Other Functions

Enum_Zlib,Random,OS Function

Random Function

- import random
- # randomly select an odd number between 1-100
- print ("randrange(1,100, 2) : ", random.randrange(1, 100, 2))
- # randomly select a number between 0-99
- print ("randrange(100) : ", random.randrange(100))

Sample and choice of Random

- Random sample is used to print anyone sample element from random set or list or tuple user can select number of elements
- Ex: random.sample(('abcdefghijklmnopqrstuvwxyz'), 1)
- The same like sample but can print only one thing
- random.choice('abcdefghij')

Shuffle

- import random

- list = [20, 16, 10, 5];
- random.shuffle(list)
- print ("Reshuffled list : ", list)

- random.shuffle(list)
- print ("Reshuffled list : ", list)

Some other Random Example

- >>> random.random() # Random float x, 0.0 <= x < 1.0
- 0.37444887175646646
- >>> random.uniform(1, 10) # Random float x, 1.0 <= x < 10.0
- 1.1800146073117523
- >>> random.randint(1, 10) # Integer from 1 to 10, endpoints included
- 7
- >>> random.randrange(0, 101, 2) # Even integer from 0 to 100
- 26
- >>> random.choice('abcdefghij') # Choose a random element
- 'c'
- >>> items = [1, 2, 3, 4, 5, 6, 7]
- >>> random.shuffle(items)
- >>> items
- [7, 3, 2, 5, 6, 4, 1]
- >>> random.sample([1, 2, 3, 4, 5], 3) # Choose 3 elements
- [4, 1, 5]

Enumerate

- A lot of times when dealing with iterators, we also get a need to keep a count of iterations. Python eases the programmers' task by providing a built-in function enumerate() for this task.
- Enumerate() method adds a counter to an iterable and returns it in a form of enumerate object. This enumerate object can then be used directly in for loops or be converted into a list of tuples using list() method.

Enum syntax

- enumerate(iterable, start=0)
- Parameters:
- Iterable: any object that supports iteration
- Start: the index value from which the counter is
- to be started, by default it is 0

Example Script

```
# Python program to illustrate
# enumerate function
l1 = ["eat","sleep","repeat"]
s1 = "geek"
print (list(enumerate(l1)))
 # changing start index to 2 from 0
print (list(enumerate(s1,2)))
```

Output:

```
Return type: <class 'enumerate'>
[(0, 'eat'), (1, 'sleep'), (2, 'repeat')]
[(2, 'g'), (3, 'e'), (4, 'e'), (5, 'k')]
```

Zlib

- The Python zlib library provides a Python interface to the zlib C library, which is a higher-level abstraction for the DEFLATE lossless compression algorithm. The data format used by the library is specified in the RFC 1950 to 1952, which is available at http://www.ietf.org/rfc/rfc1950.txt.

- The zlib compression format is free to use, and is not covered by any patent, so you can safely use it in commercial products as well. It is a lossless compression format (which means you don't lose any data between compression and decompression), and has the advantage of being portable across different platforms. Another important benefit of this compression mechanism is that it doesn't expand the data.

- The main use of the zlib library is in applications that require compression and decompression of arbitrary data, whether it be a string, structured in-memory content, or files.

Zlib Ex 1

- import zlib
- import binascii

- data = 'Hello world'
- data_en=data.encode()
- compressed_data = zlib.compress(data_en, 2)

- print('Original data: ' , data_en)
- print('Compressed data: ' , binascii.hexlify(compressed_data))

Decompressing Example

- import zlib data = 'Hello world'
- compressed_data = zlib.compress(data, 2) decompressed_data = zlib.decompress(compressed_data) print('Decompressed data: ' + decompressed_data)

Time IT

- 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.

Timeit functions

- 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.

Ex for Timeit

```
# 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))
```

OS Function

- os.system() # Executing a shell command
- os.stat() # Get the status of a file
- os.environ() # Get the users environment
- os.chdir() # Move focus to a different directory
- os.getcwd() # Returns the current working directory
- os.getgid() # Return the real group id of the current process
- os.getuid() # Return the current process’s user id

OS Function

- os.getpid() # Returns the real process ID of the current process
- os.getlogin() # Return the name of the user logged

- os.remove(path) # Remove (delete) the file path
- os.rmdir(path) # Remove (delete) the directory path
- os.makedirs(path)# Recursive directory creation function
- os.removedirs(path) # Remove directories recursively
- os.rename(src, dst) # Rename the file or directory src to dst

OS Function

- os.access() # Check read permissions
- os.chmod() # Change the mode of path to the numeric mode
- os.chown() # Change the owner and group id
- os.umask(mask) # Set the current numeric umask
- os.getsize() # Get the size of a file
- os.environ() # Get the users environment

OS Function

- os.uname() # Return information about the current operating system
- os.chroot(path) # Change the root directory of the current process to path

- os.listdir(path)# List of the entries in the directory given by path

- os.getloadavg() # Show queue averaged over the last 1, 5, and 15 minutes

- os.path.exists()# Check if a path exists

- os.walk() # Print out all directories, sub-directories and files

- os.mkdir(path) # Create a directory named path with numeric mode mode