

Editors - Python Console

Python Console.....	1
Menus.....	1
Console Menu.....	1
Edit Menu.....	1
Usage.....	2
Accessing Built-in Python Console.....	2
First look at the Console Environment.....	2
Auto Completion at work.....	2
Before tinkering with the modules.....	3
Examples.....	4
bpy.context.....	4
Try it out!.....	4
bpy.data.....	5
Try it out!.....	5
Exercise.....	5
bpy.ops.....	6
Try it out!.....	6

Python Console

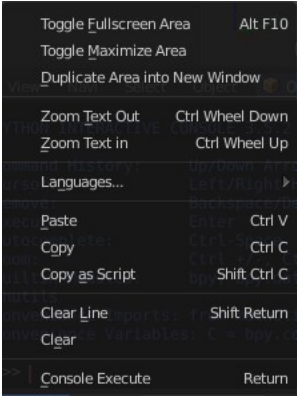
The Python console is a quick way to execute commands, with access to the entire Python API, command history and auto-complete.

Its a good way to explore possibilities, which can then be pasted into larger scripts.

Menus

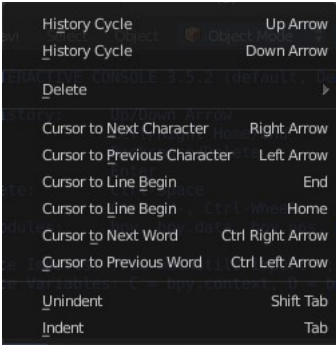
Console Menu

The Console menu provides you with the text operation and window tools.



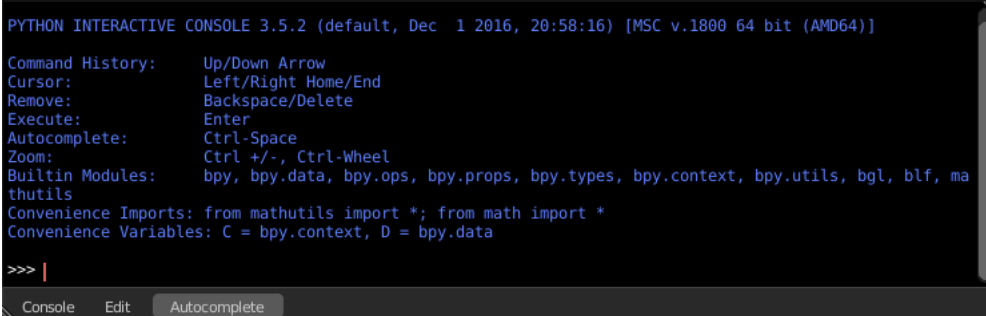
Edit Menu

The Edit menu provides you with the usual navigation tools.



Usage

Accessing Built-in Python Console



```
PYTHON INTERACTIVE CONSOLE 3.5.2 (default, Dec 1 2016, 20:58:16) [MSC v.1800 64 bit (AMD64)]

Command History: Up/Down Arrow
Cursor: Left/Right Home/End
Remove: Backspace/Delete
Execute: Enter
Autocomplete: Ctrl-Space
Zoom: Ctrl +/-, Ctrl-Wheel
Builtin Modules: bpy, bpy.data, bpy.ops, bpy.props, bpy.types, bpy.context, bpy.utils, bgl, blf, ma
thutils
Convenience Imports: from mathutils import *; from math import *
Convenience Variables: C = bpy.context, D = bpy.data

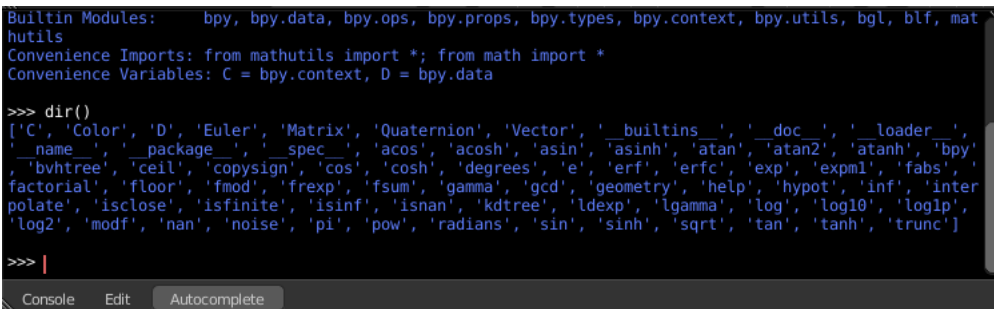
>>> |
```

From the screen shot above, you will notice that by clicking at the Autocomplete button you can enable Auto-complete feature.

The command prompt is typical for Python 3.x, the interpreter is loaded and is ready to accept commands at the prompt >>>

First look at the Console Environment

To check what is loaded into the interpreter environment, type `dir()` at the prompt and execute it.



```
Builtin Modules: bpy, bpy.data, bpy.ops, bpy.props, bpy.types, bpy.context, bpy.utils, bgl, blf, mat
hutils
Convenience Imports: from mathutils import *; from math import *
Convenience Variables: C = bpy.context, D = bpy.data

>>> dir()
['C', 'Color', 'D', 'Euler', 'Matrix', 'Quaternion', 'Vector', '__builtins__', '__doc__', '__loader__',
 '__name__', '__package__', '__spec__', 'acos', 'acosh', 'asin', 'asinh', 'atan', 'atan2', 'atanh', 'bpy',
 'bvhtree', 'ceil', 'copysign', 'cos', 'cosh', 'degrees', 'e', 'erf', 'erfc', 'exp', 'expm1', 'fabs', 'f
actorial', 'floor', 'fmod', 'frexp', 'fsum', 'gamma', 'gcd', 'geometry', 'help', 'hypot', 'inf', 'inter
polate', 'isclose', 'isfinite', 'isinf', 'isnan', 'kdtree', 'ldexp', 'lgamma', 'log', 'log10', 'log1p',
 'log2', 'modf', 'nan', 'noise', 'pi', 'pow', 'radians', 'sin', 'sinh', 'sqrt', 'tan', 'tanh', 'trunc']

>>> |
```

Following is a quick overview of the output

C

Quick access to `bpy.context`

D

Quick access to `bpy.data`

bpy

Top level Bforartists Python API module.

The rest of the commands are of various content. Most of them are mathematical expressions.

Auto Completion at work

Now, type `bpy.` and then press the Autocomplete button, and you will see the Console auto-complete feature in action.

```
Builtin Modules: bpy, bpy.data, bpy.ops, bpy.props, bpy.types, bpy.context, bpy.utils, bgl, blf, mat
hutils
Convenience Imports: from mathutils import *; from math import *
Convenience Variables: C = bpy.context, D = bpy.data

>>> dir()
['C', 'Color', 'D', 'Euler', 'Matrix', 'Quaternion', 'Vector', '_builtins_', '_doc_', '_loader_',
'_name_', '_package_', '_spec_', 'acos', 'acosh', 'asin', 'asinh', 'atan', 'atan2', 'atanh', 'bpy',
'_bvhtree_', 'ceil', 'copysign', 'cos', 'cosh', 'degrees', 'e', 'erf', 'erfc', 'exp', 'expm1', 'fabs',
'factorial', 'floor', 'fmod', 'frexp', 'fsum', 'gamma', 'gcd', 'geometry', 'help', 'hypot', 'inf', 'inter
polate', 'isclose', 'isfinite', 'isinf', 'isnan', 'kdtree', 'ldexp', 'lgamma', 'log', 'log10', 'log1p',
'log2', 'modf', 'nan', 'noise', 'pi', 'pow', 'radians', 'sin', 'sinh', 'sqrt', 'tan', 'tanh', 'trunc']

>>> bpy.
```

You will notice that a list of sub-modules inside of bpy appear. These modules encapsulate all that we can do with Bforartists Python API and are very powerful tools.

Lets list all the contents of bpy.app module.

```
['C', 'Color', 'D', 'Euler', 'Matrix', 'Quaternion', 'Vector', '_builtins_', '_doc_', '_loader_',
'_name_', '_package_', '_spec_', 'acos', 'acosh', 'asin', 'asinh', 'atan', 'atan2', 'atanh', 'bpy',
'_bvhtree_', 'ceil', 'copysign', 'cos', 'cosh', 'degrees', 'e', 'erf', 'erfc', 'exp', 'expm1', 'fabs',
'factorial', 'floor', 'fmod', 'frexp', 'fsum', 'gamma', 'gcd', 'geometry', 'help', 'hypot', 'inf', 'inter
polate', 'isclose', 'isfinite', 'isinf', 'isnan', 'kdtree', 'ldexp', 'lgamma', 'log', 'log10', 'log1p',
'log2', 'modf', 'nan', 'noise', 'pi', 'pow', 'radians', 'sin', 'sinh', 'sqrt', 'tan', 'tanh', 'trunc']

>>> bpy.app.version
(2, 78, 4)

>>> bpy.app.version_string
'2.78 (sub 4)'

>>> |
```

Notice the green output above the prompt where you enabled auto-completion. What you see is the result of auto completion listing. In the above listing all are module attribute names, but if you see any name end with '(' , then that is a function.

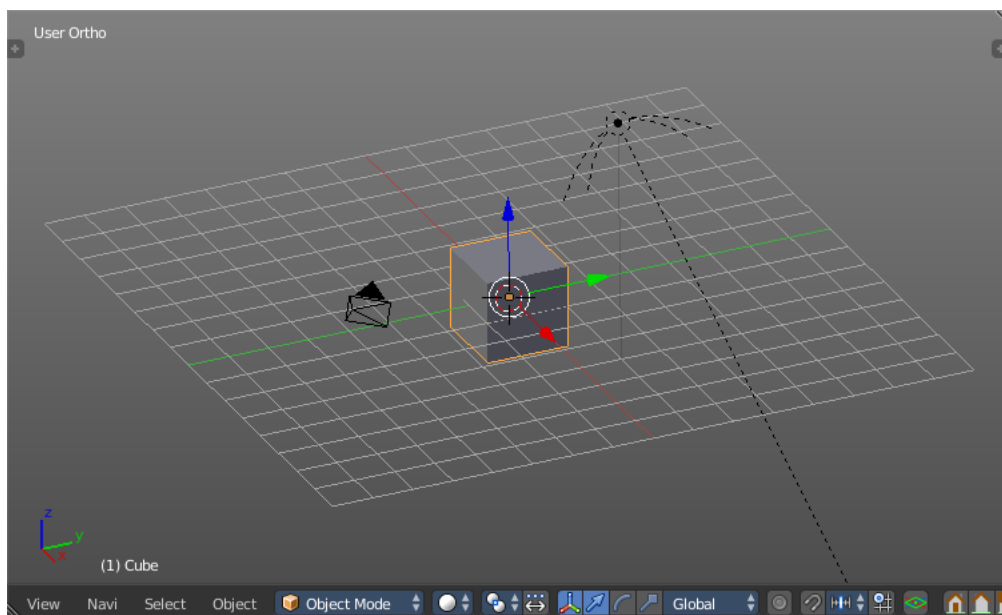
We will make use of this a lot to help our learning the API faster. Now that you got a hang of this, lets proceed to investigate some of modules in bpy.

Before tinkering with the modules..

If you look at the 3D Viewport in the default Bforartists scene, you will notice some objects. We have added a cube here too.

- All objects exist in a context and there can be various modes under which they are operated upon.
- At any instance, only one object is active and there can be more than one selected objects.
- All objects are data in the Bforartists file.
- There are operators/functions that create and modify these objects.

For all the scenarios listed above (not all were listed, mind you..) the bpy module provides functionality to access and modify data.



Examples

Note

For the commands below to show the proper output, make sure you have selected object(s) in the 3D view.

bpy.context

```
>>> bpy.context.mode
'OBJECT'

>>> bpy.context.object
bpy.data.objects['Cube']

>>> bpy.context.active_object
bpy.data.objects['Cube']

>>> bpy.context.selected_objects
[bpy.data.objects['Cube']]

>>> bpy.context.selected_objects
[bpy.data.objects['Cube'], bpy.data.objects['Lamp Hemi'], bpy.data.objects['Camera']]

>>> |
```

Try it out!

bpy.context.mode

Will print the current 3D View mode (Object, Edit, Sculpt etc.,)

bpy.context.object or bpy.context.active_object

Will give access to the active object in the 3D View

```
>>> bpy.context.object.location.x = 1
```

Change x location to a value of 1

```
>>> bpy.context.object.location.x += 0.5
```

Move object from previous x location by 0.5 unit

```
>>> bpy.context.object.location = (1, 2, 3)
```

Changes x, y, z location

```
>>> bpy.context.object.location.xyz = (1, 2, 3)
```

Same as above

```
>>> type(bpy.context.object.location)
```

Data type of objects location

```
>>> dir(bpy.context.object.location)
```

Now that is a lot of data that you have access to

bpy.context.selected_objects

Will give access to a list of all selected objects.

```
>>> bpy.context.selected_objects
```

... then press **Ctrl-Spacebar**

```
>>> bpy.context.selected_objects[0]
```

Prints out name of first object in the list

```
>>> [object for object in bpy.context.selected_objects if object != bpy.context.object]
```

Complex one... But this prints a list of objects not including the active object

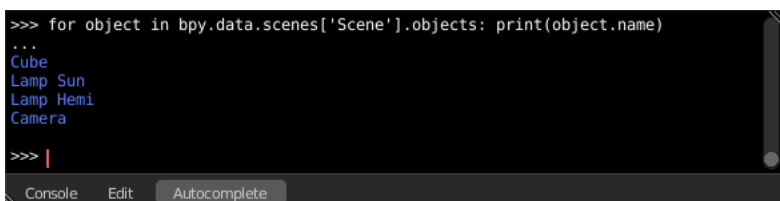
bpy.data

`bpy.data` has functions and attributes that give you access to all the data in the Bforartists file.

You can access following data in the current Bforartists file: objects, meshes, materials, textures, scenes, screens, sounds, scripts, ... etc.

That's a lot of data.

Try it out!



```
>>> for object in bpy.data.scenes['Scene'].objects: print(object.name)
...
Cube
Lamp Sun
Lamp Hemi
Camera
>>> |
```

Exercise

```
>>> for object in bpy.data.scenes['Scene'].objects: print(object.name)
```

Return twice Prints the names of all objects belonging to the Bforartists scene with name “Scene”

```
>>> bpy.data.scenes['Scene'].objects.unlink(bpy.context.active_object)
```

Unlink the active object from the Bforartists scene named ‘Scene’

```
>>> bpy.data.materials['Material'].shadows
```

```
>>> bpy.data.materials['Material'].shadows = False
```

bpy.ops

The tool/action system in Bforartists 2.5 is built around the concept of operators. These operators can be called directly from console or can be executed by click of a button or packaged in a python script. Very powerful they are..

For a list of various operator categories, [click here](#)

Lets create a set of five Cubes in the 3D Viewport. First, delete the existing Cube object by selecting it and pressing X

Try it out!

The following commands are used to specify that the objects are created in layer 1. So first we define an array variable for later reference:

```
>>> mylayers = [False] * 20
>>> mylayers[0] = True
```

We create a reference to the operator that is used for creating a cube mesh primitive

```
>>> add_cube = bpy.ops.mesh.primitive_cube_add
```

Now in a for loop, we create the five objects like this (In the screenshot above, I used another method) Press ENTER-KEY twice after entering the command at the shell prompt.

```
>>> for index in range(0, 5): add_cube(location=(index * 3, 0, 0), layers=mylayers)
```

