Python

History

- Made by **Guido van Rossum** in **Feb 1991** at **Centrum Wiskunde Informatica, Netherlands.**
- It is successor of **ABC** Language
    - **ABC- this language made for simple use of computer because in 1980 personal computer introduced so user use computer easily that's why this language made and Guido van Rossum is the part of team who implemented this.**
- Python is the mini project that is developed by **Guido van Rossum** that is use in **Amiba** Distributed Operating System also this language use for another operating system so it is platform independent language.
- Python 1.0 released in 1994.
- python 2.0 released in 2000.
- Python 3.0 released in 2008.
- There is no backward compatibility. so python 2.0 is different as compare to python 3.0 function name may be same but their working is different.
- It is extensible so we can add more pre defined function in the library.

Scripting vs programming language

- Programming language compiled while scripting language interpreted.
- Scripting languages are easy as compare to programming language.
- There is no begin or end of scripting lang while in programming begin and end must.
- Scripting lang little bit slow as compare to programming lang.

Note- Python is scripting as well as programming both.

About Python

- Object oriented
- Procedure oriented
- Imperative means no need of declaration
- Dynamically typed
- Platform Independent
- Auto memory management
- Large library

In Python you can develop variety of application

- Console app
- GUI/Desktop app
- Web app
- Mobile app
- IOT app
- AI app
- Data Science app
- ML app

Python Installation

python.org

- Download Installer
- Run the Installer

How to use Python for programming?

- **Python Shell | IDLE Shell**
  - >>> String Prompt

- **How to and where to write python program?**
  - Python program are stored in **.py** file.
  - Create a file with any name with **.py** extension. ex A1.py
  - A1.py

```
print("Hello World")
```

- **How to run python program?**
  - Using cmd (Command Line).
    - python file name
      - ex- python A1.py
  - Using double Click.
  - Using Shell.

Basics Of Python

Comments in Python

- **Single Line Comment**

  Single line comments in Python start with a hash symbol (#) and continue until the end of the line.

```
#this is single comment
```

- **Multi-Line Comment**

  Multi-line comments in Python are created using triple quotes (''' or """). These can span multiple lines and are useful for longer documentation or temporarily disabling multiple lines of code.

```
'''
This is a multi-line comment
```

```
You can write as many lines as you want
Python will ignore everything between the triple quotes
'''
```

Constants

- Data=Information=Constant
- Numbers (25, 2.5, 0, -2, 0.005, 3+4i)
- Strings (“Chirag” , “Muzaffarnagar” ,“This is long sentence”)

Variables

- Variables are nothing but reserved memory location to store values.
- This means that when you create a variable you reserve some space in memory.
- While the program is running, variables are accessed and sometimes changed, i.e. a new value will be assigned to the variable.
- Variable name can be a combination of alphabet, digit and underscore.
- variable name cannot start with digit.
- variable names are case sensitive.
- keywords cannot be used as variable name.
- In C/C++/Java
  - int a;
  - char str[10];
  - bool flag;
  - float b;

  But in **Python** we don't need to declare variables with their types - we can directly assign values:

```
x = 10        # integer
name = "John"  # string
```

```
flag = True    # boolean
```

- How to print value of variable?

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

Python is Dynamically Typed Language

- Not only the value of a variable may change during program execution but the type as well.
- type (Data Type) is category of Data.
- **How to Know the Type of variable?**
  - Example of built-in data types in python
    - int
    - float
    - str
    - complex
    - bool
  - To check the type of a variable, you can use the type() function:

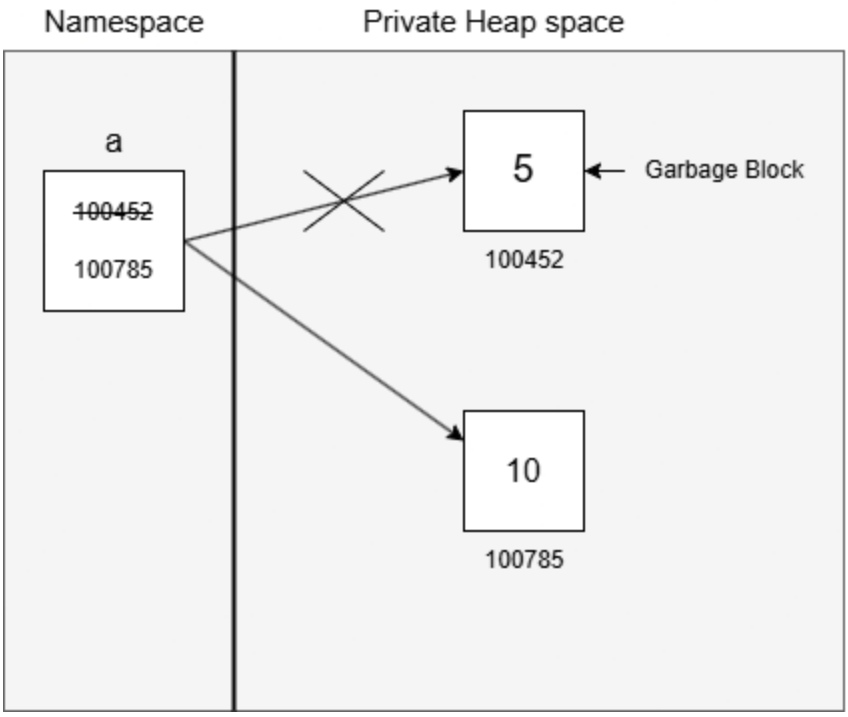
```
x = 10
print(type(x))  # Output: <class 'int'>

x = "John"
print(type(x))  # Output: <class 'str'>
```

Memory Allocation

- Every data in python is an object.

    - class == common noun
    - object == proper noun
- Object consumes memory in **private heap space.**
- The job of Python memory manager is to allocate memory for object in private heap space.
- id=address=reference=location number of an object.
- when an object is not referenced , It is called Garbage block.
- The job of Python Garbage collector is to release garbage blocks.
- a=5
- a=10



- id() use for check the address or id of a variable
- Example of using id():

```
a = 10
b = 10
```

```
print(id(a))  # Both variables point to same memory location
print(id(b))  # because they have the same value
```

- When objects share the same value, Python may reuse the memory location to optimize memory usage. This is called interning.

Keywords

- Predefined words or reserved words
- There are 35 keywords in python.
- **False, None, True** are keywords as well as data (constant).
- Two ways to know the list of keywords in python.
  - Using Python's interactive help mode:
    - In Python shell, type 'help()' and press Enter
    - Then type 'keywords' to see the list of all keywords
  - Using Python's keyword module:

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

  - This will display all the Python keywords as a list.
- **What is module in python?**
  - any .py file is a module.
- **What reusable elements can reside in a module?**
  - Variables
  - Functions
  - Classes
- **How to use reusable elements of any module in your python code?**
  - ex- example.py

```
a=5
```

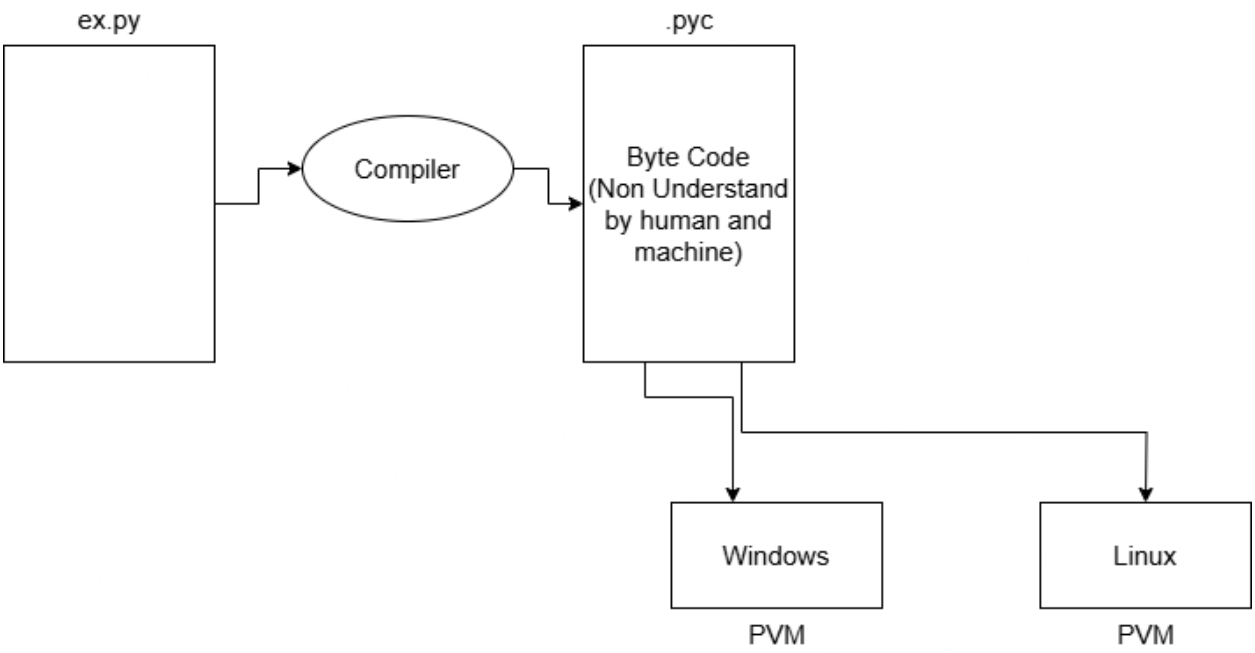
- A1.py

```
import example
print(example.a)
```

Data Types

- Every data in python is an object.
- Object is an instance of a class.
- Class defines category of data.
- Category of data==Data types.
- **Built-In Data Types**
  - **Numbers**
    - int - (5, 0, -2, 458 , 45789522265263)
    - float - (4.0, 0.0, -8.5, 457.125)
    - complex - (7+8j, 4+2j, -8+6j, 5.5+8j)
    - Note: there is no concept of double, long in python.
  - **String**
    - str - (”Chirag”, ‘dhiman’, “”””hello”””, ‘’’hi’’’)
  - **Boolean**
    - bool - (True/False)
- **int , float, complex, bool, (str, range, list, tuple, set, dict) these are sequence, NoneType. Discuss in next lecture.**

How to develop and run Python program?



- .py is the source code of python
- compiler convert it to .pyc file which is non understand by human and machine
- so PVM is software which has internal software like just-in-time-compiler
- so just-in-time-compiler is not compiler or interpreter.
- it translate line of byte code into machine understandable language and gave to CPU and after that repeat this process again and again.
- so that's way we achieve platform independency in python.
- PVM is platform dependent in python.
- **How to compiled python file?**
  - python -m py_compile file name
    - ex- python -m py_compile A1.py
    - This will create a **pycache** directory containing the compiled .pyc file
    - The .pyc file contains the bytecode that PVM can execute

Type Conversion

- You can not add different type value but you add int & float.

```
a=10
b=5.0
c="chirag"
x=a+b # 15.0 so int convert automatic into float
z=a+c # gave TypeError
```

- **Type conversion function**
  - **int()**

```
x=int("123") # 123
x=int("abc") # ValueError
x=int("5a") # ValueError
x=int(4.5) # 4
x=int(2+8j) # ValueError
```

  - **float**

```
x=float(5) # 5.0
x=float("2.5") # 2.5
x=float("abc") # ValueError
x=float(2+8j) # ValueError
```

  - **bool()**
    - in bool class every non zero value is True & Zero is False

```
x=bool(45) # True
x=bool(-85) # True
x=bool(0) # False
x=bool(2.5) # True
x=bool(0.0) # False
x=bool("Hello") # True
x=bool("") # False
```

```
x=bool(5+8j) #True
x=bool(0+0j) #False
```

- **str()**

```
x=str(5) # '5'
x=str(4.0) # '4.0'
x=str(True) # 'True'
x=str(2+5j) # '2+5j'
```

- **Note: bool() & str() will never give ValueError**
- **complex()**

```
x=complex("abc") # ValueError
x=complex(2) # 2+0j
x=complex("2+5j") # 2+5j
x=complex(2.5) # 2.5+0j
```

How to obtain unicode of any character?

- **ord()** function is used to obtain unicode of a character.
- Argument must be a single character string.

```
x=ord('A') #65
x=ord('a') #97
x=ord(' ') #32
x=ord('0') #48
```

How to obtain character of any unicode?

- **chr()** function is used to obtain character of a unicode.

```
x=chr(65) # A
x=chr(68) # D
```

Taking input from user

- we use **input()** function.
- input() function always return **str** type value.
- input() can take at most **one argument of str type.**
- input() ends when you hit **enter key.**

```
x=input() # 24 so x=24 of type str
x=input("Enter number") # Enter Number24 so x=24 of type str
```

- **print() function when prints multiple values, by default they are separated by spaces.**

Number System

- There are 4 number system decimal, binary, octal, hexadecimal.
- In Python built-in function to convert from one number system to another.
- **oct() to convert in octal number system represent with prefix 0o.**
- **bin() to convert in binary number system represent with prefix 0b.**
- **hex() to convert in hexadecimal number system represent with prefix 0x.**
- these function return **str** type value.

```
x=25
y=oct(x) # 0o31
y=bin(x) #0b11001
y=hex(x) #0x19

y=0o31
print(y) # 25
```

Operators

- Operator requires operands (data) to perform its job.

- ex- **3+4 here + is operator & 3, 4 both are operands.**
    - **Types Of Operators**
        - Arithmetic Operators
        - Relational Operators
        - Logical Operators
        - Bitwise Operator
        - Assignment Operators
        - Identity Operators
        - Membership Operators
- **No ++,- - Operator in Python**
- **No ?: Conditional Operator in Python**

Arithmetic Operator

- **** (exponential operator or power operator)**

```
x= 2**5 # 32
x= 3**3 # 27
x= -2**2 # -4
x= (-2)**2 # 4
x= 2**-2 # 0.25
x= 2**0.5 # 1.4142135623730951
x= 0.5**0.5 # 0.7071067811865476
```

- **/ (True Division) gave result always in float and mathematically correct.**

```
x=5/2 #2.5
x=25/5 #5.0
```

- **// (Division) . If numerator & denumerator both are int gave result as int if any one float so result as float.**

```
x=25//5 #5
x=26//5 #5
x=25//5.0 #5.0
x=5//2.0 #2.0
x=2//5 #0
x=2//5.0 #0.0
```

- *** (Multiply)**

```
x=2*5 #10
x=2*2.5 #5.0
x=(3+6j)*2 #6+12j
x="chirag"*3 #'chiragchiragchirag'   this is repetation operator
x="chirag"*0.2 #ValueError only we can multiply with int value.
x="chirag"*"chirag" #ValueError
```

- **+ (Addition)**

```
x=2+8 #10
x=0.5+4.5 #5.0
x="chirag"+5 #TypeError
x="chirag"+"dhiman" #'chiragdhiman"  this is concatination operator
```

- **- (Subtraction)**

```
x=5-8 #-3
x=4.2-1.2 #3.0
```

- **% (Modulation)**

```
x=5%2 #1
x=5%2.0 #0
x=26%50 #26
```

Relational Operator

- **>, <, >=, <=**
- **==, != these are the identity operator**
- identity operator does not give any error
- Relational operator always gives result **True/False**

```
5>4 #True
3!=0 #True
4<4 #False
"Ram">"Bharat" #True
5>"5" #TypeError
5==5.0 #True
5>3.5 #False
5=="5" #False
3+4j>3-5j #TypeError
3+4j==3-4j #False
3+4j==3+4j #True
5>4>3 #True
10>8>6>7 #False
True != False #True
"Muz"=="Muz" #True
"Muz"=="muz" #False
True+5==6 #True
'A'==65 #False
ord('A')==65 #True
```

Logical Operator

- **and, or, not**
- **and**

expression 1	and	expression 2	result
True	and	True	True

expression 1	and	expression 2	result
True	and	False	False
False	and	any	False

- **or**

expression 1	or	expression 2	result
False	or	False	False
False	or	True	True
True	or	any	True

- **not**

not	expression	result
not	True	False
not	False	True

- for non-bool operands logical operators gives non-bool result.
- not operator always gives result in bool type.

```
3>4 and 4>2 # False
5>0 and 6>0 # True
not 5>4 #False
not 5 #False
not -5 #False
not 0 #True
"Chirag" or "dhiman" #"Chirag"
5 or 6 #5
not "Chirag" False
"Chirag" and "Dhiman" #"Dhiman"
4 and 7 #7
"5" and 2 #2
"" and 5 # ""
5 or 5/0 # 5
5 and 5/0 #ZeroDivisionError
```

```
0 or 5 #5
0 and 5 #0
```

Bitwise Operator

- **&, |, ^, ~, >>, <<**
- **& (Bitwise and)**

1	&	1	1
1	&	0	0
0	&	1	0
0	&	0	0

- **| (Bitwise or)**

1	|	1	1
1	|	0	1
0	|	1	1
0	|	0	0

- **~ (Bitwise not)**

~	1	0
~	0	1

- **^ (Bitwise XOR)**

1	^	1	0
1	^	0	1
0	^	1	1
0	^	0	0

- **>> (Bitwise right shift)**

77>>2 = 01001101

shift 2 bits so number **00010011**

- **<< (Bitwise left shift)**

22<<3 = 00001100

shift 3 bits so number **01100000**

```
25&49 #17
85|58 #127
77>>2 #19
22<<3 #96
```

Assignment Operator

- **=, +=, -=, *=, /=, //=, &=, ^=, |=, >>=, <<=**

```
a,b=5,6
a+=2 #7
b*=4 #24
```

Identity Operator

- **is, is not**

```
x=5
y=5
x is y #True
x=5.0
y=5.0
x is y # False
x="chirag"
y="chirag"
x is y # True
x is not y #False
```

Membership Operator

- **in, not in**

- works only with sequence.

```
x="chirag dhiman"
"man" in x # True
"ags" in x # False
"chi" not in x # False
```

Print

- When you print multiple values using print() function, values are by default separated with a space.
- keyword argument **sep** is used to mention what separater we want.
- At the end print method prints a new line character by default.
- Keyword argument **end** is used to mention what end character we want.
- We can print value by using format specifier.

```
a,b,c=10,20,30
print(a,b,c) # 10 20 30
print(a,b,c,sep='-') # 10-20-30
print(a,b,c,sep=',') # 10,20,30

print(a)    # 10
print(b,c)  # 20 30
print(a,b,c,end="not new line") # 10 20 30not new line
print(a,b,c,sep='-',end='$') # 10-20-30$

print("sum of",a,"&",b,"is',c) # Sum of 10 & 20 is 30
print("Sum of %d & %d is %d"%(a,b,c)) # Sum of 10 & 20 is 30
print("Sum of {} & {} is {}".format(a,b,c)) # Sum of 10 & 20 is 30
```

Flow Control Statements

- Decision Control Statement
- Iterative Control Statement

Decision Control Statement

- if
- if else
- if elif else
- Single line if else
- **Switch** is not there in python.
- In python, block is created with the help of indentation.

if Condition-

- Syntex-
  - if condition:
    - code
    - code
    - .....

```
x=5
a=int(input("enter a number"))
if a==x:
   print("%d is equal to %d"%(a,x))
```

if else Condition-

- Syntex-
  - if Condition:
    - code
    - code
  - else:
    - code

        - code

```
x=5
if x>0:
    print("%d is Positive number"%x)
else:
    print("%d is negative or 0"%x)
```

if elif else Condition-

- Syntex-
    - if Condition:
        - code
        - code
    - elif Condition:
        - code
        - code
    - elif Condition:
        - code
        - code
    - else:
        - code

```
x=5
if x>5:
    print("%d is greater then 5"%x)
elif x<5:
    print("%d is less then 5"%x)
else:
    print("%d is equal 5"%x)
```

single line if else

- exp 1 if condition else exp2
- if condition true so exp1 execute otherwise exp2 execute

```
print("true" if 5<10 else "false") # false
```

Iterative Control Statement

- while
- for
- there is no concept of do while loop in python.

while

- while condition:
  - code
  - code

```
i=5
while i<7:
    print("hi") "hi"
    i+=1       "hi"
```

Transfer Control Statement

- break
  - It terminates execution of loop.

```
i,x=1,5
while i<=10:
    if i==x:
        break
```

```
    print(i,end=' ')   # 1 2 3 4
    i+=1
```

- continue
  - control moves to the next iteration

```
i,x=1,5
while i<=10:
    i+=1
    if i==x:
        continue
    print(i,end=' ')  # 2 3 4 6 7 8 9 10
```

- pass
  - to make empty block

```
if 2>5:
    print("if execitue")
else:
    pass          # else run
```

for loop

- **for** var **in** sequence:
  - code
  - code
- What is sequence?
  - sequence is collection of elements
  - sequence is container for elements
  - Data Types whose objects are sequence:
    - range

        - list
        - str
        - tuple
        - set
        - dict

```
for x in "chirag":
    print(chr(ord(x)-32,end=' ') # CHIRAG
```

range

- range is a class.
- range is a sequence.
- range is an iterable object.
- range is group of integers only.
- **How to create object of range?**
    - r1=range(beg,end,step)
    - beg: starting value | first element of range
    - end: end value | excluded | last element is just lesser then the end value.
    - step: difference between consecutive values
    - range(beg,end,step)
    - range(beg,end) by default step=1
    - range(end) by default step=1,beg=0
- **How to use for loop to print range elements?**
    - for var in range_object:
        - print(var,end=’ ‘)

```
r1=range(1,10,1)
for x in r1:
    print(r1,end=' ') # 1 2 3 4 5 6 7 8 9

for x in range(2,15,2):
    print(x,end=' ') # 2 4 6 8 10 12 14

for x in range(2,10,-1):
    print(x,end=' ') # empty

for x in range(10,1,-2):
    print(x,end=' ') # 10 8 6 4 2

for x in range(10,20):
    print(x,end=' ') # 10 11 12 13 14 15 16 17 18 19

for x in range(20,10):
    print(x,end=' ') # empty

for x in range(10):
    print(x,end=' ') # 0 1 2 3 4 5 6 7 8 9
```

list

- list is a class
- list is a sequence
- list is iterable
- list elements are indexed
- list is growable and shrinkable
- list can store heterogeneous elements
- list is mutable.
- **How to create a list object?**

- l1=[10, 20, 30]
- l1=[] # Empty list
- l1=[”Pune”, ”Delhi”, ”Muzaffarnagar”]
- l1=[10, ’abc’, 3.5, 5+6j, True]

- **Concept of indexing**
  - [10, 20, 30, 40]
  - 0 1 3 4
- **Concept of negative indexing**
  - 0 1 3 4
  - [10, 20, 30, 40]
  - -4 -3 -2 -1
- **How to access list elements?**
  - listObject[index]
- **How to edit list elements?**
  - listObject[index]=new Value
- **How to add more value in the list?**
  1. **Add new value at the end of list**
     - listObject.append(value)
  2. **Insert new value at specific index**
     - listObject.insert(index,value)

```
l1=[10,20,30,40]
l1[0] # 10
l1[1] # 20
l1[4] # IndexError
l1[-1] #40
l1[1]=25
l1 # [10, 25, 30, 40]
```

```
l1[4]=50 #IndexError
l1.append(50)
l1 # [10,25,30,40,50]
l1[-1] #50
l1.insert(2,45)
l1 #[10,25,45,30,40,50]
l1.insert(10,100) # Not give error its behave like append so add as a last value
l1 #[10,25,45,30,40,50,100]
l1[-1] #100
l1.insert(-2,140)
l1 #[10,25,45,30,40,140,50,100]
l1.insert(-10,500)
l1 #[500,10,25,45,30,40,140,50,100]
```

- **Common method for sequences**
  - these method doesn’t change anything on actual sequence.
  - len(sequence)
  - sum(sequence)
  - sorted(sequence) #return a sorted list
  - max(sequence)
  - min(sequence)

```
l1=[10,20,75,45,65,28]
len(l1) # 6
sum(l1) # 243
sorted(l1) # [10,20,28,45,65,75]
max(l1) # 75
min(l1) # 10
```

- **Relational Operator**

```
l1=[10,20,30]
l2=[10,15,13]
```

```
l1>l2 # True ( l1 ist element = to l2 ist element so compare 2nd elemet now
l1 2nd                element > l2 2nd element )
l1!=l2 # True
```

- **un-packing**

```
l1=[10,20,30]
a,b,c=l1
a # 10
b # 20
c # 30
x,y,z,m=l1 # Error
```

- **packing**

```
a,b,c=10,20,30
l1=[a,b,c]
```

- **concatination Operator +**

```
l1=[10,20,30]
l2=[100,200,300]
l1+l2 # [10,20,30,100,200,300]
l2+l1 # [100,200,300,10,20,30]
# + return a list not changes in actual list
l1-l2 # TypeError
```

  - — operator doesn’t work on list

- **Repeatation Operator ***

```
l1=[10,20,30]
l1*3 # [10,20,30,10,20,30,10,20,30] this is also return a list
```

- **accessing list elements using for loop**

```
l1=[10,20,30]
for i in l1:
    print(i,end=' ') # 10 20 30
```

- for hetrogeneous elements in list common fun doesn’t work

```
l1=[10,25.2,"chirag"]
sorted[l1] # Error
max(l1) # Error
min(l1) # Error
sum(l1) # Error
len(l1) # 3 It will work

l2=[2+4j,8+6j,-9+3j]
sorted(l2) # error
max(l2) # error
min(l2) # error
sum(l2) # 1+13j it will work
len(l2) # 3
```

- **list inside a list**

```
l1=[[10,0,45],[78,45,12,45],["chirag",1,2.5]]
for i in l1:
    print(i)

# [10,0,45]
#  [78,45,12,45]
#  ["chirag",1,2.5]
```

- **list() Function**
  - list is a class while list() is a function
  - list() take at most one argument of sequence type.

```
l1=list()
l1 # [] empty list
l1=list(2,5,5) # TypeError
l1=list(67) # TypeError
l1=list(range(10,51,10))
l1 # [10,20,30,40,50]
```

- **list class function**
  - index()

    ```
    l1=[10,20,30,40,50,60]
    l1.index(30) # 2
    l1.index(60) # 5
    l1.index(100) # ValueError
    ```

  - remove()

    ```
    l1=[10,20,30,40,50,60]
    l1.remove(40)
    l1 # [10,20,30,50,60]
    l1.remove() # Error argument must be present
    l1.remove(100) # ValueError
    ```

  - pop()

    ```
    l1=[10,20,30,40,50]
    l1.pop() # 50 return last value from list
    l1 # [10,20,30,40]
    ```

  - pop(index)

    ```
    l1=[10,20,50,60]
    l1.remove(2) # 50 return index value from list
    ```

```
l1 # [10,20,60]
l1.pop(10) # IndexError
```

- clear()

```
l1=[10,20,45,78]
l1.clear() # remove all value make a empty list
l1 # []
```

- sort()
  - sort is the method of list class.
  - also sort changes in the actual list.

```
l1=[45,78,12,54,63]
l1.sort()
l1 # [12,45,54,63,78]
l1.sort(reverse=True)
l1 # [78,63,54,45,12]
```

- reverse()

```
l1=[10,20,30,45,78]
l1.reverse()
l1 # [78,45,30,20,10]
```

- count()
  - It will give the frequency of a element in the list.

```
l1=[10,20,30,10,30,20,45,30,45,78]
l1.count(10) #2
l1.count(30) #3
l1.count(78) #1
l1.count(100) #0
```

- **List Comprehension**
  - l1=[exp for i in sequence]

```
l1=[i*i for i in range(1,11,1))
l1 # [1,4,9,16,25,36,49,64,81,100]
```

str

- str is a class.
- str is a sequence.
- str is iterable.
- str is immutable.
- str elements are indexed.

How to create a str object?

- ‘something’
- “something”
- “””something”””
- ‘’’something’’’

```
s1="chirag"
s2="""chirag"""
s3='chirag'
s4='''chirag'''
s5=str(123) # '123'
s6=str([10,20,30]) # '[10,20,30]'


len(s1) # 6
sum(s1) # ValueError
max(s1) # 'r'
```

```
min(s1) # 'a'
sorted(s1) # ['a','c','g','h','i','r']
```

str methods

- strObject.index()
- strObject.count()
- strObject.startswith()
- strObject.endswith()
- strObject.upper()
- strObject.lower()

```
s1="Muzaffarnagar"
s1.index('n') # 8
s1.index('f') # 4
s1.index('nag') #8

s1.count('f') # 2
s1.count('a') # 4

s1.startswith("Muz") # True
s1.startswith("muz") # False

s1.endswith("gar") # True
s1.endswith("Gar") # False

s1.upper() # 'MUZAFFARNAGAR'
s1.lower() # 'muzaffarnagar'
```

split() & join() method

- split function always return a list of strings.
- join function always join a list elements and return a string.

```
s1="i am good"

l1=s1.split(' ') # ['i','am','good'] split on the basis of character ' '.
l2=s1.split('a') # ['i ','m good'] split on the basis of character 'a'.

s2=" ".join(l1) # 'i am good' join on the basis of " " and join the list l1.

s3="HIII"
s4=s3.join(l2) # 'i HIIIm good' join on the basis of "HIII" and join the list l2.
```

Slicing Operator

- strObject[beg:end:step]
- beg include | end exclude | step difference

```
s1="Chirag Dhiman"
s1[1:10:1] # hirag Dhi
s1[5:12:3] # g Dhima
s1[10:2:-2] # mh a
s1[-1:10:-3] # n
s1[2:5] # ira
s1[::] # Chirag Dhiman
s1[::-1] # namihD garihC
```

- Note: Slicing Operator also use in list & working is same.

```
l1=[10,20,30,40,50,60]
l1[::-1] # [60.50,40,30,20,10]
```

tuple

- tuple is sequence
- tuple is class

- tuple is iterable
- tuple is immutable
- tuple may contain different type of value
- tuple elements are indexed.

how to create tuple object?

```
t1=(10,20,30)
type(t1) # <class 'tuple'>

t2=10,20,30,40
type(t2) # <class 'tuple'>
```

how to create empty tuple object?

```
t1=()
type(t1) # <class 'tuple'>
```

How to create tuple with single element?

```
t3=(10,)
type(t3) # <class 'tuple'>

t2=(10)
type(t2) # <class 'int'>
```

how to access tuple elements?

```
t1=(10,20,30)
t1[0] # 10
t1[1] # 20
t1[2] # 30
t1[3] # IndexError
```

```
t1[2]=111 # ValueError tuple is immutable
```

packing & unpacking

```
t1=(10,20,30)
a,b,c=t1
a # 10
b # 20
c # 30

t2=(a,b,c)
print(t2) # (10,20,30)
```

slicing operator

```
t1=(10,20,30,40,50}
t1[1:4:1] # (10,20,30)
```

+,*,> & more operator

```
t1=(10,20,30)
t2=(100,200)
t1+t2 # (10,20,30,100,200)

t1*3 # (10,20,30,10,20,30,10,20,30)

t1>t2 # False

l1=[1000,20000]
l1>t2 # Error different type value not compare
```

tuple() function

```
t1=tuple()
t1 # <class 'tuple'>

t2=tuple(10,20,30) #error more then one argument

t3=tuple(10) # Error int value pass

t4=tuple([10,20,30])
t4 # (10,20,30)
l1=[1,20]
tuple(l1)>t4 # False
```

index() & count()

```
t1=(10,20,30,10,20)
t1.count(10) # 2
t1.index(30) # 2
```

set

- set is class.
- set is a sequence.
- set is iterable.
- set elements are not indexed.
- Elements order in set is not preserved.
- set cannot have duplicate values.
- no slicing operator in set.
- set objects are mutable.
- set elements can be hetrogeneous.

How to create set object?

```
s1={10,20,30,40,50}
type(s1) # <class 'set'>
print(s1) # {50,40,10,20,30}
```

How to create an empty set?

```
s1=set()
type(s1) # <class 'set'>

s1={}
type(s1) # <class 'dict'>
```

set() function

```
s1=set(10) # Error int value can't pass
s1=set(10,20,30) # Error more then one argument

s1=set("chirag")
print(s1) # {'c','a','g','i','r','h'}
```

How to access set elements?

```
s1={10,20,30,40}
s1[0] # Error

for e in s1:
    print(e,end=" ")   # 40,30,10,20
```

set class method add()

```
s1={10,20,30,40}
s1.add(50)
```

```
s1 # {50,10,20,40,30}
s1.add("abc")
s1 # {50,40,20,"abc"10,30}

s1.add([112,456]) # error list can't be add in set

s1.add({10,0}) # error set also can't be add in set

s1.add((10,20))
s1 # {50,40,20,10,"abc",(10,20),30}

s1.add(range(0,5))
s1 # {50,40,20,10,"abc",(10,20),range(0,5),30}
```

set class method update()

- It takes any number of argument.
- Argument type must be sequence.
- It works like a union means it adds every element of sequence in set.

```
s2={10}
s2.update([10,20,30]}
s2 # {10,20,30}
s2.update([12,23],(1,90),"abc")
s2 # {10,23,1,'a',12,'b','c',90}
```

set class method remove(),discard(),pop()&clear()

- All method use for remove an element from set.
- discard never gives you an error.
- remove may be gives you an error.
- pop method remove any element from set and also return it.
- clear method remove all element from set & make an empty set.

```
s3={10,20,30}
s3.discard(30)
s3 # {20,10}
s3.discard(100) # No Error
s3.add(50)
s3.remove(20)
s3 # {50,10}
s3.remove(25) # Error
s3.pop() # 50 so return any element
s3.clear()
s3 # empty set.
```

union(),intersection(),issubset(),issuperset()

```
s1={10,20,30}
s2={20,30,40}
s1.union(s2) # {10,20,30,40}
s2.intersection(s1) # {20,30}
s1.issubset(s2) # False
s2.issuperset({1}) # True
```

dict

- dict is a class.
- dict is a sequence.
- dict is iterable.
- dict elements are not indexed.
- dict is short of dictionary.
- dict is mutable.
- no slicing operator.
- each element of a dict is a pair of KEY and VALUE.

How to create dict object?

```
d1={}
type(d1)  # <class 'dict'>
d1={1:"chirag",2:"piyush",3:"govil"}
```

How to access dict elements?

```
d1[1] # 'chirag'
d2[2] # 'govil'
d3[4] # Key Error
```

How to add new item & edit exist item in dict?

```
d1[101]="New Item"
d1 # {1:"chirag",2:"piyush",3:"govil",101:"New Item"}
d1[1]="update"
d1 # {1:"update",2:"piyush",3:"govil",101:"New Item"}
```

Note : For delete an item from any sequence

```
del d1[2]
d1 # {1:"update",3:"govil",101:"New Item"}
```

dict()

- takes always keyword arguments.

```
d1=dict()
d1 # empty dict
d2=dict(one=101,two=202,three=303)
d2 # {'one':101,'two':202,'three':303}
d3=dict(101:"chirag",202:"piyush") # Error
```

dict class method items() , keys(), values()

```
d1={101:"chirag",102:"piyush",103:"govil"}
d1.items() # dict_items([(101:"chirag"),(102:"piyush"),(103:"govil")])
d1.keys() # dict_keys([101,102,103])
d1.values() # dict_values(["chirag","piyush","govil")
```

clear(), pop() & popitem()

- clear use for delete all elements.
- pop delete only key passed elements.
- popitem delete without taking an argument.

```
d1={101:"chirag",102:"piyush",103:"govil"}
d1.pop(101) # (101,"chirag")
d1 # {102:"piyush",103:"govil"}
d1.popitem() # (103:"govil")
d1 # {102:"piyush"}
d1.clear()
d1 # empty dict
```

Note : we use comphrensive to store elements in dict & set

```
d2={k:k**2 for k in range(5)}
d2 # {0:0,1:1,2:4,3:9,4:16}

s2={k for k in range(5)}
s2 # {0,1,2,3,4}
```