

VTU Question Paper Solution & Scheme

Introduction to Python Programming

Session June/July 2023

Q 1 a Demonstrate with example print (), input and string replication. [6 Marks]

Explanation & Example - [3 Marks]

Explanation & Code- [3 Marks]

A 1 a print (): -The print () function displays the string value inside the parentheses on the screen.

Code: -

```
print('Hello world!')
print('What is your name?') # ask for their name
```

input (): -The input () function waits for the user to type some text on the keyboard and press enter.

Code: -

```
myName = input()
```

String replication (): - The * operator is used for multiplication when it operates on two integer or floating-point values. But when the * operator is used on one string value and one integer value, it becomes the string replication operator.

Code: -

```
>>> 'Alice' * 5
'AliceAliceAliceAliceAlice'
```

Q 1 b Develop a program to generate Fibonacci sequences of length (N). Read N from the console. [6 Marks]

Full Program- [4 Marks]

Logic- [2 Marks]

A 1 b Code: -

```

nterms = int(input("How many terms? "))
n1, n2 = 0, 1
count = 0
if nterms <= 0:
    print("Please enter a positive integer")
elif nterms == 1:
    print("Fibonacci sequence upto",nterms,":")
    print(n1)
else:
    print("Fibonacci sequence:")
    while count < nterms:
        print(n1)
        nth = n1 + n2
        n1 = n2
        n2 = nth
        count += 1

```

Q 1 c Explain elif, for, while, break and continue statements in python with examples for each. [8 Marks]

Explanation - [5 Marks]

Example & Code- [3 Marks]

A 1 c elif: - In Python, elif is short for "else if" and is used when the first if statement isn't true, but you want to check for another condition. An else statement can be combined with an if statement. An else statement contains the block of code that executes if the conditional expression in the if statement resolves to 0 or a FALSE value.

In code, an elif statement always consists of the following:

- The elif keyword
- A condition (that is, an expression that evaluates to True or False)
- A colon
- Starting on the next line, an indented block of code (called the elif clause)

Code: -

```
if name == 'Alice':  
    print('Hi, Alice.')
```

```
elif age < 12:  
    print('You are not Alice, kiddo.')
```

while Loop Statements: - Python While Loop is used to execute a block of statements repeatedly until a given condition is satisfied. And when the condition becomes false, the line immediately after the loop in the program is executed.

In code, a while statement always consists of the following:

- The while keyword
- A condition (that is, an expression that evaluates to True or False)
- A colon
- Starting on the next line, an indented block of code (called the while clause)

Code: -

```
spam = 0  
while spam < 5:  
    print('Hello, world.')
```

```
    spam = spam + 1
```

break Statements: - There is a shortcut to getting the program execution to break out of a while loop's clause early. If the execution reaches a break statement, it immediately exits the while loop's clause. In code, a break statement simply contains the break keyword.

Code: -

```
while True:  
    print('Please type your name.')
```

```
    name = input()
```

```
    if name == 'your name':
```

```
        break
```

```
print('Thank you!')
```

continue Statements: - Like break statements, continue statements are used inside loops. When the program execution reaches a continue statement, the program execution immediately jumps back to the start of the loop and reevaluates the loop's condition. (This is also what happens when the execution reaches the end of the loop.)

Code: -

```
while True:
    print('Who are you?')
    name = input()
    if name != 'Joe':
        continue
    print('Hello, Joe. What is the password? (It is a fish.)')
    password = input()
    if password == 'swordfish':
        break
print('Access granted.')
```

for loop:-

A for loop is used for iterating over a sequence (that is either a list, a tuple, a dictionary, a set, or a string). This is less like the for keyword in other programming languages, and works more like an iterator method as found in other object-orientated programming languages. With the for loop we can execute a set of statements, once for each item in a list, tuple, set etc. In code, a for statement looks something like for i in range(5): and always includes the following:

- The for keyword
- A variable name
- The in keyword
- A call to the range () method with up to three integers passed to it
- A colon
- Starting on the next line, an indented block of code (called the for clause)

Code: -

```
print('My name is')
for i in range(5):
    print('Jimmy Five Times (' + str(i) + ')')
```

Q 2 a What are user defined functions? How can we pass parameters in user defined functions? Explain with suitable examples [5 Marks]

Explanation & Example - [2 Marks]

Method & Code- [3 Marks]

A 2 a Python User-defined functions: -All the functions that are written by any of us come under the category of user-defined functions. Below are the steps for writing user-defined functions in Python.

- In Python, a def keyword is used to declare user-defined functions.
- An indented block of statements follows the function name and arguments which contains the body of the function.

Syntax:

```
def function_name():  
    statements  
    .  
    .
```

Here we have created the fun function and then called the fun() function to print the statement.

```
# Declaring a function  
def fun():  
    print("Inside function")  
# Driver's code  
# Calling function  
fun()
```

Output:

Inside function

How to pass a parameter to a function

The purpose of function parameters in Python is to allow a programmer using the function to define variables dynamically within the function.

It's important to note that after passing a parameter to a function, an argument should be passed to the parameter of the function whenever you call the key function in your program.

Note: An **argument** is a value that is supplied to the parameter whenever the function is called.

Now, let's pass a parameter to a function that will help us store the name of our customers by returning the argument (real name) we pass to that function.

Code

```
# defining a function and passing the name parameter to it  
def greet_customer(name):  
    print (f'Hi {name} welcome aboard to the home of delicacies!')  
# calling the function and passing Theophilus as the argument  
greet_customer('Theophilus')
```

```
# calling the function and passing Chidalu as the argument
greet_customer('Chidalu')
```

Output

Hi Theophilus welcome aboard to the home of delicacies!

Hi Chidalu welcome aboard to the home of delicacies!

Explanation

- In the code above, we use the def keyword to show that we are defining a function.
- The name of our function is greet_customer(name). (name) is the parameter of the function.
- The double line breaks must be included after defining a function.
- (Theophilus) and (Chidalu) are the arguments passed to the greet_customer(name) function whenever it is called.
- greet_customer(Theophilus) and greet_customer('Chidalu) are used to called the key function, greet_customer(name). Without calling the key function, the program will not execute the defined function.

Q 2b Explain local & Global scope with variables for each [8 Marks]

Explanation- [4 Marks]

*Example & Code-[4=2*2 Marks]*

A 2b A variable is only available from inside the region it is created. This is called **scope**. A variable created inside a function belongs to the *local scope* of that function, and can only be used inside that function.

Eg: def myfunc():

```
    x = 300
```

```
    print(x)
```

```
myfunc()
```

- ***Local Variables Cannot Be Used in the Global Scope***

- This code results in an error.

def spam():

```
    eggs = 31337
```

```
spam()
```

```
print(eggs)
```

- ***Local Scopes Cannot Use Variables in Other Local Scopes***

- ***Global Variables Can Be Read from a Local Scope-Example***

def spam():

```
    print(eggs)
```

```
    eggs = 42
```

```
spam()
```

```
print(eggs)
```

- It is acceptable to use the same variable name for a global variable and local variables in different scopes in Python

Q 2 c Develop a program to read the name and year of birth of a person. Display whether the person is a senior citizen or not. [7 Marks]

Full Program- [4 Marks]

Logic-[3 Marks]

A 2 c

Code:-

```
from datetime import date
def calculateAge(birthDate):
    today = date.today()
    age = today.year - birthDate.year -
        ((today.month, today.day) <
         (birthDate.month, birthDate.day))

    return age
# Driver code
print(calculateAge(date(1997, 2, 3)), "years")
```

Or

Code:-

```
age = int (input ("Enter the age of the person: "))
#Condition to check whether the person is senior citizenship or not:
if age >= 60:
    status = "Senior citizenship "
else:
    status = "Not a senior citizenship "
print ("The person is ", status)
```

Q 3 a What is List? Explain append (), insert () and remove () methods with examples. [8 Marks]

Explanation- [2 Marks]

*Method & Code- [6=2*3 Marks]*

A 3 a

Lists are just like dynamically sized arrays, declared in other languages (vector in C++ and ArrayList in Java). Lists need not be homogeneous always which makes it the most powerful tool in Python. A single list may contain DataTypes like Integers, Strings, as well as Objects.

Lists are mutable, and hence, they can be altered even after their creation. Lists have various methods, the most commonly used list method are *append()*, *insert ()*, *extend()*, etc.

Append

It adds an element at the end of the list. The argument passed in the append function is added as a single element at end of the list and the length of the list is increased by 1.

Syntax:

```
list_name.append(element)
```

The element can be a string, integer, tuple, or another list.

Example:

```
# python program to demonstrate
# working of append function
# assign list
l = ['geeks']
# use method
l.append('for')
l.append('geeks')
# display list
print(l)
```

Output:

```
['geeks', 'for', 'geeks']
```

Insert

This method can be used to insert a value at any desired position. It takes two arguments-element and the index at which the element has to be inserted.

Syntax:

```
list_name(index,element)
```

The element can be a string, object, or integer.

Example:

```
# python program to demonstrate
# working of insert function
```



```
# assign list
l = ['geeks', 'geeks']
# use method
l.insert(1, 'for')
# display list
print(l)
```

Output:

```
['geeks', 'for', 'geeks']
```

remove (item):

This method is used to remove particular item from the list.

Create a python file with the following script to see the use remove () method. If the item value that is used as the argument value of remove () method exists in the list the item will be removed. Here, the value, 'Juice' exists in the list and it will be removed after running the script.

Define the list

```
list = ['Cake', 'Pizza', 'Juice', 'Pasta', 'Burger']
```

Print the list before delete

```
print("List before delete")
print(list)
```

Remove an item

```
list.remove('Juice')
```

Print the list after delete

```
print("List after delete")
print(list)
```

Output:

The following output will appear after running the script.

```
List before delete
```

```
['Cake', 'Pizza', 'Juice', 'Pasta', 'Burger']
```

```
List after delete
```

```
['Cake', 'Pizza', 'Pasta', 'Burger']
```

Q 3 b Explain the following methods with example:

i) keys () ii) values () iii) items () in a dictionary [12 Marks]

*Explanation- [6 Marks=3*2]*

*Method & Code- [6 Marks =3*2]*

A 3 b items()

The items() method returns a view object that displays a list of dictionary's (key, value) tuple pairs. The syntax of items() method is:

dictionary.items()

Example

```
marks = {'Physics':67, 'Maths':87}
```

```
print(marks.items())
```

```
# Output: dict_items([('Physics', 67), ('Maths', 87)])
```

Values()

The values() method returns a view object. The view object contains the values of the dictionary, as a list. The view object will reflect any changes done to the dictionary, see example below.

Syntax

dictionary.values()

Example:

```
car = {
```

```
    "brand": "Ford",
```

```
    "model": "Mustang",
```

```
    "year": 1964
```

```
}
```

```
x = car.values()
```

```
print(x)
```

output:

```
dict_values(['Ford', 'Mustang', 1964])
```

Keys():

The keys() method returns a view object. The view object contains the keys of the dictionary, as a list. The view object will reflect any changes done to the dictionary, see example below

Syntax

`dictionary.keys()`

Example:

```
car = {  
    "brand": "Ford",  
    "model": "Mustang",  
    "year": 1964  
}  
x = car.keys()
```

`print(x)`

output:

`dict_keys(['brand', 'model', 'year'])`

Q 4 a How is a tuple different from a list and what function is used to convert list to tuple?

Explain. [6 Marks]

Explanation- [4 Marks]

Example code- [2 Marks]

A 4 a

Sno	LIST	TUPLE
1	<u>Lists</u> are <u>mutable</u>	<u>Tuples</u> are immutable
2	The implication of iterations is Time-consuming	The implication of iterations is comparatively Faster
3	The list is better for performing operations, such as insertion and deletion.	A Tuple data type is appropriate for accessing the elements
4	Lists consume more memory	Tuple consumes less memory as compared to the list

Sno	LIST	TUPLE
5	Lists have several built-in methods	Tuple does not have many built-in methods.
6	Unexpected changes and errors are more likely to occur	In a tuple, it is hard to take place.

Convert the List to a Tuple in Python

You can convert a list to a tuple using the built-in tuple() function. For example, you first create a list called mylist containing the integers values. Then, use the tuple() function to convert the list to a tuple and store the result in a new variable called tuples. This function returns the tuple type.

Create a list

```
my_list = [22, 42, 53, 64, 85]
```

Convert the list to a tuple

```
tuples = tuple(my_list)
```

```
print(tuples)
```

Output:

```
# (22, 42, 53, 64, 85)
```

You can also create the convert function that takes the list and returns the tuple. For example, the **convert()** function takes a single argument **my_list**, which is the list that you want to convert to a tuple. Inside the function, you call the **tuple()** function, passing it in **my_list** as an argument. The **tuple()** function converts the list to a tuple and returns it.

Q 4 b List merits of dictionary over list.[4 Marks]

*Explanation- [4 Marks=1*4]*

A 4 b Lists in Python Python possesses a list as a data structure that is an ordered sequence of elements and mutable in nature. Each item or value that is inside of a list is called an element. Just as strings are defined as characters between quotes, lists are defined by having values between square brackets ([]) separated by commas.

A list is great to use when you have to deal with the related values. As lists are mutable, you can add, update or delete the elements of the list at any moment while programming.

For Example:

```
sample_list = ["Python", "is", "Fun"]  
print(sample_list)
```

Output

```
['Python', 'is', 'Fun']
```

Dictionary in Python Dictionary is a default python data structure used to store the data collection in the form of key-value pairs. Dictionaries are written inside the curly brackets ({}), separated by commas. However, the key and value of the data are separated by placing a semi-colon between them (:). Dictionary elements are ordered, changeable, and do not allow duplicates. Remember that the key name of every data value should be unique and are case-sensitive. Later, you can access the dictionary elements by simply using the key name and retrieving its corresponding data value.

For Example:

```
sample_dict = {  
    "vegetable": "potato",  
    "fruit": "banana",  
    "chocolate": "gems"  
}  
print(sample_dict)
```

Output

```
{'vegetable': 'potato', 'fruit': 'banana', 'chocolate': 'gems'}
```

Parameters	List	Dictionary
Basics	A list refers to a collection of various index value pairs like that in the case of an array in C++.	A dictionary refers to a hashed structure of various pairs of keys and values.
Creation	We can create a list by placing all the available elements into a [] and separating them using “,” commas.	We can create a dictionary by placing all the available elements into a { } in the form of a key:value. Here, we have to separate each pair of available key-values using the “,” commas.

Data Type	The indices in the case of a list are basically integers that start from the value 0.	The keys present in a dictionary can easily be of any given data type.
Mode of Accessing	We can access the elements in a key using indices.	We can access the elements present in a dictionary using the key-values.
Order of Elements	The entered order of elements is always maintained.	We don't have any guarantee of maintaining the order of the available elements.

Q 4 c Read N numbers from the console and create a list. Develop a program to compute and print mean, variance and standard deviation with messages. [10 Marks]

Full Program- [6 Marks]

Logic- [4 Marks]

A 4 c

```
//To find mean, variance and standard deviation of given numbers
import statistics
a = list()
number = int(input("Enter the number of elements you want: "))
print('Enter numbers in array: ')
for i in range(int(number)):
    n = input("number : ")
    a.append(int(n))
print("Given 'n' numbers: ")
print(a)
m = statistics.mean(a)
v = statistics.variance(a)
st = statistics.stdev(a)
print("Mean: ",m)
print("Variance: ",v)
print("Standard Deviation: ",st)
```

```
Output:
Given 'n' numbers:
[1, 2, 3, 4, 5]
Mean: 3
Variance: 2.5
Standard Deviation: 1.5811388300841898
```

Q 5 a Explain the following methods with suitable examples. [8 Marks]

Explanation- [4 Marks]

Example code- [4 Marks]

i) upper()

ii) lower()

iii) is_upper()

iv) is_lower()

A 5 a i) upper ()

i) The upper () and lower () string methods return a new string where all the letters in the original string have been converted to uppercase or lower- case, respectively. Non-Letter characters in the string remain unchanged.

Code: -

```
>>> spam = 'Hello world!'
>>> spam = spam.upper()
>>> spam
'HELLO WORLD!'
>>> spam = spam.lower()
>>> spam
'hello world!'
```

If you want to change the original string, you have to call upper () or lower () on the string and then assign the new string to the variable where the original was stored. The upper() and lower() methods are helpful if you need to make a case-insensitive comparison.

ii) lower ()

ii) Code: -

```
print('How are you?')
feeling = input()
if feeling.lower() == 'great':
    print('I feel great too.')
else:
    print('I hope the rest of your day is good.')
```

iii) is_upper()

iii) Code: -

```
>>> spam = 'Hello world!'
>>> spam.islower()
False
>>> spam.isupper()
False
>>> 'HELLO'.isupper()
True
>>> 'abc12345'.islower()
True
>>> '12345'.islower()
False
>>> '12345'.isupper()
False
```

iv) `is_lower()`

iv) Code: -

```
True
>>> '12345'.islower()
False
>>> '12345'.isupper()
False
```

Q5 b Illustrate with examples opening of a file with `open ()` function, reading the contents of the file with `read ()` and writing to files with `write ()` [12 Marks]

Explanation- [4 Marks]

Example code- [4 Marks]

Logic- [4 Marks]

A5 b Text files with the .txt extension or Python script files with the .py extension are examples of plaintext files. These can be opened with Windows's Notepad or OS X's TextEdit application. Programs can easily read the contents of plaintext files and treat them as an ordinary string value. Binary files are all other file types, such as word processing documents, PDFs, images, spreadsheets, and executable programs.

Opening Files with the `open ()` Function

To open a file with the `open ()` function, you pass it a string path indicating the file you want to open; it can be either an absolute or relative path. The `open ()` function returns a File object.

Code: -

```
>>> helloFile = open('C:\\Users\\your_home_folder\\hello.txt')
```

Reading the Contents of Files

Now that you have a File object, you can start reading from it. If you want to read the entire contents of a file as a string value, use the File object's `read ()` method.

Code: -

```
>>> helloContent = helloFile.read()
>>> helloContent
'Hello world!'
```

Writing to Files

Python allows you to write content to a file in a way similar to how the `print ()` function “writes” strings to the screen. You can't write to a file you've opened in read mode, though. Instead, you

need to open it in “write plaintext” mode or “append plaintext” mode, or write mode and append mode for short. Write mode will overwrite the existing file and start from scratch, just like when you overwrite a variable’s value with a new value. Pass 'w' as the second argument to open () to open the file in write mode. Append mode, on the other hand, will append text to the end of the existing file.

If the filename passed to open () does not exist, both write and append mode will create a new, blank file. After reading or writing a file, call the close () method before opening the file again.

Code: -

```
>>> baconFile = open('bacon.txt', 'w')
>>> baconFile.write('Hello world!\n')
13
>>> baconFile.close()
>>> baconFile = open('bacon.txt', 'a')
>>> baconFile.write('Bacon is not a vegetable.')
25
>>> baconFile.close()
>>> baconFile = open('bacon.txt')
>>> content = baconFile.read()
>>> baconFile.close()
>>> print(content)
Hello world!
Bacon is not a vegetable.
```

Q 6a Explain the steps involved in adding bullets to Wiki, Markup, Support with appropriate code (10 Marks)

Explanation – [4 Marks]

Example code– [6 Marks]

A 6 a When editing a Wikipedia article, you can create a bulleted list by putting each list item on its own line and placing a star in front. The bulletPointAdder.py script will get the text from the clipboard, add a star and space to the beginning of each line, and then paste this new text to the clipboard. For example: -

Lists of animals

Lists of aquarium life

Lists of biologists by author abbreviation

Lists of cultivars

and then ran the bulletPointAdder.py program, the clipboard would then contain the following:

*** Lists of animals**

*** Lists of aquarium life**

*** Lists of biologists by author abbreviation**

*** Lists of cultivars**

Step 1: Copy and Paste from the Clipboard

You want the bulletPointAdder.py program to do the following:

1. Paste text from the clipboard
2. Do something to it
3. Copy the new text to the clipboard

```
#!/ python3
# bulletPointAdder.py - Adds Wikipedia bullet points to the start
# of each line of text on the clipboard.

import pyperclip
text = pyperclip.paste()

pyperclip.copy(text)
```

Step 2: Separate the Lines of Text and Add the Star

```
#!/ python3
# bulletPointAdder.py - Adds Wikipedia bullet points to the start
# of each line of text on the clipboard.

import pyperclip
text = pyperclip.paste()

# Separate lines and add stars.
lines = text.split('\n')
for i in range(len(lines)): # loop through all indexes in the "lines" list
    lines[i] = '* ' + lines[i] # add star to each string in "lines" list

pyperclip.copy(text)
```

Step 3: Join the Modified Lines

```
#!/ python3
# bulletPointAdder.py - Adds Wikipedia bullet points to the start
# of each line of text on the clipboard.

import pyperclip
text = pyperclip.paste()

# Separate lines and add stars.
lines = text.split('\n')
for i in range(len(lines)): # loop through all indexes for "lines" list
    lines[i] = '* ' + lines[i] # add star to each string in "lines" list
text = '\n'.join(lines)
pyperclip.copy(text)
```

Q 6 b Develop a program to sort the content of a text file and write the sorted contents into a separate text file. [Use strip (), len (), list methods sort (), append & file methods open(), readlines() & write()]. [10 Marks]

Full Program- [6 Marks]

Logic- [4 Marks]

A 6 b To sort the contents of a file, firstly we need to open the file in 'read' mode. After this, we will open the specific file using statement given below-

file = open("filename.extension")

`file = open("filename.extension")`

For example → `file = open("sample.txt")`

```
def sorting(filename):
    infile = open(filename)
    words = []
    for line in infile:
        temp = line.split()
        for i in temp:
            words.append(i)
    infile.close()
    words.sort()
    outfile = open("result.txt", "w")
    for i in words:
        outfile.writelines(i)
        outfile.writelines(" ")
    outfile.close()
    sorting("sample.txt")
```

Q 7 a How do you copy files and folders using the Shutil module. Explain in detail.

[6 Marks]

Explanation – [3 Marks]

Example code– [3 Marks]

A 7 a The shutil module in Python provides functions for working with files and directories. You can use shutil to copy, move, and rename files

i) Copying files: To copy a file from one location to another, use the `shutil.copy(src, dst)` method. The `src` argument is the path of the source file, and the `dst` argument is the path of the destination file. If `dst` is a directory, the file is copied with the same name to that directory

Code Snippet

```
import shutil
# Copy a file from src to dst
src = 'path/to/source/file.txt'
dst = 'path/to/destination/file.txt'
shutil.copy(src, dst)
```

ii) Moving files: To move a file from one location to another, use the `shutil.move(src, dst)` method. The `src` argument is the path of the source file, and the `dst` argument is the path of the destination file. If `dst` is a directory, the file is moved with the same name to that directory

file is moved with the same name to that directory

Code Snippet

```
import shutil
# Move a file from src to dst
src = 'path/to/source/file.txt'
dst = 'path/to/destination/file.txt'
shutil.move(src, dst)
Both shutil.copy() and shutil.move() can handle both files and directories.
```

iii) **Renaming files:** To rename a file, use the `os.rename(src, dst)` method. The `src` argument is the path of the source file, and the `dst` argument is the new name of the file import os

Code Snippet

```
# Rename a file
src = 'path/to/source/file.txt'
dst = 'path/to/source/new_file.txt'
os.rename(src, dst)
```

Q 7b What are assertions? Write the contents of an assert statement. Explain them with examples. [8 Marks]

Explanation – [4 Marks]

Example code– [4 Marks]

7 b Assertions are statements in Python that can be used to check if a condition is true. They are often used as a debugging aid to ensure that the assumptions made by the programmer about the state of the program are correct. If the assertion is false, an `AssertionError` is raised, indicating that there is a bug in the program. An assert statement consists of the `assert` keyword, followed by a condition that is expected to be true.

Example.

```
x = 5
```

```
assert x == 5, "x is not 5"
```

Here, we use the `assert` statement to check that the variable `x` has the value 5. If the condition `x == 5` is false, an `AssertionError` is raised with the message "x is not 5"

Assertions can be used to add automated tests in codes to catch bugs early in the development process. For example, we can use assertions to check that the output of a function is correct for a given input.

Code Snippet:

```
def square(x):  
    return x ** 2  
assert square(2) == 4, "square(2) should be 4"  
assert square(3) == 9, "square(3) should be 9"  
assert square(-2) == 4, "square(-2) should be 4"
```

Here, we define a function square that returns the square of a number. We then use assert statements to check that the output of the function is correct for different input values.

Q 7 c Illustrate the logging levels in python. [6 Marks]

Explanation – [3 Marks]

Example code– [3 Marks]

A 7 c Logging is a great way to understand what’s happening in your program and in what order it's happening. Python’s logging module makes it easy to create a record of custom messages that you write. These log messages will describe when the program execution has reached the logging function call and list any variables you have specified at that point in time. On the other hand, a missing log message indicates a part of the code was skipped and never executed. Logging levels provide a way to categorize your log messages by importance. There are five logging levels, described in Table from least to most important. Messages can be logged at each level using a different logging function.

```
import logging  
logging.basicConfig(level=logging.DEBUG, format=' %(asctime)s - %(levelname)s  
- %(message)s')
```

Level	Logging Function	Description
-------	------------------	-------------

DEBUG	logging.debug ()	The lowest level. Used for small details. Usually, you care about these messages only when diagnosing problems.
INFO	logging.info ()	Used to record information on general events in your program or confirm that things are working at their point in the program.
WARNING	logging.warning()	Used to indicate a potential problem that doesn't prevent the program from working but might do so in the future.
ERROR	logging.error()	Used to record an error that caused the program to fail to do something.
CRITICAL	logging.critical()	The highest level. Used to indicate a fatal error that has caused or is about to cause the program to stop running entirely.

The benefit of logging levels is that you can change what priority of logging message you want to see. Passing logging.DEBUG to the basicConfig() function's level keyword argument will show messages from all the logging levels (DEBUG being the lowest level).

Q 8 a With suitable code, explain Backing up a folder into a Zip File. Clearly mention the step involved [12 Marks]

Explanation- [3 Marks]

Full Program- [5 Marks]

Logic- [4 Marks]

A 8 a Python can be used to extract all files and folders within a zip file. This can be done through the zipfile module. The zipfile module allows us to work with zip files, doing things such as creating zip files, reading zip files, extracting all files and folders from the zip file, and adding files to zip files. So, if you want an example of a zip file if you don't have your own, you can use this basic zip file here: documents.zip. This is a simple zip file composed of 4 files: document1.txt, document2.txt, document3.txt, and document4.txt. This is a zip file, meaning it is compressed.

Code Snippet

```
>>>import zipfile
>>> import os
>>> os.getcwd()
'C:\\Users\\dlhyl\\AppData\\Local\\Programs\\Python\\Python38-32'
>>> myzipfile= zipfile. ZipFile('documents.zip')
>>> myzipfile.extractall()
>>> myzipfile.close()
```

Step involved

Step 1: Figure Out the ZIP File's Name

Step 2: Create the New ZIP File

Step 3: Walk the Directory Tree and Add to the ZIP File

Q 8 b Explain the logging module and debug the factorial of number program [8 Marks]

Explanation – [3 Marks]

Example code– [3 Marks]

Logic- [2 Marks]

A 8 b To enable the logging module to display log messages on your screen as your program runs, copy the following to the top of your program (but under the #! python shebang line):

Code: -

```
import logging
logging.basicConfig(level=logging.DEBUG, format=' %(asctime)s - %(levelname)s
- %(message)s')
```

Code: -

```
import logging
logging.basicConfig(level=logging.DEBUG, format=' %(asctime)s - %(levelname)s
- %(message)s')
logging.debug('Start of program')

def factorial(n):
    logging.debug('Start of factorial(%s%%)' % (n))
    total = 1
    for i in range(n + 1):
        total *= i
        logging.debug('i is ' + str(i) + ', total is ' + str(total))
    logging.debug('End of factorial(%s%%)' % (n))
    return total

print(factorial(5))
logging.debug('End of program')
```

Q 9 a What is a class? How to define a class in python? How to initiate a class and how the class members are accessed [8 Marks]

Explanation – [5 Marks]

Example code– [3 Marks]

A 9 a A Class is like an object constructor, or a "blueprint" for creating objects.


Example

To create a class, use the keyword class:

```
class MyClass:
```

```
    x = 5
```

```
class Point: """Represents a point in 2-D space."""
```

Chapter 15. 

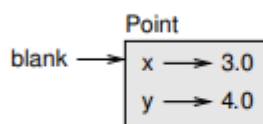


Figure 15.1: Object diagram.

The header indicates that the new class is called Point. The body is a docstring that explains what the class is for. You can define variables and methods inside a class definition, but we will get back to that later. Defining a class named Point creates a class object. >>> Point Because Point is defined at the top level, its “full name” is `__main__.Point`. The class object is like a factory for creating objects. To create a Point, you call Point as if it were a function. >>> `blank = Point()`

>>> blank <__main__.Point object at 0xb7e9d3ac> The return value is a reference to a Point object, which we assign to blank. Creating a new object is called instantiation, and the object is an instance of the class. When you print an instance, Python tells you what class it belongs to and where it is stored in memory (the prefix 0x means that the following number is in hexadecimal)

Q 9 b Define pure function. Illustrate with an example python program [8 Marks]

Explanation – [5 Marks]

Example code– [3 Marks]

A 9 b Pure Function: - A function is called pure function if it always returns the same result for same argument values and it has no side effects like modifying an argument (or global variable) or outputting something. The only result of calling a pure function is the return value.

Examples of pure functions are strlen(), pow(), sqrt() etc.

Example Code: - A function called addTime, which calculates the sum of two Times.

```
def addTime(t1, t2):
    sum = Time()
    sum.hours = t1.hours + t2.hours
    sum.minutes = t1.minutes + t2.minutes
    sum.seconds = t1.seconds + t2.seconds
    return sum
```

The function creates a new Time object, initializes its attributes, and returns a reference to the new object. This is called a pure function because it does not modify any of the objects passed to it as arguments and it has no side effects, such as displaying a value or getting user input.

```
>>> currentTime = Time()
>>> currentTime.hours = 9
>>> currentTime.minutes = 14
>>> currentTime.seconds = 30

>>> breadTime = Time()
>>> breadTime.hours = 3
>>> breadTime.minutes = 35
>>> breadTime.seconds = 0

>>> doneTime = addTime(currentTime, breadTime)
>>> printTime(doneTime)
```

We'll create two Time objects: currentTime, which contains the current time; and breadTime, which contains the amount of time it takes for a breadmaker to make bread. The output of this program is 12:49:30, which is correct.

9 c Explain Printing objects. [4 Marks]

Explanation- [2 Marks]

Example- [2 Marks]

A 9 c An Object is an instance of a Class. A class is like a blueprint while an instance is a copy of the class with actual values. When an object of a class is created, the class is said to be instantiated. All the instances share the attributes and the behavior of the class. But the values of those attributes, i.e., the state are unique for each object. A single class may have any number of instances.

Printing objects give us information about the objects we are working with

Code:-

```
# Defining a class
```

```
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)
```

```
    def __str__(self):
```

```
        return "From str method of Test: a is % s, " \
```

```
            "b is % s" % (self.a, self.b)
```

```
# Driver Code
```

```
t = Test(1234, 5678)
```

```
# This calls __str__()
```

```
print(t)
```

```
# This calls __repr__()
```

```
print([t])
```

Q 10 a What is Polymorphism? Demonstrate polymorphism with functions to find histogram to count numbers of times each letter appears in a word and in sentence [10 Marks]

Explanation – [4 Marks]

Example code– [3 Marks]

Logic- [3 Marks]

A 10 a Polymorphism: -Polymorphism is a term used in Python to refer to an object's ability to take on multiple forms. The term is derived from two distinct terms: poly, which means numerous, and morphs, which means forms.

```
def multadd (x, y, z):  
    return x * y + z
```

We can invoke it with numeric values:

```
>>> multadd (3, 2, 1)  
7
```

Or with Points:

```
>>> p1 = Point(3, 4)  
>>> p2 = Point(5, 7)  
>>> print multadd (2, p1, p2)  
(11, 15)  
>>> print multadd (p1, p2, 1)  
44
```

```
def histogram(s):  
    d = dict()  
    for c in s:  
        if c not in d:  
            d[c] = 1  
        else:  
            d[c] = d[c]+1  
    return d
```

Q 10 b Write Deck methods to add, remove, shuffle & sort cards, with illustrating the problems. [10 Marks]

Explanation – [4 Marks]

Example code– [3 Marks]

Logic- [3 Marks]

A 10 bClass definition for Deck.

The init method creates the attribute cards and generates the standard set of fifty-two cards:

Code:-

```

class Card:
    def __init__(self, suit=0, rank=2):
        self.suit = suit
        self.rank = rank

    ...

class Card:
    suitList = ["Clubs", "Diamonds", "Hearts", "Spades"]
    rankList = ["narf", "Ace", "2", "3", "4", "5", "6", "7",
                "8", "9", "10", "Jack", "Queen", "King"]

    #init method omitted

    def __str__(self):
        return (self.rankList[self.rank] + " of " +
                self.suitList[self.suit])

```

Deck: -

The following is a class definition for the Deck class. The initialization method creates the attribute `cards` and generates the standard set of fifty-two cards:

```

class Deck:
    def __init__(self):
        self.cards = []
        for suit in range(4):
            for rank in range(1, 14):
                self.cards.append(Card(suit, rank))

```

Printing the Deck: -

```

class Deck:
    ...
    def printDeck(self):
        for card in self.cards:
            print card

```

Add: -

one space more than the previous card.

```

class Deck:
    ...
    def __str__(self):
        s = ""
        for i in range(len(self.cards)):
            s = s + " " + str(self.cards[i]) + "\n"
        return s

```

Shuffling & sort the deck: -

```
class Deck:
    ...
    def shuffle(self):
        import random
        nCards = len(self.cards)
        for i in range(nCards):
            j = random.randrange(i, nCards)
            self.cards[i], self.cards[j] = self.cards[j], self.cards[i]
```

Removing: -

```
class Deck:
    ...
    def removeCard(self, card):
        if card in self.cards:
            self.cards.remove(card)
            return True
        else:
            return False
```