Python Archive

Python Language Introduction

Python is a widely used general-purpose, high level programming language. It was initially designed by Guido van Rossum in 1991 and developed by Python Software Foundation. It was mainly developed for emphasis on code readability, and its syntax allows programmers to express concepts in fewer lines of code.

Python is a programming language that lets you work quickly and integrate systems more efficiently.

There are two major Python versions- **Python 2 and Python 3**. Both are quite different.

Beginning with Python programming:

1) Finding an Interpreter:

Before we start Python programming, we need to have an interpreter to interpret and run our programs. There are certain online interpreters like **http://code.geeksforgeeks.org/**, http://ideone.com/ or http://codepad.org/ that can be used to start Python without installing an interpreter.

***Windows*:**There are many interpreters available freely to run Python scripts like IDLE (Integrated Development Environment) which is installed when you install the python software from **http://python.org/**

***Linux*:**For Linux, Python comes bundled with the linux.

2) Writing first program:

Following is first program in Python

```
# Script Begins

print("GeeksQuiz")

# Scripts Ends
```

Output:

```
GeeksQuiz
```

Let us analyze the script line by line.

***Line 1 :* [# Script Begins]** In Python comments begin with #. So this statement is for readability of code and ignored by Python interpretor.

Line 2 : [print("GeeksQuiz")] In a Python script to print something on the console print() function is used – it simply prints out a line (and also includes a newline unlike in C). One difference between Python 2 and Python 3 is the print statement. In Python 2, the "print" statement is not a function, and therefore can be invoked without a parenthesis. However, in Python 3, it is a function, and must be invoked with parentheses.

***Line 3 :* [# Script Ends]** This is just another comment like Line 1.

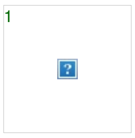
Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

GATE CS Notes (According to Official GATE 2017 Syllabus)

GATE CS Corner

See Placement Course for placement preparation, GATE Corner for GATE CS Preparation and Quiz Corner for all Quizzes on GeeksQuiz.
Category: Python Articles

Python – The new generation Language



Python designed by Guido van Rossum at CWI has become a widely used general-purpose, high-level programming language.

Prerequisites:

Knowledge of any programming language can be a plus.

Reason for increasing popularity

1. Emphasis on **code readability, shorter codes**, ease of writing
2. Programmers can express logical concepts in **fewer lines** of code in comparison to languages such as C++ or Java.
3. Python supports **multiple** programming paradigms, like object-oriented, imperative and functional programming or procedural.
4. There exists inbuilt functions for almost all of the frequently used concepts.
5. Philosophy is "Simplicity is the best".

LANGUAGE FEATURES

- **Interpreted**
  - There are no separate compilation and execution steps like C and C++.
  - Directly *run*the program from the source code.
  - Internally, Python converts the source code into an intermediate form called bytecodes which is then translated into native language of specific computer to run it.
  - No need to worry about linking and loading with libraries, etc.
- **Platform Independent**
  - Python programs can be developed and executed on multiple operating system platforms.
  - Python can be used on Linux, Windows, Macintosh, Solaris and many more.
- **Free and Open Source;** Redistributable
- **High-level Language**
  - In Python, no need to take care about low-level details such as managing the memory used by the program.
- **Simple**
  - Closer to English language;Easy to Learn
  - More emphasis on the solution to the problem rather than the syntax
- **Embeddable**
  - Python can be used within C/C++ program to give scripting capabilities for the program's users.
- **Robust**:
  - Exceptional handling features
  - Memory management techniques in built
- **Rich Library Support**
  - The Python Standard Library is vary vast.
  - Known as the **"batteries included"** philosophy of Python ;It can help do various things involving regular expressions, documentation generation, unit testing, threading, databases, web browsers, CGI, email, XML, HTML, WAV files, cryptography, GUI and many more.
  - Besides the standard library, there are various other high-quality libraries such as the Python Imaging Library which is an amazingly simple image manipulation library.

Python vs JAVA

Python	**Java**
Dynamically Typed1.No need to declare anything. An assignment statement binds a name to an object, and the object can be of any type.*2.*No type casting required when using container objects	**Statically Typed** 1.All variable names (along with their types) must be explicitly declared. Attempting to assign an object of the wrong type to a variable name triggers a type exception.*2.*Type casting is required when using container objects.
Concise Express much in limited words	**Verbose**Contains more words
Compact	**Less Compact**
Uses Indentation for structuring code	**Uses braces for structuring code**

The classical **Hello World program** illustrating the **relative verbosity** of a Java Program and Python Program

Java Code

```
public class HelloWorld
{
  public static void main (String[] args)
  {
    System.out.println("Hello, world!");
  }
}
```

Python Code

```
print "Hello, world!"
```

```
print("Hello, world!") # Python version 3
```

Similarity with Java

- Require some form of runtime on your system (JVM/Python runtime)
- Can probably be compiled to executables without the runtime (this is situational, none of them are designed to work this way)

LOOK and FEEL of the Python

2

GUI

Command Line interface

3

Currently, there are two versions of Python available **Python 2 and Python 3.** Many beginners must be wondering with which version of Python they should start. My answer to this question is usually something along the lines “just go with the version your favourite tutorial is written in, and check out the differences later on.”

Softwares making use of Python

Python has been successfully embedded in a number of software products as a scripting language.

1. GNU Debugger uses Python as a **pretty printer** to show complex structures such as C++ containers.
2. Python has also been used in artificial intelligence
3. Python is often used for **natural language processing** tasks.

Current Applications of Python

1. A number of Linux distributions use installers written in Python example in Ubuntu we have the **Ubiquity**
2. Python has seen extensive use in the **information security industry**, including in exploit development.
3. Raspberry Pi– single board computer uses Python as its principal user-programming language.
4. Python is now being used **Game Development** areas also.

Pros:

1. Ease of use
2. Multi-paradigm Approach

Cons:

1. Slow speed of execution compared to C,C++
2. Absence from mobile computing and browsers
3. For the C,C++ programmers switching to python can be irritating as the language requires proper indentation of code. Certain variable names commonly used like sum are functions in python. So C, C++ programmers have to look out for these.

Industrial Importance

Most of the companies are now looking for candidates who know about Python Programming. Those having the knowledge of python may have more chances of impressing the interviewing panel. So I would suggest that beginners should start learning python and excel in it.

GeeksforGeeks is very soon going to introduce programming in Python too.

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

Article By **Mudit Maheshwari:** **mudit**

A 3rd year B.Tech IT student from VIT University, Vellore having keen interest in coding , learning about new technology and developing softwares . Besides being passionate about coding, he also loves playing guitar and singing. Currently staying in Chennai. You can reach him at mudit94@gmail.com.

If you also wish to showcase your blog here,please see GBlog for guest blog writing on GeeksforGeeks.

GATE CS Corner   Company Wise Coding Practice

GBlog

Important differences between Python 2.x and Python 3.x with examples

- Division operator
- print function
- Unicode
- xrange
- Error Handling
- _future_ module

Division operator

If we are porting our code or executing the python 3.x code in python 2.x, it can be dangerous if integer division changes go unnoticed (since it doesn’t raise any error). It is preferred to use the floating value (like 7.0/5 or 7/5.0) to get the expected result when porting our code.

```
print 7 / 5
print -7 / 5

'''
Output in Python 2.x
1
```

```
-2
Output in Python 3.x :
1.4
-1.4

# Refer below link for details
# http://www.geeksforgeeks.org/division-operator-in-python/
'''
```

print function

This is the most well known change. In this the **print** function in Python 2.x is replaced by print() function in Python 3.x,i.e, to print in Python 3.x an extra pair of parenthesis is required.

```
print 'Hello, Geeks'      # Python 3.x doesn't support
print('Hope You like these facts')

'''
Output in Python 2.x :
Hello, Geeks
Hope You like these facts

Output in Python 3.x :
File "a.py", line 1
   print 'Hello, Geeks'
                      ^
SyntaxError: invalid syntax

Refer below link for details
http://www.geeksforgeeks.org/g-fact-25-print-single-multiple-variable-python/
'''
```

As we can see, if we use parenthesis in python 2.x then there is no issue but if we don't use parenthesis in python 3.x, we get SyntaxError.

Unicode:

In Python 2, implicit str type is ASCII. But in Python 3.x implicit str type is Unicode.

```
print(type('default string '))
print(type(b'string with b '))

'''
Output in Python 2.x (Bytes is same as str)
<type 'str'>
<type 'str'>

Output in Python 3.x (Bytes and str are different)
<class 'str'>
<class 'bytes'>
'''
```

Python 2.x also supports Unicode

```
print(type('default string '))
print(type(u'string with b '))

'''
Output in Python 2.x (Unicode and str are different)
<type 'str'>
<type 'unicode'>

Output in Python 3.x (Unicode and str are same)
<class 'str'>
<class 'str'>
'''
```

xrange:

xrange() of Python 2.x doesn't exist in Python 3.x. In Python 2.x, range returns a list i.e. range(3) returns [0, 1, 2] while xrange returns a xrange object i. e., xrange(3) returns iterator object which work similar to Java iterator and generates number when needed.
If we need to iterate over the same sequence multiple times, we prefer range() as range provides a static list. xrange() reconstructs the sequence every time. xrange() doesn't support slices and other list methods. The advantage of xrange() is, it saves memory when task is to iterate over a large range.

In Python 3.x, the range function now does what xrange does in Python 2.x, so to keep our code portable, we might want to stick to using range instead. So Python 3.x's range function *is* xrange from Python 2.x.

```
for x in xrange(1, 5):
   print(x),

for x in range(1, 5):
   print(x),

'''
Output in Python 2.x
1 2 3 4 1 2 3 4

Output in Python 3.x
NameError: name 'xrange' is not defined
'''
```

Error Handling:

There is a small change in error handling in both versions. In python 3.x, ‘as’ keyword is required.

```
try:
    trying_to_check_error
except NameError, err:
    print err, 'Error Caused'   # Would not work in Python 3.x

'''
Output in Python 2.x:
name 'trying_to_check_error' is not defined Error Caused

Output in Python 3.x :
File "a.py", line 3
    except NameError, err:
                    ^
SyntaxError: invalid syntax
'''
```

```
try:
     trying_to_check_error
except NameError as err: # 'as' is needed in Python 3.x
     print (err, 'Error Caused')

'''
Output in Python 2.x:
(NameError("name 'trying_to_check_error' is not defined",), 'Error Caused')

Output in Python 3.x :
name 'trying_to_check_error' is not defined Error Caused
'''
```

_future_module:

This is basically not a difference between two version, but useful thing to mention here. The idea of __future__ module is to help in migration. We can use Python 3.x
If we are planning Python 3.x support in our 2.x code,we can ise**_future_** imports it in our code.

For example, in below Python 2.x code, we use Python 3.x’s integer division behavior using __future__ module

```
# In below python 2.x code, division works
# same as Python 3.x because we use  __future__
from __future__ import division

print 7 / 5
print -7 / 5
```

Output :

```
1.4
-1.4
```

Another example where we use brackets in Python 2.x using __future__ module

```
from __future__ import print_function

print('GeeksforGeeks')
```

Output :

```
GeeksforGeeks
```

Refer this for more details of __future__ module.

This article is contributed by **Arpit Agarwal**. 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

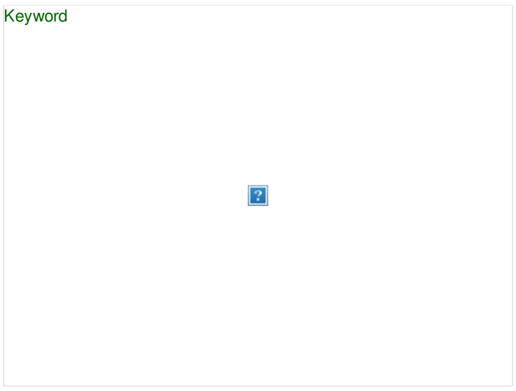
Python

How to check if a string is a valid keyword in Python?

Defining a Keyword

In programming, a keyword is a “**reserved word**” by the language which **convey a special meaning to the interpreter**. It may be a command or a parameter. Keywords **cannot be used as a variable name** in the program snippet.

Keywords in Python: Python language also reserves some of keywords that convey special meaning. Knowledge of these is necessary part of learning this language. Below is list of keywords registered by python .



How to check if a string is keyword?

Python in its language defines an inbuilt module "**keyword**" which handles certain operations related to keywords. A function "**iskeyword()**" checks if a string is keyword or not. Returns **true if a string is keyword, else returns false**.

```
#Python code to demonstrate working of iskeyword()

# importing "keyword" for keyword operations
import keyword

# initializing strings for testing
s = "for"
s1 = "geeksforgeeks"
s2 = "elif"
s3 = "elseif"
s4 = "nikhil"
s5 = "assert"
s6 = "shambhavi"
s7 = "True"
s8 = "False"
s9 = "akshat"
s10 = "akash"
s11 = "break"
s12 = "ashty"
s13 = "lambda"
s14 = "suman"
s15 = "try"
s16 = "vaishnavi"

# checking which are keywords
if keyword.iskeyword(s):
        print ( s + " is a python keyword")
else :  print ( s + " is not a python keyword")

if keyword.iskeyword(s1):
        print ( s1 + " is a python keyword")
else :  print ( s1 + " is not a python keyword")

if keyword.iskeyword(s2):
        print ( s2 + " is a python keyword")
else :  print ( s2 + " is not a python keyword")

if keyword.iskeyword(s3):
        print ( s3 + " is a python keyword")
else :  print ( s3 + " is not a python keyword")

if keyword.iskeyword(s4):
        print ( s4 + " is a python keyword")
else :  print ( s4 + " is not a python keyword")

if keyword.iskeyword(s5):
        print ( s5 + " is a python keyword")
else :  print ( s5 + " is not a python keyword")

if keyword.iskeyword(s6):
        print ( s6 + " is a python keyword")
else :  print ( s6 + " is not a python keyword")

if keyword.iskeyword(s7):
        print ( s7 + " is a python keyword")
else :  print ( s7 + " is not a python keyword")

if keyword.iskeyword(s8):
        print ( s8 + " is a python keyword")
else :  print ( s8 + " is not a python keyword")

if keyword.iskeyword(s9):
        print ( s9 + " is a python keyword")
else :  print ( s9 + " is not a python keyword")

if keyword.iskeyword(s10):
        print ( s10 + " is a python keyword")
else :  print ( s10 + " is not a python keyword")
```

```
if keyword.iskeyword(s11):
       print ( s11 + " is a python keyword")
else :  print ( s11 + " is not a python keyword")

if keyword.iskeyword(s12):
       print ( s12 + " is a python keyword")
else :  print ( s12 + " is not a python keyword")

if keyword.iskeyword(s13):
       print ( s13 + " is a python keyword")
else :  print ( s13 + " is not a python keyword")

if keyword.iskeyword(s14):
       print ( s14 + " is a python keyword")
else :  print ( s14 + " is not a python keyword")

if keyword.iskeyword(s15):
       print ( s15 + " is a python keyword")
else :  print ( s15 + " is not a python keyword")

if keyword.iskeyword(s16):
       print ( s16 + " is a python keyword")
else :  print ( s16 + " is not a python keyword")
```

Output:

```
for is a python keyword
geeksforgeeks is not a python keyword
elif is a python keyword
elseif is not a python keyword
nikhil is not a python keyword
assert is a python keyword
shambhavi is not a python keyword
True is a python keyword
False is a python keyword
akshat is not a python keyword
akash is not a python keyword
break is a python keyword
ashty is not a python keyword
lambda is a python keyword
suman is not a python keyword
try is a python keyword
vaishnavi is not a python keyword
```

How to print list of all keywords

Sometimes, remembering all the keywords can be a difficult task while assigning variable names. Hence a function “**kwlist()**” is provided in “keyword” module which **prints all the 33 python keywords**.

```
#Python code to demonstrate working of iskeyword()

# importing "keyword" for keyword operations
import keyword

# printing all keywords at once using "kwlist()"
print ("The list of keywords is : ")
print (keyword.kwlist)
```

Output:

```
The list of keywords is :
['False', 'None', 'True', 'and', 'as', 'assert', 'break', 'class',
'continue', 'def', 'del', 'elif', 'else', 'except', 'finally',
'for', 'from', 'global', 'if', 'import', 'in', 'is', 'lambda',
'nonlocal', 'not', 'or', 'pass', 'raise', 'return',
'try', 'while', 'with', 'yield']
```

Next Articles:

- Keywords in Python | Set 1
- Keywords in Python | Set 2

This article is contributed by **Manjeet Singh(S.Nandini)**. 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.

Python

Keywords in Python | Set 1

Python Keywords – Introduction

This article aims at providing a detailed insight to these keywords.

1. True : This keyword is used to represent a boolean true. If a statement is truth, “True” is printed.

2. False : This keyword is used to represent a boolean false. If a statement is False, “False” is printed.
True and False in python are same as 1 and 0.Example:

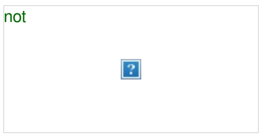
```
print False == 0
print True == 1

print True + True + True
print True + False + False
```

3. None : This is a special constant used to **denote a null value or a void**. **Its important to remember, 0, any empty container(e.g empty list) do not compute to None**.
It is an object of its own datatype – NoneType. It is not possible to create multiple None objects and can assign it to variables.

4. and : This a logical operator in python. “and” returns true if **both the operands are true**. Else returns false.The truth table for “and” is depicted below.

5. or : This a logical operator in python. “or” returns true **if any one of the operand is true**. Else returns false.The truth table for “or” is depicted below.

6. not : This logical operator **inverts the truth value**.The truth table for “not” is depicted below.



```
# Python code to demonstrate
# True, False, None, and, or , not

# showing that None is not equal to 0
# prints False as its false.
print (None == 0)

# showing objective of None
# two None value equated to None
# here x and y both are null
# hence true
x = None
y = None
print (x == y)

# showing logical operation
# or (returns True)
print (True or False)

# showing logical operation
# and (returns False)
print (False and True)

# showing logical operation
# not (returns False)
print (not True)
```

Output:

```
False
True
True
False
False
```

7. assert : This function is used for **debugging purposes**. Usually used to check the correctness of code. If a statement evaluated to true, nothing happens, but when it is false, "**AssertionError**" is raised . One can also **print a message with the error, separated by a comma**.

8. break : "break" is used to control the flow of loop. The statement is used to **break out of loop and passes the control to the statement following immediately after loop.**

9. continue : "continue" is also used to control the flow of code. The keyword **skips the current iteration of the loop**, but **does not end the loop**.

Illustrations of break and continue keywords can be seen in the article below.

Loops and Control Statements (continue, break and pass) in Python
10. class : This keyword is used to **declare user defined classes**.For more info. click here.

11. def : This keyword is used to **declare user defined functions**.For more info. click here.

12. if : It is a control statement for decision making. **Truth expression forces control to go in "if" statement block.**

13. else : It is a control statement for decision making. **False expression forces control to go in "else" statement block.**

14. elif : It is a control statement for decision making. It is short for "**else if**"

if, else and elif conditional statements are explained in detail here article.

15. del : del is used to **delete a reference to an object**. Any variable or list value can be deleted using del.

```
# Python code to demonstrate
# del and assert

# initialising list
a = [1, 2, 3]

# printing list before deleting any value
print ("The list before deleting any value")
print (a)

# using del to delete 2nd element of list
del a[1]

# printing list after deleting 2nd element
print ("The list after deleting 2nd element")
print (a)

# demonstrating use of assert
# prints AssertionError
assert 5 < 3, "5 is not smaller than 3"
```

Output:

```
The list before deleting any value
[1, 2, 3]
The list after deleting 2nd element
[1, 3]
```

Runtime Error:

```
Traceback (most recent call last):
  File "9e957ae60b718765ec2376b8ab4225ab.py", line 19, in
    assert 5<3, "5 is not smaller than 3"
AssertionError: 5 is not smaller than 3
```

Next Article – Keywords in Python | Set 2

This article is contributed by **Manjeet Singh(S. Nandini)**. 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.

Python

Keywords in Python | Set 2

Python Keywords – Introduction
Keywords in Python | Set 1

More keywords:

16. try : This keyword is used for exception handling**,** used to catch the errors in the code using the keyword except. Code in "try" block is checked, if there is any type of error, except block is executed.

17. except : As explained above, this works together with "try" to catch exceptions.

18. raise : Also used for exception handling to explicitly raise exceptions.

19. finally : No matter what is result of the "try" block, block termed "finally" is always executed. Detailed article –Exception Handling in Python

20. for : This keyword is used to control flow and for looping.

21. while : Has a similar working like “for” , used to control flow and for looping.

22. pass : It is the null statement in python. Nothing happens when this is encountered. This is used to prevent indentation errors and used as a placeholder

Detailed Article – for, while, pass

23. import : This statement is used to include a particular module into current program.

24. from : Generally used with import, from is used to import particular functionality from the module imported.

25. as : This keyword is used to create the alias for the module imported. i.e giving a new name to the imported module.. E.g import math as mymath.

Detailed Article – import, from and as

26. lambda : This keyword is used to make inline returning functions with no statements allowed internally. Detailed Article – map, filter, lambda

27. return : This keyword is used to return from the function. Detailed article – Return values in Python.

28. yield : This keyword is used like return statement but is used to return a generator. Detailed Article – yield keyword

29. with : This keyword is used to wrap the execution of block of code within methods defined by context manager.This keyword is not used much in day to day programming.

30. in : This keyword is used to check if a container contains a value. This keyword is also used to loop through the container.

31. is : This keyword is used to test object identity, i.e to check if both the objects take same memory location or not.

```
# Python code to demonstrate working of
# in and is

# using "in" to check
if 's' in 'geeksforgeeks':
      print ("s is part of geeksforgeeks")
else : print ("s is not part of geeksforgeeks")

# using "in" to loop through
for i in 'geeksforgeeks':
    print (i,end=" ")

print ("\r")

# using is to check object identity
# string is immutable( cannot be changed once alloted)
# hence occupy same memory location
print (' ' is ' ')

# using is to check object identity
# dictionary is mutable( can be changed once alloted)
# hence occupy different memory location
print ({} is {})
```

Output:

```
s is part of geeksforgeeks
g e e k s f o r g e e k s
True
False
```

32. global : This keyword is used to define a variable inside the function to be of a global scope.

33. non-local : This keyword works similar to the global, but rather than global, this keyword declares a variable to point to variable of outside enclosing function, in case of nested functions.

```
# Python code to demonstrate working of
# global and non local

#initializing variable globally
a = 10

# used to read the variable
def read():
    print (a)

# changing the value of globally defined variable
def mod1():
    global a
    a = 5

# changing value of only local variable
def mod2():
    a = 15

# reading initial value of a
# prints 10
read()

# calling mod 1 function to modify value
# modifies value of global a to 5
mod1()

# reading modified value
# prints 5
read()

# calling mod 2 function to modify value
# modifies value of local a to 15, doesn't effect global value
mod2()

# reading modified value
# again prints 5
read()

# demonstrating non local
# inner loop changing the value of outer a
# prints 10
print ("Value of a using nonlocal is : ",end="")
def outer():
   a = 5
   def inner():
      nonlocal a
      a = 10
   inner()
   print (a)

outer()

# demonstrating without non local
# inner loop not changing the value of outer a
# prints 5
print ("Value of a without using nonlocal is : ",end="")
def outer():
   a = 5
   def inner():
      a = 10
   inner()
   print (a)

outer()
```

Output:

```
10
5
5
Value of a using nonlocal is : 10
Value of a without using nonlocal is : 5
```

This article is contributed by **Manjeet Singh(S. Nandini)**. 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

Python | Set 2 (Variables, Expressions, Conditions and Functions)

Introduction to Python has been dealt with in this article. Now, let us begin with learning python.

Running your First Code in Python

Python programs are not compiled, rather they are interpreted. Now, let us move to writing a python code and running it. Please make sure that python is installed on the system you are working. If it is not installed, download it from here. We will be using python 2.7.

Making a Python file:

Python files are stored with the extension ".py". Open text editor and save a file with the name "hello.py". Open it and write the following code:

```
print "Hello World"
# Notice that NO semi-colon is to be used
```

Reading the file contents:

Linux System – Move to the directory from terminal where the created file (hello.py) is stored by using the 'cd' command, and then type the following in the terminal :

```
python hello.py
```

Windows system – Open command prompt and move to the directory where the file is stored by using the 'cd' command and then run the file by writing the file name as command.

Variables in Python

Variables need not be declared first in python. They can be used directly. Variables in python are case sensitive as most of the other programming languages.

Example:

```
a = 3
```

```
A = 4
print a
print A
```

The output is :

```
3
4
```

Expressions in Python

Arithmetic operations in python can be performed by using arithmetic operators and some of the in-built functions.

```
a = 2
b = 3
c = a + b
print c
d = a * b
print d
```

The output is :

```
5
6
```

Conditions in Python

Conditional output in python can be obtained by using if-else and elif (else if) statements.

```
a = 3
b = 9
if b % a == 0 :
    print "b is divisible by a"
elif b + 1 == 10:
    print "Increment in b produces 10"
else:
    print "You are in else statement"
```

The output is :

```
b is divisible by a
```

Functions in Python

A function in python is declared by the keyword 'def' before the name of the function. The return type of the function need not be specified explicitly in python. The function can be invoked by writing the function name followed by the parameter list in the brackets.

```
# Function for checking the divisibility
# Notice the indentation after function declaration
# and if and else statements
def checkDivisibility(a, b):
 if a % b == 0 :
     print "a is divisible by b"
 else:
     print "a is not divisible by b"
#Driver program to test the above function
checkDivisibility(4, 2)
```

The output is :

```
a is divisible by b
```

So, python is a very simplified and less cumbersome language to code in. This easiness of python is itself promoting its wide use.

- Next Article- Python Data Types
- Quiz – Functions in Python

This article has been contributed by **Nikhil Kumar Singh.** Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

GATE CS Notes (According to Official GATE 2017 Syllabus)

GATE CS Corner

See Placement Course for placement preparation, GATE Corner for GATE CS Preparation and Quiz Corner for all Quizzes on GeeksQuiz.
Category: Python Articles

What is the maximum possible value of an integer in Python ?

Consider below Python program.

```
# A Python program to demonstrate that we can store
# large numbers in Python

x = 10000000000000000000000000000000000000000000;
x = x + 1
print (x)
```

Output :

```
10000000000000000000000000000000000000000001
```

In Python, value of an integer is not restricted by the number of bits and can expand to the limit of the available memory (Sources : this and this). Thus we never need any special arrangement for storing large numbers (Imagine doing above arithmetic in C/C++).

As a side note, in Python 3, there is only one type “int” for all type of integers. In Python 2.7. there are two separate types “int” (which is 32 bit) and “long int” that is same as “int” of Python 3.x, i.e., can store arbitrarily large numbers.

```
# A Python program to show that there are two types in
# Python 2.7 : int and long int
# And in Python 3 there is only one type : int

x = 10
print(type(x))

x = 10000000000000000000000000000000000000000000
print(type(x))
```

Output in Python 2.7 :

```
<type 'int'>
<type 'long'>
```

Output in Python 3 :

```
<type 'int'>
<type 'int'>
```

We may want to try more interesting programs like below :

```
# Printing 100 raise to power 100
print(100**100)
```

This article is contributed by **Abhay Rathi**. 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
large-numbers
Python

Transpose a matrix in Single line in Python

Transpose of a matrix is a task we all can perform very easily in python (Using a nested loop). But there are some interesting ways to do the same in a single line.
In Python, we can implement a matrix as nested list (list inside a list). Each element is treated as a row of the matrix. For example m = [[1, 2], [4, 5], [3, 6]] represents a matrix of 3 rows and 2 columns.
First element of the list – **m[0]** and element in first row, first column – **m[0][0]**.

1. **Using Nested List Comprehension:** Nested list comprehension are used to iterate through each element in the matrix.In the given example ,we iterate through each element of matrix (m) in column major manner and assign the result to rez matrix which is the transpose of m.

   ```
   m = [[1,2],[3,4],[5,6]]
   for row in m :
    print(row)
   rez = [[m[j][i] for j in range(len(m))] for i in range(len(m[0]))]
   print("\n")
   for row in rez:
    print(row)
   ```

   Output:

   ```
   [1, 2]
   [3, 4]
   [5, 6]


   [1, 3, 5]
   [2, 4, 6]
   ```

2. **Using zip:** Zip returns an iterator of tuples, where the i-th tuple contains the i-th element from each of the argument sequences or iterables. In this example we unzip our array using * and then zip it to get the transpose.

```
matrix=[(1,2,3),(4,5,6),(7,8,9),(10,11,12)]
for row in matrix:
 print(row)
print("\n")
t_matrix = zip(*matrix)
for row in t_matrix:
 print row
```

Output:

```
(1, 2, 3)
(4, 5, 6)
(7, 8, 9)
(10, 11, 12)

(1, 4, 7, 10)
(2, 5, 8, 11)
(3, 6, 9, 12)
```

Note :- If you want your result in the form [[1,4,7,10][2,5,8,11][3,6,9,12]] , you can use t_matrix=map(list, zip(*matrix)).

3. **Using numpy:** NumPy is a general-purpose array-processing package designed to efficiently manipulate large multi-dimensional arrays. The transpose method returns a transposed view of the passed multi-dimensional matrix.

```
# You need to install numpy in order to import it
# Numpy transpose returns similar result when
# applied on 1D matrix
import numpy
matrix=[[1,2,3],[4,5,6]]
print(matrix)
print("\n")
print(numpy.transpose(matrix))
```

This article is contributed by **Mayank Rawat** .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

Global and Local Variables in Python

Global variables are the one that are defined and declared outside a function and we need to use them inside a function.

```
# This function uses global variable s
def f():
    print s

# Global scope
s = "I love Geeksforgeeks"
f()
```

Output:

```

I love Geeksforgeeks
```

If a variable with same name is defined inside the scope of function as well then it will print the value given inside the function only and not the global value.

```
# This function has a variable with
# name same as s.
def f():
    s = "Me too."
    print s

# Global scope
s = "I love Geeksforgeeks"
f()
print s
```

Output:

```
Me too.
I love Geeksforgeeks.
```

The variable s is defined as the string "I hate spam", before we call the function f(). The only statement in f() is the "print s" statement. As there is no local s, the value from the global s will be used.

The question is, what will happen, if we change the value of s inside of the function f()? Will it affect the global s as well? We test it in the following piece of code:

```
def f():
    print s

    # This program will NOT show error
    # if we comment below line.
    s = "Me too."

    print s

# Global scope
s = "I love Geeksforgeeks"
f()
print s
```

Output:

```
Line 2: undefined: Error: local variable 's' referenced before assignment
```

To make the above program work, we need to use “global” keyword. We only need to use global keyword in a function if we want to do assignments / change them. global is not needed for printing and accessing. Why? Python “assumes” that we want a local variable due to the assignment to s inside of f(), so the first print statement throws this error message. Any variable which is changed or created inside of a function is local, if it hasn’t been declared as a global variable. To tell Python, that we want to use the global variable, we have to use the keyword **“global”**, as can be seen in the following example:

```
# This function modifies global variable 's'
def f():
    global s
    print s
    s = "Look for Geeksforgeeks Python Section"
    print s

# Global Scope
s = "Python is great!"
f()
print s
```

Now there is no ambiguity.
Output:

```
Python is great!
Look for Geeksforgeeks Python Section.
Look for Geeksforgeeks Python Section.
```

A good Example

```
a = 1

# Uses global because there is no local 'a'
def f():
    print 'Inside f() : ', a

# Variable 'a' is redefined as a local
def g():
    a = 2
    print 'Inside g() : ',a

# Uses global keyword to modify global 'a'
def h():
    global a
    a = 3
    print 'Inside h() : ',a

# Global scope
print 'global : ',a
f()
print 'global : ',a
g()
print 'global : ',a
h()
print 'global : ',a
```

Output:

```
global :  1
Inside f() :  1
global :  1
Inside g() :  2
global :  1
Inside h() :
global :  3
```

This article is contributed by **Shwetanshu Rohatgi**. 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.

Partial Functions in Python

Partial functions allow us to fix a certain number of arguments of a function and generate a new function.

Example:

```
from functools import partial

# A normal function
def f(a, b, c, x):
    return 1000*a + 100*b + 10*c + x

# A partial function that calls f with
# a as 3, b as 1 and c as 4.
g = partial(f, 3, 1, 4)

# Calling g()
print(g(5))
```

Output:

```
3145
```

In the example we have pre-filled our function with some constant values of a, b and c. And g() just takes a single argument i.e. the variable x.

Another Example :

```
from functools import *

# A normal function
def add(a, b, c):
   return 100*a + 10*b  + c

# A partial function with b = 1 and c = 2
add_part = partial(add, c=2, b=1)

# Calling partial function
print(add_part(3))
```

Output:

```
312
```

- Partial functions can be used to derive specialized functions from general functions and therefore help us to reuse our code.
- This feature is similar to bind in C++.

This article is contributed by **Mayank Rawat** .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

Packing and Unpacking Arguments in Python

We use two operators * (for tuples) and ** (for dictionaries).

Background

Consider a situation where we have a function that receives four arguments. We want to make call to this function and we have a list of size 4 with us that has all arguments for the function. If we simply pass list to the function, the call doesn't work.

```
# A Python program to demonstrate need
# of packing and unpacking

# A sample function that takes 4 arguments
# and prints them.
def fun(a, b, c, d):
    print(a, b, c, d)

# Driver Code
my_list = [1, 2, 3, 4]

# This doesn't work
fun(my_list)
```

Output :

```
TypeError: fun() takes exactly 4 arguments (1 given)
```

Unpacking

We can us * to unpack the list so that all elements of it can be passed as different parameters.

```
# A sample function that takes 4 arguments
# and prints the,
def fun(a, b, c, d):
    print(a, b, c, d)

# Driver Code
my_list = [1, 2, 3, 4]

# Unpacking list into four arguments
fun(*my_list)
```

Output :

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

As another example, consider the built-in range() function that expects separate start and stop arguments. If they are not available separately, write the function call with the *-operator to unpack the arguments out of a list or tuple:

```
>>>
>>> range(3, 6)  # normal call with separate arguments
[3, 4, 5]
>>> args = [3, 6]
>>> range(*args)  # call with arguments unpacked from a list
[3, 4, 5]
```

Packing

When we don’t know how many arguments need to be passed to a python function, we can use Packing to pack all arguments in a tuple.

```
# A Python program to demonstrate use
# of packing

# This function uses packing to sum
# unknown number of arguments
def mySum(*args):
    sum = 0
    for i in range(0, len(args)):
        sum = sum + args[i]
    return sum

# Driver code
print(mySum(1, 2, 3, 4, 5))
print(mySum(10, 20))
```

Output:

```
15
30
```

The above function mySum() does ‘packing’ to pack all the arguments that this method call receives into one single variable. Once we have this ‘packed’ variable, we can do things with it that we would with a normal tuple. args[0] and args[1] would give you the first and second argument, respectively. Since our tuples are immutable so if you convert the args tuple to a list you can also modify, delete and re-arrange items in i just like a normal list.

Packing and Unpacking

Below is an example that shows both packing and unpacking.

```
# A Python program to demonstrate both packing and
# unpacking.

# A sample python function that takes three arguments
# and prints them
def fun1(a, b, c):
    print(a, b, c)

# Another sample function.
# This is an example of PACKING. All arguments passed
# to fun2 are packed into tuple *args.
def fun2(*args):

    # Convert args tuple to a list so we can modify it
    args = list(args)

    # Modifying args
    args[0] = 'Geeksforgeeks'
    args[1] = 'awesome'

    # UNPACKING args and calling fun1()
    fun1(*args)

# Driver code
fun2('Hello', 'beautiful', 'world!')
```

Output:

```
(Geeksforgeeks, awesome, world!)
```

**** is used for dictionaries**

```
# A sample program to demonstrate unpacking of
# dictionary items using **
def fun(a, b, c):
    print(a, b, c)

# A call with unpacking of dictionary
d = {'a':2, 'b':4, 'c':10}
fun(**d)
```

Output-

```
2 4 10
```

Here ** unpacked the dictionary used with it, and passed the items in the dictionary as keyword arguments to the function. So writing “fun(1, **d)” was equivalent to writing “fun(1, b=8, c=16)”.

```
# A Python program to demonstrate packing of
# dictionary items using **
def fun(**kwargs):

    # kwargs is a dict
    print(type(kwargs))

    # Printing dictionary items
    for key in kwargs:
        print("%s = %s" % (key, kwargs[key]))

# Driver code
fun(name="geeks", ID="101", language="Python")
```

Output :

```
<class 'dict'>
language = Python
name = geeks
ID = 101
```

Applications and Important Points
1. Used in socket programming to send a infinite no. of requests to server.
2. Used in Django framework to send variable arguments to view functions.
3. There are wrapper functions that require us to pass in variable argurents.
4. Modification of arguments become easy but at the same time validation is not proper so they must be used with care.

Reference :
http://hangar.runway7.net/python/packing-unpacking-arguments

This article is contributed by **Shwetanshu Rohatgi** .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

Python | end parameter in print()

By default python’s print() function ends with a newline. A programmer with C/C++ background may wonder how to print without newline.

Python’s print() function comes with a parameter called ‘end’. By default, the value of this parameter is ‘\n’, i.e. the new line character. You can end a print statement with any character/string using this parameter.

```
# This Python program must be run with
# Python 3 as it won't work with 2.7.

# ends the output with a <space>
print("Welcome to" , end = ' ')
print("GeeksforGeeks", end = ' ')
```

Output :

```
Welcome to GeeksforGeeks
```

One more program to demonstrate working of end parameter.

```
# This Python program must be run with
# Python 3 as it won't work with 2.7.

# ends the output with '@'
print("Python" , end = '@')
print("GeeksforGeeks")
```

Output :

```
Python@GeeksforGeeks
```

This article is contributed by **Ankit Bindal**. 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

Type Conversion in Python

Python defines type conversion functions to directly convert one data type to another which is useful in day to day and competitive programming. This article is aimed at providing the information about certain conversion functions.

1. int(a,base) : This function converts **any data type to integer**. 'Base' specifies the **base in which string is** if data type is string.

2. float() : This function is used to convert **any data type to a floating point number**
.

```
# Python code to demonstrate Type conversion
# using int(), float()

# initializing string
s = "10010"

# printing string converting to int base 2
c = int(s,2)
print ("After converting to integer base 2 : ", end="")
print (c)

# printing string converting to float
e = float(s)
print ("After converting to float : ", end="")
print (e)
```

Output:

```
After converting to integer base 2 : 18
After converting to float : 10010.0
```

3. ord() : This function is used to convert a **character to integer.**

4. hex() : This function is to convert **integer to hexadecimal string**.

5. oct() : This function is to convert **integer to octal string**.

```
# Python code to demonstrate Type conversion
# using  ord(), hex(), oct()

# initializing integer
s = '4'

# printing character converting to integer
c = ord(s)
print ("After converting character to integer : ",end="")
print (c)

# printing integer converting to hexadecimal string
c = hex(56)
print ("After converting 56 to hexadecimal string : ",end="")
print (c)

# printing integer converting to octal string
c = oct(56)
print ("After converting 56 to octal string : ",end="")
print (c)
```

Output:

```
After converting character to integer : 52
After converting 56 to hexadecimal string : 0x38
After converting 56 to octal string : 0o70
```

6. tuple() : This function is used to **convert to a tuple**.

7. set() : This function returns the **type after converting to set**.

8. list() : This function is used to convert **any data type to a list type**.

```
# Python code to demonstrate Type conversion
# using  tuple(), set(), list()

# initializing string
s = 'geeks'

# printing string converting to tuple
c = tuple(s)
print ("After converting string to tuple : ",end="")
print (c)

# printing string converting to set
c = set(s)
print ("After converting string to set : ",end="")
print (c)

# printing string converting to list
c = list(s)
print ("After converting string to list : ",end="")
print (c)
```

Output:

```
After converting string to tuple : ('g', 'e', 'e', 'k', 's')
After converting string to set : {'k', 'e', 's', 'g'}
After converting string to list : ['g', 'e', 'e', 'k', 's']
```

9. dict() : This function is used to **convert a tuple of order (key,value) into a dictionary**.

10. str() : Used to **convert integer into a string.**

11. complex(real,imag) : : This function **converts real numbers to complex(real,imag) number.**

```
# Python code to demonstrate Type conversion
# using  dict(), complex(), str()

# initializing integers
a = 1
b = 2

# initializing tuple
tup = (('a', 1) ,('f', 2), ('g', 3))

# printing integer converting to complex number
c = complex(1,2)
print ("After converting integer to complex number : ",end="")
print (c)

# printing integer converting to string
c = str(a)
print ("After converting integer to string : ",end="")
print (c)

# printing tuple converting to expression dictionary
c = dict(tup)
print ("After converting tuple to dictionary : ",end="")
print (c)
```

Output:

```
After converting integer to complex number : (1+2j)
After converting integer to string : 1
After converting tuple to dictionary : {'a': 1, 'f': 2, 'g': 3}
```

This article is contributed by **Manjeet Singh(S. Nandini)**. 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

Byte Objects vs String in Python

In Python 2, both str and bytes are the same typeByte objects whereas in Python 3 Byte objects, defined in Python 3 are "**sequence of bytes**" and similar to "**unicode**" objects from Python 2. However, there are many differences in strings and Byte objects. Some of them are depicted below:
`

- Byte objects are sequence of **Bytes**, whereas Strings are sequence of **characters**.
- Byte objects are in **machine readable** form internally, Strings are only in **human readable** form.
- Since Byte objects are machine readable, they can be **directly stored on the disk**. Whereas, Strings **need encoding** before which they can be stored on disk.

There are methods to convert a byte object to String and String to byte objects.

Encoding

PNG, JPEG, MP3, WAV, ASCII, UTF-8 etc are different forms of encodings. An encoding is a format to represent audio, images, text, etc in bytes. Converting **Strings to byte** objects is termed as encoding. This is necessary so that the text can be stored on disk using mapping using **ASCII** or **UTF-8** encoding techniques.

This task is achieved using **encode()**. It take encoding technique as argument. Default technique is “**UTF-8**” technique.

```
# Python code to demonstate String encoding

# initialising a String
a = 'GeeksforGeeks'

# initialising a byte object
c = b'GeeksforGeeks'

# using encode() to encode the String
# encoded version of a is stored in d
# using ASCII mapping
d = a.encode('ASCII')

# checking if a is converted to bytes or not
if (d==c):
    print ("Encoding successful")
else : print ("Encoding Unsuccessful")
```

Output:

```
Encoding successful
```

Decoding

Similarly, Decoding is process to convert a **Byte object to String**. It is implemented using **decode()** . A byte string can be decoded back into a character string, if you know which encoding was used to encode it. Encoding and Decoding are **inverse** processes.

```
# Python code to demonstate Byte Decoding

# initialising a String
a = 'GeeksforGeeks'

# initialising a byte object
c = b'GeeksforGeeks'

# using decode() to decode the Byte object
# decoded version of c is stored in d
# using ASCII mapping
d = c.decode('ASCII')

# checking if c is converted to String or not
if (d==a):
    print ("Decoding successful")
else : print ("Decoding Unsuccessful")
```

Output:

```
Decoding successful
```

This article is contributed by **Manjeet Singh**. 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

G-Fact 19 (Logical and Bitwise Not Operators on Boolean)

Most of the languages including C, C++, Java and Python provide the boolean type that can be either set to **False** or **True**.

Consider below programs that use **Logical Not (or !)** operator on boolean.

Python

```
# A Python program that uses Logical Not or ! on boolean
a = not True
b = not False
print a
print b
```

```
# Output: False
#        True
```

C/C++

```
// A C/C++ program that uses Logical Not or ! on boolean
#include <stdio.h>
#include <stdbool.h>

int main()
{
    bool a = 1, b = 0;
    a = !a;
    b = !b;
    printf("%d\n%d", a, b);
    return 0;
}
// Output: 0
//        1
```

Java

```
// A Java program that uses Logical Not or ! on boolean
import java.io.*;

class GFG
{
    public static void main (String[] args)
    {
        boolean a = true, b = false;
        System.out.println(!a);
        System.out.println(!b);
    }
}
// Output: False
//        True
```

The outputs of above programs are as expected, but the outputs following programs may not be as expected if we have not used **Bitwise Not (or ~)** operator before.

Python

```
# A Python program that uses Bitwise Not or ~ on boolean
a = True
b = False
print ~a
print ~b
```

C/C++

```
// C/C++ program that uses Bitwise Not or ~ on boolean
#include <bits/stdc++.h>
using namespace std;
int main()
{
    bool a = true, b = false;
    cout << ~a << endl << ~b;
    return 0;
}
```

Output:

```
-2
-1
```

Reason: The bitwise not operator ~ returns the complement of a number i.e., it switches each 1 to 0 and each 0 to 1. Booleans True and False have values 1 and 0 respectively.

˜being the bitwise not operator,

- The expression "˜True" returns bitwise inverse of 1.
- The expression "˜False" returns bitwise inverse of 0.

Java doesn't allow ~ operator to be applied on boolean values. For example, the below program produces compiler error.

```
// A Java program that uses Bitwise Not or ~ on boolean
import java.io.*;

class GFG
```

```
{
    public static void main (String[] args)
    {
        boolean a = true, b = false;
        System.out.println(~a);
        System.out.println(~b);
    }
}
```

Output:

```
6: error: bad operand type boolean for unary operator '~'
  System.out.println(~a);
                     ^
7: error: bad operand type boolean for unary operator '~'
  System.out.println(~b);
                     ^
2 errors
```

Conclusion:
“Logical not or !” is meant for boolean values and “bitwise not or ~” is for integers. Languages like C/C++ and python do auto promotion of boolean to integer type when an integer operator is applied. But Java doesn’t do it.

This article is contributed by **Arpit Agarwal**. 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

GFacts
Python

Ternary Operator in Python

Ternary operators also known as conditional expressions are operators that evaluate something based on a condition being true or false. It was added to Python in version 2.5. It simply allows to test a condition in a **single line** replacing the multiline if-else making the code compact.

Syntax :

```
[on_true] if [expression] else [on_false]
```

Example :

```
# Program to demonstrate conditional operator
a, b = 10, 20

# Copy value of a in min if a < b else copy b
min = a if a < b else b

print(min)
```

Output :

```
10
```

Important Points:

- First the given condition is evaluated (a < b), then either a or b is returned based on the Boolean value returned by the condition
- Order of the arguments in the operator is different from other languages like C/C++ (See C/C++ ternary operators).
- Conditional expressions have the lowest priority amongst all Python operations.

Method used prior to 2.5 when ternary operator was not present
In an expression like the one given below , the interpreter checks for the expression if this is true then on_true is evaluated, else the on_false is evaluated.

Syntax :

```
'''When condition becomes true, expression [on_false]
   is not executed and value of "True and [on_true]"
   is returned.  Else value of "False or [on_false]"
   is returned.
   Note that "True and x" is equal to x.
   And "False or x" is equal to x. '''
[expression] and [on_true] or [on_false]
```

Example :

```
# Program to demonstrate conditional operator
a, b = 10, 20

# If a is less than b, then a is assigned
# else b is assigned (Note : it doesn't
# work if a is 0.
min = a < b and a or b

print(min)
```

Output:

```
10
```

Note : The only drawback of this method is that **on_true must not be zero or False**. If this happens on_false will be evaluated always. The reason for that is if expression is true, the interpreter will check for the on_true, if that will be zero or false, that will force the interpreter to check for on_false to give the final result of whole expression.

This article is contributed by **Mayank Rawat** .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

Increment and Decrement Operators in Python

If you're familiar with Python, you would have known Increment and Decrement operators (both pre and post) are not allowed in it.

Python is designed to be consistent and readable. One common error by a novice programmer in languages with ++ and -- operators is mixing up the differences (both in precedence and in return value) between pre and post increment/decrement operators. Simple increment and decrement operators aren't needed as much as in other languages.

You don't write things like :

```
for (int i = 0; i < 5; ++i)
```

In Python, instead we write it like

```
# A Sample Python program to show loop (unlike many
# other languages, it doesn't use ++)
for i in range(0, 5):
   print(i)
```

Output:

```
0
1
2
3
4
```

We can almost always avoid use of ++ and --. For example, **x++** can be written as **x += 1** and **x--** can be written as **x -= 1**.

This article is contributed by **Harshit Gupta**. 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
Python

Division Operators in Python

Consider below statements in Python 2.7

```
# A Python 2.7 program to demonstrate use of
# "/" for integers
print 5/2
print -5/2
```

Output:

```
2
-3
```

First output is fine, but the second one may be surprising if we are coming Java/C++ world. In Python 2.7, the "/" operator works as a floor division for integer arguments. However, the operator / returns a float value if one of the arguments is a float (this is similar to C++)

```
# A Python 2.7 program to demonstrate use of
# "/" for floating point numbers
print 5.0/2
print -5.0/2
```

Output:

```
2.5
-2.5
```

The real floor division operator is "//". It returns floor value for both integer and floating point arguments.

```
# A Python 2.7 program to demonstrate use of
# "//" for both integers and floating points
print 5//2
print -5//2
print 5.0//2
print -5.0//2
```

Output:

```
2
-3
2.0
-3.0
```

How about Python 3?
Here is another surprise, In Python 3, '/' operator does floating point division for both int and float arguments.

```
# A Python 3 program to demonstrate use of
# "/" for both integers and floating points
print (5/2)
print (-5/2)
print (5.0/2)
print (-5.0/2)
```

Output:

```
2.5
-2.5
2.5
-2.5
```

Behavior of "//" is same Python 2.7 and Python 3. See this for example.

This article is contributed by **Arpit Agrawal**. 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
Python

Any & All in Python

Any and All are two built ins provided in python used for successive And/Or.

Any

Returns true if any of the items is True. It returns False if empty or all are false. Any can be thought of as a sequence of OR operations on the provided iterables.
It short circuit the execution i.e. stop the execution as soon as the result is known.
Syntax : any(list of iterables)

```
# Since all are false, false is returned
print (any([False, False, False, False]))

# Here the method will short-circuit at the
# second item (True) and will return True.
print (any([False, True, False, False]))

# Here the method will short-circuit at the
# first (True) and will return True.
print (any([True, False, False, False]))
```

Output :

```
False
True
True
```

All

Returns true if all of the items are True (or if the iterable is empty). All can be thought of as a sequence of AND operations on the provided iterables. It also short circuit the execution i.e. stop the execution as soon as the result is known.
Syntax : all(list of iterables)

```
# Here all the iterables are True so all
# will return True and the same will be printed
print (all([True, True, True, True]))

# Here the method will short-circuit at the
# first item (False) and will return False.
print (all([False, True, True, False]))

# This statement will return False, as no
# True is found in the iterables
print (all([False, False, False]))
```

Output :

```
True
False
False
```

Truth table :-

Capture

This article is contributed by **Mayank Rawat** .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

Inplace vs Standard Operators in Python

Inplace Operators – Set 1, Set 2

Normal operators do the simple assigning job. On other hand, Inplace operators behave similar to normal operators **except** that they act in a different manner in case of mutable and Immutable targets.

- The **_add_** method, does simple addition, takes two arguments, returns the sum and stores it in other variable without modifying any of the argument.
- On the other hand, **_iadd_** method also takes two arguments, but it makes **in-place change** in 1st argument passed by storing the sum in it. As object mutation is needed in this process, immutable targets such as numbers, strings and tuples, **shouldn't have _iadd_ method**.
- **Normal operator's "add()"** method, implements "**a+b**" and stores the result in the mentioned variable.
- **Inplace operator's "iadd()"** method, implements "**a+=b**" if it exists (i.e in case of immutable targets, it doesn't exist) and changes the value of passed argument. But **if not, "a+b" is implemented**.

In both the cases assignment is required to do to store the value.

Case 1 : **Immutable Targets.**
In Immutable targets, such as numbers, strings and tuples. Inplace operator behave same as normal operators, i.e only assignment takes place, no modification is taken place in the passed arguments.

```
# Python code to demonstrate difference between
# Inplace and Normal operators in Immutable Targets

# importing operator to handle operator operations
import operator

# Initializing values
x = 5
y = 6
a = 5
b = 6

# using add() to add the arguments passed
z = operator.add(a,b)

# using iadd() to add the arguments passed
p = operator.iadd(x,y)

# printing the modified value
print ("Value after adding using normal operator : ",end="")
print (z)

# printing the modified value
print ("Value after adding using Inplace operator : ",end="")
print (p)

# printing value of first argument
# value is unchanged
print ("Value of first argument using normal operator : ",end="")
print (a)

# printing value of first argument
# value is unchanged
print ("Value of first argument using Inplace operator : ",end="")
print (x)
```

Output:

```
Value after adding using normal operator : 11
Value after adding using Inplace operator : 11
Value of first argument using normal operator : 5
Value of first argument using Inplace operator : 5
```

Case 2 : **Mutable Targets**
The behaviour of Inplace operators in mutable targets, such as list and dictionaries, is different from normal operators.The **updation and assignment both are carried out** in case of mutable targets.

```
# Python code to demonstrate difference between
# Inplace and Normal operators in mutable Targets

# importing operator to handle operator operations
import operator

# Initializing list
a = [1, 2, 4, 5]

# using add() to add the arguments passed
z = operator.add(a,[1, 2, 3])

# printing the modified value
print ("Value after adding using normal operator : ",end="")
print (z)

# printing value of first argument
# value is unchanged
print ("Value of first argument using normal operator : ",end="")
print (a)

# using iadd() to add the arguments passed
# performs a+=[1, 2, 3]
p = operator.iadd(a,[1, 2, 3])

# printing the modified value
print ("Value after adding using Inplace operator : ",end="")
print (p)

# printing value of first argument
# value is changed
print ("Value of first argument using Inplace operator : ",end="")
print (a)
```

Output:

```
Value after adding using normal operator : [1, 2, 4, 5, 1, 2, 3]
Value of first argument using normal operator : [1, 2, 4, 5]
Value after adding using Inplace operator : [1, 2, 4, 5, 1, 2, 3]
Value of first argument using Inplace operator : [1, 2, 4, 5, 1, 2, 3]
```

This article is contributed by **Manjeet Singh**. 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

Operator Functions in Python | Set 1

Python has predefined functions for many mathematical, logical, relational, bitwise etc operations under the module "operator". Some of the basic functions are covered in this article.

1. add(a, b) :- This functions returns **addition** of the given arguments.
Operation – **a + b.**

2. sub(a, b) :- This functions returns **difference** of the given arguments.
Operation – **a – b.**

3. mul(a, b) :- This functions returns **product** of the given arguments.
Operation – **a * b.**

```
# Python code to demonstrate working of
# add(), sub(), mul()

# importing operator module
import operator
```

```
# Initializing variables
a = 4

b = 3

# using add() to add two numbers
print ("The addition of numbers is :",end="");
print (operator.add(a, b))

# using sub() to subtract two numbers
print ("The difference of numbers is :",end="");
print (operator.sub(a, b))

# using mul() to multiply two numbers
print ("The product of numbers is :",end="");
print (operator.mul(a, b))
```

Output:

```
The addition of numbers is :7
The difference of numbers is :1
The product of numbers is :12
```

4. truediv(a,b) :- This functions returns **division** of the given arguments.
Operation – **a** / **b.**

5. floordiv(a,b) :- This functions also returns division of the given arguments. But the value is floored value i.e. **returns greatest small integer**.
Operation – **a** // **b.**

6. pow(a,b) :- This functions returns **exponentiation** of the given arguments.
Operation – **a ** b.**

7. mod(a,b) :- This functions returns **modulus** of the given arguments.
Operation – **a % b.**

```
# Python code to demonstrate working of
# truediv(), floordiv(), pow(), mod()

# importing operator module
import operator

# Initializing variables
a = 5

b = 2

# using truediv() to divide two numbers
print ("The true division of numbers is : ",end="");
print (operator.truediv(a,b))

# using floordiv() to divide two numbers
print ("The floor division of numbers is : ",end="");
print (operator.floordiv(a,b))

# using pow() to exponentiate two numbers
print ("The exponentiation of numbers is : ",end="");
print (operator.pow(a,b))

# using mod() to take modulus of two numbers
print ("The modulus of numbers is : ",end="");
print (operator.mod(a,b))
```

Output:

```
The true division of numbers is : 2.5
The floor division of numbers is : 2
The exponentiation of numbers is : 25
The modulus of numbers is : 1
```

8. lt(a, b) :- This function is used to **check if a is less than b or not**. Returns true if a is less than b, else returns false.
Operation – **a < b**.

9. le(a, b) :- This function is used to **check if a is less than or equal to b or not**. Returns true if a is less than or equal to b, else returns false.
Operation – **a <= b**.

10. eq(a, b) :- This function is used to **check if a is equal to b or not**. Returns true if a is equal to b, else returns false.
Operation – **a == b**.

```
# Python code to demonstrate working of
# lt(), le() and eq()

# importing operator module
import operator

# Initializing variables
a = 3

b = 3
```

```
# using lt() to check if a is less than b
if(operator.lt(a,b)):
    print ("3 is less than 3")
else : print ("3 is not less than 3")

# using le() to check if a is less than or equal to b
if(operator.le(a,b)):
    print ("3 is less than or equal to 3")
else : print ("3 is not less than or equal to 3")

# using eq() to check if a is equal to b
if (operator.eq(a,b)):
    print ("3 is equal to 3")
else : print ("3 is not equal to 3")
```

Output:

```
3 is not less than 3
3 is less than or equal to 3
3 is equal to 3
```

11. gt(a,b) :- This function is used to **check if a is greater than b or not**. Returns true if a is greater than b, else returns false.
Operation – **a > b**.

12. ge(a,b) :- This function is used to **check if a is greater than or equal to b or not**. Returns true if a is greater than or equal to b, else returns false.
Operation – **a >= b**.

13. ne(a,b) :- This function is used to **check if a is not equal to b or is equal**. Returns true if a is not equal to b, else returns false.
Operation – **a != b**.

```
# Python code to demonstrate working of
# gt(), ge() and ne()

# importing operator module
import operator

# Initializing variables
a = 4

b = 3

# using gt() to check if a is greater than b
if (operator.gt(a,b)):
    print ("4 is greater than 3")
else : print ("4 is not greater than 3")

# using ge() to check if a is greater than or equal to b
if (operator.ge(a,b)):
    print ("4 is greater than or equal to 3")
else : print ("4 is not greater than or equal to 3")

# using ne() to check if a is not equal to b
if (operator.ne(a,b)):
    print ("4 is not equal to 3")
else : print ("4 is equal to 3")
```

Output:

```
4 is greater than 3
4 is greater than or equal to 3
4 is not equal to 3
```

This article is contributed by **Manjeet Singh**. 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

Operator Functions in Python | Set 2

Operator Functions in Python | Set 1

More functions are discussed in this article.

1. setitem(ob, pos, val) :- This function is used to **assign** the value at a **particular position** in the container.
Operation – **ob[pos] = val**

2. delitem(ob, pos) :- This function is used to **delete** the value at a **particular position** in the container.
Operation – **del ob[pos]**

3. getitem(ob, pos) :- This function is used to **access** the value at a **particular position** in the container.
Operation – **ob[pos]**

```
# Python code to demonstrate working of
# setitem(), delitem() and getitem()

# importing operator module
import operator

# Initializing list
li = [1, 5, 6, 7, 8]

# printing original list
print ("The original list is : ",end="")
for i in range(0,len(li)):
    print (li[i],end=" ")

print ("\r")

# using setitem() to assign 3 at 4th position
operator.setitem(li,3,3)

# printing modified list after setitem()
print ("The modified list after setitem() is : ",end="")
for i in range(0,len(li)):
    print (li[i],end=" ")

print ("\r")

# using delitem() to delete value at 2nd index
operator.delitem(li,1)

# printing modified list after delitem()
print ("The modified list after delitem() is : ",end="")
for i in range(0,len(li)):
    print (li[i],end=" ")

print ("\r")

# using getitem() to access 4th element
print ("The 4th element of list is : ",end="")
print (operator.getitem(li,3))
```

Output:

```
The original list is : 1 5 6 7 8
The modified list after setitem() is : 1 5 6 3 8
The modified list after delitem() is : 1 6 3 8
The 4th element of list is : 8
```

4. setitem(ob, slice(a,b), vals) :- This function is used to **set the values in a particular range** in the container.

Operation – **obj[a:b] = vals**

5. delitem(ob, slice(a,b)) :- This function is used to **delete the values from a particular range** in the container.

Operation – **del obj[a:b]**

6. getitem(ob, slice(a,b)) :- This function is used to **access the values in a particular range** in the container.

Operation – **obj[a:b]**

```
# Python code to demonstrate working of
# setitem(), delitem() and getitem()

# importing operator module
import operator

# Initializing list
li = [1, 5, 6, 7, 8]

# printing original list
print ("The original list is : ",end="")
for i in range(0,len(li)):
    print (li[i],end=" ")

print ("\r")

# using setitem() to assign 2,3,4 at 2nd,3rd and 4th index
operator.setitem(li,slice(1,4),[2,3,4])

# printing modified list after setitem()
print ("The modified list after setitem() is : ",end="")
for i in range(0,len(li)):
    print (li[i],end=" ")

print ("\r")

# using delitem() to delete value at 3rd and 4th index
operator.delitem(li,slice(2,4))

# printing modified list after delitem()
print ("The modified list after delitem() is : ",end="")
for i in range(0,len(li)):
    print (li[i],end=" ")
print ("\r")

# using getitem() to access 1st and 2nd element
print ("The 1st and 2nd element of list is : ",end="")
print (operator.getitem(li,slice(0,2)))
```

Output:

```
The original list is : 1 5 6 7 8
The modified list after setitem() is : 1 2 3 4 8
The modified list after delitem() is : 1 2 8
The 1st and 2nd element of list is : [1, 2]
```

7. concat(ob1,obj2) :- This function is used to **concatenate** two containers.
Operation – **obj1 + obj2**

8. contains(ob1,obj2) :- This function is used to **check if obj2 in present in obj1**.
Operation – **obj2 in obj1**

```
# Python code to demonstrate working of
# concat() and contains()

# importing operator module
import operator

# Initializing string 1
s1 = "geeksfor"

# Initializing string 2
s2 = "geeks"

# using concat() to concatenate two strings
print ("The concatenated string is : ",end="")
print (operator.concat(s1,s2))

# using contains() to check if s1 contains s2
if (operator.contains(s1,s2)):
      print ("geeksfor contains geeks")
else : print ("geeksfor does not contain geeks")
```

Output:

```
The concatenated string is : geeksforgeeks
geeksfor contains geeks
```

9. and_(a,b) :- This function is used to compute **bitwise and** of the mentioned arguments.
Operation – **a & b**

10. or_(a,b) :- This function is used to compute **bitwise or** of the mentioned arguments.
Operation – **a | b**

11. xor(a,b) :- This function is used to compute **bitwise xor** of the mentioned arguments.
Operation – **a ^ b**

12. invert(a) :- This function is used to compute **bitwise inversion** of the mentioned argument.
Operation – **~ a**

```
# Python code to demonstrate working of
# and_(), or_(), xor(), invert()

# importing operator module
import operator

# Initializing a and b

a = 1

b = 0

# using and_() to display bitwise and operation
print ("The bitwise and of a and b is : ",end="")
print (operator.and_(a,b))

# using or_() to display bitwise or operation
print ("The bitwise or of a and b is : ",end="")
print (operator.or_(a,b))

# using xor() to display bitwise exclusive or operation
print ("The bitwise xor of a and b is : ",end="")
print (operator.xor(a,b))

# using invert() to invert value of a
operator.invert(a)
```

```
# printing modified value
print ("The inverted value of a is : ",end="")
print (a)
```

Output:

```
The bitwise and of a and b is : 0
The bitwise or of a and b is : 1
The bitwise xor of a and b is : 1
The inverted value of a is : 1
```

This article is contributed by **Manjeet Singh**. 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

Loops and Control Statements (continue, break and pass) in Python

Python programming language provides following types of loops to handle looping requirements.

While Loop

Syntax :

```
while expression:
   statement(s)
```

In Python, all the statements indented by the same number of character spaces after a programming construct are considered to be part of a single block of code. Python uses indentation as its method of grouping statements.

```
# prints Hello Geek 3 Times
count = 0
while (count < 3):
   count = count+1
   print("Hello Geek")
```

Output:

```
Hello Geek
Hello Geek
Hello Geek
```

See this for an example where while loop is used for iterators. As mentioned in the article, it is not recommended to use while loop for iterators in python.

For in Loop

In Python, there is no C style for loop, i.e., for (i=0; i<n; i++). There is “for in” loop which is similar to for each loop in other languages.
Syntax:

```
for iterator_var in sequence:
   statements(s)
```

It can be used to iterate over iterators and a range.

```
# Iterating over a list
print("List Iteration")
l = ["geeks", "for", "geeks"]
for i in l:
   print(i)

# Iterating over a tuple (immutable)
print("\nTuple Iteration")
t = ("geeks", "for", "geeks")
for i in t:
   print(i)

# Iterating over a String
print("\nString Iteration")
s = "Geeks"
for i in s :
   print(i)

# Iterating over dictionary
print("\nDictionary Iteration")
d = dict()
d['xyz'] = 123
d['abc'] = 345
for i in d :
   print("%s  %d" %(i, d[i]))
```

Output:

```
List Iteration
geeks
for
geeks

Tuple Iteration
geeks
for
geeks

String Iteration
G
e
e
k
s

Dictionary Iteration
xyz  123
abc  345
```

We can use for in loop for user defined iterators. See this for example.

Nested Loops

Python programming language allows to use one loop inside another loop. Following section shows few examples to illustrate the concept.
Syntax:

```
for iterator_var in sequence:
    for iterator_var in sequence:
        statements(s)
        statements(s)
```

The syntax for a nested while loop statement in Python programming language is as follows:

```
while expression:
    while expression:
        statement(s)
        statement(s)
```

A final note on loop nesting is that we can put any type of loop inside of any other type of loop. For example a for loop can be inside a while loop or vice versa.

```
from __future__ import print_function
for i in range(1, 5):
    for j in range(i):
         print(i, end=' ')
    print()
```

Output:

```
1
2 2
3 3 3
4 4 4 4
```

Loop Control Statements

Loop control statements change execution from its normal sequence. When execution leaves a scope, all automatic objects that were created in that scope are destroyed. Python supports the following control statements.

Continue Statement

It returns the control to the beginning of the loop.

```
# Prints all letters except 'e' and 's'
for letter in 'geeksforgeeks':
    if letter == 'e' or letter == 's':
         continue
    print 'Current Letter :', letter
    var = 10
```

Output:

```
Current Letter : g
Current Letter : k
Current Letter : f
Current Letter : o
Current Letter : r
Current Letter : g
Current Letter : k
```

Break Statement

It brings control out of the loop

```
for letter in 'geeksforgeeks':

    # break the loop as soon it sees 'e'
    # or 's'
    if letter == 'e' or letter == 's':
        break

print 'Current Letter :', letter
```

Output:

```
Current Letter : e
```

Pass Statement
We use pass statement to write empty loops. Pass is also used for empty control statement, function and classes.

```
# An empty loop
for letter in 'geeksforgeeks':
    pass
print 'Last Letter :', letter
```

Output:

```
Last Letter : s
```

Exercise:
How to print a list in reverse order (from last to first item) using while and for in loops.

This article is contributed by **Ashirwad Kumar.** 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

Using Iterations in Python Effectively

Prerequisite : Iterators in Python

Following are different ways to use iterators.

C-style approach:

This approach requires prior knowledge of total number of iterations.

```
# A C-style way of accessing list elements
cars = ["Aston", "Audi", "McLaren"]
i = 0
while (i < len(cars)):
    print cars[i]
    i += 1
```

Output:

```
Aston
Audi
McLaren
```

Important Points:

- This style of looping is rarely used by python programmers.
- This 4-step approach creates no compactness with single-view looping construct.
- This is also prone to errors in large-scale programs or designs.
- There is no C-Style for loop in Python, i.e., a loop like for (int i=0; i<n; i++)

Use of for-in (or for each) style:

This style is used in python containing iterator of lists, dictonary, n dimensional-arrays etc. The iterator fetches each component and prints data while looping. The iterator is automatically incremented/decremented in this construct.

```
# Accessing items using for-in loop

cars = ["Aston", "Audi", "McLaren"]
for x in cars:
    print x
```

Output:

```
Aston
Audi
McLaren
```

See this for more examples of different data types.

Indexing using Range function: We can also use indexing using range() in Python.

```
# Accessing items using indexes and for-in

cars = ["Aston", "Audi", "McLaren"]
for i in range(len(cars)):
    print cars[i]
```

Output:

```
Aston
Audi
McLaren
```

Enumerate:

Enumerate is built-in python function that takes input as iterator, list etc and returns a tuple containing index and data at that index in the iterator sequence. For example, enumerate(cars), returns a iterator that will return (0, cars[0]), (1, cars[1]), (2, cars[2]), and so on.

```
# Accessing items using enumerate()

cars = ["Aston" , "Audi", "McLaren "]
for i, x in enumerate(cars):
    print (x)
```

Output :

```
Aston
Audi
McLaren
```

Below solution also works.

```
# Accessing items and indexes enumerate()

cars = ["Aston" , "Audi", "McLaren "]
for x in enumerate(cars):
    print (x[0], x[1])
```

Output :

```
(0, 'Aston')
(1, 'Audi')
(2, 'McLaren ')
```

We can also directly print returned value of enumerate() to see what it returns.

```
# Printing return value of enumerate()

cars = ["Aston" , "Audi", "McLaren "]
print enumerate(cars)
```

Output :

```
[(0, 'Aston'), (1, 'Audi'), (2, 'McLaren ')]
```

Enumerate takes parameter start which is default set to zero. We can change this parameter to any value we like. In the below code we have used start as 1.

```
# demonstrating use of start in enumerate

cars = ["Aston" , "Audi", "McLaren "]
for x in enumerate(cars, start=1):
    print (x[0], x[1])
```

Output :

```
(1, 'Aston')
(2, 'Audi')
(3, 'McLaren ')
```

enumerate() helps to embed solution for accessing each data item in iterator and fetching index of each data item.

Looping extensions:

i) Two iterators for a single looping construct: In this case, a list and dictionary are to be used for each iteration in a single looping block using enumerate function. Let us see example.

```
# Two separate lists
cars = ["Aston", "Audi", "McLaren"]
accessories = ["GPS kit", "Car repair-tool kit"]
```

```
# Single dictionary holds prices of cars and
# its accessories.
# First two items store prices of cars and
# next three items store prices of accessories.
prices = {1:"570000$", 2:"68000$", 3:"450000$",
        4:"890000$", 5:"4500$"}

# Printing prices of cars
for index, c in enumerate(cars, start=1):
    print "Car: %s Price: %s"%(c, prices[index])

# Printing prices of accessories
for index, a in enumerate(accessories,start=1):
    print ("Accessory: %s Price: %s"\
          %(a,prices[index+len(cars)]))
```

Output:

```
Car: Aston Price: 570000$
Car: Audi Price: 68000$
Car: McLaren Price: 450000$
Accessory: GPS kit Price: 890000$
Accessory: Car repair-tool kit Price: 4500$
```

ii) zip function (Both iterators to be used in single looping construct):
This function is helpful to combine similar type iterators(list-list or dict- dict etc) data items at ith position. It uses shortest length of these input iterators. Other items of larger length iterators are skipped. In case of empty iterators it returns No output.

For example, the use of zip for two lists (iterators) helped to combine a single car and its required accessory.

```
# Python program to demonstrate working of zip

# Two separate lists
cars = ["Aston", "Audi", "McLaren"]
accessories = ["GPS", "Car Repair Kit",
             "Dolby sound kit"]

# Combining lists and printing
for c, a in zip(cars, accessories):
    print "Car: %s, Accessory required: %s"\
         %(c, a)
```

Output:

```
Car: Aston, Accessory required: GPS
Car: Audi, Accessory required: Car Repair Kit
Car: McLaren, Accessory required: Dolby sound kit
```

The reverse of getting iterators from zip function is known as unzipping using “*” operator.

Use of enumerate function and zip function helps to achieve an effective extension of iteration logic in python and solves many more sub-problems of a huge task or problem.

```
# Python program to demonstrate unzip (reverse
# of zip)using * with zip function

# Unzip lists
l1,l2 = zip(*[('Aston', 'GPS'),
            ('Audi', 'Car Repair'),
            ('McLaren', 'Dolby sound kit')
          ])

# Printing unzipped lists
print(l1)
print(l2)
```

Output:

```
('Aston', 'Audi', 'McLaren')
('GPS', 'Car Repair', 'Dolby sound kit')
```

References:

1. https://docs.python.org/2/library/functions.html#enumerate
2. https://docs.python.org/2/library/functions.html#zip

This article is contributed by **Krishnasagar Subhedarpage**. 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

Counters in Python | Set 1 (Initialization and Updation)

What is counter?
Counter is a **container** included in the collections module.

What is Container?
Containers are objects that hold objects. They provide a way to access the contained objects and iterate over them. Examples of built in containers are Tuple, list and dictionary. Others are included in Collections module.

A Counter is a subclass of dict. Therefore it is an unordered collection where elements and their respective count are stored as dictionary. This is equivalent to bag or multiset of other languages.

Syntax :
class collections.Counter([iterable-or-mapping])

Initialization :
The constructor of counter can be called in any one of the following ways :

- With sequence of items
- With dictionary containing keys and counts
- With keyword arguments mapping string names to counts

Example of each type of initialization :

```
# A Python program to show different ways to create
# Counter
from collections import Counter

# With sequence of items
print Counter(['B','B','A','B','C','A','B','B','A','C'])

# with dictionary
print Counter({'A':3, 'B':5, 'C':2})

# with keyword arguments
print Counter(A=3, B=5, C=2)
```

Output of all the three lines is same :

```
Counter({'B': 5, 'A': 3, 'C': 2})
Counter({'B': 5, 'A': 3, 'C': 2})
Counter({'B': 5, 'A': 3, 'C': 2})
```

Updation :
We can also create an empty counter in the following manner :

```
coun = collections.Counter()
```

And can be updated via update() method .Syntax for the same :

```
coun.update(Data)
```

```
# A Python program to demonstrate update()
from collections import Counter
coun = Counter()

coun.update([1, 2, 3, 1, 2, 1, 1, 2])
print(coun)

coun.update([1, 2, 4])
print(coun)
```

Output :

```
Counter({1: 4, 2: 3, 3: 1})
Counter({1: 5, 2: 4, 3: 1, 4: 1})
```

- Data can be provided in any of the three ways as mentioned in initialization and the counter's data will be increased not replaced.
- Counts can be zero and negative also.

  ```
  # Python program to demonstrate that counts in
  # Counter can be 0 and negative
  from collections import Counter

  c1 = Counter(A=4,  B=3, C=10)
  c2 = Counter(A=10, B=3, C=4)

  c1.subtract(c2)
  print(c1)
  ```

  Output :

  ```
   Counter({'c': 6, 'B': 0, 'A': -6})
  ```

- We can use Counter to count distinct elements of a list or other collections.

  ```
  # An example program where different list items are
  # counted using counter
  from collections import Counter
  ```

```
# Create a list
z = ['blue', 'red', 'blue', 'yellow', 'blue', 'red']

# Count distinct elements and print Counter abojecct
print(Counter(z))
```

Output:

```
Counter({'blue': 3, 'red': 2, 'yellow': 1})
```

This article is contributed by **Mayank Rawat** .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

Counters in Python | Set 2 (Accessing Counters)

Counters in Python | Set 1 (Initialization and Updation)

Once initialized, counters are accessed just like dictionaries. Also, it does not raise the KeyValue error (if key is not present) instead the value's count is shown as 0.

Example :

```
# Python program to demonstrate accessing of
# Counter elements
from collections import Counter

# Create a list
z = ['blue', 'red', 'blue', 'yellow', 'blue', 'red']
col_count = Counter(z)
print(col_count)

col = ['blue','red','yellow','green']

# Here green is not in col_count
# so count of green will be zero
for color in col:
    print (color, col_count[color])
```

Output:

```
Counter({'blue': 3, 'red': 2, 'yellow': 1})
blue 3
red 2
yellow 1
green 0
```

elements() :
The elements() method returns an iterator that produces all of the items known to the Counter.
Note : Elements with count <= 0 are not included.

Example :

```
# Python example to demonstrate elements() on
# Counter (gives back list)
from collections import Counter

coun = Counter(a=1, b=2, c=3)
print(coun)

print(list(coun.elements()))
```

Output :

```
Counter({'c': 3, 'b': 2, 'a': 1})
['a', 'b', 'b', 'c', 'c', 'c']
```

most_common() :
most_common() is used to produce a sequence of the n most frequently encountered input values and their respective counts.

```
# Python example to demonstrate most_elements() on
# Counter
from collections import Counter

coun = Counter(a=1, b=2, c=3, d=120, e=1, f=219)

# This prints 3 most frequent characters
for letter, count in coun.most_common(3):
    print('%s: %d' % (letter, count))
```

Output :

```
f: 219
d: 120
c: 3
```

This article is contributed by **Mayank Rawat** .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

Iterators in Python

Iterator in python is any python type that can be used with a 'for in loop'. Python lists, tuples, dicts and sets are all examples of inbuilt iterators. These types are iterators because they implement following methods. In fact, any object that wants to be an iterator must implement following methods.

1. **__iter__** method that is called on initialization of an iterator. This should return an object that has a next or __next__ (in Python 3) method.
2. **next (__next__ in Python 3)** The iterator next method should return the next value for the iterable. When an iterator is used with a 'for in' loop, the for loop implicitly calls next() on the iterator object. This method should raise a StopIteration to signal the end of the iteration.

Below is a simple Python program that creates iterator type that iterates from 10 to given limit. For example, if limit is 15, then it prints 10 11 12 13 14 15. And if limit is 5, then it prints nothing.

```
# A simple Python program to demonstrate
# working of iterators using an example type
# that iterates from 10 to given value

# An iterable user defined type
class Test:

    # Cosntructor
    def __init__(self, limit):
        self.limit = limit

    # Called when iteration is initialized
    def __iter__(self):
        self.x = 10
        return self

    # To move to next element. In Python 3,
    # we should replace next with __next__
    def next(self):

        # Store current value ofx
        x = self.x

        # Stop iteration if limit is reached
        if x > self.limit:
            raise StopIteration

        # Else increment and return old value
        self.x = x + 1;
        return x

# Prints numbers from 10 to 15
for i in Test(15):
    print(i)

# Prints nothing
for i in Test(5):
    print(i)
```

Output :

```
10
11
12
13
14
15
```

Examples of inbuilt iterator types.

```
# Sample built-in iterators

# Iterating over a list
print("List Iteration")
l = ["geeks", "for", "geeks"]
for i in l:
    print(i)
```

```
# Iterating over a tuple (immutable)
print("\nTuple Iteration")
t = ("geeks", "for", "geeks")
for i in t:
    print(i)

# Iterating over a String
print("\nString Iteration")
s = "Geeks"
for i in s :
    print(i)

# Iterating over dictionary
print("\nDictionary Iteration")
d = dict()
d['xyz'] = 123
d['abc'] = 345
for i in d :
    print("%s  %d" %(i, d[i]))
```

Output :

```
List Iteration
geeks
for
geeks

Tuple Iteration
geeks
for
geeks

String Iteration
G
e
e
k
s

Dictionary Iteration
xyz  123
abc  345
```

Generators in Python

This article is contributed by **Shwetanshu Rohatgi**. 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

Iterator Functions in Python | Set 1

Perquisite: Iterators in Python

Python in its definition also allows some interesting and useful iterator functions for efficient looping and making execution of the code faster. There are many build-in iterators in the module "**itertools**".
This module implements a number of iterator building blocks.
Some useful Iterators :

1. accumulate(iter, func) :- This iterator takes **two arguments, iterable target and the function which would be followed at each iteration of value in target**. If no function is passed, **addition** takes place by default.If the input iterable is empty, the output iterable will also be empty.

2. chain(iter1, iter2..) :- This function is used to print all the **values in iterable targets** one after another mentioned in its arguments.

```
# Python code to demonstrate the working of
# accumulate() and chain()

# importing "itertools" for iterator operations
import itertools

# importing "operator" for operator operations
import operator

# initializing list 1
li1 = [1, 4, 5, 7]

# initializing list 2
li2 = [1, 6, 5, 9]

# initializing list 3
li3 = [8, 10, 5, 4]

# using accumulate()
# prints the successive summation of elements
```

```
print ("The sum after each iteration is : ",end="")
print (list(itertools.accumulate(li1)))

# using accumulate()
# prints the successive multiplication of elements
print ("The product after each iteration is : ",end="")
print (list(itertools.accumulate(li1,operator.mul)))

# using chain() to print all elements of lists
print ("All values in mentioned chain are : ",end="")
print (list(itertools.chain(li1,li2,li3)))
```

Output:

```
The sum after each iteration is : [1, 5, 10, 17]
The product after each iteration is : [1, 4, 20, 140]
All values in mentioned chain are : [1, 4, 5, 7, 1, 6, 5, 9, 8, 10, 5, 4]
```

3. chain.from_iterable() :- This function is implemented similarly as chain() but the argument here is a **list of lists or any other iterable container**.

4. compress(iter, selector) :- This iterator **selectively picks the values to print** from the passed container **according to the boolean list** value passed as other argument. The arguments **corresponding to boolean true are printed** else all are skipped.

```
# Python code to demonstrate the working of
# chain.from_iterable() and compress()

# importing "itertools" for iterator operations
import itertools

# initializing list 1
li1 = [1, 4, 5, 7]

# initializing list 2
li2 = [1, 6, 5, 9]

# initializing list 3
li3 = [8, 10, 5, 4]

# intializing list of list
li4 = [li1, li2, li3]

# using chain.from_iterable() to print all elements of lists
print ("All values in mentioned chain are : ",end="")
print (list(itertools.chain.from_iterable(li4)))

# using compress() selectively print data values
print ("The compressed values in string are : ",end="")
print (list(itertools.compress('GEEKSFORGEEKS',[1,0,0,0,0,1,0,0,1,0,0,0,0])))
```

Output:

```
All values in mentioned chain are : [1, 4, 5, 7, 1, 6, 5, 9, 8, 10, 5, 4]
The compressed values in string are : ['G', 'F', 'G']
```

5. dropwhile(func, seq) :- This iterator starts printing the characters only **after the func. in argument returns false** for the first time.

6. filterfalse(func, seq) :- As the name suggests, **this iterator prints only values that return false** for the passed function.

```
# Python code to demonstrate the working of
# dropwhile() and filterfalse()

# importing "itertools" for iterator operations
import itertools

# initializing list
li = [2, 4, 5, 7, 8]

# using dropwhile() to start displaying after condition is false
print ("The values after condition returns false : ",end="")
print (list(itertools.dropwhile(lambda x : x%2==0,li)))

# using filterfalse() to print false values
print ("The values that return false to function are : ",end="")
print (list(itertools.filterfalse(lambda x : x%2==0,li)))
```

Output:

```
The values after condition returns false : [5, 7, 8]
The values that return false to function are : [5, 7]
```

Reference: https://docs.python.org/dev/library/itertools.html

This article is contributed by **Manjeet Singh**. 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.

Iterator Functions in Python | Set 2 (islice(), starmap(), tee()..)

Iterator Functions in Python | Set 1

1. islice(iterable, start, stop, step) :- This iterator **selectively prints the values mentioned in its iterable container** passed as argument. This iterator takes **4 arguments, iterable container, starting pos., ending position and step.**

2. starmap(func., tuple list) :- This iterator takes a **function and tuple list** as argument and returns the **value according to the function** from each tuple of list.

```
# Python code to demonstrate the working of
# islice() and starmap()

# importing "itertools" for iterator operations
import itertools

# initializing list
li = [2, 4, 5, 7, 8, 10, 20]

# initializing tuple list
li1 = [ (1, 10, 5), (8, 4, 1), (5, 4, 9), (11, 10 , 1) ]


# using islice() to slice the list acc. to need
# starts printing from 2nd index till 6th skipping 2
print ("The sliced list values are : ",end="")
print (list(itertools.islice(li,1, 6, 2)))

# using starmap() for selection value acc. to function
# selects min of all tuple values
print ("The values acc. to function are : ",end="")
print (list(itertools.starmap(min,li1)))
```

Output:

```
The sliced list values are : [4, 7, 10]
The values acc. to function are : [1, 1, 4, 1]
```

3. takewhile(func, iterable) :- This iterator is opposite of dropwhile(), it prints the values **till the function returns false** for 1st time.

4. tee(iterator, count) :- This iterator **splits the container into a number of iterators** mentioned in the argument.

```
# Python code to demonstrate the working of
# takewhile() and tee()

# importing "itertools" for iterator operations
import itertools

# initializing list
li = [2, 4, 6, 7, 8, 10, 20]

# storing list in iterator
iti = iter(li)

# using takewhile() to print values till condition is false.
print ("The list values till 1st false value are : ",end="")
print (list(itertools.takewhile(lambda x : x%2==0,li )))

# using tee() to make a list of iterators
# makes list of 3 iterators having same values.
it = itertools.tee(iti, 3)

# printing the values of iterators
print ("The iterators are : ")
for i in range (0,3):
    print (list(it[i]))
```

Output:

```
The list values till 1st false value are : [2, 4, 6]
The iterators are :
[2, 4, 6, 7, 8, 10, 20]
[2, 4, 6, 7, 8, 10, 20]
[2, 4, 6, 7, 8, 10, 20]
```

5. zip_longest(iterable1, iterable2, fillval.) :- This iterator prints the **values of iterables alternatively in sequence**. If one of the iterables is printed fully, **remaining values are filled by the values assigned to fillvalue**.

```
# Python code to demonstrate the working of
# zip_longest()

# importing "itertools" for iterator operations
import itertools
```

```
# using zip_longest() to combine two iterables.
print ("The combined values of iterables is  : ")
print (*(itertools.zip_longest('GesoGes','ekfrek',fillvalue='_' )))
```

Output:

```
The combined values of iterables is  :
('G', 'e') ('e', 'k') ('s', 'f') ('o', 'r') ('G', 'e') ('e', 'k') ('s', '_')
```

Combinatoric Iterators

1. product(iter1, iter2) :- This iterator prints the **cartesian product** of the two iterable containers passed as arguments.

2. permutations(iter, group_size) :- This iterator prints all **possible permutation** of all elements of iterable. The **size** of each permuted group is **decided by group_size** argument.

```
# Python code to demonstrate the working of
# product() and permutations()

# importing "itertools" for iterator operations
import itertools

# using product() to print the cartesian product
print ("The cartesian product of the containers is : ")
print (list(itertools.product('AB','12')))

# using permutations to compute all possible permutations
print ("All the permutations of the given container is : ")
print (list(itertools.permutations('GfG',2)))
```

Output:

```
The cartesian product of the containers is :
[('A', '1'), ('A', '2'), ('B', '1'), ('B', '2')]
All the permutations of the given container is :
[('G', 'f'), ('G', 'G'), ('f', 'G'), ('f', 'G'), ('G', 'G'), ('G', 'f')]
```

3. combinations(iterable, group_size) :- This iterator prints **all the possible combinations(without replacement)** of the container passed in arguments in the **specified group size** in **sorted order.**

4. combinations_with_replacement(iterable, group_size) :- This iterator prints **all the possible combinations(with replacement)** of the container passed in arguments in the **specified group size** in **sorted order.**

```
# Python code to demonstrate the working of
# combination() and combination_with_replacement()

# importing "itertools" for iterator operations
import itertools

# using combinations() to print every combination
# (without replacement)
print ("All the combination of container in sorted order(without replacement) is : ")
print (list(itertools.combinations('1234',2)))

# using combinations_with_replacement() to print every combination
# with replacement
print ("All the combination of container in sorted order(with replacement) is : ")
print (list(itertools.combinations_with_replacement('GfG',2)))
```

Output:

```
All the combination of container in sorted order(without replacement) is :
[('1', '2'), ('1', '3'), ('1', '4'), ('2', '3'), ('2', '4'), ('3', '4')]
All the combination of container in sorted order(with replacement) is :
[('G', 'G'), ('G', 'f'), ('G', 'G'), ('f', 'f'), ('f', 'G'), ('G', 'G')]
```

Infinite Iterators

1. count(start, step) :- This iterator **starts printing from the “start” number and prints infinitely**. If steps are mentioned, the numbers are skipped else step is 1 by default.

Example :

```
iterator.count(5,2) prints -- 5,7,9,11...infinitely
```

2. cycle(iterable) :- This iterator prints all values in order from the passed container. It restarts **printing from beginning again when all elements are printed in a cyclic manner**.

Example :

```
iterator.cycle([1,2,3,4]) prints -- 1,2,3,4,1,2,3,4,1...infinitely
```

3. repeat(val, num) :- This iterator **repeatedly prints** the passed value infinite number of times. If num. is mentioned, them till that number.

```
# Python code to demonstrate the working of
# repeat()
# importing "itertools" for iterator operations
import itertools

# using repeat() to repeatedly print number
print ("Printing the numbers repeatedly : ")
print (list(itertools.repeat(25,4)))
```

Output:

```
Printing the numbers repeatedly :
[25, 25, 25, 25]
```

This article is contributed by **Manjeet Singh**. 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

Generators in Python

Prerequisites: Yield Keyword and Iterators

There are two terms involved when we discuss generators.

1. **Generator-Function :** A generator-function is defined like a normal function, but whenever it needs to generate a value, it does so with the yield keyword rather than return. If the body of a def contains yield, the function automatically becomes a generator function.

   ```
   # A generator function that yields 1 for first time,
   # 2 second time and 3 third time
   def simpleGeneratorFun():
       yield 1
       yield 2
       yield 3

   # Driver code to check above generator function
   for value in simpleGeneratorFun():
       print(value)
   ```

   Output :

   ```
   1
   2
   3
   ```

2. **Generator-Object :** Generator functions return a generator object. Generator objects are used either by calling the next method on the generator object or using the generator object in a “for in” loop (as shown in the above program).

   ```
   # A Python program to demonstrate use of
   # generator object with next()

   # A generator function
   def simpleGeneratorFun():
       yield 1
       yield 2
       yield 3

   # x is a generator object
   x = simpleGeneratorFun()

   # Iterating over the generator object using next
   print(x.next()); # In Python 3, __next__()
   print(x.next());
   print(x.next());
   ```

   Output :

   ```
   1
   2
   3
   ```

So a generator function returns an generator object that is iterable, i.e., can be used as an Iterators .

As another example, below is a generator for Fibonacci Numbers.

```
# A simple generator for Fibonacci Numbers
def fib(limit):

    # Initialize first two Fibonacci Numbers
    a, b = 0, 1

    # One by one yield next Fibonacci Number
    while a < limit:
        yield a
```

```
        a, b = b, a + b

# Create a generator object
x = fib(5)

# Iterating over the generator object using next
print(x.next()); # In Python 3, __next__()
print(x.next());
print(x.next());
print(x.next());
print(x.next());

# Iterating over the generator object using for
# in loop.
print("\nUsing for in loop")
for i in fib(5):
    print(i)
```

Output :

```
0
1
1
2
3

Using for in loop
0
1
1
2
3
```

Applications : Suppose we to create a stream of Fibonacci numbers, adopting the generator approach makes it trivial; we just have to call next(x) to get the next Fibonacci number without bothering about where or when the stream of numbers ends.
A more practical type of stream processing is handling large data files such as log files. Generators provide a space efficient method for such data processing as only parts of the file are handled at one given point in time. We can also use Iterators for these purposes, but Generator provides a quick way (We don't need to write __next__ and __iter__ methods here).

Refer below link for more advanced applications of generators in Python.
http://www.dabeaz.com/finalgenerator/

This article is contributed by **Shwetanshu Rohatgi**. 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

Looping Techniques in Python

Python supports various looping techniques by certain inbuilt functions, in various sequential containers. These methods are primarily verya useful in competitive programming and also in various project which require a specific technique with loops maintaining the overall structure of code.

Where they are used ?

Different looping techniques are primarily useful in the places where we don't need to actually manipulate the structure and ordering of overall container, rather only print the elements for a single use instance, no inplace change occurs in the container. This can also be used in instances to save time.

Different looping techniques using Python data structures are:

- **Using enumerate():** enumerate() is used to loop through the containers printing the index number along with the value present in that particular index.
- **Using zip():** zip() is used to combine 2 similar containers(list-list or dict-dict) printing the values sequentially. The loop exists only till the smaller container ends. The detailed explanation of zip() and enumerate() can be found here.
- **Using iteritem():** iteritems() is used to loop through the dictionary printing the dictionary key-value pair sequentially.
- **Using items():** items() performs the similar task on dictionary as iteritems() but have certain disadvantages when compared with iteritems().
    - It is **very time consuming**. Calling it on large dictionaries consumes quite a lot of time.
    - It takes a **lot of memory**. Sometimes takes double the memory when called on dictionary.

```
# python code to demonstrate working of iteritems(),items()

d = { "geeks" : "for", "only" : "geeks" }

# using iteritems to print the dictionary key-value pair
print ("The key value pair using iteritems is : ")
for i,j in d.iteritems():
    print i,j

# using items to print the dictionary key-value pair
print ("The key value pair using items is : ")
for i,j in d.items():
    print i,j
```

Output:

```
The key value pair using iteritems is :
geeks for
only geeks
The key value pair using items is :
geeks for
only geeks
```

- **Using sorted():** sorted() is used to print the **container is sorted order**. It **doesn't sort the container**, but just prints the container in sorted order for 1 instance. Use of **set() can be combined to remove duplicate** occurrences.

```
# python code to demonstrate working of sorted()

# initializing list
lis = [ 1 , 3, 5, 6, 2, 1, 3 ]

# using sorted() to print the list in sorted order
print ("The list in sorted order is : ")
for i in sorted(lis) :
    print (i,end=" ")

print ("\r")

# using sorted() and set() to print the list in sorted order
# use of set() removes duplicates.
print ("The list in sorted order (without duplicates) is : ")
for i in sorted(set(lis)) :
    print (i,end=" ")
```

Output:

```
The list in sorted order is :
1 1 2 3 3 5 6
The list in sorted order (without duplicates) is :
1 2 3 5 6
```

- **Using reversed():** reversed() is used to print the values of **container in the descending order** as declared.

```
# python code to demonstrate working of reversed()

# initializing list
lis = [ 1 , 3, 5, 6, 2, 1, 3 ]


# using revered() to print the list in reversed order
print ("The list in reversed order is : ")
for i in reversed(lis) :
    print (i,end=" ")
```

Output:

```
The list in reversed order is :
3 1 2 6 5 3 1
```

Advantage of using above techniques over for, while loop

- These techniques are quick to use and reduces coding effort. for, while loops needs the entire structure of container to be changed.
- These Looping techniques do not require any structural changes to container. They have keywords which present the exact purpose of usage. Whereas, no pre-predictions or guesses can be made in for, while loop i.e not easily understandable the purpose at a glance.
- Looping technique make the code more concise than using for, while loopings.

References : https://docs.python.org/3/tutorial/datastructures.html#looping-techniques
This article is contributed by **Manjeet Singh**. 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

Programs for printing pyramid patterns in Python

Patterns can be printed in python using simple for loops. **First outer loop** is used to handle **number of rows** and **Inner nested loop** is used to handle the **number of columns**. Manipulating the print statements, different number patterns, alphabet patterns or star patterns can be printed.
Some of the Patterns are shown in this article.

- **Simple pyramid pattern**

```
# Python 3.x code to demonstrate star pattern

# Function to demonstrate printing pattern
def pypart(n):
    # outer loop to handle number of rows
    # n in this case
    for i in range(0, n):

        # inner loop to handle number of columns
        # values changing acc. to outer loop
        for j in range(0, i+1):

            # printing stars
            print("* ",end="")

        # ending line after each row
        print("\r")

# Driver Code
n = 5
pypart(n)
```

Output:

```
*
* *
* * *
* * * *
* * * * *
```

- **After 180 degree rotation**

```
# Python 3.x code to demonstrate star pattern

# Function to demonstrate printing pattern
def pypart2(n):

    # number of spaces
    k = 2*n - 2

    # outer loop to handle number of rows
    for i in range(0, n):

        # inner loop to handle number spaces
        # values changing acc. to requirement
        for j in range(0, k):
            print(end=" ")

        # decrementing k after each loop
        k = k - 2

        # inner loop to handle number of columns
        # values changing acc. to outer loop
        for j in range(0, i+1):

            # printing stars
            print("* ", end="")

        # ending line after each row
        print("\r")

# Driver Code
n = 5
pypart2(n)
```

Output:

```
        *
      * *
    * * *
  * * * *
* * * * *
```

- **Printing Triangle**

```
# Python 3.x code to demonstrate star pattern

# Function to demonstrate printing pattern triangle
def triangle(n):

    # number of spaces
    k = 2*n - 2

    # outer loop to handle number of rows
    for i in range(0, n):

        # inner loop to handle number spaces
        # values changing acc. to requirement
        for j in range(0, k):
            print(end=" ")

        # decrementing k after each loop
        k = k - 1
        # inner loop to handle number of columns
        # values changing acc. to outer loop
        for j in range(0, i+1):
        
            # printing stars
            print("* ", end="")
        
        # ending line after each row
        print("\r")

# Driver Code
n = 5
triangle(n)
```

Output:

```
    *
   * *
  * * *
 * * * *
* * * * *
```

- **Number Pattern**

```
# Python 3.x code to demonstrate star pattern

# Function to demonstrate printing pattern of numbers
def numpat(n):

    # initialising starting number
    num = 1

    # outer loop to handle number of rows
    for i in range(0, n):

        # re assigning num
        num = 1

        # inner loop to handle number of columns
            # values changing acc. to outer loop
        for j in range(0, i+1):

                # printing number
            print(num, end=" ")

            # incrementing number at each column
            num = num + 1

        # ending line after each row
        print("\r")

# Driver code
n = 5
numpat(n)
```

Output:

```
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
```

- **Numbers without re assigning**

```
# Python 3.x code to demonstrate star pattern

# Function to demonstrate printing pattern of numbers
def contnum(n):

    # initializing starting number
    num = 1

    # outer loop to handle number of rows
    for i in range(0, n):

        # not re assigning num
        # num = 1

        # inner loop to handle number of columns
        # values changing acc. to outer loop
        for j in range(0, i+1):

            # printing number
            print(num, end=" ")

            # incrementing number at each column
            num = num + 1
        # ending line after each row
        print("\r")

n = 5

# sending 5 as argument
# calling Function
contnum(n)
```

Output:

```
1
2 3
4 5 6
7 8 9 10
11 12 13 14 15
```

- **Character Pattern**

```
# Python 3.x code to demonstrate star pattern

# Function to demonstrate printing pattern of alphabets
def alphapat(n):

    # initializing value corresponding to 'A'
    # ASCII value
    num = 65

    # outer loop to handle number of rows
    # 5 in this case
    for i in range(0, n):

        # inner loop to handle number of columns
        # values changing acc. to outer loop
        for j in range(0, i+1):

            # explicitely converting to char
            ch = chr(num)

            # printing char value
            print(ch, end=" ")

        # incrementing number
        num = num + 1

        # ending line after each row
        print("\r")

# Driver Code
n = 5
alphapat(n)
```

Output:

```
A
B B
C C C
D D D D
E E E E E
```

- **Continuous Character pattern**

```
# Python code 3.x to demonstrate star pattern

# Function to demonstrate printing pattern of alphabets
def  contalpha(n):

    # initializing value corresponding to 'A'
    # ASCII value
    num = 65

    # outer loop to handle number of rows
-   for i in range(0, n):

        # inner loop to handle number of columns
        # values changing acc. to outer loop
        for j in range(0, i+1):

            # explicitely converting to char
            ch = chr(num)

            # printing char value
            print(ch, end=" ")

            # incrementing at each column
            num = num +1
        # ending line after each row
        print("\r")

# Driver code
n = 5
contalpha(n)
```

Output:

```
A
B C
D E F
G H I J
K L M N O
```

This article is contributed by **Manjeet Singh(S.Nupur)** . 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
pattern-printing

range() vs xrange() in Python

range() and xrange() are two functions that could be used to iterate a certain number of times in for loops in Python. In Python 3, there is no xrange , but the range function behaves like xrange in Python 2.If you want to write code that will run on both Python 2 and Python 3, you should use 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**".

Both are implemented in different ways and have different characteristics associated with them. The points of comparisons are:

- Return Type
- Memory
- Operation Usage
- Speed

Return Type

range() returns – the **list** as return type.
xrange() returns – **xrange()** object.

```
# Python code to demonstrate range() vs xrange()
# on  basis of return type

# initializing a with range()
a = range(1,10000)

# initializing a with xrange()
x = xrange(1,10000)

# testing the type of a
print ("The return type of range() is : ")
print (type(a))

# testing the type of x
print ("The return type of xrange() is : ")
print (type(x))
```

Output:

```
The return type of range() is :

The return type of xrange() is :
```

Memory

The variable storing the **range** created by range() **takes more memory** as compared to variable storing the range using xrange(). The basic reason for this is the return type of range() is list and xrange() is xrange() object.

```
# Python code to demonstrate range() vs xrange()
# on  basis of memory

import sys

# initializing a with range()
a = range(1,10000)

# initializing a with xrange()
x = xrange(1,10000)
```

```
# testing the size of a
# range() takes more memory
print ("The size allotted using range() is : ")
print (sys.getsizeof(a))

# testing the size of a
# range() takes less memory
print ("The size allotted using xrange() is : ")
print (sys.getsizeof(x))
```

Output:

```
The size allotted using range() is :
80064
The size allotted using xrange() is :
40
```

Operations usage

As range() returns the list, all the operations that **can** be applied on the list can be used on it. On the other hand, as xrange() returns the xrange object, operations associated to list **cannot** be applied on them, hence a disadvantage.

```
# Python code to demonstrate range() vs xrange()
# on  basis of operations usage

# initializing a with range()
a = range(1,6)

# initializing a with xrange()
x = xrange(1,6)

# testing usage of slice operation on range()
# prints without error
print ("The list after slicing using range is : ")
print (a[2:5])

# testing usage of slice operation on xrange()
# raises error
print ("The list after slicing using xrange is : ")
print (x[2:5])
```

Error:

```
Traceback (most recent call last):
  File "1f2d94c59aea6aed795b05a19e44474d.py", line 18, in
    print (x[2:5])
TypeError: sequence index must be integer, not 'slice'
```

Output:

```
The list after slicing using range is :
[3, 4, 5]
The list after slicing using xrange is :
```

Speed

Because of the fact that xrange() evaluates only the generator object containing only the values that are required by lazy evaluation, therefore is **faster** in implementation than range().

Important Points:

- If you want to write code that will run on both Python 2 and Python 3, use range() as the xrange funtion is deprecated in Python 3
- range() is faster if iterating over the same sequence multiple times.
- xrange() has to reconstruct the integer object every time, but range() will have real integer objects. (It will always perform worse in terms of memory however)

This article is contributed by **Manjeet Singh**. 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

Function Decorators in Python | Set 1 (Introduction)

Background

Following are important facts about functions in Python that are useful to understand decorator functions.

1. In Python, we can define a function inside another function.
2. In Python, a function can be passed as parameter to another function (a function can also return another function).

```
# A Python program to demonstrate that a function
# can be defined inside another function and a
# function can be passed as parameter.

# Adds a welcome message to the string
def messageWithWelcome(str):

    # Nested function
    def addWelcome():
        return "Welcome to "

    # Return concatenation of addWelcome()
    # and str.
    return  addWelcome() + str

# To get site name to which welcome is added
def site(site_name):
    return site_name

print messageWithWelcome(site("GeeksforGeeks"))
```

Output:

```
Welcome to GeeksforGeeks
```

Function Decorator

A decorator is a function that takes a function as its only parameter and returns a function. This is helpful to “wrap” functionality with the same code over and over again. For example, above code can be re-written as following.
We use @func_name to specify a decorator to be applied on another function.

```
# Adds a welcome message to the string
# returned by fun(). Takes fun() as
# parameter and returns welcome().
def decorate_message(fun):

    # Nested function
    def addWelcome(site_name):
        return "Welcome to " + fun(site_name)

    # Decorator returns a function
    return addWelcome

@decorate_message
def site(site_name):
    return site_name;

# Driver code

# This call is equivalent to call to
# decorate_message() with function
# site("GeeksforGeeks") as parameter
print site("GeeksforGeeks")
```

Output:

```
Welcome to GeeksforGeeks
```

Decorators can also be useful to attach data (or add attribute) to functions.

```
# A Python example to demonstrate that
# decorators can be useful attach data

# A decorator function to attach
# data to func
def attach_data(func):
       func.data = 3
       return func

@attach_data
def add (x, y):
       return x + y

# Driver code

# This call is equivalent to attach_data()
# with add() as parameter
print(add(2, 3))

print(add.data)
```

Output:

```
5
3
```

‘add()’ returns sum of x and y passed as arguments but it is wrapped by a decorator function, calling add(2, 3) would simply give sum of two numbers but when we call add.data then ‘add’ function is passed into then decorator function ‘attach_data’ as argument and this function returns ‘add’ function with an attribute ‘data’ that is set to 3 and hence prints it.

Python decorators are a powerful tool to remove redundancy.

This article is contributed by **Shwetanshu Rohatgi**. 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

How to write an empty function in Python – pass statement?

In C/C++ and Java, we can write empty function as following

```
// An empty function in C/C++/Java
void fun() {  }
```

In Python, if we write something like following in Python, it would produce compiler error.

```
# Incorrect empty function in Python
def fun():
```

Output :

```
IndentationError: expected an indented block
```

In Python, to write empty functions, we use **pass** statement. pass is a special statement in Python that does nothing. It only works as a dummy statement.

```
# Correct way of writing empty function
# in Python
def fun():
    pass
```

We can use pass in empty while statement also.

```
# Empty loop in Python
mutex = True
while (mutex == True) :
    pass
```

We can use pass in empty if else statements.

```
# Empty in if/else in Python
mutex = True
if (mutex == True) :
    pass
else :
    print("False")
```

This article is contributed by **Shivam Gupta**. 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

When to use yield instead of return in Python?

The yield statement suspends function's execution and sends a value back to caller, but retains enough state to enable function to resume where it is left off. When resumed, the function continues execution immediately after the last yield run. This allows its code to produce a series of values over time, rather them computing them at once and sending them back like a list.

Let's see with an example:

```
# A Simple Python program to demonstrate working
# of yield

# A generator function that yields 1 for first time,
# 2 second time and 3 third time
def simpleGeneratorFun():
    yield 1
    yield 2
    yield 3

# Driver code to check above generator function
for value in simpleGeneratorFun():
    print(value)
```

Output:

```
1
```

```
2
3
```

Return sends a specified value back to its caller whereas **Yield** can produce a sequence of values. We should use yield when we want to iterate over a sequence, but don't want to store the entire sequence in memory.

Yield are used in Python **generators**. A generator function is defined like a normal function, but whenever it needs to generate a value, it does so with the yield keyword rather than return. If the body of a def contains yield, the function automatically becomes a generator function.

```
# A Python program to generate squares from 1
# to 100 using yield and therefore generator

# An infinite generator function that prints
# next square number. It starts with 1
def nextSquare():
    i = 1;

    # An Infinite loop to generate squares
    while True:
        yield i*i
        i += 1  # Next execution resumes
                # from this point

# Driver code to test above generator
# function
for num in nextSquare():
    if num > 100:
         break
    print(num)
```

Output:

```
1
4
9
16
25
36
49
64
81
100
```

This article is contributed by **Arpit Agarwal**. 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
Python

Returning Multiple Values in Python

In Python, we can return multiple values from a function. Following are different ways

1) Using Object: This is similar to C/C++ and Java, we can create a class (in C, struct) to hold multiple values and return an object of the class.

```
# A Python program to to return multiple
# values from a method using class
class Test:
    def __init__(self):
        self.str = "geeksforgeeks"
        self.x = 20

# This function returns an object of Test
def fun():
    return Test()

# Driver code to test above method
t = fun()
print(t.str)
print(t.x)
```

Output:

```
geeksforgeeks
20
```

Below are interesting methods for somebody shifting C++/Java world.

2) Using Tuple: A Tuple is a comma separated sequence of items. It is created with or without (). Tuples are immutable. See this for details of tuple and list.

```
# A Python program to to return multiple
# values from a method using tuple

# This function returns a tuple
def fun():
    str = "geeksforgeeks"
    x   = 20
    return str, x;  # Return tuple, we could also
                    # write (str, x)

# Driver code to test above method
str, x = fun() # Assign returned tuple
print(str)
print(x)
```

Output:

```
geeksforgeeks
20
```

3) Using a list: A list is like an array of items created using square brackets. They are different from arrays as they can contain items of different types. Lists are different from tuples as they mutable.

```
# A Python program to to return multiple
# values from a method using list

# This function returns a list
def fun():
    str = "geeksforgeeks"
    x = 20
    return [str, x];

# Driver code to test above method
list = fun()
print(list)
```

Output:

```
['geeksforgeeks', 20]
```

4) Using a Dictionary: A Dictionary is similar to hash or map in other languages. See this for details of dictionary.

```
# A Python program to to return multiple
# values from a method using dictionary

# This function returns a dictionary
def fun():
    d = dict();
    d['str'] = "GeeksforGeeks"
    d['x']   = 20
    return d

# Driver code to test above method
d = fun()
print(d)
```

Output:

```
{'x': 20, 'str': 'GeeksforGeeks'}
```

Reference:
http://stackoverflow.com/questions/354883/how-do-you-return-multiple-values-in-python

This article is contributed by **Shubham Agrawal**. 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
Python

Precision Handling in Python

Python in its definition allows to handle precision of floating point numbers in several ways using different functions. Most of them are defined under the “**math**” module. Some of the most used operations are discussed in this article.

1. trunc() :- This function is used to **eliminate all decimal part** of the floating point number and return the integer without the decimal part.

2. ceil() :- This function is used to print the **least integer greater than the given number**.

3. floor() :- This function is used to print the **greatest integer smaller than the given integer**.

```
# Python code to demonstrate ceil(), trunc()
# and floor()

# importing "math" for precision function
import math

# initializing value
a = 3.4536

# using trunc() to print integer after truncating
print ("The integral value of number is : ",end="")
print (math.trunc(a))

# using ceil() to print number after ceiling
print ("The smallest integer greater than number is : ",end="")
print (math.ceil(a))

# using floor() to print number after flooring
print ("The greatest integer smaller than number is : ",end="")
print (math.floor(a))
```

Output :

```
The integral value of number is : 3
The smallest integer greater than number is : 4
The greatest integer smaller than number is : 3
```

Setting Precision

There are many ways to set precision of floating point value. Some of them is discussed below.

1. Using "%" :- "%" operator is used to format as well as set precision in python. This is similar to "printf" statement in C programming.

2. Using format() :- This is yet another way to format the string for setting precision.

3. Using round(x,n) :- This function takes 2 arguments, **number and the number till which we want decimal part rounded.**

```
# Python code to demonstrate precision
# and round()

# initializing value
a = 3.4536

# using "%" to print value till 2 decimal places
print ("The value of number till 2 decimal place(using %) is : ",end="")
print ('%.2f'%a)

# using format() to print value till 2 decimal places
print ("The value of number till 2 decimal place(using format()) is : ",end="")
print ("{0:.2f}".format(a))

# using round() to print value till 2 decimal places
print ("The value of number till 2 decimal place(using round()) is : ",end="")
print (round(a,2))
```

Output :

```
The value of number till 2 decimal place(using %) is : 3.45
The value of number till 2 decimal place(using format()) is : 3.45
The value of number till 2 decimal place(using round()) is : 3.45
```

This article is contributed by **Manjeet Singh**. 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

Python Modules

A module is a file containing Python definitions and statements. A module can define functions, classes and variables. A module can also include runnable code. Grouping related code into a module makes the code easier to understand and use.

Example:

```
# A simple module, calc.py

def add(x, y):
    return (x+y)
```

```
def subtract(x, y):
    return (x-y)
```

The *import* statement

We can use any Python source file as a module by executing an import statement in some other Python source file.

When interpreter encounters an import statement, it imports the module if the module is present in the search path. A search path is a list of directories that the interpreter searches for importing a module. For example, to import the module calc.py, we need to put the following command at the top of the script :

```
# importing  module calc.py
import calc

print add(10, 2)
```

Output:

```
12
```

The *from import* Statement

Python's *from* statement lets you import specific attributes from a module. The *from .. import ..* has the following syntax :

```
# importing sqrt() and factorial from the
# module math
from math import sqrt, factorial

# if we simply do "import math", then
# math.sqrt(16) and math.factorial()
# are required.
print sqrt(16)
print factorial(6)
```

Output:

```
4.0
720
```

The dir() function

The dir() built-in function returns a sorted list of strings containing the names defined by a module. The list contains the names of all the modules, variables and functions that are defined in a module.

```
#  Import built-in module  random
import  random
print  dir(math)
```

Output:

```
['BPF', 'LOG4', 'NV_MAGICCONST', 'RECIP_BPF', 'Random',
'SG_MAGICCONST', 'SystemRandom', 'TWOPI', 'WichmannHill',
'_BuiltinMethodType', '_MethodType', '__all__',
'__builtins__', '__doc__', '__file__', '__name__',
'__package__', '_acos', '_ceil', '_cos', '_e', '_exp',
'_hashlib', '_hexlify', '_inst', '_log', '_pi', '_random',
'_sin', '_sqrt', '_test', '_test_generator', '_urandom',
'_warn', 'betavariate', 'choice', 'division',
'expovariate', 'gammavariate', 'gauss', 'getrandbits',
'getstate', 'jumpahead', 'lognormvariate', 'normalvariate',
'paretovariate', 'randint', 'random', 'randrange',
'sample', 'seed', 'setstate', 'shuffle', 'triangular',
'uniform', 'vonmisesvariate', 'weibullvariate']
```

Code Snippet illustrating python built-in modules:

```
# importing built-in module math
import math

# using square root(sqrt) function contained
# in math module
print math.sqrt(25)

# using pi function contained in math module
print math.pi

# 2 radians = 114.59 degreees
print math.degrees(2)

# 60 degrees = 1.04 radians
print math.radians(60)

# Sine of 2 radians
print math.sin(2)
```

```
# Cosine of 0.5 radians
print math.cos(0.5)

# Tangent of 0.23 radians
print math.tan(0.23)

# 1 * 2 * 3 * 4 = 24
print math.factorial(4)


# importing built in module random
import random

# printing random integer between 0 and 5
print random.randint(0, 5)

# print random floating point number between 0 and 1
print random.random()

# random number between 0 and 100
print random.random() * 100

List = [1, 4, True, 800, "python", 27, "hello"]

# using choice function in random module for choosing
# a random element from a set such as a list
print random.choice(List)


# importing built in module datetime
import datetime
from datetime import date
import time

# Returns the number of seconds since the
# Unix Epoch, January 1st 1970
print time.time()

# Converts a number of seconds to a date object
print date.fromtimestamp(454554)
```

Output:

```
5.0
3.14159265359
114.591559026
1.0471975512
0.909297426826
0.87758256189
0.234143362351
24
3
0.401533172951
88.4917616788
True
1461425771.87
1970-01-06
```

This article is contributed by **Gaurav Shrestha**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above. 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.

GATE CS Corner   Company Wise Coding Practice

Python

Mathematical Functions in Python | Set 1 (Numeric Functions)

In python a number of mathematical operations can be performed with ease by importing a module named “math” which defines various functions which makes our tasks easier.

1. ceil() :- This function returns the **smallest integral value greater than the number**. If number is already integer, same number is returned.

2. floor() :- This function returns the **greatest integral value smaller than the number**. If number is already integer, same number is returned.

```
# Python code to demonstrate the working of
# ceil() and floor()

# importing "math" for mathematical operations
import math

a = 2.3

# returning the ceil of 2.3
print ("The ceil of 2.3 is : ", end="")
print (math.ceil(a))
```

```
# returning the floor of 2.3
print ("The floor of 2.3 is : ", end="")
print (math.floor(a))
```

Output:

```
The ceil of 2.3 is : 3
The floor of 2.3 is : 2
```

3. fabs() :- This function returns the **absolute value** of the number.

4. factorial() :- This function returns the **factorial** of the number. An error message is displayed if number is not integral.

```
# Python code to demonstrate the working of
# fabs() and factorial()

# importing "math" for mathematical operations
import math

a = -10

b= 5

# returning the absolute value.
print ("The absolute value of -10 is : ", end="")
print (math.fabs(a))

# returning the factorial of 5
print ("The factorial of 5 is : ", end="")
print (math.factorial(b))
```

Output:

```
The absolute value of -10 is : 10.0
The factorial of 5 is : 120
```

5. copysign(a, b) :- This function returns the number with the **value of 'a' but with the sign of 'b'**. The returned value is float type.

6. gcd() :- This function is used to compute the **greatest common divisor of 2 numbers** mentioned in its arguments. This function works in python 3.5 and above.

```
# Python code to demonstrate the working of
# copysign() and gcd()

# importing "math" for mathematical operations
import math

a = -10
b = 5.5
c = 15
d = 5

# returning the copysigned value.
print ("The copysigned value of -10 and 5.5 is : ", end="")
print (math.copysign(5.5, -10))

# returning the gcd of 15 and 5
print ("The gcd of 5 and 15 is : ", end="")
print (math.gcd(5,15))
```

Output:

```
The copysigned value of -10 and 5.5 is : -5.5
The gcd of 5 and 15 is : 5
```

This article is contributed by **Manjeet Singh** .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

Mathematical Functions in Python | Set 2 (Logarithmic and Power Functions)

Numeric functions are discussed in set 1 below

Mathematical Functions in Python | Set 1 (Numeric Functions)

Logarithmic and power functions are discussed in this set.

1. exp(a) :- This function returns the value of **e raised to the power a (e**a)** .

2. log(a, b) :- This function returns the logarithmic **value of a with base b**. If base is not mentioned, the computed value is of natural log.

```
# Python code to demonstrate the working of
# exp() and log()

# importing "math" for mathematical operations
import math

# returning the exp of 4
print ("The e**4 value is : ", end="")
print (math.exp(4))

# returning the log of 2,3
print ("The value of log 2 with base 3 is : ", end="")
print (math.log(2,3))
```

Output:

```
The e**4 value is : 54.598150033144236
The value of log 2 with base 3 is : 0.6309297535714574
```

3. log2(a) :- This function computes value of **log a with base 2**. This value is **more accurate** than the value of the function discussed above.

4. log10(a) :- This function computes value of **log a with base 10**. This value is **more accurate** than the value of the function discussed above.

```
# Python code to demonstrate the working of
# log2() and log10()

# importing "math" for mathematical operations
import math

# returning the log2 of 16
print ("The value of log2 of 16 is : ", end="")
print (math.log2(16))

# returning the log10 of 10000
print ("The value of log10 of 10000 is : ", end="")
print (math.log10(10000))
```

Output:

```
The value of log2 of 16 is : 4.0
The value of log10 of 10000 is : 4.0
```

5. pow(a, b) :- This function is used to compute value of **a raised to the power b (a**b)**.

6. sqrt() :- This function returns the **square root** of the number.

```
# Python code to demonstrate the working of
# pow() and sqrt()

# importing "math" for mathematical operations
import math

# returning the value of 3**2
print ("The value of 3 to the power 2 is : ", end="")
print (math.pow(3,2))

# returning the square root of 25
print ("The value of square root of 25 : ", end="")
print (math.sqrt(25))
```

Output:

```
The value of 3 to the power 2 is : 9.0
The value of square root of 25 : 5.0
```

This article is contributed by **Manjeet Singh** .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

Mathematical Functions in Python | Set 3 (Trigonometric and Angular Functions)

Some of the mathematical functions are discussed in below set 1 and set 2

Mathematical Functions in Python | Set 1 (Numeric Functions)
Mathematical Functions in Python | Set 2 (Logarithmic and Power Functions)

Trigonometric and angular functions are discussed in this article.

1. sin() :- This function returns the **sine** of value passed as argument. The value passed in this function should be in **radians**.

2. cos() :- This function returns the **cosine** of value passed as argument. The value passed in this function should be in **radians**.

```
# Python code to demonstrate the working of
# sin() and cos()

# importing "math" for mathematical operations
import math

a = math.pi/6

# returning the value of sine of pi/6
print ("The value of sine of pi/6 is : ", end="")
print (math.sin(a))

# returning the value of cosine of pi/6
print ("The value of cosine of pi/6 is : ", end="")
print (math.cos(a))
```

Output:

```
The value of sine of pi/6 is : 0.49999999999999994
The value of cosine of pi/6 is : 0.8660254037844387
```

3. tan() :- This function returns the **tangent** of value passed as argument. The value passed in this function should be in **radians**.

4. hypot(a, b) :- This returns the **hypotenuse** of the values passed in arguments. Numerically, it returns the value of **sqrt(a*a + b*b)**.

```
# Python code to demonstrate the working of
# tan() and hypot()

# importing "math" for mathematical operations
import math

a = math.pi/6
b = 3
c = 4

# returning the value of tangent of pi/6
print ("The value of tangent of pi/6 is : ", end="")
print (math.tan(a))

# returning the value of hypotenuse of 3 and 4
print ("The value of hypotenuse of 3 and 4 is : ", end="")
print (math.hypot(b,c))
```

Output:

```
The value of tangent of pi/6 is : 0.5773502691896257
The value of hypotenuse of 3 and 4 is : 5.0
```

5. degrees() :- This function is used to **convert argument value from radians to degrees**.

6. radians() :- This function is used to **convert argument value from degrees to radians**.

```
# Python code to demonstrate the working of
# degrees() and radians()

# importing "math" for mathematical operations
import math

a = math.pi/6
b = 30

# returning the converted value from radians to degrees
print ("The converted value from radians to degrees is : ", end="")
print (math.degrees(a))

# returning the converted value from degrees to radians
print ("The converted value from degrees to radians is : ", end="")
print (math.radians(b))
```

Output:

```
The converted value from radians to degrees is : 29.999999999999996
The converted value from degrees to radians is : 0.5235987755982988
```

This article is contributed by **Manjeet Singh**. 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.

Mathematical Functions in Python | Set 4 (Special Functions and Constants)

Some of the mathematical functions are discussed in below set 1, set 2 and set 3
Mathematical Functions in Python | Set 1 (Numeric Functions)
Mathematical Functions in Python | Set 2 (Logarithmic and Power Functions)
Mathematical Functions in Python | Set 3 (Trigonometric and Angular Functions)

Special Functions and constants are discussed in this article.

1. gamma() :- This function is used to return the **gamma function** of the argument.

```
# Python code to demonstrate the working of
# gamma()

# importing "math" for mathematical operations
import math

a = 4

# returning the gamma() of 4
print ("The gamma() of 4 is : ", end="")
print (math.gamma(a))
```

Output:

```
The gamma() of 4 is : 6.0
```

2. pi :- This is an inbuilt constant that outputs the **value of pi(3.141592)**.

3. e :- This is an inbuilt constant that outputs the **value of e(2.718281)**.

```
# Python code to demonstrate the working of
# const. pi and e

# importing "math" for mathematical operations
import math

# returning the value of const. pi
print ("The value of const. pi is : ", end="")
print (math.pi)

# returning the value of const. e
print ("The value of const. e is : ", end="")
print (math.e)
```

Output:

```
The value of const. pi is : 3.141592653589793
The value of const. e is : 2.718281828459045
```

4. inf :- This is a **positive floating point infinity constant**. -inf is used to denote the negative floating point infinity. This constant is defined in python 3.5 and above.

5. isinf() :- This function is used to **check** whether the value is an **infinity or not.**

6. nan :- This constant denotes "Not a number" in python. This constant is defined in python 3.5 and above.

7. isnan() :- This function returns **true if the number is "nan"** else returns false.

```
# Python code to demonstrate the working of
# inf, nan, isinf(), isnan()

# importing "math" for mathematical operations
import math

# checking if number is nan
if (math.isnan(math.nan)):
    print ("The number is nan")
else : print ("The number is not nan")

# checking if number is positive infinity
if (math.isinf(math.inf)):
    print ("The number is positive infinity")
else : print ("The number is not positive infinity")
```

Output:

```
The number is nan
The number is positive infinity
```

This article is contributed by **Manjeet Singh**. 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

Inplace Operators in Python | Set 1 (iadd(), isub(), iconcat()…)

Python in its definition provides methods to perform inplace operations, i.e **doing assignment and computation in a single statement** using “**operator**” module. For example,

```
x += y is equivalent to x = operator.iadd(x, y)
```

Some Important Inplace operations :

1. iadd() :- This function is used to **assign and add the current value**. This operation does “**a+=b**” operation. Assigning is **not** performed in case of immutable containers, such as strings, numbers and tuples.

2. iconcat() :- This function is used to **concat** one string at end of second.

```
# Python code to demonstrate the working of
# iadd() and iconcat()

# importing operator to handle operator operations
import operator

# using iadd() to add and assign value
x = operator.iadd(2, 3);

# printing the modified value
print ("The value after adding and assigning : ", end="")
print (x)

# initializing values
y = "geeks"

z = "forgeeks"

# using iconcat() to concat the sequences
y = operator.iconcat(y, z)

# using iconcat() to concat sequences
print ("The string after concatenation is : ", end="")
print (y)
```

Output:

```
The value after adding and assigning : 5
The string after concatenation is : geeksforgeeks
```

3. isub() :- This function is used to **assign and subtract the current value**. This operation does “**a-=b**” operation. Assigning is **not** performed in case of immutable containers, such as strings, numbers and tuples.

4. imul() :- This function is used to **assign and multiply the current value**. This operation does “**a*=b**” operation. Assigning is **not** performed in case of immutable containers, such as strings, numbers and tuples.

```
# Python code to demonstrate the working of
# isub() and imul()

# importing operator to handle operator operations
import operator

# using isub() to subtract and assign value
x = operator.isub(2, 3);

# printing the modified value
print ("The value after subtracting and assigning : ", end="")
print (x)

# using imul() to multiply and assign value
x = operator.imul(2, 3);

# printing the modified value
print ("The value after multiplying and assigning : ", end="")
print (x)
```

Output:

```
The value after subtracting and assigning : -1
The value after multiplying and assigning : 6
```

5. itruediv() :- This function is used to **assign and divide the current value**. This operation does “**a/=b**” operation. Assigning is **not** performed in case of immutable containers, such as strings, numbers and tuples.

6. imod() :- This function is used to **assign and return remainder** . This operation does “**a%=b**” operation. Assigning is **not** performed in case of immutable containers, such as strings, numbers and tuples.

```
# Python code to demonstrate the working of
# itruediv() and imod()

# importing operator to handle operator operations
import operator

# using itruediv() to divide and assign value
x = operator.itruediv(10, 5);
```

```
# printing the modified value
print ("The value after dividing and assigning : ", end="")
print (x)

# using imod() to modulus and assign value
x = operator.imod(10, 6);

# printing the modified value
print ("The value after modulus and assigning : ", end="")
print (x)
```

Output:

```
The value after dividing and assigning : 2.0
The value after modulus and assigning : 4
```

Next Articles

- Inplace Operators in Python | Set 2
- Inplace vs Standard Operators

This article is contributed by **Manjeet Singh**. 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

Inplace Operators in Python | Set 2 (ixor(), iand(), ipow(),…)

Inplace Operators in Python | Set 1(iadd(), isub(), iconcat()…)

More functions are discussed in this articles.

1. ixor() :- This function is used to **assign and xor the current value**. This operation does “**a^ = b**” operation. Assigning is **not** performed in case of immutable containers, such as strings, numbers and tuples.

2. ipow() :- This function is used to **assign and exponentiate the current value**. This operation does “**a ** = b**” operation. Assigning is **not** performed in case of immutable containers, such as strings, numbers and tuples.

```
# Python code to demonstrate the working of
# ixor() and ipow()

# importing operator to handle operator operations
import operator

# using ixor() to exclusive or and assign value
x = operator.ixor(10,5);

# printing the modified value
print ("The value after xoring and assigning : ",end="")
print (x)

# using ipow() to exponentiate and assign value
x = operator.ipow(5,4);

# printing the modified value
print ("The value after exponentiating and assigning : ",end="")
print (x)
```

Output:

```
The value after xoring and assigning : 15
The value after exponentiating and assigning : 625
```

3. iand() :- This function is used to **assign and bitwise and the current value**. This operation does “**a &= b**” operation. Assigning is **not** performed in case of immutable containers, such as strings, numbers and tuples.

4. ior() :- This function is used to **assign and bitwise or the current value**. This operation does “**a |=b** ” operation. Assigning is **not** performed in case of immutable containers, such as strings, numbers and tuples.

```
# Python code to demonstrate the working of
# ior() and iand()

# importing operator to handle operator operations
import operator

# using ior() to or, and assign value
x = operator.ior(10,5);

# printing the modified value
print ("The value after bitwise or, and assigning : ",end="")
print (x)
```

```
# using iand() to and, and assign value
x = operator.iand(5,4);

# printing the modified value
print ("The value after bitwise and, and assigning : ",end="")
print (x)
```

Output:

```
The value after bitwise or, and assigning : 15
The value after bitwise and, and assigning : 4
```

5. ilshift() :- This function is used to **assign and bitwise leftshift the current value by second argument**. This operation does "**a <<=b** " operation. Assigning is **not** performed in case of immutable containers, such as strings, numbers and tuples.

6. irshift() :- This function is used to **assign and bitwise rightshift the current value by second argument**. This operation does "**a >>=b** " operation. Assigning is **not** performed in case of immutable containers, such as strings, numbers and tuples.

```
# Python code to demonstrate the working of
# ilshift() and irshift()

# importing operator to handle operator operations
import operator

# using ilshift() to bitwise left shift and assign value
x = operator.ilshift(8,2);

# printing the modified value
print ("The value after bitwise left shift and assigning : ",end="")
print (x)

# using irshift() to bitwise right shift and assign value
x = operator.irshift(8,2);

# printing the modified value
print ("The value after bitwise right shift and assigning : ",end="")
print (x)
```

Output:

```
The value after bitwise left shift and assigning : 32
The value after bitwise right shift and assigning : 2
```

Next Article – Inplace vs Standard Operators

This article is contributed by **Manjeet Singh**. 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
XOR

Calendar Functions in Python | Set 1(calendar(), month(), isleap()…)

Python defines an inbuilt module "**calendar**" which handles operations related to calendar.

Operations on calendar :

1. calendar(year, w, l, c) :- This function displays the year, width of characters, no. of lines per week and column separations.

2. firstweekday() :- This function returns the **first week day number**. By default 0 (Monday).

```
# Python code to demonstrate the working of
# calendar() and firstweeksday()

# importing calendar module for calendar operations
import calendar

# using calender to print calendar of year
# prints calendar of 2012
print ("The calender of year 2012 is : ")
print (calendar.calendar(2012,2,1,6))

#using firstweekday() to print starting day number
print ("The starting day number in calendar is : ",end="")
print (calendar.firstweekday())
```

Output:

```
The calendar of year 2012 is :
                    2012

     January              February              March
```

```
Mo Tu We Th Fr Sa Su      Mo Tu We Th Fr Sa Su      Mo Tu We Th Fr Sa Su
                   1             1  2  3  4  5                1  2  3  4
 2  3  4  5  6  7  8       6  7  8  9 10 11 12       5  6  7  8  9 10 11
 9 10 11 12 13 14 15      13 14 15 16 17 18 19      12 13 14 15 16 17 18
16 17 18 19 20 21 22      20 21 22 23 24 25 26      19 20 21 22 23 24 25
23 24 25 26 27 28 29      27 28 29                  26 27 28 29 30 31
30 31

       April                      May                       June
Mo Tu We Th Fr Sa Su      Mo Tu We Th Fr Sa Su      Mo Tu We Th Fr Sa Su
                   1          1  2  3  4  5  6                   1  2  3
 2  3  4  5  6  7  8       7  8  9 10 11 12 13       4  5  6  7  8  9 10
 9 10 11 12 13 14 15      14 15 16 17 18 19 20      11 12 13 14 15 16 17
16 17 18 19 20 21 22      21 22 23 24 25 26 27      18 19 20 21 22 23 24
23 24 25 26 27 28 29      28 29 30 31               25 26 27 28 29 30
30

        July                     August                  September
Mo Tu We Th Fr Sa Su      Mo Tu We Th Fr Sa Su      Mo Tu We Th Fr Sa Su
                   1             1  2  3  4  5                      1  2
 2  3  4  5  6  7  8       6  7  8  9 10 11 12       3  4  5  6  7  8  9
 9 10 11 12 13 14 15      13 14 15 16 17 18 19      10 11 12 13 14 15 16
16 17 18 19 20 21 22      20 21 22 23 24 25 26      17 18 19 20 21 22 23
23 24 25 26 27 28 29      27 28 29 30 31            24 25 26 27 28 29 30
30 31

      October                   November                  December
Mo Tu We Th Fr Sa Su      Mo Tu We Th Fr Sa Su      Mo Tu We Th Fr Sa Su
 1  2  3  4  5  6  7                1  2  3  4                      1  2
 8  9 10 11 12 13 14       5  6  7  8  9 10 11       3  4  5  6  7  8  9
15 16 17 18 19 20 21      12 13 14 15 16 17 18      10 11 12 13 14 15 16
22 23 24 25 26 27 28      19 20 21 22 23 24 25      17 18 19 20 21 22 23
29 30 31                  26 27 28 29 30            24 25 26 27 28 29 30
                                                    31

The starting day number in calendar is : 0
```

3. isleap (year) :- This function checks if year mentioned in argument is **leap or not**.

4. leapdays (year1, year2) :- This function returns the **number of leap days between the specified years** in arguments.

```
# Python code to demonstrate the working of
# isleap() and leapdays()

# importing calendar module for calendar operations
import calendar

# using isleap() to check if year is leap or not
if (calendar.isleap(2008)):
    print ("The year is leap")
else : print ("The year is not leap")

#using leapdays() to print leap days between years
print ("The leap days between 1950 and 2000 are : ",end="")
print (calendar.leapdays(1950, 2000))
```

Output:

```
The year is leap
The leap days between 1950 and 2000 are : 12
```

5. month (year, month, w, l) :- This function prints the **month of a specific year** mentioned in arguments. **It takes 4 arguments, year, month, width of characters and no. of lines taken by a week**.

```
# Python code to demonstrate the working of
# month()

# importing calendar module for calendar operations
import calendar

# using month() to display month of specific year
print ("The month 5th of 2016 is :")
print (calendar.month(2016,5,2,1))
```

Output:

```
The month 5th of 2016 is :
      May 2016
Mo Tu We Th Fr Sa Su
                   1
 2  3  4  5  6  7  8
 9 10 11 12 13 14 15
16 17 18 19 20 21 22
23 24 25 26 27 28 29
30 31
```

This article is contributed by **Manjeet Singh**. 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.

Calendar Functions in Python | Set 2(monthrange(), prcal(), weekday()…)

Some of calendar functions are discussed in the Set 1

1. monthrange(year, month) :- This function returns **two integers, first, the starting day number of week(0 as monday) , second, the number of days in the month**.

2. prcal(year, w, l, c) :- This function also **prints the calendar of specific year** but there is no need of "print" operation to execute this.

```
# Python code to demonstrate the working of
# monthrange() and prcal()

# importing calendar module for calendar operations
import calendar

# using monthrange() to print start week day and
# number of month
print ("The start week number and no. of days of month : ",end="")
print (calendar.monthrange(2008, 2))

# using prcal() to print calendar of 1997
print ("The calendar of 1997 is : ")
calendar.prcal(1997, 2,1,6)
```

Output:

```
The start week number and no. of days of month : (4, 29)
The calendar of 1997 is :
                              1997

      January                 February                March
Mo Tu We Th Fr Sa Su      Mo Tu We Th Fr Sa Su      Mo Tu We Th Fr Sa Su
       1  2  3  4  5                      1  2                      1  2
 6  7  8  9 10 11 12       3  4  5  6  7  8  9       3  4  5  6  7  8  9
13 14 15 16 17 18 19      10 11 12 13 14 15 16      10 11 12 13 14 15 16
20 21 22 23 24 25 26      17 18 19 20 21 22 23      17 18 19 20 21 22 23
27 28 29 30 31            24 25 26 27 28            24 25 26 27 28 29 30
                                                    31

       April                   May                     June
Mo Tu We Th Fr Sa Su      Mo Tu We Th Fr Sa Su      Mo Tu We Th Fr Sa Su
    1  2  3  4  5  6                1  2  3  4                         1
 7  8  9 10 11 12 13       5  6  7  8  9 10 11       2  3  4  5  6  7  8
14 15 16 17 18 19 20      12 13 14 15 16 17 18       9 10 11 12 13 14 15
21 22 23 24 25 26 27      19 20 21 22 23 24 25      16 17 18 19 20 21 22
28 29 30                  26 27 28 29 30 31         23 24 25 26 27 28 29
                                                    30

        July                   August                September
Mo Tu We Th Fr Sa Su      Mo Tu We Th Fr Sa Su      Mo Tu We Th Fr Sa Su
    1  2  3  4  5  6                   1  2  3       1  2  3  4  5  6  7
 7  8  9 10 11 12 13       4  5  6  7  8  9 10       8  9 10 11 12 13 14
14 15 16 17 18 19 20      11 12 13 14 15 16 17      15 16 17 18 19 20 21
21 22 23 24 25 26 27      18 19 20 21 22 23 24      22 23 24 25 26 27 28
28 29 30 31               25 26 27 28 29 30 31      29 30

      October                 November                December
Mo Tu We Th Fr Sa Su      Mo Tu We Th Fr Sa Su      Mo Tu We Th Fr Sa Su
       1  2  3  4  5                      1  2       1  2  3  4  5  6  7
 6  7  8  9 10 11 12       3  4  5  6  7  8  9       8  9 10 11 12 13 14
13 14 15 16 17 18 19      10 11 12 13 14 15 16      15 16 17 18 19 20 21
20 21 22 23 24 25 26      17 18 19 20 21 22 23      22 23 24 25 26 27 28
27 28 29 30 31            24 25 26 27 28 29 30      29 30 31
```

3. prmonth(year, month, w, l) :- This function also **prints the month of specific year** but there is no need of "print" operation to execute this.

4. setfirstweekday(num) :- This function sets the **day start number** of week.

```
# Python code to demonstrate the working of
# prmonth() and setfirstweekday()

# importing calendar module for calendar operations
import calendar

# using prmonth() to print calendar of 1997
print ("The 4th month of 1997 is : ")
calendar.prmonth(1997, 4, 2, 1)


# using setfirstweekday() to set first week day number
calendar.setfirstweekday(4)
```

```
print ("\r")

# using firstweekday() to check the changed day
print ("The new week day number is : ",end="")
print (calendar.firstweekday())
```

Output:

```
The 4th month of 1997 is :
    April 1997
Mo Tu We Th Fr Sa Su
    1  2  3  4  5  6
 7  8  9 10 11 12 13
14 15 16 17 18 19 20
21 22 23 24 25 26 27
28 29 30

The new week day number is : 4
```

5. weekday(year, month, date) :- This function returns the **week day number**(0 is Monday) of the date specified in its arguments.

```
# Python code to demonstrate the working of
# weekday()

# importing calendar module for calendar operations
import calendar

# using weekday() to print day number of date
print ("The day number of 25 April 1997 is : ",end="")
print (calendar.weekday(1997,4,25))
```

Output:

```
The day number of 25 April 1997 is : 4
```

This article is contributed by **Manjeet Singh**. 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

Complex Numbers in Python | Set 1 (Introduction)

Not only real numbers, Python can also handle complex numbers and its associated functions using the file "cmath". Complex numbers have their uses in many applications related to mathematics and python provides useful tools to handle and manipulate them.

Converting real numbers to complex number

An complex number is represented by " **x + yi** ". Python converts the real numbers x and y into complex using the function **complex(x,y)**. The real part can be accessed using the function **real()** and imaginary part can be represented by **imag()**.

```
# Python code to demonstrate the working of
# complex(), real() and imag()

# importing "cmath" for complex number operations
import cmath

# Initializing real numbers
x = 5
y = 3

# converting x and y into complex number
z = complex(x,y);

# printing real and imaginary part of complex number
print ("The real part of complex number is : ",end="")
print (z.real)

print ("The imaginary part of complex number is : ",end="")
print (z.imag)
```

Output:

```
The real part of complex number is : 5.0
The imaginary part of complex number is : 3.0
```

Phase of complex number

Geometrically, the phase of a complex number is the **angle between the positive real axis and the vector representing complex number**. This is also known as **argument** of complex number. Phase is returned using **phase()**, which takes complex number as argument. The range of phase lies from **-pi to +pi.** i.e from **-3.14 to +3.14**.

```
# Python code to demonstrate the working of
# phase()

# importing "cmath" for complex number operations
import cmath

# Initializing real numbers
x = -1.0
y = 0.0

# converting x and y into complex number
z = complex(x,y);

# printing phase of a complex number using phase()
print ("The phase of complex number is : ",end="")
print (cmath.phase(z))
```

Output:

```
The phase of complex number is : 3.141592653589793
```

Converting from polar to rectangular form and vice versa

Conversion to polar is done using **polar()**, which returns a **pair(r,ph)** denoting the **modulus r** and phase **angle ph**. modulus can be displayed using **abs()** and phase using **phase()**.
A complex number converts into rectangular coordinates by using **rect(r, ph)**, where **r is modulus** and **ph is phase angle**. It returns a value numerically equal to **r * (math.cos(ph) + math.sin(ph)*1j)**

```
# Python code to demonstrate the working of
# polar() and rect()

# importing "cmath" for complex number operations
import cmath
import math

# Initializing real numbers
x = 1.0
y = 1.0

# converting x and y into complex number
z = complex(x,y);

# converting complex number into polar using polar()
w = cmath.polar(z)

# printing modulus and argument of polar complex number
print ("The modulus and argument of polar complex number is : ",end="")
print (w)

# converting complex number into rectangular using rect()
w = cmath.rect(1.4142135623730951, 0.7853981633974483)

# printing rectangular form of complex number
print ("The rectangular form of complex number is : ",end="")
print (w)
```

Output:

```
The modulus and argument of polar complex number is : (1.4142135623730951, 0.7853981633974483)
The rectangular form of complex number is : (1.0000000000000002+1j)
```

Complex Numbers in Python | Set 2 (Important Functions and Constants)

This article is contributed by **Manjeet Singh**. 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.

Complex Numbers in Python | Set 2 (Important Functions and Constants)

Introduction to python complex numbers: Complex Numbers in Python | Set 1 (Introduction)
Some more important functions and constants are discussed in this article.
Operations on complex numbers :

1. exp() :- This function returns the **exponent** of the complex number mentioned in its argument.

2. log(x,b) :- This function returns the **logarithmic value of x with the base b**, both mentioned in its arguments. If base is not specified, natural log of x is returned.

```
# Python code to demonstrate the working of
# exp(), log()

# importing "cmath" for complex number operations
import cmath
import math

# Initializing real numbers
x = 1.0
y = 1.0

# converting x and y into complex number
z = complex(x, y);

# printing exponent of complex number
print ("The exponent of complex number is : ", end="")
print (cmath.exp(z))

# printing log form of complex number
print ("The log(base 10) of complex number is : ", end="")
print (cmath.log(z,10))
```

Output:

```
The exponent of complex number is : (1.4686939399158851+2.2873552871788423j)
The log(base 10) of complex number is : (0.15051499783199057+0.3410940884604603j)
```

3. log10() :- This function returns the **log base 10** of a complex number.

4. sqrt() :- This computes the **square root** of a complex number.

```
# Python code to demonstrate the working of
# log10(), sqrt()
# importing "cmath" for complex number operations
import cmath
import math

# Initializing real numbers
x = 1.0
y = 1.0

# converting x and y into complex number
z = complex(x, y);

# printing log10 of complex number
print ("The log10 of complex number is : ", end="")
print (cmath.log10(z))

# printing square root form of complex number
print ("The square root of complex number is : ", end="")
print (cmath.sqrt(z))
```

Output:

```
The log10 of complex number is : (0.15051499783199057+0.3410940884604603j)
The square root of complex number is : (1.09868411346781+0.45508986056222733j)
```

5. isfinite() :- Returns **true if both real and imaginary part** of complex number are **finite**, else returns false.

6. isinf() :- Returns **true if either real or imaginary part** of complex number is/are **infinite**, else returns false.

7. isnan() :- Returns true if **either real or imaginary part** of complex number is **NaN** , else returns false.

```
# Python code to demonstrate the working of
# isnan(), isinf(), isfinite()

# importing "cmath" for complex number operations
import cmath
import math

# Initializing real numbers
x = 1.0
y = 1.0
a = math.inf
b = math.nan

# converting x and y into complex number
z = complex(x,y);

# converting x and a into complex number
w = complex(x,a);

# converting x and b into complex number
v = complex(x,b);
```

```
# checking if both numbers are finite
if cmath.isfinite(z):
      print ("Complex number is finite")
else : print ("Complex number is infinite")

# checking if either number is/are infinite
if cmath.isinf(w):
      print ("Complex number is infinite")
else : print ("Complex number is finite")

# checking if either number is/are infinite
if cmath.isnan(v):
      print ("Complex number is NaN")
else : print ("Complex number is not NaN")
```

Output:

```
Complex number is finite
Complex number is infinite
Complex number is NaN
```

Constants

There are two constants defined in cmath module, **"pi"**, which returns the numerical value of pi. The second one is **"e"** which returns the numerical value of exponent.

```
# Python code to demonstrate the working of
# pi and e

# importing "cmath" for complex number operations
import cmath
import math

# printing the value of pi
print ("The value of pi is : ", end="")
print (cmath.pi)

# printing the value of e
print ("The value of exponent is : ", end="")
print (cmath.e)
```

Output:

```
The value of pi is : 3.141592653589793
The value of exponent is : 2.718281828459045
```

Complex Numbers in Python | Set 3 (Trigonometric and Hyperbolic Functions)

This article is contributed by **Manjeet Singh**. 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

Complex Numbers in Python | Set 3 (Trigonometric and Hyperbolic Functions)

Some of the Important Complex number functions are discussed in the articles below

Complex Numbers in Python | Set 1 (Introduction)
Complex Numbers in Python | Set 2 (Important Functions and constants)

Trigonometric and Hyperbolic Functions are discussed in this article.

Trigonometric Functions

1. sin() : This function returns the **sine** of the complex number passed in argument.

2. cos() : This function returns the **cosine** of the complex number passed in argument.

3. tan() : This function returns the **tangent** of the complex number passed in argument.

```
# Python code to demonstrate the working of
# sin(), cos(), tan()

# importing "cmath" for complex number operations
import cmath

# Initializing real numbers
x = 1.0

y = 1.0

# converting x and y into complex number z
```

```
z = complex(x,y);

# printing sine of the complex number
print ("The sine value of complex number is : ",end="")
print (cmath.sin(z))

# printing cosine of the complex number
print ("The cosine value of complex number is : ",end="")
print (cmath.cos(z))

# printing tangent of the complex number
print ("The tangent value of complex number is : ",end="")
print (cmath.tan(z))
```

Output:

```
The sine value of complex number is : (1.2984575814159773+0.6349639147847361j)
The cosine value of complex number is : (0.8337300251311491-0.9888977057628651j)
The tangent value of complex number is : (0.2717525853195118+1.0839233273386946j)
```

4. asin() : This function returns the **arc sine** of the complex number passed in argument.

5. acos() : This function returns the **arc cosine** of the complex number passed in argument.

6. atan() : This function returns the **arc tangent** of the complex number passed in argument.

```
# Python code to demonstrate the working of
# asin(), acos(), atan()

# importing "cmath" for complex number operations
import cmath

# Initializing real numbers
x = 1.0

y = 1.0

# converting x and y into complex number z
z = complex(x,y);

# printing arc sine of the complex number
print ("The arc sine value of complex number is : ",end="")
print (cmath.asin(z))

# printing arc cosine of the complex number
print ("The arc cosine value of complex number is : ",end="")
print (cmath.acos(z))

# printing arc tangent of the complex number
print ("The arc tangent value of complex number is : ",end="")
print (cmath.atan(z))
```

Output:

```
The arc sine value of complex number is : (0.6662394324925153+1.0612750619050357j)
The arc cosine value of complex number is : (0.9045568943023814-1.0612750619050357j)
The arc tangent value of complex number is : (1.0172219678978514+0.40235947810852507j)
```

Hyperbolic Functions

1. sinh() : This function returns the **hyperbolic sine** of the complex number passed in argument.

2. cosh() : This function returns the **hyperbolic cosine** of the complex number passed in argument.

3. tanh() : This function returns the **hyperbolic tangent** of the complex number passed in argument.

```
# Python code to demonstrate the working of
# sinh(), cosh(), tanh()

# importing "cmath" for complex number operations
import cmath

# Initializing real numbers
x = 1.0

y = 1.0

# converting x and y into complex number z
z = complex(x,y);

# printing hyperbolic sine of the complex number
print ("The hyperbolic sine value of complex number is : ",end="")
print (cmath.sinh(z))

# printing hyperbolic cosine of the complex number
print ("The hyperbolic cosine value of complex number is : ",end="")
print (cmath.cosh(z))

# printing hyperbolic tangent of the complex number
print ("The hyperbolic tangent value of complex number is : ",end="")
print (cmath.tanh(z))
```

Output:

```
The hyperbolic sine value of complex number is : (0.6349639147847361+1.2984575814159773j)
The hyperbolic cosine value of complex number is : (0.8337300251311491+0.9888977057628651j)
The hyperbolic tangent value of complex number is : (1.0839233273386946+0.2717525853195117j)
```

4. asinh() : This function returns the **inverse hyperbolic sine** of the complex number passed in argument.

5. acosh() : This function returns the **inverse hyperbolic cosine** of the complex number passed in argument.

6. atanh() : This function returns the **inverse hyperbolic tangent** of the complex number passed in argument.

```
# Python code to demonstrate the working of
# asinh(), acosh(), atanh()

# importing "cmath" for complex number operations
import cmath

# Initializing real numbers
x = 1.0

y = 1.0

# converting x and y into complex number z
z = complex(x,y);

# printing inverse hyperbolic sine of the complex number
print ("The inverse hyperbolic sine value of complex number is : ",end="")
print (cmath.asinh(z))

# printing inverse hyperbolic cosine of the complex number
print ("The inverse hyperbolic cosine value of complex number is : ",end="")
print (cmath.acosh(z))

# printing inverse hyperbolic tangent of the complex number
print ("The inverse hyperbolic tangent value of complex number is : ",end="")
print (cmath.atanh(z))
```

Output:

```
The inverse hyperbolic sine value of complex number is : (1.0612750619050357+0.6662394324925153j)
The inverse hyperbolic cosine value of complex number is : (1.0612750619050357+0.9045568943023813j)
The inverse hyperbolic tangent value of complex number is : (0.40235947810852507+1.0172219678978514j)
```

This article is contributed by **Manjeet Singh**. 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

Time Functions in Python | Set 1 (time(), ctime(), sleep()…)

Python has defined a module, “time” which allows us to handle various operations regarding time, its conversions and representations, which find its use in various applications in life. The beginning of time is started measuring from **1 January, 12:00 am, 1970** and this very time is termed as “**epoch**” in Python.

Operations on Time :

1. time() :- This function is used to count the number of **seconds elapsed since the epoch**.

2. gmtime(sec) :- This function returns a **structure with 9 values** each representing a time attribute in sequence. It converts **seconds into time attributes(days, years, months etc.)** till specified seconds from epoch. If no seconds are mentioned, time is calculated till present. The structure attribute table is given below.

Screenshot (24)

```
# Python code to demonstrate the working of
# time() and gmtime()

#  importing "time" module for time operations
```

```
import time

# using time() to display time since epoch
print ("Seconds elapsed since the epoch are : ",end="")
print (time.time())


# using gmtime() to return the time attribute structure
print ("Time calculated acc. to given seconds is : ")
print (time.gmtime())
```

Output:

```
Seconds elapsed since the epoch are : 1470121951.9536893
Time calculated acc. to given seconds is :
time.struct_time(tm_year=2016, tm_mon=8, tm_mday=2,
tm_hour=7, tm_min=12, tm_sec=31, tm_wday=1,
tm_yday=215, tm_isdst=0)
```

3. asctime("time") :- This function takes a time attributed string produced by gmtime() and returns a **24 character string denoting time**.

4. ctime(sec) :- This function returns a **24 character time string** but takes seconds as argument and **computes time till mentioned seconds**. If no argument is passed, time is calculated till present.

```
# Python code to demonstrate the working of
# asctime() and ctime()

#  importing "time" module for time operations
import time

# initializing time using gmtime()
ti = time.gmtime()

# using asctime() to display time acc. to time mentioned
print ("Time calculated using asctime() is : ",end="")
print (time.asctime(ti))


# using ctime() to diplay time string using seconds
print ("Time calculated using ctime() is : ", end="")
print (time.ctime())
```

Output:

```
Time calculated using asctime() is : Tue Aug  2 07:47:02 2016
Time calculated using ctime() is : Tue Aug  2 07:47:02 2016
```

5. sleep(sec) :- This method is used to **hault the program execution** for the time specified in the arguments.

```
# Python code to demonstrate the working of
# sleep()

#  importing "time" module for time operations
import time

# using ctime() to show present time
print ("Start Execution : ",end="")
print (time.ctime())

# using sleep() to hault execution
time.sleep(4)

# using ctime() to show present time
print ("Stop Execution : ",end="")
print (time.ctime())
```

Output:

```
Start Execution : Tue Aug  2 07:59:03 2016
Stop Execution : Tue Aug  2 07:59:07 2016
```

Reference :
http://www.tutorialspoint.com/python/python_date_time.htm

This article is contributed by **Manjeet Singh**. 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.

Time Functions in Python | Set-2 (Date Manipulations)

Some of Time Functions are discussed in Set 1

Date manipulation can also be performed using Python using “datetime” module and using “date” class in it.

Operations on Date :

1. MINYEAR :- It displays the **minimum year** that can be represented using date class.

2. MAXYEAR :- It displays the **maximum year** that can be represented using date class.

```
# Python code to demonstrate the working of
# MINYEAR and MAXYEAR

# importing built in module datetime
import datetime
from datetime import date

# using MINYEAR to print minimum representable year
print ("Minimum representable year is : ",end="")
print (datetime.MINYEAR)

# using MAXYEAR to print maximum representable year
print ("Maximum representable year is : ",end="")
print (datetime.MAXYEAR)
```

Output:

```
Minimum representable year is : 1
Maximum representable year is : 9999
```

3. date(yyyy-mm-dd) :- This function returns a string with passed arguments in order of year, months and date.

4. today() :- Returns the **date of present day** in the format yyyy-mm-dd.

```
# Python code to demonstrate the working of
# date() and today()

# importing built in module datetime
import datetime
from datetime import date

# using date() to represent date
print ("The represented date is : ",end="")
print (datetime.date(1997,4,1))

# using today() to print present date
print ("Present date is : ",end="")
print (date.today())
```

Output:

```
The represented date is : 1997-04-01
Present date is : 2016-08-02
```

5. fromtimestamp(sec) :- It returns the **date calculated from the seconds** elapsed since epoch mentioned in arguments.

6. min() :- This returns the **minimum date** that can be represented by date class.

7. max() :- This returns the **maximum date** that can be represented by date class.

```
# Python code to demonstrate the working of
# fromtimestamp(), min() and max()

# importing built in module datetime
import datetime
from datetime import date

# using fromtimestamp() to calculate date
print ("The calculated date from seconds is : ",end="")
print (date.fromtimestamp(3452435))

# using min() to print minimum representable date
print ("Minimum representable date is : ",end="")
print (date.min)

# using max() to print minimum representable date
print ("Maximum representable date is : ",end="")
print (date.max)
```

Output:

```
The calculated date from seconds is : 1970-02-09
Minimum representable date is : 0001-01-01
Maximum representable date is : 9999-12-31
```

This article is contributed by **Manjeet Singh**. 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

Random Numbers in Python

Python defines a set of functions that are used to generate or manipulate random numbers. This particular type of functions are used in a lot of games, lotteries or any application requiring random number generation.

Randon Number Operations :

1. choice() :- This function is used to **generate 1 random number** from a container.

2. randrange(beg, end, step) :- This function is also used to **generate random number but within a range** specified in its arguments. **This function takes 3 arguments, beginning number (included in generation), last number (excluded in generation) and step (to skip numbers in range while selecting).**

```
# Python code to demonstrate the working of
# choice() and randrange()

# importing "random" for random operations
import random

# using choice() to generate a random number from a
# given list of numbers.
print ("A random number from list is : ",end="")
print (random.choice([1, 4, 8, 10, 3]))

# using randrange() to generate in range from 20
# to 50. The last parameter 3 is step size to skip
# three numbers when selecting.
print ("A random number from range is : ",end="")
print (random.randrange(20, 50, 3))
```

Output:

```
A random number from list is : 4
A random number from range is : 41
```

3. random() :- This number is used to generate a **float random number less than 1 and greater or equal to 0.**

4. seed() :- This function **maps a particular random number with the seed argument** mentioned. All random numbers called after the seeded value returns the mapped number.

```
# Python code to demonstrate the working of
# random() and seed()

# importing "random" for random operations
import random

# using random() to generate a random number
# between 0 and 1
print ("A random number between 0 and 1 is : ", end="")
print (random.random())

# using seed() to seed a random number
random.seed(5)

# printing mapped random number
print ("The mapped random number with 5 is : ", end="")
print (random.random())

# using seed() to seed different random number
random.seed(7)

# printing mapped random number
print ("The mapped random number with 7 is : ", end="")
print (random.random())

# using seed() to seed to 5 again
random.seed(5)

# printing mapped random number
print ("The mapped random number with 5 is : ",end="")
print (random.random())

# using seed() to seed to 7 again
random.seed(7)

# printing mapped random number
print ("The mapped random number with 7 is : ",end="")
print (random.random())
```

Output:

```
A random number between 0 and 1 is : 0.510721762520941
The mapped random number with 5 is : 0.6229016948897019
The mapped random number with 7 is : 0.32383276483316237
The mapped random number with 5 is : 0.6229016948897019
The mapped random number with 7 is : 0.32383276483316237
```

5. shuffle() :- This function is used to **shuffle the entire list** to randomly arrange them.

6. uniform(a, b) :- This function is used to generate a **floating point random number between the numbers** mentioned in its arguments. **It takes two arguments, lower limit(included in generation) and upper limit(not included in generation).**

```
# Python code to demonstrate the working of
# shuffle() and uniform()

# importing "random" for random operations
import random

# Initializing list
li = [1, 4, 5, 10, 2]

# Printing list before shuffling
print ("The list before shuffling is : ", end="")
for i in range(0, len(li)):
    print (li[i], end=" ")
print("\r")

# using shuffle() to shuffle the list
random.shuffle(li)

# Printing list after shuffling
print ("The list after shuffling is : ", end="")
for i in range(0, len(li)):
    print (li[i], end=" ")
print("\r")

# using uniform() to generate random floating number in range
# prints number between 5 and 10
print ("The random floating point number between 5 and 10 is : ",end="")
print (random.uniform(5,10))
```

Output:

```
The list before shuffling is : 1 4 5 10 2
The list after shuffling is : 2 1 4 5 10
The random floating point number between 5 and 10 is : 5.183697823553464
```

This article is contributed by **Manjeet Singh**. 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

reduce() in Python

The **reduce(fun,seq)** function is used to **apply a particular function passed in its argument to all of the list elements** mentioned in the sequence passed along.This function is defined in “**functools**” module.

Working :

- At first step, first two elements of sequence are picked and the result is obtained.
- Next step is to apply the same function to the previously attained result and the number just succeeding the second element and the result is again stored.
- This process continues till no more elements are left in the container.
- The final returned result is returned and printed on console.

```
# python code to demonstrate working of reduce()

# importing functools for reduce()
import functools

# initializing list
lis = [ 1 , 3, 5, 6, 2, ]

# using reduce to compute sum of list
print ("The sum of the list elements is : ",end="")
print (functools.reduce(lambda a,b : a+b,lis))

# using reduce to compute maximum element from list
print ("The maximum element of the list is : ",end="")
print (functools.reduce(lambda a,b : a if a > b else b,lis))
```

Output:

```
The sum of the list elements is : 17
The maximum element of the list is : 6
```

Using Operator Functions

reduce() can also be combined with operator functions to achieve the similar functionality as with lambda functions and makes the code more readable.

```
# python code to demonstrate working of reduce()
# using operator functions

# importing functools for reduce()
import functools

# importing operator for operator functions
import operator

# initializing list
lis = [ 1 , 3, 5, 6, 2, ]

# using reduce to compute sum of list
# using operator functions
print ("The sum of the list elements is : ",end="")
print (functools.reduce(operator.add,lis))

# using reduce to compute product
# using operator functions
print ("The product of list elements is : ",end="")
print (functools.reduce(operator.mul,lis))

# using reduce to concatenate string
print ("The concatenated product is : ",end="")
print (functools.reduce(operator.add,["geeks","for","geeks"]))
```

Output

```
The sum of the list elements is : 17
The product of list elements is : 180
The concatenated product is : geeksforgeeks
```

reduce() vs accumulate()

Both reduce() and accumulate() can be used to calculate the summation of a sequence elements. But there are differences in the implementation aspects in both of these.

- reduce() is defined in “functools” module, accumulate() in “itertools” module.
- reduce() stores the intermediate result and only returns the final summation value. Whereas, accumulate() returns a list containing the intermediate results. The last number of the list returned is summation value of the list.
- reduce(fun,seq) takes function as 1st and sequence as 2nd argument. In contrast accumulate(seq,fun) takes sequence as 1st argument and function as 2nd argument.

```
# python code to demonstrate summation
# using reduce() and accumulate()

# importing itertools for accumulate()
import itertools

# importing functools for reduce()
import functools

# initializing list
lis = [ 1, 3, 4, 10, 4 ]

# priting summation using accumulate()
print ("The summation of list using accumulate is :",end="")
print (list(itertools.accumulate(lis,lambda x,y : x+y)))

# priting summation using reduce()
print ("The summation of list using reduce is :",end="")
print (functools.reduce(lambda x,y:x+y,lis))
```

Output:

```
The summation of list using accumulate is :[1, 4, 8, 18, 22]
The summation of list using reduce is :22
```

This article is contributed by **Manjeet Singh(S.Nandini)**. 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

struct module in Python

This module performs conversions between Python values and C structs represented as Python bytes objects. Format strings are the mechanism used to specify the expected layout when packing and unpacking data. Module struct is available in Python 3.x and not on 2.x, thus these codes will run on Python3 interpreter.

Struct Functions

- **struct.pack()**

  Syntax:
  struct.pack(format, v1, v2, ...)

  Return a string containing the values v1, v2, … , that are packed according to the given format (Format strings are the mechanism used to specify the expected layout when packing and unpacking data).The values followed by the format must be as per the format only, else struct.error is raised.

```
import struct

# Format: h is short in C type
# Format: l is long in C type
# Format 'hhl' stands for 'short short long'
var = struct.pack('hhl',1,2,3)
print(var)

# Format: i is int in C type
# Format 'iii' stands for 'int int int'
var = struct.pack('iii',1,2,3)
print(var)
```

  Output:

```
b'\x01\x00\x02\x00\x00\x00\x00\x00\x03\x00\x00\x00\x00\x00\x00\x00'
b'\x01\x00\x00\x00\x02\x00\x00\x00\x03\x00\x00\x00'
```

- **struct.unpack()**

  Syntax:
  struct.unpack(fmt, string)

  Return the values v1, v2, … , that are unpacked according to the given format(1st argument). Values returned by this function are returned as tuples of size that is equal to the number of values passed through struct.pack() during packing.

```
import struct

# '?' -> _BOOL , 'h' -> short, 'i' -> int and 'l' -> long
var = struct.pack('?hil', True, 2, 5, 445)
print(var)

# struct.unpack() return a tuples
# Variables V1, V2, V3,.. are returned as elements of tuple
tup = struct.unpack('?hil', var)
print(tup)

# q -> long long int and f -> float
var = struct.pack('qf', 5, 2.3)
print(var)
tup = struct.unpack('qf', var)
print(tup)
```

  Output:

```
b'\x01\x00\x02\x00\x05\x00\x00\x00\xbd\x01\x00\x00\x00\x00\x00\x00'
(True, 2, 5, 445)
b'\x05\x00\x00\x00\x00\x00\x00\x0033\x13@'
(5, 2.299999952316284)
```

  Note: 'b' in the Output stands for binary.

- **struct.calcsize()**

  Syntax:
  struct.calcsize(fmt)
  fmt: format

  Return the size of the struct (and hence of the string) corresponding to the given format. calcsize() is important function, and is required for function such as struct.pack_into() and struct.unpack_from(), which require offset value and buffer as well.

```
import struct
var = struct.pack('?hil', True, 2, 5, 445)
print(var)
# Returns the size of the structure
print(struct.calcsize('?hil'))
print(struct.calcsize('qf'))
```

  Output:

```
b'\x01\x00\x02\x00\x05\x00\x00\x00\xbd\x01\x00\x00\x00\x00\x00\x00'
16
12
```

```
import struct
var = struct.pack('bi', 56, 0x12131415)
print(var)
print(struct.calcsize('bi'))
var = struct.pack('ib', 0x12131415, 56)
print(var)
print(struct.calcsize('ib'))
```

Output:

```
b'8\x00\x00\x00\x15\x14\x13\x12'
8
b'\x15\x14\x13\x128'
5
```

Note: The ordering of format characters may have an impact on size.

- **Exception struct.error**
  Exception struct.error describes what is wrong at passing arguments, when a wrong argument is passed struct.error is raised.

```
from struct import error
print(error)
```

Note: This is piece of code is not useful, anywhere other than exception handling, and is used to show that 'error' upon interpreted shows about the class.

- **struct.pack_into()**

```
Syntax:
struct.pack_into(fmt, buffer, offset, v1, v2, ...)
fmt: data type format
buffer: writable buffer which starts at offset (optional)
v1,v2.. : values
```

- **struct.unpack_from()**

```
Syntax:
struct.unpack_from(fmt, buffer[,offset = 0])fmt: data type format
buffer: writable buffer which starts at offset (optional)
```

Returns a tuple, similar to struct.unpack()

```
import struct

# ctypes in imported to create string buffer
import ctypes

# SIZE of the format is calculated using calcsize()
siz = struct.calcsize('hhl')
print(siz)

# Buffer 'buff' is created
buff = ctypes.create_string_buffer(siz)

# struct.pack() returns packed data
# struct.unpack() returns unpacked data
x = struct.pack('hhl', 2, 2, 3)
print(x)
print(struct.unpack('hhl', x))

# struct.pack_into() packs data into buff, doesn't return any value
# struct.unpack_from() unpacks data from buff, returns a tuple of values
struct.pack_into('hhl', buff, 0, 2, 2, 3)
print(struct.unpack_from('hhl', buff, 0))
```

Output:

```
16
b'\x02\x00\x02\x00\x00\x00\x00\x00\x03\x00\x00\x00\x00\x00\x00\x00'
(2, 2, 3)
(2, 2, 3)
```

Reference https://docs.python.org/2/library/struct.html

This article is contributed by **Piyush Doorwar**. 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.

Mouse and keyboard automation using Python

This article illustrates how to automate movements of mouse and keyboard using **pyautogui** module in python. This module is not preloaded with python. So to install it run the following command:

```
pip install pyautogui
```

Controlling mouse movements using pyautogui module

Python tracks and controls mouse using coordinate system of screen. Suppose the resolution of your screen is 1920X1080 , then your screen's coordinate system looks like :
gui in python

- **size():** This function is used to get Screen resolution.

  ```
  import pyautogui
  print(pyautogui.size())
  ```

  Save this file with .py extension, and then run the file.
  This python code use size() function to output your screen resolution in x,y format:
  Output:

  ```
   (1920, 1080)
  ```

  Note: Some of the codes provided in this article might not run on geeksforgeeks IDE, since geeksforgeeks IDE doesnot have required modules to run these codes. But these code can be easily run locally on your PC by installing python and following the instructions provided in the article.

- **moveTo():** use this function move the mouse in pyautogui module.

  ```
  import pyautogui
  pyautogui.moveTo(100, 100, duration= 1)
  ```

  This code uses moveTo() function, which takes x and y coordinates, and an optional duration argument. This function moves your mouse pointer from it's current location to x,y coordinate, and takes time as specified by duration argument to do so. Save and run this python script to see your mouse pointer magically moving from its current location to coordinates (100, 100) , taking 1 second in this process.

- **moveRel() function:** moves the mouse pointer relative to its previous position.

  ```
  import pyautogui
  pyautogui.moveRel(0, 50, duration=1)
  ```

  This code will move mouse pointer at (0, 50) relative to its original position. For example, if mouse position before running the code was (1000, 1000), then this code will move the pointer to coordinates (1000, 1050) in duration 1 second.

- **position():** function to get current position of the mouse pointer.

  ```
  import pyautogui
  print(pyautogui.position())
  ```

  Output: coordinates where your mouse was residing at the time of executing the program.

- **click():** Function used for clicking and dragging the mouse.

  ```
  import pyautogui
  pyautogui.click(100, 100)
  ```

  This code performs a typical mouse click at the location (100, 100).
  We have two functions associated with drag operation of mouse, **dragTo and dragRel**. They perform similar to moveTo and moveRel functions, except they hold the left mouse button while moving, thus initiating a drag.
  This functionality can be used at various places, like moving a dialog box, or drawing something automatically using pencil tool in MS Paint. To draw a square in paint:

  ```
  import time
  import time

  #a module which has functions related to time.
  # It can be installed using cmd command:
  # pip install time, in the same way as pyautogui.
  import pyautogui
  time.sleep(10)

  #makes program execution pause for 10 sec
  ```

```
pyautogui.moveTo(1000, 1000, duration=1)

#moves mouse to 1000, 1000.
pyautogui.dragRel(100, 0, duration=1)

#drags mouse 100, 0 relative to its previous position,
# thus dragging it to 1100, 1000
pyautogui.dragRel(0, 100, duration=1)
pyautogui.dragRel(-100, 0, duration=1)
pyautogui.dragRel(0, -100, duration=1)
```

Before running the code, open MS paint in background with pencil tool selected. Now run the code, quickly switch to MS paint before 10 seconds (since we have given 10 second pause time using sleep() function before running the program).
After 10 seconds, you will see a square being drawn in MS paint, with its top left edge at 1000,1000 and edge length 100 pixels.

- **scroll():** scroll function takes no. of pixels as argument, and scrolls the screen up to given number of pixels.

```
import pyautogui
pyautogui.scroll(200)
```

This code scrolls the active screen up to 200 pixels.

- **typewrite():** You can automate typing of string by using typewrite() function. just pass the string which you want to type as argument of this function.

```
import pyautogui
pyautogui.click(100, 100)
pyautogui.typewrite("hello Geeks!")
```

Suppose a text field was present at coordinates 100, 100 on screen, then this code will click the text field to make it active and type "hello Geeks!" in it.

- **Passing key names:** You can pass key names separately through typewrite() function.

```
import pyautogui
pyautogui.typewrite(["a", "left", "ctrlleft"])
```

This code is automatic equivalent of typing "a", pressing left arrow key, and pressing left control key.

- **Pressing hotkey combinations:** Use hotkey() function to press combination of keys like ctrl-c, ctrl-a etc. Ex:

```
import pyautogui
pyautogui.hotkey("ctrlleft", "a")
```

This code is automatic equivalent of pressing left ctrl and "a"simultaneously. Thus in windows, this will result in selection of all text present on screen.

This article is contributed by **tkkhhaarree**. 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.

GATE CS Corner   Company Wise Coding Practice

Python

Timeit in Python with Examples

This article will introduce you to a method of measuring the execution time of your python code snippets.
We will be using an in-built python library timeit.

This module provides a simple way to find the execution time of small bits of Python code.

Why timeit?

- Well, how about using simple time module? Just save the time before and after the execution of code and subtract them! But this method is not precise as there might be a background process momentarily running which disrupts the code execution and you will get significant variations in running time of small code snippets.
- timeit runs your snippet of code millions of time (default value is 1000000) so that you get the statistically most relevant measurement of code execution time!
- timeit is pretty simple to use and has a command line interface as well as a callable one.

So now, let's start exploring this handy library!

The module function **timeit.timeit(stmt, setup, timer, number)** accepts four arguments:

- **stmt** which is the statement you want to measure; it defaults to 'pass'.
- **setup** which is the code that you run before running the **stmt**; it defaults to 'pass'.
  We generally use this to import the required modules for our code.
- **timer** which is a **timeit.Timer** object; it usually has a sensible default value so you don't have to worry about it.
- **number** which is the number of executions you'd like to run the **stmt**.

Where the **timeit.timeit()** function returns the number of seconds it took to execute the code.

Example 1

Let us see a basic example first.

```
# importing the required module
import timeit

# code snippet to be executed only once
```

```
mysetup = "from math import sqrt"

# code snippet whose execution time is to be measured
mycode = '''
def example():
    mylist = []
    for x in range(100):
        mylist.append(sqrt(x))
'''

# timeit statement
print timeit.timeit(setup = mysetup,
                    stmt = mycode,
                    number = 10000)
```

- The output of above program will be the execution time(in seconds) for 10000 iterations of the code snippet passed to **timeit.timeit()** function.
  Note: *Pay attention to the fact that the output is the execution time of **number** times iteration of the code snippet, not the single iteration. For a single iteration exec. time, divide the output time by **number**.*
- The program is pretty straight-forward. All we need to do is to pass the code as a string to the **timeit.timeit()** function.
- It is advisable to keep the import statements and other static pieces of code in setup argument.

Example 2

Let's see another practical example in which we will compare two searching techniques, namely, **Binary search** and **Linear search**.
Also, here I demonstrate two more features, **timeit.repeat** function and calling the functions already defined in our program.

```
# importing the required modules
import timeit

# binary search function
def binary_search(mylist, find):
    while len(mylist) > 0:
        mid = (len(mylist))//2
        if mylist[mid] == find:
            return True
        elif mylist[mid] < find:
            mylist = mylist[:mid]
        else:
            mylist = mylist[mid + 1:]
    return False


# linear search function
def linear_search(mylist, find):
    for x in mylist:
        if x == find:
            return True
    return False


# compute binary search time
def binary_time():
    SETUP_CODE = '''
from __main__ import binary_search
from random import randint'''

    TEST_CODE = '''
mylist = [x for x in range(10000)]
find = randint(0, len(mylist))
binary_search(mylist, find)'''

    # timeit.repeat statement
    times = timeit.repeat(setup = SETUP_CODE,
                          stmt = TEST_CODE,
                          repeat = 3,
                          number = 10000)

    # priniting minimum exec. time
    print('Binary search time: {}'.format(min(times)))


# compute linear search time
def linear_time():
    SETUP_CODE = '''
from __main__ import linear_search
from random import randint'''

    TEST_CODE = '''
mylist = [x for x in range(10000)]
find = randint(0, len(mylist))
linear_search(mylist, find)
    '''
    # timeit.repeat statement
    times = timeit.repeat(setup = SETUP_CODE,
                          stmt = TEST_CODE,
                          repeat = 3,
                          number = 10000)

    # priniting minimum exec. time
    print('Linear search time: {}'.format(min(times)))

if __name__ == "__main__":
    linear_time()
    binary_time()
```

- The output of above program will be the minimum value in the list **times**.
  This is how a sample output looks like:
- **timeit.repeat()** function accepts one extra argument, **repeat**. The output will be a list of the execution times of all code runs repeated a specified no. of times.
- In setup argument, we passed:

  ```
  from __main__ import binary_search
  from random import randint
  ```

  This will import the definition of function **binary_search**, already defined in the program and **random** library function **randint**.
- As expected, we notice that execution time of binary search is significantly lower than linear search!

Example 3
Finally, I demonstrate below how you can utilize the command line interface of **timeit** module:

3

Here I explain each term individually:

2

So, this was a brief yet concise introduction to **timeit** module and its practical applications.
Its a pretty handy tool for python programmers when they need a quick glance of the execution time of their code snippets.

This article is contributed by **Nikhil Kumar**. 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.

Python

NumPy in Python | Set 1 (Introduction)

This article will help you get acquainted with the widely used array-processing library in Python, NumPy.

numpy-logo

What is NumPy?

NumPy is a general-purpose array-processing package. It provides a high-performance multidimensional array object, and tools for working with these arrays.

It is the fundamental package for scientific computing with Python. It contains various features including these important ones:

- A powerful N-dimensional array object
- Sophisticated (broadcasting) functions
- Tools for integrating C/C++ and Fortran code
- Useful linear algebra, Fourier transform, and random number capabilities

Besides its obvious scientific uses, NumPy can also be used as an efficient multi-dimensional container of generic data.
Arbitrary data-types can be defined using Numpy which allows NumPy to seamlessly and speedily integrate with a wide variety of databases.

Installation:

- **Mac** and **Linux** users can install NumPy via pip command:

  ```
  pip install numpy
  ```

- **Windows** does not have any package manager analogous to that in linux or mac.
  Please download the pre-built windows installer for NumPy from here (according to your system configuration and Python version).
  And then install the packages manually.

Note: All the examples discussed below will not run on an **online IDE.**

1. Arrays in NumPy: NumPy's main object is the homogeneous multidimensional array.

- It is a table of elements (usually numbers), all of the same type, indexed by a tuple of positive integers.
- In NumPy dimensions are called *axes*. The number of axes is *rank*.
- NumPy's array class is called **ndarray**. It is also known by the alias **array**.

Example:

```
[[ 1, 2, 3],
 [ 4, 2, 5]]
Here,
rank = 2 (as it is 2-dimensional or it has 2 axes)
first dimension(axis) length = 2, second dimension has length = 3
overall shape can be expressed as: (2, 3)
```

```
# Python program to demonstrate basic array characteristics
import numpy as np

# creating array object
arr = np.array([[ 1, 2, 3],
                [ 4, 2, 5]])

# printing type of arr object
print "Array is of type: ", type(arr)

# printing array dimensions (axes)
print "No. of dimensions: ", arr.ndim

# printing shape of array
print "Shape of array: ", arr.shape

# printing size (total number of elements) of array
print "Size of array: ", arr.size

# printing type of elements in array
print "Array stores elements of type: ", arr.dtype
```

Output:

```
Array is of type:
No. of dimensions:  2
Shape of array:  (2L, 3L)
Size of array:  6
Array stores elements of type:  int32
```

2. Array creation: There are various ways to create arrays in NumPy.

- For example, you can create an array from a regular Python **list** or **tuple** using the **array** function. The type of the resulting array is deduced from the type of the elements in the sequences.
- Often, the elements of an array are originally unknown, but its size is known. Hence, NumPy offers several functions to create arrays with **initial placeholder content**. These minimize the necessity of growing arrays, an expensive operation.
  For example: np.zeros, np.ones, np.full, np.empty, etc.
- To create sequences of numbers, NumPy provides a function analogous to range that returns arrays instead of lists.
- **arange:** returns evenly spaced values within a given interval. **step** size is specified.
- **linspace:** returns evenly spaced values within a given interval. **num** no. of elements are returned.
- **Reshaping array:** We can use **reshape** method to reshape an array. Consider an array with shape (a1, a2, a3, …, aN). We can reshape and convert it into another array with shape (b1, b2, b3, …, bM). The only required condition is:
  a1 x a2 x a3 … x aN = b1 x b2 x b3 … x bM . (i.e original size of array remains unchanged.)
- **Flatten array:** We can use **flatten** method to get a copy of array collapsed into **one dimension**. It accepts *order* argument. Default value is 'C' (for row-major order). Use 'F' for column major order.

Note: Type of array can be explicitly defined while creating array.

```
# Python program to demonstrate array creation techniques
import numpy as np

# creating array from list with type float
a = np.array([[1, 2, 4], [5, 8, 7]], dtype = 'float')
print "Array created using passed list:\n", a

# creating array from tuple
b = np.array((1 , 3, 2))
print "\nArray created using passed tuple:\n", b

# creating a 3X4 array with all zeros
c = np.zeros((3, 4))
print "\nAn array initialized with all zeros:\n", c
```

```
# create a constant value array of complex type
d = np.full((3, 3), 6, dtype = 'complex')
print "\nAn array initialized with all 6s. Array type is complex:\n", d

# create an array with random values
e = np.random.random((2,2))
print "\nA random array:\n", e

# create a sequence of integers from 0 to 30 with steps of 5
f = np.arange(0, 30, 5)
print "\nA sequential array with steps of 5:\n", f

# create a sequence of 10 values in range 0 to 5
g = np.linspace(0, 5, 10)
print "\nA sequential array with 10 values between 0 and 5:\n", g

# reshaping 3X4 array to 2X2X3 array
arr = np.array([[1, 2, 3, 4],
                [5, 2, 4, 2],
                [1, 2, 0, 1]])
newarr = arr.reshape(2, 2, 3)
print "\nOriginal array:\n", arr
print "Reshaped array:\n", newarr

# flatten array
arr = np.array([[1, 2, 3], [4, 5, 6]])
flarr = arr.flatten()
print "\nOriginal array:\n", arr
print "Fattened array:\n", flarr
```

Output:

```
Array created using passed list:
[[ 1.  2.  4.]
 [ 5.  8.  7.]]

Array created using passed tuple:
[1 3 2]

An array initialized with all zeros:
[[ 0.  0.  0.  0.]
 [ 0.  0.  0.  0.]
 [ 0.  0.  0.  0.]]

An array initialized with all 6s. Array type is complex:
[[ 6.+0.j  6.+0.j  6.+0.j]
 [ 6.+0.j  6.+0.j  6.+0.j]
 [ 6.+0.j  6.+0.j  6.+0.j]]

A random array:
[[ 0.10997504  0.68059294]
 [ 0.31917537  0.87140165]]

A sequential array with steps of 5:
[ 0  5 10 15 20 25]

A sequential array with 10 values between 0 and 5:
[ 0.          0.55555556  1.11111111  1.66666667  2.22222222  2.77777778
  3.33333333  3.88888889  4.44444444  5.        ]

Original array:
[[1 2 3 4]
 [5 2 4 2]
 [1 2 0 1]]
Reshaped array:
[[[1 2 3]
  [4 5 2]]

 [[4 2 1]
  [2 0 1]]]

Original array:
[[1 2 3]
 [4 5 6]]
Fattened array:
[1 2 3 4 5 6]
```

3. Array Indexing: Knowing the basics of array indexing is important for analysing and manipulating the array object. NumPy offers many ways to do array indexing.

- **Slicing:** Just like lists in python, NumPy arrays can be sliced. As arrays can be multidimensional, you need to specify a slice for each dimension of the array.
- **Integer array indexing:** In this method, lists are passed for indexing for each dimension. One to one mapping of corresponding elements is done to construct a new arbitrary array.
- **Boolean array indexing:** This method is used when we want to pick elements from array which satisfy some condition.

```
# Python program to demonstrate indexing in numpy
import numpy as np

# an exemplar array
arr = np.array([[-1, 2, 0, 4],
              [4, -0.5, 6, 0],
              [2.6, 0, 7, 8],
              [3, -7, 4, 2.0]])

# slicing
temp = arr[:2, ::2]
print "Array with first 2 rows and alternate columns(0 and 2):\n", temp

# integer array indexing example
temp = arr[[0, 1, 2, 3], [3, 2, 1, 0]]
print "\nElements at indices (0, 3), (1, 2), (2, 1), (3, 0):\n", temp

# boolean array indexing example
cond = arr > 0    # cond is a boolean array
temp = arr[cond]
print "\nElements greater than 0:\n", temp
```

Output:

```
Array with first 2 rows and alternate columns(0 and 2):
[[-1.  0.]
 [ 4.  6.]]

Elements at indices (0, 3), (1, 2), (2, 1), (3, 0):
[ 4.  6.  0.  3.]

Elements greater than 0:
[ 2.   4.   4.   6.   2.6  7.   8.   3.   4.   2. ]
```

4. Basic operations: Plethora of built-in arithmetic functions are provided in NumPy.

- **Operations on single array:** We can use overloaded arithmetic operators to do element-wise operation on array to create a new array. In case of +=, -=, *= operators, the exsisting array is modified.

```
# Python program to demonstrate basic operations on single array
import numpy as np

a = np.array([1, 2, 5, 3])

# add 1 to every element
print "Adding 1 to every element:", a+1

# subtract 3 from each element
print "Subtracting 3 from each element:", a-3

# multiply each element by 10
print "Multiplying each element by 10:", a*10

# square each element
print "Squaring each element:", a**2

# modify existing array
a *= 2
print "Doubled each element of original array:", a

# transpose of array
a = np.array([[1, 2, 3], [3, 4, 5], [9, 6, 0]])
print "\nOriginal array:\n", a
print "Transpose of array:\n", a.T
```

Output:

```
Adding 1 to every element: [2 3 6 4]
Subtracting 3 from each element: [-2 -1  2  0]
Multiplying each element by 10: [10 20 50 30]
Squaring each element: [ 1  4 25  9]
Doubled each element of original array: [ 2  4 10  6]

Original array:
[[1 2 3]
 [3 4 5]
 [9 6 0]]
Transpose of array:
[[1 3 9]
 [2 4 6]
 [3 5 0]]
```

- **Unary operators:** Many unary operations are provided as a method of **ndarray** class. This includes sum, min, max, etc. These functions can also be applied row-wise or column-wise by setting an axis parameter.

```
# Python program to demonstrate unary operators in numpy
import numpy as np

arr = np.array([[1, 5, 6],
              [4, 7, 2],
              [3, 1, 9]])
```

```
# maximum element of array
print "Largest element is:", arr.max()
print "Row-wise maximum elements:", arr.max(axis = 1)

# minimum element of array
print "Column-wise minimum elements:", arr.min(axis = 0)

# sum of array elements
print "Sum of all array elements:", arr.sum()

# cumulative sum along each row
print "Cumulative sum along each row:\n", arr.cumsum(axis = 1)
```

Output:

```
Largest element is: 9
Row-wise maximum elements: [6 7 9]
Column-wise minimum elements: [1 1 2]
Sum of all array elements: 38
Cumulative sum along each row:
[[ 1  6 12]
 [ 4 11 13]
 [ 3  4 13]]
```

- **Binary operators:** These operations apply on array elementwise and a new array is created. You can use all basic arithmetic operators like +, -, /, *, etc. In case of +=, -=, *= operators, the exsisting array is modified.*

```
# Python program to demonstrate binary operators in Numpy
import numpy as np

a = np.array([[1, 2],
            [3, 4]])
b = np.array([[4, 3],
            [2, 1]])

# add arrays
print "Array sum:\n", a + b

# multiply arrays (elementwise multiplication)
print "Array multiplication:\n", a*b

# matrix multiplication
print "Matrix multiplication:\n", a.dot(b)
```

Output:

```
Array sum:
[[5 5]
 [5 5]]
Array multiplication:
[[4 6]
 [6 4]]
Matrix multiplication:
[[ 8  5]
 [20 13]]
```

- **Universal functions (ufunc):** NumPy provides familiar mathematical functions such as sin, cos, exp, etc. These functions also operate elementwise on an array, producing an array as output.

Note: All the operations we did above using overloaded operators can be done using ufuncs like np.add, np.subtract, np.multiply, np.divide, np.sum, etc.

```
# Python program to demonstrate universal functions in numpy
import numpy as np

# create an array of sine values
a = np.array([0, np.pi/2, np.pi])
print "Sine values of array elements:", np.sin(a)

# exponential values
a = np.array([0, 1, 2, 3])
print "Exponent of array elements:", np.exp(a)

# square root of array values
print "Square root of array elements:", np.sqrt(a)
```

Output:

```
Sine values of array elements: [  0.00000000e+00   1.00000000e+00   1.22464680e-16]
Exponent of array elements: [  1.          2.71828183   7.3890561   20.08553692]
Square root of array elements: [ 0.          1.          1.41421356  1.73205081]
```

4. Sorting array: There is a simple **np.sort** method for sorting NumPy arrays. Let’s explore it a bit.

```
# Python program to demonstrate sorting in numpy
import numpy as np
```

```
a = np.array([[1, 4, 2],
  [3, 4,6],
  [0, -1, 5]])

# sorted array
print "Array elements in sorted order:\n", np.sort(a, axis = None)

# sort array row-wise
print "Row-wise sorted array:\n", np.sort(a, axis = 1)

# specify sort algorithm
print "Column wise sort by applying merge-sort:\n", np.sort
(a, axis = 0, kind = 'mergesort')

# example to show sorting of structured array
## set alias names for dtypes
dtypes = [('name', 'S10'), ('grad_year', int), ('cgpa', float)]
## values to be put in array
values = [('Hrithik', 2009, 8.5), ('Ajay', 2008, 8.7), ('Pankaj', 2008, 7.9),
('Aakash', 2009, 9.0)]
## creating array
arr = np.array(values, dtype = dtypes)
print "\nArray sorted by names:\n", np.sort(arr, order = 'name')
print "Array sorted by grauation year and then cgpa:\n", np.sort(arr, order
= ['grad_year', 'cgpa'])
```

Output:

```
Array elements in sorted order:
[-1  0  1  2  3  4  4  5  6]
Row-wise sorted array:
[[ 1  2  4]
 [ 3  4  6]
 [-1  0  5]]
Column wise sort by applying merge-sort:
[[ 0 -1  2]
 [ 1  4  5]
 [ 3  4  6]]

Array sorted by names:
[('Aakash', 2009, 9.0) ('Ajay', 2008, 8.7) ('Hrithik', 2009, 8.5)
 ('Pankaj', 2008, 7.9)]
Array sorted by grauation year and then cgpa:
[('Pankaj', 2008, 7.9) ('Ajay', 2008, 8.7) ('Hrithik', 2009, 8.5)
 ('Aakash', 2009, 9.0)]
```

So, this was a brief yet concise introduction-cum-tutorial of the NumPy library.
For more detailed study, please refer NumPy Reference Guide .
This article is contributed by Nikhil Kumar. 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.

Python
Python-numpy

GET and POST requests using Python

This post discusses two HTTP (Hypertext Transfer Protocol) request methods GET and POST requests in Python and their implementation in python.

What is HTTP?

HTTP is a set of protocols designed to enable communication between clients and servers. It works as a request-response protocol between a client and server.
A web browser may be the client, and an application on a computer that hosts a web site may be the server.

So, to request a response from the server, there are mainly two methods:

1. **GET** : to request data from the server.
2. **POST** : to submit data to be processed to the server.

Here is a simple diagram which explains the basic concept of GET and POST methods.

iservice_post_get

Now, to make HTTP requests in python, we can use several HTTP libraries like:

- httplib
- urllib
- requests

The most elegant and simplest of above listed libraries is Requests. We will be using requests library in this article. To download and install Requests library, use following command:

```
pip install requests
```

OR, download it from here and install manually.

Making a Get request

```
# importing the requests library
import requests

# api-endpoint
URL = "http://maps.googleapis.com/maps/api/geocode/json"

# location given here
location = "delhi technological university"

# defining a params dict for the parameters to be sent to the API
PARAMS = {'address':location}

# sending get request and saving the response as response object
r = requests.get(url = URL, params = PARAMS)

# extracting data in json format
data = r.json()


# extracting latitude, longitude and formatted address
# of the first matching location
latitude = data['results'][0]['geometry']['location']['lat']
longitude = data['results'][0]['geometry']['location']['lng']
formatted_address = data['results'][0]['formatted_address']

# printing the output
print("Latitude:%s\nLongitude:%s\nFormatted Address:%s"
    %(latitude, longitude,formatted_address))
```

Output:

1

The above example finds latitude, longitude and formatted address of a given location by sending a GET request to the Google Maps API. An API (Application Programming Interface) enables you to access the internal features of a program in a limited fashion. And in most cases, the data provided is in JSON(JavaScript Object Notation) format (which is implemented as dictionary objects in Python!).

Important points to infer :

```
PARAMS = {'address':location}
```

The URL for a GET request generally carries some parameters with it. For requests library, parameters can be defined as a dictionary. These parameters are later parsed down and added to the base url or the api-endpoint.
To understand the parameters role, try to print **r.url** after the response object is created. You will see something like this:

```
http://maps.googleapis.com/maps/api/geocode/json?address=delhi+technological+university
```

This is the actual URL on which GET request is made

```
r = requests.get(url = URL, params = PARAMS)
```

Here we create a response object ‘r’ which will store the request-response. We use requests.get() method since we are sending a GET request. The two arguments we pass are url and the parameters dictionary.

```
data = r.json()
```

Now, in order to retrieve the data from the response object, we need to convert the raw response content into a JSON type data structure. This is achieved by using json() method. Finally, we extract the required information by parsing down the JSON type object.

Making a POST request

```
# importing the requests library
import requests

# defining the api-endpoint
API_ENDPOINT = "http://pastebin.com/api/api_post.php"

# your API key here
API_KEY = "XXXXXXXXXXXXXXXXX"

# your source code here
source_code = '''
print("Hello, world!")
a = 1
b = 2
print(a + b)
'''

# data to be sent to api
data = {'api_dev_key':API_KEY,
        'api_option':'paste',
        'api_paste_code':source_code,
        'api_paste_format':'python'}

# sending post request and saving response as response object
r = requests.post(url = API_ENDPOINT, data = data)

# extracting response text
pastebin_url = r.text
print("The pastebin URL is:%s"%pastebin_url)
```

This example explains how to paste your **source_code** to pastebin.com by sending POST request to the PASTEBIN API.
First of all, you will need to generate an API key by signing up here and then access your API key here.

Important features of this code:

```
data = {'api_dev_key':API_KEY,
        'api_option':'paste',
        'api_paste_code':source_code,
        'api_paste_format':'python'}
```

Here again, we will need to pass some data to API server. We store this data as a dictionary.

```
r = requests.post(url = API_ENDPOINT, data = data)
```

Here we create a response object ‘r’ which will store the request-response. We use requests.post() method since we are sending a POST request. The two arguments we pass are url and the data dictionary.

```
pastebin_url = r.text
```

In response, the server processes the data sent to it and sends the pastebin URL of your **source_code** which can be simply accessed by **r.text .**

requests.post method could be used for many other tasks as well like filling and submitting the web forms, posting on your FB timeline using the Facebook Graph API, etc.

Here are some important points to ponder upon:

- When the method is GET, all form data is encoded into the URL, appended to the **action** URL as query string parameters. With POST, form data appears within the **message body** of the HTTP request.
- In GET method, the parameter data is limited to what we can stuff into the request line (URL). Safest to use less than 2K of parameters, some servers handle up to 64K.No such problem in POST method since we send data in **message body** of the HTTP request, not the URL.
- Only ASCII characters are allowed for data to be sent in GET method.There is no such restriction in POST method.
- GET is less secure compared to POST because data sent is part of the URL. So, GET method should not be used when sending passwords or other sensitive information.

This blog is contributed by **Nikhil Kumar**. 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.

Regular Expression in Python with Examples | Set 1

Module Regular Expressions(RE) specifies a set of strings(pattern) that matches it.

To understand the RE analogy, MetaCharacters are useful, important and will be used in functions of module re.

There are a total of 14 metacharcters and will be discussed as they follow into functions:

```
\   Used to drop the special meaning of character
    following it (discussed below)
[]  Represent a character class
^   Matches the beginning
$   Matches the end
.   Matches any character except newline
?   Matches zero or one occurrence.
|   Means OR (Matches with any of the characters
    separated by it.
*   Any number of occurrences (including 0 occurrences)
+   One ore more occurrences
{}  Indicate number of occurrences of a preceding RE
    to match.
()  Enclose a group of REs
```

- **Function compile()**

  Regular expressions are compiled into pattern objects, which have methods for various operations such as searching for pattern matches or performing string substitutions.

```
# Module Regular Expression is imported using __import__().
import re

# compile() creates regular expression character class [a-e],
# which is equivalent to [abcde].
# class [abcde] will match with string with 'a', 'b', 'c', 'd', 'e'.
p = re.compile('[a-e]')

# findall() searches for the Regular Expression and return a list upon finding
print(p.findall("Aye, said Mr. Gibenson Stark"))
```

Output:

```
['e', 'a', 'd', 'b', 'e', 'a']
```

Understanding the Output:

First occurrence is ‘e’ in “Aye” and not ‘A’, as it being Case Sensitive.

Next Occurrence is ‘a’ in “said”, then ‘d’ in “said”, followed by ‘b’ and ‘e’ in “Gibenson”, the Last ‘a’ matches with “Stark”.

Metacharacter blackslash ‘\’ has a very important role as it signals various sequences. If the blackslash is to be used without its special meaning as metacharcter, use’\\’

```
\d   Matches any decimal digit, this is equivalent
     to the set class [0-9].
\D   Matches any non-digit character.
\s   Matches any whitespace character.
\S   Matches any non-whitespace character
\w   Matches any alphanumeric character, this is
     equivalent to the class [a-zA-Z0-9_].
\W   Matches any non-alphanumeric character.
```

Set class [\s,.] will match any whitespace character, ‘,’, or,’.’ .

```
import re

# \d is equivalent to [0-9].
p = re.compile('\d')
print(p.findall("I went to him at 11 A.M. on 4th July 1886"))

# \d+ will match a group on [0-9], group of one or greater size
p = re.compile('\d+')
print(p.findall("I went to him at 11 A.M. on 4th July 1886"))
```

Output:

```
['1', '1', '4', '1', '8', '8', '6']
['11', '4', '1886']
```

```
import re

# \w is equivalent to [a-zA-Z0-9_].
p = re.compile('\w')
print(p.findall("He said * in some_lang."))
```

```
# \w+ matches to group of alphanumeric charcter.
p = re.compile('\w+')
print(p.findall("I went to him at 11 A.M., he said *** in some_language."))

# \W matches to non alphanumeric characters.
p = re.compile('\W')
print(p.findall("he said *** in some_language."))
```

Output:

```
['H', 'e', 's', 'a', 'i', 'd', 'i', 'n', 's', 'o', 'm', 'e', '_', 'l', 'a', 'n', 'g']
['I', 'went', 'to', 'him', 'at', '11', 'A', 'M', 'he', 'said', 'in', 'some_language']
[' ', ' ', '*', '*', '*', ' ', ' ', '.']
```

```
import re

# '*' replaces the no. of occurrence of a character.
p = re.compile('ab*')
print(p.findall("ababbaabbb"))
```

Output:

```
['ab', 'abb', 'a', 'abbb']
```

Understanding the Output:
Our RE is ab*, which ‘a’ accompanied by any no. of ‘b’s, starting from 0.
Output ‘ab’, is valid because of singe ‘a’ accompanied by single ‘b’.
Output ‘abb’, is valid because of singe ‘a’ accompanied by 2 ‘b’.
Output ‘a’, is valid because of singe ‘a’ accompanied by 0 ‘b’.
Output ‘abbb’, is valid because of singe ‘a’ accompanied by 3 ‘b’.

- **Function split()**
  Split string by the occurrences of a character or a pattern, upon finding that pattern, the remaining characters from the string are returned as part of the resulting list.
  Syntax :

```
re.split(pattern, string, maxsplit=0, flags=0)
```

The First parameter, pattern denotes the regular expression, string is the given string in which pattern will be searched for and in which splitting occurs, maxsplit if not provided is considered to be zero ‘0’, and if any nonzero value is provided, then at most that many splits occurs. If maxsplit = 1, then the string will split once only, resulting in a list of length 2. The flags are very useful and can help to shorten code, they are not necessary parameters, eg: flags = re.IGNORECASE, In this split, case will be ignored.

```
from re import split

# '\W+' denotes Non-Alphanumeric Characters or group of characters
# Upon finding ',' or whitespace ' ', the split(), splits the string from that point
print(split('\W+', 'Words, words , Words'))
print(split('\W+', "Word's words Words"))

# Here ':', ' ' ,',' are not AlphaNumeric thus, the point where splitting occurs
print(split('\W+', 'On 12th Jan 2016, at 11:02 AM'))

# '\d+' denotes Numeric Characters or group of characters
# Spliting occurs at '12', '2016', '11', '02' only
print(split('\d+', 'On 12th Jan 2016, at 11:02 AM'))
```

Output:

```
['Words', 'words', 'Words']
['Word', 's', 'words', 'Words']
['On', '12th', 'Jan', '2016', 'at', '11', '02', 'AM']
['On ', 'th Jan ', ', at ', ':', ' AM']
```

```
import re

# Splitting will occurs only once, at '12', returned list will have length 2
print(re.split('\d+', 'On 12th Jan 2016, at 11:02 AM', 1))

# 'Boy' and 'boy' will be treated same when flags = re.IGNORECASE
print(re.split('[a-f]+', 'Aey, Boy oh boy, come here', flags = re.IGNORECASE))
print(re.split('[a-f]+', 'Aey, Boy oh boy, come here'))
```

Output:

```
['On ', 'th Jan 2016, at 11:02 AM']
['', 'y, ', 'oy oh ', 'oy, ', 'om', ' h', 'r', '']
['A', 'y, Boy oh ', 'oy, ', 'om', ' h', 'r', '']
```

- **Function sub()**
  Syntax:

```
re.sub(pattern, repl, string, count=0, flags=0)
```

The ‘sub’ in the function stands for SubString, a certain regular expression pattern is searched in the given string(3rd parameter), and upon finding the substring pattern is replaced by by repl(2nd parameter), count checks and maintains the number of times this occurs.

```
import re

# Regular Expression pattern 'ub' matches the string at "Subject" and "Uber".
# As the CASE has been ignored, using Flag, 'ub' should match twice with the string
# Upon matching, 'ub' is replaced by '~*' in "Subject", and in "Uber", 'Ub' is replaced.
print(re.sub('ub', '~*' , 'Subject has Uber booked already', flags = re.IGNORECASE))

# Consider the Case Senstivity, 'Ub' in "Uber", will not be reaplced.
print(re.sub('ub', '~*' , 'Subject has Uber booked already'))

# As count has been given value 1, the maximum times replacement occurs is 1
print(re.sub('ub', '~*' , 'Subject has Uber booked already', count=1, flags = re.IGNORECASE))

# 'r' before the patter denotes RE, \s is for start and end of a String.
print(re.sub(r'\sAND\s', ' & ', 'Baked Beans And Spam', flags=re.IGNORECASE))
```

Output

```
S~*ject has ~*er booked already
S~*ject has Uber booked already
S~*ject has Uber booked already
Baked Beans & Spam
```

- **Function subn()**
  Syntax:

```
 re.subn(pattern, repl, string, count=0, flags=0)
```

subn() is similar to sub() in all ways, except in its way to providing output. It returns a tuple with count of total of replacement and the new string rather than just the string.

```
import re
print(re.subn('ub', '~*' , 'Subject has Uber booked already'))
t = re.subn('ub', '~*' , 'Subject has Uber booked already', flags = re.IGNORECASE)
print(t)
print(len(t))

# This will give same output as sub() would have
print(t[0])
```

Output

```
('S~*ject has Uber booked already', 1)
('S~*ject has ~*er booked already', 2)
Length of Tuple is:  2
S~*ject has ~*er booked already
```

- **Function escape()**
  Syntax:

```
re.escape(string)
```

Return string with all non-alphanumerics backslashed, this is useful if you want to match an arbitrary literal string that may have regular expression metacharacters in it.

```
import re

# escape() returns a string with BackSlash '\', before every Non-Alphanumeric Character
# In 1st case only ' ', is not alphanumeric
# In 2nd case, ' ', caret '^', '-', '[]', '\' are not alphanumeric
print(re.escape("This is Awseome even 1 AM"))
print(re.escape("I Asked what is this [a-9], he said \t ^WoW"))
```

Output

```
This\ is\ Awseome\ even\ 1\ AM
I\ Asked\ what\ is\ this\ \[a\-9\]\,\ he\ said\ \ \ \^WoW
```

Related Article :
http://www.geeksforgeeks.org/regular-expressions-python-set-1-search-match-find/

Reference:
https://docs.python.org/2/library/re.html

This article is contributed by **Piyush Doorwar**. 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

Regular Expressions in Python | Set 2 (Search, Match and Find All)

Regular Expression in Python with Examples | Set 1

The module **re** provides support for regular expressions in Python. Below are main methods in this module.

Searching an occurrence of pattern

re.search() : This method either returns None (if the pattern doesn't match), or a re.MatchObject that contains information about the matching part of the string. This method stops after the first match, so this is best suited for testing a regular expression more than extracting data.

```
# A Python program to demonstrate working of re.match().
import re

# Lets use a regular expression to match a date string
# in the form of Month name followed by day number
regex = r"([a-zA-Z]+) (\d+)"

match = re.search(regex, "I was born on June 24")

if match != None:

    # We reach here when the expression "([a-zA-Z]+) (\d+)"
    # matches the date string.

    # This will print [14, 21), since it matches at index 14
    # and ends at 21.
    print "Match at index %s, %s" % (match.start(), match.end())

    # We us group() method to get all the matches and
    # captured groups. The groups contain the matched values.
    # In particular:
    #    match.group(0) always returns the fully matched string
    #    match.group(1) match.group(2), ... return the capture
    #    groups in order from left to right in the input string
    #    match.group() is equivalent to match.group(0)

    # So this will print "June 24"
    print "Full match: %s" % (match.group(0))

    # So this will print "June"
    print "Month: %s" % (match.group(1))

    # So this will print "24"
    print "Day: %s" % (match.group(2))

else:
    print "The regex pattern does not match."
```

Output :

```
Match at index 14, 21
Full match: June 24
Month: June
Day: 24
```

Matching a Pattern with Text

re.match() : This function attempts to match pattern to whole string. The re.match function returns a match object on success, None on failure.

```
re.match(pattern, string, flags=0)

pattern : Regular expression to be matched.
string : String where p attern is searched
flags : We can specify different flags
        using bitwise OR (|).
```

```
# A Python program to demonstrate working
# of re.match().
import re

# a sample function that uses regular expressions
# to find month and day of a date.
def findMonthAndDate(string):

    regex = r"([a-zA-Z]+) (\d+)"
    match = re.match(regex, string)

    if match == None:
        print "Not a valid date"
        return

    print "Given Data: %s" % (match.group())
    print "Month: %s" % (match.group(1))
    print "Day: %s" % (match.group(2))


# Driver Code
findMonthAndDate("Jun 24")
```

```
print("")
findMonthAndDate("I was born on June 24")
```

Finding all occurrences of a pattern

re.findall() : Return all non-overlapping matches of pattern in string, as a list of strings. The string is scanned left-to-right, and matches are returned in the order found (Source : Python Docs).

```
# A Python program to demonstrate working of
# findall()
import re

# A sample text string where regular expression
# is searched.
string  = """Hello my Number is 123456789 and
          my friend's number is 987654321"""

# A sample regular expression to find digits.
regex = '\d+'

match = re.findall(regex, string)
print(match)

# This example is contributed by Ayush Saluja.
```

Output :

```
['123456789', '987654321']
```

Regular expression is a vast topic. It's a complete library. Regular expressions can do a lot of stuff. You can Match, Search, Replace, Extract a lot of data. For example, below small code is so powerful that it can extract email address from a text. So we can make our own Web Crawlers and scrappers in python with easy.Look at below regex.

```

# extract all email addresses and add them into the resulting set
new_emails = set(re.findall(r"[a-z0-9\.\-+_]+@[a-z0-9\.\-+_]+\.[a-z]+",
                        text, re.I))
```

We will soon be discussing more methods on regular expressions.

This article is contributed by **Shwetanshu Rohatgi**. 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

OS Module in Python with Examples

The OS module in python provides functions for interacting with the operating system. OS, comes under Python's standard utility modules. This module provides a portable way of using operating system dependent functionality. The *os* and *os.path* modules include many functions to interact with the file system.

Following are some functions in OS module:

1. os.name: This function gives the name of the operating system dependent module imported. The following names have currently been registered: 'posix', 'nt', 'os2', 'ce', 'java' and 'riscos'

```
import os
print(os.name)
```

Output:

```
posix
```

Note: It may give different output on different interpreters, such as 'posix' when you run the code here.

2. os.getcwd(): Function os.getcwd(), returns the Current Working Directory(CWD) of the file used to execute the code, can vary from system to system.

```
import os
print(os.getcwd())
# To print absolute path on your system
# os.path.abspath('.')

# To print files and directories in the current directory
# on your system
# os.listdir('.')
```

Output:

```
C:\Users\GFG\Desktop\ModuleOS
```

Note: In case of GFG interpreter, directory used is \root.

3. os.error: All functions in this module raise OSError in the case of invalid or inaccessible file names and paths, or other arguments that have the correct type, but are not accepted by the operating system. os.error is an alias for built-in OSError exception.

```
import os
try:
    # If the file does not exist,
    # then it would throw an IOError
    filename = 'GFG.txt'
    f = open(filename, 'rU')
    text = f.read()
    f.close()

# Control jumps directly to here if
#any of the above lines throws IOError.
except IOError:

    # print(os.error) will <class 'OSError'>
    print('Problem reading: ' + filename)

# In any case, the code then continues with
# the line after the try/except
```

Output:

```
Problem reading: GFG.txt
```

File Object Manipulation

4. os.popen(): This method opens a pipe to or from command. The return value can be read or written depending on whether mode is ‘r’ or ‘w’.
Syntax:

```
 os.popen(command[, mode[, bufsize]])
```

Parameters mode & bufsize are not necessary parameters, if not provided, default ‘r’ is taken for mode.

```
import os
fd = "GFG.txt"

# popen() is similar to open()
file = open(fd, 'w')
file.write("Hello")
file.close()
file = open(fd, 'r')
text = file.read()
print(text)

# popen() provides a pipe/gateway and accesses the file directly
file = os.popen(fd, 'w')
file.write("Hello")
# File not closed, shown in next function.
```

Output:

```
Hello
```

Note: Output for popen() will not be shown, there would be direct changes into the file.

5. os.close(): Close file descriptor fd. A file opened using open(), can be closed by close()only. But file opened through os.popen(), can be closed with close() or os.close(). If we try closing a file opened with open(), using os.close(), Python would throw TypeError.

```
import os
fd = "GFG.txt"
file = open(fd, 'r')
text = file.read()
print(text)
os.close(file)
```

Output:

```
Traceback (most recent call last):
  File "C:\Users\GFG\Desktop\GeeksForGeeksOSFile.py", line 6, in
    os.close(file)
TypeError: an integer is required (got type _io.TextIOWrapper)
```

Note: The same error may not be thrown, due to non-existent of file or permission privilege.

6. os.rename(): A file old.txt can be renamed to new.txt, using the function os.rename(). The name of the file changes only if, the file exists and user has sufficient privilege permission to change the file.

```
import os
fd = "GFG.txt"
os.rename(fd,'New.txt')
os.rename(fd,'New.txt')
```

Output:

```
Traceback (most recent call last):
  File "C:\Users\GFG\Desktop\ModuleOS\GeeksForGeeksOSFile.py", line 3, in
    os.rename(fd,'New.txt')
FileNotFoundError: [WinError 2] The system cannot find the
file specified: 'GFG.txt' -> 'New.txt'
```

Understanding the Output: A file name “GFG.txt” exists, thus when os.rename() is used the first time, the file gets renamed. Upon calling the fuction os.rename() second time, file “New.txt” exists and not “GFG.txt”
thus Python throws FileNotFoundError.

Reference: https://docs.python.org/2/library/os.html

This article is contributed by **Piyush Doorwar**. 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

copy in Python (Deep Copy and Shallow Copy)

Python defines a module which allows to deep copy or shallow copy mutable object using the inbuilt functions present in the module “**copy**“.
Assignment statements in Python do not copy objects, they create bindings between a target and an object. For collections that are mutable or contain mutable items, a copy is sometimes needed so one can change one copy without changing the other.

Deep copy

Deep

Image Source : http://docs.roguewave.com/sourcepro/11.1/html/toolsug/6-4.html

In case of deep copy, a copy of object is copied in other object. It means that **any changes** made to a copy of object **do not reflect** in the original object.
In python, this is implemented using “**deepcopy()**” function.

```
# Python code to demonstrate copy operations

# importing "copy" for copy operations
import copy

# initializing list 1
li1 = [1, 2, [3,5], 4]

# using deepcopy to deep copy
li2 = copy.deepcopy(li1)

# original elements of list
print ("The original elements before deep copying")
for i in range(0,len(li1)):
 print (li1[i],end=" ")

print("\r")

# adding and element to new list
li2[2][0] = 7

# Change is reflected in l2
print ("The new list of elements after deep copying ")
for i in range(0,len( li1)):
 print (li2[i],end=" ")

print("\r")

# Change is NOT reflected in original list
# as it is a deep copy
print ("The original elements after deep copying")
for i in range(0,len( li1)):
 print (li1[i],end=" ")
```

Output:

```
The original elements before deep copying
1 2 [3, 5] 4
```

```
The new list of elements after deep copying
1 2 [7, 5] 4
The original elements after deep copying
1 2 [3, 5] 4
```

In the above example, the change made in the list **did not** effect in other list, indicating the list is deep copied.

Shallow copy

Image Source : http://docs.roguewave.com/sourcepro/11.1/html/toolsug/6-4.html

In case of shallow copy, a reference of object is copied in other object. It means that **any changes** made to a copy of object **do reflect** in the original object.
In python, this is implemented using “**copy()**” function.

```
# Python code to demonstrate copy operations

# importing "copy" for copy operations
import copy

# initializing list 1
li1 = [1, 2, [3,5], 4]

# using copy to shallow copy
li2 = copy.copy(li1)

# original elements of list
print ("The original elements before shallow copying")
for i in range(0,len(li1)):
    print (li1[i],end=" ")

print("\r")

# adding and element to new list
li2[2][0] = 7

# checking if change is reflected
print ("The original elements after shallow copying")
for i in range(0,len( li1)):
    print (li1[i],end=" ")
```

Output:

```
The original elements before shallow copying
1 2 [3, 5] 4
The original elements after shallow copying
1 2 [7, 5] 4
```

In the above example, the change made in the list **did** effect in other list, indicating the list is shallow copied.

Important Points:
The difference between shallow and deep copying is only relevant for compound objects (objects that contain other objects, like lists or class instances):

- A shallow copy constructs a new compound object and then (to the extent possible) inserts references into it to the objects found in the original.
- A deep copy constructs a new compound object and then, recursively, inserts copies into it of the objects found in the original.

Reference: Python official Documentation

This article is contributed by **Manjeet Singh**. 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

Import module in Python

Import in python is similar to #include header_file in C/C++. Python modules can get access to code from another module by importing the file/function using import. The import statement is the most common way of invoking the import machinery, but it is not the only way.

import module_name
When import is used, it searches for the module initially in the local scope by calling __import__() function. The value returned by the function are then reflected in the output of

the initial code.

```
import math
print(math.pi)
```

Output:

```
3.141592653589793
```

import module_name.member_name
In the above code module math is imported, and its variables can be accessed by considering it to be a class and pi as its object.
The value of pi is returned by __import__().
pi as whole can be imported into our intial code, rather than importing the whole module.

```
from math import pi

# Note that in the above example,
# we used math.pi. Here we have used
# pi directly.
print(pi)
```

Output:

```
3.141592653589793
```

from module_name import *
In the above code module math is not imported, rather just pi has been imported as a variable.
All the functions and constants can be imported using *.

```
from math import *
print(pi)
print(factorial(6))
```

Output:

```
3.141592653589793
720
```

As said above import uses __import__() to search for module, and if not found, it would raise ImportError

```
import mathematics
print(mathematics.pi)
```

Output:

```
Traceback (most recent call last):
  File "C:/Users/GFG/Tuples/xxx.py", line 1, in
    import mathematics
ImportError: No module named 'mathematics'
```

This article is contributed by **Piyush Doorwar**. 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.

GATE CS Corner   Company Wise Coding Practice

Python

Reloading modules in Python

reload() reloads a previously imported module. This is useful if you have edited the module source file using an external editor and want to try out the new version without leaving the Python interpreter. The return value is the module object.

Note: The argument should be a module which has been successfully imported.

Usage:

For Python2.x

```
reload(module)
```

For above 2.x and <=Python3.3

```
import imp
imp.reload(module)
```

For >=Python3.4

```
import importlib
```

```
importlib.reload(module)
```

For more information, check out reload().

This article is contributed by **Sri Sanketh Uppalapati**. 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

Deque in Python

Deque can be implemented in python using the module “**collections**“. Deque is preferred over list in the cases where we need quicker append and pop operations from both the ends of container, as deque provides an **O(1)** time complexity for append and pop operations as compared to list which provides O(n) time complexity.

Operations on deque :

1. append() :- This function is used to **insert** the value in its argument to the **right end** of deque.

2. appendleft() :- This function is used to **insert** the value in its argument to the **left end** of deque.

3. pop() :- This function is used to **delete** an argument from the **right end** of deque.

4. popleft() :- This function is used to **delete** an argument from the **left end** of deque.

```
# Python code to demonstrate working of
# append(), appendleft(), pop(), and popleft()

# importing "collections" for deque operations
import collections

# initializing deque
de = collections.deque([1,2,3])

# using append() to insert element at right end
# inserts 4 at the end of deque
de.append(4)

# printing modified deque
print ("The deque after appending at right is : ")
print (de)

# using appendleft() to insert element at right end
# inserts 6 at the beginning of deque
de.appendleft(6)

# printing modified deque
print ("The deque after appending at left is : ")
print (de)

# using pop() to delete element from right end
# deletes 4 from the right end of deque
de.pop()

# printing modified deque
print ("The deque after deleting from right is : ")
print (de)

# using popleft() to delete element from left end
# deletes 6 from the left end of deque
de.popleft()

# printing modified deque
print ("The deque after deleting from left is : ")
print (de)
```

Output:

```
The deque after appending at right is :
deque([1, 2, 3, 4])
The deque after appending at left is :
deque([6, 1, 2, 3, 4])
The deque after deleting from right is :
deque([6, 1, 2, 3])
The deque after deleting from left is :
deque([1, 2, 3])
```

5. index(ele, beg, end) :- This function **returns the first index of the value** mentioned in arguments, **starting searching from beg till end** index.

6. insert(i, a) :- This function **inserts the value** mentioned in arguments(a) **at index(i)** specified in arguments.

7. remove() :- This function **removes the first occurrence** of value mentioned in arguments.

8. count() :- This function **counts the number of occurrences** of value mentioned in arguments.

```
# Python code to demonstrate working of
# insert(), index(), remove(), count()

# importing "collections" for deque operations
import collections

# initializing deque
de = collections.deque([1, 2, 3, 3, 4, 2, 4])

# using index() to print the first occurrence of 4
print ("The number 4 first occurs at a position : ")
print (de.index(4,2,5))

# using insert() to insert the value 3 at 5th position
de.insert(4,3)

# printing modified deque
print ("The deque after inserting 3 at 5th position is : ")
print (de)

# using count() to count the occurrences of 3
print ("The count of 3 in deque is : ")
print (de.count(3))

# using remove() to remove the first occurrence of 3
de.remove(3)

# printing modified deque
print ("The deque after deleting first occurrence of 3 is : ")
print (de)
```

Output:

```
The number 4 first occurs at a position :
4
The deque after inserting 3 at 5th position is :
deque([1, 2, 3, 3, 3, 4, 2, 4])
The count of 3 in deque is :
3
The deque after deleting first occurrence of 3 is :
deque([1, 2, 3, 3, 4, 2, 4])
```

9. extend(iterable) :- This function is used to **add multiple values at the right end** of deque. The argument passed is an iterable.

10. extendleft(iterable) :- This function is used to **add multiple values at the left end** of deque. The argument passed is an iterable. **Order is reversed** as a result of left appends.

11. reverse() :- This function is used to **reverse order** of deque elements.

12. rotate() :- This function **rotates the deque** by the number specified in arguments. **If the number specified is negative, rotation occurs to left. Else rotation is to right.**

```
# Python code to demonstrate working of
# extend(), extendleft(), rotate(), reverse()

# importing "collections" for deque operations
import collections

# initializing deque
de = collections.deque([1, 2, 3,])

# using extend() to add numbers to right end
# adds 4,5,6 to right end
de.extend([4,5,6])

# printing modified deque
print ("The deque after extending deque at end is : ")
print (de)

# using extendleft() to add numbers to left end
# adds 7,8,9 to right end
de.extendleft([7,8,9])

# printing modified deque
print ("The deque after extending deque at beginning is : ")
print (de)

# using rotate() to rotate the deque
# rotates by 3 to left
de.rotate(-3)

# printing modified deque
print ("The deque after rotating deque is : ")
print (de)

# using reverse() to reverse the deque
de.reverse()

# printing modified deque
print ("The deque after reversing deque is : ")
print (de)
```

Output :

```
The deque after extending deque at end is :
deque([1, 2, 3, 4, 5, 6])
The deque after extending deque at beginning is :
deque([9, 8, 7, 1, 2, 3, 4, 5, 6])
The deque after rotating deque is :
deque([1, 2, 3, 4, 5, 6, 9, 8, 7])
The deque after reversing deque is :
deque([7, 8, 9, 6, 5, 4, 3, 2, 1])
```

This article is contributed by **Manjeet Singh**. 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

Namedtuple in Python

Python supports a type of container like dictionaries called "**namedtuples()**" present in module, "**collection**". Like dictionaries they contain keys that are hashed to a particular value. But on contrary, it supports both access from key value and iteration, the functionality that dictionaries lack.

Operations on namedtuple() :

Access Operations

1. Access by index : The attribute values of namedtuple() are ordered and can be accessed using the index number unlike dictionaries which are not accessible by index.

2. Access by keyname : Access by keyname is also allowed as in dictionaries.

3. using getattr() :- This is yet another way to access the value by giving namedtuple and key value as its argument.

```
# Python code to demonstrate namedtuple() and
# Access by name, index and getattr()

# importing "collections" for namedtuple()
import collections

# Declaring namedtuple()
Student = collections.namedtuple('Student',['name','age','DOB'])

# Adding values
S = Student('Nandini','19','2541997')

# Access using index
print ("The Student age using index is : ",end ="")
print (S[1])

# Access using name
print ("The Student name using keyname is : ",end ="")
print (S.name)

# Access using getattr()
print ("The Student DOB using getattr() is : ",end ="")
print (getattr(S,'DOB'))
```

Output :

```
The Student age using index is : 19
The Student name using keyname is : Nandini
The Student DOB using getattr() is : 2541997
```

Conversion Operations

1. _make() :- This function is used to return a **namedtuple() from the iterable** passed as argument.

2. _asdict() :- This function returns **the OrdereDict()** as constructed from the mapped values of namedtuple().

3. using "" (double star) operator** :- This function is used to **convert a dictionary into the namedtuple().**

```
# Python code to demonstrate namedtuple() and
# _make(), _asdict() and "**" operator

# importing "collections" for namedtuple()
import collections

# Declaring namedtuple()
Student = collections.namedtuple('Student',['name','age','DOB'])

# Adding values
S = Student('Nandini','19','2541997')

# initializing iterable
li = ['Manjeet', '19', '411997' ]
```

```
# initializing dict
di = { 'name' : "Nikhil", 'age' : 19 , 'DOB' : '1391997' }

# using _make() to return namedtuple()
print ("The namedtuple instance using iterable is  : ")
print (Student._make(li))

# using _asdict() to return an OrderedDict()
print ("The OrderedDict instance using namedtuple is  : ")
print (S._asdict())

# using ** operator to return namedtuple from dictionary
print ("The namedtuple instance from dict is  : ")
print (Student(**di))
```

Output :

```
The namedtuple instance using iterable is  :
Student(name='Manjeet', age='19', DOB='411997')
The OrderedDict instance using namedtuple is  :
OrderedDict([('name', 'Nandini'), ('age', '19'), ('DOB', '2541997')])
The namedtuple instance from dict is  :
Student(name='Nikhil', age=19, DOB='1391997')
```

Additional Operations

1. _fields :- This function is used to return **all the keynames** of the namespace declared.

2. _replace() :- This function is used to **change the values** mapped with the passed keyname.

```
# Python code to demonstrate namedtuple() and
# _fields and _replace()

# importing "collections" for namedtuple()
import collections

# Declaring namedtuple()
Student = collections.namedtuple('Student',['name','age','DOB'])

# Adding values
S = Student('Nandini','19','2541997')

# using _fields to display all the keynames of namedtuple()
print ("All the fields of students are : ")
print (S._fields)

# using _replace() to change the attribute values of namedtuple
print ("The modified namedtuple is : ")
print(S._replace(name = 'Manjeet'))
```

Output :

```
All the fields of students are :
('name', 'age', 'DOB')
The modified namedtuple is :
Student(name='Manjeet', age='19', DOB='2541997')
```

This article is contributed by **Manjeet Singh**. 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

Heap queue (or heapq) in Python

Heap data structure is mainly used to represent a priority queue. In Python, it is available using "**heapq**" module. The property of this data structure in python is that each time the **smallest of heap element is popped(min heap)**. Whenever elements are pushed or popped, **heap structure in maintained**. The heap[0] element also returns the smallest element each time.

Operations on heap :

1. heapify(iterable) :- This function is used to **convert the iterable into a heap** data structure. i.e. in heap order.

2. heappush(heap, ele) :- This function is used to **insert the element** mentioned in its arguments into heap. The **order is adjusted**, so as **heap structure is maintained**.

3. heappop(heap) :- This function is used to **remove and return the smallest element** from heap. The **order is adjusted**, so as **heap structure is maintained**.

```
# Python code to demonstrate working of
# heapify(), heappush() and heappop()

# importing "heapq" to implement heap queue
import heapq

# initializing list
```

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

# using heapify to convert list into heap
heapq.heapify(li)

# printing created heap
print ("The created heap is : ",end="")
print (list(li))

# using heappush() to push elements into heap
# pushes 4
heapq.heappush(li,4)

# printing modified heap
print ("The modified heap after push is : ",end="")
print (list(li))

# using heappop() to pop smallest element
print ("The popped and smallest element is : ",end="")
print (heapq.heappop(li))
```

Output :

```
The created heap is : [1, 3, 9, 7, 5]
The modified heap after push is : [1, 3, 4, 7, 5, 9]
The popped and smallest element is : 1
```

4. heappushpop(heap, ele) :- This function **combines the functioning of both push and pop operations** in one statement, increasing efficiency. Heap order is maintained after this operation.

5. heapreplace(heap, ele) :- This function also inserts and pops element in one statement, but it is different from above function. In this, **element is first popped, then element is pushed.i.e, the value larger than the pushed value can be returned.**

```
# Python code to demonstrate working of
# heappushpop() and heapreplce()

# importing "heapq" to implement heap queue
import heapq

# initializing list 1
li1 = [5, 7, 9, 4, 3]

# initializing list 2
li2 = [5, 7, 9, 4, 3]

# using heapify() to convert list into heap
heapq.heapify(li1)
heapq.heapify(li2)

# using heappushpop() to push and pop items simultaneously
# pops 2
print ("The popped item using heappushpop() is : ",end="")
print (heapq.heappushpop(li1, 2))

# using heapreplace() to push and pop items simultaneously
# pops 3
print ("The popped item using heapreplace() is : ",end="")
print (heapq.heapreplace(li2, 2))
```

Output :

```
The popped item using heappushpop() is : 2
The popped item using heapreplace() is : 3
```

6. nlargest(k, iterable, key = fun) :- This function is used to **return the k largest elements from the iterable specified and satisfying the key if mentioned.**

7. nsmallest(k, iterable, key = fun) :- This function is used to **return the k smallest elements from the iterable specified and satisfying the key if mentioned.**

```
# Python code to demonstrate working of
# nlargest() and nsmallest()

# importing "heapq" to implement heap queue
import heapq

# initializing list
li1 = [6, 7, 9, 4, 3, 5, 8, 10, 1]

# using heapify() to convert list into heap
heapq.heapify(li1)

# using nlargest to print 3 largest numbers
# prints 10, 9 and 8
print("The 3 largest numbers in list are : ",end="")
print(heapq.nlargest(3, li1))

# using nsmallest to print 3 smallest numbers
# prints 1, 3 and 4
print("The 3 smallest numbers in list are : ",end="")
print(heapq.nsmallest(3, li1))
```

Output :

```
The 3 largest numbers in list are : [10, 9, 8]
The 3 smallest numbers in list are : [1, 3, 4]
```

This article is contributed by **Manjeet Singh**. 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

enum in Python

Enumerations in Python are implemented by using the module named "**enum**".Enumerations are created using **classes**. Enums have **names and values** associated with them.

Properties of enum:

1. Enums can be displayed as **string** or repr.

2. Enums can be checked for their types using **type()**.

3. "**name**" keyword is used to display the name of the enum member.

```
# Python code to demonstrate enumerations

# importing enum for enumerations
import enum

# creating enumerations using class
class Animal(enum.Enum):
    dog = 1
    cat = 2
    lion = 3

# printing enum member as string
print ("The string representation of enum member is : ",end="")
print (Animal.dog)

# printing enum member as repr
print ("The repr representation of enum member is : ",end="")
print (repr(Animal.dog))

# printing the type of enum member using type()
print ("The type of enum member is : ",end ="")
print (type(Animal.dog))

# printing name of enum member using "name" keyword
print ("The name of enum member is : ",end ="")
print (Animal.dog.name)
```

Output :

```
The string representation of enum member is : Animal.dog
The repr representation of enum member is : <Animal.dog: 1>
The type of enum member is : <enum 'Animal'>
The name of enum member is : dog
```

4. Enumerations are **iterable**. They can be iterated using loops

5. Enumerations support **hashing**. Enums can be used in dictionaries or sets.

```
# Python code to demonstrate enumerations
# iterations and hashing
# importing enum for enumerations
import enum

# creating enumerations using class
class Animal(enum.Enum):
    dog = 1
    cat = 2
    lion = 3

# printing all enum members using loop
print ("All the enum values are : ")
for Anim in (Animal):
    print(Anim)

# Hashing enum member as dictionary
di = {}
di[Animal.dog] = 'bark'
di[Animal.lion] = 'roar'

# checking if enum values are hashed successfully
if di=={Animal.dog : 'bark',Animal.lion : 'roar'}:
    print ("Enum is hashed")
else: print ("Enum is not hashed")
```

Output :

```
All the enum values are :
Animal.dog
Animal.cat
Animal.lion
Enum is hashed
```

Accessing Modes : Enum members can be accessed by two ways

1. By value :- In this method, the value of enum member is passed.

2. By name :- In this method, the name of enum member is passed.

Seperate value or name can also be accessed using “**name**” or “**value**” keyword.

Comparison : Enumerations supports two types of comparisons

1. Identity :- These are checked using keywords “**is**” and “**is not**“.

2. Equality :- Equality comparisons of “**==**” and “**!=**” types are also supported.

```
# Python code to demonstrate enumerations
# Access and comparison

# importing enum for enumerations
import enum

# creating enumerations using class
class Animal(enum.Enum):
    dog = 1
    cat = 2
    lion = 3

# Accessing enum member using value
print ("The enum member associated with value 2 is : ",end="")
print (Animal(2))

# Accessing enum member using name
print ("The enum member associated with name lion is : ",end="")
print (Animal['lion'])

# Assigning enum member
mem = Animal.dog

# Displaying value
print ("The value associated with dog is : ",end="")
print (mem.value)

# Displaying name
print ("The name associated with dog is : ",end="")
print (mem.name)

# Comparison using "is"
if Animal.dog is Animal.cat:
       print ("Dog and cat are same animals")
else : print ("Dog and cat are different animals")

# Comparison using "!="
if Animal.lion != Animal.cat:
       print ("Lions and cat are different")
else : print ("Lions and cat are same")
```

Output :

```
The enum member associated with value 2 is : Animal.cat
The enum member associated with name lion is : Animal.lion
The value associated with dog is : 1
The name associated with dog is : dog
Dog and cat are different animals
Lions and cat are different
```

This article is contributed by **Manjeet Singh**. 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

Statistical Functions in Python | Set 1 (Averages and Measure of Central Location)

Python has the ability to manipulate some statistical data and calculate results of various statistical operations using the file “**statistics**“, useful in domain of mathematics.

Important Average and measure of central location functions :

1. mean() :- This function returns the **mean or average** of the data passed in its arguments. If passed argument is empty, **StatisticsError** is raised.

2. mode() :- This function returns the the **number with maximum number of occurrences**. If passed argument is empty, **StatisticsError** is raised.

```
# Python code to demonstrate the working of
# mean() and mode()

# importing statistics to handle statistical operations
import statistics

# initializing list
li = [1, 2, 3, 3, 2, 2, 2, 1]

# using mean() to calculate average of list elements
print ("The average of list values is : ",end="")
print (statistics.mean(li))

# using mode() to print maximum occurring of list elements
print ("The maximum occurring element is  : ",end="")
print (statistics.mode(li))
```

Output:

```
The average of list values is : 2.0
The maximum occurring element is  : 2
```

3. median() :- This function is used to calculate the median, i.e **middle element of data.** If passed argument is empty, **StatisticsError** is raised.

4. median_low() :- This function returns the median of data in case of odd number of elements, but in case of even number of elements, returns the **lower of two middle** elements. If passed argument is empty, **StatisticsError** is raised.

5. median_high() :- This function returns the median of data in case of odd number of elements, but in case of even number of elements, **returns the higher of two middle** elements. If passed argument is empty, **StatisticsError** is raised.

```
# Python code to demonstrate the working of
# median(), median_low() and median_high()

# importing statistics to handle statistical operations
import statistics

# initializing list
li = [1, 2, 2, 3, 3, 3]

# using median() to print median of list elements
print ("The median of list element is : ",end="")
print (statistics.median(li))

# using median_low() to print low median of list elements
print ("The lower median of list element is : ",end="")
print (statistics.median_low(li))

# using median_high() to print high median of list elements
print ("The higher median of list element is : ",end="")
print (statistics.median_high(li))
```

Output:

```
The median of list element is : 2.5
The lower median of list element is : 2
The higher median of list element is : 3
```

6. median_grouped() :- This function is used to compute group median, i.e **50th percentile** of the data. If passed argument is empty, **StatisticsError** is raised.

```
# Python code to demonstrate the working of
# median_grouped()

# importing statistics to handle statistical operations
import statistics

# initializing list
li = [1, 2, 2, 3, 3, 3]

# using median_grouped() to calculate 50th percentile
print ("The 50th percentile of data is : ",end="")
print (statistics.median_grouped(li))
```

Output:

```
The 50th percentile of data is : 2.5
```

Statistical Functions in Python | Set 2 (Measure of Spread)

This article is contributed by **Manjeet Singh**. 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

Statistical Functions in Python | Set 2 (Measure of Spread)

Statistical Functions in Python | Set 1(Averages and Measure of Central Location)

Measure of spread functions of statistics are discussed in this article.

1. variance() :- This function calculates the variance i.e **measure of deviation of data, more the value of variance, more the data values are spread**. Sample variance is computed in this function, assuming data is of a part of population. If passed argument is empty, **StatisticsError** is raised.

2. pvariance() :- This function computes the **variance of the entire population**. The data is interpreted as it is of the whole population. If passed argument is empty, **StatisticsError** is raised.

```
# Python code to demonstrate the working of
# variance() and pvariance()

# importing statistics to handle statistical operations
import statistics

# initializing list
li = [1.5, 2.5, 2.5, 3.5, 3.5, 3.5]

# using variance to calculate variance of data
print ("The variance of data is : ",end="")
print (statistics.variance(li))

# using pvariance to calculate population variance of data
print ("The population variance of data is : ",end="")
print (statistics.pvariance(li))
```

Output:

```
The variance of data is : 0.6666666666666667
The population variance of data is : 0.5555555555555556
```

3. stdev() :- This function returns the **standard deviation (square root of sample variance)** of the data. If passed argument is empty, **StatisticsError** is raised.

4. pstdev() :- This function returns the population **standard deviation (square root of population variance)** of the data. If passed argument is empty, **StatisticsError** is raised.

```
# Python code to demonstrate the working of
# stdev() and pstdev()

# importing statistics to handle statistical operations
import statistics

# initializing list
li = [1.5, 2.5, 2.5, 3.5, 3.5, 3.5]

# using stdev to calculate standard deviation of data
print ("The standard deviation of data is : ",end="")
print (statistics.stdev(li))

# using pstdev to calculate population standard deviation of data
print ("The population standard deviation of data is : ",end="")
print (statistics.pstdev(li))
```

Output:

```
The standard deviation of data is : 0.816496580927726
The population standard deviation of data is : 0.7453559924999299
```

This article is contributed by **Manjeet Singh**. 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

Bisect Algorithm Functions in Python

The purpose of Bisect algorithm is to find a position in list where an element needs to be inserted to keep the list sorted.

Python in its definition provides the bisect algorithms using the module “**bisect**” which allows to keep the list in sorted order after insertion of each element. This is essential as this reduces overhead time required to sort the list again and again after insertion of each element.

Important Bisection Functions

1. bisect(list, num, beg, end) :- This function returns the **position** in the **sorted** list, where the number passed in argument can be placed so as to **maintain the resultant list in sorted order**. If the element is already present in the list, the **right most position** where element has to be inserted is returned. **This function takes 4 arguments, list which has to be worked with, number to insert, starting position in list to consider, ending position which has to be considered**.

2. bisect_left(list, num, beg, end) :- This function returns the **position** in the **sorted** list, where the number passed in argument can be placed so as to **maintain the resultant list in sorted order**. If the element is already present in the list, the **left most position** where element has to be inserted is returned. **This function takes 4 arguments, list which has to be worked with, number to insert, starting position in list to consider, ending position which has to be considered**.

3. bisect_right(list, num, beg, end) :- This function works similar to the "**bisect()**" and mentioned above.

```
# Python code to demonstrate the working of
# bisect(), bisect_left() and bisect_right()

# importing "bisect" for bisection operations
import bisect

# initializing list
li = [1, 3, 4, 4, 4, 6, 7]

# using bisect() to find index to insert new element
# returns 5 ( right most possible index )
print ("The rightmost index to insert, so list remains sorted is  : ", end="")
print (bisect.bisect(li, 4))

# using bisect_left() to find index to insert new element
# returns 2 ( left most possible index )
print ("The leftmost index to insert, so list remains sorted is  : ", end="")
print (bisect.bisect_left(li, 4))

# using bisect_right() to find index to insert new element
# returns 4 ( right most possible index )
print ("The rightmost index to insert, so list remains sorted is  : ", end="")
print (bisect.bisect_right(li, 4, 0, 4))
```

Output:

```
The rightmost index to insert, so list remains sorted is  : 5
The leftmost index to insert, so list remains sorted is  : 2
The rightmost index to insert, so list remains sorted is  : 4
```

4. insort(list, num, beg, end) :- This function returns the **sorted list after inserting number in appropriate position**, if the element is already present in the list, the element is inserted at the **rightmost possible position. This function takes 4 arguments, list which has to be worked with, number to insert, starting position in list to consider, ending position which has to be considered**.

5. insort_left(list, num, beg, end) :- This function returns the **sorted list after inserting number in appropriate position**, if the element is already present in the list, the element is inserted at the **leftmost possible position. This function takes 4 arguments, list which has to be worked with, number to insert, starting position in list to consider, ending position which has to be considered**.

6. insort_right(list, num, beg, end) :- This function works similar to the "insort()" as mentioned above.

```
# Python code to demonstrate the working of
# insort(), insort_left() and insort_right()

# importing "bisect" for bisection operations
import bisect

# initializing list
li1 = [1, 3, 4, 4, 4, 6, 7]

# initializing list
li2 = [1, 3, 4, 4, 4, 6, 7]

# initializing list
li3 = [1, 3, 4, 4, 4, 6, 7]

# using insort() to insert 5 at appropriate position
# inserts at 6th position
bisect.insort(li1, 5)

print ("The list after inserting new element using insort() is : ")
for i in range(0, 7):
    print(li1[i], end=" ")

# using insort_left() to insert 5 at appropriate position
# inserts at 6th position
bisect.insort_left(li2, 5)

print("\r")

print ("The list after inserting new element using insort_left() is : ")
for i in range(0, 7):
    print(li2[i], end=" ")

print("\r")
```

```
# using insort_right() to insert 5 at appropriate position
# inserts at 5th position
bisect.insort_right(li3, 5, 0, 4)

print ("The list after inserting new element using insort_right() is : ")
for i in range(0, 7):
    print(li3[i], end=" ")
```

Output:

```
The list after inserting new element using insort() is :
1 3 4 4 4 5 6
The list after inserting new element using insort_left() is :
1 3 4 4 4 5 6
The list after inserting new element using insort_right() is :
1 3 4 4 5 4 6
```

This article is contributed by **Manjeet Singh**. 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

Decimal Functions in Python | Set 1

Python in its definition provides certain methods to perform faster decimal floating point arithmetic using the module “decimal”.

Important operations on Decimals

1. sqrt() :- This function computes the **square root** of the decimal number.

2. exp() :- This function returns the **e^x (exponent)** of the decimal number.

```
# Python code to demonstrate the working of
# sqrt() and exp()

# importing "decimal" module to use decimal functions
import decimal

# using exp() to compute the exponent of decimal number
a = decimal.Decimal(4.5).exp()

# using sqrt() to compute the square root of decimal number
b = decimal.Decimal(4.5).sqrt()

# printing the exponent
print ("The exponent of decimal number is : ",end="")
print (a)

# printing the square root
print ("The square root of decimal number is : ",end="")
print (b)
```

Output:

```
The exponent of decimal number is : 90.01713130052181355011545675
The square root of decimal number is : 2.121320343559642573202533086
```

3. ln() :- This function is used to compute **natural logarithm** of the decimal number.

4. log10() :- This function is used to compute **log(base 10)** of a decimal number.

```
# Python code to demonstrate the working of
# ln() and log10()

# importing "decimal" module to use decimal functions
import decimal

# using ln() to compute the natural log of decimal number
a = decimal.Decimal(4.5).ln()

# using sqrt() to compute the log10 of decimal number
b = decimal.Decimal(4.5).log10()

# printing the natural logarithm
print ("The natural logarithm of decimal number is : ",end="")
print (a)

# printing the log10
print ("The log(base 10) of decimal number is : ",end="")
print (b)
```

Output:

```
The natural logarithm of decimal number is : 1.504077396776274073373258352
The log(base 10) of decimal number is : 0.6532125137753436793763169118
```

5. as_tuple() :- Returns the decimal number as tuple containing **3 arguments, sign(0 for +, 1 for -), digits and exponent value**.

6. fma(a,b) :- This “fma” stands for **fused multiply and add**. It computes **(num*a)+b** from the numbers in argument. **No rounding of (num*a)** takes place in this function.

Example :

```
decimal.Decimal(5).fma(2,3) --> (5*2)+3 = 13
```

```
# Python code to demonstrate the working of
# as_tuple() and fma()

# importing "decimal" module to use decimal functions
import decimal

# using as_tuple() to return decimal number as tuple
a = decimal.Decimal(-4.5).as_tuple()

# using fma() to compute fused multiply and addition
b = decimal.Decimal(5).fma(2,3)

# printing the tuple
print ("The tuple form of decimal number is : ",end="")
print (a)

# printing the fused multiple and addition
print ("The fused multiply and addition of decimal number is : ",end="")
print (b)
```

Output:

```
The tuple form of decimal number is : DecimalTuple(sign=1, digits=(4, 5), exponent=-1)
The fused multiply and addition of decimal number is : 13
```

7. compare() :- This function is used to compare decimal numbers. **Returns 1 if 1st Decimal argument is greater than 2nd, -1 if 1st Decimal argument is smaller than 2nd and 0 if both are equal.**

8. compare_total_mag() :- Compares the total magnitute of decimal numbers. **Returns 1 if 1st Decimal argument is greater than 2nd(ignoring sign), -1 if 1st Decimal argument is smaller than 2nd(ignoring sign) and 0 if both are equal(ignoring sign).**

```
# Python code to demonstrate the working of
# compare() and compare_total_mag()

# importing "decimal" module to use decimal functions
import decimal

# Initializing decimal number
a = decimal.Decimal(9.53)

# Initializing decimal number
b = decimal.Decimal(-9.56)

# comparing decimal numbers using compare()
print ("The result of comparison using compare() is : ",end="")
print (a.compare(b))

# comparing decimal numbers using compare_total_mag()
print ("The result of comparison using compare_total_mag() is : ",end="")
print (a.compare_total_mag(b))
```

Output:

```
The result of comparison using compare() is : 1
The result of comparison using compare_total_mag() is : -1
```

9. copy_abs() :- This function prints the **absolute** value of decimal argument.

10. copy_negate() :- This function prints the **negation** of decimal argument.

11. copy_sign() :- This function prints the **first argument by copying the sign from 2nd argument**.

```
# Python code to demonstrate the working of
# copy_abs(),copy_sign() and copy_negate()

# importing "decimal" module to use decimal functions
import decimal

# Initializing decimal number
a = decimal.Decimal(9.53)

# Initializing decimal number
b = decimal.Decimal(-9.56)

# printing absolute value using copy_abs()
print ("The absolute value using copy_abs() is : ",end="")
print (b.copy_abs())
```

```
# printing negated value using copy_negate()
print ("The negated value using copy_negate() is : ",end="")
print (b.copy_negate())

# printing sign effected value using copy_sign()
print ("The sign effected value using copy_sign() is : ",end="")
print (a.copy_sign(b))
```

Output:

```
The absolute value using copy_abs() is : 9.5600000000000004973799150320701301097869873046875
The negated value using copy_negate() is : 9.5600000000000004973799150320701301097869873046875
The sign effected value using copy_sign() is : -9.5299999999999993605115378159098327159881591796875
```

12. max() :- This function computes the **maximum** of two decimal numbers.

13. min() :- This function computes the **minimum** of two decimal numbers.

```
# Python code to demonstrate the working of
# min() and max()

# importing "decimal" module to use decimal functions
import decimal

# Initializing decimal number
a = decimal.Decimal(9.53)

# Initializing decimal number
b = decimal.Decimal(7.43)

# printing minimum of both values
print ("The minimum of two numbers is : ",end="")
print (a.min(b))

# printing maximum of both values
print ("The maximum of two numbers is : ",end="")
print (a.max(b))
```

Output:

```
The minimum of two numbers is : 7.429999999999999715782905696
The maximum of two numbers is : 9.529999999999999360511537816
```

This article is contributed by **Manjeet Singh**. 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

Decimal Functions in Python | Set 2 (logical_and(), normalize(), quantize(), rotate() …)

Some of the Decimal functions have been discussed in Set 1 below

Decimal Functions in Python | Set 1

More functions are discussed in this article.

1. logical_and() :- This function computes digit-wise **logical "and"** operation of the number. Digits can only have the values **0 or 1**.

2. logical_or() :- This function computes digit-wise **logical "or"** operation of the number. Digits can only have the values **0 or 1**.

3. logical_xor() :- This function computes digit-wise **logical "xor"** operation of the number. Digits can only have the values **0 or 1**.

4. logical_invert() :- This function computes digit-wise **logical "invert"** operation of the number. Digits can only have the values **0 or 1**.

```
# Python code to demonstrate the working of
# logical_and(), logical_or(), logical_xor()
# and logical_invert()

# importing "decimal" module to use decimal functions
import decimal

# Initializing decimal number
a = decimal.Decimal(1000)

# Initializing decimal number
b = decimal.Decimal(1110)

# printing logical_and of two numbers
print ("The logical_and() of two numbers is : ",end="")
print (a.logical_and(b))
```

```
# printing logical_or of two numbers
print ("The logical_or() of two numbers is : ",end="")
print (a.logical_or(b))

# printing exclusive or of two numbers
print ("The exclusive or of two numbers is : ",end="")
print (a.logical_xor(b))

# printing logical inversion of number
print ("The logical inversion of number is : ",end="")
print (a.logical_invert())
```

Output:

```
The logical_and() of two numbers is : 1000
The logical_or() of two numbers is : 1110
The exclusive or of two numbers is : 110
The logical inversion of number is : 1111111111111111111111110111
```

5. next_plus() :- This function returns the **smallest number** that can be represented, **larger than the given number.**

6. next_plus() :- This function returns the **largest number** that can be represented, **smaller than the given number.**

```
# Python code to demonstrate the working of
# next_plus() and next_plus()

# importing "decimal" module to use decimal functions
import decimal

# Initializing decimal number
a = decimal.Decimal(101.34)

# printing the actual decimal number
print ("The original number is : ",end="")
print (a)

# printing number after using next_plus()
print ("The smallest number larger than current number : ",end="")
print (a.next_plus())

# printing number after using next_minus()
print ("The largest number smaller than current number : ",end="")
print (a.next_minus())
```

Output:

```
The original number is : 101.340000000000003410605131648480892181396484375
The smallest number larger than current number : 101.3400000000000034106051317
The largest number smaller than current number : 101.3400000000000034106051316
```

7. next_toward() :- This function returns the **number nearest to the 1st argument in the direction of the second** argument. In case Both the numbers are equal, returns the **2nd number with the sign of first** number.

8. normalize() :- This function prints the number after **erasing all the rightmost trailing zeroes** in the number.

```
# Python code to demonstrate the working of
# next_toward() and normalize()

# importing "decimal" module to use decimal functions
import decimal

# Initializing decimal number
a = decimal.Decimal(101.34)

# Initializing decimal number
b = decimal.Decimal(-101.34)

# Initializing decimal number
c = decimal.Decimal(-58.68)

# Initializing decimal number
d = decimal.Decimal(14.010000000)

# printing the number using next_toward()
print ("The number closest to 1st number in direction of second number : ")
print (a.next_toward(c))

# printing the number using next_toward()
# when equal
print ("The second number with sign of first number is : ",end="")
print (b.next_toward(a))

# printing number after erasing rightmost trailing zeroes
print ("Number after erasing rightmost trailing zeroes : ",end="")
print (d.normalize())
```

Output:

```
The number closest to 1st number in direction of second number :
```

```
101.3400000000000034106051316
The second number with sign of first number is : -101.3400000000000034106051316
Number after erasing rightmost trailing zeroes : 14.01
```

9. quantize() :- This function returns the 1st argument with the number of **digits in decimal part(exponent) shortened** by the **number of digits** in **decimal part(exponent) of 2nd argument.**

10. same_quantum() :- This function **returns 0 if both the numbers have different exponent and 1 if both numbers have same exponent.**

```
# Python code to demonstrate the working of
# quantize() and same_quantum()

# importing "decimal" module to use decimal functions
import decimal

# Initializing decimal number
a = decimal.Decimal(20.76548)

# Initializing decimal number
b = decimal.Decimal(12.25)

# Initializing decimal number
c = decimal.Decimal(6.25)

# printing quantized first number
print ("The quantized first number is : ",end="")
print (a.quantize(b))

# checking if both number have same exponent
if (b.same_quantum(c)):
    print ("Both the numbers have same exponent")
else : print ("Both numbers have different exponent")
```

Output:

```
The quantized first number is : 20.77
Both the numbers have same exponent
```

11. rotate() :- This function **rotates** the first argument by the **amount mentioned in the second argument**. If the sign of second argument is **positive, rotation is towards left**, **else the rotation is towards right**. The sign of first argument is unchanged.

12. shift() :- This function **shifts** the first argument by the **amount mentioned in the second argument**. If the sign of second argument is **positive, shifting is towards left, else the shifting is towards right**. The sign of first argument is unchanged. Digit shifted are **replaced by 0**.

```
# Python code to demonstrate the working of
# rotate() and shift()

# importing "decimal" module to use decimal functions
import decimal

# Initializing decimal number
a = decimal.Decimal(2343509394029424234334563465)

# using rotate() to rotate the first argument
# rotates to right by 2 positions
print ("The rotated value is : ",end="")
print (a.rotate(-2))

# using shift() to shift the first argument
# rotates to left by 2 positions
print ("The shifted value is : ",end="")
print (a.shift(2))
```

Output:

```
The rotated value is : 6523435093940294242343345634
The shifted value is : 4350939402942423433456346500
```

13. remainder_near() :- Returns the value "**1st – (n*2nd)**" where **n is the integer value nearest to the result of 1st/2nd**. If 2 integers have **exactly similar proximity, even one is choosen.**

14. scaleb() :- This function **shifts the exponent** of 1st number by the **value of second argument**.

```
# Python code to demonstrate the working of
# remainder_near() and scaleb()

# importing "decimal" module to use decimal functions
import decimal

# Initializing decimal number
a = decimal.Decimal(23.765)

# Initializing decimal number
b = decimal.Decimal(12)

# Initializing decimal number
c = decimal.Decimal(8)
```

```
# using remainder_near to compute value
print ("The computed value using remainder_near() is : ",end="")
print (b.remainder_near(c))

# using scaleb() to shift exponont
print ("The value after shifting exponent : ",end="")
print (a.scaleb(2))
```

Output:

```
The computed value using remainder_near() is : -4
The value after shifting exponent : 2376.500000000000056843418861
```

This article is contributed by **Manjeet Singh**. 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

Python | Set 3 (Strings, Lists, Tuples, Iterations)

In the previous article, we read about basics of Python. Now, we continue with some more python concepts.

Strings in Python

A string is a sequence of characters. It can be declared in python by using double quotes. Strings are immutable, i.e., they cannot be changed.

```
# Assigning string to a variable
a = "This is a string"
print a
```

Lists in Python

Lists are one of the most powerful tools in python. They are just like the arrays declared in other languages. But the most powerful thing is that list need not be always homogenous. A single list can contain strings, integers, as well as objects. Lists can also be used for implementing stacks and queues. Lists are mutable, i.e., they can be altered once declared.

```
# Declaring a list
L = [1, "a" , "string" , 1+2]
print L
L.append(6)
print L
L.pop()
print L
print L[1]
```

The output is :

```
[1, 'a', 'string', 3]
[1, 'a', 'string', 3, 6]
[1, 'a', 'string', 3]
a
```

Tuples in Python

A tuple is a sequence of immutable Python objects. Tuples are just like lists with the exception that tuples cannot be changed once declared. Tuples are usually faster than lists.

```
tup = (1, "a", "string", 1+2)
print tup
print tup[1]
```

The output is :

```
(1, 'a', 'string', 3)
a
```

Iterations in Python

Iterations or looping can be performed in python by ‘for’ and ‘while’ loops. Apart from iterating upon a particular condition, we can also iterate on strings, lists, and tuples.

Example1: Iteration by while loop for a condition

```
i = 1
while (i < 10):
 i += 1
 print i,
```

The output is :

```
2 3 4 5 6 7 8 9 10
```

Example 2: Iteration by for loop on string

```
s = "Hello World"
for i in s :
 print i
```

The output is :

```
H
e
l
l
o

W
o
r
l
d
```

Example 3: Iteration by for loop on list

```
L = [1, 4, 5, 7, 8, 9]
for  i in L:
 print i,
```

The output is :

```
1 4 5 7 8 9
```

Example 4 : Iteration by for loop for range

```
for  i in range(0, 10):
    print i,
```

The output is :

```
0 1 2 3 4 5 6 7 8 9
```

- Next Article – Python: Dictionary and Keywords
- Quiz on Data Types in Python

This article has been contributed by **Nikhil Kumar Singh.** Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

GATE CS Notes (According to Official GATE 2017 Syllabus)

GATE CS Corner

See Placement Course for placement preparation, GATE Corner for GATE CS Preparation and Quiz Corner for all Quizzes on GeeksQuiz.
Category: Python Articles

Interesting facts about strings in Python | Set 1

1. Strings are Immutable

Once a string is defined, it cannot be changed.

```
# A python program to show that string
# cannot be changed

a = 'Geeks'
print a       # output is displayed
a[2] = 'E'
print a       # causes error
```

Output:

```
Traceback (most recent call last):
   line 3, in
    a[2] = 'E'
TypeError: 'str' object does not support item assignment
```

But below code works fine.

```
# A python program to show that a string
# can be appended to a string.

a = 'Geeks'
print a # output is displayed
a = a + 'for'
```

```
print a # works fine
```

Output:

```
Geeks
Geeksfor
```

In the second program, interpreter makes a copy of the original string and then work on it and modifies it. So the expression **a = a +'for'** doesn't change string but reassigns the variable **a** to the new string generated by the result and drops down the previous string.

2. Three ways to create strings:
Strings in python can be created using single quotes or double quotes or a triple quotes .

The single quotes and double quotes works same for the string creation. Example of single quote and double quote string. Now talking about triple quotes, these are used when we have to write a string in multiple lines and printing as it is without using any escape sequence.

```
# A python program to create strings in three
# different ways and concatenate them.

a = 'Geeks' # string with single quotes
b = "for"   # string with double quotes
c = '''Geeks
a portal
for
geeks'''    # string with triple quotes
print a
print b
print c

# Concatenation of strings created using
# different quotes
print a + b + c
```

Output:

```
Geeks
for
Geeks
a portal
for
geeks
GeeksforGeeks
a portal
for
geeks
```

How to print single quote or double quote on screen?
We can do that in the following two ways:

- First one is to use escape character to display the additional quote.
- The second way is by using mix quote, i.e., when we want to print single quote then using double quotes as delimiters and vice-versa.

Example-

```
print "Hi Mr Geek."

# use of escape sequence
print "He said, \"Welcome to GeeksforGeeks\""

print 'Hey so happy to be here'

# use of mix quotes
print 'Getting Geeky, "Loving it"'
```

Output:

```
Hi Mr Geek.
He said, "Welcome to GeeksforGeeks"
Hey so happy to be here
Getting Geeky, "Loving it"
```

This article is contributed by **Arpit Agarwal**. 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

Python
Python

Interesting facts about strings in Python | Set 2 (Slicing)

Interesting facts about strings in Python -Set 1

Like other programming languages, it's possible to access individual characters of a string by using array-like indexing syntax. In this we can access each and every element of string through their index number and the indexing starts from 0. Python does index out of bound checking.

So, we can obtain the required character using syntax, **string_name[index_position]**:

- The positive index_position denotes the element from the starting(0) and the negative index shows the index from the end(-1).

Example-

```
# A python program to illustrate slicing in strings

x = "Geeks at work"

# Prints 3rd character beginning from 0
print x[2]

# Prints 7th character
print x[6]

# Prints 3rd character from rear beginning from -1
print x[-3]

# Length of string is 10 so it is out of bound
print x[15]
```

Output:

```
Traceback (most recent call last):
  File "8a33ebbf716678c881331d75e0b85fe6.py", line 15, in
    print x[15]
IndexError: string index out of range
```

```
e
a
o
```

Slicing

To extract substring from the whole string then then we use the syntax like

string_name[beginning: end : step]

- beginning represents the starting index of string
- end denotes the end index of string which is not inclusive
- steps denotes the distance between the two words.

Note: We can also slice the string using beginning and only and ***steps are optional.***

Example-

```
# A python program to illustrate
# print substrings of a string
x = "Welcome to GeeksforGeeks"

# Prints substring from 2nd to 5th character
print x[2:5]

# Prints substring stepping up 2nd character
# from 4th to 10th character
print x[4:10:2]

# Prints 3rd character from rear from 3 to 5
print x[-5:-3]
```

Output:

```
lco
oet
Ge
```

This article is contributed by **Arpit Agarwal.** 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
Python

Python String Methods | Set 1 (find, rfind, startwith, endwith, islower, isupper,

lower, upper, swapcase & title)

Some of the string basics have been covered in the below articles

Strings Part-1

Strings Part-2

The important string methods will be discussed in this article

1. find("string", beg, end) :- This function is used to find the position of the substring within a string.It takes 3 arguments, **substring , starting index(by default 0) and ending index(by default string length)**.

- It returns "-1 " if string is not found in given range.
- It returns first occurrence of string if found.

2. rfind("string", beg, end) :- This function has the similar working as find(), but it returns the position of the **last occurrence** of string.

```
# Python code to demonstrate working of
# find() and rfind()
str = "geeksforgeeks is for geeks"
str2 = "geeks"

# using find() to find first occurrence of str2
# returns 8
print ("The first occurrence of str2 is at : ", end="")
print (str.find( str2, 4) )

# using rfind() to find last occurrence of str2
# returns 21
print ("The last occurrence of str2 is at : ", end="")
print ( str.rfind( str2, 4) )
```

Output:

```
The first occurrence of str2 is at : 8
The last occurrence of str2 is at : 21
```

3. startswith("string", beg, end) :- The purpose of this function is to return true if the function **begins with mentioned string(prefix)** else return false.

4. endswith("string", beg, end) :- The purpose of this function is to return true if the **function ends with mentioned string(suffix)** else return false.

```
# Python code to demonstrate working of
# startswith() and endswith()
str = "geeksforgeeks"
str1 = "geeks"

# using startswith() to find if str starts with str1
if    str.startswith(str1):
        print ("str begins with str1")
else :  print ("str does not begin with str1")

# using endswith() to find if str ends with str1
if str.startswith(str1):
       print ("str ends with str1")
else : print ("str does not end with str1")
```

Output:

```
str begins with str1
str ends with str1
```

5. islower("string") :- This function returns true if all the letters in the string are **lower cased,** otherwise false.

6. isupper("string") :- This function returns true if all the letters in the string are **upper cased**, otherwise false.

```
# Python code to demonstrate working of
# isupper() and islower()
str = "GeeksforGeeks"
str1 = "geeks"

# checking if all characters in str are upper cased
if str.isupper() :
       print ("All characters in str are upper cased")
else : print ("All characters in str are not upper cased")

# checking if all characters in str1 are lower cased
if str1.islower() :
       print ("All characters in str1 are lower cased")
else : print ("All characters in str1 are not lower cased")
```

Output:

```
All characters in str are not upper cased
All characters in str1 are lower cased
```

7. lower() :- This function returns the new string with all the letters **converted into its lower case**.

8. upper() :- This function returns the new string with all the letters **converted into its upper case**.

9. swapcase() :- This function is used to swap the cases of string i.e upper case is converted to lower case and vice versa.

10. title() :- This function converts the string to its **title case** i.e the first letter of every word of string is upper cased and else all are lower cased.

```
# Python code to demonstrate working of
# upper(), lower(), swapcase() and title()
str = "GeeksForGeeks is fOr GeeKs"

# Coverting string into its lower case
str1 = str.lower();
print (" The lower case converted string is : " + str1)

# Coverting string into its upper case
str2 = str.upper();
print (" The upper case converted string is : " + str2)

# Coverting string into its swapped case
str3 = str.swapcase();
print (" The swap case converted string is : " + str3)

# Coverting string into its title case
str4 = str.title();
print (" The title case converted string is : " + str4)
```

Output:

```
The lower case converted string is : geeksforgeeks is for geeks
The upper case converted string is : GEEKSFORGEEKS IS FOR GEEKS
The swap case converted string is : gEEKSfORgEEKS IS FoR gEEkS
The title case converted string is : Geeksforgeeks Is For Geeks
```

This article is contributed by **Manjeet Singh** .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

Python String Methods | Set 2 (len, count, center, ljust, rjust, isalpha, isalnum, isspace & join)

Some of the string methods are covered in the set 3 below

String Methods Part- 1

More methods are discussed in this article

1. len() :- This function returns the **length** of the string.

2. count("string", beg, end) :- This function **counts** the occurrence of mentioned **substring** in whole string. This function takes 3 arguments, s**ubstring, beginning position(by default 0) and end position(by default string length).**

```
# Python code to demonstrate working of
# len() and count()
str = "geeksforgeeks is for geeks"

# Printing length of string using len()
print (" The length of string is : ", len(str));

# Printing occurrence of "geeks" in string
# Prints 2 as it only checks till 15th element
print (" Number of appearance of ""geeks"" is : ",end="")
print (str.count("geeks",0,15))
```

Output:

```
The length of string is :  26
Number of appearance of geeks is : 2
```

3. centre() :- This function is used to **surround the string with a character** repeated both sides of string multiple times. By default the character is a space. Takes 2 arguments, **length of string and the character.**

4. ljust() :- This function returns the **original string shifted to left** that has a **character at its right**. It left adjusts the string. By default the character is space. It also takes two arguments, **length of string and the character.**

5. rjust() :- This function returns the **original string shifted to right** that has a **character at its left**. It right adjusts the string. By default the character is space. It also takes two arguments, **length of string and the character.**

```
# Python code to demonstrate working of
# center(), ljust() and rjust()
str = "geeksforgeeks"
```

```
# Printing the string after centering with '-'
print ("The string after centering with '-' is : ",end="")
print ( str.center(20,'-'))

# Printing the string after ljust()
print ("The string after ljust is : ",end="")
print ( str.ljust(20,'-'))

# Printing the string after rjust()
print ("The string after rjust is : ",end="")
print ( str.rjust(20,'-'))
```

Output:

```
The string after centering with '-' is : ---geeksforgeeks----
The string after ljust is : geeksforgeeks-------
The string after rjust is : -------geeksforgeeks
```

6. isalpha() :- This function returns true when all the characters in the string are **alphabets** else returns false.

7. isalnum() :- This function returns true when all the characters in the string are **numbers** else returns false.

8. isspace() :- This function returns true when all the characters in the string are **spaces** else returns false.

```
# Python code to demonstrate working of
# isalpha(), isalnum(), isspace()
str = "geeksforgeeks"
str1 = "123"

# Checking if str has all alphabets
if (str.isalpha()):
    print ("All characters are alphabets in str")
else : print ("All characters are not alphabets in str")

# Checking if str1 has all numbers
if (str1.isalnum()):
    print ("All characters are numbers in str1")
else : print ("All characters are not numbers in str1")

# Checking if str1 has all spaces
if (str1.isspace()):
    print ("All characters are spaces in str1")
else : print ("All characters are not spaces in str1")
```

Output:

```
All characters are alphabets in str
All characters are numbers in str1
All characters are not spaces in str1
```

9. join() :- This function is used to **join a sequence** of strings mentioned in its arguments with the string.

```
# Python code to demonstrate working of
# join()
str = "_"
str1 = ( "geeks", "for", "geeks" )

# using join() to join sequence str1 with str
print ("The string after joining is : ", end="")
print ( str.join(str1))
```

Output:

```
The string after joining is : geeks_for_geeks
```

This article is contributed by **Manjeet Singh** .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

Python String Methods | Set 3 (strip, lstrip, rstrip, min, max, maketrans, translate & relplace)

Some of the string methods are covered in below sets.

String Methods Part- 1
String Methods Part- 2

More methods are discussed in this article

1. strip() :- This method is used to **delete all the leading and trailing** characters mentioned in its argument.

2. lstrip() :- This method is used to **delete all the trailing** characters mentioned in its argument.

3. rstrip() :- This method is used to **delete all the leading** characters mentioned in its argument.

```
# Python code to demonstrate working of
# strip(), lstrip() and rstrip()
str = "---geeksforgeeks---"

# using strip() to delete all '-'
print ( " String after stripping all '-' is : ", end="")
print ( str.strip('-') )

# using lstrip() to delete all trailing '-'
print ( " String after stripping all trailing '-' is : ", end="")
print ( str.lstrip('-') )

# using rstrip() to delete all leading '-'
print ( " String after stripping all leading '-' is : ", end="")
print ( str.rstrip('-') )
```

Output:

```
 String after stripping all '-' is : geeksforgeeks
 String after stripping all trailing '-' is : geeksforgeeks---
 String after stripping all leading '-' is : ---geeksforgeeks
```

4. min("string") :- This function returns the **minimum value alphabet** from string.

5. max("string") :- This function returns the **maximum value alphabet** from string.

```
# Python code to demonstrate working of
# min() and max()
str = "geeksforgeeks"

# using min() to print the smallest character
# prints 'e'
print ("The minimum value character is : " + min(str));

# using max() to print the largest character
# prints 's'
print ("The maximum value character is : " + max(str));
```

Output:

```
The minimum value character is : e
The maximum value character is : s
```

6. maketrans() :- It is used to **map the contents of string 1 with string 2** with respective indices to be translated later using translate().

7. translate() :- This is used to **swap the string elements mapped** with the help of maketrans().

```
# Python code to demonstrate working of
# maketrans() and translate()
from string import maketrans # for maketrans()

str = "geeksforgeeks"

str1 = "gfo"
str2 = "abc"

# using maktrans() to map elements of str2 with str1
mapped = maketrans( str1, str2 );

# using translate() to translate using the mapping
print "The string after translation using mapped elements is : "
print  str.translate(mapped) ;
```

Output:

```
The string after translation using mapped elements is :
aeeksbcraeeks
```

In the above code, 'g' is replaced by a, 'f' is replaced by b and 'o' is replaced by 'c' in the string using translate function.

8. replace() :- This function is used to **replace the substring with a new substring** in the string. This function has 3 arguments. **The string to replace, new string which would replace and max value denoting the limit to replace action (by default unlimited).**

```
# Python code to demonstrate working of
# replace()

str = "nerdsfornerds is for nerds"

str1 = "nerds"
str2 = "geeks"

# using replace() to replace str2 with str1 in str
# only changes 2 occurrences
print ("The string after replacing strings is : ", end="")
print (str.replace( str1, str2, 2))
```

Output:

```
The string after replacing strings is : geeksforgeeks is for nerds
```

This article is contributed by **Manjeet Singh** .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

Logical Operators on String in Python

For strings in python, boolean operators (and , or) work. Let us consider the two strings namely str1 and str2 and try boolean operators on them:

```
str1 = ''
str2 = 'geeks'

# repr is used to print the string along with the quotes
print repr(str1 and str2) # Returns str1
print repr(str2 and str1) # Returns str1
print repr(str1 or str2) # Returns str2
print repr(str2 or str1) # Returns str2

str1 = 'for'

print repr(str1 and str2) # Returns str2
print repr(str2 and  str1) # Returns str1
print repr(str1 or str2) # Returns str1
print repr(str2 or str1) # Returns str2

# Coded by Nikhil Kumar Singh(nickzuck_007)
```

Output:

```
''
''
'geeks'
'geeks'
'geeks'
'for'
'for'
'geeks'
```

The output of the boolean operations between the strings depends on following things:

1. Python considers empty strings as having boolean value of 'false' and non-empty string as having boolean value of 'true'.
2. For 'and' operator if left value is true, then right value is checked and returned. If left value is false, then it is returned
3. For 'or' operator if left value is true, then it is returned, otherwise if right value is false, then right value is returned.

Note that the bitwise operators (| , &) don't work for strings.

This article is contributed by **Nikhil Kumar Singh**.

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
Python

How to split a string in C/C++, Python and Java?

Splitting a string by some delimiter is a very common task. For example, we have a comma separated list of items from a file and we want individual items in an array.

Almost all programming languages, provide function split a string by some delimiter.

In C/C++:

```
// Splits str[] according to given delimiters.
// and returns next token. It needs to be called
// in a loop to get all tokens. It returns NULL
// when there are no more tokens.
char * strtok(char str[], const char *delims);
```

```
// A C/C++ program for splitting a string
// using strtok()
#include <stdio.h>
#include <string.h>

int main()
{
    char str[] = "Geeks-for-Geeks";

    // Returns first token
    char *token = strtok(str, "-");

    // Keep printing tokens while one of the
    // delimiters present in str[].
    while (token != NULL)
    {
        printf("%s\n", token);
        token = strtok(NULL, "-");
    }

    return 0;
}
```

Output:

```
Geeks
for
Geeks
```

In Java :

In Java, split() is a method in String class.

```
// expregexp is the delimiting regular expression;
// limit is the number of returned strings
public String[] split(String regexp, int limit);

// We can call split() without limit also
public String[] split(String regexp)
```

```
// A Java program for splitting a string
// using split()
import java.io.*;
public class Test
{
    public static void main(String args[])
    {
        String Str = new String("Geeks-for-Geeks");

        // Split above string in at-most two strings
        for (String val: Str.split("-", 2))
            System.out.println(val);

        System.out.println("");

        // Splits Str into all possible tokens
        for (String val: Str.split("-"))
            System.out.println(val);
    }
}
```

Output:

```
Geeks
for-Geeks

Geeks
for
Geeks
```

In Python:

```
  // regexp is the delimiting regular expression;
  // limit is limit the number of splits to be made
  str.split(regexp = "", limit = string.count(str))
```

```
line = "Geek1 \nGeek2 \nGeek3";
print line.split()
print line.split(' ', 1)
```

Output:

```
['Geek1', 'Geek2', 'Geek3']
```

```
['Geek1', '\nGeek2 \nGeek3']
```

This article is contributed by **Aditya Chatterjee.** 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

C/C++ Puzzles
Java
Python
Strings
cpp-library
cpp-string

String Formatting in Python using %

In Python a string of required formatting can be achieved by different methods.
Some of them are ;
1) Using %
2) Using {}
3) Using Template Strings

In this article the formatting using % is discussed.

The formatting using % is similar to that of 'printf' in C programming language.
%d – integer
%f – float
%s – string
%x – hexadecimal
%o – octal

The below example describes the use of formatting using % in Python

```
# Python program to demonstrate the use of formatting using %

# Initialize variable as a string
variable = '15'
string = "Variable as string = %s" %(variable)
print string

# Printing as raw data
# Thanks to Himanshu Pant for this
print "Variable as raw data = %r" %(variable)

# Convert the variable to integer
# And perform check other formatting options

variable = int(variable) # Without this the below statement
    # will give error.
string = "Variable as integer = %d" %(variable)
print string
print "Variable as float = %f" %(variable)
print "Variable as hexadecimal = %x" %(variable)
print "Variable as octal = %o" %(variable)
```

Output :

```
Variable as string = 15
Variable as raw data = '15'
Variable as integer = 15
Variable as float = 15.000000
Variable as hexadecimal = f
Variable as octal = 17
```

This article is contributed by **Nikhil Kumar Singh (nickzuck_007)**

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

GATE CS Notes (According to Official GATE 2017 Syllabus)

GATE CS Corner

See Placement Course for placement preparation, GATE Corner for GATE CS Preparation and Quiz Corner for all Quizzes on GeeksQuiz.
Category: Python Articles

String Template Class in Python

In String module, Template Class allows us to create simplified syntax for output specification. The format uses placeholder names formed by $ with valid Python identifiers (alphanumeric characters and underscores). Surrounding the placeholder with braces allows it to be followed by more alphanumeric letters with no intervening spaces. Writing $$ creates a single escaped $:

Below is a simple example.

```
# A Simple Python templaye example
from string import Template

# Create a template that has placeholder for value of x
t = Template('x is $x')

# Substitute value of x in above template
print (t.substitute({'x' : 1}))
```

Output:

```
x is 1
```

Following is another example where we import names and marks of students from a list and print them using template.

```
# A Python program to demonstrate working of string template
from string import Template

# List Student stores the name and marks of three students
Student = [('Ram',90), ('Ankit',78), ('Bob',92)]

# We are creating a basic structure to print the name and
# marks of the students.
t = Template('Hi $name, you have got $marks marks')

for i in Student:
     print (t.substitute(name = i[0], marks = i[1]))
```

Output:

```
Hi Ram, you have got 90 marks
Hi Ankit, you have got 78 marks
Hi Bob, you have got 92 marks
```

The substitute() method raises a KeyError when a placeholder is not supplied in a dictionary or a keyword argument. For mail-merge style applications, user supplied data may be incomplete and the safe_substitute() method may be more appropriate — it will leave placeholders unchanged if data is missing:

Another application for template is separating program logic from the details of multiple output formats. This makes it possible to substitute custom templates for XML files, plain text reports, and HTML web reports.

Note that there are other ways also to print formatted output like %d for integer, %f for float, etc (Refer this for details)

Reference: https://docs.python.org/3.3/tutorial/stdlib2.html

This article is contributed by **Siddharth Lalwani.** 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

Regular Expressions in Python | Set 2 (Search, Match and Find All)

Regular Expression in Python with Examples | Set 1

The module **re** provides support for regular expressions in Python. Below are main methods in this module.

Searching an occurrence of pattern

re.search() : This method either returns None (if the pattern doesn't match), or a re.MatchObject that contains information about the matching part of the string. This method stops after the first match, so this is best suited for testing a regular expression more than extracting data.

```
# A Python program to demonstrate working of re.match().
import re

# Lets use a regular expression to match a date string
# in the form of Month name followed by day number
regex = r"([a-zA-Z]+) (\d+)"

match = re.search(regex, "I was born on June 24")

if match != None:

    # We reach here when the expression "([a-zA-Z]+) (\d+)"
    # matches the date string.

    # This will print [14, 21), since it matches at index 14
    # and ends at 21.
    print "Match at index %s, %s" % (match.start(), match.end())
    # We us group() method to get all the matches and
    # captured groups. The groups contain the matched values.
    # In particular:
    #    match.group(0) always returns the fully matched string
    #    match.group(1) match.group(2), ... return the capture
    #    groups in order from left to right in the input string
    #    match.group() is equivalent to match.group(0)

    # So this will print "June 24"
    print "Full match: %s" % (match.group(0))

    # So this will print "June"
    print "Month: %s" % (match.group(1))

    # So this will print "24"
    print "Day: %s" % (match.group(2))

else:
    print "The regex pattern does not match."
```

Output :

```
Match at index 14, 21
Full match: June 24
Month: June
Day: 24
```

Matching a Pattern with Text

re.match() : This function attempts to match pattern to whole string. The re.match function returns a match object on success, None on failure.

```
re.match(pattern, string, flags=0)

pattern : Regular expression to be matched.
string : String where p attern is searched
flags : We can specify different flags
        using bitwise OR (|).
```

```
# A Python program to demonstrate working
# of re.match().
import re

# a sample function that uses regular expressions
# to find month and day of a date.
def findMonthAndDate(string):

    regex = r"([a-zA-Z]+) (\d+)"
    match = re.match(regex, string)

    if match == None:
        print "Not a valid date"
        return

    print "Given Data: %s" % (match.group())
    print "Month: %s" % (match.group(1))
    print "Day: %s" % (match.group(2))


# Driver Code
findMonthAndDate("Jun 24")
print("")
findMonthAndDate("I was born on June 24")
```

Finding all occurrences of a pattern

re.findall() : Return all non-overlapping matches of pattern in string, as a list of strings. The string is scanned left-to-right, and matches are returned in the order found (Source : Python Docs).

```
# A Python program to demonstrate working of
# findall()
import re

# A sample text string where regular expression
# is searched.
string  = """Hello my Number is 123456789 and
             my friend's number is 987654321"""

# A sample regular expression to find digits.
regex = '\d+'

match = re.findall(regex, string)
print(match)

# This example is contributed by Ayush Saluja.
```

Output :

```
['123456789', '987654321']
```

Regular expression is a vast topic. It's a complete library. Regular expressions can do a lot of stuff. You can Match, Search, Replace, Extract a lot of data. For example, below small code is so powerful that it can extract email address from a text. So we can make our own Web Crawlers and scrappers in python with easy.Look at below regex.

```
# extract all email addresses and add them into the resulting set
new_emails = set(re.findall(r"[a-z0-9\.\-+_]+@[a-z0-9\.\-+_]+\.[a-z]+",
                    text, re.I))
```

We will soon be discussing more methods on regular expressions.

This article is contributed by **Shwetanshu Rohatgi**. 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

Python List Comprehension and Slicing

List comprehension is an elegant way to define and create list in python. We can create lists just like mathematical statements and in one line only. The syntax of list comprehension is easier to grasp.

```
A list comprehension generally consist of these parts :
   Output expression,
   input sequence,
   a variable representing member of input sequence and
   an optional predicate part.

For example :

lst  =  [x ** 2  for x in range (1, 11)   if  x % 2 == 1]

here, x ** 2 is output expression,
      range (1, 11)  is input sequence,
      x is variable and
      if x % 2 == 1 is predicate part.
```

```
# Python program to demonstrate list comprehension in Python

# below list contains square of all odd numbers from
# range 1 to 10
odd_square = [x ** 2 for x in range(1, 11) if x % 2 == 1]
print odd_square

# for understanding, above generation is same as,
odd_square = []
for x in range(1, 11):
 if x % 2 == 1:
  odd_square.append(x**2)
print odd_square

# below list contains power of 2 from 1 to 8
power_of_2 = [2 ** x for x in range(1, 9)]
print power_of_2

# below list contains prime and non-prime in range 1 to 50
noprimes = [j for i in range(2, 8) for j in range(i*2, 50, i)]
primes = [x for x in range(2, 50) if x not in noprimes]
print primes

# list for lowering the characters
print [x.lower() for x in ["A","B","C"]]

# list which extracts number
string = "my phone number is : 11122 !!"

print("\nExtracted digits")
numbers = [x for x in string if x.isdigit()]
print numbers

# A list of list for multiplication table
a = 5
table = [[a, b, a * b] for b in range(1, 11)]

print("\nMultiplication Table")
for i in table:
 print i
```

Output:

```
[1, 9, 25, 49, 81]
[1, 9, 25, 49, 81]
[2, 4, 8, 16, 32, 64, 128, 256]
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47]
```

```
['a', 'b', 'c']

Extracted digits
['1', '1', '1', '2', '2']

Multiplication Table
[5, 1, 5]
[5, 2, 10]
[5, 3, 15]
[5, 4, 20]
[5, 5, 25]
[5, 6, 30]
[5, 7, 35]
[5, 8, 40]
[5, 9, 45]
[5, 10, 50]
```

After getting the list, we can get a part of it using python's slicing operator which has following syntax :

```
   [start : stop : steps]

which means that slicing will start from index start
 will go up to stop in step of steps.
 Default value of start is 0, stop is last index of list
 and for step it is 1
```

So **[: stop]** will slice list from starting till stop index and **[start :]** will slice list from start index till end Negative value of steps shows right to left traversal instead of left to right traversal that is why **[: : -1]** prints list in reverse order.

```
# Let us first create a list to demonstrate slicing
# lst contains all number from 1 to 10
lst = range(1, 11)
print lst

#  below list has numbers from 2 to 5
lst1_5 = lst[1 : 5]
print lst1_5

#  below list has numbers from 6 to 8
lst5_8 = lst[5 : 8]
print lst5_8

#  below list has numbers from 2 to 10
lst1_ = lst[1 : ]
print lst1_

#  below list has numbers from 1 to 5
lst_5 = lst[: 5]
print lst_5

#  below list has numbers from 2 to 8 in step 2
lst1_8_2 = lst[1 : 8 : 2]
print lst1_8_2

#  below list has numbers from 10 to 1
lst_rev = lst[ : : -1]
print lst_rev

#  below list has numbers from 10 to 6 in step 2
lst_rev_9_5_2 = lst[9 : 4 : -2]
print lst_rev_9_5_2
```

Output:

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
[2, 3, 4, 5]
[6, 7, 8]
[2, 3, 4, 5, 6, 7, 8, 9, 10]
[1, 2, 3, 4, 5]
[2, 4, 6, 8]
[10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
[10, 8, 6]
```

We can use **filter** function to filter a list based on some condition provided as a **lambda expression** as first argument and list as second argument, example of which is shown below :

```
#  filtering odd numbers
lst = filter(lambda x : x % 2 == 1, range(1, 20))
print lst

#  filtering odd square which are divisble by 5
lst = filter(lambda x : x % 5 == 0,
      [x ** 2 for x in range(1, 11) if x % 2 == 1])
print lst

#   filtering negative numbers
```

```
lst = filter((lambda x: x < 0), range(-5,5))
print lst

#  implementing max() function, using
print reduce(lambda a,b: a if (a > b) else b, [7, 12, 45, 100, 15])
```

Output:

```
[1, 3, 5, 7, 9, 11, 13, 15, 17, 19]
[25]
[-5, -4, -3, -2, -1]
100
```

This article is contributed by **Utkarsh Trivedi**. 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
Python

List Methods in Python | Set 1 (in, not in, len(), min(), max()…)

List basics have been covered in python in the set below

Data Types-List, Tuples and Iterations

List methods are discussed in this articles.

1. "in" operator :- This operator is used to **check if an element is present** in the list or not. Returns true if element is present in list else returns false.

2. "not in" operator :- This operator is used to **check if an element is not present** in the list or not. Returns true if element is not present in list else returns false.

```
# Python code to demonstrate the working of
# "in" and "not in"
# initializing list
lis = [1, 4, 3, 2, 5]

# checking if 4 is in list using "in"
if 4 in lis:
        print ("List is having element with value 4")
else :  print ("List is not having element with value 4")

# checking if 4 is not list using "not in"
if 4 not in lis:
        print ("List is not having element with value 4")
else :  print ("List is having element with value 4")
```

Output:

```
List is having element with value 4
List is having element with value 4
```

3. len() :- This function returns the **length** of list.

4. min() :- This function returns the **minimum** element of list.

5. max() :- This function returns the **maximum** element of list.

```
# Python code to demonstrate the working of
# len(), min() and max()
# initializing list 1
lis = [2, 1, 3, 5, 4]

# using len() to print length of list
print ("The length of list is : ", end="")
print (len(lis))

# using min() to print minimum element of list
print ("The minimum element of list is : ", end="")
print (min(lis))

# using max() to print maximum element of list
print ("The maximum element of list is : ", end="")
print (max(lis))
```

Output:

```
The length of list is : 5
The minimum element of list is : 1
The maximum element of list is : 5
```

6. "+" operator :- This operator is used to **concatenate** two lists into a single list.

7. "*" operator :- This operator is used to **multiply the list "n" times** and return the single list.

```
# Python code to demonstrate the working of
# "+" and "*"
# initializing list 1
lis = [1, 2, 3]

# initializing list 2
lis1 = [4, 5, 6]

# using "+" to concatenate lists
lis2= lis + lis1

# priting concatenated lists
print ("list after concatenation is : ", end="")
for i in range(0,len(lis2)):
        print (lis2[i], end=" ")

print ("\r")

#using '*' to combine lists
lis3 = lis * 3

# priting combined lists
print ("list after combining is : ", end="")
for i in range(0,len(lis3)):
        print (lis3[i], end=" ")
```

Output:

```
list after concatenation is : 1 2 3 4 5 6
list after combining is : 1 2 3 1 2 3 1 2 3
```

8. index(ele, beg, end) :- This function returns the **index of first occurrence of element** after beg and before end.

9. count() :- This function counts the **number of occurrences** of elements in list.

```
# Python code to demonstrate the working of
# index() and count()
# initializing list 1
lis = [2, 1, 3, 5, 4, 3]

# using index() to print first occurrence of 3
# prints 5
print ("The first occurrence of 3 after 3rd position is : ", end="")
print (lis.index(3, 3, 6))

# using count() to count number of occurrence of 3
print ("The number of occurrences of 3 is : ", end="")
print (lis.count(3))
```

Output:

```
The first occurrence of 3 after 3rd position is : 5
The number of occurrences of 3 is : 2
```

Related articles:

List methods in Python

List Methods in Python | Set 2 (del, remove(), sort(), insert(), pop(), extend()…)

This article is contributed by **Manjeet Singh** .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

List Methods in Python | Set 2 (del, remove(), sort(), insert(), pop(), extend()…)

Some of the list methods are mentioned in set 1 below

List Methods in Python | Set 1 (in, not in, len(), min(), max()…)

More methods are discussed in this article.

1. del[a : b] :- This method **deletes all the elements in range** starting from index 'a' till 'b' mentioned in arguments.

2. pop() :- This method **deletes** the element **at the position** mentioned in its arguments.

```
# Python code to demonstrate the working of
# del and pop()

# initializing list
lis = [2, 1, 3, 5, 4, 3, 8]

# using del to delete elements from pos. 2 to 5
# deletes 3,5,4
del lis[2 : 5]
```

```
# displaying list after deleting
print ("List elements after deleting are : ",end="")
for i in range(0, len(lis)):
    print(lis[i], end=" ")

print("\r")

# using pop() to delete element at pos 2
# deletes 3
lis.pop(2)

# displaying list after popping
print ("List elements after popping are : ", end="")
for i in range(0, len(lis)):
    print(lis[i], end=" ")
```

Output:

```
List elements after deleting are : 2 1 3 8
List elements after popping are : 2 1 8
```

3. insert(a, x) :- This function **inserts an element at the position** mentioned in its arguments. It takes 2 arguments, **position and element to be added at respective position.**

4. remove() :- This function is used to **delete the first occurrence** of number mentioned in its arguments.

```
# Python code to demonstrate the working of
# insert() and remove()

# initializing list
lis = [2, 1, 3, 5, 3, 8]

# using insert() to insert 4 at 3rd pos
lis.insert(3, 4)

# displaying list after inserting
print("List elements after inserting 4 are : ", end="")
for i in range(0, len(lis)):
    print(lis[i], end=" ")

print("\r")

# using remove() to remove first occurrence of 3
# removes 3 at pos 2
lis.remove(3)

# displaying list after removing
print ("List elements after removing are : ", end="")
for i in range(0, len(lis)):
    print(lis[i], end=" ")
```

Output:

```
List elements after inserting 4 are : 2 1 3 4 5 3 8
List elements after removing are : 2 1 4 5 3 8
```

5. sort() :- This function **sorts** the list in **increasing order**.

6. reverse() :- This function **reverses** the elements of list.

```
# Python code to demonstrate the working of
# sort() and reverse()

# initializing list
lis = [2, 1, 3, 5, 3, 8]

# using sort() to sort the list
lis.sort()

# displaying list after sorting
print ("List elements after sorting are : ", end="")
for i in range(0, len(lis)):
    print(lis[i], end=" ")

print("\r")

# using reverse() to reverse the list
lis.reverse()

# displaying list after reversing
print ("List elements after reversing are : ", end="")
for i in range(0, len(lis)):
    print(lis[i], end=" ")
```

Output:

```
List elements after sorting are : 1 2 3 3 5 8
List elements after reversing are : 8 5 3 3 2 1
```

7. extend(b) :- This function is used to **extend the list with** the elements present in **another list**. This function takes **another list as its argument**.

8. clear() :- This function is used to **erase all the elements** of list. After this operation, list becomes empty.

```
# Python code to demonstrate the working of
# extend() and clear()

# initializing list 1
lis1 = [2, 1, 3, 5]

# initializing list 1
lis2 = [6, 4, 3]

# using extend() to add elements of lis2 in lis1
lis1.extend(lis2)

# displaying list after sorting
print ("List elements after extending are : ", end="")
for i in range(0, len(lis1)):
    print(lis1[i], end=" ")

print ("\r")

# using clear() to delete all lis1 contents
lis1.clear()

# displaying list after clearing
print ("List elements after clearing are : ", end="")
for i in range(0, len(lis1)):
    print(lis1[i], end=" ")
```

Output:

```
List elements after extending are : 2 1 3 5 6 4 3
List elements after clearing are :
```

Related articles:
List methods in Python
List Methods in Python | Set 1 (in, not in, len(), min(), max()…)

This article is contributed by **Manjeet Singh** .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.

List methods in Python

This article is extension of below articles :
List Methods in Python | Set 1 (in, not in, len(), min(), max()…)
List Methods in Python | Set 2 (del, remove(), sort(), insert(), pop(), extend()…)

Adding and Appending

- **append():** Used for appending and adding elements to List.It is used to add elements to the last position of List.
  Syntax:

  ```
  list.append (element)
  ```

  ```
  # Adds List Element as value of List.
  List = ['Mathematics', 'chemistry', 1997, 2000]
  List.append(20544)
  print(List)
  ```

  Output:

  ```
  ['Mathematics', 'chemistry', 1997, 2000, 20544]
  ```

- **insert():** Inserts an elements at specified position.
  Syntax:

  ```
  list.insert(<position, element)
  ```

  Note: Position mentioned should be within the range of List, as in this case between 0 and 4, elsewise would throw IndexError.

  ```
  List = ['Mathematics', 'chemistry', 1997, 2000]
  # Insert at index 2 value 10087
  List.insert(2,10087)
  print(List)
  ```

Output:

```
['Mathematics', 'chemistry', 10087, 1997, 2000, 20544]
```

- **extend():** Adds contents to List2 to the end of List1.
  Syntax:

```
List1.extend(List2)
```

```
List1 = [1, 2, 3]
List2 = [2, 3, 4, 5]

# Add List2 to List1
List1.extend(List2)
print(List1)

#Add List1 to List2 now
List2.extend(List1)
print(List2)
```

Output:

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

sum(), count(), index(), min() and max() functions of List

- **sum() :** Calculates sum of all the elements of List.
  Syntax:

```
 sum(List)
```

```
List = [1, 2, 3, 4, 5]
print(sum(List))
```

Output:

```
15
```

What happens if numeric value is not used a parameter?

Sum is calculated only for Numeric values, elsewise throws TypeError.

See example:

```
List = ['gfg', 'abc', 3]
print(sum(List))
```

Output:

```
Traceback (most recent call last):
  File "", line 1, in
    sum(List)
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

- **count():**Calculates total occurrence of given element of List.
  Syntax:

```
List.count(element)
```

```
List = [1, 2, 3, 1, 2, 1, 2, 3, 2, 1]
print(List.count(1))
```

Output:

```
4
```

- **index():** Returns the index of first occurrence. Start and End index are not necessary parameters.
  Syntax:

```
List.index(element[,start[,end]])
```

```
List = [1, 2, 3, 1, 2, 1, 2, 3, 2, 1]
print(List.index(2))
```

Output:

```
1
```

Another example:

```
List = [1, 2, 3, 1, 2, 1, 2, 3, 2, 1]
print(List.index(2,2))
```

Output:

```
4
```

```
List = [1, 2, 3, 1, 2, 1, 2, 3, 2, 1]

# will check from index 2 to 3.
print(List.index(2,2,4))
```

Output:

```
Traceback (most recent call last):
  File "", line 1, in
    List.index(2,2,4)
ValueError: tuple.index(x): x not in tuple
```

- **min() :** Calculates minimum of all the elements of List.
  Syntax:

```
min(List)
```

```
List = [2.3, 4.445, 3, 5.33, 1.054, 2.5]
print(min(List))
```

Output:

```
1.054
```

- **max():** Calculates maximum of all the elements of List.
  Syntax:

```
max(List)
```

```
List = [2.3, 4.445, 3, 5.33, 1.054, 2.5]
print(max(List))
```

Output:

```
5.33
```

sort() and reverse() functions

- **reverse():** Sort the given data structure (both tuple and list) in ascending order. Key and reverse_flag are not necessary parameter and reverse_flag is set to False, if nothing is passed through sorted().
  Syntax:

```
sorted([list[,key[,Reverse_Flag]]])
 list.sort([key,[Reverse_flag]])
```

```
List = [2.3, 4.445, 3, 5.33, 1.054, 2.5]

#Reverse flag is set True
List.sort(reverse=True)

#List.sort().reverse(), reverses the sorted list
print(List)
```

Output:

```
[5.33, 4.445, 3, 2.5, 2.3, 1.054]
```

```
List = [2.3, 4.445, 3, 5.33, 1.054, 2.5]
sorted(List)
print(List)
```

Output:

```
[1.054, 2.3, 2.5, 3, 4.445, 5.33]
```

Deletion of List Elements

To Delete one or more elements, i.e. remove an element, many built in functions can be used, such as pop() & remove() and keywords such as del.

- **pop():** Index is not a necessary parameter, if not mentioned takes the last index.
  Syntax:

```
 list.pop([index])
```

Note: Index must be in range of the List, elsewise IndexErrors occurs.

```
List = [2.3, 4.445, 3, 5.33, 1.054, 2.5]
```

```
print(List.pop())
```

Output:

```
2.5
```

```
List = [2.3, 4.445, 3, 5.33, 1.054, 2.5]
print(List.pop(0))
```

Output:

```
2.3
```

- **del() :** Element to be deleted is mentioned using list name and index.
  Syntax:

  ```
  del list.[index]
  ```

  ```
  List = [2.3, 4.445, 3, 5.33, 1.054, 2.5]
  del List[0]
  print(List)
  ```

  Output:

  ```
  [4.445, 3, 5.33, 1.054, 2.5]
  ```

- **remove():** Element to be deleted is mentioned using list name and element.
  Syntax:

  ```
   list.remove(element)
  ```

  ```
  List = [2.3, 4.445, 3, 5.33, 1.054, 2.5]
  List.remove(3)
  print(List)
  ```

  Output:

  ```
  [4.445, 5.33, 1.054, 2.5]
  ```

This article is contributed by **Piyush Doorwar**. 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.

Tuples in Python

A Tuple is a collection of Python objects separated by commas. In someways a tuple is similar to a list in terms of indexing, nested objects and repetition but a tuple is immutable unlike lists which are mutable.

Creating Tuples

```
# An empty tuple
empty_tuple = ()
print (empty_tuple)
```

Output:

```
 ()
```

```
# Creating non-empty tuples

# One way of creation
tup = 'python', 'geeks'
print(tup)

# Another for doing the same
tup = ('python', 'geeks')
print(tup)
```

Output

```
('python', 'geeks')
('python', 'geeks')
```

Note: In case your generating a tuple with a single element, make sure to add a comma after the element.

Concatenation of Tuples

```
# Code for concatenating 2 tuples

tuple1 = (0, 1, 2, 3)
tuple2 = ('python', 'geek')

# Concatenating above two
print(tuple1 + tuple2)
```

Output:

```
(0, 1, 2, 3, 'python', 'geek')
```

Nesting of Tuples

```
# Code for creating nested tuples

tuple1 = (0, 1, 2, 3)
tuple2 = ('python', 'geek')
tuple3 = (tuple1, tuple2)
print(tuple3)
```

Output :

```
((0, 1, 2, 3), ('python', 'geek'))
```

Repetition in Tuples

```
# Code to create a tuple with repetition

tuple3 = ('python',)*3
print(tuple3)
```

Output

```
 ('python', 'python', 'python')
```

Try the above without a comma and check. You will get tuple3 as a string 'pythonpythonpython'.

Immutable Tuples

```
#code to test that tuples are immutable

tuple1 = (0, 1, 2, 3)
tuple1[0] = 4
print(tuple1)
```

Output

```
Traceback (most recent call last):
  File "e0eaddff843a8695575daec34506f126.py", line 3, in
    tuple1[0]=4
TypeError: 'tuple' object does not support item assignment
```

Slicing in Tuples

```
# code to test slicing

tuple1 = (0 ,1, 2, 3)
print(tuple1[1:])
print(tuple1[::-1])
print(tuple1[2:4])
```

Output

```
(1, 2, 3)
(3, 2, 1, 0)
(2, 3)
```

Deleting a Tuple

```
# Code for deleting a tuple

tuple3 = ( 0, 1)
```

```
del tuple3
print(tuple3)
```

Error:

```
Traceback (most recent call last):
  File "d92694727db1dc9118a5250bf04dafbd.py", line 6, in <module>
    print(tuple3)
NameError: name 'tuple3' is not defined
```

Output:

```
(0, 1)
```

Finding Length of a Tuple

```
# Code for printing the length of a tuple

tuple2 = ('python', 'geek')
print(len(tuple2))
```

Output

```
 2
```

Converting list to a Tuple

```
# Code for converting a list and a string into a tuple

list1 = [0, 1, 2]
print(tuple(list1))
print(tuple('python')) # string 'python'
```

Output

```
(0, 1, 2)
('p', 'y', 't', 'h', 'o', 'n')
```

Takes a single parameter which may be a list,string,set or even a dictionary(only keys are taken as elements) and converts them to a tuple.

Tuples in a loop

```
#python code for creating tuples in a loop

tup = ('geek',)
n = 5  #Number of time loop runs
for i in range(int(n)):
    tup = (tup,)
    print(tup)
```

Output :

```
(('geek',),)
((('geek',),),)
(((('geek',),),),)
((((('geek',),),),),)
(((((('geek',),),),),),)
```

Using cmp(), max() , min()

```
# A python program to demonstrate the use of
# cmp(), max(), min()

tuple1 = ('python', 'geek')
tuple2 = ('coder', 1)

if (cmp(tuple1, tuple2) != 0):

    # cmp() returns 0 if matched, 1 when not tuple1
    # is longer and -1 when tuple1 is shoter
    print('Not the same')
else:
    print('Same')
print ('Maximum element in tuples 1,2: ' +
        str(max(tuple1)) +  ',' +
        str(max(tuple2)))
print ('Minimum element in tuples 1,2: ' +
     str(min(tuple1)) + ','  + str(min(tuple2)))
```

Output

```
Not the same
Maximum element in tuples 1,2: python,coder
Minimum element in tuples 1,2: geek,1
```

Note: max() and min() checks the based on ASCII values. If there are two strings in a tuple, then the first different character in the strings are checked.

This article is contributed by **Sri Sanketh Uppalapati.** Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

GATE CS Notes (According to Official GATE 2017 Syllabus)

GATE CS Corner

See Placement Course for placement preparation, GATE Corner for GATE CS Preparation and Quiz Corner for all Quizzes on GeeksQuiz.
Category: Python

Sets in Python

A Set is an unordered collection data type that is iterable, mutable, and has no duplicate elements. Python's set class represents the mathematical notion of a set. The major advantage of using a set, as opposed to a list, is that it has a highly optimized method for checking whether a speci?c element is contained in the set. This is based on a data structure known as a hash table.

Frozen Sets Frozen sets are immutable objects that only support methods and operators that produce a result without a?ecting the frozen set or sets to which they are applied.

```
# Python program to demonstrate differences
# between normal and frozen set

# Same as {"a", "b","c"}
normal_set = set(["a", "b","c"])

# Adding an element to normal set is fine
normal_set.add("d")

print("Normal Set")
print(normal_set)

# A frozen set
frozen_set = frozenset(["e", "f", "g"])

print("Frozen Set")
print(frozen_set)

# Uncommenting below line would cause error as
# we are trying to add element to a frozen set
# frozen_set.add("h")
```

Output:

```
Normal Set
set(['a', 'c', 'b', 'd'])
Frozen Set
frozenset(['e', 'g', 'f'])
```

Methods for Sets

1. add(x) Method: Adds the item x to set if it is not already present in the set.

```
people = {"Jay", "Idrish", "Archil"}
people.add("Daxit")
```

-> This will add Daxit in people set.

2. **union(s) Method**: Returns a union of two set.Using the '|' operator between 2 sets is the same as writing set1.union(set2)

```
people = {"Jay", "Idrish", "Archil"}
vampires = {"Karan", "Arjun"}
population = people.union(vampires)
```

OR

```
population = people|vampires
```

-> Set population set will have components of both people and vampire

3. **intersect(s) Method:** Returns an intersection of two sets.The '&' operator comes can also be used in this case.

```
victims = people.intersection(vampires)
```

-> Set victims will contain the common element of people and vampire

4. difference(s) Method: Returns a set containing all the elements of invoking set but not of the second set. We can use '-' operator here.

```
safe = people.difference(vampires)
```

OR

```
safe = people – vampires
```

-> Set safe will have all the elements that are in people but not vampire
5. clear() Method: Empties the whole set.

```
victims.clear()
```

-> Clears victim set

However there are two major pitfalls in Python sets:

1. The set doesn't maintain elements in any particular order.
2. Only instances of immutable types can be added to a Python set.

Operators for Sets

Sets and frozen sets support the following operators:

key in s   # containment check

key not in s   # non-containment check

s1 == s2   # s1 is equivalent to s2

s1 != s2   # s1 is not equivalent to s2

s1 <= s2   # s1is subset of s2

s1 < s2   # s1 is proper subset of s2

s1 >= s2   # s1is superset of s2

s1 > s2   # s1 is proper superset of s2

s1 | s2   # the union of s1 and s2

s1 & s2 # the intersection of s1 and s2

s1 – s2   # the set of elements in s1 but not s2

s1 ˆ s2   # the set of elements in precisely one of s1 or s2

Code Snippet to illustrate all Set operations in Python

```
# Python program to demonstrate working# of
# Set in Python

# Creating two sets
set1 = set()
set2 = set()

# Adding elements to set1
for i in range(1, 6):
    set1.add(i)

# Adding elements to set2
for i in range(3, 8):
    set2.add(i)

print("Set1 = ", set1)
print("Set2 = ", set2)
print("\n")

# Union of set1 and set2
set3 = set1 | set2# set1.union(set2)
print("Union of Set1 & Set2: Set3 = ", set3)

# Intersection of set1 and set2
set4 = set1 & set2# set1.intersection(set2)
print("Intersection of Set1 & Set2: Set4 = ", set4)
print("\n")

# Checking relation between set3 and set4
if set3 > set4: # set3.issuperset(set4)
    print("Set3 is superset of Set4")
elif set3 < set4: # set3.issubset(set4)
    print("Set3 is subset of Set4")
else : # set3 == set4
    print("Set3 is same as Set4")

# displaying relation between set4 and set3
if set4 < set3: # set4.issubset(set3)
    print("Set4 is subset of Set3")
    print("\n")
```

```
# difference between set3 and set4
set5 = set3 - set4
print("Elements in Set3 and not in Set4: Set5 = ", set5)
print("\n")

# checkv if set4 and set5 are disjoint sets
if set4.isdisjoint(set5):
    print("Set4 and Set5 have nothing in common\n")

# Removing all the values of set5
set5.clear()

print("After applying clear on sets Set5: ")
print("Set5 = ", set5)
```

Output:

```
('Set1 = ', set([1, 2, 3, 4, 5]))
('Set2 = ', set([3, 4, 5, 6, 7]))


('Union of Set1 & Set2: Set3 = ', set([1, 2, 3, 4, 5, 6, 7]))
('Intersection of Set1 & Set2: Set4 = ', set([3, 4, 5]))


Set3 is superset of Set4
Set4 is subset of Set3


('Elements in Set3 and not in Set4: Set5 = ', set([1, 2, 6, 7]))


Set4 and Set5 have nothing in common

After applying clear on sets Set5:
('Set5 = ', set([]))
```

This article is contributed by **Jay Patel.** 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

Array in Python | Set 1 (Introduction and Functions)

Other than some generic containers like list, Python in its definition can also handle containers with specified data types. Array can be handled in python by module named "**array**". They can be useful when we have to manipulate only a specific data type values.

Operations on Array :

1. array(data type, value list) :- This function is used to **create** an array with data type and value list specified in its arguments. Some of the data types are mentioned in the table below.

2. append() :- This function is used to **add the value** mentioned in its arguments at the **end** of the array.

3. insert(i,x) :- This function is used to **add the value at the position** specified in its argument.

```
# Python code to demonstrate the working of
# array(), append(), insert()

# importing "array" for array operations
import array

# initializing array with array values
# initializes array with signed integers
arr = array.array('i', [1, 2, 3])

# printing original array
print ("The new created array is : ",end="")
for i in range (0,3):
    print (arr[i], end=" ")

print ("\r")

# using append() to insert new value at end
arr.append(4);

# printing appended array
print ("The appended array is : ", end="")
for i in range (0, 4):
    print (arr[i], end=" ")

# using insert() to insert value at specific position
# inserts 5 at 2nd position
arr.insert(2, 5)

print("\r")

# printing array after insertion
print ("The array after insertion is : ", end="")
for i in range (0, 5):
    print (arr[i], end=" ")
```

Output:

```
The new created array is : 1 2 3
The appended array is : 1 2 3 4
The array after insertion is : 1 2 5 3 4
```

4. pop() :- This function **removes the element at the position** mentioned in its argument, and returns it.

5. remove() :- This function is used to **remove the first occurrence** of the value mentioned in its arguments.

```
# Python code to demonstrate the working of
# pop() and remove()

# importing "array" for array operations
import array

# initializing array with array values
# initializes array with signed integers
arr= array.array('i',[1, 2, 3, 1, 5])

# printing original array
print ("The new created array is : ",end="")
for i in range (0,5):
    print (arr[i],end=" ")

print ("\r")

# using pop() to remove element at 2nd position
print ("The popped element is : ",end="")
print (arr.pop(2));

# printing array after popping
print ("The array after popping is : ",end="")
for i in range (0,4):
    print (arr[i],end=" ")

print("\r")

# using remove() to remove 1st occurrence of 1
arr.remove(1)

# printing array after removing
print ("The array after removing is : ",end="")
for i in range (0,3):
    print (arr[i],end=" ")
```

Output:

```
The new created array is : 1 2 3 1 5
The popped element is : 3
The array after popping is : 1 2 1 5
The array after removing is : 2 1 5
```

6. index() :- This function returns the **index of the first occurrence** of value mentioned in arguments.

7. reverse() :- This function **reverses** the array.

```
# Python code to demonstrate the working of
# index() and reverse()

# importing "array" for array operations
import array

# initializing array with array values
# initializes array with signed integers
arr= array.array('i',[1, 2, 3, 1, 2, 5])

# printing original array
print ("The new created array is : ",end="")
for i in range (0,6):
    print (arr[i],end=" ")

print ("\r")

# using index() to print index of 1st occurrenece of 2
print ("The index of 1st occurrence of 2 is : ",end="")
print (arr.index(2))

#using reverse() to reverse the array
arr.reverse()

# printing array after reversing
print ("The array after reversing is : ",end="")
for i in range (0,6):
    print (arr[i],end=" ")
```

Output:

```
The new created array is : 1 2 3 1 2 5
The index of 1st occurrence of 2 is : 1
The array after reversing is : 5 2 1 3 2 1
```

Reference :

https://docs.python.org/3/library/array.html#module-array

This article is contributed by **Manjeet Singh**. 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

Array in Python | Set 2 (Important Functions)

Array in Python | Set 1 (Introduction and Functions)

Below are some more functions.

1. typecode :- This function **returns the data type** by which array is initialised.

2. itemsize :- This function returns **size** in bytes of a **single array element.**

3. buffer_info() :- Returns a tuple representing the **address in which array is stored and number of elements** in it.

```
# Python code to demonstrate the working of
# typecode, itemsize, buffer_info()

# importing "array" for array operations
import array

# initializing array with array values
# initializes array with signed integers
arr= array.array('i',[1, 2, 3, 1, 2, 5])

# using typecode to print datatype of array
print ("The datatype of array is : ",end="")
print (arr.typecode)

# using itemsize to print itemsize of array
print ("The itemsize of array is : ",end="")
print (arr.itemsize)

# using buffer_info() to print buffer info. of array
print ("The buffer info. of array is : ",end="")
print (arr.buffer_info())
```

Output:

```
The datatype of array is : i
The itemsize of array is : 4
The buffer info. of array is : (32497808, 6)
```

4. count() :- This function **counts the number of occurrences** of argument mentioned in array.

5. extend(arr) :- This function **appends a whole array** mentioned in its arguments to the specified array.

```
# Python code to demonstrate the working of
# count() and extend()

# importing "array" for array operations
import array

# initializing array 1 with array values
# initializes array with signed integers
arr1 = array.array('i',[1, 2, 3, 1, 2, 5])

# initializing array 2 with array values
# initializes array with signed integers
arr2 = array.array('i',[1, 2, 3])
```

```
# using count() to count occurrences of 1 in array
print ("The occurrences of 1 in array is : ",end="")
print (arr1.count(1))

# using extend() to add array 2 elements to array 1
arr1.extend(arr2)

print ("The modified array is : ",end="")
for i in range (0,9):
    print (arr1[i],end=" ")
```

Output:

```
The occurrences of 1 in array is : 2
The modified array is : 1 2 3 1 2 5 1 2 3
```

6. fromlist(list) :- This function is used to **append a list** mentioned in its argument **to end of array**.

7. tolist() :- This function is used to **transform an array into a list**.

```
# Python code to demonstrate the working of
# fromlist() and tolist()

# importing "array" for array operations
import array

# initializing array with array values
# initializes array with signed integers
arr = array.array('i',[1, 2, 3, 1, 2, 5])

# initializing list
li = [1, 2, 3]

# using fromlist() to append list at end of array
arr.fromlist(li)

# printing the modified array
print ("The modified array is : ",end="")
for i in range (0,9):
    print (arr[i],end=" ")

# using tolist() to convert array into list
li2 = arr.tolist()

print ("\r")

# printing the new list
print ("The new list created is : ",end="")
for i in range (0,len(li2)):
    print (li2[i],end=" ")
```

Output:

```
The modified array is : 1 2 3 1 2 5 1 2 3
The new list created is : 1 2 3 1 2 5 1 2 3
```

This article is contributed by **Manjeet Singh**. 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

Python | Set 4 (Dictionary, Keywords in Python)

In the previous two articles (Set 2 and Set 3), we discussed the basics of python. In this article, we will learn more about python and feel the power of python.

Dictionary in Python

In python, dictionary is similar to hash or maps in other languages. It consists of key value pairs. The value can be accessed by unique key in the dictionary.

```
# Create a new dictionary
d = dict() # or d = {}

# Add a key - value pairs to dictionary
d['xyz'] = 123
d['abc'] = 345

# print the whole dictionary
print d
```

```
# print only the keys
print d.keys()

# print only values
print d.values()

# iterate over dictionary
for i in d :
    print "%s  %d" %(i, d[i])

# another method of iteration
for index, value in enumerate(d):
    print index, value , d[value]

# check if key exist
print 'xyz' in d

# delete the key-value pair
del d['xyz']

# check again
print "xyz" in d
```

Output:

```
{'xyz': 123, 'abc': 345}
['xyz', 'abc']
[123, 345]
xyz 123
abc 345
0 xyz 123
1 abc 345
True
False
```

break, continue, pass in Python

break – takes you out of the current loop.

continue – ends the current iteration in the loop and moves to the next iteration.

pass – The pass statement does nothing. It can be used when a statement is required. syntactically but the program requires no action. It is commonly used for creating minimal classes.

```
# Function to illustrate break in loop
def breakTest(arr):
    for i in arr:
        if i == 5:
            break
        print i,
    # For new line
    print


# Function to illustrate continue in loop
def continueTest(arr):
    for i in arr:
        if i == 5:
            continue
        print i,

    # For new line
    print

# Function to illustrate pass
def passTest(arr):
        pass


# Driver program to test above functions

# Array to be used for above functions:
arr = [1, 3 , 4, 5, 6 , 7]

# Illustrate break
print "Break method output"
breakTest(arr)

# Illustrate continue
print "Continue method output"
continueTest(arr)

# Illustrate pass- Does nothing
```

```
passTest(arr)
```

Output:

```
Break method output
1 3 4
Continue method output
1 3 4 6 7
```

map, filter, lambda

map – 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.

filter – It takes a function returning True or False and applies it to a sequence, returning a list of only those members of the sequence for which the function returned True.

lambda- Python provides the ability to create a simple (no statements allowed internally) anonymous inline function called lambda function. Using lambda and map you can have two for loops in one line.

```
# Python program to test map, filter and lambda

# Function to test map
def cube(x):
    return x**2

# Driver to test above function

# Program for working of map
print "MAP EXAMPLES"
cubes = map(cube, range(10))
print cubes

print "LAMBDA EXAMPLES"

# first parentheses contains a lambda form, that is
# a squaring function and second parentheses represents
# calling lambda
print (lambda x: x**2)(5)

# Make function of two arguments that return their product
print (lambda x, y: x*y)(3, 4)


print "FILTER EXAMPLE"
special_cubes = filter(lambda x: x > 9 and x < 60, cubes)
print special_cubes
```

Output:

```
MAP EXAMPLES
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
LAMBDA EXAMPLES
25
12
FILTER EXAMPLE
[16, 25, 36, 49]
```

For more clarity about map, filter and lambda, you can have a look at the below example:

```
#code without using map, filter and lambda

# Find the number which are odd in the list
# and multiply them by 5 and create a new list

# Declare a new list
x = [2, 3, 4, 5, 6]

# Empty list for answer
y = []

# Perform the operations and print the answer
for v in x:
    if v % 2:
        y += [v*5]
print y
```

Output:

```
[15, 25]
```

The same operation can be performed in two lines using map, filter and lambda as :

```
#above code with map, filter and lambda

# Declare a list
x = [2, 3, 4, 5, 6]

# Perform the same operation as  in above post
y = map(lambda v : v * 5, filter(lambda u : u % 2, x))
print y
```

Output:

```
[15, 25]
```

References :
https://docs.python.org/2/tutorial/controlflow.html#break-and-continue-statements-and-else-clauses-on-loops
http://www.u.arizona.edu/~erdmann/mse350/topics/list_comprehensions.html

This article is contributed by **Nikhil Kumar Singh**

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

GATE CS Notes (According to Official GATE 2017 Syllabus)

GATE CS Corner

See Placement Course for placement preparation, GATE Corner for GATE CS Preparation and Quiz Corner for all Quizzes on GeeksQuiz.
Category: Python Articles

Dictionary Methods in Python | Set 1 (cmp(), len(), items()…)

Python Dictionary Basics have been discussed in the article below

Dictionary

Some dictionary methods are discussed in this article.

1. str(dic) :- This method is used to **return the string**, denoting all the dictionary keys with their values.

2. items() :- This method is used to **return the list** with all dictionary keys with values.

```
# Python code to demonstrate working of
# str() and items()

# Initializing dictionary
dic = { 'Name' : 'Nandini', 'Age' : 19 }

# using str() to display dic as string
print ("The constituents of dictionary as string are : ")
print (str(dic))

# using str() to display dic as list
print ("The constituents of dictionary as list are : ")
print (dic.items())
```

Output:

```
The constituents of dictionary as string are :
{'Name': 'Nandini', 'Age': 19}
The constituents of dictionary as list are :
dict_items([('Name', 'Nandini'), ('Age', 19)])
```

3. len() :- It returns the **count of key entities** of the dictionary elements.

4. type() :- This function **returns the data type** of the argument.

```
# Python code to demonstrate working of
# len() and type()

# Initializing dictionary
dic = { 'Name' : 'Nandini', 'Age' : 19, 'ID' : 2541997 }

# Initializing list
li = [ 1, 3, 5, 6 ]
```

```
# using len() to display dic size
print ("The size of dic is : ",end="")
print (len(dic))

# using type() to display data type
print ("The data type of dic is : ",end="")
print (type(dic))

# using type() to display data type
print ("The data type of li is : ",end="")
print (type(li))
```

Output:

```
The size of dic is : 3
The data type of dic is :
The data type of li is :
```

5. copy() :- This function creates the **shallow copy of the dictionary** into other dictionary.

6. clear() :- This function is used to **clear the dictionary** contents.

```
# Python code to demonstrate working of
# clear() and copy()

# Initializing dictionary
dic1 = { 'Name' : 'Nandini', 'Age' : 19 }

# Initializing dictionary
dic3 = {}

# using copy() to make shallow copy of dictionary
dic3 = dic1.copy()

# printing new dictionary
print ("The new copied dictionary is : ")
print (dic3.items())

# clearing the dictionary
dic1.clear()

# printing cleared dictionary
print ("The contents of deleted dictionary is : ",end="")
print (dic1.items())
```

Output:

```
The new copied dictionary is :
dict_items([('Age', 19), ('Name', 'Nandini')])
The contents of deleted dictionary is : dict_items([])
```

This article is contributed by **Manjeet Singh**. 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

Dictionary Methods in Python | Set 2 (update(), has_key(), fromkeys()…)

Some of the Dictionary methods are discussed in set 1 below

Dictionary Methods in Python | Set 1 (cmp(), len(), items()…)

More methods are discussed in this article.

1. fromkeys(seq,value) :- This method is used to declare a **new dictionary from the sequence** mentioned in its arguments. This function can also **initialize the declared dictionary with "value" argument**.

2. update(dic) :- This function is used to **update the dictionary to add other dictionary keys**.

```
# Python code to demonstrate working of
# fromkeys() and update()

# Initializing dictionary 1
dic1 = { 'Name' : 'Nandini', 'Age' : 19 }

# Initializing dictionary 2
dic2 = { 'ID' : 2541997 }

# Initializing sequence
sequ = ('Name', 'Age', 'ID')

# using update to add dic2 values in dic 1
```

```
dic1.update(dic2)

# printing updated dictionary values
print ("The updated dictionary is : ")
print (str(dic1))

# using fromkeys() to transform sequence into dictionary
dict = dict.fromkeys(sequ,5)

# printing new dictionary values
print ("The new dictionary values are : ")
print (str(dict))
```

Output:

```
The updated dictionary is :
{'Age': 19, 'Name': 'Nandini', 'ID': 2541997}
The new dictionary values are :
{'Age': 5, 'Name': 5, 'ID': 5}
```

3. has_key() :- This function returns true if **specified key is present** in the dictionary, else returns false.

4. get(key, def_val) :- This function return the **key value associated with the key** mentioned in arguments. If key is not present, the default value is returned.

```
# Python code to demonstrate working of
# has_key() and get()

# Initializing dictionary
dict = { 'Name' : 'Nandini', 'Age' : 19 }

# using has_key() to check if dic1 has a key
if dict.has_key('Name'):
    print ("Name is a key")
else : print ("Name is not a key")

# using get() to print a key value
print ("The value associated with ID is : ")
print (dict.get('ID', "Not Present"))

# printing dictionary values
print ("The dictionary values are : ")
print (str(dict))
```

Output:

```
Name is a key
The value associated with ID is :
Not Present
The dictionary values are :
{'Name': 'Nandini', 'Age': 19}
```

5. setdefault(key, def_value) :- This function also **searches for a key** and displays its value like get() but, it **creates new key with def_value** if key is not present.

```
# Python code to demonstrate working of
# setdefault()

# Initializing dictionary
dict = { 'Name' : 'Nandini', 'Age' : 19 }

# using setdefault() to print a key value
print ("The value associated with Age is : ",end="")
print (dict.setdefault('ID', "No ID"))

# printing dictionary values
print ("The dictionary values are : ")
print (str(dict))
```

Output:

```
The value associated with Age is : No ID
The dictionary values are :
{'Name': 'Nandini', 'Age': 19, 'ID': 'No ID'}
```

Python

Get() method for dictionaries in Python

In python dictionaries, following is a conventional method to access a value for a key.

```
dic = {"A":1, "B":2}
print(dic["A"])
print(dic["C"])
```

The problem that arises here is that the 3rd line of the code returns a key error :

```
Traceback (most recent call last):
  File ".\dic.py", line 3, in
    print (dic["C"])
KeyError: 'C'
```

The **get()** method is used to avoid such situations. This method returns the value for the given key, if present in the dictionary. If not, then it will return None (if get() is used with only one argument).

Syntax :

Dict.get(key, default=None)

Example:

```
dic = {"A":1, "B":2}
print(dic.get("A"))
print(dic.get("C"))
print(dic.get("C","Not Found ! "))
```

Output:

```
1
None
Not Found !
```

This article is contributed by **Mayank Rawat** .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

Handling missing keys in Python dictionaries

In python, dictionaries are containers which map one key to its value with access time complexity to be **O(1)**. But in many applications, the user doesn't know all the keys present in the dictionaries. In such instances, **if user tries to access a missing key, an error is popped indicating missing keys**.

```
# Python code to demonstrate Dictionary and
# missing value error

# initializing Dictionary
d = { 'a' : 1 , 'b' : 2 }

# trying to output value of absent key
print ("The value associated with 'c' is : ")
print (d['c'])
```

Error :

```
Traceback (most recent call last):
  File "46a9aac96614587f5b794e451a8f4f5f.py", line 9, in
    print (d['c'])
KeyError: 'c'
```

In the above example, no key named 'c' in dictionary, popped a runtime error. To avoid such conditions, and to make aware user that a particular key is absent or to pop a default message in that place, there are several methods to handle missing keys.

Method 1 : Using get()

get(key,def_val) method is useful when we have to check for the key. If the key is present, value associated with the key is printed, else the def_value passed in arguments is returned.

Method 2 : Using setdefault()

setdefault(key, def_value) works in a similar way as get(), but the difference is that each time a **key is absent, a new key is created** with the def_value associated to the key passed in arguments.

For implementation of above functions, click here.

Method 3 : Using defaultdict

"**defaultdict**" is a container that is defined in module named "**collections**". It takes a **function(default factory) as its argument**. By default, **default factory is set to "int" i.e 0**. If a **key is not present** is defaultdict, the **default factory value is returned** and displayed. It has advantages over get() or setdefault().

- A default value is set at the **declaration**. There is **no need** to invoke the function again and again and pass similar value as arguments. Hence **saving time**.
- The implementation of defaultdict is **faster** than get() or setdefault().

```
# Python code to demonstrate defaultdict

# importing "collections" for defaultdict
import collections

# declaring defaultdict
# sets default value 'Key Not found' to absent keys
defd = collections.defaultdict(lambda : 'Key Not found')

# initializing values
defd['a'] = 1

# initializing values
defd['b'] = 2

# printing value
print ("The value associated with 'a' is : ",end="")
print (defd['a'])

# printing value associated with 'c'
print ("The value associated with 'c' is : ",end="")
print (defd['c'])
```

Output :

```
The value associated with 'a' is : 1
The value associated with 'c' is : Key Not found
```

This article is contributed by **Manjeet Singh**. 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

ChainMap in Python

Python also contains a container called “**ChainMap**” which encapsulates many dictionaries into one unit. ChainMap is member of module “**collections**“.

Operations on ChainMap

Access Operations

1. keys() :- This function is used to display all the **keys** of all the dictionaries in ChainMap.

2. values() :- This function is used to display **values** of all the dictionaries in ChainMap.

3. maps :- This function is used to display **keys with corresponding values** of all the dictionaries in ChainMap.

```
# Please select Python 3 for running this code in IDE
# Python code to demonstrate ChainMap and
# keys(), values() and maps

# importing collections for ChainMap operations
import collections

# initializing dictionaries
dic1 = { 'a' : 1, 'b' : 2 }
dic2 = { 'b' : 3, 'c' : 4 }

# initializing ChainMap
chain = collections.ChainMap(dic1, dic2)

# printing chainMap using maps
print ("All the ChainMap contents are : ")
print (chain.maps)

# printing keys using keys()
print ("All keys of ChainMap are : ")
print (list(chain.keys()))

# printing keys using keys()
print ("All values of ChainMap are : ")
print (list(chain.values()))
```

Output :

```
All the ChainMap contents are :
[{'b': 2, 'a': 1}, {'c': 4, 'b': 3}]
All keys of ChainMap are :
['a', 'c', 'b']
All values of ChainMap are :
[1, 4, 2]
```

Note : Notice the key named “b” exists in both dictionaries, but only first dictionary key is taken as key value of “b”. Ordering is done as the dictionaries are passed in function.

Manipulating Operations

1. new_child() :- This function adds a new dictionary in the beginning of the ChainMap.

2. reversed() :- This function reverses the relative ordering of dictionaries in the ChainMap.

```
# Please select Python 3 for running this code in IDE
# Python code to demonstrate ChainMap and
# reversed() and new_child()

# importing collections for ChainMap operations
import collections

# initializing dictionaries
dic1 = { 'a' : 1, 'b' : 2 }
dic2 = { 'b' : 3, 'c' : 4 }
dic3 = { 'f' : 5 }

# initializing ChainMap
chain = collections.ChainMap(dic1, dic2)

# printing chainMap using map
print ("All the ChainMap contents are : ")
print (chain.maps)

# using new_child() to add new dictionary
chain1 = chain.new_child(dic3)

# printing chainMap using map
print ("Displaying new ChainMap : ")
print (chain1.maps)

# displaying value associated with b before reversing
print ("Value associated with b before reversing is : ",end="")
print (chain1['b'])

# reversing the ChainMap
chain1.maps = reversed(chain1.maps)

# displaying value associated with b after reversing
print ("Value associated with b after reversing is : ",end="")
print (chain1['b'])
```

Output :

```
All the ChainMap contents are :
[{'b': 2, 'a': 1}, {'b': 3, 'c': 4}]
Displaying new ChainMap :
[{'f': 5}, {'b': 2, 'a': 1}, {'b': 3, 'c': 4}]
Value associated with b before reversing is : 2
Value associated with b after reversing is : 3
```

This article is contributed by **Manjeet Singh**. 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

OrderedDict in Python

An **OrderedDict** is a dictionary subclass that remembers the order that keys were first inserted. The only difference between dict() and OrderedDict() is that:

OrderedDict **preserves the order** in which the keys are inserted. A regular dict doesn’t track the insertion order, and iterating it gives the values in an arbitrary order. By contrast, the order the items are inserted is remembered by OrderedDict.

```
# A Python program to demonstrate working of OrderedDict
from collections import OrderedDict

print("This is a Dict:\n")
d = {}
d['a'] = 1
d['b'] = 2
d['c'] = 3
d['d'] = 4

for key, value in d.items():
    print(key, value)

print("\nThis is an Ordered Dict:\n")
od = OrderedDict()
od['a'] = 1
```

```
od['b'] = 2
od['c'] = 3
od['d'] = 4

for key, value in od.items():
    print(key, value)
```

Output:

```
This is a Dict:
('a', 1)
('c', 3)
('b', 2)
('d', 4)

This is an Ordered Dict:
('a', 1)
('b', 2)
('c', 3)
('d', 4)
```

Important Points:

1. **Key value Change:** If the value of a certain key is changed, the position of the key remains unchanged in OrderedDict.

   ```
   # A Python program to demonstrate working of key
   # value change in OrderedDict
   from collections import OrderedDict

   print("Before:\n")
   od = OrderedDict()
   od['a'] = 1
   od['b'] = 2
   od['c'] = 3
   od['d'] = 4
   for key, value in od.items():
       print(key, value)

   print("\nAfter:\n")
   od['c'] = 5
   for key, value in od.items():
       print(key, value)
   ```

   Output:

   ```
   Before:

   ('a', 1)
   ('b', 2)
   ('c', 3)
   ('d', 4)

   After:

   ('a', 1)
   ('b', 2)
   ('c', 5)
   ('d', 4)
   ```

2. **Deletion and Re-Inserting**: Deleting and re-inserting the same key will push it to the back as OrderedDict however maintains the order of insertion.

   ```
   # A Python program to demonstrate working of deletion
   # re-inserion in OrderedDict
   from collections import OrderedDict

   print("Before deleting:\n")
   od = OrderedDict()
   od['a'] = 1
   od['b'] = 2
   od['c'] = 3
   od['d'] = 4

   for key, value in od.items():
       print(key, value)

   print("\nAfter deleting:\n")
   od.pop('c')
   for key, value in od.items():
       print(key, value)

   print("\nAfter re-inserting:\n")
   od['c'] = 3
   for key, value in od.items():
       print(key, value)
   ```

   Output:

   ```
   Before deleting:

   ('a', 1)
   ```

```
('b', 2)
('c', 3)
('d', 4)

After deleting:

('a', 1)
('b', 2)
('d', 4)

After re-inserting:

('a', 1)
('b', 2)
('d', 4)
('c', 3)
```

This article is contributed by **Sri Sanketh Uppalapati**. 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

Object Oriented Programming in Python | Set 1 (Class, Object and Members)

Below is a simple Python program that creates a class with single method.

```
# A simple example class
class Test:

    # A sample method
    def fun(self):
        print("Hello")

# Driver code
obj = Test()
obj.fun()
```

Output:

```
Hello
```

As we can see above, we create a new class using the class statement and the name of the class. This is followed by an indented block of statements which form the body of the class. In this case, we have defined a single method in the class.

Next, we create an object/instance of this class using the name of the class followed by a pair of parentheses.

The self

1. Class methods must have an extra first parameter in method definition. We do not give a value for this parameter when we call the method, Python provides it
2. If we have a method which takes no arguments, then we still have to have one argument – the self. See fun() in above simple example.
3. This is similar to this pointer in C++ and this reference in Java.

When we call a method of this object as myobject.method(arg1, arg2), this is automatically converted by Python into MyClass.method(myobject, arg1, arg2) – this is all the special self is about.

The __init__ method
The __init__ method is similar to constructors in C++ and Java. It is run as soon as an object of a class is instantiated. The method is useful to do any initialization you want to do with your object.

```
# A Sample class with init method
class Person:

    # init method or constructor
    def __init__(self, name):
        self.name = name

    # Sample Method
    def say_hi(self):
        print('Hello, my name is', self.name)

p = Person('Shwetanshu')
p.say_hi()
```

Output:

```
Hello, my name is Shwetanshu
```

Here, we define the __init__ method as taking a parameter name (along with the usual self). .

Class and Instance Variables (Or attributes)

In Python, instance variables are variables whose value is assigned inside a constructor or method with self.

Class variables are variables whose value is assigned in class.

```
# Python program to show that the variables with a value
# assigned in class declaration, are class variables and
# variables inside methods and constructors are instance
# variables.

# Class for Computer Science Student
class CSStudent:

    # Class Variable
    stream = 'cse'

    # The init method or constructor
    def __init__(self, roll):

        # Instance Variable
        self.roll = roll

# Objects of CSStudent class
a = CSStudent(101)
b = CSStudent(102)

print(a.stream)  # prints "cse"
print(b.stream)  # prints "cse"
print(a.roll)    # prints 101

# Class variables can be accessed using class
# name also
print(CSStudent.stream) # prints "cse"
```

We can define instance variables inside normal methods also.

```
# Python program to show that we can create
# instance variables inside methods

# Class for Computer Science Student
class CSStudent:

    # Class Variable
    stream = 'cse'

    # The init method or constructor
    def __init__(self, roll):

        # Instance Variable
        self.roll = roll

    # Adds an instance variable
    def setAddress(self, address):
        self.address = address

    # Retrieves instance variable
    def getAddress(self):
        return self.address

# Driver Code
a = CSStudent(101)
a.setAddress("Noida, UP")
print(a.getAddress())
```

Output :

```
Noida, UP
```

How to create an empty class?

We can create an empty class using pass statement in Python.

```
# An empty class
class Test:
    pass
```

Object Oriented Programming in Python | Set 2 (Data Hiding and Object Printing)

This article is contributed by **Shwetanshu Rohatgi**. 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

Object Oriented Programming in Python | Set 2 (Data Hiding and Object Printing)

Prerequisite : Object Oriented Programming in Python | Set 1 (Class, Object and Members)

Data hiding

In Python, we use double underscore (Or __) before the attributes name and those attributes will not be directly visible outside.

```
class MyClass:

    # Hidden member of MyClass
    __hiddenVariable = 0

    # A member method that changes
    # __hiddenVariable
    def add(self, increment):
        self.__hiddenVariable += increment
        print (self.__hiddenVariable)

# Driver code
myObject = MyClass()
myObject.add(2)
myObject.add(5)

# This line causes error
print (myObject.__hiddenVariable)
```

Output :

```
2
7
Traceback (most recent call last):
  File "filename.py", line 13, in
    print (myObject.__hiddenVariable)
AttributeError: MyClass instance has
no attribute '__hiddenVariable'
```

In the above program, we tried to access hidden variable outside the class using object and it threw an exception.

We can access the value of hidden attribute by a tricky syntax:

```
# A Python program to demonstrate that hidden
# members can be accessed outside a class
class MyClass:

    # Hidden member of MyClass
    __hiddenVariable = 10

# Driver code
myObject = MyClass()
print(myObject._MyClass__hiddenVariable)
```

Output :

```
10
```

Private methods are accessible outside their class, just not easily accessible. **Nothing in Python is truly private; internally**, the names of private methods and attributes are mangled and unmangled on the fly to make them seem inaccessible by their given names [See this for source].

Printing Objects

Printing objects gives us information about objects we are working with. In C++, we can do this by adding a friend ostream& operator From str method of Test: a is 1234,b is 5678 [Test a:1234 b:5678]

Important Points about Printing:

- If no __str__ method is defined, print t (or print str(t)) uses __repr__.

```
class Test:
    def __init__(self, a, b):
        self.a = a
        self.b = b

    def __repr__(self):
        return "Test a:%s b:%s" % (self.a, self.b)

# Driver Code
t = Test(1234, 5678)
print(t)
```

  Output :

```
Test a:1234 b:5678
```

- If no __repr__ method is defined then the default is used.

```
class Test:
    def __init__(self, a, b):
        self.a = a
        self.b = b

# Driver Code
t = Test(1234, 5678)
print(t)
```

Output :

```
<__main__.test at="" instance="">
```

This article is contributed by **Shwetanshu Rohatgi**. 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

OOP in Python | Set 3 (Inheritance, examples of object, issubclass and super)

We have discussed following topics on Object Oriented Programming in Python

- Object Oriented Programming in Python | set-1
- Object Oriented Programming in Python | Set 2 (Data Hiding and Object Printing)

In this article, Inheritance is introduced.

One of the major advantages of Object Oriented Programming is re-use. Inheritance is one of the mechanisms to achieve the same. In inheritance, a class (usually called superclass) is inherited by another class (usually called subclass). The subclass adds some attributes to superclass.

Below is a sample Python program to show how inheritance is implemented in Python.

```

# A Python program to demonstrate inheritance

# Base or Super class. Note object in bracket.
# (Generally, object is made ancestor of all classes)
# In Python 3.x "class Person" is
# equivalent to "class Person(object)"
class Person(object):

    # Constructor
    def __init__(self, name):
        self.name = name

    # To get name
    def getName(self):
        return self.name

    # To check if this person is employee
    def isEmployee(self):
        return False


# Inherited or Sub class (Note Person in bracket)
class Employee(Person):

    # Here we return true
    def isEmployee(self):
        return True

# Driver code
emp = Person("Geek1")  # An Object of Person
print(emp.getName(), emp.isEmployee())

emp = Employee("Geek2") # An Object of Employee
print(emp.getName(), emp.isEmployee())
```

Output :

```
Geek1 False
Geek2 True
```

How to check if a class is subclass of another?
Python provides a function issubclass() that directly tells us if a class is subclass of another class.

```
# Python example to check if a class is
# subclass of another
```

```
class Base(object):
    pass   # Empty Class

class Derived(Base):
    pass   # Empty Class

# Driver Code
print(issubclass(Derived, Base))
print(issubclass(Base, Derived))

d = Derived()
b = Base()

# b is not an instance of Derived
print(isinstance(b, Derived))

# But d is an instance of Base
print(isinstance(d, Base))
```

Output :

```
True
False
False
True
```

What is object class?
Like Java Object class, in Python (from version 3.x), object is root of all classes.

In Python 3.x, "class Test(object)" and "class Test" are same.
In Python 2.x, "class Test(object)" creates a class with object as parent (called new style class) and "class Test" creates old style class (without object parent). Refer this for more details.

Does Python support Multiple Inheritance?
Unlike Java and like C++, Python supports multiple inheritance. We specify all parent classes as comma separated list in bracket.

```
# Python example to show working of multiple
# inheritance
class Base1(object):
    def __init__(self):
        self.str1 = "Geek1"
        print "Base1"

class Base2(object):
    def __init__(self):
        self.str2 = "Geek2"
        print "Base2"

class Derived(Base1, Base2):
    def __init__(self):

        # Calling constructors of Base1
        # and Base2 classes
        Base1.__init__(self)
        Base2.__init__(self)
        print "Derived"

    def printStrs(self):
        print(self.str1, self.str2)


ob = Derived()
ob.printStrs()
```

Output :

```
Geek1 True E101
```

How to access parent members in a subclass?

1. **Using Parent class name**

```
# Python example to show that base
# class members can be accessed in
# derived class using base class name
class Base(object):

    # Constructor
    def __init__(self, x):
        self.x = x

class Derived(Base):

    # Constructor
    def __init__(self, x, y):
        Base.x = x
        self.y = y

    def printXY(self):

       # print(self.x, self.y) will also work
       print(Base.x, self.y)


# Driver Code
d = Derived(10, 20)
d.printXY()
```

Output :

```
10, 20
```

2. **Using super()**

   We can also access parent class members using super.

```
# Python example to show that base
# class members can be accessed in
# derived class using super()
class Base(object):

    # Constructor
    def __init__(self, x):
        self.x = x

class Derived(Base):

    # Constructor
    def __init__(self, x, y):

        ''' In Python 3.x, "super().__init__(name)"
            also works'''
        super(Derived, self).__init__(x)
        self.y = y

    def printXY(self):

       # Note that Base.x won't work here
       # because super() is used in constructor
       print(self.x, self.y)


# Driver Code
d = Derived(10, 20)
d.printXY()
```

Output :

```
(10, 20)
```

Note that the above two methods are not exactly same. In the next article on inheritance, we will covering following topics.

1) How super works? How accessing a member through super and parent class name are different?

2) How Diamond problem is handled in Python?

Exercise:

Predict the output of following Python programs

```


class X(object):
 def __init__(self,a):
  self.num = a
 def doubleup(self):
  self.num *= 2

class Y(X):
 def __init__(self,a):
  X.__init__(self, a)
 def tripleup(self):
  self.num *= 3

obj = Y(4)
print(obj.num)

obj.doubleup()
print(obj.num)

obj.tripleup()
print(obj.num)
```

```
# Base or Super class
class Person(object):
    def __init__(self, name):
        self.name = name

    def getName(self):
        return self.name

    def isEmployee(self):
        return False

# Inherited or Subclass (Note Person in bracket)
class Employee(Person):
    def __init__(self, name, eid):

        ''' In Python 3.0+, "super().__init__(name)"
            also works'''
        super(Employee, self).__init__(name)
        self.empID = eid

    def isEmployee(self):
        return True

    def getID(self):
        return self.empID

# Driver code
emp = Employee("Geek1", "E101")
print(emp.getName(), emp.isEmployee(), emp.getID())
```

This article is contributed by **Shwetanshu Rohatgi** and **Mayank Rawat**. 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

Class or Static Variables in Python

Class or static variables are shared by all objects. Instance or non-static variables are different for different objects (every object has a copy of it).

For example, let a Computer Science Student be represented by class **CSStudent**. The class may have a static variable whose value is "cse" for all objects. And class may also have non-static members like **name** and **roll**.

In C++ and Java, we can use static keyword to make a variable as class variable. The variables which don't have preceding static keyword are instance variables. See this for Java example and this for C++ example.

The **Python** approach is simple, it doesn't require a static keyword. *All variables which are assigned a value in class declaration are class variables. And variables which are assigned values inside class methods are instance variables.*

```
# Python program to show that the variables with a value
# assigned in class declaration, are class variables

# Class for Computer Science Student
class CSStudent:
    stream = 'cse'                # Class Variable
    def __init__(self,name,roll):
        self.name = name          # Instance Variable
        self.roll = roll          # Instance Variable

# Objects of CSStudent class
a = CSStudent('Geek', 1)
b = CSStudent('Nerd', 2)

print(a.stream)  # prints "cse"
print(b.stream)  # prints "cse"
print(a.name)    # prints "Geek"
print(b.name)    # prints "Nerd"
print(a.roll)    # prints "1"
print(b.roll)    # prints "2"

# Class variables can be accessed using class
# name also
print(CSStudent.stream) # prints "cse"
```

Output:

```
cse
cse
Geek
Nerd
1
2
cse
```

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.

Changing Class Members in Python

In the previous fact, we have seen that Python doesn't have static keyword. All variables that are assigned a value in class declaration are class variables

We should be careful when changing value of class variable. If we try to change class variable using object, a new instance (or non-static) variable for that particular object is created and this variable shadows the class variables. Below is Python program to demonstrate the same.

```
# Class for Computer Science Student
class CSStudent:
    stream = 'cse'     # Class Variable
    def __init__(self, name, roll):
        self.name = name
        self.roll = roll

# Driver program to test the functionality
# Creating objects of CSStudent class
a = CSStudent("Geek", 1)
b = CSStudent("Nerd", 2)

print "Initially"
print "a.stream =", a.stream
print "b.stream =", b.stream

# This thing doesn't change class(static) variable
# Instead creates instance variable for the object
# 'a' that shadows class member.
a.stream = "ece"

print "\nAfter changing a.stream"
print "a.stream =", a.stream
print "b.stream =", b.stream
```

Output:

```
Initially
a.stream = cse
b.stream = cse

After changing a.stream
a.stream = ece
b.stream = cse
```

We should change class variables using class name only.

```
# Program to show how to make changes to the
# class variable in Python

# Class for Computer Science Student
class CSStudent:
    stream = 'cse'     # Class Variable
    def __init__(self, name, roll):
        self.name = name
        self.roll = roll

# New object for further implementation
a = CSStudent("check", 3)
print "a.tream =", a.stream

# Correct way to change the value of class variable
CSStudent.stream = "mec"
print "\nClass variable changes to mec"

# New object for further implementation
b = CSStudent("carter", 4)

print "\nValue of variable steam for each object"
print "a.stream =", a.stream
print "b.stream =", b.stream
```

Output:

```
a.tream = cse

Class variable changes to mec

Value of variable steam for each object
a.stream = mec
b.stream = mec
```

This article is contributed by **Nikhil Kumar Singh**. 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

How to input multiple values from user in one line in Python?

For instance, in C we can do something like this:

```
// Reads two values in one line
scanf("%d %d", &x, &y)
```

One solution is to use raw_input() two times.

```
x, y = raw_input(),  raw_input()
```

Another solution is to use split()

```
x, y = raw_input().split()
```

Note that we don't have to explicitly specify split(' ') because split() uses any whitespace characters as delimiter as default.

One thing to note in above Python code is, both x and y would be of string. We can convert them to int using another line

```
x, y = [int(x), int(y)]

# We can also use  list comprehension
x, y = [int(x) for x in [x, y]]
```

Below is complete one line code to read two integer variables from standard input using split and list comprehension

```
# Reads two numbers from input and typecasts them to int using
# list comprehension
x, y = [int(x) for x in raw_input().split()]
```

Note that in Python 3, we use input() in place of raw_input().

This article is contributed by **Abhishek Shukla**. 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

Python | Set 6 (Command Line and Variable Arguments)

Previous Python Articles (Set 1 | Set 2 | Set 3 | Set 4 | Set 5)

This article is focused on command line arguments as well as variable arguments (args and kwargs) for the functions in python.

Command Line Arguments

Till now, we have taken input in python using raw_input() or input() [for integers]. There is another method that uses command line arguments. The command line arguments must be given whenever we want to give the input before the start of the script, while on the other hand, raw_input() is used to get the input while the python program / script is running.
For example, in the UNIX environment, the arguments '-a' and '-l' for the 'ls' command give different results.

The command line arguments in python can be processed by using either 'sys' module or 'argparse' module.

```
# Python code to demonstrate the use of 'sys' module
# for command line arguments

import sys

# command line arguments are stored in the form
# of list in sys.argv
argumentList = sys.argv
print argumentList

# Print the name of file
print sys.argv[0]

# Print the first argument after the name of file
print sys.argv[1]
```

OUTPUT :

```
['program1.py', 'test', '1']
program1.py
test
```

NOTE : The above code runs only on command line. We need to fire the below command given that the program is saved as program1.py

python program1.py test 123

Please note the following points about the above program :

- The sys.argv takes the command line arguments in the form of a list.
- The first element in the list is the name of the file.
- The arguments always come in the form of a string even if we type an integer in the argument list. We need to use int() function to convert the string to integer.

We can use the command line arguments to write the programs which do frequently used tasks. For example, we need to find factorial many times. We can keep the following program in a file named factorial.py in our computer and get the output by simply writing the command for getting the factorial of a number, say 5.

python factorial.py 5

```
import sys
from math import factorial

print factorial(int(sys.argv[1]))
```

Variable Arguments

args(*) and kwargs(**)

Both 'args' and 'kwargs' are used to get arbitrary number of arguments in a function.

args will give us all the function parameters in the form of a list and kwargs will give us all the keyword arguments except for those corresponding to formal parameter as dictionary.

```
# Python program to illustrate the use of args which
# multiplies all the values given to the function as parameter

def multiplyAll(*values):
    mul = 1

    # values(args) will be in the form of tuple
    print values
    print "Type = ", type(values)


    # Multiplying the all the parameters and return the result
    for i in values:
        mul = mul * i

    return mul


# Driver program to test above function

# Multiply two numbers using above function
ans = multiplyAll(1,2)
print "The multiplication of 1 and 2 is ", ans

# Multiply 5 numbers using above function
ans = multiplyAll(3, 4, 5, 6, 7)
print "The multiplication of 3 to 7 is ", ans
```

OUTPUT :

```
(1, 2)
Type =
The multiplication of 1 and 2 is  2
(3, 4, 5, 6, 7)
Type =
The multiplication of 3 to 7 is  2520
```

Note that **args** are denoted by using a single star and **kwargs** are denoted by two stars before the formal parameters in the function.

```
# Program to illustrate the use of kwargs in Python

# Function that print the details of a student
# The number of details per student may vary
def printDetails(**details):

    # Variable 'details' contains the details in the
    # form of dictionary
    print "Parameter details contains"
    print details
    print "Type = ", type(details)

    # Print first name
    print "First Name = ", details['firstName']

    # Print the department of student
    print "Department = ", details['department']
    print "" # Extra line break


# Driver program to test above function

# Calling the function with three arguments
printDetails(firstName = "Nikhil",
             rollNumber = "007",
             department = "CSE")

# Calling the function with two arguments
printDetails(firstName = "Abhay",
             department = "ECE")
```

Output:

```
Parameter details contains
{'department': 'CSE', 'rollNumber': '007', 'firstName': 'Nikhil'}
Type =
First Name =  Nikhil
Department =  CSE

Parameter details contains
{'department': 'ECE', 'firstName': 'Abhay'}
Type =
First Name =  Abhay
Department =  ECE
```

Please **note** that if you are using both args and kwargs in a function then the args parameter must precede before the kwarg parameters.
Example :

```
# Function containing both args and kwargs
def cheeseshop(kind, *arguments, **keywords):
    print "-- Do you have any", kind, "?"
    print "-- I'm sorry, we're all out of", kind
    for arg in arguments:
        print arg
    print "-" * 40
    keys = sorted(keywords.keys())
    for kw in keys:
        print kw, ":", keywords[kw]

# Driver program to test above function
cheeseshop("Burger", "It's very funny, sir.",
           "It's really very, VERY funny, sir.",
           shopkeeper='Michael Palin',
           client="John Cleese",
           sketch="Cheese Shop Sketch")
```

This article is contributed by Nikhil Kumar Singh

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

GATE CS Notes (According to Official GATE 2017 Syllabus)

GATE CS Corner

See Placement Course for placement preparation, GATE Corner for GATE CS Preparation and Quiz Corner for all Quizzes on GeeksQuiz.
Category: Python Articles

trunc() in Python

Truncate in Python

There are many built-in modules in python. Out of these module there is one interesting module known as math module which have several functions in it like, ceil, floor, truncate, factorial, fabs, etc.

Out of these functions there is an interesting function called **truncate** which behaves as a ceiling function for negative number and floor function for positive number.

In case of positive number

```
# Python program to show output of floor(), ceil()
# truncate() for a positive number.
import math
print math.floor(3.5) # floor
print math.trunc(3.5) # work as floor
print math.ceil(3.5)  # ceil
```

Output:

```
3.0
3
4.0
```

In case of negative number

```
# Python program to show output of floor(), ceil()
# truncate() for a negative number.
import math
print math.floor(-3.5) # floor
print math.trunc(-3.5) # work as ceil
print math.ceil(-3.5)  # floor
```

Output:

```
-4.0
-3
-3.0
```

This is because the ceiling function is used to round up, i.e., towards positive infinity and floor function is used to round down, i.e., towards negative infinity.

But the truncate function is used to round up or down towards zero.

Diagrammatic representation of truncate function:-

inf

This article is contributed by **Arpit Agarwal**. 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

Print Single and Multiple variable in Python

Consider below two Python code snippets in **Python 2.x**.

```
# Code 1
print 1
# Output: 1
```

```
# Code 2
print(1)
# Output: 1
```

*There is no difference between code 1 and code 2 in case of single variable in **Python 2.X**, but in case of multiple variables, variable with brackets -() is treated as “tuple”.*

For multiple variable:

- “print variable” prints the variables without any brackets ‘()’ and splitted by a space
- “print(variable)” prints the variables with brackets ‘()’ and splitted by a coma ‘,’ so it’s **treated as a tuple.**

Examples-

```
# Code 3
print 1, 2
# Output: 1 2
```

```
# Code 4
print (1, 2)
# Output: (1, 2)
```

Note:

In **Python 3.0**, the print statement is changed to print() function. Below are equivalent codes in Python 3.0.

```
# Equivalent codes in Python 3.0
# (Produces same output)

# Code 1:
print(1)
# Output: 1

# Code 2 :
print((1))
# Output: 1

# Code 3:
print(1, 2)
# Output: 1 2

# Code 4:
print((1, 2))
# Output: (1, 2)
```

This article is contributed by **Arpit Agarwal**. 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
Python

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:

```
The Zen of Python, by Tim Peters

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)
```

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)
```

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
```

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)
```

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)
```

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")
```

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])
```

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
```

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

Technical Scripter
Python

Generate all permutation of a set in Python

Permutation is an arrangement of objects in a specific order. Order of arrangement of object is very important. The number of permutations on a set of n elements is given by n!. For example, there are 2! = 2*1 = 2 permutations of {1, 2}, namely {1, 2} and {2, 1}, and 3! = 3*2*1 = 6 permutations of {1, 2, 3}, namely {1, 2, 3}, {1, 3, 2}, {2, 1, 3}, {2, 3, 1}, {3, 1, 2} and {3, 2, 1}.

Method 1 (Backtracking)

We can use the backtracking based recursive solution discussed here.

Method 2

The idea is to one by one extract all elements, place them at first position and recur for remaining list.

```
# Python function to print permutations of a given list
def permutation(lst):

    # If lst is empty then there are no permutations
    if len(lst) == 0:
        return []

    # If there is only one element in lst then, only
    # one permuatation is possible
    if len(lst) == 1:
        return [lst]
    # Find the permutations for lst if there are
    # more than 1 characters

    l = [] # empty list that will store current permutation

    # Iterate the input(lst) and calculate the permutation
    for i in range(len(lst)):
       m = lst[i]

       # Extract lst[i] or m from the list.  remLst is
       # remaining list
       remLst = lst[:i] + lst[i+1:]

       # Generating all permutations where m is first
       # element
       for p in permutation(remLst):
           l.append([m] + p)
    return l


# Driver program to test above function
data = list('123')
for p in permutation(data):
    print p
```

Output:

```
['1', '2', '3']
['1', '3', '2']
['2', '1', '3']
['2', '3', '1']
['3', '1', '2']
['3', '2', '1']
```

Method 3 (Direct Function)
We can do it by simply using the built-in permutation function in itertools library. It is the shortest technique to find the permutation.

```
from itertools import permutations
l = list(permutations(range(1, 4)))
print l
```

Output:

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

This article is contributed by **Arpit Agarwal**. 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

Algorithm
permutation
Python

How to swap two variables in one line in C/C++, Python and Java?

We have discussed different approaches to swap two integers without the temporary variable. How to swap in a single line without using library function?

Python: In Python, there is a simple and syntactically neat construct to swap variables, we just need to write "x, y = y, x".

C/C++: Below is one generally provided classical solution

```
// Swap using bitwise XOR (Wrong Solution in C/C++)
x ^= y ^= x ^= y;
```

The above solution is wrong in C/C++ as it causes undefined behaviour (compiler is free to behave in any way). The reason is, modifying a variable more than once in an expression causes undefined behaviour if there is no sequence point between the modifications.

However, we can use comma to introduce sequence points. So the modified solution is

```
// Swap using bitwise XOR (Correct Solution in C/C++)
// sequence point introduced using comma.
(x ^= y), (y ^= x), (x ^= y);
```

Java: In Java, rules for subexpression evaluations are clearly defined. The left hand operand is always evaluated before right hand operand (See this for more details). In Java, the expression "x ^= y ^= x ^= y;" doesn't produce the correct result according to Java rules. It makes x = 0. However, we can use "x = x ^ y ^ (y = x);" Note the expressions are evaluated from left to right. If x = 5 and y = 10 initially, the expression is equivalent to "x = 5 ^ 10 ^ (y = 5);". Note that we can't use this in C/C++ as in C/C++, it is not defined whether left operand or right operand is executed for any operator (See this for more details)

C/C++

```
// C/C++ program to swap two variables in single line
```

```
#include <stdio.h>
int main()
{
    int x = 5, y = 10;
    (x ^= y), (y ^= x), (x ^= y);
    printf("After Swapping values of x and y are %d %d",
            x, y);
    return 0;
}
```

Java

```
// Java program to swap two variables in single line
class GFG
{
   public static void main (String[] args)
   {
      int x = 5, y = 10;
      x = x ^ y ^ (y = x);
      System.out.println("After Swapping values of x and y are "
                      + x + " " + y);
   }
}
```

Python

```
# Python program to swap two variables in single line
x = 5
y = 10
x, y = y, x
print "After Swapping values of x and y are", x, y
```

Output:

```
After Swapping values of x and y are 10 5
```

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

C/C++ Puzzles
Java
Python
XOR

str() vs repr() in Python

str() and repr() both are used to get a string representation of object.

1. **Example of str():**

   ```
   s = 'Hello, Geeks.'
   print str(s)
   print str(2.0/11.0)
   ```

   Output:

   ```
   Hello, Geeks.
   0.181818181818
   ```

.

2. **Example of repr():**

   ```
   s = 'Hello, Geeks.'
   print repr(s)
   print repr(2.0/11.0)
   ```

   Output:

   ```
   'Hello, Geeks.'
   0.18181818181818182
   ```

From above output, we can see if we print string using repr() function then it prints with a pair of quotes and if we calculate a value we get more precise value than str() function.

Following are differences:

- ***str() is used for creating output for end user while repr() is mainly used for debugging and development. repr's goal is to be unambiguous and str's is to be***

readable. For example, if we suspect a float has a small rounding error, repr will show us while str may not.

- repr() compute the “official” string representation of an object (a representation that has all information about the abject) and str() is used to compute the “informal” string representation of an object (a representation that is useful for printing the object).
- The print statement and str() built-in function uses __str__ to display the string representation of the object while the repr() built-in function uses __repr__ to display the object.

Let understand this by an example:-

```
import datetime
today = datetime.datetime.now()

# Prints readable format for date-time object
print str(today)

# prints the official format of date-time object
print repr(today)
```

Output:

```
2016-02-22 19:32:04.078030
datetime.datetime(2016, 2, 22, 19, 32, 4, 78030)
```

str() displays today’s date in a way that the user can understand the date and time.

repr() prints “official” representation of a date-time object (means using the “official” string representation we can reconstruct the object).

How to make them work for our own defined classes?
A user defined class should also have a __repr__ if we need detailed information for debugging. And if we think it would be useful to have a string version for users, we create a __str__ function.

```
# Python program to demonstrate writing of __repr__ and
# __str__ for user defined classes

# A user defined class to represent Complex numbers
class Complex:

    # Constructor
    def __init__(self, real, imag):
       self.real = real
       self.imag = imag

    # For call to repr(). Prints object's information
    def __repr__(self):
       return 'Rational(%s, %s)' % (self.real, self.imag)

    # For call to str(). Prints readable form
    def __str__(self):
       return '%s + i%s' % (self.real, self.imag)


# Driver program to test above
t = Complex(10, 20)

print str(t)  # Same as "print t"
print repr(t)
```

Output:

```
10 + i20
Rational(10, 20)
```

This article is contributed by **Arpit Agarwal.** 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

Python
Python

Command Line Interface Programming in Python

This article discusses how you can create a CLI for your python programs using an example in which we make a basic “text file manager”.
Let us discuss some basics first.

What is a Command Line Interface(CLI)?

A command-line interface or command language interpreter (CLI), also known as command-line user interface, console user interface, and character user interface (CUI), is a means of interacting with a computer program where the user (or client) issues commands to the program in the form of successive lines of text (command lines).(Wiki)

Advantages of CLI:

- Requires fewer resources
- Concise and powerful

- Expert-friendly
- Easier to automate via scripting

Why to use CLI in your python program?

- Having even just a very basic command-line interface (CLI) for your program can make everyone's life easier for modifying parameters, including programmers, but also non-programmers.
- A CLI for your program can also make it easier to automate running and modifying variables within your program, for when you want to run your program with a cronjob or maybe an os.system call.

Now, let us start making our "Text file manager". Here, we will be using a built-in python library called **Argparse**.

About Argparse:

- It makes it easy to write user-friendly command-line interfaces.
- The program defines what arguments it requires, and argparse will figure out how to parse those out of sys.argv.
- The argparse module also automatically generates help and usage messages and issues errors when users give the program invalid arguments.

Ok let's start with a really basic program to get a feel of what argparse does.

```
# importing required modules
import argparse

# create a parser object
parser = argparse.ArgumentParser(description = "An addition program")

# add argument
parser.add_argument("add", nargs = '*', metavar = "num", type = int,
                    help = "All the numbers separated by spaces will be added.")

# parse the arguments from standard input
args = parser.parse_args()

# check if add argument has any input data.
# If it has, then print sum of the given numbers
if len(args.add) != 0:
    print(sum(args.add))
```

Let us go through some important points related to above program:

- First of all, we imported the argparse module.
- Then, created a **ArgumentParser** object and also provided a description of our program.
- Now, we can fill up our parser object with information by adding arguments. In this example, we created an argument **add.** A lot of arguments can be passed to the **add_argument** function. Here I explain the ones I have used in above example:
  argument 1: ("add") It is nothing but the name of the argument. We will use this name to access the **add** arguments by typing **args.add**.
  argument 2: (nargs = '*') The number of command-line arguments that should be consumed. Specifying it to '*' means it can be any no. of arguments i.e, from 0 to anything.
  argument 3: (metavar = 'num') A name for the argument in usage messages.
  argument 4: (type = int) The type to which the command-line argument should be converted. It is str by default.
  argument 5: (help) A brief description of what the argument does.
- Once we have specified all the arguments, it is the time to parse the arguments from the standard command line input stream. For this, we use parse_args() function.
- Now, one can simply check if the input has invoked a specific argument. Here, we check the length of args.add to check if there is any data received from input. Note that values of an argument are obtained as a **list.**
- There are two types of arguments: Positional and Optional.
  Positional ones are those which do not need any specification to be invoked. Whereas, optional arguments need to be specified by their name first (which starts with '–' sign, '-' is also a shorthand.)
- One can always use –help or -h optional argument to see the help message.
  Here is an example (The python script has been saved as add.py):
  2
- Now, let us have a look at another example where our positional argument **add** is invoked.
  3
- One more special feature worth mentioning is how argparse issues errors when users give the program invalid arguments.
  4

So, this was a basic example so that you can get comfortable with argparse and CLI concept. Now, let us move on to our **"Text file manager"** program.

```
# importing the required modules
import os
import argparse

# error messages
INVALID_FILETYPE_MSG = "Error: Invalid file format. %s must be a .txt file."
INVALID_PATH_MSG = "Error: Invalid file path/name. Path %s does not exist."


def validate_file(file_name):
 '''
 validate file name and path.
 '''
 if not valid_path(file_name):
  print(INVALID_PATH_MSG%(file_name))
  quit()
 elif not valid_filetype(file_name):
  print(INVALID_FILETYPE_MSG%(file_name))
  quit()
 return

def valid_filetype(file_name):
 # validate file type
 return file_name.endswith('.txt')

def valid_path(path):
 # validate file path
 return os.path.exists(path)



def read(args):
 # get the file name/path
 file_name = args.read[0]

 # validate the file name/path
 validate_file(file_name)

 # read and print the file content
 with open(file_name, 'r') as f:
  print(f.read())


def show(args):
 # get path to directory
 dir_path = args.show[0]

 # validate path
 if not valid_path(dir_path):
  print("Error: No such directory found.")
  exit()

 # get text files in directory
 files = [f for f in os.listdir(dir_path) if valid_filetype(f)]
 print("{} text files found.".format(len(files)))
 print('\n'.join(f for f in files))


def delete(args):
 # get the file name/path
 file_name = args.delete[0]

 # validate the file name/path
 validate_file(file_name)

 # delete the file
 os.remove(file_name)
 print("Successfully deleted {}.".format(file_name))


def copy(args):
 # file to be copied
 file1 = args.copy[0]
 # file to copy upon
 file2 = args.copy[1]

 # validate the file to be copied
 validate_file(file1)

 # validate the type of file 2
 if not valid_filetype(file2):
  print(INVALID_FILETYPE_MSG%(file2))
  exit()

 # copy file1 to file2
 with open(file1, 'r') as f1:
  with open(file2, 'w') as f2:
   f2.write(f1.read())
 print("Successfully copied {} to {}.".format(file1, file2))


def rename(args):
 # old file name
 old_filename = args.rename[0]
 # new file name
 new_filename = args.rename[1]

 # validate the file to be renamed
 validate_file(old_filename)

 # validate the type of new file name
 if not valid_filetype(new_filename):
  print(INVALID_FILETYPE_MSG%(new_filename))
  exit()

 # renaming
 os.rename(old_filename, new_filename)
 print("Successfully renamed {} to {}.".format(old_filename, new_filename))
def main():
 # create parser object
 parser = argparse.ArgumentParser(description = "A text file manager!")

 # defining arguments for parser object
 parser.add_argument("-r", "--read", type = str, nargs = 1,
      metavar = "file_name", default = None,
      help = "Opens and reads the specified text file.")

 parser.add_argument("-s", "--show", type = str, nargs = 1,
      metavar = "path", default = None,
      help = "Shows all the text files on specified directory path.\
      Type '.' for current directory.")

 parser.add_argument("-d", "--delete", type = str, nargs = 1,
      metavar = "file_name", default = None,
      help = "Deletes the specified text file.")

 parser.add_argument("-c", "--copy", type = str, nargs = 2,
      metavar = ('file1','file2'), help = "Copy file1 contents to \
      file2 Warning: file2 will get overwritten.")

 parser.add_argument("--rename", type = str, nargs = 2,
      metavar = ('old_name','new_name'),
      help = "Renames the specified file to a new name.")

 # parse the arguments from standard input
 args = parser.parse_args()

 # calling functions depending on type of argument
 if args.read != None:
  read(args)
 elif args.show != None:
  show(args)
 elif args.delete !=None:
  delete(args)
 elif args.copy != None:
  copy(args)
 elif args.rename != None:
  rename(args)


if __name__ == "__main__":
 # calling the main function
 main()
```

After the previous example, the above code seems self explanatory.
All we did was to add a set of arguments for our file manager program. Note that all these arguments are optional arguments. So, we use some if-elif statements to match the command line input with correct argument type function so that query could be processed.
Here are a few screenshots which describe the usage of above program:

- Help message (The python script has been saved as tfmanager.py):
  5

- Here are examples of operations using the text file manager:

9

So, this was an example of a simple CLI python program which we made. Many complex CLIs could be easily created by the Argparse module. I hope that these examples will give you a head start in this area.

This article is contributed by **Nikhil Kumar**. 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

How to check if a string is a valid keyword in Python?

Defining a Keyword

In programming, a keyword is a "**reserved word**" by the language which **convey a special meaning to the interpreter**. It may be a command or a parameter. Keywords **cannot be used as a variable name** in the program snippet.

Keywords in Python: Python language also reserves some of keywords that convey special meaning. Knowledge of these is necessary part of learning this language. Below is list of keywords registered by python .

Keyword

How to check if a string is keyword?

Python in its language defines an inbuilt module "**keyword**" which handles certain operations related to keywords. A function "**iskeyword()**" checks if a string is keyword or not. Returns **true if a string is keyword, else returns false**.

```
#Python code to demonstrate working of iskeyword()

# importing "keyword" for keyword operations
import keyword

# initializing strings for testing
s = "for"
s1 = "geeksforgeeks"
s2 = "elif"
s3 = "elseif"
s4 = "nikhil"
s5 = "assert"
s6 = "shambhavi"
s7 = "True"
s8 = "False"
s9 = "akshat"
s10 = "akash"
s11 = "break"
s12 = "ashty"
s13 = "lambda"
s14 = "suman"
s15 = "try"
s16 = "vaishnavi"
```

```
# checking which are keywords
if keyword.iskeyword(s):
       print ( s + " is a python keyword")
else :  print ( s + " is not a python keyword")

if keyword.iskeyword(s1):
       print ( s1 + " is a python keyword")
else :  print ( s1 + " is not a python keyword")

if keyword.iskeyword(s2):
       print ( s2 + " is a python keyword")
else :  print ( s2 + " is not a python keyword")

if keyword.iskeyword(s3):
       print ( s3 + " is a python keyword")
else :  print ( s3 + " is not a python keyword")

if keyword.iskeyword(s4):
       print ( s4 + " is a python keyword")
else :  print ( s4 + " is not a python keyword")

if keyword.iskeyword(s5):
       print ( s5 + " is a python keyword")
else :  print ( s5 + " is not a python keyword")

if keyword.iskeyword(s6):
       print ( s6 + " is a python keyword")
else :  print ( s6 + " is not a python keyword")

if keyword.iskeyword(s7):
       print ( s7 + " is a python keyword")
else :  print ( s7 + " is not a python keyword")

if keyword.iskeyword(s8):
       print ( s8 + " is a python keyword")
else :  print ( s8 + " is not a python keyword")

if keyword.iskeyword(s9):
       print ( s9 + " is a python keyword")
else :  print ( s9 + " is not a python keyword")

if keyword.iskeyword(s10):
       print ( s10 + " is a python keyword")
else :  print ( s10 + " is not a python keyword")

if keyword.iskeyword(s11):
       print ( s11 + " is a python keyword")
else :  print ( s11 + " is not a python keyword")

if keyword.iskeyword(s12):
       print ( s12 + " is a python keyword")
else :  print ( s12 + " is not a python keyword")

if keyword.iskeyword(s13):
       print ( s13 + " is a python keyword")
else :  print ( s13 + " is not a python keyword")

if keyword.iskeyword(s14):
       print ( s14 + " is a python keyword")
else :  print ( s14 + " is not a python keyword")

if keyword.iskeyword(s15):
       print ( s15 + " is a python keyword")
else :  print ( s15 + " is not a python keyword")

if keyword.iskeyword(s16):
       print ( s16 + " is a python keyword")
else :  print ( s16 + " is not a python keyword")
```

Output:

```
for is a python keyword
geeksforgeeks is not a python keyword
elif is a python keyword
elseif is not a python keyword
nikhil is not a python keyword
assert is a python keyword
shambhavi is not a python keyword
True is a python keyword
False is a python keyword
akshat is not a python keyword
akash is not a python keyword
break is a python keyword
ashty is not a python keyword
lambda is a python keyword
suman is not a python keyword
try is a python keyword
vaishnavi is not a python keyword
```

How to print list of all keywords

Sometimes, remembering all the keywords can be a difficult task while assigning variable names. Hence a function “**kwlist()**” is provided in “keyword” module which **prints all the 33 python keywords**.

```
#Python code to demonstrate working of iskeyword()

# importing "keyword" for keyword operations
import keyword

# printing all keywords at once using "kwlist()"
print ("The list of keywords is : ")
print (keyword.kwlist)
```

Output:

```
The list of keywords is :
['False', 'None', 'True', 'and', 'as', 'assert', 'break', 'class',
'continue', 'def', 'del', 'elif', 'else', 'except', 'finally',
'for', 'from', 'global', 'if', 'import', 'in', 'is', 'lambda',
'nonlocal', 'not', 'or', 'pass', 'raise', 'return',
'try', 'while', 'with', 'yield']
```

Next Articles:

- Keywords in Python | Set 1
- Keywords in Python | Set 2

This article is contributed by **Manjeet Singh(S.Nandini)**. 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.

Python | sep parameter in print()

The separator between the arguments to print() function in Python is space by default (softspace feature) , which can be modified and can be made to any character, integer or string as per our choice. The ‘sep’ parameter is used to achieve the same, it is found only in python 3.x or later. It is also used for formatting the output strings.

Examples:

```
#code for disabling the softspace feature
print('G','F','G', sep='')

#for formatting a date
print('09','12','2016', sep='-')

#another example
print('pratik','geeksforgeeks', sep='@')
```

Output:

```

GFG
09-12-2016
pratik@geeksforgeeks
```

The sep parameter when used with end parameter it produces awesome results. Some examples by combining the sep and end parameter.

```
print('G','F', sep='', end='')
print('G')
#n provides new line after printing the year
print('09','12', sep='-', end='-2016n')

print('prtk','agarwal', sep='', end='@')
print('geeksforgeeks')
```

Output:

```
GFG
09-12-2016
prtkagarwal@geeksforgeeks
```

Note: Please change the language from Python to Python 3 in the online ide.

Go to your interactive python ide by typing python in your cmd (windows) or terminal (linux)

```
#import the below module and see what happens
import antigravity
```

```
#NOTE - it wont work on online ide
```

This article is contributed by **Pratik Agarwal**. 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

JSON Formatting in Python

Javascript Object Notation abbreviated as JSON is a light-weight data interchange format. It Encode Python objects as JSON strings, and decode JSON strings into Python objects .

- Many of the APIs like Github. send their results in this format. JSON is probably most widely used for communicating between the web server and client in an AJAX application, but is not limited to that problem domain.
- For example, if you are trying to build an exciting project like this, we need to format the JSON output to render necessary results. So lets dive into **json** module which Python offers for formatting JSON output.

Functions

- **json.dump(obj, fileObj)**: Serializes *obj* as a JSON formatted stream to *fileObj*.
- **json.dumps(obj)** : Serializes *obj* as JSON formatted string.
- **json.load(JSONfile)** : De-serializes *JSONfile* to a Python object.
- **json.loads(JSONfile)** : De-serializes *JSONfile*(type: string) to a Python object.

Classes

- **JSONEncoder:** An encoder class to convert Python objects to JSON format.
- **JSONDecoder:** A decoder class to convert JSON format file into Python obj.

The conversions are based on this conversion table.

Implementation

Encoding

We will be using dump(), dumps() and JSON.Encoder class.

```
#Code will run in Python 3

from io import StringIO
import json

fileObj = StringIO()
json.dump(["Hello", "Geeks"], fileObj)
print("Using json.dump(): "+str(fileObj.getvalue()))

class TypeEncoder(json.JSONEncoder):
    def default(self, obj):
        if isinstance(obj, type):
            return str(obj)

print("Using json.dumps(): "+str(json.dumps(type(str), cls=TypeEncoder)))
print("Using json.JSONEncoder().encode"+
      str(TypeEncoder().encode(type(list))))
print("Using json.JSONEncoder().iterencode"+
      str(list(TypeEncoder().iterencode(type(dict)))))
```

Output:

```
Using json.dump(): ["Hello", "Geeks"]
Using json.dumps(): ""
Using json.JSONEncoder().encode""
Using json.JSONEncoder().iterencode['""']
```

Decoding

We will be using load(), loads() and JSON.Decoder class.

```
#Code will run in Python 3

from io import StringIO
import json

fileObj = StringIO('["Geeks for Geeks"]')
print("Using json.load(): "+str(json.load(fileObj)))
print("Using json.loads(): "+str(json.loads('{"Geeks": 1, "for": 2, "Geeks": 3}')))
print("Using json.JSONDecoder().decode(): " +
 str(json.JSONDecoder().decode('{"Geeks": 1, "for": 2, "Geeks": 3}')))
print("Using json.JSONDecoder().raw_decode(): " +
 str(json.JSONDecoder().raw_decode('{"Geeks": 1, "for": 2, "Geeks": 3}')))
```

Output:

```
Using json.load(): ['Geeks for Geeks']
Using json.loads(): {'for': 2, 'Geeks': 3}
Using json.JSONDecoder().decode(): {'for': 2, 'Geeks': 3}
Using json.JSONDecoder().raw_decode(): ({'for': 2, 'Geeks': 3}, 34)
```

Reference:

- https://docs.python.org/3/library/json.html

This article is contributed by **Sri Sanketh Uppalapati**. 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

Quine in Python

Quine is a program which takes no input but outputs a copy of its own code. We have discussed quine in C.

The shortest possible quine in python is just a single line of code!

```
_='_=%r;print _%%_';print _%_
```

In case of Python3.x

```
_='_=%r;print (_%%_)';print (_%_)
```

Explanation:
The above code is a classic use of string formatting. Firstly, we are defining a variable _ and assigning it '_=%r;print _%%_'. Secondly, we are printing _%_. Here we are printing _ with _ as input to string formatting. So *%r* in _ gets the value of _. You can even use *%s* instead of *%r*. We used double *%* in '_=%r;print _%%_' to escape *%*.

But you may say that the below code is the smallest, right!

```
print open(__file__).read()
```

You need to note that it is indeed the smallest python program that can print its own source code but it is not a quine because a quine should not use *open()* function to print out its source code.

This article is contributed by **Sri Sanketh Uppalapati**. 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

Python | Set 5 (Exception Handling)

We have explored basic python till now from Set 1 to 4 (Set 1 | Set 2 | Set 3 | Set 4).
If you are a beginner, then you might be getting a lot of Tracebacks till now. The tracebacks are generated due to runtime errors.

Like other languages, python also provides the runtime errors via exception handling method with the help of try-except. Some of the standard exceptions which are most frequent include IndexError, ImportError, IOError, ZeroDivisionError, TypeError.

Exception is the base class for all the exceptions in python. You can check the exception hierarchy here.

Let us try to access the array element whose index is out of bound and handle the corresponding exception.

```
# Python program to handle simple runtime error

a = [1, 2, 3]
try:
    print "Second element = %d" %(a[1])

    # Throws error since there are only 3 elements in array
    print "Fourth element = %d" %(a[3])

except IndexError:
    print "An error occurred"
```

Output:

```
Second element = 2
An error occurred
```

A try statement can have more than one except clause, to specify handlers for different exceptions. Please note that at most one handler will be executed.

```
# Program to handle multiple errors with one except statement
try :
    a = 3
    if a < 4 :

        # throws ZeroDivisionError for a = 3
        b = a/(a-3)

    # throws NameError if a >= 4
    print "Value of b = ", b

# note that braces () are necessary here for multiple exceptions
except(ZeroDivisionError, NameError):
    print "\nError Occurred and Handled"
```

Output:

```
Error Occurred and Handled
```

If you change the value of ‘a’ to greater than or equal to 4, the the output will be

```
Value of b =
Error Occurred and Handled
```

The output above is so because as soon as python tries to access the value of b, NameError occurs.

Else Clause:
In python, you can also use else clause on try-except block which must be present after all the except clauses. The code enters the else block only if the try clause does not raise an exception.

```
# Program to depict else clause with try-except

# Function which returns a/b
def AbyB(a , b):
    try:
        c = ((a+b) / (a-b))
    except ZeroDivisionError:
        print "a/b result in 0"
    else:
        print c

# Driver program to test above function
AbyB(2.0, 3.0)
AbyB(3.0, 3.0)
```

The output for above program will be :

```
-5.0
a/b result in 0
```

Raising Exception:
The raise statement allows the programmer to force a specific exception to occur. The sole argument in raise indicates the exception to be raised. This must be either an exception instance or an exception class (a class that derives from Exception).

```
# Program to depict Raising Exception

try:
    raise NameError("Hi there")  # Raise Error
except NameError:
    print "An exception"
    raise  # To determine whether the exception was raised or not
```

The output of the above code will simply line printed as “An exception” but a Runtime error will also occur in the last due to raise statement in the last line. So, the output on your command line will look like

```
Traceback (most recent call last):
  File "003dff3d748c75816b7f849be98b91b8.py", line 4, in
    raise NameError("Hi there") # Raise Error
NameError: Hi there
```

This article is contributed by Nikhil Kumar Singh(nickzuck_007)

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

GATE CS Notes (According to Official GATE 2017 Syllabus)

User-defined Exceptions in Python with Examples

Prerequisite- This article is an extension to Exception Handling.

Python throws errors and exceptions, when there is a code gone wrong, which may cause program to stop abruptly. Python also provides exception handling method with the help of try-except. Some of the standard exceptions which are most frequent include IndexError, ImportError, IOError, ZeroDivisionError, TypeError and FileNotFoundError. A user can create his own error using exception class.

Creating User-defined Exception

Programmers may name their own exceptions by creating a new exception class. Exceptions need to be derived from the Exception class, either directly or indirectly. Although not mandatory, most of the exceptions are named as names that end in **"Error"** similar to naming of the standard exceptions in python. For example:

```
# A python program to create user-defined exception

# class MyError is derived from super class Exception
class MyError(Exception):

    # Constructor or Initializer
    def __init__(self, value):
        self.value = value

    # __str__ is to print() the value
    def __str__(self):
        return(repr(self.value))

try:
    raise(MyError(3*2))

# Value of Exception is stored in error
except MyError as error:
    print('A New Exception occured: ',error.value)
```

Output:

```
('A New Exception occured: ', 6)
```

Knowing all about Exception Class

To know more about about class Exception, run the code below

```
help(Exception)
```

Deriving Error from Super Class Exception

Super class Exceptions are created when a module needs to handle several distinct errors. One of the common way of doing this is to create a base class for exceptions defined by that module. Further, various subclasses are defined to create specific exception classes for different error conditions.

```
# class Error is derived from super class Exception
class Error(Exception):

    # Error is derived class for Exception, but
    # Base class for exceptions in this module
    pass

class TransitionError(Error):

    # Raised when an operation attempts a state
    # transition that's not allowed.
    def __init__(self, prev, nex, msg):
        self.prev = prev
        self.next = nex

        # Error message thrown is saved in msg
        self.msg = msg
try:
    raise(TransitionError(2,3*2,"Not Allowed"))

# Value of Exception is stored in error
except TransitionError as error:
    print('Exception occured: ',error.msg)
```

Output:

```
('Exception occured: ', 'Not Allowed')
```

How to use standard Exceptions as base class?

Runtime error is a class is a standard exception which is raised when a generated error does not fall into any category. This program illustrates how to use runtime error as base class and network error as derived class. In a similar way, any exception can be derived from the standard exceptions of Python.

```
# NetworkError has base RuntimeError
# and not Exception
class Networkerror(RuntimeError):
 def __init__(self, arg):
  self.args = arg

try:
 raise Networkerror("Error")

except Networkerror as e:
 print (e.args)
```

Output:

```
('E', 'r', 'r', 'o', 'r')
```

This article is contributed by **Piyush Doorwar**. 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.

Creating a Proxy Webserver in Python | Set 1

Socket programming in python is very user friendly as compared to c. The programmer need not worry about minute details regarding sockets. In python, the user has more chance of focusing on the application layer rather than the network layer. In this tutorial we would be developing a simple multi-threaded proxy server capable of handling HTTP traffic. It would be mostly based on the basic socket programming ideas. If you are not sure about the basics then i would recommend that you brush them up before going through this tutorial.

This is a naive implementation of a proxy server. We would be gradually developing it into a quite useful server in the upcoming tutorials.

To begin with, we would achieve the process in 3 easy steps

1. Creating an incoming socket
We create a socket serverSocket in the __init__ method of the Server Class. This creates a socket for the incoming connections. We then bind the socket and then wait for the clients to connect.

```
def __init__(self, config):
    # Shutdown on Ctrl+C
    signal.signal(signal.SIGINT, self.shutdown)

    # Create a TCP socket
    self.serverSocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

    # Re-use the socket
    self.serverSocket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)

    # bind the socket to a public host, and a port
    self.serverSocket.bind((config['HOST_NAME'], config['BIND_PORT']))

    self.serverSocket.listen(10) # become a server socket
    self.__clients = {}
```

2. Accept client and process
This is the easiest yet the most important of all the steps. We wait for the client's connection request and once a successful connection is made, we dispatch the request in a separate thread, making ourselves available for the next request. This allows us to handle multiple requests simultaneously which boosts the performance of the server multifold times.

```
while True:

    # Establish the connection
    (clientSocket, client_address) = self.serverSocket.accept()

    d = threading.Thread(name=self._getClientName(client_address),
    target = self.proxy_thread, args=(clientSocket, client_address))
    d.setDaemon(True)
    d.start()
```

3. Redirecting the traffic
The main feature of a proxy server is to act as an intermediate between source and destination. Here, we would be fetching data from source and then pass it to the client.

- First, we extract the URL from the received request data.

```
# get the request from browser
request = conn.recv(config['MAX_REQUEST_LEN'])

# parse the first line
first_line = request.split('\n')[0]

# get url
url = first_line.split(' ')[1]
```

- Then, we find the destination address of the request. Address is a tuple of **(destination_ip_address, destination_port_no)**. We will be receiving data from this address.

```
http_pos = url.find("://") # find pos of ://
if (http_pos==-1):
    temp = url
else:
    temp = url[(http_pos+3):] # get the rest of url

port_pos = temp.find(":") # find the port pos (if any)

# find end of web server
webserver_pos = temp.find("/")
if webserver_pos == -1:
    webserver_pos = len(temp)

webserver = ""
port = -1
if (port_pos==-1 or webserver_pos < port_pos):

    # default port
    port = 80
    webserver = temp[:webserver_pos]

else: # specific port
    port = int((temp[(port_pos+1):])[:webserver_pos-port_pos-1])
    webserver = temp[:port_pos]
```

- Now, we setup a new connection to the destination server (or remote server), and then send a copy of the original request to the server. The server will then respond with a response. All the response messages use the generic message format of **RFC 822**.

```
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.settimeout(config['CONNECTION_TIMEOUT'])
s.connect((webserver, port))
s.sendall(request)
```

- We then redirect the server's response to the client. conn is the original connection to the client. The response may be bigger then MAX_REQUEST_LEN that we are receiving in one call, so, a null response marks the end of the response.

```
while 1:
    # receive data from web server
    data = s.recv(config['MAX_REQUEST_LEN'])

    if (len(data) > 0):
        conn.send(data) # send to browser/client
    else:
        break
```

We then close the server connections appropriately and do the error handling to make sure the server works as expected.

How to test the server?
1. Run the server on a terminal. Keep it running and switch to your favorite browser.
2. Go to your browser's proxy settings and change the proxy server to 'localhost' and port to '12345'.
3. Now open any HTTP website (not HTTPS), for eg. geeksforgeeks.org and volla !! you should be able to access the content on the browser.

Once the server is running, we can monitor the requests coming to the client. We can use that data to monitor the content that is going or we can develop statistics based on the content.
We can even restrict access to a website or blacklist an IP address. We would be dealing with more such features in the upcoming tutorials.
What next?
We would be adding the following features in our proxy server in the upcoming tutorials.
– Blacklisting Domains
– Content monitoring
– Logging
– HTTP WebServer + ProxyServer

The whole working source code of this tutorial is available here

Creating a Proxy Webserver in Python | Set 2

If you have any questions/comments then feel free to post them in the comments section.

About the Author:

Pinkesh Badjatiya hails from IIIT Hyderabad .He is a geek at heart with ample projects worth looking for. His project work can be seen here.

If you also wish to showcase your blog here, please see GBlog for guest blog writing on GeeksforGeeks.

GATE CS Corner   Company Wise Coding Practice

GBlog
Project
Python

Creating a Proxy Webserver in Python | Set 2

Prerequisite: Creating a Proxy Webserver in Python – Set1

In this tutorial, few interesting features are added to make it more useful.

- **Add blacklisting of domains**. For Ex. google.com, facebook.com. Create a list of BLACKLIST_DOMAINS in our configuration dict. For now, just ignore/drop the requests received for blacklisted domains. (Ideally we must respond with a forbidden response.)

```
# Check if the host:port is blacklisted
for i in range(0, len(config['BLACKLIST_DOMAINS'])):
    if config['BLACKLIST_DOMAINS'][i] in url:
        conn.close()
return
```

- **To add host blocking:** Say, you may need to allow connections from a particular subnet or connection for a particular person. To add this, create a list of all the allowed hosts. Since the hosts can be a subnet as well, add regex for matching the IP addresses, specifically IPV4 addresses. *" IPv4 addresses are canonically represented in dot-decimal notation, which consists of four decimal numbers, each ranging from 0 to 255, separated by dots, e.g., 172.16.254.1. Each part represents a group of 8 bits (octet) of the address."*
- **Using regex to match correct IP addresses:**
  - Create a new method, _ishostAllowed in Server class, and use fnmatch module to match regexes. Iterate through all the regexes and allow request if it matches any of them. If a client address is not found to be a part of any regex, then send a FORBIDDEN response. Again, for now skip this response creation part.

Note: We would be creating a full fledged custom webserver in upcoming tutorials, there creation of a createResponse function will be done to handle the generic response creation.

```
def _ishostAllowed(self, host):

    """ Check if host is allowed to access
        the content """
    for wildcard in config['HOST_ALLOWED']:
        if fnmatch.fnmatch(host, wildcard):
            return True
    return False
```

Default host match regex would be '*' to match all the hosts. Though, regex of the form '192.168.*' can also be used. Server currently processes requests but does not show any messages, so we are not aware of the state of the server. Its messages should be logged onto console. For this purpose , use the logging module as it is thread safe. (server is multi-threaded if you remember.)

Import module and setup its initial configuration.

```
logging.basicConfig(level = logging.DEBUG,
format = '[%(CurrentTime)-10s] (%(ThreadName)-10s) %(message)s',)
```

- **Create a separate method that logs every message** : Pass it as argument, with additional data such as thread-name and current-time to keep track of the logs. Also create a function that colorizes the logs so that the looks pretty on STDOUT.
  To achieve this, add a boolean in configuration, COLORED_LOGGING and create a new function that colorizes every msg passed to it based on the LOG_LEVEL.

```
def log(self, log_level, client, msg):

    """ Log the messages to appropriate place """
    LoggerDict = {
       'CurrentTime' : strftime("%a, %d %b %Y %X", localtime()),
       'ThreadName' : threading.currentThread().getName()
    }
    if client == -1: # Main Thread
        formatedMSG = msg
    else: # Child threads or Request Threads
        formatedMSG = '{0}:{1} {2}'.format(client[0], client[1], msg)
    logging.debug('%s', utils.colorizeLog(config['COLORED_LOGGING'],
    log_level, formatedMSG), extra=LoggerDict)
```

- **Create a new module, ColorizePython.py:** It contains a pycolors class which maintains a list of color codes. Separate this into another module in order to make code modular and to follow PEP8 standards.

```
# ColorizePython.py
class pycolors:
HEADER = '\033[95m'
OKBLUE = '\033[94m'
OKGREEN = '\033[92m'
WARNING = '\033[93m'
FAIL = '\033[91m'
ENDC = '\033[0m' # End color
BOLD = '\033[1m'
UNDERLINE = '\033[4m'
```

Module:

```
import ColorizePython
```

Method:

```
def colorizeLog(shouldColorize, log_level, msg):
    ## Higher is the log_level in the log()
    ## argument, the lower is its priority.
    colorize_log = {
    "NORMAL": ColorizePython.pycolors.ENDC,
    "WARNING": ColorizePython.pycolors.WARNING,
    "SUCCESS": ColorizePython.pycolors.OKGREEN,
    "FAIL": ColorizePython.pycolors.FAIL,
    "RESET": ColorizePython.pycolors.ENDC
    }

    if shouldColorize.lower() == "true":
        if log_level in colorize_log:
            return colorize_log[str(log_level)] + msg + colorize_log['RESET']
        return colorize_log["NORMAL"] + msg + colorize_log["RESET"]
    return msg
```

- Since the colorizeLog is not a function of a server class, it is created as a separate module named utils.py which stores all the utility that make code easier to understand and put this method there. *Add appropriate log messages wherever required, especially whenever the state of server changes.*
- Modify shutdown method in server to exit all the running threads before exiting the application. *threading.enumerate()* iterates over all the running threads, so we do not need to maintain a list of them. The behavior of threading module is unexpected when we try to end the main_thread. The official documentation also states this:

"join() raises a RuntimeError if an attempt is made to join the current thread as that would cause a deadlock. It is also an error to join() a thread before it has been started and attempts to do so raises the same exception."

So, skip it appropriately. Here's the code for the same.

```
def shutdown(self, signum, frame):
    """ Handle the exiting server. Clean all traces """
    self.log("WARNING", -1, 'Shutting down gracefully...')
    main_thread = threading.currentThread() # Wait for all clients to exit
    for t in threading.enumerate():
        if t is main_thread:
            continue
            self.log("FAIL", -1, 'joining ' + t.getName())
        t.join()
        self.serverSocket.close()
    sys.exit(0)
```

Directory Structure: 1

The full code directory can be downloaded from here.

If you have any comments/suggestions/queries then feel free to ask.

About the Author:

Pinkesh Badjatiya hails from IIIT Hyderabad .He is a geek at heart with ample projects worth looking for. His project work can be seen here.

If you also wish to showcase your blog here, please see GBlog for guest blog writing on GeeksforGeeks.

GATE CS Corner   Company Wise Coding Practice

GBlog
Project
Python

Send message to FB friend using Python

The power of Python comes because of the large number of modules it has. This time we are going to use one of those. Everyone of us, one time or another, has a wish of messaging (or spamming -.-) our Facebook friend. This is a program which can do something similar. So without further delay, let's jump right in.

```
import fbchat
from getpass import getpass
username = str(raw_input("Username: "))
client = fbchat.Client(username, getpass())
no_of_friends = int(raw_input("Number of friends: "))
for i in xrange(no_of_friends):
 name = str(raw_input("Name: "))
 friends = client.getUsers(name)  # return a list of names
 friend = friends[0]
 msg = str(raw_input("Message: "))
 sent = client.send(friend.uid, msg)
 if sent:
    print("Message sent successfully!")
```

Now, let's try to understand the program step by step…

Modules required – fbchat (Can be downloaded from here: Github link); getpass (usually it is pre-installed)

fbchat Installation:

```
sudo pip install fbchat
```

In case you get the error: ** make sure the development packages of libxml2 and libxslt are installed **

In Ubuntu, installing following packages might help:

```
sudo apt-get install python-dev libxml2-dev libxslt1-dev zlib1g-dev
```

Program explanation: The program can be broken down into several steps:

Step – 1: Getting the user credentials

This part is very easy. Using *raw_input()* and *getpass()* we can get the username and password. There are some things to keep in mind in this step.

1. Your facebook account should have a username. You can check that (or set that) by going to your general settings.
2. We are not using raw_input to get password because as soon as the characters (or even the password length) is out, we have got a security breach.

Step – 2: Entering the facebook friend's name
Now that we have signed in, we can enter the number of friends we want to send the message to and for each of those friends, we can enter the custom message.

Step – 3: Spamming **evil**

> *Caution – I am not responsible for extensive usage of the program which can get you banned from facebook or getting blocked by your friend. Get your own list of guinea pigs!*

Because of some reason, if you want to send the same message several times, you can use a simple for loop. Nothing difficult about that ☺

What you can try out now?

- Send message to a group chat.
- Instead of text only, send images as well.
- Create your own 'desktop' messenger.

Facebook hack – Send blank message
Using the normal facebook chat or messenger, it is not possible to send a blank message unless you are aware about the **alt+0173** trick. But, with this program you can send blank messages as well!! All you have to do is enter a blank message. That is, when the program asks for the message to be sent, just press enter and voila!! Your friend will be receiving a series of blank messages…

If you have any other projects in mind concerned with this or if you have prepared some similar to this one, please do share in comments section!

This article is contributed by **Vishwesh Ravi Shrimali**. 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.

Creating a C/C++ Code Formatting tool with help of Clang tools

Today we are going to discuss formatting files in the user's workspace by their extension. For this we are going to make use of Clang's format tools.

Prerequisites:

- Linux Machine
- Python
- Clang Tool

Setup:

- Install Python using the following command:
  ```
  sudo apt-get install python
  ```
- Install Clang Format Tools
  ```
  sudo apt-get install clang-format-3.5
  ```
- Create a python file named format-code.py at any location where you have read and write permissions. In this example we are going to create it in /home/user/. It shall contain the following code:
  ```
  # Python program to format C/C++ files using clang-format
  import os

  # File Extension filter. You can add new extension
  cpp_extensions = (".cxx",".cpp",".c", ".hxx", ".hh", ".cc", ".hpp")

  # Set the current working directory for scanning c/c++ sources (including
  # header files) and apply the clang formatting
  # Please note "-style" is for standard style options
  # and "-i" is in-place editing
  for root, dirs, files in os.walk(os.getcwd()):
      for file in files:
  ```

```
      if file.endswith(cpp_extensions):
        os.system("clang-format-3.5 -i -style=file " + root + "/" + file)
```

- Create format specification file and copy it to project's top level directory , e.g., /home/user/myproject/
  1. Create formatting file (in example, we are creating google coding style tool)

     ```
     clang-format-3.5 -style=google -dump-config > .clang-format
     ```

  2. Copy it to project's directory i.e., it's location becomes: /home/user/myproject/.clang-format

How to use it?

- Navigate to the directory whose files you want to format, e.g.,

  ```
  cd  /home/user/myproject/c-source/
  ```

- Run the format-code file that you created earlier

  ```
  python /home/user/format-code.py
  ```

This shall format all the files in our source directory with the extension same as that mentioned in the code.

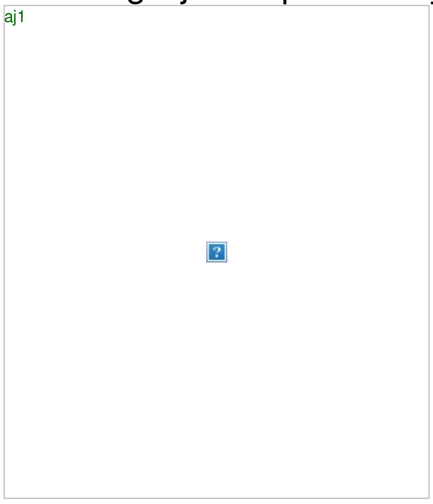
This article is contributed by **Nitin Deokate** .If you like GeeksforGeeks and would like to contribute, you can also write an article using contribute.geeksforgeeks.org or mail your article to contribute@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.

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

Project

Handling Ajax request in Django

aj1


Introduction

This tutorial explains how to carry out a ajax request in Django web framework. We will create a simple post-liking app as a part of example.

Glossary

- Project Initialization
- Create models
- Create views
- Write urls
- Carry out request with Jquery AJax.
- Register models to admin and add some posts.

Implementation:

1. Initiate the Django Project – Here I am assuming that you are done with Django Installation.

- To Create a Django Project execute:

- After creating a project we need to create a django app. To create an app say "post" execute the following: startapp

- Go to django_example/settings.py add the post app
- Now you will have files something like this:

directory_structure

2. Create models: To create models, go to post directory and open models.py .

- In models.py, first create post table. To create post table you'll need to write:

```
class Post(models.Model):
   post_heading = models.CharField(max_length=200)
   post_text = models.TextField()
      def __unicode__(self):     # If python2 use __str__ if python3
          return unicode(self.user)
```

- Then In models.py, create like table. To create like table you'll need to write:

```
class Like(models.Model):
   post = models.ForeignKey(Post)
```

- Make Migration and migrate step: migrations

After completing this steps, we have our database tables ready to use.

3. Create Views:

To create views, we need to go to post directory and open views.py

- First, import Previously created Models and HTTPresponse

```
 from .models import Post, Like
 from django.http import HttpResponse
```

- Create index view to render all the posts. Code sample:

```
def index(request):
   posts = Post.objects.all()  # Getting all the posts from database
   return render(request, 'post/index.html', { 'posts': posts })
```

- Create likePost view to like a post. This view will be called when we will hit a “like button”. Code sample:

```
def likePost(request):
   if request.method == 'GET':
        post_id = request.GET['post_id']
        likedpost = Post.obejcts.get(pk=post_id) #getting the liked posts
        m = Like(post=likedpost) # Creating Like Object
        m.save()  # saving it to store in database
        return HttpResponse("Success!") # Sending an success response
   else:
        return HttpResponse("Request method is not a GET")
```

Once our view get created we will move to write template and jQuery to perform ajax request.

4. Create URLs:

To create urls, open django_example/urls.py. Your django_example/urls.py should look something like this:

```
from django.conf.urls import include, url
from django.contrib import admin
urlpatterns = [
    url(r'^admin/', include(admin.site.urls)),
    url(r'^', include('post.urls')),   # To make post app available at /
  ]
```

To create urls, create file post/urls.py. Your post/urls.py should look something like this:

```
from django.conf.urls import url
from . import views
urlpatterns = [
      url(r'^/$', views.index, name='index'),  # index view at /
```

```
        url(r'^likepost/$', views.likePost, name='likepost'),   # likepost view at /likepost
    ]
```

5. Making templates and carrying out ajax request:

- Create a file post/templates/post/index.html. Code sample:

```
<!DOCTYPE html>
<html>
<head>
    <title>Like Post App</title>
</head>
<body>
    <p id="message"></p>
    {% for post in posts %}
    <h3>{{ forloop.counter }}) {{ post.post_heading }}</h3>
    <p>{{ post.post_text }} </p>
    <a class="likebutton" id="like{{post.id}}" href="#" data-catid="{{ post.id }}">Like</a>
    {% endfor %}
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/1.12.0/jquery.min.js"></script>
    <script type="text/javascript">
    $('.likebutton').click(function(){
    var catid;
    catid = $(this).attr("data-catid");
    $.ajax(
    {
        type:"GET",
        url: "/likepost",
        data:{
                 post_id: catid
        },
        success: function( data )
        {
            $( '#like'+ catid ).remove();
            $( '#message' ).text(data);
        }
     })
});
</script>
</body>
</html>
Basically, what we are doing here is - we are making an ajax get request -> /likepost?post_id=<id_of_liked_post>
```

6. To Register models to admin and add some posts:

- Open post/admin.py.
- Import Models to admin.py.

```
from .models import Post, Like
```

- Register your models:

```
admin.site.register(Post)
admin.site.register(Like)
```

Now add some posts using django default admin portal. Visit http://localhost:8000/ to view and like posts. I also have added this sample app to my github which you may use for reference.

References:

- https://docs.djangoproject.com/en/1.9/intro/tutorial01/

Happy Djangoing !!!

This blog is written by Anshul Singhal. **If you also wish to showcase your blog here, please see GBlog for guest blog writing on GeeksforGeeks.**

GATE CS Corner   Company Wise Coding Practice

GBlog
Project

Twitter Sentiment Analysis using Python

This article covers the sentiment analysis of any topic by parsing the tweets fetched from Twitter using Python.

sentiment-analysis

What is sentiment analysis?
Sentiment Analysis is the process of 'computationally' determining whether a piece of writing is positive, negative or neutral. It's also known as **opinion mining**, deriving the opinion or attitude of a speaker.

Why sentiment analysis?

- **Business:** In marketing field companies use it to develop their strategies, to understand customers' feelings towards products or brand, how people respond to their campaigns or product launches and why consumers don't buy some
products.
- **Politics:** In political field, it is used to keep track of political view, to detect consistency and inconsistency between statements and actions at the government level. It can be used to predict election results as well!
- **Public Actions:** Sentiment analysis also is used to monitor and analyse social phenomena, for the spotting of potentially dangerous situations and determining the general mood of the blogosphere.

Installation:

- **Tweepy:** tweepy is the python client for the official Twitter API.
  Install it using following pip command:
  ```
  pip install tweepy
  ```
- **TextBlob:** textblob is the python library for processing textual data.
  Install it using following pip command:
  ```
  pip install textblob
  ```
  Also, we need to install some NLTK corpora using following command:
  ```
  python -m textblob.download_corpora
  ```
  (Corpora is nothing but a large and structured set of texts.)

Authentication:
In order to fetch tweets through Twitter API, one needs to register an App through their twitter account. Follow these steps for the same:

- Open this link and click the button: 'Create New App'
- Fill the application details. You can leave the callback url field empty.
- Once the app is created, you will be redirected to the app page.
- Open the 'Keys and Access Tokens' tab.
- Copy 'Consumer Key', 'Consumer Secret', 'Access token' and 'Access Token Secret'.

Implementation:

```
import re
import tweepy
from tweepy import OAuthHandler
from textblob import TextBlob

class TwitterClient(object):
 '''
 Generic Twitter Class for sentiment analysis.
 '''
 def __init__(self):
  '''
  Class constructor or initialization method.
  '''
  # keys and tokens from the Twitter Dev Console
  consumer_key = 'XXXXXXXXXXXXXXXXXXXXXXXX'
  consumer_secret = 'XXXXXXXXXXXXXXXXXXXXXXXXXXXX'
  access_token = 'XXXXXXXXXXXXXXXXXXXXXXXXXXXX'
  access_token_secret = 'XXXXXXXXXXXXXXXXXXXXXXXXXX'

  # attempt authentication
  try:
   # create OAuthHandler object
   self.auth = OAuthHandler(consumer_key, consumer_secret)
   # set access token and secret
   self.auth.set_access_token(access_token, access_token_secret)
   # create tweepy API object to fetch tweets
   self.api = tweepy.API(self.auth)
  except:
```

```
   print("Error: Authentication Failed")

 def clean_tweet(self, tweet):
  '''
  Utility function to clean tweet text by removing links, special characters
  using simple regex statements.
  '''
  return ' '.join(re.sub("(@[A-Za-z0-9]+)|([^0-9A-Za-z \t])
                           |(\w+:\/\/\S+)", " ", tweet).split())

 def get_tweet_sentiment(self, tweet):
  '''
  Utility function to classify sentiment of passed tweet
  using textblob's sentiment method
  '''
  # create TextBlob object of passed tweet text
  analysis = TextBlob(self.clean_tweet(tweet))
  # set sentiment
  if analysis.sentiment.polarity > 0:
   return 'positive'
  elif analysis.sentiment.polarity == 0:
   return 'neutral'
  else:
   return 'negative'

 def get_tweets(self, query, count = 10):
  '''
  Main function to fetch tweets and parse them.
  '''
  # empty list to store parsed tweets
  tweets = []

  try:
   # call twitter api to fetch tweets
   fetched_tweets = self.api.search(q = query, count = count)

   # parsing tweets one by one
   for tweet in fetched_tweets:
    # empty dictionary to store required params of a tweet
    parsed_tweet = {}

    # saving text of tweet
    parsed_tweet['text'] = tweet.text
    # saving sentiment of tweet
    parsed_tweet['sentiment'] = self.get_tweet_sentiment(tweet.text)

    # appending parsed tweet to tweets list
    if tweet.retweet_count > 0:
     # if tweet has retweets, ensure that it is appended only once
     if parsed_tweet not in tweets:
      tweets.append(parsed_tweet)
    else:
     tweets.append(parsed_tweet)

   # return parsed tweets
   return tweets

  except tweepy.TweepError as e:
   # print error (if any)
   print("Error : " + str(e))

def main():
 # creating object of TwitterClient Class
 api = TwitterClient()
 # calling function to get tweets
 tweets = api.get_tweets(query = 'Donald Trump', count = 200)

 # picking positive tweets from tweets
 ptweets = [tweet for tweet in tweets if tweet['sentiment'] == 'positive']
 # percentage of positive tweets
 print("Positive tweets percentage: {} %".format(100*len(ptweets)/len(tweets)))
 # picking negative tweets from tweets
 ntweets = [tweet for tweet in tweets if tweet['sentiment'] == 'negative']
 # percentage of negative tweets
 print("Negative tweets percentage: {} %".format(100*len(ntweets)/len(tweets)))
 # percentage of neutral tweets
 print("Neutral tweets percentage: {} % \
  ".format(100*len(tweets - ntweets - ptweets)/len(tweets)))

 # printing first 5 positive tweets
 print("\n\nPositive tweets:")
 for tweet in ptweets[:10]:
  print(tweet['text'])

 # printing first 5 negative tweets
 print("\n\nNegative tweets:")
 for tweet in ntweets[:10]:
  print(tweet['text'])

if __name__ == "__main__":
 # calling main function
```

```
main()
```

Here is how a sample output looks like when above program is run:

```
Positive tweets percentage: 22 %
Negative tweets percentage: 15 %


Positive tweets:
RT @JohnGGalt: Amazing—after years of attacking Donald Trump the media managed
to turn #InaugurationDay into all about themselves.
#MakeAme…
RT @vooda1: CNN Declines to Air White House Press Conference Live YES!
THANK YOU @CNN FOR NOT LEGITIMI…
RT @Muheeb_Shawwa: Donald J. Trump's speech sounded eerily familiar...
POTUS plans new deal for UK as Theresa May to be first foreign leader to meet new
president since inauguration
.@realdonaldtrump #Syria #Mexico #Russia & now #Afghanistan.
Another #DearDonaldTrump Letter worth a read @AJEnglish


Negative tweets:
RT @Slate: Donald Trump's administration: "Government by the worst men."
RT @RVAwonk: Trump, Sean Spicer, et al. lie for a reason.
Their lies are not just lies. Their lies are authoritarian propaganda.
RT @KomptonMusic: Me: I hate corn
Donald Trump: I hate corn too
Me: https://t.co/GPgy8R8HB5
It's ridiculous that people are more annoyed at this than Donald Trump's sexism.
RT @tony_broach: Chris Wallace on Fox news right now talking crap
about Donald Trump news conference it seems he can't face the truth eithe…
RT @fravel: With False Claims, Donald Trump Attacks Media on Crowd Turnout
Aziz Ansari Just Hit Donald Trump Hard In An Epic Saturday NIght Live Monologue
```

We follow these 3 major steps in our program:

- Authorize twitter API client.
- Make a GET request to Twitter API to fetch tweets for a particular query.
- Parse the tweets. Classify each tweet as positive, negative or neutral.

Now, let us try to understand the above piece of code:

- First of all, we create a **TwitterClient** class. This class contains all the methods to interact with Twitter API and parsing tweets. We use **__init__** function to handle the authentication of API client.
- In **get_tweets** function, we use:

  ```
  fetched_tweets = self.api.search(q = query, count = count)
  ```

  to call the Twitter API to fetch tweets.
- In **get_tweet_sentiment** we use textblob module.

  ```
  analysis = TextBlob(self.clean_tweet(tweet))
  ```

  TextBlob is actually a high level library built over top of NLTK library. First we call **clean_tweet** method to remove links, special characters, etc. from the tweet using some simple regex.
  Then, as we pass **tweet** to create a **TextBlob** object, following processing is done over text by textblob library:

  - Tokenize the tweet ,i.e split words from body of text.
  - Remove stopwords from the tokens.(stopwords are the commonly used words which are irrelevant in text analysis like I, am, you, are, etc.)
  - Do POS(part of speech) tagging of the tokens and select only significant features/tokens like adjectives, adverbs, etc.
  - Pass the tokens to a **sentiment classifier** which classifies the tweet sentiment as positive, negative or neutral by assigning it a polarity between -1.0 to 1.0 .

  Here is how **sentiment classifier** is created:

  - **TextBlob** uses a Movies Reviews dataset in which reviews have already been labelled as positive or negative.
  - Positive and negative features are extracted from each positive and negative review respectively.
  - Training data now consists of labelled positive and negative features. This data is trained on a Naive Bayes Classifier.

  Then, we use **sentiment.polarity** method of **TextBlob** class to get the polarity of tweet between -1 to 1.
  Then, we classify polarity as:

  ```
  if analysis.sentiment.polarity > 0:
      return 'positive'
  elif analysis.sentiment.polarity == 0:
      return 'neutral'
  else:
      return 'negative'
  ```

- Finally, parsed tweets are returned. Then, we can do various type of statistical analysis on the tweets. For example, in above program, we tried to find the percentage of positive, negative and neutral tweets about a query.

References:

- http://www.ijcaonline.org/research/volume125/number3/dandrea-2015-ijca-905866.pdf
- https://textblob.readthedocs.io/en/dev/quickstart.html#sentiment-analysis
- textblob.readthedocs.io/en/dev/_modules/textblob/en/sentiments.html

This article is contributed by **Nikhil Kumar**. 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

GBlog
Project
Python

Working with Images in Python

PIL is the Python Imaging Library which provides the python interpreter with image editing capabilities. It was developed by Fredrik Lundh and several other contributors. Pillow is the friendly PIL fork and an easy to use library developed by Alex Clark and other contributors. We'll be working with Pillow.

Installation:

- **Linux:** On linux terminal type the following:

  ```
  pip install Pillow
  ```

  Installing pip via terminal:

  ```
  sudo apt-get update
  sudo apt-get install python-pip
  ```

- **Windows:** Download the appropriate Pillow package according to your python version. Make sure to download according to the python version you have.

We'll be working with the Image Module here which provides a class of the same name and provides a lot of functions to work on our images.To import the Image module, our code should begin with the following line:

```
from PIL import Image
```

Operations with Images:

- **Open a particular image from a path:**

  ```
  #img  = Image.open(path)
  # On successful execution of this statement,
  # an object of Image type is returned and stored in img variable)

  try:
      img  = Image.open(path)
  except IOError:
      pass
  # Use the above statement within try block, as it can
  # raise an IOError if file cannot be found,
  # or image cannot be opened.
  ```

- **Retrieve size of image**: The instances of Image class that are created have many attributes, one of its useful attribute is size.

  ```
  from PIL import Image

  filename = "image.png"
  with Image.open(filename) as image:
      width, height = image.size
  #Image.size gives a 2-tuple and the width, height can be obtained
  ```

  Some other attributes are: Image.width, Image.height, Image.format, Image.info etc.

- **Save changes in image:** To save any changes that you have made to the image file, we need to give path as well as image format.

  ```
  img.save(path, format)
  # format is optional, if no format is specified,
  #it is determined from the filename extension
  ```

- **Rotating an Image:** The image rotation needs angle as parameter to get the image rotated.

  ```
  from PIL import Image

  def main():
   try:
      #Relative Path
    img = Image.open("picture.jpg")

    #Angle given
    img = img.rotate(180)

     #Saved in the same relative location
    img.save("rotated_picture.jpg")
   except IOError:
    pass

  if __name__ == "__main__":
   main()
  ```

rotating an image in python

Note: There is an optional expand flag available as one of the argument of the rotate method, which if set true, expands the output image to make it large enough to hold the full rotated image.
As seen in the above code snippet, I have used a relative path where my image is located in the same directory as my python code file, an absolute path can be used as well.

- **Cropping an Image:** Image.crop(box) takes a 4-tuple (left, upper, right, lower) pixel coordinate, and returns a rectangular region from the used image.

```
from PIL import Image

def main():
 try:
  #Relative Path
  img = Image.open("picture.jpg")
  width, height = img.size

  area = (0, 0, width/2, height/2)
  img = img.crop(area)

  #Saved in the same relative location
  img.save("cropped_picture.jpg")

 except IOError:
  pass

if __name__ == "__main__":
 main()
```

cropping an image in python

- **Resizing an Image:** Image.resize(size)- Here size is provided as a 2-tuple width and height.

```
from PIL import Image

def main():
 try:
     #Relative Path
  img = Image.open("picture.jpg")
  width, height = img.size

  img = img.resize((width/2, height/2))

  #Saved in the same relative location
  img.save("resized_picture.jpg")
 except IOError:
  pass

if __name__ == "__main__":
 main()
```

resizing an image in python

- **Pasting an image on another image:** The second argument can be a 2-tuple (specifying the top left corner), or a 4-tuple (left, upper, right, lower) – in this case the size of pasted image must match the size of this box region, or None which is equivalent to (0, 0).

```
from PIL import Image

def main():
 try:
  #Relative Path
  #Image on which we want to paste
  img = Image.open("picture.jpg")

  #Relative Path
  #Image which we want to paste
  img2 = Image.open("picture2.jpg")
  img.paste(img2, (50, 50))

  #Saved in the same relative location
  img.save("pasted_picture.jpg")

 except IOError:
  pass

if __name__ == "__main__":
 main()

##An additional argument for an optional image mask image is also available.
```

pasting an image on other in Python

- **Getting a Histogram of an Image:** This will return a histogram of the image as a list of pixel counts, one for each pixel in the image. (A histogram of an image is a graphical representation of the tonal distribution in a digital image. It contains what all the brightness values contained in an image are. It plots the number of pixels for each brightness value. It helps in doing the exposure settings.)
  from PIL import Image

```
def main():
 try:
  #Relative Path
  img = Image.open("picture.jpg")

  #Getting histogram of image
  print img.histogram()

 except IOError:
  pass

if __name__ == "__main__":
 main()
```

getting a histogram of image in python

- **Transposing an Image:** This feature gives us the mirror image of an image

```
 from PIL import Image

def main():
 try:
    #Relative Path
  img = Image.open("picture.jpg")

  #transposing image
  transposed_img = img.transpose(Image.FLIP_LEFT_RIGHT)

  #Save transposed image
  transposed_img.save("transposed.jpg")
 except IOError:
  pass

if __name__ == "__main__":
 main()
```

transposing an image in python

- **Split an image into individual bands:** Splitting an image in RGB mode, creates three new images each containing a copy of the original individual bands.

```
from PIL import Image

def main():
 try:
    #Relative Path
  img = Image.open("picture.jpg")

  #splitting the image
  print img.split()
 except IOError:
  pass

if __name__ == "__main__":
 main()
```

split an image in python

- **tobitmap:** Converting an image to an X11 bitmap (A plain text binary image format). It returns a string containing an X11 bitmap, it can only be used for mode “1” images, i.e. 1 bit pixel black and white images.
  from PIL import Image

```
def main():
 try:
    #Relative Path
  img = Image.open("picture.jpg")
  print img.mode

  #converting image to bitmap
  print img.tobitmap()

  print type(img.tobitmap())
 except IOError:
  pass

if __name__ == "__main__":
 main()
```

converting image to bitmap in python

converting image to bitmao in python

- **Creating a thumbnail:** This method creates a thumbnail of the image that is opened. It does not return a new image object, it makes in-place modification to the currently opened image object itself. If you do not want to change the original image object, create a copy and then apply this method. This method also evaluates the appropriate to maintain the aspect ratio of the image according to the size passed.
  from PIL import Image

```
def main():
  try:
     #Relative Path
   img = Image.open("picture.jpg")

   #In-place modification
   img.thumbnail((200, 200))

   img.save("thumb.jpg")
  except IOError:
   pass

 if __name__ == "__main__":
  main()
```

creating thumbnail of image in python

This article is contributed by **Mohit Agarwal**. 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

Project
Python
Image-Processing

Graph Plotting in Python | Set 1

This series will introduce you to graphing in python with Matplotlib, which is arguably the most popular graphing and data visualization library for Python.

Installation

Easiest way to install matplotlib is to use pip. Type following command in terminal:

```
pip install matplotlib
```

OR, you can download it from here and install it manually.

Getting started (Plotting a line)

```
# importing the required module
import matplotlib.pyplot as plt

# x axis values
x = [1,2,3]
# corresponding y axis values
y = [2,4,1]

# plotting the points
plt.plot(x, y)

# naming the x axis
plt.xlabel('x - axis')
# naming the y axis
plt.ylabel('y - axis')

# giving a title to my graph
plt.title('My first graph!')

# function to show the plot
plt.show()
```

Output:

mp1

The code seems self explanatory. Following steps were followed:

- Define the x-axis and corresponding y-axis values as lists.
- Plot them on canvas using **.plot()** function.
- Give a name to x-axis and y-axis using **.xlabel()** and **.ylabel()** functions.
- Give a title to your plot using **.title()** function.
- Finally, to view your plot, we use **.show()** function.

Plotting two or more lines on same plot

```
import matplotlib.pyplot as plt

# line 1 points
x1 = [1,2,3]
y1 = [2,4,1]
# plotting the line 1 points
plt.plot(x1, y1, label = "line 1")

# line 2 points
x2 = [1,2,3]
y2 = [4,1,3]
# plotting the line 2 points
plt.plot(x2, y2, label = "line 2")

# naming the x axis
plt.xlabel('x - axis')
# naming the y axis
plt.ylabel('y - axis')
# giving a title to my graph
plt.title('Two lines on same graph!')

# show a legend on the plot
plt.legend()

# function to show the plot
plt.show()
```

Output:

mp2

- Here, we plot two lines on same graph. We differentiate between them by giving them a name(**label**) which is passed as an argument of .plot() function.
- The small rectangular box giving information about type of line and its color is called legend. We can add a legend to our plot using **.legend()** function.

Customization of Plots

Here, we discuss some elementary customizations applicable on almost any plot.

```
import matplotlib.pyplot as plt

# x axis values
x = [1,2,3,4,5,6]
# corresponding y axis values
y = [2,4,1,5,2,6]

# plotting the points
plt.plot(x, y, color='green', linestyle='dashed', linewidth = 3,
        marker='o', markerfacecolor='blue', markersize=12)
```

```
# setting x and y axis range
plt.ylim(1,8)
plt.xlim(1,8)

# naming the x axis
plt.xlabel('x - axis')
# naming the y axis
plt.ylabel('y - axis')

# giving a title to my graph
plt.title('Some cool customizations!')

# function to show the plot
plt.show()
```

Output:
mp3

As you can see, we have done several customizations like

- setting the line-width, line-style, line-color.
- setting the marker, marker's face color, marker's size.
- overriding the x and y axis range. If overriding is not done, pyplot module uses auto-scale feature to set the axis range and scale.

Bar Chart

```
import matplotlib.pyplot as plt

# x-coordinates of left sides of bars
left = [1, 2, 3, 4, 5]

# heights of bars
height = [10, 24, 36, 40, 5]

# labels for bars
tick_label = ['one', 'two', 'three', 'four', 'five']

# plotting a bar chart
plt.bar(left, height, tick_label = tick_label,
        width = 0.8, color = ['red', 'blue'])

# naming the x-axis
plt.xlabel('x - axis')
# naming the y-axis
plt.ylabel('y - axis')
# plot title
plt.title('My bar chart!')

# function to show the plot
plt.show()
```

Output :

mp4

- Here, we use **plt.bar()** function to plot a bar chart.
- x-coordinates of left side of bars are passed along with heights of bars.
- you can also give some name to x-axis coordinates by defining **tick_labels**

Histogram

```
import matplotlib.pyplot as plt

# frequencies
ages = [2,5,70,40,30,45,50,45,43,40,44,
        60,7,13,57,18,90,77,32,21,20,40]

# setting the ranges and no. of intervals
range = (0, 100)
bins = 10

# plotting a histogram
plt.hist(ages, bins, range, color = 'red',
        histtype = 'bar', rwidth = 0.8)

# x-axis label
plt.xlabel('age')
# frequency label
plt.ylabel('No. of people')
# plot title
plt.title('My histogram')

# function to show the plot
plt.show()
```

Output:

mp5

- Here, we use **plt.hist()** function to plot a histogram.
- frequencies are passed as the **ages** list.
- Range could be set by defining a tuple containing min and max value.
- Next step is to “**bin**” the range of values—that is, divide the entire range of values into a series of intervals—and then count how many values fall into each interval. Here we have defined **bins** = 10. So, there are a total of 100/10 = 10 intervals.

Scatter plot

```
import matplotlib.pyplot as plt

# x-axis values
x = [1,2,3,4,5,6,7,8,9,10]
# y-axis values
y = [2,4,5,7,6,8,9,11,12,12]

# plotting points as a scatter plot
plt.scatter(x, y, label= "stars", color= "m",
            marker= "*", s=30)

# x-axis label
plt.xlabel('x - axis')
# frequency label
plt.ylabel('y - axis')
# plot title
plt.title('My scatter plot!')
# showing legend
plt.legend()

# function to show the plot
plt.show()
```

Output:

mp6

- Here, we use **plt.scatter()** function to plot a scatter plot.
- Like a line, we define x and corresponding y – axis values here as well.
- **marker** argument is used to set the character to use as marker. Its size can be defined using **s** parameter.

Pie-chart

```
import matplotlib.pyplot as plt

# defining labels
activities = ['eat', 'sleep', 'work', 'play']

# portion covered by each label
slices = [3, 7, 8, 6]

# color for each label
colors = ['r', 'm', 'g', 'b']

# plotting the pie chart
plt.pie(slices, labels = activities, colors=colors,
        startangle=90, shadow = True, explode = (0, 0, 0.1, 0),
        radius = 1.2, autopct = '%1.1f%%')

# plotting legend
plt.legend()

# showing the plot
plt.show()
```

Output of above program looks like this:

mp7

- Here, we plot a pie chart by using **plt.pie()** method.
- First of all, we define the **labels** using a list called **activities**.
- Then, portion of each label can be defined using another list called **slices**.
- Color for each label is defined using a list called **colors**.
- **shadow = True** will show a shadow beneath each label in pie-chart.
- **startangle** rotates the start of the pie chart by given degrees counterclockwise from the x-axis.
- **explode** is used to set the fraction of radius with which we offset each wedge.
- **autopct** is used to format the value of each label. Here, we have set it to show the percentage value only upto 1 decimal place.

Plotting curves of given equation

```
# importing the required modules
import matplotlib.pyplot as plt
import numpy as np

# setting the x - coordinates
x = np.arange(0, 2*(np.pi), 0.1)
# setting the corresponding y - coordinates
y = np.sin(x)

# potting the points
plt.plot(x, y)

# function to show the plot
plt.show()
```

Output of above program looks like this:
mp8

Here, we use **NumPy** which is a general-purpose array-processing package in python.

- To set the x – axis values, we use **np.arange()** method in which first two arguments are for range and third one for step-wise increment. The result is a numpy array.
- To get corresponding y-axis values, we simply use predefined **np.sin()** method on the numpy array.
- Finally, we plot the points by passing x and y arrays to the **plt.plot()** function.

So, in this part, we discussed various types of plots we can create in matplotlib. There are more plots which haven't been covered but the most significant ones are discussed here –

- Graph Plotting in Python | Set 2
- Graph Plotting in Python | Set 3

This article is contributed by **Nikhil Kumar**. 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.

Python

Graph Plotting in Python | Set 2

Graph Plotting in Python | Set 1

Subplots

Subplots are required when we want to show two or more plots in same figure. We can do it in two ways using two slightly different methods.

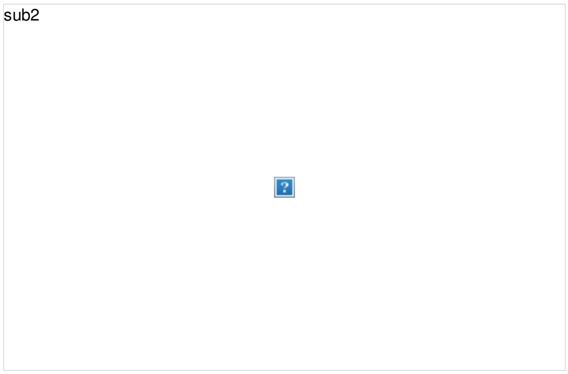
Method 1

```
# importing required modules
import matplotlib.pyplot as plt
import numpy as np

# function to generate coordinates
def create_plot(ptype):
    # setting the x-axis vaues
    x = np.arange(-10, 10, 0.01)

    # setting the y-axis values
    if ptype == 'linear':
        y = x
    elif ptype == 'quadratic':
        y = x**2
    elif ptype == 'cubic':
        y = x**3
    elif ptype == 'quartic':
        y = x**4

    return(x, y)

# setting a style to use
plt.style.use('fivethirtyeight')

# create a figure
fig = plt.figure()

# define subplots and their positions in figure
plt1 = fig.add_subplot(221)
plt2 = fig.add_subplot(222)
plt3 = fig.add_subplot(223)
plt4 = fig.add_subplot(224)

# plotting points on each subplot
x, y = create_plot('linear')
plt1.plot(x, y, color ='r')
plt1.set_title('$y_1 = x$')

x, y = create_plot('quadratic')
plt2.plot(x, y, color ='b')
```

```
plt2.set_title('$y_2 = x^2$')

x, y = create_plot('cubic')
plt3.plot(x, y, color ='g')
plt3.set_title('$y_3 = x^3$')

x, y = create_plot('quartic')
plt4.plot(x, y, color ='k')
plt4.set_title('$y_4 = x^4$')

# adjusting space between subplots
fig.subplots_adjust(hspace=.5,wspace=0.5)

# function to show the plot
plt.show()
```

Output:

sub2



Let us go through this program step by step:

```
plt.style.use('fivethirtyeight')
```

The styling of plots can be configured by setting different styles available or setting your own. You can learn more about this feature here

```
fig = plt.figure()
```

Figure acts as a top level container for all plot elements. So, we define a figure as **fig** which will contain all our subplots.

```
plt1 = fig.add_subplot(221)
plt2 = fig.add_subplot(222)
plt3 = fig.add_subplot(223)
plt4 = fig.add_subplot(224)
```

Here we use fig.add_subplot method to define subplots and their positions. The function prototype is like this:

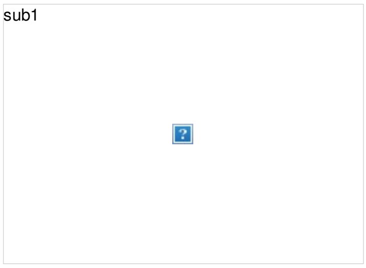
```
add_subplot(nrows, ncols, plot_number)
```

If a subplot is applied to a figure, the figure will be notionally split into ‘nrows’ * ‘ncols’ sub-axes. The parameter ‘plot_number’ identifies the subplot that the function call has to create. ‘plot_number’ can range from 1 to a maximum of ‘nrows’ * ‘ncols’.

If the values of the three parameters are less than 10, the function subplot can be called with one int parameter, where the hundreds represent ‘nrows’, the tens represent ‘ncols’ and the units represent ‘plot_number’. This means: Instead of **subplot(2, 3, 4)** we can write **subplot(234)**.

This figure will make it clear that how positions are specified:

sub1



```
x, y = create_plot('linear')
plt1.plot(x, y, color ='r')
plt1.set_title('$y_1 = x$')
```

Next, we plot our points on each subplot. First, we generate x and y axis coordinates using **create_plot** function by specifying the type of curve we want.
Then, we plot those points on our subplot using **.plot** method. Title of subplot is set by using **set_title** method. Using **$** at starting and end of the title text will ensure that ‘_'(underscore) is read as a subscript and ‘^’ is read as a superscript.

```
fig.subplots_adjust(hspace=.5,wspace=0.5)
```

This is another utility method which creates space between subplots.

```
plt.show()
```

In the end, we call plt.show() method which will show the current figure.

Method 2

```
# importing required modules
import matplotlib.pyplot as plt
import numpy as np

# function to generate coordinates
def create_plot(ptype):
    # setting the x-axis vaues
    x = np.arange(0, 5, 0.01)

    # setting y-axis values
    if ptype == 'sin':
        # a sine wave
        y = np.sin(2*np.pi*x)
    elif ptype == 'exp':
        # negative exponential function
        y = np.exp(-x)
    elif ptype == 'hybrid':
        # a damped sine wave
        y = (np.sin(2*np.pi*x))*(np.exp(-x))

    return(x, y)

# setting a style to use
plt.style.use('ggplot')

# defining subplots and their positions
plt1 = plt.subplot2grid((11,1), (0,0), rowspan = 3, colspan = 1)
plt2 = plt.subplot2grid((11,1), (4,0), rowspan = 3, colspan = 1)
plt3 = plt.subplot2grid((11,1), (8,0), rowspan = 3, colspan = 1)

# plotting points on each subplot
x, y = create_plot('sin')
plt1.plot(x, y, label = 'sine wave', color ='b')
x, y = create_plot('exp')
plt2.plot(x, y, label = 'negative exponential', color = 'r')
x, y = create_plot('hybrid')
plt3.plot(x, y, label = 'damped sine wave', color = 'g')

# show legends of each subplot
plt1.legend()
plt2.legend()
plt3.legend()

# function to show plot
plt.show()
```

Output:
sub3

Let us go through important parts of this program as well:

```
plt1 = plt.subplot2grid((11,1), (0,0), rowspan = 3, colspan = 1)
plt2 = plt.subplot2grid((11,1), (4,0), rowspan = 3, colspan = 1)
plt3 = plt.subplot2grid((11,1), (8,0), rowspan = 3, colspan = 1)
```

subplot2grid is similar to “pyplot.subplot” but uses 0-based indexing and let subplot to occupy multiple cells.

Let us try to understand the arguments of the **subplot2grid** method:

1. argument 1 : geometry of the grid
2. argument 2: location of the subplot in the grid
3. argument 3: (rowspan) No. of rows covered by subplot.
4. argument 4: (colspan) No. of columns covered by subplot.

This figure will make this concept more clear:

sub4

In our example, each subplot spans over 3 rows and 1 column with two empty rows (row no. 4,8) .

```
x, y = create_plot('sin')
plt1.plot(x, y, label = 'sine wave', color ='b')
```

Nothing special in this part as the syntax to plot points on a subplot remains same.

```
plt1.legend()
```

This will show the label of the subplot on the figure.

```
plt.show()
```

Finally, we call the plt.show() function to show the current plot.

Note: After going through the above two examples, we can infer that one should use **subplot()** method when the plots are of uniform size where as **subplot2grid()** method should be preferred when we want more flexibility on position and sizes of our subplots.

3-D plotting

We can easily plot 3-D figures in matplotlib. Now, we discuss some important and commonly used 3-D plots.

- **Plotting points**

```
from mpl_toolkits.mplot3d import axes3d
import matplotlib.pyplot as plt
from matplotlib import style
import numpy as np

# setting a custom style to use
style.use('ggplot')

# create a new figure for plotting
fig = plt.figure()

# create a new subplot on our figure
# and set projection as 3d
ax1 = fig.add_subplot(111, projection='3d')

# defining x, y, z co-ordinates
x = np.random.randint(0, 10, size = 20)
y = np.random.randint(0, 10, size = 20)
z = np.random.randint(0, 10, size = 20)

# plotting the points on subplot


# setting labels for the axes
ax1.set_xlabel('x-axis')
ax1.set_ylabel('y-axis')
ax1.set_zlabel('z-axis')

# function to show the plot
plt.show()
```

Output of above program will provide you with a window which can rotate or enlarge the plot. Here is a screenshot: (dark points are nearer than light ones)

figure_1-1

Let us try to understand some important aspects of this code now.

```
from mpl_toolkits.mplot3d import axes3d
```

This is the module required to plot on 3-D space.

```
ax1 = fig.add_subplot(111, projection='3d')
```

Here, we create a subplot on our figure and set projection argument as 3d.

```
ax1.scatter(x, y, z, c = 'm', marker = 'o')
```

Now we use **.scatter()** function to plot the points in XYZ plane.

- **Plotting lines**

```
# importing required modules
from mpl_toolkits.mplot3d import axes3d
import matplotlib.pyplot as plt
from matplotlib import style
import numpy as np

# setting a custom style to use
style.use('ggplot')

# create a new figure for plotting
fig = plt.figure()

# create a new subplot on our figure
ax1 = fig.add_subplot(111, projection='3d')

# defining x, y, z co-ordinates
x = np.random.randint(0, 10, size = 5)
y = np.random.randint(0, 10, size = 5)
z = np.random.randint(0, 10, size = 5)

# plotting the points on subplot
ax1.plot_wireframe(x,y,z)

# setting the labels
ax1.set_xlabel('x-axis')
ax1.set_ylabel('y-axis')
ax1.set_zlabel('z-axis')

plt.show()
```

A screenshot of the 3-D plot of above program will look like:

figure_1

The main difference in this program with previous one is:

```
ax1.plot_wireframe(x,y,z)
```

We used **.plot_wireframe()** method to plot lines over a given set of 3-D points.

- **Plotting Bars**

```
# importing required modules
from mpl_toolkits.mplot3d import axes3d
import matplotlib.pyplot as plt
from matplotlib import style
import numpy as np

# setting a custom style to use
style.use('ggplot')

# create a new figure for plotting
fig = plt.figure()

# create a new subplot on our figure
ax1 = fig.add_subplot(111, projection='3d')

# defining x, y, z co-ordinates for bar position
x = [1,2,3,4,5,6,7,8,9,10]
y = [4,3,1,6,5,3,7,5,3,7]
z = np.zeros(10)

# size of bars
dx = np.ones(10)            # length along x-axis
dy = np.ones(10)            # length along y-axs
dz = [1,3,4,2,6,7,5,5,10,9]   # height of bar

# setting color scheme
color = []
for h in dz:
    if h &gt; 5:
        color.append('r')
    else:
        color.append('b')

# plotting the bars
ax1.bar3d(x, y, z, dx, dy, dz, color = color)

# setting axes labels
ax1.set_xlabel('x-axis')
ax1.set_ylabel('y-axis')
ax1.set_zlabel('z-axis')

plt.show()
```

A screenshot of the 3-D environment created is here:

figure_1-2

Let us go through important aspects of this program:

```
x = [1,2,3,4,5,6,7,8,9,10]
y = [4,3,1,6,5,3,7,5,3,7]
z = np.zeros(10)
```

Here, we define the base positions of bars. Setting z = 0 means all bars start from XY plane.

```
dx = np.ones(10)           # length along x-axis
dy = np.ones(10)           # length along y-axs
dz = [1,3,4,2,6,7,5,5,10,9]  # height of bar
```

dx, dy, dz denote the size of bar. Consider he bar as a cuboid, then dx, dy, dz are its expansions along x, y, z axis respectively.

```
for h in dz:
    if h > 5:
        color.append('r')
    else:
        color.append('b')
```

Here, we set the color for each bar as a list. The color scheme is red for bars with height greater than 5 and blue otherwise.

```
ax1.bar3d(x, y, z, dx, dy, dz, color = color)
```

Finally, to plot the bars, we use **.bar3d()** function.

- **Plotting curves**

```
# importing required modules
from mpl_toolkits.mplot3d import axes3d
import matplotlib.pyplot as plt
from matplotlib import style
import numpy as np

# setting a custom style to use
style.use('ggplot')

# create a new figure for plotting
fig = plt.figure()

# create a new subplot on our figure
ax1 = fig.add_subplot(111, projection='3d')

# get points for a mesh grid
u, v = np.mgrid[0:2*np.pi:200j, 0:np.pi:100j]

# setting x, y, z co-ordinates
x=np.cos(u)*np.sin(v)
y=np.sin(u)*np.sin(v)
z=np.cos(v)

# plotting the curve
ax1.plot_wireframe(x, y, z, rstride = 5, cstride = 5, linewidth = 1)

plt.show()
```

Output of this program will look like this:

figure_1-3

Here, we plotted a sphere as a mesh grid.
Let us go through some important parts:

```
u, v = np.mgrid[0:2*np.pi:200j, 0:np.pi:100j]
```

We use np.mgrid in order to get points so that we can create a mesh.
You can read more about this here.

```
x=np.cos(u)*np.sin(v)
y=np.sin(u)*np.sin(v)
z=np.cos(v)
```

This is nothing but the parametric equation of a sphere.

```
ax1.plot_wireframe(x, y, z, rstride = 5, cstride = 5, linewidth = 1)
```

Agan, we use **.plot_wireframe()** method. Here, **rstride** and **cstride** arguments can be used to set how much dense our mesh must be.

Next Article: Graph Plotting in Python | Set 3

This article is contributed by **Nikhil Kumar**. 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.

Graph Plotting in Python | Set 3

Graph Plotting in Python | Set 1
Graph Plotting in Python | Set 2

Matplotlib is a pretty extensive library which supports **Animations** of graphs as well. The animation tools center around the **matplotlib.animation** base class, which provides a framework around which the animation functionality is built. The main interfaces are **TimedAnimation** and **FuncAnimation** and out of the two, **FuncAnimation** is the most convenient one to use.

Installation:

- **Matplotlib**: Refer to Graph Plotting in Python | Set 1
- **Numpy:** You can install numpy module using following pip command:

```
pip install numpy
```

- **FFMPEG**: It is required only for saving the animation as a video. The executable can be downloaded from here.

Implementation:

```
# importing required modules
import matplotlib.pyplot as plt
import matplotlib.animation as animation
import numpy as np

# create a figure, axis and plot element
fig = plt.figure()
ax = plt.axes(xlim=(-50, 50), ylim=(-50, 50))
line, = ax.plot([], [], lw=2)

# initialization function
def init():
    # creating an empty plot/frame
    line.set_data([], [])
    return line,

# lists to store x and y axis points
xdata, ydata = [], []

# animation function
def animate(i):
    # t is a parameter
    t = 0.1*i

    # x, y values to be plotted
    x = t*np.sin(t)
    y = t*np.cos(t)

    # appending new points to x, y axes points list
    xdata.append(x)
    ydata.append(y)

    # set/update the x and y axes data
    line.set_data(xdata, ydata)

    # return line object
    return line,

# setting a title for the plot
plt.title('A growing coil!')
# hiding the axis details
plt.axis('off')

# call the animator
anim = animation.FuncAnimation(fig, animate, init_func=init,
                               frames=500, interval=20, blit=True)

# save the animation as mp4 video file
anim.save('animated_coil.mp4', writer = 'ffmpeg', fps = 30)

# show the plot
plt.show()
```

Here is how the output animation looks like:

Now, let us try to understand the code in pieces:

```
fig = plt.figure()
ax = plt.axes(xlim=(-50, 50), ylim=(-50, 50))
line, = ax.plot([], [], lw=2)
```

Here, we first create a figure, i.e a top level container for all our subplots.

Then we create an axes element **ax** which acts as a subplot. The range/limit for x and y axis are also defined while creating the axes element.

Finally, we create the **plot** element, named as **line** . Initially, the x and y axis points have been defined as empty lists and line-width **(lw)** has been set as 2.

```
def init():
    line.set_data([], [])
    return line,
```

Now, we declare a initialization function, **init** . This function is called by animator to create the first frame.

```
def animate(i):
    # t is a parameter
    t = 0.1*i
    # x, y values to be plotted
    x = t*np.sin(t)
    y = t*np.cos(t)

    # appending new points to x, y axes points list
    xdata.append(x)
    ydata.append(y)

    # set/update the x and y axes data
    line.set_data(xdata, ydata)

    # return line object
    return line,
```

This is the most important function of above program. **animate()** function is called again and again by the animator to create each frame. The number of times this function will be called is determined by number of frames, which is passed as **frames** argument to animator.
animate() function takes the index of ith frame as argument.

```
t = 0.1*i
```

Here, we cleverly use the index of current frame as a parameter!

```
x = t*np.sin(t)
y = t*np.cos(t)
```

Now, since we have the parameter **t**, we can easily plot any parametric equation. For example, here, we are plotting a spiral using its parametric equation.

```
line.set_data(xdata, ydata)
return line,
```

Finally, we use **set_data()** function to set x and y data and then return plot object, **line** .

```
anim = animation.FuncAnimation(fig, animate, init_func=init,
                               frames=500, interval=20, blit=True)
```

Now, we create the FuncAnimation object, **anim** . It takes various arguments explained below:
fig : figure to be plotted.
animate : the function to be called repeatedly for each frame**.**
init_func : function used to draw a clear frame. It is called once before the first frame.
frames : number of frames. (Note: **frames** can also be an iterable or generator.)
interval : duration between frames (in milliseconds)
blit : setting blit=True means that only those parts will be drawn, which have changed.

```
anim.save('animated_coil.mp4', writer = 'ffmpeg', fps = 30)
```

Now, we save the animator object as a video file using **save()** function. You will need a movie writer for saving the animation video. In this example, we have used FFMPEG movie writer. So, **writer** is set as 'ffmpeg'.
fps stands for frame per second.

Example 2

This example shows how one can make a rotating curve by applying some simple mathematics!

```
# importing required modules
import matplotlib.pyplot as plt
import matplotlib.animation as animation
import numpy as np

# create a figure, axis and plot element
fig = plt.figure()
ax = plt.axes(xlim=(-25, 25), ylim=(-25, 25))
line, = ax.plot([], [], lw=2)

# initialization function
def init():
    # creating an empty plot/frame
    line.set_data([], [])
    return line,

# set of points for a star (could be any curve)
p = np.arange(0, 4*np.pi, 0.1)
x = 12*np.cos(p) + 8*np.cos(1.5*p)
y = 12*np.sin(p) - 8*np.sin(1.5*p)

# animation function
def animate(i):
    # t is a parameter
    t = 0.1*i

    # x, y values to be plotted
    X = x*np.cos(t) - y*np.sin(t)
    Y = y*np.cos(t) + x*np.sin(t)

    # set/update the x and y axes data
    line.set_data(X, Y)
    # return line object
    return line,

# setting a title for the plot
plt.title('A rotating star!')
# hiding the axis details
plt.axis('off')

# call the animator
anim = animation.FuncAnimation(fig, animate, init_func=init,
                               frames=100, interval=100, blit=True)

# save the animation as mp4 video file
anim.save('basic_animation.mp4', writer = 'ffmpeg', fps = 10)

# show the plot
plt.show()
```

Here is how the output of above program looks like:

Here, we have used some simple mathematics to rotate a given curve.

- The star shape is obtained by putting k = 2.5 and 0<t<4*pi in the parametric equation given below:

  1

  The same has been applied here:

  ```
  p = np.arange(0, 4*np.pi, 0.1)
  x = 12*np.cos(p) + 8*np.cos(1.5*p)
  y = 12*np.sin(p) - 8*np.sin(1.5*p)
  ```

- Now, in each frame, we rotate the star curve using concept of rotation in complex numbers. Let x, y be two ordinates. Then after rotation by angle theta, the new ordinates are:

  2

  The same has been applied here:

  ```
  X = x*np.cos(t) - y*np.sin(t)
  Y = y*np.cos(t) + x*np.sin(t)
  ```

All in all, animations are a great tool to create amazing stuff and many more things can be created using them.

So, this was how animated plots can be generated and saved using Matplotlib.

This article is contributed by **Nikhil Kumar**. 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.

Python

XML parsing in Python

This article focuses on how one can parse a given XML file and extract some useful data out of it in a structured way.

XML: XML stands for eXtensible Markup Language. It was designed to store and transport data. It was designed to be both human- and machine-readable.That's why, the design goals of XML emphasize simplicity, generality, and usability across the Internet.
The XML file to be parsed in this tutorial is actually a RSS feed.

RSS: RSS(Rich Site Summary, often called Really Simple Syndication) uses a family of standard web feed formats to publish frequently updated informationlike blog entries, news headlines, audio, video. RSS is XML formatted plain text.

- The RSS format itself is relatively easy to read both by automated processes and by humans alike.
- The RSS processed in this tutorial is the RSS feed of top news stories from a popular news website. You can check it out here. Our goal is to process this RSS feed (or XML file) and save it in some other format for future use.

Python Module used: This article will focus on using inbuilt xml module in python for parsing XML and the main focus will be on the ElementTree XML API of this module.

Implementation:

```
#Python code to illustrate parsing of XML files
# importing the required modules
import csv
import requests
import xml.etree.ElementTree as ET

def loadRSS():

    # url of rss feed
    url = 'http://www.hindustantimes.com/rss/topnews/rssfeed.xml'

    # creating HTTP response object from given url
    resp = requests.get(url)

    # saving the xml file
    with open('topnewsfeed.xml', 'wb') as f:
        f.write(resp.content)


def parseXML(xmlfile):

    # create element tree object
    tree = ET.parse(xmlfile)

    # get root element
    root = tree.getroot()

    # create empty list for news items
    newsitems = []

    # iterate news items
    for item in root.findall('./channel/item'):

        # empty news dictionary
        news = {}

        # iterate child elements of item
        for child in item:

            # special checking for namespace object content:media
            if child.tag == '{http://search.yahoo.com/mrss/}content':
                news['media'] = child.attrib['url']
            else:
                news[child.tag] = child.text.encode('utf8')

        # append news dictionary to news items list
        newsitems.append(news)

    # return news items list
    return newsitems


def savetoCSV(newsitems, filename):

    # specifying the fields for csv file
    fields = ['guid', 'title', 'pubDate', 'description', 'link', 'media']

    # writing to csv file
    with open(filename, 'w') as csvfile:

        # creating a csv dict writer object
        writer = csv.DictWriter(csvfile, fieldnames = fields)

        # writing headers (field names)
        writer.writeheader()

        # writing data rows
        writer.writerows(newsitems)


def main():
    # load rss from web to update existing xml file
    loadRSS()
    # parse xml file
    newsitems = parseXML('topnewsfeed.xml')

    # store news items in a csv file
    savetoCSV(newsitems, 'topnews.csv')


if __name__ == "__main__":

    # calling main function
    main()
```

Above code will:

- Load RSS feed from specified URL and save it as an XML file.
- Parse the XML file to save news as a list of dictionaries where each dictionary is a single news item.
- Save the news items into a CSV file.

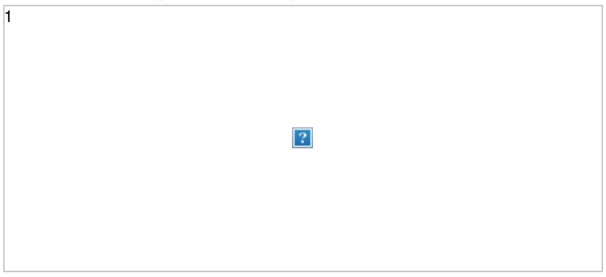
Let us try to understand the code in pieces:

- **Loading and saving RSS feed**

  ```
  def loadRSS():
      # url of rss feed
      url = 'http://www.hindustantimes.com/rss/topnews/rssfeed.xml'
      # creating HTTP response object from given url
      resp = requests.get(url)
      # saving the xml file
      with open('topnewsfeed.xml', 'wb') as f:
          f.write(resp.content)
  ```

  Here, we first created a HTTP response object by sending an HTTP request to the URL of the RSS feed. The content of response now contains the XML file data which we save as **topnewsfeed.xml** in our local directory.
  For more insight on how requests module works, follow this article:
  GET and POST requests using Python

- **Parsing XML**
  We have created **parseXML()** function to parse XML file. We know that XML is an inherently hierarchical data format, and the most natural way to represent it is with a tree. Look at the image below for example:

  1
  

  Here, we are using **xml.etree.ElementTree** (call it ET, in short) module. Element Tree has two classes for this purpose – **ElementTree** represents the whole XML document as a tree, and **Element** represents a single node in this tree. Interactions with the whole document (reading and writing to/from files) are usually done on the **ElementTree** level. Interactions with a single XML element and its sub-elements are done on the **Element** level.
  Ok, so let's go through the **parseXML()** function now:

  ```
  tree = ET.parse(xmlfile)
  ```

  Here, we create an **ElementTree** object by parsing the passed **xmlfile.**

  ```
  root = tree.getroot()
  ```

  getroot() function return the root of **tree** as an **Element** object.

  ```
  for item in root.findall('./channel/item'):
  ```

  Now, once you have taken a look at the structure of your XML file, you will notice that we are interested only in **item** element.
  ./channel/item is actually XPath syntax (XPath is a language for addressing parts of an XML document). Here, we want to find all **item** grand-children of **channel** children of the **root**(denoted by '.') element.
  You can read more about supported XPath syntax here.

  ```
  for item in root.findall('./channel/item'):

      # empty news dictionary
      news = {}

      # iterate child elements of item
      for child in item:

          # special checking for namespace object content:media
          if child.tag == '{http://search.yahoo.com/mrss/}content':
              news['media'] = child.attrib['url']
          else:
              news[child.tag] = child.text.encode('utf8')
  ```

```
        # append news dictionary to news items list
        newsitems.append(news)
```

Now, we know that we are iterating through **item** elements where each **item** element contains one news. So, we create an empty **news** dictionary in which we will store all data available about news item. To iterate though each child element of an element, we simply iterate through it, like this:

```
for child in item:
```

Now, notice a sample item element here:

1

We will have to handle namespace tags separately as they get expanded to their original value, when parsed. So, we do something like this:

```
if child.tag == '{http://search.yahoo.com/mrss/}content':
            news['media'] = child.attrib['url']
```

child.attrib is a dictionary of all the attributes related to an element. Here, we are interested in **url** attribute of **media:content** namespace tag.
Now, for all other children, we simply do:

```
news[child.tag] = child.text.encode('utf8')
```

child.tag contains the name of child element. **child.text** stores all the text inside that child element. So, finally, a sample item element is converted to a dictionary and looks like this:

```
{'description': 'Ignis has a tough competition already, from Hyun.... ,
 'guid': 'http://www.hindustantimes.com/autos/maruti-ignis-launch.... ,
 'link': 'http://www.hindustantimes.com/autos/maruti-ignis-launch.... ,
 'media': 'http://www.hindustantimes.com/rf/image_size_630x354/HT/... ,
 'pubDate': 'Thu, 12 Jan 2017 12:33:04 GMT ',
 'title': 'Maruti Ignis launches on Jan 13: Five cars that threa..... }
```

Then, we simply append this dict element to the list **newsitems**.
Finally, this list is returned.

- **Saving data to a CSV file**
  Now, we simply save the list of news items to a CSV file so that it could be used or modified easily in future using **savetoCSV()** function. To know more about writing dictionary elements to a CSV file, go through this article:
  Working with CSV files in Python

So now, here is how our formatted data looks like now:

2

As you can see, the hierarchical XML file data has been converted to a simple CSV file so that all news stories are stored in form of a table. This makes it easier to extend the database too.
Also, one can use the JSON-like data directly in their applications! This is the best alternative for extracting data from websites which do not provide a public API but provide some RSS feeds.

All the code and files used in above article can be found here.

What next?

- You can have a look at more rss feeds of the news website used in above example. You can try to create an extended version of above example by parsing other rss feeds too.
- Are you a cricket fan? Then this rss feed must be of your interest! You can parse this XML file to scrape information about the live cricket matches and use to make a desktop notifier!

Quiz of HTML and XML

This article is contributed by **Nikhil Kumar.** 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

GBlog
Project
Python

Working with PDF files in Python

All of you must be familiar with what PDFs are. In-fact, they are one of the most important and widely used digital media. PDF stands for **Portable Document Format**. It uses **.pdf** extension. It is used to present and exchange documents reliably, independent of software, hardware, or operating system.

Invented by **Adobe**, PDF is now an open standard maintained by the International Organization for Standardization (ISO). PDFs can contain links and buttons, form fields, audio, video, and business logic.

In this article, we will learn, how we can do various operations like:

- Extracting text from PDF
- Rotating PDF pages
- Merging PDFs
- Splitting PDF
- Adding watermark to PDF pages

using simple python scripts!

Installation

We will be using a third-party module, PyPDF2.

PyPDF2 is a python library built as a PDF toolkit. It is capable of:

- Extracting document information (title, author, …)
- Splitting documents page by page
- Merging documents page by page
- Cropping pages
- Merging multiple pages into a single page
- Encrypting and decrypting PDF files
- and more!

To install PyPDF2, run following command from command line:

```
pip install PyPDF2
```

This module name is case sensitive, so make sure the **y** is lowercase and everything else is uppercase. All the code and PDF files used in this tutorial/article are available here.

1. Extracting text from PDF file

```
# importing required modules
import PyPDF2

# creating a pdf file object
pdfFileObj = open('example.pdf', 'rb')

# creating a pdf reader object
pdfReader = PyPDF2.PdfFileReader(pdfFileObj)

# printing number of pages in pdf file
print(pdfReader.numPages)

# creating a page object
pageObj = pdfReader.getPage(0)

# extracting text from page
print(pageObj.extractText())
```

```
# closing the pdf file object
pdfFileObj.close()
```

Output of above program looks like this:

```
20
PythonBasics
S.R.Doty
August27,2008
Contents

1Preliminaries
4
1.1WhatisPython?...................................
..4
1.2Installationanddocumentation....................
.........4 [and some more lines...]
```

Let us try to understand the above code in chunks:

```
pdfFileObj = open('example.pdf', 'rb')
```

We opened the **example.pdf** in binary mode. and saved the file object as **pdfFileObj**.

```
pdfReader = PyPDF2.PdfFileReader(pdfFileObj)
```

Here, we create an object of **PdfFileReader** class of PyPDF2 module and pass the pdf file object & get a pdf reader object.

```
print(pdfReader.numPages)
```

numPages property gives the number of pages in the pdf file. For example, in our case, it is 20 (see first line of output).

```
pageObj = pdfReader.getPage(0)
```

Now, we create an object of **PageObject** class of PyPDF2 module. pdf reader object has function **getPage()** which takes page number (starting form index 0) as argument and returns the page object.

```
print(pageObj.extractText())
```

Page object has function **extractText()** to extract text from the pdf page.

```
pdfFileObj.close()
```

At last, we close the pdf file object.

Note: While PDF files are great for laying out text in a way that's easy for people to print and read, they're not straightforward for software to parse into plaintext. As such, PyPDF2 might make mistakes when extracting text from a PDF and may even be unable to open some PDFs at all. There isn't much you can do about this, unfortunately. PyPDF2 may simply be unable to work with some of your particular PDF files.

2. Rotating PDF pages

```
# importing the required modules
import PyPDF2

def PDFrotate(origFileName, newFileName, rotation):

    # creating a pdf File object of original pdf
    pdfFileObj = open(origFileName, 'rb')

    # creating a pdf Reader object
    pdfReader = PyPDF2.PdfFileReader(pdfFileObj)

    # creating a pdf writer object for new pdf
    pdfWriter = PyPDF2.PdfFileWriter()

    # rotating each page
    for page in range(pdfReader.numPages):

        # creating rotated page object
        pageObj = pdfReader.getPage(page)
        pageObj.rotateClockwise(rotation)

        # adding rotated page object to pdf writer
        pdfWriter.addPage(pageObj)

    # new pdf file object
    newFile = open(newFileName, 'wb')

    # writing rotated pages to new file
    pdfWriter.write(newFile)

    # closing the original pdf file object
    pdfFileObj.close()
    # closing the new pdf file object
    newFile.close()


def main():

    # original pdf file name
    origFileName = 'example.pdf'

    # new pdf file name
    newFileName = 'rotated_example.pdf'

    # rotation angle
    rotation = 270

    # calling the PDFrotate function
    PDFrotate(origFileName, newFileName, rotation)

if __name__ == "__main__":
    # calling the main function
    main()
```

Here you can see how the first page of **rotated_example.pdf** looks like (right image) after rotation:

Rotating a pdf file

Some important points related to above code:

- For rotation, we first create pdf reader object of the original pdf.

  ```
  pdfWriter = PyPDF2.PdfFileWriter()
  ```

  Rotated pages will be written to a new pdf. For writing to pdfs, we use object of **PdfFileWriter** class of PyPDF2 module.

  ```
  for page in range(pdfReader.numPages):
          pageObj = pdfReader.getPage(page)
          pageObj.rotateClockwise(rotation)
          pdfWriter.addPage(pageObj)
  ```

  Now, we iterate each page of original pdf. We get page object by **getPage()** method of pdf reader class. Now, we rotate the page by **rotateClockwise()** method of page object class. Then, we add page to pdf writer object using **addPage()** method of pdf writer class by passing the rotated page object.

  ```
  newFile = open(newFileName, 'wb')
  pdfWriter.write(newFile)
  pdfFileObj.close()
  newFile.close()
  ```

  Now, we have to write the pdf pages to a new pdf file. Firstly we open the new file object and write pdf pages to it using **write()** method of pdf writer object. Finally, we close the original pdf file object and the new file object.

3. Merging PDF files

```
# importing required modules
import PyPDF2

def PDFmerge(pdfs, output):
    # creating pdf file merger object
    pdfMerger = PyPDF2.PdfFileMerger()

    # appending pdfs one by one
    for pdf in pdfs:
        with open(pdf, 'rb') as f:
            pdfMerger.append(f)

    # writing combined pdf to output pdf file
    with open(output, 'wb') as f:
        pdfMerger.write(f)

def main():
    # pdf files to merge
    pdfs = ['example.pdf', 'rotated_example.pdf']

    # output pdf file name
    output  = 'combined_example.pdf'
```

```
    # calling pdf merge function
    PDFmerge(pdfs = pdfs, output = output)

if __name__ == "__main__":
    # calling the main function
    main()
```

Output of above program is a combined pdf, **combined_example.pdf** obtained by merging **example.pdf** and **rotated_example.pdf**.

Let us have a look at important aspects of this program:

```
pdfMerger = PyPDF2.PdfFileMerger()
```

For merging, we use a pre-built class, **PdfFileMerger** of PyPDF2 module.
Here, we create an object **pdfMerger** of pdf merger class

```
for pdf in pdfs:
        with open(pdf, 'rb') as f:
            pdfMerger.append(f)
```

Now, we append file object of each pdf to pdf merger object using **append()** method.

```
with open(output, 'wb') as f:
        pdfMerger.write(f)
```

Finally, we write the pdf pages to the output pdf file using **write** method of pdf merger object.

4. Splitting PDF file

```
# importing the required modules
import PyPDF2

def PDFsplit(pdf, splits):
    # creating input pdf file object
    pdfFileObj = open(pdf, 'rb')

    # creating pdf reader object
    pdfReader = PyPDF2.PdfFileReader(pdfFileObj)

    # starting index of first slice
    start = 0

    # starting index of last slice
    end = splits[0]


    for i in range(len(splits)+1):
        # creating pdf writer object for (i+1)th split
        pdfWriter = PyPDF2.PdfFileWriter()

        # output pdf file name
        outputpdf = pdf.split('.pdf')[0] + str(i) + '.pdf'

        # adding pages to pdf writer object
        for page in range(start,end):
            pdfWriter.addPage(pdfReader.getPage(page))

        # writing split pdf pages to pdf file
        with open(outputpdf, "wb") as f:
            pdfWriter.write(f)

        # interchanging page split start position for next split
        start = end
        try:
            # setting split end positon for next split
            end = splits[i+1]
        except IndexError:
            # setting split end position for last split
            end = pdfReader.numPages

    # closing the input pdf file object
    pdfFileObj.close()

def main():
    # pdf file to split
    pdf = 'example.pdf'

    # split page positions
    splits = [2,4]

    # calling PDFsplit function to split pdf
    PDFsplit(pdf, splits)

if __name__ == "__main__":
    # calling the main function
    main()
```

Output will be three new PDF files with **split 1 (page 0,1), split 2(page 2,3), split 3(page 4-end)**.

No new function or class has been used in above python program. Using simple logic and iterations, we created the splits of passed **pdf** according to the passed list **splits**.

5. Adding watermark to PDF pages

```
# importing the required modules
import PyPDF2

def add_watermark(wmFile, pageObj):
    # opening watermark pdf file
    wmFileObj = open(wmFile, 'rb')

    # creating pdf reader object of watermark pdf file
    pdfReader = PyPDF2.PdfFileReader(wmFileObj)

    # merging watermark pdf's first page with passed page object.
    pageObj.mergePage(pdfReader.getPage(0))

    # closing the watermark pdf file object
    wmFileObj.close()

    # returning watermarked page object
    return pageObj

def main():
    # watermark pdf file name
    mywatermark = 'watermark.pdf'

    # original pdf file name
    origFileName = 'example.pdf'

    # new pdf file name
    newFileName = 'watermarked_example.pdf'

    # creating pdf File object of original pdf
    pdfFileObj = open(origFileName, 'rb')

    # creating a pdf Reader object
    pdfReader = PyPDF2.PdfFileReader(pdfFileObj)

    # creating a pdf writer object for new pdf
    pdfWriter = PyPDF2.PdfFileWriter()

    # adding watermark to each page
    for page in range(pdfReader.numPages):
        # creating watermarked page object
        wmpageObj = add_watermark(mywatermark, pdfReader.getPage(page))

        # adding watermarked page object to pdf writer
        pdfWriter.addPage(wmpageObj)

    # new pdf file object
    newFile = open(newFileName, 'wb')

    # writing watermarked pages to new file
    pdfWriter.write(newFile)

    # closing the original pdf file object
    pdfFileObj.close()
    # closing the new pdf file object
    newFile.close()

if __name__ == "__main__":
    # calling the main function
    main()
```

Here is how first page of original (left) and watermarked (right) pdf file looks like:

- All the process is same as the page rotation example. Only difference is:

  ```
  wmpageObj = add_watermark(mywatermark, pdfReader.getPage(page))
  ```

page object is converted to watermarked page object using **add_watermark()** function.

- Let us try to understand **add_watermark()** function:

  ```
  wmFileObj = open(wmFile, 'rb')
  pdfReader = PyPDF2.PdfFileReader(wmFileObj)
  pageObj.mergePage(pdfReader.getPage(0))
  wmFileObj.close()
  return pageObj
  ```

  First of all, we create a pdf reader object of **watermark.pdf**. To the passed page object, we use **mergePage()** function and pass the page object of first page of watermark pdf reader object. This will overlay the watermark over the passed page object.

And here we reach the end of this long tutorial on working with PDF files in python.
Now, you can easily create your own PDF manager!

References:

- https://automatetheboringstuff.com/chapter13/
- https://pythonhosted.org/PyPDF2/

This article is contributed by **Nikhil Kumar**. 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

GBlog
Python

Packaging and Publishing Python code

If you have been coding in python, even for a while, you must be familiar with the concept of 'pip' by now. It is a package management system used to install and manage software packages/libraries written in Python.
Then one may ask that where are all these packages/libraries stored? It is obvious that there must be a large "on-line repository" which stores all this code. And the answer to this question is Python Package Index (PyPI).

pypi.png

PyPI is the official third-party software repository for Python. At the time of writing this article, PyPI was already hosting 95971 packages!
pip uses PyPI as the default source for packages and their dependencies. So whenever you type:

```
pip install package_name
```

pip will look for that package on PyPI and if found, it will download and install the package on your local system.

In this article, I will demonstrate how you can publish your own Python package on PyPI so that it is one-line installable and easily available to all other python users online! I will take the example of a sample package and show you the complete process. The example package is hosted on Github.

Step 1: Get the python scripts ready

First step is, of course, getting your python program (which you want to publish on PyPI) ready!

It could be any python script. Here I am using my own python script which I have named '**locator.py**' (I am using this name just for reference purpose. Feel free to save your python scripts by any name.) This file is available here.

Step 2: Getting the package-directory structure ready

This is the most important step. Now, we have to follow some pre-defined structure for our package's directory.
As a reference, feel free to check out the Github repository of the sample project which is used in this tutorial. You can clone this repository and make some modifications to create your own package.

The directory structure has to be like this:

1.PNG

Ok let's discuss what all these files will contain.

- **setup.py :** It is the most important file. It's the file where various aspects of your project are configured. The primary feature of setup.py is that it contains a global setup() function. The keyword arguments to this function are how specific details of your project are defined.
  You will need to install this setuptools library using pip:

  ```
  pip install setuptools
  ```

  Here is how my **setup.py** looks like:

  ```
  from setuptools import setup

  # reading long description from file
  with open('DESCRIPTION.txt') as file:
      long_description = file.read()


  # specify requirements of your package here
  REQUIREMENTS = ['requests']

  # some more details
  CLASSIFIERS = [
      'Development Status :: 4 - Beta',
      'Intended Audience :: Developers',
      'Topic :: Internet',
      'License :: OSI Approved :: MIT License',
      'Programming Language :: Python',
      'Programming Language :: Python :: 2',
      'Programming Language :: Python :: 2.6',
      'Programming Language :: Python :: 2.7',
      'Programming Language :: Python :: 3',
      'Programming Language :: Python :: 3.3',
      'Programming Language :: Python :: 3.4',
      'Programming Language :: Python :: 3.5',
      ]

  # calling the setup function
  setup(name='mygmap',
        version='1.0.0',
        description='A small wrapper around google maps api',
        long_description=long_description,
        url='https://github.com/nikhilkumarsingh/mygmap',
        author='Nikhil Kumar Singh',
        author_email='nikhilksingh97@gmail.com',
        license='MIT',
        packages=['geo'],
        classifiers=CLASSIFIERS,
        install_requires=REQUIREMENTS,
        keywords='maps location address'
        )
  ```

  Let us see what different arguments of setup function do:

  1. **name**: It is the name of your project. Your package will be listed by this name on PyPI.
  2. **version**: It is a string in which you can specify the current version of your project.
     It is totally your choice how you want to set the scheme of the series of versions (You may use '1.0' or '0.1' or even '0.0.1').
     This version is displayed on PyPI for each release if you publish your project. Every-time you upload a new version, you will have to change this argument as well.
  3. **description:** A short description about the package. You can use long_description argument to write long descriptions.
  4. **long_description:** We can use rich text for better description of our files. The default file format is reStructuredText . You can have a look at DESCRIPTION.txt to get a feel of the syntax.
  5. **url:** A homepage URL for your project. This makes it easier for people to follow or contribute to your project.
  6. **author, author_email:** Details about the author
  7. **license:** specify the type of license you are using.
  8. **classifiers:** It is a list of strings in which we can specify more details about our project like its development status, topic, license and supported python versions for your project. You can see more classifiers here.
  9. **install_requires:** It can be used to specify what third-party libraries your package needs to run. These dependencies will be installed by pip when someone installs your package.
  10. **keywords:** List keywords to describe your project.
- **DESCRIPTION.txt** : This file contains the long description about our package to show on the PyPI page. We use reStructuredText file format here. Check the file used in our package here.
- **LICENSE.txt**: It is a good practice to set a license for the usage of your project. You can use any of the freely available templates. Most commonly used is the **MIT** license.
  The license I am using for this project is available here.(You may replace the Author's name for using this license in your projects)
- **README.md**: This file has got nothing to do with our PyPI package. It contains description to be shown on the Github page. You can use it for PyPI page too but it will need some more modifications to our code. For now, lets keep it simple.
- **__init__.py**: The primary use of __init__.py is to initialize a python package.
  The inclusion of this file in a directory indicates to the Python interpreter that the directory should be treated like a Python package.
  You can leave this file empty.

Step 3: Create your accounts
Now, its time to create an account on PyPI and Test PyPI. Test PyPI is just a testing site where we will upload our code first to see if everything works correctly or not.

Once accounts have been made, create this **.pypirc** file in home directory of your system and enter the account details.

```
[distutils]
index-servers =
  pypi
  pypitest

[pypi]
repository=https://pypi.python.org/pypi
username= your_username
password= your_password

[pypitest]
repository=https://testpypi.python.org/pypi
username= your_username
password= your_password
```

Note*: If you are on a windows system, just type* ***echo %USERPROFILE%*** *in command prompt to know the home directory of your PC. Put the* ***.pypirc*** *file there.*

Step 4: Upload the package

Finally, we are ready to upload our package on PyPI!

- First of all, we will check if our package installs correctly on Test PyPI.
  Open command prompt/terminal in the root directory of your package.
  Run this in terminal:

  ```
  python setup.py register -r pypitest
  ```

  This will attempt to register your package against PyPI's test server, just to make sure you've set up everything correctly.

  Now, run this:

  ```
  python setup.py sdist upload -r pypitest
  ```

  You should get no errors, and should also now be able to see your library in the test PyPI repository.
- Once you've successfully uploaded to PyPI Test, perform the same steps but point to the live PyPI server instead.
  To register on PyPI, run:

  ```
  python setup.py register -r pypi
  ```

  Then, run:

  ```
  python setup.py sdist upload -r pypi
  ```

And you're all done! Your package is now publicly available on PyPI and could be easily installed by a simple pip command!
The package we created using this tutorial is available here.

Just type on terminal,

```
pip install your_package_name
```

to check if installation process is getting completed successfully.

References:

- http://peterdowns.com/posts/first-time-with-pypi.html
- https://packaging.python.org/distributing/

This article is contributed by **Nikhil Kumar**. 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

GBlog
Python

Data analysis and Visualization with Python

Python is a great language for doing data analysis, primarily because of the fantastic ecosystem of data-centric Python packages. **Pandas** is one of those packages, and makes importing and analyzing data much easier. In this article, I have used Pandas to analyze data on Country Data.csv file from UN public Data Sets of a popular 'statweb.stanford.edu' website.
As I have analyzed the Indian Country Data, I have introduced Pandas key concepts as below. Before going through this article, have a rough idea of basics from matplotlib and csv.

Installation
Easiest way to install pandas is to use pip:

```
pip install pandas
```

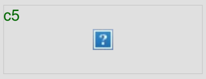
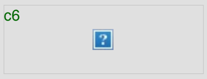
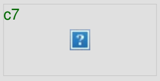
or, Download it from here

Creating A DataFrame in Pandas

Creation of dataframe is done by passing multiple Series into the DataFrame class using **pd.Series** method. Here, it is passed in the two Series objects, s1 as the first row, and s2 as the second row.
Example:

```
# assigning two series to s1 and s2
s1 = pd.Series([1,2])
s2 = pd.Series(["Ashish", "Sid"])
# framing series objects into data
df = pd.DataFrame([s1,s2])
# show the data frame
df

# data framing in another way
# taking index and column values
dframe = pd.DataFrame([[1,2],["Ashish", "Sid"]],
        index=["r1", "r2"],
        columns=["c1", "c2"])
dframe

# framing in another way
# dict-like container
dframe = pd.DataFrame({
        "c1": [1, "Ashish"],
        "c2": [2, "Sid"]})
dframe
```

Output:

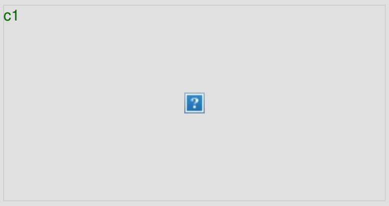
c5

c6

c7


Importing Data with Pandas

The first step is to read the data. The data is stored as a comma-separated values, or csv, file, where each row is separated by a new line, and each column by a comma (,). In order to be able to work with the data in Python, it is needed to read the csv file into a Pandas DataFrame. A DataFrame is a way to represent and work with tabular data. Tabular data has rows and columns, just like this csv file(Click Download).
Example:

```
# Import the pandas library, renamed as pd
import pandas as pd

# Read IND_data.csv into a DataFrame, assigned to df
df = pd.read_csv("IND_data.csv")

# Prints the first 5 rows of a DataFrame as default
df.head()

# Prints no. of rows and columns of a DataFrame
df.shape
```

Output:

c1


```
29,10
```

Indexing DataFrames with Pandas

Indexing can be possible using the **pandas.DataFrame.iloc** method. The iloc method allows to retrieve as many as rows and columns by position.
Examples:

```
# prints first 5 rows and every column which replicates df.head()
df.iloc[0:5,:]
# prints entire rows and columns
df.iloc[:,:]
# prints from 5th rows and first 5 columns
df.iloc[5:,:5]
```

Indexing Using Labels in Pandas

Indexing can be worked with labels using the **pandas.DataFrame.loc** method, which allows to index using labels instead of positions.
Examples:

```
# prints first five rows including 5th index and every columns of df
df.loc[0:5,:]
```

```
# prints from 5th rows onwards and entire columns
df = df.loc[5:,:]
```

The above doesn’t actually look much different from df.iloc[0:5,:]. This is because while row labels can take on any values, our row labels match the positions exactly. But column labels can make things much easier when working with data. Example:

```
# Prints the first 5 rows of Time period
# value
df.loc[:5,"Time period"]
```

c2

DataFrame Math with Pandas

Computation of data frames can be done by using Statistical Functions of pandas tools.
Examples:

```
# computes various summary statistics, excluding NaN values
df.describe()
# for computing correlations
df.corr()
# computes numerical data ranks
df.rank()
```

c4

c10

Pandas Plotting

Plots in these examples are made using standard convention for referencing the matplotlib API which provides the basics in pandas to easily create decent looking plots.
Examples:

```
# import the required module
import matplotlib.pyplot as plt
# plot a histogram
df['Observation Value'].hist(bins=10)

# shows presence of a lot of outliers/extreme values
df.boxplot(column='Observation Value', by = 'Time period')

# plotting points as a scatter plot
x = df["Observation Value"]
y = df["Time period"]
plt.scatter(x, y, label= "stars", color= "m",
            marker= "*", s=30)
# x-axis label
plt.xlabel('Observation Value')
# frequency label
plt.ylabel('Time period')
# function to show the plot
plt.show()
```

figure_1

figure_2

figure_3

Reference:

- http://pandas.pydata.org/pandas-docs/stable/tutorials.html
- https://www.datacamp.com

This article is contributed by **Afzal_Saan**. 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

Working with zip files in Python

This article explains how one can perform various operations on a zip file using a simple python program.

What is a zip file?

ZIP is an archive file format that supports lossless data compression. By lossless compression, we mean that the compression algorithm allows the original data to be perfectly reconstructed from the compressed data. So, a ZIP file is a single file containing one or more compressed files, offering an ideal way to make large files smaller and keep related files together.

Why do we need zip files?

- To reduce storage requirements.
- To improve transfer speed over standard connections.

To work on zip files using python, we will use an inbuilt python module called zipfile.

1. Extracting a zip file

```
# importing required modules
from zipfile import ZipFile

# specifying the zip file name
file_name = "my_python_files.zip"

# opening the zip file in READ mode
with ZipFile(file_name, 'r') as zip:
    # printing all the contents of the zip file
    zip.printdir()

    # extracting all the files
    print('Extracting all the files now...')
    zip.extractall()
    print('Done!')
```

The above program extracts a zip file named “my_python_files.zip” in the same directory as of this python script.
The output of above program may look like this:

zip3

Let us try to understand the above code in pieces:

```
from zipfile import ZipFile
```

ZipFile is a class of zipfile module for reading and writing zip files. Here we import only class ZipFile from zipfile module.

```
with ZipFile(file_name, 'r') as zip:
```

Here, a ZipFile object is made by calling ZipFile constructor which accepts zip file name and mode parameters. We create a ZipFile object in **READ** mode and name it as **zip**.

```
zip.printdir()
```

printdir() method prints a table of contents for the archive.

```
zip.extractall()
```

extractall() method will extract all the contents of the zip file to the current working directory. You can also call **extract()** method to extract any file by specifying its path in the zip file.
For example:

```
zip.extract('python_files/python_wiki.txt')
```

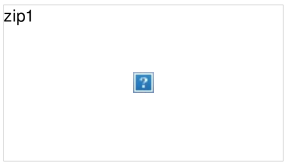
This will extract only the specified file.

If you want to read some specific file, you can go like this:

```
data = zip.read(name_of_file_to_read)
```

2. Writing to a zip file

Consider a directory (folder) with such a format:



Here, we will need to crawl whole directory and its sub-directories in order to get a list of all file-paths before writing them to a zip file.
The following program does this by crawling the directory to be zipped:

```
# importing required modules
from zipfile import ZipFile
import os

def get_all_file_paths(directory):

    # initializing empty file paths list
    file_paths = []

    # crawling through directory and subdirectories
    for root, directories, files in os.walk(directory):
        for filename in files:
            # join the two strings in order to form the full filepath.
            filepath = os.path.join(root, filename)
            file_paths.append(filepath)

    # returning all file paths
    return file_paths

def main():
    # path to folder which needs to be zipped
    directory = './python_files'

    # calling function to get all file paths in the directory
    file_paths = get_all_file_paths(directory)

    # printing the list of all files to be zipped
    print('Following files will be zipped:')
    for file_name in file_paths:
        print(file_name)

    # writing files to a zipfile
    with ZipFile('my_python_files.zip','w') as zip:
        # writing each file one by one
        for file in file_paths:
            zip.write(file)

    print('All files zipped successfully!')


if __name__ == "__main__":
    main()
```

The output of above program looks like this:

zip2

Let us try to understand above code by dividing into fragments:

```
def get_all_file_paths(directory):
    file_paths = []

    for root, directories, files in os.walk(directory):
        for filename in files:
            filepath = os.path.join(root, filename)
            file_paths.append(filepath)

    return file_paths
```

First of all, to get all file paths in our directory, we have created this function which uses the **os.walk()** method. In each iteration, all files present in that directory are appended to a list called **file_paths**.
In the end, we return all the file paths.

```
file_paths = get_all_file_paths(directory)
```

Here we pass the directory to be zipped to the **get_all_file_paths()** function and obtain a list containing all file paths.

```
with ZipFile('my_python_files.zip','w') as zip:
```

Here, we create a ZipFile object in WRITE mode this time.

```
for file in file_paths:
            zip.write(file)
```

Here, we write all the files to the zip file one by one using **write** method.

3. Getting all information about a zip file

```
# importing required modules
from zipfile import ZipFile
import datetime

# specifying the zip file name
file_name = "example.zip"

# opening the zip file in READ mode
with ZipFile(file_name, 'r') as zip:
    for info in zip.infolist():
            print(info.filename)
            print('\tModified:\t' + str(datetime.datetime(*info.date_time)))
            print('\tSystem:\t\t' + str(info.create_system) + '(0 = Windows, 3 = Unix)')
            print('\tZIP version:\t' + str(info.create_version))
            print('\tCompressed:\t' + str(info.compress_size) + ' bytes')
            print('\tUncompressed:\t' + str(info.file_size) + ' bytes')
```

The output of above program may look like this:

zip4

```
for info in zip.infolist():
```

Here, **infolist()** method creates an instance of **ZipInfo** class which contains all the information about the zip file.
We can access all information like last modification date of files, file names, system on which files were created, Zip version, size of files in compressed and uncompressed form, etc.

This article is contributed by **Nikhil Kumar**. 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.

Working with csv files in Python

This article explains how to load and parse a CSV file in Python.

First of all, what is a CSV ?

CSV (Comma Separated Values) is a simple **file format** used to store tabular data, such as a spreadsheet or database. A CSV file stores tabular data (numbers and text) in plain text. Each line of the file is a data record. Each record consists of one or more fields, separated by commas. The use of the comma as a field separator is the source of the name for this file format.

For working CSV files in python, there is an inbuilt module called **csv**.

Reading a CSV file

```
# importing csv module
import csv

# csv file name
filename = "aapl.csv"

# initializing the titles and rows list
fields = []
rows = []

# reading csv file
with open(filename, 'r') as csvfile:
    # creating a csv reader object
    csvreader = csv.reader(csvfile)

    # extracting field names through first row
    fields = csvreader.next()

    # extracting each data row one by one
    for row in csvreader:
        rows.append(row)

    # get total number of rows
    print("Total no. of rows: %d"%(csvreader.line_num))

# printing the field names
print('Field names are:' + ', '.join(field for field in fields))

#  printing first 5 rows
print('\nFirst 5 rows are:\n')
for row in rows[:5]:
    # parsing each column of a row
    for col in row:
        print("%10s"%col),
    print('\n')
```

The output of above program looks like this:

csv1

The above example uses a CSV file aapl.csv which can be downloaded from here.
Run this program with the aapl.csv file in same directory.

Let us try to understand this piece of code.

```
with open(filename, 'r') as csvfile:
    csvreader = csv.reader(csvfile)
```

Here, we first open the CSV file in READ mode. The file object is named as **csvfile**. The file object is converted to csv.reader object. We save the csv.reader object as **csvreader**.

```
fields = csvreader.next()
```

csvreader is an iterable object. Hence, .next() method returns the current row and advances the iterator to the next row. Since the first row of our csv file contains the headers (or field names), we save them in a list called **fields**.

```
for row in csvreader:
        rows.append(row)
```

Now, we iterate through remaining rows using a for loop. Each row is appended to a list called **rows**. If you try to print each row, one can find that row is nothing but a list containing all the field values.

```
print("Total no. of rows: %d"%(csvreader.line_num))
```

csvreader.line_num is nothing but a counter which returns the number of rows which have been iterated.

Writing to a CSV file

```
# importing the csv module
import csv

# field names
fields = ['Name', 'Branch', 'Year', 'CGPA']

# data rows of csv file
rows = [ ['Nikhil', 'COE', '2', '9.0'],
        ['Sanchit', 'COE', '2', '9.1'],
        ['Aditya', 'IT', '2', '9.3'],
        ['Sagar', 'SE', '1', '9.5'],
        ['Prateek', 'MCE', '3', '7.8'],
        ['Sahil', 'EP', '2', '9.1']]

# name of csv file
filename = "university_records.csv"

# writing to csv file
with open(filename, 'w') as csvfile:
    # creating a csv writer object
    csvwriter = csv.writer(csvfile)

    # writing the fields
    csvwriter.writerow(fields)

    # writing the data rows
    csvwriter.writerows(rows)
```

Let us try to understand the above code in pieces.

- **fields** and **rows** have been already defined. fields is a list containing all the field names. **rows** is a list of lists. Each row is a list containing the field values of that row.

```
with open(filename, 'w') as csvfile:
    csvwriter = csv.writer(csvfile)
```

Here, we first open the CSV file in WRITE mode. The file object is named as **csvfile**. The file object is converted to csv.writer object. We save the csv.writer object as **csvwriter**.

```
csvwriter.writerow(fields)
```

Now we use **writerow** method to write the first row which is nothing but the field names.

```
 csvwriter.writerows(rows)
```

We use **writerows** method to write multiple rows at once.

Writing a dictionary to a CSV file

```
# importing the csv module
import csv

# my data rows as dictionary objects
mydict =[{'branch': 'COE', 'cgpa': '9.0', 'name': 'Nikhil', 'year': '2'},
        {'branch': 'COE', 'cgpa': '9.1', 'name': 'Sanchit', 'year': '2'},
        {'branch': 'IT', 'cgpa': '9.3', 'name': 'Aditya', 'year': '2'},
        {'branch': 'SE', 'cgpa': '9.5', 'name': 'Sagar', 'year': '1'},
        {'branch': 'MCE', 'cgpa': '7.8', 'name': 'Prateek', 'year': '3'},
        {'branch': 'EP', 'cgpa': '9.1', 'name': 'Sahil', 'year': '2'}]

# field names
fields = ['name', 'branch', 'year', 'cgpa']

# name of csv file
filename = "university_records.csv"

# writing to csv file
with open(filename, 'w') as csvfile:
    # creating a csv dict writer object
    writer = csv.DictWriter(csvfile, fieldnames = fields)

    # writing headers (field names)
    writer.writeheader()

    # writing data rows
    writer.writerows(mydict)
```

In this example, we write a dictionary **mydict** to a CSV file.

```
with open(filename, 'w') as csvfile:
    writer = csv.DictWriter(csvfile, fieldnames = fields)
```

Here, the file object (**csvfile**) is converted to a DictWriter object.
Here, we specify the **fieldnames** as an argument.

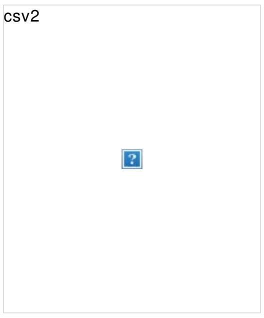
```
writer.writeheader()
```

writeheader method simply writes the first row of your csv file using the pre-specified fieldnames.

```
writer.writerows(mydict)
```

writerows method simply writes all the rows but in each row, it writes only the values(not keys).

So, in the end, our CSV file looks like this:


csv2

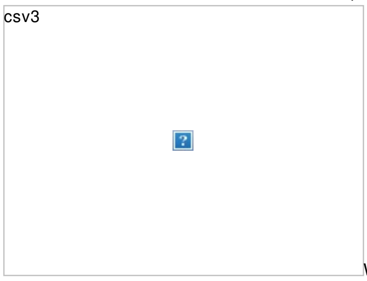
Important Points:

- In csv modules, an optional *dialect* parameter can be given which is used to define a set of parameters specific to a particular *CSV format*. By default, csv module uses *excel* dialect which makes them compatible with excel spreadsheets. You can define your own dialect using **register_dialect** method.
  Here is an example:

```
 csv.register_dialect(
'mydialect',
delimiter = ',',
quotechar = '"',
doublequote = True,
skipinitialspace = True,
lineterminator = '\r\n',
quoting = csv.QUOTE_MINIMAL)
```

Now, while defining a csv.reader or csv.writer object, we can specify the dialect like
this:

```
csvreader = csv.reader(csvfile, dialect='mydialect')
```

- Now, consider that a CSV file looks like this in plain-text:
  
  csv3
  We notice that the delimiter is not a comma but a semi-colon. Also, the rows are separated by two newlines instead of one. In such cases, we can specify the delimiter and line terminator as follows:

```
csvreader = csv.reader(csvfile, delimiter = ';', lineterminator = '\n\n')
```

So, this was a brief, yet concise discussion on how to load and parse CSV files in a python program.

This blog is contributed by **Nikhil Kumar**. 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

GBlog
Python

Downloading files from web using Python

Requests is a versatile HTTP library in python with various applications. One of its applications is to download a file from web using the file URL.

Installation: First of all, you would need to download the requests library. You can directly install it using pip by typing following command:

```
pip install requests
```

Or download it directly from here and install manually.

Downloading files

```
# imported the requests library
import requests
image_url = "https://www.python.org/static/community_logos/python-logo-master-v3-TM.png"

# URL of the image to be downloaded is defined as image_url
r = requests.get(image_url) # create HTTP response object

# send a HTTP request to the server and save
# the HTTP response in a response object called r
with open("python_logo.png",'wb') as f:

    # Saving received content as a png file in
    # binary format

    # write the contents of the response (r.content)
    # to a new file in binary mode.
    f.write(r.content)
```

This small piece of code written above will download the following image from the web. Now check your local directory(the folder where this script resides), and you will find this image:

All we need is the URL of the image source. (You can get the URL of image source by right-clicking on the image and selecting the View Image option.)

Download large files

The HTTP response content (**r.content**) is nothing but a string which is storing the file data. So, it won't be possible to save all the data in a single string in case of large files. To overcome this problem, we do some changes to our program:

- Since all file data can't be stored by a single string, we use **r.iter_content** method to load data in chunks, specifying the chunk size.

```
r = requests.get(URL, stream = True)
```

Setting **stream** parameter to **True** will cause the download of response headers only and the connection remains open. This avoids reading the content all at once into memory for large responses. A fixed chunk will be loaded each time while **r.iter_content** is iterated.

Here is an example:

```
import requests
file_url = "http://codex.cs.yale.edu/avi/db-book/db4/slide-dir/ch1-2.pdf"

r = requests.get(file_url, stream = True)

with open("python.pdf","wb") as pdf:
    for chunk in r.iter_content(chunk_size=1024):

         # writing one chunk at a time to pdf file
         if chunk:
             pdf.write(chunk)
```

Downloading Videos

In this example, we are interested in downloading all the video lectures available on this web-page. All the archives of this lecture are available here. So, we first scrape the webpage to extract all video links and then download the videos one by one.

```
import requests
from bs4 import BeautifulSoup

'''
URL of the archive web-page which provides link to
all video lectures. It would have been tiring to
download each video manually.
In this example, we first crawl the webpage to extract
all the links and then download videos.
'''

# specify the URL of the archive here
archive_url = "http://www-personal.umich.edu/~csev/books/py4inf/media/"

def get_video_links():

    # create response object
    r = requests.get(archive_url)
    # create beautiful-soup object
    soup = BeautifulSoup(r.content,'html5lib')

    # find all links on web-page
    links = soup.findAll('a')

    # filter the link sending with .mp4
    video_links = [archive_url + link['href'] for link in links if link['href'].endswith('mp4')]

    return video_links


def download_video_series(video_links):

    for link in video_links:

        '''iterate through all links in video_links
        and download them one by one'''

        # obtain filename by splitting url and getting
        # last string
        file_name = link.split('/')[-1]

        print "Downloading file:%s"%file_name

        # create response object
        r = requests.get(link, stream = True)

        # download started
        with open(file_name, 'wb') as f:
            for chunk in r.iter_content(chunk_size = 1024*1024):
                if chunk:
                    f.write(chunk)

        print "%s downloaded!\n"%file_name

    print "All videos downloaded!"
    return


if __name__ == "__main__":

    # getting all video links
    video_links = get_video_links()

    # download all videos
    download_video_series(video_links)
```

Advantages of using Requests library to download web files are:

- One can easily download the web directories by iterating recursively through the website!
- This is a browser-independent method and much faster!
- One can simply scrape a web page to get all the file URLs on a webpage and hence, download all files in a single command- Implementing Web Scraping in Python with BeautifulSoup

This blog is contributed by Nikhil Kumar. 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

GBlog
Python

Whatsapp using Python!

Have you ever wished to automatically wish your friends on their birthdays, or send a set of messages to your friend (or any Whastapp contact!) automatically at a pre-set time, or send your friends by sending thousands of random text on whatsapp! Using **Browser Automation** you can do all of it and much more!
First you must install these:-

1) **Python Bindings for Selenium (Browser Automation software)**

```
pip install selenium
```

2) **Chrome webdriver**
Download Chrome driver from here: Chromedriver download page(choose your specific version)
Extract it in a known location , as **we need the location later**

If you get stuck somewhere, Refer To the documentation: Documentation link

3) **Chromium Web Browser(Open source version of chrome browser)**

```
sudo apt-get install chromium-browser
```

That's it! You are all set.

Lets dive in right away-

```
from selenium import webdriver
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
from selenium.webdriver.common.keys import Keys
from selenium.webdriver.common.by import By
import time

# Replace below path with the absolute path
# to chromedriver in your computer
driver = webdriver.Chrome('/home/saket/Downloads/chromedriver')

driver.get("https://web.whatsapp.com/")
wait = WebDriverWait(driver, 600)

# Replace 'Friend's Name' with the name of your friend
# or the name of a group
target = '"Friend\'s Name"'

# Replace the below string with your own message
string = "Message sent using Python!!!"

x_arg = '//span[contains(@title,' + target + ')]'
group_title = wait.until(EC.presence_of_element_located((
    By.XPATH, x_arg)))
group_title.click()
inp_xpath = '//div[@class="input"][@dir="auto"][@data-tab="1"]'
input_box = wait.until(EC.presence_of_element_located((
    By.XPATH, inp_xpath)))
for i in range(100):
    input_box.send_keys(string + Keys.ENTER)
    time.sleep(1)
```

Keep your mobile phone with you. Choose whatsapp web from the top bar in whatsapp(3 dots)

Screenshot2

Then Run the script **(make sure that you have added the absolute path for chromedriver and have replaced target variable with your friends name).** Scan the QR code that appears on the screen and enjoy the power of python!

Screenshot3

Please use this script only for educational purposes, i am not responsible if your friends (or even Whatsapp) block you.

Feel free to modify the code. Try to :

1. Text Multiple Groups at once
2. Send the messages from a predefined list of messages randomly or
3. Send complete random text.

Comment below about your experience!

When it comes to browser automation, this is just the tip of the iceberg. Will write more articles on browser automation to give you a glimpse of its power!

Python
TechTips

Opencv Python Program for face detection

The objective of the program given is to detect object of interest(face) in real time and to keep tracking of the same object.This is a simple example of how to detect face in Python. You can try to use training samples of any other object of your choice to be detected by training the classifier on required objects.

Here is the steps to download the requirements below.

Steps:

1. Download Python 2.7.x version, numpy and Opencv 2.7.x version.Check if your Windows either 32 bit or 64 bit is compatible and install accordingly.
2. Make sure that numpy is running in your python then try to install opencv.
3. Put the haarcascade_eye.xml & haarcascade_frontalface_default.xml files in the same folder(links given in below code).

Implementation

```
# OpenCV program to detect face in real time
# import libraries of python OpenCV
# where its functionality resides
import cv2

# load the required trained XML classifiers
# https://github.com/Itseez/opencv/blob/master/
# data/haarcascades/haarcascade_frontalface_default.xml
# Trained XML classifiers describes some features of some
# object we want to detect a cascade function is trained
# from a lot of positive(faces) and negative(non-faces)
# images.
face_cascade = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')

# https://github.com/Itseez/opencv/blob/master
# /data/haarcascades/haarcascade_eye.xml
# Trained XML file for detecting eyes
eye_cascade = cv2.CascadeClassifier('haarcascade_eye.xml')

# capture frames from a camera
cap = cv2.VideoCapture(0)

# loop runs if capturing has been initialized.
while 1:

 # reads frames from a camera
 ret, img = cap.read()

 # convert to gray scale of each frames
 gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

 # Detects faces of different sizes in the input image
 faces = face_cascade.detectMultiScale(gray, 1.3, 5)

 for (x,y,w,h) in faces:
  # To draw a rectangle in a face
  cv2.rectangle(img,(x,y),(x+w,y+h),(255,255,0),2)
  roi_gray = gray[y:y+h, x:x+w]
  roi_color = img[y:y+h, x:x+w]

  # Detects eyes of different sizes in the input image
  eyes = eye_cascade.detectMultiScale(roi_gray)

  #To draw a rectangle in eyes
  for (ex,ey,ew,eh) in eyes:
   cv2.rectangle(roi_color,(ex,ey),(ex+ew,ey+eh),(0,127,255),2)

 # Display an image in a window
 cv2.imshow('img',img)

 # Wait for Esc key to stop
 k = cv2.waitKey(30) & 0xff
 if k == 27:
  break

# Close the window
cap.release()

# De-allocate any associated memory usage
cv2.destroyAllWindows()
```

Output:

References:

- https://www.youtube.com/v=WfdYYNamHZ8
- http://docs.opencv.org/2.4/modules/objdetect/doc/cascade_classification.html?highlight=cascadeclassifier#cascadeclassifier
- http://www.multimedia-computing.de/mediawiki//images/5/52/MRL-TR-May02-revised-Dec02.pdf

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

Project
Python
Image-Processing
OpenCV

Implementing Web Scraping in Python with BeautifulSoup

There are mainly two ways to extract data from a website:

- Use the API of the website (if it exists). For example, Facebook has the Facebook Graph API which allows retrieval of data posted on Facebook.
- Access the HTML of the webpage and extract useful information/data from it. This technique is called web scraping or web harvesting or web data extraction.

This article discusses the steps involved in web scraping using implementation of Web Scraping in Python with Beautiful Soup

Steps involved in web scraping:

1. Send a HTTP request to the URL of the webpage you want to access. The server responds to the request by returning the HTML content of the webpage. For this task, we will use a third-party HTTP library for python requests.
2. Once we have accessed the HTML content, we are left with the task of parsing the data. Since most of the HTML data is nested, we cannot extract data simply through string processing. One needs a parser which can create a nested/tree structure of the HTML data.
   There are many HTML parser libraries available but the most advanced one is html5lib.
3. Now, all we need to do is navigating and searching the parse tree that we created, i.e. tree traversal. For this task, we will be using another third-party python library, Beautiful Soup. It is a Python library for pulling data out of HTML and XML files.

Step 1: Installing the required third-party libraries

- Easiest way to install external libraries in python is to use pip. **pip** is a package management system used to install and manage software packages written in Python.
  All you need to do is:

```
pip install requests
pip install html5lib
pip install bs4
```

- Another way is to download them manually from these links:
  - requests
  - html5lib
  - beautifulsoup4

Step 2: Accessing the HTML content from webpage

```
import requests
URL = "http://www.geeksforgeeks.org/data-structures/"
r = requests.get(URL)
print(r.content)
```

Let us try to understand this piece of code.

- First of all import the requests library.
- Then, specify the URL of the webpage you want to scrape.
- Send a HTTP request to the specified URL and save the response from server in a response object called r.
- Now, as print r.content to get the **raw HTML content** of the webpage. It is of 'string' type.

Step 3: Parsing the HTML content

```
#This will not run on online IDE
import requests
from bs4 import BeautifulSoup

URL = "http://www.values.com/inspirational-quotes"
r = requests.get(URL)

soup = BeautifulSoup(r.content, 'html5lib')
print(soup.prettify())
```

A really nice thing about BeautifulSoup library is that it is build on the top of the HTML parsing libraries like html5lib, lxml, html.parser, etc. So BeautifulSoup object and specify the parser library can be created at the same time.

In the example above,

```
soup = BeautifulSoup(r.content, 'html5lib')
```

We create a BeautifulSoup object by passing two arguments:

- **r.content** : It is the raw HTML content.
- **html5lib** : Specifying the HTML parser we want to use.

Now **soup.prettify()** is printed, it gives the visual representation of the parse tree created from the raw HTML content.

Step 4: Searching and navigating through the parse tree

Now, we would like to extract some useful data from the HTML content. The soup object contains all the data in nested structure which could be programmatically extracted. In our example, we are scraping a webpage consisting some quotes. So, we would like to create a program to save those quotes (and all relevant information about them).

```
#Python program to scrape website
#and save quotes from website
import requests
from bs4 import BeautifulSoup
import csv

URL = "http://www.values.com/inspirational-quotes"
r = requests.get(URL)

soup = BeautifulSoup(r.content, 'html5lib')

quotes=[]  # a list to store quotes

table = soup.find('div', attrs = {'id':'container'})

for row in table.findAll('div', attrs = {'class':'quote'}):
    quote = {}
    quote['theme'] = row.h5.text
    quote['url'] = row.a['href']
    quote['img'] = row.img['src']
    quote['lines'] = row.h6.text
    quote['author'] = row.p.text
    quotes.append(quote)

filename = 'inspirational_quotes.csv'
with open(filename, 'wb') as f:
    w = csv.DictWriter(f,['theme','url','img','lines','author'])
    w.writeheader()
    for quote in quotes:
        w.writerow(quote)
```

Before moving on, we recommend you to go through the HTML content of the webpage which we printed using soup.prettify() method and try to find a pattern or a way to navigate to the quotes.

- It is noticed that all the quotes are inside a div container whose id is container. So, we find that div element (termed as table in above code) using **find()** method :

  ```
  table = soup.find('div', attrs = {'id':'container'})
  ```

  The first argument is the HTML tag you want to search and second argument is a dictionary type element to specify the additional attributes associated with that tag. **find()** method returns the first matching element. You can try to print **table.prettify()** to get a sense of what this piece of code does.

- Now, in the table element, one can notice that each quote is inside a div container whose class is quote. So, we iterate through each div container whose class is quote. Here, we use findAll() method which is similar to find method in terms of arguments but it returns a list of all matching elements. Each quote is now iterated using a variable called **row.**

  Here is one sample row HTML content for better understanding:
  Now consider this piece of code:

  ```
  for row in table.findAll('div', attrs = {'class':'quote'}):
      quote = {}
      quote['theme'] = row.h5.text
      quote['url'] = row.a['href']
      quote['img'] = row.img['src']
      quote['lines'] = row.h6.text
      quote['author'] = row.p.text
      quotes.append(quote)
  ```

  We create a dictionary to save all information about a quote. The nested structure can be accessed using dot notation. To access the text inside an HTML element, we use **.text :**

  ```
  quote['theme'] = row.h5.text
  ```

  We can add, remove, modify and access a tag's attributes. This is done by treating the tag as a dictionary:

  ```
  quote['url'] = row.a['href']
  ```

  Lastly, all the quotes are appended to the list called **quotes.**

- Finally, we would like to save all our data in some CSV file.

  ```
  filename = 'inspirational_quotes.csv'
  ```

```
with open(filename, 'wb') as f:
    w = csv.DictWriter(f,['theme','url','img','lines','author'])
    w.writeheader()
    for quote in quotes:
        w.writerow(quote)
```

Here we create a CSV file called inspirational_quotes.csv and save all the quotes in it for any further use.

So, this was a simple example of how to create a web scraper in Python. From here, you can try to scrap any other website of your choice. In case of any queries, post them below in comments section.

Note : Web Scraping is considered as illegal in many cases. It may also also cause your IP to be blocked permanently by a website.

This blog is contributed by **Nikhil Kumar**. 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

GBlog
Project
Python

Implementing Artificial Neural Network training process in Python

An Artificial Neural Network (ANN) is an information processing paradigm that is inspired the brain. ANNs, like people, learn by example. An ANN is configured for a specific application, such as pattern recognition or data classification, through a learning process. Learning largely involves adjustments to the synaptic connections that exist between the neurons. Artificial neural network

The brain consists of hundreds of billion of cells called neurons. These neurons are connected together by synapses which are nothing but the connections across which a neuron can send an impulse to another neuron. When a neuron sends an excitatory signal to another neuron, then this signal will be added to all of the other inputs of that neuron. If it exceeds a given threshold then it will cause the target neuron to fire an action signal forward — this is how the thinking process works internally.

In Computer Science, we model this process by creating “networks” on a computer using matrices. These networks can be understood as abstraction of neurons without all the biological complexities taken into account. To keep things simple, we will just model a simple NN, with two layers capable of solving linear classification problem.

Artificial neural network1

Let’s say we have a problem where we want to predict output given a set of inputs and outputs as training example like so:

Artificial neural network2

Note that the output is directly related to third column i.e. the values of input 3 is what the output is in every training example in fig. 2. So for the test example output value should be 1.

The training process consists of the following steps:

1. **Forward Propagation:**
   Take the inputs, multiply by the weights (just use random numbers as weights)

Let $Y = W_iI_i = W_1I_1+W_2I_2+W_3I_3$

Pass the result through a sigmoid formula to calculate the neuron's output. The Sigmoid function is used to normalise the result between 0 and 1:

$1/1 + e^{-y}$

2. **Back Propagation**

   Calculate the error i.e the difference between the actual output and the expected output. Depending on the error, adjust the weights by multiplying the error with the input and again with the gradient of the Sigmoid curve:

   Weight += Error Input Output (1-Output) ,here Output (1-Output) is derivative of sigmoid curve.

Note: Repeat the whole process for a few thousands iterations.

Let's code up the whole process in Python. We'll be using Numpy library to help us with all the calculations on matrices easily. You'd need to install numpy library on your system to run the code

Command to install numpy:

```
sudo apt -get install python-numpy
```

Implementation:

```
from numpy import *

class NeuralNet(object):
    def __init__(self):
        # Generate random numbers
        random.seed(1)

        # Assign random weights to a 3 x 1 matrix,
        self.synaptic_weights = 2 * random.random((3, 1)) - 1

    # The Sigmoid function
    def __sigmoid(self, x):
        return 1 / (1 + exp(-x))

    # The derivative of the Sigmoid function.
    # This is the gradient of the Sigmoid curve.
    def __sigmoid_derivative(self, x):
        return x * (1 - x)

    # Train the neural network and adjust the weights each time.
    def train(self, inputs, outputs, training_iterations):
        for iteration in xrange(training_iterations):

            # Pass the training set through the network.
            output = self.learn(inputs)

            # Calculate the error
            error = outputs - output

            # Adjust the weights by a factor
            factor = dot(inputs.T, error * self.__sigmoid_derivative(output))
            self.synaptic_weights += factor

    # The neural network thinks.
    def learn(self, inputs):
        return self.__sigmoid(dot(inputs, self.synaptic_weights))

if __name__ == "__main__":

    #Initialize
    neural_network = NeuralNet()

    # The training set.
    inputs = array([[0, 1, 1], [1, 0, 0], [1, 0, 1]])
    outputs = array([[1, 0, 1]]).T

    # Train the neural network
    neural_network.train(inputs, outputs, 10000)

    # Test the neural network with a test example.
    print neural_network.learn(array([1, 0, 1]))
```

Expected Output: After 10 iterations our neural network predicts the value to be 0.65980921. It looks not good as the answer should really be 1. If we increase the number of iterations to 100, we get 0.87680541. Our network is getting smarter! Subsequently, for 10000 iterations we get 0.9897704 which is pretty close and indeed a satisfactory output.

References:

- NEURAL NETWORKS by Christos Stergiou and Dimitrios Siganos
- Fundamentals of Deep Learning – Starting with Artificial Neural Network
- Tinker With a Neural Network Right Here in Your Browser
- Neural Networks Demystified

Simple Multithreaded Download Manager in Python

Introduction

A Download Manager is basically a computer program dedicated to the task of downloading stand alone files from internet. Here, we are going to create a simple Download Manager with the help of threads in Python. Using multi-threading a file can be downloaded in the form of chunks simultaneously from different threads. To implement this, we are going to create simple command line tool which accepts the URL of the file and then downloads it.

Prerequisites
Windows machine with Python installed.

Setup
Download the below mentioned packages from command prompt.

1. **Click package:** Click is a Python package for creating beautiful command line interfaces with as little code as necessary. It's the "Command Line Interface Creation Kit".
   ```
   pip install click
   ```
2. **Requests package:** In this tool, we are going to download a file based on the URL(HTTP addresses). Requests is an HTTP Library written in Python which allows you to send HTTP requests. You can add headers, form data, multi-part files, and parameters with simple Python dictionaries and access the response data in the same way.
   ```
   pip install requests
   ```
3. **threading package:** To work with threads we need threading package.
   ```
   pip install threading
   ```

Implementation

(**Note:** The program has been split into parts to make it easy to understand. Make sure that you are not missing any part of the code while running the code.)

- Create new python file in editor
- First import the required packages to that you'll need to write

```
#N ote: This code will not work on online IDE

# Importing the required packages
import click
import requests
import threading

# The below code is used for each chunk of file handled
# by each thread for downloading the content from specified
# location to storage
def Handler(start, end, url, filename):

    # specify the starting and ending of the file
    headers = {'Range': 'bytes=%d-%d' % (start, end)}

    # request the specified part and get into variable
    r = requests.get(url, headers=headers, stream=True)

    # open the file and write the content of the html page
    # into file.
    with open(filename, "r+b") as fp:

        fp.seek(start)
        var = fp.tell()
        fp.write(r.content)
```

Now we are going to implement the actual functionality of in the download_file function.

- The first step is to decorate the function with click.command() so that we can add command line arguments. We can also give options for respective commands.
- For our implementation upon entering command –help we are going to display the options that can be used. In our program there are two options that can be used. One is "number_of_threads" and the other is "name". By default "number_of_threads" is taken as 4. To change it, we can specify while running the program.
- "name" option is given so that we can give our own name to the file that is going to be downloaded. The arguments for the function can be specified using click.argument().
- For our program we need to give the URL of the file we want to download.

```
#Note: This code will not work on online IDE

@click.command(help="It downloads the specified file with specified name")
@click.option('—number_of_threads',default=4, help="No of Threads")
@click.option('--name',type=click.Path(),help="Name of the file with extension")
@click.argument('url_of_file',type=click.Path())
@click.pass_context
def download_file(ctx,url_of_file,name,number_of_threads):
```

The following code goes under the "download_file" function.

- In this function we first check for the “name”. If the “name” is not given then use the name from url.
- Next step is to connect to the URL and Get the content and size of the content.

```
r = requests.head(url_of_file)
if name:
    file_name = name
else:
    file_name = url_of_file.split('/')[-1]
try:
    file_size = int(r.headers['content-length'])
except:
    print "Invalid URL"
    return
```

Create file with size of the content

```
part = int(file_size) / number_of_threads
fp = open(file_name, "wb")
fp.write('\0' * file_size)
fp.close()
```

Now we create Threads and pass the Handler function which has the main functionality :

```
for i in range(number_of_threads):
    start = part * i
    end = start + part

       # create a Thread with start and end locations
       t = threading.Thread(target=Handler,
               kwargs={'start': start, 'end': end, 'url': url_of_file, 'filename': file_name})
       t.setDaemon(True)
       t.start()
```

Finally join the threads and call the “download_file” function from main

```
main_thread = threading.current_thread()
for t in threading.enumerate():
    if t is main_thread:
        continue
    t.join()
print '%s downloaded' % file_name

if __name__ == '__main__':
    download_file(obj={})
```

We are done with coding part and now follow the commands showed below to run the .py file.

1

```
“python filename.py” –-help
```

This command shows the “Usage” of the click command tool and options that the tool can accepts.
Below is the sample command where we try to download an jpg image file from a URL and also gave a name and number_of_threads.

2

Finally, we are successfully done with it and this is one of the way to build a simple multithreaded download manager in Python.

This article is contributed by **Rahul Bojanapally**. 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

GBlog
Python

Create a Website Alarm Using Python

We use Bookmarks to remember the important websites which we think we will use them very often in future.
Here’s a simple python code which takes the URL of the website as its first input and the time you want to open it as the second input.

As the time reaches your given value of the time, it’ll open the URL that you requested in the web browser automatically.
In this code, we’ll import two python modules – Time and Webbrowser.

```
# Import the time module, it provides various time-related
# functions.
import time

# Import the webbrowser module, it is used to
# display various HTML documents to the user.
import webbrowser

# First Input: It is the time of the form
# 'Hours:Minutes:Seconds' that you'll
# use to set the alarm.
Set_Alarm = raw_input("Set the website alarm as H:M:S:")

# Second Input: It is the URL that you want
# to open on the given time.
url = raw_input("Enter the website you want to open:")

# This is the actual time that we will use to print.
Actual_Time = time.strftime("%I:%M:%S")

# This is the while loop that'll print the time
# until it's value will be equal to the alarm time
while (Actual_Time != Set_Alarm):
    print "The time is " + Actual_Time
    Actual_Time = time.strftime("%I:%M:%S")
    time.sleep(1)


# This if statement plays the role when its the
# alarm time and executes the code within it.
if (Actual_Time == Set_Alarm):
    print "You should see your webpage now :-)"

    # We are calling the open()
    # function from the webrowser module.
    webbrowser.open(url)
```

You can contribute to the code by going here.

This article is contributed by **Shivam Sharma**. 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.

You can contribute to the code by going here.

GATE CS Corner   Company Wise Coding Practice

Python

Output of Python Program | Set 1

Predict the output of following python programs:

Program 1:

```
r = lambda q: q * 2
s = lambda q: q * 3
x = 2
x = r(x)
x = s(x)
x = r(x)
print x
```

Output:

```
24
```

Explanation : In the above program r and s are lambda functions or anonymous functions and q is the argument to both of the functions. In first step we have initialized x to 2. In second step we have passed x as argument to the lambda function r, this will return x*2 which is stored in x. That is, x = 4 now. Similarly in third step we have passed x to lambda function s, So x = 4*3. i.e, x = 12 now. Again in the last step, x is multiplied by 2 by passing it to function r. Therefore, x = 24.

Program 2:

```
a = 4.5
b = 2
print a//b
```

Output:

```
2.0
```

Explanation : This type of division is called truncating division where the remainder is truncated or dropped.

Program 3:

```
a = True
b = False
c = False

if a or b and c:
    print "GEEKSFORGEEKS"
else:
    print "geeksforgeeks"
```

Output:

```
GEEKSFORGEEKS
```

Explanation : In Python, AND operator has higher precedence than OR operator. So, it is evaluated first. i.e, (b and c) evaluates to false.Now OR operator is evaluated. Here, (True or False) evaluates to True. So the if condition becomes True and GEEKSFORGEEKS is printed as output.

Program 4:

```
a = True
b = False
c = False

if not a or b:
    print 1
elif not a or not b and c:
    print 2
elif not a or b or not b and a:
    print 3
else:
    print 4
```

Output:

```
3
```

Explanation: In Python the precedence order is first NOT then AND and in last OR. So the if condition and all the elif conditions evaluates to False. So the else block is executed and 4 is printed as output.

Program 5:

```
count = 1

def doThis():

    global count

    for i in (1, 2, 3):
        count += 1

doThis()

print count
```

Output:

```
 4
```

Explanation: The variable count declared outside the function is global variable and also the count variable being referenced in the function is the same global variable defined outside of the function. So, the changes made to variable in the function is reflected to the original variable. So, the output of the program is 4.

Python Quizzes

This article is contributed by **Harsh Agarwal**. 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

Output
Python
Python-Output

Output of python program | Set 2

Difficulty level : Intermediate

Predict the output of following Python Programs.

Program 1:

```
class Acc:
    def __init__(self, id):
        self.id = id
        id = 555

acc = Acc(111)
print acc.id
```

Output:

```
111
```

Explanation: Instantiation of the class "Acc" automatically calls the method __init__ and passes the object as the self parameter. 111 is assigned to data attribute of the object called id.
The value "555" is not retained in the object as it is not assigned to a data attribute of the class/object. So, the output of the program is "111"

Program 2:

```
for i in  range(2):
    print i

for i in range(4,6):
    print i
```

Output:

```
0
1
4
5
```

Explanation: If only single argument is passed to the range method, Python considers this argument as the end of the range and the default start value of range is 0. So, it will print all the numbers starting from 0 and before the supplied argument.
For the second for loop the starting value is explicitly supplied as 4 and ending is 5.

Program 3:

```
values = [1, 2, 3, 4]
numbers = set(values)

def checknums(num):
    if num in numbers:
        return True
    else:
        return False

for i in  filter(checknums, values):
    print i
```

Output:

```
1
2
3
4
```

Explanation: The function "filter" will return all items from list values which return True when passed to the function "checkit". "checkit" will check if the value is in the set. Since all the numbers in the set come from the values list, all of the original values in the list will return True.

Program 4:

```
counter = {}

def addToCounter(country):
    if country in  counter:
        counter[country] += 1
    else:
        counter[country] = 1

addToCounter('China')
addToCounter('Japan')
addToCounter('china')

print len(counter)
```

Output:

```
3
```

Explanation: The task of "len" function is to return number of keys in a dictionary. Here 3 keys are added to the dictionary "country" using the "addToCounter" function.
Please note carefully – The keys to a dictionary are **case sensitive.**

Try yourself: What happens If same key is passed twice??

This article is contributed by **Pratik Agarwal**. 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

Output of Python Program | Set 3

Difficulty level : Intermediate

Predict the output of following Python Programs.

Program 1:

```
class Geeks:
    def __init__(self, id):
        self.id = id

manager = Geeks(100)

manager.__dict__['life'] = 49

print manager.life + len(manager.__dict__)
```

Output:

```
51
```

Explanation : In the above program we are creating a member variable having name 'life' by adding it directly to the dictionary of the object 'manager' of class 'Geeks'. Total numbers of items in the dictionary is 2, the variables 'life' and 'id'. Therefore the size or the length of the dictionary is 2 and the variable 'life' is assigned a value '49'. So the sum of the variable 'life' and the size of the dictionary is 49 + 2 = 51.

Program 2:

```
a = "GeeksforGeeks "

b = 13

print a + b
```

Output:

```
An error is shown.
```

Explanation : As you can see the variable 'b' is of type integer and the variable 'a' is of type string. Also as Python is a strongly typed language we cannot simply concatenate an integer with a string. We have to first convert the integer variable to the type string to concatenate it with a string variable. So, trying to concatenate an integer variable to a string variable, an exception of type "TypeError" is occurred.

Program 3:

```
dictionary = {}
dictionary[1] = 1
dictionary['1'] = 2
dictionary[1] += 1

sum = 0
for k in dictionary:
    sum += dictionary[k]

print sum
```

Output:

```
4
```

Explanation : In the above dictionary, the key 1 enclosed between single quotes and only 1 represents two different keys as one of them is integer and other is string. So, the output of the program is 4.

Program 4:

```
dictionary = {1:'1', 2:'2', 3:'3'}
del dictionary[1]
dictionary[1] = '10'
del dictionary[2]
```

```
print len(dictionary)
```

Output:

```
2
```

Explanation : The task of the ‘del’ function is to remove key-value pairs from a dictionary. Initially the size of the given dictionary was 3. Then the key value pair for key 1 is first removed and then added back with a new value. Then the key value pair for key 2 is removed. So, finally the size of the dictionary is 2.

This article is contributed by **Pratik Agarwal**. 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

Output
Python
Python-Output

Output of Python Program | Set 4

Difficulty level : Intermediate
Predict the output of following Python Programs.

Program 1:

```
nameList = ['Harsh', 'Pratik', 'Bob', 'Dhruv']

print nameList[1][-1]
```

Output:

```
k
```

Explanation:
The index position -1 represents either the last element in a list or the last character in a String. In the above given list of names “nameList”, the index 1 represents the second element i.e, the second string “Pratik” and the index -1 represents the last character in the string “Pratik”. So, the output is “k”.

Program 2:

```
nameList = ['Harsh', 'Pratik', 'Bob', 'Dhruv']

pos = nameList.index("GeeksforGeeks")

print pos * 5
```

Output:

```
An Exception is thrown, ValueError: 'GeeksforGeeks' is not in list
```

Explanation:
The task of index is to find the position of a supplied value in a given list. In the above program the supplied value is “GeeksforGeeks” and list is nameList. As GeeksforGeeks is not present in the list, an exception is thrown.

Program 3:

```
geekCodes = [1, 2, 3, 4]

geekCodes.append([5,6,7,8])

print len(geekCodes)
```

Output:

```
5
```

Explanation:
The task of append() method is to append a passed *obj* into an existing list. But instead of passing a list to the append method will not merge the two lists, the entire list which is passed is added as an element of the list. So the output is 5.

Program 4:

```
def addToList(listcontainer):
    listcontainer += [10]

mylistContainer = [10, 20, 30, 40]
addToList(mylistContainer)
print len(mylistContainer)
```

Output:

```
5
```

Explanation:
In Python, everything is a reference and references are passed by value. Parameter passing in Python is same as reference passing in Java. As a consequence, the function can modify the value referred by passed argument, i.e. the value of the variable in the caller's scope can be changed. Here the task of the function "addToList" is to add an element 10 in the list, So this will increase the length of list by 1. So the output of program is 5. See this for details.

This article is contributed by **Pratik Agarwal**. 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.

Output of Python program | Set 5

Predict the output of the following programs:

Program 1:

```
def gfgFunction():
    "Geeksforgeeks is cool website for boosting up technical skills"
    return 1

print gfgFunction.__doc__[17:21]
```

Output:

```
cool
```

Explanation:
There is a docstring defined for this method, by putting a string on the first line after the start of the function definition. The docstring can be referenced using the __doc__ attribute of the function.
And hence it prints the indexed string.

Program 2:

```
class A(object):
    val = 1

class B(A):
    pass

class C(A):
    pass

print A.val, B.val, C.val
B.val = 2
print A.val, B.val, C.val
A.val = 3
print A.val, B.val, C.val
```

Output:

```
1 1 1
1 2 1
3 2 3
```

Explanation:
In Python, class variables are internally handled as dictionaries. If a variable name is not found in the dictionary of the current class, the class hierarchy (i.e., its parent classes) are searched until the referenced variable name is found, if the variable is not found error is being thrown.
So, in the above program the first call to print() prints the initialized value i.e, 1.
In the second call since B. val is set to 2, the output is 1 2 1.
The last output 3 2 3 may be surprising. Instead of 3 3 3, here B.val reflects 2 instead of 3 since it is overridden earlier.

Program 3:

```
check1 = ['Learn', 'Quiz', 'Practice', 'Contribute']
check2 = check1
check3 = check1[:]

check2[0] = 'Code'
check3[1] = 'Mcq'

count = 0
for c in (check1, check2, check3):
    if c[0] == 'Code':
        count += 1
    if c[1] == 'Mcq':
        count += 10

print count
```

Output:

```
12
```

Explanation:

When assigning check1 to check2, we create a second reference to the same list. Changes to check2 affects check1. When assigning the slice of all elements in check1 to check3, we are creating a full copy of check1 which can be modified independently (i.e, any change in check3 will not affect check1).
So, while checking check1 ‘Code’ gets matched and count increases to 1, but Mcq doest gets matched since its available only in check3.
Now checking check2 here also ‘Code’ gets matched resulting in count value to 2.
Finally while checking check3 which is separate than both check1 and check2 here only Mcq gets matched and count becomes 12.

Program 4:

```
def gfg(x,l=[]):
    for i in range(x):
        l.append(i*i)
    print(l)

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

Output:

```
[0, 1]
[3, 2, 1, 0, 1, 4]
[0, 1, 0, 1, 4]
```

Explanation:

The first function call should be fairly obvious, the loop appends 0 and then 1 to the empty list, l. l is a name for a variable that points to a list stored in memory. The second call starts off by creating a new list in a new block of memory. l then refers to this new list. It then appends 0, 1 and 4 to this new list. So that’s great. The third function call is the weird one. It uses the original list stored in the original memory block. That is why it starts off with 0 and 1.

This article is contributed by **Harsh Agarwal**. 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

Output
Python
Python-Output