🚀 Blender Python Cheatsheet

This cheatsheet has a collection of most frequently used Python API functions and solutions to common problems that you will encounter when writing scripts, macros or developing addons for Blender.

Command Line

```
# Run Blender and enter REPL
$ blender --python-console

# Run a script from command line
$ blender model.blend -P script.py

# Run script in background (without UI)
$ blender model.blend -b -P script.py

# Passing custom command line arguments
# Anything after -- will be ignored by Blender
$ blender model.blend -b -P script.py -- arg1 arg2

# And you can access them using sys.argv
>>> import sys
>>> sys.argv[6]

# NOTE: You need to replace 'blender' with full path to
# the Blender executable if it is not on PATH
```

Objects

Get / Set active object

```
# Get active object
bpy.context.active_object

# Set active object
bpy.context.view_layer.objects.active = object_to_set_active
```

Select / Deselect objects

```
# Select an object
obj.select_set(True)

# Deelect an object
obj.select_set(False)
```

Check if object is selected

```
obj.select_get()
# Returns True if selected, False if otherwise
```

Select / Deselect all objects

```
# Select all
bpy.ops.object.select_all(action='SELECT')

# Deselect all
bpy.ops.object.select_all(action='DESELECT')
```

Convert objects to type

```
bpy.ops.object.convert(target='OBJECT_TYPE')

# E.g. Convert all selected objects to Mesh
bpy.ops.object.convert(target='MESH')
```

Get all objects of type

```
def get_objects_of_type(obj_type):
    objects = []
    for obj in data.objects:
        if obj.type == obj_type:
            objects.append(obj)
    return objects

# Then you can call it using any type like:
get_objects_of_type('MESH')  # meshes
get_objects_of_type('CURVE')  # curves
get_objects_of_type('LIGHT')  # light
```

Object Modes

```
# Toggle Edit Mode
bpy.ops.object.editmode_toggle()

# Get current mode
bpy.context.mode
# Returns 'OBJECT' in Object Mode, 'EDIT_MESH' in Edit Mode, etc.

# Currently there's a discrepancy between this and
# bpy.ops.object.mode_set
# for Edit Mode. The latter only acceps 'EDIT' as the value

# Set mode
bpy.ops.object.mode_set(mode='MODE')

# E.g. To set Edit Mode
bpy.ops.object.mode_set(mode='EDIT')
```

Transforms

Select / Deselect objects

```
# Get location of an object
obj.location
obj.location.x  # individual components
obj.location[0]  # individual components using index

# Set location of an object
obj.location = 1, 1, 1  # x, y, x
obj.location.x = 3

# Set location relative to current location
obj.location += 1, 0, 1
obj.location.y += 2

# NOTE: The same concepts apply to rotation and scale
# You can also change the 'dimensions' for precision

# In case of rotation, Blender expects radians as opposed
# to degrees that you use in UI. To handle coversions, do:
obj.rotation_euler.x = math.radians(45)  # import math
```

Reset Transforms

```
bpy.ops.object.location_clear()  # Location
bpy.ops.object.rotation_clear()  # Rotation
bpy.ops.object.scale_clear()  # Scale
```

Apply Transforms

```
bpy.ops.object.transform_apply(location, rotation, scale)

# Provide location, rotation and scale params with True or False
# depending on what you want to apply. E.g.
bpy.ops.object.transform_apply(location=True)
```

🚀 Blender Python Cheatsheet

UV Maps

Create new UV map

```
bpy.context.object.data.uv_layers.new(name="uv_map_name")
```

Set active UV map (Viewport)

```
bpy.context.object.data.uv_layers.active_index = 2  # index

# Alternatively, if you have a UV Map object stored in
# a variable, you can also use the 'active' attribute.
# E.g. If you have a variable uv_map_obj like this ...
uv_map_obj = bpy.context.object.data.uv_layers["UVMap"]

# ... you can set the active UV layer like:
bpy.context.object.data.uv_layers.active = uv_map_obj
```

Set active UV map (Render)

```
bpy.object.data.uv_layers['UVMap'].active_render = True
```

Delete a UV map

```
# First set the active UV map for the viewport
bpy.context.object.data.uv_layers.active = uv_map_obj

# And then do:
bpy.ops.mesh.uv_texture_remove()
```

Smart UV Project

```
ops.uv.smart_project(angle_limit, island_margin, area_weight)

# 'angle_limit' takes radian values. All three are optional
ops.uv.smart_project(angle_limit=math.radians(45))
```

Nodes

Examples in this section are written for Material Nodes, but the same concepts are applicable to compositing or geometry nodes.

Get the node tree of a material

```
# If you already have the material in a variable
material.node_tree

# If you don't have the material in a variable
bpy.data.materials['mat_name'].node_tree

# Alternatively, you can access a material two more ways
# 1. Via object > material slots > material like:
bpy.data.objects['obj_name'].material_slots[index].material.node_tree

# 2. Via object > data > material like:
bpy.data.objects['obj_name'].data.materials['mat_name'].node_tree
```

Get active node

```
node_tree.nodes.active
```

Set active node

```
node_tree.nodes.active = node

# E.g. To make an Image Texture node active
node_tree.nodes.active = node_tree.nodes['Image Texture']
```

Select/Deselect a node

```
node.select = True  # or False

# E.g. To select/deselect an Image Texture node active
node_tree.nodes['Image Texture'].select = True  # or False
```

Create a new node

```
node_tree.nodes.new('NodeType')

# E.g. To add an Image Texture
node_tree.nodes.new('ShaderNodeTexImage')
```

Delete a node

```
node_tree.nodes.remove(node)

# E.g. To remove an Image Texture Node
node_tree.remove(node_tree.nodes['Image Texture'])
```

Connect node sockets

```
node_tree.links.new(
    node_a.outputs['socket_name'],
    node_b.inputs['socket_name']
)

# E.g. To connect an Image Texure's Color socket to a
# Principled BSDF's Base Color socket, you would:
material = bpy.data.materials['Wood'].node_tree
node_image = node_tree.nodes['Image Texture']
node_principled = node_tree.nodes['Principled BSDF']

node_tree.links.new(
    node_image.outputs['Color'],
    node_principled.inputs['Base Color']
)
```

Cut a link between two sockets

```
node_tree.links.remove(link)

# This one is a little more tricky though.
# For sockets that take only one input:
node = node_tree.nodes['Principled BSDF']
link = node.inputs['Base Color'].links[0]
node_tree.links.remove(link)

# In the above example, .links[index] is always gonna
# take 0 because Base Color can have only one connection
#
# On the other hand, for output sockets that can be connected
# to several input sockets, you first need to find out the
# right link.
#
# E.g. Assuming a Texture Coordinate node is
# connected to a Mapping Node, an Image Texture and a
# Vector Math node and you want to cut the link to the
# Image Texture node, you would do something like:
node_tex_coords = node_tree.nodes['Texture Coordinate']

for link in node_tex_coords.outputs['UV'].links:
    if link.to_node.name == 'Image Texture':
        node_tree.links.remove(link)
```

🚀 Blender Python Cheatsheet

Check if a socket has any links

```
if node.inputs['socket_name'].links != ():
    # Do something

# Similarly for an output socket
if node.outputs['socket_name'].links != ():
    # Do something

# Instead of comparing with an empty tuple (),
# you can also simply do:
if node.outputs['socket_name'].links:
    # Do something
```

Get/Set value of an unconnected socket

```
# For input sockets
node.inputs['socket_name'].default_value = desired_value

# For output sockets
node.outputs['socket_name'].default_value = desired_value

# E.g. To set the roughness of a Principled BSDF node:
node = bpy.data.materials['Wood'].node_tree.nodes['Principled BSDF']
node.inputs['Roughness'].default_value = 0.75

# Similarly to set value of a color or vector socket, you can
# provide a tuple. E.g. To set Scale of a Mapping node
node.inputs['Scale'].default_value = 1, 2, 1  # x, y, z
```

Images

```
# Create a new image
bpy.data.images.new(name, width, height)

# For example:
bpy.data.images.new('table', 2048, 2048)

# Load an image from the disk
bpy.data.images.load('path/to/image')

# Save image to disk
image.save_render(filepath='path/to/save')
# The file extension in the path will determine
# saved image's format
```

Collections

```
# Create a new collection
bpy.data.collections.new(name)

# E.g. To create a collection 'MyCollection'
coll = bpy.data.collections.new("MyCollection")

# Link collection a scene
bpy.context.scene.collection.children.link(collection_obj)

# E.g. To link the collection 'coll' we created above
bpy.context.scene.collection.children.link(coll)

# NOTE: When you create a new collection, by default it won't
# show up in the scene collection and has to be linked mamnually
# as shown above.
#
# The same applied to objects created using bpy.data
# E.g. If you create an object 'MyObject' like ... :
obj_data = bpy.data.meshes.new('MyObjectMesh')
obj = bpy.data.objects.new('MyObject', obj_data)

#  ... you need to link it to the collection like:
bpy.context.scene.collection.objects.link(obj)
```

Files

Get current file's path

```
# Get current file's path
bpy.data.filepath
```

Save / Save As file

```
# If the file was already saved to disk (Save)
bpy.ops.wm.save_as_mainfile()

# If the file is not saved to disk yet (Save As)
bpy.ops.wm.save_as_mainfile(filepath)

# NOTE: This takes keyword only arguments. So you
# need to include 'filepath=' beofre the path:
bpy.ops.wm.save_as_mainfile(
    filepath="C:\\Users\\Blender User\\Desktop\\model.blend"
)

# This will throw an error
bpy.ops.wm.save_as_mainfile(
    "C:\\Users\\Blender User\\Desktop\\model.blend"
)
```

Pack all external files

```
# Pack all external files
bpy.ops.file.pack_all()
```

Cleanup file (purge orphan data)

```
# Cleanup file (purge orphan data)
bpy.ops.outliner.orphans_purge()
```

Import / Export

```
# Depends on the filetype to be imported
# For FBX, OBJ and GLTF you can:
bpy.ops.import_scene.fbx(filepath)
bpy.ops.import_scene.obj(filepath)
bpy.ops.import_scene.gltf(filepath)

# Similarly for export, same conventions apply
bpy.ops.export_scene.gltf(filepath)

# Every importer or exporter has it's own set of
# additional parameters that can be provided as
# optional keyword arguments. E.g. For GLTF:
bpy.ops.export_scene.gltf(
    filepath="C:\\Users\\BlenderUser\\Desktop\\model.gltf",
    export_format='GLTF_SEPARATE', export_animations=False
)  # GLTF + bin + textures instead of GLB and w/o animations

# Similarly mesh only formats like PLY and STL are
# inside .import_mesh or .export_mesh:
bpy.ops.import_mesh.ply(filepath)
bpy.ops.export_mesh.stl(filepath)

# While other formats like COLLADA and USD are in .wm:
bpy.ops.wm.collada_import(filepath)
bpy.ops.wm.usd_export(filepath)
```

Quit Blender

```
# Quit Blender
bpy.ops.wm.quit_blender()
```