Model Answer

VTU-ADP-18CS55-Feb 2021

Dr. Paras Nath Singh, Professor(CSE/ISE), 2021-22

1.

a. Demonstrate with example print(), input() and string replication. 6 M

Ans:

The print() function: The print() function displays the string value and value of the variable inside the parentheses on the screen.

```
>>> print("sona-sona")
sona-sona
>>> n=20
>>> print("The value of n is",n)
The value of n is 20
```

The input() function: The input() function waits and reads the entered text by the user from the keyboard.

```
>>> name=input("Enter your name :")
Enter your name :Kumar
>>> print("Hi",name)
Hi Kumar
>>> num=int(input("Enter a number : "))
Enter a number : 78
```

b. Explain elif, for, while, break and continue statements in Python with examples for each. 10 M

Ans:

elif statement:

The elif statement is an "else if" statement that always follows an if or another elif statement. It provides another condition that is checked only if any of the previous conditions were False.

Example:

```
if marks>= 90':
    print(You are in A+ grade.')
elif marks>=60:
    print('You are passed with first class.')
else:
    printf("You are below first class.")
```

for statement:

for is a looping statement runs for certain number of times and for number of items in a collection:

for var in range orcollection:

    statements

Example

```
>>> for val in range(10):
        print(val,end=' ')

0 1 2 3 4 5 6 7 8 9
>>> for name in ['Ina','Meena','Dika']:
        print(name,end=' ')

Ina Meena Dika
```

while statement:

while is a looping statement under which block of statements execute till condition is satisfied:

while condition:

    statements

Example:

```
>>> n=13
>>> while n<100:
        print(n,end=' ')
        n+=13  #to print multiples of 13 up to 100

13 26 39 52 65 78 91
```

break statement:

break statement terminates the loop in which it is defined.

Example:

```
#checkprime.py
n=int(input("Enter a number to check whether it is prime: "))
d=2
```

```
import math
prime=True
while d <= math.sqrt(n):
   if n%d == 0:
      prime=False
      break #no need to check further, terminate the loop
   d=d+1
if prime:
   print('Yes, %d is a prime number.' %n)
else:
   print('No, %d is not a prime number.' %n)

#Expected output
Enter a number to check whether it is prime: 97
Yes, 97 is a prime number.
```

c. Write a python program where a given number is even or add. 4 M

Ans:

```
#oddeven.py
num=int(input("Enter a number : "))
if num%2 ==0:
   print(num,"is an even number.")
else:
   print(num,"is an odd number.")

Expected output:
Enter a number : 123
123 is an odd number.
```

2.

a. How can be pass parameters in user defined functions? Explain with suitable example. 5M

Ans:

A parameter is a variable that an argument is stored in when a function is called. An user defined function may or may not have parameters. Parameters are also known as arguments.
Example:

```
def hello(name): #parameter name is received
    print('Hello ' + name)

hello('Rubiya') #parameter is passed
```

```
#output
Hello Rubiya
```

```
#ispalin.py
def ispalin(str1): #parameter s is received as str1
   if str1 == str1[::-1]:
      return True
   else: return False

s=input("Enter string :")
if(ispalin(s)): print("Yes,\""+s+"\" is a palindrome.")
else: print("No,\""+s+"\" is not a palindrome.")
```

```
#Expected output
Enter string :malayalam
Yes,"malayalam" is a palindrome.
#Again
Enter string :Dr. Paras
No,"Dr. Paras" is not a palindrome.
```

Note: Keyword arguments are often used for (to replace) optional parameters

b. Explain local and global scope with local and global variables. 8M

Ans:

Parameters and variables that are assigned in a called function are said to exist in that function's local scope. Variables that are assigned outside all functions are said to exist in the global scope.

- Variables that are assigned outside all functions are said to exist in the global scope & they can be modified/corrupted by any of the function.
- A variable must be one or the other; it cannot be both local and global.

- A local scope is created whenever a function is called. Any variables assigned in this function exist within the local scope
- If we ever want to modify the value stored in a global variable from in a function, we must use a global statement on that variable.
  Example

```
def func1():
   global name
   name="Ratan"
   print("Here name in func1 is "+name);

name='Paras'
print("Here name outside of functions is "+name);
func1()
print("Here name outside of functions is "+name);

#Expected output
Here name outside of functions is Paras
Here name in func1 is Ratan
Here name outside of functions is Ratan
```

c. Demonstrate the concept of exception. Implement a code which prompts the user for Celsius temperature to Fahrenheit, and printout the converted temperature by handling the exception. 7M

Ans:

Exception: Exceptions are run time errors mostly occur due to invalid input. Exceptions (input may causing error) can be handled with **try and except** statements. The code that could potentially have an error is put in a try clause. The program execution moves to the start of a following except clause if an error happens.

Here we can put the code Celsius temperature to convert into Fahrenheit code in a try clause to check whether a number is given or not. An except clause contain code to handle what happens when this error occurs.

```
#ctof.py Celsius to Fahrenheit
inp = input('Enter Celsius Temperature:')
try:
    cel = float(inp)
```

```
    fahr = cel * 9.0 / 5.0 + 32
    print("Equivalent Temparature in Fahrenheit : ",fahr )
except:
    print('Please enter a number')

#Expected output
Enter Celsius Temperature:100
Equivalent Temparature in Fahrenheit :  212.0

#again
Enter Celsius Temperature:-40
Equivalent Temparature in Fahrenheit :  -40.0
```

3.

a. What is list? Explain append(), insert(), and remove() methods with examples. 8M

Ans:

- A list is a **"collection" of values** that contains multiple values in an ordered sequence.
- The term list value refers to the list itself (which is a value that can be stored in a variable or passed to a function like any other value), not the values inside the list value.
- To define an empty list: **lst1 = []** or **lst1=list()**

```
>>> vals = [23,3.45,'Harsha',False]
#Here the list vals contains integer, float, string & logical (Boolean) values
```

- Values inside the list are also called items. Items are separated with commas. Index of items starts with 0 to total number of items -1.
- A list value looks like this: ['cat', 'bat', 'rat', 'elephant'].

```
>>> marks=[65,67,87,76]
```

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

```
>>> marks.append(78)
>>> print(marks)
[65, 67, 87, 76, 78]
```

insert(): The insert() method can insert a value at any index in the list.
The first argument to insert() is the index for the new value, and the second argument is the new value to be inserted. If the index is out of boundary then it appends after last item of the list.

```
>>> marks.insert(2, 98)
>>> print(marks)
[65, 67, 98, 87, 76, 78]
>>> marks.insert(7,99)
>>> print(marks)
[65, 67, 98, 87, 76, 78, 99]
```

b. How is tuple different from a list and which function is used to convert list to tuple? 5M

Ans:

Lists are useful collections of different types of data since they allow us to write code that works on a modifiable number of values in a single variable. Lists are mutable, meaning that their contents can change.

Tuples and strings, although list-like in some respects, are immutable and cannot be changed.

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]. Syntactically, a tuple is a comma-separated list of values:**

Tuples are also comparable and hashable so we can sort lists of them and use tuples as key values in Python dictionaries.

list( ) and tuple( ) functions are used to convert to list and tuple respectively. Examples:

```
>>> tuple(['cat', 'dog', 5])
('cat', 'dog', 5)
>>> list(('cat', 'dog', 5))
['cat', 'dog', 5]
>>> list('hello')
['h', 'e', 'l', 'l', 'o']
```

c. Create a function to print out a blank tic-tac-toe board 7M

Ans:

```
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'])

theBoard = {'top-L': ' ', 'top-M': ' ', 'top-R': ' ', 'mid-L': ' ', 'mid-M': ' ', 'mid-R':
' ', 'low-L': ' ', 'low-M': ' ', 'low-R': ' '}
printBoard(theBoard)

#Expected output:
  | |
-+-+-
  | |
-+-+-
  | |
```

4.

a. Discuss get(), items(), keys() and values() dictionary method in Python with examples. 8M

Ans:

get( ) method:

get() method of dictionary takes two arguments: the key of the value to retrieve and a fallback value to return if that key does not exist.

Example:

```
>>> examitems = {'pen':2, 'pencil':2, 'scale':1}
>>> print('I am bringing ' + str(examitems.get('pencil', 0)) + ' pencils.')
I am bringing 2 pencils.
```

The keys(), values(), and items() Methods

There are three dictionary methods that will return list-like values of the dictionary's keys, values, or both keys and values: keys(), values(), and items(). The values returned by these methods are not true lists: They cannot be modified and do not have an append() method. But these data types (dict_keys, dict_values, and dict_items, respectively) can be used in for loops.

Examples:

items() method : Keys and values both

```
>>> marks={'DSA':76, 'SE': 78, 'CO': 80, 'Python':91}
>>> for a in marks.items():
        print(a)

('DSA', 76)
('SE', 78)
('CO', 80)
('Python', 91)
```

keys() method : Only keys

```
>>> for b in marks.keys():
        print(b)

DSA
SE
CO
Python
```

values() method : Only values

```
>>> for c in marks.values():
        print(c)


76
78
80
91
```

b. With example code explain join() and split() string methods. 6M

Ans:

join() method:

The join() method is useful when you have a list of strings that need to be joined together into a single string value. The join() method is called on a string, gets passed a list of strings, and returns a string. The returned string is the concatenation of each string in the passed-in list.

```
>>> ' '.join(['My', 'Name', 'is','Paras'])
'My Name is Paras'
```

```
>>> 'Sir'.join(['My', 'Name', 'is','Paras'])
'MySirNameSirisSirParas'
```

Note: The string join() calls on is inserted between each string of the list argument.

split() method:

The split() method does the opposite of join method: It's called on a string value and returns a list of
strings.

```
>>> name="My name is Paras"
>>> print(name.split())
['My', 'name', 'is', 'Paras']
```

c. Develop a program to accept a sentence from the user and display the longest word of that sentence along with its length. 6M

Ans:

We can put **key=len** argument in max for split strings for logest string and len() will find its length:

```
#longetsword.py
sentence = input("Enter sentence: ")
longest = max(sentence.split(), key=len)
print("Longest word is: ", longest)
print("And its length is: ", len(longest))

#Expected output:
Enter sentence: I was astonished looking her aura
Longest word is:  astonished
And its length is:  10
```

5.

a. What are regular expressions? Describe question mark, star, plus and dot regex symbols with suitable Python code snippet. 9M

Ans:

- Task of searching and extracting is so common that Python has a very powerful library called regular expressions that handles many of these tasks quite elegantly.
- The regular expression library *re* must be imported into our program before we can use it. The simplest use of the regular expression library is the search() function.
  While there are several steps to using regular expressions in Python, each step is fairly simple.
  - Import the regex module with import re.
  - Create a Regex object with the re.compile() function. (Remember to use a raw string.)
  - Pass the string you want to search into the Regex object's search() method. This returns a Match object.
  - Call the Match object's group() method to return a string of the actual matched text.

  Now importing re module executing the asked regex:

  >>> import re

  The regex ? (Question Marks):

  The ? character flags the group that precedes it as an optional part of the pattern.

```
>>> 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 regex * (star):

  - The * in regex means "match zero or more"—the group that precedes the star can occur any number of times in the text. It can be completely absent or repeated over and over again.

```
>>> 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 regex * (star):

The + (or plus) means “match one or more.”
Unlike the star, which does not require its group to appear in the matched string, the group preceding a plus must appear at least once. It is not optional.

```
>>> batRegex = re.compile(r'Bat(wo)+man')
>>> mo1 = batRegex.search('The Adventures of Batwoman') #no output
>>> mo1.group()
'Batwoman'
```

The regex * (star):

The . (or dot) character in a regular expression is called a wildcard and will match any character except for a newline.

```
>>> atRegex = re.compile(r'.at')
>>> atRegex.findall('The cat in the hat sat on the flat mat.')
['cat', 'hat', 'sat', 'lat', 'mat']
```

b. With code snippet, explain saving variables using the shelve module and Pprint Pformat() functions. 6M

Ans:

We can save variables in our Python programs to binary shelf files using the shelve module. This way, our program can restore data to variables from the hard drive. The shelve module will let us **add, Save, and Open** features to our program. For example, if we ran a program and entered some configuration settings, we could save those settings to a shelf file and then have the program load them the next time it is run.

```
>>> import shelve
>>> shelfFile = shelve.open('mydata')
>>> cats = ['Zophie', 'Pooka', 'Simon']
>>> shelfFile['cats'] = cats
>>> shelfFile.close()
```

Saving Variables with the pprint.pformat() Function:

The pprint.pprint() function will “pretty print” the contents of a list or dictionary to the screen, while the pprint.pformat() function will return this same text as a string instead of printing it. Not only is this string formatted to be easy to read, but it is also syntactically correct Python code. Say you have a dictionary stored in a variable and you want to save this variable and its contents for future use. Using pprint.pformat() will give you a string that you can write to *.py* file. This file will be your very own module that you can import whenever you want to use the variable stored in it.

```
>>> import pprint
>>> cats = [{'name': 'Zophie', 'desc': 'chubby'}, {'name': 'Pooka', 'desc': 'fluffy'}]
>>> pprint.pformat(cats)
"[{'desc': 'chubby', 'name': 'Zophie'}, {'desc': 'fluffy', 'name': 'Pooka'}]"
>>> fileObj = open('myCats.py', 'w')
>>> fileObj.write('cats = ' + pprint.pformat(cats) + '\n')
83
>>> fileObj.close()
```

c. Write a program that reads a string with 5 characters which starts with ‘a’ and ends with ‘z’. Print search successful if pattern matches string. 5M

Ans:

```
#accept and prints string with 5 characters, starts with a and ends with z.
import re
while True:
  str1=input("Enter string : (nomore to end) ")
  if str1 == 'nomore':
    break
  if len(str1)== 5 and re.findall("a.*z", str1):
    print("Yes, the string has 5 characters a...z :",str1)
  else:
    print("Not taken")

#expected output
Enter string : (nomore to end) kumar
Not taken
Enter string : (nomore to end) arbaz
Yes, the string has 5 characters a...z : arbaz
Enter string : (nomore to end) axxxz
Yes, the string has 5 characters a...z : axxxz
```

Enter string : (nomore to end) arakz
Yes, the string has 5 characters a...z : arakz
Enter string : (nomore to end) nomore

6.

a. Explain functions of Shutil module with examples. 8M

Ans:

The shutil (or shell utilities) module has functions to let you copy, move, rename, and delete files in our Python programs. To use the shutil functions, we will first need to use import shutil.

```
>>> import shutil
```

Copying Files and Folders

```
shutil.copy(source, destination)
>>> shutil.copy('e:\\5A-ADP-2021\\Assign1.docx','d:\\pnsback')
'd:\\pnsback\\Assign1.docx'
>>>shutil.copy('e:\\5C-ADP-2020\\Assign1.docx','d:\\pnsback\\5AAssign1.docx')
'd:\\pnsback\\5AAssign1.docx‘
```

shutil.copytree() copies the directory tree

Moving and Renaming Files and Folders

```
>>>shutil.move(‘d:\\pnsback\\5aAssign1.docx‘, ‘e:\\5A-ADP-2021’)  #moving
#existing file in target directory will be overwritten
#if there is no folder 5A-ADP-2021 then file will be moved with name 5A-ADP-2021
>>>shutil.move(‘d:\\pnsback\\5aAssign1.docx‘,  ‘e:\\%B-ADP-2020\\Assign15c.docx’)
```

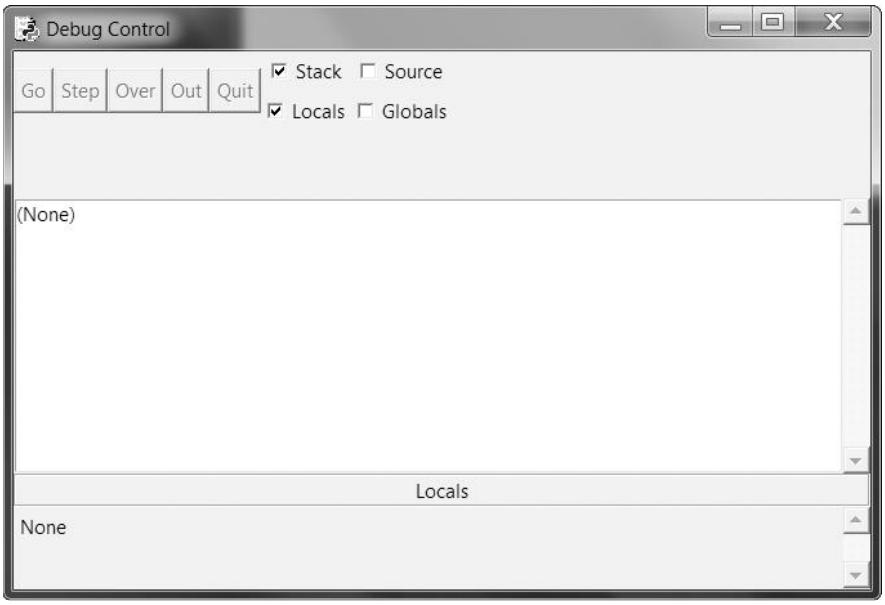
#Moving & Renaming : if the folder/sub-folder is not in the mentioned path, python will throw an exception

b. Explain buttons in Debug control window. 5M

Ans:

To enable IDLE’s debugger, click **Debug**4**Debugger** in the interactive shell window. This will bring up the Debug Control window as shown in the figure below. When the Debug Control window appears, select all four of the **Stack**, **Locals**, **Source**, and **Globals** checkboxes so that the window shows the full set of debug information. While the Debug Control window is displayed, any time we run a program from the file editor, the debugger will pause execution before the first instruction and display the following:

• The line of code that is about to be executed
• A list of all local variables and their values
• A list of all global variables and their values



The program will stay paused until you press one of the five buttons in the Debug Control window: Go, Step, Over, Out, or Quit.

Go

Clicking the Go button will cause the program to execute normally until it terminates or reaches a *breakpoint*.

Step

Clicking the Step button will cause the debugger to execute the next line of code and then pause again. The Debug Control window's list of global and local variables will be updated if their values change.

Over

Clicking the Over button will execute the next line of code, similar to the Step button. However, if the next line of code is a function call, the Over button will "step over" the code in the function.

Out

Clicking the Out button will cause the debugger to execute lines of code at full speed until it returns from the current function. If you have stepped into a function call with the Step button and now simply want to keep executing instructions until you get back out, click the **Out** button to "step out" of the current function call.

Quit

If you want to stop debugging entirely and not bother to continue executing the rest of the program, click the **Quit** button. The Quit button will immediately terminate the program.

c. What is meant by compressing files? Explain reading, extracting and creating zip files with code snippets. 7M

Ans:

A zip file can hold the compressed contents of many other files. Compressing a file reduces its size, which is useful when transferring it over the Internet.
Since a ZIP file can also contain multiple files and subfolders, it's a handy way to package several files into one. This single file, called an archive file, can then be, say, attached to an email. Our Python programs can both create and open (or extract) ZIP files using functions in the zipfile module.

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).

```
>>> import zipfile, os
>>> os.chdir('C:\\') # move to the folder with example.zip
>>> exampleZip = zipfile.ZipFile('example.zip')
>>> exampleZip.namelist()
['spam.txt', 'cats/', 'cats/catnames.txt', 'cats/zophie.jpg']
>>> spamInfo = exampleZip.getinfo('spam.txt')
>>> spamInfo.file_size
13908
>>> spamInfo.compress_size
3828
u >>> 'Compressed file is %sx smaller!' % (round(spamInfo.file_size / spamInfo
.compress_size, 2))
'Compressed file is 3.63x smaller!'
>>> exampleZip.close()
```

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')
u >>> exampleZip.extractall()
>>> exampleZip.close()
```

Creating and Adding to ZIP Files

To create your own compressed ZIP files, you must open the ZipFile object in write mode by passing 'w' as the second argument. (This is similar to opening a text file in write mode by passing 'w' to the open() function.)

```
>>> import zipfile
>>> newZip = zipfile.ZipFile('new.zip', 'w')
>>> newZip.write('spam.txt', compress_type=zipfile.ZIP_DEFLATED)
>>> newZip.close()
```

7.

a. What is class, object, attributes. Explain copy-copy() with example. 6M

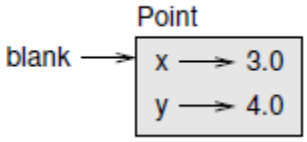
Ans:

Class:

A class is a group of similar objects. It represents object's type also.

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

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



In the given figure the header indicates that the new class is called Point. The body is a docstring that explains what the class is for. You can define variables and methods inside a class definition, but we will get back to that later.

Defining a class named Point creates a **class object**.

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

Because Point is defined at the top level, its "full name" is __main__.Point.

Object:

An object is instance of class. An object has behavior and characteristics. All the real world entities are objects. The class is also sometimes called object. The class object is like a factory for creating objects. To create a Point, we call Point as if it were a function.

```
>>> blank = Point()
>>> blank
<__main__.Point object at 0xb7e9d3ac>
```

The return value is a reference to a Point object, which we assign to blank. Creating a new object is called instantiation, and the object is an instance of the class. When we print an instance, Python tells us what class it belongs to and where it is stored. Objects are mutable.

Attributes:

Attributes are specific information about entity/object. We are assigning values to named elements of an object. These elements are called **attributes**.

```
>>> blank.x = 3.0
>>> blank.y = 4.0
```

A state diagram that shows an object and its attributes is called an **object diagram as shown in the figure**.

b. Explain pure functions and modifiers with examples. 8M

Ans:

Pure function:

The function creates a new 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 effect, like displaying a value or getting user input, other than returning a value.

Here is a simple prototype of add_time:

```
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
```

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

The problem is that this function does not deal with cases where the number of seconds or minutes adds up to more than sixty.

Modifiers:

Sometimes it is useful for a function to modify the objects it gets as parameters. In that case, the changes are visible to the caller. Functions that work this way are called **modifiers**.

```
#addtime.py
class Time:
   def gettime(self):
      self.hr=int(input("Enter hours: "))
      self.min=int(input("Enter minutes: "))
      self.sec=int(input("Enter seconds: "))

   def add_time(self,t1,t2):
      sumt=Time()
      sumt.sec=t1.sec+t2.sec
      xmin, rsec = divmod(sumt.sec, 60)
      sumt.sec=rsec
      sumt.min=t1.min+t2.min+xmin
      xhr, rmin = divmod(sumt.min, 60)
      sumt.min=rmin
      sumt.hr=t1.hr+t2.hr+xhr
      return sumt

t1=Time()
t1.gettime()
t2=Time()
t2.gettime()
t3=Time()
t3=t3.add_time(t1,t2)
print("Adding both times : ",t3.hr,t3.min,t3.sec)
```

#expected output

```
Enter hours: 23
Enter minutes: 45
Enter seconds: 54
Enter hours: 9
Enter minutes: 32
Enter seconds: 51
Adding both times :  33 18 45
```

c. Use the date time module to write a program that gets and prints that day of the week. 6M

Ans:

```
#weekday.py
import datetime
import calendar
def findDay(date1):
   wd = datetime.datetime.strptime(date1, '%d %m %Y').weekday()
   #weekday returns day of week starting from Monday as 0..Sunday 6
   return (calendar.day_name[wd])

print("Entering date:")
dd=int(input("Enter Day (dd): "))
mm=int(input("Enter Month (mm): "))
yyyy=int(input("Enter year (yyyy) : "))
date1=str(dd)+' '+str(mm)+' '+str(yyyy)
print("Weekday of date",date1,"is",findDay(date1))
```

#Expected output:

```
Entering date:
Enter Day (dd): 20
Enter Month (mm): 03
Enter year (yyyy) : 1958
Weekday of date 20 3 1958 is Thursday
```

8.

a. Explain operators overloading and polymorphism with example. 8M

Ans:

Changing the behavior of an operator so that it works with programmer-defined types is called **operator overloading**. For every operator in Python there is a corresponding special method, like __add__ for +

By defining other special methods, we can specify the behavior of operators on programmer-defined types. For example, if we define a method named __add__ for the Time class, we can use the + operator on Time objects.

```
#operators' overloading
def int_to_time(seconds):
    minutes, second = divmod(seconds, 60)
    hour, minute = divmod(minutes, 60)
    time = Time(hour, minute, second)
    return time

class Time:
    def __init__(self, hour=0, minute=0, second=0):
        self.hour = hour
        self.minute = minute
        self.second = second
    def __add__(self, other):
        seconds = self.time_to_int() + other.time_to_int()
        return int_to_time(seconds)
    def time_to_int(self):
        minutes = self.hour * 60 + self.minute
        seconds = minutes * 60 + self.second
        return seconds
    def __str__(self):
        return '%.2d:%.2d:%.2d' % (self.hour, self.minute, self.second)

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

#expected output:
11:20:00
```

b. Illustrate the concept of inheritance and class diagrams with examples.8M

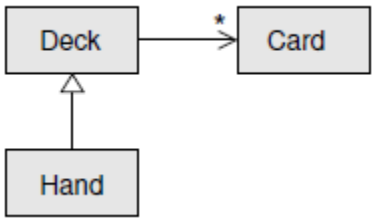
Ans:

Inheritance:

Objects of the derived class inherit the properties of base class. Inheritance is the ability to define a new class that is a modified version of an existing Class.

"Mammal is the derived class from base class animal."
"BE_CSE is derived from course"

One class might inherit from another. This relationship is called **IS-A**, as in, "a Hand is a kind of a Deck."
A **class diagram** is a graphical representation of these relationships. For example, The figure (given below) shows the relationships between Card, Deck and Hand.



A class diagram is a more abstract representation of the structure of a program. Instead of showing individual objects, it shows classes and the relationships between them.

The arrow with a hollow triangle head represents an IS-A relationship; in this case it indicates that Hand inherits from Deck. The standard arrow head represents a HAS-A relationship; in this case a Deck has references to Card objects. The star (*) near the arrow head is a **multiplicity**; it indicates how many Cards a Deck has.
A multiplicity can be a simple number, like 52, a range, like 5..7 or a star, which indicates that a Deck can have any number of Cards.
There are no dependencies in this diagram. They would normally be shown with a dashed arrow. Or if there are a lot of dependencies, they are sometimes omitted.

The inheritance may be multilevel, multiple, hybrid and hierarchical.

c. Write a function called print_time that takes a time object and print it in the form of hour: minute: second. 4M

Ans:

```
#timex.py
class Time(object):
   def print_time(self,time):
```

```
    print ("%.2d:%.2d:%.2d" % (time.hour, time.minute, time.second))
    #%.2d prints value in 2 digits filling the leading blank with 0

time = Time()
time.hour = int(input("Enter hour: "))
time.minute = int(input("Enter Minutes: "))
time.second = int(input("Enter Seconds: "))
time.print_time(time)

#Expected output
Enter hour: 9
Enter Minutes: 23
Enter Seconds: 7
09:23:07
```

9.

a. Explain parsing HTML with the BeautifulSoup module with code snippet for creating finding an element and getting data. 9M

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).

Here parsing means the BeautifulSoup can analyze and identify the parts of) an HTML file on the hard drive. We can open a new file editor window in IDLE, enter the following, and save it as *example.html*.

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

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

This code uses requests.get() to download the main page from the No Starch Press website and then passes the text attribute of the response to bs4.BeautifulSoup(). The BeautifulSoup object that it returns is stored in a variable named noStarchSoup.

b. What methods do Selenium's web element object have for simulating mouse clicks and keyboard keys. Explain with Python code snippet.6M

Ans:

Actions class is an ability provided by Selenium for handling keyboard and mouse events. In Selenium WebDriver, handling these events includes operations such as drag and drop, clicking on multiple elements with the control key, among others. These operations are performed using the advanced user interactions API. It mainly consists of *Actions* that are needed while performing these operations.

Action class is defined and invoked using the following syntax:

```
Actions action = new Actions(driver);
action.moveToElement(element).click().perform();
```

```
driver.get("cmrit.ac.in");
Actions action = new Actions(driver);
element = driver.findElement(By.linkText("Academic"));
action.moveToElement(element).click();
```

Mouse Actions in Selenium:

doubleClick(): Performs double click on the element
clickAndHold(): Performs long click on the mouse without releasing it
dragAndDrop(): Drags the element from one point and drops to another
moveToElement(): Shifts the mouse pointer to the center of the element
contextClick(): Performs right-click on the mouse

Keyboard Actions in Selenium:

sendKeys(): Sends a series of keys to the element
keyUp(): Performs key release
keyDown(): Performs keypress without release

```
#code snippet
from selenium import webdriver
from selenium.webdriver.common.keys import Keys
browser = webdriver.Firefox()
browser.get('http://nostarch.com')
htmlElem = browser.find_element_by_tag_name('html')
htmlElem.send_keys(Keys.END) # scrolls to bottom
htmlElem.send_keys(Keys.HOME) # scrolls to top
```

c. Write Python program to access cell in a worksheet. 5M

Ans:

Example of worksheet:

	A	B	C
1	4/5/2015 1:34:02 PM	Apples	73
2	4/5/2015 3:41:23 AM	Cherries	85
3	4/6/2015 12:46:51 PM	Pears	14
4	4/8/2015 8:59:43 AM	Oranges	52
5	4/10/2015 2:07:00 AM	Apples	152
6	4/10/2015 6:10:37 PM	Bananas	23
7	4/10/2015 2:40:46 AM	Strawberries	98

Once we have a Worksheet object, we can access a Cell object by its name.

```
import openpyxl
wb = openpyxl.load_workbook('example.xlsx')
sheet = wb.get_sheet_by_name('Sheet1')
sheet['A1'] #<Cell Sheet1.A1>
print(sheet['A1'].value) #datetime.datetime(2015, 4, 5, 13, 34, 2)
c = sheet['B1']
print(c.value) #'Apples'
print('Row ' + str(c.row) + ', Column ' + c.column + ' is ' + c.value)
#'Row 1, Column B is Apples'
print('Cell ' + c.coordinate + ' is ' + c.value) #'Cell B1 is Apples'
print(sheet['C1'].value) #73
```

10.

a. Write a program to get a list of files with the pdf extension in the current working directory and sort them. 6M

Ans:

The glob module finds all the pathnames matching a specified pattern according to the rules used by the operating system shell. We will use glob.glob() function for the solution directory listing. Copying in a list we can sort the list.

```
#filelist.py
import glob
list1 = glob.glob(r"D:\pnsback\*.pdf")
list1 = sorted(list1)
for file in list1:   #displaying sorted list of files
    print(file)

#Expected output in ASCII collating sequence
D:\pnsback\Calendar of Events odd sem 17-18.pdf
D:\pnsback\DivyamanasaUCDAVIS.pdf
D:\pnsback\DrPNSingh-External-MVJCE.pdf
D:\pnsback\Guide List.pdf
D:\pnsback\Ramya.pdf
D:\pnsback\SwatiMTech-PhD-IIIT.pdf
D:\pnsback\VTU-LIC-MEMBER.pdf
D:\pnsback\cds1q.pdf
D:\pnsback\cds2q.pdf
D:\pnsback\cse4syll.pdf
D:\pnsback\tavleen.pdf
```

b. Demonstrate the json module with Python program. 6M

Ans:

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. JSON can't store *every* kind of Python value. It can contain values of only the following data types: **strings, integers, floats, Booleans, lists, dictionaries, and NoneType.** JSON cannot represent Python-specific objects, such as File objects, CSV Reader or Writer objects, Regex objects, or Selenium WebElement objects.

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

Example Program:

```
pythonValue = {'isFaculty': True, 'Failed': 0, 'Name': 'DrPNSingh', 'Weak': None}
import json
stringOfJsonData = json.dumps(pythonValue)
print(stringOfJsonData)

#Expected output
{"isFaculty": true, "Failed": 0, "Name": "DrPNSingh", "Weak": null}
```

c. What are the advantages of CSV files? Explain the reader objects and writer objects with Python codes. 8M

Ans:

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.

CSV files are simple so lacking many of the following features of an Excel spreadsheet.

- 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

Reader Object: A Reader object lets us iterate over lines in the CSV file. To read data from a CSV file with the csv module, you need to create a Reader object.

```
import csv
exampleFile = open('example.csv')
exampleReader = csv.reader(exampleFile)
exampleData = list(exampleReader)
print(exampleData)

#expected output
```

Printing data of example.csv

Writer object: A Writer object lets you write data to a CSV file. To create a Writer object, we use the csv.writer() function. Python code given on shell prompt:

```
>>> import csv
>>> outputFile = open('output.csv', 'w', newline='') #w for write mode
#newline='' to suppress the double space
>>> outputWriter = csv.writer(outputFile)
>>> outputWriter.writerow(['Rinku', 'Pinku', 'Tinku', 'Minku'])
# 25 Number of characters written in file by writer object
>>> outputWriter.writerow(['Pranam, Sir', 'Pranam Madam', 'How are you?'])
# 41 Number of characters written in file41
>>> outputWriter.writerow([1261, 1391, 3.141592, "Ramu",True])
# 30 Number of characters written in file
```

#Expected output in output.csv (sheet mode)

Rinku	Pinku	Tinku	Minku	
Pranam, Sir	Pranam Madam	How are you?		
1261	1391	3.141592	Ramu	TRUE