Argparse - compulsory , optional args , default values , help

Object Oriented Programming in Python

Classes, Objects

Constructors, Destructors and Data Hidding

Inheritance(implicit, override, and altered) and Compositions

Class Variable and Object Variable

Deleting Attributes and Objects

MRO

```
class Vehicle(object):
  __Private = 0
  def __init__(self, fuel, current_capacity):
    self.fuel = fuel
    self.current_capacity = current_capacity

  def fill_fuel(self, quantity):
    self.current_capacity += quantity

  def check_fuel_status(self):
    print('Current {} Availabile is : {}'.format(self.fuel, self.current_capacity))
  def __del__(self):
    print('Deleted object')
    del self

v1 = Vehicle('Petrol', 6)
v2 = Vehicle('Diesel', 6)
v1.fill_fuel(7)
v1.check_fuel_status()
v2.fill_fuel(7)
v2.check_fuel_status()

print(dir(Vehicle))
del v1.current_capacity
del v1
```

```
Deleted object
Deleted object
Current Petrol Availabile is : 13
Current Diesel Availabile is : 13
['_Vehicle__Private', '__class__', '__del__', '__delattr__', '__dict__', '__c
Deleted object
```

```
# Inheritance(implicit, override, and altered) and Compositions


# Implicit and Override
class Base(object):
  def __init__(self):
    print('Base is initialized')
  def display_base(self):
    print('Display from Base')
```

```
class Derived(Base):
  def __init__(self):
    print('Derived is initialized')


  def display_base(self):
    super().display_base()
    print('Display from Derived for base, altered')


  def display(self):
    print('Derived display')

d = Derived()
d.display()
d.display_base()
```

```
Derived is initialized
Derived display
Display from Base
Display from Derived for base, altered
```

```
#Altered
class Base(object):
  def __init__(self, value):
    self.b = value
    print('Base is initialized')
  def display_base(self):
    print('B: {}'.format(self.b))


class Derived(Base):
  def __init__(self, value1, value2):
    super().__init__(value1)
    self.d = value2
    print('Derived is initialized')
  def display(self):
    print('D: {}'.format(self.d))

d = Derived(7, 8)
d.display()
d.display_base()
```

```
Base is initialized
Derived is initialized
D: 8
B: 7
[__main__.Derived, __main__.Base, object]
```

```
# Compositions:

class Tyre(object):
  def __init__(self, brand, ttype):
    self.brand = brand
    self.ttype = ttype
  def display(self):
    print('Brand {}, Type{}'.format(self.brand, self.ttype))

t = Tyre('CEAT', 'Tubeless')
t.display()

class Car(object):
  def __init__(self, brand, tyre):
    self.brand = brand
    self.tyre = tyre
  def display(self):
    print('Car Brand {} and Tyre Details {} {}'.format(self.brand,
```

```
                                                self.tyre.brand,
                                               self.tyre.ttype))

c = Car('Honda', t)
c.display()
```

```
Brand CEAT, TypeTubeless
Car Brand Honda and Tyre Details CEAT Tubeless
```

```
#MRO

class A(object):
  pass

class B(object):
  pass

class C(B, A):
  pass

class D(A, B):
  pass

C.mro()
```

```
[__main__.C, __main__.B, __main__.A, object]
```

Standard Python Modules

os, string, sys, subprocess, datetime, argparse

OS

getcwd, environ, chdir,mkdir,walk,path, path.join st_size, st_mtime

```
import os

for dir in os.listdir(os.getcwd()):
  print(dir, os.stat(dir).st_size, os.stat(dir).st_mtime)
```

```
.config 4096 1536858497.0
sample_data 4096 1536859733.0
demorw.txt 36 1536947376.6886268
demo.txt 42 1536946754.3595612
demow.txt 42 1536947376.6856265
```

```
''' OS '''
import os

print(os.environ)
os.getcwd()
os.getcwd()
os.mkdir(os.path.join(os.getcwd(), 'test'))
print(os.listdir(os.getcwd()))
print(os.stat(os.path.join(os.getcwd(), 'test')).st_size)
os.rmdir(os.path.join(os.getcwd(), 'test'))

file_list = dict()
sub_dir_list = dict()

for root, sub_dirs, files in os.walk(os.getcwd(), topdown=True):
  file_list[root] = list()
```

```
    sub_dir_list[root] = list()
    for file in files:
      file_list[root].append(file)
    for sub_dir in sub_dir_list:
      sub_dir_list[root].append(sub_dir)

'''
from pprint import pprint
pprint(file_list)
pprint(sub_dir_list)
'''
```

```
environ({'LANG': 'en_US.UTF-8', 'DATALAB_DEBUG': 'true', 'SHELL': '/bin/bash'
['.config', 'sample_data', 'test']
4096
'\nfrom pprint import pprint\npprint(file_list)\npprint(sub_dir_list)\n'
```

String

str.upper(), str.lower(), str.isalnum(), str.isalpha(), str.islower(), str.isnumeric(), str.isspace(), str.istitle(), str.isupper()

in, not in, +, *, endswith , startswith split, join

```
test_str = 'Hello World '
test_str.upper()
test_str.lower()

print('Hello' in test_str)

print(test_str.isupper())
print(test_str.endswith('ld'))
print(test_str.startswith('He'))
print(test_str*5)
```

```
True
False
False
True
Hello World Hello World Hello World Hello World Hello World
```

sys

sys.argv, sys.byteorder, sys.builtin_module_names, sys.platform

```
import sys
sys.argv
sys.builtin_module_names
sys.byteorder
sys.platform
sys.path
sys.executable
sys.version_info
```

```
sys.version_info(major=3, minor=6, micro=3, releaselevel='final', serial=0)
```

File handling in python

Open , Modes, Reading, Writing

"r" Open text file for reading. The stream is positioned at the beginning of the file.

"r+" Open for reading and writing. The stream is positioned at the beginning of the file.

"w" Truncate file to zero length or create text file for writing. The stream is positioned at the beginning of the file.

"w+" Open for reading and writing. The file is created if it does not exist, otherwise it is truncated. The stream is positioned at the beginning of the file.

"a" Open for writing. The file is created if it does not exist. The stream is positioned at the end of the file. Subsequent writes to the file will always end up at the then current end of file, irrespective of any intervening fseek(3) or similar.

"a+" Open for reading and writing. The file is created if it does not exist. The stream is positioned at the end of the file. Subse- quent writes to the file will always end up at the then current end of file, irrespective of any intervening fseek(3) or similar.

read, readline and readlines

File attributes - closed, mode, name

File Positions - seek, tell

Rename , Delete os.remove, os.rename

Context Management

```
# Write

f =  open('demow.txt', 'w')
f.write('Hello world \n')
f.write('Python is beautiful languages')
f.close()

# Read, name , closed

f =  open('demo.txt', 'r')
print(f.readline())
print(f.readline())
print(f.closed)
print(f.name)
f.close()
print(f.closed)

# Append
f =  open('demorw.txt', 'a')
f.write('Hello \n')
f.write('World \n')

# R+ , W+ A+
f =  open('demorw.txt', 'r+')
f.write('Written here by r+')
f.close()
# W+
f =  open('demorw.txt', 'w+')
f.write('Written here by w+')
f.close()

# a+
f =  open('demorw.txt', 'a+')
f.write('Written here by a+')
```

```
f.close()

f =  open('demorw.txt', 'a+')
f.seek(0)

for line in f.readlines():
   print(line.strip())
f.close()
```

```
Hello world

Python is beautiful languages
False
demo.txt
True
Written here by w+Written here by a+
```

Some Advanced Python Concepts, Tips & Tricks

Exception handling - try,catch,finally, Custom Exception

Function Decorators - decorators, decorators with arguments

lambda and generators

```
# Exception handling

class CustomError(BaseException):
  def __str__(self):
    return "this is custom exception raised"


def divide(a, b):
    try:
        raise CustomError()
    finally:
        print('I will always clean up the things')


try:
    divide(2, 3)
except CustomError as e:
    print(e)
```

```
I will always clean up the things
this is custom exception raised
```

```
# Decorator
def validate_io(func):
  def wrapper(num):
    if num < 0:
      print('Input can not be negative, invalid input {}'.format(num))
      return
    else:
      out = func(num)
      if out > 1000:
        print('Warning output exceeeded the 1000')
      return out
  return wrapper
```

```
@validate_io
def squareit(num):
  return num*num

@validate_io
def cubeit(num):
  return num*num*num


print(squareit(100))
print(cubeit(11))
```

```
Warning output exceeeded the 1000
10000
Warning output exceeeded the 1000
1331
```

```
# Parameterized Decorator
def validate_io_with_bound(low, max):
  def validate_io(func):
    def wrapper(num):
      if num < low:
        print('Input can not be lower than {}, invalid input {}'.format(low, num))
        return
      else:
        out = func(num)
        if out > max:
          print('Warning output exceeeded the {}'.format(max))
        return out
    return wrapper
  return validate_io

@validate_io_with_bound(10, 500)
def squareit(num):
  return num*num

@validate_io_with_bound(10, 1000)
def cubeit(num):
  return num*num*num


print(squareit(1))
print(cubeit(1))
```

```
Input can not be lower than 10, invalid input 1
None
Input can not be lower than 10, invalid input 1
None
```

```
# Generators

def square_it(data):
  return [num*num for num in data]

# print(square_it(range(1,15)))

def lazy_square_it(data):
  for num in data:
    yield num*num

out = lazy_square_it(range(1,15))
# print(list(out))


from cProfile import run

run('square_it(range(1,1000000))')
run('lazy_square_it(range(1,1000000))')
```

```
         5 function calls in 0.122 seconds

   Ordered by: standard name

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
        1    0.000    0.000    0.110    0.110 <ipython-input-44-7f464d10dc8f>
        1    0.110    0.110    0.110    0.110 <ipython-input-44-7f464d10dc8f>
        1    0.012    0.012    0.122    0.122 <string>:1(<module>)
        1    0.000    0.000    0.122    0.122 {built-in method builtins.exec}
        1    0.000    0.000    0.000    0.000 {method 'disable' of '_lsprof.P


         4 function calls in 0.000 seconds

   Ordered by: standard name

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
        1    0.000    0.000    0.000    0.000 <ipython-input-44-7f464d10dc8f>
        1    0.000    0.000    0.000    0.000 <string>:1(<module>)
        1    0.000    0.000    0.000    0.000 {built-in method builtins.exec}
        1    0.000    0.000    0.000    0.000 {method 'disable' of '_lsprof.P
```

```
fun = lambda x:x**2

print(fun(8))

def get_multiplier(n):
  return lambda x: x**n

doubler = get_multiplier(2)
tripler = get_multiplier(3)
quadruple = get_multiplier(4)

print(doubler(2), tripler(2), quadruple(2))
```

```
64
4 8 16
```

```
# lambda with map
data = range(1, 2)
cubedata = list(map(lambda x : x**3 , data))
print(cubedata)
```

```
[1, 8, 27, 64, 125, 216, 343, 512, 729, 1000, 1331]
```

```
# lambda with filter
data = list(range(1,30))
odd_nums = filter(lambda x: x%3==0, data)
print(list(odd_nums))
```

```
[3, 6, 9, 12, 15, 18, 21, 24, 27]
```

```
# intro to opencv and Image Processing

import cv2
import numpy as np

# 1 - Color 0 - BW
```

```
image = cv2.imread('images/first.JPG')

black = np.zeros([250, 250, 1], 'uint8')

white = np.ones([250, 250, 1])

white = white[:, :] * (2**8-1)

ones = np.ones([250, 250, 3], 'uint8')
blue = np.ones([250, 250, 3], 'uint8')
green = np.ones([250, 250, 3], 'uint8')
red = np.ones([250, 250, 3], 'uint8')

blue[:, :, 0] = ones[:, :, 0] * (2**8-1)

green[:, :, 1] = ones[:, :, 1] * (2**8-1)

red[:, :, 2] = ones[:, :, 2] * (2**8-1)


cv2.imwrite('images/white.jpg', white)

print(black.shape)

# cv2.imshow('Image', image)
# cv2.imshow('White', white)
# cv2.imshow('Black Image', black)
# cv2.imshow('Blue Channel', image[:, :, 0])
# cv2.imshow('Green Channel', image[:, :, 1])
# cv2.imshow('Red Channel', image[:, :, 2])

cv2.imshow('Blue', blue)
cv2.imshow('Green', green)
cv2.imshow('Red', red)


cv2.waitKey()
cv2.destroyAllWindows()
```