PYBITES PYTHON TIPS

Real World Python Tips for the Well-Rounded Developer

BOB BELDERBOS JULIAN SEQUEIRA

PyBites Python Tips

Real World Python Tips for
the Well-Rounded Developer

(Sample ebook)

Bob Belderbos & Julian Sequeira

PyBites Python Tips: Real World Python Tips for the Well-Rounded Developer

By Bob Belderbos & Julian Sequeira

Version 4.0.0

For any issues or inquiries email us: support@pybit.es

Thanks to Robin Beer and Chris May for proof reading and a special thanks to our technical reviewer Andrew Jarcho.

Cover design by @ryanurz

Contents

Introduction

Tip: a small but useful piece of practical advice. - Google search

Beautiful is better than ugly. - Zen of Python

Welcome to our Python tips book. It has been a long time coming and we are so proud to finally get it into your hands (digitally speaking).

Great developers read and write a lot of code and our tips have helped thousands of them improve their Python.

Python is a beautiful language with a rich standard library but it's still a huge undertaking to become proficient with it.

It can be difficult to discover awesome features that will make you shine as a developer.

The Zen of Python says: *There should be one-- and preferably only one --obvious way to do it.*

This can potentially take years to figure out which is why we distilled our years of experience into these practical snippets that will help you get there faster.

Regularly reviewing our tips is also the ideal *spaced repetition* that will make you a more effective developer. It will save you lines of code, will make your code more idiomatic ("Pythonic") and you will impress your colleagues and tech recruiters with your increasing knowledge of the language.

A little bit of history

We started sharing tips on *Twitter* roughly 2 years ago and the beautiful code images (produced with Carbon) gained traction immediately. This is what inspired us to write this book.

Practical Python Tips

In this free ebook you will find 10 real world PyBites Python Tips. Enjoy!

3. Create a dictionary using zip

Create a `dict` of two sequences using the `zip` built-in:

```
>>> names = 'bob julian tim sara'.split()
>>> ages = '11 22 33 44'.split()
>>> zip(names, ages)
<zip object at 0x7fae75920d20>
>>> list(zip(names, ages))
[('bob', '11'), ('julian', '22'), ('tim', '33'), ('sara', '44')]
>>> dict(zip(names, ages))
{'bob': '11', 'julian': '22', 'tim': '33', 'sara': '44'}
```

Explanation

The `dict` constructor can receive a list of tuples.

Here we use `zip` to combine names and ages. This built-in creates an iterator intertwining two or more sequences ("iterables").

By feeding this into `dict` we get a dictionary back where the first elements of each tuple pair are the keys and the second elements are the values.

It only works with 2 element tuples, giving it 3 you'd get a `ValueError: dictionary update sequence element #0 has length 3; 2 is required`.

Resources

https://stackoverflow.com/a/209854

7. Enumerate

If you need the index inside a loop in Python use `enumerate`:

```
>>> names = 'bob julian tim sara'.split()
>>> for i, name in enumerate(names, start=1):
...     print(i, name)
...
1 bob
2 julian
3 tim
4 sara
```

Explanation

Wrapping `enumerate` around an iterator you get a counter for free.

By default it starts at 0, but you can change that using the optional `start` keyword arg.

Resources

https://docs.python.org/3/library/functions.html#enumerate

Exercise

Bite 15. Enumerate 2 sequences

10. Is vs == (object equality)

The difference in Python between comparing objects and their values:

```
>>> a = [1, 2, 3]
>>> b = [1, 2, 3]
>>> c = a
>>> a == b  # same content
True
>>> a == c  # also same content
True
>>> a is c  # same object
True
>>> a is b  # not the same object
False
# to check for equal objects you can check their
# identity using id()
>>> id(a), id(b), id(c)
(140611808855040, 140611819909632, 140611808855040)
```

Explanation

In Python `is` checks that 2 arguments refer to the same object, `==` is used to check that they have the same value.

To check whether variables refer to the same object you can use the `id()` built-in which, as per the docs, returns an “identity” integer which is guaranteed to be unique and constant for the object's lifetime.

Resources

https://stackoverflow.com/a/15008404
https://docs.python.org/3/library/functions.html#id

Exercise

Bite 80. Check equality of two lists

14. Collections.Counter

For counting in Python look no further than `collections.Counter`:

```
>>> from collections import Counter
>>> languages = 'Python Java Perl Python JS C++ JS Python'.split()
>>> Counter(languages)
Counter({'Python': 3, 'JS': 2, 'Java': 1, 'Perl': 1, 'C++': 1})
>>> Counter(languages).most_common(2)
[('Python', 3), ('JS', 2)]
```

Explanation

It does not get more Pythonic than this ;)

`Counter()` can receive an iterable, a mapping or keyword args (nice!)

`most_common` is useful to get, well, the most common elements.

Resources

https://docs.python.org/3.9/library/collections.html#collections.Counter

Exercise

Bite 18. Find the most common word

20. Zfill

Give a number leading zeros in Python using `zfill`:

```
>>> for i in range(1, 6):
...     str(i).zfill(3)
...
'001'
'002'
'003'
'004'
'005'

>>> for i in range(1, 6):
...     str(i).zfill(5)
...
'00001'
'00002'
'00003'
'00004'
'00005'

>>> for i in range(-3, 2):
...     str(i).zfill(3)
...
'-03'
'-02'
'-01'
'000'
'001'
```

Explanation

This is a great technique to print titles in your app, e.g. "Bite 02"

Resources

https://docs.python.org/3/library/stdtypes.html?highlight=zfill#str.zfill

31. Set operations

You want to compare 2 sequences in Python? Enter set operations:

```
>>> a = {1, 2, 3, 4, 5}  # or use set() on a list
>>> b = {1, 2, 3, 6, 7, 8}
# unique to a
>>> a - b
{4, 5}
# unique to b
>>> b - a
{8, 6, 7}
# in both sets
>>> a & b
{1, 2, 3}
# in either one or the other
>>> a ^ b
{4, 5, 6, 7, 8}
# no need for more verbose (and probably slower) looping
>>> line1 = ['You', 'can', 'do', 'anything', 'but', 'not', 'everything']
>>> line2 = ['We', 'are', 'what', 'we', 'repeatedly', 'do']
>>> for word in line1:
...     if word in line2: print(word)
...
do
>>> set(line1) & set(line2)
{'do'}
```

Explanation

`set` operations are a very powerful feature. As you can see in the code example they can save you a lot of code / looping.

You want to have this trick up your sleeve, so practice the linked exercise below!

Resources

https://docs.python.org/3.8/library/stdtypes.html#set-types-set-frozenset

Exercise

Bite 78. Find programmers with common languages

75. Freeze a portion of a function

Python's `functools.partial` lets you put a basic wrapper around an existing function:

```
>>> from functools import partial
>>> print_no_newline = partial(print, end=', ')
>>> for _ in range(3): print('test')
...
test
test
test
>>> for _ in range(3): print_no_newline('test')
...
test, test, test, >>>
```

Explanation

Python's `functools.partial` lets you put a basic wrapper around an existing function so that you can set a default value where there normally wouldn't be one.

Here we make our own `print` defaulting the `end` keyword to a comma (overwriting `print`'s default of adding a newline (`\n`) to the end).

So this is a nice way to make a "shortcut" if you always call a function with the same arguments.

Resources

https://docs.python.org/3/library/functools.html#functools.partial

Exercise

Bite 172. Having fun with Python Partials

229. Testing floating point numbers

Sometimes you need a bit of tolerance in your tests, for example when dealing with `float`s:

```
$ more script.py
def sum_numbers(*numbers):
    return sum(numbers)

$ more test.py
from script import sum_numbers

def test_sum_numbers_ints():
    assert sum_numbers(1, 2, 3) == 6

def test_sum_numbers_floats():
    assert sum_numbers(0.1, 0.2) == 0.3  # uh-oh

$ pytest test.py
...
E       assert 0.30000000000000004 == 0.3
E        +  where 0.30000000000000004 = sum_numbers(0.1, 0.2)
...
1 failed, 1 passed in 0.06s

$ more test.py
from pytest import approx
...
def test_sum_numbers_floats():
    assert sum_numbers(0.1, 0.2) == approx(0.3)  # this passes
```

Explanation

`pytest`'s `approx` asserts that two numbers (or two sets of numbers) are equal to each other within some tolerance. Here we see a good example of `float`'s inherent imprecision and the trouble it may cause in testing. But no worries, `approx` asserts that `0.30000000000000004` equals `0.3`.

Resources

https://docs.pytest.org/en/latest/reference.html#pytest-approx
https://cs50.stackexchange.com/a/15645

241. Create an entry point to your package

Adding a `__main__.py` file to your package you can call it with `python -m`:

```
# given this simple package:
$ cat my_package/file_1.py
def add_two_numbers(a, b):
    return a + b


# we cannot run it as a package:
$ python -m my_package
... No module named my_package.__main__; 'my_package' is a package and cannot
be directly executed


# adding a __main__.py you can add an entry point to your package:
$ cat my_package/__main__.py
from . import file_1

def foo():
    print(file_1.add_two_numbers(3, 4))

if __name__ == '__main__':
    foo()


# now you can run the package like this:
$ python -m my_package
7
```

Explanation

Similarly to the `if __name__ == "__main__":` entry point for a script (see Tip #46), you can create an entry point to your package by adding a `__main__.py` module to it, making it callable using: `python -m my_package`.

Another way is to add the `entry_points` keyword argument to `setuptools.setup()` in `setup.py` (or `[tool.poetry.scripts]` in `pyproject.toml` if you use `poetry`).

Explanation

https://docs.python.org/3/using/cmdline.html#cmdoption-m
https://python-packaging.readthedocs.io/en/latest/command-line-scripts.html

247. Make a retry decorator (with optional argument)

Here we make a `retry` decorator that tries to call a function N times before giving up:

```
>>> from functools import wraps, partial
>>> import requests
>>> def retry(func=None, *, times=3):
...     if func is None:
...         return partial(retry, times=times)
...     @wraps(func)
...     def wrapper(*args, **kwargs):
...         attempt = 0
...         while attempt < times:
...             try:
...                 return func(*args, **kwargs)
...             except Exception as exc:
...                 attempt += 1
...                 print(f"Exception {func}: {exc} (attempt: {attempt})")
...         return func(*args, **kwargs)
...     return wrapper
>>> @retry  # or: @retry(times=<int>)
... def get(url):
...     resp = requests.get(url)
...     resp.raise_for_status()
>>> get('https://httpbin.org/status/200')
>>> get('https://httpbin.org/status/404')
Exception <function get at 0x7fb4592dc280>: 404 Client Error: NOT FOUND ...
Exception <function get at 0x7fb4592dc280>: 404 Client Error: NOT FOUND ...
Exception <function get at 0x7fb4592dc280>: 404 Client Error: NOT FOUND ...
...
requests.exceptions.HTTPError: 404 Client Error: NOT FOUND ...
```

Explanation

Here we try to call an endpoint using the `requests` module (Tip #39). If it raises an exception (using `raise_for_status()`, see Tip #236), it tries again, up till `times` attempts. We use `partial` (Tip #75) to have the decorator accept an optional argument. This is quite mind-blowing code which took us various attempts. Luckily we stumbled upon this recipe in Python Cookbook 3rd ed (see all attempts in the article below).

Resources

https://pybit.es/decorator-optional-argument.html

This was only 4% …

We hope you enjoyed these 10 free tips.

To get the other **240 real world Python tips**, buy our book using the link in the post you got this sample ebook from ...

Thanks,
Bob & Julian