pymove3d Documentation

Release en

Blender- and Pythonenthusiasts

February 19, 2017

Contents

Contents:

CHAPTER 1

Introduction

You heard about this course and now you are reading course material. Whatever your motivations are, here you will find information about Python and 3D modelling program Blender, but also new ideas how to use computers and software at school or just for fun. Take your time and read the following few chapters to get the best experience.

Legend

Objectives

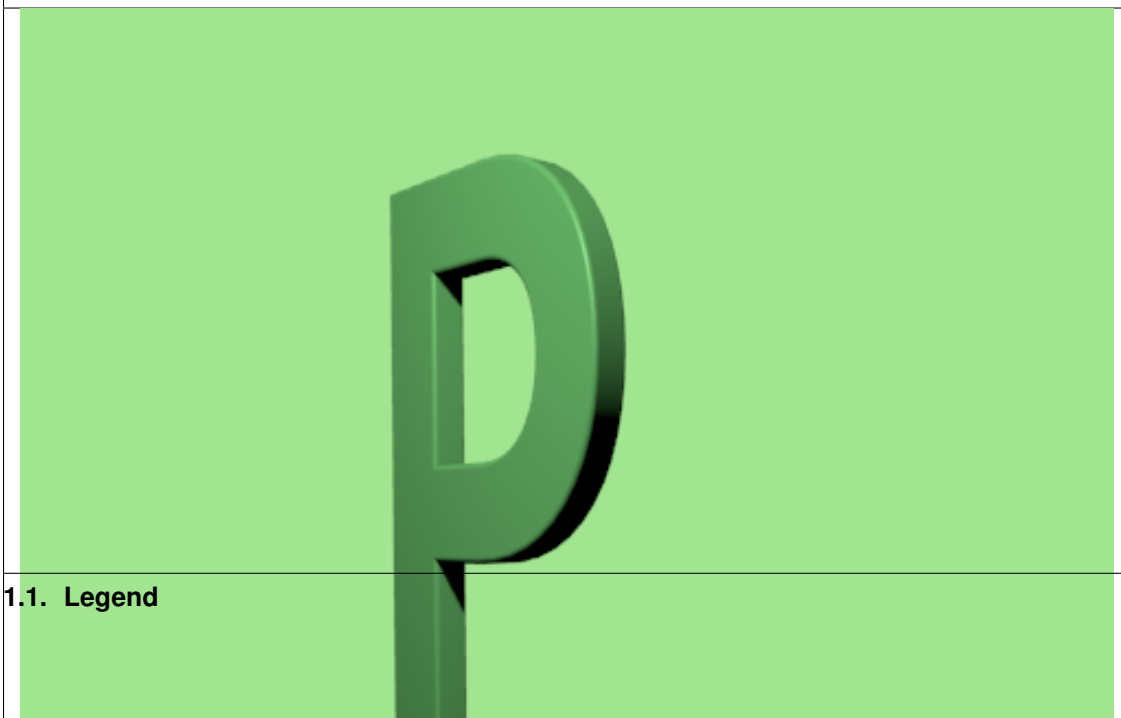
	Some sections apply only to a specific operating system. Icons are used to indica

Instructions

Tasks

1. Remember the operating system and programs you are using and their versions.
2. If the marked section is not for your program or operating system, you can skip it.

Legend

	Blender
	

How to use the course

Objectives

	How to use this course material? Some advice is given.

Instructions

Tasks

1. Read the following instructions, they give you the »big picture«.
2. Switch to the index und select one interesting topic.
3. Select the glossary in the Appendix.

Course structure

- every learning unit is named *station*
- every *station* is a isolated learning unit, but sometimes learning units depend on each other.
- each *station* is divided in tree parts
    1. objectives
    2. tasks/instructions
    3. descriptions and samples

How to use the course material?

1. First read the *objectives*.
2. It this *station* is your target, secondly read the *tasks* – read carefully, but only read, do not try to solve the exercises!
3. If the goal and the exercises match your needs, you can start reading and master the examples. Repeat the shown examples/demos.
4. Now you are ready to solve the tasks and exercises from the instruction section above.

5. Don't forget to create variations of the given tasks.

Looking for solutions

If you are looking for a solution, you have additional options:

- global search,
- index,
- glossary

The appendix also includes information on various topics in compressed form, for example, link lists to supplement the material presented here.

About the course

Objectives

What is this course for? The table of contents shows the topics of this course, bu

Instructions

Tasks

1. Use this course material, have fun!
2. If you have an idea to add or improve the material, then we won't stop you.
3. Tell other people about this course material.

At the beginning

In 2012 at the »PyCon DE« in Leipzig an proposal to establish a programming contest was discussed. It should teach young people Python, our preferred programming language and one we use in our jobs. Soon, the conference was over but nothing happened!

In march 2013, Peter Koppatz asked Reimar Bauer at the »Linuxtag« in Berlin about the progress with the idea of a programming contest. Reimar answered that nothing had happened since Leipzig!

So Peter presented the idea to combine programming with python and building 3D worlds. Still skeptical, Reimar listened to Peter and he recognized the huge potential of this idea. And so the stone began to roll. Now the first competition is over. And its success is motivation to improve the course material and prepare a version that can easily translated to other languages.

Hall of fame

People who help to create and improve this course material will be honored in our list of names (alphabetical order):

- **Anderson, Tony** – translation
- **Babenia, Anatoli** – translation, pulishing on read the docs
- **Bauer, Reimar** – coordinator, author
- **Bestwalter, Oliver** – translation
- **Boscaini, Maurizio** – translation
- **Fabula. Thomas** – community relation
- **Jens, Horst** – testing, marketing
- **Koppatz, Peter** – coordinator, author
- **Krasnitzky, Marcus** – testing
- **Meijer, Joroen** – development gui
- **Pratz, Valentin** – testing, autor
- **Retel, Joren** – testing, author
- **Stross-Radschinski, Armin** – marketing
- **Trabucci, Stefania** – Design
- **Wais, Alexander** – testing

CHAPTER 2

Blender and Python Console

In this course we use Python, which conveniently comes with Blender. This introduction explains basics of Blender and the use of its Python console.

Download/Install Blender

Objectives

For the competition you need the software *Blender*. The installation shouldn't be a problem. Blender files have the file e

Instructions

Tasks

1. Search and watch videos about Blender projects.
2. Download and installation

What can we do with Blender?

Many things, for example: video production, game development, architecture, art, ... Here's a little clip which shows what has already been done with Blender:

Arranged by: Mike Pan, Musik: Jan Morgenstern.

Where can I get the program?

You can find it here: http://www.blender.org

And here, a little clip about security:

The operation of Blender

Objectives

Blender is a window-oriented application. Some concepts are different from what you may expect, but it is not difficult. Just follow next videos and experiment a little bit to become familiar with how blender works.

Instructions

Tasks

1. Watch the clips.
2. Split the window and close unused windows.
3. Change the window type, for example to »File Browser«
4. Look at the table in the notes with the most important keyboard shortcuts.

Paneling for Programmers

The paneling in Blender is very varied and depends on the tasks which should be done. In this course we want to get to know about Python and we want to use it. Because of that the following classification is very good for the exercises:

Window Management

Here a little clip to this topic:

Numpad

The views of the 3D-View can changed in all directions. The introduction to this topic is shown by this short video clip.

When there are other special functions of Blender required, the explanation takes place at the given place.

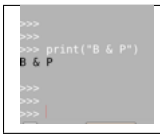
More tutorials about the usage and the operation of Blender should be present on YouTube.

Close and split windows

In the video was shown one variant to open and close windows. A second variant is a right mouseclick onto the dividing line between two windows.

Console with Blender/Python

Objectives

	Familiarize youself with the console and the interactive Python interpreter in Blender.

Instructions

Tasks

1. Which version has the system-used Python interpreter?
2. Which paths are searched by the Python interpreter?
3. Print the *sys*-modules's documentation.
4. What is the difference between the methods »version« and »version_info«
5. Where is your version of Blender located??
6. What happens if you type in Python the command: *import this*? Discuss the output in a team or in twos!

Install the interpreter

For the work with Python you have to install software - Python interpreter. In the course we use a special form of Python, because:

```
- we want to show the results at the same time in a *3D-World*,
- all users should work with the same version,
- operating system-specific questions should stay in the background and Python the focus of attentio
- the 3D-functions of Blender should be of interest only as far as necessary.
```

Blender is the software which is used in the course/competition. Download the version 2.68 from the website http://www.blender.org/download. Note the installation and start instructions on the Blender website!

Find/start console

Download the following blend file:

`Blender file with opened »console window«`

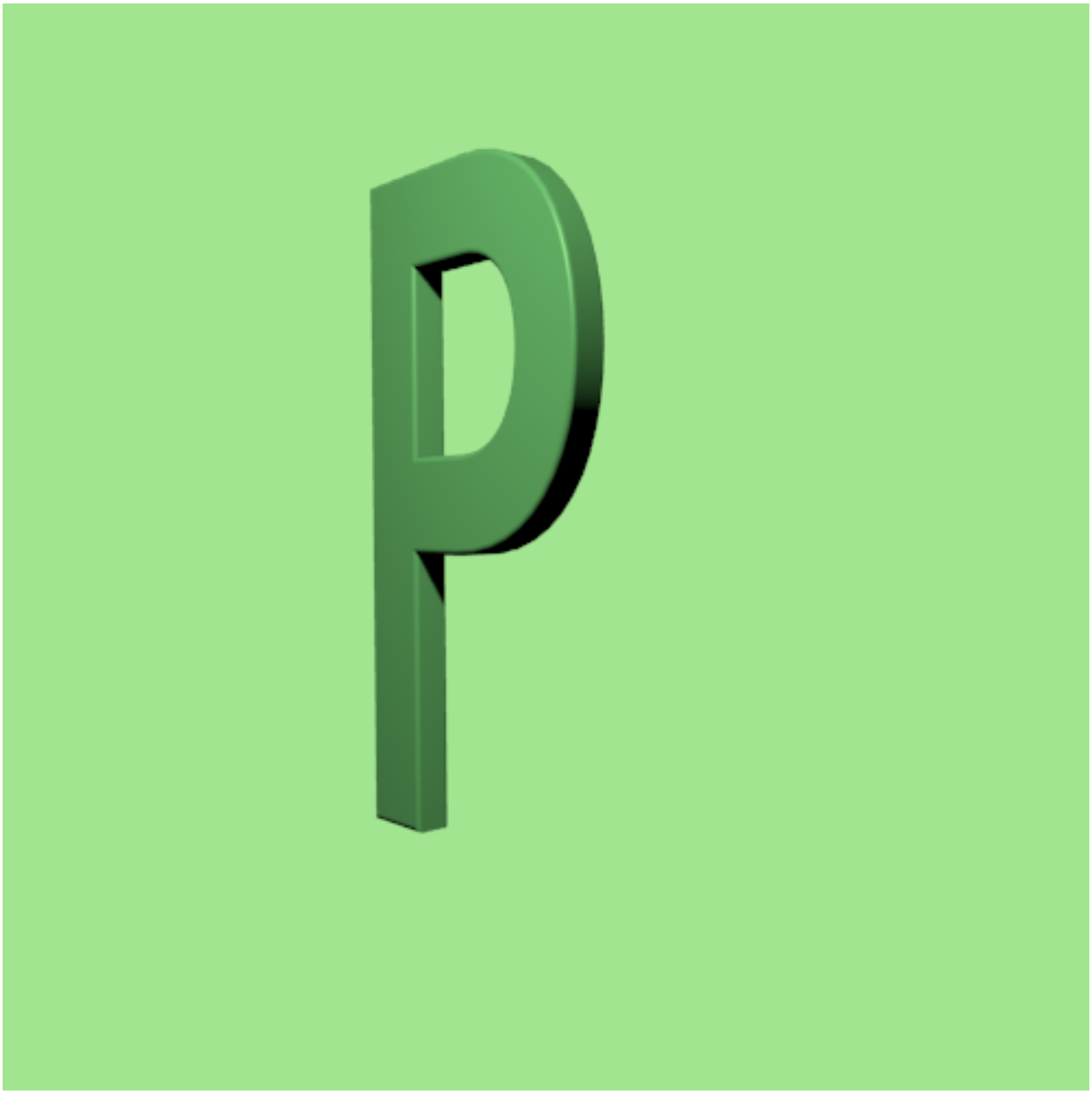
and open it over the File-Open dialog or a double click, when the file extension *.blend* is linked with the program *Blender*.

Alternatively you can switch on to the console directly:

In both cases you get the following result:

```
PYTHON INTERACTIVE CONSOLE 3.3.0 (default, Nov 18 2012, 18:00:38)  [GCC 4.2.1
66))]

Command History:     Up/Down Arrow
Cursor:              Left/Right Home/End
Remove:              Backspace/Delete
Execute:             Enter
Autocomplete:        Ctrl-Space
Zoom:                Ctrl +/-, Ctrl-Wheel
Builtin Modules:     bpy, bpy.data, bpy.ops, bpy.props, bpy.types, bpy.context
Convenience Imports: from mathutils import *; from math import *
Convenience Variables: C = bpy.context, D = bpy.data

>>>
```

System info



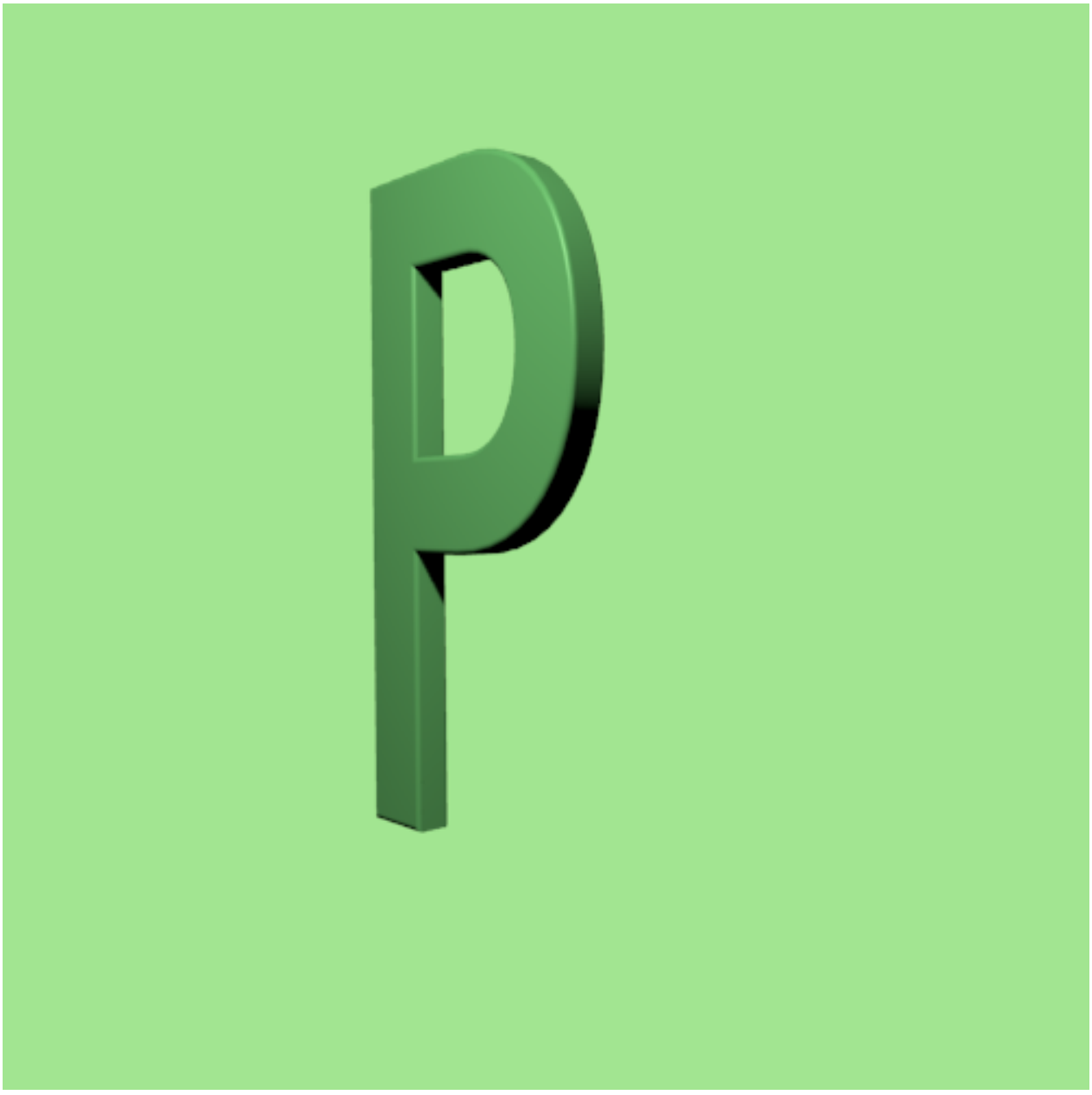
After the prompt: **>>>** you can type in instructions/commands. You finish the input with the enter key.

The output looks about this:

```
>>> dir()
['C', 'Color', 'D', 'Euler', 'Matrix', 'Quaternion', 'Vector', '__builtins__'
, '__doc__', '__name__', 'acos', 'acosh', 'asin', 'asinh', 'atan', 'atan2', '
atanh', 'bpy', 'ceil', 'copysign', 'cos', 'cosh', 'degrees', 'e', 'erf', 'erf
c', 'exp', 'expm1', 'fabs', 'factorial', 'floor', 'fmod', 'frexp', 'fsum', 'g
amma', 'geometry', 'help', 'hypot', 'isfinite', 'isinf', 'isnan', 'ldexp', 'l
gamma', 'log', 'log10', 'log1p', 'log2', 'modf', 'noise', 'pi', 'pow', 'radia
ns', 'sin', 'sinh', 'sqrt', 'tan', 'tanh', 'trunc']

>>>
```

You get a list with various names. The list is limited by square brackets and contains entries separated by commas. All names which start and end with »__« are Python system variables. You shouldn't use this notation for own names.

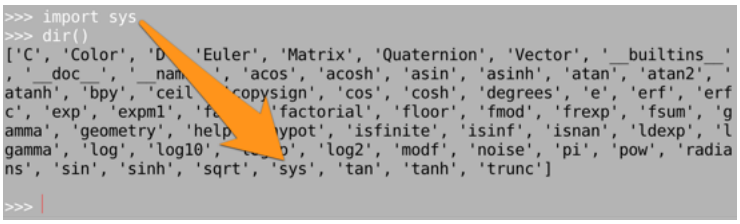
The *sys*-module



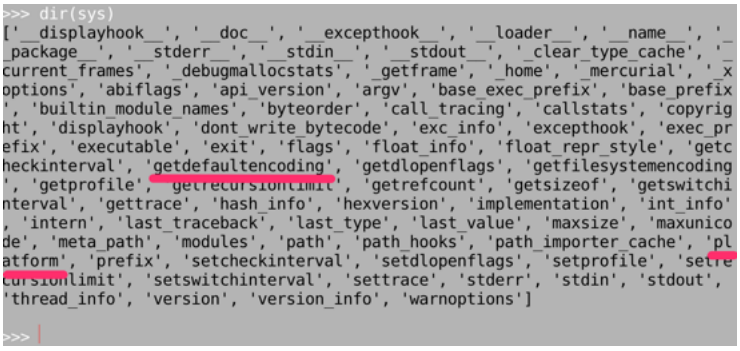
Programs are often very extensive, therefore the programs for special tasks are divided into separate modules or files. Python is modular, too. Additional functionality can be added to your Python program with the *import* function.

```
>>>import sys
```

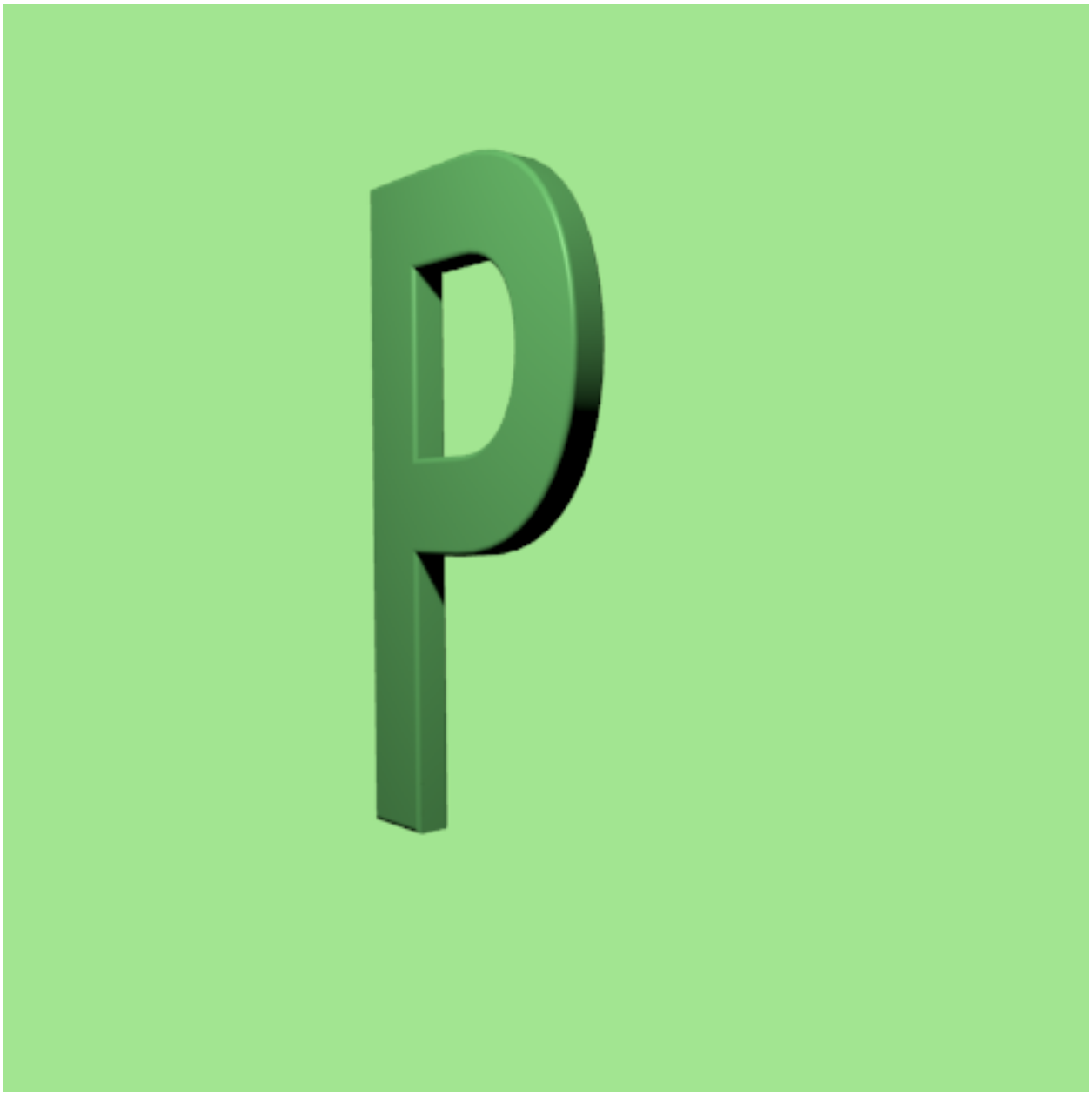
After the import of *sys*, dir() shows you a longer list:



Which functions offers the *sys-module*? Enter the the name of the module into the brackets, to get more information about it. With that you can get a first overview over a module.



Output with print



The universal output function is print. Because *platform* is defined in the module *sys*, the dot notation is used for a clear call. Like a track connects the train stations, the point connects the single names to a correct path. And now we are already midst in the object-oriented programming.

```
>>> import sys
>>> sys.platform
'darwin'

>>>
```

If the name a function/method, you have to add a pair of brackets to get the return-value. The first call in the picture returns only the method-name in a »strange packaging«. The second call adds a pair of brackets and gets the answer, in our case the description of the operation system that we use.

```
>>> sys.getdefaultencoding
<built-in function getdefaultencoding>

>>> sys.getdefaultencoding()
'utf-8'

>>>
```

Everything and more as a video clip

CHAPTER 3

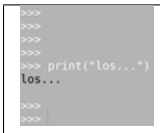
Blender Basics

This is the main part of the course. It is useful for beginners. If you are a Python beginner too, have a look to the section *Python basics*. The learning units starts with simple use cases followed by more complicated and complex scenarios.

First Steps

One template for all scripts

Objectives



Entering individual commands in the interpreter or the console is tedious and error prone. Therefore it is better to save

Exercises

Tasks

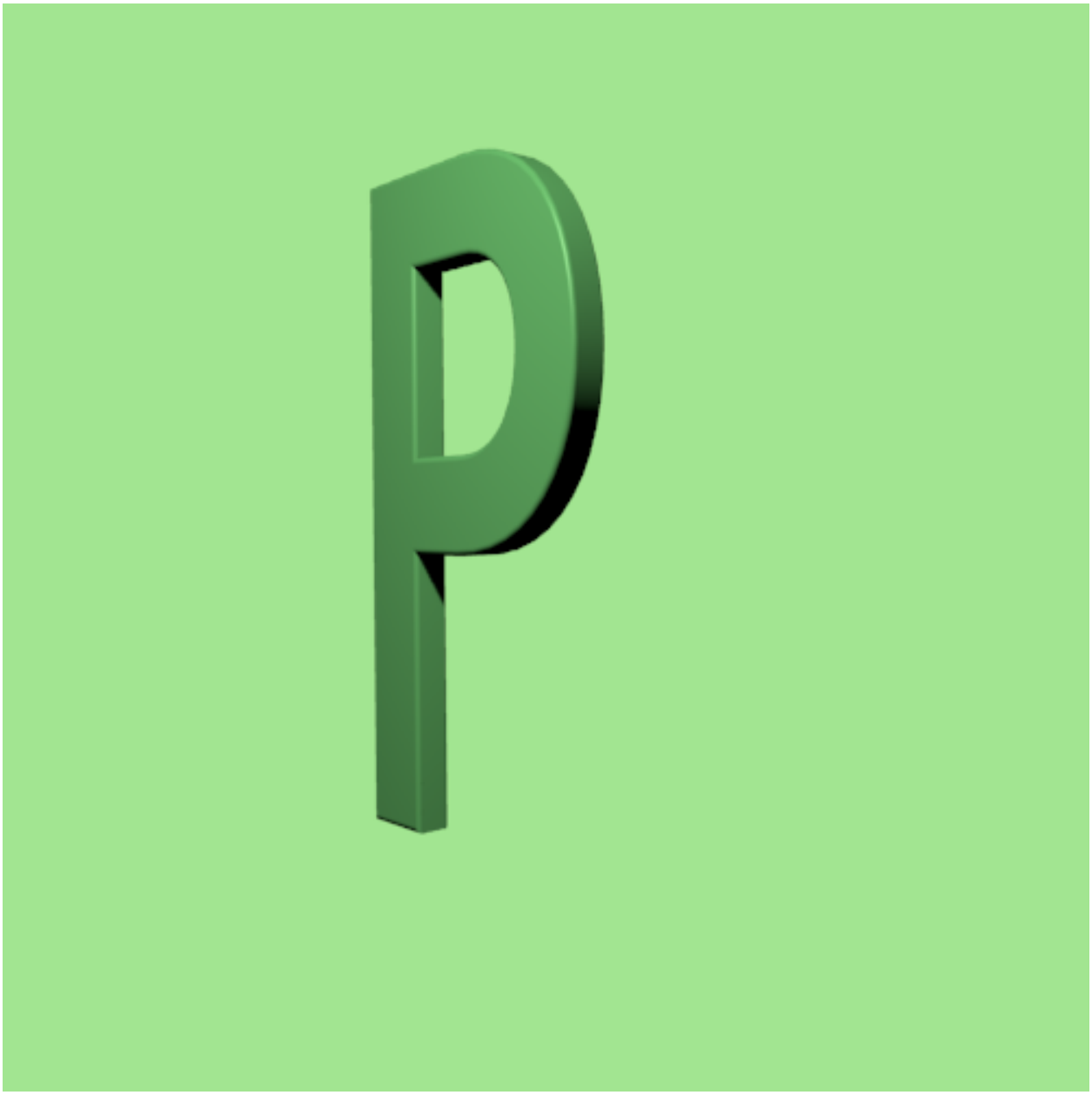
1. Open the editor in Blender, add a new file and copy the template.
2. Start customizing the template.
3. Replace the default name »Text« with a useful name. The name should have the extension »py«.
4. Activate one of the three functions and run the script.

Template for exercises

First of all we show the template as a whole. The same template is listed in the appendix, so if you need a new fresh file also have a look there *sample code (snippets)*.

```
#!bpy
"""
Name: 'Template'
Blender: 2.69
Group: 'Sample'
Tooltip: 'Template for new scripts, copy it and start coding...'
```

```
"""
import bpy


def function_1():
    """ One line describing the task of this function  """
    pass


def function_2():
    """ One line describing the task of this function  """
    print(__name__)
    print(50 * '*')


def function_3():
    """ One line describing the task of this function  """
    bpy.ops.mesh.primitive_cone_add(location=(1, 2, 1))


if __name__ == '__main__':
    # call the function for testing
    function_1()
    #function_2()
    #function_3()
```



Comments and documentation

If you count the lines in the template (31) and the lines used as comments (10) you can see that documentation is important. *If your code is not documented, it is broken!*

Read the following tips for a good programming style:

- Filenames: use a name that describes what the file is used for. Just open the file if you want to see what it does.
- Comments are used to automatically generate the documentation of a project.
- You have a better understanding what other authors have written.
- With comments you can describe algorithms and use cases.

Two kinds of comments are possible:

```
'''
This one liner says all about a module (file) in one sentence.

After an empty line, more text as documentation is possible.
Instead triples of »'«, also »"« can be used.
'''
```

In our template the lines 2-7, 12, 17 and 23 are comments. In big projects often a style guide is used. Most of them follow the guidelines in PEP-0257 und PEP-0008.

http://www.python.org/peps/pep-0008.html

http://www.python.org/peps/pep-0257.html

The second type of comments starts with a hash or number sign (#). In our template lines 28, 30 and 31 are examples for these kinds of comments. Line 28 is a real comment. On lines 30 and 31 the code is commented out, which prevents its execution.

Functions

Functions group code together to help organize it. A function has a name by which it can be called to execute the containing code. Not every task can be completed in one line and a function with a good name can make code easier to understand by describing what a specific piece of code does.

Three functions are defined in the template, each starts with the keyword *def* followed by the name of the function.

After the name of the function a list of parameters in braces is possible, if left empty (like in our template), the definition of the function ends with a pair of empty braces. The colon at the end of the line starts a new code block - this is not only true for functions - all kinds of blocks can be started with a colon.

All following lines (but at least one line) are indented with four spaces. If you want to end the block (in this case the function) you do that by unindenting the code that should not be part of the block anymore.

With this cool indentation trick it is not necessary to use curly braces or other constructs to define code blocks.

Have a look to the lines 10 to 13:

```
def function_1():
    """ One line describing the task of this function  """
    pass
```

This function consists of two lines. The first is a so called »docstring« followed by the command *pass*. Because of the rule: every block needs at least one line, we use the keyword *pass* and it does what it should do: nothing, but the block building rule is satisfied.

Calling a function

A function is called by writing its name followed by a pair of braces like in line 29-31. The last two lines thought are commented out and won't be executed. If you remove the number signs (#) at the beginning of the line, those functions will also be executed.

```
    function_1()
    #function_2()
    #function_3()
```

Function print

In the function named »function_2« we use the function *print*. The result is visible in a console.

```
    print(__name__)
    print(50 * '*')
```

The first print gives us the name of our script, the second returns fifty stars.

Two ways to use a script

Line number 27 contains a little bit of magic! In the *if*-statement we check the content of a variable named »__name__«. This variable is set automatically by the Python interpreter depending on how the file is run. Two values are possible:

1. the name of the file
2. the name »__main__«

If the value is »__main__«, all indented commands after the colon are executed - so the colon starts a new block here. If the value is different the commands in the block are not executed. With this trick you are able to run your file as a script that calls the functions directly and reuse the functions defined in your script from a different file by importing it.

Module specific functions

In the function »function_3«. we call a special Blender function. its name is »primitive_cone_add« and part of an object hierarchy. The path to the function is constructed with dots as delimiters. functions which are part of objects are called *method*. The method »primitive_cone_add« needs one parameter to place the result of the method call in our 3D-World. Have a look in the 3D Window after execution.

```
    bpy.ops.mesh.primitive_cone_add(location=(1, 2, 1))
```

Import modules

You can use elements from other files by importing them. This helps you to organize your code in several files and makes code reusable. This also gives you the possibility to use e.g. the function of Blender in your modules by importing them:

```
import bpy
```

Run a script

Objectives

Now that you learned about the basic structure of a script, we will save, load and run the script to check if it does exact

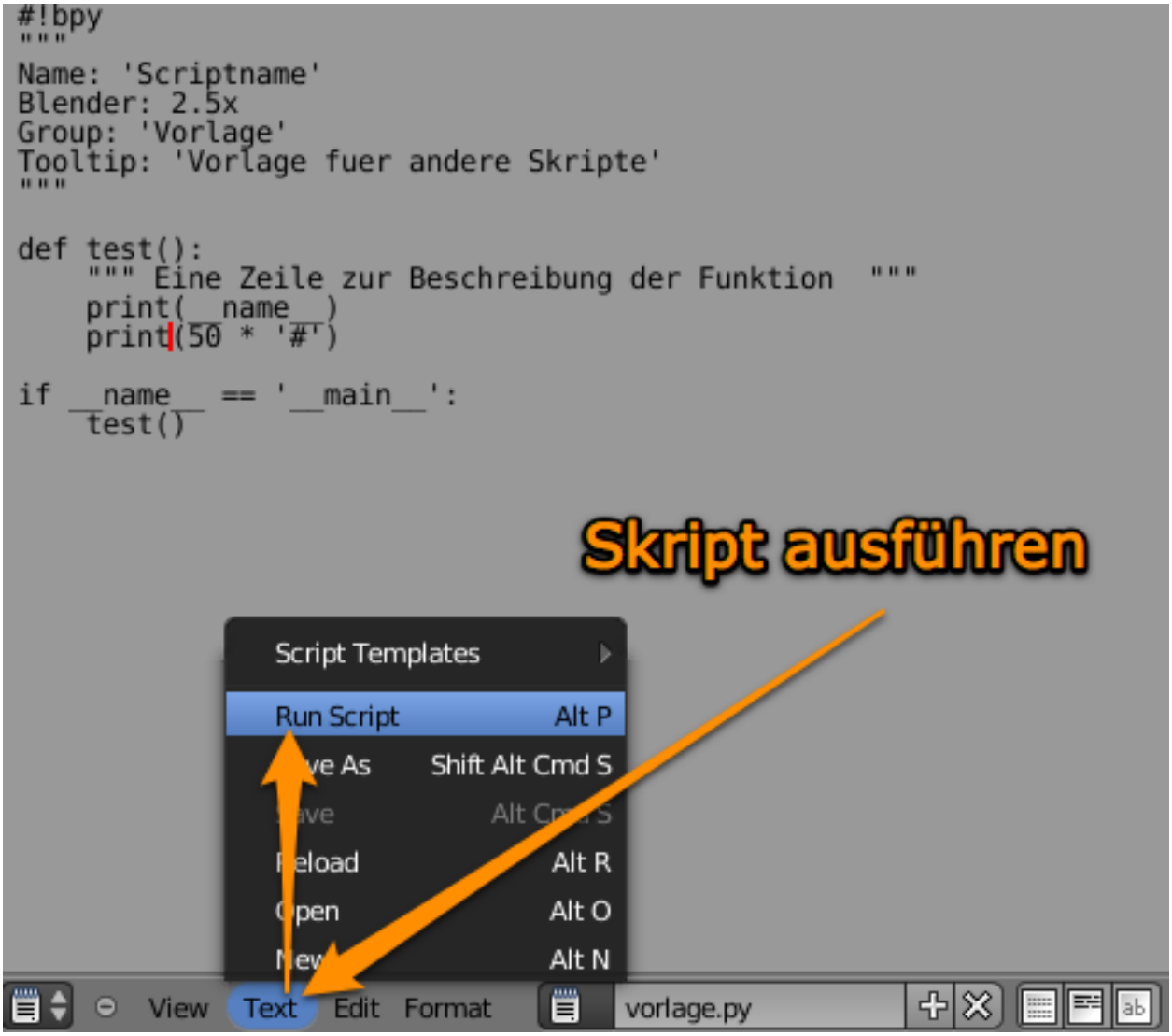
Instructions

Tasks

1. Create a project folder for all scripts.
2. Check and run the script discussed in the last station.
3. Activate all functions from the template.
4. Replace *pass* in function_1 with a *print* function.
5. Add a fourth function that is creating a new sphere in the 3D World at location (1,1,3)

Load script ...

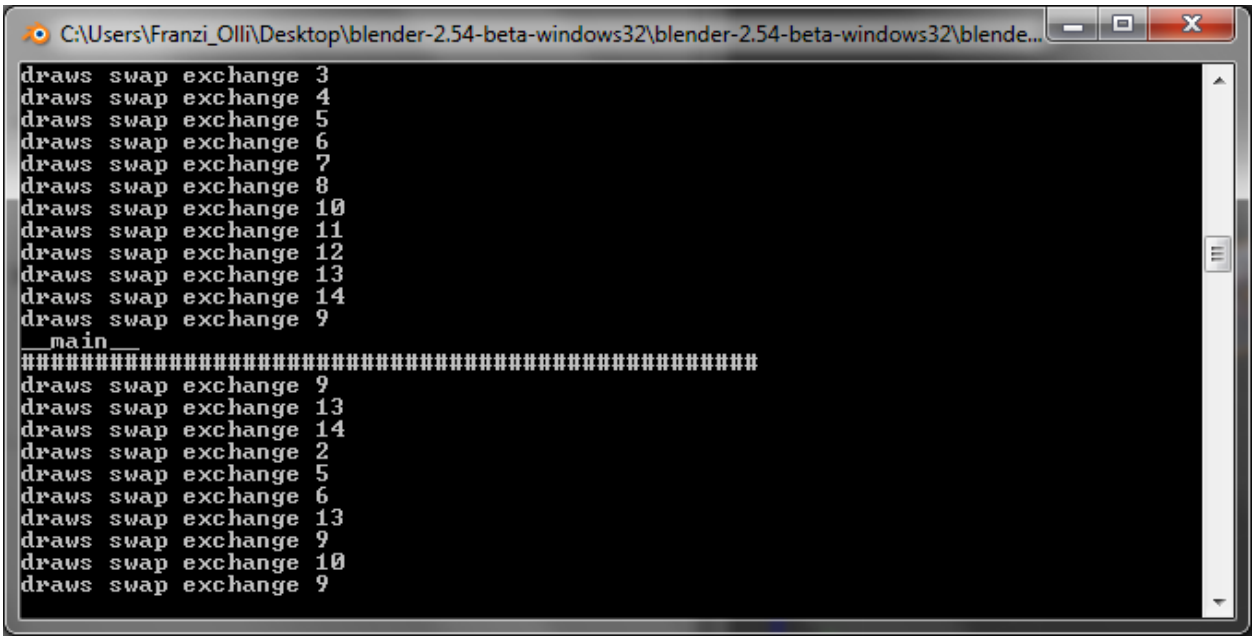
Run script...



Output...

```
draws swap exchange 18
__main__
##################################################
draws swap exchange 17
```

Windows: output


C:\Users\Franzi_Olli\Desktop\blender-2.54-beta-windows32\blender-2.54-beta-windows32\blende...
draws swap exchange 3
draws swap exchange 4
draws swap exchange 5
draws swap exchange 6
draws swap exchange 7
draws swap exchange 8
draws swap exchange 10
draws swap exchange 11
draws swap exchange 12
draws swap exchange 13
draws swap exchange 14
draws swap exchange 9
__main__
##
draws swap exchange 9
draws swap exchange 13
draws swap exchange 14
draws swap exchange 2
draws swap exchange 5
draws swap exchange 6
draws swap exchange 13
draws swap exchange 9
draws swap exchange 10
draws swap exchange 9

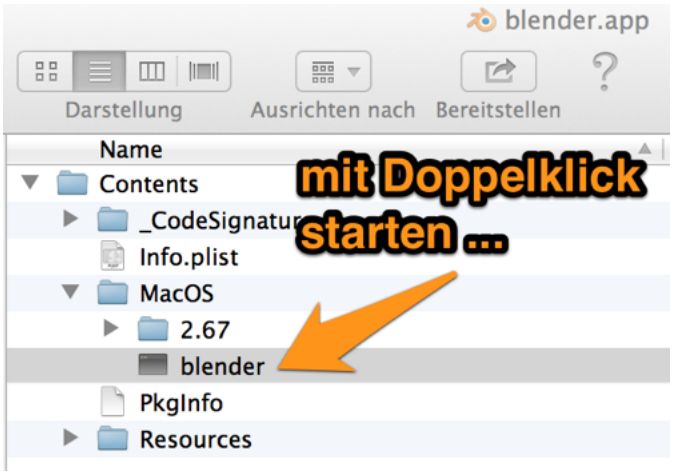
Start with console on MacOS

If you start Blender with a double click, no output will be shown. As a workaround start Blender within a console. Everybody who is familiar with a terminal, moves to the folder of blender.app and starts the application in the background with the following command:

```
./blender.app/Contents/MacOS/blender &
```

A second way to start Blender, search Blender with *Finder* and open the package with a right mouse click:

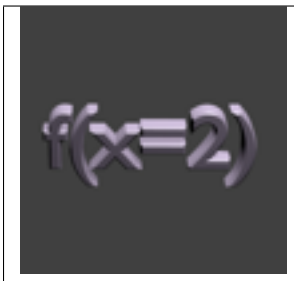
move to the blender file and start it with a double click:



Blender is starting and a terminal window is opened. Do not close this window until you want to quit Blender. Errors and print output is now visible in the terminal window.

Parameter and functions

Objectives



You learned about the structure of a script and how to run it. Now we define a new function with a parame

Instructions

Tasks

1. Create a new scrtipt and save it as *bprint.py* in your project folder.
2. Run the script and check the output in the 3D View.
3. Rewrite the script, the output function should set the text on a given position in the 3D-View. Use a second parameter for this task.
4. How is the function call for the mascot of Blender, the monkey named »Suzanne«?

bprint – output into the 3D view

How can you create text in a 3D world? With our own function we can add new functionality. A new function named bprint (b stands for Blender) will be created and used. Have a look at the complete script.

```
#!bpy
"""
Name: 'bprint.py'
Blender: 2.69
Group: 'Experiment'
Tooltip: 'Output of text in a scene'
"""

import bpy


def bprint(print_text):
    """Places a given text in the 3D view """

    bpy.ops.object.text_add(location=(0, 0, 0), rotation=(0, 0, 0))
    bpy.ops.object.editmode_toggle()
    bpy.ops.font.delete()
    bpy.ops.font.text_insert(text=print_text)
    bpy.ops.object.editmode_toggle()


if __name__ == '__main__':
    # Stop edit mode
    if bpy.ops.object.mode_set.poll():
        bpy.ops.object.mode_set(mode='OBJECT')

    # delete all mesh objects from a scene
    bpy.ops.object.select_by_type(type='MESH')
    bpy.ops.object.delete()

    # delete all text objects from a scene
    bpy.ops.object.select_by_type(type='FONT')
    bpy.ops.object.delete()

    # call the new function
    bprint("If in doubt, just do it")

```

Defining parameters

In our template we used a parameter with the function *print*:

```
print(__name__)
```

The definition of the new funktion *bprint* defines a parameter as well. The parameter is named »print_text« - the name of the parameter(s) can be choosen freely.

```
#!bpy
"""
Name: 'bprint.py'
Blender: 2.69
Group: 'Experiment'
Tooltip: 'Output of text in a scene'
"""

import bpy


def bprint(print_text):
    """Places a given text in the 3D view """

    bpy.ops.object.text_add(location=(0, 0, 0), rotation=(0, 0, 0))
    bpy.ops.object.editmode_toggle()
    bpy.ops.font.delete()
    bpy.ops.font.text_insert(text=print_text)
    bpy.ops.object.editmode_toggle()


if __name__ == '__main__':
    # Stop edit mode
    if bpy.ops.object.mode_set.poll():
        bpy.ops.object.mode_set(mode='OBJECT')

    # delete all mesh objects from a scene
    bpy.ops.object.select_by_type(type='MESH')
    bpy.ops.object.delete()

    # delete all text objects from a scene
    bpy.ops.object.select_by_type(type='FONT')
    bpy.ops.object.delete()

    # call the new function
    bprint("If in doubt, just do it")
```

Note Parameters are placed between the rounded braces behind the name of the function.

Optional parameters

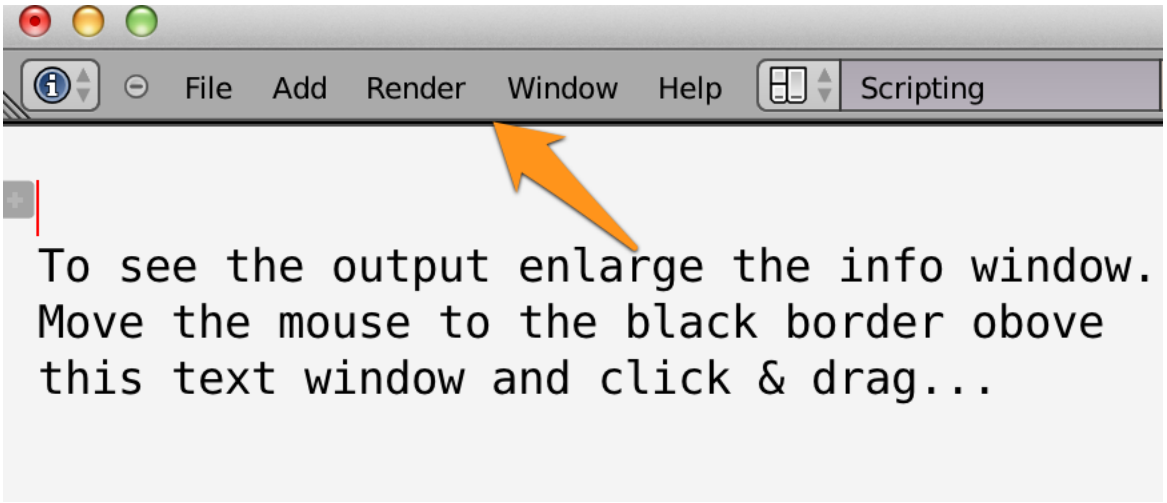
If you define a function like in the example above, you can not call the function without passing an argument into the function. If you want to make the parameter optional you can give the parameter a default value. If that parameter does not get fed an argument when the function is called, the default value is used instead. This is useful if many parameters are possible but not all are mandatory. Examples for optional parameters:

```
def my_functions(location=(0,0,0))
def my_function(rotation=(0,1,1))
def my_function(text="hello, world")
```

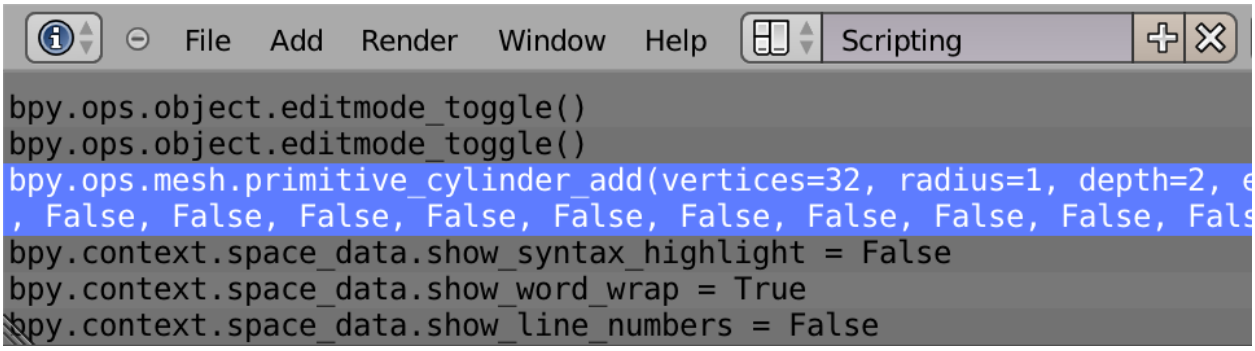
note An optional parameter is defined as a key-value-pair, as shown in the examples above.

Blender showcase for parameters

Now let's have a look at an example in Blender. First of all enlarge the info window in Blender:



You are now able to see all functioncalls that your are doing. If you add a cylinder, selecting the menu *Add » Mesh » Cylinder* you will see a very long function call.
You can select a line by right clicking it - then you can copy and paste the line to your text window.



For better readability of this »ASCII-tapeworm« all parameters are broken up to separate lines:

```
bpy.ops.mesh.primitive_cylinder_add(vertices=32,
                                    radius=1,
                                    depth=2,
                                    end_fill_type='NGON',
                                    view_align=False,
                                    enter_editmode=False,
                                    location=(-4.93998, -5.74176, 4.41665),
                                    rotation=(0, 0, 0),
                                    layers=(True,
                                            False, False, False, False,
                                            False, False, False, False,
                                            False, False, False, False,
                                            False, False, False, False,
                                            False, False, False))
```

Most of the lines are optional. The shortest version is:

```
bpy.ops.mesh.primitive_cylinder_add(location=(-4.93998, -5.74176, 4.41665))
```

And with some better readable parameter values:

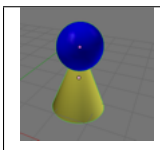
```
bpy.ops.mesh.primitive_cylinder_add(location=(2, 2, 2))
```

Now you have a new way to discover useful and/or new function in Blender.

API

Search the API

Objectives



Creating objects from a menu as with *Add » Mesh »* ... is easy. The same gaol is also achievable within a Python script

Instructions

Tasks

1. Create an new Python script.
2. Create a function that lists all methods available in Blender who create a mesh objects (primitive) .
3. Create one instance from each object in your scene. No object should touch an other.
4. Create a figure as a composition from a cone and a sphere (UV_Sphere). There are two possible solutions. Maybe you will find a third version.

Structure of the API

All available functionality is grouped by different modules. Only the first two are required in the beginning.

Application Modules

- *Data Access (bpy.data)*
- *Operators (bpy.ops)*
- Types (bpy.types)
- Utilities (bpy.utils)
- Path Utilities (bpy.path)
- Application Data (bpy.app)
- Property Definitions (bpy.props)

Standalone Modules

- Math Types & Utilities (mathutils)
- Font Drawing (blf)
- Audio System (aud)

Game Engine Modules

- Game Engine bge.types Module
- Game Engine bge.logic Module
- Game Engine bge.render Module
- Game Engine bge.events module

Which »primitives« are available?

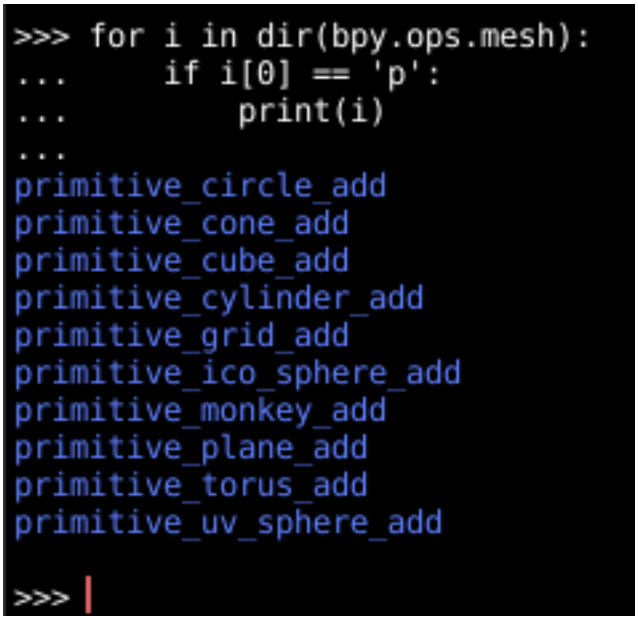
If you start creating a complex world in 3D/Blender, often a good starting point is to create simple objects. Lets create the available mesh objects. But which objects are available?

1. First and easiest solution: have a look at the menu *Add » Mesh » ...* and look at the tooltip at the mouse pointer.
2. RTFM (read the fine manuals)
   (a) Online documentation, search for »mesh primitive« on Blenderdocs
   (b) Search the file »OperatorList.txt«.
   You can open it in the Blender editor using the menu sequence:
   Help » Operator Sheet Sheet.
   A copy is also included in the appendix: *OperatorList.txt*
3. If you are familiar with the structures in Blender, the console is a good starting point to discover the API.
4. If your are not successful searching the API, ask the community. A linklist is available in the appendix.

   Note It is essential to work with the API, nobody is able to remember all possibilities!

Finding the right method(s)

Add mesh objects with Python instead using the menu is shown in a screenshot where the console is used. This is error prone, using a script is a good alternative.



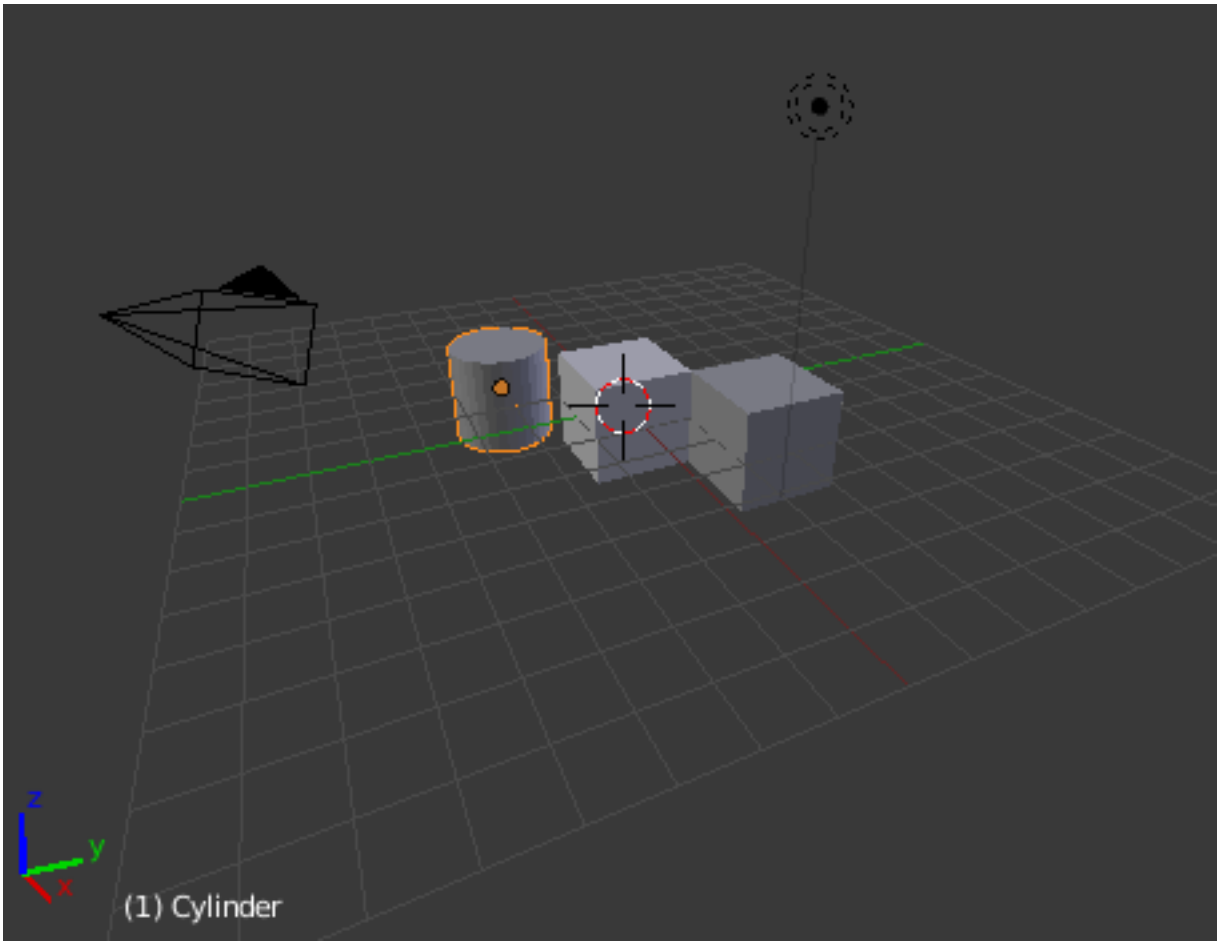
Create the objects

Now that we know how to create new objects lets create a script for this task. Where to place the new object, is controlled by the parameter *location*, a *tuple* (values in rounded braces) with tree Values for the x-, y- und z-aches.

Here is a fragment...

```
bpy.ops.mesh.primitive_cube_add(location=(2, 2, 0))
bpy.ops.mesh.primitive_cylinder_add(location=(-2, -2, 0))
```

And so the final solution might look like:



Short video about using the API

Showcase

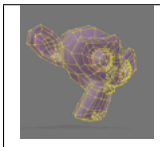
Try to recreate the following construct unsing only cubes and the menu *Add » Mesh »* ...!

Todo

create a script...

Search the scene

Objectives



If the objects are created, in a second step further manipulations are necessary. The next question is: Which objects are

Instructions

Tasks

1. What is possible with lists? If in doubt, have a look at *Python basics: lists*
2. Create a script that executes the code from this station.
3. Add a new object to the scene and run the script again.

Show all objects

In an new Blender file, in a scene some objects are placed by default. How to get a list of available objects is shown with the following script:

```
#!bpy
"""
Name: 'object-list.py'
Blender: 2.69
Group: 'Discover'
Tooltip: 'Find objects in a scene'
"""

import bpy


def get_list_of_objects():
    """ Print list of objects in a scene """

    for i in bpy.data.objects:
        print(i)


if __name__ == '__main__':
    # call the new function
    get_list_of_objects()
```

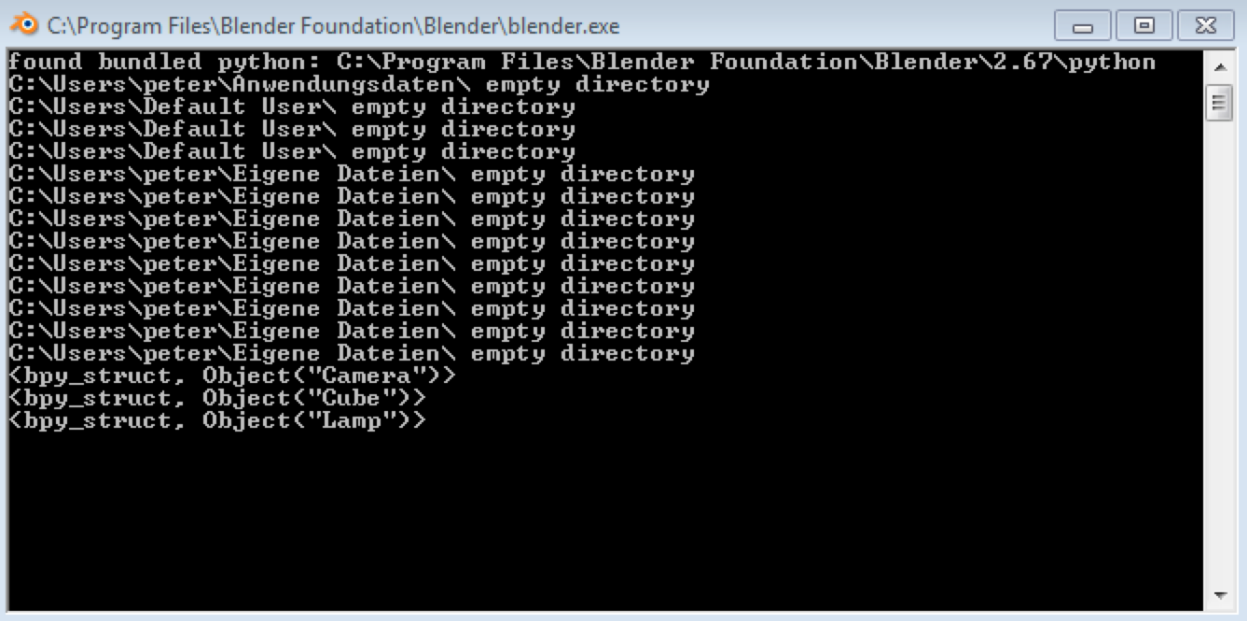
The modul *bpy.data* contains as structure *objects*. From this you can get a list of objects of the context, the scene. A for loop is used to print all objects:

Result:

```
<bpy_struct, Object("Camera")>
<bpy_struct, Object("Cube")>
<bpy_struct, Object("Lamp")>
```

You can imagine, what sort of objects are available in the scene.

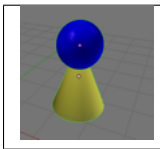
This is the output on a command line on windows.



Compositions

Composite pieces

Objectives



In this station a figure for the halma game will be created. It is a composition from mesh objects (primitives).

Instructions

Tasks

1. Create a composite pice as shown in this learning unit.
2. Move the new object with the mouse.
3. Join the parts and name the object »halma_figure«.
4. Move the new object again.
5. Duplicate the object and have a look at the counter in the header of the info window.

First version

```
#! bpy
"""
Name: 'halma_figure.py'
Blender: 269
```

```
Group: 'Example composit piece'
Tooltip: 'Crate and add a halma figure'
"""
import bpy


def halma_figure():
    '''Crate a halma figure from a ball and a cone'''

    # get the context
    scn = bpy.context.scene

    # names of the parts
    parts = ['head', 'body', 'halma_figure']

    # crate the head
    bpy.ops.mesh.primitive_uv_sphere_add(location=(2, 2, .9))
    obj = bpy.context.object
    obj.scale[0] = .6
    obj.scale[1] = .6
    obj.scale[2] = .6
    obj.name = parts[0]

    scn.objects.active = scn.objects[parts[0]]

    # create the body
    bpy.ops.mesh.primitive_cone_add(location=(2, 2, 0))
    obj = bpy.context.object
    obj.scale[0] = .8
    obj.scale[1] = .8
    obj.name = parts[1]
    scn.objects.active = scn.objects[parts[1]]

if __name__ == '__main__':
    bpy.ops.object.select_by_type(type='MESH')
    bpy.ops.object.delete()
    halma_figur()
```

Select the context

Selecting the *context* is the first step. Next step is creating and positioning objects. The *context* is assigned to a variable named *scn*.

```
    # get the context
    scn = bpy.context.scene
```

Name it

If you name the objects, later on it is easier to select one of them.

```
    # names of the parts
    parts = ['head', 'body', 'halma_figure']
```

First part of the composite pice

The sphere is the first part. We also scale the object afterwards and name it »head«.

```
    # crate the head
    bpy.ops.mesh.primitive_uv_sphere_add(location=(2, 2, .9))
    obj = bpy.context.object
    obj.scale[0] = .6
    obj.scale[1] = .6
    obj.scale[2] = .6
    obj.name = parts[0]
```

Second part of the composite pice

Same procedure as with the first part.

```
    # create the body
    bpy.ops.mesh.primitive_cone_add(location=(2, 2, 0))
    obj = bpy.context.object
    obj.scale[0] = .8
    obj.scale[1] = .8
    obj.name = parts[1]
    scn.objects.active = scn.objects[parts[1]]
```

Call the function

Let's have a look to the final version. Are all pieces on the right place and have the right dimension? The old objects are deleted, so a frech and empty scene is used. You can check the info window. There are also statistics about the number of objects.

```
if __name__ == '__main__':
    bpy.ops.object.select_by_type(type='MESH')
    bpy.ops.object.delete()
    halma_figur()
```

How many objects are selected/available?

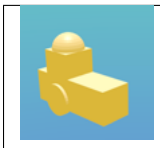
How to connect the parts to one?

Add the following lines of code to glue together the two pieces to a new one.

```
# connect parts to a new one
scn.objects[parts[0]].select = True
scn.objects[parts[1]].select = True
bpy.ops.object.join()
obj.name = parts[2]
```

Rotate/move/assemble

Objectives



Creating new Objects is easy, afterwards we have to scale and move the pieces to a new location. How this can achieve

Instructions

Tasks

1. Move the pieces of the example to a different place.
2. Join the pieces to a single one and name it »marsmobil«.
3. Create the following object, shown in this picture:

4. Create your own object.

Creating objects

The complete script, parts of the script are explained afterwards.

```
#!bpy
"""
Name: 'mars_mobile.py'
Blender: 2.69
Group: 'Composition'
Tooltip: 'Rotate, locate and scale objects assembled to a futuristic prototype'
"""
import bpy


def create_objects():
    """Create objects from a list of attributes

    List values:

    object name    -- string
    object type    -- string
    location       -- tupel of integers
    """
    objectlist = [('cu1', 'cube', (-6, 1, 1)),
                  ('cu2', 'cube', (-3, 1, 1)),
                  ('cy1', 'cylinder', (0, 1, 1)),
                  ('cy2', 'cylinder', (3, 1, 1)),
                  ('uv1', 'uv_sphere', (6, 1, 1)),
                  ('uv2', 'uv_sphere', (9, 1, 1))]

    for element in objectlist:
        if element[1] == 'cube':
            bpy.ops.mesh.primitive_cube_add(location=element[2])
        if element[1] == 'cylinder':
            bpy.ops.mesh.primitive_cylinder_add(location=element[2])
        if element[1] == 'uv_sphere':
            bpy.ops.mesh.primitive_uv_sphere_add(location=element[2])
        # give it a name
        obj = bpy.context.object
        obj.name = element[0]


def select_cubes():
    """Sample: select two objects by name """

    bpy.ops.object.select_pattern(pattern="cu2")
```

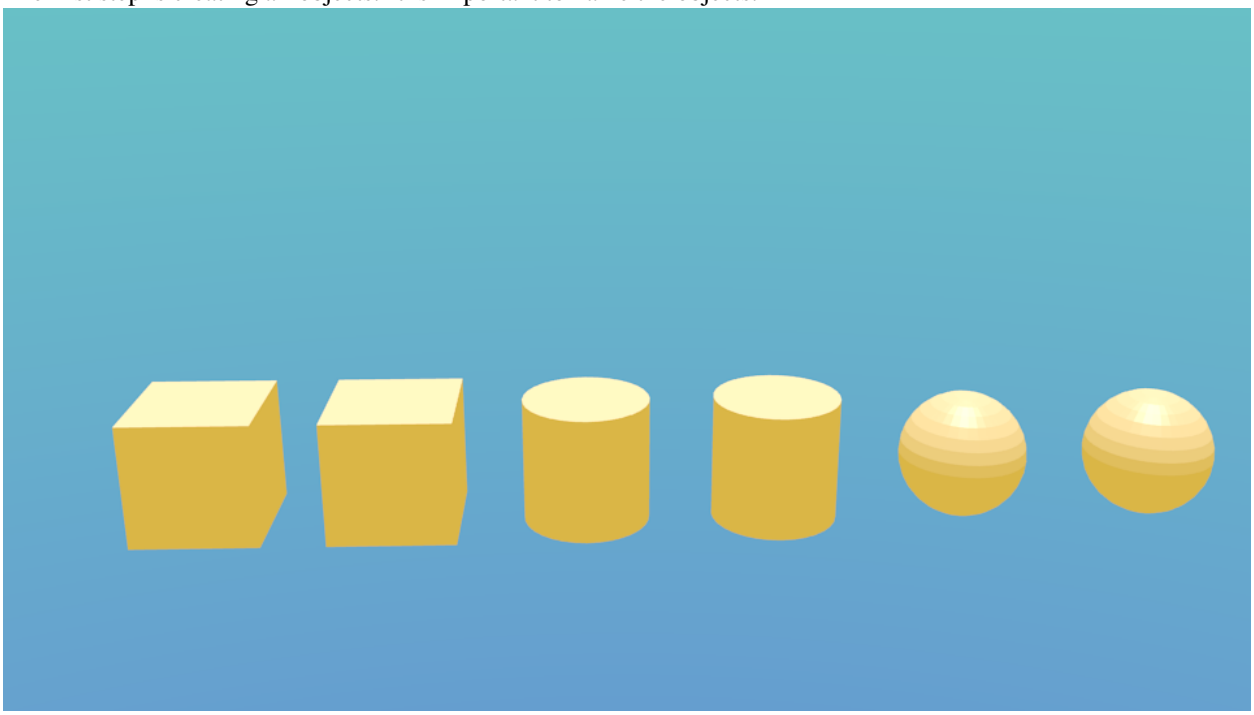
```
    bpy.ops.object.select_pattern(pattern="cu1")


def activate_object():
    """Sample: activate an object by name"""

    bpy.context.scene.objects.active = bpy.data.objects["cu2"]


def scale_cu2():
    """Select, scale and move cube 2"""

    bpy.context.scene.objects.active = bpy.data.objects["cu2"]
    obj = bpy.context.scene.objects.active
    obj.scale = (1, 1, 3)
    obj.location = (-3, 1, 3)


def scale_cy():
    """Select cylinders, scale and move"""

    bpy.context.scene.objects.active = bpy.data.objects["cy1"]
    obj = bpy.context.scene.objects.active
    obj.scale = (1, 1, .2)
    obj.location = (0, 1, .2)

    bpy.context.scene.objects.active = bpy.data.objects["cy2"]
    obj = bpy.context.scene.objects.active
    obj.scale = (1, 1, .2)
    obj.location = (3, 1, .2)


def assemble_mars_mobile():
    """Create a composite piece """

    pi_half = 3.141592/2

    # body
    bpy.context.scene.objects.active = bpy.data.objects["cu2"]
    obj = bpy.context.scene.objects.active
    obj.location = (-3, 1, 1.5)
    # rotation
    obj.rotation_euler = [pi_half, 0, 0]

    # wheel (right)
    bpy.context.scene.objects.active = bpy.data.objects["cy1"]
    obj = bpy.context.scene.objects.active
    obj.location = (-4, 2, 1)
    obj.rotation_euler = [0, pi_half, 0]

    # wheel (left)
    bpy.context.scene.objects.active = bpy.data.objects["cy2"]
    obj = bpy.context.scene.objects.active
    obj.location = (-2, 2, 1)
    obj.rotation_euler = [0, pi_half, 0]

    # wheel (front)
    bpy.context.scene.objects.active = bpy.data.objects["uv1"]
```

```
    obj = bpy.context.scene.objects.active
    obj.location = (-3, -1, 1)

    # cabin
    bpy.context.scene.objects.active = bpy.data.objects["cu1"]
    obj = bpy.context.scene.objects.active
    obj.location = (-3, 2, 3)

    bpy.context.scene.objects.active = bpy.data.objects["uv2"]
    obj = bpy.context.scene.objects.active
    obj.location = (-3, 2, 4)


if __name__ == '__main__':
    if bpy.ops.object.mode_set.poll():
        bpy.ops.object.mode_set(mode='OBJECT')
    # delete all meshes from a szene
    bpy.ops.object.select_by_type(type='MESH')
    bpy.ops.object.delete()
    create_objects()
    ## comment and uncomment the lines as you like
    # select_cubes()
    # aktivate_object()
    # scale_cu2()
    # scale_cy()
    # assemble_mars_mobile()
```

How it works

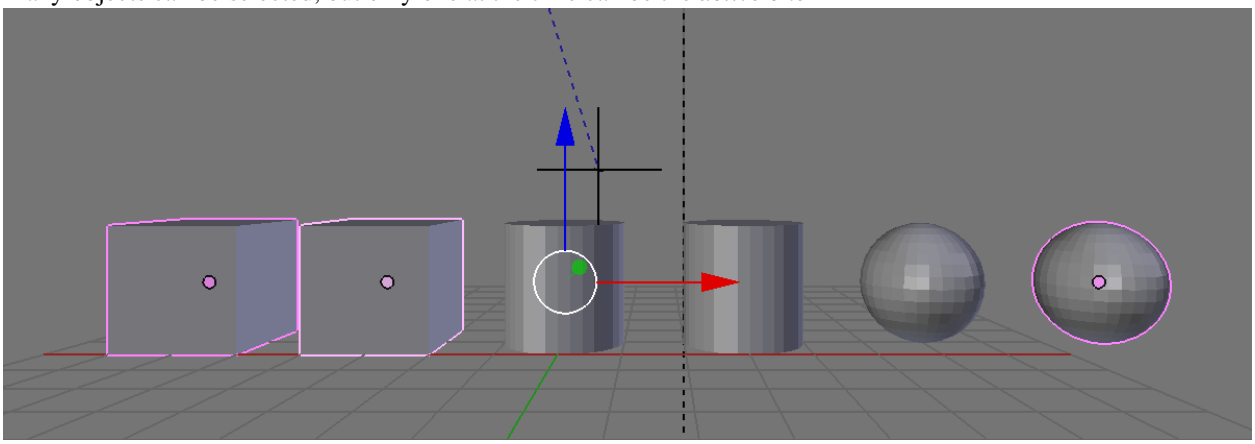
Every function is solving a particular problem. You can test the different functions if you comment or uncomment the function calls at the end of the script.

The first step is creating all objects. It is important to name the objects.



You have to distinguish between selected objects an one *active object*.

Many objects can be selected, but only one at the time can be the *active one*



How many objects are selected, is visible in the info window.

Select objects

```
def select_cubes():
    """Sample: select two objects by name """

    bpy.ops.object.select_pattern(pattern="cu2")
    bpy.ops.object.select_pattern(pattern="cu1")
```

Activate an object

Manipulations are only possible with the active object.

```
def activate_object():
    """Sample: activate an object by name"""

    bpy.context.scene.objects.active = bpy.data.objects["cu2"]
```

Scale objectes

Scaling an object is possible in tree dimensions. If no scaling should happened, the scaling factor is 1. In our example only the z-achs is scaled. Afterwards we move the center of the objects, therefore they are positioned on the ground again.

```
def scale_cu2():
    """Select, scale and move cube 2"""
```

```
    bpy.context.scene.objects.active = bpy.data.objects["cu2"]
    obj = bpy.context.scene.objects.active
    obj.scale = (1, 1, 3)
    obj.location = (-3, 1, 3)
```

Both cylinders are minimized.

```
def scale_cy():
    """Select cylinders, scale and move"""

    bpy.context.scene.objects.active = bpy.data.objects["cy1"]
    obj = bpy.context.scene.objects.active
    obj.scale = (1, 1, .2)
    obj.location = (0, 1, .2)

    bpy.context.scene.objects.active = bpy.data.objects["cy2"]
    obj = bpy.context.scene.objects.active
    obj.scale = (1, 1, .2)
    obj.location = (3, 1, .2)
```

Rotation with Euler

All objects are scaled and in the next step they can be assembled. Some parts have to be rotated. We calculate the value once:

```
pi_half = 3.141592/2
```

and use it as often as needed and *pi_half*. It is always a rotation of 90°.

```
obj.rotation_euler = [pi_half, 0, 0]
```

More about calculating a rotation is explained at Rotating an object

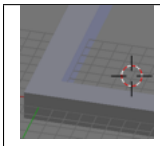
Now we have finished our prototype and the next car race at the mars can start :-)

Todo

translate station *eurlerrotaton* linked from here...

Labyrinth/maze

Objectives



This station is a repetition of techniques already shown. But wait, it is also the starting point for a game that we build i

Instructions

Tasks

1. Construct your own level.
2. Create objects with different shape for each symbol.

Labyrinth

Our starting point is a maze build out of cubes.

```
#! bpy
"""
Name: 'labyrinth.py'
Blender: 269
Group: 'Example'
Tooltip: 'Creating a maze'
"""
import bpy

level_00 = ["####################",
            "#.                 #",
            "#         $        #",
            "#                  #",
            "#                  #",
            "#         @        #",
            "#                  #",
            "#                  #",
            "#                  #",
            "#                  #",
            "####################"]


level_01 = ["     #####          ",
            "     #   #          ",
            "     #$  #          ",
            "   ###  $##         ",
            "   #  $ $ #         ",
            "### # ## #   ###### ",
            "#    # ## #####  ..# ",
            "# $  $          ..# ",
            "##### ### #@##  ..# ",
            "     #     ######### ",
            "     #######        "]


```

```
36 def sokobanLevel(level):
37     """Create a maze by adding cube objects"""
38
39     cols = len(level[0])
40     rows = len(level)
41
42     for row in range(rows):
43         for i in range(cols):
44             if level[row][i] == '#':
45                 bpy.ops.mesh.primitive_cube_add(location=(row*2, i*2, 0))
46
47 if __name__ == '__main__':
48     sokobanLevel(level_00)
```

Iterate with for

The function *len* gives us the number to iterate in a for loop through the lists.

```
    cols = len(level[0])
    rows = len(level)
```

Comparison

With each comparison we decide to generate a new object otherwise nothing happened.

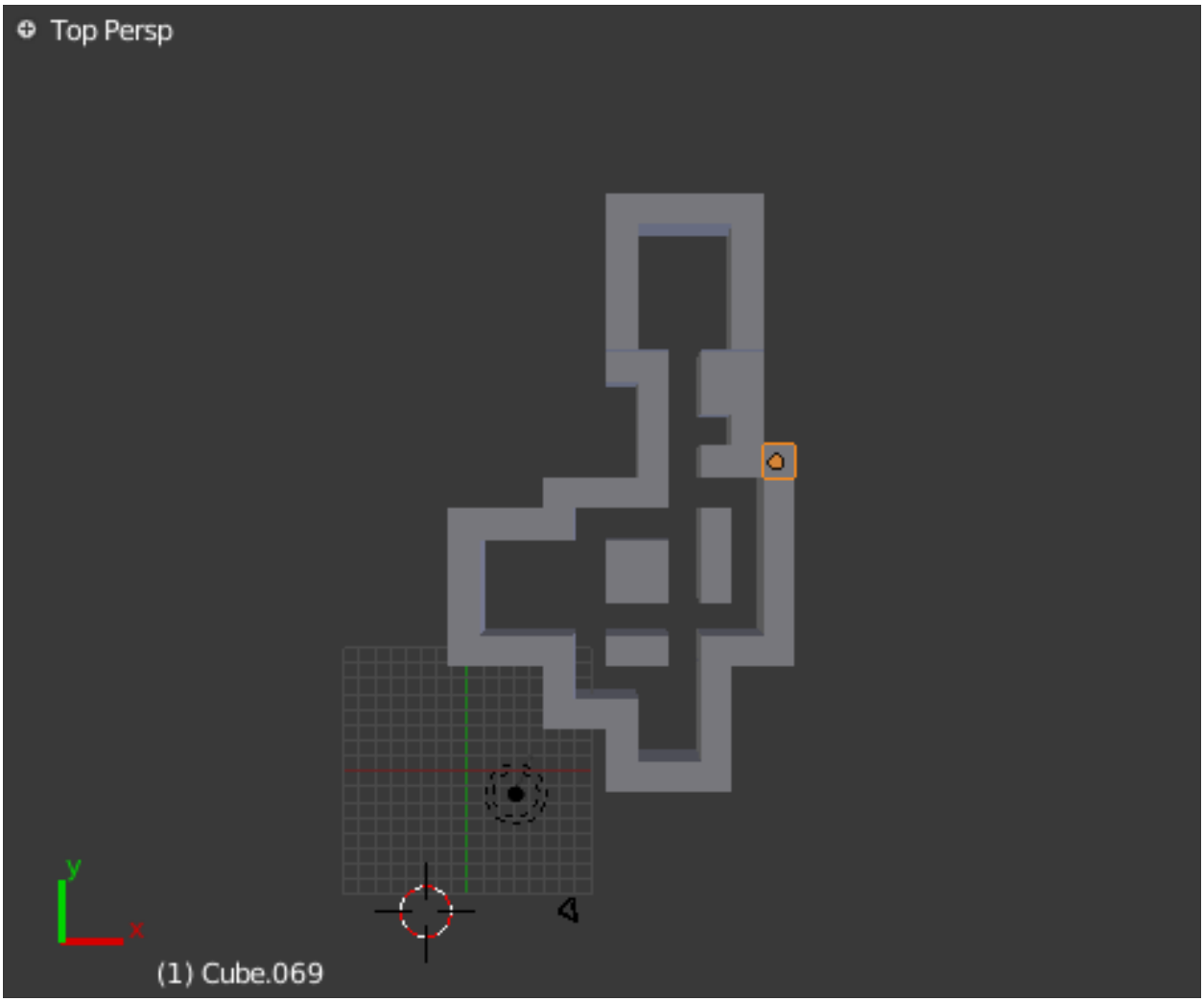
```
    for row in range(rows):
        for i in range(cols):
            if level[row][i] == '#':
                bpy.ops.mesh.primitive_cube_add(location=(row*2, i*2, 0))
```

Function call with parameter

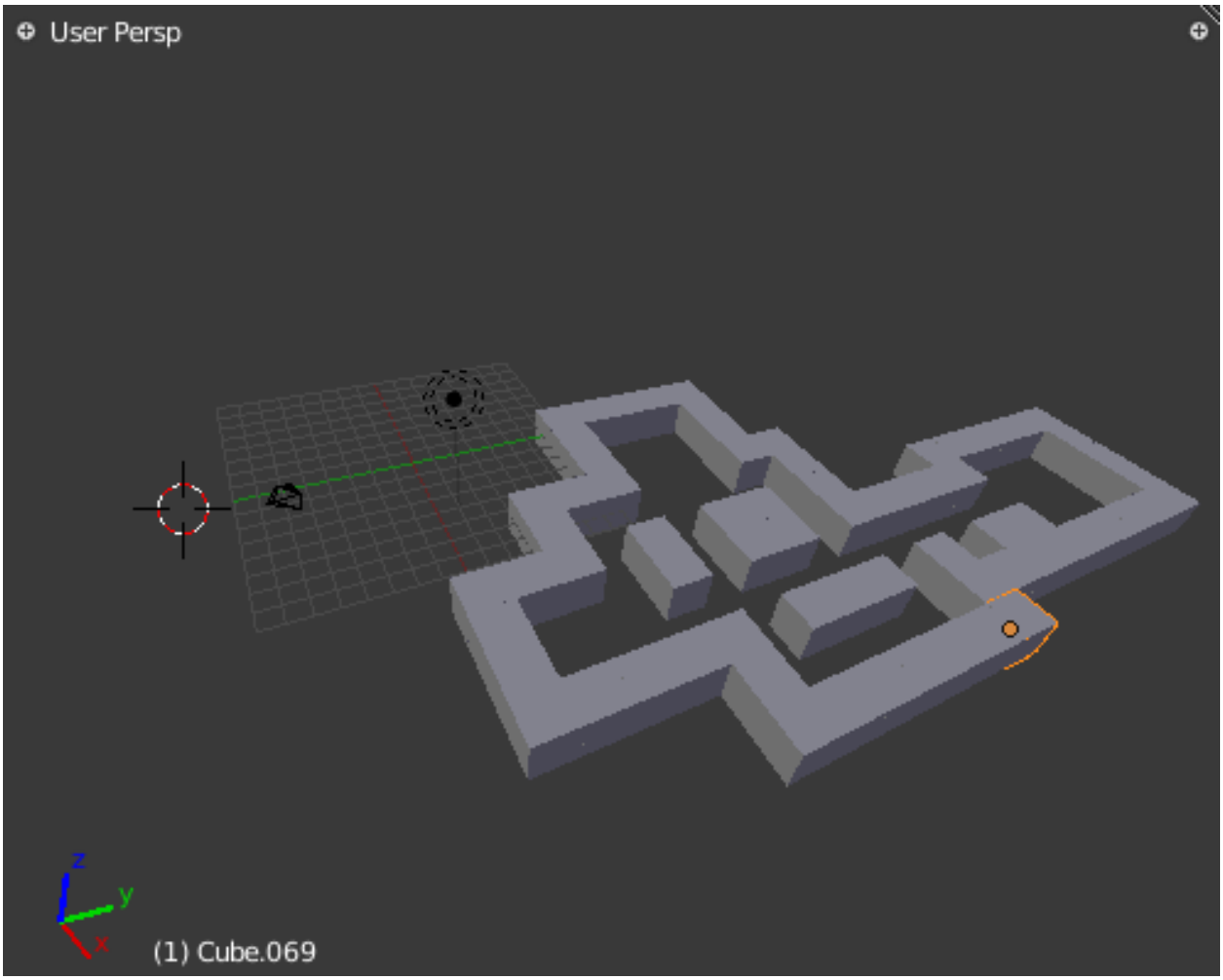
The function is now called with one of the available levels (level_00, level_01,level_02...).

```
if __name__ == '__main__':
    sokobanLevel(level_00)
```

Result of the script



View from top...

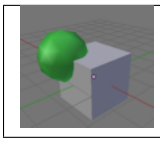


View from an other perspective ...

Material

Material – colored labyrinth

Objectives



So far, our objects are unique grey. Now lets make the new world a little bit colorful. Therefor we are using materials a

Instructions

Tasks

1. Construct your own new level as shown in the last station.
2. Create new types of objects for the character $ and @ (ie. cone, cylinder).

3. Add different colors for every type of object.

Das Labyrinth

Creating the labyrinth is the same procedure as before but in addition we add a material.

```
#! bpy
"""
Name: 'colored_labyrinth.py'
Blender: 269
Group: 'Materials'
Tooltip: 'Generates colorful objects by materials'
"""
import bpy

level_00 = ["####################",
            "#.                 #",
            "#         $        #",
            "#                  #",
            "#                  #",
            "#         @        #",
            "#                  #",
            "#                  #",
            "#                  #",
            "#                  #",
            "####################"]


level_01 = ["    #####           ",
            "    #   #           ",
            "    #$  #           ",
            "  ###  $##          ",
            "  #  $ $ #          ",
            "### # ## #   ###### ",
            "#   # ## #####  ..# ",
            "# $  $          ..# ",
            "##### ### #@##  ..# ",
            "    #     ######### ",
            "    #######         "]

MATERIAL_RED = bpy.data.materials.new('Red Material')


def sokobanLevel(level):
    """Generating a labyrinth with colored cubes"""

    cols = len(level[0])
    rows = len(level)

    for row in range(rows):
        for i in range(cols):
            if level[row][i] == '#':
                bpy.ops.mesh.primitive_cube_add(location=(row*2, i*2, 0))
                obj = bpy.context.object
                setColor(obj, MATERIAL_RED, (1, 0, 0))


def setColor(obj, material, color):
```

```
53      material.diffuse_color = color
54      material.specular_hardness = 200
55      obj.data.materials.append(material)
56
57
58  if __name__ == '__main__':
59      sokobanLevel(level_00)
```

Create a new material

Materials in Blender are distinguished by name and exists independent from objects. Therefor it is possible to put one material to different objects.

```
MATERIAL_RED = bpy.data.materials.new('Red Material')
```

Save object

The new object is allocated to a variable named »*obj*«.

```
    for row in range(rows):
        for i in range(cols):
            if level[row][i] == '#':
                bpy.ops.mesh.primitive_cube_add(location=(row*2, i*2, 0))
                obj = bpy.context.object
                setColor(obj, MATERIAL_RED, (1, 0, 0))
```

Function: setColor

The function *setColor* needs tree parameters: object, material and color.

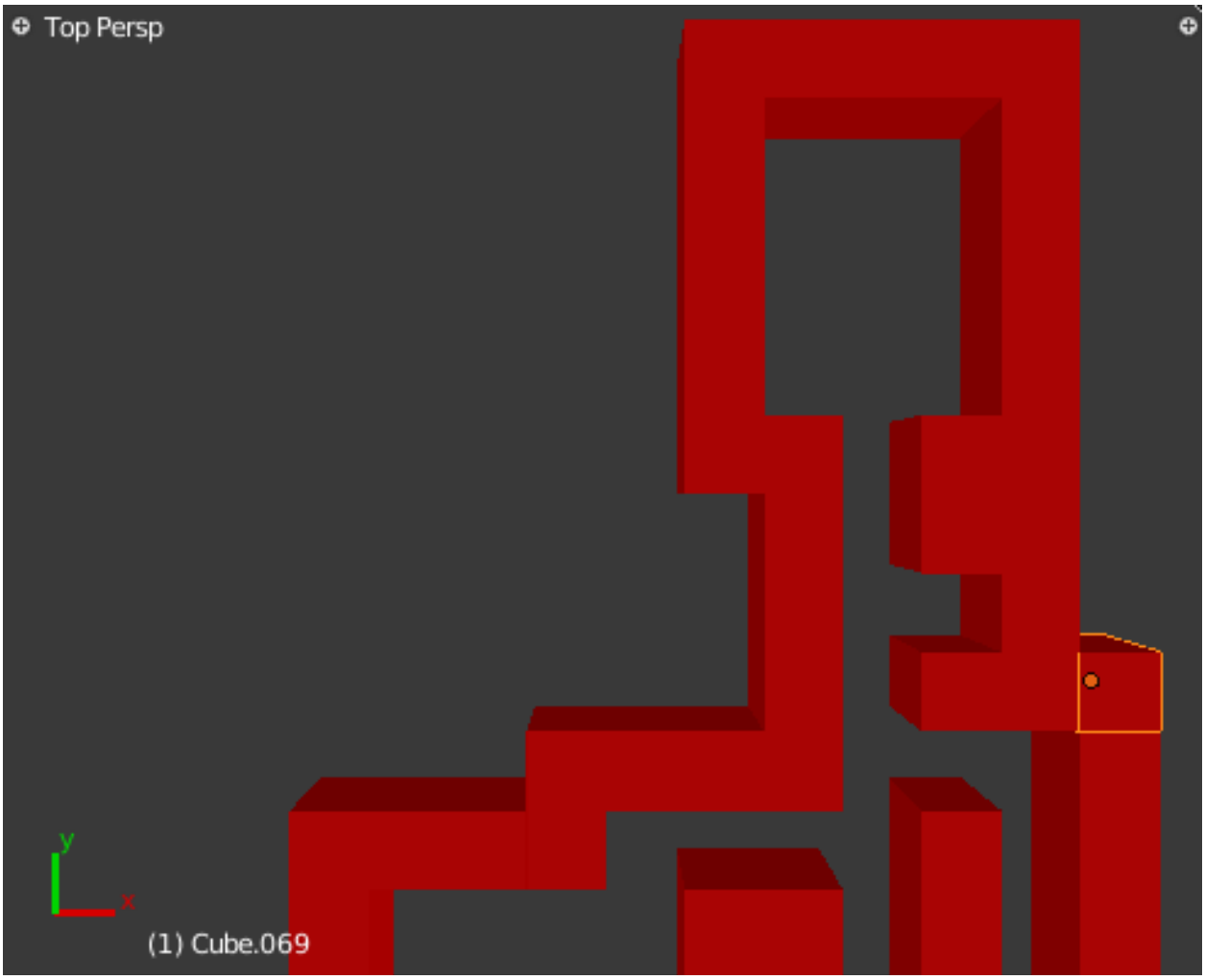
```
                setColor(obj, MATERIAL_RED, (1, 0, 0))
```

Allocate material

Within setColor the parameters are used to change the given object.

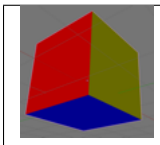
```
def setColor(obj, material, color):
    material.diffuse_color = color
    material.specular_hardness = 200
    obj.data.materials.append(material)
```

Result



Material – colored faces

Objectives

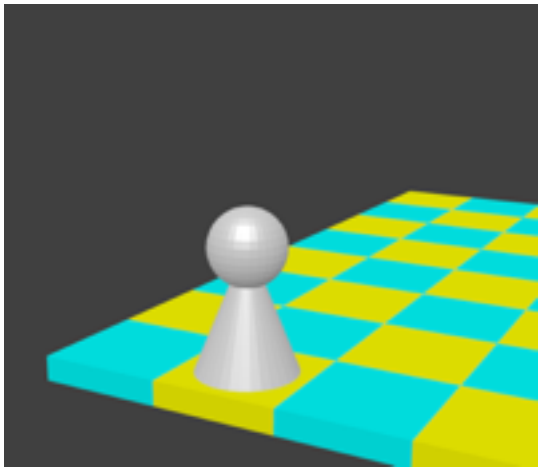


In the last station we applied the material to the hole object. Now we select a few faces to color only parts of the objec

Instructions

Tasks

1. Create a cylinder and change the color of the top and bottom base.
2. Create a chessboard like in this image:



3. Create an set of tokens for the game draughts (or checkers) an the board.

Louis-Léopold Boilly (1761–1845)
Painting of a family game of checkers
(»jeu des dames«)

A colored cube

To be able to color of one or more faces, select them. At first the script:

```
#!bpy
"""
Name: 'colored_cube.py'
Blender: 2.69
Group: 'Material'
Tooltip: 'Colored faces'
"""
import bpy

red = bpy.data.materials.new('Red')
blue = bpy.data.materials.new('Blue')
yellow = bpy.data.materials.new('Yellow')


def setColor(obj, material, color):
    material.diffuse_color = color
    material.specular_hardness = 200
    obj.data.materials.append(material)

def colored_cube():
    """ A three colored cube """

    bpy.ops.mesh.primitive_cube_add(location=(0,0,0))
    ob = bpy.context.object
    ob.name = 'three'

    setColor(ob, red, (1, 0, 0))
    setColor(ob, yellow, (1, 1, 0))
    setColor(ob, blue, (0, 0, 1))

    # apply the colors
    for face in ob.data.polygons:
        face.material_index = face.index % 3

if __name__ == '__main__':
    bpy.ops.object.select_by_type(type='MESH')
    bpy.ops.object.delete()
    colored_cube()
```

Name the colors

Every color should have a name. This is the first Step.

```
red = bpy.data.materials.new('Red')
blue = bpy.data.materials.new('Blue')
yellow = bpy.data.materials.new('Yellow')
```

Add the materials

This is little bit tricky, to add a color for each face. Step by step:

- All faces of an object (a cube has six) are saved in a list of polygons.

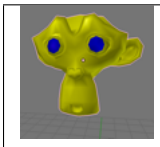
- In a for loop wie iterate over this list.
- The index number from the list is divided by three with the modulo operator. So we get three possible results 0, 1 or 2.
- Each number is connected with a color.
- Finally the color number is assigned to the material index, because materials are also managed with an index.

```
    # apply the colors
    for face in ob.data.polygons:
        face.material_index = face.index % 3
```

hint In old tutorials before 2.68 faces are selected from a list named *faces*. Now the new list and name is *polygons*.

Find faces

Objectives



Often an object is collection of huge number of polygons, it is not easy to find the index values for a spezific area or nu

Instructions

Tasks

1. Test the script and discover how the functions work.
2. Change the color of eyes from »Suzanne«.
3. Try to colorize the mouth of »Suzanne«.
4. Try colorize a part of a figure that you created yourself.

The script

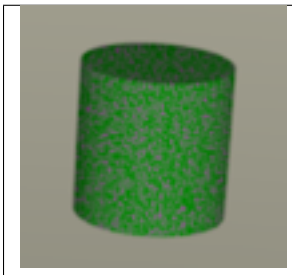
```
#!bpy
"""
Name: 'show_faces.py'
Blender: 2.68
Group: 'Material'
Tooltip: 'Materialien an einzelne Flächen zuweisen'
"""
import bpy

red = bpy.data.materials.new('Red')
blue = bpy.data.materials.new('Blue')
yellow = bpy.data.materials.new('Yellow')


def setColor(obj, material, color):
    material.diffuse_color = color
    material.specular_hardness = 200
```

```
    obj.data.materials.append(material)


def monkey():
    """ a new monkey """

    bpy.ops.mesh.primitive_monkey_add(location=(0, 0, 1), rotation=(0, 0, .5))
    ob = bpy.context.object
    ob.name = 'Suzanne'

    setColor(ob, red, (1, 0, 0))
    setColor(ob, yellow, (1, 1, 0))
    setColor(ob, blue, (0, 0, 1))

    # colorize the whole object
    for poly in ob.data.polygons:
        poly.material_index = 1


def coloredEyes(name):
    bpy.context.scene.objects.active = bpy.data.objects[name]
    me = bpy.context.object

    # changes are visible after switch to the edit mode
    bpy.ops.object.mode_set(mode='EDIT')
    bpy.ops.object.mode_set(mode='OBJECT')
    polygons_augen = [48, 49, 50, 51, 52, 53, 54, 55,
                      56, 57, 58, 59, 60, 61, 62, 63]
    # blue Eyes
    for poly in me.data.polygons:
        if poly.index in polygons_augen:
            poly.material_index = 2
            print(poly.index, polygons_augen)


def getIndexOfFaces(name):
    """ Print the index of faces to the console

        - Switch to the edit-Mode
        - select faces
        - the output ist shown in the console
    """
    bpy.context.scene.objects.active = bpy.data.objects[name]
    bpy.ops.object.mode_set(mode='EDIT')
    bpy.ops.object.mode_set(mode='OBJECT')
    me = bpy.context.object.data
    for poly in me.polygons:
        if poly.select:
            print("Polygon index: %d, length: %d" % (poly.index,
                                                     poly.loop_total))
    bpy.ops.object.mode_set(mode='EDIT')


if __name__ == '__main__':
    # collect the object names
    obj_names = []
    for obj in bpy.context.scene.objects:
        obj_names.append(obj.name)
```

```
    # if no Suzanne is in the list, create one
    if not obj_names.count("Suzanne"):
        monkey()
    # switch in the edit mode (3D View) and selct one ore more faces
    # selected index nummers can be found on the command line
    getIndexOfFaces('Suzanne')

    # If you put the printed index values in a function
    # the process is repeatable

    coloredEyes('Suzanne')
```

Materials/Textures – procedural

Objectives



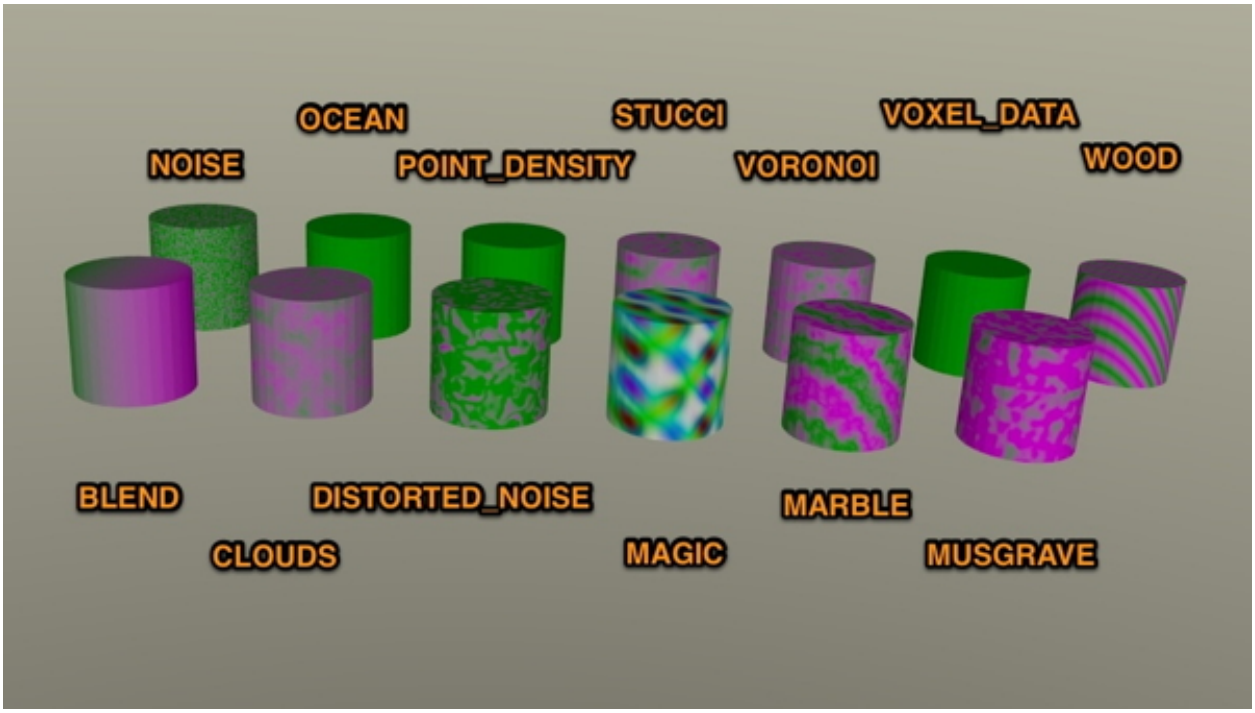
Procedural textures are explained in this station. It is only the starting point to this complex topic.

Instructions

Tasks

1. Run the script and check the functionality of the functions.
2. Change the parameters of a texture.
3. Set material and texture to a new object.

All types in one image – the mapping types will be shown in a separate station. On the picture you can see the names of the textures and how the rendered objects looks like.



Code used to render the image

```
#!bpy
"""
Name: 'material_procedural.py'
Blender: 2.69
Group: 'Materials and Textures'
Tooltip: 'Texturen procedurale'
"""

import bpy


def materialcheck(obj, materialtype):
    """ Put a procedural textur on a object."""

    # used names
    matname = "mat" + materialart
    texname = "tex" + materialart

    # new material
    material = bpy.data.materials.new(matname)
    material.diffuse_color = (0, .5, 0)
    obj.data.materials.append(material)

    # new texture
    textur = bpy.data.textures.new(texname, type=materialart)

    # lits all properties and methods of a texture
    # print(dir(textur))

    # connect texture with material
```

```
    bpy.data.materials[matname].texture_slots.add()
    bpy.data.materials[matname].active_texture = textur


if __name__ == '__main__':

    # switch to object mode if edit mode is activ
    if bpy.ops.object.mode_set.poll():
        bpy.ops.object.mode_set(mode='OBJECT')

    # clear the scene
    bpy.ops.object.select_by_type(type='MESH')
    bpy.ops.object.delete()
    # all names of procedurale texturen, without IMAGE and LANDSCAPE
    materials = ['BLEND',
                 'CLOUDS',
                 'DISTORTED_NOISE',
                 'MAGIC',
                 'MARBLE',
                 'MUSGRAVE',
                 'NOISE',
                 'OCEAN',
                 'POINT_DENSITY',
                 'STUCCI',
                 'VORONOI',
                 'VOXEL_DATA',
                 'WOOD']

    x, y, z = 1, 0, 1
    for i in materials:
        # new line in the middle of the list
        if z % 7 == 0:
            y = 4
            x = 1
        bpy.ops.mesh.primitive_cylinder_add(location=(x, y, 0))
        obj = bpy.context.scene.objects.active

        obj.name = 'obj-%0.2d' % (z)
        obj = bpy.context.scene.objects['obj-%0.2d' % (z)]

        x += 3
        z += 1
        materialcheck(obj, materialtype=i)
```

hint Run the script, set the camera on the right position and start the menu sequence: Render » Render Image.

Every texture has more or less parameters. In the property panel all parameters are changeable. The names of parameters are printed with the *dir* command.

```
print(dir(textur))
```

Output example for the texture »WOOD«:

```
['__doc__', '__module__', '__slots__', 'animation_data',
 'animation_data_clear', 'animation_data_create', 'bl_rna',
 'color_ramp', 'contrast', 'copy', 'evaluate', 'factor_blue',
 'factor_green', 'factor_red', 'intensity', 'is_library_indirect',
 'is_updated', 'is_updated_data', 'library', 'nabla', 'name',
 'node_tree', 'noise_basis', 'noise_basis_2', 'noise_scale',
 'noise_type', 'rna_type', 'saturation', 'tag', 'turbulence',
 'type', 'update_tag', 'use_color_ramp', 'use_fake_user',
 'use_nodes', 'use_preview_alpha', 'user_clear', 'users',
 'wood_type']
```

Materials/Textures – mapping

Objectives

Mapping is a common method to put pictures on objects. With simple objects it is possible to use a script.

Instructions

Tasks

1. Start the functions and have a look to the resulting output.
2. Replace the images with your own.
3. Alter the code, let one face empty.

Sample images

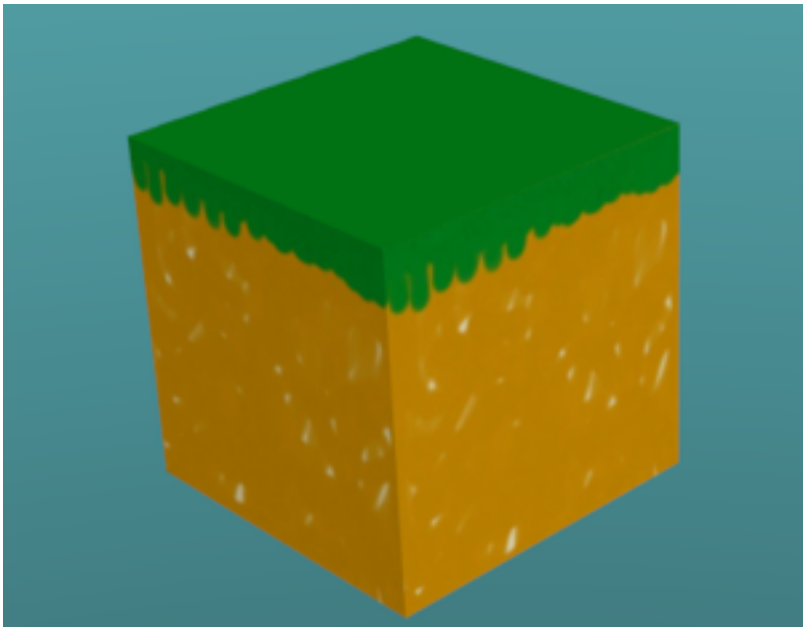
`Download (pictures_uv_mapping.zip)`

Putting images (2D) on a body (3D) is named UV-Mapping. The letters U and V are used, because X, Y and Z always used for positioning an object in a 3D-World. Like materials textures are managed independent form any object. Textures are also never directly connected to an object, instead a material is used. So wie have a chain: texture » material » body.

In our first example a picture is placed on one site of a cube:

The second example is showing different images on all faces of a cube:

Have I heard the term »Mindcraft« (hava a look at exercise tree):



One image on all faces

```
#!bpy
"""
Name: 'mat1.py'
Blender: 2.69
Group: 'Sample'
Tooltip: 'Put one image on each site of a cube'
"""
import bpy
import os


def uvMapperCube(obj):
    """ Put image to an object"""

    # used names
    matname = "matUVMapping"
    texname = "texUVMapping"

    # new material
    if not matname in bpy.data.materials:
        material = bpy.data.materials.new(matname)
        material.diffuse_color = (0, .5, .4)
        obj.data.materials.append(material)

    # new texture
    texUV = bpy.data.textures.new(texname, type="IMAGE")
    image_path = os.path.expanduser("//wuerfelbilder/blume.jpg")
    image = bpy.data.images.load(image_path)
    texUV.image = image

    # connect textur with material
    bpy.data.materials[matname].texture_slots.add()
    bpy.data.materials[matname].active_texture = texUV
```

```
    bpy.data.materials[matname].texture_slots[0].texture_coords = "GLOBAL"
    bpy.data.materials[matname].texture_slots[0].mapping = "CUBE"


def delete_old_stuff():

    # escape edit mode
    if bpy.ops.object.mode_set.poll():
        bpy.ops.object.mode_set(mode='OBJECT')

    # delete all mesh objects
    bpy.ops.object.select_by_type(type='MESH')
    bpy.ops.object.delete()

    # delete all materials
    for i in bpy.data.materials.values():
        bpy.data.materials.remove(i)

    # delete all textures
    for i in bpy.data.textures.values():
        bpy.data.textures.remove(i)

    # delete all images
    for i in bpy.data.images.values():
        # delete image path, this is only possible without a user
        i.user_clear()
        # delete all, except »Render Result«
        if i.name != "Render Result":
            bpy.data.images.remove(i)

if __name__ == "__main__":
    delete_old_stuff()
    bpy.ops.mesh.primitive_cube_add(location=(0, 0, 0))
    obj = bpy.context.scene.objects.active
    obj.name = 'image-as-uvmapping'
    obj = bpy.context.scene.objects['image-as-uvmapping']
    uvMapperCube(obj)
```

One image on each Face

```
#!bpy
"""
Name: 'mat2.py'
Blender: 2.69
Group: 'Sample'
Tooltip: 'Put six different images to cube'
"""
import bpy
import os


def uvMapperCube(obj):
    """ Put images to an object"""

    # used names
    namen = {'blume.jpg': ('mat01', 'tex01'),
             'boot.jpg': ('mat02', ' tex02'),
```

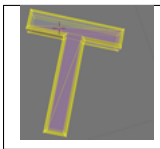
```
            'jueterbog.jpg': ('mat03', 'tex03'),
            'kopf.jpg': ('mat04', 'tex04'),
            'moster.jpg': ('mat05', 'tex05'),
            'telefon.jpg': ('mat06', 'tex06'),
            #'warnung.jpg': ('mat07', 'tex07')
            }

    for i in namen.keys():

        # new material
        matname = namen[i][0]
        if not matname in bpy.data.materials:
            material = bpy.data.materials.new(matname)
            material.diffuse_color = (0, .5, .4)
            obj.data.materials.append(material)

        # new texture
        texUV = bpy.data.textures.new(namen[i][1], type="IMAGE")
        image_path = os.path.expanduser("//wuerfelbilder/{}".format(i))
        image = bpy.data.images.load(image_path)
        texUV.image = image

        # connect texture with material
        bpy.data.materials[matname].texture_slots.add()
        bpy.data.materials[matname].active_texture = texUV
        bpy.data.materials[matname].texture_slots[0].texture_coords = "ORCO"
        bpy.data.materials[matname].texture_slots[0].mapping = "CUBE"

    # apply one material to one face
    for face in obj.data.polygons:
        face.material_index = face.index


def delete_old_stuff():

    # escape edit mode
    if bpy.ops.object.mode_set.poll():
        bpy.ops.object.mode_set(mode='OBJECT')

    # delete all mesh objects
    bpy.ops.object.select_by_type(type='MESH')
    bpy.ops.object.delete()

    # delete all materials
    for i in bpy.data.materials.values():
        bpy.data.materials.remove(i)

    # delete all textures
    for i in bpy.data.textures.values():
        bpy.data.textures.remove(i)

    # delete all images
    for i in bpy.data.images.values():
        # delete image path, this is only possible without a user
        i.user_clear()
        # delete all, except »Render Result«
        if i.name != "Render Result":
            bpy.data.images.remove(i)
```

```
if __name__ == "__main__":
    if bpy.ops.object.mode_set.poll():
        bpy.ops.object.mode_set(mode='OBJECT')

    delete_old_stuff()

    bpy.ops.mesh.primitive_cube_add(location=(0, 0, 0))
    obj = bpy.context.scene.objects.active
    obj.name = 'image-as-uvmapping'
    obj = bpy.context.scene.objects['image-as-uvmapping']
    uvMapperCube(obj)
```

Effects

Text: Bevel & Extrude

Objectives



Text is often used in opening and closing credits of a film. In this station we create a 3D Version from plain text.

Instructions

Tasks

1. Create your own text.
2. Configure the environment (camera, lightning, ... see also »Configure your world«
3. Add a material to the text.
4. Render the final image.

3D-Effekt

```
#!bpy
"""
Name: 'text.py'
Blender: 2.69
Group: 'Text'
Tooltip: 'Text with 3D-Effekt'
"""

import bpy


def add_a_text(text="hallo"):
    """ Add text """
```

```
    if not 'Text' in bpy.data.objects:
        bpy.ops.object.text_add()

    bpy.data.objects['Text'].data.body = text


def extrudeText():
    """ Text: extrude """

    bpy.context.object.data.extrude = 0.04


def bevelText():
    """ Text: bevel """

    bpy.context.object.data.bevel_depth = 0.02
    bpy.context.object.data.bevel_resolution = 8


def text2mesh():
    """ Text: transform to a mesh """

    # ATTENTION: Characters are not editable any more!
    bpy.ops.object.convert(target='MESH', keep_original=False)


if __name__ == "__main__":
    bpy.ops.object.select_by_type(type='MESH')
    bpy.ops.object.delete()
    add_a_text("Hallo Du Pappnase")
    bpy.context.scene.objects.active = bpy.data.objects['Text']
    extrudeText()
    bevelText()
    #text2mesh()
```

Example

This is one version, experiment with the control elements in the property window of Blender.


Hallo Du Pappnase
Hallo Du Pappnase

CHAPTER 4

Blender Extended

If a project grows up, you need technics to manage objects and files. Two of them are shown in the next stations. Modules are useful to split the code in smaller units. Classes are useful to follow an object oriented paradigm while programming applications.

Modules

Module I

Objectives

Large projects can no longer manage them in a file. The source code will quickly become confusing. Ther

Instructions

Tasks

1. Move parts of the source code in a second file (levels.py) from.
2. Import the outsourced parts.
3. Check whether the program behaves, as previously working.
4. Add a third level in the file levels.py.

One module for levels

To separate level definitions from the code that creates the maze we create two files. The first one is named *levels.py*. From now on, a game designer could create new levels, while a programmer cares about the game logic.

```
#! bpy
"""
Name: 'levels.py'
Blender: 269
Group: 'Modules'
Tooltip: 'Maze definitions'
"""

level_00 = ["####################",
            "#.                 #",
            "#         $         #",
            "#                   #",
            "#                   #",
            "#         @         #",
            "#                   #",
            "#                   #",
            "#                   #",
            "#                   #",
            "####################"]


level_01 = ["    #####           ",
            "    #   #           ",
            "    #$  #           ",
            "  ###  $##          ",
            "  #  $ $ #          ",
            "### # ## #   ###### ",
            "#   # ## #####  ..# ",
            "# $  $         ..# ",
            "##### ### #@##  ..# ",
            "    #     ######### ",
            "    #######         "]
```

The main module

Reusing code from other files (other authors/programmers) is achieved with the *import* statement.

```
#! bpy
"""
Name: 'sokoban.py'
Blender: 269
Group: 'Modules'
Tooltip: 'Creating a game called Sokoban and using modules'
"""
import bpy
from levels import level_00
from levels import level_01


def sokobanLevel(level):
    """Create a maze with cubes"""

    cols = len(level[0])
    rows = len(level)

    for row in range(rows):
        for i in range(cols):
```

```
21              if level[row][i] == '#':
22                  bpy.ops.mesh.primitive_cube_add(location=(row*2, i*2, 0))
23
24  if __name__ == '__main__':
25      sokobanLevel(level_00)
26      sokobanLevel(level_01)
```

Errors

Again and again, errors occur and it is important to find the cause. For example a small typo occurred. The errors are always shown as a **traceback** in the console.

```
Traceback (most recent call last):
File "/Users/.../sokoban.py", line 10, in <module>
ImportError: cannot import name Level_01
Error: Python script fail, look in the console for now...
```

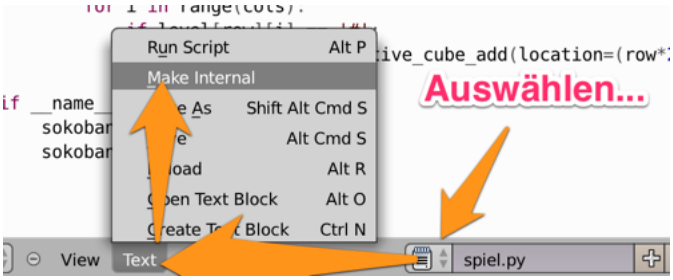
The file path in this Traceback is shorten. Important is the line number, often indicating the main cause of the error. Also important is the type of error, in this case the *ImportError*, because Python is case-sensitive. So the Error will disappear, if you replace *L* with *l*.

Connecting Python scripts with a Blender file

The classical way to find modules in Python is limited in Blender. The easiest workaround is to include the Python scripts into the Blender file.

Step by step

1. Create a new blend file.
2. Load every Python script.
3. For each file call the menu: Text | Make internal.



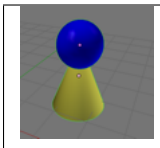
Save the Blender file and all modules are available.

External Editors

The internal Editor is not so bad, but if you like to use your prefered Editor, for example emacs, vi on Linux or PyCharm on Windows, you can edit the external Python scripts. A small icon is indicating, that you have altered the content of the file and you are forced to reload the script, starting with a right click on this icon.

Module II

Objectives

	Now we connect tree Python script to build a complete szene

Instructions

Tasks

1. What are the differences to the modules, used before.
2. Create your own scene and manage the objects in separate modules (files).
3. Add a module for the ground. Let it look like a chessboard.
   See also learning station: *materials*

Module: checker.py

```
#! bpy
"""
Name: 'checker.py'
Blender: 269
Group: 'Modules'
Tooltip: 'Create a checker'
"""
import bpy


def checker(location=(0, 0, 0)):
    '''Checker build from a cone and a sphere'''
```

```
    # save the context
    scn = bpy.context.scene

    # list of names
    parts = ['head', 'body', 'checker']

    # head
    bpy.ops.mesh.primitive_uv_sphere_add(location=(2, 2, .9))
    obj = bpy.context.object
    obj.scale[0] = .6
    obj.scale[1] = .6
    obj.scale[2] = .6
    obj.name = parts[0]

    scn.objects.active = scn.objects[parts[0]]

    # body
    bpy.ops.mesh.primitive_cone_add(location=(2, 2, 0))
    obj = bpy.context.object
    obj.scale[0] = .8
    obj.scale[1] = .8
    obj.name = parts[1]
    scn.objects.active = scn.objects[parts[1]]

    # connecting the parts
    scn.objects[parts[0]].select = True
    scn.objects[parts[1]].select = True
    bpy.ops.object.join()

    obj.name = parts[2]
    obj.location = location

if __name__ == '__main__':
    bpy.ops.object.select_by_type(type='MESH')
    bpy.ops.object.delete()
    checker(location=(2, 2, 2))
```

Module: levels.py

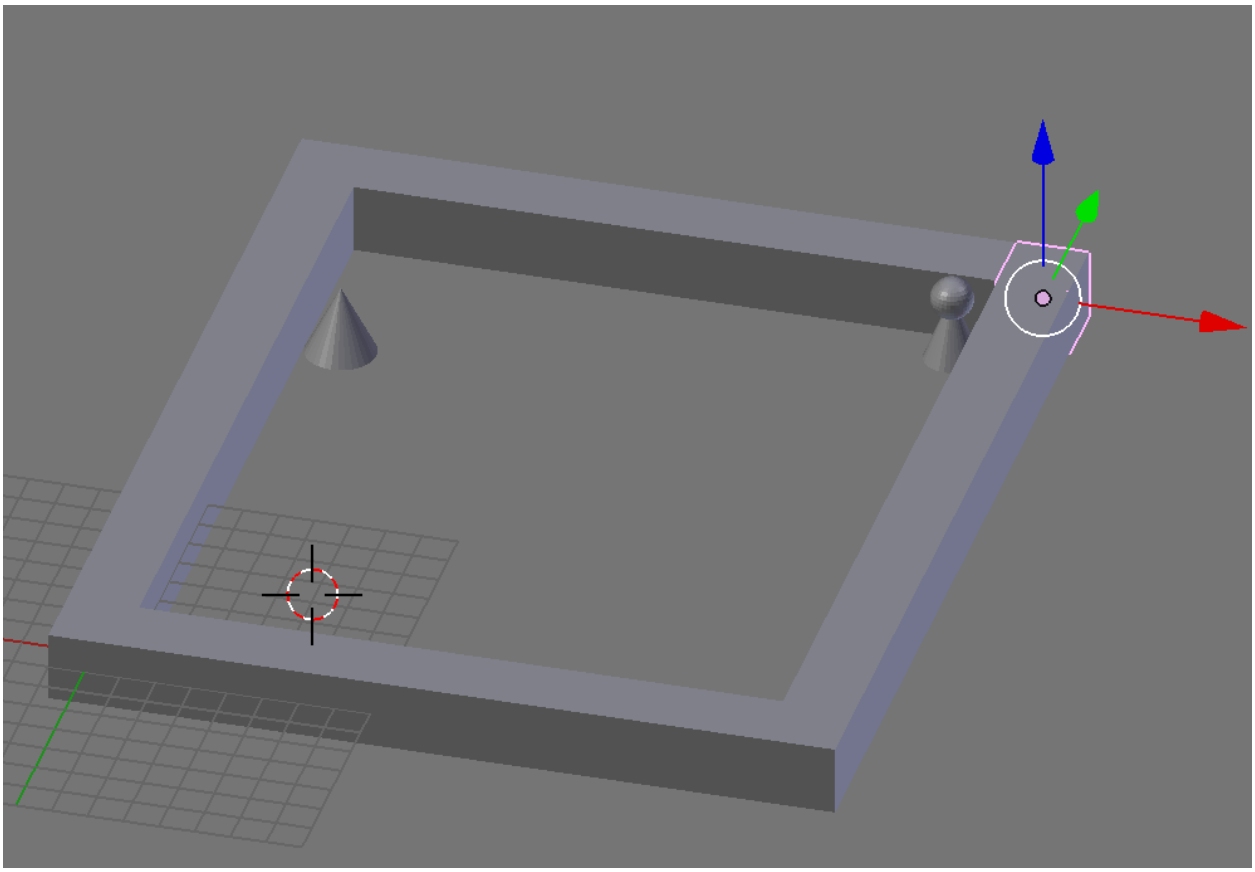
```
#! bpy
"""
Name: 'levels.py'
Blender: 269
Group: 'Modules'
Tooltip: 'Maze definitions'
"""

level_00 = ["####################",
            "#.                 #",
            "#        $         #",
            "#                  #",
            "#                  #",
            "#        @         #",
            "#                  #",
            "#                  #",
            "#                  #",
            "#                  #",
```

```
            "####################"]


level_01 = ["     #####            ",
            "     #   #            ",
            "     #$  #            ",
            "   ###  $##           ",
            "   #  $ $ #           ",
            "### # ## #   ###### ",
            "#    # ## #####  ..# ",
            "# $  $          ..# ",
            "##### ### #@##  ..# ",
            "     #     ######### ",
            "     #######          "]
```

Main programm: sokoban.py

```
#! bpy
"""
Name: 'sokoban_m2.py'
Blender: 269
Group: 'Modules'
Tooltip: 'Create a maze'
"""
import bpy
import levels
from checker import checker


def sokobanLevel(level):
    """Create a level managed by modules"""

    cols = len(level[0])
    rows = len(level)

    for row in range(rows):
        for i in range(cols):
            coords = (row*2, i*2, 0)
            if level[row][i] == '#':
                bpy.ops.mesh.primitive_cube_add(location=coords)
            elif level[row][i] == '@':
                checker(location=coords)
            elif level[row][i] == '$':
                bpy.ops.mesh.primitive_cone_add(location=coords)

if __name__ == '__main__':
    bpy.ops.object.select_by_type(type='MESH')
    bpy.ops.object.delete()
    sokobanLevel(levels.level_00)
```

Result



Classes

Class: Simple

Objectives

Classes are part of a programming paradigma. We use Python which is a object oriented programming language, With

Instructions

Tasks

1. Read the text about objectoriented programming on wikipedia.
2. Build and test the class shown in this station.
3. Replace the text with your own.
4. Mix the concept of a class with positioning a Text in the 3D-View (*Text in 3D*).

Simple class

This example is taken from: Blender API.

The code:

```
import bpy


class HelloWorldOperator(bpy.types.Operator):
    bl_idname = "wm.hello_world"
    bl_label = "Minimal Operator"

    def execute(self, context):
        print("Hello World")
        return {'FINISHED'}

bpy.utils.register_class(HelloWorldOperator)

# test call to the newly defined operator
bpy.ops.wm.hello_world()
```

Notes about the code (step by step)

A class definition is introduced by the keyword class, followed by a self-selected name and finally a pair of parentheses. As in our case, the class can also inherit properties of another class, in our example *bpy.types.Operator*. A colon marks the start of the block, the next line(s) must be indented.

```
class HelloWorldOperator(bpy.types.Operator):
```

Two variables are defined that. This ist necessary for the integration of our class in the Blender system.

```
    bl_idname = "wm.hello_world"
    bl_label = "Minimal Operator"
```

No class without methods. A method is introduced by the keyword *def* and the first parameter used, is *self*. With *self* the connection between function and class is defined. In our example, the second parameter *context* is mandatory and used by Blender. The function does not really do much. Finally a typical blender statement, gives the value *{'FINISHED'}* back.

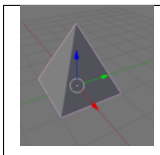
```
    def execute(self, context):
        print("Hello World")
        return {'FINISHED'}
```

Finally, the new class is registered in the Blender system and also a trial statement is called. The result can be seen in the console. There, the *Hello World* should be printed as expected.

```
bpy.utils.register_class(HelloWorldOperator)

# test call to the newly defined operator
bpy.ops.wm.hello_world()
```

Class: Mesh I

Objectives



In an object-oriented world, parts of the model are mapped into classes. This has many advantages and the most impor

Instructions

Tasks

1. Repeat the sub-steps shown here to create a class until you have a working version.
2. Create a class of its own »primitive«, eg. a wedge or the base form of a hex key.
3. Add a material to the class, which is given to the new object.

Design data for a mesh

We start with the ground of a pyramid. The first step is defining the vertices an there location in a 3D scene. After that, then for each area, the index values of each surface section summarizes (a face is defined). The latter determines whether an area is closed or remains open, if it is not defined.

```
def pyramid_values():
    """
    This function takes inputs and returns vertex and face arrays.
    no actual mesh data creation is done here.
    """
    verts =  [(-1, +1, 0),
              (+1, +1, 0),
              (+1, -1, 0),
              (-1, -1, 0)]

    faces = [(0, 1, 2, 3),]

 return verts, faces
```

The coordinates of each point are given by the focus of our future pyramid. The whole thing still possible in a clockwise sense.

Todo

Picture with coordinats

Class: AddPyramid

After the keyword *class* follows a self-selected class name *AddPyramid*.

As in the example with the simple class, our class inherits from a class in Blender: *bpy.types.Operator*. We need tree variable, for the correkt integration of our new class into Blender, its menues or lists for selection.

```
class AddPyramid(bpy.types.Operator):
    """Add a simple pyramid mesh"""

    bl_idname = "mesh.primitive_pyramid_add"
    bl_label = "Add Pyramid"
    bl_options = {'REGISTER', 'UNDO'}
```

Properties

Also tree parameters are nessesary to place a new pyramid in a 3D-World.

```
    """Add a simple pyramid mesh"""

    bl_idname = "mesh.primitive_pyramid_add"
    bl_label = "Add Pyramid"
    bl_options = {'REGISTER', 'UNDO'}
```

Method: execute

The methode *execute* is using the coordintate given by the function *pyramid_values()*. The values are transfered to locale variables: *verts_loc* and *faces*.

Now two new objects are created mit the operator *new()*.

- one *Mesh* for the coordinates
- one bmesh for ...

Todo

bmash description

```
    def execute(self, context):

        verts_loc, faces = pyramid_values()
        mesh = bpy.data.meshes.new("Pyramid")
        bm = bmesh.new()
```

Coordinates second part

With two for loops the coordintes are added to the new objects.

```
        for v_co in verts_loc:
            bm.verts.new(v_co)

        for f_idx in faces:
            bm.faces.new([bm.verts[i] for i in f_idx])
```

Create objects

Afte the preparation, a new object can be created and placed on a given location.

```
        bm.to_mesh(mesh)
        mesh.update()
        self.location = (0, 0, 0)
```

Display the object

With *object_utils* the object gets visible in a szene. If all works fine, the return value *{'FINISHED'}* is given back.

```
        # set the new object into the szene
        from bpy_extras import object_utils
        object_utils.object_data_add(context, mesh, operator=self)

        return {'FINISHED'}
```

Register/Unregister

Blender needs a registration and unregistreation command for the new class.

```
def register():
    bpy.utils.register_class(AddPyramid)


def unregister():
    bpy.utils.unregister_class(AddPyramid)
```

Test run

Lets test the creation of a new pyramid, so far we only have the base.

```
if __name__ == "__main__":
    register()

    # create a instanz of a pyramid
    bpy.ops.mesh.primitive_pyramid_add()
```

With the next step, we will finish the pyramid.

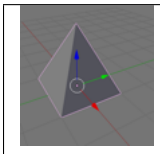
The code for the class

```
import bpy
import bmesh
from bpy.props import BoolProperty, FloatVectorProperty


def pyramid_values():
    """
    This function takes inputs and returns vertex and face arrays.
    no actual mesh data creation is done here.
    """
    verts = [(-1, +1, 0),
```

```
              (+1, +1, 0),
              (+1, -1, 0),
              (-1, -1, 0)]

    faces = [(0, 1, 2, 3), ]
    return verts, faces


class AddPyramid(bpy.types.Operator):
    """Add a simple pyramid mesh"""

    bl_idname = "mesh.primitive_pyramid_add"
    bl_label = "Add Pyramid"
    bl_options = {'REGISTER', 'UNDO'}

    # generic transform props
    view_align = BoolProperty(name="Align to View",
                              default=False)
    location = FloatVectorProperty(name="Location",
                                   subtype='TRANSLATION')
    rotation = FloatVectorProperty(name="Rotation",
                                   subtype='EULER')

    def execute(self, context):

        verts_loc, faces = pyramid_values()
        mesh = bpy.data.meshes.new("Pyramid")
        bm = bmesh.new()

        for v_co in verts_loc:
            bm.verts.new(v_co)

        for f_idx in faces:
            bm.faces.new([bm.verts[i] for i in f_idx])

        bm.to_mesh(mesh)
        mesh.update()
        self.location = (0, 0, 0)

        # set the new object into the szene
        from bpy_extras import object_utils
        object_utils.object_data_add(context, mesh, operator=self)

        return {'FINISHED'}


def register():
    bpy.utils.register_class(AddPyramid)


def unregister():
    bpy.utils.unregister_class(AddPyramid)

if __name__ == "__main__":
    register()

    # create a instanz of a pyramid
    bpy.ops.mesh.primitive_pyramid_add()
```

Class: Mesh II

Objectives



Now we will finish the pyramid class

Instructions

Tasks

1. Repeat the sub-steps shown here to create a class until you have a working version.
2. Create a class of its own »primitive«, eg. a wedge or the base form of a hex key.
3. Add a material to the class, which is given to the new object.

Design data for a mesh

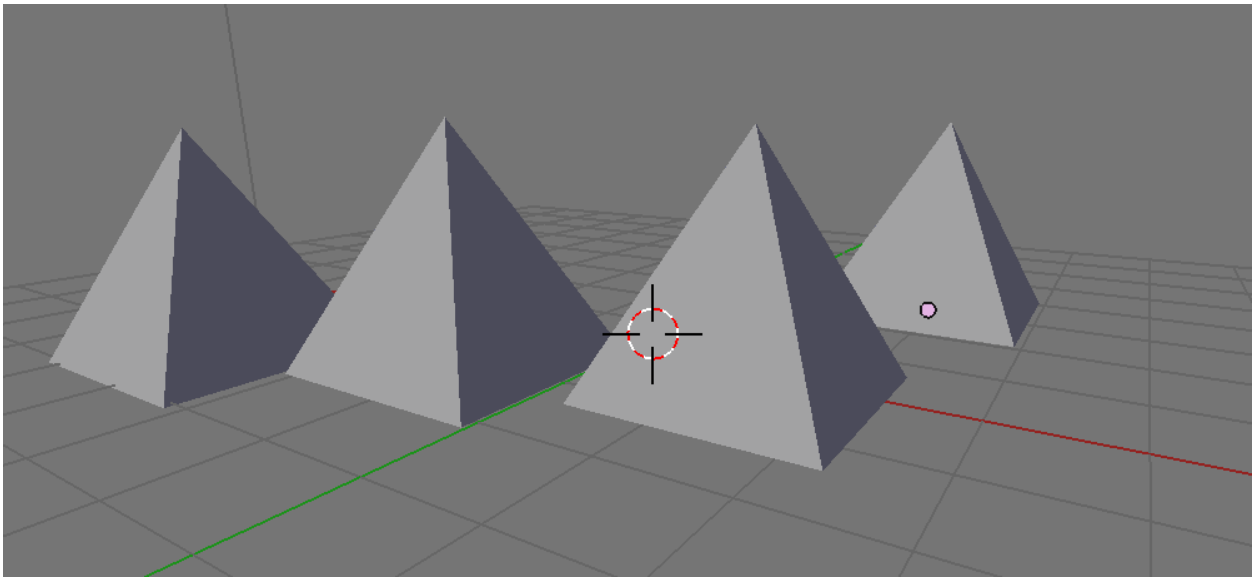
Define the tip of the pyramid and defining the faces, is the last step.

```
def pyramid_values():
    """
    This function takes inputs and returns vertex and face arrays.
    no actual mesh data creation is done here.
    """
    verts = [(-1, +1, 0),
             (+1, +1, 0),
             (+1, -1, 0),
             (-1, -1, 0),
             (0, 0, +2)]

    faces = [(0, 1, 2, 3),
             (0,1,4),
             (1,2,4),
             (2,3,4),
             (3,0,4)]

    return verts, faces
```

Defining the faces is done clockwise and remember the indes starts with 0.

The result

Now you are able to create as much pyramids as you like.



The final version of the class

```
import bpy
import bmesh
from bpy.props import BoolProperty, FloatVectorProperty


def pyramid_values():
    """
    This function takes inputs and returns vertex and face arrays.
    no actual mesh data creation is done here.
    """
    verts = [(-1, +1, 0),
             (+1, +1, 0),
             (+1, -1, 0),
             (-1, -1, 0),
             (0, 0, +2)]

    faces = [(0, 1, 2, 3),
             (0, 1, 4),
             (1, 2, 4),
             (2, 3, 4),
             (3, 0, 4)]

    return verts, faces


class AddPyramid(bpy.types.Operator):
    """Add a simple pyramid mesh"""

    bl_idname = "mesh.primitive_pyramid_add"
    bl_label = "Add Pyramid"
    bl_options = {'REGISTER', 'UNDO'}

    # generic transform props
    view_align = BoolProperty(name="Align to View",
                              default=False)
```

```
    location = FloatVectorProperty(name="Location",
                                   subtype='TRANSLATION')

    def execute(self, context):

        verts_loc, faces = pyramid_values()
        mesh = bpy.data.meshes.new("Pyramid")
        bm = bmesh.new()

        for v_co in verts_loc:
            bm.verts.new(v_co)

        for f_idx in faces:
            bm.faces.new([bm.verts[i] for i in f_idx])

        bm.to_mesh(mesh)
        mesh.update()
        self.location = (0, 0, 0)

        # neues Objekt in die Szene setzen
        from bpy_extras import object_utils
        object_utils.object_data_add(context, mesh, operator=self)

        return {'FINISHED'}


def register():
    bpy.utils.register_class(AddPyramid)


def unregister():
    bpy.utils.unregister_class(AddPyramid)

if __name__ == "__main__":
    register()

    # create a instanz of a pyramid
    bpy.ops.mesh.primitive_pyramid_add()
```

CHAPTER 5

Blender at School

The lessons should be exciting, informative and challenging. Is Blender the right choice? The interplay between a 3D world and the training at school will be shown. Have a look at some examples.

Blender & Python at school

Objectives

Each of the following lessons takes a topic from the school curriculum and uses a 3D world visualization

Instructions

Tasks

1. Have a look at the learning units in this section.
2. Do you have a new idea?
3. Start with Python and Blender if you are not familiar with programming and/or the Python programming language.
4. Use the available examples.
5. Create your own exercises.

Ideas

- Representation of molecules (chemistry) [finished/available]
- Laws of geometry (mathematics) [idea]
- Examples of plants (biology) [idea]

- Free fall (physics) [idea]
- Inclined plane (physics) [idea]
- Programming constructs like object oriented programming, control flow, modules (informatics) [finished, look at section Blender and Python basics]

Combination of two and more subjects

- language and informatics [idea]
- Your own ideas ...

Examples/Links

Adriana Mikolaskova – Blender at School – Art and Design (in german)

László Sátori – Virtual electronics lab made in blender.

Video-Link: http://youtu.be/RQH1tnzD9rQ

Chris Wilmer produced a video about fuel storage - how filling fuel tanks with sand and other types of crystals can vastly increase the amount of storage inside for gaseous fuels. This video has won a prize in the 2011 International Science & Engineering Visualization Challenge.

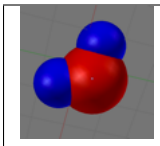
Video-Link: http://youtu.be/QaKSekjAnqY

Chemestry: molecules

Author(s) Joren Retel & Peter Koppatz

Tags Chemestry, molecules

Objectives

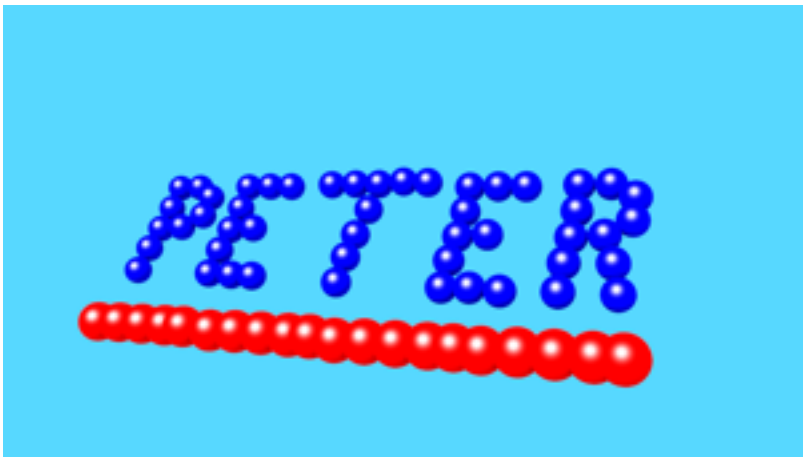


Atoms and molecules – an exiting use case for 3D. With this material it is possible to create molecules, other experime

Instructions

Tasks

1. Download: `Zipfile with example data (PDB)`
2. Open the blend file and have a look at the examples.
3. Add new molecules to the collection.
4. Let students construct a molecule from a given set of atoms.
5. Working with atoms should be fun, how about constructing a name from a set of atoms.



If a real molecule is intergrated in the name, as student can earn some extra points.

Hint All mentioned files are available as a collection within the *blend*-file.

PDB-Format

We use the structure of a PDB-File. The documentation about the PDB format is available at:

- http://www-lehre.inf.uos.de/~okrone/DIP/node25.html

or the short version in the following table:

COLUMNS	DATA TYPE	CONTENTS
1 - 6	Record name	“ATOM”
7 - 11	Integer	Atom serial number.
13 - 16	Atom	Atom name.
17	Character	Alternate location indicator.
18 - 20	Residue name	Residue name.
22	Character	Chain identifier.
23 - 26	Integer	Residue sequence number.
27	AChar	Code for insertion of residues.
31 - 38	Real(8.3)	Orthogonal coordinates for X in Angstroms.
39 - 46	Real(8.3)	Orthogonal coordinates for Y in Angstroms.
47 - 54	Real(8.3)	Orthogonal coordinates for Z in Angstroms.
55 - 60	Real(6.2)	Occupancy.
61 - 66	Real(6.2)	Temperature factor (Default = 0.0).
73 - 76	LString(4)	Segment identifier, left-justified.
77 - 78	LString(2)	Element symbol, right-justified.
79 - 80	LString(2)	Charge on the atom.

Part of a PDB file:

```
         1         2         3         4         5         6         7         8
12345678901234567890123456789012345678901234567890123456789012345678901234567890
ATOM    145  N   VAL A  25      32.433  16.336  57.540  1.00 11.92      A1   N
ATOM    146  CA  VAL A  25      31.132  16.439  58.160  1.00 11.85      A1   C
ATOM    147  C   VAL A  25      30.447  15.105  58.363  1.00 12.34      A1   C
ATOM    148  O   VAL A  25      29.520  15.059  59.174  1.00 15.65      A1   O
```

Simple version or teaching

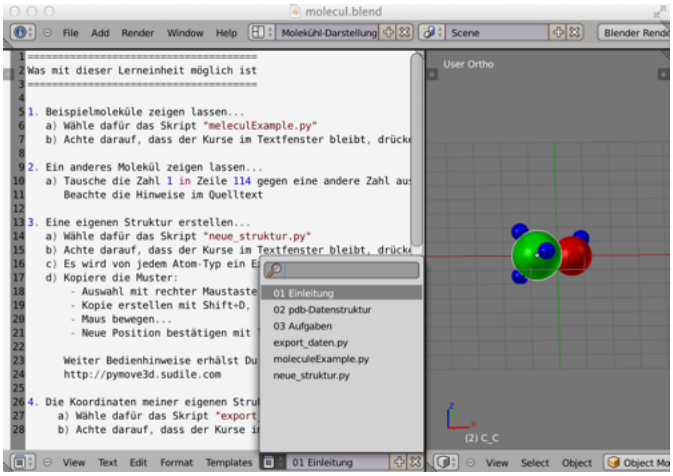
To experiment with this file format we only need the coordinates an the type of atom. So the following simpler structure remains (a water molecule in this example):

```
########## PDB-Struktur(vereinfacht) ##########
# H20 (Wasser, Water)
ATOM                              -0.544  -0.257  -0.228                      H
ATOM                               0.391   0.456   0.228                      H
ATOM                              -0.332   0.44   -0.016                      O
```

Important for the representation is the use of different diameters and colors for each type of atom:

Atom	Diameter	Color
Hydrogen	0,32	blue
Carbon	0,91	green
Knitted fabric	0,75	blue
Phospor	1,06	red
Oxygen	0,73	red
unknown Atom	1,0	black

Content of the Blender file



- 01 Introduction (listing of all possibilities)
- 02 PDB data structure (with examples, used by the standard)
- 03 Tasks
- export_data.py (save the coordinates for a new structure to a file)
- moleculeExample.py (Hydrogen, part of a DNA, methanol ...)
- new_struktur.py (a template with one unit of each type of atom)

More informations about this topic

A extension for Blender, with more possibilities as in this station shown...

http://wiki.blender.org/index.php/Extensions:2.6/Py/Scripts/Import-Export/PDB

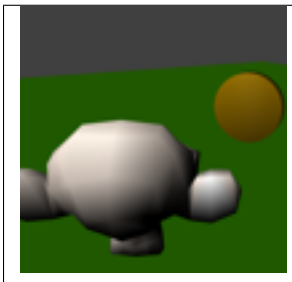
CHAPTER 6

Blender Game Engine

There are two ways to create games with Blender and Python. Firstly the object-oriented way, wherein every object has it's own script, which is executed every frame. Secondly the way, wherein the game runs in a mainloop, which calls the different funcions of the objects (like in pygame). Both use cases are shown in the following sections.

Starting the Blender Game Engine (BGE)

Objectives



Programming games in Blender is possibel in two ways but starting the Game Engine, in short BGE or *bg*

The *bge* module

The *bge* module contains nearly all you need to write a game in Python. But if you write

```
import bge
```

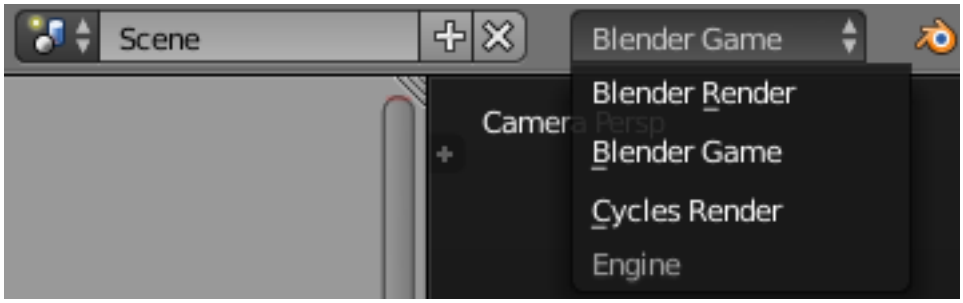
in the interactive interpreter or a script, which you start with »Run Script«, you'll get the Error:

```
ImportError: No module named 'bge'
```

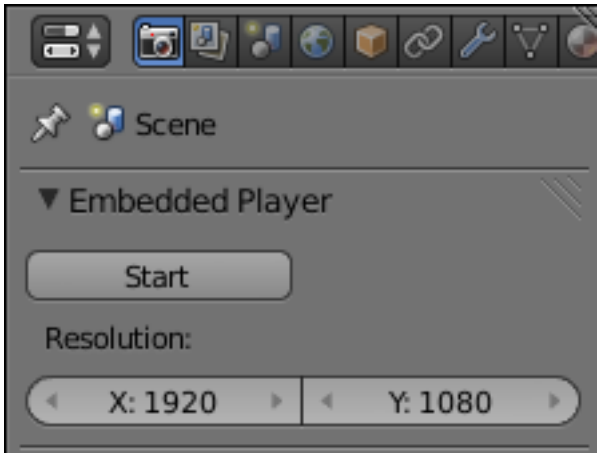
That's because the bge module can only be imported, if the Blender Game Engine is running.

Start BGE the right way

1. Change the Render Type from »Blender Render« to »Blender Game«



2. You can start the Blender Game Engine now by clicking »P« while the cursor is over the »3D-View« or by clicking »Start« in the *Render Properties*.



Game I

Mainloop

Objectives

In this part you learn how to write a little program for the game engine which uses a mainloop.

Instructions

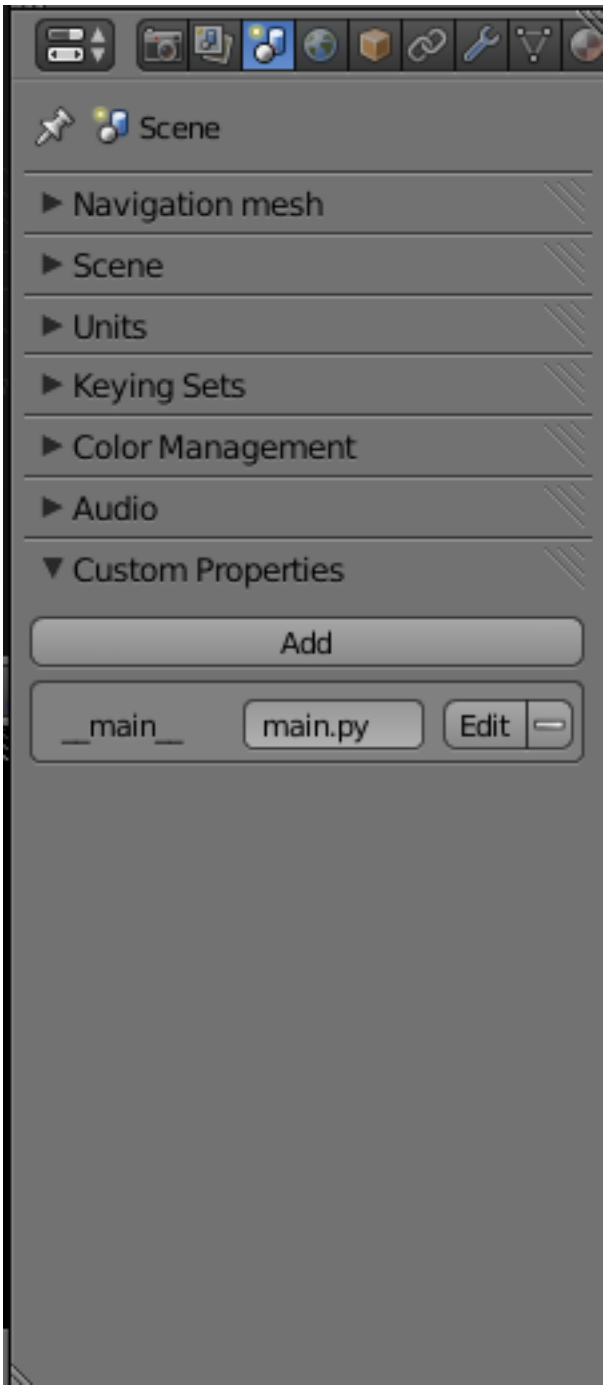
Tasks

1. Copy the example
2. Add keys and functions

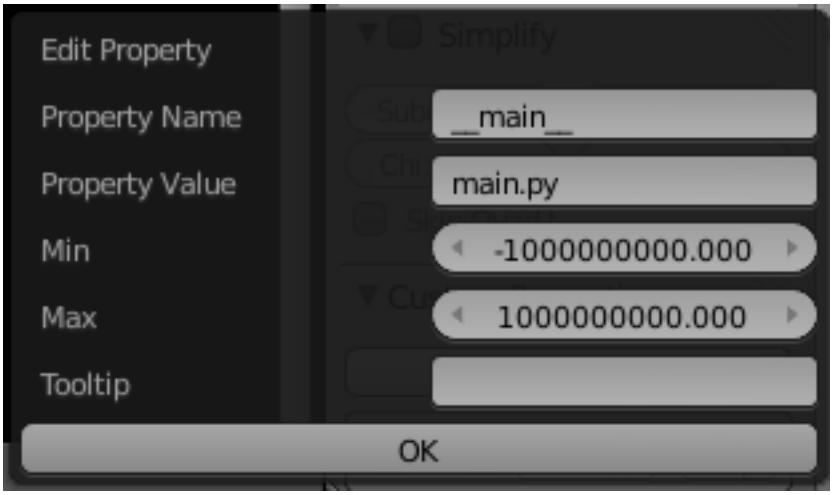
Preparations

1. Open a new blend-file
2. Change the render engine from »Blender Render« to »Blender Game«

3. Change to the scripting area and create the file main.py in the text editor
4. Click now in the scene settings in the menu item *Custom Properties* on »add«



5. Click on »edit« set *Property Name* to »__main__« and *Property Value* to »main.py«



The script

First import the module of the Blender Game Engine, bge.

```
import bge
```

ATTENTION You can't run the script with *Run Script* now, because the module bge can only be imported, if the Game Enigne is active. You can start the program now by clicking »P« while the cursor is over the »3D-View«.

```
#! bpy
import bge
import time
import timeit


# active scene cube
scene = bge.logic.getCurrentScene()
# the object "Cube"
scene.objects["Cube"]

keyboard = bge.logic.keyboard

LOOP = True
FPS = 60  # Frames per Second
last_time = timeit.default_timer()

while LOOP:
    # mainloop
    # wait until next frame
    current_time = timeit.default_timer()
    # sleep time until next frame
    sleep_time = 1 / FPS - (current_time - last_time)
    if sleep_time > 0:
        time.sleep(sleep_time)
    last_time = current_time

    # change scene
    # read keyboard input
    k_events = bge.logic.KX_INPUT_ACTIVE
```

```
    # uparrow key pressed
    if keyboard.events[bge.events.UPARROWKEY] == k_events:
        # move up cube .005 Blender-Units
        cube.position[2] += .01
    # downarrow key pressed
    if keyboard.events[bge.events.DOWNARROWKEY] == k_events:
        # move down cube .005 Blender-Units
        cube.position[2] -= .01
    # escape key pressed
    if keyboard.events[bge.events.ESCKEY] == k_events:
        # Stop mainloop -> exit program
        LOOP = False
    # render next frame
    bge.logic.NextFrame()
```

You can find the keys under Blender Game Engine – events.

More about Blender and the Game Enigne under Blender Game Engine – API

Object Oriented

Objectives

In this part you learn how to write a little game for the game engine which is splitted in many scripts, which are executed

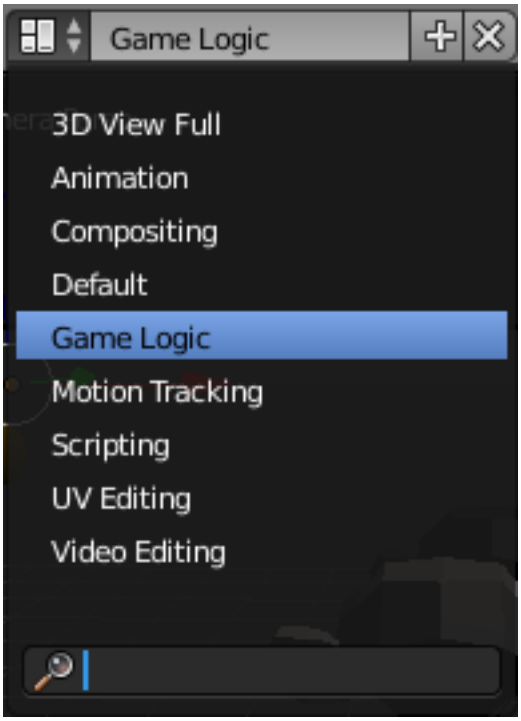
Instructions

Tasks

1. Download the .blend file from `here`
2. Add functions

Preparations

1. Open the the .blend file in Blender
2. Change the *Scene layout* to »Game Logic«



Running scripts for an object

If you want to run a script for a Blender-Object, you can use the *logic bricks* to do that. First you have to add an »Always« sensor. Click to the left ‘’‘, for executing the script every frame.

Then add an »Python« controller and set the *Script* Property to your script.

Then connect them.

The scripts

The script for our Player (The monkey named »Player«).

```
#! bpy
import bge
import timeit
import aud
import os


def main():
    """Contains all changes"""
    # get scene
    scene = bge.logic.getCurrentScene()
    # get contoller
    con = bge.logic.getCurrentController()
    # get player
    player = con.owner
    # get keyboard
    keyboard = bge.logic.keyboard
    # get the text object
    text = scene.objects["Text"]
    # check if start has already been run
    if "lives" not in player:
        start(player)
    # check for collisions
    for object in scene.objects:
        # does object collide
        # the distance is from origin to origin, so we subtract a bit
        if player.getDistanceTo(object) - 2 <= 0:
            # Is object an enemy ...
            if object.name.startswith("Enemy"):
                if player["can_collide"]:
                    player["lives"] -= 1
                    # play enemy hit sound
                    aud.device().play(player["enemy_sound"])
                    # Set that player can't collide
                    player["can_collide"] = False
                    # register time of collision
                    player["last_col"] = timeit.default_timer()
            # ... or is object a coin?
            elif object.name.startswith("Coin"):
                player["coins"] += 1
                # play collecting sound with the aud module
                aud.device().play(player["coin_sound"])
                # remove the coin from the scene
                object.endObject()

    # player hasn't got lives?
    if player["lives"] <= 0:
        print("lost")
        # remove player from scene
        player.endObject()
        # stop game
        bge.logic.endGame()
    # player got all coins?
    if player["coins"] >= 10:
        print("Win")
```

```
        bge.logic.endGame()
    # player can collide again?
    if not player["can_collide"]:
        if (timeit.default_timer() - player["last_col"]) >= 1:
            player["can_collide"] = True

    # move player
    # keyboard events
    k_events = bge.logic.KX_INPUT_ACTIVE
    # up arrow pressed?
    if keyboard.events[bge.events.UPARROWKEY] == k_events:
        # move player forwards on its local y axis
        player.applyMovement((0, 0, .2), True)
    # down arrow pressed
    if keyboard.events[bge.events.DOWNARROWKEY] == k_events:
        # move player backwards on its local y axis
        player.applyMovement((0, 0, -.2), True)
    # left arrow pressed
    if keyboard.events[bge.events.LEFTARROWKEY] == k_events:
        # rotate to the left
        player.applyRotation((0, 0, .05))
    # right arrow pressed
    if keyboard.events[bge.events.RIGHTARROWKEY] == k_events:
        # rotate to the right
        player.applyRotation((0, 0, -.05))
    # Set the text
    text.text = "Lives: {}; Coins: {}".format(player["lives"], player["coins"])


def start(player):
    """Setup the player"""
    player["lives"] = 3
    player["coins"] = 0
    player["can_collide"] = True
    # sound with the aud module
    # coin collect sound
    path = bge.logic.expandPath("//snd/coin.wav")
    player["coin_sound"] = aud.Factory.file(path)
    # enemy hit sound
    path = bge.logic.expandPath("//snd/enemy.wav")
    player["enemy_sound"] = aud.Factory.file(path)


main()
```

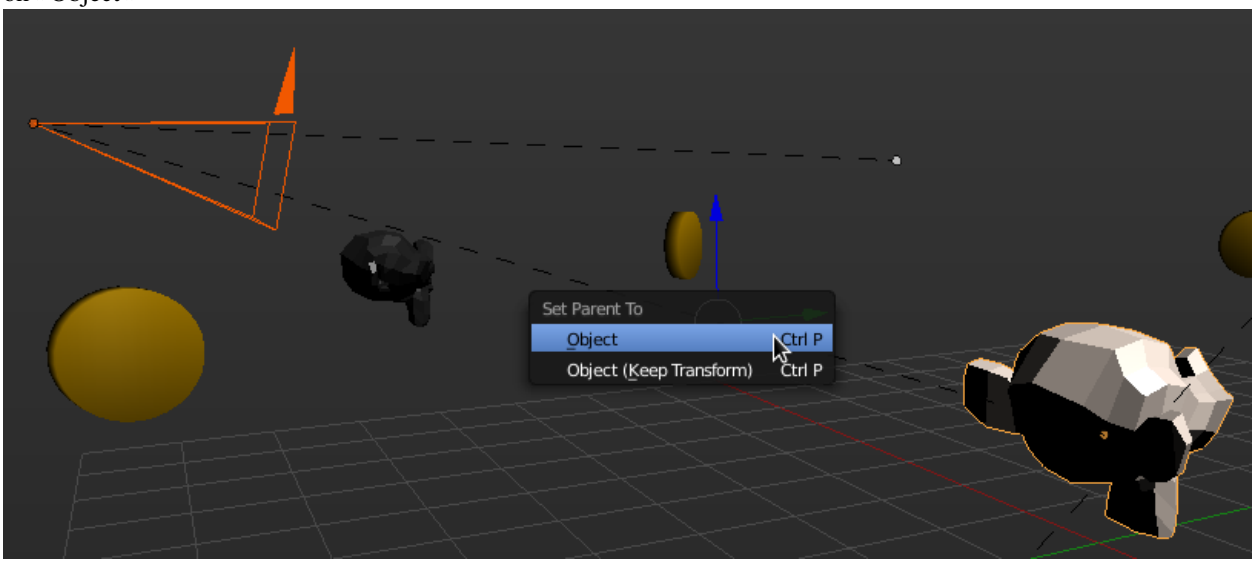
And the Script for our Enemys:

```
#! bpy
import bge
from random import randint


def main():
    """Contains all changes"""
    # get scene
    scene = bge.logic.getCurrentScene()
    # get contoller
    con = bge.logic.getCurrentController()
    # get player
```

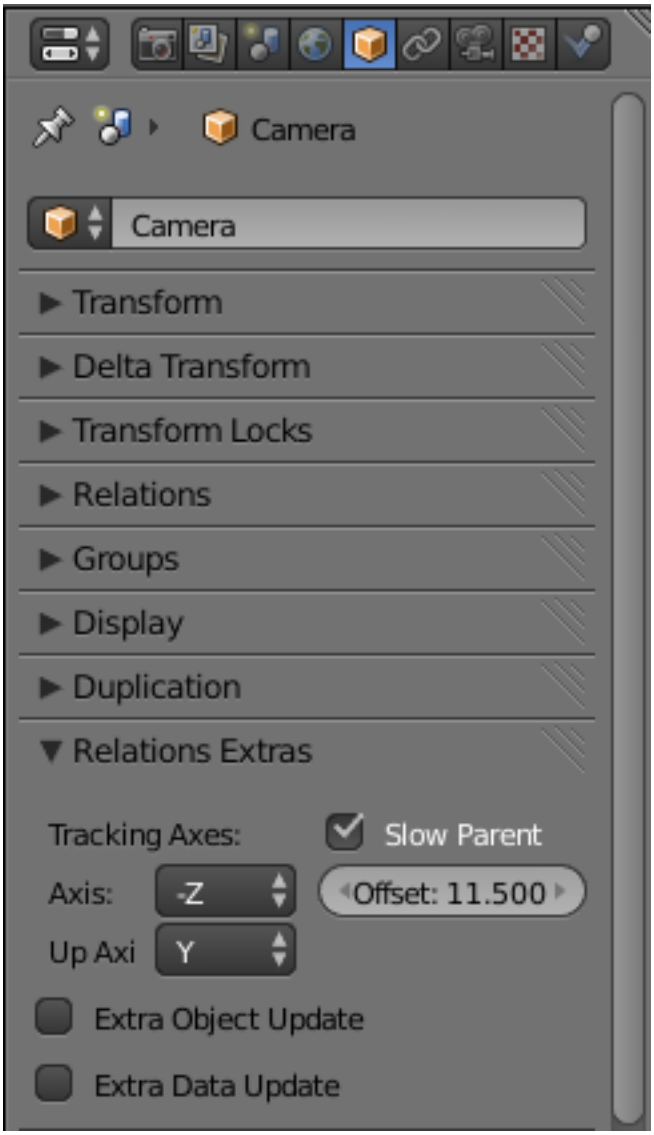
```
    enemy = con.owner

    # check, if enemy is outside the field
    if abs(enemy.position[0]) >= 20 or abs(enemy.position[1]) >= 20:
        # reverse direction
        enemy.applyRotation((0, 0, 180))
    # change direction in ca. 1% of the calls
    if randint(1, 100) == 1:
        # change direction
        enemy.applyRotation((0, 0, randint(1, 360)))
    # check for collisions
    for object in scene.objects:
        # does object collide
        # the distance is from origin to origin, so we subtract a bit
        if enemy.getDistanceTo(object) - 2 <= 0 and object != enemy:
            # Is object an enemy ...
            if object.name.startswith("Enemy"):
                # reverse direction
                enemy.applyRotation((0, 0, 180))
    # move
    enemy.applyMovement((0, 0, .15), True)


main()
```

The camera

Connecting the camera to the player: 1. Deselect all (»A« toggles selection) 2. Select the camera (with a right click) 3. Press shift while selecting the player, so you selected both and the player is active 4. Press »Ctrl« + »P« and click on »Object«



5. Select the camera again and enable »Slow Parent« in the *Relations Extras*
6. Set the *Offset* to »11.5«



More about Blender and the Game Enigne under Blender Game Engine – API

Game II

Sokoban - the idea

Objectives

(All information are taken form http://en.wikipedia.org/wiki/Sokoban)

Sokoban (warehouse keeper) is a type of transport puzzle, in which the player pushes boxes or crates arou

Instructions

Tasks

1. Download the `blend-file`
2. Read the code explanation in the next station
3. Start the game.
4. Create a new level.

History

Sokoban was created in 1981 by Hiroyuki Imabayashi, and published in 1982 by Thinking Rabbit, a software house based in Takarazuka, Japan.

Rules

The game is played on a board of squares, where each square is a floor or a wall. Some floor squares are marked as storage locations, and some of them have boxes.

The player is confined to the board, and may move horizontally or vertically onto empty squares (never through walls or boxes). The player can also move into a box, which pushes it into the square beyond. Boxes may not be pushed into other boxes or walls, and they cannot be pulled. The puzzle is solved when all boxes are at storage locations.

Leveldesign

A good article about leveldesign and sokoban:

- http://www.games4brains.de/sokoban-leveldesign.htm

Sokoban - the code

Objectives

This part shows the code for a basic sokoban game implemented with Blender ar

Instructions

Tasks

1. If you haven't done it so far, download the `blend-file`
2. Read the code, improve it and try to understand it
3. Add materials and textures to the objects
4. Create own levels

5. Save highscores to a file

The blend-file

Blender has a System with 20 Layers, which can switched on and off. You can do that with the keyboard or with the GUI.

- **Keyboard:** Change the layer: Press a (not numpad) number from 1 - 20

  Activate a layer: Press Shift + a (not numpad) number from 1 - 20

- **GUI:** Change the layer: Klick on an layer

  Activate a layer: Shift + klick on a layer

Note: The layer of the selected object includes a yellow point. Active layers are darker than unactivated. If you want to change the layer of the selected object, click M + the layers number.

If you start the game, only the 1st layer is allowed to be active. The 2nd layer with your objects must be inactive or you'll get an error.

The 1st layer

On the first layer are laying only 5 Objects:

- Two cameras, one for the game view and one for displaying different screens
- Two emptys, one for adding game objects and one for adding screens
- One lamp

The 2nd layer

All the other objects, which are needed for the game, are laying on the 2nd layer and the programm will add them to the game:

- player
- static/moveable box
- about, help and info screen
- end point
- ground

The Script

The Script is called every frame (that makes an always sensor which calls the script), so we can't store global variables in our script. So we need an object which exists all the time to store our variables there. I used the »add_empty« to do this and applied the always sensor to it, too. So we can save variables like this:

```
        a["state"] = "intro"
```

»a« is our »add_empty« and we store our state, which says if the player is in the intro or playing a level. Now we are on the second point: We execute always the same script, but we want to do different actions - displaying the intro and playing levels. So we must save what we did last and where we want to continue. So we need states, in our case

in named them »intro« and »play«. Each state has an own loop method, which is called when it's active. If the state changes, another method is called and adds the object for the state. This all happens in the method main():

```
def main():
    scene = bge.logic.getCurrentScene()
    # we use the empty to store variables
    a = scene.objects["add_empty"]
    if "state" not in a:
        # first method call, set state to intro and setup intro scene
        a["state"] = "intro"
        a["level_index"] = 0
        add_intro_objects()
    if a["state"] == "intro":
        # intro is active
        get = intro_loop()
        if get == "end":
            bge.logic.endGame()
        elif get == "start":
            # play first level
            a["state"] = "play"
            create_level(data.levels[a["level_index"]])
            a["level_index"] += 1
    elif a["state"] == "play":
        get = play_loop()
        if get == "end":
            # go back to intro
            a["state"] = "intro"
            add_intro_objects()
            a["level_index"] = 0
        elif get == "next":
            if a["level_index"] > len(data.levels) - 1:
                # all levels are played, go back to intro
                a["state"] = "intro"
                add_intro_objects()
                a["level_index"] = 0
            else:
                # play next level
                create_level(data.levels[a["level_index"]])
                a["level_index"] += 1
```

The Splash Screen (intro state)

Before starting the game, the player will see a splash screen. There he can call help and about. First, we have to set the camera active:

```
    camera = scene.objects["screen_cam"]
    scene.active_camera = camera
```

On keyboard input can show help, about or start the game. Here we look if »h« was pressed and if true, we hide intro and about screen and display the help screen.

```
    elif keyboard.events[bge.events.HKEY] == k_events:
        # display the help
        scene.objects["help"].setVisible(True)
        scene.objects["intro"].setVisible(False)
        scene.objects["about"].setVisible(False)
```

Adding objects

For adding an object, you need an object in the scene which is active yet. In our case we have the empty »add_empty« for adding objects and »screen_empty« for adding screens. The new objects will be added on the empty's position with it's rotation and scale.

I'll show you here some parts of the code, you can find the whole code at the end of the page. Before you setup a new scene, you have to clear the old one. That can be done with the following lines:

```
    for object in scene.objects:
        if object.name not in ['Camera', 'Lamp', 'screen_empty',
                               'add_empty', 'screen_cam']:
            object.endObject()
```

After that, we can add new objets, for example a static box:

```
                # the static box object
                static_box_obj = scene.objectsInactive["static_box"]
                # add static box to scene
                scene.addObject(static_box_obj, a)
```

The variable »a« is in this case an empty on the place where the object will be added.

The Game (play state)

After adding the objects and setting up the new camera, we convert the level in an easier format without end points. We do this because the end points could be overwritten if player or box is on it.

```
    # make copy that can be changed and store it
    a["field"] = [list(string.replace(".", " ").replace("*", "$").
                  replace("+", "@")) for string in level[:]]
```

We don't need them because we check by the positions if a box is over an end point, for example to set other colors.

```
def update_box_color(objects):
    """Gives the box another color, if it is over an end point"""
    # get all end points by name
    end_points = [object for object in objects if
                  object.name.startswith("end_point")]
    # get all moveable boxes by name
    moveable_boxes = [object for object in objects if
                      object.name.startswith("moveable_box")]
    for box in moveable_boxes:
        for point in end_points:
            # check if positions are the same
            if (int(box.position[0]), int(box.position[1])) \
               == (int(point.position[0]), int(point.position[1])):
                # change box color (red, green, blue, alpha)
                box.color = (.281, .232, .106, 1)
                break
            else:
                box.color = (.9, .872, .134, 1)
```

For more details read the comments in the code.

```
#! bpy

import aud  # for audio
```

```
import bge
import data
import timeit
import time
import os
from random import randint
from random import choice
from math import radians  # from degrees to radians
from math import degrees  # from radians to degrees


def play_loop():
    """Creates a new player and controls the game"""
    # our scene
    scene = bge.logic.getCurrentScene()
    # the object with the variables
    a = scene.objects["add_empty"]
    keyboard = bge.logic.keyboard
    # set box colors
    update_box_color(scene.objects)
    player = scene.objects["player"]
    # change the scene
    # read keyboard input
    k_events = bge.logic.KX_INPUT_JUST_ACTIVATED
    # uparrow key pressed
    if keyboard.events[bge.events.UPARROWKEY] == k_events:
        # move player forwards if he can
        move = can_move_player(player, a["field"], scene)
        if move:
            # player can move
            if move == 1:
                # move only the player
                move_forwards_player(player, a["field"])
            else:
                # move the player and the moveable box in front of him
                box = get_front_box(a["field"], player, scene.objects, scene)
                move_forwards_player(player, a["field"], box)
                # set box colors
                update_box_color(scene.objects)
                # check, if level is won
                if is_level_won(a["level"], a["field"]):
                    # end level
                    print("Won")
                    # play next level
                    return "next"
    elif keyboard.events[bge.events.LEFTARROWKEY] == k_events:
        rotate_left(player)
    elif keyboard.events[bge.events.RIGHTARROWKEY] == k_events:
        rotate_right(player)
    if keyboard.events[bge.events.QKEY] == k_events:
        # end level, go back to intro
        return "end"


def create_level(level):
    """Add the objects for a sobokan level"""
    # our active scene
    scene = bge.logic.getCurrentScene()
```

```
    # clear scene
    for object in scene.objects:
        if object.name not in ['Camera', 'Lamp', 'screen_empty',
                               'add_empty', 'screen_cam']:
            object.endObject()
    # our empty for adding
    a = scene.objects["add_empty"]
    # store the level variable
    a["level"] = level
    # our columns
    cols = len(level[0])
    # our rows
    rows = len(level)
    # the names of our created objects

    # add ground
    ground_pos = (rows - 1, cols - 1, -1)
    # the ground object
    ground_obj = scene.objectsInactive["ground"]
    # setup ground's adding position
    a.worldPosition = ground_pos
    # add the ground object to the scene
    ground = scene.addObject(ground_obj, a)
    # scale the ground
    ground.localScale[0] = rows
    ground.localScale[1] = cols
    # setup the camera
    camera = scene.objects["Camera"]
    scene.active_camera = camera
    # add the othe objects
    for row in range(rows):
        for i in range(cols):
            coords = (row * 2, i * 2, 0)
            # change adding position to coords
            a.worldPosition = coords
            if level[row][i] == "#":
                # the static box object
                static_box_obj = scene.objectsInactive["static_box"]
                # add static box to scene
                scene.addObject(static_box_obj, a)
            elif level[row][i] == "@":
                # the player object
                player_obj = scene.objectsInactive["player"]
                # Add the player
                player = scene.addObject(player_obj, a)
                player.applyRotation((radians(90), 0, 0))
            elif level[row][i] == "$":
                # the moveable box object
                moveable_box_obj = scene.objectsInactive["moveable_box"]
                # add a moveable box
                scene.addObject(moveable_box_obj, a)
            elif level[row][i] == ".":
                # the end point object
                end_point_obj = scene.objectsInactive["end_point"]
                # set the adding position to ground height
                a.worldPosition[2] = -.99
                # Add an end point
                scene.addObject(end_point_obj, a)
```

```
            elif level[row][i] == "*":
                # moveable box and end point on one field
                # the moveable box object
                moveable_box_obj = scene.objectsInactive["moveable_box"]
                # add a moveable box
                scene.addObject(moveable_box_obj, a)

                # the end point object
                end_point_obj = scene.objectsInactive["end_point"]
                # set the adding position to ground height
                a.worldPosition[2] = -.99
                # Add an end point
                scene.addObject(end_point_obj, a)
            elif level[row][i] == "+":
                # player and end point on one field
                # the player object
                player_obj = scene.objectsInactive["player"]
                # Add the player
                player = scene.addObject(player_obj, a)
                player.applyRotation((radians(90), 0, 0))

                # the end point object
                end_point_obj = scene.objectsInactive["end_point"]
                # set the adding position to ground height
                a.worldPosition[2] = -.99
                # Add an end point
                scene.addObject(end_point_obj, a)
    # make copy that can be changed and store it
    a["field"] = [list(string.replace(".", " ").replace("*", "$").
                  replace("+", "@")) for string in level[:]]


def move_forwards_player(game_object, field, box=None):
    """Moves the object (and a box, if given) one step forwards"""
    if box is not None:
        # update field
        field[int(box.worldPosition[0] / 2)][int(box.worldPosition[1] / 2)]\
            = " "
        # move the box in the »3D-View«
        box.orientation = game_object.orientation
        move(box, 2)
    # update field
    field[int(game_object.worldPosition[0] / 2)][int(
        game_object.worldPosition[1] / 2)] = " "
    # move the player in the »3D-View«
    move(game_object, 2)
    # update »3D-View«
    bge.logic.NextFrame()
    # update field
    if box:
        field[int(box.worldPosition[0] / 2)][int(box.worldPosition[1] / 2)]\
            = "$"
    field[int(game_object.worldPosition[0] / 2)][int(
        game_object.worldPosition[1] / 2)] = "@"


def move(game_object, distance):
    """Moves the object"""
```

```
    # get the objects orientation and move to the right direction
    if int(degrees(game_object.orientation.to_euler().z)) % 360 == 0:
        game_object.position[1] -= distance
    elif int(degrees(game_object.orientation.to_euler().z)) % 360 == 90:
        game_object.position[0] += distance
    elif int(degrees(game_object.orientation.to_euler().z)) % 360 == 180:
        game_object.position[1] += distance
    else:
        game_object.position[0] -= distance


def rotate(game_object, degrees):
    """Rotates the object around z axis"""
    game_object.applyRotation((0, 0, radians(degrees)))


def rotate_left(game_object):
    """Rotates the object 90 degrees to the left"""
    rotate(game_object, 90)


def rotate_right(game_object):
    """Rotates the object 90 degrees to the right"""
    rotate(game_object, -90)


def can_move_player(game_object, field, scene):
    """Check, if the player can be moved"""
    def check_for_obstruction(object_forwards, object_d_forwards):
        """Check, if object_forwards/object_d_forwards is an obstruction"""
        # place in front of the player is free
        if object_forwards == " " or object_forwards == ".":
            return 1
        # an obstruction is in front of the player
        elif object_forwards == "#":
            return 0
        # a movable box is in front of the player
        else:
            # there is nothing behind the box
            if object_d_forwards == " " or object_d_forwards == ".":
                return 2
            # there is something behind the box
            return 0

    # players position in the field
    player_position = (int(game_object.worldPosition[0] / 2),
                       int(game_object.worldPosition[1] / 2))
    try:
        # the object's orientation
        if int(degrees(game_object.orientation.to_euler().z)) % 360 == 0:
            # the object in front of the player
            object_forwards = field[player_position[0]][player_position[1] - 1]
            # the object after the object in front of he player
            object_d_forwards = field[player_position[0]][player_position[1]
                                                          - 2]
            return check_for_obstruction(object_forwards, object_d_forwards)
        elif int(degrees(game_object.orientation.to_euler().z)) % 360 == 90:
            object_forwards = field[player_position[0] + 1][player_position[1]]
```

```
            object_d_forwards = field[player_position[0]
                                      + 2][player_position[1]]
            return check_for_obstruction(object_forwards, object_d_forwards)
        elif int(degrees(game_object.orientation.to_euler().z)) % 360 == 180:
            object_forwards = field[player_position[0]][player_position[1] + 1]
            object_d_forwards = field[player_position[0]
                                      ][player_position[1] + 2]
            return check_for_obstruction(object_forwards, object_d_forwards)
        else:
            object_forwards = field[player_position[0] - 1][player_position[1]]
            object_d_forwards = field[player_position[0]
                                      - 2][player_position[1]]
            return check_for_obstruction(object_forwards, object_d_forwards)
    except IndexError:
        # object_d_forwards not in field so object_forwards must be static cube
        return 0


def get_front_box(field, player, objects, scene):
    """Returns the Blender object of the box in front of the player"""
    # players position in the field
    player_position = (int(player.worldPosition[0] / 2),
                       int(player.worldPosition[1] / 2))
    box = None
    # get player's orientation and check for a box
    if int(degrees(player.orientation.to_euler().z)) % 360 == 0:
        object_forwards = (player_position[0], player_position[1] - 1)
    elif int(degrees(player.orientation.to_euler().z)) % 360 == 90:
        object_forwards = (player_position[0] + 1, player_position[1])
    elif int(degrees(player.orientation.to_euler().z)) % 360 == 180:
        object_forwards = (player_position[0], player_position[1] + 1)
    else:
        object_forwards = (player_position[0] - 1, player_position[1])
    box = get_object_from_position(object_forwards,
                                   [object for object in scene.objects if
                                    object.name.startswith("moveable_box")])
    return box


def get_object_from_position(position, objects):
    """Get the object on position *position*"""
    for object in objects:
        if (int(object.position[0] / 2),
                int(object.position[1] / 2)) == position:
            return object


def is_level_won(level, field):
    """Look, if all moveable boxes are on an end point"""
    cols = len(level[0])
    rows = len(level)
    for row in range(rows):
        for i in range(cols):
            # is moveable box on an end point
            if field[row][i] == "$" and not (level[row][i] == "." or
                                             level[row][i] == "+" or
                                             level[row][i] == "*"):
                return False
```

```
    return True


def update_box_color(objects):
    """Gives the box another color, if it is over an end point"""
    # get all end points by name
    end_points = [object for object in objects if
                  object.name.startswith("end_point")]
    # get all moveable boxes by name
    moveable_boxes = [object for object in objects if
                      object.name.startswith("moveable_box")]
    for box in moveable_boxes:
        for point in end_points:
            # check if positions are the same
            if (int(box.position[0]), int(box.position[1])) \
               == (int(point.position[0]), int(point.position[1])):
                # change box color (red, green, blue, alpha)
                box.color = (.281, .232, .106, 1)
                break
            else:
                box.color = (.9, .872, .134, 1)


def intro_loop():
    """The intro before the first level is starting"""
    # our active scene
    scene = bge.logic.getCurrentScene()
    keyboard = bge.logic.keyboard
    # look if a key was pressed
    k_events = bge.logic.KX_INPUT_JUST_ACTIVATED
    # uparrow key pressed
    if keyboard.events[bge.events.SPACEKEY] == k_events:
        # start level
        return "start"
    elif keyboard.events[bge.events.HKEY] == k_events:
        # display the help
        scene.objects["help"].setVisible(True)
        scene.objects["intro"].setVisible(False)
        scene.objects["about"].setVisible(False)
    elif keyboard.events[bge.events.AKEY] == k_events:
        # show the about on the screen
        scene.objects["help"].setVisible(False)
        scene.objects["intro"].setVisible(False)
        scene.objects["about"].setVisible(True)
    if keyboard.events[bge.events.QKEY] == k_events:
        # if intro screen is shown, quit program
        return "end"
    return "ok"


def add_intro_objects():
    """Adds the objects which are needed for the intro"""
    # the active scene
    scene = bge.logic.getCurrentScene()
    # clear scene
    for object in scene.objects:
        if object.name not in ['Camera', 'Lamp', 'screen_empty',
                               'add_empty', 'screen_cam']:
```

```
            object.endObject()
    # the add_empty for adding objects
    a = scene.objects["add_empty"]
    # the screen_empty for adding screens
    s = scene.objects["screen_empty"]
    # set active camera
    camera = scene.objects["screen_cam"]
    scene.active_camera = camera
    # the intro screen object
    intro_screen_obj = scene.objectsInactive["intro"]
    # add intro screen
    scene.addObject(intro_screen_obj, s)
    # the help screen object
    help_obj = scene.objectsInactive["help"]
    # add the help screen
    help = scene.addObject(help_obj, s)
    # hide help screen
    help.setVisible(False)
    # the about screen object
    about_obj = scene.objectsInactive["about"]
    # add the about screen
    about = scene.addObject(about_obj, s)
    # hide about screen
    about.setVisible(False)


def main():
    scene = bge.logic.getCurrentScene()
    # we use the empty to store variables
    a = scene.objects["add_empty"]
    if "state" not in a:
        # first method call, set state to intro and setup intro scene
        a["state"] = "intro"
        a["level_index"] = 0
        add_intro_objects()
    if a["state"] == "intro":
        # intro is active
        get = intro_loop()
        if get == "end":
            bge.logic.endGame()
        elif get == "start":
            # play first level
            a["state"] = "play"
            create_level(data.levels[a["level_index"]])
            a["level_index"] += 1
    elif a["state"] == "play":
        get = play_loop()
        if get == "end":
            # go back to intro
            a["state"] = "intro"
            add_intro_objects()
            a["level_index"] = 0
        elif get == "next":
            if a["level_index"] > len(data.levels) - 1:
                # all levels are played, go back to intro
                a["state"] = "intro"
                add_intro_objects()
                a["level_index"] = 0
```

```
            else:
                # play next level
                create_level(data.levels[a["level_index"]])
                a["level_index"] += 1


main()
```

CHAPTER 7

Python Basics

There are many Python courses available. We only discuss some special corner cases. Have also a look at the linklist below.

Data types: Overview

Objectives

Python provides a set of data types. As a supplement and for a quick entry, a brie

Instructions

Tasks

1. Have a look to the standard documentation in addition to the given examples.
2. Execute the example in the next chapters. Do not forget to create variations.

Numbers

- Integer
- Float
- Long
- Complex

Sequences

- *Strings*
- *Tuple*

- *Lists*

Mappings/Collections

- *Dictionary*
- Set

Tecniques useful for different data types

- *Slicing*
- *Modulo-Operator (string formatting or interpolation operator)*
- *Modulo: the remainder is important*

Data type: string

Objectives

Every programming language is using strings as a data type. The Python wa

Instructions

Tasks

1. Save the following text, line by line into variables:

```
"Shall I compare thee to a summer's day?
Thou art more lovely and more temperate..."

Lines from Shakespeare's Sonnet 18.
```

2. Return the values of all variables with the print-Function. Don't forget the empty line.

Properties and rules

- A string is enclosed in the following characters: single quotation, double quotation and triples of the quotation signs.
- Strings are immutable, if you change a string, the computer has to reserve new memory and afterward release the old memory.
- Each character is callable with an index. The index counting starts at zero.

More hints about strings kann be found at:

Wikipedia: String (computer science)

Assign strings

Each variable get's own value.

todo: find a good equivalent to the german version...

Concatenate strings

Building longer Strings from substrings, is an often used technique.

Hint The same result can be achieved with other data types tuple and *list*.

Extract parts of string

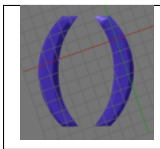
This task is possible with the slicing operator. This operation is available for all sequence data types. See also *slicing*.

Wildcards and strings

It is also posseble to include strings in an other string with predefined slots. The percent character(%) is used as a place-marker, followed by a single character s (%s). More about this techniques can be found in the learning station *modulo*.

Data type: tuple & for loop

Objectives



The specifics of the tuple data type, short and sweet.

Instructions

Tasks

1. Create a second file: tuple_and_lists.py (Note: files must not be named for reserved words, because it can lead to name conflicts!)
2. Save your address in a tuple.
3. Enter the address of the tuple.
4. Use a for loop to output the tuple.

Content

Properties ...

- Immutable like strings
- External characteristic in the source code are the parentheses
- Can take any object
- Parts of a tuple can be extract. The index starts with 0.
- Attempt to change the value of a tuple entry is not possible!

Example:

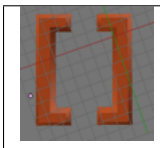
In these examples, the for loop is used for the control flow.

```
T1 = () # an empty tupel (considered unusual and purely academic)
t2 = ('The', 'quick', 'brown', 'fox', 'jumps', 'over', 7, 'stones')
print(t2[0], t2[1], t2[2], t2[3], t2[4], t2[5], t2[6], t2[7])
for i in t2:
    print(i, )
print()
print(t3)
t2[4] = 9 # Will that work?
```

As you can extract parts of a tuple is in the station *slicing* shown.

Data type: Lists & while

Objectives



Introduction to the datatype *list*.

Instructions

Tasks

1. Discover the methods a list supports. Use the command *dir([])* at the command line.
2. Create a list with things available in your environment.
3. Print some elements or all from your list.

Properties

The essential properties of a list are:

- Square brackets - the external indicator if parts of list are used.

- A list can accommodate a wide variety of object types, for example, Strings, numbers ...
- She has a variety of methods to manage the list items.

For more information about lists, have a look at: Built-in Types

Repetition with while

With a while loop, you can reach the same results as with the for loop! A termination condition leads to the end of the loop or to a premature demolition.

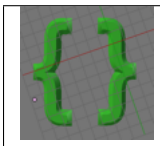
```
L1 = []  # empty List
L2 = ['The', 'brave', 'tailor', 'hit', 7, 'on', 'a', 'string']
print(L2[0], L2[1], L2[2], L2[3], L2[4], L2[5], L2[6], L2[7])
i = 0
while i < len(L2):
    print(L2[i])
    i = i + 1           # The variable i must Modify the value,
                        # Otherwise it will loop indefinitely !!


L2.reverse()
print(L2)
```

With the slicing operator further sub output can be realized.

See also: *Slicing*

Data type: dictionary & if

Objectives



In other programming langauges a dictionary is named »Associative array«, in Python it is named »Dictionary«. Some

Instructions

Tasks

1. Create a script: dictionaries.py
2. Create a second version of the refrigerator example according to the station for data type lists.
3. Expand the mushroom example and generate a complete HTML page with *print* function.

Properties of a dictionary

- Dictionaries are called mapping-type.
- They are similar to the associative array of other programming languages.
- They are represented by a pair of curly brackets *{}* and always contain name/value pairs that are separated by a colon. Names are also called *keys*

- The order of the key values in the dictionary is random.

Examples

```
cars = {'BMW':'Germany', 'Renault':'France', 'Volvo':'Sweden'}

# other stucture...

cars = {'BMW':'Germany',
        'Renault':'France',
        'Volvo':'Sweden'}
```

Method	Note	Example, comment
clear()	removes all entries from a dictionary	cars.clear() the result os an empty dictionary {}
copy()	copy all entries af a give dictionary	copy_of_cars = cars.copy()
has_key()	returns the value 1 if the key exists if not 0	cars.has_key('Volvo')
items()	returns all name/value pairs each as a tuple	cars.items()
keys()	returns a list of all keys	cars.keys() |
update()	add a new name/value, if the key exists the value will be replaced	cars['Skoda'] = 'Czech Republic'
get()	return the value to a give name	cars.get('Volvo') cars.get('xyz','???')

Second Example

```
# create a dictionary...

mushrooms = {} # empty dictionary
mushrooms['cep'] = ['eatable x times', "mixed forest", "Boletus edulis"]
mushrooms['fly amanita'] = ["eatable once", "coniferous forest", "Amanita muscaria"]
mushrooms['bay bolete'] = ["eatable x times", "mixed forest", "Boletus badius"]
mushrooms['Coprinus'] = ['eatable x times', "meadow", "Coprinus"]

# print the content:

print('''Pretty print dictionary of mushrooms:\n ''')

for i in mushrooms.keys():
    print("""kind = {}, eatable = {}, be found = {}; """.format(i,
                                                               mushrooms[i][0],
                                                               mushrooms[i][1]))
```

In this example some other data types are used or created.

- Dictionary = mushrooms
- List = as result of the mehtod call keys()
- Tupel = as parameter of the format method

Many more variations are possible eg. combined with a *Slicing* -operator.

Module: Formating not calculating

Objectives

If all calculations are finished, print is often used to present all values as a string. Beside th

Instructions

Tasks

1. Change the examples presented in the old as in the new notation.
2. Print a list, as it is listed on a sales slip.
3. **Look at the examples in the Python documentation at:** http://docs.python.org/2/library/string.html#format-examples

String formating with the Moduluo-Operator %

Principles to the old version

- The % sign is a wildcard, followed by a letter indicating the data type.
- Examples: %s, %d, %f, %2d, %2.3f
- The number after the percent sign specifies the precision to be used for output.
- The last % sign initiates the handover of parmaeter(s) to be used for the placeholder.
- The number of parameters (list, tuple) must match the number of placeholders.

Special case: Dictionary as parameter

- If the parameter is a dictionary, then the wildcard must matching the names (keys) of the dictinary.

Example 1

```
for i in range(1,11):
  print("%s * %s = %s" % (i,10,i*10))
```

Example 2

```
print("The %s f%sx j%ss" % ("brown","o","ump"))
```

Example 3

```
# create a dictionary...

mushrooms = {} # empty dictionary
mushrooms['cep'] = ['eatable x times', "mixed forest", "Boletus edulis"]
mushrooms['fly amanita'] = ["eatable once", "coniferous forest", "Amanita muscaria"]
mushrooms['bay bolete'] = ["eatable x times", "mixed forest", "Boletus badius"]
mushrooms['Coprinus'] = ['eatable x times', "meadow", "Coprinus"]

# print the content:

print('''Pretty print dictionary of mushrooms:\n ''')

for i in mushrooms.keys():
    print("""kind = %s, eatable = %s, be found = %s; """.format(i,
                                                              mushrooms[i][0],
                                                              mushrooms[i][1]))
```

Example 4

```
adress = {"firstnamename": "Robin",
          "lastname": "Hood",
          "location": "Sharewood Forest",
          "mark": "Bow & Arrow"}

print("""
         Wanted %(lastname)s, %(firstname)s
         special mark %(mark)s
         lives in %(location)s
      """ % (adress)
```

String formating with format()

This new method was discussed in PEP 3101 and is available since Python version 2.7.

Principles to the new version

- placehoder is a pair of curly braces
- the parameter are placed in the *format* method

The following examples repeat the old examples above and should the cause the same result.

Example 1

```
for i in range(1,11):
  print('{} * {} = {}'.format(i,10,i*10))
```

The same example, but more table like:

```
for i in range(1,11):
  print('{:2d} * {:2d} = {:3d}'.format(i,10,i*10))
```

```
x = 10
print('{0:2d} {1:3d} {2:4d}'.format(x, x*x, x*x*x))
```

```
print("The {} f{}x j{}s".format("brown","o","ump"))
```

Example 3

```
# create a dictionary...

mushrooms = {} # empty dictionary
mushrooms['cep'] = ['eatable x times', "mixed forest", "Boletus edulis"]
mushrooms['fly amanita'] = ["eatable once", "coniferous forest", "Amanita muscaria"]
mushrooms['bay bolete'] = ["eatable x times", "mixed forest", "Boletus badius"]
mushrooms['Coprinus'] = ['eatable x times', "meadow", "Coprinus"]

# print the content:

print('''Pretty print dictionary of mushrooms:\n ''')

for i in mushrooms.keys():
    print("""kind = {}, eatable = {}, be found = {}; """.format(i,
                                                                mushrooms[i][0],
                                                                mushrooms[i][1]))
```

Excample 4

```
adress = {"firstnamename": "Robin",
          "lastname": "Hood",
          "location": "Sharewood Forest",
          "mark": "Bow & Arrow"}
print("""
         Wanted {lastname}, {firstname}
         special mark {mark}
         lives in {location}
      """ % (**adress)
```

Example 5

A little collection fo exotic versions...

```
>>> name ="Peter"
>>> "{!r:>10}".format(name)
"   'Peter'"

>>> '{:x>42}'.format(name)
'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxPeter'
```

Modulo: the rest is important

Objectives

The modulus operator is used twice in Python. Once for the formatted output of texts and secondly for the calculation

Instructions

Tasks

1. What is the result if the calculation of the number 3 is performed?
2. Search for some aother examples you can find on the internet.

Calculating with the modulo operator %

This example is using a for loop:

```
for i in range(1,11):
   print('{} divided by {} is {} and the reminder is {}'.format(i, 2, i/2, i % 2))
```

The result:

```
1 divided by 2 is 0 and the reminder is 1
2 divided by 2 is 1 and the reminder is 0
3 divided by 2 is 1 and the reminder is 1
4 divided by 2 is 2 and the reminder is 0
5 divided by 2 is 2 and the reminder is 1
6 divided by 2 is 3 and the reminder is 0
7 divided by 2 is 3 and the reminder is 1
8 divided by 2 is 4 and the reminder is 0
9 divided by 2 is 4 and the reminder is 1
10 divided by 2 is 5 and the reminder is 0
```

Question: What can you achieve?

Answer: It is clear which number is even (rest = 0) and that number is odd (residual = 1).

Question: Where do I need this?

Answer: For example, the different coloring of the table rows. No matter how long the table is, the color is always exchanged in the exchange, and thus the table more readable.

Die Liste aller Rechner

Schulungsraum 1

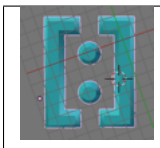
Nummer	Status	Anmerkung	Rechner	IP
01	Schrott	Dozentenrechner	R2C19.ffm.e-cademy.de	192.168.64.37
02	Schulungsrechner	R1Fx	R1C06.ffm.e-cademy.de	192.168.62.15
03	Schulungsrechner	R1Fx	R1C05.ffm.e-cademy.de	192.168.62.35
04	Schulungsrechner	R1Fx	R1C03.ffm.e-cademy.de	192.168.62.21

In Blender we use the modulo operator to assign materials in exchange, each plot.

Other examples can be found under http://en.wikipedia.org/wiki/Euclidean_division

Slicing

Objectives



Slicing is a useful technique to operate on all sequenz data types. Helpful examples are given in this station.

Instructions

Tasks

1. Create a tuple or a list with the following sentence and store each word as a value: “Simple is better than complex”
2. Return the last value of the tuple or list.
3. Create a dictionary with ten concept pairs <first language> – <second language>.
4. Print the content of the dictionare in order.
5. Construct a data structure which stores the position of a chess game.
6. Return the position as ASCII art.

What is slicing?

It is the selection of a subset of a sequence of values. This works with strings, tuples and lists alike.

Cut part range from - to:

```
s = "Big Bug Bunny"
part = s[8:10]
print(s, part)
```

Cut the last part:

```
s = "Big Bug Bunny"
part = s[-1:]
print(s, part)
```

Cut first element:

```
s = "Big Bug Bunny"
part = s[0]
print(s, part)
```

Cut all elements:

```
s = "Big Bug Bunny"
part = s[:] # same as s without braces
print(s, part)
```

Nested Views

What applies to the strings, can also be applied to *tuples* and *lists*. If the element is a character string, it can also partly cut again:

```
t1 = ('The', 'quick', 'brown fox', 'jumps', 'over', 'the', 'lazy', 'dog')
t2 = t1[2][3:8]
print(t2)
```

Dictionaries and slicing

For the data type * dictionary * there is no slicing operator! Various methods returns a list as a result. If the result is a list, slicing is possible again.

```
# create a dictionary...

mushrooms = {} # empty dictionary
mushrooms['cep'] = ['eatable x times', "mixed forest", "Boletus edulis"]
mushrooms['fly amanita'] = ["eatable once", "coniferous forest", "Amanita muscaria"]
mushrooms['bay bolete'] = ["eatable x times", "mixed forest", "Boletus badius"]
mushrooms['Coprinus'] = ['eatable x times', "meadow", "Coprinus"]

# print the content:

print('''Pretty print dictionary of mushrooms:\n ''')

for i in mushrooms.keys():
    print("""kind = {}, eatable = {}, be found = {}; """.format(i,
                                                                mushrooms[i][0],
                                                                mushrooms[i][1]))
```

Other Python courses

- https://developers.google.com/edu/python

CHAPTER 8

Python Specials

CHAPTER 9

Appendix

Blender: often used commands

Table of contents

Template: section blender basics

```
#!bpy
"""
Name: 'Template'
Blender: 2.69
Group: 'Sample'
Tooltip: 'Template for new scripts, copy it and start coding...'
"""
import bpy

def function_1():
    """ One line describing the task of this function  """
    pass


def function_2():
    """ One line describing the task of this function  """
    print(__name__)
    print(50 * '*')


def function_3():
    """ One line describing the task of this function  """
```

```
    bpy.ops.mesh.primitive_cone_add(location=(1, 2, 1))


if __name__ == '__main__':
    # call the function for testing
    function_1()
    #function_2()
    #function_3()
```

Template: simple sample

```
#!bpy
"""
Name: '???'
Blender: 2.69
Group: 'Experiment'
Tooltip: 'Try...'
"""

import bpy


def test():
    """ One line describing the task of this function """
    pass

if __name__ == '__main__':
    # Stop edit mode
    if bpy.ops.object.mode_set.poll():
        bpy.ops.object.mode_set(mode='OBJECT')

    # delete all mesh objects from a scene
    bpy.ops.object.select_by_type(type='MESH')
    bpy.ops.object.delete()

    # call the new function
    test()
```

Find Objects by name

All objecte as list of names ...

Version I

```
all = [item.name for item in bpy.data.objects]
for name in all:
    print(name)
```

Version II

```
# collect all names
obj_names = []
for obj in bpy.context.scene.objects:
    obj_names.append(obj.name)
```

Selection by type ...

```
bpy.ops.object.select_by_type(type='MESH')
bpy.ops.object.select_by_type(type='CURVE')
```

Selection by pattern...

```
bpy.ops.object.select_pattern(pattern="Mein Auto")
```

Deaktivate all objects

```
for obj in bpy.context.selectable_objects:
    obj.select = False
```

Switch between edit- and object mode

Works only if a object is selected.

```
bpy.ops.object.editmode_toggle()
```

```
bpy.ops.object.mode_set(mode = 'OBJECT')
bpy.ops.object.mode_set(mode = 'EDIT')
```

Delete objects

Works only, if objects are selected ...

```
bpy.ops.object.select_by_type(type='MESH')
bpy.ops.object.delete()
```

Activate object

```
bpy.context.scene.objects.active = bpy.data.objects["Cube"]
```

Todo

check all variants, to set a path ...

Detect the index of a face

```
def getIndexOfFaces(name):
    """ Print the index of faces to the console

        - Switch to the edit-Mode
        - select faces
        - the output ist shown in the console
    """
    bpy.context.scene.objects.active = bpy.data.objects[name]
    bpy.ops.object.mode_set(mode='EDIT')
    bpy.ops.object.mode_set(mode='OBJECT')
    me = bpy.context.object.data
```

```
    for poly in me.polygons:
        if poly.select:
            print("Polygon index: %d, length: %d" % (poly.index,
                                                     poly.loop_total))
    bpy.ops.object.mode_set(mode='EDIT')
```

Operator list

Objectives

```
#   1469
# Operators
```

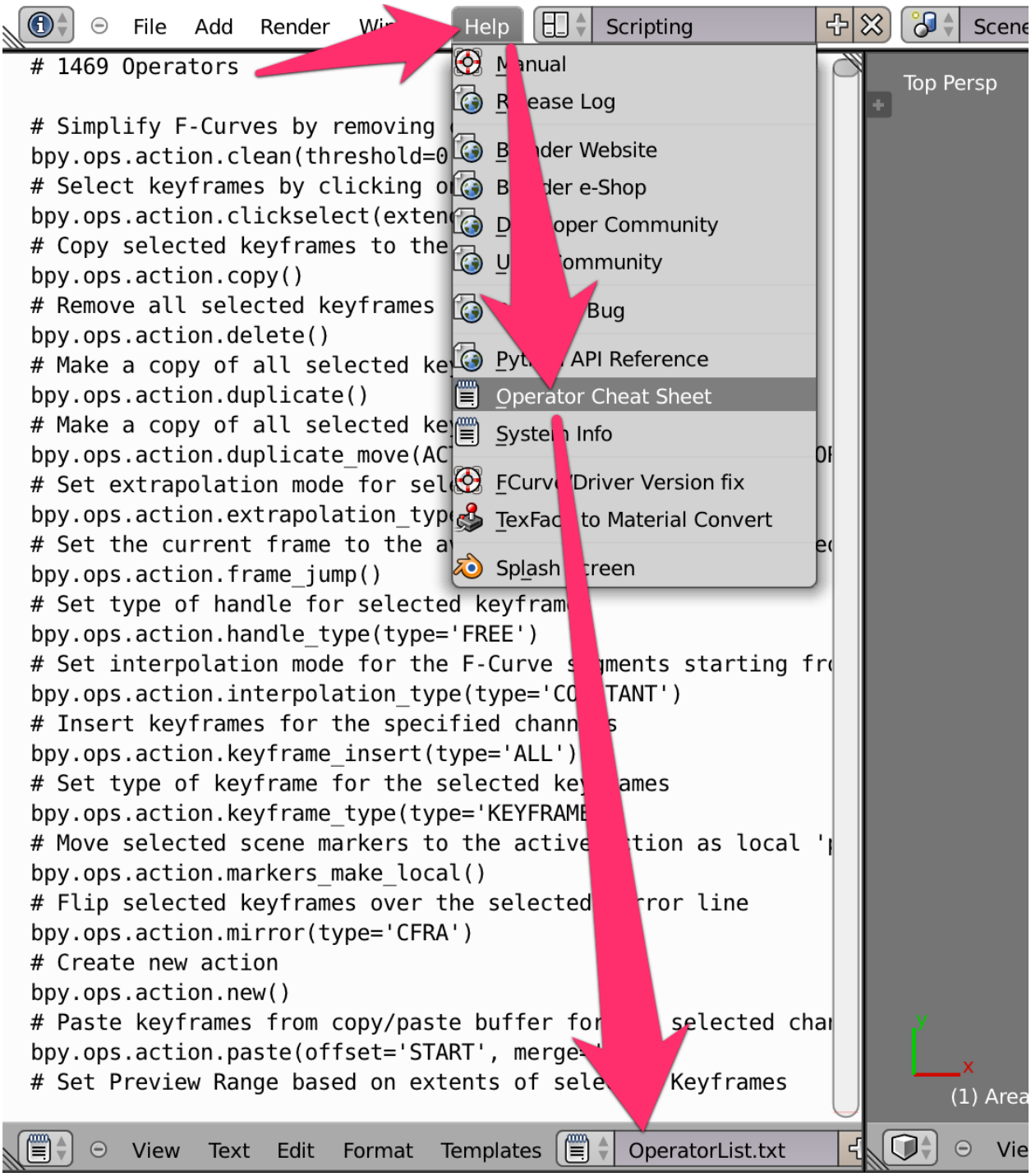
This is a copy from the file *OperatorList.txt*. It is also available with the menu sequence *Help » Operator Cheat Sheet*.

Instructions

Tasks

1. Select the menu *Help » Operator Cheat Sheet*
2. Switch the text file and search for » **mesh.primitive_** «

Call the menu



List of operators

Start searching in most browsers (Ctrl+f), more information are available in the API.

http://www.blender.org/documentation/blender_python_api_2_69_1/

1469 Operators

```
# Simplify F-Curves by removing closely spaced keyframes
bpy.ops.action.clean(threshold=0.001)
# Select keyframes by clicking on them
bpy.ops.action.clickselect(extend=False, column=False)
# Copy selected keyframes to the copy/paste buffer
bpy.ops.action.copy()
# Remove all selected keyframes
bpy.ops.action.delete()
# Make a copy of all selected keyframes
bpy.ops.action.duplicate()
# Make a copy of all selected keyframes and move them
bpy.ops.action.duplicate_move(ACTION_OT_duplicate={},
TRANSFORM_OT_transform={"mode":'TRANSLATION', "value":(0, 0, 0, 0), "axis":(0, 0, 0),
"constraint_axis":(False, False, False), "constraint_orientation":'GLOBAL', "mirror":False,
"proportional":'DISABLED', "proportional_edit_falloff":'SMOOTH', "proportional_size":1, "snap":False,
"snap_target":'CLOSEST', "snap_point":(0, 0, 0), "snap_align":False, "snap_normal":(0, 0, 0),
"release_confirm":False})
# Set extrapolation mode for selected F-Curves
bpy.ops.action.extrapolation_type(type='CONSTANT')
# Set the current frame to the average frame value of selected keyframes
bpy.ops.action.frame_jump()
# Set type of handle for selected keyframes
bpy.ops.action.handle_type(type='FREE')
# Set interpolation mode for the F-Curve segments starting from the selected keyframes
bpy.ops.action.interpolation_type(type='CONSTANT')
# Insert keyframes for the specified channels
bpy.ops.action.keyframe_insert(type='ALL')
# Set type of keyframe for the selected keyframes
bpy.ops.action.keyframe_type(type='KEYFRAME')
# Move selected scene markers to the active Action as local 'pose' markers
bpy.ops.action.markers_make_local()
# Flip selected keyframes over the selected mirror line
bpy.ops.action.mirror(type='CFRA')
# Create new action
bpy.ops.action.new()
# Paste keyframes from copy/paste buffer for the selected channels, starting on the current frame
bpy.ops.action.paste(offset='START', merge='MIX')
# Set Preview Range based on extents of selected Keyframes
bpy.ops.action.previewrange_set()
# Add keyframes on every frame between the selected keyframes
bpy.ops.action.sample()
# Toggle selection of all keyframes
bpy.ops.action.select_all_toggle(invert=False)
# Select all keyframes within the specified region
bpy.ops.action.select_border(gesture_mode=0, xmin=0, xmax=0, ymin=0, ymax=0, extend=True, axis_range=False)
```

```
# Select all keyframes on the specified frame(s)
bpy.ops.action.select_column(mode='KEYS')
# Select keyframes to the left or the right of the current frame
bpy.ops.action.select_leftright(mode='CHECK', extend=False)
# Deselect keyframes on ends of selection islands
bpy.ops.action.select_less()
# Select keyframes occurring in the same F-Curves as selected ones
bpy.ops.action.select_linked()
# Select keyframes beside already selected ones
bpy.ops.action.select_more()
# Snap selected keyframes to the times specified
bpy.ops.action.snap(type='CFRA')
# Reset viewable area to show full keyframe range
bpy.ops.action.view_all()
# Reset viewable area to show selected keyframes range
bpy.ops.action.view_selected()


# Interactively change the current frame number
bpy.ops.anim.change_frame(frame=0)
# Handle mouse-clicks over animation channels
bpy.ops.anim.channels_click(extend=False, children_only=False)
# Collapse (i.e. close) all selected expandable animation channels
bpy.ops.anim.channels_collapse(all=True)
# Delete all selected animation channels
bpy.ops.anim.channels_delete()
# Toggle editability of selected channels
bpy.ops.anim.channels_editable_toggle(mode='TOGGLE', type='PROTECT')
# Expand (i.e. open) all selected expandable animation channels
bpy.ops.anim.channels_expand(all=True)
# Clears 'disabled' tag from all F-Curves to get broken F-Curves working again
bpy.ops.anim.channels_fcurves_enable()
# Add selected F-Curves to a new group
bpy.ops.anim.channels_group(name="New Group")
# Rearrange selected animation channels
bpy.ops.anim.channels_move(direction='DOWN')
# Rename animation channel under mouse
bpy.ops.anim.channels_rename()
# Toggle selection of all animation channels
bpy.ops.anim.channels_select_all_toggle(invert=False)
# Select all animation channels within the specified region
bpy.ops.anim.channels_select_border(gesture_mode=0, xmin=0, xmax=0, ymin=0, ymax=0, extend=True)
# Disable specified setting on all selected animation channels
bpy.ops.anim.channels_setting_disable(mode='DISABLE', type='PROTECT')
# Enable specified setting on all selected animation channels
bpy.ops.anim.channels_setting_enable(mode='ENABLE', type='PROTECT')
# Toggle specified setting on all selected animation channels
```

```
bpy.ops.anim.channels_setting_toggle(mode='TOGGLE', type='PROTECT')
# Remove selected F-Curves from their current groups
bpy.ops.anim.channels_ungroup()
# Make only the selected animation channels visible in the Graph Editor
bpy.ops.anim.channels_visibility_set()
# Toggle visibility in Graph Editor of all selected animation channels
bpy.ops.anim.channels_visibility_toggle()
# Mark actions with no F-Curves for deletion after save & reload of file preserving "action libraries"
bpy.ops.anim.clear_useless_actions(only_unused=True)
# Copy the driver for the highlighted button
bpy.ops.anim.copy_driver_button()
# Add driver(s) for the property(s) connected represented by the highlighted button
bpy.ops.anim.driver_button_add(all=True)
# Remove the driver(s) for the property(s) connected represented by the highlighted button
bpy.ops.anim.driver_button_remove(all=True)
# Clear all keyframes on the currently active property
bpy.ops.anim.keyframe_clear_button(all=True)
# Remove all keyframe animation for selected objects
bpy.ops.anim.keyframe_clear_v3d()
# Delete keyframes on the current frame for all properties in the specified Keying Set
bpy.ops.anim.keyframe_delete(type='DEFAULT', confirm_success=True)
# Delete current keyframe of current UI-active property
bpy.ops.anim.keyframe_delete_button(all=True)
# Remove keyframes on current frame for selected objects
bpy.ops.anim.keyframe_delete_v3d()
# Insert keyframes on the current frame for all properties in the specified Keying Set
bpy.ops.anim.keyframe_insert(type='DEFAULT', confirm_success=True)
# Insert a keyframe for current UI-active property
bpy.ops.anim.keyframe_insert_button(all=True)
# Insert Keyframes for specified Keying Set, with menu of available Keying Sets if undefined
bpy.ops.anim.keyframe_insert_menu(type='DEFAULT', confirm_success=False, always_prompt=False)
# Select a new keying set as the active one
bpy.ops.anim.keying_set_active_set(type='DEFAULT')
# Add a new (empty) Keying Set to the active Scene
bpy.ops.anim.keying_set_add()
# Export Keying Set to a python script
bpy.ops.anim.keying_set_export(filepath="", filter_folder=True, filter_text=True, filter_python=True)
# Add empty path to active Keying Set
bpy.ops.anim.keying_set_path_add()
# Remove active Path from active Keying Set
bpy.ops.anim.keying_set_path_remove()
# Remove the active Keying Set
bpy.ops.anim.keying_set_remove()
# Add current UI-active property to current keying set
bpy.ops.anim.keyingset_button_add(all=True)
# Remove current UI-active property from current keying set
bpy.ops.anim.keyingset_button_remove()
```

# Paste the driver in the copy/paste buffer for the highlighted button
bpy.ops.anim.paste_driver_button()
# Clear Preview Range
bpy.ops.anim.previewrange_clear()
# Interactively define frame range used for playback
bpy.ops.anim.previewrange_set(xmin=0, xmax=0, ymin=0, ymax=0)
# Update data paths from 2.56 and previous versions, modifying data paths of drivers and fcurves
bpy.ops.anim.update_data_paths()

# Align selected bones to the active bone (or to their parent)
bpy.ops.armature.align()
# Change the visible armature layers
bpy.ops.armature.armature_layers(layers=(False, False))
# Automatically renames the selected bones according to which side of the target axis they fall on
bpy.ops.armature.autoside_names(type='XAXIS')
# Change the layers that the selected bones belong to
bpy.ops.armature.bone_layers(layers=(False, False))
# Add a new bone located at the 3D-Cursor
bpy.ops.armature.bone_primitive_add(name="Bone")
# Automatically fix alignment of select bones' axes
bpy.ops.armature.calculate_roll(type='X', axis_flip=False, axis_only=False)
# Create a new bone going from the last selected joint to the mouse position
bpy.ops.armature.click_extrude()
# Remove selected bones from the armature
bpy.ops.armature.delete()
# Make copies of the selected bones within the same armature
bpy.ops.armature.duplicate()
# Make copies of the selected bones within the same armature and move them
bpy.ops.armature.duplicate_move(ARMATURE_OT_duplicate={}, TRANSFORM_OT_translate={"value":(0, 0, 0), "constraint_axis":(False, False, False), "constraint_orientation":'GLOBAL', "mirror":False, "proportional":'DISABLED', "proportional_edit_falloff":'SMOOTH', "proportional_size":1, "snap":False, "snap_target":'CLOSEST', "snap_point":(0, 0, 0), "snap_align":False, "snap_normal":(0, 0, 0), "texture_space":False, "release_confirm":False})
# Create new bones from the selected joints
bpy.ops.armature.extrude(forked=False)
# Create new bones from the selected joints and move them
bpy.ops.armature.extrude_forked(ARMATURE_OT_extrude={"forked":False}, TRANSFORM_OT_translate={"value":(0, 0, 0), "constraint_axis":(False, False, False), "constraint_orientation":'GLOBAL', "mirror":False, "proportional":'DISABLED', "proportional_edit_falloff":'SMOOTH', "proportional_size":1, "snap":False, "snap_target":'CLOSEST', "snap_point":(0, 0, 0), "snap_align":False, "snap_normal":(0, 0, 0), "texture_space":False, "release_confirm":False})
# Create new bones from the selected joints and move them
bpy.ops.armature.extrude_move(ARMATURE_OT_extrude={"forked":False}, TRANSFORM_OT_translate={"value":(0, 0, 0), "constraint_axis":(False, False, False),

“constraint_orientation”:’GLOBAL’, “mirror”:False, “proportional”:’DISABLED’,
“proportional_edit_falloff”:’SMOOTH’, “proportional_size”:1, “snap”:False, “snap_target”:’CLOSEST’,
“snap_point”:(0, 0, 0), “snap_align”:False, “snap_normal”:(0, 0, 0), “texture_space”:False, “release_confirm”:False})

```
# Add bone between selected joint(s) and/or 3D-Cursor
bpy.ops.armature.fill()
# Flips (and corrects) the axis suffixes of the names of selected bones
bpy.ops.armature.flip_names()
# Tag selected bones to not be visible in Edit Mode
bpy.ops.armature.hide(unselected=False)
# Make all armature layers visible
bpy.ops.armature.layers_show_all(all=True)
# Merge continuous chains of selected bones
bpy.ops.armature.merge(type='WITHIN_CHAIN')
# Remove the parent-child relationship between selected bones and their parents
bpy.ops.armature.parent_clear(type='CLEAR')
# Set the active bone as the parent of the selected bones
bpy.ops.armature.parent_set(type='CONNECTED')
# Unhide all bones that have been tagged to be hidden in Edit Mode
bpy.ops.armature.reveal()
# Toggle selection status of all bones
bpy.ops.armature.select_all(action='TOGGLE')
# Select immediate parent/children of selected bones
bpy.ops.armature.select_hierarchy(direction='PARENT', extend=False)
# Flip the selection status of bones (selected -> unselected, unselected -> selected)
bpy.ops.armature.select_inverse()
# Select bones related to selected ones by parent/child relationships
bpy.ops.armature.select_linked(extend=False)
# Select similar bones by property types
bpy.ops.armature.select_similar(type='LENGTH', threshold=0.1)
# Isolate selected bones into a separate armature
bpy.ops.armature.separate()
# Break selected bones into chains of smaller bones
bpy.ops.armature.subdivide(number_cuts=1)
# Change the direction that a chain of bones points in (head <-> tail swap)
bpy.ops.armature.switch_direction()


# Add a boid rule to the current boid state
bpy.ops.boid.rule_add(type='GOAL')
# Delete current boid rule
bpy.ops.boid.rule_del()
# Move boid rule down in the list
bpy.ops.boid.rule_move_down()
# Move boid rule up in the list
bpy.ops.boid.rule_move_up()
# Add a boid state to the particle system
bpy.ops.boid.state_add()
# Delete current boid state
```

```
bpy.ops.boid.state_del()
# Move boid state down in the list
bpy.ops.boid.state_move_down()
# Move boid state up in the list
bpy.ops.boid.state_move_up()


# Set active sculpt/paint brush from it's number
bpy.ops.brush.active_index_set(mode="", index=0)
# Add brush by mode type
bpy.ops.brush.add()
# Set brush shape
bpy.ops.brush.curve_preset(shape='SMOOTH')
# Return brush to defaults based on current tool
bpy.ops.brush.reset()
# Change brush size by a scalar
bpy.ops.brush.scale_size(scalar=1)
# Control the stencil brush
bpy.ops.brush.stencil_control(mode='TRANSLATION', texmode='PRIMARY')
# When using an image texture, adjust the stencil size to fit the image aspect ratio
bpy.ops.brush.stencil_fit_image_aspect(use_repeat=True, use_scale=True, mask=False)
# Reset the stencil transformation to the default
bpy.ops.brush.stencil_reset_transform(mask=False)
# Set the UV sculpt tool
bpy.ops.brush.uv_sculpt_tool_set(tool='PINCH')


# Open a directory browser, Hold Shift to open the file, Alt to browse containing directory
bpy.ops.buttons.directory_browse(directory="", filter_blender=False, filter_backup=False, filter_image=False,
filter_movie=False, filter_python=False, filter_font=False, filter_sound=False, filter_text=False, filter_btx=False,
filter_collada=False, filter_folder=False, filemode=9, relative_path=True,
display_type='FILE_DEFAULTDISPLAY')
# Open a file browser, Hold Shift to open the file, Alt to browse containing directory
bpy.ops.buttons.file_browse(filepath="", filter_blender=False, filter_backup=False, filter_image=False,
filter_movie=False, filter_python=False, filter_font=False, filter_sound=False, filter_text=False, filter_btx=False,
filter_collada=False, filter_folder=False, filemode=9, relative_path=True,
display_type='FILE_DEFAULTDISPLAY')
# Display button panel toolbox
bpy.ops.buttons.toolbox()


# Add a Camera Preset
bpy.ops.camera.preset_add(remove_active=False, name="")


# Place new marker at specified location
bpy.ops.clip.add_marker(location=(0, 0))
# Place new marker at the desired (clicked) position
bpy.ops.clip.add_marker_at_click()
```

```
# Add new marker and move it on movie
bpy.ops.clip.add_marker_move(CLIP_OT_add_marker={"location":(0, 0)},
TRANSFORM_OT_translate={"value":(0, 0, 0), "constraint_axis":(False, False, False),
"constraint_orientation":'GLOBAL', "mirror":False, "proportional":'DISABLED',
"proportional_edit_falloff":'SMOOTH', "proportional_size":1, "snap":False, "snap_target":'CLOSEST',
"snap_point":(0, 0, 0), "snap_align":False, "snap_normal":(0, 0, 0), "texture_space":False, "release_confirm":False})
# Add new marker and slide it with mouse until mouse button release
bpy.ops.clip.add_marker_slide(CLIP_OT_add_marker={"location":(0, 0)},
TRANSFORM_OT_translate={"value":(0, 0, 0), "constraint_axis":(False, False, False),
"constraint_orientation":'GLOBAL', "mirror":False, "proportional":'DISABLED',
"proportional_edit_falloff":'SMOOTH', "proportional_size":1, "snap":False, "snap_target":'CLOSEST',
"snap_point":(0, 0, 0), "snap_align":False, "snap_normal":(0, 0, 0), "texture_space":False, "release_confirm":False})
# Apply scale on solution itself to make distance between selected tracks equals to desired
bpy.ops.clip.apply_solution_scale(distance=0)
# Create vertex cloud using coordinates of reconstructed tracks
bpy.ops.clip.bundles_to_mesh()
# Add a Tracking Camera Intrinsics Preset
bpy.ops.clip.camera_preset_add(remove_active=False, name="")
# Interactively change the current frame number
bpy.ops.clip.change_frame(frame=0)
# Clean tracks with high error values or few frames
bpy.ops.clip.clean_tracks(frames=0, error=0, action='SELECT')
# Clear all calculated data
bpy.ops.clip.clear_solution()
# Clear tracks after/before current position or clear the whole track
bpy.ops.clip.clear_track_path(action='REMAINED', clear_active=False)
# Create F-Curves for object which will copy object's movement caused by this constraint
bpy.ops.clip.constraint_to_fcurve()
# Copy selected tracks to clipboard
bpy.ops.clip.copy_tracks()
# Delete marker for current frame from selected tracks
bpy.ops.clip.delete_marker()
# Delete movie clip proxy files from the hard drive
bpy.ops.clip.delete_proxy()
# Delete selected tracks
bpy.ops.clip.delete_track()
# Automatically detect features and place markers to track
bpy.ops.clip.detect_features(placement='FRAME', margin=16, min_trackability=16, min_distance=120)
# Disable/enable selected markers
bpy.ops.clip.disable_markers(action='DISABLE')
# Select movie tracking channel
bpy.ops.clip.dopesheet_select_channel(location=(0, 0), extend=False)
# Reset viewable area to show full keyframe range
bpy.ops.clip.dopesheet_view_all()
# Jump to special frame
bpy.ops.clip.frame_jump(position='PATHSTART')
# Scroll view so current frame would be centered
bpy.ops.clip.graph_center_current_frame()
```

```
# Delete selected curves
bpy.ops.clip.graph_delete_curve()
# Delete curve knots
bpy.ops.clip.graph_delete_knot()
# Disable/enable selected markers
bpy.ops.clip.graph_disable_markers(action='DISABLE')
# Select graph curves
bpy.ops.clip.graph_select(location=(0, 0), extend=False)
# Change selection of all markers of active track
bpy.ops.clip.graph_select_all_markers(action='TOGGLE')
# Select curve points using border selection
bpy.ops.clip.graph_select_border(gesture_mode=0, xmin=0, xmax=0, ymin=0, ymax=0, extend=True)
# View all curves in editor
bpy.ops.clip.graph_view_all()
# Hide selected tracks
bpy.ops.clip.hide_tracks(unselected=False)
# Clear hide selected tracks
bpy.ops.clip.hide_tracks_clear()
# Join selected tracks
bpy.ops.clip.join_tracks()
# Lock/unlock selected tracks
bpy.ops.clip.lock_tracks(action='LOCK')
# Set the clip interaction mode
bpy.ops.clip.mode_set(mode='TRACKING')
# Load a sequence of frames or a movie file
bpy.ops.clip.open(directory="", files=[], filter_blender=False, filter_backup=False, filter_image=True,
filter_movie=True, filter_python=False, filter_font=False, filter_sound=False, filter_text=False, filter_btx=False,
filter_collada=False, filter_folder=True, filemode=9, relative_path=True, display_type='FILE_DEFAULTDISPLAY')
# Paste tracks from clipboard
bpy.ops.clip.paste_tracks()
# Prefetch frames from disk for faster playback/tracking
bpy.ops.clip.prefetch()
# Toggle clip properties panel
bpy.ops.clip.properties()
# Rebuild all selected proxies and timecode indices in the background
bpy.ops.clip.rebuild_proxy()
# Refine selected markers positions by running the tracker from track's reference to current frame
bpy.ops.clip.refine_markers(backwards=False)
# Reload clip
bpy.ops.clip.reload()
# Select tracking markers
bpy.ops.clip.select(extend=False, location=(0, 0))
# Change selection of all tracking markers
bpy.ops.clip.select_all(action='TOGGLE')
# Select markers using border selection
bpy.ops.clip.select_border(gesture_mode=0, xmin=0, xmax=0, ymin=0, ymax=0, extend=True)
# Select markers using circle selection
```

```
bpy.ops.clip.select_circle(x=0, y=0, radius=0, gesture_mode=0)
# Select all tracks from specified group
bpy.ops.clip.select_grouped(group='ESTIMATED')
# Select markers using lasso selection
bpy.ops.clip.select_lasso(path=[], deselect=False, extend=True)
# Set direction of scene axis rotating camera (or it's parent if present) and assuming selected track lies on real axis joining it with the origin
bpy.ops.clip.set_axis(axis='X')
# Set optical center to center of footage
bpy.ops.clip.set_center_principal()
# Set active marker as origin by moving camera (or it's parent if present) in 3D space
bpy.ops.clip.set_origin(use_median=False)
# Set plane based on 3 selected bundles by moving camera (or it's parent if present) in 3D space
bpy.ops.clip.set_plane(plane='FLOOR')
# Set scale of scene by scaling camera (or it's parent if present)
bpy.ops.clip.set_scale(distance=0)
# Set scene's start and end frame to match clip's start frame and length
bpy.ops.clip.set_scene_frames()
# Set object solution scale using distance between two selected tracks
bpy.ops.clip.set_solution_scale(distance=0)
# Set keyframe used by solver
bpy.ops.clip.set_solver_keyframe(keyframe='KEYFRAME_A')
# Set current movie clip as a camera background in 3D view-port (works only when a 3D view-port is visible)
bpy.ops.clip.set_viewport_background()
# Prepare scene for compositing 3D objects into this footage
bpy.ops.clip.setup_tracking_scene()
# Slide marker areas
bpy.ops.clip.slide_marker(offset=(0, 0))
# Solve camera motion from tracks
bpy.ops.clip.solve_camera()
# Add selected tracks to 2D stabilization tool
bpy.ops.clip.stabilize_2d_add()
# Remove selected track from stabilization
bpy.ops.clip.stabilize_2d_remove()
# Select track which are used for stabilization
bpy.ops.clip.stabilize_2d_select()
# Use active track to compensate rotation when doing 2D stabilization
bpy.ops.clip.stabilize_2d_set_rotation()
# Toggle clip tools panel
bpy.ops.clip.tools()
# Add a Clip Track Color Preset
bpy.ops.clip.track_color_preset_add(remove_active=False, name="")
# Copy color to all selected tracks
bpy.ops.clip.track_copy_color()
# Track selected markers
bpy.ops.clip.track_markers(backwards=False, sequence=False)
# Copy tracking settings from active track to default settings
```

```
bpy.ops.clip.track_settings_as_default()
# Create an Empty object which will be copying movement of active track
bpy.ops.clip.track_to_empty()
# Add new object for tracking
bpy.ops.clip.tracking_object_new()
# Remove object for tracking
bpy.ops.clip.tracking_object_remove()
# Add a motion tracking settings preset
bpy.ops.clip.tracking_settings_preset_add(remove_active=False, name="")
# View whole image with markers
bpy.ops.clip.view_all(fit_view=False)
# Use a 3D mouse device to pan/zoom the view
bpy.ops.clip.view_ndof()
# Pan the view
bpy.ops.clip.view_pan(offset=(0, 0))
# View all selected elements
bpy.ops.clip.view_selected()
# Zoom in/out the view
bpy.ops.clip.view_zoom(factor=0)
# Zoom in the view
bpy.ops.clip.view_zoom_in(location=(0, 0))
# Zoom out the view
bpy.ops.clip.view_zoom_out(location=(0, 0))
# Set the zoom ratio (based on clip size)
bpy.ops.clip.view_zoom_ratio(ratio=0)


# Add a Cloth Preset
bpy.ops.cloth.preset_add(remove_active=False, name="")


# Evaluate the namespace up until the cursor and give a list of options or complete the name if there is only one
bpy.ops.console.autocomplete()
# Print a message when the terminal initializes
bpy.ops.console.banner()
# Clear text by type
bpy.ops.console.clear(scrollback=True, history=False)
# Clear the line and store in history
bpy.ops.console.clear_line()
# Copy selected text to clipboard
bpy.ops.console.copy()
# Copy the console contents for use in a script
bpy.ops.console.copy_as_script()
# Delete text by cursor position
bpy.ops.console.delete(type='NEXT_CHARACTER')
# Execute the current console line as a python expression
bpy.ops.console.execute(interactive=False)
# Append history at cursor position
```

```
bpy.ops.console.history_append(text="", current_character=0, remove_duplicates=False)
# Cycle through history
bpy.ops.console.history_cycle(reverse=False)
# Add 4 spaces at line beginning
bpy.ops.console.indent()
# Insert text at cursor position
bpy.ops.console.insert(text="")
# Set the current language for this console
bpy.ops.console.language(language="")
# Move cursor position
bpy.ops.console.move(type='LINE_BEGIN')
# Paste text from clipboard
bpy.ops.console.paste()
# Append scrollback text by type
bpy.ops.console.scrollback_append(text="", type='OUTPUT')
# Set the console selection
bpy.ops.console.select_set()
# Delete 4 spaces from line beginning
bpy.ops.console.unindent()


# Clear inverse correction for ChildOf constraint
bpy.ops.constraint.childof_clear_inverse(constraint="", owner='OBJECT')
# Set inverse correction for ChildOf constraint
bpy.ops.constraint.childof_set_inverse(constraint="", owner='OBJECT')
# Remove constraint from constraint stack
bpy.ops.constraint.delete()
# Add default animation for path used by constraint if it isn't animated already
bpy.ops.constraint.followpath_path_animate(constraint="", owner='OBJECT', frame_start=1, length=100)
# Reset limiting distance for Limit Distance Constraint
bpy.ops.constraint.limitdistance_reset(constraint="", owner='OBJECT')
# Move constraint down in constraint stack
bpy.ops.constraint.move_down(constraint="", owner='OBJECT')
# Move constraint up in constraint stack
bpy.ops.constraint.move_up(constraint="", owner='OBJECT')
# Clear inverse correction for ObjectSolver constraint
bpy.ops.constraint.objectsolver_clear_inverse(constraint="", owner='OBJECT')
# Set inverse correction for ObjectSolver constraint
bpy.ops.constraint.objectsolver_set_inverse(constraint="", owner='OBJECT')
# Reset original length of bone for Stretch To Constraint
bpy.ops.constraint.stretchto_reset(constraint="", owner='OBJECT')


# Make active spline closed/opened loop
bpy.ops.curve.cyclic_toggle(direction='CYCLIC_U')
# (De)select first of visible part of each NURBS
bpy.ops.curve.de_select_first()
# (De)select last of visible part of each NURBS
```

bpy.ops.curve.de_select_last()

Delete selected control points or segments

bpy.ops.curve.delete(type='SELECTED')

Duplicate selected control points and segments between them

bpy.ops.curve.duplicate()

Duplicate curve and move

bpy.ops.curve.duplicate_move(CURVE_OT_duplicate={}, TRANSFORM_OT_translate={"value":(0, 0, 0), "constraint_axis":(False, False, False), "constraint_orientation":'GLOBAL', "mirror":False, "proportional":'DISABLED', "proportional_edit_falloff":'SMOOTH', "proportional_size":1, "snap":False, "snap_target":'CLOSEST', "snap_point":(0, 0, 0), "snap_align":False, "snap_normal":(0, 0, 0), "texture_space":False, "release_confirm":False})

Extrude selected control point(s)

bpy.ops.curve.extrude(mode='TRANSLATION')

Extrude curve and move result

bpy.ops.curve.extrude_move(CURVE_OT_extrude={"mode":'TRANSLATION'}, TRANSFORM_OT_translate={"value":(0, 0, 0), "constraint_axis":(False, False, False), "constraint_orientation":'GLOBAL', "mirror":False, "proportional":'DISABLED', "proportional_edit_falloff":'SMOOTH', "proportional_size":1, "snap":False, "snap_target":'CLOSEST', "snap_point":(0, 0, 0), "snap_align":False, "snap_normal":(0, 0, 0), "texture_space":False, "release_confirm":False})

Set type of handles for selected control points

bpy.ops.curve.handle_type_set(type='AUTOMATIC')

Hide (un)selected control points

bpy.ops.curve.hide(unselected=False)

Join two curves by their selected ends

bpy.ops.curve.make_segment()

Construct a Bezier Circle

bpy.ops.curve.primitive_bezier_circle_add(view_align=False, enter_editmode=False, location=(0, 0, 0), rotation=(0, 0, 0), layers=(False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False))

Construct a Bezier Curve

bpy.ops.curve.primitive_bezier_curve_add(view_align=False, enter_editmode=False, location=(0, 0, 0), rotation=(0, 0, 0), layers=(False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False))

Construct a Nurbs Circle

bpy.ops.curve.primitive_nurbs_circle_add(view_align=False, enter_editmode=False, location=(0, 0, 0), rotation=(0, 0, 0), layers=(False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False))

Construct a Nurbs Curve

bpy.ops.curve.primitive_nurbs_curve_add(view_align=False, enter_editmode=False, location=(0, 0, 0), rotation=(0, 0, 0), layers=(False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False))

Construct a Path

bpy.ops.curve.primitive_nurbs_path_add(view_align=False, enter_editmode=False, location=(0, 0, 0), rotation=(0, 0, 0), layers=(False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False))

Set per-point radius which is used for bevel tapering

bpy.ops.curve.radius_set(radius=1)

Show again hidden control points

bpy.ops.curve.reveal()

```
# (De)select all control points
bpy.ops.curve.select_all(action='TOGGLE')
# Reduce current selection by deselecting boundary elements
bpy.ops.curve.select_less()
# Select all control points linked to active one
bpy.ops.curve.select_linked()
# Select all control points linked to already selected ones
bpy.ops.curve.select_linked_pick(deselect=False)
# Select control points directly linked to already selected ones
bpy.ops.curve.select_more()
# Select control points following already selected ones along the curves
bpy.ops.curve.select_next()
# Deselect every other vertex
bpy.ops.curve.select_nth(nth=2)
# Select control points preceding already selected ones along the curves
bpy.ops.curve.select_previous()
# Randomly select some control points
bpy.ops.curve.select_random(percent=50, extend=False)
# Select a row of control points including active one
bpy.ops.curve.select_row()
# Separate (partly) selected curves or surfaces into a new object
bpy.ops.curve.separate()
# Set shading to flat
bpy.ops.curve.shade_flat()
# Set shading to smooth
bpy.ops.curve.shade_smooth()
# Flatten angles of selected points
bpy.ops.curve.smooth()
# Flatten radii of selected points
bpy.ops.curve.smooth_radius()
# Extrude selected boundary row around pivot point and current view axis
bpy.ops.curve.spin(center=(0, 0, 0), axis=(0, 0, 0))
# Set type of active spline
bpy.ops.curve.spline_type_set(type='POLY', use_handles=False)
# Set softbody goal weight for selected points
bpy.ops.curve.spline_weight_set(weight=1)
# Subdivide selected segments
bpy.ops.curve.subdivide(number_cuts=1)
# Switch direction of selected splines
bpy.ops.curve.switch_direction()
# Clear the tilt of selected control points
bpy.ops.curve.tilt_clear()
# Add a new control point (linked to only selected end-curve one, if any)
bpy.ops.curve.vertex_add(location=(0, 0, 0))


# Enable nodes on a material, world or lamp
```

bpy.ops.cycles.use_shading_nodes()

# Bake dynamic paint image sequence surface
bpy.ops.dpaint.bake()
# Add or remove Dynamic Paint output data layer
bpy.ops.dpaint.output_toggle(output='A')
# Add a new Dynamic Paint surface slot
bpy.ops.dpaint.surface_slot_add()
# Remove the selected surface slot
bpy.ops.dpaint.surface_slot_remove()
# Toggle whether given type is active or not
bpy.ops.dpaint.type_toggle(type='CANVAS')

# Redo previous action
bpy.ops.ed.redo()
# Undo previous action
bpy.ops.ed.undo()
# Redo specific action in history
bpy.ops.ed.undo_history(item=0)
# Add an undo state (internal use only)
bpy.ops.ed.undo_push(message="Add an undo step *function may be moved*")

# Save a BVH motion capture file from an armature
bpy.ops.export_anim.bvh(check_existing=True, filepath="", filter_glob="*.bvh", global_scale=1, frame_start=0, frame_end=0, rotate_mode='NATIVE', root_transform_only=False)

# Export a single object as a Stanford PLY with normals, colors and texture coordinates
bpy.ops.export_mesh.ply(check_existing=True, filepath="", filter_glob="*.ply", use_mesh_modifiers=True, use_normals=True, use_uv_coords=True, use_colors=True, axis_forward='Y', axis_up='Z', global_scale=1)
# Save STL triangle mesh data from the active object
bpy.ops.export_mesh.stl(check_existing=True, filepath="", filter_glob="*.stl", ascii=False, use_mesh_modifiers=True, axis_forward='Y', axis_up='Z', global_scale=1)

# Export to 3DS file format (.3ds)
bpy.ops.export_scene.autodesk_3ds(check_existing=True, filepath="", filter_glob="*.3ds", use_selection=False, axis_forward='Y', axis_up='Z')
# Selection to an ASCII Autodesk FBX
bpy.ops.export_scene.fbx(check_existing=True, filepath="", filter_glob="*.fbx", use_selection=False, global_scale=1, axis_forward='-Z', axis_up='Y', object_types={'EMPTY', 'CAMERA', 'LAMP', 'ARMATURE', 'MESH'}, use_mesh_modifiers=True, mesh_smooth_type='FACE', use_mesh_edges=False, use_armature_deform_only=False, use_anim=True, use_anim_action_all=True, use_default_take=True, use_anim_optimize=True, anim_optimize_precision=6, path_mode='AUTO', use_rotate_workaround=False, xna_validate=False, batch_mode='OFF', use_batch_own_dir=True, use_metadata=True)
# Save a Wavefront OBJ File

bpy.ops.export_scene.obj(check_existing=True, filepath="", filter_glob="*.obj;*.mtl", use_selection=False, use_animation=False, use_mesh_modifiers=True, use_edges=True, use_smooth_groups=False, use_normals=False, use_uvs=True, use_materials=True, use_triangles=False, use_nurbs=False, use_vertex_groups=False, use_blen_objects=True, group_by_object=False, group_by_material=False, keep_vertex_order=False, axis_forward='-Z', axis_up='Y', global_scale=1, path_mode='AUTO')
# Export selection to Extensible 3D file (.x3d)
bpy.ops.export_scene.x3d(check_existing=True, filepath="", filter_glob="*.x3d", use_selection=False, use_mesh_modifiers=True, use_triangulate=False, use_normals=False, use_compress=False, use_hierarchy=True, name_decorations=True, use_h3d=False, axis_forward='Z', axis_up='Y', global_scale=1, path_mode='AUTO')

# Add a bookmark for the selected/active directory
bpy.ops.file.bookmark_add()
# Toggle bookmarks display
bpy.ops.file.bookmark_toggle()
# Cancel loading of selected file
bpy.ops.file.cancel()
# Delete selected files
bpy.ops.file.delete()
# Delete selected bookmark
bpy.ops.file.delete_bookmark(index=-1)
# Enter a directory name
bpy.ops.file.directory()
# Create a new directory
bpy.ops.file.directory_new(directory="")
# Execute selected file
bpy.ops.file.execute(need_active=False)
# Increment number in filename
bpy.ops.file.filenum(increment=1)
# Try to find missing external files
bpy.ops.file.find_missing_files(find_all=False, directory="", filter_blender=False, filter_backup=False, filter_image=False, filter_movie=False, filter_python=False, filter_font=False, filter_sound=False, filter_text=False, filter_btx=False, filter_collada=False, filter_folder=False, filemode=9, display_type='FILE_DEFAULTDISPLAY')
# Toggle hide hidden dot files
bpy.ops.file.hidedot()
# Highlight selected file(s)
bpy.ops.file.highlight()
# Make all paths to external files absolute
bpy.ops.file.make_paths_absolute()
# Make all paths to external files relative to current .blend
bpy.ops.file.make_paths_relative()
# Move to next folder
bpy.ops.file.next()
# Pack all used external files into the .blend
bpy.ops.file.pack_all()
# Pack all used Blender library files into the current .blend
bpy.ops.file.pack_libraries()
# Move to parent directory

```
bpy.ops.file.parent()
# Move to previous folder
bpy.ops.file.previous()
# Refresh the file list
bpy.ops.file.refresh()
# Rename file or file directory
bpy.ops.file.rename()
# Report all missing external files
bpy.ops.file.report_missing_files()
# Reset Recent files
bpy.ops.file.reset_recent()
# Activate/select file
bpy.ops.file.select(extend=False, fill=False, open=True)
# Select or deselect all files
bpy.ops.file.select_all_toggle()
# Select a bookmarked directory
bpy.ops.file.select_bookmark(dir="")
# Activate/select the file(s) contained in the border
bpy.ops.file.select_border(gesture_mode=0, xmin=0, xmax=0, ymin=0, ymax=0, extend=True)
# Smooth scroll to make editable file visible
bpy.ops.file.smoothscroll()
# Unpack all files packed into this .blend to external ones
bpy.ops.file.unpack_all(method='USE_LOCAL')
# Unpack this file to an external file
bpy.ops.file.unpack_item(method='USE_LOCAL', id_name="", id_type=19785)
# Unpack all used Blender library files from this .blend file
bpy.ops.file.unpack_libraries()


# Bake fluid simulation
bpy.ops.fluid.bake()
# Add a Fluid Preset
bpy.ops.fluid.preset_add(remove_active=False, name="")


# Set font case
bpy.ops.font.case_set(case='LOWER')
# Toggle font case
bpy.ops.font.case_toggle()
# Change font character code
bpy.ops.font.change_character(delta=1)
# Change font spacing
bpy.ops.font.change_spacing(delta=1)
# Delete text by cursor position
bpy.ops.font.delete(type='ALL')
# Paste contents from file
bpy.ops.font.file_paste(filepath="", filter_blender=False, filter_backup=False, filter_image=False,
filter_movie=False, filter_python=False, filter_font=False, filter_sound=False, filter_text=True, filter_btx=False,
```

filter_collada=False, filter_folder=True, filemode=9, display_type='FILE_DEFAULTDISPLAY')
# Insert placeholder text
bpy.ops.font.insert_lorem()
# Insert line break at cursor position
bpy.ops.font.line_break()
# Move cursor to position type
bpy.ops.font.move(type='LINE_BEGIN')
# Make selection from current cursor position to new cursor position type
bpy.ops.font.move_select(type='LINE_BEGIN')
# Load a new font from a file
bpy.ops.font.open(filepath="", filter_blender=False, filter_backup=False, filter_image=False, filter_movie=False, filter_python=False, filter_font=True, filter_sound=False, filter_text=False, filter_btx=False, filter_collada=False, filter_folder=True, filemode=9, relative_path=True, display_type='FILE_DEFAULTDISPLAY')
# Set font style
bpy.ops.font.style_set(style='BOLD', clear=False)
# Toggle font style
bpy.ops.font.style_toggle(style='BOLD')
# Copy selected text to clipboard
bpy.ops.font.text_copy()
# Cut selected text to clipboard
bpy.ops.font.text_cut()
# Insert text at cursor position
bpy.ops.font.text_insert(text="", accent=False)
# Paste text from clipboard
bpy.ops.font.text_paste()
# Add a new text box
bpy.ops.font.textbox_add()
# Remove the textbox
bpy.ops.font.textbox_remove(index=0)
# Unlink active font data block
bpy.ops.font.unlink()

# Delete the active frame for the active Grease Pencil datablock
bpy.ops.gpencil.active_frame_delete()
# Convert the active Grease Pencil layer to a new Curve Object
bpy.ops.gpencil.convert(type='PATH', use_normalize_weights=True, radius_multiplier=1, use_link_strokes=True, timing_mode='<UNKNOWN ENUM>', frame_range=100, start_frame=1, use_realtime=False, end_frame=250, gap_duration=0, gap_randomness=0, seed=0, use_timing_data=False)
# Add new Grease Pencil datablock
bpy.ops.gpencil.data_add()
# Unlink active Grease Pencil datablock
bpy.ops.gpencil.data_unlink()
# Make annotations on the active data
bpy.ops.gpencil.draw(mode='DRAW', stroke=[])
# Add new Grease Pencil layer for the active Grease Pencil datablock
bpy.ops.gpencil.layer_add()

```
# Bake selected F-Curves to a set of sampled points defining a similar curve
bpy.ops.graph.bake()
# Simplify F-Curves by removing closely spaced keyframes
bpy.ops.graph.clean(threshold=0.001)
# Insert new keyframe at the cursor position for the active F-Curve
bpy.ops.graph.click_insert(frame=1, value=1)
# Select keyframes by clicking on them
bpy.ops.graph.clickselect(extend=False, column=False, curves=False)
# Copy selected keyframes to the copy/paste buffer
bpy.ops.graph.copy()
# Interactively set the current frame number and value cursor
bpy.ops.graph.cursor_set(frame=0, value=0)
# Remove all selected keyframes
bpy.ops.graph.delete()
# Make a copy of all selected keyframes
bpy.ops.graph.duplicate(mode='TRANSLATION')
# Make a copy of all selected keyframes and move them
bpy.ops.graph.duplicate_move(GRAPH_OT_duplicate={"mode":'TRANSLATION'},
TRANSFORM_OT_transform={"mode":'TRANSLATION', "value":(0, 0, 0, 0), "axis":(0, 0, 0),
"constraint_axis":(False, False, False), "constraint_orientation":'GLOBAL', "mirror":False,
"proportional":'DISABLED', "proportional_edit_falloff":'SMOOTH', "proportional_size":1, "snap":False,
"snap_target":'CLOSEST', "snap_point":(0, 0, 0), "snap_align":False, "snap_normal":(0, 0, 0),
"release_confirm":False})
# Fix large jumps and flips in the selected Euler Rotation F-Curves arising from rotation values being clipped when
baking physics
bpy.ops.graph.euler_filter()
# Set extrapolation mode for selected F-Curves
bpy.ops.graph.extrapolation_type(type='CONSTANT')
# Add F-Modifiers to the selected F-Curves
bpy.ops.graph.fmodifier_add(type='NULL', only_active=True)
# Copy the F-Modifier(s) of the active F-Curve
bpy.ops.graph.fmodifier_copy()
# Add copied F-Modifiers to the selected F-Curves
bpy.ops.graph.fmodifier_paste()
# Place the cursor on the midpoint of selected keyframes
bpy.ops.graph.frame_jump()
# Clear F-Curve snapshots (Ghosts) for active Graph Editor
bpy.ops.graph.ghost_curves_clear()
# Create snapshot (Ghosts) of selected F-Curves as background aid for active Graph Editor
bpy.ops.graph.ghost_curves_create()
# Set type of handle for selected keyframes
bpy.ops.graph.handle_type(type='FREE')
# Set interpolation mode for the F-Curve segments starting from the selected keyframes
bpy.ops.graph.interpolation_type(type='CONSTANT')
# Insert keyframes for the specified channels
bpy.ops.graph.keyframe_insert(type='ALL')
# Flip selected keyframes over the selected mirror line
bpy.ops.graph.mirror(type='CFRA')
```

```
# Paste keyframes from copy/paste buffer for the selected channels, starting on the current frame
bpy.ops.graph.paste(offset='START', merge='MIX')
# Automatically set Preview Range based on range of keyframes
bpy.ops.graph.previewrange_set()
# Toggle display properties panel
bpy.ops.graph.properties()
# Add keyframes on every frame between the selected keyframes
bpy.ops.graph.sample()
# Toggle selection of all keyframes
bpy.ops.graph.select_all_toggle(invert=False)
# Select all keyframes within the specified region
bpy.ops.graph.select_border(gesture_mode=0, xmin=0, xmax=0, ymin=0, ymax=0, extend=True, axis_range=False,
include_handles=False)
# Select all keyframes on the specified frame(s)
bpy.ops.graph.select_column(mode='KEYS')
# Select keyframes to the left or the right of the current frame
bpy.ops.graph.select_leftright(mode='CHECK', extend=False)
# Deselect keyframes on ends of selection islands
bpy.ops.graph.select_less()
# Select keyframes occurring in the same F-Curves as selected ones
bpy.ops.graph.select_linked()
# Select keyframes beside already selected ones
bpy.ops.graph.select_more()
# Apply weighted moving means to make selected F-Curves less bumpy
bpy.ops.graph.smooth()
# Snap selected keyframes to the chosen times/values
bpy.ops.graph.snap(type='CFRA')
# Bakes a sound wave to selected F-Curves
bpy.ops.graph.sound_bake(filepath="", filter_blender=False, filter_backup=False, filter_image=False,
filter_movie=True, filter_python=False, filter_font=False, filter_sound=True, filter_text=False, filter_btx=False,
filter_collada=False, filter_folder=True, filemode=9, display_type='FILE_DEFAULTDISPLAY', low=0,
high=100000, attack=0.005, release=0.2, threshold=0, use_accumulate=False, use_additive=False, use_square=False,
sthreshold=0.1)
# Reset viewable area to show full keyframe range
bpy.ops.graph.view_all(include_handles=True)
# Reset viewable area to show selected keyframe range
bpy.ops.graph.view_selected(include_handles=True)


# Create an object group from selected objects
bpy.ops.group.create(name="Group")
# Add the object to an object group that contains the active object
bpy.ops.group.objects_add_active(group='<UNKNOWN ENUM>')
# Remove selected objects from all groups or a selected group
bpy.ops.group.objects_remove(group='<UNKNOWN ENUM>')
# Remove the object from an object group that contains the active object
bpy.ops.group.objects_remove_active()
# Remove selected objects from all groups or a selected group
```

bpy.ops.group.objects_remove_all()

# Set black point or white point for curves
bpy.ops.image.curves_point_set(point='BLACK_POINT')
# Cycle through all non-void render slots
bpy.ops.image.cycle_render_slot(reverse=False)
# Edit image in an external application
bpy.ops.image.external_edit(filepath="")
# Invert image's channels
bpy.ops.image.invert(invert_r=False, invert_g=False, invert_b=False, invert_a=False)
# Set image's user's length to the one of this video
bpy.ops.image.match_movie_length()
# Create a new image
bpy.ops.image.new(name="Untitled", width=1024, height=1024, color=(0, 0, 0, 1), alpha=True, generated_type='BLANK', float=False)
# Open image
bpy.ops.image.open(filepath="", filter_blender=False, filter_backup=False, filter_image=True, filter_movie=True, filter_python=False, filter_font=False, filter_sound=False, filter_text=False, filter_btx=False, filter_collada=False, filter_folder=True, filemode=9, relative_path=True, display_type='FILE_DEFAULTDISPLAY')
# Pack an image as embedded data into the .blend file
bpy.ops.image.pack(as_png=False)
# Project edited image back onto the object
bpy.ops.image.project_apply()
# Edit a snapshot of the view-port in an external image editor
bpy.ops.image.project_edit()
# Toggle display properties panel
bpy.ops.image.properties()
# Reload current image from disk
bpy.ops.image.reload()
# Replace current image by another one from disk
bpy.ops.image.replace(filepath="", filter_blender=False, filter_backup=False, filter_image=True, filter_movie=True, filter_python=False, filter_font=False, filter_sound=False, filter_text=False, filter_btx=False, filter_collada=False, filter_folder=True, filemode=9, relative_path=True, display_type='FILE_DEFAULTDISPLAY')
# Use mouse to sample a color in current image
bpy.ops.image.sample()
# Sample a line and show it in Scope panels
bpy.ops.image.sample_line(xstart=0, xend=0, ystart=0, yend=0, cursor=1)
# Save the image with current name and settings
bpy.ops.image.save()
# Save the image with another name and/or settings
bpy.ops.image.save_as(save_as_render=False, copy=False, filepath="", check_existing=True, filter_blender=False, filter_backup=False, filter_image=True, filter_movie=True, filter_python=False, filter_font=False, filter_sound=False, filter_text=False, filter_btx=False, filter_collada=False, filter_folder=True, filemode=9, relative_path=True, display_type='FILE_DEFAULTDISPLAY')
# Save all modified textures
bpy.ops.image.save_dirty()
# Save a sequence of images

```
bpy.ops.image.save_sequence()
# Toggle display scopes panel
bpy.ops.image.scopes()
# Save an image packed in the .blend file to disk
bpy.ops.image.unpack(method='USE_LOCAL', id="")
# View the entire image
bpy.ops.image.view_all()
# Use a 3D mouse device to pan/zoom the view
bpy.ops.image.view_ndof()
# Pan the view
bpy.ops.image.view_pan(offset=(0, 0))
# View all selected UVs
bpy.ops.image.view_selected()
# Zoom in/out the image
bpy.ops.image.view_zoom(factor=0)
# Zoom in the image (centered around 2D cursor)
bpy.ops.image.view_zoom_in(location=(0, 0))
# Zoom out the image (centered around 2D cursor)
bpy.ops.image.view_zoom_out(location=(0, 0))
# Set zoom ratio of the view
bpy.ops.image.view_zoom_ratio(ratio=0)


# Load a BVH motion capture file
bpy.ops.import_anim.bvh(filepath="", filter_glob="*.bvh", target='ARMATURE', global_scale=1, frame_start=1,
use_fps_scale=False, use_cyclic=False, rotate_mode='NATIVE', axis_forward='-Z', axis_up='Y')


# Load a SVG file
bpy.ops.import_curve.svg(filepath="", filter_glob="*.svg")


# Load a PLY geometry file
bpy.ops.import_mesh.ply(filepath="", files=[], directory="", filter_glob="*.ply")
# Load STL triangle mesh data
bpy.ops.import_mesh.stl(filepath="", filter_glob="*.stl", files=[], directory="")


# Import from 3DS file format (.3ds)
bpy.ops.import_scene.autodesk_3ds(filepath="", filter_glob="*.3ds", constrain_size=10, use_image_search=True,
use_apply_transform=True, axis_forward='Y', axis_up='Z')
# Load a Wavefront OBJ File
bpy.ops.import_scene.obj(filepath="", filter_glob="*.obj;*.mtl", use_ngons=True, use_edges=True,
use_smooth_groups=True, use_split_objects=True, use_split_groups=True, use_groups_as_vgroups=False,
use_image_search=True, split_mode='ON', global_clamp_size=0, axis_forward='-Z', axis_up='Y')
# Import an X3D or VRML2 file
bpy.ops.import_scene.x3d(filepath="", filter_glob="*.x3d;*.wrl", axis_forward='Z', axis_up='Y')
```

```
# Copy selected reports to Clipboard
bpy.ops.info.report_copy()
# Delete selected reports
bpy.ops.info.report_delete()
# Replay selected reports
bpy.ops.info.report_replay()
# Update the display of reports in Blender UI (internal use)
bpy.ops.info.reports_display_update()
# Select or deselect all reports
bpy.ops.info.select_all_toggle()
# Toggle border selection
bpy.ops.info.select_border(gesture_mode=0, xmin=0, xmax=0, ymin=0, ymax=0, extend=True)
# Select reports by index
bpy.ops.info.select_pick(report_index=0)


# Add a Sky & Atmosphere Preset
bpy.ops.lamp.sunsky_preset_add(remove_active=False, name="")


# Mirror all control points without inverting the lattice deform
bpy.ops.lattice.flip(axis='U')
# Set UVW control points a uniform distance apart
bpy.ops.lattice.make_regular()
# Change selection of all UVW control points
bpy.ops.lattice.select_all(action='TOGGLE')
# Select vertices without a group
bpy.ops.lattice.select_ungrouped(extend=False)


# Add an actuator to the active object
bpy.ops.logic.actuator_add(type='MOTION', name="", object="")
# Move Actuator
bpy.ops.logic.actuator_move(actuator="", object="", direction='UP')
# Remove an actuator from the active object
bpy.ops.logic.actuator_remove(actuator="", object="")
# Add a controller to the active object
bpy.ops.logic.controller_add(type='LOGIC_AND', name="", object="")
# Move Controller
bpy.ops.logic.controller_move(controller="", object="", direction='UP')
# Remove a controller from the active object
bpy.ops.logic.controller_remove(controller="", object="")
# Remove logic brick connections
bpy.ops.logic.links_cut(path=[], cursor=9)
# Toggle display properties panel
bpy.ops.logic.properties()
# Add a sensor to the active object
bpy.ops.logic.sensor_add(type='ALWAYS', name="", object="")
```

```
# Move Sensor
bpy.ops.logic.sensor_move(sensor="", object="", direction='UP')
# Remove a sensor from the active object
bpy.ops.logic.sensor_remove(sensor="", object="")
# Convert old texface settings into material. It may create new materials if needed
bpy.ops.logic.texface_convert()
# Resize view so you can see all logic bricks
bpy.ops.logic.view_all()


# Add a new time marker
bpy.ops.marker.add()
# Bind the active camera to selected markers(s)
bpy.ops.marker.camera_bind()
# Delete selected time marker(s)
bpy.ops.marker.delete()
# Duplicate selected time marker(s)
bpy.ops.marker.duplicate(frames=0)
# Copy selected markers to another scene
bpy.ops.marker.make_links_scene(scene='Scene')
# Move selected time marker(s)
bpy.ops.marker.move(frames=0)
# Rename first selected time marker
bpy.ops.marker.rename(name="RenamedMarker")
# Select time marker(s)
bpy.ops.marker.select(extend=False, camera=False)
# Change selection of all time markers
bpy.ops.marker.select_all(action='TOGGLE')
# Select all time markers using border selection
bpy.ops.marker.select_border(gesture_mode=0, xmin=0, xmax=0, ymin=0, ymax=0, extend=True)


# Add vertex to feather
bpy.ops.mask.add_feather_vertex(location=(0, 0))
# Add new vertex to feather and slide it
bpy.ops.mask.add_feather_vertex_slide(MASK_OT_add_feather_vertex={"location":(0, 0)},
MASK_OT_slide_point={"slide_feather":False})
# Add vertex to active spline
bpy.ops.mask.add_vertex(location=(0, 0))
# Add new vertex and slide it
bpy.ops.mask.add_vertex_slide(MASK_OT_add_vertex={"location":(0, 0)},
TRANSFORM_OT_translate={"value":(0, 0, 0), "constraint_axis":(False, False, False),
"constraint_orientation":'GLOBAL', "mirror":False, "proportional":'DISABLED',
"proportional_edit_falloff":'SMOOTH', "proportional_size":1, "snap":False, "snap_target":'CLOSEST',
"snap_point":(0, 0, 0), "snap_align":False, "snap_normal":(0, 0, 0), "texture_space":False, "release_confirm":False})
# Toggle cyclic for selected splines
bpy.ops.mask.cyclic_toggle()
# Delete selected control points or splines
```

```
bpy.ops.mask.delete()
# Reset the feather weight to zero
bpy.ops.mask.feather_weight_clear()
# Set type of handles for selected control points
bpy.ops.mask.handle_type_set(type='AUTO')
# Reveal the layer by setting the hide flag
bpy.ops.mask.hide_view_clear()
# Hide the layer by setting the hide flag
bpy.ops.mask.hide_view_set(unselected=False)
# Move the active layer up/down in the list
bpy.ops.mask.layer_move(direction='UP')
# Add new mask layer for masking
bpy.ops.mask.layer_new(name="")
# Remove mask layer
bpy.ops.mask.layer_remove()
# Create new mask
bpy.ops.mask.new(name="")
# Re-calculate the direction of selected handles
bpy.ops.mask.normals_make_consistent()
# Clear the mask's parenting
bpy.ops.mask.parent_clear()
# Set the mask's parenting
bpy.ops.mask.parent_set()
# Select spline points
bpy.ops.mask.select(extend=False, deselect=False, toggle=False, location=(0, 0))
# Change selection of all curve points
bpy.ops.mask.select_all(action='TOGGLE')
# Select markers using border selection
bpy.ops.mask.select_border(gesture_mode=0, xmin=0, xmax=0, ymin=0, ymax=0, extend=True)
# Select markers using circle selection
bpy.ops.mask.select_circle(x=0, y=0, radius=0, gesture_mode=0)
# Select markers using lasso selection
bpy.ops.mask.select_lasso(path=[], deselect=False, extend=True)
# Select all vertices linked to the active mesh
bpy.ops.mask.select_linked()
# (De)select all points linked to the curve under the mouse cursor
bpy.ops.mask.select_linked_pick(deselect=False)
#
bpy.ops.mask.shape_key_clear()
# Reset feather weights on all selected points animation values
bpy.ops.mask.shape_key_feather_reset()
#
bpy.ops.mask.shape_key_insert()
# Recalculate animation data on selected points for frames selected in the dopesheet
bpy.ops.mask.shape_key_rekey(location=True, feather=True)
# Slide control points
bpy.ops.mask.slide_point(slide_feather=False)
```

```
# Switch direction of selected splines
bpy.ops.mask.switch_direction()


# Copy the material settings and nodes
bpy.ops.material.copy()
# Add a new material
bpy.ops.material.new()
# Paste the material settings and nodes
bpy.ops.material.paste()
# Add a Subsurface Scattering Preset
bpy.ops.material.sss_preset_add(remove_active=False, name="")


# Delete selected metaelement(s)
bpy.ops.mball.delete_metaelems()
# Duplicate selected metaelement(s)
bpy.ops.mball.duplicate_metaelems(mode='TRANSLATION')
# Hide (un)selected metaelement(s)
bpy.ops.mball.hide_metaelems(unselected=False)
# Reveal all hidden metaelements
bpy.ops.mball.reveal_metaelems()
# Change selection of all meta elements
bpy.ops.mball.select_all(action='TOGGLE')
# Randomly select metaelements
bpy.ops.mball.select_random_metaelems(percent=0.5)


# Rearrange some faces to try to get less degenerated geometry
bpy.ops.mesh.beautify_fill()
# Edge Bevel
bpy.ops.mesh.bevel(offset=0, segments=1, vertex_only=False)
# Blend in shape from a shape key
bpy.ops.mesh.blend_from_shape(shape='<UNKNOWN ENUM>', blend=1, add=True)
# Make faces between two or more edge loops
bpy.ops.mesh.bridge_edge_loops(type='SINGLE', use_merge=False, merge_factor=0.5, number_cuts=0,
interpolation='PATH', smoothness=1, profile_shape_factor=0, profile_shape='SMOOTH')
# Flip direction of vertex colors inside faces
bpy.ops.mesh.colors_reverse()
# Rotate vertex colors inside faces
bpy.ops.mesh.colors_rotate(use_ccw=False)
# Enclose selected vertices in a convex polyhedron
bpy.ops.mesh.convex_hull(delete_unused=True, use_existing_faces=True, make_holes=False, join_triangles=True,
limit=0.698132, uvs=False, vcols=False, sharp=False, materials=False)
# Clear vertex sculpt masking data from the mesh
bpy.ops.mesh.customdata_clear_mask()
# Clear vertex skin layer
bpy.ops.mesh.customdata_clear_skin()
```

```
# Delete selected vertices, edges or faces
bpy.ops.mesh.delete(type='VERT')
# Delete an edge loop by merging the faces on each side
bpy.ops.mesh.delete_edgeloop(use_face_split=True)
# Dissolve geometry
bpy.ops.mesh.dissolve_edges(use_verts=False, use_face_split=False)
# Dissolve geometry
bpy.ops.mesh.dissolve_faces(use_verts=False)
# Dissolve selected edges and verts, limited by the angle of surrounding geometry
bpy.ops.mesh.dissolve_limited(angle_limit=0.0872665, use_dissolve_boundaries=False, delimit=set())
# Dissolve geometry
bpy.ops.mesh.dissolve_verts(use_face_split=False)
# Assign Image to active UV Map, or create an UV Map
bpy.ops.mesh.drop_named_image(name="Image", filepath="Path")
# Duplicate and extrude selected vertices, edges or faces towards the mouse cursor
bpy.ops.mesh.dupli_extrude_cursor(rotate_source=True)
# Duplicate selected vertices, edges or faces
bpy.ops.mesh.duplicate(mode=1)
# Duplicate mesh and move
bpy.ops.mesh.duplicate_move(MESH_OT_duplicate={"mode":1}, TRANSFORM_OT_translate={"value":(0, 0, 0),
"constraint_axis":(False, False, False), "constraint_orientation":'GLOBAL', "mirror":False,
"proportional":'DISABLED', "proportional_edit_falloff":'SMOOTH', "proportional_size":1, "snap":False,
"snap_target":'CLOSEST', "snap_point":(0, 0, 0), "snap_align":False, "snap_normal":(0, 0, 0),
"texture_space":False, "release_confirm":False})
# Collapse selected edges
bpy.ops.mesh.edge_collapse()
# Add an edge or face to selected
bpy.ops.mesh.edge_face_add()
# Rotate selected edge or adjoining faces
bpy.ops.mesh.edge_rotate(use_ccw=False)
# Split selected edges so that each neighbor face gets its own copy
bpy.ops.mesh.edge_split()
# Select an edge ring
bpy.ops.mesh.edgering_select(extend=False, deselect=False, toggle=False, ring=True)
# Select all sharp-enough edges
bpy.ops.mesh.edges_select_sharp(sharpness=0.523599)
# Extrude individual edges only
bpy.ops.mesh.extrude_edges_indiv(mirror=False)
# Extrude edges and move result
bpy.ops.mesh.extrude_edges_move(MESH_OT_extrude_edges_indiv={"mirror":False},
TRANSFORM_OT_translate={"value":(0, 0, 0), "constraint_axis":(False, False, False),
"constraint_orientation":'GLOBAL', "mirror":False, "proportional":'DISABLED',
"proportional_edit_falloff":'SMOOTH', "proportional_size":1, "snap":False, "snap_target":'CLOSEST',
"snap_point":(0, 0, 0), "snap_align":False, "snap_normal":(0, 0, 0), "texture_space":False, "release_confirm":False})
# Extrude individual faces only
bpy.ops.mesh.extrude_faces_indiv(mirror=False)
# Extrude faces and move result
bpy.ops.mesh.extrude_faces_move(MESH_OT_extrude_faces_indiv={"mirror":False},
```

TRANSFORM_OT_shrink_fatten={“value”:0, “mirror”:False, “proportional”:’DISABLED’, “proportional_edit_falloff”:’SMOOTH’, “proportional_size”:1, “snap”:False, “snap_target”:’CLOSEST’, “snap_point”:(0, 0, 0), “snap_align”:False, “snap_normal”:(0, 0, 0), “release_confirm”:False})

Extrude region of faces

bpy.ops.mesh.extrude_region(mirror=False)

Extrude region and move result

bpy.ops.mesh.extrude_region_move(MESH_OT_extrude_region={“mirror”:False}, TRANSFORM_OT_translate={“value”:(0, 0, 0), “constraint_axis”:(False, False, False), “constraint_orientation”:’GLOBAL’, “mirror”:False, “proportional”:’DISABLED’, “proportional_edit_falloff”:’SMOOTH’, “proportional_size”:1, “snap”:False, “snap_target”:’CLOSEST’, “snap_point”:(0, 0, 0), “snap_align”:False, “snap_normal”:(0, 0, 0), “texture_space”:False, “release_confirm”:False})

Extrude selected vertices, edges or faces repeatedly

bpy.ops.mesh.extrude_repeat(offset=2, steps=10)

Extrude vertices and move result

bpy.ops.mesh.extrude_vertices_move(MESH_OT_extrude_verts_indiv={“mirror”:False}, TRANSFORM_OT_translate={“value”:(0, 0, 0), “constraint_axis”:(False, False, False), “constraint_orientation”:’GLOBAL’, “mirror”:False, “proportional”:’DISABLED’, “proportional_edit_falloff”:’SMOOTH’, “proportional_size”:1, “snap”:False, “snap_target”:’CLOSEST’, “snap_point”:(0, 0, 0), “snap_align”:False, “snap_normal”:(0, 0, 0), “texture_space”:False, “release_confirm”:False})

Extrude individual vertices only

bpy.ops.mesh.extrude_verts_indiv(mirror=False)

Copy mirror UV coordinates on the X axis based on a mirrored mesh

bpy.ops.mesh.faces_mirror_uv(direction=’POSITIVE’, precision=3)

Select linked faces by angle

bpy.ops.mesh.faces_select_linked_flat(sharpness=0.0174533)

Display faces flat

bpy.ops.mesh.faces_shade_flat()

Display faces smooth (using vertex normals)

bpy.ops.mesh.faces_shade_smooth()

Fill a selected edge loop with faces

bpy.ops.mesh.fill(use_beauty=True)

Fill grid from two loops

bpy.ops.mesh.fill_grid()

Flip the direction of selected faces’ normals (and of their vertices)

bpy.ops.mesh.flip_normals()

Hide (un)selected vertices, edges or faces

bpy.ops.mesh.hide(unselected=False)

Inset new faces into selected faces

bpy.ops.mesh.inset(use_boundary=True, use_even_offset=True, use_relative_offset=False, thickness=0.01, depth=0, use_outset=False, use_select_inset=True, use_individual=False, use_interpolate=True)

Use other objects outlines & boundaries to project knife cuts

bpy.ops.mesh.knife_project()

Cut new topology

bpy.ops.mesh.knife_tool(use_occlude_geometry=True, only_selected=False)

Select a loop of connected edges by connection type

bpy.ops.mesh.loop_multi_select(ring=False)

Select a loop of connected edges

bpy.ops.mesh.loop_select(extend=False, deselect=False, toggle=False, ring=False)

# Select region of faces inside of a selected loop of edges
bpy.ops.mesh.loop_to_region(select_bigger=False)
# Add a new loop between existing loops
bpy.ops.mesh.loopcut(number_cuts=1, smoothness=0, falloff='ROOT', edge_index=-1,
mesh_select_mode_init=(False, False, False))
# Cut mesh loop and slide it
bpy.ops.mesh.loopcut_slide(MESH_OT_loopcut={"number_cuts":1, "smoothness":0, "falloff":'ROOT',
"edge_index":-1, "mesh_select_mode_init":(False, False, False)}, TRANSFORM_OT_edge_slide={"value":0,
"mirror":False, "snap":False, "snap_target":'CLOSEST', "snap_point":(0, 0, 0), "snap_align":False,
"snap_normal":(0, 0, 0), "correct_uv":False, "release_confirm":False})
# (Un)mark selected edges as Freestyle feature edges
bpy.ops.mesh.mark_freestyle_edge(clear=False)
# (Un)mark selected faces for exclusion from Freestyle feature edge detection
bpy.ops.mesh.mark_freestyle_face(clear=False)
# (Un)mark selected edges as a seam
bpy.ops.mesh.mark_seam(clear=False)
# (Un)mark selected edges as sharp
bpy.ops.mesh.mark_sharp(clear=False)
# Merge selected vertices
bpy.ops.mesh.merge(type='CENTER', uvs=False)
# Remove navmesh data from this mesh
bpy.ops.mesh.navmesh_clear()
# Add a new index and assign it to selected faces
bpy.ops.mesh.navmesh_face_add()
# Copy the index from the active face
bpy.ops.mesh.navmesh_face_copy()
# Create navigation mesh for selected objects
bpy.ops.mesh.navmesh_make()
# Assign a new index to every face
bpy.ops.mesh.navmesh_reset()
# Use vertex coordinate as texture coordinate
bpy.ops.mesh.noise(factor=0.1)
# Make face and vertex normals point either outside or inside the mesh
bpy.ops.mesh.normals_make_consistent(inside=False)
# Split a face into a fan
bpy.ops.mesh.poke(offset=0, use_relative_offset=False, center_mode='MEAN_WEIGHTED')
# Construct a circle mesh
bpy.ops.mesh.primitive_circle_add(vertices=32, radius=1, fill_type='NOTHING', view_align=False,
enter_editmode=False, location=(0, 0, 0), rotation=(0, 0, 0), layers=(False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False, False, False, False, False, False))
Construct a conic mesh
bpy.ops.mesh.primitive_cone_add(vertices=32, radius1=1, radius2=0, depth=2, end_fill_type='NGON',
view_align=False, enter_editmode=False, location=(0, 0, 0), rotation=(0, 0, 0), layers=(False, False, False, False,
False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False))
Construct a cube mesh
bpy.ops.mesh.primitive_cube_add(radius=1, view_align=False, enter_editmode=False, location=(0, 0, 0),
rotation=(0, 0, 0), layers=(False, False, False, False, False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False))

Construct a cylinder mesh

bpy.ops.mesh.primitive_cylinder_add(vertices=32, radius=1, depth=2, end_fill_type='NGON', view_align=False, enter_editmode=False, location=(0, 0, 0), rotation=(0, 0, 0), layers=(False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False))

Construct a grid mesh

bpy.ops.mesh.primitive_grid_add(x_subdivisions=10, y_subdivisions=10, radius=1, view_align=False, enter_editmode=False, location=(0, 0, 0), rotation=(0, 0, 0), layers=(False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False))

Construct an Icosphere mesh

bpy.ops.mesh.primitive_ico_sphere_add(subdivisions=2, size=1, view_align=False, enter_editmode=False, location=(0, 0, 0), rotation=(0, 0, 0), layers=(False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False))

Construct a Suzanne mesh

bpy.ops.mesh.primitive_monkey_add(radius=1, view_align=False, enter_editmode=False, location=(0, 0, 0), rotation=(0, 0, 0), layers=(False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False))

Construct a filled planar mesh with 4 vertices

bpy.ops.mesh.primitive_plane_add(radius=1, view_align=False, enter_editmode=False, location=(0, 0, 0), rotation=(0, 0, 0), layers=(False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False))

Add a torus mesh

bpy.ops.mesh.primitive_torus_add(rotation=(0, 0, 0), view_align=False, location=(0, 0, 0), major_radius=1, minor_radius=0.25, major_segments=48, minor_segments=12, use_abso=False, abso_major_rad=1, abso_minor_rad=0.5)

Construct a UV sphere mesh

bpy.ops.mesh.primitive_uv_sphere_add(segments=32, ring_count=16, size=1, view_align=False, enter_editmode=False, location=(0, 0, 0), rotation=(0, 0, 0), layers=(False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False))

Triangulate selected faces

bpy.ops.mesh.quads_convert_to_tris(use_beauty=True)

Select boundary edges around the selected faces

bpy.ops.mesh.region_to_loop()

Remove duplicate vertices

bpy.ops.mesh.remove_doubles(threshold=0.0001, use_unselected=False)

Reveal all hidden vertices, edges and faces

bpy.ops.mesh.reveal()

Disconnect vertex or edges from connected geometry

bpy.ops.mesh.rip(mirror=False, proportional='DISABLED', proportional_edit_falloff='SMOOTH', proportional_size=1, release_confirm=False, use_fill=False)

Rip polygons and move the result

bpy.ops.mesh.rip_move(MESH_OT_rip={"mirror":False, "proportional":'DISABLED', "proportional_edit_falloff":'SMOOTH', "proportional_size":1, "release_confirm":False, "use_fill":False}, TRANSFORM_OT_translate={"value":(0, 0, 0), "constraint_axis":(False, False, False), "constraint_orientation":'GLOBAL', "mirror":False, "proportional":'DISABLED', "proportional_edit_falloff":'SMOOTH', "proportional_size":1, "snap":False, "snap_target":'CLOSEST', "snap_point":(0, 0, 0), "snap_align":False, "snap_normal":(0, 0, 0), "texture_space":False, "release_confirm":False})

Rip-fill polygons and move the result

bpy.ops.mesh.rip_move_fill(MESH_OT_rip={"mirror":False, "proportional":'DISABLED', "proportional_edit_falloff":'SMOOTH', "proportional_size":1, "release_confirm":False, "use_fill":False}, TRANSFORM_OT_translate={"value":(0, 0, 0), "constraint_axis":(False, False, False),

“constraint_orientation”:’GLOBAL’, “mirror”:False, “proportional”:’DISABLED’,
“proportional_edit_falloff”:’SMOOTH’, “proportional_size”:1, “snap”:False, “snap_target”:’CLOSEST’,
“snap_point”:(0, 0, 0), “snap_align”:False, “snap_normal”:(0, 0, 0), “texture_space”:False, “release_confirm”:False})
# Extrude selected vertices in screw-shaped rotation around the cursor in indicated viewport
bpy.ops.mesh.screw(steps=9, turns=1, center=(0, 0, 0), axis=(0, 0, 0))
# (De)select all vertices, edges or faces
bpy.ops.mesh.select_all(action=’TOGGLE’)
# Select all data in the mesh on a single axis
bpy.ops.mesh.select_axis(mode=’POSITIVE’, axis=’X_AXIS’)
# Select vertices or faces by the number of polygon sides
bpy.ops.mesh.select_face_by_sides(number=4, type=’EQUAL’, extend=True)
# Select faces where all edges have more than 2 face users
bpy.ops.mesh.select_interior_faces()
# Deselect vertices, edges or faces at the boundary of each selection region
bpy.ops.mesh.select_less()
# Select all vertices linked to the active mesh
bpy.ops.mesh.select_linked(limit=False)
# (De)select all vertices linked to the edge under the mouse cursor
bpy.ops.mesh.select_linked_pick(deselect=False, limit=False)
# Select loose geometry based on the selection mode
bpy.ops.mesh.select_loose(extend=False)
# Select mesh items at mirrored locations
bpy.ops.mesh.select_mirror(extend=False)
# Change selection mode
bpy.ops.mesh.select_mode(use_extend=False, use_expand=False, type=’VERT’, action=’TOGGLE’)
# Select more vertices, edges or faces connected to initial selection
bpy.ops.mesh.select_more()
# Select next edge loop adjacent to a selected loop
bpy.ops.mesh.select_next_loop()
# Select all non-manifold vertices or edges
bpy.ops.mesh.select_non_manifold(extend=True)
# Deselect every Nth element starting from the active vertex, edge or face
bpy.ops.mesh.select_nth(nth=2, offset=0)
# Randomly select vertices
bpy.ops.mesh.select_random(percent=50, extend=False)
# Select similar vertices, edges or faces by property types
bpy.ops.mesh.select_similar(type=’NORMAL’, compare=’EQUAL’, threshold=0)
# Select vertices without a group
bpy.ops.mesh.select_ungrouped(extend=False)
# Separate selected geometry into a new mesh
bpy.ops.mesh.separate(type=’SELECTED’)
# Apply selected vertex locations to all other shape keys
bpy.ops.mesh.shape_propagate_to_all()
# Select shortest path between two selections
bpy.ops.mesh.shortest_path_pick(extend=False)
# Selected vertex path between two vertices
bpy.ops.mesh.shortest_path_select(use_length=True)

```
# Create a solid skin by extruding, compensating for sharp angles
bpy.ops.mesh.solidify(thickness=0.01)
# The order of selected vertices/edges/faces is modified, based on a given method
bpy.ops.mesh.sort_elements(type='VIEW_ZAXIS', elements=set(), reverse=False, seed=0)
# Extrude selected vertices in a circle around the cursor in indicated viewport
bpy.ops.mesh.spin(steps=9, dupli=False, angle=1.5708, center=(0, 0, 0), axis=(0, 0, 0))
# Split off selected geometry from connected unselected geometry
bpy.ops.mesh.split()
# Subdivide selected edges
bpy.ops.mesh.subdivide(number_cuts=1, smoothness=0, quadtri=False, quadcorner='STRAIGHT_CUT', fractal=0,
fractal_along_normal=0, seed=0)
#
bpy.ops.mesh.subdivide_edgering(number_cuts=10, interpolation='PATH', smoothness=1, profile_shape_factor=0,
profile_shape='SMOOTH')
# Enforce symmetry (both form and topological) across an axis
bpy.ops.mesh.symmetrize(direction='NEGATIVE_X')
# Snap vertex pairs to their mirrored locations
bpy.ops.mesh.symmetry_snap(direction='NEGATIVE_X', threshold=0.05, factor=0.5, use_center=True)
# Join triangles into quads
bpy.ops.mesh.tris_convert_to_quads(limit=0.698132, uvs=False, vcols=False, sharp=False, materials=False)
# UnSubdivide selected edges & faces
bpy.ops.mesh.unsubdivide(iterations=2)
# Add UV Map
bpy.ops.mesh.uv_texture_add()
# Remove UV Map
bpy.ops.mesh.uv_texture_remove()
# Flip direction of UV coordinates inside faces
bpy.ops.mesh.uvs_reverse()
# Rotate UV coordinates inside faces
bpy.ops.mesh.uvs_rotate(use_ccw=False)
# Connect 2 vertices of a face by an edge, splitting the face in two
bpy.ops.mesh.vert_connect()
# Add vertex color layer
bpy.ops.mesh.vertex_color_add()
# Remove vertex color layer
bpy.ops.mesh.vertex_color_remove()
# Flatten angles of selected vertices
bpy.ops.mesh.vertices_smooth(repeat=1, xaxis=True, yaxis=True, zaxis=True)
# Laplacian smooth of selected vertices
bpy.ops.mesh.vertices_smooth_laplacian(repeat=1, lambda_factor=5e-05, lambda_border=5e-05, use_x=True,
use_y=True, use_z=True, preserve_volume=True)
# Inset new faces into selected faces
bpy.ops.mesh.wireframe(use_boundary=True, use_even_offset=True, use_relative_offset=False, use_crease=False,
thickness=0.01, use_replace=True)


# Synchronize the length of the referenced Action with the length used in the strip
bpy.ops.nla.action_sync_length(active=True)
```

```
# Add an Action-Clip strip (i.e. an NLA Strip referencing an Action) to the active track
bpy.ops.nla.actionclip_add(action='<UNKNOWN ENUM>')
# Apply scaling of selected strips to their referenced Actions
bpy.ops.nla.apply_scale()
# Bake object/pose loc/scale/rotation animation to a new action
bpy.ops.nla.bake(frame_start=1, frame_end=250, step=1, only_selected=True, clear_constraints=False,
clear_parents=False, bake_types={'POSE'})
# Handle clicks to select NLA channels
bpy.ops.nla.channels_click(extend=False)
# Reset scaling of selected strips
bpy.ops.nla.clear_scale()
# Handle clicks to select NLA Strips
bpy.ops.nla.click_select(extend=False)
# Delete selected strips
bpy.ops.nla.delete()
# Duplicate selected NLA-Strips, adding the new strips in new tracks above the originals
bpy.ops.nla.duplicate(mode='TRANSLATION')
# Add a F-Modifier of the specified type to the selected NLA-Strips
bpy.ops.nla.fmodifier_add(type='NULL', only_active=False)
# Copy the F-Modifier(s) of the active NLA-Strip
bpy.ops.nla.fmodifier_copy()
# Add copied F-Modifiers to the selected NLA-Strips
bpy.ops.nla.fmodifier_paste()
# Add new meta-strips incorporating the selected strips
bpy.ops.nla.meta_add()
# Separate out the strips held by the selected meta-strips
bpy.ops.nla.meta_remove()
# Move selected strips down a track if there's room
bpy.ops.nla.move_down()
# Move selected strips up a track if there's room
bpy.ops.nla.move_up()
# Mute or un-mute selected strips
bpy.ops.nla.mute_toggle()
# Toggle display properties panel
bpy.ops.nla.properties()
# Select or deselect all NLA-Strips
bpy.ops.nla.select_all_toggle(invert=False)
# Use box selection to grab NLA-Strips
bpy.ops.nla.select_border(gesture_mode=0, xmin=0, xmax=0, ymin=0, ymax=0, extend=True, axis_range=False)
# Select strips to the left or the right of the current frame
bpy.ops.nla.select_leftright(mode='CHECK', extend=False)
# Make selected objects appear in NLA Editor by adding Animation Data
bpy.ops.nla.selected_objects_add()
# Move start of strips to specified time
bpy.ops.nla.snap(type='CFRA')
# Add a strip for controlling when speaker plays its sound clip
bpy.ops.nla.soundclip_add()
```

```
# Split selected strips at their midpoints
bpy.ops.nla.split()
# Swap order of selected strips within tracks
bpy.ops.nla.swap()
# Add NLA-Tracks above/after the selected tracks
bpy.ops.nla.tracks_add(above_selected=False)
# Delete selected NLA-Tracks and the strips they contain
bpy.ops.nla.tracks_delete()
# Add a transition strip between two adjacent selected strips
bpy.ops.nla.transition_add()
# Enter tweaking mode for the action referenced by the active strip
bpy.ops.nla.tweakmode_enter()
# Exit tweaking mode for the action referenced by the active strip
bpy.ops.nla.tweakmode_exit()
# Reset viewable area to show full strips range
bpy.ops.nla.view_all()
# Reset viewable area to show selected strips range
bpy.ops.nla.view_selected()


# Add a node to the active tree and link to an existing socket
bpy.ops.node.add_and_link_node(use_transform=False, settings=[], type="", link_socket_index=0)
# Add a file node to the current node editor
bpy.ops.node.add_file(filepath="", filter_blender=False, filter_backup=False, filter_image=True, filter_movie=False,
filter_python=False, filter_font=False, filter_sound=False, filter_text=False, filter_btx=False, filter_collada=False,
filter_folder=True, filemode=9, display_type='FILE_DEFAULTDISPLAY', name="Image")
# Add a node to the active tree
bpy.ops.node.add_node(use_transform=False, settings=[], type="")
# Add a reroute node
bpy.ops.node.add_reroute(path=[], cursor=6)
# Add a node to the active tree
bpy.ops.node.add_search(use_transform=False, settings=[], type="", node_item='<UNKNOWN ENUM>')
# Attach active node to a frame
bpy.ops.node.attach()
# Move Node backdrop
bpy.ops.node.backimage_move()
# Use mouse to sample background image
bpy.ops.node.backimage_sample()
# Zoom in/out the background image
bpy.ops.node.backimage_zoom(factor=1.2)
# Copies selected nodes to the clipboard
bpy.ops.node.clipboard_copy()
# Pastes nodes from the clipboard to the active node tree
bpy.ops.node.clipboard_paste()
# Toggle collapsed nodes and hide unused sockets
bpy.ops.node.collapse_hide_unused_toggle()
# Delete selected nodes
bpy.ops.node.delete()
```

# Delete nodes; will reconnect nodes as if deletion was muted
bpy.ops.node.delete_reconnect()
# Detach selected nodes from parents
bpy.ops.node.detach()
# Detach nodes, move and attach to frame
bpy.ops.node.detach_translate_attach(NODE_OT_detach={}, TRANSFORM_OT_translate={"value":(0, 0, 0), "constraint_axis":(False, False, False), "constraint_orientation":'GLOBAL', "mirror":False, "proportional":'DISABLED', "proportional_edit_falloff":'SMOOTH', "proportional_size":1, "snap":False, "snap_target":'CLOSEST', "snap_point":(0, 0, 0), "snap_align":False, "snap_normal":(0, 0, 0), "texture_space":False, "release_confirm":False}, NODE_OT_attach={})
# Duplicate selected nodes
bpy.ops.node.duplicate(keep_inputs=False)
# Duplicate selected nodes and move them
bpy.ops.node.duplicate_move(NODE_OT_duplicate={"keep_inputs":False}, NODE_OT_translate_attach={"TRANSFORM_OT_translate":{"value":(0, 0, 0), "constraint_axis":(False, False, False), "constraint_orientation":'GLOBAL', "mirror":False, "proportional":'DISABLED', "proportional_edit_falloff":'SMOOTH', "proportional_size":1, "snap":False, "snap_target":'CLOSEST', "snap_point":(0, 0, 0), "snap_align":False, "snap_normal":(0, 0, 0), "texture_space":False, "release_confirm":False}, "NODE_OT_attach":{}})
# Duplicate selected nodes keeping input links and move them
bpy.ops.node.duplicate_move_keep_inputs(NODE_OT_duplicate={"keep_inputs":False}, NODE_OT_translate_attach={"TRANSFORM_OT_translate":{"value":(0, 0, 0), "constraint_axis":(False, False, False), "constraint_orientation":'GLOBAL', "mirror":False, "proportional":'DISABLED', "proportional_edit_falloff":'SMOOTH', "proportional_size":1, "snap":False, "snap_target":'CLOSEST', "snap_point":(0, 0, 0), "snap_align":False, "snap_normal":(0, 0, 0), "texture_space":False, "release_confirm":False}, "NODE_OT_attach":{}})
# Search for named node and allow to select and activate it
bpy.ops.node.find_node(prev=False)
# Edit node group
bpy.ops.node.group_edit(exit=False)
# Insert selected nodes into a node group
bpy.ops.node.group_insert()
# Make group from selected nodes
bpy.ops.node.group_make()
# Separate selected nodes from the node group
bpy.ops.node.group_separate(type='COPY')
# Ungroup selected nodes
bpy.ops.node.group_ungroup()
# Toggle unused node socket display
bpy.ops.node.hide_socket_toggle()
# Toggle hiding of selected nodes
bpy.ops.node.hide_toggle()
# Attach selected nodes to a new common frame
bpy.ops.node.join()
# Use the mouse to create a link between two nodes
bpy.ops.node.link(detach=False)
# Makes a link between selected output in input sockets
bpy.ops.node.link_make(replace=False)
# Link to viewer node

bpy.ops.node.link_viewer()
# Use the mouse to cut (remove) some links
bpy.ops.node.links_cut(path=[], cursor=9)
# Remove all links to selected nodes, and try to connect neighbor nodes together
bpy.ops.node.links_detach()
# Move a node to detach links
bpy.ops.node.move_detach_links(NODE_OT_links_detach={}, TRANSFORM_OT_translate={"value":(0, 0, 0), "constraint_axis":(False, False, False), "constraint_orientation":'GLOBAL', "mirror":False, "proportional":'DISABLED', "proportional_edit_falloff":'SMOOTH', "proportional_size":1, "snap":False, "snap_target":'CLOSEST', "snap_point":(0, 0, 0), "snap_align":False, "snap_normal":(0, 0, 0), "texture_space":False, "release_confirm":False})
# Move a node to detach links
bpy.ops.node.move_detach_links_release(NODE_OT_links_detach={}, NODE_OT_translate_attach={"TRANSFORM_OT_translate":{"value":(0, 0, 0), "constraint_axis":(False, False, False), "constraint_orientation":'GLOBAL', "mirror":False, "proportional":'DISABLED', "proportional_edit_falloff":'SMOOTH', "proportional_size":1, "snap":False, "snap_target":'CLOSEST', "snap_point":(0, 0, 0), "snap_align":False, "snap_normal":(0, 0, 0), "texture_space":False, "release_confirm":False}, "NODE_OT_attach":{}})
# Toggle muting of the nodes
bpy.ops.node.mute_toggle()
# Create a new node tree
bpy.ops.node.new_node_tree(type='CompositorNodeTree', name="NodeTree")
# Add a Node Color Preset
bpy.ops.node.node_color_preset_add(remove_active=False, name="")
# Copy color to all selected nodes
bpy.ops.node.node_copy_color()
# Toggle option buttons display for selected nodes
bpy.ops.node.options_toggle()
# Add a new input to a file output node
bpy.ops.node.output_file_add_socket(file_path="Image")
# Move the active input of a file output node up or down the list
bpy.ops.node.output_file_move_active_socket(direction='DOWN')
# Remove active input from a file output node
bpy.ops.node.output_file_remove_active_socket()
# Detach selected nodes
bpy.ops.node.parent_clear()
# Attach selected nodes
bpy.ops.node.parent_set()
# Toggle preview display for selected nodes
bpy.ops.node.preview_toggle()
# Toggles the properties panel display
bpy.ops.node.properties()
# Read all render layers of current scene, in full sample
bpy.ops.node.read_fullsamplelayers()
# Read all render layers of all used scenes
bpy.ops.node.read_renderlayers()
# Render current scene, when input node's layer has been changed
bpy.ops.node.render_changed()

```
# Resize a node
bpy.ops.node.resize()
# Select the node under the cursor
bpy.ops.node.select(mouse_x=0, mouse_y=0, extend=False)
# (De)select all nodes
bpy.ops.node.select_all(action='TOGGLE')
# Use box selection to select nodes
bpy.ops.node.select_border(gesture_mode=0, xmin=0, xmax=0, ymin=0, ymax=0, extend=True, tweak=False)
# Select nodes using lasso selection
bpy.ops.node.select_lasso(path=[], deselect=False, extend=True)
# Select node and link it to a viewer node
bpy.ops.node.select_link_viewer(NODE_OT_select={"mouse_x":0, "mouse_y":0, "extend":False},
NODE_OT_link_viewer={})
# Select nodes linked from the selected ones
bpy.ops.node.select_linked_from()
# Select nodes linked to the selected ones
bpy.ops.node.select_linked_to()
# Select all the nodes of the same type
bpy.ops.node.select_same_type()
# Activate and view same node type, step by step
bpy.ops.node.select_same_type_step(prev=False)
# Update shader script node with new sockets and options from the script
bpy.ops.node.shader_script_update()
# Sort the nodes and show the cyclic dependencies between the nodes
bpy.ops.node.show_cyclic_dependencies()
# Toggles tool shelf display
bpy.ops.node.toolbar()
# Move nodes and attach to frame
bpy.ops.node.translate_attach(TRANSFORM_OT_translate={"value":(0, 0, 0), "constraint_axis":(False, False,
False), "constraint_orientation":'GLOBAL', "mirror":False, "proportional":'DISABLED',
"proportional_edit_falloff":'SMOOTH', "proportional_size":1, "snap":False, "snap_target":'CLOSEST',
"snap_point":(0, 0, 0), "snap_align":False, "snap_normal":(0, 0, 0), "texture_space":False, "release_confirm":False},
NODE_OT_attach={})
# Go to parent node tree
bpy.ops.node.tree_path_parent()
# Add an input or output socket to the current node tree
bpy.ops.node.tree_socket_add(in_out='IN')
# Move a socket up or down in the current node tree's sockets stack
bpy.ops.node.tree_socket_move(direction='UP')
# Remove an input or output socket to the current node tree
bpy.ops.node.tree_socket_remove()
# Resize view so you can see all nodes
bpy.ops.node.view_all()
# Resize view so you can see selected nodes
bpy.ops.node.view_selected()
# Set the boundaries for viewer operations
bpy.ops.node.viewer_border(gesture_mode=0, xmin=0, xmax=0, ymin=0, ymax=0, extend=True)
```

# Add an object to the scene
bpy.ops.object.add(type='EMPTY', view_align=False, enter_editmode=False, location=(0, 0, 0), rotation=(0, 0, 0), layers=(False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False))
# Add named object
bpy.ops.object.add_named(linked=False, name="Cube")
# Align Objects
bpy.ops.object.align(bb_quality=True, align_mode='OPT_2', relative_to='OPT_4', align_axis=set())
# Convert object animation for normal transforms to delta transforms
bpy.ops.object.anim_transforms_to_deltas()
# Add an armature object to the scene
bpy.ops.object.armature_add(view_align=False, enter_editmode=False, location=(0, 0, 0), rotation=(0, 0, 0), layers=(False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False))
# Bake image textures of selected objects
bpy.ops.object.bake_image()
# Add a camera object to the scene
bpy.ops.object.camera_add(view_align=False, enter_editmode=False, location=(0, 0, 0), rotation=(0, 0, 0), layers=(False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False))
# Add a constraint to the active object
bpy.ops.object.constraint_add(type='<UNKNOWN ENUM>')
# Add a constraint to the active object, with target (where applicable) set to the selected Objects/Bones
bpy.ops.object.constraint_add_with_targets(type='<UNKNOWN ENUM>')
# Clear all the constraints for the active Object only
bpy.ops.object.constraints_clear()
# Copy constraints to other selected objects
bpy.ops.object.constraints_copy()
# Convert selected objects to another type
bpy.ops.object.convert(target='MESH', keep_original=False)
# Delete selected objects
bpy.ops.object.delete(use_global=False)
# Add an empty image type to scene with data
bpy.ops.object.drop_named_image(filepath="", name="", view_align=False, location=(0, 0, 0), rotation=(0, 0, 0), layers=(False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False))
#
bpy.ops.object.drop_named_material(name="Material")
# Set offset used for DupliGroup based on cursor position
bpy.ops.object.dupli_offset_from_cursor(group=0)
# Duplicate selected objects
bpy.ops.object.duplicate(linked=False, mode='TRANSLATION')
# Duplicate selected objects and move them
bpy.ops.object.duplicate_move(OBJECT_OT_duplicate={"linked":False, "mode":'TRANSLATION'}, TRANSFORM_OT_translate={"value":(0, 0, 0), "constraint_axis":(False, False, False), "constraint_orientation":'GLOBAL', "mirror":False, "proportional":'DISABLED', "proportional_edit_falloff":'SMOOTH', "proportional_size":1, "snap":False, "snap_target":'CLOSEST', "snap_point":(0, 0, 0), "snap_align":False, "snap_normal":(0, 0, 0), "texture_space":False, "release_confirm":False})

# Duplicate selected objects and move them
bpy.ops.object.duplicate_move_linked(OBJECT_OT_duplicate={"linked":False, "mode":'TRANSLATION'}, TRANSFORM_OT_translate={"value":(0, 0, 0), "constraint_axis":(False, False, False), "constraint_orientation":'GLOBAL', "mirror":False, "proportional":'DISABLED', "proportional_edit_falloff":'SMOOTH', "proportional_size":1, "snap":False, "snap_target":'CLOSEST', "snap_point":(0, 0, 0), "snap_align":False, "snap_normal":(0, 0, 0), "texture_space":False, "release_confirm":False})
# Make dupli objects attached to this object real
bpy.ops.object.duplicates_make_real(use_base_parent=False, use_hierarchy=False)
# Toggle object's editmode
bpy.ops.object.editmode_toggle()
# Add an empty object with a physics effector to the scene
bpy.ops.object.effector_add(type='FORCE', view_align=False, enter_editmode=False, location=(0, 0, 0), rotation=(0, 0, 0), layers=(False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False))
# Add an empty object to the scene
bpy.ops.object.empty_add(type='PLAIN_AXES', view_align=False, location=(0, 0, 0), rotation=(0, 0, 0), layers=(False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False))
# Refresh data in the Explode modifier
bpy.ops.object.explode_refresh(modifier="")
# Toggle object's force field
bpy.ops.object.forcefield_toggle()
# Copy game physics properties to other selected objects
bpy.ops.object.game_physics_copy()
# Remove all game properties from all selected objects
bpy.ops.object.game_property_clear()
# Copy/merge/replace a game property from active object to all selected objects
bpy.ops.object.game_property_copy(operation='COPY', property='<UNKNOWN ENUM>')
# Create a new property available to the game engine
bpy.ops.object.game_property_new(type='FLOAT', name="")
# Remove game property
bpy.ops.object.game_property_remove(index=0)
# Add an object to a new group
bpy.ops.object.group_add()
# Add a dupligroup instance
bpy.ops.object.group_instance_add(name="Group", group='<UNKNOWN ENUM>', view_align=False, location=(0, 0, 0), rotation=(0, 0, 0), layers=(False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False))
# Add an object to an existing group
bpy.ops.object.group_link(group='<UNKNOWN ENUM>')
# Remove the active object from this group
bpy.ops.object.group_remove()
# Reveal the render object by setting the hide render flag
bpy.ops.object.hide_render_clear()
# Reveal all render objects by setting the hide render flag
bpy.ops.object.hide_render_clear_all()
# Hide the render object by setting the hide render flag
bpy.ops.object.hide_render_set(unselected=False)

```
# Reveal the object by setting the hide flag
bpy.ops.object.hide_view_clear()
# Hide the object by setting the hide flag
bpy.ops.object.hide_view_set(unselected=False)
# Hook selected vertices to the first selected Object
bpy.ops.object.hook_add_newob()
# Hook selected vertices to the first selected Object
bpy.ops.object.hook_add_selob(use_bone=False)
# Assign the selected vertices to a hook
bpy.ops.object.hook_assign(modifier='<UNKNOWN ENUM>')
# Set hook center to cursor position
bpy.ops.object.hook_recenter(modifier='<UNKNOWN ENUM>')
# Remove a hook from the active object
bpy.ops.object.hook_remove(modifier='<UNKNOWN ENUM>')
# Recalculate and clear offset transformation
bpy.ops.object.hook_reset(modifier='<UNKNOWN ENUM>')
# Select affected vertices on mesh
bpy.ops.object.hook_select(modifier='<UNKNOWN ENUM>')
# Hide unselected render objects of same type as active by setting the hide render flag
bpy.ops.object.isolate_type_render()
# Join selected objects into active object
bpy.ops.object.join()
# Merge selected objects to shapes of active object
bpy.ops.object.join_shapes()
# Transfer UV Layouts from active to selected objects (needs matching geometry)
bpy.ops.object.join_uvs()
# Add a lamp object to the scene
bpy.ops.object.lamp_add(type='POINT', view_align=False, location=(0, 0, 0), rotation=(0, 0, 0), layers=(False,
False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False,
False))
# Clear the object's location
bpy.ops.object.location_clear()
# Copy logic bricks to other selected objects
bpy.ops.object.logic_bricks_copy()
# Make linked objects into dupli-faces
bpy.ops.object.make_dupli_face()
# Make links from the active object to other selected objects
bpy.ops.object.make_links_data(type='OBDATA')
# Link selection to another scene
bpy.ops.object.make_links_scene(scene='Scene')
# Make library linked datablocks local to this file
bpy.ops.object.make_local(type='SELECT_OBJECT')
# Make linked data local to each object
bpy.ops.object.make_single_user(type='SELECTED_OBJECTS', object=False, obdata=False, material=False,
texture=False, animation=False)
# Add a new material slot
bpy.ops.object.material_slot_add()
```

# Assign active material slot to selection
bpy.ops.object.material_slot_assign()
# Copies materials to other selected objects
bpy.ops.object.material_slot_copy()
# Deselect by active material slot
bpy.ops.object.material_slot_deselect()
# Remove the selected material slot
bpy.ops.object.material_slot_remove()
# Select by active material slot
bpy.ops.object.material_slot_select()
# Bind mesh to cage in mesh deform modifier
bpy.ops.object.meshdeform_bind(modifier="")
# Add an metaball object to the scene
bpy.ops.object.metaball_add(type='BALL', view_align=False, enter_editmode=False, location=(0, 0, 0), rotation=(0, 0, 0), layers=(False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False))
# Sets the object interaction mode
bpy.ops.object.mode_set(mode='OBJECT', toggle=False)
# Add a modifier to the active object
bpy.ops.object.modifier_add(type='SUBSURF')
# Apply modifier and remove from the stack
bpy.ops.object.modifier_apply(apply_as='DATA', modifier="")
# Convert particles to a mesh object
bpy.ops.object.modifier_convert(modifier="")
# Duplicate modifier at the same position in the stack
bpy.ops.object.modifier_copy(modifier="")
# Move modifier down in the stack
bpy.ops.object.modifier_move_down(modifier="")
# Move modifier up in the stack
bpy.ops.object.modifier_move_up(modifier="")
# Remove a modifier from the active object
bpy.ops.object.modifier_remove(modifier="")
# Move the object to different layers
bpy.ops.object.move_to_layer(layers=(False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False))
# Modify the base mesh to conform to the displaced mesh
bpy.ops.object.multires_base_apply(modifier="")
# Pack displacements from an external file
bpy.ops.object.multires_external_pack()
# Save displacements to an external file
bpy.ops.object.multires_external_save(filepath="", check_existing=True, filter_blender=False, filter_backup=False, filter_image=False, filter_movie=False, filter_python=False, filter_font=False, filter_sound=False, filter_text=False, filter_btx=True, filter_collada=False, filter_folder=True, filemode=9, relative_path=True, display_type='FILE_DEFAULTDISPLAY', modifier="")
# Deletes the higher resolution mesh, potential loss of detail
bpy.ops.object.multires_higher_levels_delete(modifier="")
# Copy vertex coordinates from other object
bpy.ops.object.multires_reshape(modifier="")

```
# Add a new level of subdivision
bpy.ops.object.multires_subdivide(modifier="")
# Bake an image sequence of ocean data
bpy.ops.object.ocean_bake(modifier="", free=False)
# Clear the object's origin
bpy.ops.object.origin_clear()
# Set the object's origin, by either moving the data, or set to center of data, or use 3D cursor
bpy.ops.object.origin_set(type='GEOMETRY_ORIGIN', center='MEDIAN')
# Clear the object's parenting
bpy.ops.object.parent_clear(type='CLEAR')
# Set the object's parenting without setting the inverse parent correction
bpy.ops.object.parent_no_inverse_set()
# Set the object's parenting
bpy.ops.object.parent_set(type='OBJECT', xmirror=False, keep_transform=False)
# Add a particle system
bpy.ops.object.particle_system_add()
# Remove the selected particle system
bpy.ops.object.particle_system_remove()
# Calculate motion paths for the selected objects
bpy.ops.object.paths_calculate(start_frame=1, end_frame=250)
# Clear path caches for selected objects
bpy.ops.object.paths_clear()
# Recalculate paths for selected objects
bpy.ops.object.paths_update()
# Enable or disable posing/selecting bones
bpy.ops.object.posemode_toggle()
# Add empty object to become local replacement data of a library-linked object
bpy.ops.object.proxy_make(object='DEFAULT')
#
bpy.ops.object.quick_explode(style='EXPLODE', amount=100, frame_duration=50, frame_start=1, frame_end=10,
velocity=1, fade=True)
#
bpy.ops.object.quick_fluid(style='BASIC', initial_velocity=(0, 0, 0), show_flows=False, start_baking=False)
#
bpy.ops.object.quick_fur(density='MEDIUM', view_percentage=10, length=0.1)
#
bpy.ops.object.quick_smoke(style='STREAM', show_flows=False)
# Randomize objects loc/rot/scale
bpy.ops.object.randomize_transform(random_seed=0, use_delta=False, use_loc=True, loc=(0, 0, 0), use_rot=True,
rot=(0, 0, 0), use_scale=True, scale_even=False, scale=(1, 1, 1))
# Clear the object's rotation
bpy.ops.object.rotation_clear()
# Clear the object's scale
bpy.ops.object.scale_clear()
# Change selection of all visible objects in scene
bpy.ops.object.select_all(action='TOGGLE')
# Select all visible objects on a layer
```

```
bpy.ops.object.select_by_layer(match='EXACT', extend=False, layers=1)
# Select all visible objects that are of a type
bpy.ops.object.select_by_type(extend=False, type='MESH')
# Select the active camera
bpy.ops.object.select_camera(extend=False)
# Select all visible objects grouped by various properties
bpy.ops.object.select_grouped(extend=False, type='CHILDREN_RECURSIVE')
# Select object relative to the active object's position in the hierarchy
bpy.ops.object.select_hierarchy(direction='PARENT', extend=False)
# Select all visible objects that are linked
bpy.ops.object.select_linked(extend=False, type='OBDATA')
# Select the Mirror objects of the selected object eg. L.sword -> R.sword
bpy.ops.object.select_mirror(extend=False)
# Select objects matching a naming pattern
bpy.ops.object.select_pattern(pattern="*", case_sensitive=False, extend=True)
# Set select on random visible objects
bpy.ops.object.select_random(percent=50, extend=False)
# Select object in the same group
bpy.ops.object.select_same_group(group="")
# Render and display faces uniform, using Face Normals
bpy.ops.object.shade_flat()
# Render and display faces smooth, using interpolated Vertex Normals
bpy.ops.object.shade_smooth()
# Add shape key to the object
bpy.ops.object.shape_key_add(from_mix=True)
# Clear weights for all shape keys
bpy.ops.object.shape_key_clear()
# Mirror the current shape key along the local X axis
bpy.ops.object.shape_key_mirror(use_topology=False)
# Move the active shape key up/down in the list
bpy.ops.object.shape_key_move(type='UP')
# Remove shape key from the object
bpy.ops.object.shape_key_remove(all=False)
# Resets the timing for absolute shape keys
bpy.ops.object.shape_key_retime()
# Copy another selected objects active shape to this one by applying the relative offsets
bpy.ops.object.shape_key_transfer(mode='OFFSET', use_clamp=False)
# Create an armature that parallels the skin layout
bpy.ops.object.skin_armature_create(modifier="")
# Mark/clear selected vertices as loose
bpy.ops.object.skin_loose_mark_clear(action='MARK')
# Make skin radii of selected vertices equal on each axis
bpy.ops.object.skin_radii_equalize()
# Mark selected vertices as roots
bpy.ops.object.skin_root_mark()
# Clear the object's slow parent
bpy.ops.object.slow_parent_clear()
```

# Set the object's slow parent
bpy.ops.object.slow_parent_set()
# Add a speaker object to the scene
bpy.ops.object.speaker_add(view_align=False, enter_editmode=False, location=(0, 0, 0), rotation=(0, 0, 0), layers=(False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False))
# Sets a Subdivision Surface Level (1-5)
bpy.ops.object.subdivision_set(level=1, relative=False)
# Add a text object to the scene
bpy.ops.object.text_add(view_align=False, enter_editmode=False, location=(0, 0, 0), rotation=(0, 0, 0), layers=(False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False))
# Clear tracking constraint or flag from object
bpy.ops.object.track_clear(type='CLEAR')
# Make the object track another object, either by constraint or old way or locked track
bpy.ops.object.track_set(type='DAMPTRACK')
# Apply the object's transformation to its data
bpy.ops.object.transform_apply(location=False, rotation=False, scale=False)
# Add a new vertex group to the active object
bpy.ops.object.vertex_group_add()
# Assign the selected vertices to the active vertex group
bpy.ops.object.vertex_group_assign()
# Assign the selected vertices to a new vertex group
bpy.ops.object.vertex_group_assign_new()
# Blend selected vertex weights with unselected for the active group
bpy.ops.object.vertex_group_blend(factor=1)
# Remove Vertex Group assignments which aren't required
bpy.ops.object.vertex_group_clean(group_select_mode='ACTIVE', limit=0, keep_single=False)
# Make a copy of the active vertex group
bpy.ops.object.vertex_group_copy()
# Copy Vertex Groups to all users of the same Geometry data
bpy.ops.object.vertex_group_copy_to_linked()
# Copy Vertex Groups to other selected objects with matching indices
bpy.ops.object.vertex_group_copy_to_selected()
# Deselect all selected vertices assigned to the active vertex group
bpy.ops.object.vertex_group_deselect()
# Modify the position of selected vertices by changing only their respective groups' weights (this tool may be slow for many vertices)
bpy.ops.object.vertex_group_fix(dist=0, strength=1, accuracy=1)
# Invert active vertex group's weights
bpy.ops.object.vertex_group_invert(group_select_mode='ACTIVE', auto_assign=True, auto_remove=True)
# Add some offset and multiply with some gain the weights of the active vertex group
bpy.ops.object.vertex_group_levels(group_select_mode='ACTIVE', offset=0, gain=1)
# Limit deform weights associated with a vertex to a specified number by removing lowest weights
bpy.ops.object.vertex_group_limit_total(group_select_mode='ALL', limit=4)
# Change the lock state of all vertex groups of active object
bpy.ops.object.vertex_group_lock(action='TOGGLE')
# Mirror all vertex groups, flip weights and/or names, editing only selected vertices, flipping when both sides are

selected otherwise copy from unselected
bpy.ops.object.vertex_group_mirror(mirror_weights=True, flip_group_names=True, all_groups=False, use_topology=False)
# Move the active vertex group up/down in the list
bpy.ops.object.vertex_group_move(direction='UP')
# Normalize weights of the active vertex group, so that the highest ones are now 1.0
bpy.ops.object.vertex_group_normalize()
# Normalize all weights of all vertex groups, so that for each vertex, the sum of all weights is 1.0
bpy.ops.object.vertex_group_normalize_all(lock_active=True)
# Delete the active vertex group
bpy.ops.object.vertex_group_remove(all=False)
# Remove the selected vertices from active or all vertex group(s)
bpy.ops.object.vertex_group_remove_from(use_all_groups=False, use_all_verts=False)
# Select all the vertices assigned to the active vertex group
bpy.ops.object.vertex_group_select()
# Set the active vertex group
bpy.ops.object.vertex_group_set_active(group='<UNKNOWN ENUM>')
# Sorts vertex groups alphabetically
bpy.ops.object.vertex_group_sort()
# Transfer weight paint to active from selected mesh

bpy.ops.object.vertex_group_transfer_weight(WT_vertex_group_mode='WT_REPLACE_ACTIVE_VERTEX_GROUP', WT_method='WT_BY_NEAREST_FACE', WT_replace_mode='WT_REPLACE_ALL_WEIGHTS')
# Parent selected objects to the selected vertices
bpy.ops.object.vertex_parent_set()
# Copy weights from Active to selected
bpy.ops.object.vertex_weight_copy()
# Delete this weight from the vertex (disabled if vertex Group is locked)
bpy.ops.object.vertex_weight_delete(weight_group=-1)
# Normalize Active Vert Weights
bpy.ops.object.vertex_weight_normalize_active_vertex()
# Copy this group's weight to other selected verts (disabled if vertex Group is locked)
bpy.ops.object.vertex_weight_paste(weight_group=-1)
# Set as active Vertex Group
bpy.ops.object.vertex_weight_set_active(weight_group=-1)
# Apply the object's visual transformation to its data
bpy.ops.object.visual_transform_apply()

# Change the active action used
bpy.ops.outliner.action_set(action='<UNKNOWN ENUM>')
#
bpy.ops.outliner.animdata_operation(type='SET_ACT')
#
bpy.ops.outliner.data_operation(type='SELECT')
# Add drivers to selected items
bpy.ops.outliner.drivers_add_selected()
# Delete drivers assigned to selected items

```
bpy.ops.outliner.drivers_delete_selected()
# Expand/Collapse all items
bpy.ops.outliner.expanded_toggle()
#
bpy.ops.outliner.group_operation(type='UNLINK')
#
bpy.ops.outliner.id_operation(type='UNLINK')
# Handle mouse clicks to activate/select items
bpy.ops.outliner.item_activate(extend=True, recursive=False)
# Toggle whether item under cursor is enabled or closed
bpy.ops.outliner.item_openclose(all=True)
# Rename item under cursor
bpy.ops.outliner.item_rename()
# Add selected items (blue-gray rows) to active Keying Set
bpy.ops.outliner.keyingset_add_selected()
# Remove selected items (blue-gray rows) from active Keying Set
bpy.ops.outliner.keyingset_remove_selected()
# Drag material to object in Outliner
bpy.ops.outliner.material_drop(object="Object", material="Material")
#
bpy.ops.outliner.object_operation(type='SELECT')
# Context menu for item operations
bpy.ops.outliner.operation()
# Drag to clear parent in Outliner
bpy.ops.outliner.parent_clear(dragged_obj="Object", type='CLEAR')
# Drag to parent in Outliner
bpy.ops.outliner.parent_drop(child="Object", parent="Object", type='OBJECT')
# Toggle the renderability of selected items
bpy.ops.outliner.renderability_toggle()
# Drag object to scene in Outliner
bpy.ops.outliner.scene_drop(object="Object", scene="Scene")
# Scroll page up or down
bpy.ops.outliner.scroll_page(up=False)
# Use box selection to select tree elements
bpy.ops.outliner.select_border(gesture_mode=0, xmin=0, xmax=0, ymin=0, ymax=0)
# Toggle the selectability
bpy.ops.outliner.selectability_toggle()
# Toggle the Outliner selection of items
bpy.ops.outliner.selected_toggle()
# Adjust the view so that the active Object is shown centered
bpy.ops.outliner.show_active()
# Open all object entries and close all others
bpy.ops.outliner.show_hierarchy()
# Expand/collapse all entries by one level
bpy.ops.outliner.show_one_level(open=True)
# Toggle the visibility of selected items
bpy.ops.outliner.visibility_toggle()
```

```
# Select a paint mode's brush by tool type
bpy.ops.paint.brush_select(paint_mode='ACTIVE', sculpt_tool='BLOB', vertex_paint_tool='MIX',
weight_paint_tool='MIX', texture_paint_tool='DRAW', toggle=False, create_missing=False)
# Change selection for all faces
bpy.ops.paint.face_select_all(action='TOGGLE')
# Hide selected faces
bpy.ops.paint.face_select_hide(unselected=False)
# Select linked faces
bpy.ops.paint.face_select_linked()
# Select linked faces
bpy.ops.paint.face_select_linked_pick(extend=False)
# Reveal hidden faces
bpy.ops.paint.face_select_reveal(unselected=False)
# Move the clone source image
bpy.ops.paint.grab_clone(delta=(0, 0))
# Hide/show some vertices
bpy.ops.paint.hide_show(action='HIDE', area='INSIDE', xmin=0, xmax=0, ymin=0, ymax=0)
# Make an image from the current 3D view for re-projection
bpy.ops.paint.image_from_view(filepath="")
# Paint a stroke into the image
bpy.ops.paint.image_paint(mode='NORMAL', stroke=[])
# Fill the whole mask with a given value, or invert its values
bpy.ops.paint.mask_flood_fill(mode='VALUE', value=0)
# Project an edited render from the active camera back onto the object
bpy.ops.paint.project_image(image='Render Result')
# Use the mouse to sample a color in the image
bpy.ops.paint.sample_color(location=(0, 0))
# Toggle texture paint mode in 3D view
bpy.ops.paint.texture_paint_toggle()
# Change selection for all vertices
bpy.ops.paint.vert_select_all(action='TOGGLE')
# Select vertices without a group
bpy.ops.paint.vert_select_ungrouped(extend=False)
#
bpy.ops.paint.vertex_color_dirt(blur_strength=1, blur_iterations=1, clean_angle=3.14159, dirt_angle=0,
dirt_only=False)
# Fill the active vertex color layer with the current paint color
bpy.ops.paint.vertex_color_set()
# Smooth colors across vertices
bpy.ops.paint.vertex_color_smooth()
# Paint a stroke in the active vertex color layer
bpy.ops.paint.vertex_paint(stroke=[])
# Toggle the vertex paint mode in 3D view
bpy.ops.paint.vertex_paint_toggle()
# Set the weights of the groups matching the attached armature's selected bones, using the distance between the
vertices and the bones
bpy.ops.paint.weight_from_bones(type='AUTOMATIC')
```

```
# Sample a line and show it in Scope panels
bpy.ops.paint.weight_gradient(type='LINEAR', xstart=0, xend=0, ystart=0, yend=0, cursor=1)
# Paint a stroke in the current vertex group's weights
bpy.ops.paint.weight_paint(stroke=[])
# Toggle weight paint mode in 3D view
bpy.ops.paint.weight_paint_toggle()
# Use the mouse to sample a weight in the 3D view
bpy.ops.paint.weight_sample()
# Select one of the vertex groups available under current mouse position
bpy.ops.paint.weight_sample_group(group='<UNKNOWN ENUM>')
# Fill the active vertex group with the current paint weight
bpy.ops.paint.weight_set()


# Apply a stroke of brush to the particles
bpy.ops.particle.brush_edit(stroke=[])
# Connect hair to the emitter mesh
bpy.ops.particle.connect_hair(all=False)
# Delete selected particles or keys
bpy.ops.particle.delete(type='PARTICLE')
# Disconnect hair from the emitter mesh
bpy.ops.particle.disconnect_hair(all=False)
# Duplicate the current dupliobject
bpy.ops.particle.dupliob_copy()
# Move dupli object down in the list
bpy.ops.particle.dupliob_move_down()
# Move dupli object up in the list
bpy.ops.particle.dupliob_move_up()
# Remove the selected dupliobject
bpy.ops.particle.dupliob_remove()
# Undo all edition performed on the particle system
bpy.ops.particle.edited_clear()
# Hide selected particles
bpy.ops.particle.hide(unselected=False)
# Duplicate and mirror the selected particles along the local X axis
bpy.ops.particle.mirror()
# Add new particle settings
bpy.ops.particle.new()
# Add a new particle target
bpy.ops.particle.new_target()
# Toggle particle edit mode
bpy.ops.particle.particle_edit_toggle()
# Change the number of keys of selected particles (root and tip keys included)
bpy.ops.particle.rekey(keys_number=2)
# Remove selected particles close enough of others
bpy.ops.particle.remove_doubles(threshold=0.0002)
# Show hidden particles
```

```
bpy.ops.particle.reveal()
# (De)select all particles' keys
bpy.ops.particle.select_all(action='TOGGLE')
# Deselect boundary selected keys of each particle
bpy.ops.particle.select_less()
# Select nearest particle from mouse pointer
bpy.ops.particle.select_linked(deselect=False, location=(0, 0))
# Select keys linked to boundary selected keys of each particle
bpy.ops.particle.select_more()
# Select roots of all visible particles
bpy.ops.particle.select_roots(action='SELECT')
# Select tips of all visible particles
bpy.ops.particle.select_tips(action='SELECT')
# Subdivide selected particles segments (adds keys)
bpy.ops.particle.subdivide()
# Move particle target down in the list
bpy.ops.particle.target_move_down()
# Move particle target up in the list
bpy.ops.particle.target_move_up()
# Remove the selected particle target
bpy.ops.particle.target_remove()
# Set the weight of selected keys
bpy.ops.particle.weight_set(factor=1)


# Apply the current pose as the new rest pose
bpy.ops.pose.armature_apply()
# Change the visible armature layers
bpy.ops.pose.armature_layers(layers=(False, False, False, False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False,
False, False))
# Automatically renames the selected bones according to which side of the target axis they fall on
bpy.ops.pose.autoside_names(axis='XAXIS')
# Change the layers that the selected bones belong to
bpy.ops.pose.bone_layers(layers=(False, False, False, False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False,
False, False))
# Create a suitable breakdown pose on the current frame
bpy.ops.pose.breakdown(prev_frame=0, next_frame=0, percentage=0.5)
# Add a constraint to the active bone
bpy.ops.pose.constraint_add(type='<UNKNOWN ENUM>')
# Add a constraint to the active bone, with target (where applicable) set to the selected Objects/Bones
bpy.ops.pose.constraint_add_with_targets(type='<UNKNOWN ENUM>')
# Clear all the constraints for the selected bones
bpy.ops.pose.constraints_clear()
# Copy constraints to other selected bones
bpy.ops.pose.constraints_copy()
# Copies the current pose of the selected bones to copy/paste buffer
```

```
bpy.ops.pose.copy()
# Flips (and corrects) the axis suffixes of the the names of selected bones
bpy.ops.pose.flip_names()
# Add a new bone group
bpy.ops.pose.group_add()
# Add selected bones to the chosen bone group
bpy.ops.pose.group_assign(type=0)
# Deselect bones of active Bone Group
bpy.ops.pose.group_deselect()
# Change position of active Bone Group in list of Bone Groups
bpy.ops.pose.group_move(direction='UP')
# Remove the active bone group
bpy.ops.pose.group_remove()
# Select bones in active Bone Group
bpy.ops.pose.group_select()
# Sort Bone Groups by their names in ascending order
bpy.ops.pose.group_sort()
# Remove selected bones from all bone groups
bpy.ops.pose.group_unassign()
# Tag selected bones to not be visible in Pose Mode
bpy.ops.pose.hide(unselected=False)
# Add IK Constraint to the active Bone
bpy.ops.pose.ik_add(with_targets=True)
# Remove all IK Constraints from selected bones
bpy.ops.pose.ik_clear()
# Reset locations of selected bones to their default values
bpy.ops.pose.loc_clear()
# Paste the stored pose on to the current pose
bpy.ops.pose.paste(flipped=False, selected_mask=False)
# Calculate paths for the selected bones
bpy.ops.pose.paths_calculate(start_frame=1, end_frame=250, bake_location='TAILS')
# Clear path caches for selected bones
bpy.ops.pose.paths_clear()
# Recalculate paths for bones that already have them
bpy.ops.pose.paths_update()
# Copy selected aspects of the current pose to subsequent poses already keyframed
bpy.ops.pose.propagate(mode='WHILE_HELD', end_frame=250)
# Exaggerate the current pose
bpy.ops.pose.push(prev_frame=0, next_frame=0, percentage=0.5)
# Flip quaternion values to achieve desired rotations, while maintaining the same orientations
bpy.ops.pose.quaternions_flip()
# Make the current pose more similar to its surrounding ones
bpy.ops.pose.relax(prev_frame=0, next_frame=0, percentage=0.5)
# Unhide all bones that have been tagged to be hidden in Pose Mode
bpy.ops.pose.reveal()
# Reset rotations of selected bones to their default values
bpy.ops.pose.rot_clear()
```

```
# Set the rotation representation used by selected bones
bpy.ops.pose.rotation_mode_set(type='QUATERNION')
# Reset scaling of selected bones to their default values
bpy.ops.pose.scale_clear()
# Toggle selection status of all bones
bpy.ops.pose.select_all(action='TOGGLE')
# Select bones used as targets for the currently selected bones
bpy.ops.pose.select_constraint_target()
# Activate the bone with a flipped name
bpy.ops.pose.select_flip_active()
# Select all visible bones grouped by similar properties
bpy.ops.pose.select_grouped(extend=False, type='LAYER')
# Select immediate parent/children of selected bones
bpy.ops.pose.select_hierarchy(direction='PARENT', extend=False)
# Select bones related to selected ones by parent/child relationships
bpy.ops.pose.select_linked(extend=False)
# Select bones that are parents of the currently selected bones
bpy.ops.pose.select_parent()
# Reset location, rotation, and scaling of selected bones to their default values
bpy.ops.pose.transforms_clear()
# Reset pose on selected bones to keyframed state
bpy.ops.pose.user_transforms_clear(only_selected=True)
# Apply final constrained position of pose bones to their transform
bpy.ops.pose.visual_transform_apply()


# Make action suitable for use as a Pose Library
bpy.ops.poselib.action_sanitize()
# Apply specified Pose Library pose to the rig
bpy.ops.poselib.apply_pose(pose_index=-1)
# Interactively browse poses in 3D-View
bpy.ops.poselib.browse_interactive(pose_index=-1)
# Add New Pose Library to active Object
bpy.ops.poselib.new()
# Add the current Pose to the active Pose Library
bpy.ops.poselib.pose_add(frame=1, name="Pose")
# Remove nth pose from the active Pose Library
bpy.ops.poselib.pose_remove(pose='<UNKNOWN ENUM>')
# Rename specified pose from the active Pose Library
bpy.ops.poselib.pose_rename(name="RenamedPose", pose='<UNKNOWN ENUM>')
# Remove Pose Library from active Object
bpy.ops.poselib.unlink()


# Add new cache
bpy.ops.ptcache.add()
# Bake physics
bpy.ops.ptcache.bake(bake=False)
```

```
# Bake all physics
bpy.ops.ptcache.bake_all(bake=True)
# Bake from cache
bpy.ops.ptcache.bake_from_cache()
# Free physics bake
bpy.ops.ptcache.free_bake()
# Free all baked caches of all objects in the current scene
bpy.ops.ptcache.free_bake_all()
# Delete current cache
bpy.ops.ptcache.remove()


# Add an Integrator Preset
bpy.ops.render.cycles_integrator_preset_add(remove_active=False, name="")
# OpenGL render active viewport
bpy.ops.render.opengl(animation=False, sequencer=False, write_still=False, view_context=True)
# Play back rendered frames/movies using an external player
bpy.ops.render.play_rendered_anim()
# Add a Render Preset
bpy.ops.render.preset_add(remove_active=False, name="")
# Render active scene
bpy.ops.render.render(animation=False, write_still=False, layer="", scene="")
# Cancel show render view
bpy.ops.render.view_cancel()
# Toggle show render view
bpy.ops.render.view_show()


# Bake rigid body transformations of selected objects to keyframes
bpy.ops.rigidbody.bake_to_keyframes(frame_start=1, frame_end=250, step=1)
# Create rigid body constraints between selected rigid bodies
bpy.ops.rigidbody.connect(con_type='FIXED', pivot_type='CENTER',
connection_pattern='SELECTED_TO_ACTIVE')
# Add Rigid Body Constraint to active object
bpy.ops.rigidbody.constraint_add(type='FIXED')
# Remove Rigid Body Constraint from Object
bpy.ops.rigidbody.constraint_remove()
# Automatically calculate mass values for Rigid Body Objects based on volume
bpy.ops.rigidbody.mass_calculate(material='Air', density=1)
# Add active object as Rigid Body
bpy.ops.rigidbody.object_add(type='ACTIVE')
# Remove Rigid Body settings from Object
bpy.ops.rigidbody.object_remove()
# Copy Rigid Body settings from active object to selected
bpy.ops.rigidbody.object_settings_copy()
# Add selected objects as Rigid Bodies
bpy.ops.rigidbody.objects_add(type='ACTIVE')
# Remove selected objects from Rigid Body simulation
```

```
bpy.ops.rigidbody.objects_remove()
# Change collision shapes for selected Rigid Body Objects
bpy.ops.rigidbody.shape_change(type='MESH')
# Add Rigid Body simulation world to the current scene
bpy.ops.rigidbody.world_add()
# Remove Rigid Body simulation world from the current scene
bpy.ops.rigidbody.world_remove()


# Delete active scene
bpy.ops.scene.delete()
# Add the data paths to the Freestyle Edge Mark property of selected edges to the active keying set
bpy.ops.scene.freestyle_add_edge_marks_to_keying_set()
# Add the data paths to the Freestyle Face Mark property of selected polygons to the active keying set
bpy.ops.scene.freestyle_add_face_marks_to_keying_set()
# Add an alpha transparency modifier to the line style associated with the active lineset
bpy.ops.scene.freestyle_alpha_modifier_add(type='ALONG_STROKE')
# Add a line color modifier to the line style associated with the active lineset
bpy.ops.scene.freestyle_color_modifier_add(type='ALONG_STROKE')
# Fill the Range Min/Max entries by the min/max distance between selected mesh objects and the source object
bpy.ops.scene.freestyle_fill_range_by_selection(type='COLOR', name="")
# Add a stroke geometry modifier to the line style associated with the active lineset
bpy.ops.scene.freestyle_geometry_modifier_add(type='2D_OFFSET')
# Add a line set into the list of line sets
bpy.ops.scene.freestyle_lineset_add()
# Copy the active line set to a buffer
bpy.ops.scene.freestyle_lineset_copy()
# Change the position of the active line set within the list of line sets
bpy.ops.scene.freestyle_lineset_move(direction='UP')
# Paste the buffer content to the active line set
bpy.ops.scene.freestyle_lineset_paste()
# Remove the active line set from the list of line sets
bpy.ops.scene.freestyle_lineset_remove()
# Create a new line style, reusable by multiple line sets
bpy.ops.scene.freestyle_linestyle_new()
# Duplicate the modifier within the list of modifiers
bpy.ops.scene.freestyle_modifier_copy()
# Move the modifier within the list of modifiers
bpy.ops.scene.freestyle_modifier_move(direction='UP')
# Remove the modifier from the list of modifiers
bpy.ops.scene.freestyle_modifier_remove()
# Add a style module into the list of modules
bpy.ops.scene.freestyle_module_add()
# Change the position of the style module within in the list of style modules
bpy.ops.scene.freestyle_module_move(direction='UP')
# Open a style module file
bpy.ops.scene.freestyle_module_open(filepath="", make_internal=True)
```

```
# Remove the style module from the stack
bpy.ops.scene.freestyle_module_remove()
# Add a line thickness modifier to the line style associated with the active lineset
bpy.ops.scene.freestyle_thickness_modifier_add(type='ALONG_STROKE')
# Add new scene by type
bpy.ops.scene.new(type='NEW')
# Add a render layer
bpy.ops.scene.render_layer_add()
# Remove the selected render layer
bpy.ops.scene.render_layer_remove()


# Handle area action zones for mouse actions/gestures
bpy.ops.screen.actionzone(modifier=0)
# Cancel animation, returning to the original frame
bpy.ops.screen.animation_cancel(restore_frame=True)
# Play animation
bpy.ops.screen.animation_play(reverse=False, sync=False)
# Step through animation by position
bpy.ops.screen.animation_step()
# Duplicate selected area into new window
bpy.ops.screen.area_dupli()
# Join selected areas into new window
bpy.ops.screen.area_join(min_x=-100, min_y=-100, max_x=-100, max_y=-100)
# Move selected area edges
bpy.ops.screen.area_move(x=0, y=0, delta=0)
# Operations for splitting and merging
bpy.ops.screen.area_options()
# Split selected area into new windows
bpy.ops.screen.area_split(direction='HORIZONTAL', factor=0.5, mouse_x=-100, mouse_y=-100)
# Swap selected areas screen positions
bpy.ops.screen.area_swap()
# Revert back to the original screen layout, before fullscreen area overlay
bpy.ops.screen.back_to_previous()
# Delete active screen
bpy.ops.screen.delete()
# Jump to first/last frame in frame range
bpy.ops.screen.frame_jump(end=False)
# Move current frame forward/backward by a given number
bpy.ops.screen.frame_offset(delta=0)
# Toggle the header over/below the main window area
bpy.ops.screen.header_flip()
# Show or hide the header pulldown menus
bpy.ops.screen.header_toggle_menus()
# Display header region toolbox
bpy.ops.screen.header_toolbox()
# Jump to previous/next keyframe
```

```
bpy.ops.screen.keyframe_jump(next=True)
# Add a new screen
bpy.ops.screen.new()
# Display menu for last action performed
bpy.ops.screen.redo_last()
# Blend in and out overlapping region
bpy.ops.screen.region_blend()
# Toggle the region's alignment (left/right or top/bottom)
bpy.ops.screen.region_flip()
# Split selected area into camera, front, right & top views
bpy.ops.screen.region_quadview()
# Scale selected area
bpy.ops.screen.region_scale()
# Display menu for previous actions performed
bpy.ops.screen.repeat_history(index=0)
# Repeat last action
bpy.ops.screen.repeat_last()
# Toggle display selected area as fullscreen
bpy.ops.screen.screen_full_area()
# Cycle through available screens
bpy.ops.screen.screen_set(delta=0)
# Capture a video of the active area or whole Blender window
bpy.ops.screen.screencast(filepath="", full=True)
# Capture a picture of the active area or whole Blender window
bpy.ops.screen.screenshot(filepath="", check_existing=True, filter_blender=False, filter_backup=False,
filter_image=True, filter_movie=False, filter_python=False, filter_font=False, filter_sound=False, filter_text=False,
filter_btx=False, filter_collada=False, filter_folder=True, filemode=9, display_type='FILE_DEFAULTDISPLAY',
full=True)
# Remove unused settings for invisible editors
bpy.ops.screen.spacedata_cleanup()
# Show user preferences
bpy.ops.screen.userpref_show()


# Ignore autoexec warning
bpy.ops.script.autoexec_warn_clear()
# Execute a preset
bpy.ops.script.execute_preset(filepath="", menu_idname="")
# Run Python file
bpy.ops.script.python_file_run(filepath="")
# Reload Scripts
bpy.ops.script.reload()


# Sculpt a stroke into the geometry
bpy.ops.sculpt.brush_stroke(stroke=[], mode='NORMAL', ignore_background_click=False)
# Dynamic topology alters the mesh topology while sculpting
bpy.ops.sculpt.dynamic_topology_toggle()
```

```
# Recalculate the sculpt BVH to improve performance
bpy.ops.sculpt.optimize()
# Toggle sculpt mode in 3D view
bpy.ops.sculpt.sculptmode_toggle()
# Reset the copy of the mesh that is being sculpted on
bpy.ops.sculpt.set_persistent_base()
# Symmetrize the topology modifications
bpy.ops.sculpt.symmetrize()
# Sculpt UVs using a brush
bpy.ops.sculpt.uv_sculpt_stroke(mode='NORMAL')


#
bpy.ops.sequencer.change_effect_input(swap='A_B')
#
bpy.ops.sequencer.change_effect_type(type='CROSS')
#
bpy.ops.sequencer.change_path(filepath="", directory="", files=[], filter_blender=False, filter_backup=False,
filter_image=True, filter_movie=True, filter_python=False, filter_font=False, filter_sound=False, filter_text=False,
filter_btx=False, filter_collada=False, filter_folder=True, filemode=9, relative_path=True,
display_type='FILE_DEFAULTDISPLAY')
#
bpy.ops.sequencer.copy()
# Do cross-fading volume animation of two selected sound strips
bpy.ops.sequencer.crossfade_sounds()
# Cut the selected strips
bpy.ops.sequencer.cut(frame=0, type='SOFT', side='BOTH')
# Cut multi-cam strip and select camera
bpy.ops.sequencer.cut_multicam(camera=1)
# Deinterlace all selected movie sources
bpy.ops.sequencer.deinterlace_selected_movies()
# Erase selected strips from the sequencer
bpy.ops.sequencer.delete()
# Duplicate the selected strips
bpy.ops.sequencer.duplicate(mode='TRANSLATION')
# Duplicate selected strips and move them
bpy.ops.sequencer.duplicate_move(SEQUENCER_OT_duplicate={"mode":'TRANSLATION'},
TRANSFORM_OT_translate={"value":(0, 0, 0), "constraint_axis":(False, False, False),
"constraint_orientation":'GLOBAL', "mirror":False, "proportional":'DISABLED',
"proportional_edit_falloff":'SMOOTH', "proportional_size":1, "snap":False, "snap_target":'CLOSEST',
"snap_point":(0, 0, 0), "snap_align":False, "snap_normal":(0, 0, 0), "texture_space":False, "release_confirm":False})
# Add an effect to the sequencer, most are applied on top of existing strips
bpy.ops.sequencer.effect_strip_add(filepath="", filter_blender=False, filter_backup=False, filter_image=False,
filter_movie=False, filter_python=False, filter_font=False, filter_sound=False, filter_text=False, filter_btx=False,
filter_collada=False, filter_folder=False, filemode=9, relative_path=True,
display_type='FILE_DEFAULTDISPLAY', frame_start=0, frame_end=0, channel=1, replace_sel=True,
overlap=False, type='CROSS', color=(0, 0, 0))
# Insert gap at current frame to first strips at the right, independent of selection or locked state of strips
bpy.ops.sequencer.gap_insert(frames=10)
```

# Remove gap at current frame to first strip at the right, independent of selection or locked state of strips
bpy.ops.sequencer.gap_remove(all=False)
# Add an image or image sequence to the sequencer
bpy.ops.sequencer.image_strip_add(directory="", files=[], filter_blender=False, filter_backup=False, filter_image=True, filter_movie=False, filter_python=False, filter_font=False, filter_sound=False, filter_text=False, filter_btx=False, filter_collada=False, filter_folder=True, filemode=9, relative_path=True, display_type='FILE_DEFAULTDISPLAY', frame_start=0, frame_end=0, channel=1, replace_sel=True, overlap=False)
# On image sequence strips, it returns a strip for each image
bpy.ops.sequencer.images_separate(length=1)
# Lock the active strip so that it can't be transformed
bpy.ops.sequencer.lock()
# Add a mask strip to the sequencer
bpy.ops.sequencer.mask_strip_add(frame_start=0, channel=1, replace_sel=True, overlap=False, mask='<UNKNOWN ENUM>')
# Group selected strips into a metastrip
bpy.ops.sequencer.meta_make()
# Put the contents of a metastrip back in the sequencer
bpy.ops.sequencer.meta_separate()
# Toggle a metastrip (to edit enclosed strips)
bpy.ops.sequencer.meta_toggle()
# Add a movie strip to the sequencer
bpy.ops.sequencer.movie_strip_add(filepath="", files=[], filter_blender=False, filter_backup=False, filter_image=False, filter_movie=True, filter_python=False, filter_font=False, filter_sound=False, filter_text=False, filter_btx=False, filter_collada=False, filter_folder=True, filemode=9, relative_path=True, display_type='FILE_DEFAULTDISPLAY', frame_start=0, channel=1, replace_sel=True, overlap=False, sound=True)
# Add a movieclip strip to the sequencer
bpy.ops.sequencer.movieclip_strip_add(frame_start=0, channel=1, replace_sel=True, overlap=False, clip='<UNKNOWN ENUM>')
# Mute selected strips
bpy.ops.sequencer.mute(unselected=False)
# Clear strip offsets from the start and end frames
bpy.ops.sequencer.offset_clear()
#
bpy.ops.sequencer.paste()
# Open sequencer properties panel
bpy.ops.sequencer.properties()
# Reassign the inputs for the effect strip
bpy.ops.sequencer.reassign_inputs()
# Rebuild all selected proxies and timecode indices using the job system
bpy.ops.sequencer.rebuild_proxy()
# Refresh the sequencer editor
bpy.ops.sequencer.refresh_all()
# Reload strips in the sequencer
bpy.ops.sequencer.reload(adjust_length=False)
# Set render size and aspect from active sequence
bpy.ops.sequencer.rendersize()

```
# Use mouse to sample color in current frame
bpy.ops.sequencer.sample()
# Add a strip to the sequencer using a blender scene as a source
bpy.ops.sequencer.scene_strip_add(frame_start=0, channel=1, replace_sel=True, overlap=False, scene='Scene')
# Select a strip (last selected becomes the "active strip")
bpy.ops.sequencer.select(extend=False, linked_handle=False, left_right=False, linked_time=False)
# Select strips on the nominated side of the active strip
bpy.ops.sequencer.select_active_side(side='BOTH')
# Select or deselect all strips
bpy.ops.sequencer.select_all(action='TOGGLE')
# Enable border select mode
bpy.ops.sequencer.select_border(gesture_mode=0, xmin=0, xmax=0, ymin=0, ymax=0, extend=True)
# Select all strips grouped by various properties
bpy.ops.sequencer.select_grouped(extend=False, type='TYPE')
# Select manipulator handles on the sides of the selected strip
bpy.ops.sequencer.select_handles(side='BOTH')
# Shrink the current selection of adjacent selected strips
bpy.ops.sequencer.select_less()
# Select all strips adjacent to the current selection
bpy.ops.sequencer.select_linked()
# Select a chain of linked strips nearest to the mouse pointer
bpy.ops.sequencer.select_linked_pick(extend=False)
# Select more strips adjacent to the current selection
bpy.ops.sequencer.select_more()
# Frame where selected strips will be snapped
bpy.ops.sequencer.snap(frame=0)
# Add a sound strip to the sequencer
bpy.ops.sequencer.sound_strip_add(filepath="", files=[], filter_blender=False, filter_backup=False,
filter_image=False, filter_movie=False, filter_python=False, filter_font=False, filter_sound=True, filter_text=False,
filter_btx=False, filter_collada=False, filter_folder=True, filemode=9, relative_path=True,
display_type='FILE_DEFAULTDISPLAY', frame_start=0, channel=1, replace_sel=True, overlap=False,
cache=False)
# Move frame to previous edit point
bpy.ops.sequencer.strip_jump(next=True, center=True)
# Add a modifier to strip
bpy.ops.sequencer.strip_modifier_add(type='COLOR_BALANCE')
# Move modifier up and down in the stack
bpy.ops.sequencer.strip_modifier_move(name="Name", direction='UP')
# Add a modifier to strip
bpy.ops.sequencer.strip_modifier_remove(name="Name")
# Swap active strip with strip to the right or left
bpy.ops.sequencer.swap(side='RIGHT')
# Swap 2 sequencer strips
bpy.ops.sequencer.swap_data()
# Swap the first two inputs for the effect strip
bpy.ops.sequencer.swap_inputs()
# Unlock the active strip so that it can't be transformed
```

```
bpy.ops.sequencer.unlock()
# Un-Mute unselected rather than selected strips
bpy.ops.sequencer.unmute(unselected=False)
# View all the strips in the sequencer
bpy.ops.sequencer.view_all()
# Zoom preview to fit in the area
bpy.ops.sequencer.view_all_preview()
# Enable border select mode
bpy.ops.sequencer.view_ghost_border(gesture_mode=0, xmin=0, xmax=0, ymin=0, ymax=0)
# Zoom the sequencer on the selected strips
bpy.ops.sequencer.view_selected()
# Toggle between sequencer views (sequence, preview, both)
bpy.ops.sequencer.view_toggle()
# Change zoom ratio of sequencer preview
bpy.ops.sequencer.view_zoom_ratio(ratio=1)


# Cancel the current sketch stroke
bpy.ops.sketch.cancel_stroke()
# Convert the selected sketch strokes to bone chains
bpy.ops.sketch.convert()
# Delete a sketch stroke
bpy.ops.sketch.delete()
# Draw preview of current sketch stroke (internal use)
bpy.ops.sketch.draw_preview(snap=False)
# Start to draw a sketch stroke
bpy.ops.sketch.draw_stroke(snap=False)
# End and keep the current sketch stroke
bpy.ops.sketch.finish_stroke()
# Start to draw a gesture stroke
bpy.ops.sketch.gesture(snap=False)
# Select a sketch stroke
bpy.ops.sketch.select()


# Update the audio animation cache
bpy.ops.sound.bake_animation()
# Mixes the scene's audio to a sound file
bpy.ops.sound.mixdown(filepath="", check_existing=True, filter_blender=False, filter_backup=False,
filter_image=False, filter_movie=False, filter_python=False, filter_font=False, filter_sound=True, filter_text=False,
filter_btx=False, filter_collada=False, filter_folder=True, filemode=9, relative_path=True,
display_type='FILE_DEFAULTDISPLAY', accuracy=1024, container='FLAC', codec='FLAC', format='S16',
bitrate=192, split_channels=False)
# Load a sound file
bpy.ops.sound.open(filepath="", filter_blender=False, filter_backup=False, filter_image=False, filter_movie=True,
filter_python=False, filter_font=False, filter_sound=True, filter_text=False, filter_btx=False, filter_collada=False,
filter_folder=True, filemode=9, relative_path=True, display_type='FILE_DEFAULTDISPLAY', cache=False,
mono=False)
# Load a sound file as mono
```

bpy.ops.sound.open_mono(filepath="", filter_blender=False, filter_backup=False, filter_image=False, filter_movie=True, filter_python=False, filter_font=False, filter_sound=True, filter_text=False, filter_btx=False, filter_collada=False, filter_folder=True, filemode=9, relative_path=True, display_type='FILE_DEFAULTDISPLAY', cache=False, mono=True)

Pack the sound into the current blend file

bpy.ops.sound.pack()

Unpack the sound to the samples filename

bpy.ops.sound.unpack(method='USE_LOCAL', id="")

Update animation flags

bpy.ops.sound.update_animation_flags()

Construct a Nurbs surface Circle

bpy.ops.surface.primitive_nurbs_surface_circle_add(view_align=False, enter_editmode=False, location=(0, 0, 0), rotation=(0, 0, 0), layers=(False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False))

Construct a Nurbs surface Curve

bpy.ops.surface.primitive_nurbs_surface_curve_add(view_align=False, enter_editmode=False, location=(0, 0, 0), rotation=(0, 0, 0), layers=(False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False))

Construct a Nurbs surface Cylinder

bpy.ops.surface.primitive_nurbs_surface_cylinder_add(view_align=False, enter_editmode=False, location=(0, 0, 0), rotation=(0, 0, 0), layers=(False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False))

Construct a Nurbs surface Sphere

bpy.ops.surface.primitive_nurbs_surface_sphere_add(view_align=False, enter_editmode=False, location=(0, 0, 0), rotation=(0, 0, 0), layers=(False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False))

Construct a Nurbs surface Patch

bpy.ops.surface.primitive_nurbs_surface_surface_add(view_align=False, enter_editmode=False, location=(0, 0, 0), rotation=(0, 0, 0), layers=(False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False))

Construct a Nurbs surface Torus

bpy.ops.surface.primitive_nurbs_surface_torus_add(view_align=False, enter_editmode=False, location=(0, 0, 0), rotation=(0, 0, 0), layers=(False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False))

Show a list of used text in the open document

bpy.ops.text.autocomplete()

Convert selected text to comment

bpy.ops.text.comment()

Convert whitespaces by type

bpy.ops.text.convert_whitespace(type='SPACES')

Copy selected text to clipboard

bpy.ops.text.copy()

Set cursor position

bpy.ops.text.cursor_set(x=0, y=0)

Cut selected text to clipboard

bpy.ops.text.cut()

```
# Delete text by cursor position
bpy.ops.text.delete(type='NEXT_CHARACTER')
# Duplicate the current line
bpy.ops.text.duplicate_line()
# Find specified text
bpy.ops.text.find()
# Find specified text and set as selected
bpy.ops.text.find_set_selected()
# Indent selected text
bpy.ops.text.indent()
# Insert text at cursor position
bpy.ops.text.insert(text="")
# Jump cursor to line
bpy.ops.text.jump(line=1)
# Insert line break at cursor position
bpy.ops.text.line_break()
# The current line number
bpy.ops.text.line_number()
# Make active text file internal
bpy.ops.text.make_internal()
# Move cursor to position type
bpy.ops.text.move(type='LINE_BEGIN')
# Move the currently selected line(s) up/down
bpy.ops.text.move_lines(direction='DOWN')
# Make selection from current cursor position to new cursor position type
bpy.ops.text.move_select(type='LINE_BEGIN')
# Create a new text data block
bpy.ops.text.new()
# Open a new text data block
bpy.ops.text.open(filepath="", filter_blender=False, filter_backup=False, filter_image=False, filter_movie=False,
filter_python=True, filter_font=False, filter_sound=False, filter_text=True, filter_btx=False, filter_collada=False,
filter_folder=True, filemode=9, display_type='FILE_DEFAULTDISPLAY', internal=False)
# Toggle overwrite while typing
bpy.ops.text.overwrite_toggle()
# Paste text from clipboard
bpy.ops.text.paste(selection=False)
# Toggle text properties panel
bpy.ops.text.properties()
# Refresh all pyconstraints
bpy.ops.text.refresh_pyconstraints()
# Reload active text data block from its file
bpy.ops.text.reload()
# Replace text with the specified text
bpy.ops.text.replace()
# Replace text with specified text and set as selected
bpy.ops.text.replace_set_selected()
# When external text is out of sync, resolve the conflict
```

bpy.ops.text.resolve_conflict(resolution='IGNORE')
# Run active script
bpy.ops.text.run_script()
# Save active text data block
bpy.ops.text.save()
# Save active text file with options
bpy.ops.text.save_as(filepath="", check_existing=True, filter_blender=False, filter_backup=False, filter_image=False, filter_movie=False, filter_python=True, filter_font=False, filter_sound=False, filter_text=True, filter_btx=False, filter_collada=False, filter_folder=True, filemode=9, display_type='FILE_DEFAULTDISPLAY')
# Scroll text screen
bpy.ops.text.scroll(lines=1)
# Scroll text screen
bpy.ops.text.scroll_bar(lines=1)
# Select all text
bpy.ops.text.select_all()
# Select text by line
bpy.ops.text.select_line()
# Select word under cursor
bpy.ops.text.select_word()
# Set cursor selection
bpy.ops.text.selection_set(select=False)
# Start searching text
bpy.ops.text.start_find()
# Create 3D text object from active text data block
bpy.ops.text.to_3d_object(split_lines=False)
# Convert selected comment to text
bpy.ops.text.uncomment()
# Unindent selected text
bpy.ops.text.unindent()
# Unlink active text data block
bpy.ops.text.unlink()

# Discard the environment map and free it from memory
bpy.ops.texture.envmap_clear()
# Discard all environment maps in the .blend file and free them from memory
bpy.ops.texture.envmap_clear_all()
# Save the current generated Environment map to an image file
bpy.ops.texture.envmap_save(layout=(0, 0, 1, 0, 2, 0, 0, 1, 1, 1, 2, 1), filepath="", check_existing=True, filter_blender=False, filter_backup=False, filter_image=True, filter_movie=True, filter_python=False, filter_font=False, filter_sound=False, filter_text=False, filter_btx=False, filter_collada=False, filter_folder=True, filemode=9, display_type='FILE_DEFAULTDISPLAY')
# Add a new texture
bpy.ops.texture.new()
# Copy the material texture settings and nodes
bpy.ops.texture.slot_copy()
# Move texture slots up and down
bpy.ops.texture.slot_move(type='UP')

```
# Copy the texture settings and nodes
bpy.ops.texture.slot_paste()


# Set the end frame
bpy.ops.time.end_frame_set()
# Set the start frame
bpy.ops.time.start_frame_set()
# Show the entire playable frame range
bpy.ops.time.view_all()


# Create transformation orientation from selection
bpy.ops.transform.create_orientation(name="", use_view=False, use=False, overwrite=False)
# Delete transformation orientation
bpy.ops.transform.delete_orientation()
# Change the bevel weight of edges
bpy.ops.transform.edge_bevelweight(value=0, snap=False, snap_target='CLOSEST', snap_point=(0, 0, 0),
snap_align=False, snap_normal=(0, 0, 0), release_confirm=False)
# Change the crease of edges
bpy.ops.transform.edge_crease(value=0, snap=False, snap_target='CLOSEST', snap_point=(0, 0, 0),
snap_align=False, snap_normal=(0, 0, 0), release_confirm=False)
# Slide an edge loop along a mesh
bpy.ops.transform.edge_slide(value=0, mirror=False, snap=False, snap_target='CLOSEST', snap_point=(0, 0, 0),
snap_align=False, snap_normal=(0, 0, 0), correct_uv=False, release_confirm=False)
# Mirror selected vertices around one or more axes
bpy.ops.transform.mirror(constraint_axis=(False, False, False), constraint_orientation='GLOBAL',
proportional='DISABLED', proportional_edit_falloff='SMOOTH', proportional_size=1, release_confirm=False)
# Push/Pull selected items
bpy.ops.transform.push_pull(value=0, mirror=False, proportional='DISABLED',
proportional_edit_falloff='SMOOTH', proportional_size=1, snap=False, snap_target='CLOSEST', snap_point=(0,
0, 0), snap_align=False, snap_normal=(0, 0, 0), release_confirm=False)
# Scale (resize) selected items
bpy.ops.transform.resize(value=(1, 1, 1), constraint_axis=(False, False, False), constraint_orientation='GLOBAL',
mirror=False, proportional='DISABLED', proportional_edit_falloff='SMOOTH', proportional_size=1, snap=False,
snap_target='CLOSEST', snap_point=(0, 0, 0), snap_align=False, snap_normal=(0, 0, 0), texture_space=False,
release_confirm=False)
# Rotate selected items
bpy.ops.transform.rotate(value=0, axis=(0, 0, 0), constraint_axis=(False, False, False),
constraint_orientation='GLOBAL', mirror=False, proportional='DISABLED',
proportional_edit_falloff='SMOOTH', proportional_size=1, snap=False, snap_target='CLOSEST', snap_point=(0,
0, 0), snap_align=False, snap_normal=(0, 0, 0), release_confirm=False)
# Select transformation orientation
bpy.ops.transform.select_orientation(orientation='GLOBAL')
# Slide a sequence strip in time
bpy.ops.transform.seq_slide(value=(0, 0), snap=False, snap_target='CLOSEST', snap_point=(0, 0, 0),
snap_align=False, snap_normal=(0, 0, 0), release_confirm=False)
# Shear selected items along the horizontal screen axis
bpy.ops.transform.shear(value=0, mirror=False, proportional='DISABLED', proportional_edit_falloff='SMOOTH',
proportional_size=1, snap=False, snap_target='CLOSEST', snap_point=(0, 0, 0), snap_align=False,
```

snap_normal=(0, 0, 0), release_confirm=False)

Shrink/fatten selected vertices along normals

bpy.ops.transform.shrink_fatten(value=0, mirror=False, proportional='DISABLED', proportional_edit_falloff='SMOOTH', proportional_size=1, snap=False, snap_target='CLOSEST', snap_point=(0, 0, 0), snap_align=False, snap_normal=(0, 0, 0), release_confirm=False)

Scale selected vertices' skin radii

bpy.ops.transform.skin_resize(value=(1, 1, 1), constraint_axis=(False, False, False), constraint_orientation='GLOBAL', mirror=False, proportional='DISABLED', proportional_edit_falloff='SMOOTH', proportional_size=1, snap=False, snap_target='CLOSEST', snap_point=(0, 0, 0), snap_align=False, snap_normal=(0, 0, 0), texture_space=False, release_confirm=False)

Tilt selected control vertices of 3D curve

bpy.ops.transform.tilt(value=0, mirror=False, proportional='DISABLED', proportional_edit_falloff='SMOOTH', proportional_size=1, snap=False, snap_target='CLOSEST', snap_point=(0, 0, 0), snap_align=False, snap_normal=(0, 0, 0), release_confirm=False)

Move selected vertices outward in a spherical shape around mesh center

bpy.ops.transform.tosphere(value=0, mirror=False, proportional='DISABLED', proportional_edit_falloff='SMOOTH', proportional_size=1, snap=False, snap_target='CLOSEST', snap_point=(0, 0, 0), snap_align=False, snap_normal=(0, 0, 0), release_confirm=False)

Trackball style rotation of selected items

bpy.ops.transform.trackball(value=(0, 0), mirror=False, proportional='DISABLED', proportional_edit_falloff='SMOOTH', proportional_size=1, snap=False, snap_target='CLOSEST', snap_point=(0, 0, 0), snap_align=False, snap_normal=(0, 0, 0), release_confirm=False)

Transform selected items by mode type

bpy.ops.transform.transform(mode='TRANSLATION', value=(0, 0, 0, 0), axis=(0, 0, 0), constraint_axis=(False, False, False), constraint_orientation='GLOBAL', mirror=False, proportional='DISABLED', proportional_edit_falloff='SMOOTH', proportional_size=1, snap=False, snap_target='CLOSEST', snap_point=(0, 0, 0), snap_align=False, snap_normal=(0, 0, 0), release_confirm=False)

Translate (move) selected items

bpy.ops.transform.translate(value=(0, 0, 0), constraint_axis=(False, False, False), constraint_orientation='GLOBAL', mirror=False, proportional='DISABLED', proportional_edit_falloff='SMOOTH', proportional_size=1, snap=False, snap_target='CLOSEST', snap_point=(0, 0, 0), snap_align=False, snap_normal=(0, 0, 0), texture_space=False, release_confirm=False)

Slide a vertex along a mesh

bpy.ops.transform.vert_slide(value=0, mirror=False, snap=False, snap_target='CLOSEST', snap_point=(0, 0, 0), snap_align=False, snap_normal=(0, 0, 0), release_confirm=False)

Warp selected items around the cursor

bpy.ops.transform.warp(value=(0,), mirror=False, proportional='DISABLED', proportional_edit_falloff='SMOOTH', proportional_size=1, snap=False, snap_target='CLOSEST', snap_point=(0, 0, 0), snap_align=False, snap_normal=(0, 0, 0), release_confirm=False)

Copy the RNA data path for this property to the clipboard

bpy.ops.ui.copy_data_path_button()

Copy property from this object to selected objects or bones

bpy.ops.ui.copy_to_selected_button(all=True)

Edit UI source code of the active button

bpy.ops.ui.editsource()

Edit i18n in current language for the active button

bpy.ops.ui.edittranslation_init()

Sample a color from the Blender Window to store in a property

```
bpy.ops.ui.eyedropper()
# Force a full reload of UI translation
bpy.ops.ui.reloadtranslation()
# Write the reports
bpy.ops.ui.reports_to_textblock()
# Reset this property's value to its default value
bpy.ops.ui.reset_default_button(all=True)
# Reset to the default theme colors
bpy.ops.ui.reset_default_theme()


# Align selected UV vertices to an axis
bpy.ops.uv.align(axis='ALIGN_AUTO')
# Average the size of separate UV islands, based on their area in 3D space
bpy.ops.uv.average_islands_scale()
# Select UV vertices using circle selection
bpy.ops.uv.circle_select(x=0, y=0, radius=0, gesture_mode=0)
# Project the UV vertices of the mesh over the six faces of a cube
bpy.ops.uv.cube_project(cube_size=1, correct_aspect=True, clip_to_bounds=False, scale_to_bounds=False)
# Set 2D cursor location
bpy.ops.uv.cursor_set(location=(0, 0))
# Project the UV vertices of the mesh over the curved wall of a cylinder
bpy.ops.uv.cylinder_project(direction='VIEW_ON_EQUATOR', align='POLAR_ZX', radius=1,
correct_aspect=True, clip_to_bounds=False, scale_to_bounds=False)
# Export UV layout to file
bpy.ops.uv.export_layout(filepath="", check_existing=True, export_all=False, modified=False, mode='PNG',
size=(1024, 1024), opacity=0.25, tessellated=False)
# Follow UVs from active quads along continuous face loops
bpy.ops.uv.follow_active_quads(mode='LENGTH_AVERAGE')
# Hide (un)selected UV vertices
bpy.ops.uv.hide(unselected=False)
# Follow UVs from active quads along continuous face loops
bpy.ops.uv.lightmap_pack(PREF_CONTEXT='SEL_FACES', PREF_PACK_IN_ONE=True,
PREF_NEW_UVLAYER=False, PREF_APPLY_IMAGE=False, PREF_IMG_PX_SIZE=512,
PREF_BOX_DIV=12, PREF_MARGIN_DIV=0.1)
# Mark selected UV edges as seams
bpy.ops.uv.mark_seam()
# Reduce UV stretching by relaxing angles
bpy.ops.uv.minimize_stretch(fill_holes=True, blend=0, iterations=0)
# Transform all islands so that they fill up the UV space as much as possible
bpy.ops.uv.pack_islands(margin=0.001)
# Set/clear selected UV vertices as anchored between multiple unwrap operations
bpy.ops.uv.pin(clear=False)
# Project the UV vertices of the mesh as seen in current 3D view
bpy.ops.uv.project_from_view(orthographic=False, camera_bounds=True, correct_aspect=True,
clip_to_bounds=False, scale_to_bounds=False)
# Selected UV vertices that are within a radius of each other are welded together
bpy.ops.uv.remove_doubles(threshold=0.02, use_unselected=False)
```

```
# Reset UV projection
bpy.ops.uv.reset()
# Reveal all hidden UV vertices
bpy.ops.uv.reveal()
# Set mesh seams according to island setup in the UV editor
bpy.ops.uv.seams_from_islands(mark_seams=True, mark_sharp=False)
# Select UV vertices
bpy.ops.uv.select(extend=False, location=(0, 0))
# Change selection of all UV vertices
bpy.ops.uv.select_all(action='TOGGLE')
# Select UV vertices using border selection
bpy.ops.uv.select_border(pinned=False, gesture_mode=0, xmin=0, xmax=0, ymin=0, ymax=0, extend=True)
# Select UVs using lasso selection
bpy.ops.uv.select_lasso(path=[], deselect=False, extend=True)
# Deselect UV vertices at the boundary of each selection region
bpy.ops.uv.select_less()
# Select all UV vertices linked to the active UV map
bpy.ops.uv.select_linked(extend=False)
# Select all UV vertices linked under the mouse
bpy.ops.uv.select_linked_pick(extend=False, location=(0, 0))
# Select a loop of connected UV vertices
bpy.ops.uv.select_loop(extend=False, location=(0, 0))
# Select more UV vertices connected to initial selection
bpy.ops.uv.select_more()
# Select all pinned UV vertices
bpy.ops.uv.select_pinned()
# Select only entirely selected faces
bpy.ops.uv.select_split()
# This script projection unwraps the selected faces of a mesh (it operates on all selected mesh objects, and can be
used to unwrap selected faces, or all faces)
bpy.ops.uv.smart_project(angle_limit=66, island_margin=0, user_area_weight=0)
# Snap cursor to target type
bpy.ops.uv.snap_cursor(target='PIXELS')
# Snap selected UV vertices to target type
bpy.ops.uv.snap_selected(target='PIXELS')
# Project the UV vertices of the mesh over the curved surface of a sphere
bpy.ops.uv.sphere_project(direction='VIEW_ON_EQUATOR', align='POLAR_ZX', correct_aspect=True,
clip_to_bounds=False, scale_to_bounds=False)
# Stitch selected UV vertices by proximity
bpy.ops.uv.stitch(use_limit=False, snap_islands=True, limit=0.01, static_island=0, midpoint_snap=False,
clear_seams=True, mode='VERTEX', stored_mode='VERTEX', selection=[])
# Set UV image tile coordinates
bpy.ops.uv.tile_set(tile=(0, 0))
# Unwrap the mesh of the object being edited
bpy.ops.uv.unwrap(method='ANGLE_BASED', fill_holes=True, correct_aspect=True, use_subsurf_data=False,
margin=0.001)
# Weld selected UV vertices together
```

```
bpy.ops.uv.weld()


# Pan the view
bpy.ops.view2d.pan(deltax=0, deltay=0)
# Reset the view
bpy.ops.view2d.reset()
# Scroll the view down
bpy.ops.view2d.scroll_down(deltax=0, deltay=0, page=False)
# Scroll the view left
bpy.ops.view2d.scroll_left(deltax=0, deltay=0)
# Scroll the view right
bpy.ops.view2d.scroll_right(deltax=0, deltay=0)
# Scroll the view up
bpy.ops.view2d.scroll_up(deltax=0, deltay=0, page=False)
# Scroll view by mouse click and drag
bpy.ops.view2d.scroller_activate()
# Zoom in the view to the nearest item contained in the border
bpy.ops.view2d.smoothview(gesture_mode=0, xmin=0, xmax=0, ymin=0, ymax=0)
# Zoom in/out the view
bpy.ops.view2d.zoom(deltax=0, deltay=0)
# Zoom in the view to the nearest item contained in the border
bpy.ops.view2d.zoom_border(gesture_mode=0, xmin=0, xmax=0, ymin=0, ymax=0)
# Zoom in the view
bpy.ops.view2d.zoom_in(zoomfacx=0, zoomfacy=0)
# Zoom out the view
bpy.ops.view2d.zoom_out(zoomfacx=0, zoomfacy=0)


# Add a new background image
bpy.ops.view3d.background_image_add(name="Image", filepath="Path")
# Remove a background image from the 3D view
bpy.ops.view3d.background_image_remove(index=0)
# Set camera view to active view
bpy.ops.view3d.camera_to_view()
# Move the camera so selected objects are framed
bpy.ops.view3d.camera_to_view_selected()
# Clear the boundaries of the border render and disable border render
bpy.ops.view3d.clear_render_border()
# Set the view clipping border
bpy.ops.view3d.clip_border(xmin=0, xmax=0, ymin=0, ymax=0)
# Selected objects are saved in a temp file
bpy.ops.view3d.copybuffer()
# Set the location of the 3D cursor
bpy.ops.view3d.cursor3d()
# Dolly in/out in the view
bpy.ops.view3d.dolly(delta=0, mx=0, my=0)
# Extrude individual elements and move
```

bpy.ops.view3d.edit_mesh_extrude_individual_move()
# Extrude and move along normals
bpy.ops.view3d.edit_mesh_extrude_move_normal()
# Enable the transform manipulator for use
bpy.ops.view3d.enable_manipulator(translate=False, rotate=False, scale=False)
# Interactively fly around the scene
bpy.ops.view3d.fly()
# Start game engine
bpy.ops.view3d.game_start()
# Toggle layer(s) visibility
bpy.ops.view3d.layers(nr=1, extend=False, toggle=True)
# Toggle display of selected object(s) separately and centered in view
bpy.ops.view3d.localview()
# Manipulate selected item by axis
bpy.ops.view3d.manipulator(constraint_axis=(False, False, False), constraint_orientation='GLOBAL',
release_confirm=False)
# Move the view
bpy.ops.view3d.move()
# Position your viewpoint with the 3D mouse
bpy.ops.view3d.ndof_all()
# Explore every angle of an object using the 3D mouse
bpy.ops.view3d.ndof_orbit()
# Explore every angle of an object using the 3D mouse
bpy.ops.view3d.ndof_orbit_zoom()
# Position your viewpoint with the 3D mouse
bpy.ops.view3d.ndof_pan()
# Set the active object as the active camera for this view or scene
bpy.ops.view3d.object_as_camera()
# Contents of copy buffer gets pasted
bpy.ops.view3d.pastebuffer()
# Toggles the properties panel display
bpy.ops.view3d.properties()
# Set the boundaries of the border render and enable border render
bpy.ops.view3d.render_border(xmin=0, xmax=0, ymin=0, ymax=0, camera_only=False)
# Rotate the view
bpy.ops.view3d.rotate()
# Interactive ruler
bpy.ops.view3d.ruler()
# Activate/select item(s)
bpy.ops.view3d.select(extend=False, deselect=False, toggle=False, center=False, enumerate=False, object=False,
location=(0, 0))
# Select items using border selection
bpy.ops.view3d.select_border(gesture_mode=0, xmin=0, xmax=0, ymin=0, ymax=0, extend=True)
# Select items using circle selection
bpy.ops.view3d.select_circle(x=0, y=0, radius=0, gesture_mode=0)
# Select items using lasso selection
bpy.ops.view3d.select_lasso(path=[], deselect=False, extend=True)

```
# Menu object selection
bpy.ops.view3d.select_menu(name='<UNKNOWN ENUM>', toggle=False)
# Select element under the mouse, deselect everything is there's nothing under the mouse
bpy.ops.view3d.select_or_deselect_all(extend=False, toggle=False, deselect=False, center=False, enumerate=False,
object=False)
# The time to animate the change of view (in milliseconds)
bpy.ops.view3d.smoothview()
# Snap cursor to active item
bpy.ops.view3d.snap_cursor_to_active()
# Snap cursor to the Center
bpy.ops.view3d.snap_cursor_to_center()
# Snap cursor to nearest grid node
bpy.ops.view3d.snap_cursor_to_grid()
# Snap cursor to center of selected item(s)
bpy.ops.view3d.snap_cursor_to_selected()
# Snap selected item(s) to cursor
bpy.ops.view3d.snap_selected_to_cursor()
# Snap selected item(s) to nearest grid node
bpy.ops.view3d.snap_selected_to_grid()
# Toggles tool shelf display
bpy.ops.view3d.toolshelf()
# View all objects in scene
bpy.ops.view3d.view_all(use_all_regions=False, center=False)
# Center the camera view
bpy.ops.view3d.view_center_camera()
# Center the view so that the cursor is in the middle of the view
bpy.ops.view3d.view_center_cursor()
# Center the view to the Z-depth position under the mouse cursor
bpy.ops.view3d.view_center_pick()
# Clear all view locking
bpy.ops.view3d.view_lock_clear()
# Lock the view to the active object/bone
bpy.ops.view3d.view_lock_to_active()
# Orbit the view
bpy.ops.view3d.view_orbit(type='ORBITLEFT')
# Pan the view
bpy.ops.view3d.view_pan(type='PANLEFT')
# Switch the current view from perspective/orthographic projection
bpy.ops.view3d.view_persportho()
# Move the view to the selection center
bpy.ops.view3d.view_selected(use_all_regions=False)
# Use a preset viewpoint
bpy.ops.view3d.viewnumpad(type='FRONT', align_active=False)
# Zoom in/out in the view
bpy.ops.view3d.zoom(delta=0, mx=0, my=0)
# Zoom in the view to the nearest object contained in the border
bpy.ops.view3d.zoom_border(gesture_mode=0, xmin=0, xmax=0, ymin=0, ymax=0)
```

# Match the camera to 1:1 to the render output
bpy.ops.view3d.zoom_camera_1_to_1()

# Disable an addon
bpy.ops.wm.addon_disable(module="")
# Enable an addon
bpy.ops.wm.addon_enable(module="")
# Display more information on this addon
bpy.ops.wm.addon_expand(module="")
# Install an addon
bpy.ops.wm.addon_install(overwrite=True, target='DEFAULT', filepath="", filter_folder=True, filter_python=True, filter_glob="*.py;*.zip")
# Disable an addon
bpy.ops.wm.addon_remove(module="")
#
bpy.ops.wm.appconfig_activate(filepath="")
#
bpy.ops.wm.appconfig_default()
# Launch the blender-player with the current blend-file
bpy.ops.wm.blenderplayer_start()
# Call (draw) a pre-defined menu
bpy.ops.wm.call_menu(name="")
# Save a Collada file
bpy.ops.wm.collada_export(filepath="", check_existing=True, filter_blender=False, filter_backup=False, filter_image=False, filter_movie=False, filter_python=False, filter_font=False, filter_sound=False, filter_text=False, filter_btx=False, filter_collada=True, filter_folder=True, filemode=8, display_type='FILE_DEFAULTDISPLAY', apply_modifiers=False, export_mesh_type=0, export_mesh_type_selection='view', selected=False, include_children=False, include_armatures=False, include_shapekeys=True, deform_bones_only=False, active_uv_only=False, include_uv_textures=False, include_material_textures=False, use_texture_copies=True, triangulate=True, use_object_instantiation=True, sort_by_name=False, export_transformation_type=0, export_transformation_type_selection='matrix', second_life=False)
# Load a Collada file
bpy.ops.wm.collada_import(filepath="", filter_blender=False, filter_backup=False, filter_image=False, filter_movie=False, filter_python=False, filter_font=False, filter_sound=False, filter_text=False, filter_btx=False, filter_collada=True, filter_folder=True, filemode=8, display_type='FILE_DEFAULTDISPLAY', import_units=False)
# Set boolean values for a collection of items
bpy.ops.wm.context_collection_boolean_set(data_path_iter="", data_path_item="", type='TOGGLE')
# Set a context array value (useful for cycling the active mesh edit mode)
bpy.ops.wm.context_cycle_array(data_path="", reverse=False)
# Toggle a context value
bpy.ops.wm.context_cycle_enum(data_path="", reverse=False)
# Set a context value (useful for cycling active material, vertex keys, groups, etc.)
bpy.ops.wm.context_cycle_int(data_path="", reverse=False)
#
bpy.ops.wm.context_menu_enum(data_path="")
# Adjust arbitrary values with mouse input
bpy.ops.wm.context_modal_mouse(data_path_iter="", data_path_item="", header_text="", input_scale=0.01, invert=False, initial_x=0)

```
# Scale an int context value
bpy.ops.wm.context_scale_int(data_path="", value=1, always_step=True)
# Set a context value
bpy.ops.wm.context_set_boolean(data_path="", value=True)
# Set a context value
bpy.ops.wm.context_set_enum(data_path="", value="")
# Set a context value
bpy.ops.wm.context_set_float(data_path="", value=0, relative=False)
# Toggle a context value
bpy.ops.wm.context_set_id(data_path="", value="")
# Set a context value
bpy.ops.wm.context_set_int(data_path="", value=0, relative=False)
# Set a context value
bpy.ops.wm.context_set_string(data_path="", value="")
# Set a context value
bpy.ops.wm.context_set_value(data_path="", value="")
# Toggle a context value
bpy.ops.wm.context_toggle(data_path="")
# Toggle a context value
bpy.ops.wm.context_toggle_enum(data_path="", value_1="", value_2="")
# Copy settings from previous version
bpy.ops.wm.copy_prev_settings()
# Open a popup to set the debug level
bpy.ops.wm.debug_menu(debug_value=0)
# Print dependency graph relations to the console
bpy.ops.wm.dependency_relations()
# Load online reference docs
bpy.ops.wm.doc_edit(doc_id="", doc_new="")
# Load online reference docs
bpy.ops.wm.doc_view(doc_id="")
# Load online manual
bpy.ops.wm.doc_view_manual(doc_id="")
# Add an Application Interaction Preset
bpy.ops.wm.interaction_preset_add(remove_active=False, name="")
# Add a theme preset
bpy.ops.wm.interface_theme_preset_add(remove_active=False, name="")
#
bpy.ops.wm.keyconfig_activate(filepath="")
# Export key configuration to a python script
bpy.ops.wm.keyconfig_export(filepath="keymap.py", filter_folder=True, filter_text=True, filter_python=True)
# Import key configuration from a python script
bpy.ops.wm.keyconfig_import(filepath="keymap.py", filter_folder=True, filter_text=True, filter_python=True,
keep_original=True)
# Add a Key-config Preset
bpy.ops.wm.keyconfig_preset_add(remove_active=False, name="")
# Remove key config
bpy.ops.wm.keyconfig_remove()
```

```
# Test key-config for conflicts
bpy.ops.wm.keyconfig_test()
# Add key map item
bpy.ops.wm.keyitem_add()
# Remove key map item
bpy.ops.wm.keyitem_remove(item_id=0)
# Restore key map item
bpy.ops.wm.keyitem_restore(item_id=0)
# Restore key map(s)
bpy.ops.wm.keymap_restore(all=False)
# Link or Append from a Library .blend file
bpy.ops.wm.link_append(filepath="", directory="", filename="", files=[], filter_blender=True, filter_backup=False,
filter_image=False, filter_movie=False, filter_python=False, filter_font=False, filter_sound=False, filter_text=False,
filter_btx=False, filter_collada=False, filter_folder=True, filemode=1, relative_path=True,
display_type='FILE_DEFAULTDISPLAY', link=True, autoselect=True, active_layer=True, instance_groups=True)
# Print memory statistics to the console
bpy.ops.wm.memory_statistics()
# Change NDOF sensitivity
bpy.ops.wm.ndof_sensitivity_change(decrease=True, fast=False)
# Open a Blender file
bpy.ops.wm.open_mainfile(filepath="", filter_blender=True, filter_backup=False, filter_image=False,
filter_movie=False, filter_python=False, filter_font=False, filter_sound=False, filter_text=False, filter_btx=False,
filter_collada=False, filter_folder=True, filemode=8, display_type='FILE_DEFAULTDISPLAY', load_ui=True,
use_scripts=True)
#
bpy.ops.wm.operator_cheat_sheet()
# Set the active operator to its default values
bpy.ops.wm.operator_defaults()
# Add an Operator Preset
bpy.ops.wm.operator_preset_add(remove_active=False, name="", operator="")
# Open a path in a file browser
bpy.ops.wm.path_open(filepath="")
#
bpy.ops.wm.properties_add(data_path="")
# Jump to a different tab inside the properties editor
bpy.ops.wm.properties_context_change(context="")
#
bpy.ops.wm.properties_edit(data_path="", property="", value="", min=0, max=1, description="")
# Internal use (edit a property data_path)
bpy.ops.wm.properties_remove(data_path="", property="")
# Quit Blender
bpy.ops.wm.quit_blender()
# Set some size property (like e.g. brush size) with mouse wheel
bpy.ops.wm.radial_control(data_path_primary="", data_path_secondary="", use_secondary="", rotation_path="",
color_path="", fill_color_path="", zoom_path="", image_id="")
# Load default file and user preferences
bpy.ops.wm.read_factory_settings()
# Reloads history and bookmarks
```

bpy.ops.wm.read_history()
# Open the default file (doesn't save the current file)
bpy.ops.wm.read_homefile()
# Open an automatically saved file to recover it
bpy.ops.wm.recover_auto_save(filepath="", filter_blender=True, filter_backup=False, filter_image=False, filter_movie=False, filter_python=False, filter_font=False, filter_sound=False, filter_text=False, filter_btx=False, filter_collada=False, filter_folder=False, filemode=8, display_type='FILE_LONGDISPLAY')
# Open the last closed file ("quit.blend")
bpy.ops.wm.recover_last_session()
# Simple redraw timer to test the speed of updating the interface
bpy.ops.wm.redraw_timer(type='DRAW', iterations=10)
# Save the current file in the desired location
bpy.ops.wm.save_as_mainfile(filepath="", check_existing=True, filter_blender=True, filter_backup=False, filter_image=False, filter_movie=False, filter_python=False, filter_font=False, filter_sound=False, filter_text=False, filter_btx=False, filter_collada=False, filter_folder=True, filemode=8, display_type='FILE_DEFAULTDISPLAY', compress=False, relative_remap=True, copy=False, use_mesh_compat=False)
# Make the current file the default .blend file, includes preferences
bpy.ops.wm.save_homefile()
# Save the current Blender file
bpy.ops.wm.save_mainfile(filepath="", check_existing=True, filter_blender=True, filter_backup=False, filter_image=False, filter_movie=False, filter_python=False, filter_font=False, filter_sound=False, filter_text=False, filter_btx=False, filter_collada=False, filter_folder=True, filemode=8, display_type='FILE_DEFAULTDISPLAY', compress=False, relative_remap=False)
# Save user preferences separately, overrides startup file preferences
bpy.ops.wm.save_userpref()
# Pop-up a search menu over all available operators in current context
bpy.ops.wm.search_menu()
# Opens a blocking popup region with release info
bpy.ops.wm.splash()
# Generate System Info
bpy.ops.wm.sysinfo()
# Load and apply a Blender XML theme file
bpy.ops.wm.theme_install(overwrite=True, filepath="", filter_folder=True, filter_glob="*.xml")
# Open a website in the web-browser
bpy.ops.wm.url_open(url="")
# (undocumented operator)
bpy.ops.wm.userpref_autoexec_path_add()
# (undocumented operator)
bpy.ops.wm.userpref_autoexec_path_remove(index=0)
# Duplicate the current Blender window
bpy.ops.wm.window_duplicate()
# Toggle the current window fullscreen
bpy.ops.wm.window_fullscreen_toggle()

# Add a new world
bpy.ops.world.new()

CHAPTER 10

Contests

Programming Python is fun.
Programming Python in Blender is more fun.
Showing how funny it is, a programming contest is offered.

Programming contest 2013

Two winners of a beginners programming competition showed their results in the opening keynote of the German Python conference in Cologne. They have to use Python to program Blender. Both showed slides and also their animation movie.

A 15 year old girl visualized Conway's "Game of life" and a boy, 13 years showed a card game "skat" having good software design skills.

The conference and also the competition is driven by the german Python Software Verband. This is a national equivalent to the Python Software Foundation. The contest will continue and likely start again in november and been celebrated at the next EuroPython in Berlin 21.7 – 27.7.2014.

Programming contest 2014

The contest will continue and likely start again in november and been celebrated at the next EuroPython in Berlin 21.7 – 27.7.2014.

CHAPTER 11

Incubator

Please insert new learning units (stations) here, as long as it is not clear where to place the new content. If you have an proposal, enhancement or bug use the ticket system at bitbucket.org.

CHAPTER 12

Indices and tables

- genindex
- search

P

R

S

T