Unit III

Conditionals: Boolean values and operators, conditional (if), alternative (if-else), chained conditional (if-elif-else); Iteration: state, while, for, break, continue, pass; Fruitful functions: return values, parameters, local and global scope, function composition, recursion; Strings: string slices, immutability, string functions and methods, string module; Lists as arrays.

APEC

Boolean Values and Operators

- The Boolean data type is a data type, having two values : true or false.
- When converting:
  - bool to an int, the integer value is always 0 or 1
  - int to a bool, the boolean value is True for all integers except 0.

- **Boolean and logical operators**

  Boolean values respond to logical operators and / or
  - True and False = False
  - True and True = True
  - False and True = False
  - False or True = True
  - False or False = False

APEC

Conditional Statements in Python

- **Conditional statements** ➔ Used to perform different computations or actions depending on whether a condition evaluates to true or false.
  - Condition uses comparisons and arithmetic expressions with variables, evaluated to the Boolean values True or False.

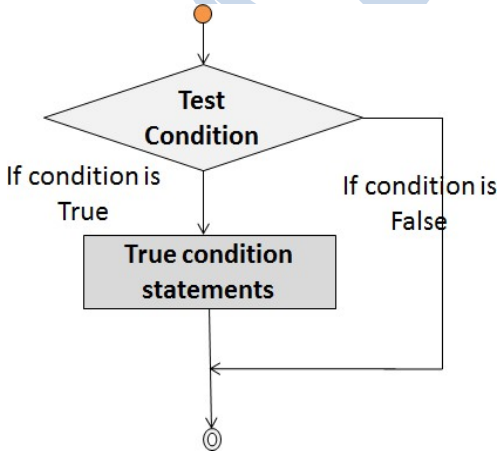


Fig: Structure of conditional statement

Conditional Statements (Cont….)

- Types of conditional checking statements in python:
  - **if statements**
  - **if…else statements**
  - **if…elif…else statements(or) chained conditionals**
  - **Nested if statements**

- ***Rules*:**
  - The colon (:) is required at the end of the condition.
  - The body of the if statement is indicated by the indentation. In Python, four spaces are used for indenting.
  - All lines indented the same amount after the colon will be executed for the true condition.
  - Python interprets non-zero values as True. None and 0 are interpreted as False.

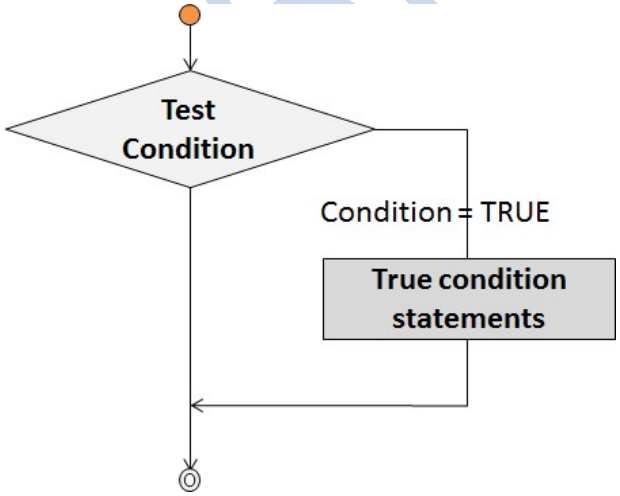
APEC

Conditional Statements (Cont....)

- ***if statements:***
  - The statement is executed only if the condition is *true,* if false then not executed.
  - ***Syntax:***

```
if condition:
    True block statement
```

  - ***Flowchart for if statement:***



Conditional Statements (Cont....)

- ***if statements (cont..)***

Example program for if statement:

Program :

```
1. age = input("What's your age = ")
2. if ( age >= 60 ) :
3.     print "You are a Senior Citizen"
4. print "Good bye!"
```

Execution

```
sh-4.3$ python program1.py
What's your age = 68
You are a Senior Citizen
Good bye!
```

APEC

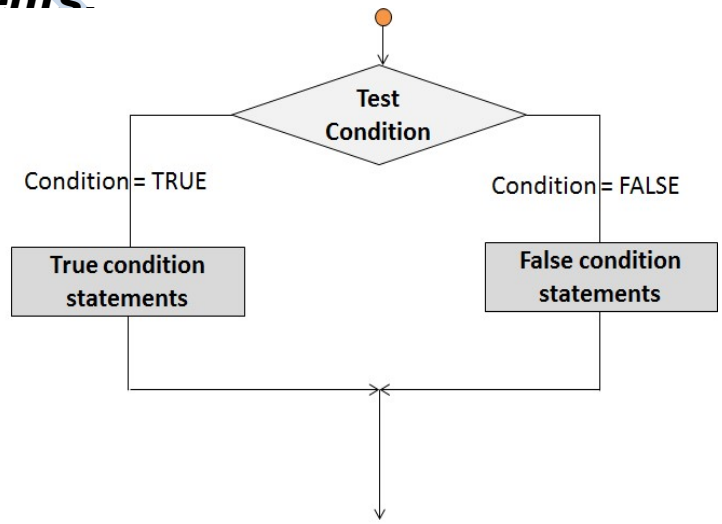
Conditional Statements (Cont....)

- ***if...else statements:***
  - Executes the true statement block only when test condition is True.
  - If False, then false statement block is executed.
  - Indentation is used to separate the blocks.

➢ **Syntax:**

```
if condition:
        True statement block.
else:
        False statement block.
```

➢ ***Flowchart for if...else statements:***



Conditional Statements (Cont….)

- ***if…else statements (cont..)***

Example program for if…else statement

Program :

```
1.  age = input("What's your age = ")
2.  if ( age >= 60 ) :
3.      print "Sorry! You are a Senior Citizen"
4.      print "Good bye!"
5.  else:
6.      print "You are selected"
7.  print "Have a great day!"
```

Execution

```
sh-4.3$ python program2.py
What's your age = 45
You are selected
Have a great day!
sh-4.3$ python program2.py
What's your age = 63
Sorry! You are a Senior Citizen
Good Bye!!
```

APEC

Conditional Statements (Cont....)

- ***Ternary operator: C and Python***
  - The ternary operator(? : ;) in C takes three arguments. The first argument is a comparison argument, the second is the result upon a true comparison, and the third is the result upon a false comparison.
  - **Syntax in C:** ***if condition ? value_if_true : value_if_false;***
  - Example : max = (a > b) ? a : b;
  - In Python, the syntax for ternary operator is

Syntax 1: value_if_true if_condition value_if_false

Syntax 2: (value_if_true , value_if_false) [if_condition]

  - ***Example***: *max = a if (a > b) else b or max = (a,b)[(a > b)]*

Conditional Statements (Cont….)

- ***Ternary operator: C and Python (cont..)***

Program :

```
1. a = raw_input("Enter a = ")
2. b = raw_input("Enter b = ")
3.  max = (a,b)[(a > b)]
4. max = a if (a > b) else b# work in some compilers
   only
5. print "Maximum of two numbers = ", max
```

Execution

```
sh-4.3$ python program4.py
Enter a = 89
Enter b = 568
Maximum of two numbers =  568
 Enter a = 56
Enter b = 41
Maximum of two numbers =  41
```

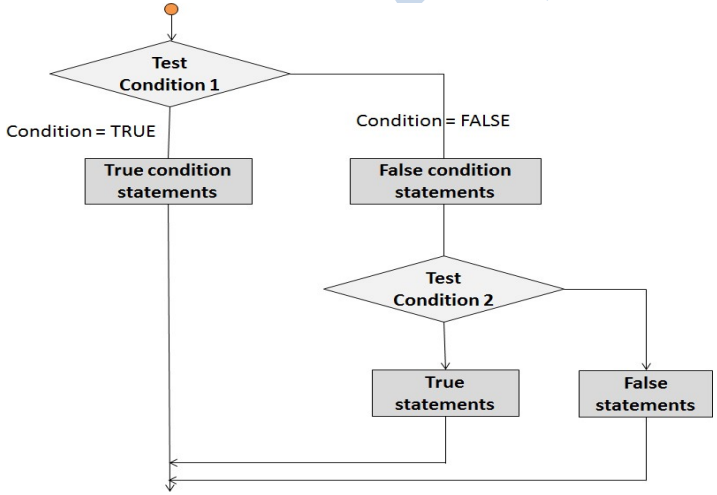
APEC

Conditional Statements (Cont....)

- ***if...elif...else (or) chained conditionals***
- The elif is short form for else if, used to check multiple conditions.
- If the condition 1 is false, it checks the condition 2 of the next elif block and so on.
- If all the conditions are False, then the else statement is executed.
- ***Syntax***:

```
if condition 1:
        True statement block for condition 1.
elif condition 2:
        True statement block for condition 2.
elif condition 3:
        True statement block for condition 3.
else:
        False statement block
```

- ***Flowchart for if...elif...else statements:***



Conditional Statements (Cont....)

- ***if...elif...else (or) chained conditionals (cont...)***

Example program for if...elif...else statement

Program:

```
1.  age = input("What's your age = ")
2.   if ( age >= 60 ) :
3.  print "Sorry! You are a Senior Citizen"
4.  print "Good bye!"
5.  elif (age < 60 and age > 40 ):
6.      print "You are selected for the post of Senior Manager"
7.  print "Have a great day!"
8.  elif (age < 40 and age > 20 ):
9.      print "You are selected for the post of Junior Manager"
10. print "Have a great day!"
11. else:
12. print("Sorry! you cannot apply")
```

Execution

```
sh-4.3$ python program5.py
What's your age = 72
Sorry! You are a Senior Citizen
Good bye!
 What's your age = 55
You are selected for the post of Senior Manager
Have a great day!
 What's your age = 18
Sorry! you cannot apply
```

APEC

Conditional Statements (Cont....)

- ***Nested if statements:***

➢ if...elif...else statement can be used inside another if...elif...else statement. This is called nesting in computer programming.

➢ **Example program for Nested if statements:**

Program :

```
#check whether the given number is positive, negative or zero
number = input("Enter a number: ")
if number >= 0:
    if number == 0:
        print("Zero")
    else:
        print("Positive number")
else:
    print("Negative number")
```

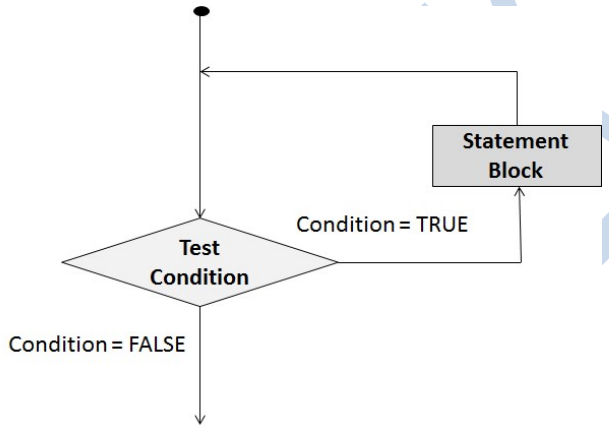
Execution

```
sh-4.3$ python program6.py
Enter a number: 8
Positive number
Enter a number: -9
Negative number
Enter a number: 0
Zero
```

APEC

Iteration (or) Looping

- Repeated execution of a set of statements is called iteration or looping.
- ***Flowchart for looping statements:***



- Types of iterative statements:
  - *The for loop*
  - *The while statement*
  - *Nested loops*

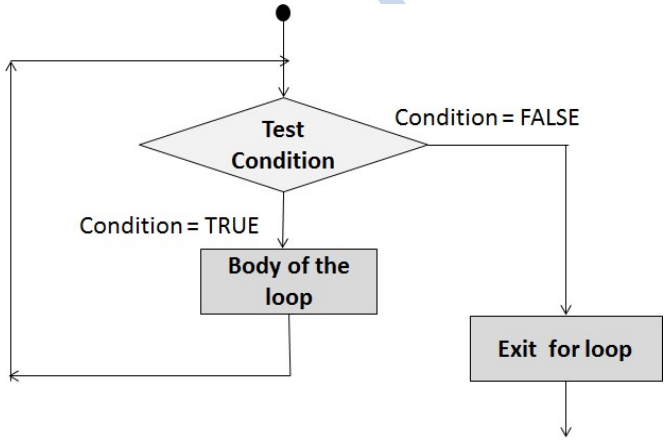
Iteration (or) Looping (cont..)

- ***The for loop:***

- For loop in Python repeats a group of statements for a specified number of times.
- For loop in python starts with the keyword "for" followed by an arbitrary variable name, which holds its values in the following sequence object.
- The else block will be executed only if the for loop hasn't been broken by a "break" statement.
- ***Syntax:***

```
for<loop_variable> in <sequence>:
<statement or statement block)>
else
    <statement or statement block>
```

- ***Flowchart for FOR loop:***



Iteration (or) Looping (cont..)

- ***The for loop(cont..)***

Example program for For loop:

Program 7:

```
1.# simple program to explain for loop
2.for i in '123':
3.        print "Welcome",i,"times"
```

Execution

```
sh-4.3$ python program7.py
Welcome 1 times
Welcome 2 times
Welcome 3 times
```

APEC

Iteration (or) Looping (cont..)

- ***The range() function***
  - Sequence of numbers are also generated using range() function.
  - range() is defined with start_element, stop_element and step size.
  - Default step size is equal to 1 if not provided.
  - ***Syntax:***

```
range(start_element, stop_element, step size)
```

  - ***Example Program :***

Program :

```
# Display numbers from 0 to 10 (11 numbers)
numbers = range(11)
print(numbers)
 # Display numbers from 5 (start_element) to 10 (stop_element)
numbers = range(5,11)
print(numbers)
 # Display numbers from 1 (start_element) to
# 11 (stop_element) with step_size = 2
numbers = range(1,11,2)
print(numbers)
# Display numbers from 10 (start_element) to
# 0 (stop_element) with step_size = -1
numbers = range(10, 0, -1)
print(numbers)
```

Execution

```
sh-4.3$ python program9.py
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
[5, 6, 7, 8, 9, 10]
[1, 3, 5, 7, 9]
[10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
```

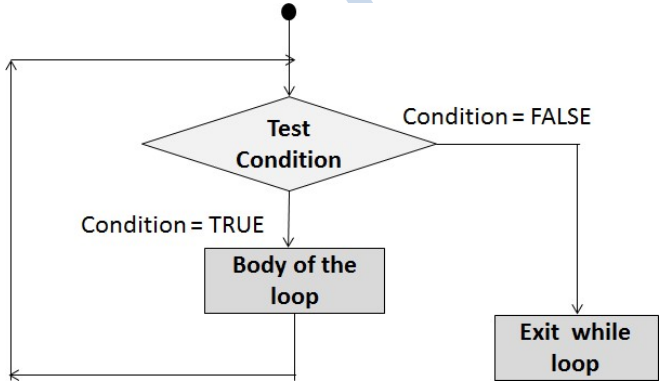
Iteration (or) Looping (cont..)

- ***The While loop:***

- The while loop iterates over a block of statements as long as the test condition is true.
- The statement body is entered only when the test condition is true. After first iteration, the test condition is checked again
- .In Python, the statement body is determined through indentation. Python interprets any non-zero value as True. None and 0 are interpreted as False.
- ***Syntax****:*

```
While condition:
    statement_1
    .....
statement_2
```

- ***Flowchart for While loop:***



Iteration (or) Looping (cont..)

- ***The While loop (cont..):***

Example program for While loop

Program 12:

```
1. # Simple While loop
2. a = 0
3. #Body of the loop is intended by four spaces
4. while a < 10:
5.     a = a + 1
6.     print a
```

Execution

```
sh-4.3$ python program12.py
1
2
3
4
5
6
7
8
9
10
```

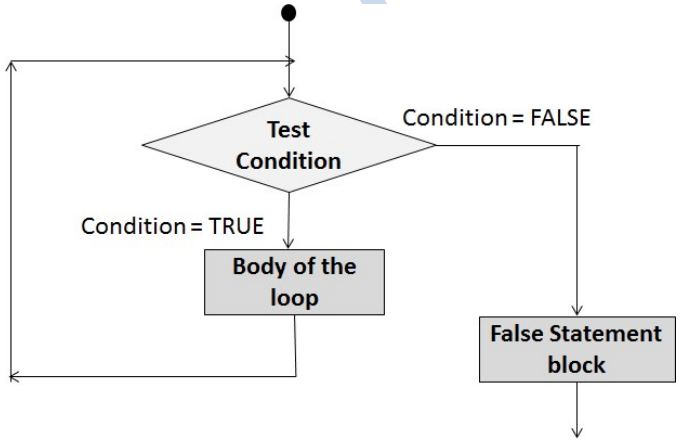
APEC

Iteration (or) Looping (cont..)

- ***While-Else loop:***
  - Python supports to have an else statement with a while loop statement.
  - The else statement is executed when the condition becomes false.
  - ***Syntax:***

```
While condition:
    statement_1
    ……….
    statement_n
Else:
    staement_1
    statement_n
```

  - ***Flowchart for While-Else loop:***



Iteration (or) Looping (cont..)

- ***While-Else loop (cont..)***

Example program for While-Else loop

Program :

```
1.#program for While-Else statement
2.count = 0
3.while count < 5:
4.    print count, " is  less than 5"
5.    count = count + 1
6.else:
7.    print count, " is not less than 5"
```

Execution

```
sh-4.3$ python program14.py
Apple
Asus
Dell
Samsung
```

APEC

Iteration (or) Looping (cont..)

- ***Nested loops statements:***

➢ Python programming language allows using one loop inside another loop.

➢ ***Syntax: Nested for loop***

```
for iterating _var  in sequence:
  for iterating _var  in sequence:
      statement(s)
  statement(s)
```

➢ ***Syntax : Nested while loop***

```
while expression:
  while expression:
      statement(s)
  statement(s)
```

➢ ***Example program for Nested loops***

Program :

```
#program to explainforloop
list1 = [["john", "Jerry"],[1, 2], ["Paris", "London"]]
for i in list1:
    print i
```

Execution

```
sh-4.3$ python program15.py
['john', 'Jerry']
[1, 2]
['Paris', 'London']
```

Iteration (or) Looping (cont..)

- ***Nested loops statements (cont..):***

Example program for Nested loops

Program:

```
1. #program to show nested loops –for  loop within
   anotherforloop
2. list1 = [["john", "Jerry"],[1, 2], ["Paris",
   "London"]]
3.  for i in list1:
4.      for x in i:
5.          print x
```

Execution

```
sh-4.3$ python program15.py
john
Jerry
1
2
Paris
London
```

APEC

Python Break, Continue And Pass Statements

- In some situations there is a need to exit the loop completely when an external condition is triggered or there may also be a situation to skip a part of the code and start next execution.
- Python provides following statements to handle these situations:
  - break
  - continue
  - pass

APEC

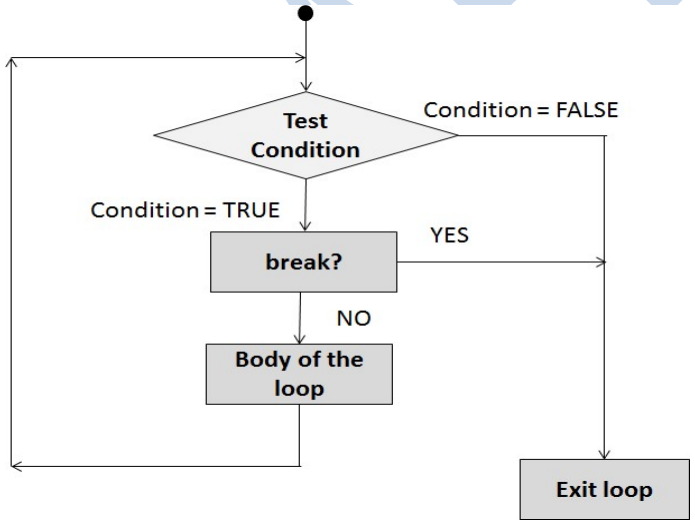
Python Break, Continue And Pass Statements (cont..)

- ***Break statement***

- The break statement in Python terminates the current loop and resumes execution at the next statement. It's just like the traditional break found in C.
- ***Syntax:***

```
break;
```

- ***Flowchart for break statement***



Python Break, Continue And Pass Statements (cont..)

- ***Break statement (cont..)***

Example program for break statement

Program :

```
1. #program to explain break statement
2. chocolates = 10
3. while (chocolates > 0):
4.     print "We have ", chocolates,"number of chocolates"
5.     chocolates=chocolates-1
6.     if chocolates == 5:
7. break
8. print('Alarm!!! Chocolate thief')
9. input('Press enter to exit')
```

Execution

```
sh-4.3$ python program17.py
We have  10 number of chocolates
We have  9 number of chocolates
We have  8 number of chocolates
We have  7 number of chocolates
We have  6 number of chocolates
Alarm!!! Chocolate thief
Press enter to exit
```

APEC

Python Break, Continue And Pass Statements (cont..)

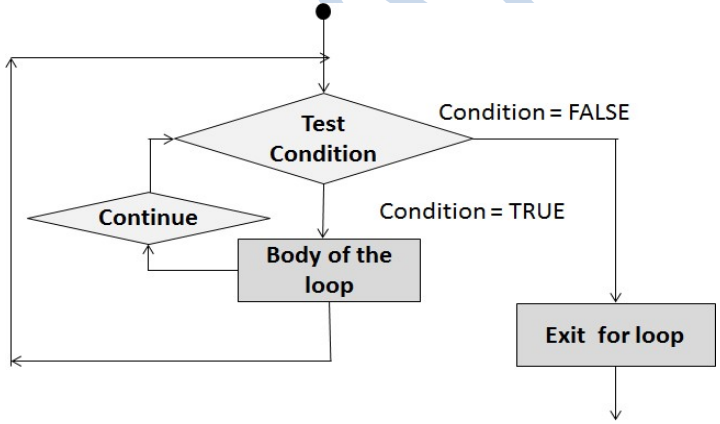
- ***Continue statement:***

➢ The continue statement rejects all the remaining statements in the current iteration of the loop and moves the control back to the top (begining)of the loop.

➢ ***Syntax:***

```
continue;
```

➢ ***Flowchart for break statement:***



Python Break, Continue And Pass Statements (cont..)

- ***Continue statement (cont..)***

Example program for continue statement

Program :

```
1.  #program to explain continue statement
2.   apples = 0
3.  oranges = 0
4.  while (oranges != -1):
5.      apples = input("Enter number of apples = ")
6.      oranges = input("Enter number of oranges = ")
7.      if (oranges == 0):
8.   continue
9.      print "Total fruits in the basket = ", apples + oranges
```

Execution

```
sh-4.3$ python program1.py
Enter number of apples = 2
Enter number of oranges = 3
Total fruits in the basket =  5
Enter number of apples = 2
Enter number of oranges = 0
Enter number of apples = 4
Enter number of oranges = 1
Total fruits in the basket =  5
Enter number of apples = 2
Enter number of oranges = -1
Total fruits in the basket =  1
```

APEC

Python Break, Continue And Pass Statements (cont..)

- ***Pass statement:***

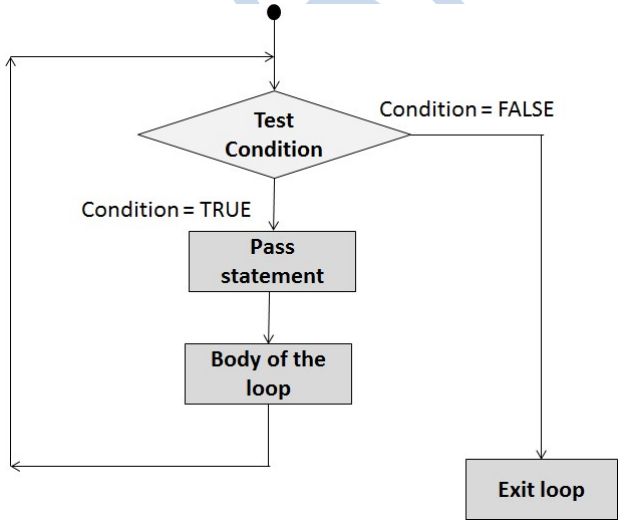
➢ The pass statement in Python is used when a statement is required syntactically but you do not want any command or code to execute.

➢ The pass statement is a null operation.

➢ ***Syntax:***

```
pass;
```

➢ ***Flowchart for pass statement:***



Python Break, Continue And Pass Statements (cont..)

- ***Pass statement (cont..):***

Example program for pass statement

Program :

```
1. #program to explain pass statement
2.  for letter in 'hello':
3.    if letter == 'o':
4.       pass
5.       print 'This is pass block'
6.    print 'Current Letter :', letter
7. print "Thank you!"
```

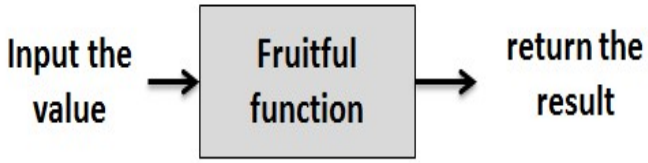
Execution

```
sh-4.3$ python program18.py
Current Letter : h
Current Letter : e
Current Letter : l
Current Letter : l
This is pass block
Current Letter : o
Thank you!
```

APEC

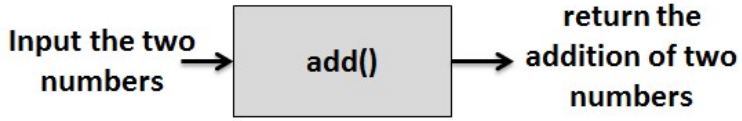
Fruitful Functions

- Functions that return values are called as *fruitful functions*.
- The return statement is followed by an expression which is evaluated. Its result is returned to the caller as the "fruit" of calling this function.



- ***Simple Example:***

  Let's create a mathematical function called add. that takes two numbers as parameters and return the result of adding two numbers.



Fruitful Functions (cont..)

- **Simple Example for fruitful function**

Program :

```
1. # Fruitful function with return statement
2. def add(x,y):
3.     sum = x + y
4.     return sum
5. a = input("Enter a = ")
6. b = input("Enter b = ")
7. print "Addition of two numbers = ", add(a,b)
```

Execution

```
sh-4.3$ python program19.py
Enter a = 2
Enter b = 5
Addition of two numbers =  7
```

APEC

Fruitful Functions (cont...)

- Sometimes it is useful to have multiple return statements, one in each branch of a conditional statement.
- Code that appears after a return statement is called dead code.
- ***Example:***

Program :

```
1.  # Fruitful function with multiple return statements
2.   def original(x):
3.       if x < 0:
4.           print "This is if"
5.           return -x
6.       else:
7.           return x
8.           print "This is else"#Dead code
9.   a = input("Enter value = ")
10. print original(a)
```

Execution

```
sh-4.3$ python program20.py
Enter value = 9
9
Enter value = -8
This is if
8
```

APEC

Fruitful Functions (cont…)

- ***Parameters in Fruitful functions:***

- A ***function*** in Python
  - ✓ Takes input data, called *parameters* or *arguments*
  - ✓ Performs some computations
  - ✓ Returns the result
- A **function definition** syntax:

```
def function(param1, param2)
#computaion
return result;
```

- Once a function is defined, it can be called from the main program or from another function.
- ***Functions call statement syntax*:**

```
function_name(param1, param2);
```

- **Parameter** is the input data that is sent from one function to another. The parameters are of two types:
  - ✓ *Formal parameters*
    - This parameter defined as part of the function definition.
    - The actual parameter value is received by the formal parameter.
  - ✓ *Actual parameters*
    - This parameter defined in the function call.

Fruitful Functions (cont...)

- Example program:

Program :

```
1. # Function with actual and formal parameters
2.  def cube(x):                    # x is the formal
    parameter
3.      return x*x*x
4. a = input("Enter the number = ")
5. b = cube(a)                      # a is the actual
    parameter
6. print  "cube of the given number =", b
```

Execution

```
sh-4.3$ python program21.py
Enter the number = 2
cube of the given number = 8
```

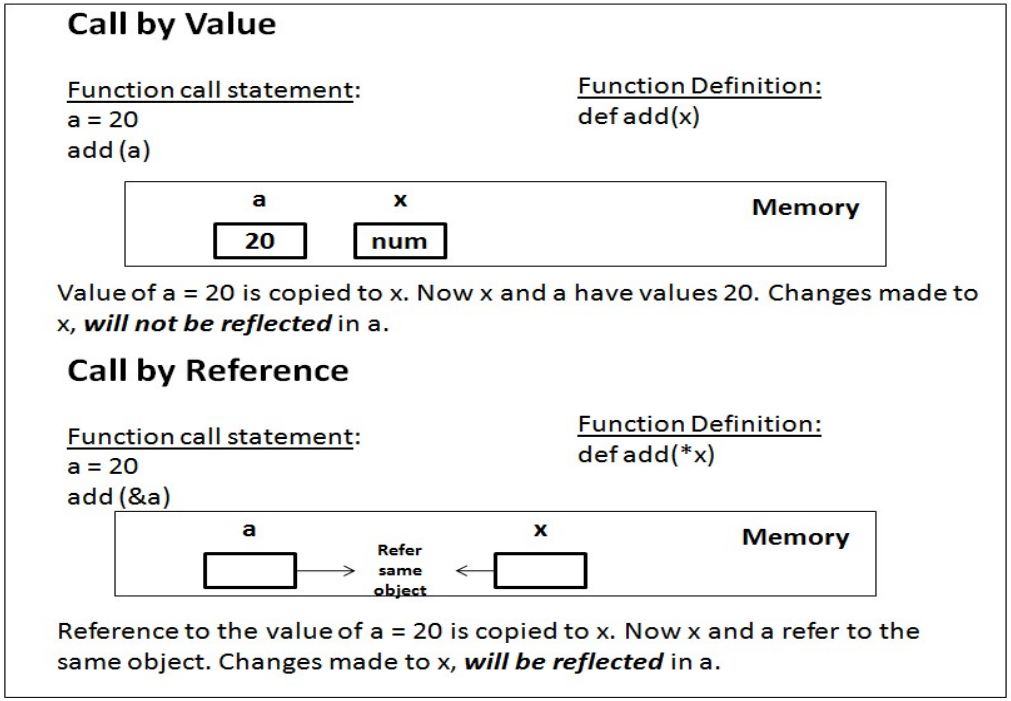
APEC

Fruitful Functions (cont...)

- ***Parameter Passing Techniques***:
- *Two types:*
  - *Call by value*

    a copy of actual arguments is passed to formal arguments and any changes made to the formal arguments have no effect on the actual arguments.
  - *Call by reference*

    the address of actual arguments is passed to formal arguments. By accessing the addresses of formal arguments, it will be reflected in the actual arguments too.



Fruitful Functions (cont...)

- ***Parameter Passing Techniques***:

Example Program(Call by value)

Program :

```
1. #Python program  - call by value example
2. k = 2
3. def sqre(n):              # n = 2, value is copied
4.     n = n * n             # n = 2 * 2 = 4
5.     square = n            # square = 4
6.     return square
7. j = sqre(k)                          # value of k =
   2 is passed to the function sqre
8. print "Square of the given number =", j
9. print k                              # k value = 2
```

Execution

```
sh-4.3$ python program22.py
Square of the given number = 4
2
```

APEC

Fruitful Functions (cont...)

- ***Parameter Passing Techniques*:**

Example Program(Call by value)

Program :

```
#Python program  - call by reference example
1.  def change_list(the_list):
2.      print 'Inside the Function =', the_list
3.      the_list.append('raj')
4.      print 'New list inside function = ', the_list
5.  my_list = ['john', 'jack', 'atlee']
6.  print 'My list before function call =', my_list
7.  change_list(my_list)
8.  print 'My list after function call =', my_list    #call by
    reference
```

Execution

```
sh-4.3$ python program23.py
My list before function call = ['john', 'jack', 'atlee']
Inside the Function = ['john', 'jack', 'atlee']
New list inside function =  ['john', 'jack', 'atlee', 'raj']
My list after function call = ['john', 'jack', 'atlee', 'raj']
```

Fruitful Functions (cont...)

- ***Void functions:***

  It is possible to compose a function without a return statement. Functions like this are called void functions and they return none.

Program :

```
1. # Void Function
2.  def simple():                    #Function definition without
    return statement
3.      print 'Hello'
4.      print 'This is an example for void function'
5.  simple()          #Function call statement
```

Execution

```
sh-4.3$ python program24.py
Hello
                        This is an example for void function
```

APEC

Fruitful Functions (cont…)

- ***Scope of the Variable***

➢ A variable in the program can be either *local variable* or *global variable.*

➢ A global variable is a variable that is declared in the main program while a local variable is a variable that is declared within the function.

```
s = 10                  #  Here, s is the Global variable
def f1():
   s= 55                #  Here, s is the Local variable
   print s       # output = 55
```

➢ Example:

Program :

 # Scope of the variable in the functions

```
1. def f():
2.     s = "Inside function!"
3.     print(s)
4. s = "Outside function!"
5. f()
6. print(s)
```

Execution

```
sh-4.3$ python program25.py
Inside function!
                                Outside function!
```

Fruitful Functions (cont…)

- **Composition**
  - When a function is called from within another function, it's called composition.
  - If this type of nested function is used, the inner function has its scope only in the outer function, so it is most often useful when the inner function is being returned or when it is being passed into another function.
  - ***Syntax:***

```
def outer()
def inner(a)
return a;
return inner;
…..
f outer( )
f
```

  - ***Example:***

Program :

 # Composition of functions

```
1.  def make_adder(x):
2.      def add(y):
3.          print "Inside inner", x + y
4.      return add
5.   plus5 = make_adder(5)
6.  print (plus5(12))
```

Execution

```
sh-4.3$ python program26.py
17
```

Fruitful Functions (cont...)

- ***Python Recursive Function:***
- A function is ***recursive*** if it calls itself and has a termination condition**.**
- ***Limitations of recursions:***

  A recursive function could hold much more memory than a traditional function as it stores memory every time its called.
- **Example – Factorial function using Recursion**

  The mathematical definition of factorial is: n! = n * (n-1)!,

  Example: 3! = 3 x 2 x 1 = 6.

Program :

 # Recursive functions

```
def factorial(n):
    if n == 0:
        return 1
    else:
        return n * factorial(n - 1)
num = input("Enter the input = ")
print "Factorial of the given number =", factorial(num)
```

Execution

```
sh-4.3$ python program27.py
Enter the input = 5
Factorial of the given number = 120
```

APEC

Strings

- A string is a sequence of characters ie., letter, a number, or a backslash.
- Python strings are "immutable" which means they cannot be changed after they are created.
- The various built-in methods are described below:

Method name	Description	Example
Create	Strings are created by enclosing characters within single quotes or double quotes or triple quotes.	str = "Hello World
Accessing the characters	Square brackets [ ] are used to access characters in a string. Positive index - start counting from front Negative index – start counting from last	str = "Hello World" print str[5] print str[-5]
Slicing Strings	Extracting chunk of characters in a string str[start:end:stetpsize]	print str[3:10:2]
Length	It returns the length of the string.	print len(str)
Count	Count how many times, the particular character is available in the string	print str.count('l') print s.count(' ')
Find	Find the location of the particular character in the string	print str.find("H")

Strings (cont..)

Method name	Description	Example
Index	Find the starting location of the substring	print str.index("World")
Concatenate strings	To concatenate strings in Python use the "+" operator.	print str1+str2
Join(str)	Add : between every char in the string	print ":".join(str)
ljust(width, [fillchar])	Padding is done using the specified fillchar for the length 'width'	print str1.ljust(50, '0')
lower()	Converts all uppercase letters to lowercase.	print str.lower()
upper()	Converts lowercase letters to uppercase.	print str.upper()
lstrip()	Removes all leading whitespace in string.	print str. lstrip()
rstrip()	Removes all trailing whitespace of string.	print str. rstrip()
strip()	Performs both lstrip() and rstrip() on string	print str. strip()
max(str)	Returns the max alphabetical character from the string str.	print max(str)
min(str)	Returns the min alphabetical character from the string str.	print min(str)
replace(old, new,[max])	Replaces all occurrences of old in string with new or at most max occurrences if max given.	print str. replace("java", "Cent OS")

Strings (cont..)

Method name	Description	Example
startswith(str)	Determines if string starts with substring str; returns true if so and false otherwise.	print str.endswith('str')
endswith(str)	Determines if string ends with substring str; returns true if so and false otherwise.	print str.endswith('str')
title()	Returns titlecased version of string, ie. all words begins with uppercase and the rest are lowercase.	print str. title()
swapcase()	Change case for all letters in string	print str.swapcase ()
Capitalize	Return the string with it's first character capitalized and the rest lowercased. If the first character is a space, the space is unchanged, the rest lowercased	print string.capitalize()
split(separator)	Split or breakup a string and add to a string array using a defined separator. If no separator is defined then whitespace will be used by default	print string.split(" ")
Repeat Strings	Repeat the string n times	print str * 3
Reverse the string	Reverse the characters in the string	print ' '.join(reversed(str))

Strings (cont..)

Method name	Description	Example
isalnum()	Returns true if string has at least 1 character and all characters are alphanumeric or false otherwise.	print str. isalnum()
isalpha()	Returns true if string has at least 1 character and all characters are alphabetic or false otherwise.	print str. isalpha()
isdigit()	Returns true if string contains only digits and false otherwise	print str. isdigit()
islower()	Returns true if string has lowercase characters and false otherwise	print str. islower()
isupper()	Returns true if string has uppercase characters and false otherwise	print str. isupper()
isnumeric()	Returns true if string contains only numeric characters and false otherwise.	print str. isnumeric()
isspace()	Returns true if string contains only whitespace characters and false otherwise.	print str. isspace()
istitle()	Returns true if string is properly "titlecased" and false otherwise.	print str. istitle()

Strings (cont..)

- ***Example program for String Manipulations***

Program :

 # String Manipulations

```
1.  str  = "Python programming "
2.  str1 = "reg_no:892898!"
3.  print "Character at location 3 = ", str[2]
4.  print "Length of the string = ", len(str)
5.  print "Number of times the given character (m) = ", str.count('m')
6.  print "Number of times , blank space = ", str.count(' ')
7.  print "Location of the character (p) = ",str.find("p")
8.  print "Index of the sub string = ", str.index("program")
9.  print "Join the character : with each character of the string = ", ":".join(str)
10. print "Join the empty space with each character of the string = "," ".join(str)
11. print "Concatenation of two strings using + symbol = ", str + str1
12. print "String in lower case = ",  str.lower()
13. print "String in upper case = ",str.upper()
14. print "Title case of the String = ",str.title()
15. print "Swap the case of the String = ",str.swapcase()
16. print "Capiltalize the String reg no. 892898 = ",str1.capitalize()
17. print "Reverse the String = ",''.join(reversed(str))
18. print "Split the String on white space = ",str.split(" ")
19. print "Split the String on character m = ",str.split("m")
20. print "Replace old string with new string = ", str. replace("Python", "Cent OS")
21. print "Replace old string with new string = ", str. replace("o", "Cent OS")
22. print "Replace old  with new string only 1 time = ", str. replace("o", "Cent OS", 1)
23. print "Maximum characters in the string reg No. 892898 = ", max(str1)
24. print "Minimum characters in the string reg No. 892898 = ", min(str1)
25. print "Padding the string by $ symbol", str1.ljust(25, '$')
26. print "&" * 10              #prints character &, 10 times
27. print str * 3               #prints str 3 times
```

ARE

Strings (cont..)

- ***Example program for String Manipulations***

```
28.str2  = "    Django programming       "
29. print "Removes all leading whitespace in string = ", str2.lstrip()
30. print "Removes all trailing whitespace in string = ", str2.rstrip()
31. print "Removes all whitespace characters in string = ", str2.strip()
32.  #String Slicing
33. print "Get first character of the string = ", str1[0]
34. print "Get only 1 character = ", str1[0:1]
35. print "Get first 3 characters in the string = ", str1[0:3]
36. print "Get last three characters = ", str1[-3:]
37. print "Get first 3 characters & last 3 characters in the string = ", str1[0:3] +
    str1[-3:]
38. print "Get the three characters from location 3 till location 7 = ", str1[3:10]
39. print "Return a character by moving forward 2 positions  = ", str1[3:10:2]
40. print "Get all characters from 3rd place = ", str1[3:]
41. print "Get all characters except last 3 characters = ", str1[:-3]
42.  str1 ="DB124"
43. print str1
44. print "Check alphanumeric characters =", str1.isalnum()
45. print "Check all are alphabetic characters =", str1.isalpha()      #false, if space
46. print "Check if string fully digits =", str1.isdigit()
47. print "Check for title words =", str1.istitle()
48. print "Check for uppercase characters =", str1.isupper()
49. print "Check for lowercase characters =", str1.islower()
50. print "Check for whitespace characters =", str1.isspace()
51. print "Check if string ends with character B =", str1.endswith('B')
52. print "Check if string ends with character B =", str1.startswith('B')
```

APEC

Strings (cont..)

- ***Example program for String Manipulations(cont)***

Execution

```
sh-4.3$ python program28.py
Character at location 3 =  t
Length of the string =  19
Number of times the given character (m) =  2
Number of times , blank space =  2
Location of the character (p) =  7
Index of the sub string =  7
Join the character : with each character of the string =  P:y:t:h:o:n:
:p:r:o:g:r:a:m:m:i:n:g:
Join the empty space with each character of the string =  P y t h o n   p r o g r a m m i n g
Concatenation of two strings using + symbol =  Python programming reg_no:892898!
String in lower case =  python programming
String in upper case =  PYTHON PROGRAMMING
Title case of the String =  Python Programming
Swap the case of the String =  pYTHON PROGRAMMING
Capitalize the String reg no. 892898 =  Reg_no:892898!
Reverse the String =   gnimmargorp nohtyP
Split the String on white space =  ['Python', 'programming', '']
Split the String on character m =  ['Python progra', '', 'ing ']
Replace old string with new string =  Cent OS programming
Replace old string with new string =  PythCent OSn prCent OSgramming
Replace old with new string only 1 time =  PythCent OSn programming
Maximum characters in the string reg No. 892898 =  r
Minimum characters in the string reg No. 892898 =  !
```

APEC

Strings (cont..)

- ***Example program for String Manipulations****(cont)*

```
Padding the string by $ symbol
reg_no:892898!$$$$$$$$$$ &&&&&&&&&&
Python programming Python programming Python programming
Removes all leading whitespace in string =  Django programming
Removes all trailing whitespace in string =      Django programming
Removes all whitespace characters in string =  Django programming
Get first character of the string =  r
Get only 1 character =  r
Get first 3 characters in the string =  reg
Get last three characters =  98!
Get first 3 characters & last 3 characters in the string =  reg98!
Get the three characters from location 3 till location 7 =  _no:892
Return a character by moving forward 2 positions  =  _o82
Get all characters from 3rd place =  _no:892898!
Get all characters except last 3 characters =  reg_no:8928
DB124
Check alphanumeric characters = True
Check all are alphabetic characters = False
Check if string fully digits = False
Check for title words = False
Check for uppercase characters = True
Check for lowercase characters = False
Check for whitespace characters = False
Check if string ends with character D = False
Check if string ends with character D = True
```

APEC

Strings (cont..)

- ## The String Module
- The string module provides additional tools to manipulate strings. This module contains a number of functions to process standard Python strings. In recent versions, string built-in functions are available as string module functions.
- ***Sample Program:***

Program :
 # String Module

```
1.  import string
2.  text = "This is python programming"
3.  print "upper", "=>", string.upper(text)
4.  print "lower", "=>", string.lower(text)
5.  print "split", "=>", string.split(text)
6.  print "join", "=>", string.join(string.split(text), "+")
7.  print "replace", "=>", string.replace(text, "python", "Java")
8.  print "find", "=>", string.find(text, "python"), string.find(text, "Java")
9.  print "count", "=>", string.count(text, "n")
```

Execution

```
sh-4.3$ python program29.py
upper => THIS IS PYTHON PROGRAMMING
lower => this is python programming
split => ['This', 'is', 'python', 'programming']
join => This+is+python+programming
replace => This is Java programming
find => 8 -1
count => 2
```

Python Lists

- Python doesn't have a native array data structure, but it has the list. List is one of the compound data type available in Python often referred to as sequences.
- A list is created by placing all the items (elements) inside a square bracket [ ], separated by commas.
- Items in a list need not be of the same data type.
- A list is mutable; it means the contents of the list are changed.

  list1 = ['physics', 'chemistry', 1997, 2000];

  list2 = [1, 2, 3, 4, 5];

  list3 = ["a", "b", "c", "d"]
- Index operator [] is used to access an item in a list. Index starts from 0. Python allows negative indexing for its sequences. The index of -1 refers to the last item. Nested list are accessed using nested indexing.

APEC

Python Lists (cont..)

- ***Sample Program:***

Program :

#Simple program for list

```
1. my_list = ['p','r','o','b','e']
2. print "First item in the List = ", (my_list[0])
3. print "Third item in the List = ",(my_list[2])
4. print "Last item in the List = ",(my_list[-1])
5. # Nested List
6. n_list = ["Happy", [2,0,1,5], ['john',78,17.3,"hi"]]
7. print "2nd item in list 1 = ", n_list[0][1]
8. print "3rd item in list 3 = ", n_list[2][3]
```

Execution:

```
sh-4.3$ python program30.py
First item in the List =  p
Third item in the List =  o
Last item in the List =  e
2nd item in list 1 =  a
3rd item in list 3 =  hi
```

APEC

Python Lists (cont..)

- ***Basic List Operations:***

  Lists respond to the + and * operators much like strings; they mean concatenation and repetition which results is a new list.

Method name	Description	Syntax
Length	Find the length of the list	len(List)
Concatenation	Concatenate two lists	L1+L2
Repetition	Repeat the item multiple times	['Hi!'] * 4
Append	Add an element to the end of the list	list.append(item)
Insert	Insert an item at the defined index	list.insert(index, item)
Extend	Add all elements of a list to the another list	list.extend(new list)
Remove	Removes an item from the list	list.remove(item)
Pop	Removes an element at the given index	list.pop()
Clear	Removes all items from the list	list.clear()
Index	Returns the index of the first matched item	list[index]
Count	Count the number of times	list.count(item)
Sort	Sort items in a list in ascending order	list.sort()
Reverse	Reverse the order of items in the list	list.reverse()

Python Lists (cont..)

- ***Example program for Basic operations in Lists:***

Program 29:

 # Basic List operations

```
1. L1 = [56,78,98,78]
2. print "Items in the list = ", L1
3. L1.insert(2,"python")
4. print "Inserted item in the list at location 2 = ", L1
5. print "Length of items in the list = ", len(L1)
6. L2 = ['h','e','l','l','o']
7. L3 = L1+L2
8. print "Concatenated items in the list = ", L3
9. print "Repeat item 4 times =", ['Hi!'] * 4
10.L4 = ["hi","john","jack"]
11. L3.extend(L4)
12. print "Extended items in the list =", L3
13. del L3[4]
14. print "Delete the item in the list at location 4 = ", L3
15. del L3[1:5]
16. print "Delete the items from location 1 to 5 = ", L3
17. L3.remove('l')
18. print "Remove an item = ", L3
```

Execution

```
sh-4.3$ python program29.py
Items in the list =  [56, 78, 98, 78]
Inserted item in the list at location 2 =  [56, 78, 'python', 98, 78]
Length of items in the list =  5
Concatenated items in the list =  [56, 78, 'python', 98, 78, 'h', 'e', 'l', 'l', 'o']
Repeat item 4 times = ['Hi!', 'Hi!', 'Hi!', 'Hi!']
Extended items in the list = [56, 78, 'python', 98, 78, 'h', 'e', 'l', 'l', 'o', 'hi', 'john', 'jack']
Delete the item in the list at location 4 =  [56, 78, 'python', 98, 'h', 'e', 'l', 'l', 'o', 'hi', 'john', 'jack']
Delete the items from location 1 to 5 =  [56, 'e', 'l', 'l', 'o', 'hi', 'john', 'jack']
Remove an item =  [56, 'e', 'l', 'o', 'hi', 'john', 'jack']
```

APEC

Python Lists (cont..)

- **Example program for built-in functions in Lists:**

Program 29:

```
# Built-in functions in lists
1.  L = [56,78,98,78]
2.  print "Items in the list = ", L
3.  L.reverse()
4.  print "Reversed items in the list = ", L
5.  L.append(568)
6.  print "Append item in the list = ", L
7.  L.insert(0, "hi")
8.  print "Insert new item after 1st iem = ", L
9.  print "Number of items 78 is in the list = ", L.count(78)
10. print "Length of items in the list = ", len(L)
11. L.sort()
12. print "Sort the items in the list = ", L
13. print "Minimum element in the list = ", min(L)
14. print "Maximum element in the list = ", max(L)
15. print "Pop last time in the list = ", L.pop()
16. print "Pop 1st time in the list = ",L.pop(0)
```

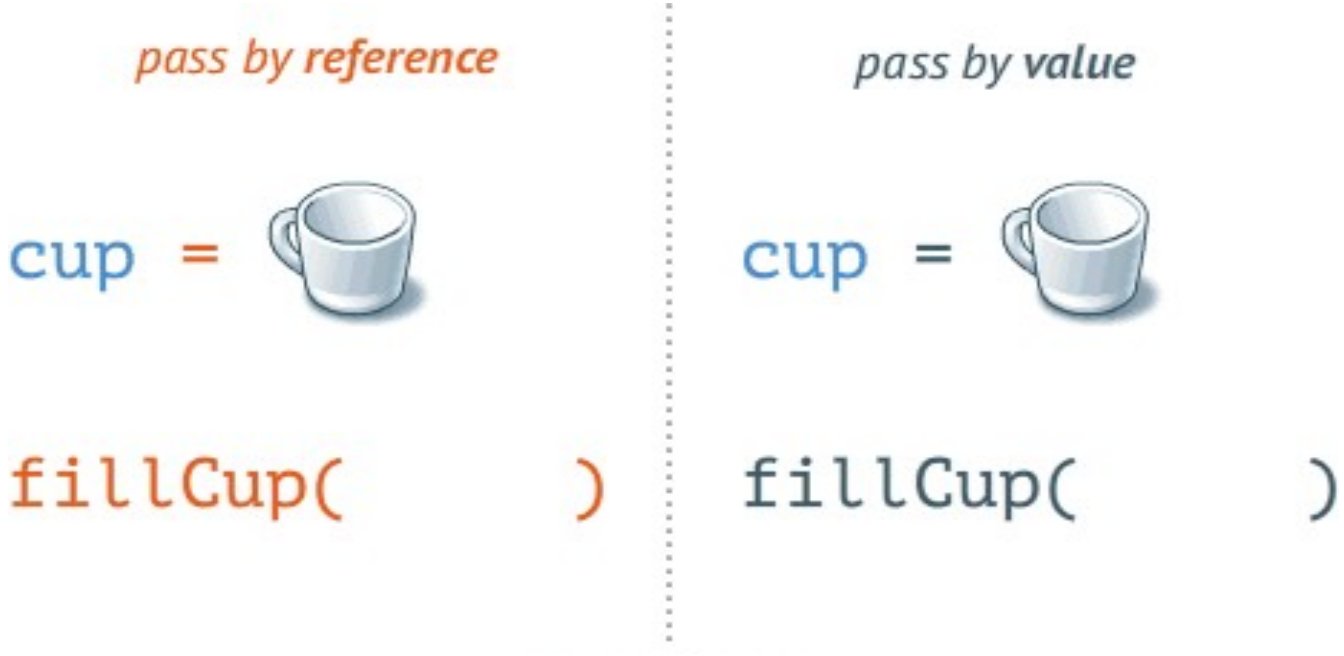
Execution

```
sh-4.3$ python program29.py
Items in the list =  [56, 78, 98, 78]
Reversed items in the list =  [78, 98, 78, 56]
Append item in the list =  [78, 98, 78, 56, 568]
Insert new item after 1st iem =  ['hi', 78, 98, 78, 56, 568]
Number of items 78 is in the list =  2
Length of items in the list =  6
Sort the items in the list =  [56, 78, 78, 98, 568, 'hi']
Minimum element in the list =  56
Maximum element in the list =  hi
Pop last time in the list =  hi
Pop 1st time in the list =  56
```

APEC

APEC

Pass by value vs Pass by address
(or)
call by value vs call by reference



- Think of the coffee in the cup as the data in a variable.
  - One is a copy and one is the original

Call by value Vs call by reference

- Call by value and call by reference are both methods of passing arguments
- **Call by value**
  - A copy of actual arguments is passed to respective formal arguments
- **Call by reference**
  - The location or address of the actual arguments is passed to the formal arguments.

- Note:
  - Parameters (or) Arguments

APEC

Call by value Vs call by reference Contd...

