An A-Z of useful Python tricks

Peter Gleeson
Aug 28, 2018 · 9 min read

Photo by Almos Bechtold on Unsplash

Python is one of the world's most popular, in-demand programming languages. This is for many reasons:

- it's easy to learn
- it's super versatile
- it has a huge range of modules and libraries

I use Python daily as an integral part of my job as a data scientist. Along the way, I've picked up a few useful tricks and tips.

Here, I've made an attempt at sharing some of them in an A-Z format.

Most of these 'tricks' are things I've used or stumbled upon during my day-to-day work. Some I found while browsing the Python Standard Library docs. A few others I found searching through PyPi.

However, credit where it is due — I discovered four or five of them over at awesome-python.com. This is a curated list of hundreds of interesting Python tools and modules. It is worth browsing for inspiration!

all or any

One of the many reasons why Python is such a popular language is because it is readable and expressive.

It is often joked that Python is 'executable pseudocode'. But when you can write code like this, it's difficult to argue otherwise:

```
x = [True, True, False]

if any(x):
    print("At least one True")

if all(x):
    print("Not one False")

if any(x) and not all(x):
    print("At least one True and one False")
```

bashplotlib

You want to plot graphs in the console?

```
$ pip install bashplotlib
```

You can have graphs in the console.

collections

Python has some great default datatypes, but sometimes they just won't behave exactly how you'd like them to.

Luckily, the Python Standard Library offers the collections module. This handy add-on provides you with further datatypes.

```
from collections import OrderedDict, Counter

# Remembers the order the keys are added!
x = OrderedDict(a=1, b=2, c=3)

# Counts the frequency of each character
y = Counter("Hello World!")
```

dir

Ever wondered how you can look inside a Python object and see what attributes it has? Of course you have.

From the command line:

```
>>> dir()
>>> dir("Hello World")
>>> dir(dir)
```

This can be a really useful feature when running Python interactively, and for dynamically exploring objects and modules you are working with.

Read more here.

emoji

Yes, really.

```
$ pip install emoji
```

Don't pretend you're not gonna try it out…

```
from emoji import emojize

print(emojize(":thumbs_up:"))
```

👍

from __future__ import

One consequence of Python's popularity is that there are always new versions under development. New versions mean new features—unless your version is out-of-date.

Fear not, however. The __future__ module lets you import functionality from future versions of Python. It's literally like time travel, or magic, or something.

```
from __future__ import print_function

print("Hello World!")
```

Why not have a go importing curly braces?

geopy

Geography can be a challenging terrain for programmers to navigate (ha, a pun!). But the geopy module makes it unnervingly easy.

```
$ pip install geopy
```

It works by abstracting the APIs of a range of different geocoding services. It enables you to obtain a place's full street address, latitude, longitude, and even altitude.

There's also a useful distance class. It calculates the distance between two locations in your favorite unit of measurement.

```
from geopy import GoogleV3

place = "221b Baker Street, London"
location = GoogleV3().geocode(place)
```

```
print(location.address)
print(location.location)
```

howdoi

Stuck on a coding problem and can't remember that solution you saw before? Need to check StackOverflow, but don't want to leave the terminal?

Then you need this useful command line tool.

```
$ pip install howdoi
```

Ask it whatever question you have, and it'll do its best to return an answer.

```
$ howdoi vertical align css
$ howdoi for loop in java
$ howdoi undo commits in git
```

Be aware though — it scrapes code from top answers from StackOverflow. It might not always give the most helpful information…

```
$ howdoi exit vim
```

inspect

Python's inspect module is great for understanding what is happening behind the scenes. You can even call its methods on itself!

The code sample below uses `inspect.getsource()` to print its own source code. It also uses `inspect.getmodule()` to print the module in which it was defined.

The last line of code prints out its own line number.

```
import inspect

print(inspect.getsource(inspect.getsource))
print(inspect.getmodule(inspect.getmodule))
print(inspect.currentframe().f_lineno)
```

Of course, beyond these trivial uses, the inspect module can prove useful for understanding what your code is doing. You could also use it for writing self-documenting code.

Jedi

The Jedi library is an autocompletion and code analysis library. It makes writing code quicker and more productive.

Unless you're developing your own IDE, you'll probably be most interested in using Jedi as an editor plugin. Luckily, there are already loads available!

You may already be using Jedi, however. The IPython project makes use of Jedi for its code autocompletion functionality.

**kwargs

When learning any language, there are many milestones along the way. With Python, understanding the mysterious `**kwargs` syntax probably counts as one.

The double-asterisk in front of a dictionary object lets you pass the contents of that dictionary as named arguments to a function.

The dictionary's keys are the argument names, and the values are the values passed to the function. You don't even need to call it `kwargs`!

```
dictionary = {"a": 1, "b": 2}

def someFunction(a, b):
    print(a + b)
    return

# these do the same thing:
someFunction(**dictionary)
someFunction(a=1, b=2)
```

This is useful when you want to write functions that can handle named arguments not defined in advance.

List comprehensions

One of my favourite things about programming in Python are its list comprehensions.

These expressions make it easy to write very clean code that reads almost like natural language.

You can read more about how to use them here.

```
numbers = [1,2,3,4,5,6,7]
evens = [x for x in numbers if x % 2 is 0]
odds = [y for y in numbers if y not in evens]

cities = ['London', 'Dublin', 'Oslo']

def visit(city):
    print("Welcome to "+city)

for city in cities:
    visit(city)
```

map

Python supports functional programming through a number of inbuilt features. One of the most useful is the `map()` function — especially in combination with lambda functions.

```
x = [1, 2, 3]
y = map(lambda x : x + 1 , x)

# prints out [2,3,4]
print(list(y))
```

In the example above, `map()` applies a simple lambda function to each element in `x` . It returns a map object, which can be converted to some iterable object such as a list or tuple.

newspaper3k

If you haven't seen it already, then be prepared to have your mind blown by Python's newspaper module.

It lets you retrieve news articles and associated meta-data from a range of leading international publications. You can retrieve images, text and author names.

It even has some inbuilt NLP functionality.

So if you were thinking of using BeautifulSoup or some other DIY webscraping library for your next project, save yourself the time and effort and `$ pip install newspaper3k` instead.

Operator overloading

Python provides support for operator overloading, which is one of those terms that make you sound like a legit computer scientist.

It's actually a simple concept. Ever wondered why Python lets you use the `+` operator to add numbers and also to concatenate strings? That's operator overloading in action.

You can define objects which use Python's standard operator symbols in their own specific way. This lets you use them in contexts relevant to the objects you're working with.

```
class Thing:
    def __init__(self, value):
        self.__value = value
    def __gt__(self, other):
        return self.__value > other.__value
    def __lt__(self, other):
        return self.__value < other.__value

something = Thing(100)
nothing = Thing(0)

# True
something > nothing

# False
something < nothing

# Error
something + nothing
```

pprint

Python's default `print` function does its job. But try printing out any large, nested object, and the result is rather ugly.

Here's where the Standard Library's pretty-print module steps in. This prints out complex structured objects in an easy-to-read format.

A must-have for any Python developer who works with non-trivial data structures.

```
import requests
import pprint

url = 'https://randomuser.me/api/?results=1'
users = requests.get(url).json()

pprint.pprint(users)
```

Queue

Python supports multithreading, and this is facilitated by the Standard Library's Queue module.

This module lets you implement queue data structures. These are data structures that let you add and retrieve entries according to a specific rule.

'First in, first out' (or FIFO) queues let you retrieve objects in the order they were added. 'Last in, first out' (LIFO) queues let you access the most recently added objects first.

Finally, priority queues let you retrieve objects according to the order in which they are sorted.

Here's an example of how to use queues for multithreaded programming in Python.

__repr__

When defining a class or an object in Python, it is useful to provide an 'official' way of representing that object as a string. For example:

```
>>> file = open('file.txt', 'r')
>>> print(file)
<open file 'file.txt', mode 'r' at 0x10d30aaf0>
```

This makes debugging code a lot easier. Add it to your class definitions as below:

```
class someClass:
    def __repr__(self):
        return "<some description here>"
```

```
someInstance = someClass()

# prints <some description here>
print(someInstance)
```

sh

Python makes a great scripting language. Sometimes using the standard os and subprocess libraries can be a bit of a headache.

The sh library provides a neat alternative.

It lets you call any program as if it were an ordinary function — useful for automating workflows and tasks, all from within Python.

```
import sh

sh.pwd()
sh.mkdir('new_folder')
sh.touch('new_file.txt')
sh.whoami()
sh.echo('This is great!')
```

Type hints

Python is a dynamically-typed language. You don't need to specify datatypes when you define variables, functions, classes etc.

This allows for rapid development times. However, there are few things more annoying than a runtime error caused by a simple typing issue.

Since Python 3.5, you have the option to provide type hints when defining functions.

```
def addTwo(x : Int) -> Int:
    return x + 2
```

You can also define type aliases:

```
from typing import List
```

```
Vector = List[float]
Matrix = List[Vector]

def addMatrix(a : Matrix, b : Matrix) -> Matrix:
  result = []
  for i,row in enumerate(a):
    result_row =[]
    for j, col in enumerate(row):
      result_row += [a[i][j] + b[i][j]]
    result += [result_row]
  return result

x = [[1.0, 0.0], [0.0, 1.0]]
y = [[2.0, 1.0], [0.0, -2.0]]

z = addMatrix(x, y)
```

Although not compulsory, type annotations can make your code easier to understand.

They also allow you to use type checking tools to catch those stray TypeErrors before runtime. Probably worthwhile if you are working on large, complex projects!

uuid

A quick and easy way to generate Universally Unique IDs (or 'UUIDs') is through the Python Standard Library's uuid module.

```
import uuid

user_id = uuid.uuid4()
print(user_id)
```

This creates a randomized 128-bit number that will almost certainly be unique.

In fact, there are over 2^{122} possible UUIDs that can be generated. That's over five undecillion (or 5,000,000,000,000,000,000,000,000,000,000,000,000).

The probability of finding duplicates in a given set is extremely low. Even with a trillion UUIDs, the probability of a duplicate existing is much, much less than one-in-a-billion.

Pretty good for two lines of code.

Virtual environments

This is probably my favorite Python thing of all.

Chances are you are working on multiple Python projects at any one time. Unfortunately, sometimes two projects will rely on different versions of the same dependency. Which do you install on your system?

Luckily, Python's support for virtual environments lets you have the best of both worlds. From the command line:

```
python -m venv my-project
source my-project/bin/activate
pip install all-the-modules
```

Now you can have standalone versions and installations of Python running on the same machine. Sorted!

wikipedia

Wikipedia has a great API that allows users programmatic access to an unrivalled body of completely free knowledge and information.

The wikipedia module makes accessing this API almost embarrassingly convenient.

```
import wikipedia

result = wikipedia.page('freeCodeCamp')
print(result.summary)
for link in result.links:
    print(link)
```

Like the real site, the module provides support for multiple languages, page disambiguation, random page retrieval, and even has a `donate()` method.

xkcd

Humour is a key feature of the Python language — after all, it is named after the British comedy sketch show Monty Python's Flying Circus. Much of Python's official documentation references the show's most famous sketches.

The sense of humour isn't restricted to the docs, though. Have a go running the line below:

```
import antigravity
```

Never change, Python. Never change.

YAML

YAML stands for 'YAML Ain't Markup Language'. It is a data formatting language, and is a superset of JSON.

Unlike JSON, it can store more complex objects and refer to its own elements. You can also write comments, making it particularly suited to writing configuration files.

The PyYAML module lets you use YAML with Python. Install with:

```
$ pip install pyyaml
```

And then import into your projects:

```
import yaml
```

PyYAML lets you store Python objects of any datatype, and instances of any user-defined classes also.

zip

One last trick for ya, and it really is a cool one. Ever needed to form a dictionary out of two lists?

```
keys = ['a', 'b', 'c']
vals = [1, 2, 3]
zipped = dict(zip(keys, vals))
```

The `zip()` inbuilt function takes a number of iterable objects and returns a list of tuples. Each tuple groups the elements of the input objects by their positional index.

You can also 'unzip' objects by calling `*zip()` on them.

Thanks for reading!

So there you have it, an A-Z of Python tricks — hopefully you've found something useful for your next project.

Python's a very diverse and well-developed language, so there's bound to be many features I haven't got round to including.

Please share any of your own favorite Python tricks by leaving a response below!

Python Programming Technology Learning Software Development