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...

Py-Facts – 10 interesting facts about Python

Python is one of the most popular programming languages nowadays on account of its code readability and simplicity. All thanks to Guido Van Rossum, its creator.

I've compiled a list of 10 interesting Facts in the Python Language. Here they are:

1. There is actually a poem written by Tim Peters named as THE ZEN OF PYTHON which can be read by just writing import this in the interpreter.

```
# Try to guess the result before you actually run it
import this
```

Output:

Run on IDE

```
Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
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!
```

2. One can return multiple values in Python. Don't believe ? See the below code snippet:

```
# Multiple Return Values in Python!
def func():
   return 1, 2, 3, 4, 5

one, two, three, four, five = func()

print(one, two, three, four, five)
```

Run on IDE

Output:

```
(1, 2, 3, 4, 5)
```

3. One can use an "else" clause with a "for" loop in Python. It's a special type of syntax that executes only if the for loop exits naturally, without any break statements.

```
def func(array):
     for num in array:
        if num%2==0:
            print(num)
            break # Case1: Break is called, so 'else' wouldn't be executed.
     else: # Case 2: 'else' executed since break is not called
        print("No call for Break. Else is executed")

print("1st Case:")
a = [2]
func(a)
print("2nd Case:")
a = [1]
func(a)
```

Run on IDE

Output:

```
1st Case:
2
2nd Case:
No call for Break. Else is executed
```

4. In Python, everything is done by reference. It doesn't support pointers.

5. Function Argument Unpacking is another awesome feature of Python. One can unpack a list or a dictionary as function arguments using * and ** respectively. This is commonly known as the Splat operator. Example here

```
def point(x, y):
    print(x,y)

foo_list = (3, 4)
bar_dict = {'y': 3, 'x': 2}

point(*foo_list) # Unpacking Lists
point(**bar_dict) # Unpacking Dictionaries
```

Run on IDE

Output:

```
3 4
2 3
```

6. Want to find the index inside a for loop? Wrap an iterable with 'enumerate' and it will yield the item along with its index. See this code snippet

```
# Know the index faster
vowels=['a','e','i','o','u']
for i, letter in enumerate(vowels):
    print (i, letter)
```

Run on IDE

Output:

```
(0, 'a')
(1, 'e')
(2, 'i')
(3, 'o')
(4, 'u')
```

7. One can chain comparison operators in Python answer= 1<x<10 is executable in Python. More examples here

```
# Chaining Comparison Operators
i = 5;

ans = 1 < i < 10
print(ans)
```

```
ans = 10 > i <= 9
print(ans)

ans = 5 == i
print(ans)
```

Run on IDE

Output:

```
True
True
True
```

8. We can't define Infinities right? But wait! Not for Python. See this amazing example

```
# Positive Infinity
p_infinity = float('Inf')

if 99999999999999 > p_infinity:
    print("The number is greater than Infinity!")
else:
    print("Infinity is greatest")

# Negetive Infinity
n_infinity = float('-Inf')
if -99999999999999 < n_infinity:
    print("The number is lesser than Negative Infinity!")
else:
    print("Negative Infinity is least")
```

Run on IDE

Output:

```
Infinity is greatest
Negative Infinity is least
```

9. Instead of building a list with a loop, one can build it more concisely with a list comprehension. See this code for more understanding.

```
# Simple List Append
a = []
for x in range(0,10):
    a.append(x)
print(a)

# List Comprehension
print([x for x in a])
```

Run on IDE

Output:

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

10. Finally, Python's special Slice Operator. It is a way to get items from lists, as well as change them. See this code snippet

```
# Slice Operator
a = [1,2,3,4,5]

print(a[0:2]) # Choose elements [0-2), upper-bound noninclusive

print(a[0:-1]) # Choose all but the last

print(a[::-1]) # Reverse the list

print(a[::2]) # Skip by 2

print(a[::-2]) # Skip by -2 from the back
```

Run on IDE

Output:

```
[1, 2]
[1, 2, 3, 4]
[5, 4, 3, 2, 1]
[1, 3, 5]
[5, 3, 1]
```

This article is contributed by **Harshit Gupta**. If you like GeeksforGeeks and would like to contribute, you can also write an article and 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:

Generate all permutation of a set in Python
How to swap two variables in one line in C/C++, Python and Java?
JSON Formatting in Python
How to check if a string is a valid keyword in Python?
*args and **kwargs in Python

(Login to Rate and Mark)

2.3 Average Difficulty : **2.3/5.0**
Based on **12** 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!

 Contact Us! About Us! Careers! Privacy Policy