CORE PYTHON

Introduction to PYTHON:

- Python is a simple, easy to learn, powerful, high level and object-oriented programming language.
- Python is an interpreted scripting language also.
- Python was developed by Guido Van Rossum and Released in 1991.
- Python is a general purpose, dynamic, high level and interpreted programming language.
- It supports Object Oriented programming approach to develop applications.
- It is simple and easy to learn and provides lots of high-level data structures.
- Python is *easy to learn* yet powerful and versatile scripting language which makes it attractive for Application Development.
- Python supports *multiple programming pattern*, including object oriented, and functional or procedural programming styles.
- We don't need to use data types to declare variable because it is *dynamically typed* so we can write a=10 to assign an integer value in an integer variable.

Features of PYTHON:

1. Easy to Learn and Use

- Python is easy to learn and use. It is developer-friendly and high level programming language.

2. Expressive Language

- Python language is more expressive means that it is more understandable and readable.

3. **Interpreted Language**

- Python is an interpreted language i.e. interpreter executes the code line by line at a time. This makes debugging easy and it's suitable for beginners.

4. **Cross-platform Language**

- Python can run equally on different platforms such as Windows, Linux, Unix and Macintosh etc. So, we can say that Python is a portable language.

5 .**Free and Open Source**

- Python language is freely available at official web address.www.python.org/downloads.
- The source-code is also available. Therefore it is open source.

6. **Object-Oriented Language**

- Python supports object oriented language and concepts of classes and objects come into existence.

7. **Extensible**

- It implies that other languages such as C/C++ can be used to compile the code and thus it can be used further in our python code.

8. **Large Standard Library**

- Python has a large and broad library and provides rich set of module and functions for rapid application development.

9. **GUI Programming Support**

- Graphical user interfaces can be developed using Python.

10. **Integrated**

- It can be easily integrated with languages like C, C++, and Java etc.

Versions of PYTHON:

- **Python 0.9.0 - February 20, 1991**
  - Python 0.9.1 - February, 1991
  - Python 0.9.2 - Autumn, 1991
  - Python 0.9.4 - December 24, 1991
  - Python 0.9.5 - January 2, 1992
  - Python 0.9.6 - April 6, 1992
  - Python 0.9.8 - January 9, 1993
  - Python 0.9.9 - July 29, 1993
- **Python 1.0 - January 1994**
  - Python 1.2 - April 10, 1995
  - Python 1.3 - October 12, 1995
  - Python 1.4 - October 25, 1996
  - Python 1.5 - December 31, 1997
  - Python 1.6 - September 5, 2000
- **Python 2.0 - October 16, 2000**
  - Python 2.1 - April 15, 2001
  - Python 2.2 - December 21, 2001
  - Python 2.3 - July 29, 2003
  - Python 2.4 - November 30, 2004
  - Python 2.5 - September 19, 2006
  - Python 2.6 - October 1, 2008
  - Python 2.7 - July 4, 2010
- **Python 3.0 - December 3, 2008**
  - Python 3.1 - June 27, 2009
  - Python 3.2 - February 20, 2011
  - Python 3.3 - September 29, 2012
  - Python 3.4 - March 16, 2014
  - Python 3.5 - September 13, 2015
  - Python 3.6 - December 23, 2016
  - Python 3.7 - June 27, 2018
  - Python 3.8- Oct 14,2019

  - ○ Python 3.9 –Oct 5,2020
  - ○ **Current Stable version is 3.10.4**

History of PYTHON:

- The implementation of Python was started in the December 1989 by **Guido Van Rossum** at National Research Institute in Netherland.
- In February 1991, van Rossum published the code.
- In 1994, Python 1.0 was released with new features like: lambda, map, filter, and reduce.
- Python 2.0 added new features like: list comprehensions, garbage collection system.
- On December 3, 2008, Python 3.0 (also called "Py3K") was released. It was designed to rectify fundamental flaw of the language.
- *ABC programming language* is said to be the predecessor of Python language which was capable of Exception Handling and interfacing with Amoeba Operating System.
- Python is influenced by following programming languages:
  - ABC language.
  - Modula-3

- **Using Python we can Develop the Following Applications**
  - ❖ Web Applications
  - ❖ Desktop GUI Applications
  - ❖ Network Programming
  - ❖ Gaming Applications
  - ❖ Data Analysis Applications
  - ❖ Console Based Applications
  - ❖ Business Applications
  - ❖ Audio and Video Based Applications

- **Steps to install python for windows**

Go to Google

|

Type: python download for windows

|

Click on Python.org

|

Click on Download python 3.10.0

|

It will be download python 3.10.0.Exe

|

Open Python 3.10.0.Exe

|

Activate the Checkbox Add Python path

|

Click on Install now

|

Click on Next, Next, Install

Different Ways to Execute Python Code

- **Using Interactive Mode**
- **Using Script Mode**
- **Using Python IDLE**
- **Using Pycharm Editor**

Interpreter vs. Compiler:

Interpreter	Compiler
1. It will check line by line and executes	1. It will check Whole program at a time
2. It gives the result line by line	2. It gives whole output at a time
3. If any error occurs interpreter stops Hence it shows only one error	3.It Checks all statements in the program and show all errors in program
4. It will not generate executable file	4. If no errors in the program then it generates executable file
5. It always executes only source code Ex: Html,Perl,Javascript,Python	5. It executes exe file Ex: C,C++,C#,Java

Python Indentation:

- Most of the programming languages like C, C++, Java use braces { } to define a block of code, but Python uses indentation.

- A code block (body of a function, loop etc.) starts with indentation and ends with the first un- indented line. The amount of indentation is up to you, but it must be consistent throughout that block.

- Generally four whitespaces are used for indentation and is preferred over tabs. Here is an example.

```
a=10
if a==10:
   print("true")
```

- The indentation in Python makes the code look neat and clean.
- Indentation can be ignored in line continuation.
- But it makes the code more readable.

Python Comments:

- Comments are very important while writing a program.

- Python Interpreter ignores comment.

- Python supports two types of comments:

Single line comment:

- In case user wants to specify a single line comment, then comment must start with (#)

Multi line comment:

- If we have comments that extend multiple lines, one way of doing it is to use hash (#) in the beginning of each line.

- Multi lined comment can be given inside triple quotes.

Identifiers:

- A Name in python program is called identifier.

- It can be a class name or function name or variable name.

- **Rules to define identifier:**

- Alphabet symbols(either lowercase or uppercase)
- Digits(0 to 9)
- Underscore symbol(_)
- Identifier should not start with digit.
- Identifiers are case sensitive
- We cannot use keywords as Identifier.
- If Identifier starts with Underscore then it is private.

Python Variables:

- Variable is a name which is used to refer memory location. Variable also known as identifier and used to hold value.
- In Python, we don't need to specify the type of variable because python is a dynamically typed.
- Variable names can be a group of both letters and digits, but they have to begin with a letter or an underscore.
- It is recommended to use lowercase letters for variable name.

Multiple Assignments:

- Python allows us to assign a value to multiple variables in a single statement which is also known as multiple assignment
- We can apply multiple assignments in two ways either by assigning a single value to multiple variables or assigning multiple values to multiple variables.

➢ **Assigning single value to multiple variables**

Ex1:

```
a=b=c=10
print(a)
print(b)
print(c)
```

Ex2:

```
a=b=c=10
print(a,b,c,sep=",")
```

➢ **Assigning multiple values to multiple variables**

Ex1:

```
a,b,c=10,20,30
print(a)
print(b)
print(c)
```

Ex2 :

```
a,b,c=10,20,30
print(a,end=",")
print(b,end=",")
print(c)
```

- The values will be assigned in the order in which variables appears.

Python Keywords:

- Python Keywords are special reserved words which convey a special meaning to the interpreter.
- Each keyword have a special meaning and a specific operation. These keywords can't be used as variable. Following is the List of Python Keywords.

Ex to get keyword list:

```
import keyword
print(keyword.kwlist)
```

Output:

```
['False', 'None', 'True', '__peg_parser__', 'and', 'as', 'assert', 'async', 'await',
'break', 'class', 'continue', 'def', 'del', 'elif', 'else', 'except', 'finally', 'for',
'from', 'global', 'if', 'import', 'in', 'is', 'lambda', 'nonlocal', 'not', 'or', 'pass',
'raise', 'return', 'try', 'while', 'with', 'yield']
```

Python Data Types:

- Data type represent the type of data present inside a variable.
- Every value in Python has a data type.
- In python not required to specify the type explicitly.
- Based on value, the type will be assigned automatically.
- Python is Dynamically Typed Language

Python contain the following Data types:

1. None
2. Numeric
3. List
4. Tuple
5. Set
6. String
7. Range
8. Dictionary or Mapping
9. Bytes
10. Bytearray
11. Frozenset

None:

- When we have a variable which is not assigned any value is called None.
- Normally in any language the keyword can be use null, but in python we use None.

Ex:

```
a=None
print(a)
print(type(a))
```

Numeric: It classified Four Types

1. Int
2. Float
3. Complex
4. Bool

Examples:

```
a=10
print(a)
print(type(a))
```

```
a=10.9
print(a)
print(type(a))
```

Ex to convert from int to float:

```
a=10
print(a)
print(type(a))

b=float(a)
print(b)
print(type(b))
```

Ex to convert from float to int:

```
a=10.8
print(a)
print(type(a))

b=int(a)
print(b)
print(type(b))
```

Program to accept input from user:

```
name=input("Enter your name:")
print("your name is:",name)
```

Program to accept integer values from user:

```
a=input("Enter Num1:")
b=input("Enter Num2:")
print("result is:",a+b)
```

Note: by default the values entered by user at runtime will be treated as string

```
a=input("Enter Num1:")
print(type(a))
b=input("Enter Num2:")
print(type(b))
print("result is:",a+b)
```

Note: we have to convert from string to integer

```
a=input("Enter Num1:")
print(type(a))
x=int(a)
b=input("Enter Num2:")
print(type(b))
y=int(b)
print("result is:",x+y)
```

Complex Data Type: A complex number is in the form of real and imaginary

Ex: a+bj

10+20j

a and b are integers or float values

Ex:

```
a=9
b=8
c=complex(a,b)
print(c)
print(type(c))
print(c.real)
print(c.imag)
```

- Note: Complex data type has some inbuilt attributes to retrieve the real part and imaginary part.
- We can use complex type generally in scientific Applications and electrical engineering Applications.

Bool:

- We can use this data type to represent Boolean values.
- The only allowed values for this data type are: True and False
- Internally Python represents True as 1 and False as 0

Ex:

```
a=5
b=6
c=a<b
print(c)
print(type(c))
print(int(c))
print(int(True))
print(int(False))
print(True+True)
print(4+True)
```

String:

- str represents String data type.
- A string is a sequence of characters enclosed within single quotes or double quotes.
- String can be create by using '' or "" or ''' ''' or """ """
- Triple quotes can be used for multiline string.
- String is immutable, that can’t be modifying directly.

Ex:

```
s="durgasoft"
print(s)
print(type(s))

s='durga\'s'
print(s)

s='''durgasoft
hyderabad
```

```
maitrivanam'''
print(s)

s="""durgasoft
hyderabad
maitrivanam"""
print(s)
```

List:

- List is ordered collection of elements.
- We can create list by using [].
- List will allow duplicate elements.
- List will allow different data type elements
- List is mutable, once we create a list that can be modified.

Ex:

```
l= [10,10,"durga",23.4,20,'A']
print(l)
print(type(l))
```

Tuple:

- Tuple is ordered collection of elements same as list.
- We can create tuple by using () but brackets are optional.
- Tuple will allow duplicate elements.
- Tuple will allow different data type elements
- Tuple is immutable, once we create a tuple that cannot be modified.
- Creating a tuple with one element is bit different, to create a tuple with one element ,after element we have to give comma(,)

Ex1:

```
t=(10,10,"durga",23.4,20,'A')
print(t)
print(type(t))
```

Ex2:

```
t=(10,)
print(t)
print(type(t))
```

Set:

- Set is unordered collection of unique elements.
- We can create by using {}
- Set will allow different data type elements.
- Set will not allow duplicate elements.
- Set is mutable, we can modify the set.
- To create an empty set then we use set() function.

Ex1:

```
s=set()
print(s)
print(type(s))
```

Ex2:

```
s={10,"sai",12.4,'A',10}
print(s)
print(type(s))
```

Dict:

- Dict is a collection of items.
- In dict each item can be a pair i.e. key and value.
- We can create dict by using {}

- In dict keys are immutable and must be unique.
- In dict values are mutable and no need to be unique.
- In dict keys and values can be of any data type.
- In dict keys cannot be modified but values can be modified.

Ex1:

```
d={}
print(d)
print(type(d))
```

Ex2:

```
d={1:"sai",2:"mohan",'a':'apple',3:34.5,1:"moha
n"}
print(d)
print(type(d))
```

Range:

- Range is used to generate range of values.
- By default range starts from 0
- Range is immutable; we cannot change the range values.

Ex:

```
r=range(10)
print(r)
print(type(r))

r=range(list(10))
print(r)

r=range(list(2,20,2))
print(r)
```

Python Operators:

- Operator is a symbol which is used to perform required operations.
- 5+2=7, here +, = are operators and 5, 2 are operands.
- Python supports the following operators
  1. Arithmetic
  2. Relational
  3. Assignment
  4. Logical
  5. Membership
  6. Identity
  7. Bitwise

Arithmetic Operators:

- Arithmetic operators are used to perform mathematical operations like addition, subtraction, multiplication and division.

Operator	Description
//	It perform floor division
+	It perform Addition
-	It Perform subtraction
*	It Perform Multiplication
/	It Perform Division
%	Return remainder after division
**	Perform Exponent or Power Operator

Ex:

```
print(2+3)
print(4-1)
print(5*2)
print(5**2)
print(5/2)
print(5//2)
print(5%2)
```

Relational Operators:

- Relational operators are used for comparing the values. It either returns True or False according to the condition. These operators are also known as Comparison Operators.

Operator	Description
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to
==	Equal to
!=	Not Equal to

Ex:

```
print(3<4)
print(4>5)
print(4<=4)
print(5>=5)
print(3==3)
print(3!=3)
```

Assignment Operators:

- Assignment Operators are used to assigning values to variables.

Operator	Description
=	Assignment
/=	Divide and Assign
+=	Add and Assign
-=	Subtract and Assign
*=	Multiply and Assign
%=	Modulus and Assign
**=	Exponent and Assign
//=	Floor division and Assign

Ex:

```
a=5
print(a)
a+=2
print(a)
a-=2
print(a)
a*=2
print(a)
a/=2
print(a)
a//=2
print(a)
```

```
a=5
print(a)
a**=2
print(a)
a%=2
print(a)
```

Logical Operators:

- Logical operators are used on conditional statements and perform logical operations.

Operator	**Description**
And	Logical AND(When both conditions are true output will be true)
Or	Logical or(if any one condition is true output will be true).
Not	i.e Reverse , complement the condition

Ex:

```
print(2<3 and 4<5)
print(2<3 and 4>5)

print(2<3 or 4>5)
print(2<3 or 1<2)
print(1>2 or 2>3)

print(not(2<3))
print(not(1>2))
```

Membership Operators:

- Membership operators are used to test if a given value or object is present in sequence or not.

Operator	Description
In	Return true if variable is available in sequence , else false.
Not in	Return true if variable is not available in sequence, else false.

Ex:

```
lst=[2,3,4,5]
print(2 in lst)
print(100 in lst)
print(100 not in lst)
print(2 not in lst)
```

Identity Operators:

- These operators are used to check whether the variables or objects are having same identity or different identity.

Operator	Description
Is	Return true if identity of two operands are same, else false.
Is not	Return true if identity of two operands are not same, else false.

Ex:

```
a=2
b=2
print(a is b)
print(a is not b)

a=2
b=3
print(a is b)
print(a is not b)
```

Id ( ):

- It is a built-in function which is used to find memory address of variables or any object in python.
- id () will return unique integer value.

Ex:

```
a=2
b=2

print(id(a))
print(id(b))
print(id(a)==id(b))

a=2
b=3

print(id(a))
print(id(b))

a='mohan'
b='Mohan'
```

```
print(id(a))
print(id(b))

print(id(a)==id(b))
```

Bitwise Operators:

- In Python, bitwise operators are used to performing bitwise calculations on integers.
- The integers are first converted into binary and then operations are performed on bit by bit, hence the name bitwise operators. Then the result is returned in decimal format.

1. Bitwise AND (&)

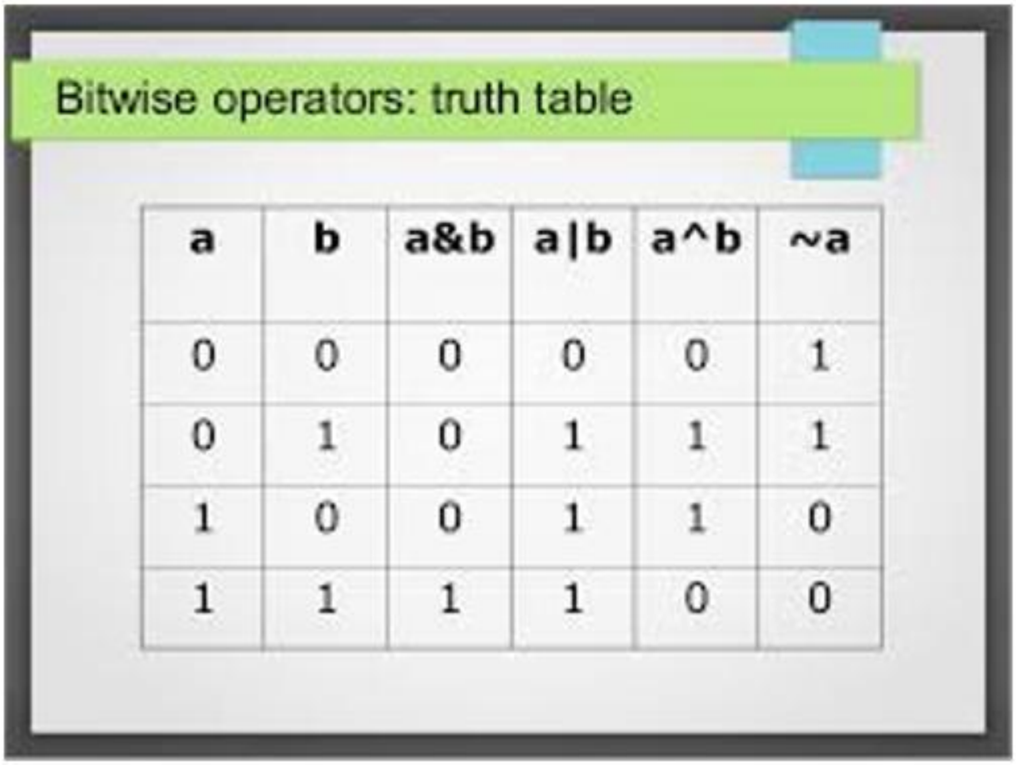
2. Bitwise OR (|)

3. Bitwise XOR (^)

4. Bitwise NOT (~)

5. Left shift (<<)

6. Right shift (>>)



Bitwise operators: truth table

a	b	a&b	a|b	a^b	~a
0	0	0	0	0	1
0	1	0	1	1	1
1	0	0	1	1	0
1	1	1	1	0	0

Ex:

```
a=9
b=5
print(bin(a))
print(bin(b))
print(bin(a&b))
print(a&b)
print(a|b)
print(a^b)
print(~a)  #~a=-(a+1)
print(a<<2)
print(a>>2)
```

Control Flow Statements:

- Flow control describes the order in which statements will be executed at runtime.

- Control flow statements divided into 3 parts.

1. Conditional statements
2. Iterative statements
3. Transfer statements

Conditional statements:

- If
- If else
- Nested if
- elif

if :

Syntax: if condition:
Statements

- If condition is true then statements will be executed.

Ex:

```
i=100
if i==100:
    print("true")
```

if else:

Syntax: if condition:
Statements
else:
Statements

- If condition is false then statements of else block will be executed.
- Else will execute only if the condition is false.

Ex:

```
i=100
if i==100:
    print("true")
else:
    print("false")
```

Ex: **program to check the given number is even or odd**

```
i=int(input("Enter a number:"))
if i%2==0:
    print(i,"is even")
else:
    print(i,"is odd")
```

Ex: **program to find biggest of two numbers**

```
i=int(input("Enter num1:"))
j=int(input("Enter num2:"))
if i>j:
    print(i,"is big")
else:
    print(j,"is big")
```

Ex: **program to check the given number is in between 1 and 100**

```
n=int(input("Enter number:"))
if n>=1 and n<=100:
    print("The number",n,"is in between 1 to
100")
else:
    print("The number", n, "is not in between 1
to 100")
```

nested if:

Syntax:
```
if condition:
        Statements
    else:
          if condition:
              Statements
          else:
               Statements
```

Ex:

```
i=int(input("Enter num1:"))
j=int(input("Enter num2:"))
if i>j:
    print(i,"is greater than",j)
else:
    if i<j:
        print(i,"is less than ",j)
    else:
        print(i,"is equal to",j)
```

elif:

Syntax:
```
if condition1:
        Statements
    elif condition2:
        Statements
    elif condition3:
        Statements
    else:
        Statements
```

Ex: **program to find biggest of three numbers**

```
i=int(input("Enter num1:"))
j=int(input("Enter num2:"))
k=int(input("Enter num3:"))

if i>j and i>k:
    print("Biggest number is:",i)
elif j>k:
    print("Biggest number is:",j)
else:
    print("Biggest number is:",k)
```

Ex:

```
n=int(input("Enter number:"))
if n==1:
    print("ONE")
elif n==2:
    print("TWO")
elif n==3:
    print("THREE")
elif n==4:
    print("FOUR")
else:
    print("Invalid number")
```

Ex:

```
n=int(input("Enter number:"))
if n==1:
    print("ONE")
else:
    if n==2:
        print("TWO")
    else:
        if n==3:
            print("THREE")
        else:
```

```
if n==4:
    print("FOUR")
else:
     print("Invalid number")
```

Iterative statements:

- If we want to execute a group of statements multiple times then we should go for iterative statements.
- **For loop**
- **While loop**

For loop:

- For loop is used to iterate the elements of collection or sequence what the order they appear.

Syntax: for variable in sequence:
Statements

Ex:

```
l=[10,20,30,40,50,60]
for i in l:
    #print(i)
     print(i,end=' ')
```

Ex:

```
for i in [10,"sai",23.4,50,'A']:
    print(i,type(i))
```

Ex:

```
for i in range(10):
    print(i)
```

Ex: **program to display even numbers from 0 to 20**

```
for i in range(21):
    if i%2==0:
        print(i)
```

Ex: **program to display sum of first n numbers**

```
n=int(input("Enter number:"))
sum=0
for i in range(n+1):
    sum=sum+i
print("sum of first",n,"numbers:",sum)
```

Nested for loop:

- A for loop which is having one more for loop within it is called nested for loop.
- A for loop inside another for loop is called nested for loop.

Syntax: for variable in sequence:
Statements
for variable in sequence:
Statements

Note: For every iteration of outer loop, inner loop should finish its all iterations then only the outer loop starts with its next iteration.

Ex:

```
numlsit=[1,2,3]
charlist=['a','b']

for n in numlsit:
    print(n)
```

```
for c in charlist:
    print(c)
```

While loop:

- If we want to execute a group of statements iteratively until some condition false, then we should go for while loop.

Syntax: while condition:
Statements

Ex:

```
i=1
while i<=10:
    print(i)
    i+=1
```

Ex: **program to display sum of first n numbers**

```
n=int(input("Enter number:"))
sum=0
i=1
while i<=n:
    sum=sum+i
    i+=1
print("Sum of first",n,"numbers is:",sum)
```

Ex: **Infinite loop**

```
i=0
while True:
    i+=1
    print("Hello",i)
```

Transfer statements:

- Transfer statements alter the way a program gets executed. These statements are often used in loop statements.

- **Break**
- **Continue**
- **Pass**

Break:

- **We can use this inside loops to stop the execution based on condition.**

Ex:

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

Ex:

```
i=1
while i<=10:
    print(i)
    i = i + 1
    if i == 5:
        break
```

Continue:

- **We can use this to skip the current iteration and continue with next iteration.**

Ex:

```
for i in range(10):
    if i==5:
        continue
    print(i)
```

Ex:

```
for i in range(10):
    if i==5 or i==7:
        continue
    print(i)
```

Ex:

```
i=1
while i<=10:
    print(i)
    i+=1
    if i==5:
        i+=1
        continue
```

Pass:

- **It is a keyword in python.**
- **If we want to define the empty block then we use pass keyword.**

Ex:

```
i=10
if i==10:
    pass
else:
    pass
```

Ex:

```
for i in range(10):
    pass
```

Working with String functions:

- **To get complete string functions**

Ex:

```
print(dir(str))
```

- **To get complete string functions and its description**

Ex:

```
help(str)
```

- **String indexing and slicing**

Ex:

```
# -9  -8  -7  -6  -5  -4  -3  -2  -1
#  d   u   r   g   a   s   o   f   t
#  0   1   2   3   4   5   6   7   8

s="hyderabad"
print(s[0])
print(s[-1])
print(s[8])
print(s[12])#index error:string index out of
range

s="hyderbad"
print(s[2:7])
print(s[:6])
print(s[1:])
print(s[:])
print(s[0:9:1])
print(s[::-1])
print(s[7:1]) #empty string
print(s[-1:-5])
print(s[-5:-1])
```

- **String Concatenation and Multiplication**

Ex:

```
s1="durga"
s2="soft"
```

```
print(s1+s2)
print(s1+" "+s2)
print("mohan"+" "+"kumar")

print(s1*3)
print((s1+" ")*3)
print(("mohan"+" ")*3)
```

- **String Split and Max Split**

Ex:

```
#s="d u r g a s o f t"
s="python is very easy and it is oop and it is interpreter"
print(s)
s1=s.split(" ",3)
print(s1)
print(type(s1))

for i in s1:
    print(i)
```

- **String capitalize and title**

Ex:

```
s="pyThon is vEry eaSy"
print(s)
s1=s.capitalize()
print(s1)
#or
print(s.capitalize())
print(s.title())
```

- **String upper and lower**

Ex:

```
s="durgasoft"
print(s)
print(s.upper())

s="DURGASOFT"
print(s)
print(s.lower())
```

- **String count**

Ex:

```
s="python is very easy and it is oop and it is interpreter"
substring="is"
print(s.count(substring))
print(s.count("and"))
print(s.count("is"))
print(s.count("x"))
print(s.count(" "))
print(s.count('a'))
```

- **String replace**

Ex:

```
s="my name is durga"
print(s)
s1=s.replace("durga","mohan")
print(s1)
#or
#print(s.replace("durga","mohan"))
```

- **String join**

Ex:

```
print(",".join("MOHAN"))
print(" ".join(["sai","mohan","raj","durga"]))
```

- **String reverse**

Ex:

```
print(" ".join(reversed("SAI")))
#or
s="SAI"
print(s[::-1])
```

- **String sort**

Ex:

```
s="python is very easy"
print(s)
s1=s.split(" ")
print(s1)
print(type(s1))
s1.sort()
print(s1)
s1.sort(reverse=True)
print(s1)
```

- **String swap case**

Ex:

```
s="DuRgAsOfT"
print(s)
print(s.swapcase())
```

- **String strip , lstrip, rstrip**

Ex:

```
s="  durga  "
print(s)
print(s.strip(" "))

s="adurga"
print(s)
print(s.strip('a'))
print(s.lstrip('a'))
print(s.rstrip('a'))
```

- **String length**

Ex:

```
print(len("durga soft"))
```

- **String find , index ,rindex**

Ex:

```
s="python is very easy and it is oop and it is
interpreter"

print(s.find("is"))
print(s.find("x"))
print(s.index("is"))
#print(s.index("x"))
print(s.rindex("is"))
```

- **String max, min**

Ex:

```
s="durgasoft"
print(max(s))
print(min(s))
s="DURGASOFT"
print(max(s))
print(min(s))
```

- **String partition**

Ex:

```
s="python is very easy and it is oop"
s1=s.partition("is")
```

```
print(s1)
print(type(s1))
```

- **String startswith , endswith**

Ex:

```
s="durgasoft"
print(s.startswith('a'))
print(s.startswith('D'))
print(s.startswith('d'))
print(s.endswith('T'))
print(s.endswith('t')
```

- **String isdigit ,isalpha, isalnum**

Ex:

```
s="12345"
print(s.isdigit())
s="12345a"
print(s.isdigit())

s="abcd"
print(s.isalpha())
s='abcd12'
print(s.isalpha())

s="abcd"
print(s.isalnum())
s="1234"
print(s.isalnum())
s="123abc"
print(s.isalnum())
s="$%#%"
print(s.isalnum())
```

Working with List functions:

- **List index**

Ex:

```
#  0  1   2    3  4    5      6   7   8  9
l=[10,20,"sai",30,40,"durga",'A',23.4,50,60]
print(l[1])
print(l[-1])
print(l[-4])
#print(l[10])#Index Error: list index out of
range
print(l[1:6])

#nested list
#  0  1   2           3         4    5  6   7
#               0   1    2
l=[10,20,"sai",[30,40,"durga"],'A',23.4,50,60]
print(l[3])
print(l[3][1])
print(l[-5][-2])
```

- **List slice**

Ex:

```
#  0  1   2    3  4    5      6   7   8  9
l=[10,20,"sai",30,40,"durga",'A',23.4,50,60]

print(l[1:5])
print(l[:4])
print(l[1:])
print(l[:])
```

- **Adding elements to the List : insert, append, extend**

Ex:

```
l=[2,3,4]
print(l)
l[1]=33# 1 is index and 33 is value
```

```
print(l)

l=[2,3,4]
print(l)
l.insert(1,33)# 1 is index and 33 is value
print(l)

l=[2,3,4]
print(l)
l.append(33)
l.extend([45,67,89,"sai"])
print(l)
```

- **Delete elements from the List : remove, pop, clear, del**

Ex:

```
l=[10,20,30,40,50]
print(l)
l.remove(20)
#l.remove(33) #ValueError: list.remove(x): x not in list

l=[10,20,30,40,50]
print(l)
l.pop(2)
l.clear()

del l
print(l)
```

- **List concatenation and multiplication**

Ex:

```
l1=["sai","durga","ram"]
l2=[10,20,30]
print(l1+l2)
print(l1*3)
```

- **List sort**

Ex:

```
l=[2,6,9,4,3,5,7,1]
print(l)
l.sort()
print(l)
l.sort(reverse=True)
print(l)
```

- **List copy**

Ex:

```
l1=[10,20,30,40,50]
print(l1)

#l2=l1
l2=l1.copy()

l1[1]=33
print(l1)
print(l2)

print(id(l1))
print(id(l2))
```

- **List count and index**

Ex:

```
l=[10,20,30,10,20,10,10,20,10,20]
print(l.count(10))
print(l.count(100))
print(l.index(30))
```

- **List creation by accepting values at runtime**

Ex:

```
l=[]

item1=int(input("Enter int vlaue:"))
item2=input("Enter string value:")
item3=float(input("Enter float value:"))

'''l.append(item1)
l.append(item2)
l.append(item3)'''

l.extend([item1,item2,item3])

print(l)
```

- **List creation by accepting values at runtime**

Ex:

```
l=[]

n=int(input("Enter length of the list:"))

for i in range(n):
    x=int(input("Enter a value:"))
    l.append(x)
print(l)
```

- **List creation using range**

Ex:

```
print(list(range(10)))
print(list(range(2,20)))
print(list(range(2,21,3)))#start,stop,step
```

Working with tuple functions:

Ex:

```
t=(10,20,30)
print(t)
print(type(t))
#t[1]=33 #TypeError: 'tuple' object does not support item assignment
del t
print(t)
```

- **Tuple membership test**

Ex:

```
print(10 in t)
print(100 in t)
print(100 not in t)
```

- **Tuple Len, max, min, sum**

Ex:

```
t=(10,20,-34,23.4,50)
print(len(t))
print(max(t))
print(min(t))
print(sum(t))
print(sum(t,4))
```

- **Converting a string into tuple**

Ex:

```
s="durgasoft"
print(s)
print(type(s))

t=tuple(s)
```

```
print(t)
print(type(t))
```

- **Converting a List into tuple**

Ex:

```
l=[10,20,30,40]
print(l)
print(type(l))

t=tuple(l)
print(t)
print(type(t))
```

- **Tuple packing and unpacking**

Ex:

```
#packing
a=10
b=20
c=30

t=a,b,c
print(t)
print(type(t))

#unpacking
t=(100,200,300)

a,b,c=t
print("a=",a)
print("b=",b)
print("c=",c)
```

Working with set functions:

- **Creating a set**

Ex:

```
s=set()
print(s)
print(type(s))
```

- **Add and update methods of set**

Ex:

```
s={10,20,30,"sai",45.7,10}
print(s)

s.add(33)
print(s)

s.update([44,55,78,"durga"])
print(s)
```

- **Discard , Remove, Clear methods of set**

Ex:

```
s={10,20,30,40,50}
print(s)
#s.discard(10)
#s.remove(10)
s1=s.discard(100) #none
print(s1)
#s.remove(100) #KeyError: 100
s.clear()
print(s)
```

- **Deleting a set**

Ex:

```
s={10,20,30,40,50}
print(s)

del s
print(s) #NameError: name 's' is not defined
```

- **Set operators : union, intersection, difference, symmetric difference**

Ex:

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

print(A|B)
print(A.union(B))

print(A&B)
print(A.intersection(B))

print(A-B)
print(A.difference(B))
print(B-A)

print(A^B)
print(A.symmetric_difference(B))
```

- **Set Len, max, min, sum**

Ex:

```
s={10,34.5,-45,89,99}
print(len(s))
print(max(s))
print(min(s))
print(sum(s))
print(sum(s,9))
```

- **Set membership test**

Ex:

```
s={10,34.5,-45,89,99}
print(10 in s)
print(9 in s)
print(99 not in s)
```

Working with dict functions:

- **Creating a dict**

Ex:

```
d={}
print(d)
print(type(d))

d={"eid":1234,"ename":"sai"}
print(d)
```

- **Access value from dict**

Ex:

```
d={"eid":1234,"ename":"sai"}
print(d)

print(d["ename"])
print(d.get("ename"))

#print(d["age"])#KeyError: 'age'
print(d.get("age"))  #none
```

- **Change the value from dict**

Ex:

```
d={"eid":123,"ename":"sai"}
print(d)

d["ename"]="mohan"
print(d)

d["age"]=37
print(d)
```

- **Deleting a dict**

Ex:

```
d={"eid":123,"ename":"sai"}
print(d)
del d["ename"]
print(d)
#del d
#print(d) #NameError: name 'd' is not defined
```

- **Dict copy**

Ex:

```
d={1:"sai",2:"mohan",3:"raja"}
print(d)

#d1=d
d1=d.copy()
print(d1)

d[1]="durga"
print(d)
print(d1)
print(id(d1))
print(id(d))
```

- **Dict items, values, keys functions**

Ex:

```
d={1:"sai",2:"mohan",3:"raja"}
print(d)
print(d.items())
print(d.keys())
print(d.values())
```

- **Dict len and membership test**
- **Note: in dict membership test is only applicable for keys**

Ex:

```
d={1:"sai",2:"mohan",3:"raja"}
print(1 in d)
print("sai" in d)
print(len(d))
```

- **Dict pop and pop item methods**

Ex:

```
d={1:"sai",2:"mohan",3:"durga"}
print(d)

#print(d.pop(1))
#print(d)

print(d.popitem())
print(d)
```

Working with bytes and byte array:

- Bytes represent byte numbers just like an array.
- The only allowed values for bytes are 0 to 255.
- Bytes is immutable, we cannot change byte values once we create.
- Byte array represent byte numbers just like an array.
- The only allowed values for byte array is 0 to 255.
- Byte array is mutable; we can change byte array values once we create.

Ex:

```
x=[10,20,30,40,255]
print(x)
print(type(x))

#b=bytes(x)
b=bytearray(x)
print(b)
print(type(b))
'''print(b[0])
print(b[1])
print(b[2])
print(b[3])'''

b[1]=33
for i in b:
    print(i)
```

Working with frozen set:

- Set and frozen set is almost same but only the difference is ,set is mutable Whereas frozen set is immutable.
- We cannot change frozen set once create it.

Ex:

```
s={10,20,30,40}
print(s)
print(type(s))
s.add(34)
print(s)

fs=frozenset(s)
print(fs)
print(type(fs))

fs.add(45) #AttributeError: 'frozenset' object
has no attribute 'add'
print(fs)
```

Python Functions:

- 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.
- Function is a group of related statements that perform a specific task.
- Functions help break our program into smaller and modular chunks.
- As our program grows larger and larger, functions make it more organized and manageable.
- Furthermore, it avoids repetition and makes code reusable
- The main advantage of functions is code Reusability.
- Functions will provide improve readability of the program.

Python supports 2 types of functions

- Built in Functions
- User Defined Functions

1. **Built in Functions:**

- The functions which are coming along with Python software automatically are called built in functions or pre-defined functions
  Ex: print (), id (), type ()   etc.

2. **User Defined Functions:**

- The functions which are developed by programmer explicitly according to their requirements are called user defined functions.

Syntax to create user defined functions:

Syntax :

```
def    function_name(parameters) :
  """ doc string"""
   Statements…
   return value
```

- Keyword def marks the start of function header.
- A function name to uniquely identify it. Function naming follows the same rules of writing identifiers in Python.
- Parameters (arguments) through which we pass values to a function. They are optional.
- A colon (:) to mark the end of function header.
- Optional documentation string (docstring) to describe what the function does.
- One or more valid python statements that make up the function body. Statements must have same indentation level (usually 4 spaces).
- An optional return statement to return a value from the function.

- **Creating a function**

Ex:

```
#creating a function
def f1():
    for i in range(10):
         print("Hello")

#calling a function
f1()
f1()
```

Parameters:

- Parameters are inputs to the function. If a function contains parameters, then at the time of calling, compulsory we should provide parameter values otherwise we will get error.
- **Create a function to find square of given number**

Ex:

```
def square(n):
    print("square is:",n*n)

#square()#TypeError: square() missing 1 required positional argument: 'n'
square(3)
square(5)
```

- **Create a function to find whether given number is even or odd**

Ex:

```
def iseven(n):
    if n%2==0:
        print(n,"is even")
    else:
        print(n,"is odd")
```

```
iseven(10)
iseven(n=int(input("Enter a number:")))
```

- **Create a function to find biggest of two numbers**

Ex:

```
def biggest(a,b):
    if a>b:
        print(a,"is big")
    else:
        print(b,"is big")

biggest(23,45)
```

- **Create a function to find biggest of three numbers**

Ex:

```
def biggest(a,b,c):
    if a>b and a>c:
        print(a,"is big")
    elif b>c:
        print(b,"is big")
    else:
        print(c,"is big")

biggest(2,3,4)
biggest(4,3,2)
biggest(3,4,2)
biggest(4,2,3)
```

- **Create a function to find sum of two numbers**

Ex:

```
def sum(a,b):
    print("sum is:",a+b)

sum(10,20)
sum(23.4,56.7)
sum(a=int(input("Enter
Num1:")),b=int(input("Enter Num2:")))
```

Return Statement:

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

- **Create a function to find sum of two numbers and return the result**

Ex:

```
def add(a,b):
    #print("sum is:",a+b)
    return a+b

r=add(10,20)
print("sum is:",r)
```

- **In python a function can return multiple values**

Ex:

```
def sum_sub(a,b):
    sum=a+b
    sub=a-b
    return sum,sub

x,y=sum_sub(20,10)
```

```
print("sum is:",x)
print("sub is:",y)
```

- **Program for arithmetic operations using functions**

Ex:

```
def add(x, y):
   return x + y
def subtract(x, y):
   return x - y
def multiply(x, y):
   return x * y
def divide(x, y):
   return x / y
print("Select operation.")
print("1.Add")
print("2.Subtract")
print("3.Multiply")
print("4.Divide")
#  input from the user
choice = input("Enter choice(1/2/3/4):")
num1 = int(input("Enter first number: "))
num2 = int(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))
else:
   print("Invalid input")
```

Function Variables:

- Python supports 2 types of variables
    1. Local variables
    2. Global variables

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, from outside of function we cannot access.

Ex:

```
def f1():
    a=10
    print(a)
    #print(b)#NameError: name 'b' is not
defined

def f2():
    b=20
    print(b)
    #print(a)#NameError: name 'a' is not
defined

f1()
f2()
```

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 or program.

Ex:

```
a=10 #global variable

def f1():
    print(a)
```

```
def f2():
    print(a)

f1()
f2()
```

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

Ex:

```
a=10

def f1():
    global a
    a=99
    print(a)

def f2():
    print(a)

f1()
f2()
```

Ex:

```
def f1():
    global a
    a=99
    print(a)

def f2():
    print(a)
```

```
f1()
f2()
```

Note: If global variable and local variable is having the same name then we can access global variable inside a function as follows.

Ex:

```
a=10

def f1():
    a=20
    print(a)
    print(globals()['a'])

def f2():
    print(a)

f1()
f2()
```

Types of arguments:

```
def f1(a,b):
     print(a+b)

f1(10,20)
```

- **a, b are formal arguments whereas 10,20 are actual arguments**
- **There are 4 types are actual arguments are allowed in Python.**

1. positional arguments
2. keyword arguments
3. default arguments
4. Variable length arguments

Positional arguments:

- The arguments which are passed to a function in correct positional order is called positional arguments.
- The number of arguments and position of arguments must be matched. If we change the order then result may be changed.
- If we change the number of arguments then we will get error.

Ex:

```
def sub(a,b):
    print("sub is:",a-b)

sub(20,10)
sub(10,20)
sub(10,20,30) #TypeError: sub() takes 2 positional arguments but 3 were given
```

Keyword arguments:

- The arguments which are passed to a function by using a keyword or parameter name are called keyword arguments.
- Here the order of arguments is not important but number of arguments must be matched.

Ex:

```
def f1(name,msg):
    print("Hello:",name,msg)

#keyword arguments
f1(name="mohan",msg="Good morning")
f1(msg="Good morning",name="mohan")
#positional arguments
f1("mohan","Good morning")
f1("Good morning","mohan")
```

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.

Ex:

```
def f1(name,msg):
    print("Hello:",name,msg)

#keyword arguments
f1(name="mohan",msg="Good morning")
f1(msg="Good morning",name="mohan")
#we can use one positional and one keyword
arguments
#but make sure that positional arguments should
be first then after keyword arguments
f1("mohan",msg="Good morning")
#f1(name="mohan","Good morning") #SyntaxError:
positional argument follows keyword argument
```

Default arguments:

- Sometimes we can provide default values for our positional arguments

Ex:

```
def f1(cource="python"):
    print("course is:",cource)

f1("c")
f1()
```

- If we are not passing any course then only default value will be considered
- After default arguments we should not take non default arguments

Ex:

```
#def f1(cource="python",name): #SyntaxError:
non-default argument follows default argument
def f1(name,cource="python"):
    print(name,"course is:",cource)

f1("sai","c")
f1("mohan")
f1("ram")
```

Variable length arguments:

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

Ex:

```
def f1(*a):
    print(a)

f1()
f1(10)
f1(10,20)
f1(10,20,30)
```

Ex:

```
def add(*n):
    s = 0
    for i in n:
        s=s+i
    print("sum is:",s)

add(10,20)
add(2,3,4)
add(2,3,4,5,6,7)
add(10,20,30,40,50,60,70)
```

What is *args and **kwargs in python:

- *args allows us to pass no of argument values to the function.
- *args will store the values in tuple collection
- The name *args is optional we can use any name like *n or *x
- *args is same as variable length arguments.

Ex:

```
def add(*args):
    s = 0
    for i in args:
        s=s+i
    print("sum is:",s)

add(10,20)
add(2,3,4)
add(2,3,4,5,6,7)
add(10,20,30,40,50,60,70)
```

- **kwargs allows us to pass no of keyword argument values to the function.
- **kwargs will store the values in dict collection
- The name **kwargs is optional we can use any name like **n or **x

Ex:

```
def f1(**kwargs):
    print(kwargs)
    for k,v in kwargs.items():
        print(k,"=",v)

f1(a=10,b=20,c=30)
f1(eid=1234,ename="sai",eaddress="hyd",esal=450
00)
```

Positional only arguments:

- If we declare a function for accepting only positional arguments then that function will work only with positional arguments.
- If we pass keyword arguments then it will give error

Ex:

```
def add(a,b,/):
    print("sum is:",a+b)

add(10,20)
#add(a=10,b=20) #TypeError: add() got some
positional-only a
```

Keyword only arguments:

- If we declare a function for accepting only keyword arguments then that function will work only with keyword arguments.
- If we pass positional arguments then it will give error

Ex:

```
def add(*,a,b):
    print("sum is:",a+b)

#add(10,20) #TypeError: add() takes 0
positional arguments but 2 were given
add(a=10,b=20)
```

Function Aliasing:

- For the existing function we can give another name, which is nothing but function aliasing.
- If we delete a name of the function still we can call a function with alias name.

Ex:

```
def f1():
    print("Hello")

f2=f1
#del f1
f2()
print(id(f1))
print(id(f2))
```

Nested Function:

- **A Function which contains one more Functions within it is known as Nested Function.**

Ex:

```
def f1():
    print("hello")

    def f2():
        print("Hai")

        def f3():
            print("welcome")

        f3()

    f2()
```

```
f1()
```

Ex:

```
def multi(a):
    def mul(b):
        def mu(c):
         return a*b*c
        return mu
    return mul
y=multi(10)(20)(2)
print(y)
```

Recursive Function:

- A function that calls itself is known as Recursive Function.
- We can reduce length of the code and improves readability.
- We can solve complex problems very easily.
- Factorial is a recursive function which calls every time itself.
- factorial(n)= n*factorial(n-1)

```
factorial(3)   =3*factorial(2)
               =3*2*factorial(1)
               =3*2*1*factorial(0)
               =3*2*1*1
               =6
```

Ex:

```
def factorial(n):
    if n==0:
        result=1
    else:
         result=n*factorial(n-1)
    return  result
print("Factorial of 4 is:",factorial(4))
print("Factorial of 5 is:",factorial(5))
```

Anonymous or Lambda Functions:

- Sometimes we can declare a function without any name, such type of nameless functions is called anonymous functions or lambda functions.
- The main purpose of anonymous function is just for instant use (for one time usage).
- Using lambda we can write concise code.
- Using lambda we can reduce the length of the code.
- Using lambda we can write mostly single line expressions.
- Using lambda we can improve readability of program.
- We can define lambda function by using lambda keyword

Syntax: lambda argument_list : expression

- **lambda function to find square of given number**

```
s=lambda n:n*n
print(s(4))
```

- **lambda function to find sum of two numbers**

```
s=lambda a,b:a+b
print(s(10,20))
```

- **lambda function to find biggest of two numbers**

```
s=lambda a,b:a if a>b else b
print(s(34,56))
```

- **lambda function to find biggest of three numbers**

```
s=lambda a,b,c:a if a>b and a>c else b if b>c
else c
print(s(2,3,4))
```

- **lambda function to find a number is even or odd**

```
s=lambda n:"is even" if n%2==0 else "is odd"
print(s(5))
print(s(8))
```

- Sometimes we can pass a function as an argument to other function in this situation lambda is best choice.
- In python some functions will expect other function as an argument in this situation lambda is best choice.
- We can use lambda functions very commonly with filter(),map() and reduce() functions, because these functions expect other function as argument.

Filter ():

- This function is used to filter the values from given sequence based on condition.
- **syntax:** filter(function, sequence)
- here first parameter function is for conditional check
- here second parameter sequence can be a list or tuple or set

Ex:

```
#without lambda
def iseven(x):
    if x%2==0:
        return True
    else:
        return False

L=[2,3,4,5,6,7,8,9,10]
L1=list(filter(iseven,L))
print(L1)
```

Ex:

```
#with lambda
L=[2,3,4,5,6,7,8,9,10]
```

```
L1=list(filter(lambda x:x%2==0,L))
print(L1)
```

Map ():

- for every element present in sequence, apply some condition and Return the new sequence of elements for this purpose we use map function.
    - **syntax:** filter(function, sequence)
- here first parameter function is for conditional check
- here second parameter sequence can be a list or tuple or set

Ex:

```
#without lambda
def dbl(x):
    return 2*x

L=[2,3,4,5,6,7,8,9,10]
L1=list(map(dbl,L))
print(L1)
```

Ex:

```
#with lambda
L=[2,3,4,5,6,7,8,9,10]
L1=list(map(lambda x:2*x,L))
print(L1)
```

- **We can apply map () function on multiple lists also, but make sure all list should have same length.**

Ex:

```
L1=[2,3,4,5,7]
L2=[5,6,7,8,9]
L3=list(map(lambda x,y:x+y,L1,L2))
print(L3)
```

Reduce ():

- Reduce () function reduces sequence of elements into a single element by applying the specified function.
- **syntax:** filter(function, sequence)
- Reduce () function present in functools module and hence we should write import statement.

Ex:

```
#without lambda
from functools import reduce

def f1(x,y):
    return x+y

L=[10,20,30,40,50,60,70]
result=reduce(f1,L)
print(result)
```

Ex:

```
#without lambda
from functools import reduce
L=[10,20,30,40,50,60,70]
result=reduce(lambda x,y:x+y,L)
print(result)
```

Modules in Python:

- A group of functions, variables and classes saved to a file, which is nothing but module.
- Every Python file (.py) acts as a module.
- Module is for reusability of the program code into other program.
- Once we write logic in one program that logic we can use into any other program by importing the program name as module.

- **In python modules can be of two types**

  1. User defined modules
  2. Predefined or built-in modules

Working User defined modules:

Sample.py :

```
x=123
def add(a,b):
    print("sum is:",a+b)
def mul(a,b):
    print("Product is:",a*b)
```

- Here sample.py acts as module.
- Sample module contains one variable and 2 functions.
- Variables and functions of module called as members.
- If we want to use members of module in our program then we should import that module.
- import modulename
- We can access members by using module name.
- modulename.variable
- modulename.function()

Test.py:

```
import sample
print(sample.x)
sample.add(10,20)
sample.mul(30,20)
```

- Here Test.py reusing the functionality of sample.py file.

Renaming a module or module aliasing:

- Just giving other name to a module is called module aliasing.
- Make sure that once we change the module name then we can access members of the modules with alias name only.

Test.py:

```
import sample as sm
print(sm.x)
sm.add(10,20)
sm.mul(30,20)
```

- **Here sample is original module name and sm is alias name.**
- **We can access members by using alias name sm**

from ... import :

- We can import particular members of module by using from ... import.
- The main advantage of this is we can access members directly without using module name.

Test.py:

```
from sample import x,add,mul
print(x)
add(10,20)
mul(20,30)
```

- **We can import all members of a module as follows**

Test.py:

```
from sample import *
print(x)
add(10,20)
mul(20,30)
```

Renaming a member or member aliasing:

Test.py:

```
from sample import x as a,add as sum
print(a)
sum(10,20)
```

- Here a is alias name of x and sum is alias name of add
- We can also import multiple modules

Ex with importing multiple modules

Sample.py :

```
x=123
def add(a,b):
    print("sum is:",a+b)
def mul(a,b):
    print("Product is:",a*b)
```

Sample1.py:

```
x=200
def sub(a,b):
    print("mul is:",a*b)
```

Test.py:

```
from sample import *
from sample1 import *
print(x)
add(2,3)
mul(3,6)

print(x)
sub(7,3)
```

Note: in the above program we notice that the value of x is 200 for two times, the reason is recent module value only effect that is from sample1, to avoid this try to change the program like below.

Test.py :

```
from sample import *

print(x)
add(2,3)
mul(3,6)

from sample1 import *
print(x)
sub(7,3)
```

The Special variable __name__:

- For every python program, a special variable __name__ will be added internally.
- This variable stores information regarding whether the program is executed as an individual program or executed from other program.
- If the program executed as an individual program then the value of this variable is __main__
- If the program executed from some other program then the value of this variable is the name of module where it is defined.
- Hence by using this __name__ variable we can identify whether the program executed directly or executed from some other program.

Sample.py :

```
def f1():
    if __name__=='__main__':
        print("Executed as an individual
program")
    else:
        print("Executed from some other
program")
f1()
```

Test.py :

```
import sample
sample.f1()
```

To get the information of particular module:

Test.py :

```
import sample
help(sample)
```

Working with pre-defined modules:

- When we install python software then automatically lot of modules gets installed for ready to use.
- In case any modules are not available then we have to install explicitly.

Working with math module:

- This module defines several functions which can be used for mathematical operations.

Ex:

```
from math import *

print(factorial(3))
print(sqrt(4))
print(pow(3,2))
print(log(10,2))
print(ceil(34.2))
print(floor(34.9))
```

Working with random module:

- This module defines several functions to generate random numbers.
- We can use these functions while developing games, in cryptography and to generate random numbers on fly for authentication.
- Functions of random module:
    1. Random()
    2. Randint()
    3. Uniform()
    4. Randrange()
    5. Choice()

Random () :

- This function always generate some float value between 0 and 1 (not inclusive)

```
from random import *

for i in range(5):
    print(random())
```

Randint () :

- This function always generate random integer values between two given numbers (inclusive)

```
from random import *

for i in range(10):
    #print(randint(2,21))
    print(randint(1000,2000))
```

Uniform () :

- This function always generate some float value between two given numbers (not inclusive)

```
from random import *

for i in range(10):
    print(uniform(2,21))
```

Randrange () :

- This function always generates random range values.
- **Syntax:** randrange(start,stop,step)

```
from random import *

for i in range(10):
    print(randrange(2,21,3))
```

Choice ():

- This function will not generate random values but it return random object.

```
 from random import *

l=["sai","mohan","raj","ram",10,34.5,"manoj"]
x=choice(l)
print(x)
print(type(x))
#in python everything is called as object
```

Working with Date time module:

- This module is used to work with date and time related tasks.

Example to display current system date and time

```
import datetime
x=datetime.datetime.now()
print(x)

from datetime import *
x=datetime.now()
print(x)
print(x.date())
print(x.time())
```

Example to display current system date and time

```
import datetime

x=datetime.datetime.now()
print(x)

import datetime as dt

x=dt.datetime.now()
print(x)

from datetime import *
x=datetime.now()
print(x)
```

strftime () function :

- This function is used to display date objects into string format

```
from datetime import *

x=datetime.now()
print(x)
print(x.strftime('%A')) #weekday full version
print(x.strftime('%a')) #weekday short version
print(x.strftime('%Y')) #year full version
print(x.strftime('%y')) #year short version
print(x.strftime('%B')) #month full version
print(x.strftime('%b')) #month short version
```

Working with Calendar module:

Example to display the calendar for specified year and month

```
from calendar import *

y=1947
m=8
print(month(y,m))
print(month(2021,11))
```

Example to display entire the year calendar

```
from calendar import *
print(calendar(2021,2,1,9))#y,w,l,c

#y---year
#w---width of the characters
#l---lines per week
#c---column separation

from calendar import *

print(leapdays(1980,2021))
print(isleap(2020))
print(isleap(2022))
```

Arrays in Python:

Array:

- Array is a user defined collection of similar data type elements.
- Array index always starts with zero and ends with size – 1
- In general arrays are two types

1. Single dimension array
2. Two dimension array

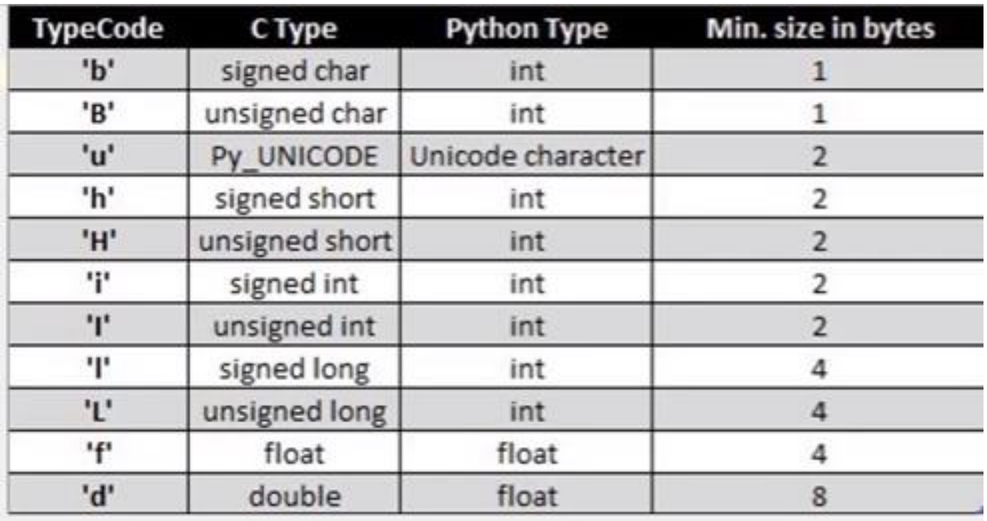
- In python we can create single dimension array by using array module.
- Using array module we cannot create two dimension array.
- To create single and two dimension arrays then we use numpy module.

Example with single dimension array:

```
import array
                    #     0   1   2   3
A=array.array('i',[10,20,30,40])
print(A)
print(type(A))

A=array.array('f',[12.3,45.6,78.9,45.2])
print(A)
print(type(A))
```

Note: to create arrays in python using array module then we use type code



TypeCode	C Type	Python Type	Min. size in bytes
'b'	signed char	int	1
'B'	unsigned char	int	1
'u'	Py_UNICODE	Unicode character	2
'h'	signed short	int	2
'H'	unsigned short	int	2
'i'	signed int	int	2
'I'	unsigned int	int	2
'l'	signed long	int	4
'L'	unsigned long	int	4
'f'	float	float	4
'd'	double	float	8

```
from array import *
A=array('i',[10,20,30,40])
print(A)
print(type(A))
print(A.typecode)
print(A[2])
A.insert(1,34)
print(A)
A.append(45)
print(A)
A.extend([67,89,90])
print(A)
A.remove(30)
print(A)
A.reverse()
print(A)
```

To display array elements using for loop:

```
from array import *

A=array('i',[100,200,300,400,500,600])

for i in A:
    print(i)

for i in range(len(A)):
    print(A[i])
```

Copying elements from one array to another:

```
from array import *

A=array('i',[2,3,4,5,6])
print(A)
#B=A
#print(B)
B=array(A.typecode,[i*2 for i in A])
print(B)
```

Creating array by accepting elements at runtime:

```
from array import *

A=array('i',[])

n=int(input("Enter length of the array:"))

for i in range(n):
    x=int(input("Enter value:"))
    A.append(x)
print(A)
```

Numpy:

- Numpy stands for numerical python
- Numpy is a multi-dimensional array processing package.
- Numpy is not available by default; we have to install this explicitly.
- Using numpy we can create single and two dimension arrays.
- Numpy provides several methods to create arrays.

Numpy methods:

1. Array()
2. Linespace()
3. Logspace()
4. Arange()
5. Zeros()
6. Ones()

Steps to install numpy package:

- Go to file menu in pycharm editor
- Click on settings
- Expand your project from left hand side
- Click on python interpreter
- Click on + symbol
- Type in search : numpy
- Select numpy

- Click on install package.

Examples with numpy arrays:

Ex1:

```
import numpy

#A=numpy.array([10,20,30,40,50])
A=numpy.array([10,20,30,40,50.8],int)
print(A)
print(type(A))
print(A.dtype)
print(A.ndim)
print(A.size)
print(A.shape)
```

Ex2:

```
from numpy import *

A=linspace(2,20,7)#start,stop,no of parts
print(A)

A=logspace(3,15,6)
print(A)

A=arange(2,20,3)#start,stop,step
print(A)

A=zeros(6,int)
print(A)

A=ones(6,int)
print(A)
```

Ex3:

```
import numpy as np

A=np.array([[10,20,30],
            [34,56,78],
            [78,23,45]])

print(A)
print(A.dtype)
print(A.ndim)
print(A.shape)
print(A.size)
print(A[1][2])
```

Ex4:

```
import numpy as np

A=np.array([[-10,20,30],
            [34,-56,78],
            [78,23,-45]])
print("max element is:",A.max())
print("min element is:",A.min())

print("col wise max element is:",A.max(axis=0))
print("col wise min element is:",A.min(axis=0))

print("row wise max element is:",A.max(axis=1))
print("row wise min element is:",A.min(axis=1))
print(A.sum())
```

Note: in numpy arrays dimensions are called axis
axis 0 means columns, axis 1 means rows

flatten () method:

- This method is used to collapse all rows from two dimension array into a single row.

```
import numpy as np
```

```
A=np.array([[-10,20,30],
            [34,-56,78],
            [78,23,-45]])

print(A)
print(A.ndim)

B= A.flatten()
print(B)
print(B.ndim)
```

THANK YOU