Quick Links for Python
Recent Articles
MCQ / Quizzes
Practice Problems

Basics
Introduction
New Generation Language
Keywords, Set 1 Set 2
Explore More...
Variables
Variables,Expressions & Functions
Global and Local Variables
Type Conversion
Explore More...
Operators
Increment and Decrement Operator
Teranry Operator & Divison Operator
Logical and Bitwise Not Operators on Boolean
Any & ALL

Operator Functions Set 1 & Set 2
Data Types
Introduction
Arrays Set 1, Set 2
String Methods Set 1, Set 2, Set 3
String Template Class & String Formatting using %
List Methods Set 1, Set 2, Set 3
Tuples & Sets
Dictionary Methods Set 1, Set 2
ChainMap
Explore More...
Control Flow
Loops and Control Statements
Counters & Accessing Counters
Iterators & Iterator Functions Set 1, Set 2
Generators
Explore More...
Functions
Function Decorators
Returning Multiple Values
Yield instead of Return
Python Closures & Coroutine
Explore More...
Modules
Introduction

Numeric Functions & Logarithmic and Power functions
Calender Functions Set 1, Set 2
Complex Numbers Introduction & Important functions
Explore More...
Object Oriented Concepts
Class, Object and Members
Data Hiding and Object Printing
Inheritance, Subclass and super
Class method vs static method & Class or Static Variables
Explore More...
Exception Handling
Exception Handling
User-Defined Exceptions
Built-in Exceptions
Libraries and Functions
Timeit
Numpy Set 1, Set 2
Get and Post
import module & reload module
Collection Modules Deque, Namedtuple & Heap
Explore More...
Machine Learning with Python
Classifying data using Support Vector Machines(SVMs) in Python

K means Clustering
How to get synonyms/antonyms from NLTK WordNet in Python?
Explore More...
Misc
Sql using Python & MongoDB and Python
Json formatting & Python Virtual environment
Metaprogramming with Metaclasses in Python
Python Input Methods for Competitive Programming
Explore More...
Applications and Projects
Creating a proxy webserver Set 1, Set 2
Send Messsage to FB friend
Twitter Sentiment Analysis & Whatsapp using Python
Desktop Notifier & Junk File Organizer
Explore More...

Optimization Tips for Python Code

In this article, some interesting optimization tips for Faster Python Code are discussed. These techniques help to produce result faster in a python code.

1. **Use builtin functions and libraries:** Builtin functions like map() are implemented in C code. So the interpreter doesn't have to execute the loop, this gives a considerable speedup.
   The map() function applies a function to every member of iterable and returns the result. If there are multiple arguments, map() returns a list consisting of tuples containing the corresponding items from all iterables.

   ```
   # Python program to illustrate library functions
   ```

```
# save time while coding with the example of map()
import time

# slower (Without map())
start = time.clock()
s = 'geeks'
U = []
for c in s:
    U.append(c.upper())
print U
elapsed = time.clock()
e1 = elapsed - start
print "Time spent in function is: ", e1

# Faster (Uses builtin function map())
s = 'geeks'
start = time.clock()
U = map(str.upper, s)
print U
elapsed = time.clock()
e2 = elapsed - start
print "Time spent in builtin function is: ", e2
```

Run on IDE

```
['G', 'E', 'E', 'K', 'S']
Time spent in function is:  0.0394747945637
['G', 'E', 'E', 'K', 'S']
Time spent in builtin function is:  0.0212335531192
```

The packages are platform-specific, which means that we need the appropriate package for the platform we're using. If we are doing string operation, consider using an existing module 'collections' like deque which is highly optimized for our purposes.

```
# Python program to illustrate
# importing list-like container with
# fast appends and pops on either end
from collections import deque
s = 'geek'

# make a new deque
d = deque(s)

# add a new entry to the right side
d.append('y')

# add a new entry to the left side
d.appendleft('h')
print d

d.pop() # return and remove the rightmost item

d.popleft() # return and remove the lefttmost item

# print list deque in reverse
print list(reversed(d))
```

Run on IDE

```
deque(['h', 'g', 'e', 'e', 'k', 'y'])
['k', 'e', 'e', 'g']
```

```
# importing iteration tools
import itertools
iter = itertools.permutations([1,2,3])
print list(iter)
```

Run on IDE

Output:

```
[(1, 2, 3), (1, 3, 2), (2, 1, 3), (2, 3, 1), (3, 1, 2), (3, 2, 1)]
```

2. **Use keys for sorts:** In Python, we should use the key argument to the built-in sort instead, which is a faster way to sort.

```
# Python program to illustrate
# using keys for sorting
somelist = [1, -3, 6, 11, 5]
somelist.sort()
print somelist

s = 'geeks'
# use sorted() if you don't want to sort in-place:
s = sorted(s)
print s
```

Run on IDE

Output:

```
[-3, 1, 5, 6, 11]
['e', 'e', 'g', 'k', 's']
```

In each case the list is sorted according to the index you select as part of the key argument. This approach works just as well with strings as it does with numbers.

3. **Optimizing loops:** Write idiomatic code: This may sound counter-intuitive but writing idiomatic code will make your code faster in most cases. This is because Python was designed to have only one obvious/correct way to do a task.
For example (String Concatenation):

```
# Python program to illustrate using
# optimized loops for faster coding

# slow O(n^2) - ( Note: In latest implementations it is O(n) )
s = 'hellogeeks'
slist = ''
for i in s:
```

```
    slist = slist + i
print slist

# string concatenation (idiomatic and fast O(n))
st = 'hellogeeks'
slist = ''.join([i for i in s])
print slist

# Better way to iterate a range
evens = [ i for i in xrange(10) if i%2 == 0]
print evens

# Less faster
i = 0
evens = []
while i < 10:
    if i %2 == 0: evens.append(i)
        i += 1
print evens

# slow
v = 'for'
s = 'geeks ' + v + ' geeks'
print s

# fast
s = 'geeks %s geeks' % v
print s
```

Run on IDE

```
hellogeeks
[0, 2, 4, 6, 8]
geeks for geeks
```

Every time running a loop to s(i), Python evaluates the method. However, if you place the evaluation in a variable, the value is already known and Python can perform tasks faster.

4. **Try multiple coding approaches**: Using precisely the same coding approach every time we create an application will almost certainly result in some situations where the application runs slower than it might.

   For example (Initializing Dictionary Elements):

```
# Python program to illustrate trying
# multiple coding approaches
# for getting faster result
# slower
mydict = {'g':1,'e':1,'e':1,'k':1}
word = 'geeksforgeeks'
for w in word:
    if w not in mydict:
        mydict[w] = 0
    mydict[w] += 1
print mydict

# faster
mydict = {'g':1,'e':1,'e':1,'k':1}
```

```
word = 'geeksforgeeks'
for w in word:
    try:
        mydict[w] += 1
    except KeyError:
        mydict[w] = 1
print mydict
```

Run on IDE

```
{'e': 5, 'g': 3, 'f': 1, 'k': 3, 'o': 1, 's': 2, 'r': 1}
```

The output is the same in both cases. The only difference is how the output is obtained.

5. **Use xrange instead of range:**range() – This returns a list of numbers created using range() function. xrange() – This function returns the generator object that can be used to display numbers only by looping. Only particular range is displayed on demand and hence called "lazy evaluation".

```
# slower
x = [i for i in range(0,10,2)]
print x

# faster
x = [i for i in xrange(0,10,2)]
print x
```

Run on IDE

```
[1, 3, 5, 7, 9]
```

This could save you system memory because xrange() will only yield one integer element in a sequence at a time. Whereas range(), it gives you an entire list, which is unnecessary overhead for looping.

6. **Use Python multiple assignment to swap variables:** This is elegant and faster in Python.

```
# Python program to illustrate swapping
# of a variable in one line

# slower
x = 2
y = 5
temp = x
x = y
y = temp
print x,y

x,y = 3,5
# faster
x, y = y, x
print x,y
```

Run on IDE

```
5 2
5 3
```

7. **Use local variable if possible:** Python is faster retrieving a local variable than retrieving a global variable. That is, avoid the "global" keyword. So if you are going to access a method often (inside a loop) consider writing it to a variable.

```
# Python program to illustrate trying
# to use local variables to make code
# run faster
class Test:
    def func(self,x):
        print x+x

# Declaring variable that assigns class method object
Obj = Test()
mytest = Obj.func # Declaring local variable
n = 2
for i in range(n):
    mytest(i) # faster than Obj.func(i)
```

Run on IDE

```
0
2
```

References:

- StackOverflow
- Python.org

This article is contributed by **Afzal Ansari**. If you like GeeksforGeeks and would like to contribute, you can also write an article using contribute.geeksforgeeks.org or mail your article to contribute@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

GATE CS Corner **Company Wise Coding Practice**

Python Technical Scripter

Recommended Posts:

Printing "GEEKS FOR GEEKS" In Brainfuck
Understanding Code Reuse and Modularity in Python 3
OOP in Python | Set 3 (Inheritance, examples of object, issubclass and super)
Noble integers in an array (count of greater elements is equal to value)
Python Virtual Environment | Introduction

(Login to Rate and Mark)

2.8 Average Difficulty : **2.8/5.0**
Based on **6** vote(s)

Add to TODO List
Mark as DONE

Writing code in comment? Please use ide.geeksforgeeks.org, generate link and share the link here.

Load Comments Share this post!