# Python scripting tutorial

Previous: Introduction to Python Index Next: FreeCAD Scripting Basics

| Tutorial |
| --- |
| **Topic** |
| Programming |
| **Level** |
| Intermediate |
| **Time to complete** |
| |
| **Authors** |
| |
| **FreeCAD version** |
| |
| **Example files** |

## Contents



Python (http://en.wikipedia.org/wiki/Python_%28programming_language%29) is a programming language, very simple to use and very fast to learn. It is open-source, multi-platform, and can be used alone for a wide array of things, from programming simple shell scripts to very complex programs. But one of its most widespread uses is as a scripting language, since it is easy to embed in other applications. That's exactly how it is used inside FreeCAD. From the python console, or from your custom scripts, you can pilot FreeCAD, and make it perform very complex actions for which there is still no graphical user interface tool.

For example, from a python script, you can:

- create new objects
- modify existing objects
- modify the 3D representation of those objects
- modify the FreeCAD interface

There are also several different ways to use python in FreeCAD:

- From the FreeCAD python interpreter, where you can issue simple commands like in a "command line"-style interface
- From macros, which are a convenient way to quickly add a missing tool to the FreeCAD interface
- From external scripts, which can be used to program much more complex things. like entire Workbenches.

In this tutorial, we'll work on a couple of simple examples to get you started, but there is also much more documentation about python scripting available

on this wiki. If you are totally new to python and want to understand how it works, we also have a basic introduction to Python.

**Important!** Before proceeding with Python scripting, go to **Edit** → **Prefences** → **General** → **Output** window and check 2 boxes:

- Redirect internal Python output to report view
- Redirect internal Python errors to report view

Then go to **View** → **Panels** and check:

- Report view

This will save you a lot of aggravation!

# Writing python code

There are two easy ways to write python code in FreeCAD: From the python console (available from the View → Panels → Python console menu) or from the Macro editor (Tools → Macros). In the console, you write python commands one by one, which are executed when you press return, while the macros can contain a more complex script made of several lines, which is executed only when the macro is executed.


The FreeCAD python console

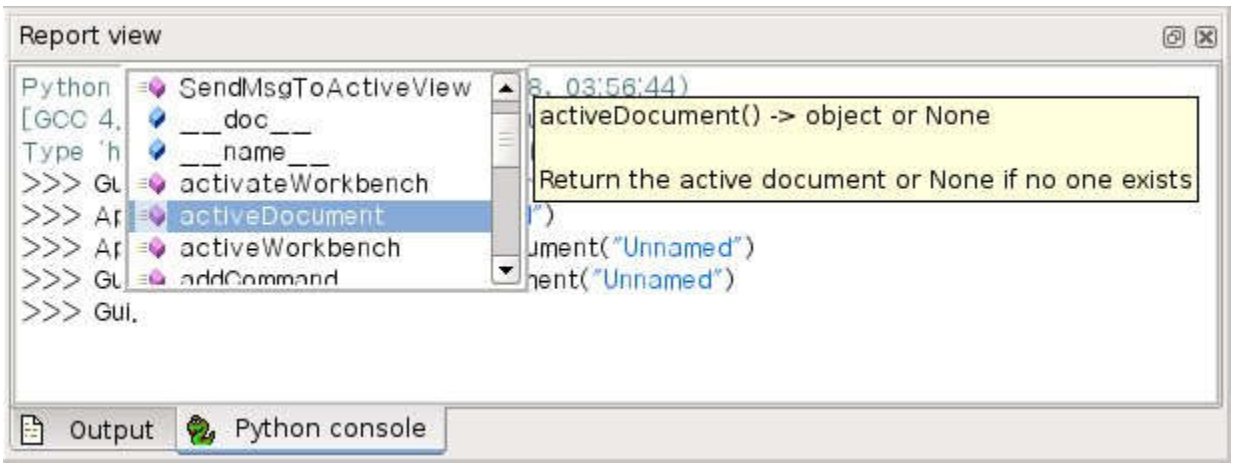
In this tutorial, you will be able to use both methods, either by copying/pasting each line one by one in the python console and pressing Return after each line, or by copying/pasting the entire code in a new Macro window.

# Exploring FreeCAD

Let's start by creating a new empty document:

```
doc = FreeCAD.newDocument()
```

If you type this in the FreeCAD python console, you will notice that as soon as you type "FreeCAD.", a windows pops up, allowing to quickly autocomplete the rest of your line. Even better, each entry in the autocomplete list has a tooltip explaining what it does. This makes it very easy to explore the functionality available. Before choosing "newDocument", have a look at the other options available.




The autocomplete mechanism of the FreeCAD python console

Now our new document will be created. This is similar to pressing the "new document" button on the toolbar. In fact, most buttons in FreeCAD do nothing else than executing a line or two of python code. Even better, you can set an option in **Edit → Preferences → General → Macro** to "show script commands in python console". This will print in the console all python code executed when you press buttons. Very useful to learn how to reproduce actions in python.

Now let's get back to our document. Let's see what we can do with it:

```
doc.
```

Explore the available options. Usually names that begin with a capital letter are attributes, they contain a value, while names that begin with small letter are functions (also called methods), they "do something". Names that begin with an underscore are usually there for the internal working of the module, and you shouldn't care about them. Let's use one of the methods to add a new object to our document:

```
box = doc.addObject("Part::Box","myBox")
```

Nothing happens. Why? Because FreeCAD is made for the big picture. One day, it will work with hundreds of complex objects, all depending one from another.

Making a small change somewhere could have a big impact, you may need to recalculate the whole document, which could take a long time... For that reason, almost no command updates the scene automatically. You must do it manually:

```
doc.recompute()
```

See? Now our box appeared! Many of the buttons that add objects in FreeCAD actually do 2 things: add the object, and recompute. If you turned on the "show script commands in python console" option above, try now adding a sphere with the GUI button, you'll see the two lines of python code being executed one after the other.

What about the "Part::Box" will you ask? How can I know what other kind of objects can be added? It's all here:

```
doc.supportedTypes()
```

Now let's explore the contents of our box:

```
box.
```

You'll immediately see a couple of very interesting things such as:

```
box.Height
```

This will print the current height of our box. Now let's try to change that:

```
box.Height = 5
```

If you select your box with the mouse, you'll see that in the properties panel, in the "Data" tab, our "Height" property appears. All properties of a FreeCAD object that appear there (and also in the "View" tab, more about that later), are directly accessible by python too, by their names, like we did with the "Height" property. Try changing the other dimensions of that box.

# Vectors and Placements

Vectors (http://en.wikipedia.org/wiki/Euclidean_vector) are a very fundamental concept in any 3D application. It is a list of 3 numbers (x, y and z), describing a point or position in the 3D space. A lot of things can be done with vectors, such as additions, subtractions, projections and much more (http://en.wikipedia.org/wiki/Vector_space). In FreeCAD vectors work like this:

```
myvec = FreeCAD.Vector(2,0,0)
myvec
myvec.x
myvec.y
othervec = FreeCAD.Vector(0,3,0)
sumvec = myvec.add(othervec)
```

Another common feature of FreeCAD objects is their placement. Each object has a Placement attributes, which contains the position (Base) and orientation (Rotation) of the object. It is easy to manipulate, for example to move our object:

```
box.Placement
box.Placement.Base
box.Placement.Base = sumvec

otherpla = FreeCAD.Placement()
box.Placement = otherpla
```

Now you must understand a couple of important concepts before we get further.

# App and Gui

FreeCAD is made from the beginning to work as a command-line application, without its user interface. As a result, almost everything is separated between a "geometry" component and a "visual" component. When you work in command-line mode, the geometry part is present, but all the visual part is simply disabled. Almost any object in FreeCAD therefore is made of two parts, an Object and a ViewObject.

To illustrate the concept, see our cube object, the geometric properties of the cube, such as its dimensions, position, etc... are stored in the object, while its visual properties, such as its color, line thickness, etc... are stored in the viewobject. This corresponds to the "Data" and "View" tabs in the property window. The view object of an object is accessed like this:

```
vo = box.ViewObject
```

Now you can also change the properties of the "View" tab:

```
vo.Transparency = 80
vo.hide()
vo.show()
```

When you start FreeCAD, the python console already loads 2 base modules: FreeCAD and FreeCADGui (which can also be accessed by their shortcuts App and Gui). They contain all kinds of generic functionality to work with documents and their objects. To illustrate our concept, see that both FreeCAD

and FreeCADGui contain an ActiveDocument attribute, which is the currently opened document. FreeCAD.ActiveDocument and FreeCADGui.ActiveDocument are not the same object. They are the two components of a FreeCAD document, and they contain different attributes and methods. For example, FreeCADGui.ActiveDocument contains ActiveView, which is the currently opened 3D view

# Modules

Now, you must be wondering, what, other than "Part::Box", can I do? The FreeCAD base application is more or less an empty container. Without its modules, it can do little more than create new, empty documents. The true power of FreeCAD is in its faithful modules. Each of them adds not only new workbenches to the interface, but also new python commands and new object types. As a result, several different or even totally incompatible object types can coexist in the same document. The most important modules in FreeCAD, that we'll look at in this tutorial, are Part, Mesh, Sketcher and Draft.

Sketcher and Draft both use the Part module to create and handle their geometry, which are BRep while Mesh is totally independent, and handles its own objects. More about that below.

You can check all the available base object types for the current document like this:

```
doc.supportedTypes()
```

The different FreeCAD modules, although they added their object types to FreeCAD, are not automatically loaded in the python console. This is to avoid having a very slow startup. Modules are loaded only when you need them. So, for example, to explore what's inside the Part module:

```
import Part
Part.
```

But we'll talk more about the Part module below.

By now, you know a bit more about the different modules of FreeCAD: The core modules (FreeCAD, FreeCADGui), and the workbench modules (Part, Mesh, Sketcher). The other important modules are the 3d scene module (pivy) and the interface module (pyside), we'll talk about them too below.

Now it's time to explore a bit deeper the important ones, which are the workbench modules.

# Mesh

Meshes (http://en.wikipedia.org/wiki/Polygon_mesh) are a very simple kind of 3D object, used for example by Sketchup (http://en.wikipedia.org/wiki/SketchUp), Blender (http://en.wikipedia.org/wiki/Blender_%28software%29) or 3D studio Max (http://en.wikipedia.org/wiki/Autodesk_3ds_Max). They are composed of 3 elements: points (also called vertices), lines (also called edges) and faces. In many applications, FreeCAD included, faces can have only 3 vertices. Of course, nothing prevents you from having a bigger plane face made of several coplanar triangles.

Meshes are simple, but because they are simple you can easily have millions of them in a single document. However, in FreeCAD they have less use and are mostly there so you can import objects in mesh formats (.stl, .obj) from other applications. The Mesh module was also used extensively as the main test module in the first month of FreeCAD's life.

Mesh objects and FreeCAD objects are different things. You can see the FreeCAD object as a container for a Mesh object (and as we'll see below, for Part objects also). So in order to add a mesh object to FreeCAD, we must first create a FreeCAD object and a Mesh object, then add the Mesh object to the FreeCAD object:

```
import Mesh
mymesh = Mesh.createSphere()
mymesh.
mymesh.Facets
mymesh.Points

meshobj = doc.addObject("Mesh::Feature","MyMesh")
meshobj.Mesh = mymesh
doc.recompute()
```

This is a standard example that uses the createSphere() method to automatically create a sphere, but you can also create custom meshes from scratch by defining their vertices and faces.

Read more about mesh scripting...

# Part

The Part Module is the most powerful module in the whole of FreeCAD. It allows you to create and manipulate BRep (http://en.wikipedia.org/wiki/Boundary_representation) objects. This kind of object, unlike meshes, can have a wide variety of components. Brep stands for Boundary Representation, which means that Brep objects are defined by their surfaces; those surfaces enclose and define an inner volume. A surface can be a variety of things such as plane faces or very complex NURBS surfaces.

The Part module is based on the powerful OpenCasCade (http://en.wikipedia.org/wiki/Open_CASCADE_Technology) library, which allows a wide range of complex operations to be easily performed on those objects, such as boolean operations, filleting, lofts, etc...

The Part module works the same way as the Mesh module: You create a FreeCAD object, a Part object, then add the Part object to the FreeCAD object:

```
import Part
myshape = Part.makeSphere(10)
myshape.
myshape.Volume
myshape.Area

shapeobj = doc.addObject("Part::Feature","MyShape")
shapeobj.Shape = myshape
doc.recompute()
```

The Part module (like the Mesh module) also has a shortcut that automatically creates a FreeCAD object and adds a shape to it, so you can shorten the last three lines above to:

```
Part.show(myshape)
```

By exploring the contents of myshape, you will notice many interesting subcomponents such as Faces, Edges, Vertexes, Solids and Shells, and a wide range of geometry operations such as cut (subtraction), common (intersection) or fuse (union). The Topological data scripting page explains all that in detail.

Read more about part scripting...

# Draft

FreeCAD features many more modules, such as Sketcher and Draft, which also create Part objects. These modules add additional parameters to the objects created, or even implement a whole new way to handle the Part geometry in them. Our box example above is a perfect example of a parametric object. All you need to define the box is to specify the parameters height, width and length. Based on those, the object will automatically calculate its Part shape. FreeCAD allows you to create such objects in python.

The Draft Module adds 2D parametric object types (which are all Part objects) such as lines and circles, and also provides some generic functions that work not only on Draft-made objects, but on any Part object. To explore what is available, simply do:

```
import Draft
```

```
Draft.
rec = Draft.makeRectangle(5,2)
mvec = FreeCAD.Vector(4,4,0)
Draft.move(rec,mvec)
Draft.move(box,mvec)
```

# Interface

The FreeCAD user interface is made with Qt (http://en.wikipedia.org/wiki/Qt_%28framework%29), a powerful graphical interface system, responsible for drawing and handling all the controls, menus, toolbars and buttons around the 3D view. Qt provides a module, PySide, which allows python to access and modify Qt interfaces such as FreeCAD. Let's try to fiddle with the Qt interface and produce a simple dialog:

```
from PySide import QtGui
QtGui.QMessageBox.information(None,"Apollo program","Houston, we have a problem")
```

Notice that the dialog that appears has the FreeCAD icon in its toolbar, meaning that Qt knows that the order has been issued from inside the FreeCAD application. We can therefore easily directly manipulate any part of the FreeCAD interface.

Qt is a very powerful interface system that allows you to do very complex things. It also has some easy-to-use tools such as the Qt Designer with which you can design dialogs graphically and then add them to the FreeCAD interface with a few lines of python code.

Read more about PySide here...

# Macros

Now that you have a good understanding of the basics, where are we going to keep our python scripts, and how are we going to launch them easily from FreeCAD? There is an easy mechanism for that, called Macros. A macro is simply a python script that can be added to a toolbar and launched via a mouse click. FreeCAD provides you with a simple text editor (Macro → Macros → Create) where you can write or paste scripts. Once the script is done, use Tools → Customiz → Macros to define a button for it that can be added to toolbars.

Now you are ready for more in-depth FreeCAD scripting. Head on to the Power users hub!

Previous: Introduction to Python Index Next: FreeCAD Scripting Basics

User documentation [Expand]