PYTHON PROGRAMMING Solution 2

MCQ:

a) If zoo = [‘lion’, tiger’], what will be zoo*2?

i) [‘lion’, ‘tiger’, ‘lion’, ‘tiger’]

b) For dictionary d = {plum : 0.66, pears : 1.25, oranges : 0.49} which ofthe following statement correctly updates the price of oranges to 0.52?i)

i)d [oranges] = 0.52

c) What is the correct syntax for creating single valued tuple?

iii) (a,)

d) The length of sys. argv is

iv)Total number of excluding the file name

e) Which of these definitions correctly describes a module?

Design and implementation of specific functionality to be incorporated into a program.

f) To include use of functions which are present in random library, we mustuse the option

i)random. h

g) _____is a string literal denoted by triple quote for providing thespecifications of certain program elements.

ii)docstring

h) The correct way of inheriting a derived class from base class is:

i) Class Derived (Base):

i) The + operator is overloaded using the method

ii) --add-- ( )

j) The syntax for using super( ) in derived class__init__( )method definitionlooks like

iv) Super ( ).--init-- (base class parameters)

k) The character___and___ matches the start and end of the string, respectively.

iv)^ and $

l) Consider a five 21-12-2016. zip. The regular expression pattern to extractdate from filename is

i) ([0-9] {2} \- [0-9] {2} \- [0-9] {4})

m) When will the else part of the try-except-else be executed?
iii)When there is no exception

n) When is the finally block executed?

ii)always

o) Which of the following is not a valid attribute of the file object file-handler.

ii)file-handler.size

p) Which of the following command is used in mango shell to show all thedatabases.

ii)Show dbs

q) The function that returns its arguments with a modified shape and themethod that modifies the arry itself respectively in Numpy are

iv) reshape, resize

r) Which of the following is used for line graph?

ii)plt. plot ( )

s) Which of the following statement is false?

i) ndarray is also known as the axis array

t) The method used in pandas to return top 5 rows

iv)head ( )

Q1) a) Write a python program to check whether the given string is palindrom or not using function. [5]

Answer:

```
def is_palindrome(str):
    for i in range(len(str)):
        if str[i] != str[len(str)-i-1]:
            return False
    return True

str = input("Enter Text = ")
if is_palindrome(str):
    print("palindrome")
else:
    print("Not")
```

b) What is lambda function in python? Explain with any (one) example.[5]

Answer: A **Lambda Function in Python** programming is an anonymous function or a function having no name. It is a small and restricted function having no more than one line. Just like a normal function, a Lambda function can have multiple arguments with one expression.

In Python, lambda expressions (or lambda forms) are utilized to construct anonymous functions. To do so, you will use the **lambda** keyword. Every anonymous function you define in Python will have 3 essential parts:

- The lambda keyword.
- The parameters (or bound variables), and
- The function body.

A lambda function can have any number of parameters, but the function body can only contain **one** expression. Moreover, a lambda is written in a single line of code and can also be invoked immediately.

Syntax: `lambda p1, p2: expression`

Example: adder = lambda x, y: x + y

print (adder (1, 2))

1. The lambda keyword used to define an anonymous function.
2. x and y are the parameters that we pass to the lambda function.
3. This is the body of the function, which adds the 2 parameters we passed. Notice that it is a single expression. You cannot write multiple statements in the body of a lambda function.
4. We call the function and print the returned value.

Example2:

```
x="some kind of a useless lambda"
(lambda x : print(x))(x)
```

Example3:

Max = lambda a, b : a if(a > b) else b

print(Max(1, 2))

In the following example we have evaluated an expression using lamda expression

Q2) a) What is string? Explain any five string functions with proper example.[5]

Answer: A string is a data structure in Python that represents a sequence of characters. It is an immutable data type, meaning that once you have created a string, you cannot change it. Strings are used widely in many different applications, such as storing and manipulating text data, representing names, addresses, and other types of data that can be represented as text.

Creating a string: str1 = 'Hello Python'

print(str1)

Slicing string: Specify the start index and the end index, separated by a colon, to return a part of the string.

b = "Hello, World!"

print(b[2:5])

Modifying String:

Example1: Convert string into upper case

upper() method returns the string in upper case:

a = "Hello, World!"

print(a.upper())

Example2: Convert string into lower case

The lower() method returns the string in lower case:

a = "Hello, World!"

print(a.lower())

Example: Remove whitespace

The strip() method removes any whitespace from the beginning or the end:
a = " Hello, World! "
print(a.strip())

Concatenate String:

To concatenate, or combine, two strings you can use the + operator.

```
a = "Hello"
b = "World"
c = a + b
print(c)
```

Format string:

The format() method takes the passed arguments, formats them, and places them in the string where the placeholders {} are

```
age = 36
txt = "hello all myself Ramesh, and I am {}"
print(txt.format(age))
```

b) Print odd numbers between 1 to 100 using for loop. [5]
Answer: start = int(input("Enter the start of range:"))
end = int(input("Enter the end of range:"))

```
# iterating each number in list
for num in range(start, end + 1):

    # checking condition
    if num % 2 != 0:
        print(num)
```

o/p: Enter the start range: 1
Enter the end of range :100
1 3 5 7 9 11--------99

Q3) Write a python program to implement student class which has method to calculate percentage. Assume suitable class variable. [10]
Answer:

```
class student:
 def calculate(self):
  print("Enter the marks of five subjects::")

  subject_1 = float (input ())
  subject_2 = float (input ())
  subject_3 = float (input ())
  subject_4 = float (input ())
  subject_5 = float (input ())

  total, average, percentage, grade = None, None, None, None

# It will calculate the Total, Average and Percentage
  total = subject_1 + subject_2 + subject_3 + subject_4 + subject_5
  average = total / 5.0
  percentage = (total / 500.0) * 100
```

```
  if average >= 90:
    grade = 'A'
  elif average >= 80 and average < 90:
    grade = 'B'
  elif average >= 70 and average < 80:
    grade = 'C'
  elif average >= 60 and average < 70:
    grade = 'D'
  else:
    grade = 'E'

   # It will produce the final output
  print ("\nThe Total marks is:   \t", total, "/ 500.00")
  print ("\nThe Average marks is: \t", average)
  print ("\nThe Percentage is:    \t", percentage, "%")
  print ("\nThe Grade is:         \t", grade)
obj=student()
obj.calculate()
```

Q4) a) Write a python program to extract email from text file. [5]
Answer:

```
import re
text = "Please contact us at contact@tutorialspoint.com for further information."+\
       " You can also give feedbacl at feedback@tp.com"


emails = re.findall(r"[a-z0-9\.\-+_]+@[a-z0-9\.\-+_]+\.[a-z]+", text)
print(emails)
```

b) Explain multithreading in python with example. [5]

Answer: Multithreading is a stringing procedure in Python programming to run various strings simultaneously by quickly exchanging between strings with a central processor help (called setting exchanging).

In addition, it makes it possible to share its data space with the primary threads of a process, making it easier than with individual processes to share information and communicate with other threads.

The goal of multithreading is to complete multiple tasks at the same time, which improves application rendering and performance.

There are two main modules of multithreading used to handle threads in Python.

1. The thread module
2. The threading module

Threading Modules

The threading module is a high-level implementation of multithreading used to deploy an application in Python. To use multithreading, we need to import the threading module in Python Program.

start()	A start() method is used to initiate the activity of a thread. And it calls only once for each thread so that the execution of the thread can begin.
run()	A run() method is used to define a thread's activity and can be overridden by a class that extends the threads class.
join()	A join() method is used to block the execution of another code until the thread terminates.

Program:

```
import threading

def print_hello(n):

    Print("Hello, how old are you? ", n)

T1 = threading.Thread( target = print_hello, args = (20, ))

T1.start()

T1.join()

Print("Thank you")
```

Thread modules

It is started with Python 3, designated as obsolete, and can only be accessed with _thread that supports backward compatibility.

Syntax:

thread.start_new_thread (function_name, args[, kwargs])

To implement the thread module in Python, we need to import a thread module and then define a function that performs some action by setting the target with a variable.

Thread.py

```
import thread # import the thread module
import time # import time module

def cal_sqre(num): # define the cal_sqre function
    print(" Calculate the square root of the given number")
    for n in num:
        time.sleep(0.3) # at each iteration it waits for 0.3 time
        print(' Square is : ', n * n)

def cal_cube(num): # define the cal_cube() function
    print(" Calculate the cube of  the given number")
    for n in num:
        time.sleep(0.3) # at each iteration it waits for 0.3 time
        print(" Cube is : ", n * n *n)

arr = [4, 5, 6, 7, 2] # given array

t1 = time.time() # get total time to execute the functions
cal_sqre(arr) # call cal_sqre() function
cal_cube(arr) # call cal_cube() function
```

print(" Total time taken by threads is :", time.time() - t1) # print the total time

Q5) Draw pie chart and bar chart using library of python with suitable example.[10]

Answer:

```
import matplotlib.pyplot as plt
import numpy as np

y = np.array([35, 25, 25, 15])
mylabels = ["Python", "Java", "MongoDB", "Operational Research"]

plt.pie(y, labels = mylabels)
plt.show()
```

```
import numpy as np

import matplotlib.pyplot as plot

# creating the dataset

data = {'C':20, 'C++':15, 'Java':30,

    'Python':35}

courses = list(data.keys())

values = list(data.values())



fig = plt.figure(figsize = (10, 5))

# creating the bar plot

plt.bar(courses, values, color ='maroon',

    width = 0.4)

plt.xlabel("Courses offered")

plt.ylabel("No. of students enrolled")

plt.title("Students enrolled in different courses")

plt.show()
```

Q6) a) How mangy ways Data frame can be created using pandas. [5]

Answer: Creating an empty dataframe:

```
import pandas as pd
```

```
# Creating Empty DataFrame and Storing it in variable df
df = pd.DataFrame()

# Printing Empty DataFrame
print(df)
```

Creating Dataframe from Lists:

```
import pandas as pd

# initialize list elements
data = [10,20,30,40,50,60]

# Create the pandas DataFrame with column name is provided explicitly
df = pd.DataFrame(data, columns=['Numbers'])

# print dataframe.
df
```

Creating Pandas DataFrame from lists of lists:

```
import pandas as pd

# initialize list of lists
data = [['tom', 10], ['nick', 15], ['juli', 14]]

# Create the pandas DataFrame
df = pd.DataFrame(data, columns=['Name', 'Age'])

# print dataframe.
df
```

Creating DataFrame from dict of narray/lists:

```
import pandas as pd

# initialize data of lists.
data = {'Name': ['Tom', 'nick', 'krish', 'jack'],
        'Age': [20, 21, 19, 18]}

# Create DataFrame
df = pd.DataFrame(data)

# Print the output.
df
```

Creating a DataFrame by proving index label explicitly.

```
import pandas as pd

# initialize data of lists.
data = {'Name': ['Tom', 'Jack', 'nick', 'juli'],
        'marks': [99, 98, 95, 90]}

# Creates pandas DataFrame.
df = pd.DataFrame(data, index=['rank1',
                               'rank2',
```

```
                                    'rank3',
                                    'rank4'])

# print the data
df
```

Creating Dataframe from list of dicts:

```
import pandas as pd

# Initialize data to lists.
data = [{'a': 1, 'b': 2, 'c': 3},
        {'a': 10, 'b': 20, 'c': 30}]

# Creates DataFrame.
df = pd.DataFrame(data)

# Print the data
Df
```

b) Explain Decorators and Generators in python. [5]

Answer: A Python decorator is a function that takes in a function and returns it by adding some functionality.In fact, any object which implements the special __call__() method is termed callable. So, in the most basic sense, a decorator is a callable that returns a callable.Basically, a decorator takes in a function, adds some functionality and returns it.

```
def make_pretty(func):
    def inner():
        print("I got decorated")
        func()
    return inner
def ordinary():
    print("I am ordinary")
```

ordinary() that prints "I am ordinary"

make_pretty() that takes a function as its argument and has a nested function named inner(), and returns the inner function.

We are calling the ordinary() function normally, so we get the output "I am ordinary". Now, let's call it using the decorator function.

```
def make_pretty(func):
    # define the inner function
    def inner():
        # add some additional behavior to decorated function
        print("I got decorated")

        # call original function
        func()
    # return the inner function
    return inner

# define ordinary function
def ordinary():
    print("I am ordinary")

# decorate the ordinary function
decorated_func = make_pretty(ordinary)

# call the decorated function
decorated_func()
```

@ Symbol With Decorator

Instead of assigning the function call to a variable, Python provides a much more elegant way to achieve this functionality using the @ symbol. For example,

```
def make_pretty(func):

    def inner():
        print("I got decorated")
        func()
    return inner

@make_pretty
def ordinary():
    print("I am ordinary")

ordinary()
```

Decorating Functions with Parameters:

```
def smart_divide(func):
    def inner(a, b):
        print("I am going to divide", a, "and", b)
        if b == 0:
            print("Whoops! cannot divide")
            return
```

```
        return func(a, b)
    return inner

@smart_divide
def divide(a, b):
    print(a/b)

divide(2,5)

divide(2,0)
```

Generator Function in Python

A generator function in Python 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 Python generator function.

Create a Generator in Python

In Python, we can create a generator function by simply using the def keyword and the yield keyword. The generator has the following syntax in Python:

```
def function_name():
    yield statement
def simpleGeneratorFun():
    yield 1
    yield 2
    yield 3

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

Generator Object

Python Generator functions return a generator object that is iterable, i.e., can be used as an Iterator. Generator objects are used either by calling the next method of the generator object or using the generator object in a “for in” loop.

```
def simpleGeneratorFun():
    yield 1
    yield 2
    yield 3

# x is a generator object
x = simpleGeneratorFun()

# Iterating over the generator object using next

# In Python 3, __next__()
print(next(x))
print(next(x))
print(next(x))
```

Q7) Write a short note (any four) : [4 × 5 = 20]

a)read ( ), readline ( ), readlines ( ):

Answer:

Read ( ): Returns the read bytes in the form of a string. Reads n bytes; if n is not specified, then reads the entire file.

```
F=open("demofile.txt","r")
print(F.read())
```

Readline ( ): Reads a line of the file and returns in the form of a string. For specified n, reads at most n bytes. readline ( ) function does not read more than one line at a time; even if n exceeds, it reads only one line. Readline ( ) function reads a line of the file and returns it in the string.

```
file = open("filename.txt", "r")
file.readline()
```

readlines ( ): Reads all the lines and returns them as a string element in a list. Readlines ( ) is used to read all the lines at a single go and then return them as a string element in a list.

```
file = open("filename.txt","r")
file.readlines()
```

b)Matplotlib:

Matplotlib is a low level graph plotting library in python that serves as a visualization utility.

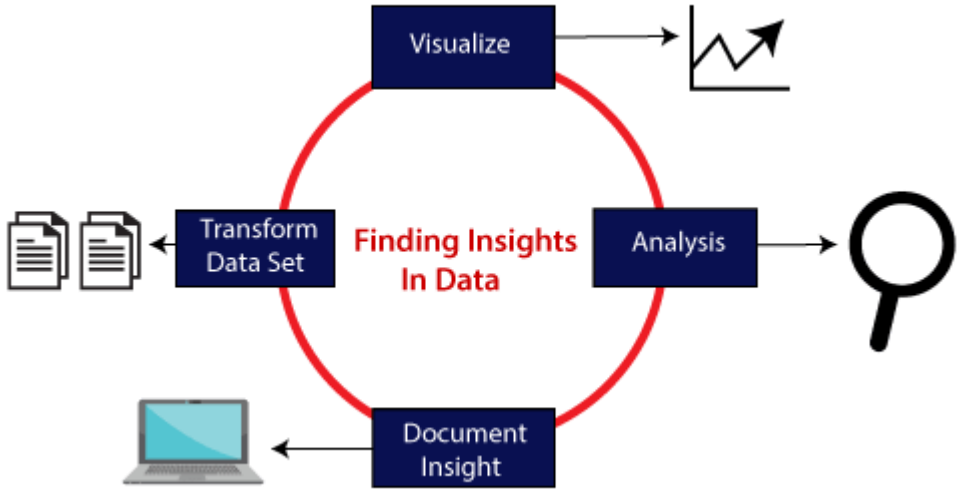
Matplotlib was created by John D. Hunter.

Matplotlib is open source and we can use it freely.

Matplotlib is mostly written in python, a few segments are written in C, Objective-C and Javascript for Platform compatibility.

Syntax for installation: pip install matplotlib

Phases of matplotlib:



- **Visualize:** We analyze the raw data, which means it makes complex data more accessible, understandable, and more usable. Tabular data representation is used where the user will look up a specific measurement, while the chart of several types is used to show patterns or relationships in the data for one or more variables.
- **Analysis:** Data analysis is defined as cleaning, inspecting, transforming, and modeling data to derive useful information. Whenever we make a decision for the business or in daily life, is by past experience. **What will happen to choose a particular decision**, it is nothing but analyzing our past. That may be affected in the future, so the proper analysis is necessary for better decisions for any business or organization.
- **Document Insight:** Document insight is the process where the useful data or information is organized in the document in the standard format.
- **Transform Data Set:** Standard data is used to make the decision more effectively.

The **matplotlib.pyplot** is the collection command style functions that make matplotlib feel like working with MATLAB. The pyplot functions are used to make some changes to figure such as create a figure, creates a plotting area in a figure, plots some lines in a plotting area, decorates the plot including labels, etc.

Show() is use to show the figure.

'b'	Using for the blue marker with default shape.
'ro'	Red circle

'-g'	Green solid line
'--'	A dashed line with the default color
'^k:'	Black triangle up markers connected by a dotted line

Example:

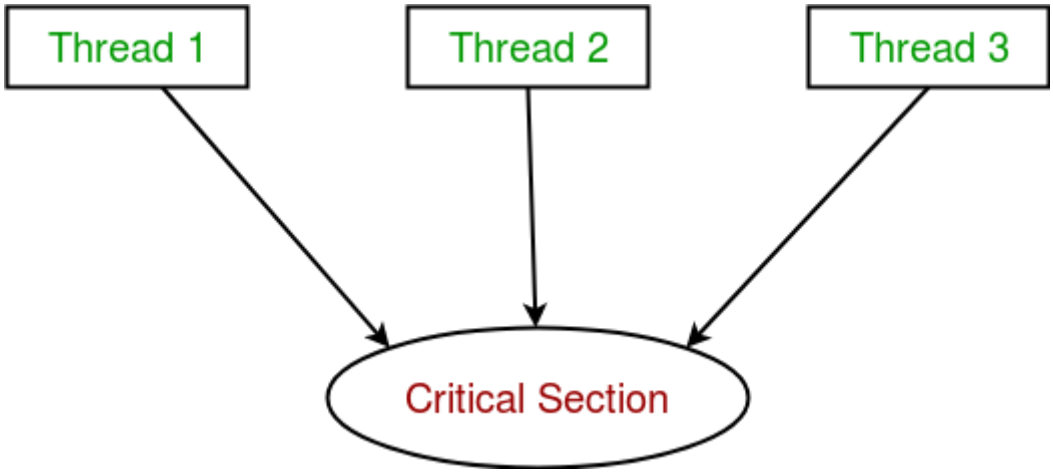
```
from matplotlib import pyplot as plt
#ploting our canvas
plt.plot([1,2,3],[4,5,1])
#display the graph
plt.show()
```

```
from matplotlib import pyplot as plt
plt.plot([1, 2, 3, 4,5], [1, 4, 9, 16,25], 'ro')
plt.axis([0, 6, 0, 20])
plt.show()
```

d) Synchronizing the threads:

What is synchronization in threading? Explain with example. [4m]

Thread synchronization is defined as a mechanism which ensures that two or more concurrent threads do not simultaneously execute some particular program segment known as **critical section**.



Concurrent accesses to shared resource can lead to **race condition**.

A race condition occurs when two or more threads can access shared data and they try to change it at the same time.

```
import threading

# global variable x
x = 0
```

```
def increment():
    """
    function to increment global variable x
    """
    global x
    x += 1

def thread_task():
    """
    task for thread
    calls increment function 100000 times.
    """
    for _ in range(100000):
        increment()

def main_task():
    global x
    # setting global variable x as 0
    x = 0

    # creating threads
    t1 = threading.Thread(target=thread_task)
    t2 = threading.Thread(target=thread_task)

    # start threads
    t1.start()
    t2.start()

    # wait until threads finish their job
    t1.join()
    t2.join()

if __name__ == "__main__":
    for i in range(10):
        main_task()
        print("Iteration {0}: x = {1}".format(i,x))
Iteration 0: x = 175005
Iteration 1: x = 200000
Iteration 2: x = 200000
Iteration 3: x = 169432
Iteration 4: x = 153316
Iteration 5: x = 200000
Iteration 6: x = 167322
Iteration 7: x = 200000
Iteration 8: x = 169917
Iteration 9: x = 153589
```

In above program:

- Two threads **t1** and **t2** are created in **main_task** function and global variable **x** is set to 0.
- Each thread has a target function **thread_task** in which **increment** function is called 100000 times.
- **increment** function will increment the global variable **x** by 1 in each call.

The expected final value of **x** is 200000 but what we get in 10 iterations of **main_task** function is some different values.

This happens due to concurrent access of threads to the shared variable **x**. This unpredictability in value of **x** is nothing but **race condition**.

threading module provides a **Lock** class to deal with the race conditions.

Lock class provides following methods:

- **acquire([blocking]) :** To acquire a lock. A lock can be blocking or non-blocking.
    - When invoked with the blocking argument set to **True** (the default), thread execution is blocked until the lock is unlocked, then lock is set to locked and return **True**.
    - When invoked with the blocking argument set to **False**, thread execution is not blocked. If lock is unlocked, then set it to locked and return **True** else return **False** immediately.
- **release() :** To release a lock.
    - When the lock is locked, reset it to unlocked, and return. If any other threads are blocked waiting for the lock to become unlocked, allow exactly one of them to proceed.
    - If lock is already unlocked, a **ThreadError** is raised.

```
import threading

# global variable x
x = 0

def increment():
    """
    function to increment global variable x
    """
    global x
    x += 1

def thread_task(lock):
    """
    task for thread
    calls increment function 100000 times.
    """
    for _ in range(100000):
        lock.acquire()
        increment()
        lock.release()

def main_task():
    global x
```

```
    # setting global variable x as 0
    x = 0

    # creating a lock
    lock = threading.Lock()

    # creating threads
    t1 = threading.Thread(target=thread_task, args=(lock,))
    t2 = threading.Thread(target=thread_task, args=(lock,))

    # start threads
    t1.start()
    t2.start()

    # wait until threads finish their job
    t1.join()
    t2.join()

if __name__ == "__main__":
    for i in range(10):
        main_task()
        print("Iteration {0}: x = {1}".format(i,x))
```