CBCS SCHEME

USN | | | | | | | | | | |

BPLCK205B / BPLCKB205

# Second Semester B.E./B.Tech. Degree Examination, June/July 2024
## Introduction to Python Programming

Time: 3 hrs. Max. Marks: 100

***Note: 1. Answer any FIVE full questions, choosing ONE full question from each module.***
***2. M : Marks , L: Bloom's level , C: Course outcomes.***

| | | Module – 1 | M | L | C |
|---|---|---|---|---|---|
| Q.1 | a. | Explain with example print( ), input( ) and len( ). | 6 | L2 | CO1 |
| | b. | Explain elif, for, while statement in python with example. | 6 | L2 | CO1 |
| | c. | Develop a program to generate Fibonacci sequence of Length(N).Read N from the console. | 8 | L3 | CO1 |
| | | **OR** | | | |
| Q.2 | a. | What are functions? Explain python function with parameters and return statement. | 6 | L2 | CO1 |
| | b. | How to handle exception in python with example. | 6 | L2 | CO1 |
| | c. | Explain Local and Global scope with variables for each. | 8 | L2 | CO1 |
| | | **Module – 2** | | | |
| Q.3 | a. | Explain the use of in and not in operator in list with examples. | 6 | L2 | CO2 |
| | b. | Explain negative Indexing, slicing, index( ), append( ), remove( ), pop( ), insert( ) and sort( ) with suitable example. | 8 | L2 | CO2 |
| | c. | Write about Mutable and Immutable data type in list. | 6 | L2 | CO2 |
| | | **OR** | | | |
| Q.4 | a. | Explain the following list methods with examples.<br>i) index ( ) ii) append( ) iii) insert( )<br>iv) sort( ) v) reverse( ) vi) List concatenation and Replication. | 10 | L2 | CO2 |
| | b. | Develop a program to read the student details like Name, USN and Marks in three subjects. Display the student details, total marks and percentage with suitable messages. | 10 | L3 | CO2 |
| | | **Module – 3** | | | |
| Q.5 | a. | Illustrate with example opening of a life with open( ) function, reading the contents of the file with read( ) and writing to files with write ( ). | 10 | L2 | CO3 |
| | b. | Explain how to save variable with the shelve module. | 10 | L2 | CO3 |

| | | | | | |
|---|---|---|---|---|---|
| | | **OR** | | | |
| **Q.6** | **a.** | Explain the following string methods with examples.<br>i) isalpha( ) ii) isalnum( ) iii) isdecimal( ) iv) isspace( ) v) istitle( ). | **10** | **L2** | **CO3** |
| | **b.** | Explain about in and not in operators in string. | **5** | **L2** | **CO3** |
| | **c.** | Explain about pyperclip module. | **5** | **L2** | **CO3** |
| | | **Module – 4** | | | |
| **Q.7** | **a.** | What are Assertions? Write the contents of an assert statement. Explain them with examples. | **10** | **L2** | **CO3** |
| | **b.** | Develop a program with a function named DivExp which takes Two parameters a, b and returns a value c(c = a/b), write 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. | **10** | **L3** | **CO3** |
| | | **OR** | | | |
| **Q.8** | **a.** | Explain about files and folders can be copied using shutil module. | **10** | **L2** | **CO3** |
| | **b.** | Explain about Debug control window. | **10** | **L2** | **CO3** |
| | | **Module – 5** | | | |
| **Q.9** | **a.** | Explain about class and objects. | **10** | **L2** | **CO4** |
| | **b.** | Explain about pure function and modifier. | **10** | **L2** | **CO4** |
| | | **OR** | | | |
| **Q.10** | **a.** | Explain the concept of prototyping Vs planning. | **10** | **L2** | **CO4** |
| | **b.** | Explain _init_ and _str_methods with examples. | **10** | **L2** | **CO4** |

* * * * *

# Introduction to Python Programming BPLCK205B
## Scheme of Solution

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

Ex: print ('Hello world!')
print ('What is your name?')

The line print ('Hello world!') means <print out the text in the string 'Hello world!'=

When Python executes this line, python is calling the print() function and String value is being passed to the function. A Value ie passed to a function call is an argument.

input() : The input() function wait for the user to type some text on the keyboard & press ENTER.

myName = input()

This function call evaluates to a String equals to user text & previous line of code assign the myName variable to this String value. The input() function call as an expression that evaluates to whatever string the user typed in. If the user entered 'Al' then the expression would evaluates to myName = 'Al'.

len() function: We can pass the len() function a string value (or variable containing a string) & the function evaluates to the integer value of the no. of characters in that string.

```
>>> print ('The length of your name:')
>>> print (len(myName))
>>> len ('hello')
5.
```

len(myName) evaluates to an integer. It is passed to print() to be displayed on the screen.

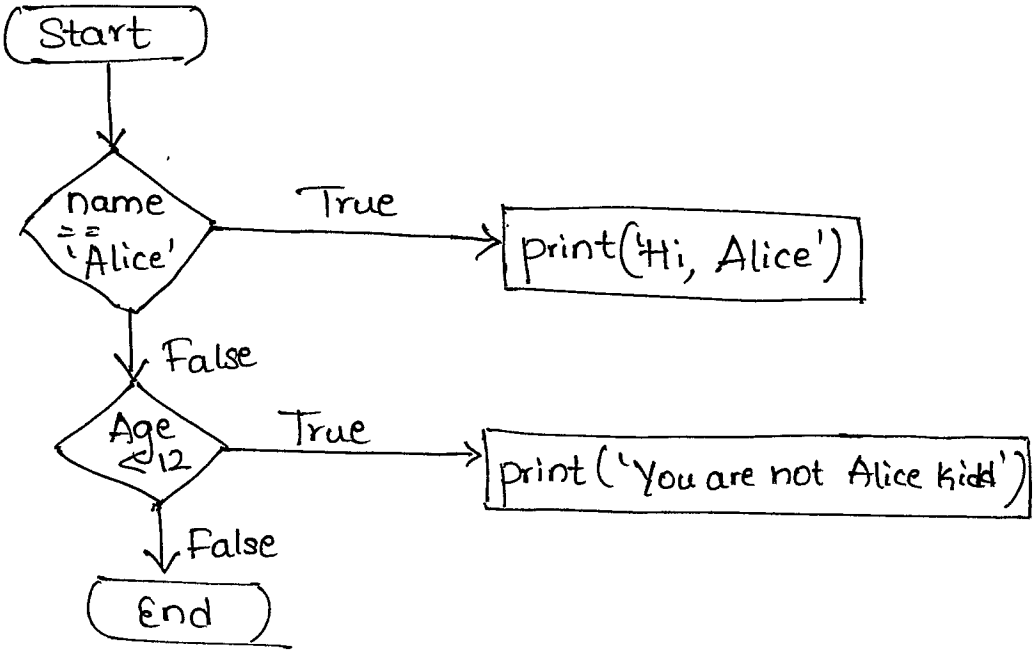
2a) elif Statement :- if or else clause will execute, we want one of many possible clauses to execute.

<else if = statement that always follow as if or another elif statement.

elif statement consists of

1) elif keyword
2) A condition
3) A colon.
4) Starting on the next line, an idented block of code.

```
if name = 'Alice':
    print ('Hi, Alice')
elif age < 12:
    print('You are not Alice, kidd')
```



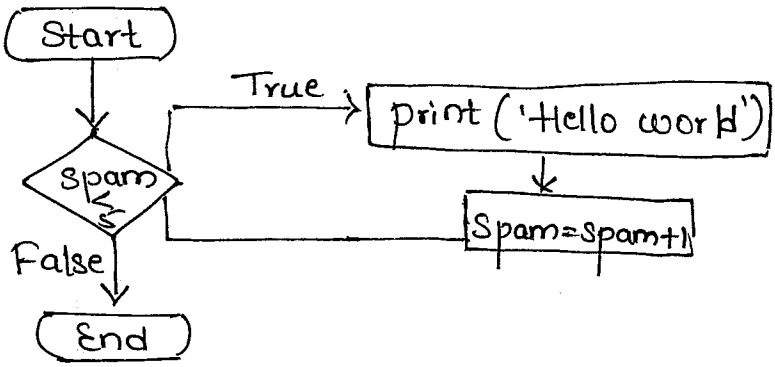

While loop Statement :-

while clause will be executed as long as while statement cond$^n$ is true. It consists of

1) while keyword
2) A cond$^n$
3) A colon
4) Starting on the next line, an indented block of code.

Ex: Spam = 0.

```
while spam < 5:
  print ('Hello world')
  Spam = Spam + 1
```




In while loop, the cond$^n$ is checked at the start of each iteration. If the cond$^n$ is true, the clause is executed & afterward, the cond$^n$ is checked again. If the cond$^n$ is false, while clause is skipped.

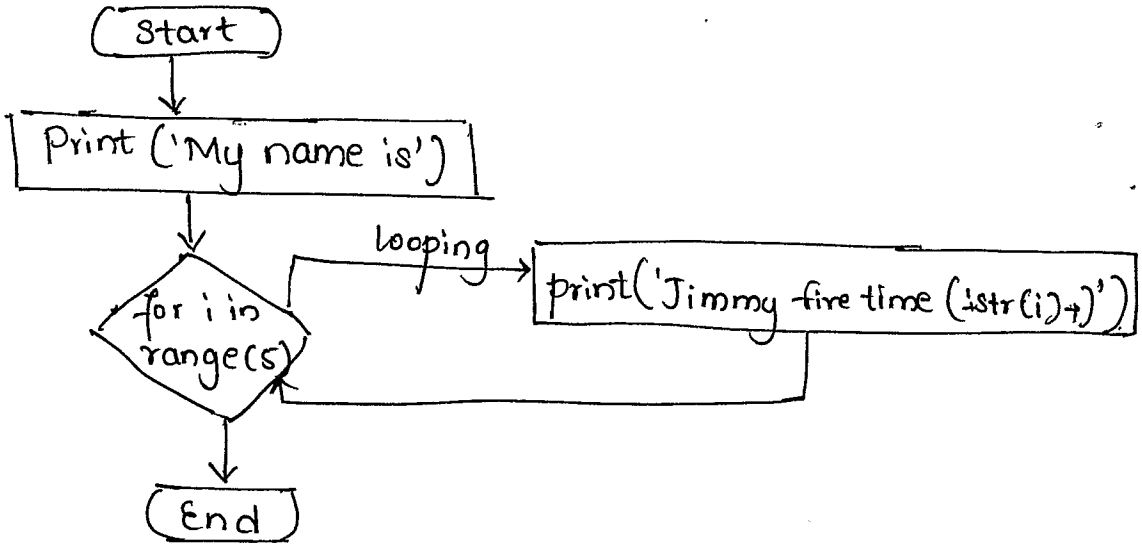
## for loop ~~function~~ statement:

If we want to execute a block of code only certain no. of times we can do for loop statement & range() functn

for statement like for i in range(5)

1) for keyword
2) A variable name
3) The in keyword
4) A call to the range() functn with upto a integer passed to it.
5) A colon
6) Starting on the next line, an indented code of block.

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




```
print ('Enter the value of n')
n = int(input())
f1 = 0
f2 = 1
print('Fibonacci series')
print(f1)
print(f2)
for x in range (2,n):
    f3 = f1 + f2
    Print (f3)
    f1 = f2
    f2 = f3
```

O/P: Enter the value of 'n':

5

Fibonacci series

0

1

1

2

3

2a) A function is like a mini program within a program

Ex:
```
def hello():
    print('How!')
    print('How!!')
    print('Hello here')

hello()
hello()
hello()
```

O/P:
```
How!
How!!
Hello here
How!
How!!
Hello here
How!
How!!
Hello here.
```

When we call print() or len() function, we pass in values called arguments, by typing them between the parenthesis.
We can also define our own functn that accepts arguments.

Ex:
```
def hello(name):
    print('Hello'+ name)
hello('Alice')
hello('Bob')
```

O/P:
```
Hello Alice
Hello Bob.
```

The defn of the hello() functn in this program has a parameter called name.
A parameter is a variable that an argument is stored, when functn is called.
When creating a functn using def statement we can specify what the return value should be with return statemen
A return statement consists of

1) return keyword
2) The value or expression that the functn should return.

When an expression is used with return statement, the return value is what this expressn evaluates to.

2b) If we don't want to crash the program due to error instead we want the program to detect errors, handle them & continue to run.

Ex:
```
def spam(divideBy):
    return 42/divideBy
print(spam(2))
print(spam(12))
print(spam(0))
print(spam(1))
```

O/P:

```
21.0
3.5
ZeroDivisionError: division by zero.
```

A ZeroDivisionError happens whenever we try to divide a no. by zero. Error can be handled with try & except statement. The code that could potientially have error is put in a try clause. The program execut$^n$ moves to the start following except clause if an error happens.

c) Parameters & variables assigned called funct$^n$ & exist in that funct$^n$ local scope. Variable that are assigned all funct$^n$ called global scope. A variable must be one or other, it cannot be local & global. When scope is destroyed, the value stored in scope's variable are forgotten. There is only one global scope, it is created when your program begins. When program terminates, global scope is destroyed and all variables are forgotten.

A local scope is created whenever a funct$^n$ is called. Any variables assigned in this funct$^n$ with local scope, is destroyed these variables are forgotten.

1) Code in the global scope cannot use only local variables
2) A local scope can access global variables.
3) Code in a funct$^n$ local scope cannot use variable in any other local scope.

4) We can use the same name for different variables if they are different scopes a local variable named spam & a global variable named spam.

Ex:
```
def spam():
    global eggs
    eggs = 'spam'
    eggs = 'global'
    spam()
    print(eggs)
```

o/p : spam.

A variable is in local scope or global scope.

1) If a variable is being used in global scope then it is always a global variables.

2) If there is a global statement for that variables in a funct$^n$ it is global variable.

3) Otherwise, if the variable is used in assignment statement in the funct$^n$ it is local variable.

4) Variable is not an assignment statement, it is global variable.

3) a) We can determine whether a value is or isn't in a list with in & out in operators. in & not in are used to express & connect two value, a value to look for in a list & list ~~val~~ where it may be found & these expression evaluate to Boolean Value.

Ex:
```
mypets = ['Zophie', 'Pooka', 'Fat-Tail']
Print('Enter a pet name')
  name = input()
if name not in mypets:
    print('I donnot have pet named' + name)
    else:
      Print(name + 'is my pet')
```

O/p: Enter a petname

Foot

I donot have a pet named foot.

b) Negative Indexing:-

We can use -ve integers for the index. The integer value -1 refers to the last index in a list, value -2 refers to 2nd to last index in a list & so on.

```
>>> spam=['cat', 'bat', 'rat', 'elephant']
>>> Spam[-1]
'elephant'
>>> spam[-3]
'bat'.
```

Slicing:- A slice is typed between square brackets like an index, it has two integers seperated by a colon.

Ex:

```
>>> spam = ['cat', 'bat', 'rat', 'elephant']
>>> spam[0:4]
['cat', 'bat', 'rat', 'elephant']
```

index(): List values have index method that can be passed a value, & if that value exists in the list, the index of the value is returned. If the value isn't the list, python produces a valueError error. When there are duplicate of the value in the list, the index of its first appearance is returned.

```
Ex: >>> spam=['cat', 'rat', 'bat']
       spam[1]
       ['rat']
```

c) Mutable & Immutable Datatype.

String:- is immutable; it cannot be changed, Trying to reassig a single character in a string results in a TypeError error.

Mutate: a string to use slicing & concatenatn to build a new strin by copying from part of old string

List: A list value is a mutable datatype. It can have value added, removed or changed.

```
>>> eggs=[1,2,3]
>>> eggs=[4,5,6]
>>> eggs
 [4,5,6]
```

Tuple:- It is almost identical to the list datatype except in 2 way

i) Tuples are typed with parentheses ( & ) instead of square bracket [ & ].

ii) Tuples instead of lists because they are immutable & their content don't change. Tuples cannot have their values modified, appended or removed.

```
>>> egg=('hello', 4.2, 0.5)
    eggs[0]
     hello.
```

```
>>> eggs[1:3]
    (4.2, 0.5)
```

4 a) i) index(): List values have index method that can be passed a value, if that value exists in the list, the index of the value is returned. If the value isn't the list, python produces a ValueError error. When there are duplicate of the value in the list, the index of its first appearance is returned.

Ex: 
```
>>> Spam=['cat','bat','rat']
    Spam[1]
    ['rat']
```

append() method: call adds the argument to the end of the list.

Ex:
```
>>> spam=['cat','dog']
>>> spam.append('bat')
>>> spam
['cat', 'dog', 'bat']
```

remove(): is passed the value to be removed from the list it is called on.

Ex:
```
>>> spam=['cat', 'bat', 'rat']
>>> spam.remove('bat')
>>> Spam
['Cat', 'rat']
```

insert(): can insert a value at any index in the list. The 1st argument to insert() is the index for new value & 2nd argument is the new value to be inserted.

Ex:
```
>>> spam=['cat','bat']
>>> Spam.insert(1,'dog')
>>> spam
['cat', 'dog', 'bat']
```

Sort(): List of no. values or lists of strings can be sorted with sort() method.

```
>>> spam = [2,5,3,-1]
>>> spam.sort()
>>> spam
-1, 2, 3, 5.
```

pop(): is used to remove an item at the specified position in a list & return it.

Ex:
```
>>> spam=['Cat', 'bat', 'rat']
>>> spam.pop(1)
['cat', 'rat']
```

ii) append() : call adds the argument to the end of the list.

Ex: >>> spam = ['cat', 'dog']
>>> spam.append('bat')
>>> spam
['cat', 'dog', 'bat']

iii) insert() : Can insert a value at any index in the list. The 1st argument to insert() is the index for new value & 2nd argument is the new value to be inserted.

Ex: >>> spam = ['cat', 'bat']
>>> spam.insert(1, 'dog')
>>> spam
['cat', 'dog', 'bat']

iv) Sort() : List of no. values or lists of strings can be sorted with sort() method.

>>> spam = [2, 5, 3, -1]
>>> spam.sort()
>>> spam
-1, 2, 3, 5

v) reverse() : method reverses the sorting order of the elements.

>>> spam = ['cat', 'bat', 'rat']
>>> spam.reverse()
>>> spam
['rat', 'bat', 'cat'].

vi) '+' Operator can combine 2 lists to Create a new list value, it combines 2 strings into a new String value.

'*' Operator used with a list & an integers value to replicate the list

Ex: >>> [1, 2, 3] + ['A', 'B', 'C']
[1, 2, 3, 'A', 'B', 'C']

```
>>> ['x', 'y', 'z'] * 3
['x', 'y', 'z', 'x', 'y', 'z', 'x', 'y', 'z']
>>> spam=[1,2,3]
>>> spam = spam + ['A', 'B', 'C']
>>> spam
[1, 2, 3, 'A', 'B', 'C']
```

b)

```
Class Student
    marks=[ ]
    def getData (self, rn, name, m1, m2, m3):
        Student.m = m
        Student.name = name
        Student.marks.append (m1)
        Student.marks.append (m2)
        Student.marks.append (m3)
    def total (self):
        return ((Student.marks[0]+Student.marks[1] + Student.
                marks[2])
    def average (self):
        return ((Student.marks[0] +Student.marks[1] +Student.
                marks[2])/3)
    def displayData (self):
        Print ("Roll no. is :", Student.rn)
        Print("Name is", Student.name)
        Print(" Marks are", Student.marks)
        Print("Total marks", self.total())
        Print("Average marks", self.average())
Print ("Enter the roll no.")
r= int(input( ))
```

```
name = input("Enter the name")
m1 = int(input("Enter the marks in first subject:")
m2 = int(input("Enter the marks in 2nd subject")
m3 = int(input("Enter the marks in 3rd subject")
S1 = Student()
S1.getData(r, name, m1, m2, m3)
S1.displayData()
```

O/P: Enter roll no.
101
Enter the name : abc
Enter the marks in first subject : 98
Enter the marks in 2nd subject: 95
Enter the marks in 3rd subject: 99

Roll no. is 101
Name is : abc
Marks are : [98, 95, 99]
Total marks are : 292
Average marks are : 97.33.

5 a) Opening the file
To open a file with open() funct$^n$, pass it a string path indicating file want to open either an absolute or relative path. The open() funct$^n$ return a file object. Creating text file named hello.txt using notepad. Type Hello world! as the Content of text file & Save it in your user home folder.

```
>>> hellofile = open("c:\\users\\your_home_folder\\hello.txt")
```

When a file opened in read mode, python only read data from file, can't write or modify it in.
Read mode is default mode for files you open in python.

```
open('users/asweight/hello.txt', 'r').
```

## Reading the contents of file.

* To read the entire contents of a file as string value, use the file objects read() method

```
>>> hellocontent = hellofile.read()
>>> hello content
'Hello world'
```

readlines() method to get a list of string values from the file, one string for each line of text.

Create a file named soni.txt in the same directory as hello.txt & write text in it.

Ex: When in disgrace with fortune
I all alone beweep.
My outcast state

```
>>> sonnetfile = open('soni.txt')
>>> sonnetfile = readlines()
```

O/P: ['When in disgrace with fortune \n', I all alone beweep
My outcast state\n'].

## Writing the file write()

To write to file opened in read mode. Need to open it in 'write plain text' mode or 'append plain text' mode or writemode & append mode for short.

Write mode will overwrite the existing file & start from scratch, when ~~you~~ overwrite a variables values with a new value. Pass 'w' as second argument to open() to open the file in writemode. Append mode on the other, will append text to end of existing file.

Pass 'a' as 2nd argument to open() to open the file in append mode.

```
>>> baconfile = open('bacon.txt', 'w')
```

```
>>> baconFile.write('Hello world\n')
   13
>>> baconfile.close()
>>> baconfile=open('bacon.txt','a')
>>> baconfile.write('Bacon is not vegetables')
   25.
>>> baconfile.close().
```

b) Save variables in python program to binary shelf files using the shelve module. Program can restore data variables from hard drive. The shelve module will add save & open features to program.

```
>>> import shelve
>>> shelffile=shelve.open('mydata')
>>> cats=['a','b','c']
>>> shelffile['cats']=cats
>>> shelffile.close()
```

* To read & write data using the shelve module, 1st modal import shelve, call shelve.open() & pass it a filename & store the returned shelf value in a variable.
* Create a list cats & write shelffile['cats']=cats to store the list in shelffile as a value associated with key 'cats'. Call close() on shelffile.
* Program can use shelve module to reopen & retrieve the data from these shelffiles. Shelf values don't have to be opened in read or write code both once opened.

```
>>> shelffile=shelve.open('mydata')
>>> type(shelffile)
<class 'shelve.DbfilenameShelf'>
>>> sheffile['cats']
['a','b','c']
```

* Shelf values have keys() & values() method will return list like values & keys in the shelf.

```
>>> shelffile= shelve.open ('mydata')
>>> list (shelffile. keys())
['cats']
>>> list (shelffile. values())
[['a', 'b', 'c']]
>>> shelffile. close().
```

6 a) i) isalpha(); return True if the string consists only of letters & is not blank.

Ex: >>>'hello'. isalpha()
True
>>>'hello123', isalpha()
False

ii) isalnum() ; return True if the string consists only of letters & numbers & is not blank.

Ex:
>>> 'hello123'. isalnum()
True.

iii) isdecimal(); return True string consists only of numeric Character & is not blank.

Ex: >>>'123'. isdecimal()
True

iv) isspace(): return True if the string only of space, tabs & newlines & is not blank

Ex: >>>' ', isspace()
True.

v) istitle() return true if string consists only of words that begins with an uppercase letter followed by lowercase letters.

Ex: >>> 'This is title case'.istitle()
True

b) The in & not in operators with strings is used with strings just like with list values. An expression with 2 strings joined using in & out evaluate to a Boolean True or False.

```
>>> 'Hello' in 'Hello world!'
True
>>> 'Hello' in 'Hello'
True
>>> 'HELLO' in 'Hello world!'
False
>>> ' ' in 'spam'
True
>>> 'cats' not in 'cats & dogs'
False.
```

c) Copying & Pasting Strings with the pyperclip Module:
The pyperclip module has copy() & paste() funct$^n$ that can send text to & receive text from Computer's Clipboard.

```
>>>import pyperclip
>>>pyperclip.copy('Hello world!')
>>>pyperclip.paste()
'Helloworld!'
```

If something outside the program changes the Clipboard contents, the paste() funct$^n$ will return it.

```
>>> pyperclip.paste()
'Hello everyone, this is python programming'.
```

7 a) An assertion is a sanity check to make sure code isn't something wrong.

* Sanity checks are performed by assert statement. Sanity check fails then Assert$^n$ Error excepth is raised.

assert consists of

1) A cond$^n$ (evaluates True or False)
2) A comma
3) A string to display when cond$^n$ is false.

Ex:- >>> podBayDoorstatus = 'open'
>>> assert podBayDoorstatus == 'open', 'The pod bay doors need to be "open".'
>>> podBayDoorstatus = 'I'm sorry, Dave. I'm afraid I can't do that."
>>> podBayDoorstatus = 'I'm sorry' 'open', 'The pod bay doors need to be "open".

Traceback (most recent call last):
File "<pyshell#10>", line 1, in <module>
assert podBayDoorstatus == 'open', 'The pod bay doors need to be "open".'
AssertionError: The pod bay doors need to be "open".

* Set podBayDoorStatus to 'open' so from now on, we fully except the value of this Variable to be 'open'.
* A lot of code under the assumption that the value is 'open' - code depends on 'open' in order to work. So we add an assert$^n$ to make sure we're right to assume podBayDoorstatus is 'open'.
* The message "The pod bay doors need to be "open" so it'll be easy what's wrong if assertion fails.

b) 
```
def diverp(a,b):
    try:
        c=a/b
        return c
    except ZeroDivisionError:
        print('invalid Argument')
Print('Enter 1st no.')
a=int(input())
print('Enter 2nd no.')
b=int(input())
res=diverp(a,b)
print('Division of two no. is', res)
```

O/P:

```
Enter 1st no.
20
Enter 2nd no.
10
Division of two no. is 2.0
Enter 1st no.
20
Enter 2nd no.
0
invalid argument
Division of two no. is None.
```

8 a) The Shutil module provides functn for copying files as well as entire folders. Calling Shutil.copy(source, destn) will copy the file at the path source to the folder at the path destinatn.

```
>>> import shutil, os
>>> from pathlib import path
>>> p=path.home()
>>> shutil.copy(p / 'spam.txt', p / 'some_folder')
```

'C:\\Users\\All\\some_folder\\spam.txt'

>>> shutil.copy(p/'eggs.txt', p/some_folder/egg2.txt')

windowsPath('C:\user/Al/some_folder/egg2.txt').

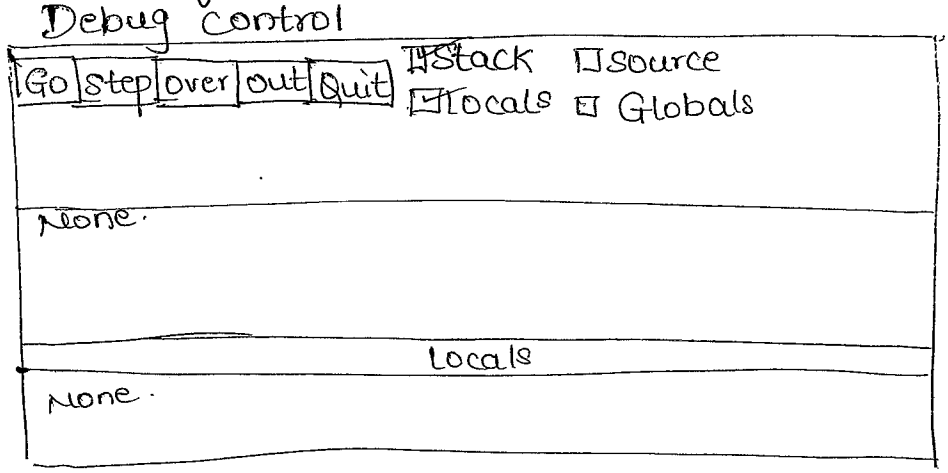
* 1st shutil.copy() call copies the files at C:\spam.txt to the folder C:\delicious. The return value is the path of newly copied file. Note a folder was specified as the destn.
* Original spam.txt filename is used for new, copied file's filename. 2nd shutil.copy() call copies the file at C:\eggs.txt to the folder.

C:\delicious but gives the copied file the name egg2.txt.

```
>>> import shutil, os
>>> from pathlib import Path
>>> p = path.home()
>>> shutil.copytree(p/'Spam', p/'spam_backup')
WindowsPath('C:\user\Al\spam_backup')
```

* Shutil.copytree() call creates a new folder named bacon_backup with same content as the original bacon folder.

b) Debug control window appears, select all 4 of stack, locals, source & Globals checkboxes so window show full sets of debug information.




Debug control window is displayed, run a program from the file editor. Debugger will pause execution before 1st instruction & display following

1) Line of code i.e about to be executed
2) A list of all local variables & their values.
3) A list of all global " " " " .
4) Program will paused until press one of 5 buttons in debug Control window.

Go: Click Go button cause the program to execute normally until it terminates or reaches a breakpoint. Done debugging & program to continue normally, Click Go button.

Step: Click Step button will cause the debugger to execute next line of code & pause again. Debug control window's list of global & local variables will updated if values change. Next line of code is funct$^n$ call, the debugger will "Step into" that funct$^n$ & jump to 1st line of code of that funct$^n$.

Over: Click over button will execute next line of code lly to step button. It will "step over" the code in funct$^n$. The funct$^n$ code will be executed at full speed & debugger will pause as soon as funct$^n$ call returns.

Quit: If want to stop debugging entirely & not bother to continue executing rest of program click quit button. It will immediately terminate the program. If want to run program normally select Debug4Debugger to disable the debugger.

9 a) A programmer-defined type called a class.

```
Class Point:
    """Represents a point in 2-D Space."""
```

The header indicates the new class is called Point. The body is a docstring that class is for. Define variables & methods inside a class def$^n$.

Defining a class named Point creates a Class object:

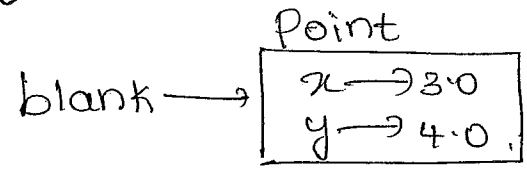
```
>>> point
<class '__main__.Point'>
```

The class object is like factory for creating objects.

```
>>> blank = Point()
>>> blank
<__main__.Point object at 0xb7ed>
```

The return value is a reference to a Point object we assign to blank.

Creating a new object is called instantiation & object is an instance of class.




-Object diagram.

b) Pure Functions: The function creates a new Time object, initializes its attributes & returns a references to new object This is called a pure function because it does not modify any of the objects passed to it as arguments & it has no effect like displaying a value or getting input user & returning a value.

```
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
    return sum
```

add_time figures.

```
>>> start = Time().
>>> start.hour=9
>>> start.minute=45
>>> start.second=0
>>> startduration = Time()
>>> duration.hour=1
>>> duration.minute=35
>>> duration.second=0
done=add_time(start, durat^n).
>>> print_tim(done)
10:80:00
```

Improved version.

```
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.
```

Modifiers:

The function to modify the object its get as parameters, the changes are visible to the caller. are called modifiers.

```
def increment (time, seconds):
    time.second += seconds.
    if time.second >=60;
        time.second -=60
        time.minute +=1
    if time.minute >=60;
        time.minute -=60
        time.hour+=1
```

The program that use pure functn are faster to develop & less error-prone than program that use modifiers. But modifier are less efficient to programs functionals.

10 a) Prototype that performed the basic calculation & tested patching errors.

A designed development, which high leve insight into the problem can make the programming much easier.

The function that converts Times to integers.

```
def time_to_int (time):
    minutes = time.hour * 60 + time.minute .
    seconds = minutes * 60 + time.second
    return seconds.
```

The funcn converts an integer to a Time.

```
def int_to_time (seconds):
    time = Time()
    minutes, time.second = divmod (seconds, 60)
    time.hour, time.minute = divmod (minutes, 60)
    return time.
```

Rewrite add_time

```
def add_time (t1, t2):
    seconds = time_to_int (t1) + time_to_int (t2)
    return int_to_time (seconds)
```

This version is shorter than original & easier to verify.

b) init method : is (initialization) a special method that gets invoked when an object is instantiated..

__init__ (two underscore characters followed by init & then two more underscore)

```
# inside class Time:
    def __init__ (self, hour=0, minute=0, second=0):
        self.hour = hour
        self.minute = minute.
        self.second = second
```

__init__ have same names as the attributes:.

```
self.hour = hour.
```

Stores the value of parameter hour as attribute self.

Time with no arguments, default values.

```
>>> time = Time()
>>> time.print_time()
00:00:00
```

If one argument, it overrides hour.

```
>>> time = Time(9)
>>> time.print_time()
09:00:00
```

If Two argument, it override hour & minute

```
>>> time = Time(9,45)
>>> time.print_time()
09:45:00
```

__str__ method is a special method like __init__ return a string representation of an object.

Str method for Time objects;

```
def __str__(self):
    return '%2d:%2d:%2d' % (self.hour, self.minute,
                            self.second)
```

It invoke str method

```
>>> time = Time(9,45)
>>> print(time)
09:45:00
```

__init__ makes it easier to instantiate objects & __str__ is useful for debugging.

Prof. Ekata.S.

HOD
CSE (AI & ML)
KLS Vishwanathrao Deshpande
Institute of Technology, Haliyal