Python (programming language) Programming Languages +1

What are some hidden features of Python?

Would love to hear about less-known, cool Python features :)

Answer Request Follow 722 Comment Share 8 Downvote •••

44 Answers

Jonathan Goldsmith, pythonista-in-training
27.4k Views

StackOverflow has a very good compilation of hidden features here: Hidden features of Python

Here are some of my favorites:

For .. else syntax

```
for i in foo:
    if i == 0:
        break

else: print("i was never 0")
```

Decorators

Wrap a function in another function to add meta-functionality. I use this all the time to memoize highly recursive functions:

```
class memoize:
    def __init__(self, f):
        self.f = f
        self.dict = {}
    def __call__(self, *args):
        if not args in self.dict:
            self.dict[args] = self.f(*args)
        return self.dict[args]

@memoize
def fib(n):
    if n < 2:
        return 1
    return fib(n-1) + fib(n-2) #turn expensive calculations into dict lo
```

get() in Dictionaries

If d['key'] does not exist then calling it will raise an exception. d.get('key') will return None if 'key' doesn't exist, and you can also pass in a value to return instead of none, e.g. `d.get('key', 0)`

'this' module

Probably Python's best known 'hidden' feature is the this module. Try typing `import this` into an interactive shell and it will print:

> The Zen of Python, by Tim Peters
>
> Beautiful is better than ugly.
> Explicit is better than implicit.
> Simple is better than complex.

Related Questions

Where can I learn the hidden features of python?

What are the Lisp-like features of Ruby and Python?

Hidden (not wildly known) features of jQuery?

What are some of the cool features of Python which most people don't know?

What are your favorite tricks or hidden features of OCaml that you use?

What are some hidden (or not so known) features of Django?

What are the hidden truths of Python programming language?

What are the major "must know," "should know," and "nice to know" features of programming in Python?

In what ways are Lisp's features better for dealing with functions beyond what Python has?

What are the features of Python?

More Related Questions

Question Stats

722 Followers
323,415 Views
Last Asked Apr 19
Edits

> Complex is better than complicated.
> Flat is better than nested.
> Sparse is better than dense.
> Readability counts.
> Special cases aren't special enough to break the rules.
> Although practicality beats purity.
> Errors should never pass silently.
> Unless explicitly silenced.
> In the face of ambiguity, refuse the temptation to guess.
> There should be one-- and preferably only one --obvious way to do it.
> Although that way may not be obvious at first unless you're Dutch.
> Now is better than never.
> Although never is often better than *right* now.
> If the implementation is hard to explain, it's a bad idea.
> If the implementation is easy to explain, it may be a good idea.
> Namespaces are one honking great idea -- let's do more of those!

Written Mar 24, 2013 · View Upvotes

Michael Staniek, Hobby Python and Java Programmer
9.1k Views · Michael has 90+ answers in Python (programming language)

Hello,

thanks for the A2A. I always wanted to write something here, but nearly all my knowledge about hidden things comes from this topic, sooo everything I knew probably isnt that much help.

But now I know some things I can talk about (focussing on things not written here already!), mainly for all those guys and gals out there that want to optimize the hell out of their code and other unknown stuff

Disassemble Python

Ever thought about what python does under the hood? With the standard library module dis, you can do that easily!

```
import dis
def test(number):
    return (str(number)+str(number))
dis.dis(test)

Result:
  8           0 LOAD_GLOBAL              0 (str)
              3 LOAD_FAST                0 (number)
              6 CALL_FUNCTION            1 (1 positional, 0 keyword pair
              9 LOAD_GLOBAL              0 (str)
             12 LOAD_FAST                0 (number)
             15 CALL_FUNCTION            1 (1 positional, 0 keyword pair
             18 BINARY_ADD
             19 RETURN_VALUE
#can we speed it up?
```

Type hints

Everyone and their mum knows that Python is dynamically typed. But since a short while ago (some will probably know that) you can add type hints to your functions, and then use extern static type checking tools (like mypy) to check your code, without even running it.

PEP 484 -- Type Hints

And the best: You CAN use it, but it wont be enforced to you. Take that, enemies of dynamically typed languages.

Transposing a Matrix

Read that somewhere on Quora, dont know where sadly, but you can EASILY transpose an array made up of lists (no numpy) with the zip operation:

```
matrix=[(1,2,3) , (4,5,6), (7,8,9)]
for row in matrix:
    print(row)
```

```
(1, 2, 3)
(4, 5, 6)
(7, 8, 9)

matrix2=zip(*matrix)
print()
for row in matrix2:
    print(row)

(1, 4, 7)
(2, 5, 8)
(3, 6, 9)
```

cProfile

Python has many scripts that are very useful. For example you can easily profile your code by starting your code like this:

python -m cProfile filename

With this, you then get an analysis of which function take how long, so you can always optimize the longest running function.

It will then look like this:

```
        807 function calls in 2.459 seconds

  Ordered by: standard name

  ncalls  tottime  percall  cumtime  percall filename:lineno(function)
       1    0.000    0.000    2.459    2.459 p.py:1(<module>)
       1    0.000    0.000    2.459    2.459 p.py:12(main)
     200    0.002    0.000    0.227    0.001 p.py:2(fast)
     200    0.002    0.000    2.027    0.010 p.py:4(slow)
       2    0.004    0.002    2.459    1.229 p.py:6(very_slow)
       1    0.000    0.000    0.000    0.000 {method 'disable' of '_lsp
     402    2.451    0.006    2.451    0.006 {time.sleep}
```

Found on google

Let me know if I could help you.

Greetings

Written Apr 19 · View Upvotes · Answer requested by George Va and Viveksai Ramagiri

Upvote | 85 Downvote Comments 2+

Bahrom Matyakubov, 2+ years of Python development
6.8k Views

One of the neat features I've seen but haven't personally used yet, is that `in` is actually syntactic sugar for `__contains__`.

For example:

```
class EvenNumbers:
    def __contains__(self, x):
        return not x%2

print(2 in EvenNumbers())
```

As I've mentioned earlier, I have yet to find a practical use for this, but I'm sure it has its use cases.

Edit:

I am going to add a few for personal reference as I come across them. Disclaimer, there may be duplicates from earlier answers - let me know if I forget to say that.

1. Just discovered clearing python shell from commandline - Python: Clear screen in shell.
2. As mentioned in one of the answers above, `_` is the result of the last shell command.

Updated Jan 7 · View Upvotes

Upvote | 42 Downvote Comment 1

Manuel Schipper, I like Python.
28.6k Views

There is some nice stuff already mentioned here, here's my 0.02$:

Enumerate

A lot of code like this:

```
animals = ['cat', 'dog', 'cow']
for i in range(len(animals)):
    print i, animals[i]
```

is better made like this:

```
animals = ['cat', 'dog', 'cow']
for count, animal in enumerate(animals):
    print count, animal
```

String methods

A lot of code is made using the regular expression module when the built-in string methods would suffice. Take a look at the replace method:

```
>>> cars = ['Red_sedan', 'Big_truck', 'Small_roadster']
>>> for car in cars:
...     car.replace('_', ' ')
...
'Red sedan'
'Big truck'
'Small roadster'
```

And partition:

```
>>> cars = ['Red_sedan', 'Big_truck', 'Small_roadster']
>>> for car in cars:
...     head, sep, tail = car.partition('_')
...     print tail
...
sedan
truck
roadster
```

And startswith:

```
>>> cars = ['Red_sedan', 'Big_truck', 'Small_roadster']
>>> for car in cars:
...     car.startswith('Big')
...
False
True
False
```

It's surprising how little I see these in production.

Collections module

This module can help solve a lot of problems. Abhijith Reddy D already mentioned the defaultdict which is very useful. Another pretty nice one is Counter. Take a look at these examples from the docs before you implement your own.

```
>>> # Tally occurrences of words in a list
>>> cnt = Counter()
>>> for word in ['red', 'blue', 'red', 'green', 'blue', 'blue']:
...      cnt[word] += 1
>>> cnt
Counter({'blue': 3, 'red': 2, 'green': 1})

>>> # Find the ten most common words in Hamlet
>>> import re
>>> words = re.findall(r'\w+', open('hamlet.txt').read().lower())
>>> Counter(words).most_common(10)
[('the', 1143), ('and', 966), ('to', 762), ('of', 669), ('i', 631), ('you
```

Comprehensions

List comprehensions are great and commonly used to make a list like this:

```
>>> squares = list()
```

```
>>> for i in range(5):
...     squares.append(i**2)
```

in a one liner like this:

```
>>> squares = [i**2 for i in range(5)]
```

In Python 3 there is also Set and Dictionary comprehensions.

Take a look at this example found here where key value pairs are swapped with a Dictionary comprehension:

```
>>> a_dict = {'a': 1, 'b': 2, 'c': 3}
>>> {value:key for key, value in a_dict.items()}
{1: 'a', 2: 'b', 3: 'c'}
```

Cheers!

Written Dec 23, 2015 · View Upvotes

Upvote | 241 Downvote Comments 5+

Vishesh Yadav
5.7k Views

I'm not aware of any *real* hidden stuff in Python. These are some small things in Python that are relatively less known I guess and are worth looking at -

- Built-in method type() in Python. It not only lets you check the type of object, but also has some other super powers. It lets you create classes on the fly. Check out this answer on StackOverflow - What is a metaclass in Python?
- namedtuple and Counter from stdlibs's collections modules are quite new and cool. Both are useful and there is good chance that you might be reinventing the wheel, particularly for Counter. (Its not a core language feature, but still I thought its worth mentioning)
- Probably not less known, but I was unaware of this for years I guess -

```
# you can do stuff like
a < b < c < d < e . . .
```

- set/frozenset has some cool binary operators.

Written Nov 9, 2013 · View Upvotes

Upvote | 9 Downvote Comments 2+

Simon Willison, Architecture at Eventbrite / Co-founder of Lanyrd.com
9.2k Views · Simon has 90+ answers in Computer Programming

Generators and Iterators are pretty amazing. These two tutorials will really open your eyes as to how powerful they can be:

- Generator Tricks for Systems Programmers
- A Curious Course on Coroutines and Concurrency

Both by David Beazley. To my mind they are the most fascinating Python tutorials out there.

Written Mar 26, 2013 · View Upvotes

Upvote | 41 Downvote Comment

Pravesh Jain, Worked in Java, Python, Scala, C++
2.4k Views

Here are some of the lesser known features that I came to know about:

- **Python dictionaries have a get() method**

  The method **get()** returns a value for the given key. If key is not available then returns default value None. It allows you to provide a default value if the key is missing:

```
dictionary.get("bogus", None)
```

returns `None`, whereas

```
dictionary["bogus"]
```

would raise a keyerror.

- **Conditional Assignment**

```
x = 3 if (y == 1) else 2
```

It does exactly what it looks like. It means "Assign 3 to x if y is 1 else assign 2 to x". Looks so much more elegant than writing the if-else block.

You can also chain such statements

```
x = 3 if (y == 1) else 2 if (y == -1) else 1
```

Although the chained version doesn't have the same catchiness to it.

- **Formatting a String**

Python provides a very simple way to format a string using arguments.

```
print("The {foo} is {bar}".format(foo='answer', bar=42))
```

The readability of this is infinitely higher than making a large string by concatenating smaller strings with variables.

- **Enumerate**

The **enumerate()** function adds a counter to an iterable.

So for each element in cursor, a tuple is produced with (counter, element); the for loop binds that to row_number and row, respectively.

```
for count, elem in enumerate(elements):
  print count, elem
```

outputs

```
0 foo
1 bar
2 baz
```

- **Handle missing keys from a Python Dictionary**

  If a subclass of dict defines a method **__missing__()** and *key* is not present, the d[key] operation calls that method with the key *key* as argument. The d[key] operation then returns or raises whatever is returned or raised by the __missing__(key) call. No other operations or methods invoke __missing__(). If __missing__() is not defined, KeyError is raised. __missing__() must be a method; it cannot be an instance variable.

```
class Counter(dict):
  def __missing__(self, key):

c[1]
```


outputs
`0`

This eliminates the need for checking dictionary keys before accessing them.

Source : Hidden features of Python

Written Nov 3, 2015 · View Upvotes

Upvote | 20 Downvote Comment 1

Chris Riederer, Computer Science PhD student, Minnesotan
2.7k Views

Tons of great answers here. I didn't see anyone mention "reduce" which I feel isn't well known and can be fun to use.

Suppose you have a giant file where each line is a JSON object, and you want a set of all categories. Here's one fun way to do it using reduce:

```
import json

categories = reduce(lambda x, y: x.union(y), (json.loads(line)['categori
```

Written Nov 23, 2014 · View Upvotes

Upvote | 8 Downvote Comments 1+

Swarun Krishna, Barnabus
3.2k Views

Chained comparisons :
You can chain multiple comparisons into a single statement, much like you would do in real life.

```
>> x=3
>> 0 < x < 5 < 2*x < 10
>> True
```

Standard operator precedence rules apply.

Written Dec 12, 2014 · View Upvotes

Upvote | 17 Downvote Comment

Lucian Nutiu, Underprivileged white person
3.4k Views

__slots__
If you don't know what this is, read more here

__slots__ in Python: Save some space and prevent member variable additions

Even if Guido recommends to avoid them, I find the mechanism very useful for enforcing a fixed set of properties for a class, because sometimes the (too) dynamic behaviour of Python is counter-productive.

Read more controversy here:
Python __slots__

Written Mar 26, 2013 · View Upvotes

Upvote | 5 Downvote Comment


```
import __hello__
```

or

```
import __phello__
```

The itertools module is also pretty cool. It "implements a number of iterator building blocks inspired by constructs from APL, Haskell, and SML." The functions available are fast, memory efficient, and make list comprehension less of a syntactic mess. Here are some recipes from its docs:

```
def take(n, iterable):
    "Return first n items of the iterable as a list"
    return list(islice(iterable, n))

def ncycles(iterable, n):
    "Returns the sequence elements n times"
    return chain.from_iterable(repeat(tuple(iterable), n))

def dotproduct(vec1, vec2):
    return sum(imap(operator.mul, vec1, vec2))

def flatten(listOfLists):
    "Flatten one level of nesting"
    return chain.from_iterable(listOfLists)

def powerset(iterable):
    "powerset([1,2,3]) --> () (1,) (2,) (3,) (1,2) (1,3) (2,3) (1,2,3)"
    s = list(iterable)
    return chain.from_iterable(combinations(s, r) for r in range(len(s)+
```

And then there's this (no pun intended):

```
>>> import this
...
# The Zen of Python, by Tim Peters
...
>>> this is True
False
>>> this is not False
True
```

Written Jan 4 · View Upvotes

Upvote | 15 Downvote Comment 1

David Rutter, Python programmer
1k Views

Here is a short piece of code I helped write which prints the nth order "abacaba" sequence along with a bar graph where the length of each bar is the index of the letter in the alphabet:

```
f=lambda n,s=1:n*"'"and"  _"*s+f(n-1,0)+"_"*(n-2)+"\n%c|%s|"%
(64+n,"_"*n)+f(n-1,0)
```

There's a couple of nice "hidden" features of python here:

- Calling a named lambda expression recursively from within that expression is totally

allowed.

- Lambda expressions can use default arguments!
- The % string formatting syntax can automatically convert a character code to a character with %c
- "and" and "or" are not binary logic operators, but rather selectors which return one of their arguments based only on their evaluation of the first argument. This is not only fast, but also makes it possible to use them in the place of conditionals in situations like this. (Here, they allow the base case to live in the same expression as the recursive calls without use of Python's annoying ternary operator.)
- Empty strings evaluate logically as false, so the same expression can be used as a conditional and a return string (when using "and" and "or" as selectors).

Do note that writing code that resembles the above is very much against the intent and spirit of Python, as is, probably, exploiting many of the above "features".

Written Apr 19 · View Upvotes