CMR INSTITUTE OF TECHNOLOGY		USN							
Internal Assesment Test - I									
Sub:	Programming Using Python							Code:	18MCA32
Date:	6.09.2019	Duration:	90 mins	Max Marks:	50	Sem:	III	Branch:	MCA

Answer Any One FULL Question from each part.

			Marks	OBE	
				CO	RBT
Part - I					
1	(a)	Explain any 5 string functions with examples.	[10]	CO2	L2
OR					
2	(a)	Input an array of n numbers and find separately the sum of positive numbers and negative numbers.	[10]	CO2	L3
Part – II					
3	(a)	Explain and construct the memory model of variables in Python.	[5]	CO1	L2
	(b)	Write functions to convert from Celsius to Faranheit and vice versa where the formula connecting them is F=C*1.8+32	[5]	CO1	L3
OR					
4	(a)	Discuss the usage of the following with respect to the print() function: i) sep argument ii) end argument iii) .format(arguments)	[6]	CO1	L2
	(b)	Write a function for finding simple interest and amount.	[4]	CO1	L3

CMR INSTITUTE OF TECHNOLOGY		USN							
Internal Assesment Test - I									
Sub:	Programming Using Java							Code:	18MCA32
Date:	6.09.2019	Duration:	90 mins	Max Marks:	50	Sem:	III	Branch:	MCA

Answer Any One FULL Question from each part.

			Marks	OBE	
				CO	RBT
Part - I					
1	(a)	Explain any 5 string functions with examples.	[10]	CO2	L2
OR					
2	(a)	Input an array of n numbers and find separately the sum of positive numbers and negative numbers.	[10]	CO2	L3
Part – II			**Part – II**		
3	(a)	Explain and construct the memory model of variables in Python.	[5]	CO1	L2
	(b)	Write functions to convert from Celsius to Faranheit and vice versa where the formula connecting them is F=C*1.8+32	[5]	CO1	L3
OR					
4	(a)	Discuss the usage of the following with respect to the print() function: i) sep argument ii) end argument iii) .format(arguments)	[6]	CO1	L2
	(b)	Write a function for finding simple interest and amount.	[4]	CO1	L3

	PART - III			
5 (a)	Describe briefly the process of designing your own modules with clear example.	[5]	CO4	L2
(b)	Write a piece of Python code to find the largest of three numbers.	[5]	CO4	L2
	OR			

6 (a)	Trace the function call and explain the memory model of the following code:
	def f(x):
	x=2*x
	return x
	x=1
	x=f(x+1)+f(x+2)
		[10]	CO2	L4

	Part – IV			
7 (a)	Write a program to search an element using linear search.	[10]	CO2	L2
	OR			
8 (a)	Explain the two ways to use python interpreter. What are the errors that can be detected by Python? Differentiate between them with one example each. Give output of the following expressions: i) 54/17 ii) -17/10 iii) -16%5 iv) 17%-10	[10]	CO5 CO1	L3
	Part – V			
9 (a)	What are the two ways of importing a module? Which one is more beneficial? Explain.	[4]	CO2	L2
(b)	Write a function to find roots of a quadratic equation. Take care of all cases	[6]	CO1	L3
	OR			
10(a)	Explain how code in Python is tested semi-automatically.	[5]	CO5	L3
(b)	Write a program to sum all the elements from n1 to n2 where n1 and n2 are positive integers.	[5]	CO2	L2

	PART - III			
5 (a)	Describe briefly the process of designing your own modules with clear example.	[5]	CO4	L2
(b)	Write a piece of Python code to find the largest of three numbers.	[5]	CO4	L2
	OR			

6 (a)	. Trace the function call and explain the memory model of the following code:
	def f(x):
	x=2*x
	return x
	x=1
	x=f(x+1)+f(x+2)
		[10]	CO2	L4

	Part – IV			
7 (a)	Write a program to search an element using linear search.	[10]	CO2	L2
	OR			
8 (a)	Explain the two ways to use python interpreter. What are the errors that can be detected by Python? Differentiate between them with one example each. Give output of the following expressions: i) 54/17 ii) -17/10 iii) -16%5 iv) 17%-10	[10]	CO5 CO1	L3
	Part – V			
9 (a)	What are the two ways of importing a module? Which one is more beneficial? Explain.	[4]	CO2	L2
(b)	Write a function to find roots of a quadratic equation. Take care of all cases	[6]	CO1	L3
	OR			
10(a)	Explain how code in Python is tested semi-automatically.	[5]	CO5	L3
(b)	Write a program to sum all the elements from n1 to n2 where n1 and n2 are positive integers.	[5]	CO2	L2

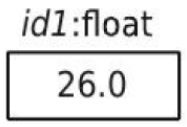
Q1.(a) Explain any 5 string functions with examples.
Sol:

```
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 '}').
```

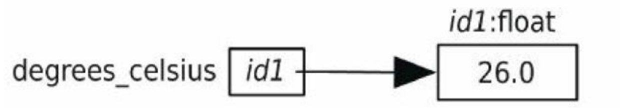
Q 2 (a) Input an array of n numbers and find separately the sum of positive numbers and negative numbers.

Sol:
```
#Input an array of n numbers and find separately the sum of positive numbers and negative numbers
int_arr =list()
totnum=input("Enter how many elements you want : ")
print 'Enter the numbers in array : '

for i in range(int(totnum)):
  n=input("Number : ")
  int_arr.append(int(n))

print 'Entered Array : ', int_arr
pos_sum=0
neg_sum=0
for i in int_arr:
  if i>0:
      pos_sum=pos_sum + i
  else:
      neg_sum=neg_sum + i
print ("Sum of Positive numbers :  "+str(pos_sum))
print ("Sum of Negative numbers :  "+str(neg_sum))
```

Q3 (a). Explain and construct the memory model of variables in Python.
Sol:
Sol: Evrey location in the computers memory has a memory address that uniquely identifies its location. Note : we name memory location by prefixing id . An example of amemory cell storing a float is shown below:



This picture shows that the value 26.0 is at the memory address id1. The box would be called an object: a value at a memory address with a type. During execution of a program every value that Python keeps track of is stored inside an object in computer memory.
In Python's memory model a variable contains the memory address of the object to which it refers



To explain the above picture the following terminology is used:

- Value 26.0 has the memory address id1.
- The object at the memory address id1 has type float and the value 26.0.
- Variable degree_celsius contains the memory address id1
- Variable degree_celsius refers to the value 26.0

Whenever Python needs to know which value degree_celsius refers to, it looks at the object at the memory address that degree_celsius contains. In this example, that memory address is id1, so Python will use the value at the memory address is1 which is 26.0.

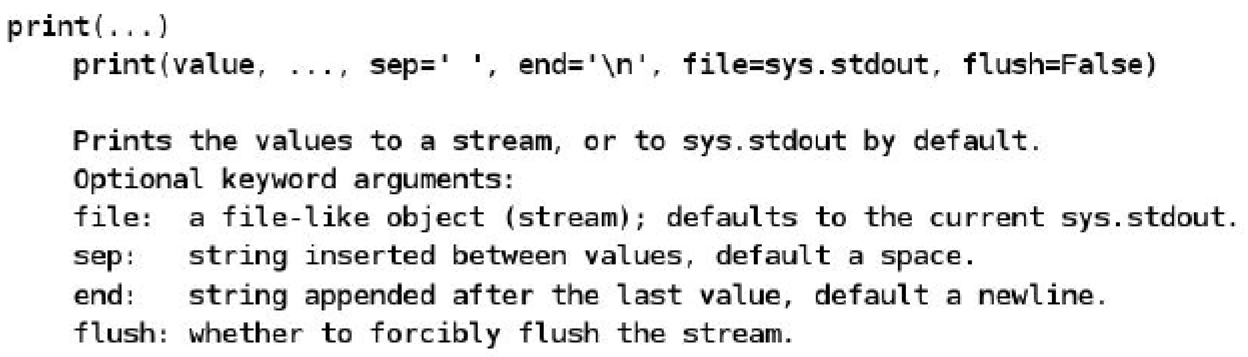
Q 3(b) Write functions to convert from Celsius to Faranheit and vice versa where the formula connecting them is F=C*1.8+32
Sol:

```
def convertCtoF(C):
        '''
                number -->number

                Converts a temperature given in Celsius to Faranheit

                >>> convertCtoF(5)
                41.0
                >>> convertCtoF(1)
                33.8
        '''
        return C*1.8+32

def convertFtoC(F):
        '''
```

```
            number -->number

            Converts a temperature given in Celsius to Faranheit

            >>> convertFtoC(32)
            0.0
            >>> round(convertFtoC(33.8))
            1.0
    '''
    return (F-32)/1.8
```

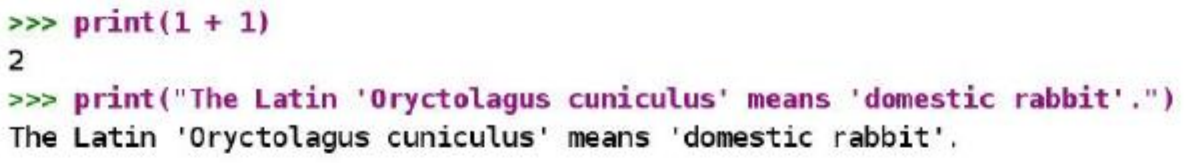
Q4.(a) Discuss the usage of the following with respect to the print() function: i) sep argument ii) end argument iii) .format(arguments)

Sol:



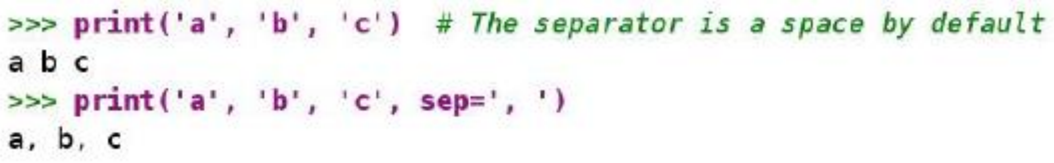
```
print(...)
    print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)

    Prints the values to a stream, or to sys.stdout by default.
    Optional keyword arguments:
    file:  a file-like object (stream); defaults to the current sys.stdout.
    sep:   string inserted between values, default a space.
    end:   string appended after the last value, default a newline.
    flush: whether to forcibly flush the stream.
```

Print function is used to output a value /expression on the screen. E.g.



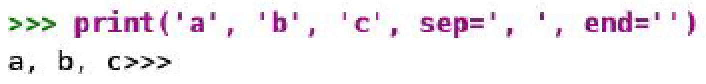
```
>>> print(1 + 1)
2
>>> print("The Latin 'Oryctolagus cuniculus' means 'domestic rabbit'.")
The Latin 'Oryctolagus cuniculus' means 'domestic rabbit'.
```

"sep" is used to specify the separators between different elements when a list is printed. The default value is a space. example



```
>>> print('a', 'b', 'c')  # The separator is a space by default
a b c
>>> print('a', 'b', 'c', sep=', ')
a, b, c
```

"end" is used to specify the character to be printed at the end of list. The default value is newline. Example



```
>>> print('a', 'b', 'c', sep=', ', end='')
a, b, c>>>
```

```
 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 '}').
```

Example:

```
>>> a=20
>>> b=30
>>> str.format("The value of a is {0}, b is {0} and their sum is {2}",a,b,a+b)
'The value of a is 20, b is 20 and their sum is 50'
```

Q4 (b) Write a function for finding simple interest and amount

Sol:

```
def simpleInterest(p, r,t):
	'''
		(number, number, number) --> (number, number)

		Returns the Simple Interest and Amount given the Principal, Rate of Interest and Time Period

		>>> simpleInterest(1000,10,3)
		(300, 1300)
		>>> simpleInterest(500,2.5,4.5)
		(300, 1300)

		>>> simpleInterest(500,2.5,4.5)
		(56.25, 556.25)
	'''
	si=p*r*t/100
	a=p+si
	return (si,a)
```

Q5(a): Describe briefly the process of designing your own modules with clear example.

Sol: 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
```

Q5(b) Write a piece of Python code to find the largest of three numbers

Sol:

```
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
```

Q6(a) Trace the function call and explain the memory model of the following code:

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

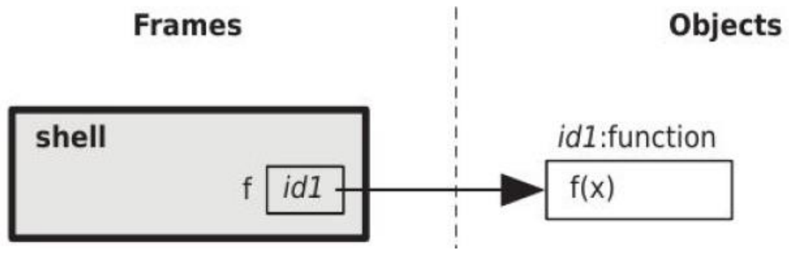
Sol:

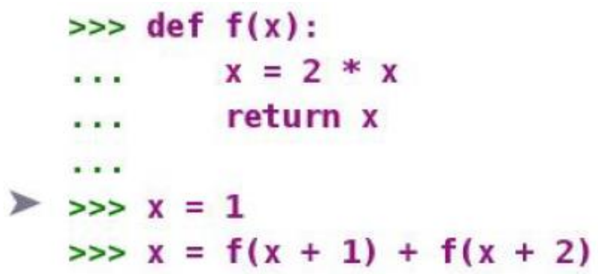
Whenever Python executes a function call it creates a amespace in which to store local variables for theat call. Python also another namespace for variables created in the shell.

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

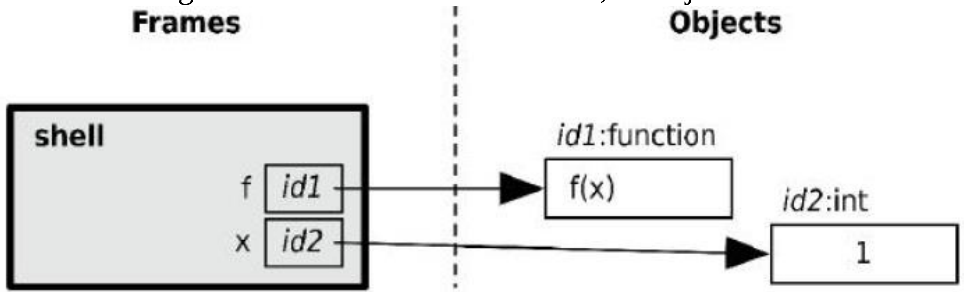
When it encounters the first line it creates a variable f in the shell frame and a function object.



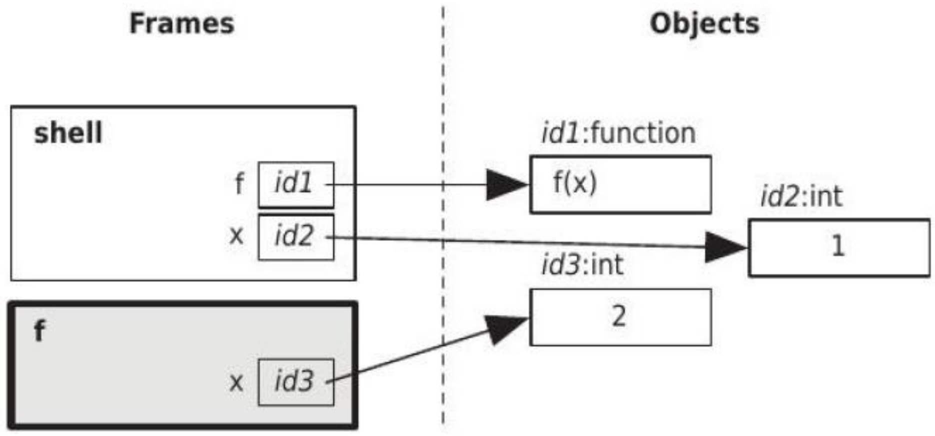


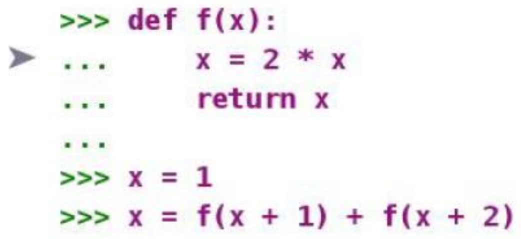
```
    >>> def f(x):
    ...     x = 2 * x
    ...     return x
    ...
➤   >>> x = 1
    >>> x = f(x + 1) + f(x + 2)
```

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



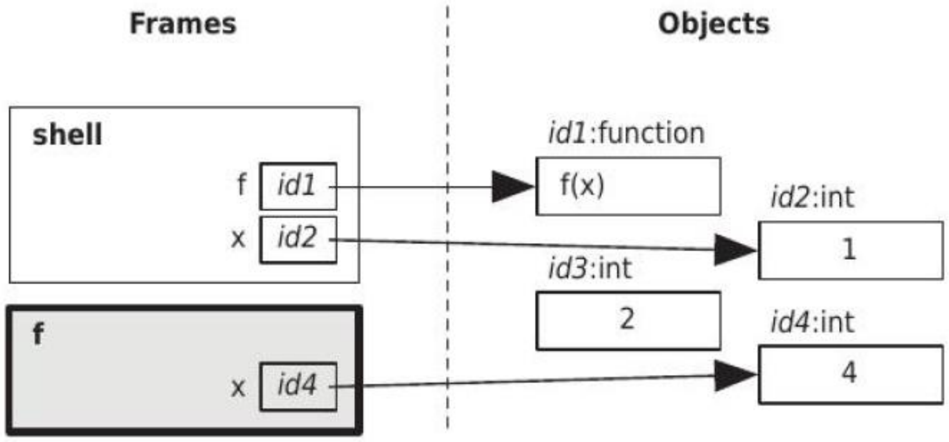
Following python rules x+1 is first evaluated to 2. The next step is to createa a namespace for the function call f as shown below



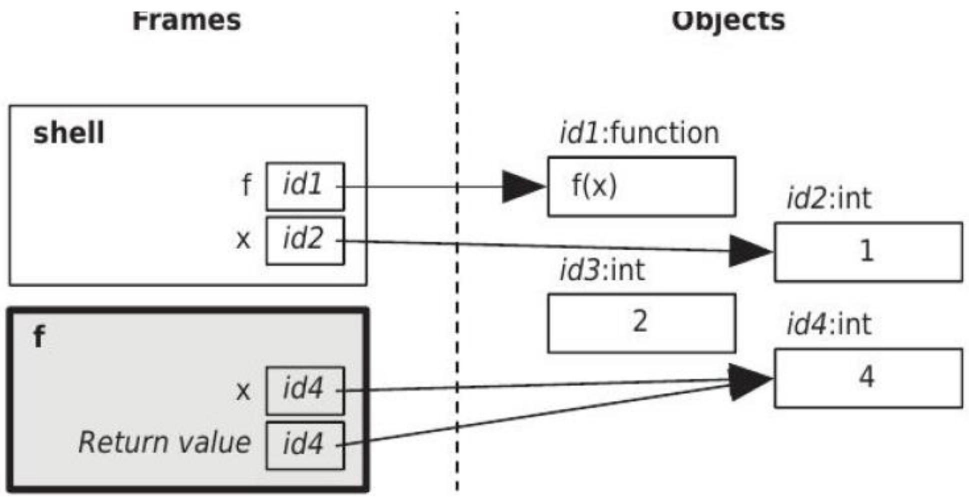


```
    >>> def f(x):
➤   ...     x = 2 * x
    ...     return x
    ...
    >>> x = 1
    >>> x = f(x + 1) + f(x + 2)
```

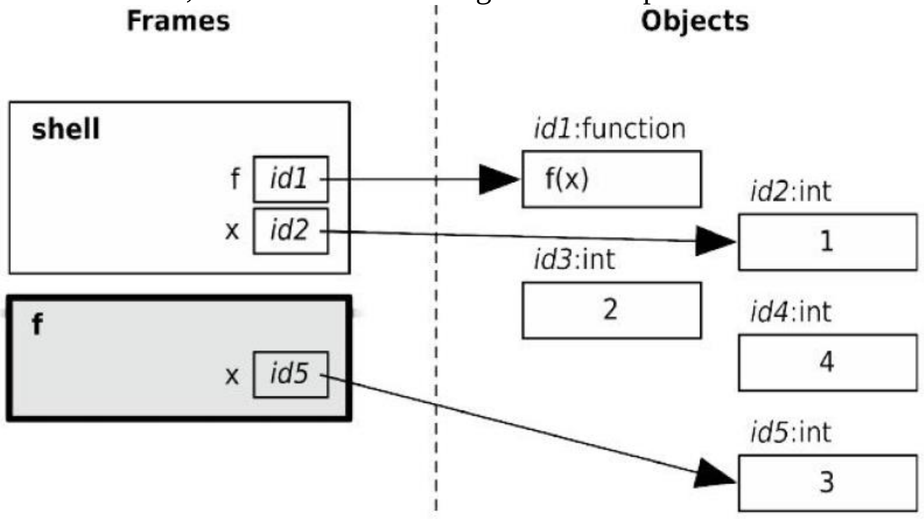
The first statement of the function f is executed to give a value of x as 2*2=4.



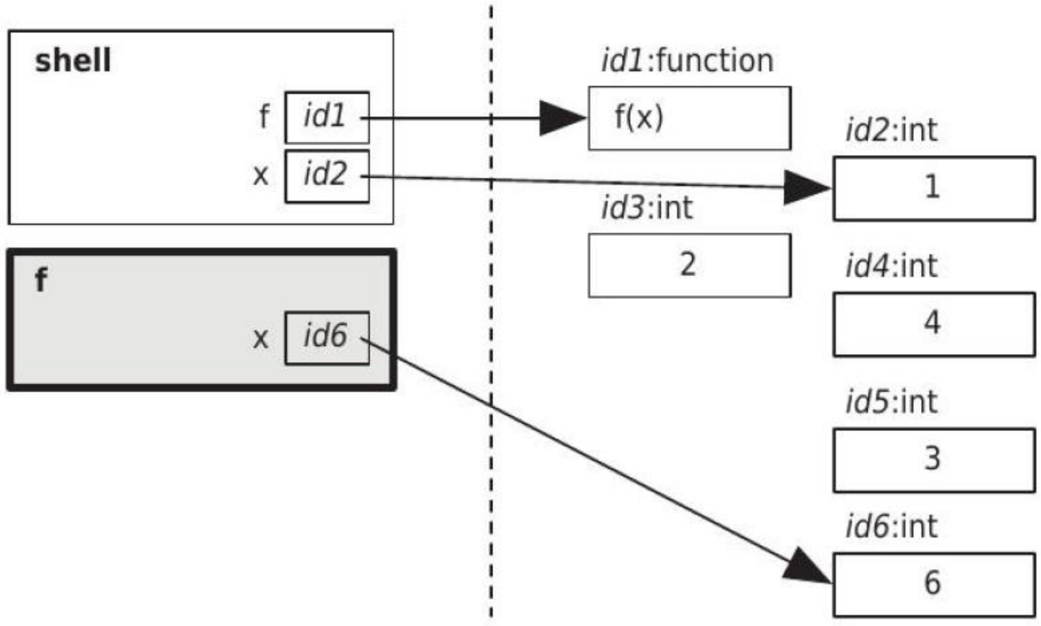
The value 4 is returned as the second statement of the function



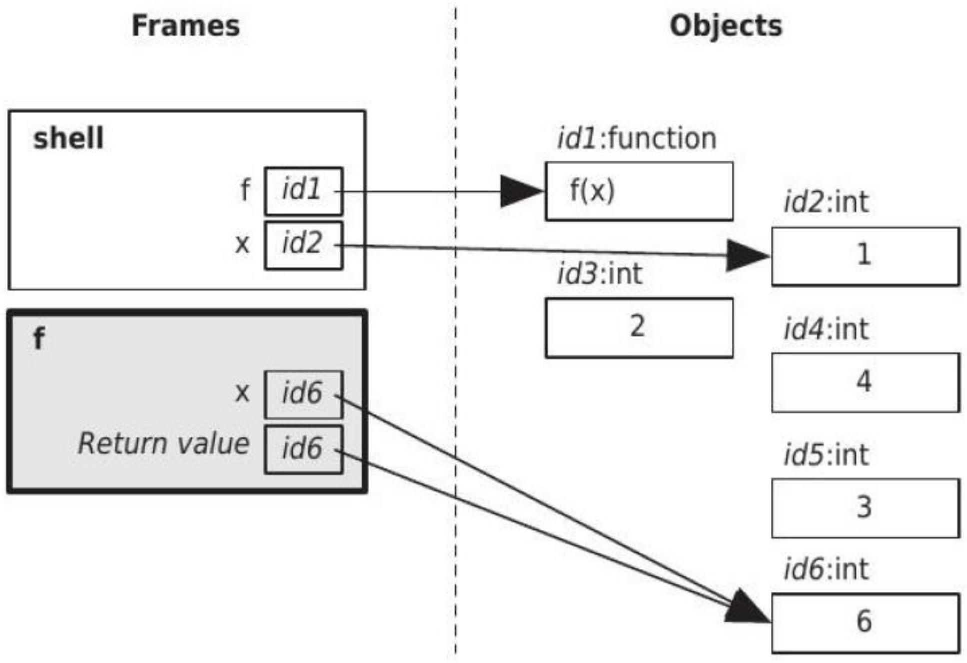
When the function returns Python now executes the right function call f(x+2). According to the rules x+2 is first evaluated , x+2 evaluates to 3. Again a namespace for the function call is created.



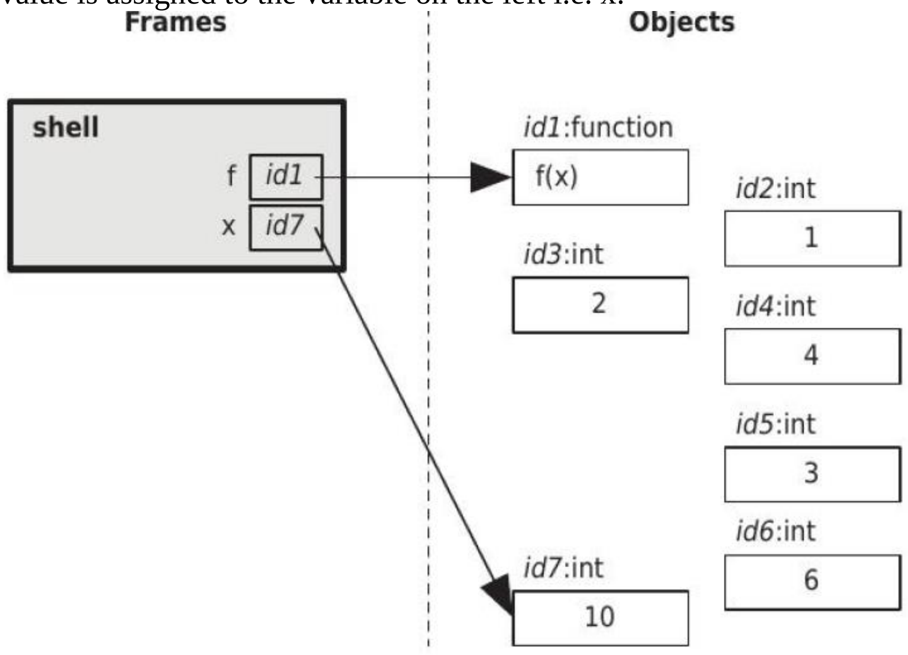
In the function the first statement is executes x evaluates to 2*3=6.



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.



Q7(a) Write a program to search an element using linear search.

Sol: #Write a program to search an element using linear search

```
int_arr =list()


totnum=input("Enter how many elements you want : ")

print 'Enter the numbers in array : '



for i in range(int(totnum)):

  n=input("Number : ")
  int_arr.append(int(n))


print 'Entered Array : ', int_arr


key=input("Enter the key value to be searched :")


flag=-1

for i in int_arr:

    if i==key:

       pos=int_arr.index(i)

       flag=1

       break



if flag==1:

   print 'Key found at position : ' , pos+1

else:

   print 'Key not found'
```

Q8(a) Explain the two ways to use python interpreter. What are the errors that can be detected by Python? Differentiate between them with one example each. Give output of the following expressions:
i) 54/17   ii) -17/10   iii) -16%5   iv) 17%-10

Sol: 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

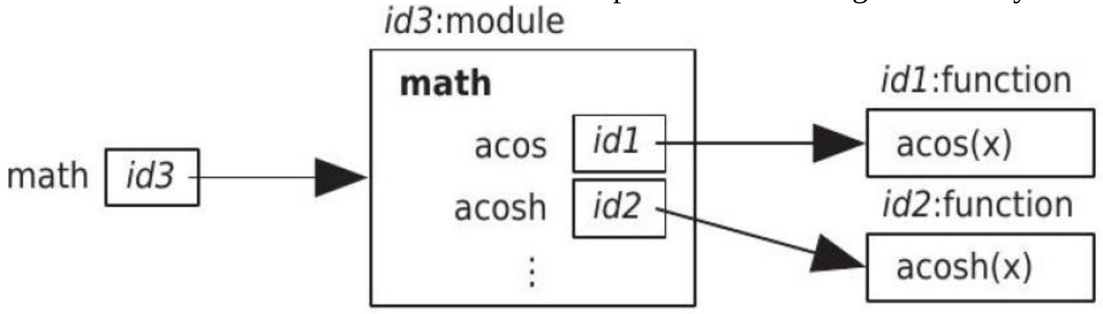
54/17 - 3,   -17/10 - -1.7,   -16%5 = 4,   17%-10 = -3

Q9(a) What are the two ways of importing a module? Which one is more beneficial? Explain.

SOl:
The two ways to import a module is to import the entire module using the statement:

import <module_name>
e.g. import math

In this case all the functions in the module are imported and are assigned memory.



A module variable math is created which points to module structure containg addresses of all the functions in the module. as shown in the above diagram. To invoke a function we use the format
<module name>.<function name>(args)
.E.g. math.sqrt(5).

Another way to import are specific functions and variables from a module using the format:
from <module name> import fn1,fn2
E.g. from math import math, pi

In such a case only the functions sqrt and variable pi are imported from math. The other functions are not stored . A function call be invoked by just calling the function without prepending the module name
e.g. sqrt(9)

Usually the second method is preferred since it only imports the functions and variables required. The first method causes all the methods to occupy memory irrespective of whether they are used or not.

Q9(b) Write a function to find roots of a quadratic equation. Take care of all cases.

Sol: # import complex math module

```
import cmath

import math



a = int(input("Enter the coefficients of a: "))

b = int(input("Enter the coefficients of b: "))

c = int(input("Enter the coefficients of c: "))



d = b**2-4*a*c # discriminant



if d < 0:

    print ("This equation has no real solution")

    x1 = (-b-cmath.sqrt(d))/(2*a)
    x2 = (-b+cmath.sqrt(d))/(2*a)

    print('The solution are {0} and {1}'.format(x1,x2))

elif d == 0:

    x = (-b+math.sqrt(d))/2*a

    print (("This equation has one solutions: "), x)

else:

    x1 = (-b+math.sqrt(d))/(2*a)

    x2 = (-b-math.sqrt(d))/(2*a)

    print ("This equation has two solutions: ", x1, " or", x2)
```

Q10(a)Explain how code in Python is tested semi-automatically.

Sol:

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.

Q10(b) Write a program to sum all the elements from n1 to n2 where n1 and n2 are positive integers.

Sol:

```
#prg to print sum of n1 to n2

n1 = input("Enter n1 :")

n2 = input("Enter n2 :")


sum=0

i=n1


while i<=n2 :

  sum = sum+i

  i=i+1


print ("Sum of Numbers from "+str(n1)+" to "+str(n2)+" = "+str(sum))
```