CBCS SCHEME

USN 1 C R 1 9 M C A 0 2

18MCA32

Third Semester MCA Degree Examination, Jan./Feb. 2021
Programming using Python

Time: 3 hrs. Max. Marks: 100

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

Important Note : 1. On completing your answers, compulsorily draw diagonal cross lines on the remaining blank pages.
2. Any revealing of identification, appeal to evaluator and /or equations written eg, 42+8 = 50, will be treated as malpractice.

Module-1

1 a. Explain any string functions with examples. **(10 Marks)**

b. Give the output of the following:

i) (-(-(-5)))
ii) 5 * 2 * * 3 – 15
iii) -9 % 2
iv) 9 % -2
v) -17/10 **(10 Marks)**

OR

2 a. Explain the two ways to use python interpreter. What are error that can be detected by Python? Differentiate between them with one example each. **(10 Marks)**

b. Write a python program to find sum of all odd and even numbers from n1 to n2 where n1 and n2 are positive integers. **(10 Marks)**

Module-2

3 a. Explain how code in python is tested semi-automatically. **(10 Marks)**

b. Describe briefly the process of designing your own module with clear example. **(10 Marks)**

OR

4 a. Trace the function call and explain the memory model of the following code:

```
def fn(x):
    x = 2 * x
    return x
x = 1
x = fn(x+1) + fn(x+2)
```

(10 Marks)

b. Write a python function to find the average of two bigger numbers of given three numbers. **(10 Marks)**

Module-3

5 a. Write a python program to search an element using binary search (Recursive). **(08 Marks)**

b. Compare list and string in python. **(04 Marks)**

c. Explain any five list methods with example. **(08 Marks)**

OR

6 a. Write a python program to compute sum of diagonals of 3 × 3 square matrix. **(10 Marks)**

b. What do you mean by slicing of lists? List and explain the various operations that can be applied on lists. **(10 Marks)**

CREATE VIEW Detailview As
Name, Addres

Module-4

7 a. Write a program to read a word and print the number of letters, vowels and percentage of vowels in the word using a dictionary. **(10 Marks)**

b. Demonstrate any 6 set operations with examples. **(10 Marks)**

OR

8 a. Write a python program to read contents of a text file and write into another. **(10 Marks)**

b. Write a function to create a dictionary where the keys are numbers between 1 and N (both included N is taken as input) and the values are square of keys. Print the contents of the dictionary. **(10 Marks)**

Module-5

9 a. Explain MVC design with the help of Tkinter program. **(10 Marks)**

b. Write a python class named square constructed by a side and two methods which will compute the area and perimeter of a square. **(10 Marks)**

OR

10 a. Demonstrate the creation of any 5 widgets using Tkinter. **(10 Marks)**

b. Explain tkinter based python program for creating a GUI that has a label, entry and a button. The values given in entry field should be updated in label on click of the button. **(10 Marks)**

* * * * *

1 a.) swapcase(...)

S.swapcase() -> string

Return a copy of the string S with uppercase characters

converted to lowercase and vice versa

strip(...)

S.strip([chars]) -> string or unicode

Return a copy of the string S with leading and trailing

whitespace removed.

If chars is given and not None, remove characters in chars instead.

startswith(...)

S.startswith(prefix[, start[, end]]) -> bool

Return True if S starts with the specified prefix, False otherwise.

With optional start, test S beginning at that position.

With optional end, stop comparing S at that position.

prefix can also be a tuple of strings to try.

split(...)

S.split([sep [,maxsplit]]) -> list of strings

Return a list of the words in the string S, using sep as the

delimiter string. If maxsplit is given, at most maxsplit

splits are done. If sep is not specified or is None, any

whitespace string is a separator and empty strings are removed

from the result.

format(...)

S.format(*args, **kwargs) -> string

Return a formatted version of S, using substitutions from args and kwargs.

The substitutions are identified by braces ('{' and '}').

1 b.) (i) (-(-(-5)))

O/P: -5

(ii) 5*2**3-15

O/P: 25

(iii) -9%2

O/P: 1

(iv) 9%-2

O/P: -1

(v) -17/10

O/P: -1.7

Q2 a: There are two ways to use the Python interpreter, one is to tell it to execute a python program that is saved in a file with a .py extension, Another is to interact with it in a program called a shell, where you type statements one at a time. The interpreter will execute each statement when you type it, do what the statement says to do, and show any outpu as text, all in one window.

There are two kinds of errors in Python:

- syntax errors -coding errors which dont follow Python syntax rules. E.g. c=a b- Here there is no operator between identifiers a and b.
- semantic errors: When you tell Python to do something that it cannot do,
  Example 1 - divide a number by 0
  Example 2 - try to use a variable that does not exist..
  Example 3 - access element outside array boundary

Q2 b: n1 = int(input("Enter n1 :"))

n2 = int(input("Enter n2 :"))

```
sum1=0
sum2=0

i=n1



while i<=n2 :
    if i%2==0:
        sum1 = sum1+i
    else :
        sum2= sum2+i
    i=i+1



print ("Sum of Even Numbers from "+str(n1)+" to "+str(n2)+" = "+str(sum1))
print ("Sum of Odd Numbers from "+str(n1)+" to "+str(n2)+" = "+str(sum2))
```

I/O: Enter n1 :2

Enter n2 :8

Sum of Even Numbers from 2 to 8 = 20

Sum of Odd Numbers from 2 to 8 = 15

Q3a.: Python has a module called doctest that allows to run tests that are included in the docstring all at once. The function that enables us to print such a report is testmod. It reports on whether the function calls return what we expect. It gives messages on how many tests succeeded and how many failed. The test cases can be specified in the docstring for a function as shown in the example below:

```
def large(a,b,c):
        '''
                (number, number, number) --> number
```

```
        Finds the largest of the three numbers given as input
        >>> large(5,1,3)
        5
        >>> large(4,10,7)
        10
    '''
    if a>b:
        if a>c:
            return a
        else:
            return c
    else:
        if  b > c:
            return b
        else:
            return c
```

In the above function to find maximum of 3 numbers the docstring specifies two tests for the numbers 5,1,3 and 4,10,7. The line next to the function call in docstring specifies the output expected. The testmod function parses the docstring runs the test specified and compares the result to the expected result to give the output. For instance importing the module containing the function above and typing the following commands in the python

```
>>>import doctest
>>> doctest.testmod()
```

tests all the functions in the current enviroment. In this case it will give the following output:

```
>>> doctest.testmod()
```

TestResults(failed=0, attempted=2)

This means two tests were run and none of them failed.

Q3 b: : Module is a collection of related function and variables. E.g. math module

Python allows us to create a module by creating a python file and including definitions of all functions and related variables. the module can then be imported by using import <filename> of file containing the module.

Example if we create a file Integer.py

```
x=10
y=20
def add(a,b):
     return a+b

def sub(a,b):
    return a-b

def mult(a,b):
   return a-b
```

Importing the module using

>>> import Integer

results in the function add, mult and sub along with the variables x and y to be available.

They can be used in the following manner:

>>> Integer.add(5,7)

12

>>>Integer.mult(3,x)

30

Q4 a:

Whenever Python executes a function call it creates a amespace in which to store local variables for theat call. Python also another namespace for variables created in the shell.

The steps involved in executing a function call are:

1. Evaluate the arguments left to right

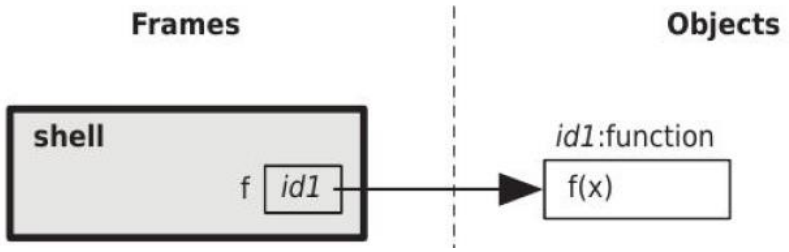
2. Create a namespace to hold the functions call's local variables , including the parameters

3. Pass the resulting argument values into the function by assigning them to the parameters

4. Execute the function body. The value of the expression in the return statement is used as the value of the function call.

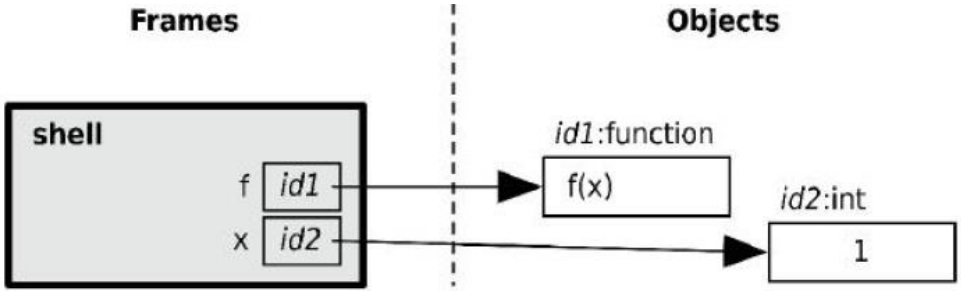
A namespace is created for each function call. This is called a frame

When it encounters the first line it creates a variable f in the shell frame and a function object.

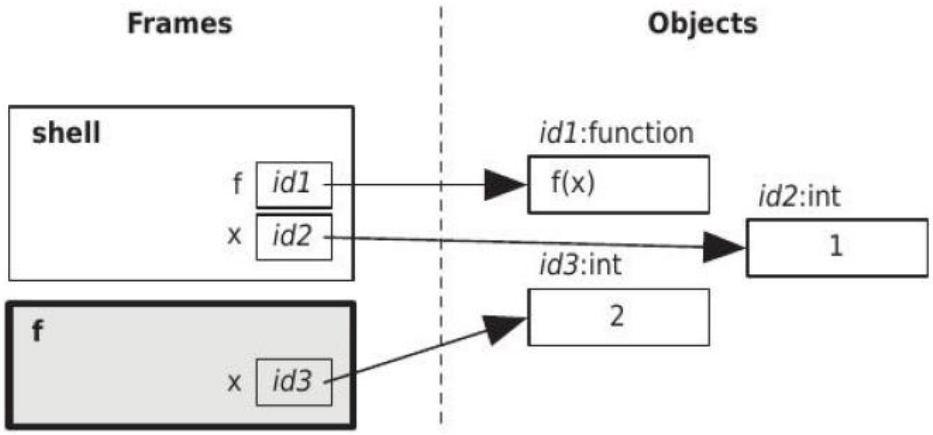



```
>>> def f(x):
...     x = 2 * x
...     return x
...
>>> x = 1
>>> x = f(x + 1) + f(x + 2)
```

On encountering the varaible initialization for x , an object and a variable x are created.

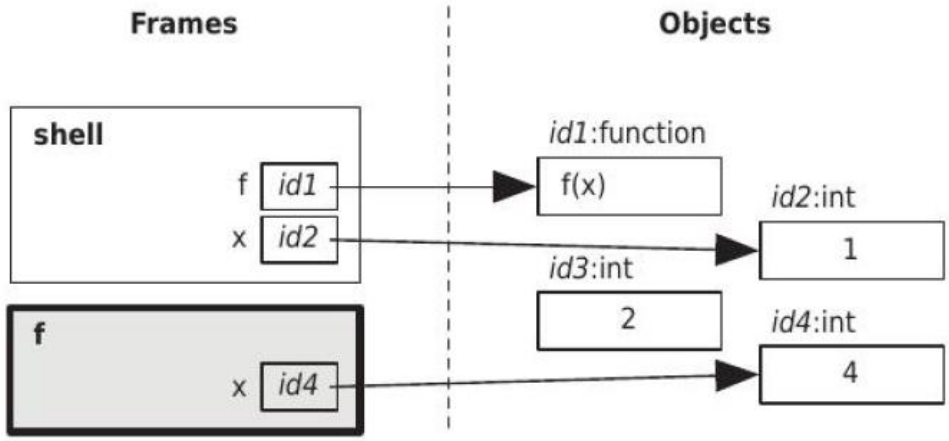


Following python rules x+1 is first evaluated to 2. The next step is to createa a namespace for the function call f as shown below

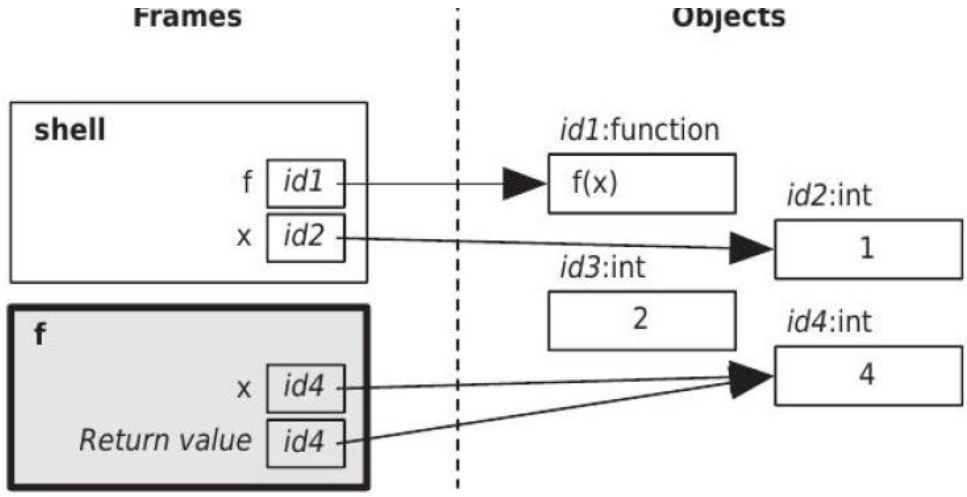



```
>>> def f(x):
...     x = 2 * x
...     return x
...
>>> x = 1
>>> x = f(x + 1) + f(x + 2)
```

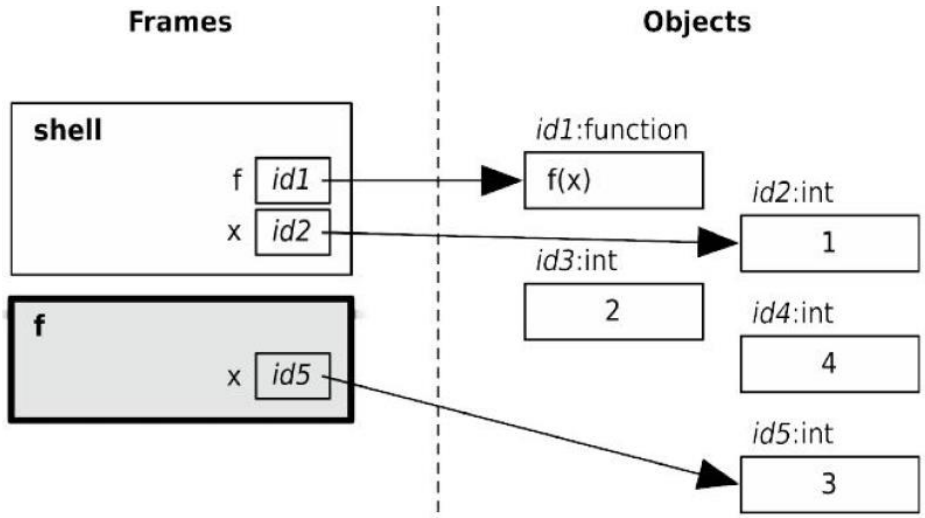
The first statement of the function f is executed to give a value of x as 2*2=4.



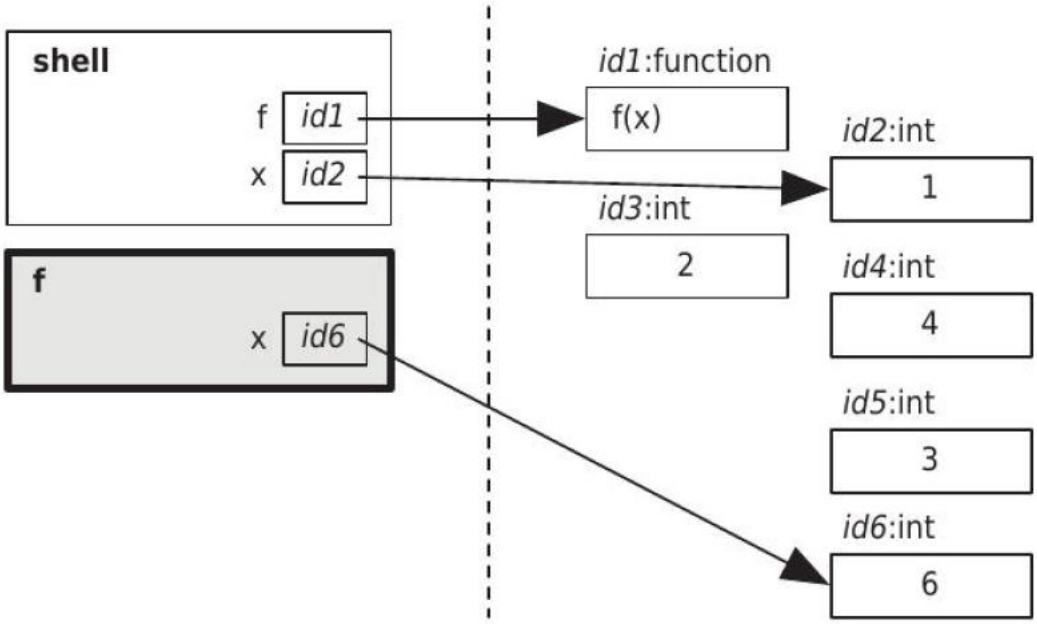
The value 4 is returned as the second statement of the function



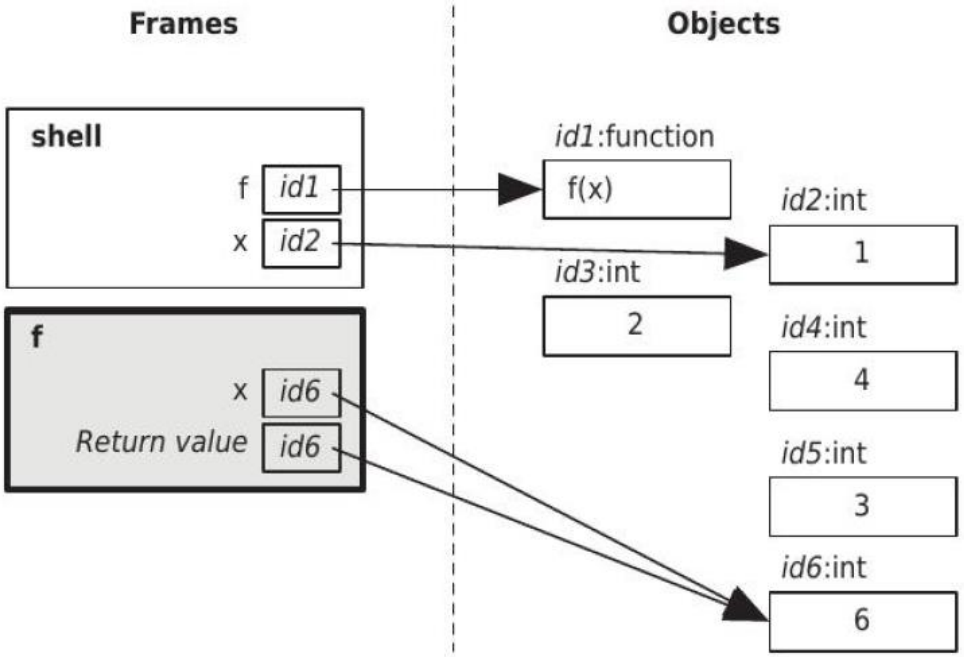
When the function returns Python now executes the right function call f(x+2). According to the rules x+2 is first evaluated , x+2 evaluates to 3. Again a namespace for the function call is created.



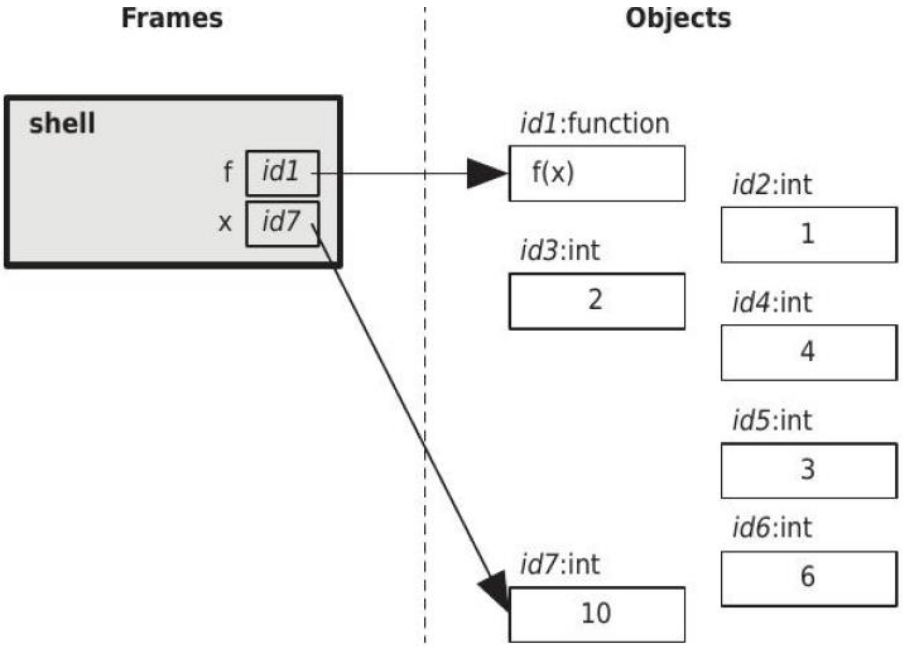
In the function the first statement is executes x evaluates to 2*3=6.



The value 6 is returned through the second statement.



When the function returns, Python comes back to expressin f(x+1)+f(x+2). This evaluates to 4+6=10. This value is assigned to the variable on the left i.e. x.



Q 4 b: num1 = float(input("Enter first number: "))

num2 = float(input("Enter second number: "))

num3 = float(input("Enter third number: "))

if (num1 < num2) and (num1 < num3):

    small = num1

elif num2<num3:

    small=num2

else :

    small= num3

```
average= (num1+num2+num3-small)/2
print("Average of two greater number is:", average)
```

I/O: Enter first number: 2

Enter second number: 3

Enter third number: 5

Average of two greater number is: 4.0

Q5a:

```
# Python 3 program for recursive binary search.
# Modifications needed for the older Python 2 are found in comments.

# Returns index of x in arr if present, else -1
def binary_search(arr, low, high, x):

    # Check base case
    if high >= low:

        mid = (high + low) // 2

        # If element is present at the middle itself
        if arr[mid] == x:
```

```
            return mid

        # If element is smaller than mid, then it can only
        # be present in left subarray
        elif arr[mid] > x:
            return binary_search(arr, low, mid - 1, x)

        # Else the element can only be present in right subarray
        else:
            return binary_search(arr, mid + 1, high, x)

    else:
        # Element is not present in the array
        return -1

# Test array
arr = [ 2, 3, 4, 10, 40 ]
x = 10
```

```
# Function call

result = binary_search(arr, 0, len(arr)-1, x)



if result != -1:

    print("Element is present at index", str(result))

else:

    print("Element is not present in array")
```

Output:
Element is present at index 3

Q5 b: Strings can only consist of characters, while **lists** can contain any data type. Because of the previous **difference**, we cannot easily make a **list** into a **string**, but we can make a **string** into a **list** of characters, simply by using the **list**() function. ... **Strings** are immutable, meaning that we cannot update them.

One simple difference between strings and lists is that lists can any type of data i.e. integers, characters, strings etc, while strings can only hold a set of characters.

We can change the value of a list after we have created it, but we cannot in case of strings.

List is mutable whereas strings are immutable.

Q5 c:

a) extend - attaches another list to the end of current list.

e.g.

```
l=[1,2,3]

l1=['a','b','c']

l.extend(l1)
```

Output:

```
l=[1,2,3,'a','b','c']
```

b) append - attaches a single element to the end of the list

l=[1,2,3]

l.append('a')

Output:

l=[1,2,3,'a']

c) remove - remove the element specified

l=[1,2,3]

l.remove(2)

Output

l=[1,3]

d)pop() -The pop() method removes the element at the specified position.

fruits = ['apple', 'banana', 'cherry']

fruits.pop(1)

e) insert() : The insert() method inserts the specified value at the specified position.

fruits = ['apple', 'banana', 'cherry']

fruits.insert(1, "orange")

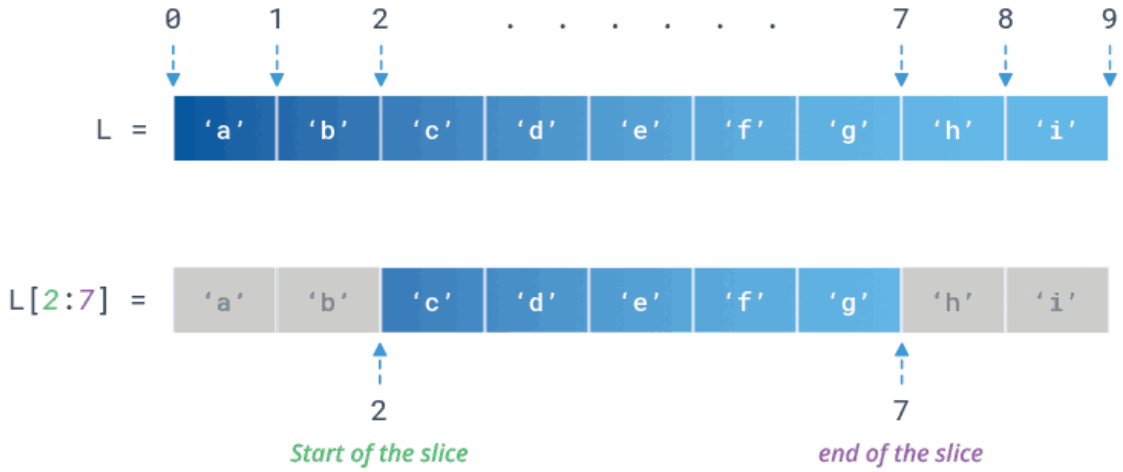
Q6a:
```
# A simple Python program to
# find sum of diagonals
MAX = 100

def printDiagonalSums(mat, n):

    principal = 0
    secondary = 0;
    for i in range(0, n):
        for j in range(0, n):

            # Condition for principal diagonal
            if (i == j):
```

```
            principal += mat[i][j]

        # Condition for secondary diagonal
        if ((i + j) == (n - 1)):
            secondary += mat[i][j]

    print("Principal Diagonal:", principal)
    print("Secondary Diagonal:", secondary)

# Driver code
a = [[ 1, 2, 3, 4 ],
     [ 5, 6, 7, 8 ],
     [ 1, 2, 3, 4 ],
      [ 5, 6, 7, 8 ]]
printDiagonalSums(a, 4)
```

Q6b: Slicing a List

If L is a list, the expression L [start : stop : step] returns the portion of the list from index start to index stop, at a step size step.

```
L = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i']
print(L[2:7])
# Prints ['c', 'd', 'e', 'f', 'g']
```



Operations on List

1. [ ]

Creates an empty list.

```
y = []
```

2. Mutable operations

These operations allow us to work with lists, but altering or modifying their previous definition.

2.1. append

Adds a single item to the bottom of the list.

```
x = [1, 2]x.append('h')print(x)Output:[1, 2, 'h']
```

2.2. extend

Adds another list to the end of a list.

```
x = [1, 2]x.extend([3, 4])print(x)Output:[1, 2, 3, 4]
```

2.3. insert

Inserts a new element at a specific position in the list, this method receives the position as a first argument, and the element to add as a second argument .

```
x = [1, 2]x.insert(0, 'y')print(x)Output:['y', 1, 2]
```

2.4. del

Deletes the element located in the specific index. This method also has the possibility to remove a section of elements from the list, through the ":" operator. You only need to define a starting and end point [start:end], keep in mind that the end point will not be considered. These points can be ignored, whereby the 0 position will be the starting point, and the last position in the list will be the end point.

```
x = [1, 2, 3]del x[1]print(x)Output:[1, 3]y = [1, 2, 3, 4, 5]del y[:2]print(y)Output:[3, 4, 5]
```

2.5. remove

Removes the first match for the specified item.

```
x = [1, 2, 'h', 3, 'h']x.remove('h')print(x)Output:[1, 2, 3, 'h']
```

2.6. reverse

Reverses the order of the elements in the list, this places the final elements at the beginning, and the initial elements at the end.

```
x = [1, 2, 'h', 3, 'h']x.reverse()print(x)Output:['h', 3, 'h', 2, 1]
```

2.7. sort

By default, this method sorts the elements of the list from smallest to largest, this behavior can be modified using the parameter reverse = True.

```
x = [3, 2, 1, 4]x.sort()print(x)Output:[1, 2, 3, 4]y = ['R', 'C', 'Python', 'Java',
'R']y.sort(reverse=True)print(y)Output:['R', 'R', 'Python', 'Java', 'C']
```

Q 7 a: vowel_count = 0

count = 0

total = 0

space = 0

vowels='aeiou'

string = raw_input("Type a word : ").lower()

for char in string:

    total += 1

    if char in 'aeiou':

        vowel_count += 1

```
    elif char ==' ':


        space += 1
    else:
        count += 1
print "Vowel Count : "+str(vowel_count)
print "Space Count : "+str(space)
print "Consonent Count : "+str(count)
print ("Percentage of Vowels : ")
print (float)(vowel_count)/(count)*100


# make a dictionary with each vowel a key and value 0
count1 = {}.fromkeys(vowels,0)


# count the vowels
for char in string:
   if char in count1:
```

```
        count1[char] += 1



print(count1)

#print('vowel count is {0}.format(vowel_count)')

#print('letter count is {0}.format(count)')
```

Output

```
mca@mca-Veriton-Series:~/pythonfiles$ python lab11.py
Type a word : GodisGood
Vowel Count : 4
Space Count : 0
Consonent Count : 5
Percentage of Vowels :
80.0
{'a': 0, 'i': 1, 'e': 0, 'u': 0, 'o': 3}
mca@mca-Veriton-Series:~/pythonfiles$ gedit lab11.py
```

Q 7 b:

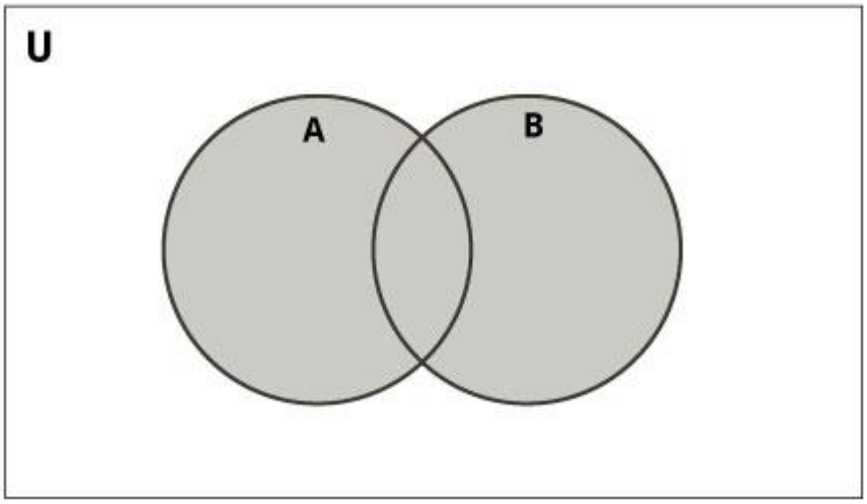
Python Set Operations

Sets can be used to carry out mathematical set operations like union, intersection, difference and symmetric difference. We can do this with operators or methods.

Let us consider the following two sets for the following operations.

```
>>> A = {1, 2, 3, 4, 5}
>>> B = {4, 5, 6, 7, 8}
```

Set Union



Set Union in Python

Union of A and B is a set of all elements from both sets.
Union is performed using | operator. Same can be accomplished using the union() method.

```
# Set union method
# initialize A and B
A = {1, 2, 3, 4, 5}
B = {4, 5, 6, 7, 8}

# use | operator
# Output: {1, 2, 3, 4, 5, 6, 7, 8}
print(A | B)
```

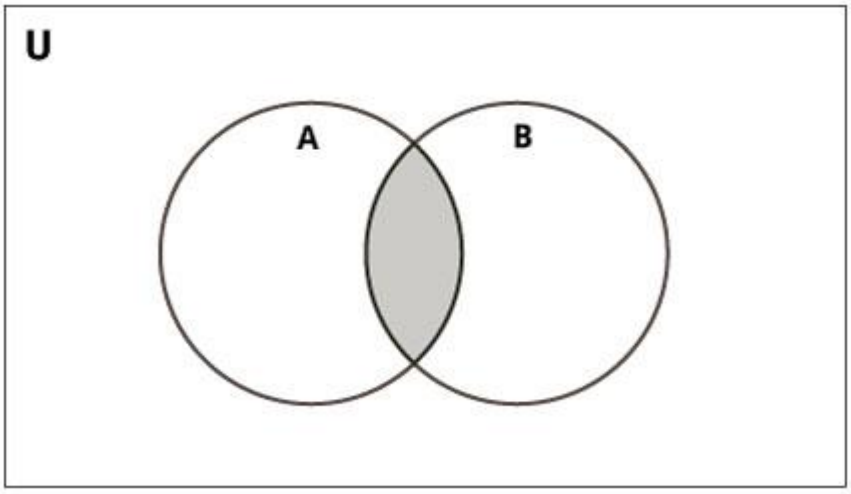
Output

```
{1, 2, 3, 4, 5, 6, 7, 8}
```

Try the following examples on Python shell.

```
# use union function
>>> A.union(B)
{1, 2, 3, 4, 5, 6, 7, 8}

# use union function on B
>>> B.union(A)
{1, 2, 3, 4, 5, 6, 7, 8}
```

Set Intersection



Set Intersection in Python

Intersection of A and B is a set of elements that are common in both the sets.
Intersection is performed using & operator. Same can be accomplished using the intersection() method.

```
# Intersection of sets
# initialize A and B
A = {1, 2, 3, 4, 5}
B = {4, 5, 6, 7, 8}

# use & operator
# Output: {4, 5}
print(A & B)
```

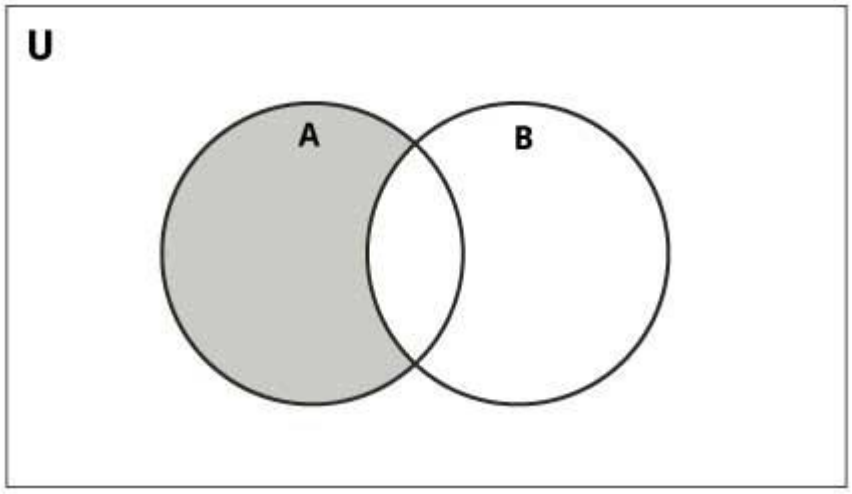
Output

```
{4, 5}
```

Try the following examples on Python shell.

```
# use intersection function on A
>>> A.intersection(B)
{4, 5}

# use intersection function on B
>>> B.intersection(A)
{4, 5}
```

Set Difference



Set Difference in Python

Difference of the set B from set A(A - B) is a set of elements that are only in A but not in B.
Similarly, B - A is a set of elements in B but not in A.
Difference is performed using - operator. Same can be accomplished using the difference() method.

```
# Difference of two sets
# initialize A and B
A = {1, 2, 3, 4, 5}
B = {4, 5, 6, 7, 8}

# use - operator on A
# Output: {1, 2, 3}
print(A - B)
```

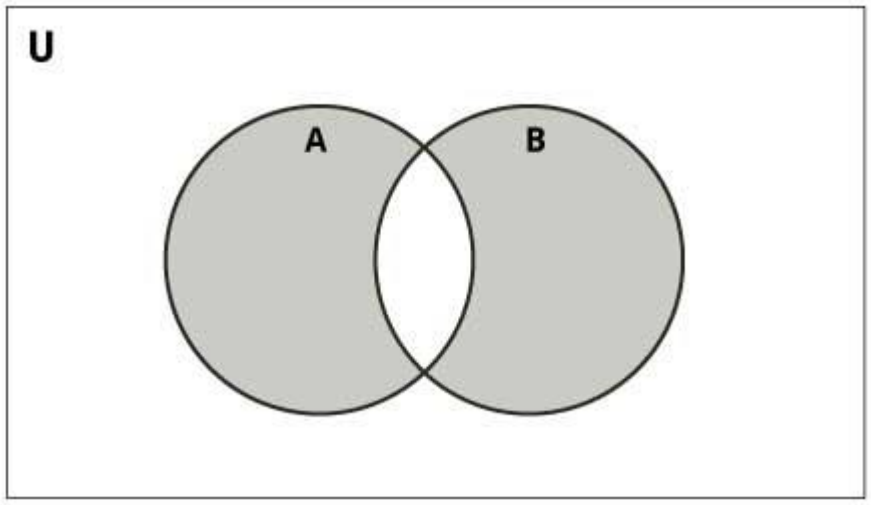
Output

```
{1, 2, 3}
```

Try the following examples on Python shell.

```
# use difference function on A
>>> A.difference(B)
{1, 2, 3}

# use - operator on B
>>> B - A
{8, 6, 7}

# use difference function on B
>>> B.difference(A)
{8, 6, 7}
```

Set Symmetric Difference



Set Symmetric Difference in Python

Symmetric Difference of A and B is a set of elements in A and B but not in both (excluding the intersection).
Symmetric difference is performed using ^ operator. Same can be accomplished using the method symmetric_difference().

```
# Symmetric difference of two sets
# initialize A and B
A = {1, 2, 3, 4, 5}
B = {4, 5, 6, 7, 8}

# use ^ operator
# Output: {1, 2, 3, 6, 7, 8}
print(A ^ B)
```

Output

```
{1, 2, 3, 6, 7, 8}
```

Try the following examples on Python shell.

```
# use symmetric_difference function on A
>>> A.symmetric_difference(B)
{1, 2, 3, 6, 7, 8}

# use symmetric_difference function on B
>>> B.symmetric_difference(A)
```

```
{1, 2, 3, 6, 7, 8}
```

Other Python Set Methods

There are many set methods, some of which we have already used above. Here is a list of all the methods that are available with the set objects:

Method	Description
add()	Adds an element to the set
clear()	Removes all elements from the set
copy()	Returns a copy of the set
difference()	Returns the difference of two or more sets as a new set
discard()	Removes an element from the set if it is a member. (Do nothing if the element is not in set)
intersection()	Returns the intersection of two sets as a new set

Q 8 a:
```
file1 = open("myfile.txt","w")
L = ["This is Delhi \n","This is Paris \n","This is London \n"]

# \n is placed to indicate EOL (End of Line)
file1.write("Hello \n")
file1.writelines(L)
file1.close() #to change file access modes

file1 = open("myfile.txt","r+")

print "Output of Read function is "
print file1.read()
print

# seek(n) takes the file handle to the nth
# bite from the beginning.
file1.seek(0)
```

```
print "Output of Readline function is "
print file1.readline()
print

file1.seek(0)

# To show difference between read and readline
print "Output of Read(9) function is "
print file1.read(9)
print

file1.seek(0)

print "Output of Readline(9) function is "
print file1.readline(9)

file1.seek(0)
# readlines function
print "Output of Readlines function is "
print file1.readlines()
print
file1.close()
```

```
Output of Read function is

Hello

This is Delhi

This is Paris

This is London



Output of Readline function is

Hello



Output of Read(9) function is

Hello

Th


Output of Readline(9) function is

Hello
```

Output of Readlines function is

['Hello \n', 'This is Delhi \n', 'This is Paris \n', 'This is London \n']

Q 8 b:

```
def fun1(N1,N2):
    d=dict()
    for x in range(n1,n2+1):
        d[x]=x**2
    print(d)



n1 = int(input("Enter n1 :"))


n2 = int(input("Enter n2 :"))
fun1(n1,n2)
```

I/O : Enter n1 :2

Enter n2 :7

{2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49}

Q 9 a: The MVC(Model View, Controllers) is a GUI design method that helps separate the parts of an application, which will make the application easier to understand and modify. The main goal of this design is to keep the representation of the data separate from the parts of the program that the user interacts with; that way, it is easier to make changes to the GUI code without affecting the code that manipulates the data.

A GUI program under MVC consists of three parts:

i) View : Component that displays information to the user, e.g. Label, Entry(also accept input). But they do not do processing or storage.
ii) Models: store data, e.g. piece of text or the cost of an object etc. They also don't do computations but keep track of the application's current state and to save that state to a file or database and reload it later). E.g. counter variable keeps track of how many times a button is clicked
iii) Controllers : convert user input into calls on functions which manipulate the data in model. Controllers may update an application's models, which in turn can trigger changes to its views.Example the function registered to be triggered on click of a button.

For example the following piece of code:

```
import tkinter
# The controller.
def click():
counter.set(counter.get() + 1)
if __name__ == '__main__':
window = tkinter.Tk()
# The model.
counter = tkinter.IntVar()
counter.set(0)
# The views.
frame = tkinter.Frame(window)
frame.pack()
button = tkinter.Button(frame, text='Click', command=click)
button.pack()
label = tkinter.Label(frame, textvariable=counter)
label.pack()
# Start the machinery!
window.mainloop()
```

Here the model is kept track of by variable counter, which refers to an IntVar so that the view will update itself automatically. The controller is function click, which updates the model whenever a button is clicked. Four objects make up the view: the root window, a Frame, a Label that shows the current value of counter, and a button that the user can click to increment the counter's value.

Q9 b:

```
class Square:
    """ A class of Python object that describes the properties of a rectangle"""
    def __init__(self, side):
        self.side = side
```

```
    def compute_perimeter(self):
        return 4*self.side


    def compute_area(self):
        return self.side * self.side


S1= Square(3)
print("Area is:", S1.compute_area())
print("Perimeter is:", S1.compute_perimeter())
```

I/O : Area is: 9

Perimeter is: 12

Q 10 a: A tkinter program is a collection of widgets along with their GUI styles and their layout.

Some of the widgets available with tkinter are

i) Button : A clickable button
ii) Checkbutton : A clickable box that can be selected or unselected
iii) Entry: A single-line text field that the user can type in
iv) Frame :A container for widgets
v) Label : A single-line display for text
vi) Menu : A drop-down menu
vii) Text : A multiline text field that the user can type in

Label

Labels are widgets that are used to display short pieces of text. Here we create a Label that belongs to the root window—its *parent widget*—and we specify the text to be displayed by assigning it to the Label's text parameter.The format for creating a label is

label = tkinter.Label(<<parent>>, text=<<*Text to be displayed in label*>>)

where <<parent>> is the container in which to put the label.

Frame

As described in Q3

Entry

Entry is a widget which let users enter a single line of text. If we associate a StringVar with the Entry, then whenever a user types anything into that Entry, the StringVar's value will automatically be updated to the contents of the Entry.

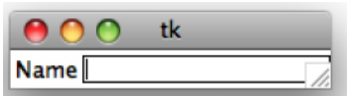
The format for creating an Entry is

entry = tkinter.Entry(<<parent>>, textvariable=<<variable name>>)

The below example covers label and Entry:

```
from Tkinter import *

window = Tk()

frame = Frame(window)

frame.pack()

label = Label(frame, text="Name")

label.pack(side="left")

entry = Entry(frame)

entry.pack(side="left")

window.mainloop()
```

Output:



Button

Button is a clickable widget with which can act as a trigger when clicked. The format for creating a button is :

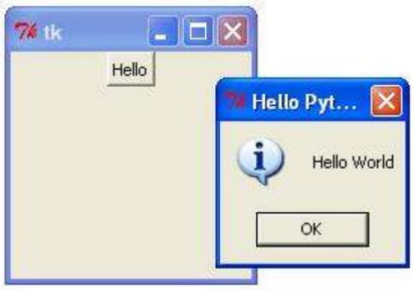
button = tkinter.Button(<<parent>>, text=<<text to be displayed on the button>>, command=<<Name of function to be called when button is clicked>>)

The third, command=<<function>>, tells it to call function <<function>> each time the user presses the button. This makes use of the fact that in Python a function is just another kind of object and can be passed as an argument like anything else.

For example the following code

```
import Tkinter
import tkMessageBox

top = Tkinter.Tk()

def helloCallBack():
   tkMessageBox.showinfo( "Hello Python", "Hello World")

B = Tkinter.Button(top, text ="Hello", command = helloCallBack)

B.pack()
top.mainloop()
```

Output:



Text

Text is a widget which is used to take multiple lines of text as input. The format of for creation of Text widget is

text = tkinter.Text(<<parent>>, height=<<h>>, width=<<w>>)

where <<parent>> is the parent frame/window, <<h>> is the number of rows and <<w>> is the number of columns.

The insert method of Text allows to enter text at the end of the text area. The format is:

text.insert(tkinter.INSERT, <<text to be inserted>>)

Text provides a much richer set of methods than the other widgets. We can embed images in the text area, put in tags, select particular lines, and so on.

For example

```
from Tkinter import *


root = Tk()

T = Text(root, height=2, width=30)

T.pack()

T.insert(END, "Just a text Widget\nin two lines\n")

mainloop()
```

The output would be

Checkbuttons:

Checkbuttons/*checkboxes*, have two states: on and off. When a user clicks a checkbutton, the state changes. We can use tkinter mutable variable to keep track of the user's selection. An IntVar variable can be used and the values 1 and 0 indicate on and off, respectively.

```
from Tkinter import *

master = Tk()

var = IntVar()

c = Checkbutton(master, text="Expand", variable=var)
c.pack()

mainloop()
```

In the above program a checkbutton 'c' is created and put in the master window and an Intvar 'var' is associated with the current state of the checkbutton.

Menu

This widget is used to display all kinds of menus used by an application. Toplevel menus are displayed just under the title bar of the root or any other toplevel windows. To create a toplevel menu, create a new Menu instance, and use **add** methods to add commands and other menu entries to it.

```
from Tkinter import *


def first():
    print "First"


def second():
```

```
    print "Second"


window=Tk()
menubar1=Menu(window)
menubar=Menu(window)
menubar.add_command(label='First',command=first)
menubar.add_command(label='Second',command=second)
menubar1.add_cascade(label='File',menu=menubar)
window.config(menu=menubar1)


window.mainloop()
```

In the above program two menu objects are created - menubar and menubar1. Items are added to the menu using the add_command method. The first argument specifies the label to be displayed and the second specifies the function that needs to be invoked on clicking on the menu option. 'menubar' object is added as a submenu of 'menubar1' using the add_cascade method invocation. The line ' window.config(menu=menubar1)' specifies that menunar1 is the main menu for the window.

Q 10 b:

```
from tkinter import *
window = Tk()
# The model.
var = StringVar()
# General controller
def click1(text):
    var1=text.get()
    label = Label(frame, text=var1)
    label.pack()
# The views.
frame = Frame(window)
frame.pack()
```

```
text= Entry(frame )
text.pack()
button = Button(frame, text="Save", command=lambda: click1(text))
button.pack()



window.mainloop()
```