Roll No.				

Internal Assessment Test 1 – Apr 2024

Sub:	**Introduction to Python Programming – SCHEME AND SOLUTION**			Sub code:	BPLCK205 B	Branch:	Chemistry Cycle	
Date:	13-04-2024	Duration:	90 min's	Max Marks: 50		Sem / Sec:	II / Chemistry Cycle	OBE

			MARKS	CO	RBT
		<u>Answer any FIVE FULL QUESTIONS</u>			
1	(a)	Explain the usage of following functions with example code i) input() ii) print() iii) range() iv) len() v)type()	[5]	CO1	L2

Correct definition/description [2 marks]

Example [3 marks]

i) input()

Gets user input from the console and assigns it to a variable

```
name = input("Enter your name: ")
```

ii) print()

Prints output to the console

```
print("Hello, World!")
```

iii) range()

Generates a range of integers from 0 to 9

```
range(0, 10)
```

iv) len()

Returns the length of a string

```
len("Python")
```

v)type()

Returns the type of the object as "str"

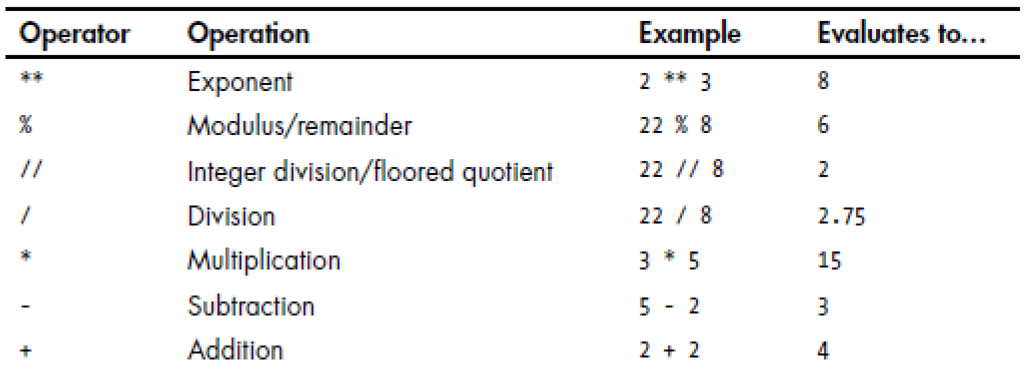
```
type("Python")
```

(b) List and explain different Math operators used in Python with example. [5] CO1 L2

Correct definition/description [2 marks]

Example and output [3 marks]

In Python, 2 + 2 is called an expression, which is the most basic kind of programming instruction in the language. Expressions consist of values (such as 2) and operators (such as +), and they can always evaluate down to a single value. A single value with no operators is also considered an expression, though it evaluates only to itself. There are plenty of operators used in Python expressions.



Operator	Operation	Example	Evaluates to...
**	Exponent	2 ** 3	8
%	Modulus/remainder	22 % 8	6
//	Integer division/floored quotient	22 // 8	2
/	Division	22 / 8	2.75
*	Multiplication	3 * 5	15
-	Subtraction	5 - 2	3
+	Addition	2 + 2	4

The order of operations (also called precedence) of Python math operators is similar to that of mathematics. The ** operator is evaluated first, then *, /, //, and % operators are evaluated next, from left to right; and the + and - operators are evaluated last (also from left to right). Parentheses can be used to override the usual precedence.

Program:

```
N= (5 - 1) * ((7 + 1) / (3 - 1))
print(N)
```

Output:

```
16.0
```

2 (a) Describe about Control statements (if, else, elif) with example [5] CO1 L2

Correct definition/description [2 marks]

Example and output [3 marks]

if Statements

The most common type of flow control statement is if statement. An if statement’s clause (that is, the block following the if statement) will execute if the statement’s condition is True. The clause is skipped if the condition is False. An if statement consists of the following:

- The if 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 if clause)

else Statements

An if clause can optionally be followed by an else statement. The else clause is executed only when the if statement's condition is False. An else statement doesn't have a condition, and in code, an else statement always consists of the following:

- The else keyword
- A colon
- Starting on the next line, an indented block of code (called the else clause)

elif Statements

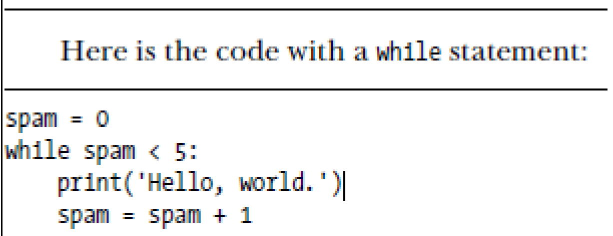
f statement is an "else if" statement that always follows an if or another elif statement. It provides another condition that is checked only if any of the previous conditions were False. In code, an elif statement always consists of the following:

- 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)

Example:

```
number = 0
if number > 0:
  print('Positive number')
elif number <0:
   print('Negative number')
else:
```

	print('Zero') **output:** Zero			

(b)	Differentiate while loop and for loop with example code snippets
	Correct definition/description [2 marks]
	Example and output [3 marks]
	while Loop Statements
	Block of code execute over and over again with a while statement. The code in a while clause will be executed as long as the while statement’s condition is True. 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)
	
	Here is the code with a while statement:
	`spam = 0`
	`while spam < 5:`
	`    print('Hello, world.')`
	`    spam = spam + 1`
	Output:
	Hello, world.
	Hello, world.
	Hello, world.
	Hello, world.
	Hello, world.
	for Loops and the range() Function:
	while loop keeps looping while its condition is True but for loop execute a block of
		[5]	CO1	L2

code only a certain number of times.

- 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)

Example:

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

Output:

```
My name is
Jimmy Five Times (0)
Jimmy Five Times (1)
Jimmy Five Times (2)
Jimmy Five Times (3)
Jimmy Five Times (4)
```

3 (a) Explain string concatenation and replication with example. [5] CO1 L2

Correct definition/description [2 marks]

Example and output [3 marks]

String Concatenation

String Concatenation is the technique of combining two strings. String Concatenation can be done using the ‘+’ Operator. This operator can be used to add multiple strings together. However, the arguments must be a string.

Program:

```
var1 = "Hello"
var2 = "Geek"
var3 = var1 + var2
print(var3)
```

output:

HelloGeek

Here, the + Operator combines the string that is stored in the var1 and var2 and stores in another variable var3.

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.

Program:

```
N=”Alice” * 5
print(N)
```

Output:

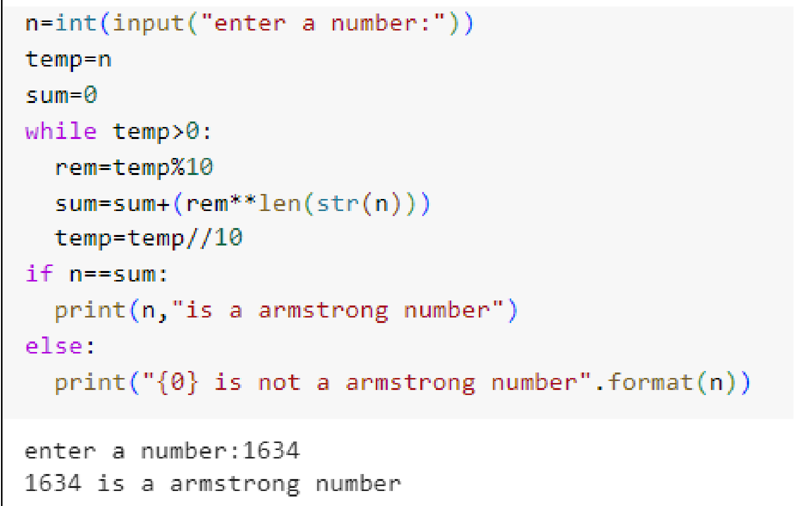
```
'AliceAliceAliceAliceAlice’
```

The expression evaluates down to a single string value that repeats the original a number of times equal to the integer value.

(b)	Write a python program to check whether a number is Armstrong number or not	[5]	CO1	L3

syntax [2 marks]

logic and output [3 marks]



```
n=int(input("enter a number:"))
temp=n
sum=0
while temp>0:
  rem=temp%10
  sum=sum+(rem**len(str(n)))
  temp=temp//10
if n==sum:
  print(n,"is a armstrong number")
else:
  print("{0} is not a armstrong number".format(n))
```

```
enter a number:1634
1634 is a armstrong number
```

4 (a)	Define the Scope of the variable. Differentiate local scope with global scope with example code snippets.	[5]	CO1	L2

Correct definition/description [2 marks]

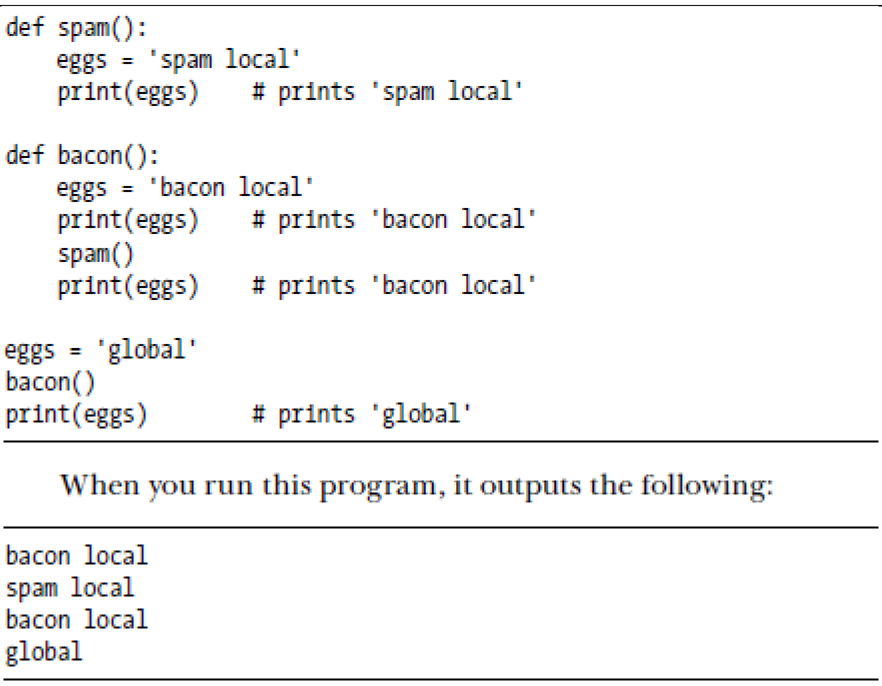
Example and output [3 marks]

Parameters and variables that are assigned inside a function are said to exist in local scope. Variables that are assigned outside all functions are said to exist in the global scope. A variable that exists in a local scope is called a local variable, while a variable that exists in the global scope is called a global variable. A variable must be one or the other; it cannot be both local and global. scope is a container for variables. When a scope is destroyed, all the values stored in the scope's variables are forgotten. There is only one global scope, and it is created when program begins. When program terminates, the global scope is destroyed, and all its variables are forgotten.

A local scope is created whenever a function is called. Any variables Assigned in this function exist within the local scope. When the function returns, the local scope is destroyed, and these variables are forgotten. The next time you call this function, the local variables will not remember the values stored in them from the last time the function was called.

Scopes matter for several reasons:

- Code in the global scope cannot use any local variables.
- However, a local scope can access global variables.
- Code in a function's local scope cannot use variables in any other local scope.
- You can use the same name for different variables if they are in different scopes.



```
def spam():
    eggs = 'spam local'
    print(eggs)    # prints 'spam local'

def bacon():
    eggs = 'bacon local'
    print(eggs)    # prints 'bacon local'
    spam()
    print(eggs)    # prints 'bacon local'

eggs = 'global'
bacon()
print(eggs)        # prints 'global'
```

When you run this program, it outputs the following:

```
bacon local
spam local
bacon local
global
```

(b)	What is a Keyword argument? Explain the use of argument in print() function with an example?	[5]	CO1	L2

Correct definition/description [2 marks]

Example and output [3 marks]

The separator between the arguments to print() function in Python is space by default, which can be modified and can be made to any character, integer or string as per our choice. However, rather than through their position, keyword arguments are identified by the keyword put before them in the function call. Keyword arguments are often used for optional parameters.

The print() function has the optional parameters **end and sep** to specify what should be printed at the end of its arguments and between its arguments (separating them), respectively.

Example:

```
print('Hello')
print('World')
```

output:

```
Hello
World
```

Using end keyword:

```
print('Hello', end=' ')

print('World')
```

output:

```
Hello World
```

Example:

```
print('cats', 'dogs', 'mice')
```

output:

```
cats dogs mice
```

using sep keyword:

```
print('cats', 'dogs', 'mice', sep=', ')
```

output:

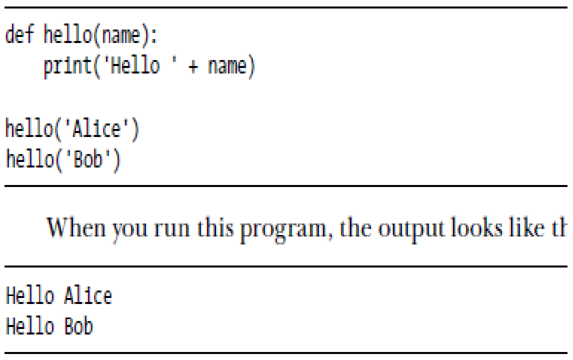
```
cats,dogs,mice
```

5(a)	How to declare and call functions in a python program? Illustrate with an example script.	[6]	CO1	L2

Correct definition/description [3 marks]

Example and output [3 marks]

A function is a block of code which only runs when it is called. You can pass data, known as parameters, into a function. A function can return data as a result. A major purpose of functions is to **group code that gets executed multiple times**.
In Python a function is defined using the **def keyword** and to **call a function**, use the function name followed by parenthesis:



```
def hello(name):
    print('Hello ' + name)

hello('Alice')
hello('Bob')
```

When you run this program, the output looks like tl

```
Hello Alice
Hello Bob
```

(b)	Write a Python Program to display Fibonacci series of length N.	[4]	CO1	L3

syntax [2 marks]

logic and output [3 marks]

```
n=int(input(“enter the number “))
a=0
b=1
sum=0
i=0
print(“fibonacci series”)
while(i<=n):
   print(sum)
   a=b
   b=sum
   sum=a+b
   i=i+1
```

output:
N=5
Fibonacci sequence =0 1 1 2 3

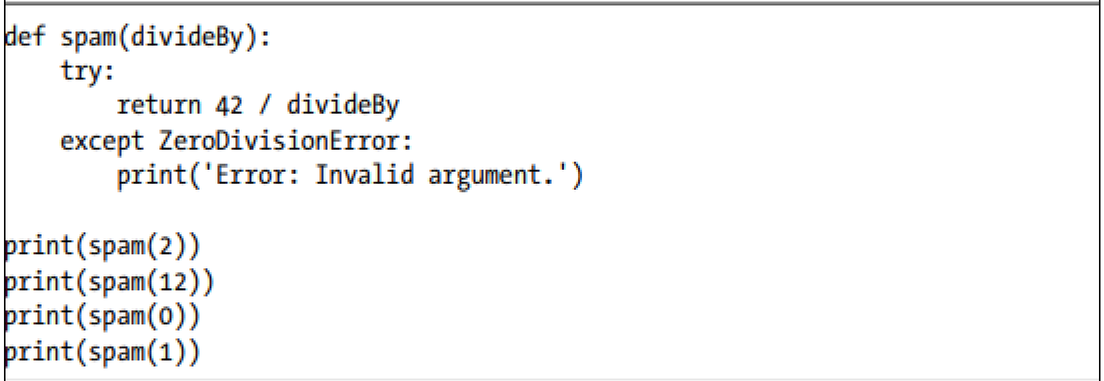
6 (a)	Explain Exception Handling in python with an example.	[6]	CO1	L2

Correct definition/description [3 marks]

Example and output [3 marks]

Exceptions are raised when the program is syntactically correct, but the code resulted in an error. This error does not stop the execution of the program, however, it changes the normal flow of the program.

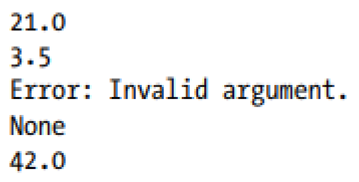
try and **except** statements are used to catch and handle exceptions in Python. Statements that can raise exceptions are kept inside the try clause and the statements that handle the exception are written inside except clause.



```
def spam(divideBy):
    try:
        return 42 / divideBy
    except ZeroDivisionError:
        print('Error: Invalid argument.')

print(spam(2))
print(spam(12))
print(spam(0))
print(spam(1))
```

Output:

	
	`21.0`
	`3.5`
	`Error: Invalid argument.`
	`None`
	`42.0`
	
	When code in a try clause causes an error, the program execution immediately moves to the code in the except clause. After running that code, the execution continues as normal.			

(b)	Write a python program to print following pattern:
	
	`*`
	`* *`
	`* * *`
	`* * * *`
	`* * * * *`
	
	syntax [2 marks]
	
	logic and output [3 marks]
	
	Program:
	
	`n = 5`
	`for i in range(0, n):`
	`    for j in range(0, i+1):`
	`        print("*", end=" ")`
	`    print()`
		[4]	CO1	L3

7 (a)	Explain Negative Indexing and slicing in List with suitable messages. **Correct definition/description [3 marks]** **Example and output [3 marks]** **NEGATIVE INDEXES** The integer value -1 refers to the last index in a list, the value -2 refers to the second-to-last index in a list, and so on. **Example:**	[6]	CO2	L2

```
>>> spam = ['cat', 'bat', 'rat', 'elephant']
>>> spam[-1]
‘elephant’
>>> spam[-3]
‘bat’
>>> 'The ' + spam[-1] + ' is afraid of the ' + spam[-3] + '.'
The elephant is afraid of the bat.
```

SLICING

Just as an index can get a single value from a list, a slice can get several values from a list, in the form of a new list. A slice is typed between square brackets, like an index, but it has two integers separated by a colon. Difference between indexes and slices is:

- spam[2] is a list with an index (one integer).
- spam[1:4] is a list with a slice (two integers).

In a slice, the first integer is the index where the slice starts. The second integer is the index where the slice ends. A slice goes up to, but will not include, the value at the second index. A slice evaluates to a new list value.

Example:

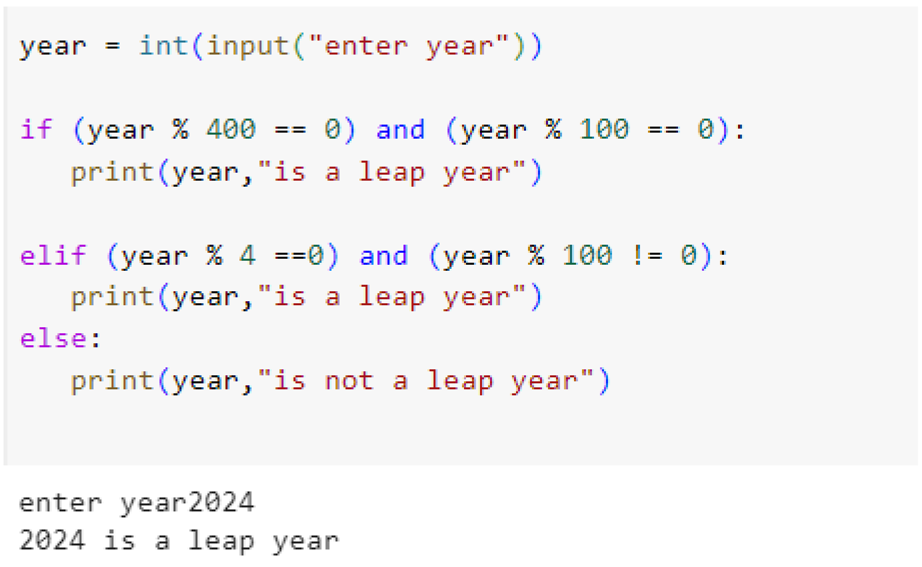
```
>>> spam = ['cat', 'bat', 'rat', 'elephant']
>>> spam[0:4]
[‘cat’, ‘bat’, ‘rat’, ‘elephant’]
>>> spam[0:-1]
['cat', 'bat', 'rat']
```

As a shortcut, we can leave out one or both of the indexes on either side of the colon in the slice. Leaving out the first index is the same as using 0, or the beginning of the list. Leaving out the second index is the same as using the length of the list, which will slice to the end of the list.

```
>>> spam = ['cat', 'bat', 'rat', 'elephant']
>>> spam[:2]
[‘cat’, ‘bat’]
>>> spam[1:]
[‘bat’, ‘rat’, ‘elephant’]
>>> spam[:]
['cat', 'bat', 'rat', 'elephant']
```

(b)	Write a python program to check whether a year is a leap year or not. **syntax [2 marks]**	[4]	CO1	L3

logic and output [3 marks]



```python
year = int(input("enter year"))

if (year % 400 == 0) and (year % 100 == 0):
   print(year,"is a leap year")

elif (year % 4 ==0) and (year % 100 != 0):
   print(year,"is a leap year")
else:
   print(year,"is not a leap year")
```

```
enter year2024
2024 is a leap year
```

CI **CCI** **HOD**