



Blender - Python API

#13



Serdar ARITAN

Department of Computer Graphics
Hacettepe University, Ankara, Turkey



Blender/Python API Add On

An add-on is simply a Python module with some additional requirements so Blender can display it in a list with useful information.

Prerequisites

- Be familiar with the basics of working in Blender.
- Know how to run a script in Blender's Text editor.
- Have an understanding of Python primitive types.
- Be familiar with the concept of Python modules.
- Have a basic understanding of classes (object orientation) in Python.



Blender/Python API Add On

The simplest possible add-on

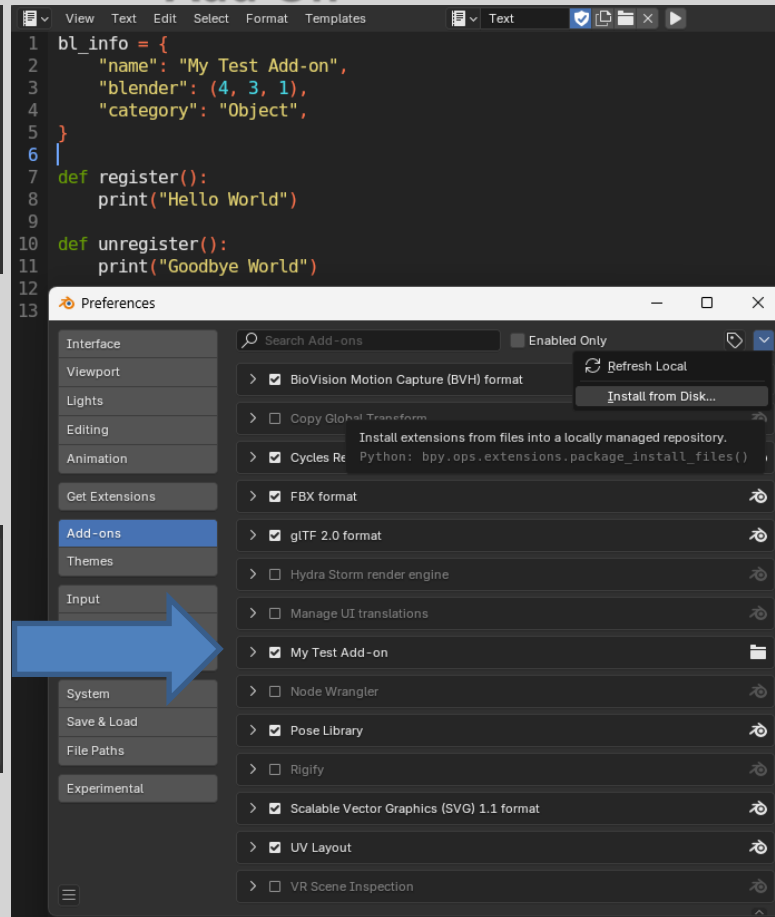
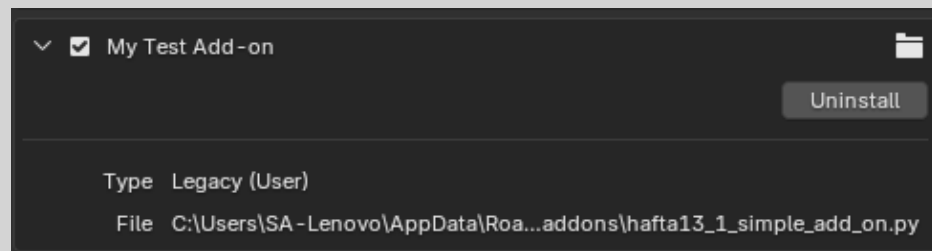
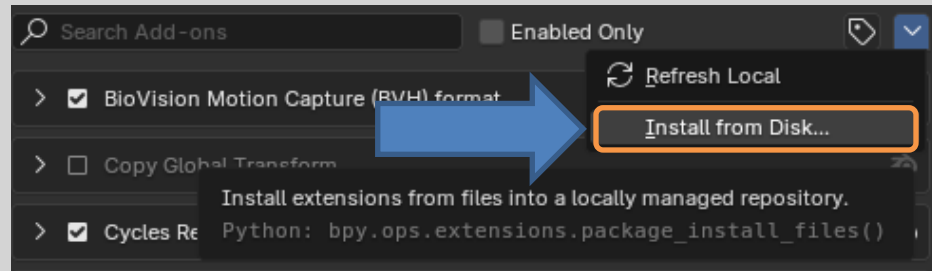
```
bl_info = {  
    "name": "My Test Add-on",  
    "blender": (4, 3, 1),  
    "category": "Object",  
}  
  
def register():  
    print("Hello World")  
  
def unregister():  
    print("Goodbye World")
```

bl_info is a dictionary containing add-on metadata such as the title, version and author to be displayed in the Preferences add-on list.

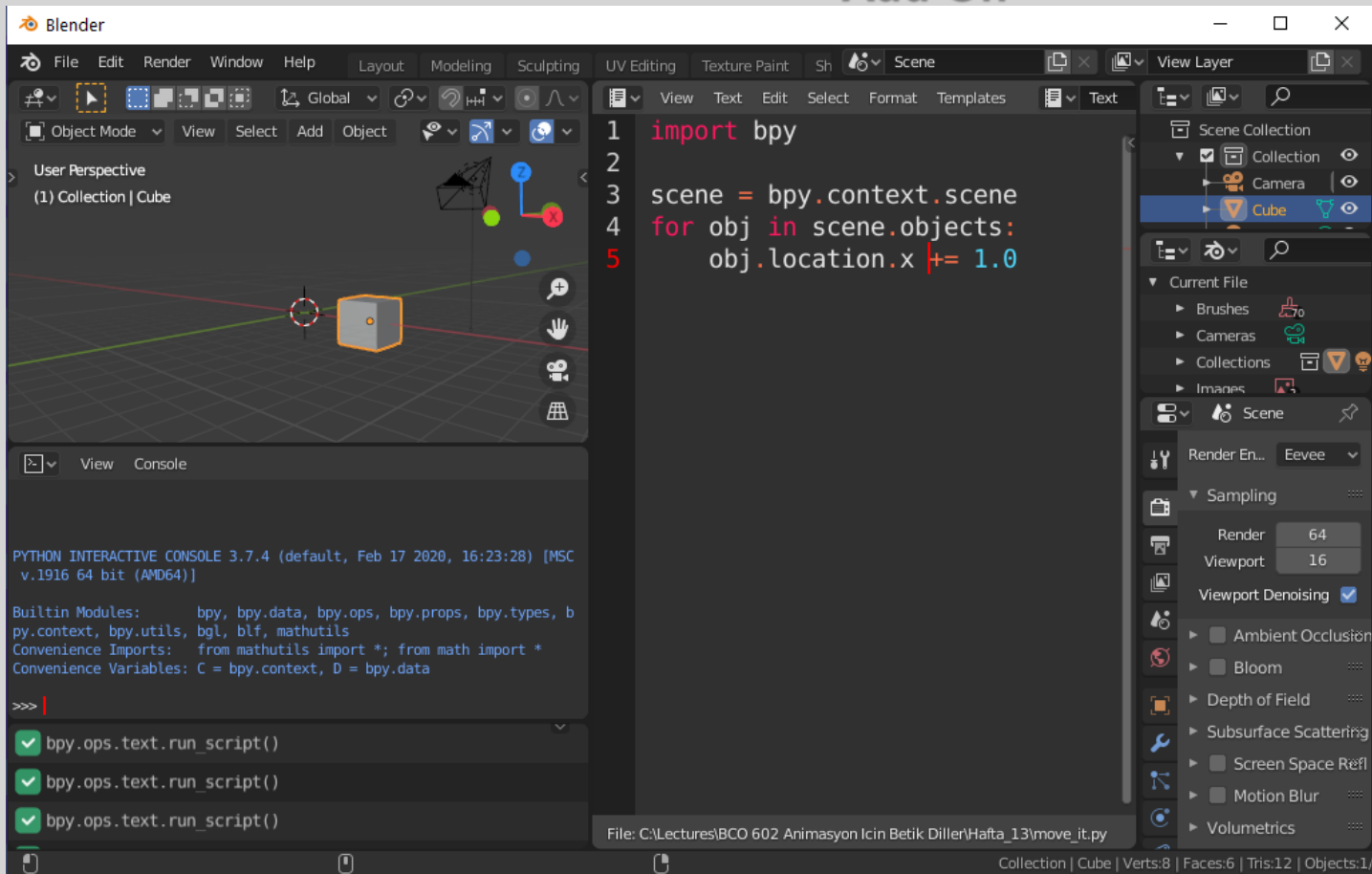
register is a function which only runs when enabling the add-on, this means the module can be loaded without activating the add-on.

unregister is a function to unload anything setup by register, this is called when the add-on is disabled.

Blender/Python API Add On



Blender/Python API Add On



The screenshot displays the Blender 2.80 interface. The central Text Editor shows a Python script that imports the bpy module and iterates through the objects in the current scene, increasing the x-coordinate of each object's location by 1.0.

```
1 import bpy
2
3 scene = bpy.context.scene
4 for obj in scene.objects:
5     obj.location.x += 1.0
```

The 3D Viewport on the left shows a cube in the center of a grid. The Properties panel on the right shows the 'Scene' collection selected, with the 'Cube' object highlighted. The 'Render' properties are visible, showing a resolution of 64x16 and 'Viewports Denoising' enabled.

The bottom status bar indicates the current state: Collection | Cube | Verts:8 | Faces:6 | Tris:12 | Objects:1/



Blender/Python API Add On

```
bl_info = {  
    "name": "Move X Axis",  
    "blender": (4, 3, 1),  
    "category": "Object",  
}  
  
import bpy  
  
class ObjectMoveX(bpy.types.Operator):  
    """My Object Moving Script""" # Use this as a tooltip for menu items and buttons.  
    bl_idname = "object.move_x" # Unique identifier for buttons and menu items  
    bl_label = "Move X by One" # Display name in the interface.  
    bl_options = {'REGISTER', 'UNDO'} # Enable undo for the operator.  
  
    def execute(self, context): # execute() is called when running the operator.  
        # The original script  
        scene = context.scene  
        for obj in scene.objects:  
            obj.location.x += 1.0  
  
    return {'FINISHED'} # Lets Blender know the operator finished successfully.
```



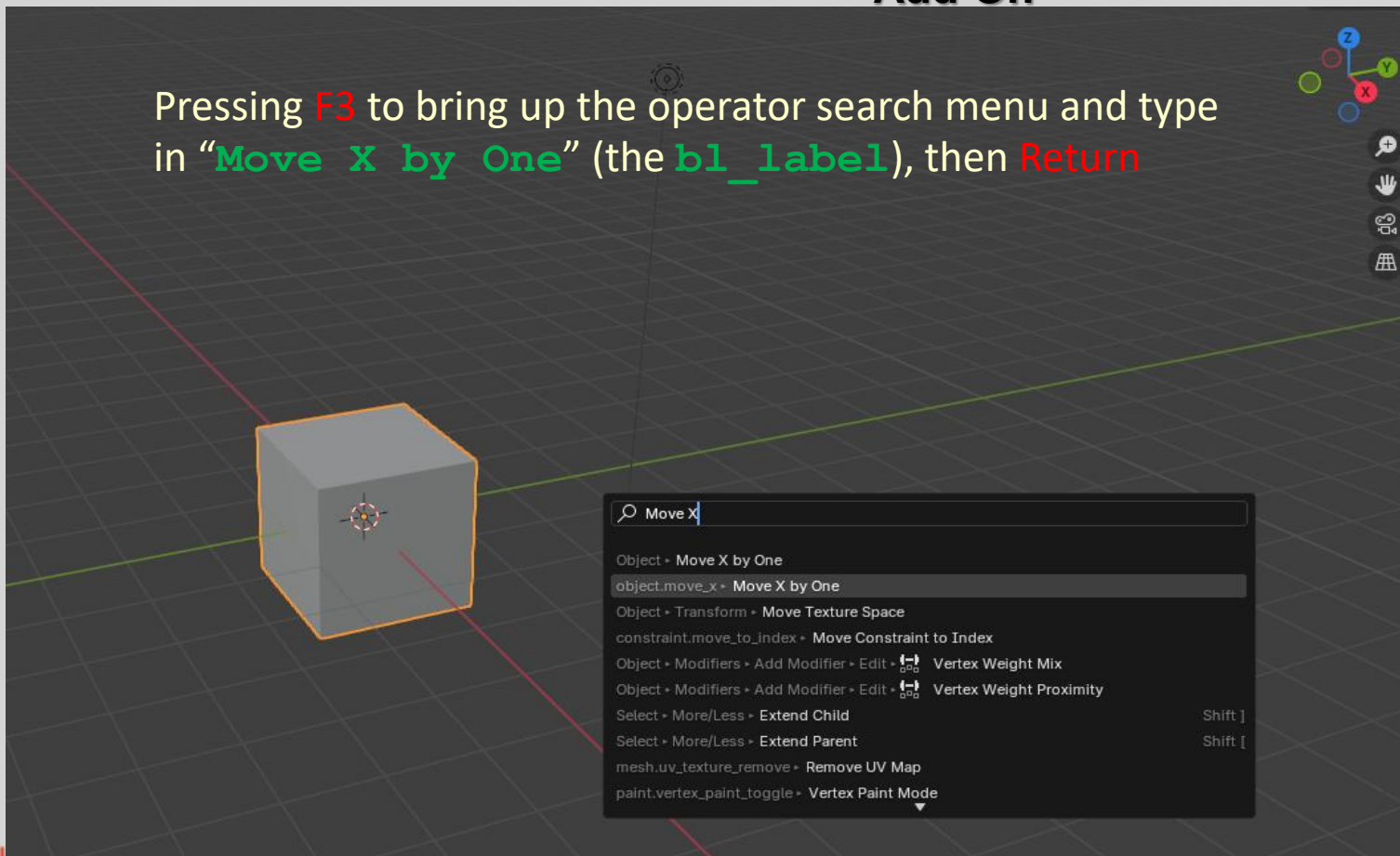
Blender/Python API Add On

```
def menu_func(self, context):  
    self.layout.operator(ObjectMoveX.bl_idname)  
  
def register():  
    bpy.utils.register_class(ObjectMoveX)  
    bpy.types.VIEW3D_MT_object.append(menu_func) # Adds the new operator to an existing menu.  
  
def unregister():  
    bpy.utils.unregister_class(ObjectMoveX)  
  
# This allows you to run the script directly from Blender's Text editor  
# to test the add-on without having to install it.  
if __name__ == "__main__":  
    register()
```

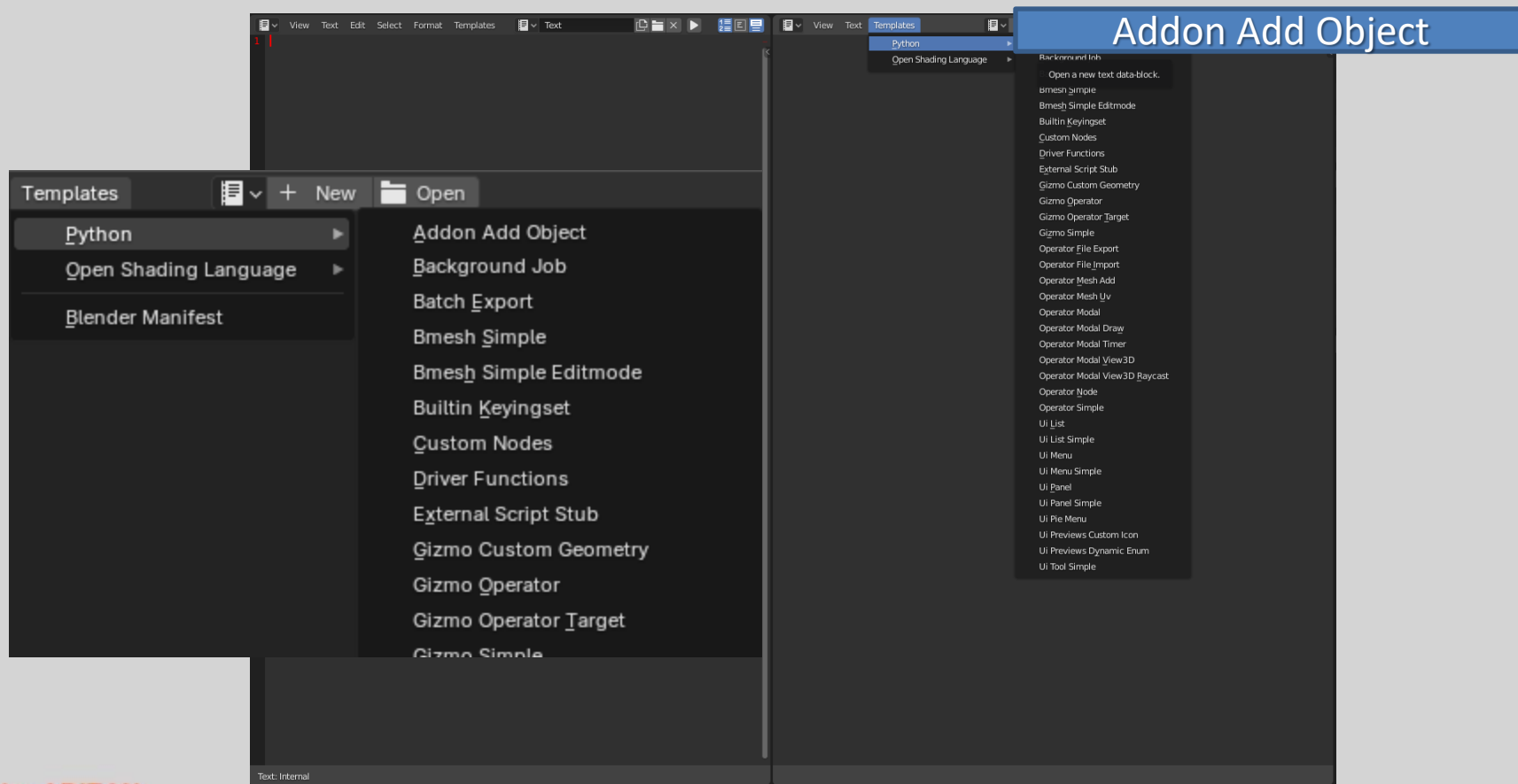



Blender/Python API Add On

Pressing **F3** to bring up the operator search menu and type in "**Move X by One**" (the **bl_label**), then **Return**



Blender/Python API Add On





Blender/Python API Add On

```
View Text Edit Select Format Templates addon_add_object.py
1 # To make this add-on installable, create an extension with it:
2 # https://docs.blender.org/manual/en/latest/advanced/extensions/getting\_started.html
3
4 import bpy
```

How to Create Extensions

Creating an extension takes only a few steps:

1. Open the directory containing the add-on code or theme file.
2. Add a [blender_manifest.toml](#) file with all the required meta-data (name, maintainer, ...).
3. Use the [Blender command-line tool](#) to build the extension .zip file.

How to publish to the [Blender Extensions Platform](#):

- [Install from Disk](#) to test if everything is working well.
- [Upload the .zip file](#) (this step requires Blender ID).

The extension will be held for [review](#), and published once the moderation team approves it.



Blender/Python API Add On

Extension files

An extension is shared as a `.zip` archive containing a manifest file and other files. The expected files depend on the extension type.

Add-on extension

Add-ons need at least the manifest and an `__init__.py` file, while more complex add-ons have a few different `.py` files or wheels together.

```
my_extension-0.0.1.zip
├─ __init__.py
├─ blender_manifest.toml
└─ (...)
```

Theme extension

A theme extension only needs the manifest and the `.xml` theme file.

```
my_extension-0.0.1.zip
├─ blender_manifest.toml
└─ theme.xml
```

Note

Extensions can optionally have all its files inside a folder (inside the archive). This is a common behavior when saving a repository as ZIP from version-control platforms.



Blender/Python API Add On

blender_manifest.toml

```
1 schema_version = "1.0.0"
2
3 # Example of manifest file for a Blender extension
4 # Change the values according to your extension
5 id = "BC0602_example_extension"
6 version = "1.0.0"
7 name = "My Example Extension"
8 tagline = "This is another extension"
9 maintainer = "Serdar ARITAN <serdar.aritan@hacettepe.edu.tr>"
10 # Supported types: "add-on", "theme"
11 type = "add-on"
12
13 # # Optional: link to documentation, support, source files, etc
14 # website = "https://extensions.blender.org/add-ons/my-example-package/"
15
16 # # Optional: tag list defined by Blender and server, see:
17 # # https://docs.blender.org/manual/en/dev/advanced/extensions/tags.html
18 # tags = ["Animation", "Sequencer"]
19
20 blender_version_min = "4.2.0"
21 # # Optional: Blender version that the extension does not support, earlier versions are supported.
22 # # This can be omitted and defined later on the extensions platform if an issue is found.
23 # blender_version_max = "5.1.0"
24
25 # License conforming to https://spdx.org/licenses/ (use "SPDX: prefix)
26 # https://docs.blender.org/manual/en/dev/advanced/extensions/licenses.html
27 license = [
28     "SPDX:GPL-3.0-or-later",
29 ]
30 |
```



Blender/Python API Add On

Required values:

| | |
|-----------------------------|--|
| blender_version_min: | Minimum supported Blender version - use at least <code>4.2.0</code> . |
| id: | Unique identifier for the extension. |
| license: | List of licenses , use SPDX license identifier . |
| maintainer: | Maintainer of the extension. |
| name: | Complete name of the extension. |
| schema_version: | Internal version of the file format - use <code>1.0.0</code> . |
| tagline: | One-line short description, up to 64 characters - cannot end with punctuation. |
| type: | "add-on", "theme". |
| version: | Version of the extension - must follow semantic versioning . |



Blender/Python API Add On

| | |
|-----------------------------|---|
| blender_version_max: | Blender version that the extension does not support, earlier versions are supported. |
| website: | Website for the extension. |
| copyright: | Some licenses require a copyright, copyrights must be "Year Name" or "Year-Year Name". |
| tags: | List of tags. See the list of available tags . |
| platforms: | List of supported platforms. If omitted, the extension will be available in all operating systems. The available options are ["windows-x64", "windows-arm64", "macos-x64", "macos-arm64", "linux-x64"] |
| wheels: | List of relative file-paths Python Wheels . |
| permissions: | Add-ons can list which resources they require. The available options are <i>files</i> , <i>network</i> , <i>clipboard</i> , <i>camera</i> , <i>microphone</i> . Each permission should be followed by an explanation (short single-sentence, up to 64 characters, with no end punctuation). |



Blender/Python API Add Ons

3D View
Add Curve
Add Mesh
Animation
Bake
Camera
Compositing
Development
Game Engine
Geometry Nodes
Grease Pencil
Import-Export
Lighting
Material
Modeling

Mesh
Node
Object
Paint
Pipeline
Physics
Render
Rigging
Scene
Sculpt
Sequencer
System
Text Editor
Tracking
User Interface
UV



Blender/Python API

Themes

Dark

Light

Colorful

Inspired By

Print

Accessibility

High Contrast



Blender/Python API Add On

Add-on extension

Add-ons need at least the manifest and an `__init__.py` file, while more complex add-ons have a few different .py files or wheels together.

```
my_extension-0.0.1.zip
├─ __init__.py
├─ blender_manifest.toml
└─ (...)
```

Theme extension

A theme extension only needs the manifest and the .xml theme file.

```
my_extension-0.0.1.zip
├─ blender_manifest.toml
└─ theme.xml
```



Blender/Python API Add On

```
1 bl_info = {
2     "name": "myButtons",
3     "author": "Serdar ARITAN",
4     "version": (1, 0),
5     "blender": (2, 83, 10),
6     "location": "View3D",
7     "description": "Custom AddOn",
8     "warning": "",
9     "doc_url": "",
10    "category": "",
11 }
12
13 import bpy
14
```

```
1 bl_info = {
2     "name": "New",
3     "author": "Your Name Here",
4     "version": (1, 0),
5     "blender": (2, 80, 0),
6     "location": "View3D > Add > Mesh",
7     "description": "Adds a new Mesh",
8     "warning": "",
9     "doc_url": "",
10    "category": "Add Mesh",
11 }
12
13
14 import bpy
15 from bpy.types import Operator
16 from bpy.props import FloatProperty
17 from bpy_extras.object_utils import object_data_add
18 from mathutils import Vector
19
20
21 def add_object(self, context):
22     scale_x = self.scale.x
23     scale_y = self.scale.y
24
25     verts = [
26         Vector((-1 * scale_x, 1 * scale_y, 0)),
27         Vector((1 * scale_x, 1 * scale_y, 0)),
28         Vector((1 * scale_x, -1 * scale_y, 0)),
29         Vector((-1 * scale_x, -1 * scale_y, 0)),
30     ]
31
32     edges = []
33     faces = [[0, 1, 2, 3]]
34
```

Operator Node



Blender/Python API Add On

```
1 import bpy
2
3 class NodeOperator(bpy.types.Operator):
4     """Tooltip"""
5     bl_idname = "node.simple_o"
6     bl_label = "Simple Node Operator"
7
8     @classmethod
9     def poll(cls, context):
10         space = context.space_data
11         return space.type == 'NODE_EDITOR'
12
13     def execute(self, context):
14         main(self, context)
15         return {'FINISHED'}
16
31 # add a link between the two nodes
32 node_link = node_tree.links.new(socket_in, socket_out)
33
34 class NodeOperator(bpy.types.Operator):
35     """Tooltip"""
36     bl_idname = "node.simple_operator"
37     bl_label = "Simple Node Operator"
38
39     @classmethod
40     def poll(cls, context):
41         space = context.space_data
42         return space.type == 'NODE_EDITOR'
43
44     def execute(self, context):
45         main(self, context)
46         return {'FINISHED'}
47
48
49
50 def menu_func(self, context):
51     self.layout.operator(NodeOperator.bl_idname, text=NodeOp
52
53
54 # Register and add to the "Node" menu (required to also use
55 def register():
56     bpy.utils.register_class(NodeOperator)
57     bpy.types.NODE_MT_node.append(menu_func)
58
59
60 def unregister():
61     bpy.utils.unregister_class(NodeOperator)
62     bpy.types.NODE_MT_node.remove(menu_func)
63
64
65 if __name__ == "__main__":
66     register()
67
```



Blender/Python API Add On

```
1 NodeOperator => ButtonOperator
2
3 class ButtonOperator(bpy.types.Operator):
4     """Tooltip"""
5     bl_idname = "my_obect.1"
6     bl_label = "Simple Button Operator"
7     bl_options = {'REGISTER', 'UNDO'}
8
9
10 def execute(self, context):
11     bpy.ops.transform.translate(value=(0, 0, 1), o
12
13     return {'FINISHED'}
14
22 return
23
24 if not node_other.outputs:
25     operator.report({'ERROR'}, "Selected node has no ou
26     return
27
28 socket_in = node_active.inputs[0]
29 socket_out = node_other.outputs[0]
30
31 # add a link between the two nodes
32 node_link = node_tree.links.new(socket_in, socket_out)
33
34
35 class NodeOperator(bpy.types.Operator):
36     """Tooltip"""
37     bl_idname = "node.simple_operator"
38     bl_label = "Simple Node Operator"
39
40 @classmethod
41 def poll(cls, context):
42     space = context.space_data
43     return space.type == 'NODE_EDITOR'
44
45 def execute(self, context):
46     main(self, context)
47     return {'FINISHED'}
48
```

Blender/Python API Add On



Ctrl-C



Blender/Python API Add On

```
1 bl_info = {
2     "name": "myButtons",
3     "author": "Serdar ARITAN",
4     "version": (1, 0),
5     "blender": (2, 83, 10),
6     "location": "View3D",
7     "description": "Custom AddOn",
8     "warning": "",
9     "doc_url": "",
10    "category": "",
11 }
12
13 import bpy
14
15 class ButtonOperator(bpy.types.Operator):
16     bl_idname = "my_object.1"
17     bl_label = ""
18     bl_options = {"REGISTER", "UNDO"}
19
20     def execute(self, context):
21         # Write here what we want to do
22         bpy.ops.transform.translate(value=(0, 0, 1), orient_type='LOCAL')
23
24         return {'FINISHED'}
```

`bpy.ops.transform.translate(`
`value=(0, 0, 1),`
`orient_type='LOCAL'`
`)`

```
13 if len(node_selected) != 1:
14     operator.report({'ERROR'}, "2 nodes must be s
15     return
16
17     node_other, = node_selected
18
19     # now we have 2 nodes to operate on
20     if not node_active.inputs:
21         operator.report({'ERROR'}, "Active node has n
22         return
23
24     if not node_other.outputs:
25         operator.report({'ERROR'}, "Selected node has
26         return
27
28     socket_in = node_active.inputs[0]
29     socket_out = node_other.outputs[0]
30
31     # add a link between the two nodes
32     node_link = node_tree.links.new(socket_in, socket
33
34
35 class NodeOperator(bpy.types.Operator):
36     """Tooltip"""
37     bl_idname = "node.simple_operator"
38     bl_label = "Simple Node Operator"
39
40     @classmethod
41     def poll(cls, context):
42         space = context.space_data
43         return space.type == 'NODE_EDITOR'
44
45     def execute(self, context):
46         main(self, context)
```




Blender/Python API Add On

```
1 import bpy
2
3 class ButtonOperator(bpy.types.Operator):
4     """Tooltip"""
5     bl_idname = "my_obect.1"
6     bl_label = "Simple Button Operator"
7     bl_options = {'REGISTER', 'UNDO'}
8
9
10 def execute(self, context):
11     bpy.ops.transform.translate(value=(0, 0, 1), o
12
13     return {'FINISHED'}
14
```

```
1 import bpy
2
3
4 class HelloWorldPanel(bpy.types.Panel):
5     """Creates a Panel in the Object prop
6     bl_label = "Hello World Panel"
7     bl_idname = "OBJECT_PT_hello"
8     bl_space_type = 'PROPERTIES'
9     bl_region_type = 'WINDOW'
10     bl_context = "object"
11
12 def draw(self, context):
13     layout = self.layout
14
15     obj = context.object
16
17     row = layout.row()
18     row.label(text="Hello world!", ic
19
20     row = layout.row()
21     row.label(text="Active object is:
22     row = layout.row()
23     row.prop(obj, "name")
24
25     row = layout.row()
26     row.operator("mesh.primitive_cube
27
28 def register():
29
30
31 def unregister():
32
33
```

Python

Open Shading Language

Blender Manifest

Addon Add Object

Background Job

Batch Export

Bmesh Simple

Bmesh Simple Editmode

Builtin Keyingset

Custom Nodes

Driver Functions

External Script Stub

Gizmo Custom Geometry

Gizmo Operator

Gizmo Operator Target

Gizmo Simple

Image Processing

Operator File Export

Operator File Import

Operator Mesh Add

Operator Mesh UV

Operator Modal

Operator Modal Draw

Operator Modal Timer

Operator Modal View3D

Operator Modal View3D Raycast

Operator Node

Operator Simple

UI Asset Shelf

UI List

UI List Generic

UI List Simple

UI Menu

UI Menu Simple

UI Panel

UI Panel Simple

UI Pie Menu

Scene Collection

Collection

Camera

Cube

Light

Current File

Cameras

Collections

Images

Lights

Line Styles

Materials

Meshe

Objects

Scene

Render En... EEVEE

Sampling

Viewport

Samples 16

Temporal Re...

Jittered Sha...

Render

Samples 64

Open: UI Panel Simple

Open a new text data-block.

Python: bpy.ops.text.open(filepath="C:\\Program Files\\Blender Foundation\\Blender 4.3\\4.3\\scripts\\templates_py\\ui_panel_simple.py", internal=True)



Blender/Python API Add On

```
1 import bpy
2
3 class ButtonOperator(bpy.types.Operator):
4     """Tooltip"""
5     bl_idname = "my_obect.1"
6     bl_label = "Simple Button Operator"
7     bl_options = {'REGISTER', 'UNDO'}
8
9
10 def execute(self, context):
11     bpy.ops.transform.translate(value=(0, 0, 1))
12
13     return {'FINISHED'}
14 |
```

```
1 import bpy
2
3 class HelloWorldPanel(bpy.types.Panel):
4     """Creates a Panel in the Object properties window"""
5     bl_label = "Hello World Panel"
6     bl_idname = "OBJECT_PT_hello"
7     bl_space_type = 'PROPERTIES'
8     bl_region_type = 'WINDOW'
9     bl_context = "object"
10
11 def draw(self, context):
12     layout = self.layout
13
14     obj = context.object
15
16     row = layout.row()
17     row.label(text="Hello world!", icon='WORLD_DATA')
18
19     row = layout.row()
20     row.label(text="Active object is: " + obj.name)
21     row = layout.row()
22     row.prop(obj, "name")
23
24     row = layout.row()
25     row.operator("mesh.primitive_cube_add")
26
27
```



Blender/Python API Add On

```
1 import bpy
2
3 class ButtonOperator(bpy.types.Operator):
4     """Tooltip"""
5     bl_idname = "my_obect.1"
6     bl_label = "Simple Button Operator"
7     bl_options = {'REGISTER', 'UNDO'}
8
9
10     def execute(self, context):
11         bpy.ops.transform.translate(value=(0, 0, 1), orient_type='LOCAL')
12
13         return {'FINISHED'}
14
15 class CustomPanel(bpy.types.Panel):
16     """Creates a Panel in the Object properties window"""
17     bl_label = "Hello World Panel"
18     bl_idname = "OBJECT_PT_Custom1"
19     bl_space_type = 'VIEW_3D'
20     bl_region_type = 'TOOLS'
21
22     def draw(self, context):
23         layout = self.layout
24
25         obj = context.object
26
27         row = layout.row()
28         row.operator(ButtonOperator.bl_idname,
29                     text = ButtonOperator.bl_label,
30                     icon = "TRIA_UP")
```

HelloWorldPanel => CustomPanel

icon = "TRIA_UP" ?



Blender/Python API Add On

Preferences

Interface

Viewport

Lights

Editing

Animation

Get Extensions

Add-ons

Themes

Input

Navigation

Keymap

System

Save & Load

File Paths

Experimental

icon

Installed

> Icon Viewer

Type "icon"
Check "icon Viewer"

File Edit Render Window Help Layout Modeling Sculpting UV Editing

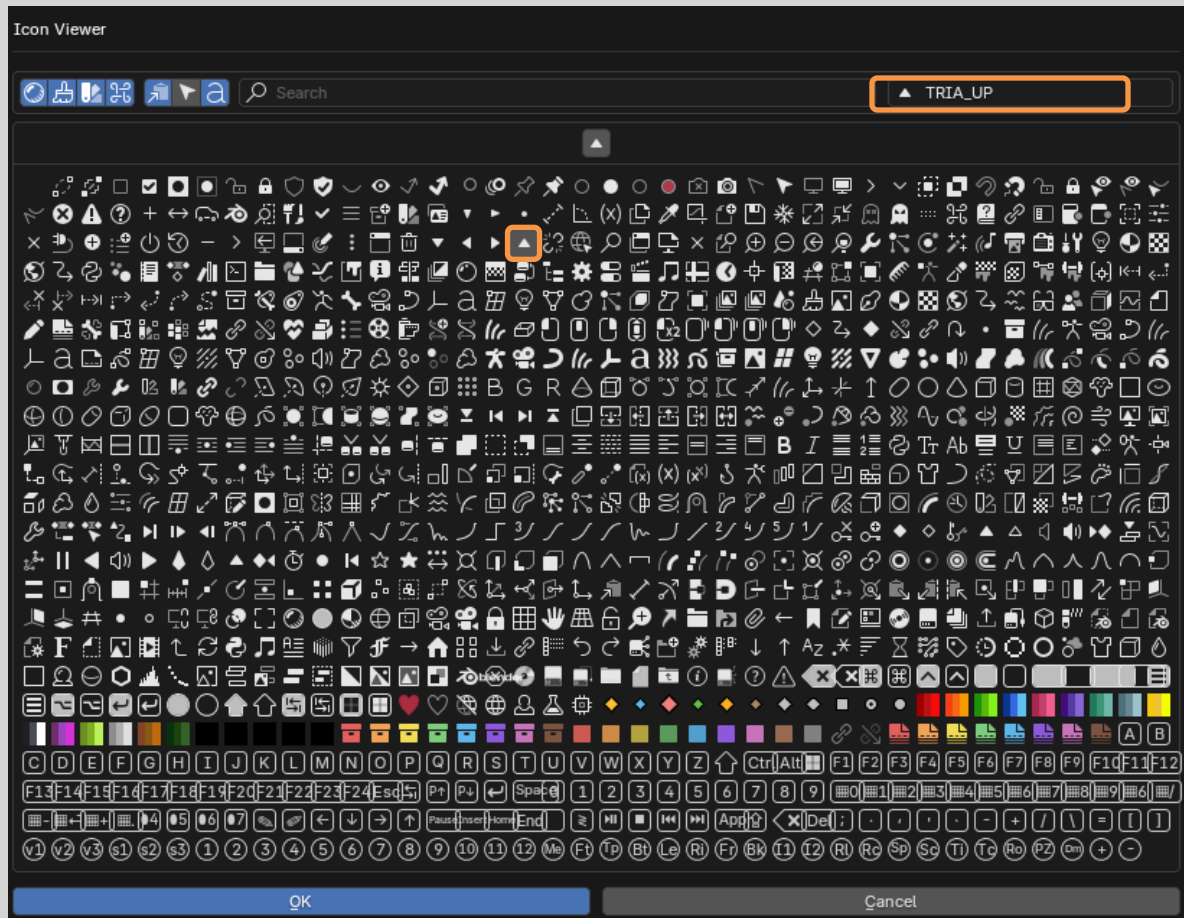
Object Mode View Select Add Object Global Options

User Perspective
(1) Collection | Cube

View Console Icon Viewer

click

Blender/Python API Add On





Register() and Unregister()

There are two functions, `register()` and `unregister()`, that are required in add-ons. These two functions are responsible for calling `bpy.utils.register_class()`, `bpy.utils.unregister_class()`, `bpy.utils.register_module()`, and `bpy.utils.unregister_module()`. Any class that inherits a `bpy.type` class needs to be registered for it to be used by Blender in the add-on. Blender uses the `unregister()` function when an add-on is switched off by the user in the user preferences.

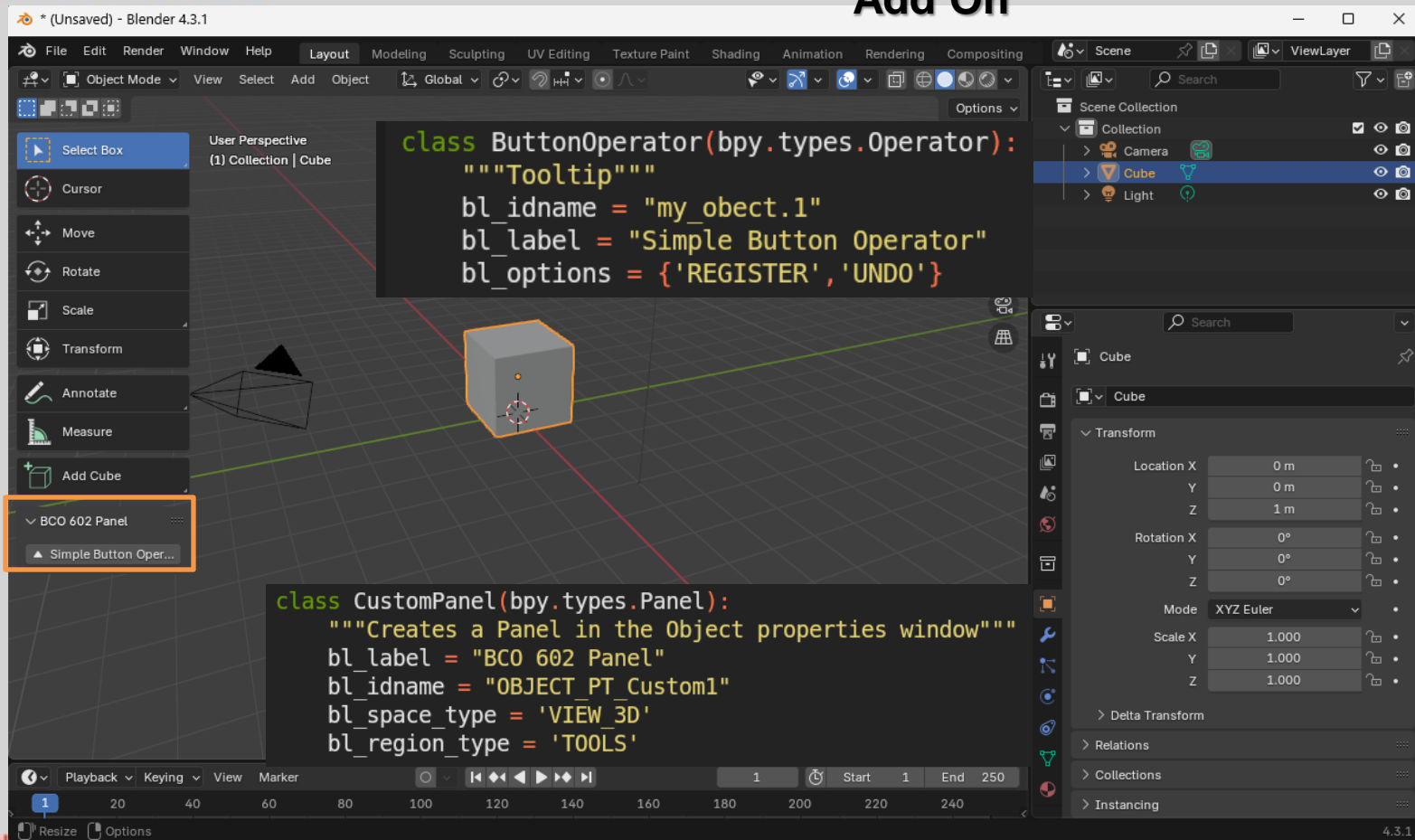


Blender/Python API Add On

```
31
32     def draw(self, context):
33         layout = self.layout
34
35         obj = context.object
36
37         row = layout.row()
38         row.operator(ButtonOperator.bl_idname,
39                     text = ButtonOperator.bl_label,
40                     icon = "TRIA_UP")
41
42     from bpy.utils import register_class, unregister_class
43
44     # Classes in the addon:
45     _classes = [
46         ButtonOperator,
47         CustomPanel
48     ]
49
50     # Registering a class loads the class definition into Blender, where it then becomes available
51     def register():
52         for cls in _classes:
53             register_class(cls)
54
55     def unregister():
56         for cls in _classes:
57             unregister_class(cls)
58
59     # The last 2 lines are only for testing:
60     if __name__ == "__main__":
61         register()
62
```

File: *C:\Lectures\BCO 602 Animasyon Icin Betik Dilleri\Hafta_13\simpleButtons.py (unsaved)

Blender/Python API Add On



The image shows the Blender 4.3.1 interface with a custom Python add-on installed. The add-on consists of two parts: a custom operator and a custom panel.

Custom Operator:

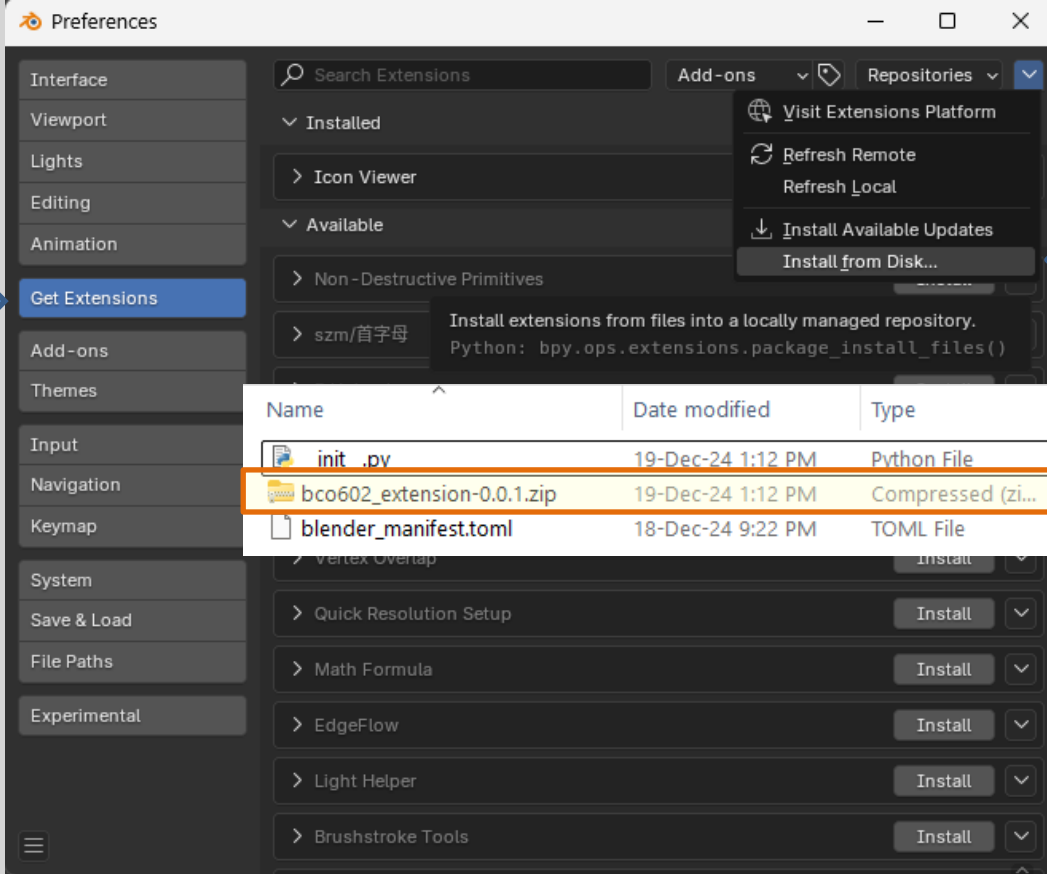
```
class ButtonOperator(bpy.types.Operator):  
    """Tooltip"""  
    bl_idname = "my_obect.1"  
    bl_label = "Simple Button Operator"  
    bl_options = {'REGISTER', 'UNDO'}  
    def execute(self, context):  
        pass
```

Custom Panel:




```
class CustomPanel(bpy.types.Panel):  
    """Creates a Panel in the Object properties window"""  
    bl_label = "BCO 602 Panel"  
    bl_idname = "OBJECT_PT_Custom1"  
    bl_space_type = 'VIEW_3D'  
    bl_region_type = 'TOOLS'  
    def draw(self, context):  
        pass
```

The interface shows the 3D Viewport with a cube selected. The Properties panel on the right shows the 'Cube' object's transform properties. The Outliner on the right shows the 'Collection' containing the 'Cube' and 'Light' objects. The Timeline at the bottom shows the current frame is 1.

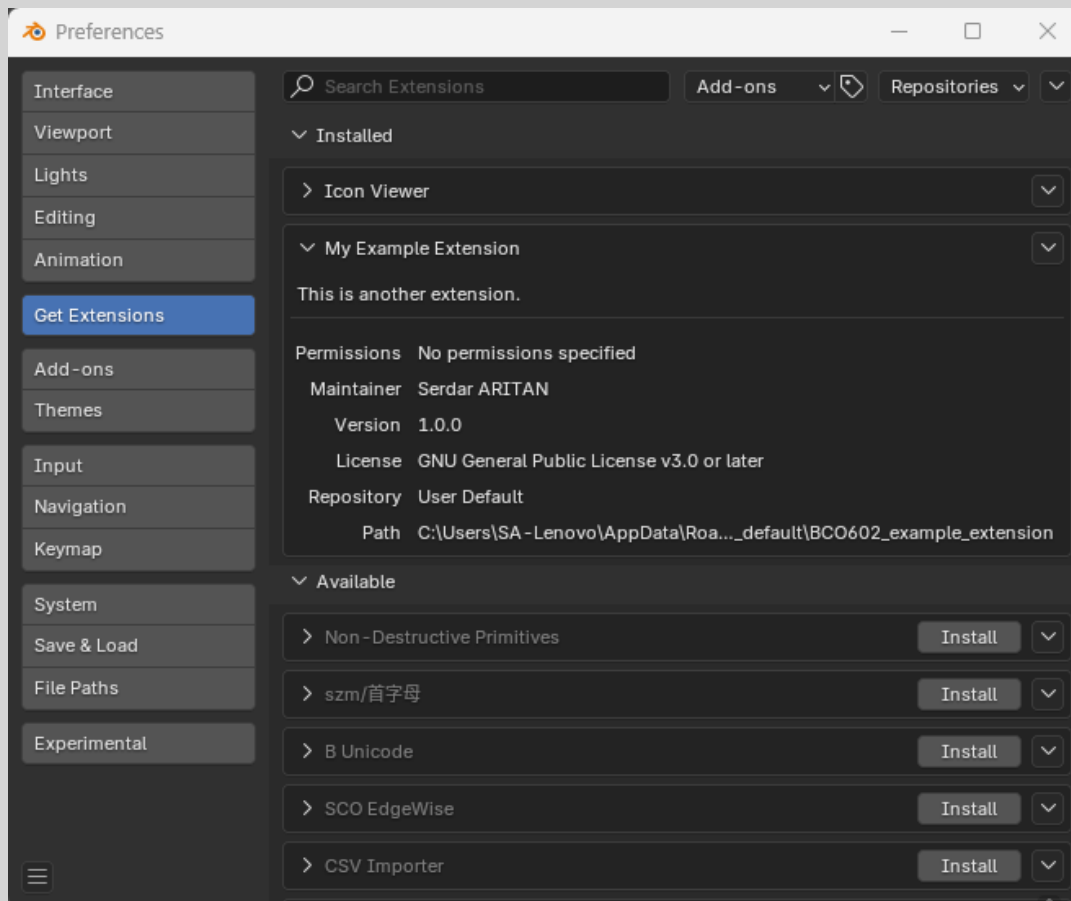
Blender/Python API Add On



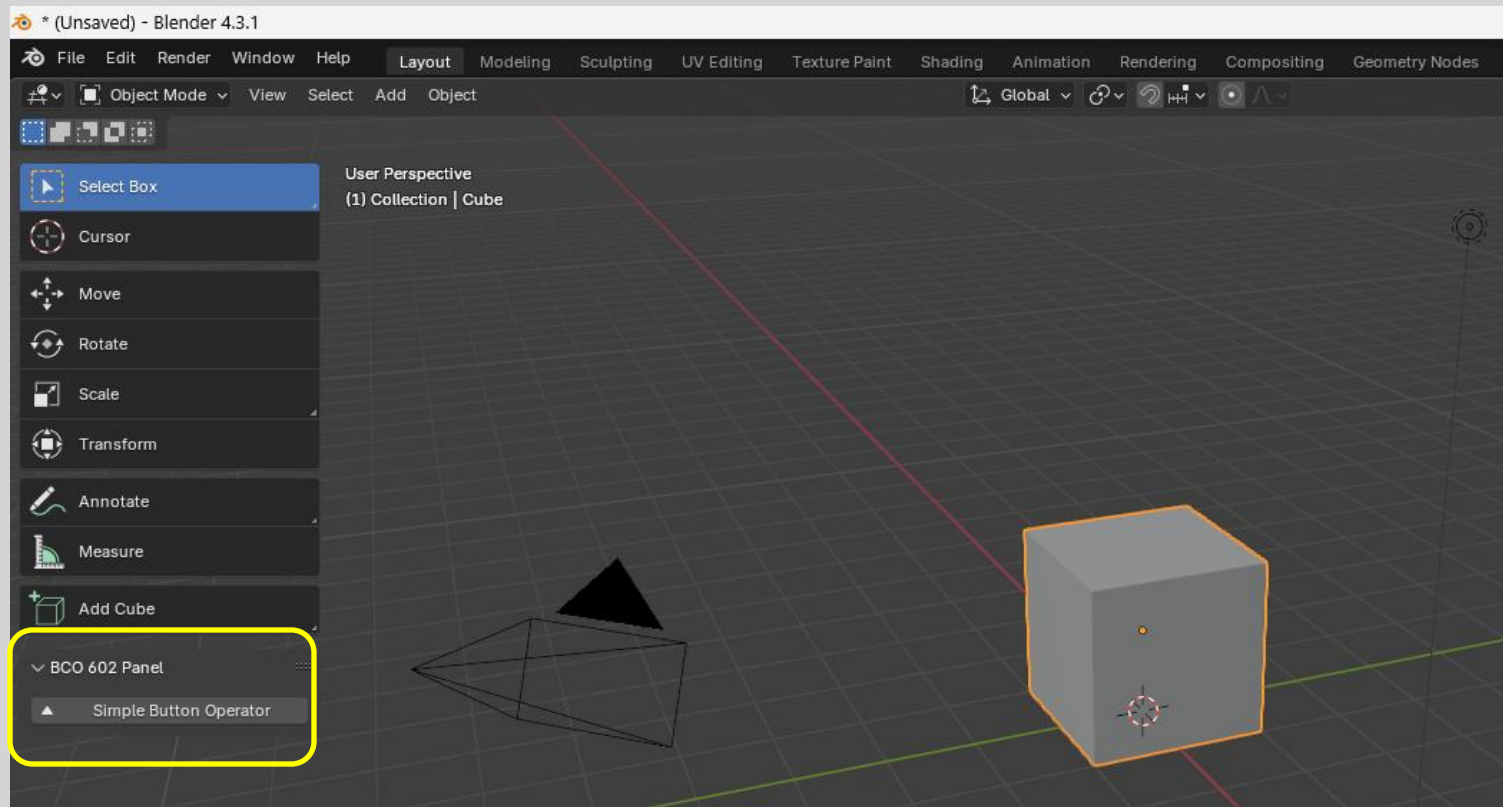
The image shows the Blender Preferences window, specifically the Add-ons panel. A blue arrow labeled '1' points to the 'Get Extensions' button in the left sidebar. Another blue arrow labeled '2' points to the 'Install from Disk...' button in the 'Available' section. A third blue arrow points to the 'bco602_extension-0.0.1.zip' file in the list of available extensions.

| Name | Date modified | Type | Size |
|--|-------------------|-------------------|------|
|  init .py | 19-Dec-24 1:12 PM | Python File | 2 KB |
|  bco602_extension-0.0.1.zip | 19-Dec-24 1:12 PM | Compressed (zi... | 2 KB |
|  blender_manifest.toml | 18-Dec-24 9:22 PM | TOML File | 2 KB |

Blender/Python API Add On

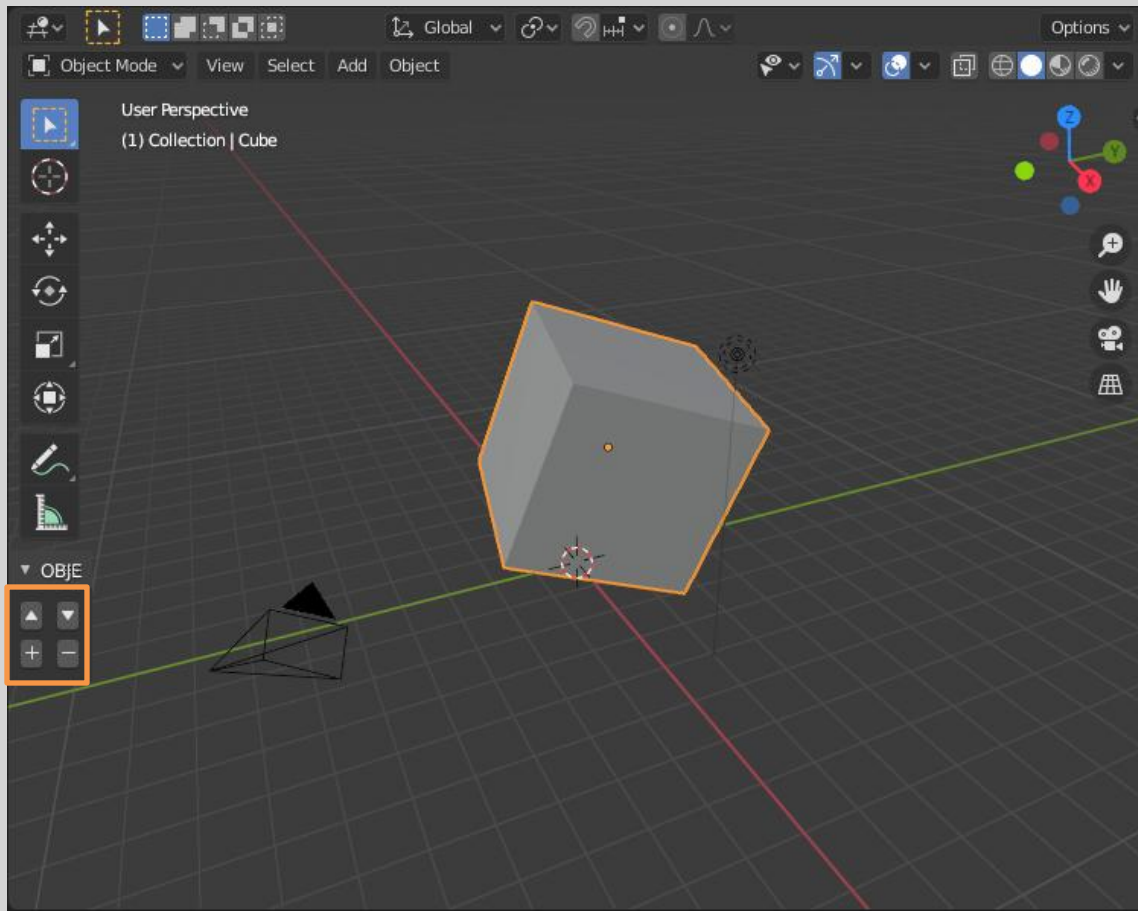


Blender/Python API Add On





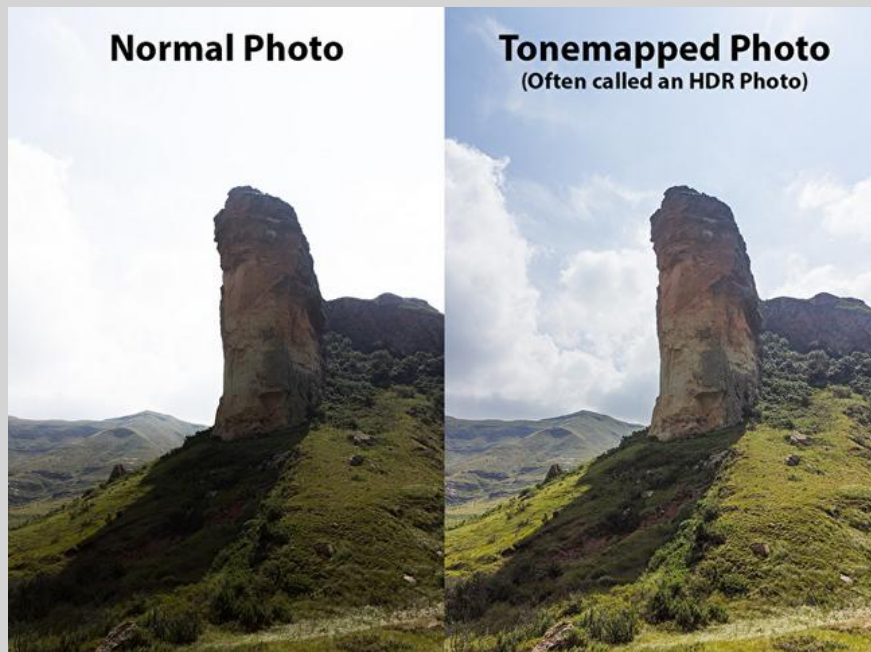
Blender/Python API Add On





HDR or HDRI

HDR means high dynamic range. An HDRI is simply an image with high dynamic range properties. So, when you hear about HDRI and HDR in digital photo editing, it's often referring to the same thing.



HDR or HDRI



HDR 1



HDR 2



HDR 3



HDR 4



HDR 5

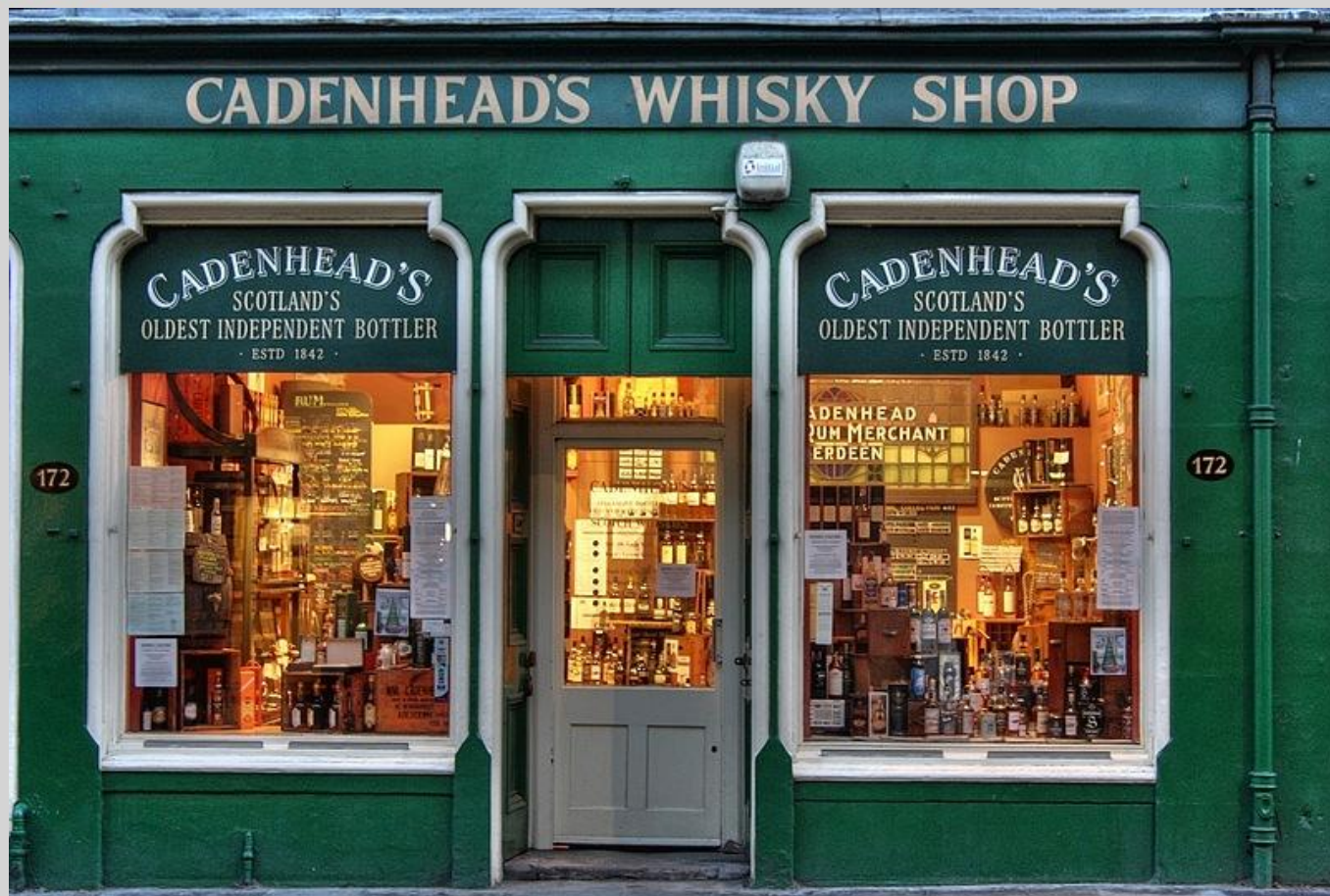


FINAL

HDR or HDRI



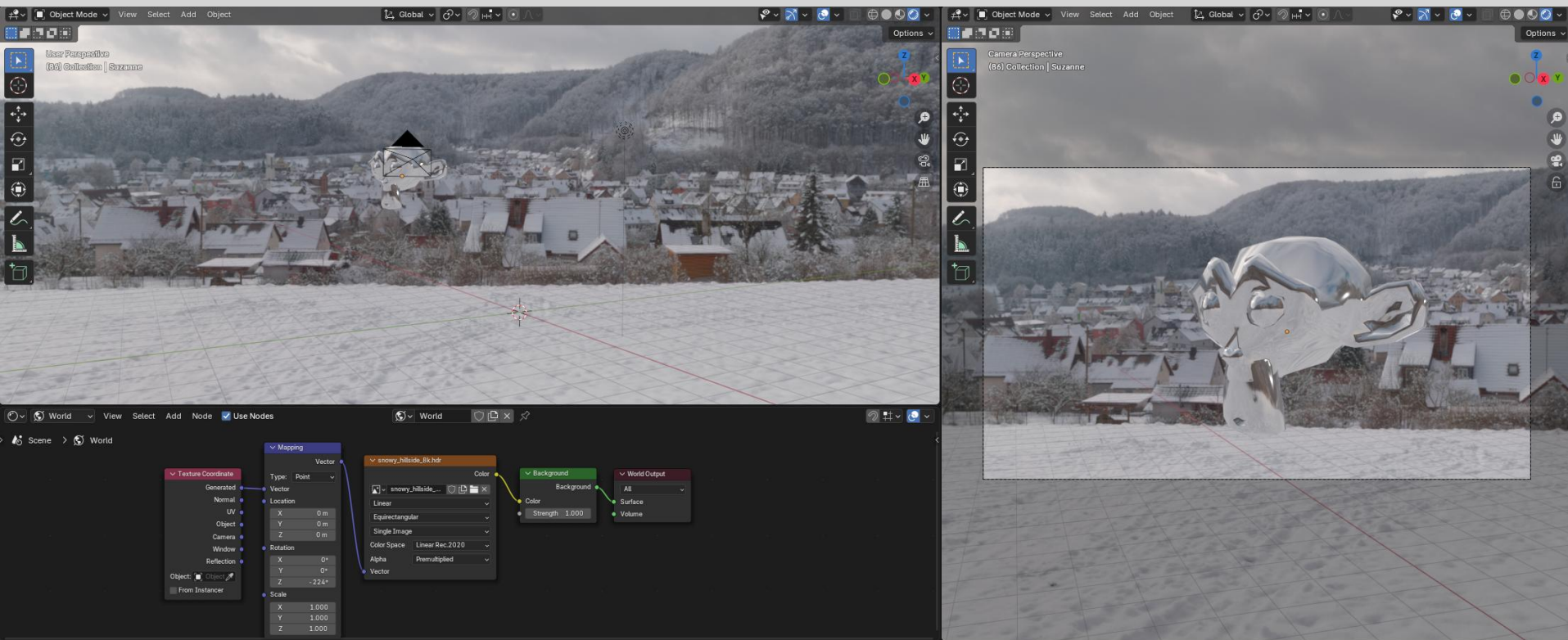






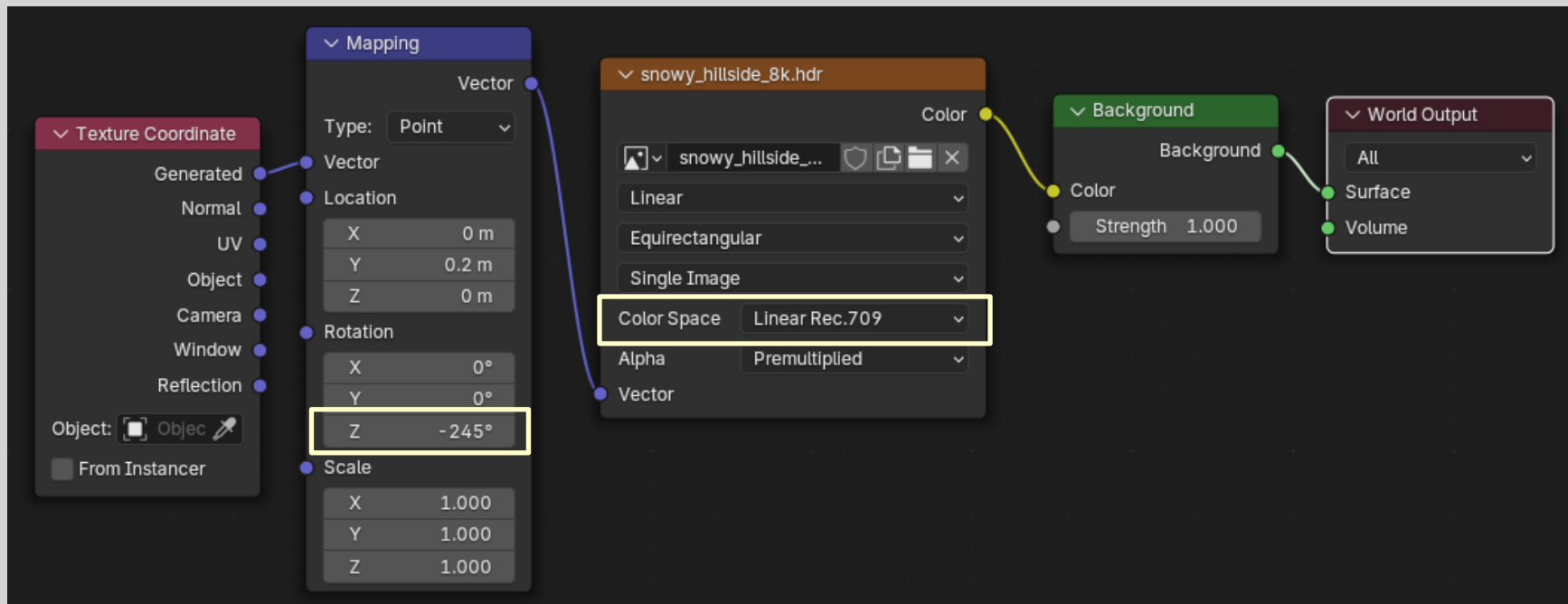








HDR or HDRI



```
bpy.data.worlds["World"].node_tree.nodes["Mapping"].inputs[2].default_value[2] = -3.90779
```