As Per New VTU Syllabus w.e.f 2018-19
Choice Based Credit System(CBCS)

SUNSTAR

SUNSTAR EXAM SCANNER

APPLICATION DEVELOPMENT USING PYTHON

(V SEM.B.E. CSE/ISE)

SYLLABUS

APPLICATION DEVELOPMENT USING PYTHON

[AS PER CHOICE BASED CREDIT SYSTEM (CBCS) SCHEME]
(EFFECTIVE FROM THE ACADEMIC YEAR 2018 - 2019)

Subject Code	18CS55	IA Marks	40
Number of Lecture Hours/Week	03	Exam Marks	60
Total Number of Lecture Hours	40	Exam Hours	03

MODULE 1

Python Basics, Entering Expressions into the Interactive Shell, The Integer, Floating-Point, and String Data Types, String Concatenation and Replication, Storing Values in Variables, Your First Program, Dissecting Your Program. **Flow control,** Boolean Values, Comparison Operators, Boolean Operators,Mixing Boolean and Comparison Operators, Elements of Flow Control, Program Execution, Flow Control Statements, Importing Modules,Ending a Program Early with sys.exit(). **Functions,** def Statements with Parameters, Return Values and return Statements,The None Value, Keyword Arguments and print(), Local and Global Scope, The global Statement, Exception Handling, A Short Program: Guess the Number

MODULE 2

Lists, The List Data Type, Working with Lists, Augmented Assignment Operators, Methods, Example Program: Magic 8 Ball with a List, List-like Types: Strings and Tuples,References, **Dictionaries and Structuring Data,** The Dictionary Data Type, Pretty Printing, Using Data Structures to Model Real-World Things, **Manipulating Strings,** Working with Strings, Useful String Methods, Project: Password Locker, Project: Adding Bullets to Wiki Markup

MODULE 3

Pattern Matching with Regular Expressions, Finding Patterns of Text Without Regular Expressions, Finding Patterns of Text with Regular Expressions,More Pattern Matching with Regular Expressions, Greedy and Nongreedy Matching, The findall() Method, Character Classes, Making Your Own Character Classes, The Caret and Dollar Sign Characters, The Wildcard Character, Review of Regex Symbols, Case-Insensitive Matching, Substituting Strings with the sub() Method, Managing Complex Regexes, Combining re .IGNORECASE, re .DOTALL, and re .VERBOSE, Project: Phone Number and Email Address Extractor, **Reading and Writing Files,** Files and File Paths, The os.path Module, The FileReading/Writing Process, Saving Variables with the shelve Module,Saving Variables with the pprint.pformat() Function, Project: Generating Random Quiz Files, Project: Multiclipboard, **Organizing Files,** The shutil Module, Walking a Directory Tree, Compressing Files with the zipfile Module, Project: Renaming Files with American-Style Dates to European-Style Dates,Project: Backing Up a Folder into a ZIP File, **Debugging,** Raising Exceptions, Getting the Traceback as a String, Assertions, Logging, IDLE's Debugger.

MODULE 4

Classes and objects, Programmer-defined types, Attributes, Rectangles, Instances as return values, Objects are mutable, Copying, Classes and functions, Time, Pure functions, Modifiers, Prototyping versus planning, Classes and methods, Object-oriented features, Printing objects, Another example, A more complicated example,The init method, The __ str__ method, Operator overloading, Type-based dispatch, Polymorphism, Interface and implementation,Inheritance, Card objects, Class attributes, Comparing cards, Decks, Printing the deck, Add, remove, shuffle and sort, Inheritance, Class diagrams, Data encapsulation

MODULE 5

Web Scraping, Project: MAPIT.PY with the webbrowser Module, Downloading Files from the Web with the requests Module, Saving Downloaded Files to the Hard Drive, HTML, Parsing HTML with the BeautifulSoup Module, Project: "I'm Feeling Lucky" Google Search,Project: Downloading All XKCD Comics, Controlling the Browser with the selenium Module, **Working with Excel Spreadsheets,** Excel Documents, Installing the openpyxl Module, Reading Excel Documents, Project: Reading Data from a Spreadsheet, Writing Excel Documents, Project: Updating a Spreadsheet, Setting the Font Style of Cells, Font Objects, Formulas, Adjusting Rows and Columns, Charts, **Working with PDF and Word Documents,** PDF Documents, Project: Combining Select Pages from Many PDFs, Word Documents, **Working with CSV files and JSON data,** The csv Module, Project: Removing the Header from CSV Files, JSON and APIs, The json Module, Project: Fetching Current Weather Data

Fifth Semester B.E. Degree Examination
CBCS - Model Question Paper - 1
APPLICATION DEVELOPMENT USING PYTHON

Time: 3 hrs. Max. Marks: 100
Note : Answer any FIVE full questions, selecting ONE full question from each module.

Module - 1

1. a. **Explain with example in python how to enter Expressions into the Interactive Shell and how to write programs in file editor with program.** **(08 Marks)**

Ans. →**Enter Expressions into the Interactive Shell** On Windows, open the Start menu, select **All Programs → Python 3.3**, and then select **IDLE (Python GUI)**.

```
Python 3.3.2 (v3.3.2:d047928ae3f6, May 16 2013, 00:06:53) [MSC v.1600 64 bit
(AMD64)] on win32 Type "copyright", "credits" or "license()" for more information.
>>> 2 + 2
4
>>> 2 + 3 * 6
20
>>> (2 + 3) * 6
30
>>> 48565873 * 578453
28093077826734
>>> 2 ** 8
256
>>> 23/7
3.2857142857142956
>>> 23 // 7
3
>>> 23 % 1
2
>>> 2          +          2
4
>>> (5 – 1) * ((7 + 1) / (3 – 1))
16.0
```

→To open the file editor in IDLE, select File→New Window. From the menu at the top of the file editor window, select File→Save As. In the Save As window, enter hello.py in the File Name field and then click Save. Select Run→Run Module or just press the F5 key.

```
# This program says hello and asks for my name.
print('Hello world!')
print('What is your name?') # ask for their name
myName = input()
print('It is good to meet you, ' + myName)
print('The length of your name is:')
print(len(myName)) print('What is your age?') # ask for their age myAge = input()
print('You will be ' + str(int(myAge) + 1) + ' in a year.')
```

b. Explain the following input(),len(),str() functions with example. **(06 Marks)**

Ans. → The **input()** function waits for the user to type some text on the keyboard and press enter.

Example :

```
myName = input()
print('It is good to meet you, ' + myName)
```

→ we can pass the **len()** function a string value (or a variable containing a string), and the function evaluates to the integer value of the number of characters in that string.

Example :

```
>>> len('hello')
    5
>>> len('My very energetic monster just scarfed nachos.')
    46
>>> len('')
    0
```

→The **str()** function can be passed an integer value and will evaluate to a string value version of it.

Example :

```
>>> str(29)
    '29'
>>> print('I am ' + str(29) + ' years old.')
    I am 29 years old.
```

Because str(29) evaluates to '29', the expression 'I am ' + str(29) + ' years old.' evaluates to 'I am ' + '29' + ' years old.', which in turn evaluates to 'I am 29 years old.'. This is the value that is passed to the print() function.

c. Write a python program to enter the name from keyboard show the length of the name entered give output. **(06 Marks)**

Ans.

```
# This program says hello and asks for my name.
print('Hello world!')
```



```
print('What is your name?') # ask for their name
myName = input()
print('It is good to meet you, ' + myName)
print('The length of your name is:')
print(len(myName)) print('What is your age?') # ask for their age myAge = input()
print
('You will be ' + str(int(myAge) + 1) + ' in a year.')
```

Output:

```
Hello world!
What is your name?
Al
It is good to meet you, Al
The length of your name is:
2
What is your age?
4
You will be 5 in a year.
>>>
```

OR

2. a. Explain flow control statements 'if statement 'and 'Else statement' with steps, python code and flow chart. **(08 Marks)**

Ans. 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. In plain English, an if statement could be read as, "If this condition is true, execute the code in the clause." In Python, an if statement consists of the following:

→Steps:

- 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)

→ code

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

→ flowchart: flow control statements end with a colon and are followed by a new block of code (the clause). This if statement's clause is the block with print ('Hi, Alice.').

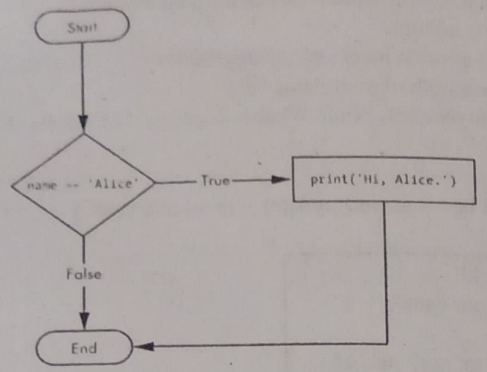


Fig Flow chart of if statement

→Else statement: 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. In plain English, an else statement could be read as, "If this condition is true, execute this code. Or else, execute that code." An else statement doesn't have a condition, and in code, an else statement always consists of the following:

STEPS:

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

CODE:

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

FLOWCHART:

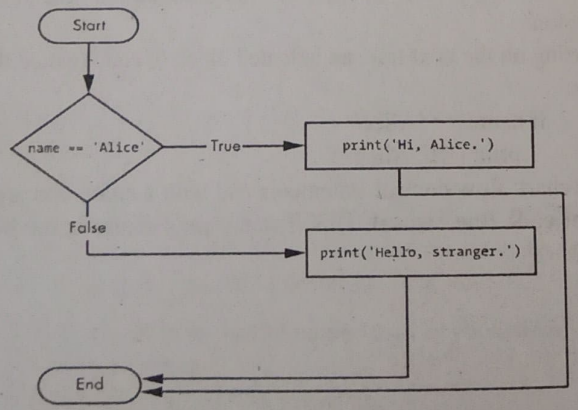


Fig Flow chart of else statement

b. Explain for Loops and the range () Function with steps, python code and flow chart. **(06 Marks)**

Ans. 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)

Let's create a new program called fiveTimes.py to help you see a for loop in action.

Code:

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

The code in the for loop's clause is run five times. The first time it is run, the variable i is set to 0. The print() call in the clause will print Jimmy Five Times (0). After Python finishes an iteration through all the code inside the for loop's clause, the execution goes back to the top of the loop, and the for statement increments i by one. This is why range(5) results in five iterations through the clause, with i being set to 0, then 1, then 2, then 3, and then 4

flowchart:

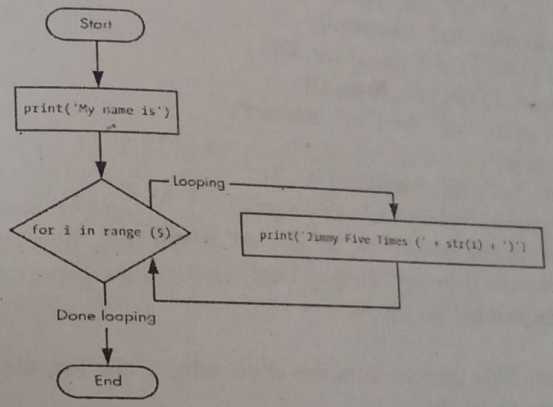


Fig Flow chart of Loops and range function

c. Explain with python code and output the def Statements with Parameters. **(06 Marks)**

Ans. We can also define our own functions that accept arguments.

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

When you run this program, the output looks like this:

Hello Alice
Hello Bob

The definition of the hello() function in this program has a parameter called name (1). A parameter is a variable that an argument is stored in when a function is called. The first time the hello ( ) function is called, it's with the argument 'Alice' (3). The program execution enters the function, and the variable name is automatically set to 'Alice', which is what gets printed by the print() statement (2).

Module-2

3. a. Illustrate list in python. Bacon contains the list [3.14, 'cat', 11, 'cat', True]. What does bacon.index('cat') evaluate to? Explain with output. (06 Marks)

Ans. LIST:

A list is a value that contains multiple values in an ordered sequence.
The term list value refers to the list itself , not the values inside the list value.
a list begins with an opening square bracket and ends with a closing square bracket, [ ].
Values inside the list are also called items. Items are separated with commas.

EXAMPLES:

```
>>> [1, 2, 3]
[1, 2, 3]
>>> ['cat', 'bat', 'rat', 'elephant']
['cat', 'bat', 'rat', 'elephant']
>>> ['hello', 3.1415, True, None, 42]
['hello', 3.1415, True, None, 42]
>>> spam = ['cat', 'bat', 'rat', 'elephant']
>>> spam
['cat', 'bat', 'rat', 'elephant']
```

On executing the following code we get:-

Bacon contains the list [3.14, 'cat', 11, 'cat', True]

```
>>> Bacon = [3.14, 'cat', 11, 'cat', True]
>>>Bacon.index('cat')
1
```

Reason: When there are duplicates of the value in the list, the index of its first appearance is returned

Therefore Bacon.index('cat') we got the value as 1 and not 3.

b. What is the difference between lists and tuples? (06 Marks)

Ans.

Sl.No.	LIST	TUPLE
1	Lists are mutable	Tuple are immutable
2	Implication of iterations is Time-consuming	Implication of iterations is comparatively Faster
3	The list is better for performing operations, such as insertion and deletion.	Tuple data type is appropriate for accessing the elements
4	Lists consume more memory	Tuple consume less memory as compared to the list
5	Lists have several built-in methods	Tuple does no have must built-in methods.
6	The unexpected changes and errors are more likely to occur	In tuple, it is hard to take place.
7.	EXAMPLE: List_data = ['a', 'b', 'c', 'd', 'e']	EXAMPLE: Tuples_data = ('a', 'b', 'c', 'd', 'e')

c. What module and function can be used to "pretty print" dictionary values? Write a python program for pretty character count in message. (08 Marks)

Ans. The module used in "pretty print" dictionary values is **pprint**

The functions used in "pretty print" dictionary values are **pprint() and pformat()**

This is helpful when you want a cleaner display of the items in a dictionary than what print() provides.

EXAMPLE:

```
import pprint
message = 'It was a bright cold day in April, and the clocks were striking thirteen.'
count = {}
for character in message:
    count.setdefault(character, 0)
    count[character] = count[character] + 1
pprint.pprint(count)
```

OUTPUT:

```
{' ': 13,
 ',': 1,
 '.': 1,
 'A': 1,
 'I': 1,
 'a': 4,
 'b': 1,
 'c': 3,
 'd': 3,
 'e': 5,
```

```
'g': 2,
'h': 3,
'i': 6,
'k': 2,
'l': 3,
'n': 4,
'o': 2,
'p': 1,
'r': 5,
's': 3,
't': 6,
'w': 2,
'y': 1}
```

OR

4. a. Write a python program to design tic-tac-toe board for following given below using dictionary and data structure show the output. (10 Marks)

i. Print the board at the start of each new turn,

ii. Get the active player's move,

iii. Updates the game board.

iv. then swaps the active player

Ans.

```
theBoard = {'top-L': ' ', 'top-M': ' ', 'top-R': ' ', 'mid-L': ' ', 'mid-M': '
', 'mid-R': ' ', 'low-L': ' ', 'low-M': ' ', 'low-R': ' '}
def printBoard(board):
  print(board['top-L'] + '|' + board['top-M'] + '|' + board['top-R'])
  print('-+-+-')
  print(board['mid-L'] + '|' + board['mid-M'] + '|' + board['mid-R'])
  print('-+-+-')
  print(board['low-L'] + '|' + board['low-M'] + '|' + board['low-R'])
turn = 'X'
for i in range(9):
  printBoard(theBoard)                    #prints at start of new turn
  print('Turn for ' + turn + '. Move on which space?')
  move = input()                          #player's active move
  theBoard[move] = turn                   #updates game board
  if turn == 'X':                         #swaps the active
        turn = 'O'
  else:
        turn = 'X'
printBoard(theBoard)
```



```
output:
  |   |
--+---+--
  |   |
--+---+--
  |   |
Turn for X. Move on which space?
mid-M
  |   |
--+---+--
  | X |
--+---+--
  |   |
Turn for 0. Move on which space?
low-L
  |   |
--+---+--
  | X |
--+---+--
 0|   |
--snip--
 0| 0| X
--+---+--
X| X| 0
--+---+--
 0|   | X
Tunr for X. Move on which space?
low-M
 0| 0| X
--+---+--
X| X| 0
--+---+--
 0| X| X
```

b. Illustrate with python code Copying and Pasting Strings with the pyperclip Module. (06 Marks)

Ans. The pyperclip module has copy() and paste() functions that can send text to and receive text from your computer's clipboard.

Sending the output of your program to the clipboard will make it easy to paste it to an email, word processor, or some other software.

Pyperclip does not come default with Python.

example:

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

If something outside of your program changes the clipboard contents, the paste() function will return it.

c. Indexing and Slicing Strings in python with example. **(06 Marks)**

Ans. Strings use indexes and slices the same way lists do.

By slicing and storing the resulting substring in another variable, you can have both the whole string and the substring handy for quick, easy access.

slicing a string does not modify the original string.

If you specify an index, you'll get the character at that position in the string. If you specify a range from one index to another, the starting index is included and the ending index is not.

You can capture a slice from one variable in a separate variable

for example, You can think of the string 'Hello world!' as a list and each character in the string as an item with a corresponding index.

```
'H e l l o   w o r l d !'
 0 1 2 3 4 5 6 7 8 9 10 11
```

(The space and exclamation point are included in the character count, so 'Hello world!' is 12 characters long, from H at index 0 to ! at index 11.)

example:

1.
```
>>> spam = 'Hello world!'
>>> spam[0]
'H'
>>> spam[4]
'o'
>>> spam[-1]
'!'
>>> spam[0:5]
'Hello'
```

Manipulating Strings 127

```
>>> spam[:5]
'Hello'
>>> spam[6:]
'world!'
```

2.
```
>>> spam = 'Hello world!'
>>> fizz = spam[0:5]
>>>fizz
  'Hello'
```


Module - 3

5. a. What is the function that creates Regex objects? Give the Review of Regular Expression Matching with suitable python code explain Grouping with Parentheses **(08 Marks)**

Ans. the function that creates Regex objects

- All the regex functions in Python are in the re module. Enter the following into the interactive shell to import this module:

```
>>> import re
```

- Review of Regular Expression Matching While there are several steps to using regular expressions in Python, each step is fairly simple.

1. Import the regex module with import re.
2. Create a Regex object with the re.compile() function. (Remember to use a raw string.)
3. Pass the string you want to search into the Regex object's search() method. This returns a Match object.
4. Call the Match object's group() method to return a string of the actual matched text.

- Grouping with Parentheses

Adding parentheses will create groups in the regex: (\d\d\d)-(\d\d\d-\d\d\d\d). Then you can use the group() match object method to grab the matching text from just one group

```
>>> phoneNumRegex = re.compile(r'(\d\d\d)-(\d\d\d-\d\d\d\d)')
>>> mo = phoneNumRegex.search('Hy number is 415-555-4242.')
>>> mo.group(l)
'415'
>>> mo.group(2)
'555-4242'
>>> mo.group(o)
'415-555-4242'
>>> mo.group()
'415-555-4242'
```

b. Explain with python code handling Absolute vs. Relative Paths. **(06 Marks)**

Ans. Handling Absolute and Relative Paths The os.path module provides functions for returning the absolute path of a relative path and for checking whether a given path is an absolute path.

- Calling os.path.abspath(path) will return a string of the absolute path of the argument. This is an easy way to convert a relative path into an absolute one.
- Calling os.path.isabs(path) will return True if the argument is an absolute path and False if it is a relative path.
- Calling os.path.relpath(path, start) will return a string of a relative path from the start path to path. If start is not provided, the current working directory is used as the start path.

Code:

```
>>> os.path.abspath('.')
'C:\\Python34'
>>> os.path.abspath(' .\\Scripts')
'C:\\Python34\\Scripts'
>>> os.path.isabs('.')
False
>>> os.path.isabs(os.path.abspath('.'))
True
```

c. What are the three "mode" arguments that can be passed to the open () function? (06 Marks)

Ans. The three "mode" arguments that can be passed to the open () function open the file in "reading plaintext" mode, or read mode. 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 mode by passing the string value 'r' as a second argument to open(). So open('/Users/asweigart/ hello.txt', 'r').

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

```
>>> 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()
>>> haconFile.close()
>>> print (content)
output :
Hello world!
Bacon is not a vegetable.
```


OR

6. a. Illustrate with python code Reading ZIP Files and Extracting from ZIP Files. (10 Marks)

Ans. **Reading ZIP Files**

To read the contents of a ZIP file, first you must create a ZipFile object (note the capital letters Z and F). ZipFile objects are conceptually similar to the File objects you saw returned by the open() function in the previous chapter: They are values through which the program interacts with the file. To create a ZipFile object, call the zipfile.ZipFile() function, passing it a string of the .zip file's filename. Note that zipfile is the name of the Python module, and ZipFile() is the name of the function.

```
>>> import zip-file, os
>>>  os.chdir{'C:\\')  # move to the folder with example.zip
>>> examplcZip = zip-file.lipFile('example.zip')
>>>  exampleZip.namelist()
['spam.txt', 'cats/', 'cats/catnames.txt', 'cats/zophie.jpg']
>>> spamInfo = exampleZip.getinfo('spam.txt')
>>> spamInfo.filesize 13908
>>> spamInfo.compress_size 3828
>>> 'Compressed file is %sx smaller!' %  (round(spamInfo.file_size / spamInfo.
compress_siie, 2))
'Conpressed file is 3.63* smaller!'
>>> exampleZip.dose()
```

Extracting from ZIP Files

The extractall() method for ZipFile objects extracts all the files and folders from a ZIP file into the current working directory.

```
>>> import zipfile, os
>>> os+chdir('C:\\')      # move to the folder with example*zip
>>>  exampleZip = zipfile.ZipFile("example.zip')
>>> exdrnpleZiptextractdllO
>>> exampleZip*close( )
```

b. Explain with example Raising Exceptions using python. (10 Marks)

Ans. Exceptions are raised with a raise statement. In code, a raise statement consists of the following:

- The raise keyword
- A call to the Exception() function
- A string with a helpful error message passed to the Exception() function

```
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)
```

```
for i in range(height - 2):
    print(symbol + (' ' * (width - 2)) + symbol)
    print(symbol * width)
for sym, w, h in (('*', 4, 4), ('0', 20, 5), (V, 1, 3), {'IV,  3, 3)): try:.
    boxPrint(symj w, h) except Exception as err:
    print('An exception happened: ' + str(err))
```

Output:

```
* * * *
*     *
*     *
* * * *
00000000000000000000
0                    0
0                    0
0                    0
00000000000000000000
```

An exception happened: Width must be greater than 2,
An exception happened: Symbol must be a single character string.

Module-4

7. a. Define class and object? Explain programmer defined types with python code. (08 Marks)

Ans. A user-defined type is also called a **class**. A class definition looks like this:

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

This header indicates that the new class is a Point, which is a kind of object, which is a built-in type. The body is a docstring that explains what the class is for.

Defining a class named Point creates a class object.

```
>>> print Point
```

output :

```
<class '__main__.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()
>>> print blank
```

output :

```
<__main__.Point instance 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).

b. Explain pure function with python code. (06 Marks)

Ans. 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)
```

The output of this program is 12:49:30.

c. Write a python program that takes a birthday as input and prints the user's age.

Ans. (06 Marks)

```
from datetime import datetime
def main()
    today = datetime.today()
    s = input('Enter your birthday in mm/dd/yyyy format: ')
    bday = datetime.strptime(s, '%m/%d/%Y')
    next_bday = bday.replace(year=today.year)
    if next_bday < today:
        next_bday = next_bday.replace(year=today.year+1)
    until_next_bday = next_bday - today
    print("Your current age:")
    last_bday = next_bday.replace(year=next_bday.year-1)
    age = last_bday.year - bday.year
    print(age)
if __name__ == '__main__':
    main()
```

Output:

```
Enter your birthday in mm/dd/yyyy format:
 1/17/2000
Your current age:
20
```

OR

8. a. **Explain the working of init method with suitable python code.** **(08 Marks)**

Ans. The init method (short for "initialization") is a special method that gets invoked when an object is instantiated. Its full name is __init__ (two underscore characters, followed by init, and then two more underscores). An init method for the Time class might look like this:

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

It is common for the parameters of __init__ to have the same names as the attributes. The statement

```
    self.hour = hour
```

stores the value of the parameter hour as an attribute of self.

The parameters are optional, so if you call Time with no arguments, you get the default values.

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

If you provide one argument, it overrides hour:

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

If you provide two arguments, they override hour and minute.

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

And if you provide three arguments, they override all three default values.

b. **Explain the concept of modifier with python code.** **(06 Marks)**

Ans. There are times when it is useful for a function to modify one or more of the objects it gets as arguments. Usually, the caller keeps a reference to the objects it passes, so any changes the function makes are visible to the caller. Functions that work this way are called **modifiers**.

increment, which adds a given number of seconds to a Time object, would be written most naturally as a modifier. A rough draft of the function looks like this:

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

  if time.seconds >= 60:
    time.seconds = time.seconds - 60
    time.minutes = time.minutes + 1
```



```
  if time.minutes >= 60:
    time.minutes = time.minutes - 60
    time.hours = time.hours + 1
```

The first line performs the basic operation; the remainder deals with the special cases.

c. **Explain the _lt_ method for comparing card with suitable python code.** **(06 Marks)**

Ans. For built-in types, there are relational operators (, ==, etc.) that compare values and determine when one is greater than, less than, or equal to another. For programmer-defined types, we can override the behavior of the built-in operators by providing a method named __lt__, which stands for "less than". __lt__ takes two parameters, self and other, and returns True if self is strictly less than other.

```
# inside class Card:
def __lt__(self, other):
 # check the suits
 if self.suit < other.suit: return True
 if self.suit > other.suit: return False
 # suits are the same... check ranks
 return self.rank < other.rank
```

You can write this more concisely using tuple comparison:

```
# inside class Card:
def __lt__(self, other):
 t1 = self.suit, self.rank
 t2 = other.suit, other.rank
 return t1 < t2.
```

Module-5

9. a. **Explain with suitable python code creating a Beautifulsoap object from HTML.** **(10 Marks)**

Ans. Beautiful Soup is a module for extracting information from an HTML page (and is much better for this purpose than regular expressions). The BeautifulSoup module's name is bs4 (for Beautiful Soup, version 4). To install it, you will need to run pip install beautifulsoup4 from the command line. (Check out Appendix A for instructions on installing third-party modules.) While beautifulsoup4 is the name used for installation, to import Beautiful Soup you run import bs4.

```
<!-- This is the example.html example file. -->
<html><headxtitle>The Website Title</titlex/head>
<body>
<p>Download my <strong>Python</strong> book from <a href="http://
inventwithpython.cam">my website</a>.</p>
<p class="slogan">Learn Python the easy way!</p>
<p>By <span id="author">Al Sweigart</spanx/p>
</bodyx/html>
```

Creating a BeautifulSoup Object from HTML The bs4.BeautifulSoup() function

needs to be called with a string containing the HTML it will parse. The bs4. BeautifulSoup() function returns is a BeautifulSoup object. Enter the following into the interactive shell while your computer is connected to the Internet:

```
>>> import requests, bs4
>>> res = requests.get('http://nostarch.com')
>>> res.raiseforstatusQ
>>> noStarchSoup = bs4.BeautifulSoup(res.text)
>>> type(noStarchSoup)
<class 'bs4*Beauti-fulSoup'>
```

b. Write a python program for Getting Rows and Columns from the Sheets with output. **(10 Marks)**

Ans. Slice Worksheet objects to get all the Cell objects in a row, column, or rectangular area of the spreadsheet. Then you can loop over all the cells in the slice.

```
>>> import openpyxl
>>> w b = openpyxl.load _workcbook('examp1e.xlsx')
>>> sheet = wb.getsheetbynameCsheetl")
>>> tuple(sheet[ 'Al':'C3'])
((<Cell Sheetl.Al), <Cell Shertl.Bl>, (Cell Sheetl.Cl>), (<Cell Sheetl.A2>,
<Cell 5heetl.B2>, (Cell Sheetl.C2>), (<Cell Sheetl.A3i, <Cell Sheetl.B3>,
<Cell Sheetl.C3>))
>>> foi lOHOftellObjects in 5heet['Al':'C3']:
for cellObj in rowOfCellObjects:
piint(cellObj.coordinate, cellObj.value)
printC— END OF ROM ---')
```

Output:

```
Al 2015-04-05 13:34:02
Bl Apples
Cl 73
--- END OF ROW ---
A2 2015-04-05 03:41:23
B2 Cherries
C2. 85
--- END OF ROW ---
A3 2015-04-06 12:46:51
B3 Pears
C3 14
--- END OF ROW ---
```

To print the values of each cell in the area, we use two for loops. The outer for loop goes over each row in the slice u. Then, for each row, the nested for loop goes through each cell in that row v.

OR

10. a. Illustrate with python program copying and rotating the pages from PDF document. **(10 Marks)**

Ans. We can use PyPDF2 to copy pages from one PDF document to another. This allows you to combine multiple PDF files, cut unwanted pages, or reorder pages.

```
>>> import PyPDF2
>>> pdflFile = openf meetingminutes.pdf', 'rb')
>>> pdf2File = open{'meetingminutes2.pdf1, 'rb*)
>>> pdflReader = PyPDF2.pdfFileReader(pdflFile)
>>> pdf2Reader = PyPDF2.PdfFileReader(pdf2File)
>>> pdfWriter = PyPDF2.PdfFileWriter{)
>>> for pageNum in range{pdflReader.numPages):
page-0bj = pdflReader.get Page(pageNum)
pdfWriter.addPage(page0bj)
>>> fox pageNum in iange(pdf2Reader.numPage£):
page0bj = pdf2Reader.getPage{pageNum)
pdfWriter.addPage(p ageObj)
>>> pdfOutputFile = openCcombinedminutes.pdf', 'wb')
>>> pdfWriter.write{pdfOutputFile)
>>> pdfOutputFile.close()
>>> pdflFile.close()
>>> pdf2File.close()
```

Open both PDF files in read binary mode and store the two resulting File objects in pdf1File and pdf2File. Call PyPDF2.PdfFileReader() and pass it pdf1File to get a PdfFileReader object for meetingminutes.pdf u. Call it again and pass it pdf2File to get a PdfFileReader object for meetingminutes2.pdf v. Then create a new PdfFileWriter object, which represents a blank PDF document w. Next, copy all the pages from the two source PDFs and add them to the PdfFileWriter object. Get the Page object by calling getPage() on a PdfFileReader object x. Then pass that Page object to your PdfFileWriter's addPage() method y. These steps are done first for pdf1Reader and then again for pdf2Reader. When you're done copying pages, write a new PDF called combinedminutes.pdf by passing a File object to the PdfFileWriter's write() method z.

b. What is CSV. Explain CSV module with suitable python code. **(10 Marks)**

Ans. CSV stands for "comma-separated values," and CSV files are simplified spreadsheets stored as plaintext files. Python's csv module makes it easy to parse CSV files.

The csv Module

Each line in a CSV file represents a row in the spreadsheet, and commas separate the cells in the row

CSV file looks like this:

```
4/5/2015 13:34,Apples,73
4/5/2015 3:41,Cherries,85
```

4/6/2015 12:46,Pears,14
4/8/2015 8:59,Oranges,52
4/10/2015 2:07,Apples,152
4/10/2015 18:10,Bananas,23
4/10/2015 2:40,Strawberries,98

CSV files are simple, lacking many of the features of an Excel spreadsheet. For example, CSV files

- Don't have types for their values-everything is a string
- Don't have settings for font size or color
- Don't have multiple worksheets
- Can't specify cell widths and heights
- Can't have merged cells
- Can't have images or charts embedded in them

The advantage of CSV files is simplicity. CSV files are widely supported by many types of programs, can be viewed in text editors (including IDLE's file editor), and are a straightforward way to represent spreadsheet data. The CSV format is exactly as advertised: It's just a text file of comma separated values.

Fifth Semester B.E. Degree Examination
CBCS - Model Question Paper - 2
APPLICATION DEVELOPMENT USING PYTHON

Time: 3 hrs. Max. Marks: 100

Note : *Answer any FIVE full questions, selecting ONE full question from each module.*

Module - 1

1. a. **What is Python? What is Interactive Shell explain the steps required to open Interactive Shell on Windows, OS X, Ubuntu. give example.** **(08 Marks)**

Ans. Python refers to the Python programming language (with syntax rules for writing what is considered valid Python code) and the Python interpreter software that reads source code (written in the Python language) and performs its instructions.

On Windows, open the Start menu, select All Programs →Python 3.3, and then select IDLE (Python GUI). On OS X, select Applications →MacPython 3.3 → IDLE. On Ubuntu, open a new Terminal window and enter idle3.

Example:

```
>>> 2 + 3 * 6
20
>>> (2 + 3) * 6
30
>>> 48565873 * 578453
28093077826734
>>> 2 ** 8
256
>>> 23/7
3.2857142857142956
>>> 23 // 7
3
>>> 23 % 1
2
>>> 2          +          2
4
>>> (5 – 1) * ((7 + 1) / (3 – 1))
16.0
```

b. **Illustrate Storing Values in Variables with suitable python code.** **(06 Marks)**

Ans. A variable is like a box in the computer's memory where you can store a single value. If you want to use the result of an evaluated expression later in your program, you can save it inside a variable.

Assignment Statements

Example:

```
>>> spam = 40
>>> spam
40
>>> eggs = 2
>>> spam + eggs
42
>>> spam + eggs + spam
82
>>> spam = spam + 2
>>> spam
42
```

Explanation: A variable is initialized (or created) the first time a value is stored in it 1. After that, you can use it in expressions with other variables and values 2. When a variable is assigned a new value 3, the old value is forgotten, which is why spam evaluated to 42 instead of 40 at the end of the example.

c. Explain and compare the flow control for if and while statement. (06 Marks)

Ans. if Statements

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.

- 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)

while Loop Statements

You can make a 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)

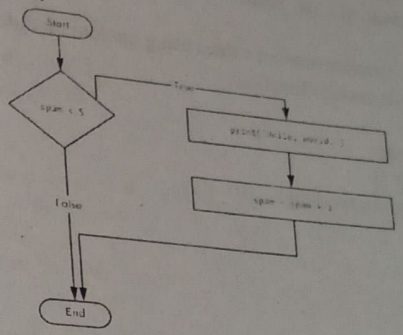
```
spam = 0
if spam < 5:
  print('Hello, world.')
spam = spam + 1
```

Here is the code with a while statement:

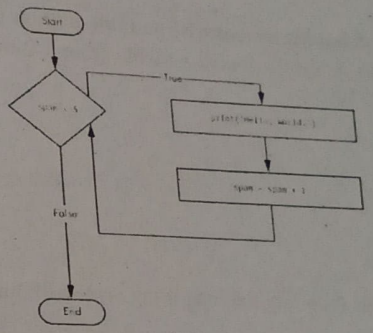
```
spam = 0
while spam < 5:
      print('Hello, world.')
spam = spam + 1
```


These statements are similar—both if and while check the value of spam, and if it's less than five, they print a message. But when you run these two code snippets, something very different happens for each one. For the if statement, the output is simply "Hello, world.". But for the while statement, it's "Hello, world." repeated five times!

Flowchart: if statement



Flowchart: While statement:



Or

2. a. Explain the import module. write a python program to generate random numbers. (06 Marks)

Ans. Python also comes with a set of modules called the standard library. Each module is a Python program that contains a related group of functions that can be embedded in your programs. For example, the math module has mathematics related functions, the random module has random number–related functions, and so on. Before you can use the functions in a module, you must import the module with an import statement. In code, an import statement consists of the following:

- The import keyword
- The name of the module

- Optionally, more module names, as long as they are separated by commas Once you import a module, you can use all the cool functions of that module.

Here's an example of an import statement that imports four different modules:

```
import random, sys, os, math
```

Program to generate random values using random module, which will give us access to the random.ranint() function.

```
import random
for i in range(5):
    print(random.randint(1, 10))
```

output:

```
4
1
8
4
1
```

b. Explain the global Statement with suitable python code. **(08 Marks)**

Ans. If you need to modify a global variable from within a function, use the global statement

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

When you run this program, the final print() call will output this:

span

Because eggs is declared global at the top of spam () **(1)** when eggs is set to "spam" **(2)** This assignment is done to the globally scoped spam. No local spam variable is created.

There are four rules to tell whether a variable is in a local scope or global scope:

1. If a variable is being used in the global scope (that is, outside of all functions), then it is always a global variable.
2. If there is a global statement for that variable in a function, it is a global variable.
3. Otherwise, if the variable is used in an assignment statement in the function, it is a local variable.
4. But if the variable is not used in an assignment statement, it is a global variable.

Example:

```
def spam():
global eggs
eggs = 'spam' # this is the global
def bacon():
eggs - 'bacon' # this is a local
def ham():
print(eggs) # this is the global
eggs = 42 # this is the global
spam()
print(eggs)
```

In the spam() function, eggs is the global eggs variable, because there's is global statement for eggs at the beginning of the function **(1)** In bacon(), eggs is a local variable, because there's an assignment statement for it in that function **(2)** In ham() **(3)** eggs is the global variable, because there is no assignment statement or global statement for it in that function. If you run

c. Write a python program to handle the exception of divide-by-zero using try and except statements. And show the output for same. **(06 Marks)**

Ans.

```
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
```

Module-2

3. a. Illustrate with python code list, tuple, and dictionary data type. **(06 Marks)**

Ans. LIST:

- A list is a value that contains multiple values in an ordered sequence.
- The term list value refers to the list itself , not the values inside the list value.
- a list begins with an opening square bracket and ends with a closing square bracket, [ ].
- Values inside the list are also called items. Items are separated with commas.

- A list value is a **mutable data type**
- It can have values added, removed, or changed.

EXAMPLE:

```
>>> eggs = [1, 2, 3]
>>>eggs
[1, 2, 3]
>>> del eggs[2]
>>> del eggs[1]
>>> del eggs[0]
>>> eggs.append(4)
>>> eggs.append(5)
>>> eggs.append(6)
>>> eggs
[4, 5, 6]
```

TUPLES:

- The tuple data type is almost identical to the list data type, except in two ways:
- First, tuples are typed with parentheses, (and), instead of square brackets, [and]
- Secondly tuples, like strings, are immutable. Tuples cannot have their values modified, appended, or removed.

EXAMPLE:

```
>>> eggs = ('hello', 42, 0.5)
>>> eggs[0]
'hello'
>>> eggs[1:3]
(42, 0.5)
>>> len(eggs)
3
>>>  eggs[1] = 99
TypeError: 'tuple' object does not support item assignment
```

DICTIONARY:

- a dictionary is a collection of many values
- indexes for dictionaries can use many different data types
- Indexes for dictionaries are called keys
- a key with its associated value is called a key-value pair

EXAMPLE:

```
>>> myCat = {'size': 'fat', 'color': 'gray', 'disposition': 'loud'}
```

here dictionary name is myCat

dictionary's keys are 'size', 'color', and 'disposition'.

The values for these keys are 'fat', 'gray', and 'loud', respectively.

```
>>> myCat['size']
'fat'
>>> 'My cat has ' + myCat['color'] + ' fur.'
'My cat has gray fur.'
```


b. **What are the operators for list concatenation and list replication?** **(06 Marks)**

Ans. **Concatenation:**

The + operator can combine two lists to create a new list value in the same way it combines two strings into a new string value.

EXAMPLE:

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

>>> spam = [1, 2, 3]
>>> spam = spam + ['A', 'B', 'C']
>>> spam
[1, 2, 3, 'A', 'B', 'C']
```

Replication:

The * operator can also be used with a list and an integer value to replicate the list.

EXAMPLE:

```
>>> ['X', 'Y', 'Z'] * 3
['X', 'Y', 'Z', 'X', 'Y', 'Z', 'X', 'Y', 'Z']

>>>['abc'] * 3
['abc', 'abc', 'abc']
```

c. **Write a python program to store data about your friends' birthdays. use a dictionary with the names as keys and the birthdays as values with output.** **(08 Marks)**

Ans. **PROGRAM:**

```
birthdays = {'Alice': 'Apr 1', 'Bob': 'Dec 12', 'Carol': 'Mar 4'}

while True:
    print('Enter a name: (blank to quit)')
    name = input()
    if name == '':
        break
    if name in birthdays:
        print(birthdays[name] + ' is the birthday of ' + name)
    else:
        print('I do not have birthday information for ' + name)
        print('What is their birthday?')
        bday = input()
        birthdays[name] = bday
        print('Birthday database updated.')
```

OUTPUT:

```
Enter a name: (blank to quit)
Alice
Apr 1 is the birthday of Alice
Enter a name: (blank to quit)
Eve
I do not have birthday information for Eve
What is their birthday?
Dec 5
Birthday database updated.
Enter a name: (blank to quit)
Eve
Dec 5 is the birthday of Eve
Enter a name: (blank to quit)
```

OR

4. a. Write a python program to print the tabular data using center(),ljust(),rjust() methods. Show output **(08 Marks)**

Ans. The rjust() and ljust() string methods return a padded version of the string they are called on, with spaces inserted to justify the text.

The first argument to both methods is an integer length for the justified string.

example:

```
>>> 'Hello'.rjust(10)
'  Hello'
>>> 'Hello'.rjust(20)
'              Hello'
>>> 'Hello World'.rjust(20)
'         Hello World'
>>> 'Hello'.ljust(10)
'Hello     '
```

'Hello'.rjust(10) says that we want to right-justify 'Hello' in a string of total length 10. 'Hello' is five characters, so five spaces will be added to its left, giving us a string of 10 characters with 'Hello' justified right.

An optional second argument to rjust() and ljust() will specify a fill character other than a space character.

example:

```
>>> 'Hello'.rjust(20, '*')
'***************Hello'
>>> 'Hello'.ljust(20, '-')
'Hello---------------'
```

The center() string method works like ljust() and rjust() but centers the text rather than justifying it to the left or right.

example:

```
>>> 'Hello'.center(20)
'       Hello        '
>>> 'Hello'.center(20, '=')
'=======Hello========'
```

These methods are especially useful when you need to print tabular data that has the correct spacing.

```
#EXAMPLE PROGRAM
def printPicnic(itemsDict, leftWidth, rightWidth):
 print('PICNIC ITEMS'.center(leftWidth + rightWidth, '-'))
 for k, v in itemsDict.items():
 print(k.ljust(leftWidth, '.') + str(v).rjust(rightWidth))
134 Chapter 6
picnicItems = {'sandwiches': 4, 'apples': 12, 'cups': 4, 'cookies': 8000}
printPicnic(picnicItems, 12, 5)
printPicnic(picnicItems, 20, 6)
OUTPUT:
-------PICNIC ITEMS-------
sandwiches.. 4
apples..... 12
cups..... 4
cookies..... 8000
-------PICNIC ITEMS-------
sandwiches..... 4
apples..... 12
cups..... 4
cookies..... 8000
```

(When you run this program, the picnic items are displayed twice. The first time the left column is 12 characters wide, and the right column is 5 characters wide. The second time they are 20 and 6 characters wide, respectively.)

Using rjust(), ljust(), and center() lets you ensure that strings are neatly aligned, even if you aren't sure how many characters long your strings are.

b. What are escape characters? What do the \n and \t escape characters represent? **(06 Marks)**

Ans. An escape character lets you use characters that are otherwise impossible to put into a string. An escape character consists of a backslash (\) followed by the character you want to add to the string.

For example, the escape character for a single quote is \'. You can use this inside a string that begins and ends with single quotes.

Python knows that since the single quote in \' has a backslash, it is not a single quote meant to end the string value. The escape characters \' and \" let you put single quotes and double quotes inside your strings, respectively.

Escape Characters	
Escape character	Prints as
\t	Tab
\n	Newline (line break)

```
>>> print("Hello there!\tHow are you?\nI\'m doing fine.")
Hello there! How are you?
I'm doing fine.
```

c. What is X String Methods with example. **(06 Marks)**

Ans. Similar to islower() and isupper(), there are several string methods that have names beginning with the word is.

These methods return a Boolean value that describes the nature of the string.

The isX string methods are helpful when you need to validate user input.

Here are some common isX string methods:

- **isalpha()** - It returns True if the string consists only of letters and is not blank.

example:

```
>>> 'hello'.isalpha()
True
>>> 'hello123'.isalpha()
False
```

- **isalnum()** - It returns True if the string consists only of letters and numbers and is not blank.

example:

```
>>> 'hello123'.isalnum()
True
>>> 'hello'.isalnum()
True
```

- **isdecimal()** - Itreturns True if the string consists only of numeric characters and is not blank.

example:

```
>>> '123'.isdecimal()
True
```

- **isspace()** - It returns True if the string consists only of spaces, tabs, and newlines and is not blank.

example:

```
>>> ' '.isspace()
True
```

- **istitle()** - It returns True if the string consists only of words that begin with an uppercase letter followed by only lowercase letters.

example:

```
>>> 'This Is Title Case'.istitle()
True
>>> 'This Is Title Case 123'.istitle()
True
>>> 'This Is not Title Case'.istitle()
False
>>> 'This Is NOT Title Case Either'.istitle()
False
```

Module-3

5. a. Explain regex symbols with python code. **(06 Marks)**

Ans. • The ? matches zero or one of the preceding group.

```
>>> batRegex = re.compile(r'Bat(wo)?man')
>>> mo1 = batRegex.search('The Adventures of Batman')
>>> mo1.group()
'Batman'
>>> mo2 = batRegex.search('The Adventures of Batwoman')
>>> mo2.group()
'Batwoman'
```

• The * matches zero or more of the preceding group.

```
>>> batRegex = re.compile(r'Bat(wo)*man')
>>> mo1 = batRegex.search('The Adventures of Batman')
>>> mo1.group()
'Batman'
>>> mo2 = batRegex.search('The Adventures of Batwoman')
>>> mo2.group()
'Batwoman'
>>> mo3 = batRegex.search('The Adventures of Batwowowowoman')
>>> mo3.group()
'Batwowowowoman'
```

• The + matches one or more of the preceding group.

```
>>> batRegex = re.compile(r'Bat(wo)+man')
>>> mo1 = batRegex.search('The Adventures of Batwoman')
>>> mo1.group()
'Batwoman'
>>> mo2 = batRegex.search('The Adventures of Batwowowowoman')
>>> mo2.group()
'Batwowowowoman'
```

```
>>> mo3 = batRegex.search('The Adventures of Batman')
>>> mo3 == None
True
```

b. Write a python program to find phone numbers and email addresses on the clipboard. **(08 Marks)**

Ans.

```
#! python3
# phoneAndEmail.py - Finds phone numbers and email addresses on the clipboard.
import pyperclip, re
phoneRegex = re.compile(r'''(
--snip--
# Find matches in clipboard text.
text = str(pyperclip.paste())
matches = []
for groups in phoneRegex.findall(text):
 phoneNum = '-'.join([groups[1], groups[3], groups[5]])
 if groups[8] != '':
 phoneNum += ' x' + groups[8]
 matches.append(phoneNum)
  for groups in emailRegex.findall(text):
 matches.append(groups[0])
```

c. Explain with python code how to check Path Validity. **(06 Marks)**

Ans. Many Python functions will crash with an error if you supply them with a path that does not exist. The os.path module provides functions to check whether a given path exists and whether it is a file or folder.

- Calling os.path.exists(path) will return True if the file or folder referred to in the argument exists and will return False if it does not exist.
- Calling os.path.isfile(path) will return True if the path argument exists and is a file and will return False otherwise.
- Calling os.path.isdir(path) will return True if the path argument exists and is a folder and will return False otherwise

Example:

```
>>> os.path.exists('C:\\Windows')
True
>>> os.path.exists('C:\\some_made_up_folder')
False
>>> os.path.isdir('C:\\Windows\\System32')
True
>>> os.path.isfile('C:\\Windows\\System32')
False
>>> os.path.isdir('C:\\Windows\\System32\\calc.exe')
False
>>> os.path.isfile('C:\\Windows\\System32\\calc.exe')
True
```


OR

6. a. Illustrate with python code the Caret and Dollar Sign Characters **(08 Marks)**

Ans. the caret symbol (^) at the start of a regex to indicate that a match must occur at the beginning of the searched text. Likewise, you can put a dollar sign ($) at the end of the regex to indicate the string must end with this regex pattern. And you can use the ^ and $ together to indicate that the entire string must match the regex.

Example:

- For example, the r'^Hello' regular expression string matches strings that begin with 'Hello

```
>>> beginsWithHello = re.compile(r'^Hello')
>>> beginsWithHello.search('Hello world!')
<_sre.SRE_Match object; span=(0, 5), match='Hello'>
>>> beginsWithHello.search('He said hello.') == None
True
```

- The r'\d$' regular expression string matches strings that end with a numeric character from 0 to 9.

```
>>> endsWithNumber = re.compile(r'\d$')
>>> endsWithNumber.search('Your number is 42')
<_sre.SRE_Match object; span=(16, 17), match='2'>
>>> endsWithNumber.search('Your number is forty two.') == None
True
```

- The r'^\d+$' regular expression string matches strings that both begin and end with one or more numeric characters.

```
>>> wholeStringIsNum = re.compile(r'^\d+$')
>>> wholeStringIsNum.search('1234567890')
<_sre.SRE_Match object; span=(0, 10), match='1234567890'>
>>> wholeStringIsNum.search('12345xyz67890') == None
True
>>> wholeStringIsNum.search('12 34567890') == None
True
```

b. Illustrate with example the reading and writing the file using python code. **(06 Marks)**

Ans. To read the entire contents of a file as a string value, use the File object's read() method

example:

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

for the contents of a file as a single large string value, the read() method returns the string that is stored in the file. Alternatively, the readlines() method to get a list of

string values from the file, one string for each line of text. For example, create a file named sonnet29.txt in the same directory as hello.txt and write the following text in it:

sonnet29.txt

When, in disgrace with fortune and men's eyes, I all alone beweep my outcast state,
And trouble deaf heaven with my bootless cries, And look upon myself and curse my fate,

```
>>> sonnetFile = open('sonnet29.txt')
>>> sonnetFile.readlines()
```

Output:

[When, in disgrace with fortune and men's eyes,\n', ' I all alone beweep my outcast state,\n', And trouble deaf heaven with my bootless cries,\n', And look upon myself and curse my fate,']

c. Explain Safe Deletes with the send2trash Module in python. (06 Marks)

Ans. Since Python's built-in shutil.rmtree() function irreversibly deletes files and folders, it can be dangerous to use. A much better way to delete files and folders is with the third-party send2trash module. You can install this module by running pip install send2trash from a Terminal window.

Using send2trash is much safer than Python's regular delete functions, because it will send folders and files to your computer's trash or recycle bin instead of permanently deleting them. If a bug in your program deletes something with send2trash you didn't intend to delete, you can later restore it from the recycle bin.

Example:

```
>>> import send2trash
>>> baconFile = open('bacon.txt', 'a') # creates the file
>>> baconFile.write('Bacon is not a vegetable.')
25
>>> baconFile.close()
>>> send2trash.send2trash('bacon.txt')
```

Module-4

7. a. Define class and object? what are programmer defined type? Explain with example (10 Marks)

Ans. Class is a user-defined data type which binds data and functions together into single entity. Class is just a prototype (or a logical entity/blue print) which will not consume any memory. An object is an instance of a class and it has physical existence. One can create any number of objects for a class. A class can have a set of variables (also known as attributes, member variables) and member functions (also known as methods).

A class in Python can be created using a keyword class. Here, we are creating an empty class without any members by just using the keyword pass within it.


```
class Point:
-----------------------
print (Point)
-----------------------
```

The output would be –

<class '__main__.Point'>

The term __main__ indicates that the class Point is in the main scope of the current module. In other words, this class is at the top level while executing the program.

The process of creating a new object is called as instantiation and the object is instance of a class.

When we print an object, Python tells which class it belongs to and where it is stored in the memory.

print(p)

The output would be –

<__main__.Point object at 0x003C1BF0>

The output displays the address (in hexadecimal format) of the object in the memory. It is now clear that, the object occupies the physical space, whereas the class does not.

b. Explain init method with python code. Explain __str__ method with python code. (10 Marks)

Ans. • The term **init** indicates initialization. As the name suggests, this method is invoked automatically when the object of a class is created. Consider the example given here –

```
import math
class Point:
    def __init__(self,a,b):
        self.x=a
        self.y=b
    def dist(self,p2):
        d=math.sqrt((self.x-p2.x)**2 + (self.y-p2.y)**2)
        return d
    def __str__(self):
        return "(%d,%d)"%(self.x, self.y)
p1=Point(10,20) #__init__() is called automatically
p2=Point(4,5) #__init__() is called automatically
print("P1 is:",p1) #__str__() is called automatically
print("P2 is:",p2) #__str__() is called automatically
d=p1.dist(p2) #explicit call for dist()
print("The distance is:",d)
```

output is –

P1 is: (10,20)

P2 is: (4,5)

Distance is: 16.15549442140351.

- **__str__** is a special method, like __init__, that is supposed to return a string representation of an object.

For example, here is a str method for Time objects:

```
class Time:
"""Represents the time of day. attributes: hour, minute, second"""
    def __init__(self, hour=0, minute=0, second=0):
        self.hour = hour
        self.minute = minute
        self.second = second

    def __str__(self):
        return '%.2d:%.2d:%.2d' % (self.hour, self.minute, self.second)
print('----------')
time1 = Time()
print(time1)
print('----------')
time2 = Time(10, 20)
print(time2)
```

Output:

```
----------
00:00:00
----------
10:20:00
```

OR

8. a. Write a function called distance_between_points that takes two points as arguments and return the distance between them. **(10 Marks)**

Ans.

```
import math

class Point(object):
    """Represents a point in 2d space."""

point_one = Point()
point_two = Point()

point_one.x, point_one.y = 6.0, 1.0
point_two.x, point_two.y = 2.0, 6.0

def distance(p1, p2):
    """Returns the distance between two points in 2d space."""
    delta_x = p2.x - p1.x
```

```
print('----------')
start = Time(9, 45)
duration = Time(1, 35)
print(start + duration)
```

Output:

```
----------
11:20:00
```

Polymorphism

Type-based dispatch is useful when it is necessary, but (fortunately) it is not always necessary. Often you can avoid it by writing functions that work correctly for arguments with different types.

Many of the functions we wrote for strings also work for other sequence types. For example, 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
```

This function also works for lists, tuples, and even dictionaries, as long as the elements of s are hashable, so they can be used as keys in d.

```
>>> t = ['spam', 'egg', 'spam', 'spam', 'bacon', 'spam']
>>> histogram(t)
{'bacon': 1, 'egg': 1, 'spam': 4}
```

Functions that work with several types are called polymorphic. Polymorphism can facilitate code reuse. For example

```
>>> t1 = Time(7, 43)
>>> t2 = Time(7, 41)
>>> t3 = Time(7, 37)
>>> total = sum([t1, t2, t3])
>>> print(total)
23:01:00
```

Module-5

9. a. List the select() Method with example to retrieve a web page element from a BeautifulSoup object. **(10 Marks)**

Ans. The various selector patterns can be combined to make sophisticated matches. For example, soup.select('p #author') will match any element that has an id attribute of author, as long as it is also inside a

element. The select() method will return a list of Tag objects, which is how Beautiful Soup represents an HTML element. The list will contain one Tag object for every

```
>>> from openpyxl.styles import Font, Style
>>> wb = openpyxl.Workbook()
>>> sheet = wb.get_sheet_by_name('Sheet')
>>> italic24Font = Font(size=24, italic=True)
>>> styleObj = Style(font=italic24Font)
>>> sheet['A'].style/styleObj
>>> sheet['A1'] = 'Hello world!'
>>> wb.save('styled.xlsx')
```

OR

10. a. Write a python program for Overlaying Pages. **(10 Marks)**

Ans.

```
>>> import PyPDF2
>>> minutesFile = open('meetingminutes.pdf', 'rb')
>>> pdfReader = PyPDF2.PdfFileReader(minutesFile)
>>> minutesFirstPage = pdfReader.getPage(0)
>>> pdfWatermarkReader = PyPDF2.PdfFileReader(open('watermark.pdf', 'rb'))
>>> minutesFirstPage.mergePage(pdfWatermarkReader.getPage(0))
>>> pdfWriter = PyPDF2.PdfFileWriter()
>>> pdfWriter.addPage(minutesFirstPage)

>>> for pageNum in range(1, pdfReader.numPages):
        pageObj = pdfReader.getPage(pageNum)
        pdfWriter.addPage(pageObj)
>>> resultPdfFile = open('watermarkedCover.pdf', 'wb')
>>> pdfWriter.write(resultPdfFile)
>>> minutesFile.close()
>>> resultPdfFile.close()
```

b. What is JSON. Explain json Module with loads() Function, and dumps() Function. **(10 Marks)**

Ans. JavaScript Object Notation is a popular way to format data as a single human-readable string. JSON is the native way that JavaScript programs write their data structures and usually resembles what Python's pprint() function would produce.

The json Module

Python's json module handles all the details of translating between a string with JSON data and Python values for the json.loads() and json.dumps() functions

Reading JSON with the loads() Function

To translate a string containing JSON data into a Python value, pass it to the json.loads() function

```
>>> stringOfJsonData = '{"name": "Zophie", "isCat": true, "miceCaught": 0,
"felineIQ": null}'
>>> import json
>>> jsonDataAsPythonValue = json.loads(stringOfJsonData)
```



```
>>> jsonDataAsPythonValue
{'isCat': True, 'miceCaught': 0, 'name': 'Zophie', 'felineIQ': None}
```

Writing JSON with the dumps() Function

The json.dumps() function (which means "dump string," not "dumps") will translate a Python value into a string of JSON-formatted data.

```
>>> pythonValue = {'isCat': True, 'miceCaught': 0, 'name': 'Zophie',
'felineIQ': None}
>>> import json
>>> stringOfJsonData = json.dumps(pythonValue)
>>> stringOfJsonData
'{"isCat": true, "felineIQ": null, "miceCaught": 0, "name": "Zophie" }'
```