Code / Software logic Explanation

Developed / Wrote by: Kevin Yang



## Preface

I must commit this is a fairly challenging project I set to myself.

With all the struggles and researches, I finally reach to a stage where this Blender add-on program is able to be released to public. To express the new knowledge and my thoughts during the development, I keen to write this as a non-academic, an informal piece of document.

In this document, I will show you the developments behind the work, as well as explaining the program logic and technical side of the program in depth.

For the ones having a programming background, this writing might seem naïve.

But yet,

I see it in similar to a class notes, **it is a documentation of my studies.**

## Development Summery / Log

Since the pandemic, the computer became pretty much my only tool of creation, whether modelling, 3D Printing, simulations or programming.

I started the briefing, or began to think about this project a few weeks before the Easter holidays –

I believe it was right after I did my Tuesday crits presentation.

During the time, I had just finished a piece of code-generated work via C# scripts. It was to generate frame shots of water simulation based on given language inputs, creating a link between language and a visual form object. For this project, there was just a script, nothing else. It runs on debugging console within the IDE, without any user interface, nor it can be directly viewed by the general audience.

In the crits, one of the tutors, Susan, mentioned all the faulty above. She reminds me of the importance of the audience factor. Especially came down to program / code-based art. I was not satisfied to stop express the linkage between language to visual form.

With this in mind, I gradually came down to the field of QR codes – one of the ingénues creation that can translate various data (languages, binaries, kanji…) into a visualized picture. It is the exact thinking I was looking for, yet not realized it is so close to my daily life.

As a sculptor student, at least before the pandemic, I was soon unsatisfied by its two-dimensional property. Thus, I created some QR cube sculptures via 3D printing and modelling.

It opened a new world.

By enhancing a 3D property, now a single cube can carry 6x amount of information. Most importantly, making it 3D allows these cubes to become building blocks of more figurative sculptures and numerous possibilities. I can perhaps build a mini statue of Liberty with QR cubes recording the federal rights, nor can I make sculptural portraits while consisting of all the model's personal information; it is endless.

However, I realized with the memory on the day of crits.

It must be accessible to the audience.

The audience can hardly make any QR cubes, nor having the ability to do modelling.

So, I need to make them capable of creating such a fun cube.

And this is where I started to develop this add-on-program, accessible to the general audience.

The development was challenging; I have no previous knowledge of both python and blender API as I only worked with C# before. But through the process of digging and researching is the API, the language and various libraries,

I managed to create this add-on, or did I?

Through the research, I went through numbers of knowledge from different incredible generous and selfless developers. The community, the great people on the internet, the founders of QR code, the people who are willing to sacrifice their capital interest and build blender and other resources opensource. It is they who managed to create this add-on with me.

Therefore, I should do the same and intended to do the same, making it open source.

Have fun with making QR Cubes ;)

## Add-on Program Logic

## 

## Pre-Installation add-on (pipcheckv1.py)

### (1) Pip / Library Installation (Operator)

The main Reason to have a Pip/Library Check and 1-Button Installation is because I Have struggled to import the PYPI through the blender console. Therefore, for the ease of general users, I decided to pack the pip python console installation script[[1]](#footnote-1) into a blender operator format.

1. class WM\_OT\_PipChecker(Operator):
2. bl\_label = "(1) PipCheck"
3. bl\_idname = "wm.pip\_check"
4. bl\_context = "object"
6. def execute(self, context):
7. scene = context.scene
8. mytool = scene.my\_tool
10. # path to python.exe
11. python\_exe = os.path.join(sys.prefix, 'bin', 'python.exe')
13. # Check The Latest Ver pip
14. subprocess.call([python\_exe, "-m", "ensurepip"])
15. subprocess.call([python\_exe, "-m", "pip", "install", "--upgrade", "pip"])
17. # install required packages
18. subprocess.call([python\_exe, "-m", "pip", "install", "Pillow"])
19. subprocess.call([python\_exe, "-m", "pip", "install", "qrcode"])
20. subprocess.call([python\_exe, "-m", "pip", "install", "lxml"]) #used for output svg files

23. print("Pip Check Complete")
25. return {'FINISHED'}

This process will install Pip[[2]](#footnote-2), and correspondingly install PIL[[3]](#footnote-3), qrcode[[4]](#footnote-4) and lxml[[5]](#footnote-5).

The installation is by subprocess.call()[[6]](#footnote-6) function, which is a call function of Py console through Blender API. – basically, it is just mimic the console lines of *-m pip install xxx*

## Main Add-on (qrc\_addon\_version101.py)

### (2) Generate Main Cube (Operator)

Since the logic is to generate 3D QR Meshes and apply Boolean with a main object cube, this operator will generate the main cube. It will also show a message when main cube is already in the scene.

1. class WM\_OT\_AddCube(Operator):
2. bl\_label = "(2) Add the Maincube"
3. bl\_idname = "wm.add\_cube"
4. bl\_context = "object"
6. def execute(self, context):
7. scene = context.scene
8. mytool = scene.my\_tool
10. cubetrue = bpy.data.collections.get('maincube') #bpy.data.collections is the data type path for collections
12. if cubetrue:
13. print("Main Cube Created already")
15. else:
16. print("Generating new Maincube")
17. # create a new cube
18. bpy.ops.mesh.primitive\_cube\_add(size=0.030)
20. # newly created cube will be automatically selected
21. maincube = bpy.context.selected\_objects[0]
22. #set new origin to center of obj mass
23. bpy.ops.object.origin\_set(type='ORIGIN\_GEOMETRY', center='BOUNDS')
24. # change name
25. maincube.name = "maincube"
27. # change its location
28. maincube.location = (0.0, 0.0, 0.0)
30. print("Main Cube Created")
32. return {'FINISHED'}

bpy.data.collections.get()[[7]](#footnote-7) will get the appearance of object in the data.

bpy.ops.mesh.primitive\_cube\_add()[[8]](#footnote-8) will add a mesh cube via the operator.

bpy.ops.object.origin\_set()[[9]](#footnote-9) It is also crucial to set the object origin to the centre of the object mass, so we can locate the distance that how far each 3d QR meshes is required to Boolean the indent.

### (3) Store Inputted String and Transform into QRC (.svg files) w/ PIL/qrcode/lxml (Operator)

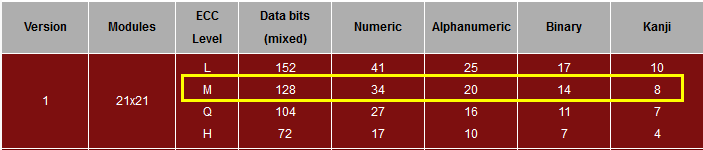
The storing of string input and transform to QR codes via requires 3 elements: string storage property, UI string input bar and the transform function operator.

The string storage property:

1. from bpy.props import (StringProperty)
2. class MyProperties(PropertyGroup):

5. my\_string1: StringProperty(
6. name="",
7. description=":",
8. default="",
9. maxlen=14, #Maxlength of string is dependend on the 21x21 size of module 1 type QR code (stores up to 14 binaries)
10. )
11. … #there is 5 more my\_string2 – my\_string5 properties, you can use list I believe.

StringProperty()[[10]](#footnote-10) is where the string data will be temporary stored, the max length is set to 14 strings due to the 14 binary limit data capacity of the version 1 module 1 type QR code[[11]](#footnote-11):



The UI input bar:

1. def draw(self, context):
2. layout = self.layout
3. scene = context.scene
4. mytool = scene.my\_tool
5. row = layout.row()
7. layout.prop(mytool, "my\_string1")
8. layout.prop(mytool, "my\_string2")
9. …
10. layout.prop(mytool, "my\_string6")

 It is the default side UI panel format, layout.prop()[[12]](#footnote-12) is to call the input bars.

The QR Code Transformer using Pillow, qrcode and lxml:

1. factory = qrcode.image.svg.SvgImage # Im using the basic factory method to get a connected svg
3. imgsvg1 = qrcode.make(mytool.my\_string1, image\_factory=factory)
4. imgsvg1.save('qrstorage/1.svg')
5. print("1.svg Created")

This is where qrcode module in the pillow library is used to transform my\_string1 in to a .svg qr code file. The application of this qr module is based on PIL qrcode documentation[[13]](#footnote-13).

.svg files are saved in the blender root folder > “qrstorage” folder, therefore, in the operation manual I warned users to create this empty qrstorage folder.

### (4) Import .svg files and Transform into 3D QR code Meshes (Operator)

This is by far the most complicated operator I developed in this add-on.

It consists of 1) file path existence checker; 2) Import files; 3) Select active object as base of join function; 4) Select all object in collection and convert into mesh type; 5) Join all the meshes to single object 6) Change to edit mode, select faces and extrude by value; 7) Reallocate the origin of the object to the centre of mass; 8) The reallocation of 3D QR Code Mesh, locate them into a cube shape for further Boolean.

1) file path existence checker[[14]](#footnote-14) and 2) Import files[[15]](#footnote-15):

1. import os
2. fp1 = 'qrstorage/1.svg'
3. if os.path.exists(fp1):
4. #import file
5. bpy.ops.import\_curve.svg(filepath = fp1)
6. print("FP1 Import Success")
8. svgt1 = bpy.data.collections.get('1.svg')

 3) Select active object as base of join function[[16]](#footnote-16):

1. bpy.context.view\_layer.objects.active = bpy.data.objects['Curve']

 4) Select all object[[17]](#footnote-17) in collection and convert into mesh type[[18]](#footnote-18) + 5) Join all the meshes[[19]](#footnote-19):

1. for obj in bpy.data.collections['1.svg'].all\_objects:
2. obj.select\_set(True)
3. #Operator Functions (Convert to mesh + Join)
4. bpy.ops.object.convert(target='MESH')
5. bpy.ops.object.join()

 6) Change to edit mode[[20]](#footnote-20), select faces[[21]](#footnote-21) and extrude by value[[22]](#footnote-22):

1. bpy.ops.object.mode\_set(mode='EDIT')# Go to edit mode, face selection mode and select all faces
2. bpy.ops.mesh.select\_mode(type='FACE')
3. bpy.ops.mesh.select\_all(action='SELECT')
4. bpy.ops.mesh.extrude\_region\_move(TRANSFORM\_OT\_translate={"value":(0, 0, 0.0045)})
6. bpy.ops.object.mode\_set( mode = 'OBJECT' )

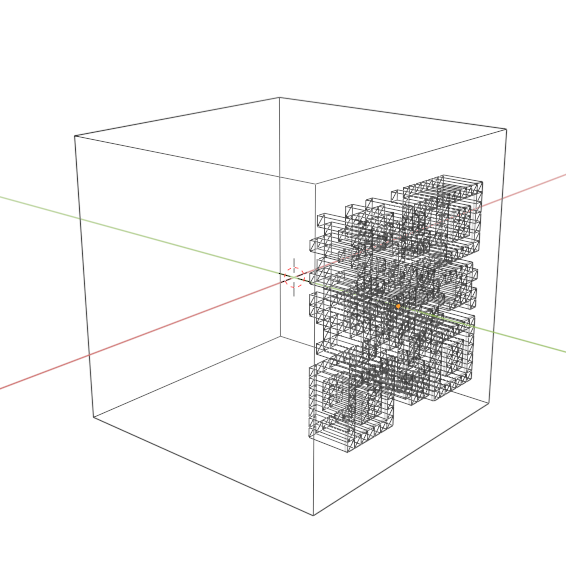
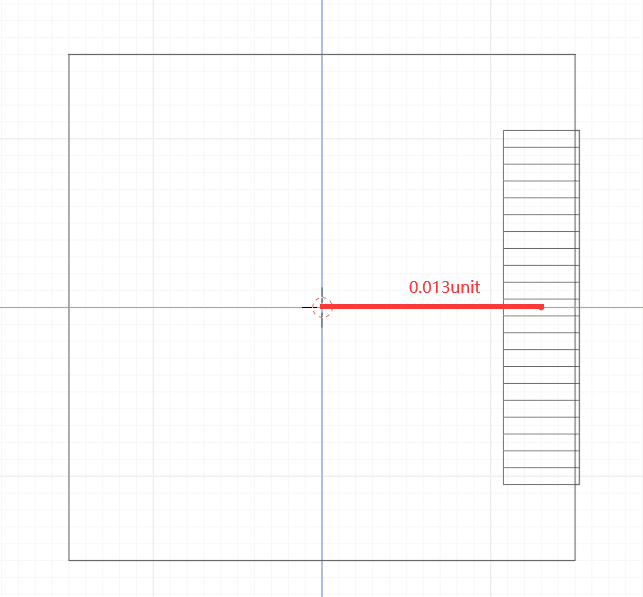
7) Reallocate the origin of the object (main cube) for further Boolean action[[23]](#footnote-23):

1. bpy.ops.object.origin\_set(type='ORIGIN\_GEOMETRY', center='BOUNDS')

8) The reallocation of 3D QR Code Mesh, locate them into a cube shape for further Boolean[[24]](#footnote-24)[[25]](#footnote-25):

1. #Locate
2. bpy.context.object.location[0] = 0.0 #x value
3. bpy.context.object.location[1] = -0.013 #y value
4. bpy.context.object.location[2] = 0.0 #z value
6. #Rotate
7. bpy.context.object.rotation\_euler[0] = 1.5708 #this number == 90 degrees

The result of the reallocation:

And this process is repeated for each faces of the cube with its corresponding vector.

### (5) Apply Blender Boolean Modules (Operator)

The Boolean module operator consists of mainly 2 parts: 1) Select main cube as active object for Boolean; 2) Apply Blender Boolean modifier scripts.

1) Select main cube as active object for Boolean (Pretty much same as the import operator):

1. cexist = scene.objects.get("maincube")
3. if cexist:
4. bpy.data.objects['Curve'].select\_set(True)
6. bpy.context.view\_layer.objects.active = bpy.data.objects['maincube']

2) Apply Blender Boolean modifier scripts:

1. #Boolean Function 1---------------------------------------------------
2. boolcube = scene.objects.get("maincube")
3. boolqrmesh = scene.objects.get("Curve")#Only Var
5. if boolcube and boolqrmesh:
6. bool = boolcube.modifiers.new(name='booltheqr', type='BOOLEAN')
7. bool.object = boolqrmesh
8. bool.operation = 'DIFFERENCE'
9. bpy.context.object.modifiers["booltheqr"].use\_self = True
10. bpy.ops.object.modifier\_apply(modifier='booltheqr', report=False)

In Boolean modifier scripts it consists of 1) Create a new blender modifier with type Boolean; 2) Set the operation as Difference Boolean; 3) Set the modifier use to self = true (so it creates indent); 4) Apply the modifier and clean the modifier bar.

1) Create a new blender modifier with type Boolean[[26]](#footnote-26):

1. bool = boolcube.modifiers.new(name='booltheqr', type='BOOLEAN')

2) Set the operation as Difference Boolean[[27]](#footnote-27):

1. bool.operation = 'DIFFERENCE'

3) Set the modifier use to self = true (so it creates indent)[[28]](#footnote-28):

1. bpy.context.object.modifiers["booltheqr"].use\_self = True

4) Apply the modifier and clean the modifier bar[[29]](#footnote-29):

1. bpy.ops.object.modifier\_apply(modifier='booltheqr', report=False)

### (6) Clean Up and Ready for Output (Operator)

This operator is the final part of the program logic, it consists of 2 parts: 1) Removal of collections; 2) delete every object in the scene apart from the main cube.

1) Removal of collections[[30]](#footnote-30):

1. collection1 = bpy.data.collections.get('1.svg')
2. if collection1:
3. bpy.data.collections.remove(collection1)

In blender API, collections are not seen as an object in the scene, more as a data type. Thus is must use …collections.remove() to erase the collections from the file.

2) delete every object in the scene apart from the main cube:

1. bpy.ops.object.select\_all()
3. mce = bpy.data.objects.get('maincube')
4. if mce:
5. bpy.data.objects['maincube'].select\_set(False)
7. bpy.ops.object.delete()

Similar to the object.get() check earlier on, it checks the existence of the main cube and deselect it. Then the line performance a blender delete operator[[31]](#footnote-31) on any other object selected in the beginning.

## Reference

### (1) From Blender API

7 **bpy.data.collections.get(),** in documentation it is wrote as **bpy\_struct.get(),** https://docs.blender.org/api/current/bpy.types.BlendDataCollections.html

8 **bpy.ops.mesh.primitive\_cube\_add(),** https://docs.blender.org/api/current/bpy.ops.mesh.html?highlight=bpy%20ops%20mesh%20primitive\_cube\_add#bpy.ops.mesh.primitive\_cube\_add

9 **bpy.ops.object.origin\_set(),** https://docs.blender.org/api/current/bpy.ops.object.html?highlight=bpy%20ops%20object%20origin\_set#bpy.ops.object.origin\_set

0 **StringProperty(),** https://docs.blender.org/api/current/bpy.props.html?highlight=stringproperty#bpy.props.StringProperty

2 **layout.prop(),** https://docs.blender.org/api/current/bpy.props.html?highlight=layout%20prop

4 **os.path.exists(),** https://docs.python.org/3/library/os.path.html?highlight=os%20path%20exists#os.path.exists

5 **Bpy.ops.import\_curve.svg(),** https://docs.blender.org/api/current/bpy.ops.import\_curve.html?highlight=bpy%20ops%20import\_curve%20svg#bpy.ops.import\_curve.svg

6 https://b3d.interplanety.org/en/how-to-set-object-mesh-to-active-in-blender-2-8-python-api/

17 **Obj.select.set(),** https://docs.blender.org/api/current/bpy.types.Object.html?highlight=obj%20select\_set#bpy.types.Object.select\_set

18 **Bpy.ops.obj.convert(),** https://docs.blender.org/api/current/bpy.ops.object.html?highlight=bpy%20ops%20object%20convert#bpy.ops.object.convert

19 **Bpy.ops.obj.join(),** https://docs.blender.org/api/current/bpy.ops.object.html?highlight=bpy%20ops%20object%20join#bpy.ops.object.join

20 **Bpy.ops.obj.mode\_set(),** https://docs.blender.org/api/current/bpy.ops.object.html?highlight=bpy%20ops%20object%20mode\_set#bpy.ops.object.mode\_set

21 **Bpy.ops.mesh.select\_mode(),** https://docs.blender.org/api/current/bpy.ops.mesh.html?highlight=bpy%20ops%20mesh%20select\_mode#bpy.ops.mesh.select\_mode

22 **bpy.ops.mesh.extrude\_region\_move(),**https://docs.blender.org/api/current/bpy.ops.mesh.html

23 https://docs.blender.org/api/current/bpy.ops.object.html?highlight=bpy%20ops%20object%20origin\_set#bpy.ops.object.origin\_set

24 **Bpy.contect.obj.location[] =,** https://docs.blender.org/api/current/info\_gotcha.html?highlight=bpy%20context%20object%20location

25 **rotation\_euler,** https://docs.blender.org/api/current/bpy.types.Object.html?highlight=bpy%20context%20object%20rotation\_euler

26 **Obj.Modifiers.new(),** https://docs.blender.org/api/current/bpy.types.ObjectModifiers.html?highlight=modifiers%20new#bpy.types.ObjectModifiers.new

27 https://docs.blender.org/api/current/bpy.types.BooleanModifier.html?highlight=bool%20operation#bpy.types.BooleanModifier.operation

28 https://docs.blender.org/api/current/bpy.types.BooleanModifier.html?highlight=bool%20operation#bpy.types.BooleanModifier.operation

29 **Modifier\_apply(),** https://docs.blender.org/api/current/bpy.ops.object.html?highlight=bpy%20ops%20object%20modifier\_apply

30 **Bpy.data.collections.remove(),** https://docs.blender.org/api/current/bpy.types.BlendDataCollections.html?highlight=bpy%20data%20collections%20remove#bpy.types.BlendDataCollections.remove

3 **Bpy.ops.object.delete()**, https://docs.blender.org/api/current/bpy.ops.object.html?highlight=bpy%20ops%20object%20delete#bpy.ops.object.delete

### (2) Other sources

In windows 3.x Python, to install pip – ‘-m ensurepip’

2 A universal python package installation tool, https://pip.pypa.io/en/stable/#

3 A python image library, https://pypi.org/project/Pillow/

4 A Pure python QR code generator, https://pypi.org/project/qrcode/

5 An XML/HTML processing tool with python, https://lxml.de/

6 **Subprocess.call()** from python framework, https://docs.python.org/3/library/subprocess.html?highlight=subprocess

0 The data capacity QR code variants, https://www.qrcode.com/en/about/version.html

3 Pillow, https://pypi.org/project/Pillow/

1. In windows 3.x Python, to install pip – ‘-m ensurepip’ [↑](#footnote-ref-1)
2. A universal python package installation tool, https://pip.pypa.io/en/stable/# [↑](#footnote-ref-2)
3. A python image library, https://pypi.org/project/Pillow/ [↑](#footnote-ref-3)
4. A Pure python QR code generator, https://pypi.org/project/qrcode/ [↑](#footnote-ref-4)
5. An XML/HTML processing tool with python, https://lxml.de/ [↑](#footnote-ref-5)
6. Subprocess.call() from python framework, https://docs.python.org/3/library/subprocess.html?highlight=subprocess [↑](#footnote-ref-6)
7. bpy.data.collections.get(), in documentation it is wrote as bpy\_struct.get(), https://docs.blender.org/api/current/bpy.types.BlendDataCollections.html [↑](#footnote-ref-7)
8. bpy.ops.mesh.primitive\_cube\_add(), https://docs.blender.org/api/current/bpy.ops.mesh.html?highlight=bpy%20ops%20mesh%20primitive\_cube\_add#bpy.ops.mesh.primitive\_cube\_add [↑](#footnote-ref-8)
9. bpy.ops.object.origin\_set(), https://docs.blender.org/api/current/bpy.ops.object.html?highlight=bpy%20ops%20object%20origin\_set#bpy.ops.object.origin\_set [↑](#footnote-ref-9)
10. StringProperty(), https://docs.blender.org/api/current/bpy.props.html?highlight=stringproperty#bpy.props.StringProperty [↑](#footnote-ref-10)
11. The data capacity QR code variants, https://www.qrcode.com/en/about/version.html [↑](#footnote-ref-11)
12. layout.prop(), https://docs.blender.org/api/current/bpy.props.html?highlight=layout%20prop [↑](#footnote-ref-12)
13. Pillow, https://pypi.org/project/Pillow/ [↑](#footnote-ref-13)
14. os.path.exists(), https://docs.python.org/3/library/os.path.html?highlight=os%20path%20exists#os.path.exists [↑](#footnote-ref-14)
15. Bpy.ops.import\_curve.svg(), https://docs.blender.org/api/current/bpy.ops.import\_curve.html?highlight=bpy%20ops%20import\_curve%20svg#bpy.ops.import\_curve.svg [↑](#footnote-ref-15)
16. https://b3d.interplanety.org/en/how-to-set-object-mesh-to-active-in-blender-2-8-python-api/ [↑](#footnote-ref-16)
17. Obj.select.set(), https://docs.blender.org/api/current/bpy.types.Object.html?highlight=obj%20select\_set#bpy.types.Object.select\_set [↑](#footnote-ref-17)
18. Bpy.ops.obj.convert(), https://docs.blender.org/api/current/bpy.ops.object.html?highlight=bpy%20ops%20object%20convert#bpy.ops.object.convert [↑](#footnote-ref-18)
19. Bpy.ops.obj.join(), https://docs.blender.org/api/current/bpy.ops.object.html?highlight=bpy%20ops%20object%20join#bpy.ops.object.join [↑](#footnote-ref-19)
20. Bpy.ops.obj.mode\_set(), https://docs.blender.org/api/current/bpy.ops.object.html?highlight=bpy%20ops%20object%20mode\_set#bpy.ops.object.mode\_set [↑](#footnote-ref-20)
21. Bpy.ops.mesh.select\_mode(), https://docs.blender.org/api/current/bpy.ops.mesh.html?highlight=bpy%20ops%20mesh%20select\_mode#bpy.ops.mesh.select\_mode [↑](#footnote-ref-21)
22. bpy.ops.mesh.extrude\_region\_move(),https://docs.blender.org/api/current/bpy.ops.mesh.html [↑](#footnote-ref-22)
23. https://docs.blender.org/api/current/bpy.ops.object.html?highlight=bpy%20ops%20object%20origin\_set#bpy.ops.object.origin\_set [↑](#footnote-ref-23)
24. Bpy.contect.obj.location[] =, https://docs.blender.org/api/current/info\_gotcha.html?highlight=bpy%20context%20object%20location [↑](#footnote-ref-24)
25. rotation\_euler, https://docs.blender.org/api/current/bpy.types.Object.html?highlight=bpy%20context%20object%20rotation\_euler [↑](#footnote-ref-25)
26. Obj.Modifiers.new(), https://docs.blender.org/api/current/bpy.types.ObjectModifiers.html?highlight=modifiers%20new#bpy.types.ObjectModifiers.new [↑](#footnote-ref-26)
27. https://docs.blender.org/api/current/bpy.types.BooleanModifier.html?highlight=bool%20operation#bpy.types.BooleanModifier.operation [↑](#footnote-ref-27)
28. https://docs.blender.org/api/current/bpy.types.BooleanModifier.html?highlight=bool%20operation#bpy.types.BooleanModifier.operation [↑](#footnote-ref-28)
29. Modifier\_apply(), https://docs.blender.org/api/current/bpy.ops.object.html?highlight=bpy%20ops%20object%20modifier\_apply [↑](#footnote-ref-29)
30. Bpy.data.collections.remove(), https://docs.blender.org/api/current/bpy.types.BlendDataCollections.html?highlight=bpy%20data%20collections%20remove#bpy.types.BlendDataCollections.remove [↑](#footnote-ref-30)
31. Bpy.ops.object.delete(), https://docs.blender.org/api/current/bpy.ops.object.html?highlight=bpy%20ops%20object%20delete#bpy.ops.object.delete [↑](#footnote-ref-31)