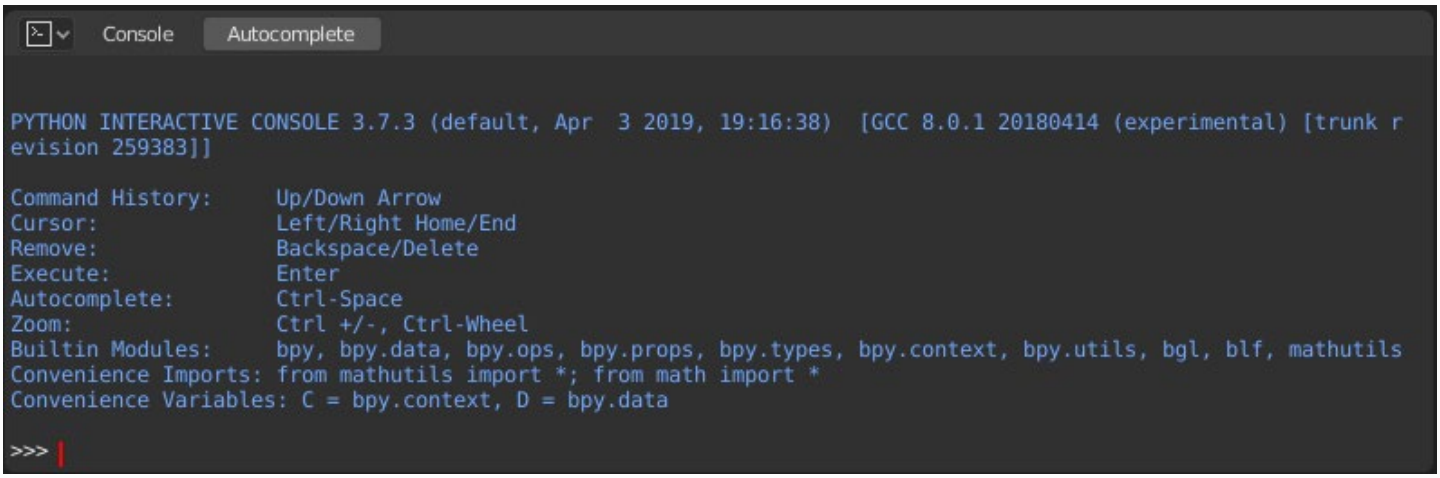
Python Console

Python Consoleは、Python API全体、コマンド履歴、オートコンプリートにアクセスし、コマンドを素早く実行する方法です。コマンドプロンプトはPython 3.xの典型的なもので、インタプリタがロードされ、プロンプト>>>でコマンドを受け入れる準備が整っています。

Python コンソールは、Blender 組み込みの Python の可能性を探るのによい方法です。Python コンソールは、Python コードの小さな断片をテストするために使用でき、その後より大きなスクリプトに貼り付けることができます。



Python Console.

Interface

Header Menus

View Menu

Zoom In / Zoom Out

コンソールテキストのフォントサイズを変化'(する大小）。

Move to Previous Word `Ctrl-Left`

カーソルを前の単語の先頭に移動させます。カーソルが単語の途中にある場合、カーソルを現在の単語の先頭に移動させる。

Move to Next Word `Ctrl-Right`

カーソルを次の単語の末尾に移動させます。カーソルが単語の途中にある場合、カーソルを現在の単語の末尾に移動させる。

Move to Line Begin Home

カーソルを現在の行の先頭に移動します。

Move to Line End End

カーソルを現在の行の末尾に移動します。

Console Menu

Clear All
コンソールをリフレッシュして、表示を新しくします。ただし、コマンド履歴はクリアされません

Clear Line `Shift-Return`.
プロンプト行からすべてを削除します。

Delete Previous Word `Ctrl-Backspace`

Deletes everything between the cursor and the beginning of the previous word (separated by periods). If the cursor is in the middle of a word, deletes everything to the beginning of the current word.

Delete Next Word `Ctrl-Delete`
カーソルから次の単語の末尾までのすべてを削除する。カーソルが単語の途中にある場合、現在の単語の終わりまでをすべて削除する。

Copy as Script `Shift-Ctrl-C`

履歴バッファをクリップボードにコピーし、テキストファイルに貼り付けてPythonスクリプトとして使用することができます。

Copy `Ctrl-C`
選択範囲をコピーします。

Paste `Ctrl-V`
コマンドラインに貼り付けます。

Indent `Tab`
選択範囲の字下げを解除します。

Unindent `Shift-Tab`
カーソルの位置にタブ文字を挿入する。

Backward in History `Up`
現在のコマンドをコマンドヒストリーに表示されている前のコマンドに変更します。

Forward in History `Down`
現在のコマンドを、コマンドヒストリーに表示されている次のコマンドに変更します。

Autocomplete `Tab`

詳しくは、Auto Completion fを参照してください。

Main View

Key Bindings

- `Left` / `Right` – Cursor motion.
- `Ctrl-Left` / `Ctrl-Right` – Cursor motion, by word.
- `Backspace` / `Delete` – Erase characters.
- `Ctrl-Backspace` / `Ctrl-Delete` – Erase words.
- `Return` – Execute command.
- `Shift-Return` – Add to command history without executing.

Usage

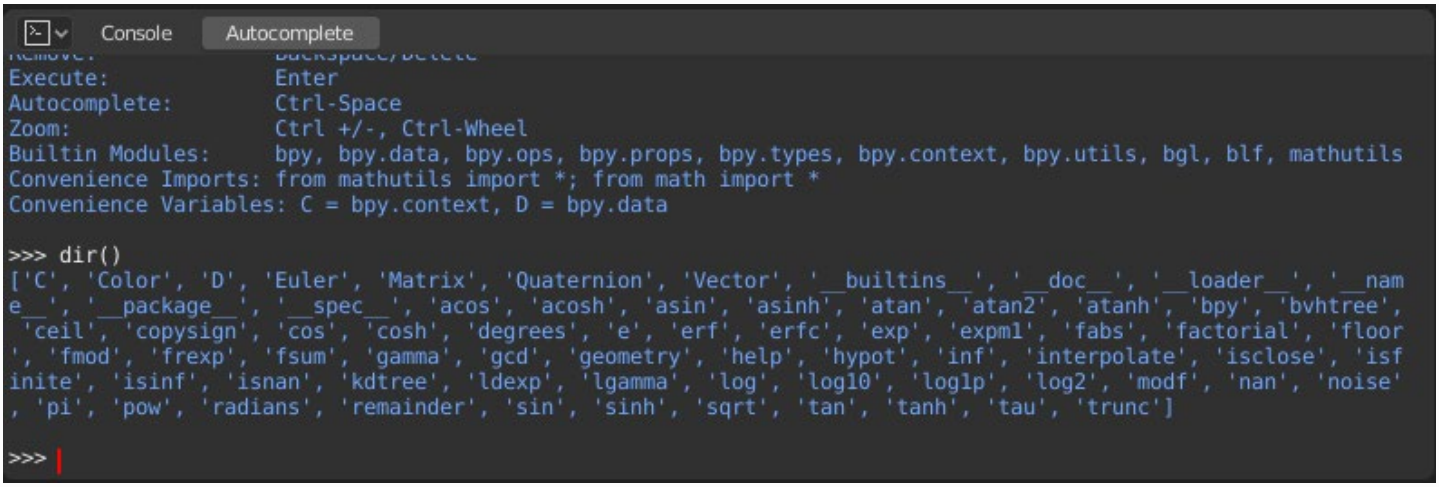
Aliases

便宜上、一部の変数やモジュールを用意しています：

- `C` : Quick access to `bpy.context`.
- `D` : Quick access to `bpy.data`.
- `bpy` : Top level Blender Python API module.

First Look at the Console Environment（コンソール環境の初見）

インタプリタ環境に何が読み込まれているかを確認するには、プロンプトでdir()と入力し、実行してみてください。



Auto Completion

ここで、bpy.と入力してTabを押すと、Consoleのオートコンプリート機能が動作していることがわかります。

bpyの中にあるサブモジュールのリストが表示されることに気づくでしょう。これらのモジュールは、Blender Python APIでできることをすべてカプセル化したもので、非常に強力なツールです。

bpy.appモジュールの内容をすべてリストアップしてみましょう。

自動補完を有効にしたプロンプトの上にある、緑色の出力に注目してください。表示されているのは、自動補完のリストアップの結果です。上のリストでは、すべてモジュールの属性名ですが、(,) で終わる名前があれば、それは関数です。

APIをより早く習得するために、この機能を大いに活用しましょう。それでは、bpyのモジュールについて調べてみましょう。す

Before Tinkering with the Modules（モジュールに手を加える前に）

デフォルトのBlenderシーンで3Dビューポートを見ると、3つのオブジェクトがあることに気づきます： キューブ（Cube）、ライト（Light）、カメラ（Camera）です。 and Camera.

- すべてのオブジェクトはコンテキストの中に存在し、それらを操作するための様々なモードが存
- 在することができます。どのような場合でも、アクティブなオブジェクトは1つだけで、選択されたオブジェクトは1つ以上存在することがあります。
- すべてのオブジェクトは、ブレンドファイル内のデータです。
- これらのオブジェクトを作成し、変更するオペレータ/ファンクションが存在します。

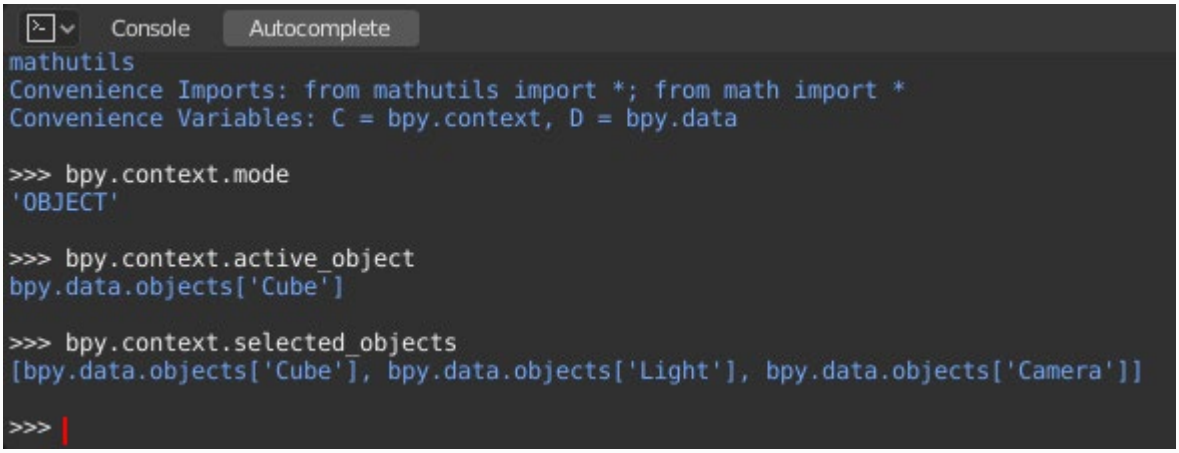
上記のすべてのシナリオ（すべてではありませんが...）に対して、bpyモジュールはデータにアクセスし、修正する機能を提供します。

Examples

bpy.context

Note

以下のコマンドで適切な出力を得るには、3Dビューポートでオブジェクトが選択されていることを確認します。



`bpy.context.mode`
: 現在の3Dビューポートモード（Object、Edit、Sculptなど）を表示します。

`bpy.context.object` or `bpy.context.active_object`

3D Viewportでアクティブなオブジェクトにアクセスできるようになります。

Xの位置の値を1に変更します：

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

オブジェクトを前のX位置から0.5単位で移動させます：

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

X、Y、Zの位置を変更する：

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

X、Y成分のみを変更する：

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

オブジェクトの位置のデータ型：

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

さて、これでアクセスできるデータが多くなりましたね：

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

`bpy.context.selected_objects`

選択されたすべてのオブジェクトのリストにアクセスできるようになります。

```
bpy.context.selected_objects
```

リストの最初のオブジェクトの名前を出力します：

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

複雑なのは…ですが,これはアクティブなオブジェクトを含まないオブジェクトのリストを表示します：

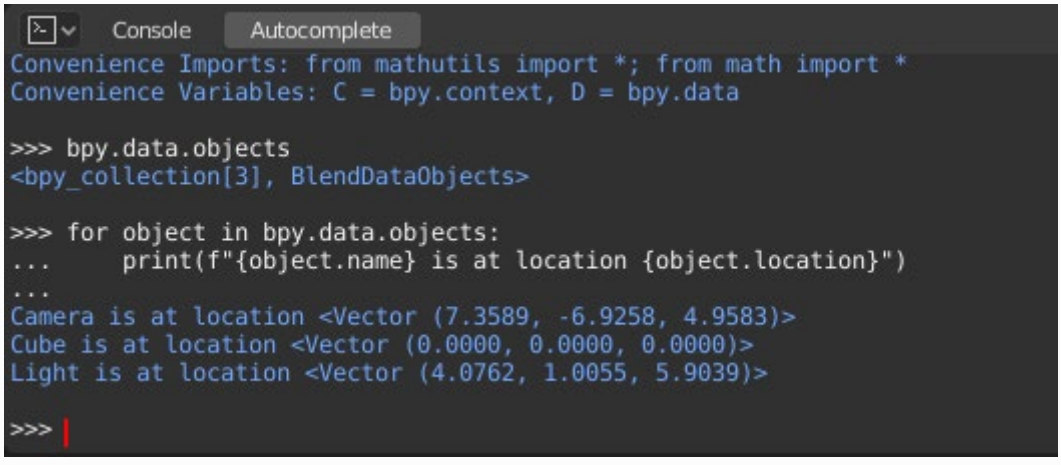
```
[obj for obj in bpy.context.selected_objects if obj != bpy.context.object]
```

bpy.data

`bpy.data`には、ブレンドファイル内のすべてのデータにアクセスするための関数と属性があります。

現在のブレンドファイル内の以下のデータにアクセスできます：オブジェクト、メッシュ、マテリアル、テクスチャ、シーン、スクリーン、サウンド、スクリプト、その他。

すごいデータ量ですね。



bpy.ops

ツールシステムは、オペレータの概念を中心に構築されています。オペレータは通常、ボタンやメニューから実行されますが、Pythonから直接呼び出すこともできます。
ボタンやメニューから実行されますが、Pythonから直接呼び出すことも可能です。

すべての演算子の一覧は、bpy.ops APIドキュメントを参照してください。

Previous Next

View Source Report issue on this page