**Pradeep Madubashena**

**Dictionaries** assesses understanding of the ability to define and use dictionary (mapping) types in a Python program, including setting values, using default values, and iterations.

Dictionaries consists with key and value pair of a data sets and we can store information inside a dictionary. The keys should be unique for each set. If we want to access a piece of information we can refer key. To construct a dictionary use {} and [] for index. For example “Jan:January, Feb: February “etc. “Jan” is the key and “January” is the value.

monthsFirstQ = { “Jan” : “January”, “Feb” : “February”, “Mar” : “March”,

} # First three months of a year in a dictionary

print (monthsFirstQ.items() )

print ( monthsFirstQ.get(“Jan”)) #gets the value of the key

print (monthsFirstQ.keys())# show keys

print (monthsFirstQ.values()) # shoe the values

print (monthsFirstQ.get(“Dec”, “Not a month in first quarter of the year”)) #default values

for key in monthsFirstQ:

print (key, monthsFirstQ [key])

**File I/O** evaluates knowledge of external file interaction using Python, including reading, writing, opening, and closing files and sockets.

Sometimes we have to open, read, write and close in external files like text, csv, HTML.

open(“company\_profile.txt”, “r”) # open and read the company\_profile.txt by giving the file path

open(“company\_profile.txt”, “w”) # open and write the company\_profile.txt by giving the file path

open(“company\_profile.txt”, “a”) # apend information without change

open(“company\_profile.txt”, “r+”) # open, read and write the company\_profile.txt

company\_profile = open(“company\_profile.txt”, “a”)#open/create file as a variable

print(company\_profile.readable()) #check the file can read or not

print(company\_profile.readline()) #read a line

print(company\_profile.readlines()[1]) # take all lines into an array and read first line

company\_profile.write(“\n Pradeep – Python development”) #write a new line in the text file

company\_profile.close() # close the file

**Python Functions** tests Python function definition syntax, parameter passing, and returning results. Lambda expressions are also covered.

Function is a collection of codes for do a specific task. If you give an input you will get an output from a function.Lamda expressions can replace functions.

def totalOftwo(a,b): #defining the fuction

total = a+b #codes inside function

return total #return results

x = totalOftwo(4,6) #giving input by calling fuction

print(“Total value of two numbers is : “, x) # take outputs

evens = filter(lambda num:num%2==0, numbers) #lamda expression

print(list(evens))

**Lists and Tuples** demonstrates knowledge of the Python list and tuple sequence aggregate types, including accessing values, simple functions, and common usage including sorting and searching.

Lists are for store large amount of data to manage, organize properly and store in the list. We can store any types of data

mylist = [“January”, “February”, “Pradeep”, True, 8, 100 ] #create a list and store data

print(mylist[2]) # print index 2 and it is Pradeep

print(mylist[-1]) # print index 4 and it is 8 from back side

print(mylist[1:]) # print from index 1

print(mylist[1:3]) # print index t to 3

numbers = [2, 5, 6, 6, 0, 10]

numbers.extend(mylist) #adding two lists

numbers.append(100) # add another item to list

mylist.insert(2, “March”) # insert an item next to index position 2

mylist.remove(“Pradeep”) # remove the item

mylist.clear() # remove all items

mylist.pop() #remove last item

evens = filter(lambda num:num%2==0, numbers) #lamda expression

print(list(evens))

print(mylist.count(“January”)) # print an item’s index

numbers.sort() # sort the numbers

mylist2 = mylist.copy() # copy the list

Tuple is a type of data structure. It can store multiple information. It is immutable. It is same as list but can not change or modify.

tuple1 = (3, 50, “January”) #create a tuple

print(tuple[1]) # print index 1 of tuple.

**Modules and Imports** tests understanding of how to use external modules in Python, how modules are defined, and how to use modules to enforce data hiding practices.

Module is a python script which is a collection of variables and methods to access to a script. Modules can import other modules in the beginning of the new script. We have to write import statements.

def fib(n): # write Fibonacci series up to n

a, b = 0, 1

while a < n:

print(a, end=' ')

a, b = b, a+b

print()

import fibo

Fibo.fib(200)

In machine learning we can import libraries such as pandas, numpy

import pandas as pd

import numpy as np

from statistics import variance as v, mean as m

list1 = [3,4,5,6,7,8,9,]

x = v(list1) #get the variance from list1

y = m(list1) # get mean from list1

print(x)

**Regular Expression Usage** tests the definition and application of Python regular expressions to locate, extract, and change text in context.

Regular expression is extract regular part of a string

Import re

myInfo = “This is my details. My email address is [madubashena89@gmail.com](mailto:madubashena89@gmail.com). My number 3169258286”

mobileNo = re.findall(‘[0-9]+’, myInfo) #find numbers 0 to 9 in the string

email = re.findall(‘\S+@\S+’, myInfo) # find email address

print(mobileNo)

print(email)

**Classes** tests knowledge of object-oriented programming using Python and object-oriented features like inheritance and polymorphism as well as subclassing, static methods, and private attributes.

Classes are collection of functions and attributes in a template which can make objects with that template. There can be many classes in object oriented programming and also some classes which have the same attributes and methods as parent class. These kind of classes can be inherit by the parent class. We call this as inheritance. Polymorphism is some classes looks same but behavior is different. Which has same methods but behavior is different so we have to adjust the behavior.

class Employee(object):

"""Models real-life employees!"""

def \_\_init\_\_(self, employee\_name):

self.employee\_name = employee\_name

def calculate\_wage(self, hours):

self.hours = hours

return hours \* 20.00

class PartTimeEmployee(Employee): # inheritance by parent class Employee

def calculate\_wage(self, hours):

self.hours = hours

return hours \* 12.00 #Polymorphism of calculate\_wage method

def full\_time\_wage(self, hours):

return super(PartTimeEmployee, self).calculate\_wage(hours)

Dan = PartTimeEmployee('Dan')

print (Dan.full\_time\_wage(100))

**Control Flow** determines knowledge of the Python flow control statements and generators (i.e., while, if, for, range, pass, yield, etc.) and what instructions are executed or evaluated depending on the value of the contextual data or the resulting expressions.

Programs can use for repetition of work for do multiple times. We can use loops such as while and for and if/else statements.

secret\_word: str = "Python"

guess = ""

guess\_limit = 3

guess\_count = 0

out\_of\_guesses = False

while guess != secret\_word and not (out\_of\_guesses): #while loop

if guess\_count < guess\_limit: # if statement

guess = input("Enter guess: ")

guess\_count < +1

else:

out\_of\_guesses = True

if out\_of\_guesses:

print("out of guesses ,you lose")

else:

print("You win")

cubes\_by\_four = [x \*\* 3 for x in range(1, 11) if ((x \*\* 3) % 4) == 0] # for loop example

print ( cubes\_by\_four)

**Python Exception Handling** evaluates skills needed to guarantee that errors reported during the execution of a Python program are properly processed to ensure proper application cleanup while maintaining useful error reporting.

Exception can use without braking or stop executing because of an error. We can use exception to find what type of error for error reporting.

try:

number = int(input(“Enter a number: “)) # we can write our program here

except ZeroDivisionError as err: # A number divide by zero will be an error.

print(err)

except ValueError:

print(“invalid input”)

**List Comprehensions** tests knowledge of how to derive a new list structure from an existing list in Python, including similar implementations using built-in functions like "map" and "filter."

If we are working with large collection of data with lists we can use maps. Area = []

For r in radii:

A = area(r)

Areas.append(a)

Map(area,radii)

Temps = [ (“LA”, 29),(“NY”, 40),(“SF”, 40)]

C\_to\_f = lambda data: (data[0], (9/5)\*data[1] +32)

List(map(C\_to\_f, temps) # map usage for large number of data to convert

Filter is used for select certain piece of data from a list.

Import statistics

Data = [2,4,5,8,2,3,6,7]

Avg = statistics.mean(Data)

Avg

List(filter(lamda x:x> avg, Data) ) # filter function use for find values above average

**Strings** evaluates knowledge of how to manipulate Python strings, including slicing operations and unicode encoding.

Sample\_url = “<http://rhsofnro.com>”

Print sample\_url[7:-4] #slicing of the string for only name of website

Encode convert string to binary format. Decord is converting to string format by binary format. UTF-8 is most popular.

Import base64

str1 = “hi may I know your name?”

str1 = base64.b64encode(str.encode(‘utf-8’)) #encode

print(“Encoded string: ”, str1)

str1 = base64.b64decode(str1).decode(‘utf-8’) #decode

print(“Decoded string: ”, str1)

**Threads** assesses knowledge of the Python threading mechanisms used to improve application responsiveness and  how to use subprocesses instead of threads to implement process-based "threading" interfaces.

Threads are used for doing number of things do at the same time. Memory is shared between all threads. Starting a tread is faster than starting a process.

from treading import Thread

Import os

Import math

Def calc():

For I in range(0,100000):

Math.sqrt(i)

Threads = []

For I in range(os.cpu\_count()):

Print(‘registering thread %d’ % i)

threads.append(Thread(target=calc))

For thread in threads:

thread.start()

For thread in threads:

thread.join()