

USN

Internal Assessment Test 2 – JUNE 2025

Sub:	INTRODUCTION TO PYTHON PROGRAMMING				Sub Code:	BPLCK205B	Branch:	ISE
Date:	20/06/2025	Duration :	90 min's	Max Marks:	50	Sem/Sec:	II ALL SECTIONS	OBE
<u>Answer any FIVE FULL Questions</u>							MARKS	CO RB T
1a)	Explain The Following Methods With Suitable Example: i) upper() ii) lower() iii) isupper() iv) islower()						[4]	CO3 L2
1b)	Explain the following file operations in Python with suitable example using shutil: i) Copying files and folders ii) Moving folder/directory iii) Permanent deletion of files and folders						[6]	CO3 L2
2 a)	Illustrate With Example Function Of Opening Of A File , Reading The Contents Of File, Writing To Files						[5]	CO3 L3
2 b)	Explain five buttons available in Debug control window						[5]	CO3 L2
3 a)	Briefly Explain Saving Variables With Shelve Module With Example.						[5]	CO3 L2
3 b)	Briefly explain assertions and raising an exception with an example.						[5]	CO3 L2

HoD Signature

CCI signature

Course Instructor

USN

Internal Assessment Test 2 – JUNE 2025

Sub:	INTRODUCTION TO PYTHON PROGRAMMING				Sub Code:	BPLCK205B	Branch:	ISE
Date:	20/06/2025	Duration :	90 min's	Max Marks:	50	Sem/Sec:	II ALL SECTIONS	OBE
<u>Answer any FIVE FULL Questions</u>							MARKS	CO RB T
1a)	Explain The Following Methods With Suitable Example: i) upper() ii) lower() iii) isupper() iv) islower()						[4]	CO3 L2
1b)	Explain the following file operations in Python with suitable example using shutil: i) Copying files and folders ii) Moving folder/directory iii) Permanent deletion of files and folders						[6]	CO3 L2
2 a)	Illustrate With Example Function Of Opening Of A File , Reading The Contents Of File, Writing To Files						[5]	CO3 L3
2 b)	Explain five buttons available in Debug control window						[5]	CO3 L2
3 a)	Briefly Explain Saving Variables With Shelve Module With Example.						[5]	CO3 L2
3 b)	Briefly explain assertions and raising an exception with an example.						[5]	CO3 L2


HoD Signature

CCI signature

Course Instructor

4a	Write A Syntax To Find I) File Size And Folder Contents Ii) Checking Path Validity	4	C03	L2
4b	Write a function named DivExp which takes TWO parameters a, b and returns a value c ($c=a/b$). Write a suitable assertion for $a>0$ in function DivExp and raise an exception for when $b=0$. Develop a suitable program which reads two values from the console and calls a function DivExp.	6	C03	L3
5a	Briefly explain the printing of objects with examples.	5	C04	L2
5b	What is a class? How to define a class in python? How to initiate a class and how the class members are accessed?	5	C04	L2
6	Develop a program to demonstrate differences between pure function and modifier functions	10	C04	L3

4a	Write A Syntax To Find I) File Size And Folder Contents Ii) Checking Path Validity	4	C03	L2
4b	Write a function named DivExp which takes TWO parameters a, b and returns a value c ($c=a/b$). Write a suitable assertion for $a>0$ in function DivExp and raise an exception for when $b=0$. Develop a suitable program which reads two values from the console and calls a function DivExp.	6	C03	L3
5a	Briefly explain the printing of objects with examples.	5	C04	L2
5b	What is a class? How to define a class in python? How to initiate a class and how the class members are accessed?	5	C04	L2
6	Develop a program to demonstrate differences between pure function and modifier functions	10	C04	L3

USN					<div><div>CELEBRATING 25 YEARS</div><div></div><div>CMR INSTITUTE OF TECHNOLOGY, BENGALURU</div><div>ACCREDITED WITH A+ GRADE BY NAAC</div></div>				
Internal Assessment Test I I – JUNE 2025									
Sub:	INTRODUCTION TO PYTHON PROGRAMMING				Sub Code:	BPLCK205B	Branch:	ISE	
Date:	Duration:	90 min's	Max Marks:	50	Sem/Sec:	II ALL SECTIONS		OBE	
Answer any FIVE FULL Questions							MARKS	CO	RBT
1a)	Explain The Following Methods With Suitable Example: i) upper() ii) lower() iii) isupper() iv) islower() Solution: 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. Nonletter characters in the string remain unchanged. spam = 'Hello world!' spam = spam.upper() spam 'HELLO WORLD!' spam = spam.lower() spam 'hello world!' These methods do not change the string itself but return new string values. The upper() and lower() methods are helpful if you need to make a case-insensitive comparison. E.G., 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.') output: How are you? GREAt I feel great too. The isupper() and islower() methods will return a Boolean True value if the string has at least one letter and all the letters are uppercase or lowercase, respectively. Otherwise, the method returns False. Enter the following into the interactive shell, and notice what each method call returns: spam = 'Hello world!' spam.islower() False spam.isupper() False 'HELLO'.isupper() True 'abc12345'.islower() True						4	CO3	L2
1b)	Explain the following file operations in Python with suitable example using shutil: i) Copying files and folders ii) Moving folder/directory iii) Permanent deletion of files and folders Solution: i) Copying files and folders The shutil (or shell utilities) module has functions to let you copy, move, rename, and delete files in your Python programs. To use the shutil functions, you will first need to use import shutil. The shutil module provides functions for copying files, as well as entire folders. Calling shutil.copy(source, destination) will copy the file at the						6	CO3	L2

	<p>path source to the folder at the path destination. (Both source and destination are strings.) If destination is a filename, it will be used as the new name of the copied file. This function returns a string of the path of the copied file. While shutil.copy() will copy a single file, shutil.copytree() will copy an entire folder and every folder and file contained in it. Calling shutil.copytree(source, destination) will copy the folder at the path source, along with all of its files and subfolders, to the folder at the path destination.</p> <pre>>>> import shutil, os >>> os.chdir('C:\\') >>> shutil.copy('C:\\spam.txt', 'C:\\delicious') 'C:\\delicious\\spam.txt' >>> shutil.copy('eggs.txt', 'C:\\delicious\\eggs2.txt') 'C:\\delicious\\eggs2.txt' >>> shutil.copytree('C:\\bacon', 'C:\\bacon_backup') 'C:\\bacon_backup'</pre> <p>ii) Moving file & Folders Calling shutil.move(source, destination) will move the file or folder at the path source to the path destination and will return a string of the absolute path of the new location. If destination points to a folder, the source file gets moved into destination and keeps its current filename.</p> <pre>>>> import shutil >>> shutil.move('C:\\bacon.txt', 'C:\\eggs') 'C:\\eggs\\bacon.txt'</pre> <p>iii) Permanently deleting file & Folders You can delete a single file or a single empty folder with functions in the os module, whereas to delete a folder and all of its contents, you use the shutil module.</p> <ul style="list-style-type: none"> • Calling os.unlink(path) will delete the file at path. • Calling os.rmdir(path) will delete the folder at path. This folder must be empty of any files or folders. • Calling shutil.rmtree(path) will remove the folder at path, and all files and folders it contains will also be deleted. <p>Code:</p> <pre>import os for filename in os.listdir(): if filename.endswith('.txt'): os.unlink(filename)</pre>			
2 a)	<p>Illustrate With Example Function Of Opening Of A File , Reading The Contents Of File, Writing To Files</p> <p>Solution: 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. Create a text file named hello.txt using Notepad or TextEdit. Type Hello, world! as the content of this text file and save it in your user home folder.</p> <p>Then enter the following into the interactive shell: helloFile = open(Path.home() / 'hello.txt') The open() function can also accept strings helloFile = open('C:\\Users\\your_home_folder\\hello.txt')</p> <p>Both these commands will open the file in “reading plaintext” mode, or read mode for short. When a file is opened in read mode, Python lets you only read data from the file; you can’t write or modify it in any way. Read mode is the default mode for files you open in Python. But if you don’t want to rely on Python’s defaults, you can explicitly specify the</p>	5	CO3	L3

	<p>mode by passing the string value 'r' as a second argument to open(). So open ('/Users/Al/hello.txt', 'r') and open('/Users/Al/hello.txt') do the same thing.</p> <p>Reading the Contents of Files</p> <p>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.</p> <pre>helloContent = helloFile.read() helloContent 'Hello, world!'</pre> <p>Alternatively, you can use the readlines() method to get a list of string values from the file, one string for each line of text.</p> <p>Writing to Files</p> <p>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.</p> <p>Write mode will overwrite the existing file and start from scratch. 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. Pass 'a' as the second argument to open() to open the file in append mode.</p> <p>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.</p> <pre>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.</pre> <p>First, we open bacon.txt in write mode. Since there isn't a bacon.txt yet, Python creates one. Calling write() on the opened file and passing write() the string argument 'Hello, world! \n' writes the string to the file and returns the number of characters written, including the newline. Then we close the file.</p> <p>To add text to the existing contents of the file instead of replacing the string we just wrote, we open the file in append mode. We write 'Bacon is not a vegetable.' to the file and close it.</p>			
2	<p>Explain five buttons available in Debug control window</p> <p>b) Solution:</p> <p>The program will stay paused until you press one of the five buttons in the Debug</p> <hr/> <p>Control window: Go, Step, Over, Out, or Quit.</p> <p>Go</p> <p>Clicking the Go button will cause the program to execute normally until it terminates or reaches a breakpoint. If you are done debugging and want the program to continue normally, click the Go button.</p> <p>Step</p> <p>Clicking the Step button will cause the debugger to execute the next line of code and then pause again. The Debug Control window's list of global and local variables will be updated if their values change. If the next line of code is a function call, the debugger will “step into” that function and jump to the first line of</p>	5	CO3	L2

	<p>code of that function.</p> <p>Over Clicking the Over button will execute the next line of code, similar to the Step button. However, if the next line of code is a function call, the Over button will “step over” the code in the function. The function’s code will be executed at full speed, and the debugger will pause as soon as the function call returns.</p> <p>Out Clicking the Out button will cause the debugger to execute lines of code at full speed until it returns from the current function. If you have stepped into a function call with the Step button and now simply want to keep executing instructions until you get back out, click the Out button to “step out” of the current function call.</p> <p>Quit If you want to stop debugging entirely and not bother to continue executing the rest of the program, click the Quit button. The Quit button will immediately terminate the program. If you want to run your program normally again, select Debug4Debugger again to disable the debugger.</p>			
3	Briefly Explain Saving Variables With Shelve Module With Example.	5	CO3	L2
a)	<p>Solution: You can save variables in your Python programs to binary shelf files using the shelve module. This way, your program can restore data to variables from the hard drive. The shelve module will let you add Save and Open features to your program.</p> <pre>import shelve shelfFile = shelve.open('mydata') cats = ['Tom', 'Simon'] shelfFile['cats'] = cats shelfFile.close()</pre> <p>To read and write data using the shelve module, you first import shelve. Call shelve.open() and pass it a filename, and then store the returned shelf value in a variable. You can make changes to the shelf value as if it were a dictionary. When you’re done, call close() on the shelf value. Here, our shelf value is stored in shelfFile.</p> <p>After running the previous code, you will see three new files in the current working directory: mydata.bak, mydata.dat, and mydata.dir. These binary files contain the data you stored in your shelf. The format of these binary files is not important; you only need to know what the shelve module does, not how it does it. The module frees you from worrying about how to store your program’s data to a file. we can use the shelve module to later reopen and retrieve the data from these shelf files.</p> <pre>shelfFile = shelve.open('mydata') shelfFile['cats'] ['Tom', 'Simon'] shelfFile.close()</pre> <p>Just like dictionaries, shelf values have keys() and values() methods that will return list-like values of the keys and values in the shelf. Since these methods return list-like values instead of true lists, you should pass them to the list() function to get them in list form.</p> <pre>shelfFile = shelve.open('mydata') list(shelfFile.keys())</pre>			

	<pre>['cats'] list(shelfFile.values()) [['Tom', 'Simon']] shelfFile.close()</pre>			
3 b)	<p>Briefly explain assertions and raising an exception with an example.</p> <p>Solution:</p> <p>Assertions</p> <p>An assertion is a sanity check to make sure your code isn't doing something obviously wrong. These sanity checks are performed by assert statements. If the sanity check fails, then an AssertionError exception is raised.</p> <p>In code, an assert statement consists of the following:</p> <ul style="list-style-type: none"> • The assert keyword • A condition (that is, an expression that evaluates to True or False) • A comma • A string to display when the condition is False. In plain English, an assert statement says, "I assert that the condition holds true, and if not, there is a bug somewhere, so immediately stop the program." For example, enter the following into the interactive shell: <pre>>>> ages = [26, 57, 92, 54, 22, 15, 17, 80, 47, 73] >>> ages.sort() >>> ages [15, 17, 22, 26, 47, 54, 57, 73, 80, 92] >>> assert ages[0] <= ages[-1], "Assert that the first age is <= the last age"</pre> <p>The assert statement here asserts that the first item in ages should be less than or equal to the last one. This is a sanity check; if the code in sort() is bug-free and did its job, then the assertion would be true.</p> <p>Because the ages[0] <= ages[-1] expression evaluates to True, the assert statement does nothing. However, let's pretend we had a bug in our code. Say we accidentally called the reverse() list method instead of the sort() list method. When we enter the following in the interactive shell, the assert statement raises an AssertionError:</p> <pre>>>> ages = [26, 57, 92, 54, 22, 15, 17, 80, 47, 73] >>> ages.reverse() >>> ages [73, 47, 80, 17, 15, 22, 54, 92, 57, 26]</pre> <p>Raising Exceptions</p> <p>Python raises an exception whenever it tries to execute invalid code. One can also raise his own exceptions in your code. Raising an exception is a way of saying, "Stop running the code in this function and move the program execution to the except statement."</p> <p>Exceptions are raised with a raise statement. In code, a raise statement consists of the following:</p> <ul style="list-style-type: none"> • the raise keyword • A call to the Exception() function • A string with a helpful error message passed to the Exception() function. <p>For example, open a new file editor tab, enter the following code, and save the program as boxPrint.py:</p> <pre>def boxPrint(symbol, width, height): if len(symbol) != 1: raise Exception('Symbol must be a single character string.') if width <= 2: raise Exception('Width must be greater than 2.') if height <= 2: raise Exception('Height must be greater than 2.') print(symbol * width)</pre>	5	CO3	L2

	<pre>width) for i in range(height - 2): print(symbol + (' ' * (width - 2)) + symbol) print(symbol * width) for sym, w, h in ((' ', 4, 4), ('O', 20, 5), ('x', 1, 3), ('ZZ', 3, 3)): try: boxPrint(sym, w, h) except Exception as err: print('An exception happened: ' + str(err))</pre> <p>Here we've defined a boxPrint() function that takes a character, a width, and a height, and uses the character to make a little picture of a box with that width and height. This box shape is printed to the screen. Say we want the character to be a single character, and the width and height to be greater than 2. We add if statements to raise exceptions if these requirements aren't satisfied. Later, when we call boxPrint() with various arguments, our try/except will handle invalid arguments. This program uses the except Exception as err form of the except statement. If an Exception object is returned from boxPrint(), this except statement will store it in a variable named err. We can then convert the Exception object to a string by passing it to str() to produce a user-friendly error message the output will look like this:</p> <pre>**** * * * * **** OOOOOOOOOOOOOOOOOOOOOO O O O O O O OOOOOOOOOOOOOOOOOOOOOO An exception happened: Width must be greater than 2. An exception happened: Symbols must be a single character string.</pre>			
4 a)	<p>Write A Syntax To Find</p> <p>I) File Size And Folder Contents</p> <p>II) Checking Path Validity</p> <p>Solution:</p> <p>Finding File Sizes and Folder Contents:</p> <p>The os.path module provides functions for finding the size of a file in bytes and the files and folders inside a given folder.</p> <ul style="list-style-type: none"> • Calling os.path.getsize(path) will return the size in bytes of the file in the path argument. • Calling os.listdir(path) will return a list of filename strings for each file in the path argument <p>E.G.,</p> <pre>os.path.getsize('C:\\Windows\\System32\\calc.exe') 27648 os.listdir('C:\\Windows\\System32') ['0409', '12520437.cpx', '12520850.cpx', '5U877.ax', 'aaclient.dll', --snip-- 'xwtpdoui.dll', 'xwtpw32.dll', 'zh-CN', 'zh-HK', 'zh-TW', 'zipfldr.dll']</pre> <p>Checking Path Validity</p> <p>Path objects have methods to check whether a given path exists and</p>	4	CO3	L2

	<p>whether it is a file or folder. Assuming that a variable p holds a Path object, you could expect the following:</p> <p>Calling p.exists() returns True if the path exists or returns False if it doesn't exist.</p> <p>Calling p.is_file() returns True if the path exists and is a file, or returns False otherwise.</p> <p>Calling p.is_dir() returns True if the path exists and is a directory, or returns False otherwise.</p> <p>winDir.exists() True</p> <p>winDir.is_dir() True</p> <p>calcFile.is_file() True</p> <p>calcFile.is_dir() False</p> <p>You can determine whether there is a DVD or flash drive currently attached to the computer by checking for it with the exists() method</p>			
4 b)	<p><u>Write a function named DivExp which takes TWO parameters a, b and returns a value c (c=a/b). Write a suitable assertion for a>0 in function DivExp and raise an exception for when b=0. Develop a suitable program which reads two values from the console and calls a function DivExp.</u></p> <p>Solution:</p> <pre>def DivExp(a, b): # Assert that a > 0 assert a > 0, "Value of a must be greater than 0" # Raise an exception if b == 0 if b == 0: raise ZeroDivisionError("Exponent b cannot be zero") return a/b # Main program try: # Read inputs from the user a = int(input("Enter value for a: ")) b = int(input("Enter value for b: ")) # Call the function and print result result = DivExp(a, b) print(f"Result: {a}^{b} = {result}") except Exception as e: print(f"An unexpected error occurred: {e}") Enter the value of a => 5 Enter the value of b => 0 b value cannot be 0</pre>	6	CO3	L3
5 a)	<p>Briefly explain the printing of objects with examples.</p> <p>Solution: Explanation 2 marks</p>	5	CO4	L2

	<p>Program 3 marks</p> <p>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.</p> <p>Printing objects give us information about the objects we are working with. In Python, this can be achieved by using <code>_repr_</code> or <code>_str_</code> methods. <code>_repr_</code> is used if we need a detailed information for debugging while <code>_str_</code> is used to print a string version for the users.</p> <p>Code:-</p> <pre> class Test: # Defining a class 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) t = Test(1234, 5678) print(t) # This calls __str__ () print([t]) # This calls __repr__() Output:- From str method of Test: a is 1234, b is 5678 [Test a:1234 b:5678] </pre>			
5 b)	<p>What is a class? How to define a class in python? How to initiate a class and how the class members are accessed?</p> <p>Solution: Explanation 3 marks Program 2 marks A programmer-defined type is called a class. A class definition looks like this: class Point: """Represents a point in 2-D space."""</p> <p>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.</p> <p>Defining a class named Point creates a class object.</p> <pre> >>> Point <class main .Point> </pre> <p>Because Point is defined at the top level, its “full name” is <code>main .Point</code>. The class object is like a factory for creating objects. To create a Point, you call Point as if it were a function.</p> <pre> >>> blank = Point() >>> blank </pre>	5	CO4	L2

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). Every object is an instance of some class, so “object” and “instance” are interchangeable.

You can assign values to an instance using dot notation:

```
>>> blank.x = 3.0
```

```
>>> blank.y = 4.0
```

though, we are assigning values to named elements of an object. These elements are called attributes. A state diagram that shows an object and its attributes is called an object diagram.

```
import math

def print_point(p):
    print('(%g, %g)' % (p.x, p.y))

def distance(q):
    d = math.sqrt((q.x)**2+(q.y)**2)
    print('Distance is %g' % (d))

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

blank = Point()
blank.x = 3.0
blank.y = 4.0

print_point(blank)
distance(blank)

(3, 4)
Distance is 5
```

6 Develop a program to demonstrate differences between pure

Solution:

Pure functions:

Explanation: 2marks

Program : 3marks

The function that creates a new Time object, initializes its attributes with new values 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 effect, like displaying a value or getting user input, other than returning a value.

```
class Time:
    """Represents the time of day.
    attributes: hour, minute, second"""

def add_time(t1, t2):
    sum = Time()
    sum.hour = t1.hour + t2.hour
    sum.minute = t1.minute + t2.minute
    sum.second = t1.second + t2.second

    if sum.second >= 60:
        sum.second -= 60
        sum.minute += 1

    if sum.minute >= 60:
        sum.minute -= 60
        sum.hour += 1

    return sum

def print_time(t):
    print('Hour: ', t.hour, '\nMinute: ', t.minute, '\nSeconds: ', t.second)

start = Time()
start.hour = 9
start.minute = 45
start.second = 0

duration = Time()
duration.hour = 1
duration.minute = 35
duration.second = 0

done = add_time(start, duration)
print_time(done)
print(start.hour)
```

10

CO4

L3

Hour: 11
Minute: 20
Seconds: 0
9

MODIFIER FUNCTION

Explanation: 2marks

Program: 3marks

sometimes it is useful for a function to modify the objects or instances it gets as parameters. In that case, the changes are visible to the caller. Functions that work this way are called **modifiers**. `increment()` function adds a given number of seconds to a Time object or instance which is visible to the called function.

```
class Time:
    """Represents the time of day.
    attributes: hour, minute, second"""

    def increment(time, seconds):
        time.second += seconds

        m = int(time.second/60)           # 5000 // 60= 83
        time.second = time.second - (m*60) # 5000-(83*60)=20
        time.minute += m                  # 45+83 = 128

        h = int(time.minute/60)           # 128//60= 2
        time.minute = time.minute - (h*60) # 128-2*60=128-120=8
        time.hour += h                    # 9 + 2 = 11

    def print_time(t):
        print('Hour:', t.hour, '\nMinute: ', t.minute, '\nSeconds: ', t.second)

start = Time()
start.hour = 9
start.minute = 45
start.second = 0

seconds = 5000
#print_time(start)
increment(start, seconds)
print_time(start)
print(start.hour)
```

Hour: 11
Minute: 8
Seconds: 20
11

Course Instructor

HoD Signature

CCI signature