BansilalRamnathAgrawal Charitable Trust’s

VISHWAKARMA INSTITUTE OF INFORMATION TECHNOLOGY

Department of Engineering &Applied Sciences

F. Y. B. Tech.

Course Material (A Brief Reference Version for Students)

Course: Python for Engineers

UNIT-II : **Python Program Flow Control, functions and packages**

Disclaimer: These notes are for internal circulation and are not meant for commercial use. These notes are meant to provide guidelines and outline of the unit. They are not necessarily complete answers to examination questions. Students must refer reference/ text books, write lecture notes for producing expected answer in examination. Charts/ diagrams must be drawn wherever necessary.

Unit II - Python Program Flow Control, functions and packages
Conditional blocks using if, else and elif, Simple for loops in python ,For loop using ranges, string, list and dictionaries ,Use of while loops in python , Loop manipulation using pass, continue, break and else Programming using Python conditional and loops block. Programming using string, list and dictionary in build functions. Organizing python codes using functions, Understanding Packages Powerful Lambda function in python Programming using functions, modules and external packages.

VIIT

1. Conditional blocks using if, else and elif:

While writing code in any language, you will have to control the flow of your program. This is generally the case when there is decision making involved - you will want to execute a certain line of codes if a condition is satisfied, and a different set of code in case it is not. In Python, you have the **if**, **elif** and the **else** statements for this purpose.

Indentation

Python relies on indentation (whitespace at the beginning of a line) to define scope in the code. Other programming languages often use curly-brackets for this purpose.

1.1 if statement:

This is the simplest example of a conditional statement. An "if statement" is written by using the **if** keyword. In Python, If Statement is used for decision making. It will run the body of code only when IF statement is true.

The syntax is:

```
if(condition):
      indented Statement Block
```

The block of lines indented the same amount after the colon (:) will be executed whenever the condition is TRUE.

The colon (:) is important because it separates the condition from the statements to be executed after the evaluation of the condition. This is specially important for statements where there is only a single statement and the bracket ( ) is not used.

Let us start with simple example. If you are a student of VIIT then, welcome to the class of Python Programming. Code is as below:

Example 1:

```
student='VIIT'

if student=='VIIT':
  print("WELCOME to the Class of Python Programming")

print("bye")
```

VIIT

Output:
```
WELCOME to the Class of Python Programming
Bye
```

Here indented statement is `print("WELCOME to the Class of Python Programming")` which belongs to the condition in if.
And `print("bye")` doesn't belongs to the if condition.
If statement, without indentation (will raise an error)
Example 2:

```
student='VIT'

if student=='VIIT':
  print("WELCOME to the Class of Python Programming")

if student!= 'VIIT':
  print("Sorry, Try again" )

print("bye")
```

Output:
```
Sorry, Try again
Bye
```

This is example of multiple if statement. Every time if condition will be checked, hence affects coding efficiency/complexity.

Example 3: Find out whether the number is even or odd

```
x=5
if x%2==0:
  print("Number is even")

if x%2!=0:
  print("Number is odd")
```
Output:
```
Number is odd
```

Short Hand If

If you have only one statement to execute, you can put it on the same line as the if statement.

Example: One line if statement:

```
if a > b: print("a is greater than b")
```
Example 3 can be written with if-else.

1.2 if-else Statement

The if-else statement is used to code the same way you would use it in the English language. The syntax for the if-else statement is:

```
if(condition):
   Indented statement block for when condition is TRUE
else:
   Indented statement block for when condition is FALSE
```

Tip: If you only have a line of code to be executed rather than multiple lines in the code following the condition, you can place it all in the same line. This is not a rule but just a common practise amongst coders.

The "else condition" is usually used when you have to judge one statement on the basis of other. If one condition goes wrong, then there should be another condition that should justify the statement or logic.

Example 3: Find out whether the number is even or odd

```
x=5
if x%2==0:
  print("Number is even")

else:
  print("Number is odd")
```

Output: `Number is odd`

Example 4: Write a code to for comparison of two numbers

```
x,y =8,4

if x<y :
    st= "x is less than y"
else:
    st= "x is greater than y"
print (st)
```

Output: `x is greater than y`

In the above example if x,y= 8,8 then the code fails.so the program can be written with the if-else-if statement.

Few important points:
1.The condition in if-statement is parenthesis free.
2.The condition can have ‘and’ and ‘or’ operator.
3.It should end with a colon.
4.It must return either True or False.
5.The zero value is considered as False otherwise True.

Short Hand If ... Else

If you have only one statement to execute, one for if, and one for else, you can put it all on the same line:

Example: One line if else statement:

```
a = 2
b = 330
print("A") if a > b else print("B")
```

1.3 if-else-if Statement

The syntax followed by the if-else-if statement is:

```
if(Condition1):
   Indented statement block for Condition1
elif(Condition2):
    Indented statement block for Condition2
else:
    Alternate statement block if all condition check above fails
```

The **elif** statment actually makes writing code easier. Imagine if the problem statement got more complex, then keeping track of every *if-else* statement within another *if* statement could easily become a nightmare!

Example 4: Write a code to for comparison of two numbers

```
x,y =8,8

if x<y :
    st= "x is less than y"
elif x==y:
    st= "x is equal to y"
else:
    st= "x is greater than y"
print (st)
```

Output: `x is equal to y`

- Code Line 1: We define two variables x, y = 8, 8
- Code Line 3: The if Statement checks for condition x<y which is **False** in this case
- Code Line 5: The flow of program control goes to the elseif condition. It checks whether x==y which is true
- Code Line 6: The variable st is set to "x is **equal to** y."
- Code Line 9: The **flow of program control exits the if Statement (it will not get to the else Statement).** And print the variable st. The output is "x is equal to y" which is correct

Example 5: Write a code to find sign of a number

```
num=-55
if num==0:
    print("It is zero")
elif num>0:
    print("Positive Number")
else:
    print(num,"is","Negative Number")
```

Output: `-55 is Negative Number`

Example 6: Read marks in percentages from student and find the grade of it as per below condition: – Above 75% : 'Distinction' – 60% to below 75%: 'First Class' – 50% to below 60%: 'Second Class' – 40% to below 50%: 'Pass Class' – Below 40%: 'Failed'

```
marks=float(input("Enter the Marks:"))

if marks>=75:
    print("Distinction")
elif marks>=60:
    print("First Class")
elif marks>=50:
    print("Second Class")
elif marks>=40:
    print("Pass Class")
else:
    print("Fail")
```

Output:
```
Enter the Marks:76
Distinction
```

Here we have used input() function. The input() function allows user input.

```
>>> a = raw_input()    # raw string input
12
>>> a
'12'
```

Example 7:

```
x = input('Enter your name:')
print('Hello, ' + x)
```

Output:
```
Enter your name:Rupa
Hello, Rupa
```

Nested if Statements

When you have an **if statement inside another if statement**, this is called nesting in the programming world. It doesn't have to always be a simple if statement but you could use the concept of if, if-else and even if-elif-else statements combined to form a more complex structure.

Any number of these statements can be nested inside one another. Indentation is the only way to figure out the level of nesting. They can get confusing, so they must be avoided unless necessary.

Example 7:

```
x = 41

if x > 10:
  print("Above ten,")
  if x > 20:
    print("and also above 20!")
  else:
    print("but not above 20.")
```

Output:
```
Above ten,
and also above 20!
```

Nesting of if-else

Example 8:– Read three numbers from keyboard and find largest of them. You are not allowed to use ‘and’ and ‘or’ operators.

```
num1 = int(input('Enter First: '))
num2 = int(input('Enter Second: '))
num3 = int(input('Enter Third: '))
if num1>num2:
    if num1>num3:
        print("Largest is",num1)
    else:
        print("Largest is",num3)
else:
    if num2>num3:
        print("Largest is",num2)
    else:
        print("Largest is",num3)
```

Output:
```
Enter First: 25
Enter Second: 56
Enter Third: 31
Largest is 56
```

Example 8: Read an year and find whether that year is leap or not?
Conditions for being leap year: –
It should be divisible by four.
If its a century year, it should be divisible by 400 also.

Leap Year:
A year is called a leap year if it contains an additional day which makes the number of the days in that year is 366. This additional day is added in February which makes it 29 days long.A leap year occurred once every 4 years.

```
year = int(input("Enter a year: "))
    if (year % 4) == 0:
       if (year % 100) == 0:
           if (year % 400) == 0:
               print(year, " is a leap year" )
           else:
               print(year, " is not a leap year")
       else:
           print(year, " is a leap year" )
    else:
       print(year, " is not a leap year")
```

Output:
```
Enter a year: 2020
2020  is a leap year
```

You can make use of boolean statements such as **or**, **and** to combine multiple conditions together. But you have to be careful to understand the boolean output of such combined statements to fully realise the flow of control of your program.

Example 8: Read an year and find whether that year is leap or not?

```
year = int(input("Enter a year: "))
if (year % 4) == 0 and (year % 100) != 0 or (year % 4) == 0:
        print(year, " is a leap year" )
else:
       print(year, " is not a leap year")
```

Output:
```
Enter a year: 2020
2020  is a leap year
```

VIT

Exercises:

1.Write a Python program to calculate profit or loss. Input is selling cost and actual cost.

2. Write a Python program to check whether a character is uppercase or lowercase alphabet.

3. Write a Python program to input electricity unit charges and calculate total electricity bill according to the given condition:
–For first 50 units Rs. 0.50/unit
– For next 100 units Rs. 0.75/unit
– For next 100 units Rs. 1.25/unit
– For unit above 250 Rs. 1.50/unit
– An additional surcharge of 17% is added to the bill

2. Simple for loops in python, For loop using ranges, string, list and dictionaries

A loop is a used for iterating over a set of statements repeatedly. In Python we have three types of loops **for**, **while** and **do-while**.

A for loop is used for iterating over a sequence (that is either a list, a tuple, a dictionary, a set, or a string).This is less like the for keyword in other programming languages, and works more like an iterator method as found in other object-orientated programming languages.

With the for loop we can execute a set of statements, once for each item in a list, tuple, set etc.

The structure of a for loop in Python is different than that in C++ or Java. That is, for(int i=0;i<n;i++) won't work here. In Python, we use the 'in' keyword.

Syntax of For loop in Python

```
for <variable> in <sequence>:
          # body_of_loop that has set of statements
          # which requires repeated execution
```

Here <variable> is a variable that is used for iterating over a <sequence>. On every iteration it takes the **next value** from <sequence> until the end of sequence is reached.

Let's start with simple examples

```
for n in 1,6,7,8,4:
    print(n, end=' ')
```

Output:
```
1 6 7 8 4
```

```
for n in 1,6,7,8,4:              #Data Operations
    print(n*n, end=' ')
```

Output:
```
1 36 49 64 16
```

```
for n in 'h',23,3.12,12:        #Mixed Elements
    print(n, end=' ')
```

Output:
```
h 23 3.12 12
```

```
# Program to print squares of all numbers present in a list

# List of integer numbers
numbers = [1, 2, 4, 6, 11, 20]

# variable to store the square of each num temporary
sq = 0

# iterating over the given list
for val in numbers:
    # calculating square of each number
    sq = val * val
    # displaying the squares
    print(sq)
```

Output:
```
1
4
16
36
121
400
```

2.1 For loop

Function range()

In the above example, we have iterated over a list using for loop. However we can also use a range() function in for loop to iterate over numbers defined by range().

range(n): generates a set of whole numbers starting from 0 to (n-1).

For example:

range(8) is equivalent to [0, 1, 2, 3, 4, 5, 6, 7]

range(start, stop): generates a set of whole numbers starting from `start` to `stop-1`.

For example:

range(5, 9) is equivalent to [5, 6, 7, 8]

range(start, stop, step_size): The default step_size is 1 which is why when we didn’t specify the step_size, the numbers generated are having difference of 1. However by specifying step_size we can generate numbers having the difference of step_size.

For example:

range(1, 10, 2) is equivalent to [1, 3, 5, 7, 9]

```
arr = range(10)
list(arr)
```

Output: `[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]`

```
arr = range(1,12,2)
list(arr)
```

Output: `[1, 3, 5, 7, 9, 11]`

Lets use the **range() function** in for loop:

```
arr = [1,2,3,4,5]
for i in arr:
    print i
```

```
    is same as
```

```
for i in [1,2,3,4,5]:
    print i
```

```
    is same as
```

```
for i in range(1, 6):
    print i
```

Output:

```
1
2
3
4
5
```

```
for x in range(1,10,2):
    print(x, end=' ')
```

Output: `1 3 5 7 9`

Note: output prints all elements on the same line .

```
arr = ["str1", "str2", "str3"]            #Iteration without index
for i in arr:
    print(i)
```

Output:

```
str1
str2
str3
```

```
arr = ["str1", "str2", "str3"]           #Iteration with index
for i in range(len(arr)):
    print (i, arr[i])
```

Output:

```
0 str1
1 str2
2 str3
```

```
for i in reversed(range(1,10,2)):        #reverse iteration
    print (i)
```

Output: `9 7 5 3 1`

```
basket = ['apple', 'orange', 'apple', 'pear']  #iteration in sorted set
for f in sorted(set(basket)):
    print (f)
```

Output:

```
apple
orange
pear
```

Example 9: Print each fruit in a fruit list

```
fruits = ["apple", "banana", "cherry"]          #list
for x in fruits:
  print(x)
```

Output:
```
apple
banana
cherry
```

Example 10: Print each element in a list

```
X = [4,7,9,1,5]                                 # list
for num in X:
    print(num, end=' ')
```

Output:
```
4 7 9 1 5
```

Example 11: Print each element in a String

```
R = 'VIIT E&AS Department'                      # string
for data in R:
    print(data, end=' ')
```

Output:
```
V I I T   E & A S   D e p a r t m e n t
```

Example 12: Print collection of String

```
col = ['Mar','Hin','San','Nep']
for name in col:
    print(name)
```

Output:
```
Mar
Hin
San
Nep
```

Example 13: Find addition of first 10 natural numbers using for loop.

```
add=0
for n in range(1,11):
    add+=n
print("sum=",add)
```

Output:
```
sum= 55
```

Example 14: Print each element from dictionary

```
thisdict =  {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
for x in thisdict:
  print(x)
```

Output:
```
brand model year
```

2.2 The else statement for for loop

A for loop may have an else statement after it. When the condition becomes false, the block under the else statement is executed.

However, it doesn't execute if you break out of the loop or if an exception is raised.

Example 15: Find addition of first 10 natural numbers using for loop.

```
add=0
for n in range(1,11):
    add+=n
else:
    print("sum=",add)
```

Output: `sum= 55`

3. while Loop

With the while loop we can execute a set of statements as long as a condition is true.

```
while condition:                                  #compulsory
        statement1 statement2
else:                                              #optional statements
```

In while loop three things are required like C.

1. Initialize
2. Condition
3. Increment/Decrement

Example 16: Print your name for 5 number of times.

```
count =0                                #initialize
while count <5:                         #condition
    print('VIIT',end=" ")
    count +=1                           # increment
```

Output: `VIIT VIIT VIIT VIIT VIIT`

Example 17: Find addition of first ten natural numbers. That is, find addition of numbers from 1 to 10.

```
count,add = 1, 0                              #initialize
while count <=10:                             #condition
    add=add+count
    count +=1                                 # increment
print('Addition is=',add)
```

Output: `Addition is= 55`

Example 18:Find addition of all the odd numbers from 1 to 20. That is, find addition of numbers 1, 3, 5 … 19.

```
count,add = 1, 0                          #initialize
while count <=20:                         #condition
    add=add+count
    count +=2                             # increment
print('Addition is=',add)
```

Output: `Addition is= 100`

Example 19: Write a program to print Fibonacci series up to 10 terms.

```
a,b=0,1
while b<10:
  print(b,end=" ")
  a,b=b,a+b
```

Output:`1 1 2 3 5 8`

3.1 More features of while loop:

A) An Infinite loop
B) The else statement for while loop
C) Single statement while

A) An Infinite loop:

Be careful while using a while loop. Because if you forget to increment the counter variable in python, or write flawed logic, the condition may never become false.

In such a case, the loop will run infinitely, and the conditions after the loop will starve.

To stop execution, press Ctrl+C. However, an infinite loop may actually be useful. **While is the only loop where we can make while loop intentionally infinite.**

Example 20: Print number upto10.

```
num =0                            #initialize
while num <10:                    #condition
    print(num,end=" ")
#counter change is missing
print("end)")
```

As here counter change is missing, the loop will run infinitely

B) The else statement for while loop

A while loop may have an else statement after it. When the condition becomes false, the block under the else statement is executed.

However, it doesn't execute if you break out of the loop or if an exception is raised.

Example 21: Print number upto10.

```
num =0                            #initialize
while num <10:                    #condition
    print(num,end=" ")
    num +=1                       #increment
```

```
else:
    print("end")
```
Output: `0 1 2 3 4 5 6 7 8 9 end`

Example 22:
```
i = 1
while i < 6:
  print(i)
  i += 1
else:
  print("i is no longer less than 6")
```
Output:
```
1
2
3
4
5
i is no longer less than 6
```

C) Single statement while

We can write single statement in while loop. This statement will be separated by semicolon.

Example 23: Print number upto10.
```
num =0
while num <10:print(num,end=" ");num +=1
```
Output: `0 1 2 3 4 5 6 7 8 9`

3.2 While VS for loop

While loop	for loop
while loop can have condition for iteration	Don’t have any loop condition
The counter can be incremented or decremented by any number	The loop can iterate through sequence by one increment only
The loop can be infinite	This loop can’t be infinite

4. Nesting of loops

You can also nest a loop inside another.
You can put a for loop inside a while, or a while inside a for, or a for inside a for, or a while inside a while.
Or you can put a loop inside a loop . You can go as far as you want.

Example 24:

```
for i in range(1,6):                     #outer loop
    for j in range(i):                   #inner loop
        print("*",end=" ")       #prints * on same line for j no of times
    print()                      #print on new line for i number of times
```

Output:

```
*
* *
* * *
* * * *
* * * * *
```

The "inner loop" will be executed one time for each iteration of the "outer loop":

Example 25:Print each adjective for every fruit

```
adj = ["red", "big", "tasty"]
fruits = ["apple", "banana", "cherry"]

for x in adj:
  for y in fruits:
    print(x, y)
```

Output:

```
red apple
red banana
red cherry
big apple
big banana
big cherry
tasty apple
tasty banana
tasty cherry
```

VIT

Exercises:

1. Read a number from user and find the factorial of the number.
2. Take a number user input and find sum of digits of this number.
3. Write a program to print all the odd numbers from 10 to 30.
4. Read a number from keyboard and print it in reverse.
5. Write a program to print Fibonacci series up to n terms.

4. Loop manipulation using pass, continue, break and else

Following are loop control statements:

1. break statement
2. continue statement
3. pass statement

1. break statement

When you put a break statement in the body of a loop, the loop stops executing, and control shifts to the first statement outside it. You can put it in a for or while loop.

Example 26: Print number upto10.

```
for i in range(10):
    if i==5:
        break
    print(i,end="  ")
```

Output: `0   1   2   3   4`

```
At code line 2,condition will be checked, if it's true under if 'break'
will be executed that is loop stops executing.
```

Example 27:

```
i = 1
while i < 6:
  print(i,end="  ")
  if i == 3:
    break
  i += 1
```

Output: `1   2   3`

Example 28: Print fruit in a fruit list

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
  print(x)
  if x == "banana":
    break
```

Output:
```
apple
Banana
```

2. continue statement

When the program control reaches the continue statement, it skips the statements after 'continue'.

It then shifts to the next item in the sequence and executes the block of code for it. You can use it with both for and while loops.

Example 29:

```
for i in range(10):
    if i%2==0:
        continue
    print(i,end="  ")
```

Output:
```
1  3  5  7  9
```

At code line 2, condition will be checked, if it's true under if 'continue' will be executed that is it skips the statements after 'continue'. So in output 2,4,6,8 is missing as condition is true(`i%2==0`). It skips statement `print(i,end="  "),for i=2,4,6,8`

Example 30:

```
i = 0
while i < 6:
  i += 1
  if i == 3:
    continue
  print(i,end="  ")
```

Output:
```
1  2  4  5  6
```

3. **pass statement**

The 'pass' statement, We use the pass statement to implement stubs. When we need a particular loop, class, or function in our program, but don't know what goes in it, we place the pass statement in it.

It is a null statement. The interpreter does not ignore it, but it performs a no-Operation (NOP).

It is used to fill indented space

`for` loops cannot be empty, but if you for some reason have a `for` loop with no content, put in the `pass` statement to avoid getting an error.

Example 31:

```
for x in [0, 1, 2]:
  pass
# having an empty for loop like this, would raise an error without the
pass statement
```

```
while True:
    pass
```

Exercises:

1. Write a program to check whether entered number if prime or not.
2. Write a program to print all the prime numbers from 5 to 50.
3. Print following pattern: 1 2 2 3 3 3 4 4 4 4 5 5 5 5 5
4. Find power of x raised to y from user input.
5. Program to Display the multiplication Table of a number

6. Programming using string, list and dictionary in build functions

6.1. Functions in Python

- A function is a block of code which only runs when it is called.
- You can pass data, known as parameters, into a function.
- A function can return data as a result.
- In Python a function is defined using the def keyword:
- Function Types : Basically, we can divide functions into the following two types: –

1. **Built-in functions - Functions that are built into Python.**
2. **User-defined functions - Functions defined by the users themselves.**

A Sample Function: Example 32:

```
# function defination
def show():
    print("Hello world")

# function call
show()
```

Output: `Hello world`

Let's see how function works

- A function is a block of code with a name.
- We can call a function by its name.
- The code inside a function only runs when it's called.
- A function can accept data from the caller program, it's called as function parameters.
- The function parameters are inside parentheses and separated by a comma. A function can accept any number of arguments.
- A function can return data to the caller program. Unlike other popular programming languages, Python functions definition doesn't specify the return type.
- We can't use **reserved keywords** as the function name. A function name must follow the **Python identifiers** definition rules.

6.1.1 How to Define a Function in Python?

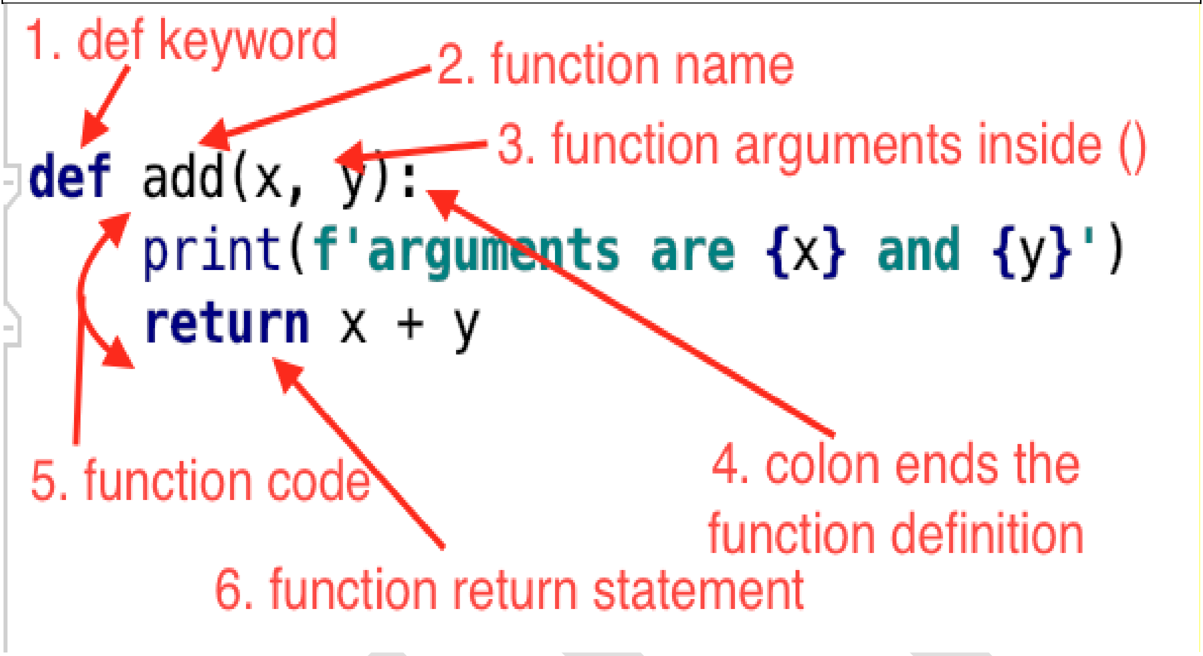
We can define a function in Python using **def keyword**. Let's look at a couple examples of a function in Python.

```
def hello():
    print('Hello World')

def add(x, y):
    print(f'arguments are {x} and {y}')
    return x + y
```

6.1.2 Based on above examples, we can define a function structure as this.

```
def function_name(arguments):
    # code statements
```



```
def add(x, y):
    print(f'arguments are {x} and {y}')
    return x + y
```

1. def keyword
2. function name
3. function arguments inside ()
4. colon ends the function definition
5. function code
6. function return statement

6.1.3 How to Call a Function in Python?

We can call a function by its name. If the function accept parameters, we have to pass them while calling the function

```
hello()
sum = add(10, 5)
print(f'sum is {sum}')
```

We are calling hello() and add() functions that are defined by us. We are also calling print() function that is one of the built-in function in Python.

6.1.4 Python Function Types

There are two types of functions in Python.

1. built-in functions: The functions provided by the Python language such as print(), len(), str(), etc.
2. user-defined functions: The functions defined by us in a Python program.

6.2 Programming using string, list and dictionary in build functions

6.2.1 Python includes the following built-in methods to manipulate strings.

Sr.No	Function	Description
1	capitalize()	Capitalizes first letter of string
2	center(width, fillchar)	Returns a space-padded string with the original string centered to a total of width columns.
3	count(str, beg= 0,end=len(string))	Counts how many times str occurs in string or in a substring of string if starting index beg and ending index end are given.
4	decode(encoding='UTF-8',errors='strict')	Decodes the string using the codec registered for encoding. encoding defaults to the default string encoding.
5	encode(encoding='UTF-8',errors='strict')	Returns encoded string version of string; on error, default is to raise a ValueError unless errors is given with 'ignore' or 'replace'.
6	endswith(suffix,beg=0, end=len(string))	Determines if string or a substring of string (if starting index beg and ending index end are given) ends with suffix; returns true if so and false otherwise.
7	expandtabs(tabsize=8)	Expands tabs in string to multiple spaces; defaults to 8 spaces per tab if tabsize not provided.
8	find(str, beg=0 end=len(string))	Determine if str occurs in string or in a substring of string if starting index beg and ending index end are given returns index if found and -1 otherwise.
9	index(str, beg=0, end=len(string))	Same as find(), but raises an exception if str not found.
10	isalnum()	Returns true if string has at least 1 character and all characters are alphanumeric and false otherwise.
11	isalpha()	Returns true if string has at least 1 character and all characters are alphabetic and false otherwise.
12	isdigit()	Returns true if string contains only digits and false otherwise.
13	islower()	Returns true if string has at least 1 cased character and all cased characters are in lowercase and false otherwise.
14	isnumeric()	Returns true if a unicode string contains only numeric characters and false otherwise.
15	isspace()	Returns true if string contains only whitespace characters and false otherwise.
16	istitle()	Returns true if string is properly "titlecased" and false otherwise.

17	**isupper()**	Returns true if string has at least one cased character and all cased characters are in uppercase and false otherwise.
18	**join(seq)**	Merges (concatenates) the string representations of elements in sequence seq into a string, with separator string.
19	**len(string)**	Returns the length of the string
20	**ljust(width[, fillchar])**	Returns a space-padded string with the original string left-justified to a total of width columns.
21	**lower()**	Converts all uppercase letters in string to lowercase.
22	**lstrip()**	Removes all leading whitespace in string.
23	**maketrans()**	Returns a translation table to be used in translate function.
24	**max(str)**	Returns the max alphabetical character from the string str.
25	**min(str)**	Returns the min alphabetical character from the string str.
26	**replace(old, new [, max])**	Replaces all occurrences of old in string with new or at most max occurrences if max given.
27	**rfind(str, beg=0,end=len(string))**	Same as find(), but search backwards in string.
28	**rindex(str, beg=0, end=len(string))**	Same as index(), but search backwards in string.
29	**rjust(width,[, fillchar])**	Returns a space-padded string with the original string right-justified to a total of width columns.
30	**rstrip()**	Removes all trailing whitespace of string.
31	**split(str="", num=string.count(str))**	Splits string according to delimiter str (space if not provided) and returns list of substrings; split into at most num substrings if given.
32	**splitlines(num=string.count('\n'))**	Splits string at all (or num) NEWLINEs and returns a list of each line with NEWLINEs removed.p>
33	**startswith(str, beg=0,end=len(string))**	Determines if string or a substring of string (if starting index beg and ending index end are given) starts with substring str; returns true if so and false otherwise.
34	**strip([chars])**	Performs both lstrip() and rstrip() on string.
35	**swapcase()**	Inverts case for all letters in string.
36	**title()**	Returns "titlecased" version of string, that

		is, all words begin with uppercase and the rest are lowercase.
37	translate(table, deletechars="")	Translates string according to translation table str(256 chars), removing those in the del string.
38	upper()	Converts lowercase letters in string to uppercase.
39	zfill (width)	Returns original string leftpadded with zeros to a total of width characters; intended for numbers, zfill() retains any sign given (less one zero).
40	isdecimal()	Returns true if a unicode string contains only decimal characters and false otherwise.

6.2.2 Python includes the following list functions

Sr.No	Function	Description
1	cmp(list1, list2)	Compares elements of both lists.
2	len(list)	Gives the total length of the list.
3	max(list)	Returns item from the list with max value.
4	min(list)	Returns item from the list with min value.
5	list(seq)	Converts a tuple into list.
Sr.No	**Methods**	**Description**
1	list.append(obj)	Appends object obj to list
2	list.count(obj)	Returns count of how many times obj occurs in list
3	list.extend(seq)	Appends the contents of seq to list
4	list.index(obj)	Returns the lowest index in list that obj appears
5	list.insert(index, obj)	Inserts object obj into list at offset index
6	list.pop(obj=list[-1])	Removes and returns last object or obj from list
7	list.remove(obj)	Removes object obj from list
8	list.reverse()	Reverses objects of list in place
9	list.sort([func])	Sorts objects of list, use compare func if given

6.2.3 Python includes the following dictionary functions

Sr.No	Function with Description	
1	**cmp(dict1, dict2)**	Compares elements of both dict.
2	**len(dict)**	Gives the total length of the dictionary. This would be equal to the number of items in the dictionary.
3	**str(dict)**	Produces a printable string representation of a dictionary
4	**type(variable)**	Returns the type of the passed variable. If passed variable is dictionary, then it would return a dictionary type.
Sr.No	**Methods**	**Description**
1	**dict.clear()**	Removes all elements of dictionary dict
2	**dict.copy()**	Returns a shallow copy of dictionary dict
3	**dict.fromkeys()**	Create a new dictionary with keys from seq and values *set to value*.
4	**dict.get(key, default=None)**	For key key, returns value or default if key not in dictionary
5	**dict.has_key(key)**	Returns true if key in dictionary *dict*, *false* otherwise
6	**dict.items()**	Returns a list of *dict's* (key, value) tuple pairs
7	**dict.keys()**	Returns list of dictionary dict's keys
8	**dict.setdefault(key, default=None)**	Similar to get(), but will set dict[key]=default if key is not already in dict
9	**dict.update(dict2)**	Adds dictionary *dict2's* key-values pairs to *dict*
10	**dict.values()**	Returns list of dictionary dict's values

Programming using string, list and dictionary in built -in functions:

6.2.1 String

Example 33: Some in build function using string

```
astring = "Hello world!"                                    #length
print("single quotes are ' "")

print(len(astring))
```

```
single quotes are ' '
```
Output:12

```
astring = "Hello world!"                                   #index
print(astring.index("o"))
```
Output:4

```
astring = "Hello world!"                                   #count
print(astring.count("l"))
```
Output:3

```
astring = "Hello world!"                                   #reverse order
print(astring[::-1])
```
Output:!dlrow olleH

```
astring = "Hello world!"                                   #upper and lower case
print(astring.upper())
print(astring.lower())
```
Output: HELLO WORLD!
hello world!

```
astring = "Hello world!"                                   #condition check
print(astring.startswith("Hello"))
print(astring.endswith("asdfasdfasdf"))
```
Output: True
False

```
astring = "Hello world!"                                   #split
afewwords = astring.split(" ")
print(afewwords)
```
Output: `['Hello', 'world!']`

Example 34: write a code to perform operation on given srting using in –build functions.

```
s = "Hey there! what should this string be?"
# Length should be 20
print("Length of s = %d" % len(s))

# First occurrence of "a" should be at index 8
print("The first occurrence of the letter a = %d" % s.index("a"))

# Number of a's should be 2
print("a occurs %d times" % s.count("a"))

# Slicing the string into bits
print("The first five characters are '%s'" % s[:5]) # Start to 5
print("The next five characters are '%s'" % s[5:10]) # 5 to 10
print("The thirteenth character is '%s'" % s[12]) # Just number 12
print("The characters with odd index are '%s'" %s[1::2]) #(0-
based indexing)
print("The last five characters are '%s'" % s[-5:]) # 5th-from-
last to end

# Convert everything to uppercase
print("String in uppercase: %s" % s.upper())

# Convert everything to lowercase
print("String in lowercase: %s" % s.lower())

# Check how a string starts
if s.startswith("Str"):
    print("String starts with 'Str'. Good!")

# Check how a string ends
if s.endswith("ome!"):
    print("String ends with 'ome!'. Good!")

# Split the string into three separate strings,
# each containing only a word
print("Split the words of the string: %s" % s.split(" "))
```

Output:
```
Length of s = 38
The first occurrence of the letter a = 13
a occurs 1 times
The first five characters are 'Hey t'
The next five characters are 'here!'
The thirteenth character is 'h'
The characters with odd index are 'e hr!wa hudti tigb?'
The last five characters are 'g be?'
String in uppercase: HEY THERE! WHAT SHOULD THIS STRING BE?
String in lowercase: hey there! what should this string be?
Split the words of the string: ['Hey', 'there!', 'what', 'should', 'this',
'string', 'be?']
```

6.2.2 List

Example 35: append(x) Adds an element at the end of the list

```
#Append
lst = ['Hello', 'Python']
print(lst)
lst.append('Programming')
print(lst)
```

Output:
```
['Hello', 'Python']
['Hello', 'Python', 'Programming']
```

Example 36: clear() Removes all the elements from the list

```
#clear
lst = ['Hello','Python','Tutorialspoint']
print(lst)
lst.clear()
print(lst)
```

Output:
```
['Hello', 'Python', 'Tutorialspoint']
[]
```

Example 37:

```
#COPY()
#Without copy
lst = ['Hello', 'Python', 'Programming']
lst1 = lst
lst1.append('Java')
print(lst)
print(lst1)
#With copy
lst = ['Hello', 'Python', 'Programming']
lst1 = lst.copy()
lst1.append("Java")
print(lst)
print(lst1)
```

Output:
```
['Hello', 'Python', 'Programming', 'Java']
['Hello', 'Python', 'Programming', 'Java']
['Hello', 'Python', 'Programming']
['Hello', 'Python', 'Programming', 'Java']
```

Example 38: count() Returns the number of elements with the specified value.

```
lst = ['Hello', 'Python', 'Programming', 'Python','Hello']
print(lst.count("Python"))
print(lst.count("Programming"))
print(lst.count(" "))
print(lst.count("Hello"))
```

Output:
```
2
1
0
2
```

Example 39: extend (iterables) Add the elements of a list (or any iterable), to the end of the current list

```
#extend(iterables)
lst = ['Hello', 'Python']
print(lst)
lst.extend(['Java', 'CSharp'])
print(lst)
```

Output:
```
['Hello', 'Python']
```

```
['Hello', 'Python', 'Java', 'CSharp']
```

Example 40: insert(i, x) Adds an element at the specified position

```
lst = ['Hello', 'Python', 'Programming', 'Python']
print(lst)
lst.insert(0, "CPlusPlus")
print(lst)
lst.insert(3, "Java")
print(lst)
```

Output:
```
['Hello', 'Python', 'Programming', 'Python']
['CPlusPlus', 'Hello', 'Python', 'Programming', 'Python']
['CPlusPlus', 'Hello', 'Python', 'Java', 'Programming', 'Python']
```

Example 41: pop([i]) Removes the element at the specified position

```
#pop()
lst = ['CPlusPlus', 'Hello', 'Python', 'Java', 'Programming', 'Python']
print(lst)
#Without index
lst.pop()
print(lst)
#With Index
lst.pop(3)
print(lst)
```

Output:
```
['CPlusPlus', 'Hello', 'Python', 'Java', 'Programming', 'Python']
['CPlusPlus', 'Hello', 'Python', 'Java', 'Programming']
['CPlusPlus', 'Hello', 'Python', 'Programming']
```

Example 42: remove(x) Removes the first item with the specified value

```
#Remove
lst = ['CPlusPlus', 'Hello', 'Python', 'Java', 'Programming', 'Python']
print(lst)
lst.remove('Python')
print(lst)
```

Output:
```
['CPlusPlus', 'Hello', 'Python', 'Java', 'Programming', 'Python']
['CPlusPlus', 'Hello', 'Java', 'Programming', 'Python']
```

Example 43: reverse() Reverses the order of the list

```
#reverse()
lst = ['CPlusPlus', 'Hello', 'Python', 'Java', 'Programming', 'Python']
print(lst)
lst.reverse()
```

Output:
```
['CPlusPlus', 'Hello', 'Python', 'Java', 'Programming', 'Python']
['Python', 'Programming', 'Java', 'Python', 'Hello', 'CPlusPlus']
```

Example 44: sort(key = None, reverse = False) Sorts the list

```
#sort()
lst = [2, 3, 7, 1, 13, 8, 49]
print(lst)
#default
lst.sort()
print(lst)
#reverse = True
lst.sort(reverse = True)
print(lst)
```

Output:
```
[2, 3, 7, 1, 13, 8, 49]
[1, 2, 3, 7, 8, 13, 49]
[49, 13, 8, 7, 3, 2, 1]
```

6.2.3 Dictionary

Example 45: keys() The method keys() returns a list of all the available keys in the dictionary.

```
dict={'Name':'Harry','Rollno':30,'Dept':'cse','Marks':97}
print(dict.keys())
```

Output:
```
dict_keys(['Name', 'Rollno', 'Dept', 'Marks'])
```

Example 46: values() This method returns list of dictionary dictionary's values from the key value pairs.

```
dict={'Name':'Harry','Rollno':30,'Dept':'cse','Marks':97}
print(dict.values())
```

Output:
```
dict_values(['Harry', 30, 'cse', 97])
```

Example 47: pop() The method pop(key) Removes and returns the value of specified key.

```
dict={'Name':'Harry','Rollno':30,'Dept':'cse','Marks':97}
dict.pop('Marks')
print(dict)
```

Output:
```
{'Name': 'Harry', 'Rollno': 30, 'Dept': 'cse'}
```

Example 48: copy() This method Returns a shallow copy of dictionary**.**

```
dict={'Name':'Harry','Rollno':30,'Dept':'cse','Marks':97}
dict_new=dict.copy()
print(dict_new)
```

Output:
```
{'Name': 'Harry', 'Rollno': 30, 'Dept': 'cse', 'Marks': 97}
```

VIT

Example 49: clear() The method clear() Removes all elements of dictionary.

```
dict={'Name':'Harry','Rollno':30,'Dept':'cse','Marks':97}
dict.clear()
print(dict)
```

Output:
```
{}
```

Example 50: get() This method returns value of given key or None as default if key is not in the dictionary.

```
dict={'Name':'Harry','Rollno':30,'Dept':'cse','Marks':97}
print('\nName: ', dict.get('Name'))
print('\nAge: ', dict.get('Age'))
```

Output:
```
Name:  Harry
Age:  None
```

Example 51: update() The update() inserts new item to the dictionary.

```
dict={'Name':'Harry','Rollno':30,'Dept':'cse','Marks':97}
dict.update({'Age':22})
print(dict)
```

Output:
```
{'Name': 'Harry', 'Rollno': 30, 'Dept': 'cse', 'Marks': 97, 'Age':
 22}
```

7. Organizing python codes using functions

7.1 User-defined functions - Functions defined by the users themselves.

1) **Parameterized Functions :**
Information can be passed to functions as parameter. Parameters are specified after the function name, inside the parentheses. You can add as many parameters as you want, just separate them with a comma.

Example 52: write a code to get a square of a number

```
def square(x):                        # function defination
    y=x*x
    print("square=",y)

num=int(input("Enter number:"))
square(num)                           # function call
```

Output:
```
Enter number:25
```

```
square= 625
```

2) **Function Returning values:**

However, when one of the return statement is reached, the function execution terminates and the value is returned to the caller.

Example 53:

```
def my_function(x):
  return 5 * x

print(my_function(3))
print(my_function(5))
print(my_function(9))
```

Output:
```
15
25
45
```

3) **Can we have multiple return statements inside a Function?**
Yes, a function can have multiple return statements.

Example 54:

```
def odd_even_checker(i):
    if i % 2 == 0:
        return 'even'
    else:
        return 'odd'
print(odd_even_checker(20))
print(odd_even_checker(15))
```

Output:
```
even
odd
```

4) **Function have default parameter value**

Python allows default values for the function parameters. If the caller doesn't pass the parameter then the default value is used.

Example 55:

```
def hello(year=2019):
    print(f'Hello World {year}')


hello(2020)   # function parameter is passed
hello()   # function parameter is not passed, so default value will be
```

Output:
```
Hello World 2020
Hello World 2019
```

5) **Can Python Function Return Multiple Values one by one?**

Python function can return multiple values one by one. It's implemented using the yield keyword. It's useful when you want a function to return a large number of values and process them. We can split the returned values into multiple chunks using yield statement. This type of function is also called a generator function.

Example 56:

```
def return_odd_ints(i):
    x = 1
    while x <= i:
        yield x
        x += 2

output = return_odd_ints(10)
for out in output:
    print(out)
```

Output:
```
1
3
5
7
9
```

Example 57:

```
#functions returning multiple values
def array(n):
  add=0
  for x in n:
      add+=x
  avg=add/len(n)
  return add,avg                  # multi return

arr=[15,35,67,88,99,34,25,44,10,11]
a,b= array(arr)
print("addition is:",a)
print("average is%.2f:",b)
```

Output:
```
addition is: 428
average is%.2f: 42.8
```

6) Programming using user defined function :

Example 58:

```
def my_function(food):
  for x in food:
    print(x)

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

my_function(fruits)
```

Output:
```
apple
banana
```

```
      cherry
```

Example 59: Write Program for a simple calculator

```
# This function adds two numbers
def add(x, y):
    return x + y

# This function subtracts two numbers
def subtract(x, y):
    return x - y

# This function multiplies two numbers
def multiply(x, y):
    return x * y

# This function divides two numbers
def divide(x, y):
    return x / y

print("Select operation.")
print("1.Add")
print("2.Subtract")
print("3.Multiply")
print("4.Divide")

while True:
    # Take input from the user
    choice = input("Enter choice(1/2/3/4): ")

    # Check if choice is one of the four options
    if choice in ('1', '2', '3', '4'):
        num1 = float(input("Enter first number: "))
        num2 = float(input("Enter second number: "))

        if choice == '1':
            print(num1, "+", num2, "=", add(num1, num2))

        elif choice == '2':
            print(num1, "-", num2, "=", subtract(num1, num2))

        elif choice == '3':
            print(num1, "*", num2, "=", multiply(num1, num2))

        elif choice == '4':
            print(num1, "/", num2, "=", divide(num1, num2))
        break
    else:
        print("Invalid Input")
```

Output:
```
Select operation.
1.Add
2.Subtract
3.Multiply
4.Divide
Enter choice(1/2/3/4): 3
Enter first number: 25
Enter second number: 5
25.0 * 5.0 = 125.0
```

Example 60: Write Program to Compute LCM

```
# Python Program to find the L.C.M. of two input number
def compute_lcm(x, y):
   # choose the greater number
   if x > y:
       greater = x
   else:
       greater = y

   while(True):
       if((greater % x == 0) and (greater % y == 0)):
           lcm = greater
           break
       greater += 1

   return lcm
num1 = 54
num2 = 24

print("The L.C.M. is", compute_lcm(num1, num2))
```

Output:
```
The L.C.M. is 216
```

Example 61:

```
# Define our 3 functions
def my_function():
    print("Hello From My Function!")

def my_function_with_args(username, greeting):
    print("Hello, %s , From My Function!, I wish you %s"%(username, gre
eting))

def sum_two_numbers(a, b):
    return a + b
```

```
# print(a simple greeting)
my_function()

#prints -
 "Hello, John Doe, From My Function!, I wish you a great year!"
my_function_with_args("Pytho", "a great year!")

# after this line x will hold the value 3!
x = sum_two_numbers(1,2)
print("sum=",x)
```

Output:
```
Hello From My Function!
 Hello, Pytho , From My Function!, I wish you a great year!
 sum= 3
```

7) Advantages of Python Functions

1. Code reusability because we can call the same function multiple times
2. Modular code since we can define different functions for different tasks
3. Improves maintainability of the code
4. Abstraction as the caller doesn’t need to know the function implementation

VIT

8. Powerful Lambda function in python Programming using functions

Understanding Packages Powerful Lambda function in python Programming using functions

In Python, anonymous function is a function that is defined without a name. While normal functions are defined using the def keyword, in Python anonymous functions are defined using the lambda keyword.

Hence, anonymous functions are also called lambda functions. A lambda function is a small anonymous function.

A lambda function can take any number of arguments, but can only have one expression.

Syntax:

lambda *arguments* : *expression*

The expression is executed and the result is returned. Lambda functions can be used wherever function objects are required.

Example 62: A lambda function that adds 10 to the number passed in as an argument, and print the result:

```
x = lambda a : a + 10
print(x(5))
```

Output: `15`

Example 63: A lambda function for square of a number

```
square=lambda x:x**2
print(square(12))
```

Output: `144`

Example 64: A lambda function that multiplies argument a with argument b and print the result

```
x = lambda a, b : a * b
print(x(5, 6))
```

Output: `30`

Example 65:A lambda function that sums argument a, b, and c and print the result

```
x = lambda a, b, c : a + b + c
print(x(5, 6, 2))
```

Output: `13`

8.1 Lambda Functions

The power of lambda is better shown when you use them as an anonymous function inside another function.

Example 66:

```
def myfunc(n):
  return lambda a : a * n

mydoubler = myfunc(2)

print(mydoubler(11))
```

Output:

```
22
```

Example 67:

```
def myfunc(n):
  return lambda a : a * n

mydoubler = myfunc(2)
mytripler = myfunc(3)

print(mydoubler(11))
print(mytripler(11))
```

Output:

```
22
33
```

8.2 Using lambda function

We use lambda functions when we require a nameless function for a short period of time. In Python, we generally use it as an argument to a higher-order function (a function that takes in other functions as arguments). Lambda functions are used along with built-in functions like filter(), map() etc.

8.2.1 The filter( ) :

The filter() function in Python takes in a function and a list as arguments. The function is called with all the items in the list and a new list is returned which contains items for which the function evaluates to True.

Example 68:

```
# a list that contains both even and odd numbers
seq=[1,2,3,5,7,12,15,20]

#result contains odd number of list
result=filter(lambda x:x%2,seq)
print("odd=",result)

#result contains even number of list
result=filter(lambda x:x%2==0,seq)
print("even=",result)
```

8.2.2 The map( ) :

The map() function in Python takes in a function and a list. The function is called with all the items in the list and a new list is returned which contains items returned by that function for each item.

Example 69:

```
def square(n):
    return n*n

#we square all numbers using map()
numbers=(1,2,3,4)
result = map(square,numbers)
print(result)

#list of strings
l=['sat','bat','cat','mat']

#map() can listify the list of strings individually
test=map(list,l)
print(test)
```

9. Modules and external packages

9.1 Modules:

Modules are used to categorize code in Python into smaller part. A module is simply a file, where classes, functions and variables are defined. Grouping similar code into a single file makes it easy to access.

- **Types**: – System modules

  – User defined modules

- **Advantages**: Python provides the following advantages for using module: –

1. Reusability: Module can be used in some other python code. Hence it provides the facility of code reusability.
2. Categorization: Similar type of attributes can be placed in one module.

9.1.1 Using system modules

• import keyword

• import as

• from import

1) The import statement

VIIT

```
>>> import math

>>> print math.sin(67)

0.855519978975

>>> print math.log(67)

4.20469261939

 >>> print math.sqrt(67)

8.18535277187

>>> print math.pi

3.14159265359

>>> print math.e

2.71828182846
```

2) Import using as

```
>>> import math as m

>>> print m.sin(90)

 0.893996663601

>>> print m.log(90)

4.49980967033

>>> print m.sqrt(90)

9.48683298051

>>> print m.pi

3.14159265359

 >>> print m.e

2.71828182846
```

3) Selective import

from…import statement is used to import particular attribute from a module. In case you do not want whole of the module to be imported then you can use from import statement.

```
>>> from math import sin, log, pi

>>> print sin(55)

0.999755173359

>>> print log(55)

4.00733318523

>>> print pi

3.14159265359

>>> from math import *

>>> print cos(55)

0.022126756262

>>> print e

2.71828182846
```

9.1.2 User defined modules

Any program stored in current working directory can be imported by import statement. When such program (module) is imported, we can use all functions, classes and global variables from that program in your program. The import, import...as and from...import will work in the same fashion..

It creates a python compiled file with extension .py in current working directory

- **What is a Module?**

Consider a module to be the same as a code library.

A file containing a set of functions you want to include in your application.

VIIT

- **Create a Module**

To create a module just save the code you want in a file with the file extension `.py`:

Example 70: Save this code in a file named `mymodule.py`

```
def greeting(name):
  print("Hello, " + name)
```

- **Use a Module**

Now we can use the module we just created, by using the `import` statement

Example 71: Import the module named mymodule, and call the greeting function:

```
import mymodule

mymodule.greeting("Jonathan")
```

Output: Hello, Jonathan

Example 72: Save this code in the file `mymodule.py`

```
person1 = {
  "name": "John",
  "age": 36,
  "country": "Norway"
}
```

Example 73: Import the module named mymodule, and access the person1 dictionary

```
import mymodule

a = mymodule.person1["age"]
print(a)
```

Output: 36

- **Naming a Module**

You can name the module file whatever you like, but it must have the file extension `.py`

- **Re-naming a Module**

You can create an alias when you import a module, by using the `as` keyword:

Example: Create an alias for `mymodule` called `mx`:

```
import mymodule as mx

a = mx.person1["age"]
print(a)
```

- **Import From Module**

You can choose to import only parts from a module, by using the `from` keyword.

Example 74: The module named `mymodule` has one function and one dictionary

```
def greeting(name):
  print("Hello, " + name)

person1 = {
  "name": "John",
  "age": 36,
  "country": "Norway"
}
```

Example 75: Import only the person1 dictionary from the module

```
from mymodule import person1

print (person1["age"])
```

Output: 36

Example 76: The module named fact.py

```
def factorial(n):
    if n<=1:
        return 1
    else:
        return n* factorial(n-1)

def hello(s):
    print("Hello",s)

name= "VISHWAKARMA INSTITUTE OF INFORMATION TECHNOLOGY"
```

Example 77: Import the module named fact.py and access the function named as factorial and hello

```
import fact
print(fact.factorial(5))
print (hello('RUPA'))
print(name)
```

Output:
```
120
Hello RUPA
VISHWAKARMA INSTITUTE OF INFORMATION TECHNOLOGY
```

9.2 External Packages:

Packages are namespaces which contain multiple packages and modules themselves. They are simply directories, but with a twist.

Each package in Python is a directory which **MUST** contain a special file called __init__.py. This file can be empty, and it indicates that the directory it contains is a Python package, so it can be imported the same way a module can be imported.

A Package is simply a collection of similar modules, sub packages etc..

9.2.1 Steps to create and import Package:

1) Create a directory, say Info

2) Place different modules inside the directory. We are placing 3 modules msg1.py, msg2.py and msg3.py respectively and place corresponding codes in respective modules. Let us place msg1() in msg1.py, msg2() in msg2.py and msg3() in msg3.py.

3) Create a file __init__.py which specifies attributes in each module.

4) Import the package and use the attributes using package.

Example: 78

File 1

```
#msg1.py
def show():
    print("Module-1")
```

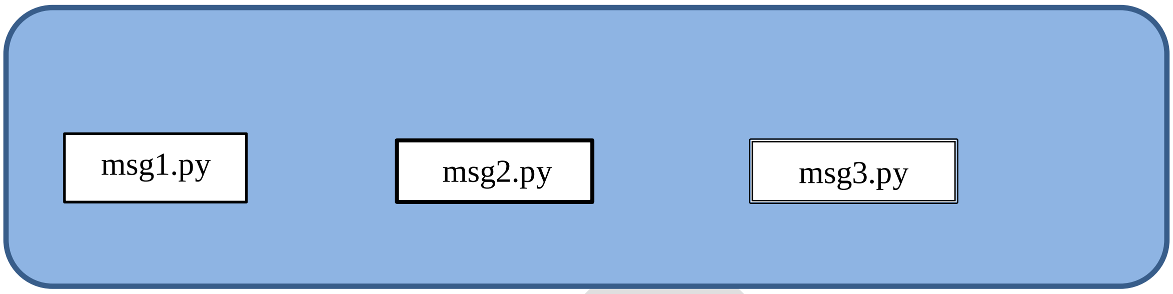
File 2

```
#msg2.py
def display():
    print("Module-2")
```

File 3

```
#msg3.py
def output():
    print("Module-3")
```

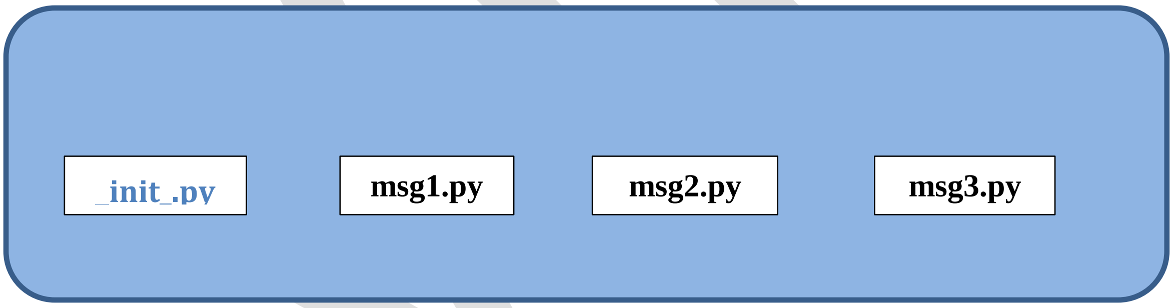
Info Folder



init.py

```
#_init_.py
from msg1 import show
from msg2 import display
from msg3 import output
```

Info Folder



Using packages

Now create a new file outside of 'info' to import the given package. We can apply all import methods to use the package.

```
import info
info.show()
info.display()
info.output()
```

It creates four new .py files in info folder.

VIIT