UNIT:3

Python statements and Loops:

if, if-else, While, for loops, break, continue, pass, Python Function;

Files I/O.

Functions: Definition, call, positional and keyword parameter. Default parameters, variable number of arguments. Modules - import mechanisms. Functional programming - map, filter, reduce, max, min. lambda function - list comprehension.

Functions:

- Function is a block of reusable code that performs a specific task or set of tasks.
- Functions provide a way to organize code into logical and modular units, making it easier to understand, debug and maintain.

(If a group of statements is repeatedly required then it is not recommended to write these statements every time separately. We have to define these statements as a single unit and we can call that unit any number of times based on our requirement without rewriting. This unit is nothing but function.)

Types of Functions:

1. Built – in Functions (Pre defined Functions)
2. User defined Functions

Built – in Functions: The functions which are coming along with Python software automatically, are called built in functions or pre-defined functions.

E.g. – id (), type (), print (), input (), eval () etc.

User defined Functions: The functions which are developed by programmer explicitly according to business requirements, are called user defined functions.

syntax: (Function definition)

def function_name (parameters):

‘”doc string ‘”

…….

…….

return value

function_name (arguments) # (Function calling)

Parameters:

Parameters are inputs to the function. If a function contains parameters, then at the time of calling, compulsory we should provide values otherwise, otherwise we will get error.

return statement:

Function can take input values as parameters and executes business logic, and returns output to the caller with return statement.

- Returning multiple values from a function: In other languages like C, C++ and Java, function can return at most one value. But in Python, a function can return any number of values.

E.g.

```
def sum_sub (a, b):
sum=a + b
sub=a - b
return sum, sub
x, y = sum_sub (100,50)
print ("The Sum is:", x)
print ("The Subtraction is:", y)
```

E.g.2

```
def calc (a, b):
```

sum = a + b

sub = a-b

mul = a*b

div = a/b

return sum, sub, mul, div

t = calc (100, 50)

print (“The results are: “)

for I in t:

print(i)

- **Types of arguments:**

def f1 (a, b):

……..

……..

……..

f1 (10,20)

a, b are formal arguments where as 10, 20 are actual arguments.

There are 4 types of actual arguments allowed in python.

1. positional arguments
2. keyword arguments
3. default arguments
4. variable length arguments
5. **positional arguments:**

These are the arguments passed to function in correct positional order.

E.g.

def sub (a, b):

print (a-b)

sub (100, 200)

sub (200, 100)

The number of arguments and position of arguments must be matched. If we change the order then result may be changed.

If we will change the number of arguments then we will get error.

1. **keyword arguments:**

we can pass argument values by keyword i.e. by parameter name.

E.g.

def wish (name, msg):

print ("Hello", name, msg)

wish (name="Bunny”, msg="Good Morning")

wish (msg="Good Morning", name="Bunny”)

Here the order of arguments is not important but number of arguments must be matched.

Note: We can use both positional and keyword arguments simultaneously. But first we have to take positional arguments and then keyword arguments, otherwise we will get syntax error.

1. **default arguments:**

Sometimes we can provide default values for our positional arguments.

E.g.

```
def wish (name = 'Guest'):

print ('Hello', name, 'Good morning')

wish ('Bunny')

wish ()
```

If we are not passing any name then only default value will be considered.

***Note:

After default arguments we should not take non default arguments.

def wish (name="Guest", msg="Good Morning"): ===>Valid

def wish (name, msg="Good Morning"): ===>Valid

def wish (name="Guest", msg): ==>Invalid

1. **variable length arguments:**

Sometimes we can pass variable number of arguments to our function, such type of arguments is called variable length arguments. We can declare a variable length argument with * symbol as follows

def f1(*n):

We can call this function by passing any number of arguments including zero number. Internally all these values represented in the form of tuple.

E.g.

```
def sum (*n):

total = 0

for n1 in n:

total = total + n1

print ('The sum = ', total)

sum ()

sum (10)

sum (10,20,30,40)

sum (10,20,30,40,50)
```

Note: After variable length argument, if we are taking any other arguments then we should provide values as keyword arguments.

- **Types of Variables**

Python supports 2 types of variables.

1. Global Variables

2. Local Variables

1. **Global Variables:** The variables which are declared outside of function are called global variables.

These variables can be accessed in all functions of that module.

E.g.

```
a = 10 #global variable

def f1 ():

print (a)

def f2 ():
```

print (a)

f1 () #output-10

f2 () #output-10

2. Local Variables: The variables which are declared inside a function are called local variables. Local variables are available only for the function in which we declared it. i.e. from outside of function we cannot access.

E.g.

def f1 ():

a = 10

print (a) #valid

def f2 ():

print (a) #invalid

f1 ()

f2 () #Error: name 'a' is not defined

'global' keyword:

We can use global keyword for the following 2 purposes:

1. To declare global variable inside function

2. To make global variable available to the function so that we can perform required modifications.

E.g.

a = 10

def f1 ():

global a

a = 777

print (a)

def f2 ():

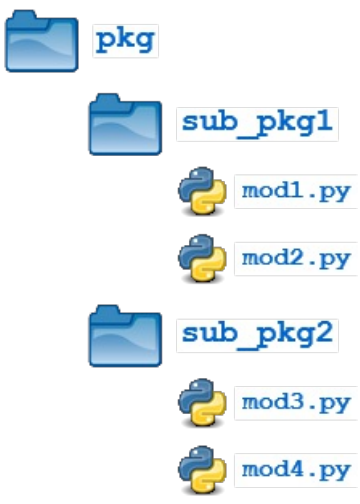
print(a)

f1 () #777

f2 () #777

Modules:

- A module is a file containing python definition, functions, variables, classes and statements. The extension of this file is '.py'.
- While python package is directory (folder) of python modules.
- A library is collection of many packages in python. Generally, there is no difference between python package and python library.



In Python, the **import** statement is used to bring in functionality from other modules or packages into the current module or script.

1. Importing a Module: You can import a module by its name using the import statement. After importing, you can access functions, classes, or variables defined in that module using dot notation (module_name. item_name).

E.g. import math

print (math. sqrt (44))

1. Importing Specific Items from a Module: You can import specific items (functions, classes, or variables) from a module rather than importing the whole module.

E.g. from math import sqrt

print (math. sqrt (4))

1. Importing with Aliases: You can import modules or items with an alias to simplify their usage in your code.

E.g. import pandas as pd

1. Importing All Items from a Module (Not Recommended):

E.g. from math import *

Functional Programming:

In functional programming: Functions are First-Class Citizens: Functions can be passed as arguments to other functions, returned from other functions, and assigned to variables.

While Python is primarily an object-oriented language, it also supports functional programming to a certain extent, allowing developers to use functional programming concepts and techniques alongside other programming paradigms.

- Functional programming constructs in Python include:
- **map (), filter (), and reduce ():**

Built-in functions for functional programming operations like applying a function to every item in an iterable, filtering items based on a condition, and reducing a list of values to a single value, respectively.

1. map () - Applies a function to all items in an iterable and returns a new iterator with the results.

syntax – map (function, iterable)

Example –

Doubling each element in a list

numbers = [1, 2, 3, 4, 5]

doubled_numbers = map (lambda x: x * 2, numbers)

print(list(doubled_numbers)) # Output: [2, 4, 6, 8, 10]

1. filter () - Filters elements from an iterable based on a function that returns True or False, and returns an iterator with the filtered elements.

syntax – filter (function, iterable)

Example –

Filtering even numbers from a list

numbers = [1, 2, 3, 4, 5]

even_numbers = filter (lambda x: x % 2 == 0, numbers)

print(list(even_numbers)) # Output: [2, 4]

1. reduce () - Applies a function of two arguments cumulatively to the items of an iterable, from left to right, to reduce the iterable to a single value.

syntax – reduce (function, iterable)

(Note: reduce () function is part of the functools module in Python 3 and needs to be imported)

Example –

Summing up elements in a list

from functools import reduce

numbers = [1, 2, 3, 4, 5]

sum_of_numbers = reduce (lambda x, y: x + y, numbers)

print(sum_of_numbers) # Output: 15

- **lambda function:**
- A lambda function is an anonymous function defined without a name and without using a ‘def’ keyword.
- It’s the simplest way to define a function in python.
- A lambda function can take any number of arguments but can only have one expression.

syntax – lambda arguments: expression

E.g. summation of numbers without lambda function

def add (x, y):

return x + y

add (10,20,30) #60

E.g.1- using lambda function

add = lambda x, y: x + y

add (20,30)

(or)

(lambda x, y: x + y) (10, 20)

E.g.-2: Addition of multiple number of arguments

addition = lambda *n: sum(n)

print (addition (90,2,3,45))

E.g.-3: (using ternary operator in lambda function)

(lambda x:('odd' if x%2 else 'even')) (45)

E.g.-4:

list1 = [40,10,45,33,15,17,56]

div_4 = list (filter (lambda x: (x%4==0), list1))

print(div_4)

- **List comprehension:**

Provides a shorter syntax while creating a new list from the existing list.

syntax –

list_name = [expression for item in iterable if condition == True]

E.g.

names = [‘Aarti’, ‘Aditi’, ‘Aditya’, ‘Ayush’, ‘Dipti’, ‘Charlie’]

a_names = [name for name in names if ‘A’ in name]

print (a_names)

Practice Questions:

01. creating squares of 1 to 10 numbers, using list comprehension

02. filtering even numbers from

list1 = [0,1,2,3,4,5,6,7,8,9]

03. i/p - ['Alice', 'Bob']

o/p - [('Alice',5), ('Bob',3)]

04. i/p – num = [1,2,3,4,5]

o/p – [‘odd’, ‘even’, ‘odd’, ‘even’, ‘odd’]