Getting started with FreeCAD scripting

J. Rabault

23rd January 2017

1 The philosophy behind FreeCAD

FreeCAD is based on several other Open Source libraries and links them in a convenient way in order to make it easy to perform CAD related tasks. At the heart of FreeCAD are **Open CASCADE** (an Open Source CAD kernel, for manipulating geometric objects), **Coin3D** (an Open Source implementation of the Open Inventor standard, which is used for the 3D rendering) and **PySide** (an Open Source Python binding for the Open Source GUI toolkit Qt). FreeCAD was designed since the beginning of the project for being modular, easy to extend and give full access to the underlying objects through a **Python API**.

Therefore, it is possible to use the full power of FreeCAD through Python programs. You can for example create, edit and modify objects and their 3D representations, perform operations on those objects etc. Actually, all of the FreeCAD Gui you used so far is just a way to interact in a user-friendly manner whith the underlying Python API. This is a great feature as it makes it easy to build on top of FreeCAD and it allows people to easily develop new functionalities and workbenches, see for example the **Fasteners workbench**:

`http://theseger.com/projects/2015/06/fasteners-workbench-for-freecad/`

`https://github.com/shaise/FreeCAD_FastenersWB`

Similarly, this lets users define complex composed objects from Python programs so that CAD can be done on virtually any geometry, not just the ones that are easily built from the FreeCAD GUI.

This 'getting started guide' is largely inspired from the FreeCAD website, but I try to gather the most important information in one single document and to give you more help to get started. You should have a look at the following page and the links presented there if you are looking for more details once you have gone through the materials presented here:

`http://www.freecadweb.org/wiki/?title=Power_users_hub`

2 A simple introduction to scripting from within FreeCAD

One option for scripting in FreeCAD is to use the **Python console**. This is what we will use here and this way, you can use Python as you would normally form the command line terminal,

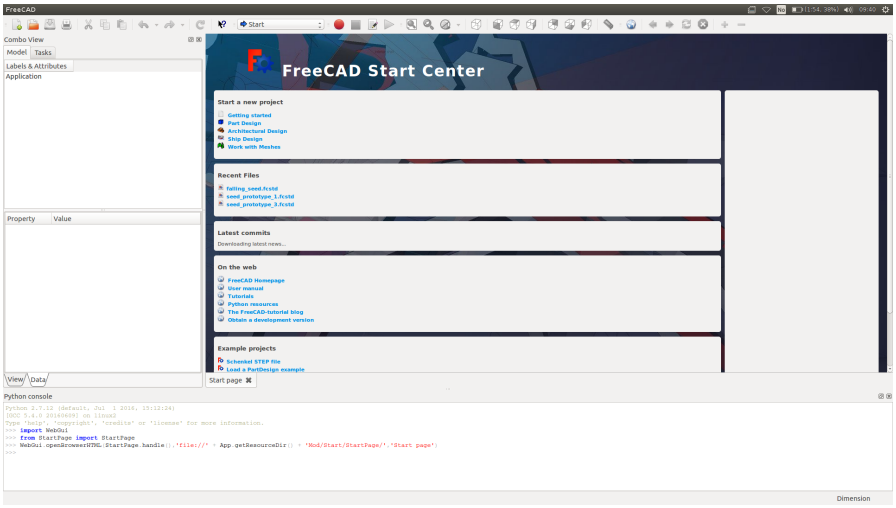


Figure 1: This is how FreeCAD should look like once the Python console is activated.

i.e. each line you type is interpreted in real time. Another possibility is to use the **Macro** tool, where you can write Python scripts that will be executed only once you launch the macro. Do your own search online when you are done with this section if you want to use macros.

Before starting to use the FreeCAD **Python console**, go to the **Edit > Preferences > General > Output window** menu and make sure that both the **Redirect internal Python output to report view** and the **Redirect internal Python errors to report view** options are checked in. Then go to the **View > Panels** menu and make sure that the **Report view** option is checked in. You can now start scripting from within FreeCAD. This tutorial will be very similar to (but maybe a bit more detailed than) the one on the FreeCAD website, so have a look at it when you are done:

`http://www.freecadweb.org/wiki/index.php?title=Python_scripting_tutorial`

- Start FreeCAD and open the **Python console** by ticking in the corresponding option in the **View > Panels** menu.

- You should get a result similar to Fig. 1. As you can see, the automatic startup routine of FreeCAD is already visible in the **Python console**. This is because, as was explained in the introduction, virtually everything you do in the FreeCAD GUI (except parametrizing the GUI itself, like moving tabs, selecting the **Data** tab instead of the **View** tab etc) is actually just commands to the Python API.

- Start by creating a new document, but from the **Python console**. For this, type:

  ```
  >>> doc = FreeCAD.newDocument("scriptingTest")
  ```

  You should notice that when you reach to the **FreeCAD.** stage while typing in the command, the **Python console** interactively displays an autocompletion list (as visible in Fig. 2) with minimalistic docstrings (available when you start scrolling up and down with the up and down arrows of your keyboard). This is a great way to quickly explore the possibilities of the FreeCAD scripting. Note that most fields that start by a capitalized letter are attributes that contain a value (which can be modified), while those that start

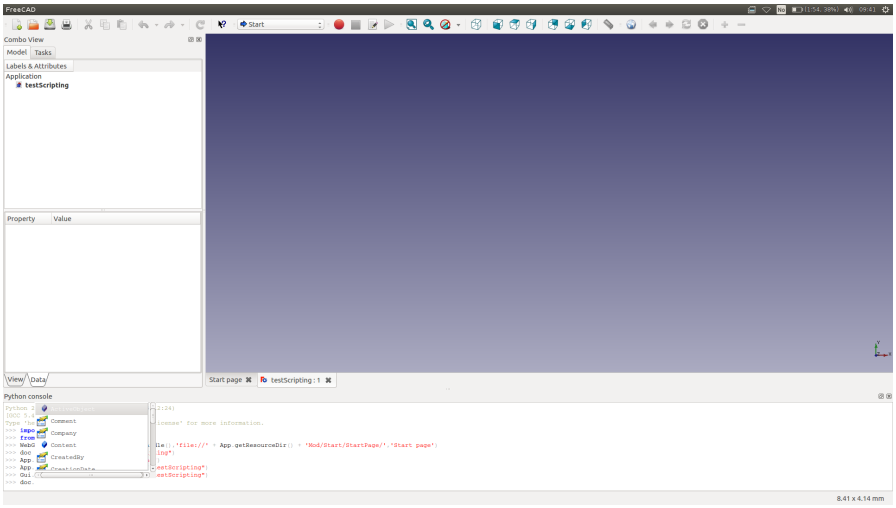


Figure 2: FreeCAD will help you explore the properties of your object.

with a small letter are functions. Actually, the command you just typed is completely equivalent to just creating a new document using the GUI button and renaming it.

- Since all actions you perform by clicking on GUI buttons are actually executed as python API commands that will get displayed in the **Python console**, this gives you a way to easily get examples of commands that you can re-use when you write Python commands. For example, move to the **Part** workbench. This is actually performed by a series of commands, that gets added in you **Python console**. Note the separation between functions that belong to the **FreeCAD** module (loaded under the short name **App**) and the **FreeCADGui** module (loaded under the short name **Gui**). While the **FreeCAD** module is used for creation of and operation on CAD objects, the **FreeCADGui** module is used for taking care of, well the GUI interface and rendering.

- Now click on **Create a cylinder**. Here also several lines are added to you **Python console**. Let us look at some of the lines that were added in the terminal.

```
>>> App.ActiveDocument.addObject("Part::Cylinder","Cylinder")
>>> App.ActiveDocument.ActiveObject.Label = "Cylinder"
```

  Those two commands are used to create a cylinder in the active document, and put a label on it.

```
>>> App.ActiveDocument.recompute()
```

  This line is used to recompute the document and update the graphical view. If it is not included, the graphical window in your GUI will not get updated.

- Let us take an example where we build a cylinder entirely from the command line. Create a new document by typing:

```
>>> doc = FreeCAD.newDocument("scriptingTest2")
```

  Note that this is exactly equivalent to:

```
>>> doc = App.newDocument("scriptingTest2")
```

because the Python **FreeCAD** module is also imported under the name **App**.

- The GUI will automatically move to the new document you created, and set it as the **ActiveDocument** in both the **App** and the **Gui** modules. The **ActiveDocument** field, as well as the **ActiveObject** field, are important for the internal working of the GUI (FreeCAD needs to keep track of which object you are editing!).

- Now create a cylinder by typing:

```
>>> cylinder_1 = FreeCAD.ActiveDocument.addObject(
    "Part::Cylinder","my_cylinder_1")
```

  You just created a **Cylinder object**, under the name (in Python) **cylinder_1**, which appears in your model tree under the **label string** that you put in the constructor (namely, **my_cylinder_1**).

- Note that nothing appeared on the screen, since we did not recompute the document. Type:

```
>>> doc.recompute()
```

  and your cylinder will appear.

- We can now edit the properties of the cylinder. By starting to type **cylinder_1.**, you will display the autocompletion for your cylinder object. Observe that it is easy to understand what some of the fields are doing, for example **Height** or **Radius**. Edit your cylinder object with:

```
>>> cylinder_1.Radius = 4
>>> cylinder_1.Height = 15
```

- As we said, all buttons of FreeCAD are just a way of calling one (or, often, several) FreeCAD Python API functions. Click for example on the **Create a cube solid** button, and then do a **Union of several shapes** between your cube and your cylinder by using the GUI. Observe the Python commands generated. You could adapt and use them if your need them in scripts.

- Of course, you also have access in addition to the full power of Python! Type for example:

```
>>> import os
>>> os.listdir(os.getcwd())
>>> import math
>>> math.sqrt(121)
>>> math.log10(1000)
```

  This will let you generate sophisticated shapes, build complex programs, and more generally use any program piece you could write to drive FreeCAD.

Now you should start to be a bit familiar with the very basis of the FreeCAD Python console, and more importantly you should start to understand the link between what happens when you use the FreeCAD GUI and what is done in the background, and how you can observe how the background works, its syntax etc.

3 Scripting from outside of FreeCAD

Of course, while it is nice to use the Python console on-the-fly from FreeCAD now and then, it is much nicer to write sophisticated scripts using your favorite text editor. This can easily be done, by programming in Python from **outside of FreeCAD**, and importing FreeCAD as any python **module**.

Open a new Python console, either directly in terminal (by typing 'python' in your command line), or through for example a jupyter notebook (look at jupyter notebook on google for more information) or even better use Atom together with the Hydrogen plugin (look at Atom + Hydrogen on google for more information). I personally like using Atom with Hydrogen, as you will see in the screenshots, but do whatever you like best.

- In order to be able to **import FreeCAD as a module**, you will need to give to Python the path to FreeCAD.
- For this, first find the path to your FreeCAD installation. This path is the one that leads to you **FreeCAD.so** or **FreeCAD.dll** file.

  If you are working on your own Linux computer, find it by using in terminal:

  ```
  $ locate FreeCAD.so
  ```

  In my case on an Ubuntu 16.04, the output I get is **/usr/lib/freecad/lib/FreeCAD.so** and the path I am looking for is therefore **/usr/lib/freecad/lib/**

  If you are using an UiO computer, you will have to locate the module you load. For this, type in terminal:

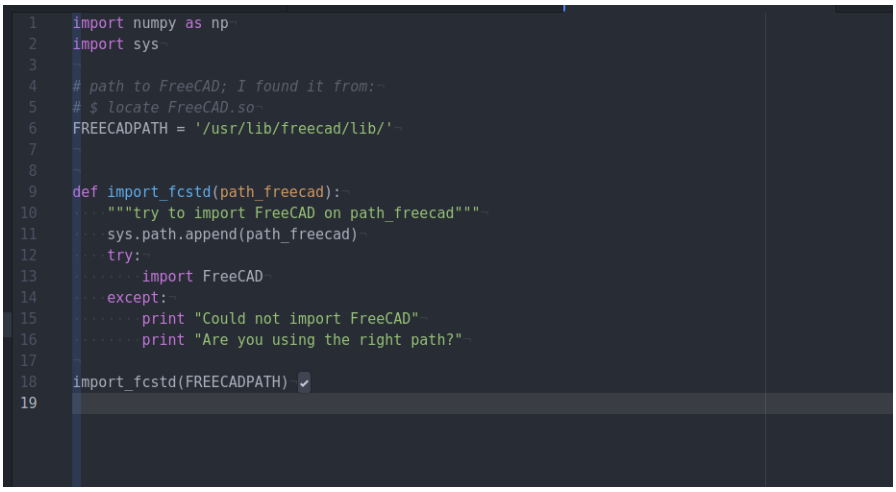
  ```
  $ module load freecad/0.16.6706
  $ which FreeCAD
  ```

  This will output the path to the FreeCAD module you just loaded. In my case, I get **/opt/uio/modules/packages/freecad/0.16.6706/bin/FreeCAD**. Then move to the corresponding directory:

  ```
  $ cd /opt/uio/modules/packages/freecad/0.16.6706/bin
  ```

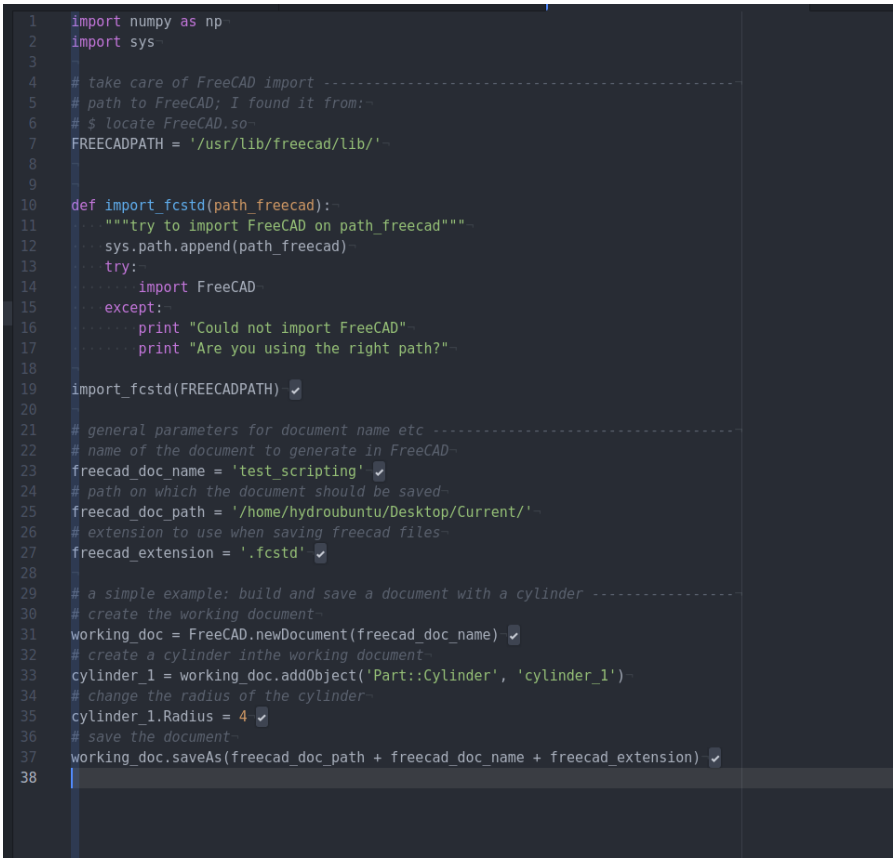
  and start exploring a bit around to find the **/lib/FreeCAD.so** (actually, without surprise, it should be in **/opt/uio/modules/packages/freecad/0.16.6706/lib**). The **/opt/uio/modules/packages/freecad/0.16.6706/lib** path, or anything else you determined on your system, is the one that you should then use.

- You can now write in the code presented in Fig. 3 and execute it in Python (adapt the FREECADPATH if necessary of course!). This should let you import FreeCAD as a Python module.
- Once FreeCAD is imported, you can use it to build whatever geometry you like the same way you would do in the FreeCAD GUI built-in Python console. For example, the script



```python
import numpy as np
import sys

# path to FreeCAD; I found it from:
# $ locate FreeCAD.so
FREECADPATH = '/usr/lib/freecad/lib/'


def import_fcstd(path_freecad):
    """try to import FreeCAD on path_freecad"""
    sys.path.append(path_freecad)
    try:
        import FreeCAD
    except:
        print "Could not import FreeCAD"
        print "Are you using the right path?"

import_fcstd(FREECADPATH)

```

Figure 3: Import FreeCAD as a Python module.



```python
import numpy as np
import sys

# take care of FreeCAD import ----------------------------------------------
# path to FreeCAD; I found it from:
# $ locate FreeCAD.so
FREECADPATH = '/usr/lib/freecad/lib/'


def import_fcstd(path_freecad):
    """try to import FreeCAD on path_freecad"""
    sys.path.append(path_freecad)
    try:
        import FreeCAD
    except:
        print "Could not import FreeCAD"
        print "Are you using the right path?"

import_fcstd(FREECADPATH)

# general parameters for document name etc ---------------------------------
# name of the document to generate in FreeCAD
freecad_doc_name = 'test_scripting'
# path on which the document should be saved
freecad_doc_path = '/home/hydroubuntu/Desktop/Current/'
# extension to use when saving freecad files
freecad_extension = '.fcstd'

# a simple example: build and save a document with a cylinder ----------------
# create the working document
working_doc = FreeCAD.newDocument(freecad_doc_name)
# create a cylinder inthe working document
cylinder_1 = working_doc.addObject('Part::Cylinder', 'cylinder_1')
# change the radius of the cylinder
cylinder_1.Radius = 4
# save the document
working_doc.saveAs(freecad_doc_path + freecad_doc_name + freecad_extension)

```

Figure 4: You can now build and save CAD models from Python.

presented in Fig. 4 will create a cylinder, change a few properties, and save it in the output path.