Python Tips

Your daily dose of bite sized python tips

Nifty Python tricks

Hi there folks. It's been a long time since I last published a post. I have been busy. However in this post I am going to share some really informative tips and tricks which you might not have known about. So without wasting any time lets get straight to them:

Enumerate

Instead of doing:

```
i = 0
for item in iterable:
    print i, item
    i += 1
```

We can do:

```
for i, item in enumerate(iterable):
    print i, item
```

Enumerate can also take a second argument. Here is an example:

```
>>> list(enumerate('abc'))
[(0, 'a'), (1, 'b'), (2, 'c')]
```

```
>>> list(enumerate('abc', 1))
[(1, 'a'), (2, 'b'), (3, 'c')]
```

Dict/Set comprehensions

You might know about list comprehensions but you might not be aware of *dict/set comprehensions*. They are simple to use and just as effective. Here is an example:

```
my_dict = {i: i * i for i in xrange(100)}
my_set = {i * 15 for i in xrange(100)}

# There is only a difference of ':' in both
```

Forcing float division:

If we divide whole numbers Python gives us the result as a whole number even if the result was a float. In order to circumvent this issue we have to do something like this:

```
result = 1.0/2
```

But there is another way to solve this problem which even I wasn't aware of. You can do:

```
from __future__ import division
result = 1/2
# print(result)
# 0.5
```

Voila! Now you don't need to append *.0* in order to get an accurate answer. Do note that this trick is for Python 2 only. In Python 3 there is no need to do the import as it handles this case by default.

Simple Server

Do you want to quickly and easily share files from a directory? You can simply do:

```
# Python2
python -m SimpleHTTPServer

# Python 3
python3 -m http.server
```

This would start up a server.

Evaluating Python expressions

We all know about *eval* but do we all know about *literal_eval*? Perhaps not. You can do:

```
import ast
my_list = ast.literal_eval(expr)
```

Instead of:

```
expr = "[1, 2, 3]"
my_list = eval(expr)
```

I am sure that it's something new for most of us but it has been a part of Python for a long time.

Profiling a script

You can easily profile a script by running it like this:

```
python -m cProfile my_script.py
```

Object introspection

You can inspect objects in Python by using *dir()*. Here is a simple example:

```
>>> foo = [1, 2, 3, 4]
>>> dir(foo)
['__add__', '__class__', '__contains__',
'__delattr__', '__delitem__', '__delslice__', ... ,
'extend', 'index', 'insert', 'pop', 'remove',
'reverse', 'sort']
```

Debugging scripts

You can easily set breakpoints in your script using the *pdb* module. Here is an example:

```
import pdb
pdb.set_trace()
```

You can write *pdb.set_trace()* anywhere in your script and it will set a breakpoint there. Super convenient. You should also read more about pdb as it has a couple of other hidden gems as well.

Simplify if constructs

If you have to check for several values you can easily do:

```
if n in [1,4,5,6]:
```

instead of:

```
if n==1 or n==4 or n==5 or n==6:
```

Reversing a list/string

You can quickly reverse a list by using:

```
>>> a = [1,2,3,4]
>>> a[::-1]
[4, 3, 2, 1]

# This creates a new reversed list.
# If you want to reverse a list in place you can do:

a.reverse()
```

and the same can be applied to a string as well:

```
>>> foo = "yasoob"
>>> foo[::-1]
'boosay'
```

Pretty print

You can print dicts and lists in a beautiful way by doing:

```
from pprint import pprint
pprint(my_dict)
```

This is more effective on dicts. Moreover, if you want to pretty print json quickly from a file then you can simply do:

```
cat file.json | python -m json.tools
```

Ternary Operators

Ternary operators are shortcut for an if-else statement, and are also known as a conditional operators. Here are some examples which you can use to make your code compact and more beautiful.

```
[on_true] if [expression] else [on_false]
x, y = 50, 25
small = x if x < y else y
```

Thats all for today! I hope you enjoyed this article and picked up a trick or two along the way. See you in the next article. Make sure that you follow us on Facebook and Twitter!

Do you have any comments or suggestions? You can write a comment or email me on yasoob.khld (at) gmail.com

RELATED:

The Python yield keyword explained
In "python"

***args and **kwargs in python explained**
In "python"

Learning Python For Data Science
In "python"

April 19, 2015 Yasoob python list comprehensions, object introspection, pdb, python, reversing string, set comprehensions

17 thoughts on "Nifty Python tricks"

GuieA_7

April 19, 2015 at 2:51 pm

>>> a = [1,2,3,4]
>>> a[::-1]
It does not reverse the list, it creates a new list which is reversed. The 'list' class has a reverse() method for that purpose.
(but if 'a' is a string — strings are immutable — the a[::-1] way is quite good)

bilalahmedawan

February 21, 2018 at 12:19 pm

do a = a[::-1] problem solved, lol

RavingNoah

April 19, 2015 at 3:18 pm

Hey, these are great tidbits. :: *hard clapping* ::

Yash Bathia

April 19, 2015 at 7:53 pm

Good tips..Thanks

tim

April 19, 2015 at 10:27 pm

Curious that you're mentioning ast.literal_eval without any reason. It's actually a much more limited eval and is not designed to evaluate general Python. The ast module is for modeling Python's syntax. Try a few statements with literal_eval; you will not get far.

Gunther Klessinger

April 28, 2015 at 7:12 pm

The limits ARE the feature: We use lit.eval all the time for *secure* (de)serializiation of structures into strings and back. The reason we prefer compared to json is this:
>>> 'é' in json.loads(json.dumps(['é']))
False
>>> 'é' in literal_eval(str(['é']))
True

Pawel

April 20, 2015 at 6:40 pm

you must use range instead of xrange for python3

Ben (@deisum)

April 21, 2015 at 12:17 pm

a[::-1] slicing is obtuse and made obsolete by the `reversed` builtin:
https://docs.python.org/2/library/functions.html#reversed

Joel

April 27, 2015 at 5:38 pm

Thank you, love the tips especially enumerate, will save me some lines!

kaisk

May 2, 2015 at 8:21 pm

I think we should use `{1,4,5,6}` instead of `[1,4,5,6]` for `in` operator because `set` can access each element by O(1).

Jim Mooney

May 4, 2015 at 10:30 am

You can even use a ternary operator in a list comprehension:
[x**2 if x > 10 else x**4 for x in range(40)]

Luk

May 20, 2015 at 12:45 am

Very good! Thank you 🙂

Pingback: Interesting Programming Links | Jack Simpson

itsmyalter

June 9, 2015 at 9:45 pm

Great work

peacengell

November 18, 2015 at 5:03 pm

Lovely site and informative.

Thanks a lot for your love of sharing.

Jay Mee

May 22, 2016 at 5:22 pm

How do i put an output of last python command into a var for example url = (scheme, resultid[0]) (gives error ??

Mark Montgomery

March 18, 2017 at 1:08 am

On simple if/else statements like shown under Ternary Operators, I like to use indexing where the test returns 0 (False) or 1 (True). So you could represent as: small = (y,x)[x<y]
