

# Variables

Before getting into variables, let us see what is a program.

- **Program** is a set of instructions written in a programming language to perform tasks.
- A program has two important elements.
  1. **Data** : It is a information such as facts and numbers upon which operations are performed.
  2. **Instructions**: A set of steps to perform operations upon data.
- Every program needs data and instructions irrespective of the programming language used.
- Instructions without data is useless.

Let us see what are Variables?.

- Data in a program is handled by variables.
- **Variables** are the names given to the data stored in a memory location.
- Variable is the identifier, reference to the data.

Lets us understand a variable with an example.

age = 15



## Another example

```
name = 'soap'  
price = 15  
weight= 10
```

Variable                      Data

- **Python variable declaration:**

- Unlike other programming languages, in python variable is declared and initialised at the same time.

```
a = 10
```

“=” Assignment operator

Declaration                      Initialisation

- Declaring a variable is giving the name to the data.
- Initialisation is storing the value in a variable.
- In python declaring a variable without initialising or say with out the data is not allowed.
  - b= 12.5 ( declaration and initialisation)
  - c        (only declaration , is not allowed)
- Unlike other programming languages like c, c++, java, in python declaration of data type for a variable is not required.
- The value assigned to the variable determines the data type of the variable.

```
Python 3.9.7 (v3.9.7:1016ef3790, Aug 30 2021, 16:39:15)
[Clang 6.0 (clang-600.0.57)] on darwin
Type "help", "copyright", "credits" or "license()" for more information.
>>>
>>> a=10
>>> print(a)
10
>>> name='Soap'
>>> print(name)
Soap
>>> b
Traceback (most recent call last):
File "<pyshell#5>", line 1, in <module>
    b
NameError: name 'b' is not defined
>>> price=15
>>>
```

- **Declaration and initialisation of multiple variables:**

- In python multiple variables can be declared and initialised in a single statement.

Example:

```
a,b,c = 5,10,15
```

Here:

a=5,b=10,c=15 , values will be stored in the respective variables in the order they are declared.

```
name, price, weight='soap', 10, 15
```

Here:

```
name='soap', price=10, weight=15
```

```
>>> a,b,c=5,10,15
>>> print(c)
15
>>> name,price,weight='toothpaste',10.5,5
>>> print(price)
10.5
>>> |
```

Example : Assigning same value to multiple variables:

$x, y, z=1,1,1$   or  $x=y=z=1$

## Python- Dynamically Typed Language

- When a variable is declared , it must be initialise as well
- Variables do not have a specific data type , it's type depends on the value assigned to it.

Ex :

X = 25 - integer type

X = 13.75 - float type

X = 'A' - string type

- For the above example we can say that, Python is a dynamically typed language meaning while defining a variable we don't have to give its data type explicitly like in any other languages such as c++ , Java
- To know what type of data is provided to the variable we use function type(x)

Ex :

```
| type 'help', 'copyright', '|
>>> x=25
>>> type(x)
<class 'int'>
>>> |
```

- The above result is <class 'int'> meaning the given data type belongs to a class of integer
- As we know every thing in python is an object and every object will have its own class
- The class is decided on the type of value assigned to the variable.

## Rules for Declaring a variable

- Although we can use any name for declaring a variable but we must follow certain rules, so that it is easy to understand by us and others too
- The variable names can be taken as:

```
a= 10  
b = 12.5  
c= ' John '
```

- Even though the above declaration is correct but we cannot extract the exact meaning of the variables.
- To make the above variables more meaningful we can declare them as :

```
roll_no = 10  
price = 12.5  
cust_name= ' John '
```

- Now the above variables are more descriptive and understandable .
- The rules for declaring a variables is as follows

Name can contain alpha-numeric characters and underscore.

Name should start with a letter or underscore character.

Keywords should not be used.

Variables are case-sensitive.

- The **first rule** says that we can mix alphabet and numbers while declaring the variables, we can even use an underscore.

- However we cannot use any special symbol like \$, &, @, #, - etc...
- Example :

x1 = 10	✓
<del>cust_name= ' John</del>	✓
address1	✓
address#1	✗
address-1	✗

- The **second rule** says that the variable must start with a letter or underscore character only.
- Although we can use alphabet and numbers in a variable but numbers cannot be used at the beginning of the variable name.
- Example :

x2= 10	✓
_x = 10	✓
1x = 10	✗

- The **third rule** states we cannot use keywords to declare a variable.
- In python program we use words, numbers or symbols.
- here the words can be categorise into two, that is identifiers and keywords .

**Identifiers** : these are the words given by the programmer, It is used for identifying something that is define exclusively by the programmer .

price = 12.5

- Variable name, function name , class name and even module name are all identifiers.

**Keywords** : the words which are predefined in the language are called keywords or reserved words.

while = 10	✗
pass = 20	✗

- The list of the keywords are given below .
- These word cannot be used while declaring the variable.

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

- The last rule states that variables are case sensitive,

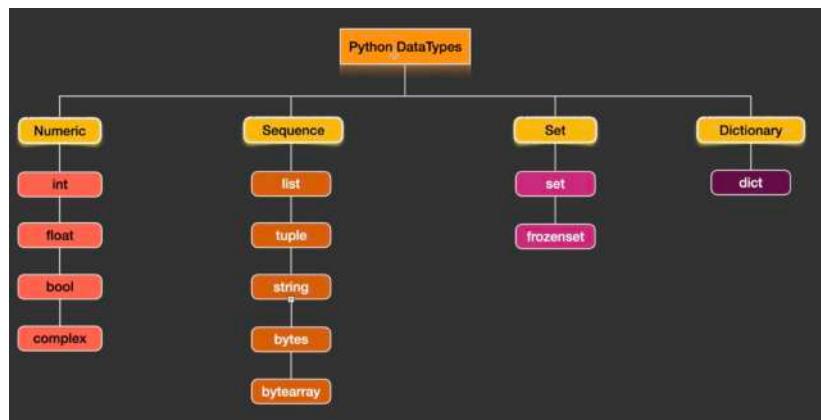
a = 10

A = 10

- The above variables are not same , ‘A’ is not same as ‘a’.

# Python Data Types

- Python is a very rich language in terms of data types as it provides various data types as shown below



- The example of **numeric** data types are

Int :

```
x = 10
```

Float :

```
y = 19.5
```

bool :

```
z = TRUE
```

Complex :

```
a = c + ib
```

- The **sequence** data type holds the collection of values.

- The **set** type of data is also collection of values , however the difference between set and sequence is , In Sequence data type , all data item have its own index where as in set there is no index value for data elements .

## Example of Sequence data type is :

List :It is a collection of items/values in a single variable. Its mutable.The values of a list can be modified.

```
[ 2,3,4,75,7 ]
```

Tuple :It is a collection of values. Tuples are immutable, they cannot be modified.

(4,6,8,2,4,6)

String: It is a collection of alphabets, numbers ,characters or symbols.

“python” , ‘ hello’

Byte :

it contain only 1 byte of data

it allows value from 0 -255

Bytearray :

it contain array of byte data

### **Example of set data type is :**

Set :

{ 2,4,8,0,12 }

Frozen set :

it is also like set but frozen mean the data cannot be modified

- Dictionary: The **Dictionary** data type is the collection of key-value pair.
- It is useful for storing and retrieving the data much faster

### **Example :**

name : “John”

rollno : 23

dept : ‘cs’

## Numeric Data Type

- The numeric data type that are available in python are:

```
int  
float  
bool  
complex
```

- Integer ( int )** is the numeric data without any decimal point .
- It can contain, both positive and negative numbers.

- Example :**

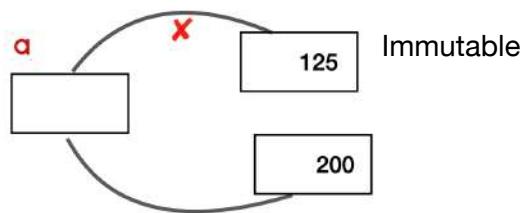
```
a = 125  
b = 2164  
c = -17  
x = 129734864
```

- In integers there is no size limit or range to an integer value .  

```
x = 12345678901234567890123456789 .
```
- Not only in integers but there is no fixed size memory taken by any datatype in python.
- Suppose you want to know the size of any value then we use **'sizeof'** method, see the example below :
- x.\_\_sizeof\_\_()**

```
>>> x=12345678901234567890123456789  
>>> print(x)  
12345678901234567890123456789  
>>> y=-17  
>>> y  
-17  
>>> print(x.__sizeof__())  
40  
>>>
```

- Lets consider an example and see what happen when we assign a variable two different values.
- `a = 125` and `a = 200`.
- Here the first `a` is the reference to the value 125 and it is available in the memory.
- when we change the value of `a` from 125 to 200, then `a` will point/refer to the new object 200.
- Here the actual value that is 125 is not changing but a new value is created and now `a` points to this new value, therefore we can say that values are immutable or it cannot be changed.



- Now the question is where is 125 in memory?. The value 125 will be garbage collected by PVM ( python virtual machine ) ,and it will be deleted from the memory.

### Floating point :

- Any numeric value with a decimal point is called a floating point number.
- They can either be positive or negative numbers.

#### Example

`a = 13.25`

`b = -17`

- Floating point number can also be in written scientific format as well ,

`12.59`

`0 . 1259 * 100`

`0 . 1259 * 102`

`0 . 1259E2 # 102 - E2 ( floating point representation )`

- Float datatype is immutable.

## Numeric Data Type ( bool & complex )

- Boolean and complex are numeric data type.
- **Boolean** data is logical data.
- Boolean data types are used in writing conditions using relational and logical operators.
- The result of a Boolean is either True or False.

True — 1                      The value of True is 1  
False — 0                      and False is 0.

- Example :

```
>>> a=True
>>> a
True
>>> int(a)
1
>>> type(a)
<class 'bool'>
>>>
```

## Complex Numbers

- Complex numbers have real part and an imaginary part.
- Complex numbers are mostly used in mathematical operations.

Mathematical representation:

$$a + ib$$

real                          imaginary



Python representation:

$$a+bj$$

real                          imaginary



- Here **i** and **j** are the pre-defined constants.
- where  $i = \sqrt{-1}$ , or  $j = \sqrt{-1}$
- In mathematics we know that the square root of negative numbers is undefined,
- lets take an example to understand this:

```

25+ √-9
25 + √-1 * 9
25 + √-1 * √ 9
25 + √-1 3
25 + i3

```

- Complex Data types can be used when an application is being developed in Python involving complex numbers.
- We can create complex numbers using integer, float value and even using functions.

```

X = 3 + 5j           // Integer
x = 3.5 + 5.9j      // float
X = complex( 3.5, 5.9) // function

```

- Operators like  $+$  ,  $-$  ,  $*$  ,  $/$  can be used to perform operations on complex numbers .

## Literals or Constants

- Literals are the direct values / data written in the program

```
price = 250
      ↑   ↑
    data literals
```

- They are 2 methods a variable can have value

by directly writing the values in the program  
by taking input from the user

- We can store any type of data in the literals
- Lets understand this with an example

### Example For taking **integer literals** :

```
a = 125 // basic literal
```

a = 12520 // this is a big number and to store it as literal we need to give an underscore to it as shown below

```
a = 12_520
```

- In python we use \_ ( under course ) so we can easily understand number system

### Example For taking **Float literals** :

```
a= 12.59 // basic float literal
```

b=13 // in float we need to have decimal value so, we take this value as b = 13. 0

- Float number can also be written in scientific notation

```
c = 1.32E2
= 1.32* 102
= 1.32*100
= 132
```

- There are some specification while giving underscore in float value that are the \_ cannot be given before and after the decimal point

d = 123_779 . 45	✓
e= 123.4_5	
f = 123 _ .45	✗ # syntax error

- When you are directly writing True / False values in your program we can use **Bool Literals**

```
a = True
b = False
```

- The T and F of True and False must be capital otherwise it will give syntax error

### **complex literal :**

- The complex number literals are as follows , it can have an \_ as well

a = 5+4j

a = 5\_1 + 4\_3j

### **string :**

- The string literals are as follows

a = ‘ John ‘

b = “ John “

c = ” John ”

## literals

Decimal : 0 - 9 its also said as base 10

Binary : its of 2 digits is said as base 2

Octal : 0 - 7 is also said as base 8

Hexa : 16 digits 0 - 9 . A - F base 16

**int** : Both positive and negative numbers including 0

a = 10  
b = 0b1010

Both a and b are same

a = 0o12 # octal  
a = 0xA # hexa

Both are 10

**float** : number with both integer and fractional parts.

a = 0b125  
f = 0b111 . 0b11      ✗

**complex** :

c = 5 + 6 j  
c = 0b101+6j      ✓

We can give a real part but not imaginary part

c = b101+b101 j      ✗

**str** :

price = input(" enter price ")

Output : enter price 0b101

' 0b101,

• But if we want in int type so we use type casting

price = int ( input ( " enter price " ) , 2 )

# base 2 cause we are entering the binary form .

enter price : 0b101

Output : 5

## Base conversions

a = 10

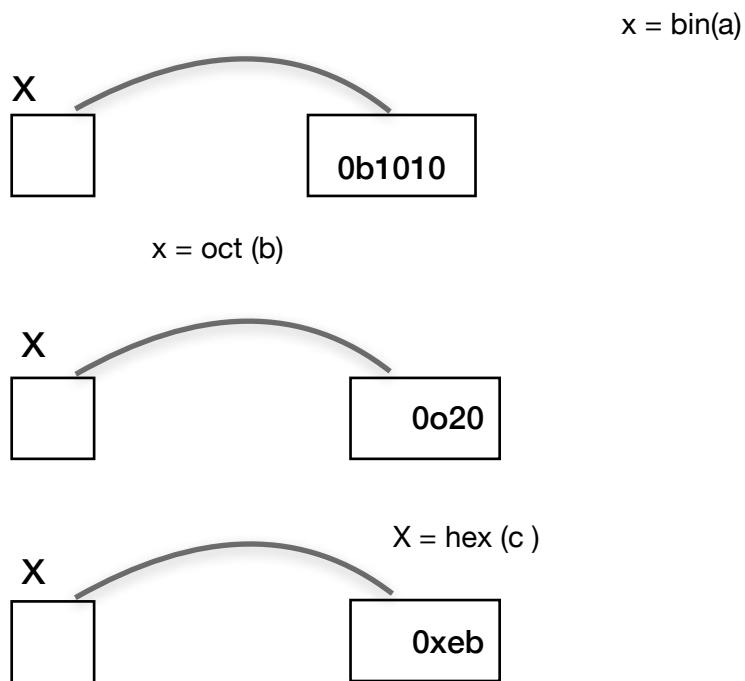
bin()

b = 16

oct()

c = 235

hex()



- Suppose we want to know the binary equivalent of any value we can call the function

$y = \text{bin}(b)$   
output : '0b10000'

## Type conversion

- It will convert one datatype to another datatype

- 1) Implicit
- 2) Explicit

**implicit** : In Implicit type conversion of data types , the Python interpreter automatically converts one data type to another without any user involvement.

**Explicit** : The programmer has to convert and programmer have to mention

```
int ()  
float()  
complex()  
bool ()  
str()
```

### Int

```
x = int (f)
```

- It will convert float to int type

f = 16.59



X

16

The output of x will be 16

- Lets try with a string

```
s1 = 'John'  
x = int ( s1 ) X # error
```

Lets give a try with int given in string

```
s2 = '123' ✓  
x = int ( s2 )
```

Its possible and the output will be 123

Lets take in binary form

```
s3 = '0b1111' ✓  
x = int ( s3 , 2 )
```



String                      base value

Output : 15

**float** : in float we can convert int to float and bool to float

x = float( i )    ✓

x = float ( b )    ✓

x = float ( c )    ✓

Lets try with string

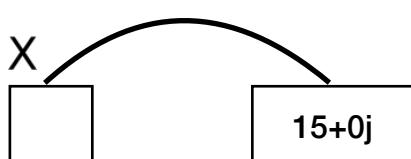
s1 = ' John'  
x = float ( s1 )    ✗

s2 = '125'  
x = float ( s2 )    ✓

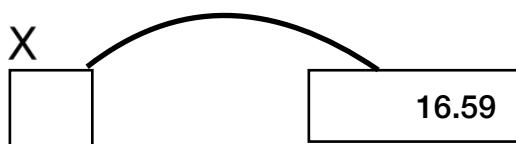
The output : 125.0

**complex** : it have real part and imaginary part

i = 15  
x = complex ( i )



x = complex( f )



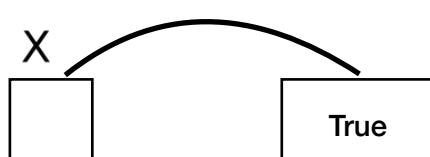
- Same goes for str

s1 = ' John'  
x = complex ( s1 )    ✗

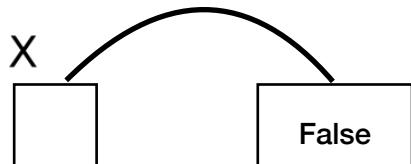
s2 = '125'  
x = complex ( s2 )    ✓

**bool** : it will convert all datatype to bool type

x = bool ( i )



`x = bool ( 0 + 0j )`



- Same goes for string

`x = bool ( ' hi ')`

Output : true

## Operators ( Arithmetic operators )

- These operator is use to perform arithmetic operation like addition , sub , div etc
- There are 7 athematic operators

$$a = 15 \quad b = 4$$

c = a+b output : 19

c = a\*b output : 60

c = a-b. Output : 11

c=a\*\*b

• Now the question is what is \*\*

It is said as power

c= a\*\*b

$$(15)^4$$

$$= 50,625$$

### Division ( / ) :

$$C = a/b = 3.5$$

dividend

↓

devisor    ↗

$$\begin{array}{r} 4 ) 15 ( 3.75 \\ \underline{14} \\ 03 \\ 28 \\ \hline 20 \\ 20 \\ \hline \end{array}$$

X ↗      reminder

### Float division ( // ) :

- If we don't want in float form we want in int form then we will use float division

$$C = a // b$$

$$= 3$$

### Mod ( % ) :

- This will divide . It will not take quotient it will take remainder

$$a = 15 \quad b = 4$$

$$Z = x//y = 3$$

$$Z = x \% y = 0$$

## Expressions

- The instruction that we write using operators is called expressions

$c = a + b$   
# c , a , b are operands  
# = , + are operators

Lets take example :

$d = 2+3*5$  # $3*5=15$   
 $d = 2+15$   
 $d=17$

- This happened because of precedence .
- First multiplication takes place because it has higher precedence than addition
- They will execute based on their precedence

$d = 2*3 + 8/2$   
 $= 6 + 4 = 10$

- To increase the precedence we use ()

$2+3*5$  # first multiplication take place because it have higher precedence than addition  
 $2+15 = 17$

By using parentheses

$(2+3) * 5$  # here addition will take place because parentheses have higher precedence than all

$5 * 5 = 25$

- What if we are having

$d=3*4*5/4$  # \*, / have same precedence  
→

So it will do left to right #first multiplication than division

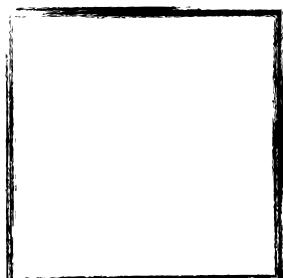
$d=2**3**2$



It will execute from right to left

**Python expressions :**  
Square .

Area of a square =  $lb$



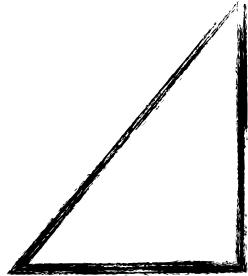
- In python we write using expressions

The area of the square can be written

$$l * b$$

**Triangle** .

Area of a triangle =  $1/2 bh$



In python =  $1/2*b*h$

**Trapezium**

Area of a trapezium =  $a+b/2 h$



- In python we write using python expressions area=  $(a+b) / 2*h$

**Displacement**

$$(v^2-u^2/2a)$$

In python

$$(v^{**2}-u^{**2})/(2*a)$$

**Equal roots**

$$(-b/2a)$$

In python using expressions

$$(-b/(2*a))$$

## Program Using Expressions.

- The input function gives string type of result.
- So, when other type of values are taken we should typecast it [i.e, convert str to int etc]

Q) Write a program for finding area of a rectangle.

#The formula for finding area of the rectangle is **a = l \* b** where l is length and b is breath.

```
length = int(input("Enter length of Rectangle : "))
breath = int(input("Enter breath of Rectangle : ")) # taking length and breath as input and typecasting it.
area = length * breath # formula for area of rectangle
print("area is ", area)
```

## Expression Student Challenge #1

1 . Triangle area

# area =  $1/2bh$

```
base = int (input ('enter base'))
height = int (input('enter height'))
area = 1/2 *base*height  # 1/2bh
print(area)
```

2 . Rhombus area

```
a=float(input('enter side a'))
b=float(input('enter side b'))
h=float(input('enter side h'))  # 3 inputs are typecasted by float
area= 1/2*(a+b)*h  #1/2(a+b)h
print(area)
```

3 . Displacement

```
v = float(input('enter v value'))  # v - final
u = float(input('enter u value'))  # u - initial velocity
a = float(input('enter a value'))  # a - acceleration
d=(v**2-u**2)/(2*a)  # d - displacement
print(d)
```

## Expression Student Challenge #2

1 . Converting km into miles.  
# 1km = 0.621371

```
km = float(input ("enter km"))
miles = km * 0.621371
print('miles =' , miles)
```

2 . Area of a circle  
#  $\pi r^{**2}$ (square)

```
import math # its a math module
raduis = float(input('enter radius'))
area = math.pi*raduis**2 # in math module we are using pi
print("Area =" , area)
```

## Expression Student Challenge #3

1. Total surface Area of a cuboid

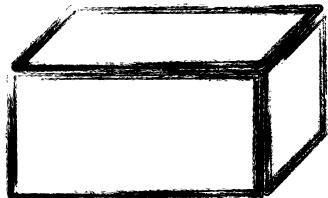
# it has 6 sides

Front and back =  $2l \times h$

Bottom and top =  $2lh$

Left and right =  $2bh$

Total =  $2(lh+lb+bh)$



```
length = float(input('enter length'))
breadth = float(input('enter breadth'))
height = float(input('enter height'))
area = 2*(length*breadth*length*height+breadth*height)
print ('Total surface area is ', area)
```



## Expression Student Challenge #4

### 1. Finding roots of quadratic equations

eg :  $ax^2 + bx + c = 0$

Roots :

$$r1 = -b + \sqrt{b^2 - 4ac} / 2a$$

$$r2 = -b - \sqrt{b^2 - 4ac} / 2a$$

If  $\sqrt{b^2 - 4ac}$  will give negative value than it will become complex number

```
import math

a = int(input('enter a value'))
b = int(input('enter b value'))
c = int(input ('enter c value'))
root1 = (-b + math.sqrt(b**2 - 4 * a * c))/(2*a) #in math module we are taking square root (sqrt)
root2 = (-b - math.sqrt(b**2 - 4 * a * c))/(2*a) #sqrt cant find the value for negative
print ('roots are ' , root1,root2)
```

## Assignment arithmetic operator

Arithmetic operator joined with assignment operator.

- suppose we have a variable

$$a=5$$

$$a=a+1$$

$$a=5+1$$

stores 6 in a

- whatever the value is stored results into it.
- It means we want to modify the value of a and increase it by 1
- The a is assigned two times the value of a is taken and increased by 1

So, this type of statements usually used for counting

$$\text{Count}=0$$

$$\text{Count}=\text{count}+1 \quad \text{this means}$$

$$1=0+1$$

- Count = count+1. This type of statements can be written in short like count+=1
- a=a+1 instead of writing this elaborated statement we can write it in short like a+=1.
- This statement was for addition
- Now, in same way if we have anything to subtract we can use subtraction.

N=10

N=n-1  
n=1

this means the same thing n- assigns 1

- if we have a variable p=10 and we want to multiply it with a variable x=5

then we may be assigning  $p=p*x$

$p^*=x$

to make the statements short this operations are given

- bitwise operators can also be used with these assignment operators.
- If we have two variables a=10, b=14 and want to perform & operation

a=a & b

a& =b

if we want to do something in other way

b=a&b this is the statement we can change to b&=a.

- when want to store the result in a then
- say a=&b this type of statement can be converted to a&=b
- if we want b to store result in b then say b=a & b this type of statement can be converted to b&=a assignment as well as bitwise operator can be joined together to make our statement shorter

most of the time counting statement is used

```
>>> a=5 # assigning value 5 to variable
```

```
>>> a=a+1# incrementing by 1
```

```
>>> a
```

6# a becomes 6 after incrementing

```
>>>count=0
```

```
>>>count+1#incremneted by1
```

1        # becomes 1

```
>>>count+=1
```

```
>>>count
```

1

```
>>>count+=1
```

```
>>>count
```

2

Instead of writing this complex statement we can write it in short

Like this

```
>>>count+=1
```

```
>>> count
```

3

## Arithmetic with all type

	+	-	*	/	//	%	**
int	✓	✓	✓	✓	— ✓	✓	✓
float	✓	✓	✓	✓	✓	✓	✓
bool	✓	✓	✓	✓	✓	✓	✓
complex	✓	✓	✓	✓	✗	✗	✓
str	✓	✗	✓	✗	✗	✗	✗

- This table shows how operators are compatible with other Datatypes.
- If you are performing float division you'll get float result.
- If you are performing floor division you'll get integer result.

True = 1 , False = 0

### Boolean

- Bool v/s Bool the result will always be integer.
- Bool v/s complex the result will always be complex.

### Complex

- Complex with integer the result is always complex.
- Complex works with all except // and % as // is used to convert float type to int type but can a complex number do all this , no so it doesn't work same goes for %.

### String

- Adding 2 strings is called 'concatenation '. If one type is integer and the other is string concatenation doesn't work. The 2 types to be added should be string only.
- In multiplication i.e, \* one type should be string and the other should be an integer only.
- Float doesn't work for strings.

## Conditional statements

- Control statements are the statements that control the flow of execution of statements so that they can be executed repeatedly and randomly
- Whenever we write a program it will execute linearly but in set of statement we add conditions it will break the linear execution and will execute the conditional statement ( if else ) then go further

### if

- First the condition is tested If condition is true it will execute . if block is false than it will execute else block .
- We can write one or more statement after colon ( : )
- if is false , then the statement mentioned after : are not executed
- Lets see the syntax :

```
if condition :  
    statements
```

- We can see in syntax there is indentation . Indentation is important in python . It is used in the beginning of the statement

### if else

- The if ....else executes statement evaluates test expression and will execute the block of if only when the test condition is True.
- If the condition is False, the body of else is executed. Indentation is used to separate the blocks.
- Lets see syntax for if else statement

```
if condition :  
    statement  
else :  
    statement
```

### Relational and logical operator :

- For writing the conditions we use relational and logical operators and the output will be always be “ True ” or “ False ”
- This is also called as comparison operators
- Lets take an example for better understanding

a = 10      b = 20

```
if a < b : T  
a <= b : T  
a > b : F  
a >= b : F  
a == b : F  
a != b : T
```

- Lets take an example by using if ... else

```
a = int ( input ( " enter a number " ) )  
if a < 0 :  
    print ( " Negative " )  
else :  
    print ( " Positive " )
```

Output : Negative



# Logical

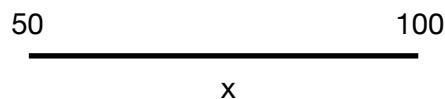
- Logical operators use to write compound conditions

## AND

- Lets take example - cond1 and cond2
- If the cond1 and cond2 both are **true** than it will return true value

True - 1              False - 0

- . AND is like multiplication and it is useful to check the range of values



If  $x \geq 50$  and  $x \leq 100$

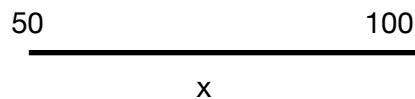
- Lets take an example about subjects (maths , phy , and chem ) where above 45 will be the passing marks

```
if math>=45 and phy >=45 and chem >=45 :  
    print ("pass")  
else :  
    print (" fail ")
```

- AND is use to check this type of conditions where everything must be true ( both conditions should be true )

## OR

- If the cond1 or cond2 both are false returns **false**
- OR is like addition



- OR is useful to check outside the range

```
if x <= 50 or x >=100 :
```

## NOT

- It will negate it
- Returns the opposite value ( If its true returns false if its false return true )
- It is not commonly used

## Condition student challenge #1

1. Find difference between 2 numbers .

# if 5 - 15 = -10 we don't want in negative one we want to know only differences

Sol :

```
no1 = int(input ('enter a number'))
no2 = int(input('enter a number'))
if no1-no2>=0:
    print(no1-no2) # no '....' cause we want it in format way
else:
    print(no2-no1)
```

2. Check if a number is odd or even .

# when we divide it by 2 the remainder must be zero then it is said as even

Sol:

```
n = int(input('Enter a number'))
if n%2==0:
    print('even')
else:
    print('odd')
```

3. Check for Age eligibility for casting a vote .

#only 18+ can vote

Sol :

```
age = int(input('enter your age'))
if age >= 18:
    print('eligible')
else:
    print('Not eligible')
```

## Student challenges # 2

1 . Check if marks of a subject are within range 0 - 100 .

if - valid  
else - invalid

```
marks = int(input('enter marks')) # taking the user input and typecasting it to int value
if x >= 0 and x <=100 : # 'and' operator where both conditions should be true
    print ('valid')
else : # if is false so else will execute
    print ('Invalid')
```

2 . Check if a person is ‘ Male ‘ or ‘ Female’

# to check person is Male or female by taking input from the user

```
gender = input ('enter gender') # we dont need to typecaste it cause input take string value
                                #by default
if gender=='m' or gender=='M': # 'or' anyone should be true
    print('Male')
else:
    print('Female')
```

3 . Check if the person is eligible to work .

```
age = int(input ('enter age')) #user input
if age >= 18 and age <= 60 : # the age should be between 18 – 60
    print('eligible')
else : # 'if' is false than else is executed
    print('not eligible')
```

## Nested if and elif

- `elif` is also a keyword in python like `if`.
- An `if` inside another `if` or another `else` statement is called nested if.
- Instead of using nested `if` python provides `elif` because using `if` will make the code more deeper and take much lines of code when compare to `elif`.

Q) Write a program taking input from user and check which one is elder among them ?

# Taking input from users and typecasting it into float value , setting if and else condition for comparison , then again applying NESTED if and else for further comparisons and printing the results based on the input given by the user.

- Nested if else without elif.

```
main.py
1 john = float(input("enter john's age : "))
2 smith = float(input("enter smith's age: "))
3 ajay = float(input("enter ajay's age: ")) # Taking input from users and type casting it into float value.
4
5 if john > smith and john > ajay: # setting if statement to compare ages
6     print("johns is elder")
7
8 else:                      # setting else for when if is not true
9     if smith > ajay:        # Nested if for more comparing statement
10        print("smith is elder")
11    else:
12        print("ajay is elder")
13
```

- Nested if else with elif.

# replacing else and if with elif hence making the code more readable and understandable.

```
main.py
1 john = float(input("enter john's age : "))
2 smith = float(input("enter smith's age: "))
3 ajay = float(input("enter ajay's age: ")) # Taking input from users and type casting it into float value.
4
5 if john > smith and john > ajay: # setting if statement to compare ages
6     print("johns is elder")
7
8 elif smith > ajay:             # replacing else and if with elif
9     print("smith is elder")
10 else:
11     print("ajay is elder")
12
```



## elif student challenge #1

1. Calculate discounted amount

```
amount <= 1000 -- 10%
1000 < amount <=5000 --20%
5000 <amount <=10000 --30%
10000 < amount --50%
```

```
amount = float(input('enter bill amount'))
if amount <= 1000 :
    discount = amount * 10 / 1000      # 10%
elif amount > 1000 and amount <=5000:
    discount = amount*20/100          #20 %
elif amount > 5000 and amount <= 5000:
    discount = amount * 30/100        #30%
else:
    discount = amount * 50/100        #50%
discamount = amount - discount
print('pay', discamount)
```

## elif student challenge #3

1. Take a day number and display day name  
# take any number and give it a day name . Its like switch case in other languages

```
day = int (input('enter day number'))
if day == 1 :
    print('Sunday')
elif day == 2:
    print('Monday')
elif day == 3:
    print('Tuesday')
elif day == 4:
    print('wednesday')
elif day == 5:
    print('Thursday')
elif day == 6:
    print('Friday')
elif day == 7:
    print('Saturday')
else:
    print('Invalid day')
```

## elif Student Challenge #2

Q ) check whether year is the leap year or not

#a year have 365 days and 1/4 day , 4 quarter day makes one day and that will be added to the feb month . Usually feb will have 28 days but in leap year( `year % 400 == 0`) it will have 29 days . Every fourth year will be the leap year . If the year is century( `year %100 == 0` ) then it is not leap year .

```
year = int ( input (" enter year"))    #user input is type casted to int

if year % 100 ==0:
    if year % 400 == 0:
        print(" leap year")   #if the remainder is 0 it said as leap year
    else :
        print("not a leap year")
elif year % 4 == 0:
    print("leap year")
else :
    print(" not a leap year")
```

## Detail Relational Operator

	<	<=	>	>=	==	!=
int	✓	✓	✓	✓	✓	✓
float	✓	✓	✓	✓	✓	✓
bool	✓	✓	✓	✓	✓	✓
complex	✗	✗	✗	✗	✓	✓
str	✓	✓	✓	✓	✓	✓

### Complex

- Relational operator can perform using int float bool and string but in complex it will not let see why.

$$a = 5+6j \quad b = 9+8j$$

`a < b` . #error

- Because it will have 2 elements that are real and imaginary . Which one should be compared a real part or imaginary part . It can't be compared . So , it is not possible .

### String

- String are compared based on there dictionary order

$$\begin{aligned} s1 &= " America" & s2 &= " Brazil" \\ s1 < s2 &\longrightarrow \text{True} \end{aligned}$$

What if a string have same alphabet

$$\begin{aligned} s1 &= "Brazil" & s2 &= " Bracket" \\ s1 < s2 &\longrightarrow \text{True} \end{aligned}$$

Because in dictionary Bracket will come first then Brazil so s1 Is less than s2

What if we have

$$s1 = " brazil" \quad s2 = "Brazil"$$

Both are not same cause python is case sensitive

$$s1 < s2 \longrightarrow \text{False}$$

Because uppercase letter is lower then the lowercase . So , s1 is not less than s2

## Logical operator

- and , or , not are logical operators
- Logical operators works upon compound conditional statement by using and , or , not

and

A	B	A and B
True	True	True
True	False	False
False	True	False
False	False	False

- They will work upon non bool type ????? Lets see

a = 5              b = 10              c = 15

if a<b and a<c  
5<10 and 5<15

- And will return true if all are true . The conditions given in if are true

0 - False        other than 0 is - True

if 5 and 6 returns True

T      T

if -5 and 7 returns True    # anything other than is True

If 0 and 6 returns False    # 0 - False

- They will work on non bool type . It will not give the result T but it will give the second number  
(Anything other than 0 is True )

```
[Clang 6.0 (clang-600.0.57) on darwin
Type "help", "copyright", "credits" or "license()" for more information.
>>>
>>> 5 and 6
6
>>> -5 and 6
6
>>> 5 or 6
5
>>> 1.2 and 1.3
1.3
>>> 0.0 and 4.0
0.0
>>> 0 and 9
0
>>> 5+2j and 3+1j
(3+1j)
>>> 5+2j and 0+0j
0j
>>> 0+0j and 5+2j
0j
>>> 'hi' or 'bye'
'hi'
>>> 'hi' and 'bye'
'bye'
>>> '' and 'bye'
''
>>>
```

- Empty string is treated as false

**Short circuit :** In `and` when the first one is false we don't have to check the second value cause it will be false cause and will return true if both the conditions are true otherwise false . If the first is True then it will check the second statement

**OR** Truth table

A	B	A or B
True	True	True
True	False	True
False	True	True
False	False	False

In OR it will check the first statement If it true than it will not check the second statement cause it will return true.

```
>>> 5 and 6
6
>>> 0 and 6
0
>>> 5 and 0
0
>>> 5 or 6
5
>>> 0 or 6
6
>>> 0 or 0
0
>>> 5 or 9
5
```



## Bitwise Operator

- The Bitwise operator are AND, OR, XOR, Complement, Left shift , Right Shift.
- These operators work on integral type of data and they perform operation on Binary representation of data i.e, 0's and 1's.
- Bitwise operators are used in applications of networking , Encryption and Decryption etc.
- Bitwise calculations starts from right hand side.

Operator	
&	AND
	OR
^	XOR
~	Complement
<<	Left Shift
>>	Right Shift

## Understanding Binary numbers system :

- To understand bitwise we first need to understand binary number system . Binary number system uses only 0's and 1's
- Decimal number system uses numbers from 0 - 9
- To convert decimal to binary we can do it in either 2 ways

0 → 0 # 0 is 0

1 → 1 # 1 is 1

+ 1 # adding 1 to 1 to make it 2  
\_\_\_\_\_

2 → 10 # binary form of decimal no 2

+ 1 # adding 1 to binary of 2 to make it 3  
\_\_\_\_\_

3 → 11 # binary form of decimal no 3 so on...

+ 1  
\_\_\_\_\_

4 → 100

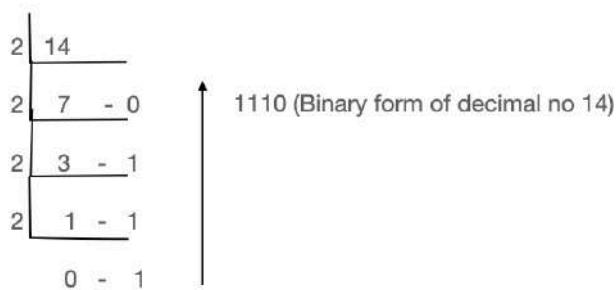
+ 1  
\_\_\_\_\_

5 → 101

Decimal	Binary
0	0
1	1
2	10
3	11
4	100
5	101
6	110
7	111
8	1000
9	1001
10	1010
11	1011
12	1100
13	1101
14	1110
15	1111

Table for decimal to binary

Consider decimal number as - 14



Let us check this in IDLE

```
Python 3.9.7 (v3.9.7:1016ef3790, Aug 30 2021, 16:39:15)
[Clang 6.0 (clang-600.0.57)] on darwin
Type "help", "copyright", "credits" or "license()" for more information.
>>>
>>> a=10
>>> format(a, 'b')
'1010'
>>> a=14
>>> format(a, 'b')
'1110'
>>> format(25,'b')
'11001'
>>> a=25
>>> bin(a)
'0b11001'
>>> a.bit_length()
5
>>>
```

## Bitwise Operations

- Let us understand all the bitwise operators with an example
- Consider  $a = 10$  ( binary of 10 is 1010 )

$$b = 13 \quad (\text{binary of } 13 \text{ is } 1101)$$

### AND operations (&)

- AND works on multiplication
- Working of AND -

- $1 * 1 = 1$
- $1 * 0 = 0$
- $0 * 1 = 0$
- $0 * 0 = 0$

Ex : a - 1010

b - 1101

$$\begin{array}{r} \text{a & b} \\ \hline 1000 \\ \hline \end{array} = 8 \text{ ( Decimal form of binary number )}$$

### OR operations (|)

- OR works on addition
- Working of OR -  $1 + 1 = 1$   
 $1 + 0 = 1$   
 $0 + 1 = 1$   
 $0 + 0 = 1$

Ex : a - 1010

b - 1101

$$\begin{array}{r} \text{a | b} \\ \hline 1111 \\ \hline \end{array} = 15 \text{ ( decimal form of binary number)}$$

### XOR operations (^)

- Working of XOR -  $1 \wedge 1 = 0$   
 $1 \wedge 0 = 1$   
 $1 \wedge 1 = 1$   
 $1 \wedge 0 = 0$

Ex: a - 1010

b - 1101

$$\begin{array}{r} \hline \end{array}$$

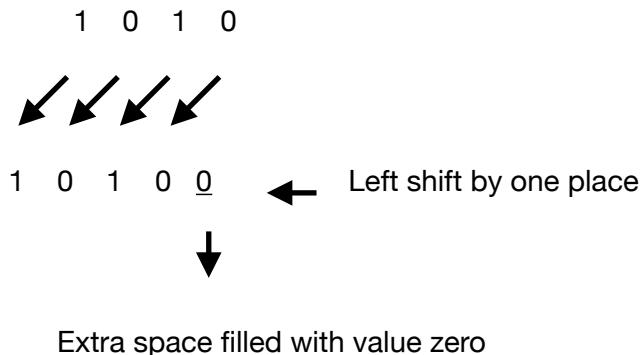
$$a \wedge b - \quad 0111 = 7 \text{ (decimal form of binary number)}$$


---

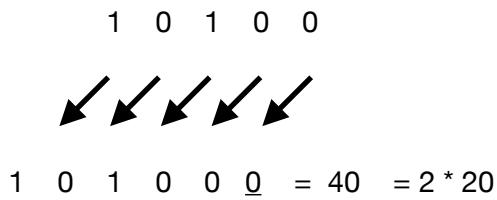
- Leading zero doesn't make sense
- Hence , we consider 111 whose decimal form is 7

### Left shift ( $<<$ ) and right shift ( $>>$ )

- $a = 10$  (binary of 10 = 1010)
- If we left shift 'a' by one place i.e,  $a << 1$  then



- After left shift by one place the bit that is freed will be taken as zero.
- Now  $10100 = 20 = 2^* 10$
- If we left shift 20 again then,



- We see that when we left shift the number will double for one place i.e;  $a << n$  ,  $a * 2$  (power n)
- If we double the left shift i.e ,  $a << 2$  then

$$a << 2 = 2(\text{power } n) * a = 4 * 10 = 40$$

- similarly , if we do  $a << 5$  , then the value gets double for 5 times

$$a << 5 = 2(\text{power } 5) * a = 32 * 10 = 320$$

- If we RIGHT SHIFT the number will become half.

$a = 10 \rightarrow$       1 0 1 0



— 1 0 1 0 (this 0 gets discarded)



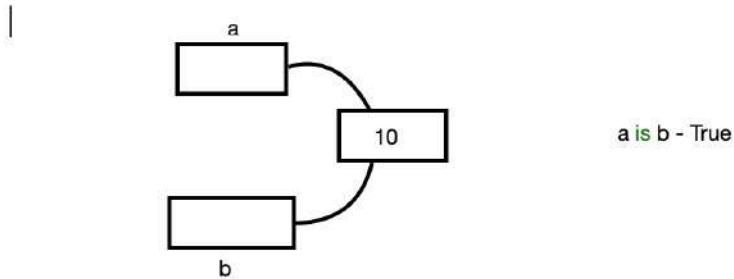
This leading empty space has no value (it becomes zero)

- Right shift operator divide by 2 i.e;  $a >> n$   $a / 2^n$  (power n)

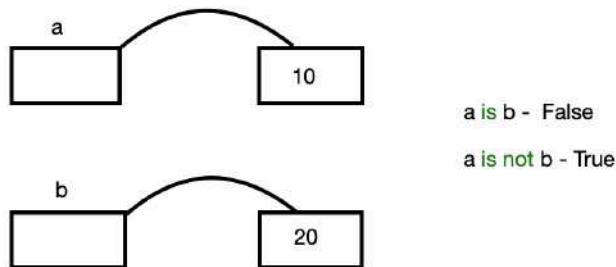
## is - is not operator.

- is and is not will check whether two variables are referring to the same memory or not.
- They return boolean value ( T, F ) as result.
- They are checked based on variable and not on value in the memory.
- For each input values ( user define ) python creates new memory even when the given values are same.
- When we are assigning values in the program python will refer it to the same memory as it is dynamically typed Programming language .

Ex 1 : a = 10  
b = 10



Ex 2 : a = 10  
b = 20



# Introduction to Loop's

- Loops are also called as repeating statements.

The loops are of 2 types in python

- I. **while loop**
- II. **for loop**

- If you want a set of statements to repeat again and again in the program then we use loops  
The statements can repeat either '**for number of times**' or '**As long as the condition is true**'
- In **while condition**, If the condition is true, the while block will get executed after the execution it will go back again and check the condition , if again the condition is true it'll get executed , once the condition becomes false it'll stop the execution.

Syntax : **while condition :**  
                  Statements

- Whatever condition you are using In while you should modify it in the statements else your loop will not work properly .

**Example 1 :** ( statements repeating number of times )

Q) Print hello 10 times using while loop.

#in while loop use the counter as condition and set its value from 1 -10 , modify the counter In the statements and get the result.

```
count = 0
while count < 10: # condition saying count should be less than or equal to 10
    print(count+1,'hello') # when count is increased print hello , starting count value should start from 1
    count = count + 1 # modifying the count
```

Output :

```
1 hello
2 hello
3 hello
4 hello
5 hello
6 hello
7 hello
8 hello
9 hello
10 hello
```

### Example 2 : ( As long as the condition is true )

Q) Write a program to print output of the given number from last to first.

# Take input from users in integer form , the condition should be a number greater than zero and divide the number and print its result in integer form only.

```
n = int(input('enter a number : ')) # typecasting input into integer value
while n > 0:
    r = n % 10 # dividing n by 10
    n = n // 10 # if given n value gives float as result upon dividing by 10 then convert it into integer using //|
    print(r)
```

Output :

```
enter a number : 1245
5
4
2
1
```

## While loop : student challenge #1

Q) Display multiplication table for a given number

#We have a number and for that we have to print a multiplication table, as the pattern for multiplying is same we are using while loop here.

```
n = 5 # given number is 5
counter = 1 # starting counter from one( 5 * 1 = 5)
while counter <= 10: # condition of while is ,the counter value should be <= 0
    print(n, 'x', counter, '=', n * counter) # displaying multiplication table of given number
    counter = counter +1 # increase value of counter |
```

Output :

```
/Users/abdulbari1/Pychar
5 x 1 = 5
5 x 2 = 10
5 x 3 = 15
5 x 4 = 20
5 x 5 = 25
5 x 6 = 30
5 x 7 = 35
5 x 8 = 40
5 x 9 = 45
5 x 10 = 50
```

If you want take the number as input from the user then you can write the same program as

```
n = int(input('enter a number for multiplication table : ')) # TAKING INPUT FROM THE USER
counter = 1 # starting counter from one( 5 * 1 = 5)
while counter <= 10: # condition of while is ,the counter value should be <= 0
    print(n, 'x', counter, '=', n * counter) # displaying multiplication table of given number
    counter = counter +1 # increase value of counter |
```

Output :

```
/Users/abdulbari1/PycharmProjects/pythonprograms/
enter a number for multiplication table : 7
7 x 1 = 7
7 x 2 = 14
7 x 3 = 21
7 x 4 = 28
7 x 5 = 35
7 x 6 = 42
7 x 7 = 49
7 x 8 = 56
7 x 9 = 63
7 x 10 = 70
```

## While loop SC #2

1Q) Counting the number of digits in a number.

#Taking input from user in integrant write a counter condition in while loop for just counting the numbers given as input from user.

```
n = int(input('enter a number : '))# taking input from user
counter = 0      # counter starting from zero
while n > 0:    # while input number is greater than zero
    n = n // 10 # divide it by 10 and make it into int using //
    counter += 1 # modify counter value

print('number of digits are ', counter)# printing the output
```

Output :

```
enter a number : 12345667889
number of digits are 11
```

2Q) Finding sum of digits in a number

#If the number is given then find its sum by using while loop

```
n = int(input('enter a number : '))# taking input from user
sum = 0          # starting sum value from zero
while n > 0:    # while input number is greater than zero
    r = n % 10 # divide given number by 10
    n = n // 10 # divide it by 10 and make it into int using //
    sum = sum + r # modify counter value

print('sum of digits are ', sum)
```

Output :

```
enter a number : 6734
sum of digits are 20
```

### 3Q) Reversing a string

#Take input from user and make that given number into reverse order using while loop

```
n = int(input('enter a number : ')) # taking input from user
rev = 0      # starting reverse value from zero
while n > 0:  # while input number is greater than zero
    r = n % 10 # divide given number by 10
    n = n // 10 # divide it by 10 and make it into integer using //
    rev = rev * 10 + r  # modify counter value

print('reverse number is ', rev)
print(n) # it will be zero because condition becomes false then loop exit and next statement is print
```

Output :

```
enter a number : 3427865
reverse number is 5687243
0
```

4Q) check if a number is a palindrome .

# if the reverse of a number is equal to the original number then we say its a palindrome

```
n = int(input('enter a number : ')) # taking input from user
m = n # storing the value of n into m
rev = 0 # starting value is zero
while n > 0: # while input number is greater than zero
    r = n % 10 # divide given number by 10
    n = n // 10 # divide it by 10 and make it into int using //
    rev = rev * 10 + r # modify counter value

if m == rev: # if value of m is equal to rev then print if block
    print('number is a palindrome')

else: # else if value is not equal print else block

    print('number is not a palindrome')
```

Output :

```
enter a number : 1221
number is a palindrome
```

## While Loop Student Challenge #3

1 ) find sum of given numbers as input

#you have to ask number of numbers by taking user input and find the sum of that inputs .  
And we have to count so we use counter for it

```
num_of_nos = int(input('enter number of numbers'))
sum = 0
count = 0      # for counting purpose
while count < num_of_nos:    # while loop should be work till the count is lessthan num_of_nos
    n=int(input('enter a number'))
    sum=sum+n
    count=count+1      # add 1 in the present count
print('sum is ', sum)
```

2 ) Find the sum of +ve and -ve number

# user may enter positive or negative  
number .

```
num_of_nos = int(input('enter number of numbers'))
Psum = 0      #Positive sum
Nsum=0      #Negative sum
count = 0      # for counting purpose
while count < num_of_nos:    # while loop should be work till the count is lessthan num_of_nos
    n=int(input('enter a number'))
    if n > 0:      # if the number is greater than 0
        Psum = Psum + n      # do sum of postive number
    else :          #else if the number is lessthan 0 it is considered as negative
        Nsum = Nsum + n
    count = count + 1      # counting

print('Positive sum is ', Psum)
print ('Negative sum is', Nsum)
```

## While Loop Student Challenge #4

3 ) find the maximum numbers from the given number .

```
num_of_nos = int ( input (' enter number of number '))      # how many number of times you want to enter numbers
count = 0
max= int (input('Enter a number '))
while count < num_of_nos - 1:
    n = int(input ('Enter a number '))
    if n > max :      # if user have entered a number which is greater than max than take max = n
        max = n
    count = count + 1
print ( 'Max number is ', max)
```

4 ) convert decimal to binary

```
n = int (input('Enter a number'))
bin = 0
while n>0:
    r = n % 2
    n = n // 2
    bin = bin * 10 + r      #the number will be in reversed form
brev = 0
while bin > 0 :
    r = bin % 10
    bin = bin // 10
    brev = brev * 10 + r      # it will reverse the number
print(brev)
```

## While Loop Student Challenge #5

Q ) guess the number between 1 - 10

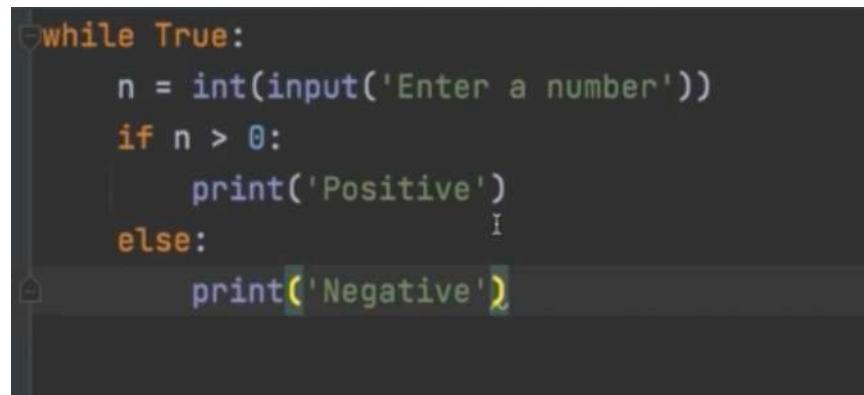
#User is prompted to enter a number. If the user guesses wrong then the prompt appears again( larger or smaller ) until the guess is correct, on successful guess, user will get a "correct answer!" message, and the program will exit.)

```
import random      #you have to import a module
n = random.randint(1,10)  #in random we take randint which will have random numbers
guess = 0
while guess != n:
    guess = int ( input ('Guess a number'))
    if guess < n :
        print (' it is smaller')
    elif guess > n :
        print('It is larger ')
    else :
        print ('correct guess')
```

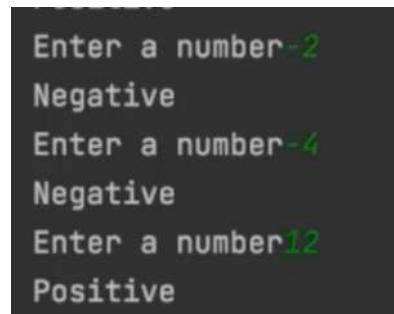
## Infinite Loop - break - continue - pass

- Infinite loop is a loop that goes on , which never stops such type of loop is called infinite loop
- The following example is of infinite loop

//This while loop never stops, you need to keep on giving input and it generated results it never stops



```
while True:
    n = int(input('Enter a number'))
    if n > 0:
        print('Positive')
    else:
        print('Negative')
```



```
Enter a number-2
Negative
Enter a number-4
Negative
Enter a number12
Positive
```

- To stop such infinite loop we can use **break** statement
- A break statement will stop the loop then and there
- A break statement can be used in other loop situations as well apart from infinite loop
- The below example shows the use of break statement

```
while True:  
    n = int(input('Enter a number'))  
    if n > 0:  
        print('Positive')  
    elif n < 0:  
        print('Negative')  
    else:  
        break;
```

```
Enter a number-3  
Negative  
Enter a number0  
  
Process finished with exit code 0
```

- **Continue** means it'll not execute the rest of the loop it'll simply go to beginning of the loop and continue its execution lets understand this with an example

```
count = 0  
  
while count < 10:  
    n = int(input('Enter a number'))  
    if n % 3 == 0:  
        continue  
    print(n)  
    count += 1
```

```
Enter a number1  
1  
Enter a number2  
2  
Enter a number3  
Enter a number4  
4  
Enter a number|
```

- Continue is used for logical design

- **Pass -** In some cases when you don't have to do anything use them just pass the it
- Pass means nothing to do
- In python when you write a block of code you must write something if there's nothing to write , then simply write pass statement.

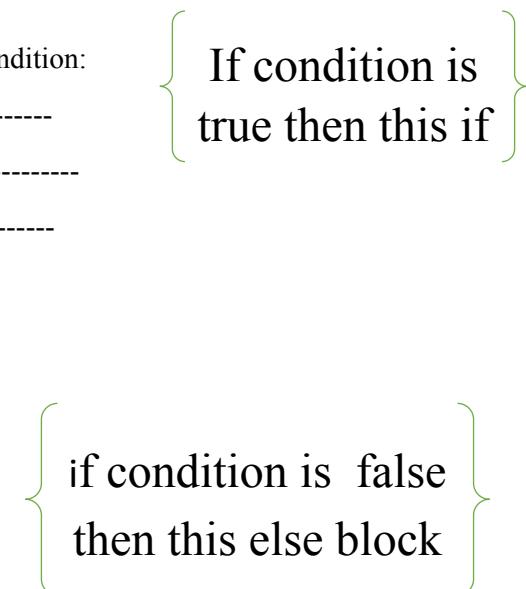
```
count = 0

while count < 10:
    n = int(input('Enter a number'))
    if n % 3 == 0:
        pass
    else:
        print(n)
    count += 1
```

```
Enter a number2
2
Enter a number3
Enter a number4
4
Enter a number5
5
Enter a number6|
```

## Else suite

```
if condition:  
-----  
-----  
-----  
  
else:  
-----  
-----  
---  
----  
-----  
-----  
-----  
-----
```



True-> if block is executed

False->else block is executed

### What is else suite?

- The else suite will be always executed irrespective of the statements in the loop are executed or not.
- If block can be written with while as well as it can also be written with for loop and also along with try block.

### How else suite is useful with while

Sample:

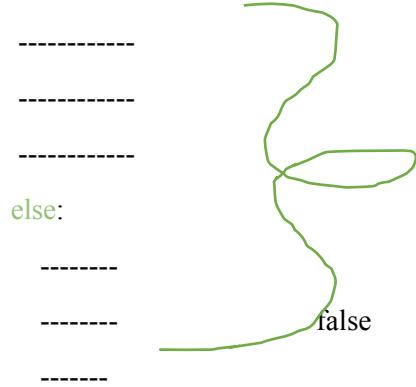
While condition

```
----- as long as this condition is true  
----- ← its get executed repeatedly  
-----  
-----
```

And once the condition becomes false it will come out of the loop and executes next statement.

We can also write else block.

while condition:



- if false it will come to else block
- Suppose if this loop is stopped using `break` statement without becoming false. Then this else block will not be executed.
- This else block will execute if only condition becomes false.
- So, it is like `if else` statement means true `if while` is executed and if it is false else will be executed.
- if `while` never becomes false then else block will never be executed.so, this else block can be used to confirm that while loop has successfully executed without any break.

### Working of else suite

## Coding

Input:

```
count=1
while count <= 10:# repeating while loop as long as the count is < or
= 10
    print(count) # printing count
    count += 1# and then count++
else:
    print('printed all 10 numbers')# confirms loop has stopped
normally by becoming false
    print('end of program')
```

Output:

1

2

3..... 10

printed all 10 numbers

- it has printed all the 10 numbers and also shows that it jumped to else on becoming false
- Now we will use break to stop the loop

Input:

```
count=1
while count <= 10:# repeating while loop as long as the count is < or
= 10
    print(count) # printing count
    count += 1# and then count++
    if count>5:# the loop stops at count 5
        break# using break to stop the loop
else:
    print('printed all 10 numbers')# confirms loop has stopped
normally by becoming false
print('end of program')
```

Output:

1  
2  
3  
4  
5

end of program

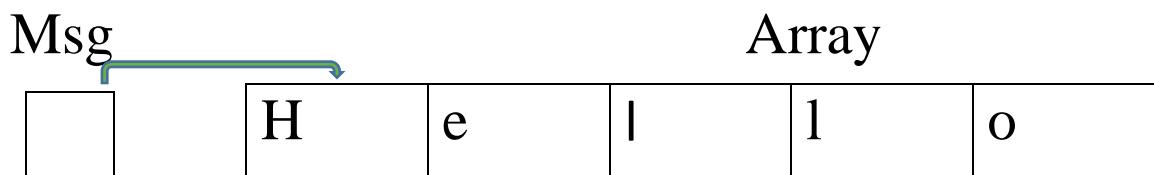
- so, else block is used to confirm that while loop has executed successfully. and it stops when it becomes false.
- If it stops abruptly using break then else will not execute.

## Introduction: For Loop

- Loops are useful for repeatedly executable statements
- While loop works on conditions whereas while loop don't

### How for loop works

Msg = 'Hello'  
↓      ↓  
Variable      string



So the string is arranged in the form of an array.

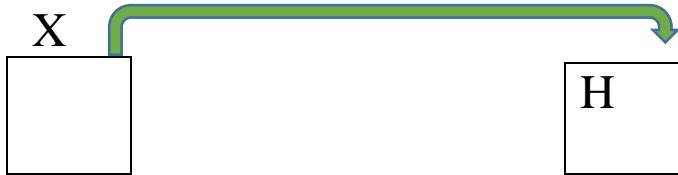
Suppose if we write For x in msg:

Print(x)

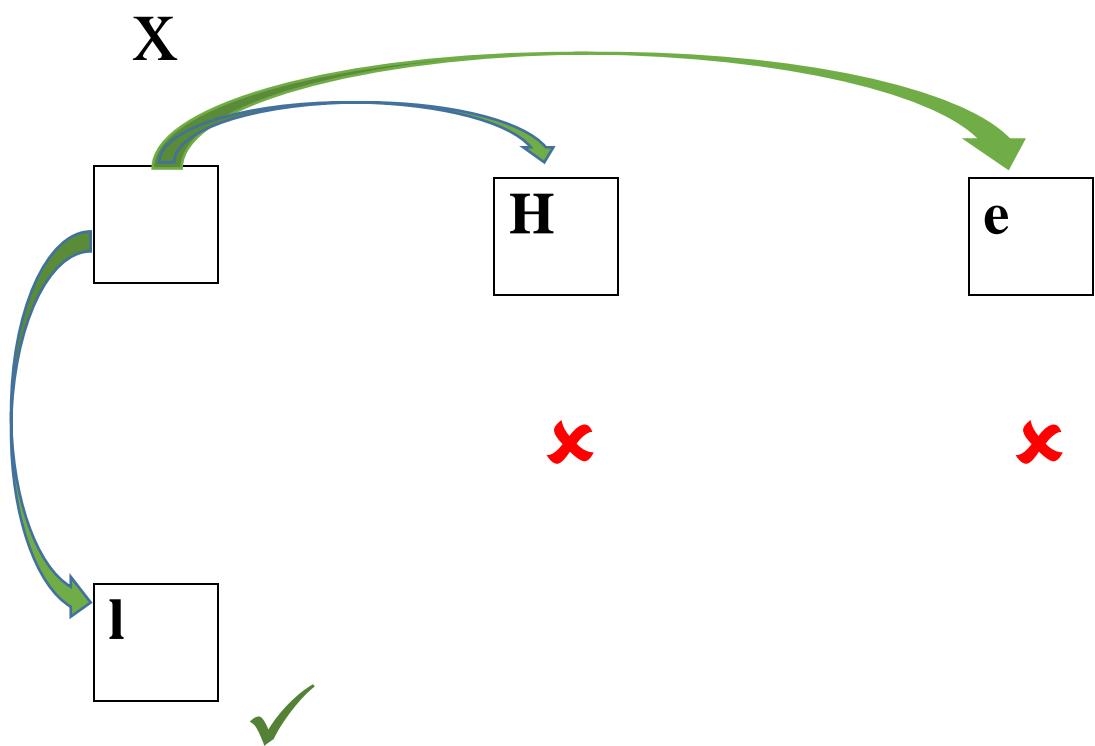
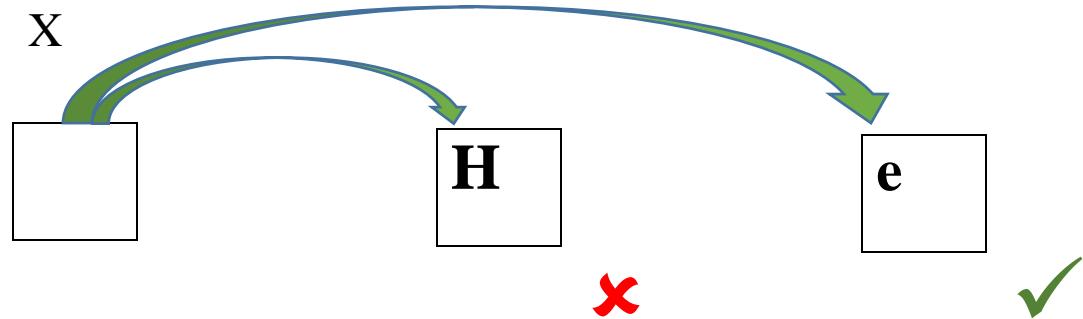
It will print like this

H  
e  
l  
l  
o

first it prints H



Then again it repeats using for and prints the next letter



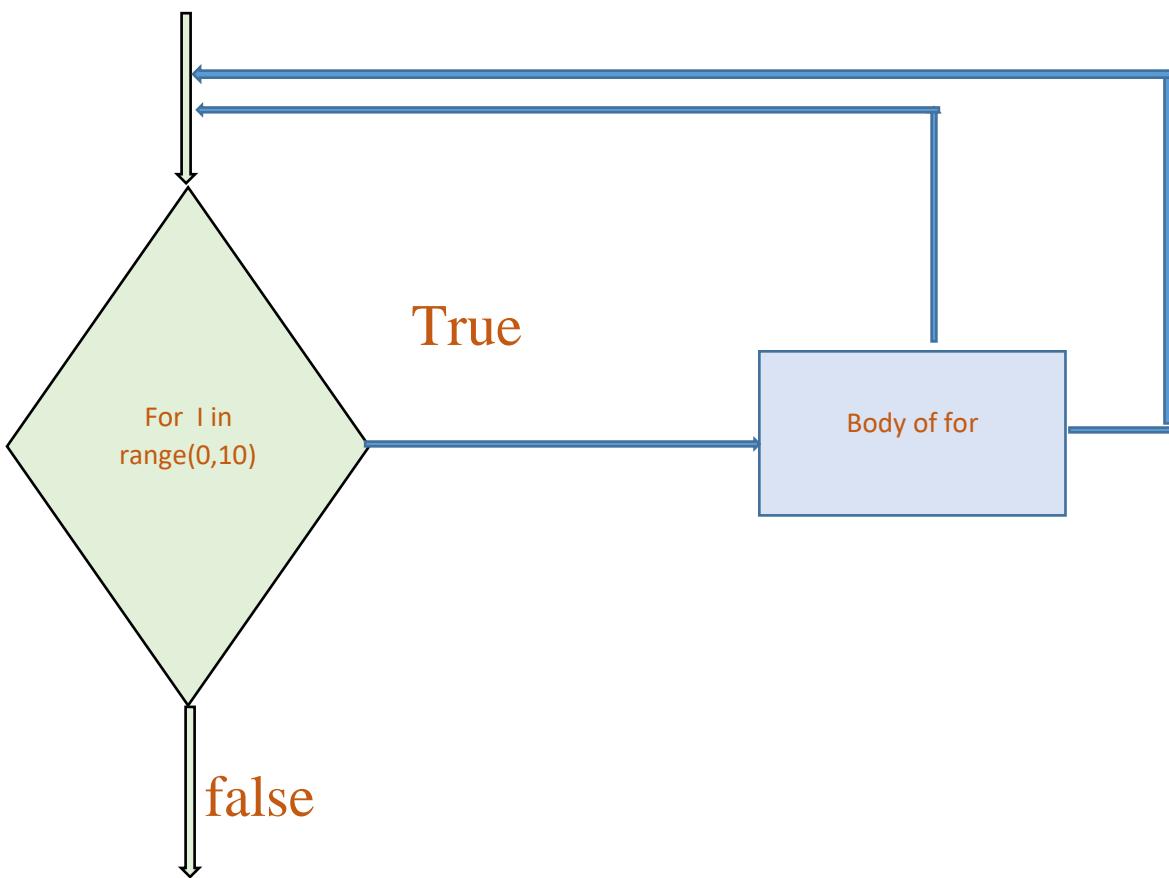
It goes on repeating till the end of an array

This loop is also called as foreach loop in languages like java and c++

Msg='hello'

For x in msg:#for loop runs for each statement in the sequence  
(print x)

## Flow chart(for loop)



so if the value condition is true so loop of the body is executed continuously and then at one point it stops.

Basically for loop depends upon the elements in a sequence.

Elements has finished means false and elements has not finished means true.

## What is Range?

- Range is a function
- Suppose range(10)
- 0,1,2,3,4,5,6,7,8,9 it will stop at just one number before the last number.
- If you give range(5,10) 5,6,7,8,9

- Start      End
- Can also give step range(1,10,2) 1,3,5,7,9
- Start      End      Step
- Range(10,0,-1) 10,9,8,7,6,5,4,3,2,1
- Start      End      Step
- Range(-10,-1,2) -10,-8,-6,-5,-4
- Start End Step
- this is how range function works

## Coding

### Input:

```
msg='Hello'  
for x in msg:  
    print(x)
```

## **Output:**

H

e

l

l

o

it's printing as H e l l o one by one . so, for loop executes for each element inside given statement.

Usually this types of things which are having elements called as sequence.

## **Input:**

```
for i in range(10): #it will take in from 0 to 10
    print(i)
```

## **Output:**

0

1

2

3

4

5

6

7

8

9

**Input:**

```
for i in range(5,10): #it will print 5 to 10
    print(i)
```

**Output:**

5  
6  
7  
8  
9

**Input:**

```
for i in range(0,10,2): #it will print on every
two steps from 0 to 10
    print(i)
```

**Output:**

0  
2  
4  
6  
8

Process finished with exit code 0

**Input:**

```
for i in range(10,0,-1): #it prints the numbers in
reverse order
    print(i)
```

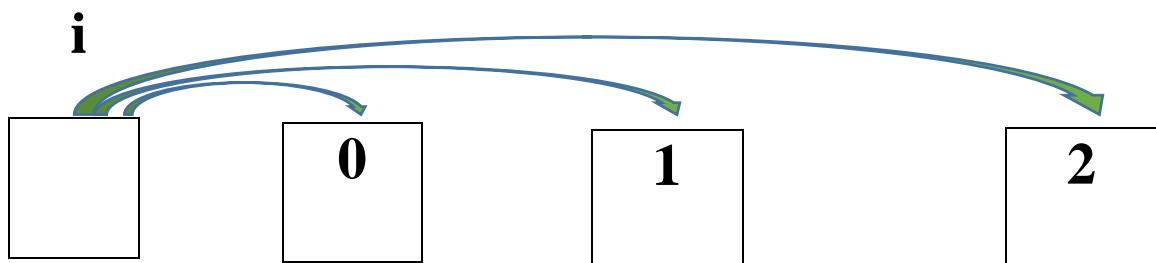
**Output:**

10  
9  
8  
7  
6  
5  
4  
3  
2  
1

How i is working ?

For i in range(0,10):

Print(i)



It works upto 10.

For each I in this range(0,10)

Print(i) will be executed

Each i is 0,1,2---upto 9

## Nested Loops

- We know that how for loops works
- For example

```
for i in range( 0,5):  
    print ( i )
```

Output : 0

```
1  
2  
3  
4
```

- Loop inside loop is said as nested loop

```
for i in range( 0,5) :  
    for j in range( 0,5) :  
        print ( i , j )
```

- The outer loop can contain any amount of inner loop
- In each iteration of the outer loop inner loop execute all its iteration. For each iteration of an outer loop the inner loop re-start and completes its execution before the outer loop can continue to its next iteration.

Working of nested loop

```
for i in range(0,5):  
    for j in range(0,5):  
        print(i,j, end=' ')  
    print()
```

```
0 0 0 1 0 2 0 3 0 4  
1 0 1 1 1 2 1 3 1 4  
2 0 2 1 2 2 2 3 2 4  
3 0 3 1 3 2 3 3 3 4  
4 0 4 1 4 2 4 3 4 4
```

- In this program, the outer `for` loop is iterate numbers from 0 to 5. The `range()` return 5 numbers. So total number of iteration of the outer loop is 5.
- In the first iteration of the nested loop, the number is 0. In the next, it 1.... and so on till 4.
- Next, For each iteration of the outer loop, the inner loop will execute five times. The inner loop will also execute five times because we are printing numbers from 0 - 5
- Total 25 times

## Nested student challenge #1

1 )To Print prime numbers 1-100.

#For finding numbers like if a number is given you have to do the count the factors of a number you have to make counts count=0. Then for I in range (1, n). From 1 to n itself we have write (n+1) .Here condition if  $n \% 1 == 0$

- Giving value zero so we have to increment to check whether a no. is prime or not we have to count the factors and find the number of factors.
- Outside of this for loop we have to write if count==2:
- Then we have to print that .Because , it's a prime number.

Otherwise doesn't need to print it.

So, the task here is printing prime number from 1to100 so, the value of this ranges from 1 to 100.

- we can write this entire logic inside a for loop

input:

```
for n in range(1,100+1):
    count = 0
    for i in range(1,n+1):
        if n%i == 0:
            count += 1
            if count == 2:
                print(n)
```

output:

2

3

4

5

6 ..... 10 ... 20 ... 30 ... 40.....100

## Match Case

Match case is used instead of switch case in languages like c++, java we use switch case it is like if else.

Input:

```
day = int(input('Enter Day number'))\n\nif day==1:\n    print('Sunday')\nelif day== 2:\n    print('Monday')\nelif day == 3:\n    print('Tuesday')\nelif day == 4:\n    print('Wednesday')\nelif day == 5:\n    print('Thursday')\nelif day == 6:\n    print('Friday')\nelif day == 7:\n    print('Saturday')\nelse:\n    print('Holiday')
```

Output:

```
Enter Day number1\nSunday
```

Input:

```
day = int(input('Enter Day number'))\n\nif day==1:\n    print('Sunday')\nelif day== 2:\n    print('Monday')\nelif day == 3:\n    print('Tuesday')\nelif day == 4:\n    print('Wednesday')\nelif day == 5:\n    print('Thursday')\nelif day == 6:\n    print('Friday')\nelif day == 7:\n    print('Saturday')\nelse:\n    print('Holiday')
```

Output:

```
Enter Day number-1
Holiday
```

It is similar to if else but syntax written is similar to switch case . But works same as if and elif>

## Introduction to string

- Anything written inside ‘ ’ , “ ” in quotes is called string
- String can contain english alphabet ,special characters like @ - \. / # ( ) , it can also contain characters from other languages also
- In memory strings looks like an **array of Character** it has +ve and -ve indexing as well
- When you call an input function normally it'll give string by default
- **len()** gives you length of a string
- We can use **for loop** for accessing every element in a string

```
>>> s = 'Hello'  
>>> type(s)  
<class 'str'>  
>>> s1='hello'  
>>> s2=input('Enter a String')  
Enter a Stringwelcome  
>>> type(s1)  
<class 'str'>  
>>> type(s2)  
<class 'str'>  
>>> s2 = input('Enter a String')  
Enter a Stringwelcome  
>>> len(s2)  
7  
>>> s1 = 'Hello'  
>>> for x in s1:  
        print(x)
```

```
H  
e  
l  
l  
o  
>>> |
```

- When a string value is given directly in the program then its called **string literals**
- String literals can be in ‘ ’ , “ ” , “ ” , “ ” “ ”
- When a string already contains a single quote ( inner quotes ) then you should enclose the string in double quotes (outer quotes ) and vice versa

Ex : s = “ John ' s ”

or

S = ' John " s '

If a string is in multiple lines then you have to use **triple single quotes (or) triple double quotes**

Ex : ' ' ' hello  
How are you ' ' '

or

" " " hello  
How are you " " "

# Operators on Strings

- The operators that work on strings are

**Concatenation**

**repetition**

**Indexing**

**Slicing**

**In**

**Not in**

- Concatenation :

- In python we can join / concatenate 2 strings using +

- Ex :**

```
s1 ='hello' s2 = 'world'
```

```
s3 = s1 + s2 = 'helloworld'
```

- Because string is immutable it will not modify it . It will create a new string
- When concatenation the 2 strings must be of string datatype only, if one is string and the other is integer this will not work

- Ex :**

```
s1 ='hello' s2 = 15 // error
```

To add a string and a number we must type case the integer value then add it

- Ex :**

```
s1 ='hello' s2 = str( 15 ) // (o/p)
```

- Repetition :

- We can multiply a string with integer numbers

- Ex :**

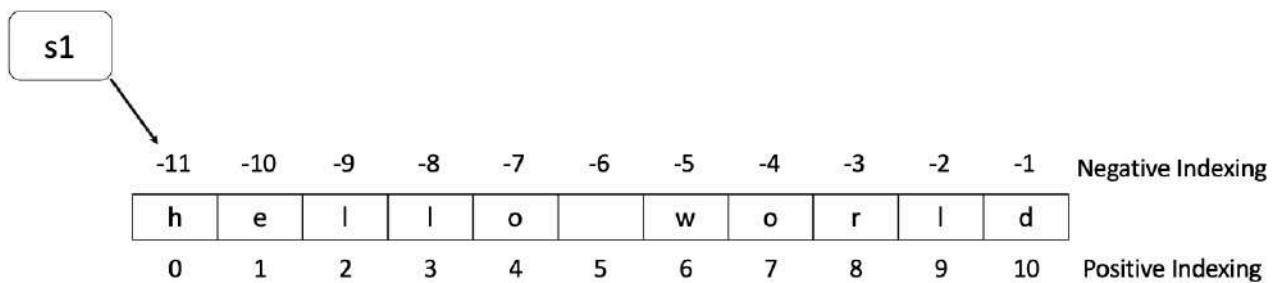
```
s1 ='hi' * 3 // (o/p) hihihi
```

- Repetition only work with a string and integer value only
- Float numbers cannot be used here .
- **Ex :**

`' hello ' * 2.5 // error`

### - Indexing :

- Suppose we are having a string s1 ='hello world ', it looks like this with indexing



- It has both +ve and -ve indexing values
- You can access any character of the string using index values.

`s1[ 0 ] -- h`

`s[ -11 ] -- h`

`s1[ 4 ] -- o`

`s1[ 6 ] -- w`

`s1[ -6 ] -- blank space`

### - Slicing :

- We can access values of strings using index
- Another way of accessing elements in a string is through slicing

- The syntax for slicing a string is

**s1[ start : stop : step ]**

- Using slicing we can extract specific portion of a string

Ex :

```
>>> s1[0:len(s1):1]
'Hello World'
>>> s1[:len(s1):1]
'Hello World'
>>> s1[::-1]
'Hello World'
>>> s1[3::]
'Hello llo'
>>> s1[6::]
'World'
>>> s1[6:8]
'Wo'
>>> s1[::-2]
'HloWrd'
>>> s1[::-1]
'dlroW olleH'
>>> s1[-1: -len(s1)-1:-1]
'dlroW olleH'
>>> s1[-1::-1] *
'dlroW olleH'
>>> s1[-1::-2]
'drWoH'
```

- **in :**

- It will say if a character is present in the string or not .
- If it is present then it will return True or else False .

Ex :

**s1 = 'hello world '**

h in s1 —— True

world in s1 —— True

me in s1 —— False

- **not in :**

- It will say TRUE if the character is not present

Ex :

**s1 = 'hello world '**

me not in s1 —— True

world not in s1 —— False

# Operators on Strings

- The operators that work on strings are

**Concatenation**

**repetition**

**Indexing**

**Slicing**

**In**

**Not in**

- Concatenation :

- In python we can join / concatenate 2 strings using +

- Ex :**

```
s1 ='hello' s2 = 'world'
```

```
s3 = s1 + s2 = 'helloworld'
```

- Because string is immutable it will not modify it . It will create a new string
- When concatenation the 2 strings must be of string datatype only, if one is string and the other is integer this will not work

- Ex :**

```
s1 ='hello' s2 = 15 // error
```

To add a string and a number we must type case the integer value then add it

- Ex :**

```
s1 ='hello' s2 = str( 15 ) // (o/p)
```

- Repetition :

- We can multiply a string with integer numbers

- Ex :**

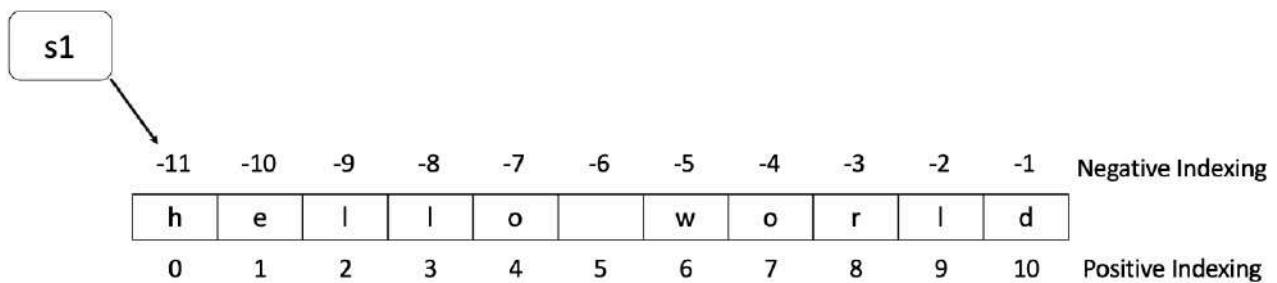
```
s1 ='hi' * 3 // (o/p) hihihi
```

- Repetition only work with a string and integer value only
- Float numbers cannot be used here .
- **Ex :**

`' hello ' * 2.5 // error`

### - Indexing :

- Suppose we are having a string s1 ='hello world ', it looks like this with indexing



- It has both +ve and -ve indexing values
- You can access any character of the string using index values.

`s1[ 0 ] -- h`

`s[ -11 ] -- h`

`s1[ 4 ] -- o`

`s1[ 6 ] -- w`

`s1[ -6 ] -- blank space`

### - Slicing :

- We can access values of strings using index
- Another way of accessing elements in a string is through slicing

- The syntax for slicing a string is

**s1[ start : stop : step ]**

- Using slicing we can extract specific portion of a string

Ex :

```
>>> s1[0:len(s1):1]
'Hello World'
>>> s1[:len(s1):1]
'Hello World'
>>> s1[::-1]
'Hello World'
>>> s1[3::]
'Hello llo'
>>> s1[6::]
'World'
>>> s1[6:8]
'Wo'
>>> s1[::-2]
'HloWrd'
>>> s1[::-1]
'dlroW olleH'
>>> s1[-1: -len(s1)-1:-1]
'dlroW olleH'
>>> s1[-1::-1] *
'dlroW olleH'
>>> s1[-1::-2]
'drWoH'
```

- **in :**

- It will say if a character is present in the string or not .
- If it is present then it will return True or else False .

Ex :

**s1 = 'hello world '**

h in s1 —— True

world in s1 —— True

me in s1 —— False

- **not in :**

- It will say TRUE if the character is not present

Ex :

**s1 = 'hello world '**

me not in s1 —— True

world not in s1 —— False

## String operator#2

### Slicing continuation

- To print a string in reverse we use -ve indexing , starting from -1
- Suppose if a string stops at -11 [ n ] then we should write -12 [ n-1 ] in the slicing operator to get -11.
- If we do not give an end point in slicing it takes the default value (I.e, where the list ends ) automatically , same goes for start point (default is 0) and stepsize (default is 1).
- Instead of using range based for loop we can use slicing to get the same result.
- Slicing gives a separate string.

### in , not in

- To check if a given sub string / character is present in a string we use `in` , `not in` operator.
- It returns Boolean values as a result I.e, [ T ,F ]

Present - True  
Not present - False

- `in` , `not in` is also called as Membership operator.
- `in` , `not in` make case sensitive comparison. (it considers lower and upper case differently)

## Introduction to string methods

- The string is a built - in class in python
- Classes contains definition / design ( there are of 2 types data and operation / methods )
- Objects is an instance of that definition design
- In string many member function are available these member functions are called **string methods**
- To know all the methods present in string class we can type a function called **dir( str )** it shows all the method
- You can access all the methods using **dot ( . )** operator (or) just want to know about particular method

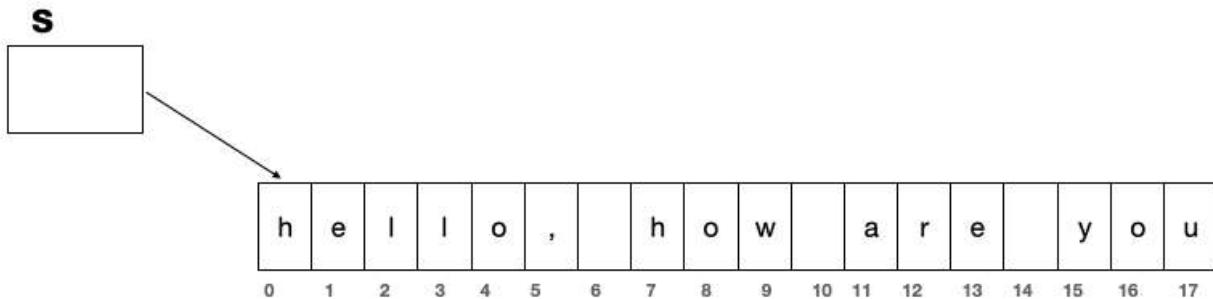
Ex : s = ' hello'  
help( s.endswith )

## String Method

- Methods are the member of the class which performs operation upon the data of an object

S = “ hello , how are you “

This is how String is stored



- It has 18 characters and it is hold by a reference that is s

**s.find ( sub [ , start [ ,end]])**

- sub ( find the occurrence of the substring )
- Methods are called by using object name that is variable name

`s.find('o')`

It will start looking for o from the left hand of the string . The result is 4 cause we found ‘ o ’ there

`s.find (' how ')`

The result will be 7

`s.find(' k ')`

The result will be -1 . Why it is -1 because the character will start from 0 . -1 means outside the range . So it is invalid position .

`s.find (' o ', 5)`

A diagram showing the parameters for the `s.find('o', 5)` method. The text `s.find('o', 5)` is at the top. Below it, two arrows point downwards to the words `sub` and `start`.

- If you want to find out after the 5th index we write 5 in starting index
- If you want to find out other substring then you should give starting and ending index

`s . find ('o', 5, 7)`

It will check from 5 to 7

- You can pass the find by single , double or 3 parameters

**s.rfind ( sub [ , start [ ,end]])**

- It is same as find but in find we where searching from left but in rfind will search substring from right

`s.rfind('o')` ————— 16 index

`s.rfind ('o ', 0 , 15 )` ————— 8 index

- In rfind ending index will work but in find starting index will work
- -1 will return if its out of string

**s.index ()**

- s.index and s.find is same but have minor differences . rindex is same as rfind

### s.count()

- It will count the number of occurrence  
Lets take an example - character ' o '

'o' — it repeated 3 times

- The count gives counting of the string. It will not gives all the indexes . It will only count

s.find (' k ') — -1

s.index (' k ') — substring not found

s.rindex('o ', 0 , 15) — 16

s.count ('me') — -0

s.count('how') — -1

## String Method #2

```
s.ljust(width[,fill])
s.rjust(width[,fill])
s.center(width[,fill])
```

- These methods are useful for text alignment

s = ' python '

- If you want to write in extra spaces like 10 spaces

`s.ljust()`

P y t h o n \_\_\_\_\_

`s.rjust()`

\_\_\_\_\_ P y t h o n

`s.center()`

\_\_\_\_\_P y t h o n \_\_\_\_

- If the string is larger than the spaces given by using

`s.ljust(3)`

- It will take the entire string it will not just take 3 letters
- If you want bigger space you mention the width bigger than the length of the string
- All this have one more parameter that is fill

`s.center(10 , '*' )`

- Python have only 6 alphabets but we want 10 spaces . \* will be filled in empty spaces
- 10 space vacant spaces with \* otherwise it will fill with spaces

s = 'python'

- String is immutable so it will not modify it will create a new string

`s.strip([ chars ])`  
`s.lstrip([ chars ])`  
`s.rstrip([ chars ])`

this is useful for removing the characters from the string

- They remove leading char , tailing characters and characters from both sides ... by default they will remove spaces

`s.lstrip` - it will remove leading char

`s.rstrip` - tailing removes character

`s.strip` - removes spaces from both the side

```
S = ' ..... ..... ++aaaaython '
```

s.lstrip(' . ') --> it will remove leading dots and stops when there is no dot

O/p - ' .....++aaaaython '

s.lstrip(' . + ') --> it will remove dot spaces and +

O/p - 'aaaaython'

- All this methods will return new string they will generate new string after performing the operations

## String Methods #3

- The methods we are discussing here are

```
s.capitalize()  
s.lower()  
s.upper()  
s.title()  
s.swapcase()  
s.casefold()
```

- Some methods are discussed below

```
>>>  
>>> s = "HeLlO WoRld"  
>>> s.capitalize()  
'Hello world'  
>>> s.lower()  
'hello world'  
>>> s.upper()  
'HELLO WORLD'  
>>> s.swapcase()  
'hElLo wOrLD'  
>>> |
```

- s.title()** , here first letter of every word in a string becomes capital
- s.casefold()** , It converts string into lowercase letters as s.lower() but there are some difference between them

```
>>> s = 'HELLO how Are YOU'  
>>> s.title()  
'Hello How Are You'  
>>> s  
'HELLO how Are YOU'  
>>> s.casefold()  
'hello how are you'  
>>> s  
'HELLO how Are YOU'  
>>> s.lower()  
'hello how are you'  
>>> s.casefold()  
'hello how are you'  
>>> s1='heLLo'  
< > s2='HELLO'  
>>> s1==s2  
False  
>>> s1.lower()==s2.lower()  
True  
>>> s1='Bu\u00DF'  
>>> s1  
'Buß'  
>>> s2='Buss'  
>>> s1==s2  
False  
>>> s1.lower()==s2.lower()  
False  
>>> s1|
```

```
>>> s1==s2
False
>>> s1.lower()==s2.lower()
False
>>> s1.casefold()==s2.casefold()
True
>>>
```

# String Methods

s.isupper()  
s.islower()  
s.istitle()  
s.isalnum()  
s.isalpha()  
s.isspace()  
s.isascii()

- It will return in boolean result ( T or F )

## s.isupper()

s = ' HELLO '

It will return true if only all the alphabets are in capital letters

## s.islower()

s = ' hellO '

It will return false because all should be in lower case but in this O is capital so its false

## s.istitle()

s = ' How Are You '

Every starting letter should be capital so it will return True and empty string will return False .

## s.isalnum()

- If the string contain alphabets and numbers it will return true . If it contain special alphabets than it will return false

s = ' abc-123' — — > False

## s.isalpha()

- If it is only alphabets then it will return true

s = ' abcd ' — —> True

s = ' abcd123' — — > False

This alpha and album works on any language

## s.isspace()

- It will check if there are any spaces present in the string then it will return true

s = ' hello world ' — —> True

## s.isascii()

- If it contain all the ASCII character , can have lower case , upper case , special character etc then it will return True

' abc12#! ' — —> True

s — —> True #empty string will return True



# String Methods

## s.isidentifier()

```
s = ' length1 ' --> True  
s = ' 1length ' --> False
```

Variable can not Start with a number so it will return false

- This method will check if it gives the valid name of the variable
- For keywords also it will return true  
s = ' if ' --> True

## s.isprintable()

- All the ASCII letters , other language letters are printable but there are escape characters \n \t \r are not printable so it will return false

```
s = ' hello ' --> True  
s = ' hello \n how are you ' --> False
```

## s.isdecimal()

It will say of there are decimal present in the string

```
s = ' 125 ' --> True  
s = ' 1.25 ' --> False
```

What are decimal ?? Decimal are the numbers between 0 - 9

## s.isdigit()

```
s = ' 168^2 ' --> True
```

- If you have special numbers it will return true
- But in s.isdecimal () it will return False

## s.isnumeric()

```
s = ' 5 ^1/2 ' --> True
```

- But in decimal and digit it will be false
- It will return true to any number



# String Methods

s = ‘ python is very easy ‘

## s.startswith(prefix [ , start [ , end ]])

- Start and end are in square brackets so it is optional
- It will say whether a given string is starts with
  - s.startswith ( ‘ python ‘ ) —→ True
  - s.startswith ( ‘is’ , 7 ) —→ True

## s.endswith(prefix [ , start [ , end ]])

- It will say if the string ends with ( certain string ) then it will return True

s.endswith(‘easy’ ) —→ False

## s.removesuffix(suffix , /)

- It will remove the substring

```
>>> s = 'python programming'
>>> s.removeprefix('py')
'thon programming'
>>> s
'python programming'
>>> s.removeprefix('java')
'python programming'
>>> s.removesuffix('ing')
'python programm'
```

## s.removeprefix(prefix , /)

- It will remove the beginning of the string if it is available and gives the original string
- The two methods( prefix , suffix ) will removes and gives back the new string . It will not modify the existing string

## s.partition( sep )

- It will divide
  - s.partition( ‘ is ‘ )
- It will check where is ‘is’ then it will form a tuple .

s.partition( ‘ is ‘ ) —→ ( ‘ python ‘ , ‘is’ , ‘ very easy ‘ )  
s.partition ( ‘ s ‘ ) —→ ( ‘ python i’ , ‘s’ , ‘ very easy ‘ )

## r.partition( sep )

- It will perform from the right hand side

r.partition ( ‘ s ‘ ) —→ ( ‘ python is very ea ‘s’ ‘y ‘ )



## String Methods #2

S.replace(old, new[,count])

S.join(iterable)

S.split([sep[, max split]])

S.rsplit([sep[, max split]])

s.splitlines([keepends])

s='a-b-c-d-e' want to replace all these hyphens with commas using these methods

S.replace('-', ',', 3)

So the first one is old and the next one is new

'a, b, c, d, e' -> This is a new string modify the old string and generate new string

We can mention the count but it is optional

```
s='a-b-c-d-e'  
s  
'a-b-c-d-e'  
s.replace('-', ',')#replacing it with comma  
'a,b,c,d,e'  
s  
'a-b-c-d-e'  
s.replace('k','m')  
'a-b-c-d-e'  
s='abcd@gmail.com'  
s.replace('gmail','yahoo')#replacing it with yahoo  
'abcd@yahoo.com'
```

This is the replace method

s.join(iterable)

The join method:

```
s1='xyz'  
s2='abc'  
s1.join(s2)# calling s1 method and joining upon s2  
'axyzbxyzc'  
s1='/'  
s1.join(s2)# letters of s1 and s2 as a separator  
'a/b/c'
```

So, the parameter is string so its taking letters of a string

**The split method:**

```
s='john smith ajay'  
s.split()# it splits
```

```
['john', 'smith', 'ajay']
s.split('h')
['jo', 'n smit', ' ajay']
s.split(',')
['john smith ajay']
s='john-smith-ajay-khan-james'
s.split('-')
['john', 'smith', 'ajay', 'khan', 'james']
```

### The s. rSplit method:

The s. Split means splitting is done from right hand side

**S.splitlines** works same as split only

```
s='aaa\n' bbb\t ccc\t
```

## ASCII vs Unicode

| Dec Chr |
|---------|---------|---------|---------|---------|
| 0 NUL   | 26 SUB  | 52 4    | 78 N    | 104 h   |
| 1 SOH   | 27 ESC  | 53 5    | 79 O    | 105 i   |
| 2 STK   | 28 FS   | 54 6    | 80 P    | 106 j   |
| 3 ETK   | 29 GS   | 55 7    | 81 Q    | 107 k   |
| 4 EOT   | 30 RS   | 56 8    | 82 R    | 108 l   |
| 5 ENQ   | 31 US   | 57 9    | 83 S    | 109 m   |
| 6 ACK   | 32 :    | 58 :    | 84 T    | 110 n   |
| 7 BEL   | 33 !    | 59 ;    | 85 U    | 111 o   |
| 8 BS    | 34 "    | 60 <    | 86 V    | 112 p   |
| 9 HT    | 35 #    | 61 =    | 87 W    | 113 q   |
| 10 LF   | 36 \$   | 62 >    | 88 X    | 114 r   |
| 11 VT   | 37 %    | 63 ?    | 89 Y    | 115 s   |
| 12 FF   | 38 &    | 64 @    | 90 Z    | 116 t   |
| 13 CR   | 39 '    | 65 A    | 91 [    | 117 u   |
| 14 SO   | 40 (    | 66 B    | 92 \    | 118 v   |
| 15 SI   | 41 )    | 67 C    | 93 ]    | 119 w   |
| 16 DLE  | 42 *    | 68 D    | 94 ^    | 120 x   |
| 17 DC1  | 43 +    | 69 E    | 95 _    | 121 y   |
| 18 DC2  | 44 ,    | 70 F    | 96 ~    | 122 z   |
| 19 DC3  | 45 -    | 71 G    | 97 a    | 123 {   |
| 20 DC4  | 46 .    | 72 H    | 98 b    | 124 }   |
| 21 NAK  | 47 /    | 73 I    | 99 c    | 125 }   |
| 22 SYN  | 48 0    | 74 J    | 100 d   | 126 ~   |
| 23 ETB  | 49 1    | 75 K    | 101 e   | 127 DEL |
| 24 CAN  | 50 2    | 76 L    | 102 f   |         |
| 25 EM   | 51 3    | 77 M    | 103 g   |         |

- Computers understand binary language.
- The data we provide to computer will be stored in binary form only.
- Ex: if we have number 10 then it is stored in the form binary i.e 1010

10 1010 both are numeric

How our computer system understand s alphabets letters, English alphabets?

As computers can understand only numeric how do we make them understand letters and alphabets?

- The solution is for English characters and other characters. We define some codes.
- Let us say 65-A
- If giving number and saying that it is a number
- If given number and saying a character then it is a code
- For English letter and other characters there are some codes. And those codes are called as ASCII codes.
- ASCII- American standard code for information interchange so, for every letter there is a code available.
- When we say a character A so, the number 65 will not appear on the screen it will appear as A only.

A->65

Z->90

a->97

Z->122

ASCII code are available for English uppercase alphabets lower case alphabets and digits as well as the special case characters and also the control characters that are available on the keyboard

ASCII code range from 0-127=128

Total 128 codes are there  $128 = 2^7$  ( 2 to the power of 7)

Within 7 bits ASCII code can be represented.

When we convert into binary form. The binary digits will be 7 not more than 7

If we take 65

1000001

1 byte= 8 bits

Enough to store ASCII Code

s="A"

ord(s)

65

ASCII code is a universal code which every machine follows in the universe.

## Unicode

- For every language there are different codes and those codes combined are called as unicode.
- It can be termed as unique code or universal code.
- For representing UTF(Unicode Transformation Format) is followed called as UTF-8 or UTF-16 and UTF-32. This type of formats are followed.
- UTF-8 means allows 1 byte also 2,4
- UTF-32-4 bytes
- Where to find unicode
- Unicode.org —>code—>select any.

How to use unicode

```
s='\u03b1'  
s  
'a'  
s='\u03b1\u0B32\u03B3'  
s  
'ାବ୍ୟ'
```

Every code represents unique alphabet. Each of these is called code byte.  
For any language we can visit any website and choose codes

```
s='\u0915\u0916\u0917'  
s  
'କଖା'  
s='u0041'  
s  
'u0041'  
s='\u0041'  
s  
'A'  
s='\x41'  
s  
'A'
```

## Escape Sequence #1

- It is the special characters used to perform operations and has meaning related to them and it is the part of the String

Escape Sequence	Meaning
\	Backslash (\)
'	Single Quote ('')
"	Double Quote ("")
\a	ASCII Bell (BEL)
\b	ASCII Backspace (BS)
\f	ASCII Formfeed (FF)
\n	ASCII Linefeed (LF)
\N	Character named <i>name</i> in the Unicode database
\r	ASCII Carriage Return (CR)
\t	ASCII Horizontal Tab (TAB)
\uxxxx	Character with 16-bit hex value (Unicode only)
\Uxxxxxxxxx	Character with 32 bit hex value (Unicode only)
\v	ASCII Vertical Tab(VT)
\ooo	Charcter with octal value ooo
\xhh	Character with hex value hh

## Escape Sequence #2

If you want to print john's

- `print('john's')`

It will cause an error because we can't use single quotes inside single quote

You can close with double quotes

`print("John's")`

Or you can use

`print('John\s')`

- \n is use for new line but if u want to print \n in the string than

`Print ('abcd\\ndef')`

Output : abcd\ndef

- If you want to print(`r'abc\ndef'`)

Here r is a raw string after that what ever we use in escape sequences it will be treated as character

Output : abc\ndef

- Without using the unicode we can give the signs for example

`s='N{pound sign}'`

Output:`£`

Like this we can print too many signs

Link :

<https://unicode.org/Public/UNIDATA/NamesList.txt>

## Print Function

- Print take 1st input as object. It can print any object like int , float , string , complex , list , dictionary even when you define your own class you can print the results

```
type help; copyright; exit
>>> a=10
>>> b='Ravi'
>>> c=86.57643
>>>
>>> print(a)
10
>>> print(c)
86.57643
>>> print(a,b,c)
10 Ravi 86.57643
>>> print('hello', 'world')
hello world
>>> print('hello'+'world')
helloworld
>>> print(a, b, c, sep='-')
10-Ravi-86.57643
>>> |
```

- Print always gives space between 2 object automatically
- But if you concatenate (+) two strings then no space is given between them when you print it
- **Sep** is specifying a separator using which we can separate 2 strings you can use any separator value like - , \ , / , etc...
- Print after printing one line will move to the next

```
a = 10
b = 'Ravi'
c = 86.54675

print(a)
print(b)
print(c)
```

```
10
Ravi
86.54675
```

- **\n** gives new line , **\t** gives tab space
- **Flush** is used to push the program from buffer (temporary memory) into monitor/screen. Flush is mostly used in multithreading

## C style formatting

C language style of printing

Formatting :

Example: roll no=10  
name='Ravi'  
avg=80.2593127

Now if we want to print it as a message

```
print('student name is %s, his roll no is %d and avg %f' % (name, rollno, avg)) # this is formed string
```

%s for name, %d for data and integer value, %f for float value

These controlled characters are used in c and as well as python uses these control characters

IDLE

Program:

```
>>> 
>>> rollno = 10
>>> name = 'Ravi'
>>> avg = 86.29714
>>>
>>> print('Student name is %s, his roll no is %d and average is %f' % (name, rollno, avg))
Student name is Ravi, his roll no is 10 and average is 86.297140
>>>
```

Counter characters	Purpose
%s	String
%d	Integer
%f	Float
%x	Hexa decimal

```
>>>
>>> rollno = 10
>>> name = 'Ravi'
>>> avg = 86.29714
>>>
>>> print('Student name is %s, his roll no is %d and average is %f' % (name, rollno, avg))
Student name is Ravi, his roll no is 10 and average is 86.297140
>>>
>>>
>>> print('Student name is %s' % name)
Student name is Ravi
>>> print('Student name is %10s' % name)
Student name is      Ravi
>>> print('roll number is %8d' % rollno)
roll number is    10
>>> print('roll number is %-8d' % rollno)
roll number is 10
>>> print('roll number is %-8d and' % rollno)
roll number is 10 and
>>> print('%2.5f' % avg)
86.29714
>>> print('%2.3f' % avg)
86.297
>>> |
```

---

## Formatted Printing

- Suppose you have 2 variables and you want to divide them using string formatting then

```
a = 22  
b = 7  
C = a / b
```

- Using these variable we'll format a string how lets see it

```
print('division of {0} and {1} is {2}'.format(a,b,c))
```

{ } - this is called a **placeholder** , values written inside them are index

- If you don't give idea number in-between { } it'll still work
- If you give different indices in-between { } it'll work but you should provide only the indices you are having and not any other index value
- In short you can give place holders in any order

```
>>> a = 22  
>>> b = 7  
>>> c = a / b  
>>> print('Division of {} and {} is {}'.format(a,b,c))  
Division of 22 and 7 is 3.142857142857143  
>>> print('Division of {0} and {1} is {2}'.format(a,b,c))  
Division of 22 and 7 is 3.142857142857143  
>>> print('Division of {2} and {1} is {0}'.format(a,b,c))  
Division of 3.142857142857143 and 7 is 22  
>>> print('Division of {2} and {1} is {0}'.format(c, b, a))  
Division of 22 and 7 is 3.142857142857143  
>>>
```

- We can mention width between strings using formatting like this

```
>>>  
>>> print('Division of {0:10} and {1:15} is {2:2.4}'.format(a,b,c))  
Division of      22 and        7 is 3.143  
>>> |
```

- We can mention the alignment as well such as <, >, ^ etc

```
>>> print('Division of {0:<10} and {1:^15} is {2:2.4}'.format(a,b,c))
Division of 22      and      7      is 3.143
>>>
```

- We can also do Conversions using formatted printing

```
>>> print('Division of {0:<10} and {1:^15} is {2:E}'.format(a,b,c))
Division of 22      and      7      is 3.142857E+00
>>> print('Division of {0:<10} and {1:^15} is {2:F}'.format(a,b,c))
Division of 22      and      7      is 3.142857
>>>
```

- Some more flag and conversation type are

flag - < > ^ := + - : ; _
Conversion - :b :c :d :e :E :f :F :g :G :o :x :X :n :%

## Formatted Printing ( New python style )

- Here in new style we can write the variable name in Formatting directly in the placeholder along with values.
- Before the start of string we should write **f** to declare that the string is formatted string

```
>>> a = 22
>>> b = 7
>>> c = a / b
>>>
>>> print(f'Division of {a:10} and {b:^15} is {c:2.4f}')
Division of    22 and      7   is 3.1429
>>>
```

## Regular Expression #1

- Regular Expression are useful for **defining patterns for a string** matching , it is useful for checking the conditions
- Regular Expressions or Regex is very powerful
- Suppose you want to match a pattern with a string we can use Regular Expressions
- Lets look at some patterns for strings

Pattern	String	
1. ‘a’	‘a’	
2. ‘a   b ’	‘a’ or ‘b’	( Either a or b is True )
3. ‘abc’	‘abc’	( Exactly abc will give True )
4. ‘ [ abc ] ’	‘a’ or ‘b’ or ‘c’	( Either a ,b ,c will give True )

- Lets try this with an example
- To use regular Expressions we need to import re module

```
>>> from re import *
>>>
>>> match('a', 'a').group()
'a'
>>> match('a|b', 'a').group()
'a'
>>> match('a|b', 'b').group()
'b'
>>> match('abc', 'abc').group()
'abc'
>>> match('abc', 'abcd').group()
'abc'
>>> match('[abc]', 'abcd').group()
'a'
>>> match('[abc]', 'bcd').group()
'b'
>>> |
```

- You can also include some characters like `+, *, ?, {m}, {m, n}` in Regular Expression

Character	Description
<code>+</code>	1 or more repetitions
<code>*</code>	0 or more repetitions
<code>?</code>	0 or 1 repetitions
<code>{ m }</code>	Exactly m occurrences
<code>{ m , n }</code>	From m to n . m defaults to 0 . n to infinity

Pattern	String
5. <code>'[abc]+'</code>	a, ab, aaaa, ababab, cccc

```

>>> match('[abc]+', 'bcd').group()
'bc'
>>> match('[abc]+', 'ccccc').group()
'ccccc'
>>> match('[abc]+', 'ababcbcbaba').group()
'ababcbcbaba'
>>> match('[abc]{5}', 'ababcbcbaba').group()
'ababc'
>>>

```

## Regular expression #2

lets take example for better understanding

**pattern**   **string.**

1)     [**a-z**]                                        'a' 'b' 'k'

- It will return true because every alphabet between a-z will return true square bracket represent single alphabet

2)     [**a-z**]**+**                                        'ask'        'python'

- It says a to z + (more than one ) it can be one or more time .but should be in lower cases

3)     [**a-zA-Z0-9**]**+**                                        ' abc123'

- Its for lower, upper and 0-9 digits or numbers

4)     [^**a-z**]**+**                                        ' AB1' 'A-5-7'

- It means except a to z everything will be the match.

5 )    [**a-z**]**+: [A-z]****+**    ' BAT'

- It will allow lower or lower alphabets it will allow only lower or only upper not mixed.

6)     [.]**+**

- Dot means anything , alphabets , numbers or special characters

7 ).      '^hell'                          ' Hello world'

- Is it starting will hello

8)      'orl\$'                          'Hello world'

- Is the string is ending will orld.

## Regular Expression #3

Sequence characters

Character	Description
\d	Digit ( 0 - 9 )
\D	Non digit
\s	White space . Ex : \t \n \r \f \w
\S	Non - white space
\w	Alphanumeric ( A to Z , a to z , 0 to 9 )
\W	Non - alphanumeric
\b	Space around words
\A	Start of the string
\Z	End of the string

Pattern	String
\d+	'012' '897' '11011'
[0-9]+'	'012' '897' '11011'
/D+'	Abcd + - ' '+ - /<>@\$'

## Functions of Regular expression

```
re.compile(pattern, flags=0)
re.search(pattern, string, flags=0)
re.match(pattern, string, flags=0)
re.findall(pattern, string, flags=0)
re.sub(pattern, repl, string, count=0, flags=0)
```

## Introduction to list

- List is a collection of **ordered objects** and **can have duplicates**
- It is created using **[]** and items inside are separated using a **( , )** comma
- A list have **+ve and -ve index** as well
- A list can be created in 2 ways that is

```
List1 = [ 1,2,3,4,5 ]  
List2 = list( ( 1,2,3,4,5 ) )
```

```
>>>  
>>> Mylist=['john', 'smith', 'mark', 'eric', 'smith']  
>>> Mylist  
['john', 'smith', 'mark', 'eric', 'smith']  
>>> print(Mylist)  
['john', 'smith', 'mark', 'eric', 'smith']  
>>> list1=list(1,2,3,4,5)  
Traceback (most recent call last):  
  File "<pyshell#4>", line 1, in <module>  
    list1=list(1,2,3,4,5)  
TypeError: list expected at most 1 argument, got 5  
>>>  
>>> list1=list((1,2,3,4,5))  
>>> list1  
[1, 2, 3, 4, 5]  
>>> Mylist  
['john', 'smith', 'mark', 'eric', 'smith']  
>>> Mylist[2]  
'mark'  
>>> Mylist[-2]  
'eric'  
>>> |
```

- List is **heterogeneous** i.e, it can contain different type of data In it

**Ex :** Mylist = [ ‘John’, 15, 14.6, True , ‘Steven’, 5+7j ]

- List is **mutable** [ changeable ] , you can change any value in a list
- **Len( )** given length of a list

```
>>> Mylist=[15, 9,12,18, 7,10]
>>> Mylist
[15, 9, 12, 18, 7, 10]
>>> Mylist[0]=30
>>> Mylist
[30, 9, 12, 18, 7, 10]
>>> Mylist[4]='john'
>>> Mylist
[30, 9, 12, 18, 'john', 10]
>>> len(Mylist)
6
>>>
```

- **Append( )** is used to add more values to a list

**Ex :** Mylist = [ 1,2,3,4,5,6 ]  
Mylist.append( 50 )

O/p:  
[ 1,2,3,4,5,6,50 ]

## List Operator #1

- The operators on list discussed here are `[]`, `[:]`
- `[]` is used for **indexing**, Both positive and negative indexing is possible on list through indexing we can read the data of list as well as write / modify a list

```
>>>
>>> list1 = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>> list1[6]
7
>>> print(list1[6])
7
>>> print(list1[-4])
7
>>> x=list1[6]
>>> x
7
>>> list1[6]=15
>>> list1
[1, 2, 3, 4, 5, 6, 15| 8, 9, 10]
>>>
```

- **[ : ] - slice / slicing operator** is similar to index , but we can take a range of values in a list
- In slicing you can give `[ start : end : stepsize ]`
- Slicing will give you new list it doesn't modify the existing list
- It is used for reading either the complete list or a section of a list
- It is discussed in detail in the example below

```
>>>
>>> list1 = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>> list1[:]
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>> list1[3:]
[4, 5, 6, 7, 8, 9, 10]
>>> list1[3:8]
[4, 5, 6, 7, 8]
>>> list1[0:10]
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>> list1[0:9]
[1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> list1[0:10:2]
[1, 3, 5, 7, 9]
>>>
>>> temp = list1[0:10:2]
>>> list1
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>> temp
[1, 3, 5, 7, 9]
>>> list1[::-1]
[10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
>>> list1[-1:-11:-1]
[10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
>>> list1[-1:-11:-2]
[10, 8, 6, 4, 2]
>>>
```

```
>>>
>>> list1 =[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>> list1[0:3]=[10,20,30]
>>>
>>> list1
[10, 20, 30, 4, 5, 6, 7, 8, 9, 10]
>>> list1[0:3]=[11,12]
>>> list1
[11, 12, 4, 5, 6, 7, 8, 9, 10]
>>>
>>> list1[0:2]=[10,20,30,40,50]
>>> list1
[10, 20, 30, 40, 50, 4, 5, 6, 7, 8, 9, 10]
>>>
>>> list1[::-2]=[11,22,33,44]
Traceback (most recent call last):
File "<pyshell#11>", line 1, in <module>
  list1[::-2]=[11,22,33,44]
ValueError: attempt to assign sequence of size 4 to extended slice of size 6
>>> list1[::-2]=[11,22,33,44,55,66]
>>> list1
[11, 20, 22, 40, 33, 4, 44, 6, 55, 8, 66, 10]
>>> |
```

## Operators on list

- ‘+’ is also known as **concatenation operator**
- In list ‘+’ will join two list to make a new list which contains all the elements of two list.
- The ‘+’ doesn’t modify the existing list it just simply generate a separate list i.e, a new list.
- If you want to add an element to a list you need to first make it as a list then concatenate it with the list in which you want to add it

Example :

```
>>> list1 = [1,2,3]
>>> list2 = list1 + [4]                                # adding 4 to list1 and storing this in
>>> list2                                              another list i.e list2
[1, 2, 3, 4]
>>>
```

- If you want to expand/ modify an existing string then you can use a method called **extend()** to include more elements.
- By using **extend()** you can change a list.

```
>>> list1 = [1,2,3]
>>> list1.extend([4,5,6])
>>> list1
[1, 2, 3, 4, 5, 6]
>>>
```

- You can modify the list as follows as well

```
>>>
>>> list2 = [7,8,9]
>>> list2 = list2 + [10,11,12]
>>> list2
[7, 8, 9, 10, 11, 12]
>>>
```

Here, list2 is modified simply , by assigning the old list2 to new list2 and concatenating old list2 with another list hence , modified new list2 is created by concatenation.

- ‘ \* ’ is a [repetition operator](#).
- Float values cannot be used for repetition only integer values should be used.

## Example:

```
>>>
>>>
>>> list1 = [1,2,3]
>>> list1 * 2
[1, 2, 3, 1, 2, 3]
>>>
>>>
>>> list1 = [1,2,3]
>>> list1 * 2.5
Traceback (most recent call last):
  File "<pyshell#22>", line 1, in <module>
    list1 * 2.5
TypeError: can't multiply sequence by non-int of type 'float'
>>> |
```

- The other two operators are [in](#) , [not in](#)
- They check if an element is present in the list or not and return boolean type of result.
- Let us see this with an example

## Example:

```
>>>
>>> list1 = [1,2,3]
>>> 2 in list1
True
>>>
>>> 10 in list1
False
>>> 10 not in list1
True
```

- If it is present then it will return true if not than it will return false

## List iteration

### Iterating a list

```
list1=
```

0	1	2	3	4
5	6	7	8	9
-5	-9	-3	-2	-1

Iterating a list means visiting a list or accessing all the list elements in a list one by one. We can also called it as traversing>

We can go reverse also.

Iteration is done using for loop also

Let's see some examples to make a list

```
>>> list1 = [5, 6, 7, 8, 9]
>>> for x in list1:
    print(x)
```

```
5
6
7
8
9
>>> |
```

For loop using range

```
>>> for i in range(len(list1)):
    print(list1[i])
```

```
5
6
7
8
9
>>> |
```

If want to Start from 2

```
>>> for i in range(2,len(list1)):
    print(list1[i])
```

```
7
8
9
>>> |
```

It is not printing element at 0 and 1 that is 5 and 6 and it is starting printing from second element that is 7

If we want to print numbers in reverse so, we can print using positive indices also negative indices also.

```
>>> list1 = [5, 6, 7, 8, 9]
>>> for i in range(len(list1)-1,-1,-1):
    print(list1[i])

9
8
7
6
5
>>> |
```



## List Method #1

- Methods are member function of a class
- The methods discussed here are **append( x ) , extend( iterable ) , insert( i, x ) , copy ()**

### append( x )

- It means adding one element at a time to a list
- It modifies the same list
- We can use slicing on append method

```
>>>
>>> L1 = [5,6,7,8,9]
>>> len(L1)
5
>>> L1.append(10)
>>> len(L1)
6
>>> L1
[5, 6, 7, 8, 9, 10]
>>> L1.append(11,12,13)
Traceback (most recent call last):
File "<pyshell#6>", line 1, in <module>
  L1.append(11,12,13)
TypeError: list.append() takes exactly one argument (3 given)
>>>           I
>>>
>>> L1 = [5,6,7,8,9]
>>>
>>> L1[len(L1):]=[10]
>>> L1
[5, 6, 7, 8, 9, 10]
>>> L1[6:6]=[11]
>>> L1
[5, 6, 7, 8, 9, 10, 11]
>>> |
```

## extend ( iterable )

- It is same as append but it can take collection of element in parameter
- It will modify the same list
- Extend can take list , tuple and string as parameter because they are iterable as well
- Slicing is also applicable on extend method

```
>>>
>>> L1 = [5,6,7,8,9]
>>> L1.extend([10,11,12])
>>> L1
[5, 6, 7, 8, 9, 10, 11, 12]
>>> L1.extend('abc')
>>> L1
[5, 6, 7, 8, 9, 10, 11, 12, 'a', 'b', 'c']
>>>
>>> L1=[5,6,7,8,9]
>>> L1[len(L1):]=[10,11,12]
>>> L1
[5, 6, 7, 8, 9, 10, 11, 12]
>>> |
```

## Insert ( i , x )

- It is used to insert any element at a given index
- Same list is modified here
- Slicing is applicable on insert method

```
>>>
>>> L1 = [5,6,7,8,9]
>>> id(L1)
140505553016192
>>> L1.insert(0,10)
>>> L1
[10, 5, 6, 7, 8, 9]
>>> id(L1)
140505553016192
>>>
>>> L1
[10, 5, 6, 7, 8, 9]
>>> L1.insert(3,20)
>>> L1
[10, 5, 6, 20, 7, 8, 9]
>>> L1[4:4]=[22]
>>> L1
[10, 5, 6, 20, 22, 7, 8, 9]
>>> L1[0:0]=[25]
>>> L1
[25, 10, 5, 6, 20, 22, 7, 8, 9]
>>>
```

## Copy

- It copy( ) will create a shallow copy of the list
- It returns a new list after copying it

```
>>>
>>> L1 = [5, 6, 7, 8, 9]
>>> L2=L1.copy()
>>> L2
[5, 6, 7, 8, 9]
>>> L1
[5, 6, 7, 8, 9]
>>> id(L1)
140265482259968
>>> id(L2)
140265443651840
>>> L1[0]
5
>>> L2[0]
5
>>> id(L1[0])
140265436354992
>>>
```

## List Methods #2

- The methods used to remove elements from a list are  
`pop( [ ] ) , remove( x ) and clear ()`

### **pop( [ ] )**

- It will delete the last element from a list if index is not given
- By mentioning any index of a list in pop() that particular element will be deleted

```
>>> l1 = [5,6,7,8]
>>> l1.pop()
8
>>> l1
[5, 6, 7]
>>> l1.pop(0)
5
>>> l1
[6, 7]
>>> l1.pop(1)
7
>>> l1
[6]
>>> |
```

The **del** keyword is used to delete an element from a list , slicing works for del keyword , we can also delete a particular element by mentioning the index number

```
>>>
>>> L1= [5, 6, 7, 8, 9]
>>> del L1[3]
>>> L1
[5, 6, 7, 9]
>>> del L1[0:2]
>>> L1
[7, 9]
>>> |
```

## remove( x )

- It search a particular element from a list and remove it if not found it gives an error
- If duplicates are present then it removes the 1st occurrence of the duplicate from the list

```
>>> L1 = [5, 6, 7, 5, 6, 7]
>>> L1.remove(6)
>>> L1
[5, 7, 5, 6, 7]
>>> L1.remove(10)
Traceback (most recent call last):
File "<pyshell#9>", line 1, in <module>
    L1.remove(10)
ValueError: list.remove(x): x not in list
>>>
```

## Clear( )

- It will clear the entire list
- Even using slicing we can clear a list
- Delete can also be used to delete a list

```
>>> L1
[5, 7, 5, 6, 7]
>>> L1.clear()
>>> L1
[]
>>> L1= [5, 6, 7, 8, 9, 10]
>>> del L1[:]
>>> L1
[]
>>>
```

## List Methods #3

- The syntax of index method is `index( x [ , start [ , end ] ] )`
- The `index()` is used to search a particular element/item in a list and when found it returns its index.
- By default index takes starting and ending value as zero if the index is not defined
- The parameter `[]` (`start` , `end`) are optional in index method
- The start and end value in index are mostly given when you want to know the index value of the duplicate element because when value is not given it takes default value as 0 and returns the index value of first occurrence of the element.

### Example :

```
>>> l1 =[5,6,7,1,2,3,6,7,9,6]
>>> l1.index(1)                                #return the index of 1 i.e, 3
      3
>>> l1.index(7)
      2
>>> l1.index(5)
      0
>>> l1.index(6,2)                            #return the index of 6 there indexing is
      6                                         starting from 2

>>>
>>> l1.index(6,2,7)                         # number to be search is 6 in range of
      6                                         index from 2 to 7
>>> |
```

- When you give an ending point in indexing they will stop one step before that value.If element is not found then we'll get an error

- The **count( x )** method is used to know how many times a particular value/element is appearing in a list . It will not give the index value it will count the given number for example .

```
>>>
>>> l1 =[5,6,7,1,2,3,6,7,9,6]
>>> l1.count(6)
3
>>> l1.count(5)
1
>>> l1.count(7)
2
>>> |
```

- When the **reverse( )** method is called it simply reverse the contents of a list.

```
>>>
>>> l1 =[5,6,7,1,2,3,6,7,9,6]
>>> l1.reverse()
>>> l1
[6, 9, 7, 6, 3, 2, 1, 7, 6, 5]
>>> |
```

- The syntax for sort method is **sort ( \* , key = none , reverse = false )**
- The **sort( )** sort the elements in the list and you'll get the elements of the list In sorted order.
- understanding the working of **sort()**

**Sort ( \* , key = none , reverse = false )**

Key = none - here we are defining our own criteria / function  
 Reverse = false - reverse is key and false is value

```
>>>
>>> l1 = ['yy','JJ','mm','BB','aa','zz']
>>> l1.sort()
>>> l1
['BB', 'JJ', 'aa', 'mm', 'yy', 'zz']      #Before applying key = none
>>>
>>> l1.sort(key = str.lower)
>>> l1
['aa', 'BB', 'JJ', 'mm', 'yy', 'zz']      #after applying key =str.lower
>>> |
```

- Upper case letter are always treated as smaller than lower case letter [ based on ASCII value it happens]
- There's a global function called **sorted()** , This function will modify the original list it gives a new sorted list.

```
>>>
>>> l1 = ['yy', 'JJ', 'mm', 'BB', 'aa', 'zz']
>>> l1.sort()                                     #Modifies the given list I.e , sort it
>>> l1
['BB', 'JJ', 'aa', 'mm', 'yy', 'zz']
>>> sorted(l1)                                    #Not modify list but create a new sorted list
['BB', 'JJ', 'aa', 'mm', 'yy', 'zz']
>>> |
```

## List Comprehension

- It is a method to create a list in a simple way from existing list (or) other iterables.
- A list can be generated from a [list , tuple , string or a set](#).
- A list can be empty, it is denoted as [ ]
- [append\(\)](#) in list adds an element to a list
- A loop can be used to append values in a list of particular range.

Example 1 :

```
>>> l1 = []
>>> for x in range(10):
...     l1.append(x)
...
...
>>> l1
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>>
```

- The above process can be done in a simple way using '[List Comprehension](#)'

Syntax of List Comprehension :

L1 = [ Expression for item in iterable ]

Example 2 :

```
>>>
>>> l1 =[x for x in range (10)]
>>> l1
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>>
```

- We can observe from this example that how using list comprehension we simplified using example 1
- Some other examples of list comprehension are



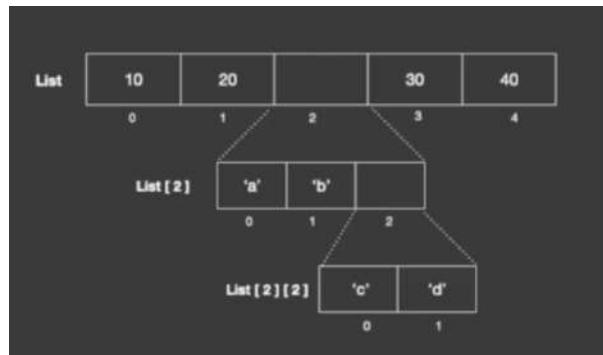


## Nested List

- A list can have heterogeneous elements like int , float etc.
- A list can have a list as an element inside it this is called nested list for example

Ex : `list = [10, 20 ['a', 'b' ['c', 'd'], 30, 40]]`

- Diagrammatically it can be represented as



- If you are having nested list you can prepare a matrix also with same type of values . Lets, see this in a program

```
A = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
B = [[9, 8, 7], [6, 5, 4], [3, 2, 1]]

C = []

for i in range(len(A)):
    S = []
    for j in range(len(A[0])):
        S.append(A[i][j] + B[i][j])
    C.append(S)

print(C)
```

# A, B are 2 list in which nested list resides  
On this 2 list we are performing addition and appending the results in C and printing the result

```
[[[10, 10, 10], [10, 10, 10], [10, 10, 10]]]
```

- Just like + you can also perform - , \* on a matrix

# Tuple Intro

## Tuple DataType

This is one more datatype in python it is similar to list . The only difference is tuple is immutable and list is mutable .means it is only readable.

Tuple is collection of elements

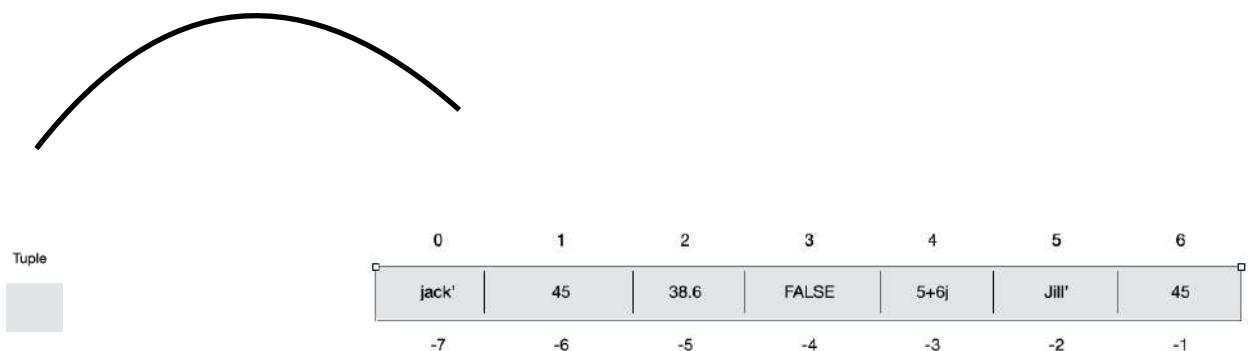
Ex: tuple 1=('jack', 45,38.6, false, 5+6j, 'Jill', 45)

Tuple can contain duplicates also.

How to create Tuple

Tuple is enclosed in round brackets

How it looks in memory?



It looks like an array. So this is an array and every array having its own index

We cannot change anything in tuple tuple does not have append method so, we cannot change the value at a particular index nor increase it.

t1=(1,2,3,4,5)#this is the method of creating a tuple

t1=(10)#single value in a tuple# not valid

t1=(10, ) # valid

t1=tuple(1,2,3,4,5)

### Program:

```
>>>
>>> t1=('Jack', 38.5, 45, 5+9j, 45)
>>> t1
('Jack', 38.5, 45, (5+9j), 45)
>>> type(t1)
<class 'tuple'>
>>>
>>> t1[2]
45
>>> t1[2]=85
Traceback (most recent call last):
File "<pyshell#6>", line 1, in <module>
t1[2]=85
TypeError: 'tuple' object does not support item assignment
>>>
>>> t1=(1,2,3,4,5)
>>> t1
(1, 2, 3, 4, 5)
>>> t2=()
>>> t2
()
>>> type(t2)
<class 'tuple'>
>>> |
```

## Packing:

t1=10,20, 30,40#giving multiple values what happens ?Let's check.

```
t1=10
t1
10
type(t1)
<class 'int'>
t1=10,20,30,40# it is called packing
type(t1)
<class 'tuple'>
```

If we give multiple values it will pack it by giving brackets this is called packing

## Unpacking:

t2=(10,20,30,40)

```
a,b,c,d=t2
t2
(10, 20, 30, 40)
a
10
b
20
c
30
d
40
```

This is called unpacking

When given values without parenthesis means packing and in contrast given values which are storing them is called unpacking are automatically used in tuple

t1=10,20,30,40

```
I1=[1,2,3]
type(I1)
<class 'list'>
a,b,c=I1
a
1
b
2
c
3
```

Unpacking is done for tuple as well as list and string. But packing happens only with tuple

## Tuple comprehensions and methods

**Comprehension:** comprehensions are short cut for creating a sequence object or tuple object.

Let's Try:

```
>>>
>>> l1= [ x for x in range(10)]
>>> l1
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> t1 = (x for x in range(10))
>>> t1
<generator object <genexpr> at 0x7fafa245fc10>
>>>
>>> t1 = tuple(x for x in range(10))
>>> t1
(0, 1, 2, 3, 4, 5, 6, 7, 8, 9)
>>> t2 = *(x for x in range(10))
SyntaxError: can't use starred expression here
>>> t2 = (*(x for x in range(10)),)
>>> t2
(0, 1, 2, 3, 4, 5, 6, 7, 8, 9)
>>>
>>> t3 =(*(x for x in range(1,10,2)),)
>>> t3
(1, 3, 5, 7, 9)
>>> t4 =(*(x for x in 'python'),)
>>> t4
('p', 'y', 't', 'h', 'o', 'n')
>>>
```

Now creating another tuple

```
>>>
>>> t5 =(*(x for x in 'PyThOn' if x.islower()),)
>>> t5
('y', 'h', 'n')
>>> t6 =tuple( x for x in 'PyThOn' if x.islower() )
>>> t6
('y', 'h', 'n')
>>> t7 =tuple( x**2 for x in (1,3,5,7,9) )
>>> t7
(1, 9, 25, 49, 81)
>>>
>>> t8 = (1,2,3,4,5,4,5,6,7)
>>> t8.count(4)
2
>>> t8.index(3)
2
>>> t8.index(10)
Traceback (most recent call last):
File "<pyshell#11>", line 1, in <module>
    t8.index(10)
ValueError: tuple.index(x): x not in tuple
>>> |
```

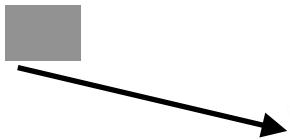
Index method gives an error when the element is not found

## Tuple Iteration and Operators

**Iteration:** traversing and visiting all the elements of a tuple .

```
t1=('jack', 45, 38.6, false, 5+6j,'Jill', 45)#this is a tuple
```

t1



0	1	2	3	4	5	6
jack'	45	38.6	FALSE	5+6j	Jill'	45
-7	-6	-5	-4	-3	-2	-1

It looks like this in memory.

Iteration can be done using for loop

Creating a tuple using for loop

```
t1=('jack', 45, 38.6, False, 5+6j,'jill', 45)
for x in t1:
    print(x)

jack
45
38.6
False
(5+6j)
jill
45
```

```
>>> for i in range(len(t1)):
    print(t1[i])

Jack
45
38.6
False
(5+6j)
Jill
45
>>>
```

This can be done using while loop also

## Operators in tuple

[]	Index
[:]	Slicing
+	Concatenation
*	Repeat
In	Membership
Not in	Membership

```
>>>
>>> t1 = ('Jack', 45, 38.6, False, 5+6j, 'Jill', 45)
>>>
>>> t1[1]
45
>>> t1[4]
(5+6j)
>>> t1[4]=25
Traceback (most recent call last):
File "<pyshell#5>", line 1, in <module>
    t1[4]=25
TypeError: 'tuple' object does not support item assignment
>>>
>>> t1[-1]
45
>>> t1[-2]
'Jill'
>>> t1[3]
('Jack', 45, 38.6, False, (5+6j), 'Jill', 45)
>>> t1[3:len(t1):-1]
(False, (5+6j), 'Jill', 45)
>>> t1[3:len(t1):-1]
()
>>> t1[-3:-len(t1):-1]
((5+6j), False, 38.6, 45)
>>> t1[:-1]
(45, 'Jill', (5+6j), False, 38.6, 45, 'Jack')
>>> t1[::-2]
('Jack', 38.6, (5+6j), 45)
>>>
>>>
>>>
>>>
>>>
>>> | ↵
...
>>> t1 =(1,2,3)
>>> t2=(4,5,6)
>>>
>>> t1+t2
(1, 2, 3, 4, 5, 6)
>>> t1
(1, 2, 3)
>>> t2
(4, 5, 6)
>>> t1 + [4,5,6]
>>> t1 + [4,5,6]
Traceback (most recent call last):
File "<pyshell#7>", line 1, in <module>
    t1 + [4,5,6]
TypeError: can only concatenate tuple (not "list") to tuple
>>>
>>> t1 + tuple([4,5,6])
(1, 2, 3, 4, 5, 6)
>>> t1 *2
(1, 2, 3, 1, 2, 3)
>>> t1 * 3
(1, 2, 3, 1, 2, 3, 1, 2, 3)
>>> temp = t1*3
>>> temp
(1, 2, 3, 1, 2, 3, 1, 2, 3)
>>> 3 in t1
True
>>> 5
```

## Set datatype

- It is a collection of elements and it is iterable  
it can contain any type of data that means it is heterogenous .
- But it is unordered . it don't have indexing. In this no duplicates are allowed.  
how to create set: by using of lower bracket

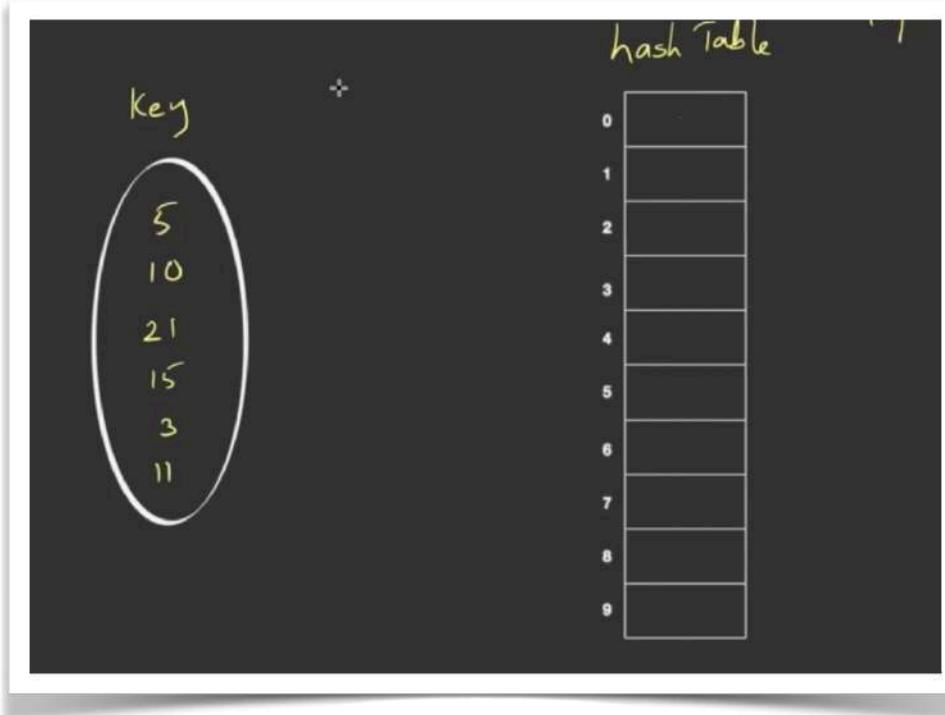
```
>>>
>>> s1 = {1,2,3,4,'jack',3,4,'jack'}
>>> s1
{1, 2, 3, 4, 'jack'}
>>> s1[0]
Traceback (most recent call last):
File "<pyshell#3>", line 1, in <module>
  s1[0]
TypeError: 'set' object is not subscriptable
>>> type(s1)
<class 'set'>
>>>
>>> s2 = set(1,2,3,4,5)
Traceback (most recent call last):
File "<pyshell#6>", line 1, in <module>
  s2 = set(1,2,3,4,5)
TypeError: set expected at most 1 argument, got 5
>>> |
```

- Indexing and slicing are not allowed cause it don't have indexing.

```
>>> s4.discard(50)
>>> s4
{20, 40, 10, 30}
>>> s4.add(100)
>>> s4
{20, 100, 40, 10, 30}
>>> s4.add(120)
>>> s4
{20, 100, 40, 10, 120, 30}
```

Set is immutable you can modify or remove but we can't replace by using index that means it is growable.

## Set internals



- Lets see how hash table works internally
- We have 5 , 10 , 21 , 15 , 3 , 11
- It uses hash function which is  $x \% 10$  ( its taking 10 because we have numbers between 0 to 10 )
- How this keys are stored in hash table lets see

we take 5 . Take it in hash function

$$h(x) = x \% 10$$

$$h(5) = 5 \% 10 = 5$$

So it is stored hash table 5

we take 10 .

$$h(10) = 10 \% 10 = 0$$

So it is stored in the place of 0

$$h(21) = 21 \% 10 = 1$$

$$h(11) = 11 \% 10 = 1$$

We got 1 for both 21 and 11 so both should be stored in 1 in hash table this is said as collision  
Then it is saved as a form of linked list



Here we are taking mode 16

$$h(10) = 10\%16=10$$

$$h(20) = 20\%16=4$$

$$h(30) = 30\%16=14$$

$$h(40) = 40\%16=8$$

$$h(50) = 50\%16=2$$

```
>>> s = {10,20,30,40,50}  
>>> s  
{50, 20, 40, 10, 30}  
>>> s.add(60)  
>>> s  
{50, 20, 40, 10, 60, 30}  
>>> s.add(70)  
>>> s  
{50, 20, 70, 40, 10, 60, 30}  
>>> |
```

$$h(60) = 10\%16=12$$

$$h(70) = 20\%16=6$$

$$h(18) = 30\%16=2$$

$$h(31) = 40\%16=15$$

# Set in Mathematics

Set

Subset

Superset

Proper subset

Proper superset

Disjoint set

$s=\{1,2,3,4,5,6,7,8,9,10\}$  # this is a set

- A set in mathematics cannot have a duplicate value

$A=\{1,2,3,5,7\}$

Comparing elements of s with A .

So it is having elements that are part of s

We say A is a subset of s

$A \subseteq s$

So A is part of s

$B=\{5, 7, 9, 10\}$

$B \subseteq s$

B is also a subset of s

- Whenever we say  $A \subseteq s$  then s is superset for A
- Whenever we say  $B \subseteq s$  then s is superset for B

$c=\{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$

All elements are present in s

$c=s$  or  $c \subseteq s$  # we can call c equals to s or c is subset of s

## Who is proper subset?

- A proper subset means it should have some values from superset not all values

$D=\{1, 2, 3, 4, 5\}$

$E=\{6, 7, 8, 9, 10\}$

## What are common among them

So, there are no elements

So, what would we call two subsets which are having no common elements.

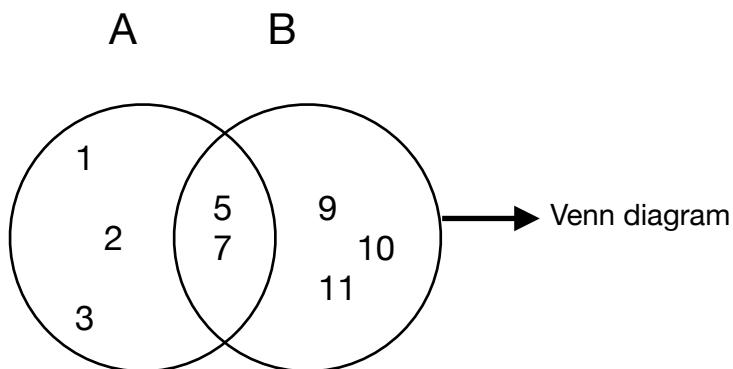
- We call them as disjoint sets.

## Set Operations in maths

- We can perform some binary operation on set
- Some of the operations are **union , intersection , difference and symmetric difference.**
- In mathematics we can perform these set operations like
- Consider 2 sets

$$A = \{ 1,2,3,5,7 \} \quad B = \{ 5,7,9,10,11 \}$$

Union :



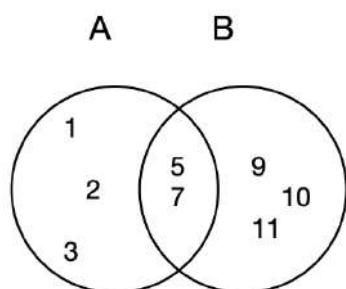
$$A \cup B = \{ 1,2,3,5,7,9,10,11 \}$$

**U** - Union symbol

- Union combines the element of set A and set B together without any duplication .
- Also ,  $A \cup B = B \cup A$

Intersection :

- Intersection means you have to take the common elements of set A and set B.



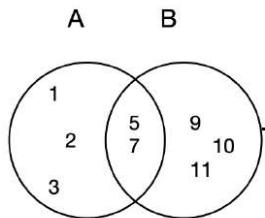
$$A \cap B = \{5, 7\}$$

**∩** - Intersection

- Also,  $A \cap B = B \cap A$

Difference :

- Difference means, Suppose you are taking  $A - B$  then take all elements of 'A' which are not present in 'B' and vice versa.



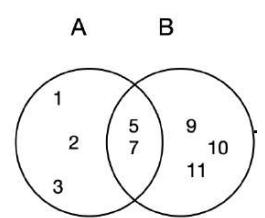
$$A - B = \{1, 2, 3\}$$

$$B - A = \{9, 10, 11\}$$

- Here,  $A - B$  Not Equal to  $B - A$

Symmetric Difference :

- Symmetric difference take the element present exclusively in sets A and set B and it doesn't take the common element.



$$A \Delta B = \{1, 2, 3, 9, 10, 11\}$$

**Δ** - symmetric difference

Here,  $A \Delta B = B \Delta A$

## Set Methods

Lets take an example

s1={1,2,3,5,7}                s2={5,7,9,10,11}

### **union(iterable)**

A.union(B)

{1,2,3,5,7,9,10,11}

- It will generate a new set without duplicate

### **intersection(iterable)**

A.intersection(B)

{5,7}

- It will print repeated one

### **difference(iterable)**

A.difference(B)

{1,2,3}

- It will print only A which is not presented in B

### **Symmetric\_difference(iterable)**

A.symmetric\_difference(B)

{1,2,3,9,10,11}

- It will not take the number present in both A and B

### **intersection\_update(iterable)**

- In intersection it will generate a new set but in difference update will update the list then A will become {5,7}
- intersection(iterable )will generate a new set
- update(iterable) will update in same string

## Set Operators

- The operators are given in the right hand side table
- Here we are taking the following sets and performing different set operators.

$s = \{ 1,2,3,4,5,6,7,8,9,10 \}$

$A = \{ 1,2,3,5,7 \} \quad B = \{ 5,7,9,10,11 \}$

- **Union** means combining both sets and removing duplicates

$C = A | B \quad \Rightarrow \quad A | B = \{ 1,2,3,5,7,9,10,11 \}$

**C** - Taking result of A , B in var C

**|** - pipe / union symbol

- **Intersection** means giving common elements

$C = A & B \quad \Rightarrow \quad A & B = \{ 5,7 \}$

**C** - Taking result of A , B in var C

**&** - Intersection symbol

- **Difference** of A - B means take all elements of A except those common in B

$C = A - B \quad \Rightarrow \quad A - B = \{ 1,2,3 \}$

- **Symmetric Difference** means take all elements of A and B except those common to both sets

$C = A ^ B \quad \Rightarrow \quad A ^ B = \{ 1,2,3,9,10,11 \}$

**C** - Taking result of A , B in var C

**^** - Symmetric difference

&
-
^
<
≤
>
≥
==
!=
in
not in

```

>>>
>>> A={1,2,3,5,7}
>>> B={5,7,9,10,11}
>>> A|B           #union
{1, 2, 3, 5, 7, 9, 10, 11}
>>> A&B          #intersection
{5, 7}
>>> A-B          #difference
{1, 2, 3}
>>> A^B          #symmetric difference
{1, 2, 3, 9, 10, 11}
>>>

```

- Now let us check other operators.
- `<, >` are useful for checking proper subset (or) proper superset
- `<=, >=` they consider equal sets as well while checking subset and superset
- `==` they check if two sets are equal or not
- `!=` they check if two sets are not equal

```

>>>
>>> S = {1,2,3,4,5,6,7,8,9,10}
>>> A={1,2,3,5,7}
>>> B={5,7,9,10,11}
>>>
>>> A<B
False
>>> B<A
False
>>> A<S
True
>>> S>B
False
>>> S==S
True
>>> S!=A
True
>>> S<=A
False
>>> S>=B
False
>>>

```

#11 Is in B but not in S so that's why false

- `in, not in` are also known as membership operators
- They check if an element is present in a set or not and return boolean type as result.

```

>>>
>>> S = {1,2,3,4,5,6,7,8,9,10}
>>> A={1,2,3,5,7}
>>> B={5,7,9,10,11}
>>>
>>> 8 in S
True
>>> 11 in S
False
>>> 11 in B
True
>>>
>>> 11 not in S
True

```

**Note :**

- If you store the result of  $A | B$  in A, then the result will be go in A itself. So, You can write this as

$A | B$  ( same as  $A = A | B$  )

$A \& B$  ( same as  $A = A \& B$  )

$A - B$  ( same as  $A = A - B$  )

## Set Methods #1

- Methods are the member function of a class
  - Some of the built in methods of sets are **add()** , **copy()** ,**update ()** , **pop()** , **discard()** , **remove()** , **clear()**.
  - In a set if you want to add element you can do this with **add()** method.
  - As sets are immutable we can add more values to a set. It will modify the same set , **add()** only add one value not multiple value at a time.
  - You can add any type of values in a set ( like str, int etc)
- 
- The **copy()** will give the same copy of a given set . It is also called cloning of a set.
  - If you want to add multiple values to a set then use **update( iterable )** method.
  - In **update()** it 1st checks the if value is available or not in the set if yes it doesn't add that value/ no duplicate and vice versa.

```
>>> s = {1,2,3,4}                                     #adding an element to a set
>>> s.add(5)
>>> s
{1, 2, 3, 4, 5}
>>> s.add(6,7)
Traceback (most recent call last):
  File "<pyshell#5>", line 1, in <module>
    s.add(6,7)
TypeError: set.add() takes exactly one argument (2 given)
>>>
>>> s.copy()                                         #making clone of the original set
{1, 2, 3, 4, 5}
>>>
>>> s.update({6,7})                                 #when you want to add more than one
s                                         #element use update
{1, 2, 3, 4, 5, 6, 7}
```

- **Pop()** will remove 1 element in a set, which element we don't know but it'll remove.

```
>>> s1 = {10,20,30,40,50,60}
>>> s1.pop()
50
>>> s1.pop()
20
>>>
>>> s1.pop(40)
Traceback (most recent call last):
  File "<pyshell#7>", line 1, in <module>
    s1.pop(40)
TypeError: set.pop() takes no arguments (1 given)
>>>
```

#**pop()** will not take argument , it will given an error if argument given.

- In **discard( x )** you have to mention the element which you want to remove.
- There is no index available in set so you have to mention the element to be discarded.
- The **remove( x )** method , removes the element from the set, work same as **discard( x )** the only difference is discussed below in the example.

Example :

```
>>>
>>> s1 = {10,20,30,40,50,60}
>>> s1.discard(70)
>>> s1
{50, 20, 40, 10, 60, 30}
>>>
>>> s1 = {10,20,30,40,50,60}
>>> s1.remove(70)
Traceback (most recent call last):
  File "<pyshell#6>", line 1, in <module>
    s1.remove(70)
KeyError: 70
```

# discard ignore if no element is found to be discarded .

#Remove gives an error when no element is found to be removed.

- The **clear()** method , will clear all contents of set, it will make it as an empty set.

```
>>>
>>> s1 = {10,20,30,40,50,60}
>>> s1.clear()
>>> s1
set()
>>> |
```

## Set comprehensions

```
S = { expression for item in iterable }
```

# set

```
S = set()
```

# method of creating an empty set

```
S1 = { x for x in range(10) }
```

# it will take values from 0 to 9

```
S2 = { x**2 for x in [-2, -1, 0, 1, 2] }
```

# it gives the square of the elements

```
S3 = { x for x in (10, 5, 7, 8, 12, 3) if x%2==0 }
```

# it contains a condition

```
S4 = { x.upper() for x in 'philippines' }
```

Unhashable type set

### IDLE:

```
s={}
type(s)
<class 'dict'>
s=set()
type(s)
<class 'set'>
for x in range(1, 11):
    s.add(x)
```

```
s
{1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
```

Same thing can be done using set comprehensions

```
s1={x for x in range(1, 11)}
s1
{1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
```

```
s2={x**2 for x in [-2, -1, 0, 1, 2]}      # it is giving the square of the element
s2
{0, 1, 4}
```

$(-2)^2 = 4$

$(-1)^2 = 1$

$0^2 = 0$

$1^2 = 1$

$2^2 = 4$

So, if we notice 4 repeating two times so, it is just taking single element.

```
s3={x for x in (10,5,7,8,12,3)}  
if(x%2==0)
```

# this is having a condition also. It means just pickup even number from the tuple.

```
s4={x.upper() for x in 'phillipines'}
```

# unhashable type set is not allowed because mutable are not hashable.

A set have all mutable types and also list cannot be a member of set

## Introduction to Dictionary

- Dictionary is a collection of **key - value pair**
- It works similar to real life dictionary which contains word and their meaning
- Searching is done based on dictionary Keys
- Dictionary is created as

```
dict = { 'fruit' : 'apple', 'vegetable' : 'carrot' , 'dish' : 'salad'}
```

fruit , vegetable and dish - **keys**  
apple , carrot and salad - **values**

- For values you can take any Datatype
- But for **Keys** you can take **only immutable Datatype** ( i.e ,excluding set and list datatype )
- We can perform the following on Dictionary I.e; Access , Insert , Update and Delete

```
>>>
>>> dict2 = { 101 : 'John' , 102 : 'Smith' , 103 : 'Mark' , 104 : 'David'}
>>>
>>> dict2[102]
'Smith'
>>> dict2[0]
Traceback (most recent call last):
File "<pyshell#4>", line 1, in <module>
    dict2[0]
KeyError: 0
# No key with 0 value so we got error
>>> dict2[103]
'Mark'
>>> dict2[103]='Mathew'
>>> dict2
{101: 'John', 102: 'Smith', 103: 'Mathew', 104: 'David'}
>>>
>>> dict2[105]='Ajay'
>>> dict2
{101: 'John', 102: 'Smith', 103: 'Mathew', 104: 'David', 105: 'Ajay'}
>>> del dict2[104]
>>> dict2
{101: 'John', 102: 'Smith', 103: 'Mathew', 105: 'Ajay'}
>>> | #Updating dictionary
#inserting in dictionary
#deleting an item using key
```

## Access

- For accessing any value use key inside [ ]

## Update

- It updates the existing value in a dictionary using respectable key

## Insert

- New keys and pair are inserted using insert in a dictionary

## del

- To delete a particular value write del keyword , dict name then key inside [ ]
- You can also delete the complete dictionary using del keyword
- You can use **for loop** for traversing through a dictionary then you'll get keys as output

```
>>> dict2 = { 101 : 'John' , 102 : 'Smith' , 103 : 'Mark' , 104 : 'David' }  
>>>  
>>> for i in dict2:  
    print(i)                                #only keys as output
```

```
101  
102  
103  
104
```

- Suppose you want to print both key value using for then do this

```
>>> for i in dict2:  
    print(i, dict2[i])  
#Key - value as output
```

```
101 John  
102 Smith  
103 Mark  
104 David  
>>> |
```

## Dictionary Comprehensions

dict1 = dict()

- Its an empty dictionary

dict5 = {}

- . Its an empty dictionary

dict2 = dict ((iterable pairs))

dict2 = dict(((1,2),(2,4),(3,6)))

This are the couple of two elements. This will all form of the dictionary  
{1:2,2:4,3:6}

dict3= dict(zip(iterable , iterable))

- If you have two iterable's ( it can be list, tuple or string)
- You can join two iterable so for this zip is used

I1={'A','B','C'} I2=['apple','ball','cat']. d={'A':'apple','B':'ball','C':'cat'}

- If the iterable is having extra value then it will be ignored

## Dictionary Comprehensions

dict4=dict(enumerate (iterable))

- It will give indexes to the elements but in dict

dict4=dict(enumerate (l1))

It will take key and value

dict6={exp:exp for item in iterable}

dict7={(exp,exp) for item in iterable}

- Both are same but in dict7 it will act as a tuple

dict8={x:y for x,y in zip(iterable,iterable)}

- It is same as dict3 but in this we use for loop

dict9={item for item in enumerate(iterable)}

- It is same as dict4 but we use for in this

## Loops in Dictionary

- Looping through a dictionary is possible in python it gives output in the form of keys

```
>>>
>>> dict1 ={ 101 : 'Production' , 102 : 'Accounts' , 103 : 'Sales & Marketing' , 104 : 'Inventory' }
>>>
>>> for x in dict1:
    print(x)

101
102
103
104
```

- If you want the key-value as output then we can loop like this

```
>>> for x in dict1:
    print(x, dict1[x]) 

101 Production
102 Accounts
103 Sales & Marketing
104 Inventory
>>> |
```

- To an index when you give a key it gives output in key - value
- We can also use get( x ) method to get the result

```
>>> for x in dict1:
    print(x, dict1.get(x) )
```

```
101 Production
102 Accounts
103 Sales & Marketing
104 Inventory
>>>
```

- The difference between `dict[x]` and `get[x]` is when there is no key present in dictionary but you still call it then `dict[x]` will give an error while `get[x]` will not give an error

```
>>>
>>> print(dict1[102])
Accounts
>>> print(dict1.get(102))
Accounts
>>> print(dict1[106])
Traceback (most recent call last):
  File "<pyshell#13>", line 1, in <module>
    print(dict1[106])
KeyError: 106
>>> print(dict1.get(106))
None
>>> print(dict1.get(106,'Unkown Dept'))
)
Unkown Dept
>>>
```

- Some other methods on dictionary which can be used in looping are `keys()`, `values()` and `items()`

```
>>> dict1 = { 101 : 'Production' , 102 : 'Accounts' , 103 : 'Sales & Marketing' , 104 : 'Inventory' }
>>>
>>> dict1.keys()
dict_keys([101, 102, 103, 104])
>>> dict1.values()
dict_values(['Production', 'Accounts', 'Sales & Marketing', 'Inventory'])
>>> dict1.items()
dict_items([(101, 'Production'), (102, 'Accounts'), (103, 'Sales & Marketing'), (104, 'Inventory')])
>>>
>>> for k in dict1.keys():
    print(k, dict1[k])

101 Production
102 Accounts
103 Sales & Marketing
104 Inventory
>>>
>>> for v in dict1.values():
    print(v)

Production
Accounts
Sales & Marketing
Inventory
>>>
```

```
>>> for x,y in dict1.items():
    print(x,y)
```

101 Production  
102 Accounts  
103 Sales & Marketing  
104 Inventory

>>> |

---

## Dictionary Methods

### copy( )

- It will create a shallow copy of a dictionary
- It will not create a new object it will be referring to new object

```
>>>
>>> dict1 = { 101 : 'Production' , 102 : 'Accounts' , 103 : 'Sales & Marketing' , 104 : 'Inventory' }
>>>
>>> dict2=dict1.copy()
>>>
>>> dict2
{101: 'Production', 102: 'Accounts', 103: 'Sales & Marketing', 104: 'Inventory'}
>>> dict2[102]='Designing'
>>>
>>> dict1
{101: 'Production', 102: 'Accounts', 103: 'Sales & Marketing', 104: 'Inventory'}
>>> dict2
{101: 'Production', 102: 'Designing', 103: 'Sales & Marketing', 104: 'Inventory'}
>>> id(dict1[101])
140284086280240
>>> id(dict2[101])
140284086280240
>>>
```

- Copy will give copy of dict1 in dict2 now we can modify , add , update , del etc can be performed

### Update( iterable )

- If you want to add more key-value pairs you can call update( )

```
>>>
>>> dict1 = { 101 : 'Production' , 102 : 'Accounts' , 103 : 'Sales & Marketing' , 104 : 'Inventory' }
>>> dict2 = {105 : 'Designing', 106 : 'Packaging'}
>>>
>>> dict1.update(dict2)
>>>
>>> dict1
{101: 'Production', 102: 'Accounts', 103: 'Sales & Marketing', 104: 'Inventory', 105: 'Designing', 106: 'Packaging'}
>>>
>>> |
```

## Setdefault( key , default )

- This method works like get() , if the key is not found then it'll insert a key and its value will be none , if value is given it'll give key - value as output.

```
>>>
>>> dict1 = { 101 : 'Production' , 102 : 'Accounts' , 103 : 'Sales & Marketing' , 104 : 'Inventory' }
>>> dict2 = { 105 : 'Designing' , 106 : 'Packaging' }
>>>
>>> dict1.update(dict2)
>>> dict1
{101: 'Production', 102: 'Accounts', 103: 'Sales & Marketing', 104: 'Inventory', 105: 'Designing', 106: 'Packaging'}
>>>
>>> dict1.get(102)
'Accounts'
>>> dict1.get(110)
>>> dict1
{101: 'Production', 102: 'Accounts', 103: 'Sales & Marketing', 104: 'Inventory', 105: 'Designing', 106: 'Packaging'}
>>>
>>> dict1.setdefault(102)
'Accounts'
>>> dict1.setdefault(110)
>>> dict1
{101: 'Production', 102: 'Accounts', 103: 'Sales & Marketing', 104: 'Inventory', 105: 'Designing', 106: 'Packaging', 110: None}
>>> dict1.setdefault(111,'Adv')
'Adv'
>>> dict1
{101: 'Production', 102: 'Accounts', 103: 'Sales & Marketing', 104: 'Inventory', 105: 'Designing', 106: 'Packaging', 110: None, 111: 'Adv'}
>>>
```

## formkeys( sequence , value )

- You can insert keys from other sequence like list , tuple , set and insert them as key value , it takes only one value and all the keys will have the same value
- Keys are important here

```
>>>
>>> L1 = [11, 22, 33, 44]
>>> dict3={}
>>> dict3.fromkeys(L1)
{11: None, 22: None, 33: None, 44: None}
>>> dict3.fromkeys(L1,100)
{11: 100, 22: 100, 33: 100, 44: 100}
>>> dict3.fromkeys(L1,'hello')
{11: 'hello', 22: 'hello', 33: 'hello', 44: 'hello'}
>>>
```

- For removing elements from dictionary following methods are used

### **Pop(key , default )**

- It will remove the key and value from dictionary and the key should be mentioned
- If key is not present I'll give an error

```
>>>
>>> dict1 = { 101 : 'Production' , 102 : 'Accounts' , 103 : 'Sales & Marketing' , 104 : 'Inventory' }

>>> dict1.pop(104)
'Inventory'
>>> dict1
{101: 'Production', 102: 'Accounts', 103: 'Sales & Marketing'}
>>> dict1.pop(110)
Traceback (most recent call last):
File "<pyshell#4>", line 1, in <module>
    dict1.pop(110)
KeyError: 110
```

### **Pop ()**

- I'll pop the last key value from the dictionary

### **Clear()**

- I'll clear the dictionary

### **del**

- If you want to delete the entire Dictionary use del keyword

```
>>> dict1
{101: 'Production', 102: 'Accounts', 103: 'Sales & Marketing', 110: 'Adv'}
>>>
>>> dict1.popitem()
(110, 'Adv')
>>> dict1.clear()
>>> dict1
{}
>>> del dict1
>>> dict1
Traceback (most recent call last):
File "<pyshell#14>", line 1, in <module>
    dict1
NameError: name 'dict1' is not defined
>>> |
```

## What are Functions ?

- A function is a piece of code which performs a specific task
  - It is used to reduce the repetition of the code hence reducing the size of the code, Less chances of making mistakes in the code
  - Functions make the task easy for the programmers as each programmer in a team can be given a specific task( function ) and at the end we can combine these task together to make single application.
- 
- **Some points to remember are :**
    - Program is a general term
    - A very big program is called application
    - An application broke into pieces are called modules that performs a specific task
    - Modules broken into pieces are called Functions / Procedure

### Advantages :

- Easy development
- Error free
- Can be developed by a team of programmers
- Reuse modules for function in other projects

### Creating a Function :

Syntax : def Name ( <parameter List >)

.....  
.....  
.....

return result

- A function must return a result and in python it returns result
- If you don't write return inside a function then it'll automatically returns **None**

## How to write a function

- We can write a function by using the keyword **def** followed by a function name.
- The function name is user define and it is suggested to take a meaningful name while defining a function
- The rules for defining a function is same as giving variable names
- Within the () you can pass parameters to a function these Parameter are called "**Formal parameter**"
- Parameters are called input to a function , a function can take multiple Parameter.
- Parameters can be of any datatype
- Returning values of the statements is Calle "**output**"
- You can call a function by using the function name and pass parameter in () , these Parameter are called "**Actual parameter**"
- The actual parameter values are copied into formal parameter which acts as input to a function they are copied in the same position / order
- When you call a functioning a result is returned you should place that result into another variable (or) print it directly
- If you don't write return in function it'll return **NONE**
- So, every function returns whether you write it or not.

Syntax :

```
def fun_name ( par1 , par2, par3 ) :      #Formal parameter
    Stat1
    Stat2
    .....
    .....
    return result      #Returning result

fun_name ( par1, par2, par3 )
return_value = fun_name ( apar1, apar2, apar3 )      #Calling a function
```

A simple example to understand function.

```
def add3(a, b, c): # defining a function
    r = a + b + c # writing statement inside function
    return r        # returning result
print(add3(10, 15, 5)) # printing the result
r = add3(1, 3, 5)
print(r) # printing and taking the result in r
```

Output :

30

9

- The values in formal parameter acts as a pointer to actual parameter
- therefore they'll be referring to the same thing
- Lets understand this with an

```
def add3(a, b, c):  
    print('inside function', id(a), id(b), id(c)) # printing id of 3 var  
    # i.e., a,b,c  
  
x, y, z = 10, 15, 5 # declaring 3 var i.e., x, y, z  
print('outside function', id(x), id(y), id(z)) # printing id of x,y,z  
print(add3(x, y, z)) # calling function  
  
# outside function gives id of actual parameter  
# Inside function gives id of formal parameter
```

Output:

```
outside function 4446765584 4446765744 4446765424  
inside function 4446765584 4446765744 4446765424  
None
```

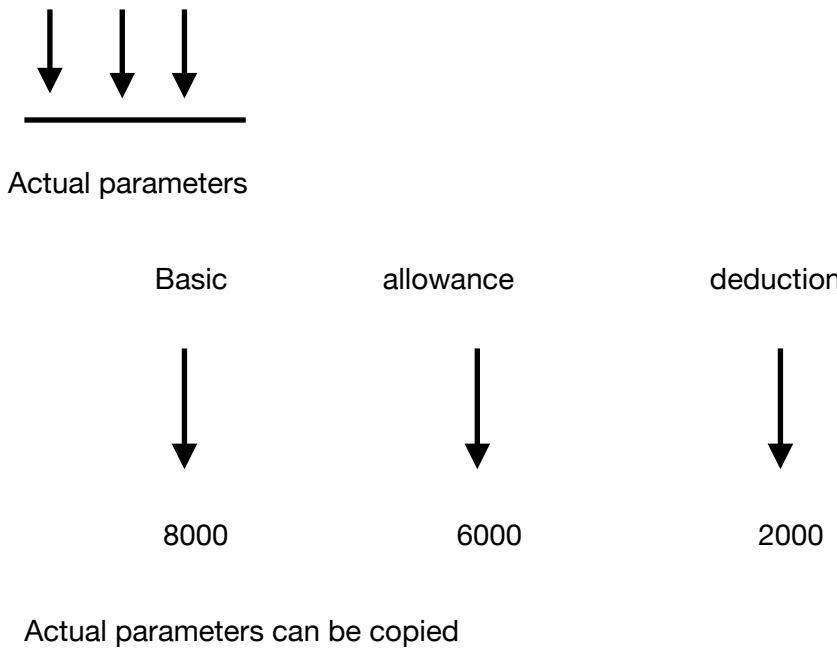
- In python object are always pass just like reference only , copy of an object will never be pass

## Positional vs keyword Arguments

```
Def net_sal(basic, allowance, deduction): # for calculating net sal of an employee.
```

```
net=basic +allowance_deduction  
Return net
```

```
n= net_sal(8000, 6000,2000)# way to call function for net sal
```



```
Def net_sal(basic, allowance, deduction): #function for net sal
```

```
net= basic+allowance-deduction
```

```
Return net
```

```
n=net_sal(basic, allowance, deduction)
```

```
print('Net salary is:', n)
```

To know which variable has print which value.

```
def net_sal(basic, allowance, deduction): #function for net sal  
    net = basic + allowance - deduction  
    return net  
  
n = net_sal(8000, 6000, 2000)  
  
print('net salary is:', n)
```

Output:

Net salary is : 12000

# so this is called as positional arguments

Program:

```
def net_sal(basic, allowance, deduction):
    print('basic', basic)
    print('allowance', allowance)
    print('deduction', deduction)
    net = basic + allowance - deduction
    return net

n = net_sal(deduction=2000, allowance=6000, basic=8000)
print('Net Salary is :', n)
```

By writing the names of parameters also we can call the function.

These are called as keyword argument. Without that it is a positional argument

```
def net_sal(basic, allowance, deduction):
    print('basic', basic)
    print('allowance', allowance)
    print('deduction', deduction)
    net = basic + allowance - deduction
    return net

n = net_sal(8000, deduction=2000, allowance=6000)
print('Net Salary is :', n)
```

# both positional and keyword arguement



## Default Argument

We can make argument default or optional

```
def add ( a,b,c ):  
    return a+b+c  
  
print( add(10,5,2)) ✓  
print( add(10,5) ✗  
print( add(10)) ✗
```

Can we pass different number of parameter ???? Yes we can pass

```
def add( a , b=0 , c=0):  
    return a+b+c  
  
add ( 5,7)
```

It will take a as 5 and b=7 and c as 0

- Here two cases are default

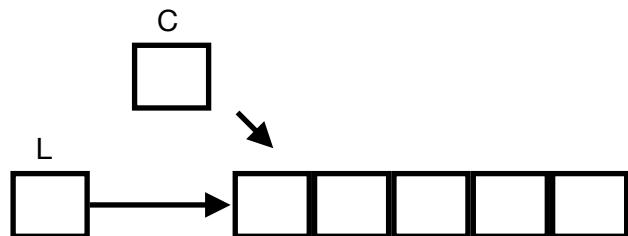
```
add(a=10,b=5,c=2) ✓  
add(b=5,c=2,10) ✗
```

## Default are created only once

```
def addition ( item,L =[ ])
```

```
    L.append(item)
```

```
    return L
```



# Mixed positional keyword

If we want a function to take only positional arguments or only keyword can enforce and make it mandatory.

Let's see how some arguments can have positional only or keywords only.

Def add(a, b, c, d, e, f):# taking 6 arguments

return a+ b+ c+ d+ e+ f # adding and returning result

if we have integer type of all so, it will add all and return the result of all

add(2, 5, 9, 7, 3, 8)and these arguments will be copied to the corresponding arguments.

In the same position to this formal parameter get copied in actual parameters.

2 goes in a

5 goes in b

.....

Add(f=8, c=9, b=5, c=3, d=7, a=2)

So we can use keyword argument also

Here 8 goes to f

.....

Def add(a,b,\ c, d, e, f):

Return a+b+c+d+e+f

Taking 6 arguments adding and returning result but here the slash means we can only give positional arguments before slash and after we can give both positional and keyword arguments both are allowed.

Add(2,5,6,7,9,4) # all positional

Add(2, 5, d=7, f=4, e=9, c=6)

This two must be positional

Where keyword passed order must not be important.

```
def add(a, b, c, d, e, f):
    return a+b+c+d+e+f
r=add(2, 5, 7, 4, 8, 9)# updating this to check whether it is working or not
```

```
def add(a, b, / ,c, d, e, f):  
    print(a, b, c, d, e, f):  
        return a+b+c+d+e+f  
r=add(f=9,e=8, a=2, c=7, b=5, d=4)  
print(r)
```

if given slash it will give an error as we have given a and b as positional argument.

Def add(a, b, / , c, d, \* ,e, f):  
Positional      ↓      any      ↓ keyword

Return a+b+c+d+e+f

Add(2,5,d=8, c=4, 9, 3)

By writing / we say that only positional arguments

By writing \* at the beginning of the statement we say keywords are mandatory

### Program

```
def add(a, b, /, c, d, *, e, f):  
    print(a, b, c, d, e, f)  
    return a+b+c+d+e+f  
r = add(2, 5, 6, 7, f=9, e=8)  
print(r)
```

## Variable Length Arguments

- The number of arguments a function takes is pre defined
- When default arguments are given we have options to pass parameter
- In python we can take as many parameters we want in a function by using variable length arguments , this is taken in the form of a tuple but the parameter name should start with \*

syntax : def fun ( \* args ):  
          Print ( args )

```
def fun1(*args):
    print(type(args), args)

fun1()
fun1(10, 20)
fun1(10, 20, 30, 40, 50, 60, 70, 80, 90, 100)
fun1(10, 'hello', 27.5, True)
```

Output :

```
:class 'tuple'> ()
:class 'tuple'> (10, 20)
:class 'tuple'> (10, 20, 30, 40, 50, 60, 70, 80, 90, 100)
:class 'tuple'> (10, 'hello', 27.5, True)
```

- If there are more parameter given before \*args then it is compulsory to give its value during function call for \*args its optional.

```
def fun1(a, b, *args):
    print(a, b, args)

fun1()
fun1(11, 22)
fun1(11, 22, 33, 44, 55, 66)
```

Output :

```
TypeError: fun1() missing 2 required positional arguments: 'a' and 'b'
```

- If you have a sequence like list, tuple, string then you can unpack that in a function call using \*

```
def fun2(a, b, c):
    print(a, b, c)

s1 = 'sky'
fun2(*s1)
```

Output:

```
s k y
```

- If a function is returning multiple values it is possible in python then all values can be unpacked in different variable or take them in a tuple

```
def fun3(a, b, c):
    return a+1, b+1, c+1

x, y, z = fun3(10, 20, 30)
print(x, y, z)
```

Output :

```
11 21 31
```

## keyword variable length Argument

### Variable length arguments

```
def fun1(*args):  
    print(args )  
    fun(10,8,12.9,’John’,16)
```

- All this values will be stored as a tuple in \*args . So you can pass as many arguments and that argument as a tuple

```
    def fun1(a,b,c)
```

Whenever we call the function we can call by positional argument or keyword arguments

```
fun1(10,20,15)  
fun1(a =10,b=20,c=15)
```

### Keyword variable length argument

```
def fun2(**kwargs):  
    print (kwargs)  
fun(name = ‘Ajay’ ,roll = 10,addr = ‘delhi’)
```

- Python allows functions to be called using keyword arguments. You can pass any argument but it should be keyword form argument
- keyword arguments must follow positional arguments.

### Mixed arguments

```
def fun3(a,b,*args , **kwargs) :  
    print (a,b,args,kwargs)
```

```
def fun3(a,b,*args , **kwargs) :  
    a , b should be positional arguments
```

```
def fun3(a,b,*args , c , **kwargs) :  
    a , b should be positional arguments and c should be keywor
```

## Iterators and Generators

- Iterators allow us to iterate through a sequence of element i.e, visiting each element once in a sequence
- We can pass any sequence to the iterator like `list` , `string` , `tuple` , `set` and `dictionary`
- In python there is a built in function called `iter` for iteration

```
L = [1, 2, 3, 4, 5]

itr = iter(L)

print(next(itr))
print(next(itr))
print(next(itr))
print(next(itr))
print(next(itr))
print(next(itr))
```

#`next(itr)` give next element in sequence

```
1
2
3
4
5
```

- We can write our own iterators which are called “**GENERATORS**” they work just like iterators

- Lets see this with an example which displays days of a week and once the sequence is completed and called again it should return the first value and so on...

```
def Days():
    L = ['Sun', 'Mon', 'Tue', 'Wed', 'Thru', 'Fri', 'Sat']
    i = 0

    while True:
        x = L[i]
        i = (i + 1) % 7
        yield x

d=Days()
print(next(d))
print(next(d))
print(next(d))
print(next(d))
print(next(d))
print(next(d))
print(next(d))
print(next(d))
```

# For a loop to remain in same state we use **yield** it'll not stop the function but keep function on hold and return value

```
Sun
Mon
Tue
Wed
Thru
Fri
Sat
```

## Local vs global variable

```
g=10(gv)
Def fun1(a, b)
c=a+b
print('local', c)
print('global',g)
fun1(4,8)
```

- Access the g value
- Variables of the python file are the global variables
- The variables declared outside the function is global variable  
Def fun1(a, b):  
c=a+b  
print(c)  
print(g)  
fun1(4,8)
- Global variable can be declared anywhere inside any function
- Global variable cannot be modified in any function for modification

### The Local Variable

- When we declare a variable inside a function it becomes a local variable.
- A local variable is variable whose scope is limited to only that function and where it is created>
- That means the local variable value is available only in that function where it is created.

### The Global Variable

- Sometimes , the global variable and the local variable has the same name . In that case, the function, by default, refers to the local variable and ignores the global variable.
- So, the global variable is not accessible.

## Recursive Function

- A function calling itself is called recursive function
- It is same as the recursive function in mathematics
- A factorial is defined in terms of factorial only ,therefore it is called recursive function

Ex :  $n! = 1 * 2 * 3 * 4 * 5 \dots * n$

$$5! = 5 * 4 * 3 * 2 * 1$$

$$5! = 5 * 4!$$

$$n! = n * (n - 1)$$

- In maths recursive function is also define as

$$\text{fact}(n) = \begin{cases} 1 & \text{if } n=0 \\ n * \text{fact}(n-1) & \text{if } n>0 \end{cases}$$

- In python we can write the same thing as

Ex:

```
def fact ( n ):  
    If n == 0 :  
        return 1  
  
    else :  
        Return n * fact ( n-1 )
```

```
def fact(n): # taking I/P that should be integer  
    if n == 0: # if given n is equal to 0  
        return 1  
    else:  
        return n * fact(n-1) # return factorial of that number  
r = fact(5) # calling and storing the function in a temporary variable  
print(r) # printing the result
```

Output :

120

## Build-in Function #1

- They are many Build-in Functions / global functions available in python
- The Build-in Functions are

Build - in Functions			
abs()	filter()	len()	set()
ascii()	float()	list()	slice()
bin()	format()	locals()	sorted()
bool()	frozenset()	map()	str()
bytearray()	globals()	max()	sum()
bytes()		min()	super()
callable()		next()	tuple()
chr()	hasattr()	object()	type()
complex()	hash()	oct()	
	help()	open()	
	hex()	ord()	zip()
dict()		pow()	
dir()	id()	print()	
divmod()	input()		
enumerate()	int()	range()	
eval()	isinstance()	reversed()	
exec()	issubclass()	round()	
	iter()		

- Lets see the working of these functions
- abs() , ascii()

```
>>>
>>> a = -15
>>> abs(a)
15
>>> b = -17.86
>>>
>>> abs(b)
17.86
>>>
>>> abs(3+4j)
5.0
>>> ascii('A')
'"A"'
>>> ascii(10)
'10'
>>> letter = '\u0521'
>>> letter
'\u0521'
>>> ascii(letter)
"\u0521"
>>>
```

- **bin()** it takes number and convert it into binary form
- **bool()** convert anything into bool type
- **bytearray()** and **bytes()** they both are similar the difference is byte array is mutable where as bytes is immutable

```
>>>
>>> ba = bytearray(5)
>>> ba
bytearray(b'\x00\x00\x00\x00\x00')
>>> s1 = 'abcde'
>>> ba = bytearray(s1.encode())
>>> ba
bytearray(b'abcde')
>>>
>>> for i in ba:
    print(i)
```

```
97
98
99
100
101
>>> ba.append(102)
>>> ba
bytearray(b'abcdef')
>>> b = bytes(s1.encode())
>>> b
b'abcde'
>>> |
```

- `callable()` we can know if the given identifier is a function or not
- `chr()` gives you the character for any given ascii code
- `complex()` used for creating complex datatype

```
>>> def add(a,b):
        return a+b

>>> s1 = 'abcd'
>>>
>>> n = 10
>>>
>>> callable(n)
False
>>> callable(s1)
False
>>> callable(add)
True
>>>
>>> chr(65)
'A'
>>> ord('A')
65
>>>
```

- `dict()` used for creating a Dictionary
- `dir()` give details of particular class
- `divmod()` takes 2 parameters and gives division as well as modulus as result

```
>>>
>>> divmod(11,3)
(3, 2)
>>>
>>> q , r = divmod(13,4)
>>> q
3
>>> r
1
>>> divmod(14.3,3.2)
(4.0, 1.5)
>>> |
```

- `enumerate()` gives indexing for all items in given sequence

```
>>>
>>> L = ['A', 'B', 'C', 'D', 'E']
>>> e = enumerate(L)
>>>
>>> e
<enumerate object at 0x7f9e7bc3ed80>
>>>
>>> for i in e:
    print(i)

|
(0, 'A')
(1, 'B')
(2, 'C')
(3, 'D')
(4, 'E')
>>>
```

- `eval()` evaluates an expression
- `exec()` execute python statements

```
>>>
>>> eval('3 * 10 + 15 / 3')
35.0
>>> eval('2 ** 4 + 9')
25
>>>
>>> s ='x=10\ny=20\nprint(x+y)'
>>>
>>> exec(s)
30
>>> x
10
>>> |
```

## Build-in Functions #2

- **filter( )** - filter objects from any iterable based on function you give

```
>>>
>>> L = [3, 6, 7, 9, 12, 14, 19, 21]
>>>
>>> def even(x):
...     if x % 2 == 0:
...         return True
...     else:
...         return False

>>> filter(even, L)
<filter object at 0x7f8ce252e8e0>
>>> f = filter(even, L)
>>>
>>> for i in f:
...     print(i)
```

6  
12  
14

- **float( )** - Convert datatype into float type
- **format( )** - same as string formatting
- **frozenset( )** - Takes any literal and convert it into immutable set
- **globals( )** - gives all the global variables declared inside a python program

```
>>> f = 12.54634
>>> format(f, 'E')
'1.254634E+01'
>>>
>>> L = [1,2,3,4,5]
>>> fz = frozenset(L)
>>>
>>> fz
frozenset({1, 2, 3, 4, 5})
>>> globals()
{'__name__': '__main__', '__doc__': None, '__package__': None, '__loader__': <class '_frozen_importlib.BuiltinImporter'>, '__spec__': None, '__annotations__': {}, '__builtins__': <module 'builtins' (built-in)>, 'f': 12.54634, 'L': [1, 2, 3, 4, 5], 'fz': frozenset({1, 2, 3, 4, 5})}
>>>
>>> locals()
{'__name__': '__main__', '__doc__': None, '__package__': None, '__loader__': <class '_frozen_importlib.BuiltinImporter'>, '__spec__': None, '__annotations__': {}, '__builtins__': <module 'builtins' (built-in)>, 'f': 12.54634, 'L': [1, 2, 3, 4, 5], 'fz': frozenset({1, 2, 3, 4, 5})}
>>> |
```

- `hasattr()` - verifying if you are having so and so attribute or not
- `hash()` - it'll show the hash value of the particular object you called
- `help()` - gives details about any method or class
- `hex()` - base conversion

```
>>>
>>> s1 = 'abcde'
>>>
>>> hasattr(s1, 'lower')
True
>>> hasattr(s1, 'isdigit')
True
>>> hasattr(s1, 'total')
```

**SyntaxError: EOL while scanning string literal**

```
>>> hasattr(s1, 'total')
False
>>> hash(s1)
-2359171732725835016
>>> n = 10
>>> hash(n)
10
>>> f = 12.345
>>> hash(f)
795515838178725900
>>>
```

```
>>>
>>> s1 = 'abcde'
>>>
>>> help(s1.lower)
Help on built-in function lower:

lower() method of builtins.str instance
    Return a copy of the string converted to lowercase.
```

```
>>> help(s1.isdigit)
Help on built-in function isdigit:
```

```
isdigit() method of builtins.str instance
    Return True if the string is a digit string, False otherwise.

A string is a digit string if all characters in the string are digits and there
is at least one character in the string.
```

- `isinstance()` - check if an object is an instance of particular class
- `issubclass()` - tells if a class is a subclass or not
- `iter()` - helps us iterate over every element in a sequence used along with `next()`

```
>>>
>>> s1 = 'abcde'
>>> n=10
>>> f = 12.34
>>>
>>> isinstance(s1 , str)
True
>>> isinstance(n , int)
True
>>> isinstance(n , float)
False
>>> False
False
>>>
>>> L = [10 , 'john' , 15.76, 'James']
>>> itr = iter(L)
>>> next(itr)
10
>>> next(itr)
'john'
>>> next(itr)
15.76
>>> next(itr)
'James'
>>>
```

## Build-in Function #3

- Functions we already know
  - `len()` - find length of any object
  - `list()` - creating items of list
  - `local()` - give local variables of given function
  - `object()` - base class for all python object
  - `oct()` - base conversion function
  - `open()` - opening a file
  - `ord()` - gives ascii code for given class
  - `print()` - prints the result
  - `range()` - gives value in specific range
  - `set()` - create object of set class
  - `str()` - create object of str class
  - `super()` - refers object of super class
  - `tuple()` - create object of tuple class
  - `Type()` - gives type if datatype

## - New Functions

- `map()` - map element of one sequence into another
- `max()` - gives maximum value in a sequence
- `min()` - gives minimum value in a sequence
- `Sum()` - gives sum of all elements in a sequence
- `sorted()` - sort a sequence
- `slice()` - gives slice object of a sequence

```
>>> L1 = [1, 2, 3, 4, 5]
>>> L2 = [5, 6, 7, 8, 9]
>>>
>>> m = map(lambda x:x**2,L1)
>>> list(m)
[1, 4, 9, 16, 25]
>>>
>>> m = map(lambda x,y:x+y,L1,L2
           )
>>> list(m)
[6, 8, 10, 12, 14]
>>>
>>> max(L1)
5
>>> min(L1)
1
>>> sum(L1)
15
>>> sorted(L1)
[1, 2, 3, 4, 5]
>>> sorted(L1,reverse=True)
[5, 4, 3, 2, 1]
>>> s = slice(0,2)
>>> L2[s]
[5, 6]
>>>
```

- `zip()` - join elements from corresponding 2 sequence
- `reversed()` - same as iterator but perform reverse iteration
- `pow()` - gives power value of a given number
- `round()` - round up the integer value

```
>>> L1 = ['A', 'B', 'C', 'D']
>>> L2 = [2, 4, 6, 8, 10, 12]
>>>
>>> z = zip(L1,L2)
>>>
>>> z
<zip object at 0x7fa3c99af180>
>>>
>>> for x,y in z:
    print(x,y)
```

```
A 2
B 4
C 6
D 8
```

```
>>> rev = reversed(L1)
>>> next(rev)
'D'
>>> next(rev)
'C'
>>> pow(2,4)
16
>>> 2**4
16
>>> round(12.3333)
12
>>>
```

## Module

- A module can contain predefined variable , functions , classes and we use this things in different programs with the help of import
- Modules are nothing but python files . Normal programs can be modules too
- 

```
total_amount = 2000

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

def sub(x,y):
    return x-y
```

- 
- Lets save this program as ModuleOne

```
import ModuleOne as mo

n1 = 10

n2 = 20

print('Sum',mo.add(n1,n2))

print('Diff',mo.sub(n2,n1))

print(mo.total_amount)
```

- 
- And this program as MyProgram
- In program ModuleOne we are defining the function and in program Myprogram we are calling the function ( add , sub )
- What happens if we write print function in ModuleOne program??

- it will print that function in MyProgram output
- If we want to execute only MyProgram.py but not ModuleOne.py(add and subtract ) print function
- For this we can ass print (\_\_name\_\_)
- It means the printing should be done only if its in main . If it is importing then don't execute the print function
- We can also write

```
if __name__=='__main__':
    print(__name__)
    print('From Module One',add(20,30))
    print('From Module One',sub(50,10))
```

## what are Exceptions and types of error

- Exceptions are runtime error

### Syntax error :-

If there is the typing mistake in the program then it will show syntax error

```
if x>y  
    print("something")
```

In if statement :(colon) is missing and in print indentation is missing

This types of errors is said as syntax error. If there is syntax error then the program will not run

### Logical error :-

```
a = 20  
b = 7  
c = 2  
d = a / b-c  
d = 20/7-2 = 4
```

```
a = int(input('Enter first number'))  
b = int(input('Enter second number'))  
c = int(input('Enter third number'))  
  
d = a // b - c  
  
print(d)
```

```
Enter first number20  
Enter second number7  
Enter third number2  
0
```

How can we correct this logical error . By adding parentheses to it

```
d = a // (b-c)
```

By doing this the answer will be 4

# Programmer user exception

Developer develops software or app and client uses it

## Types of errors

1. syntax error
2. Logical error
3. Runtime error

a developer has to give error free software to user

1. **Syntax error:** developer is responsible for syntax error.

2. **Logical error:** These errors depicts flaws in the logic of program. The program might be using wrong formula or the design of the program is wrong itself. Logical errors are not detected either by python compiler or Pam. The programmer is solely responsible for them.

3. **Runtime error:** The reason could be giving invalid input  
File not available, no connection, etc.

- Runtime errors are caused due to mishandling a software at client side
- Due to runtime error program crashes stops suddenly
- User is responsible for runtime error
- But when developer gives robust idea it can be handled means Exception handling given in program.

How to handle the runtime error?

Example:

A printer :

When the manufacturer gives any property on which the printer will blink if paper is not inserted.

It blinks because it needs some input means it is raising an exception.

Like wise also a developer can raise an exception asking user to check if there is anything missing.

# Examples of Exception

1. Index error
2. Key error
3. Value error and type error
4. Zero division error

When these errors occur

```
L=[10, 20, 30, 40, 50]
```

```
index=int(input('enter a index') # user has to enter value here  
print(L[index])
```

## Program:

```
L=[10, 20, 30, 40, 50] # LIST OF ELEMENT  
index= int(input('Enter index')) # ASKING USER TO ENTER INDEX  
print(L[index])
```

- If we print valid index it will print else
- If not valid index it will print error
- So ,user itself is responsible for giving wrong index

## What our program can do ?

- It can give a message "please enter a index which is from 0 to 4 only"
- Now there in output the error will be given as traceback, means it is showing in which line the error is appearing.

**Index error:** List index out of range



Name of the error.      This is the description of the error

**Value Error and Type error:**

```
val= int('abc') # here we are doing type conversion
```

```
res='2'+3 # this type is called as value error
```



We are saying here '2'+3  
2 to be concatenate with 3  
But it is a string how can a type int be concatenated with string

```
a=10
s='number' → THIS IS TYPE ERROR
print(s+a)
```

### Key error:

```
D=[1:'a', 2:'b', 3:'c']# dictionary list
key=int(input('Enter a key'))
print(L[key])
D={1:'a', 2:'b', 3:'c'}
key= int(input('Enter a key'))
print(D[key]) # it will show key error
```

Two possibilities of getting an error

1. Not entering a proper key and
2. entering string instead of integers

### Zero Division Error:

```
a= int(input('Enter first number'))
b=int(input('Enter second number'))
res=a/b
print(res)# it will show zero division error
```

# Exception Handling construct

We will learn how to handle exception in python.

## Python skeleton for try-except

syntax:

```
1)---  
2)---  
3)---  
Try  
4)---  
5)---  
6)---  
Except:  
7)---  
8)---  
9)---  
10)---
```



- This may cause Exception . So the lines which may cause Exception are written inside the block.
- Inside except we may print the message about the exception.
- The program starts running from the first line then continue 2,3,4,5,6 then it will not go to line 7 as there is no error.

### What happens if there is no error?

- First line , second line gets executed then third fourth . Suppose there is an error in 5t line.The exception raised in 5th line then it will not execute in 6th line.
- It will directly jump to 7th line i.e except block.
- If there is any exception in try block then it will execute except block. If condition gets in 4th , 5th and ,6th will not execute and remaining 8910 execute properly.
- The programme executes successfully it will not get terminated or crash abruptly it gets terminated gracefully
- If suppose try and accept has not been used.
- If there is an exception. In the 4th line . The program will crash

Program

Input

```
L=[10, 20, 30, 40, 50]#list of 5 elements
index=int(input('enter index'))#taking input as index
print(L[index])#printing the element at given each given index
print('terminated gracefully')#message
```

Output:enter index3

40

terminated gracefully

enter index9

Traceback (most recent call last):

File "/Users/abdulbari/PycharmProjects/pythonProject/python programs.py", line 3, in <module>  
 print(L[index])#printing the element at given each given index

IndexError: list index out of range

If the index is out of range it will get an exception error i.e index error

If we give an invalid index it will raise an exception

Program 2:

Input

```
L=[10, 20, 30, 40, 50]#List of 5 elements
try:
    index=int(input('enter index'))#taking input as index
    print(L[index])#printing the element at given each given index
except:
    print('invalid index')
print('terminated gracefully')#message
```

Output 1:

enter index3

40

terminated gracefully

Output2:

enter index9

invalid index

terminated gracefully

Now the program does not crash it executes safely.

## Multiple Exceptions

- If the Index is not given properly it'll give **index error**
- You can handle a particular exception as well by simply defining except with that error, then this except block will not handle any other except block other than what is defined
- If exception block is not defined it'll handle all type of exceptions.

```
l = [10, 20, 30, 40, 50]
try:
    index = int(input('enter index')) # take int as I/P from user
    print(l[index]) # printing the result
    print('end of try block') # indicating end of try block

except: # if problem occur it enters except block
    print('invalid index') # prints an exception has occurred

print('terminate gracefully') # prints that a program has ended gracefully
```

output 1

```
enter index9
invalid index
terminate gracefully
```

output 2

```
enter indexxyz
invalid index
terminate gracefully
```

- You can write multiple except block to handle different type of exception

```

l = [10, 20, 30, 40, 50]
try:
    index = int(input('enter index')) # take int as I/P from user
    print(l[index]) # printing the result
    print('end of try block') # indicating end of try block

except IndexError: # when wrong/no index this exception is raised
    print('invalid index')
except ValueError: # when proper value is not given this exception is raised
    print('enter only integer value')

print('terminate gracefully') # prints that a program has ended gracefully

```

Output 1

```

enter index9
invalid index
terminate gracefully

```

output 2

```

enter indexxyz
enter only integer value
terminate gracefully

```

- Instead of writing two except block we can write a single except block that can handle both the exception

```

l = [10, 20, 30, 40, 50]
try:
    index = int(input('enter index')) # take int as I/P from user
    print(l[index]) # printing the result
    print('end of try block') # indicating end of try block

except (IndexError, ValueError) as msg: # writing multiple exceptions
    # in single block
    print(msg)

print('terminate gracefully') # prints that a program has ended gracefully

```

Output 1

output 2

```

enter index9
list index out of range
terminate gracefully

```

```

enter indexxyz
invalid literal for int() with base 10: 'xyz'
terminate gracefully

```

## Why Try Except

Can we handle our program without using try and except?

Yes

Why try except is required

ZeroDivisionError

```
a=int(input('enter first number'))#taking integer input and storing it into a
b=int(input('enter second number'))
res=a//b # floor div but this div may cause error if a number is divided by zero
print(res)

#this program may cause zero division error

a=int(input('enter first number'))#taking integer input and storing it into a
b=int(input('enter second number'))
c=div(a, b)

If c== -1
    print('zero division error')
else:
    print(c)
```

It's working without Try-Except also

But problem arises when we use codes

If a function able to divide it will

If it's not able to divide then it raises the exception

So, there are two possibilities so, here div function is also having two possibilities

```
def div(a, b):
    if b != 0:
        c = a // b
        return c
    else:
        return -1

a = int(input('Enter first number'))
b = int(input('Enter second number'))

c = div(a, b)
if c == -1:
    print('Zero division error')
else:
    print(c)
```

```

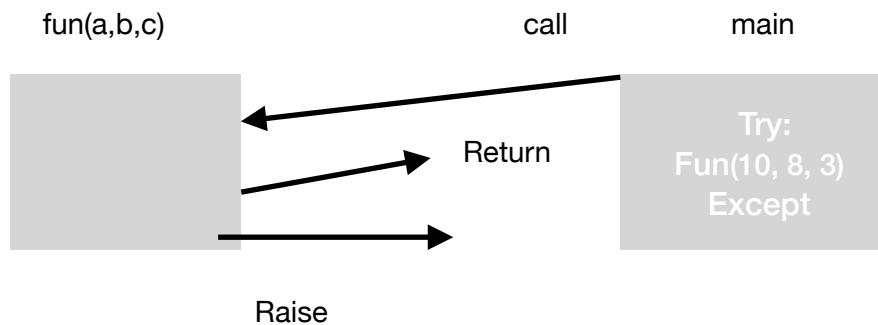
def div(a, b):
    if b != 0:
        c = a // b
        return c
    else:
        raise ZeroDivisionError

a = int(input('Enter first number'))
b = int(input('Enter second number'))

try:
    c = div(a, b)
    print(c)
except:
    print('Zero division error')

```

The output will be same but the internal working program will change



When you call a function there are two things possible:

Values return or

Exception raised

Main block should try and except

So this is the reason why try and except is introduced

So, when we communicate between two functions it's difficult to work using if and so, try and except is suitable.

## Try else

```
1 _____  
2 _____  
3 _____  
try :  
4 _____  
5 _____  
6 _____  
except :  
7 _____  
else :  
8 _____  
9 _____  
10 _____
```

### Try-except-else

As we already know we use for else, while else.

For.....

.....

Else:

.....

In for loop this else block is executed. If it is not breaking in between.

If for loop is stopping with break statement then else block will not be executed.

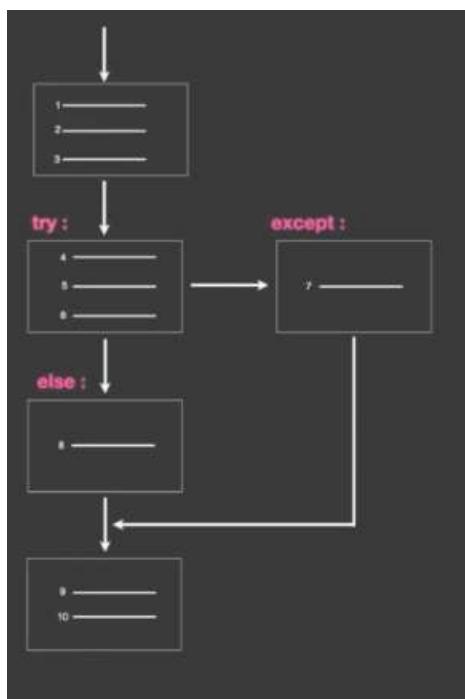
While as long as the condition written... in while loop is false  
Else: go to else condition

In both the conditions if for loop gets executed. Successfully the else will execute

In both the conditions if while loop gets executed. Successfully the else will execute

Same way if try block is executing without any exception raised then else block is executed

It means if there is any exception in try block then except block will execute and else will not be execute.



Else block confirms that try block executed successfully

```
print('before try')

try:
    a = int(input('Enter numerator'))
    b = int(input('Enter Denominator'))
    c = a // b
    print('Try Block executed successfully')
except ZeroDivisionError as err:
    print(err)
else:
    print('Division is', c)

print('outside try-except-else')
```

### Finally block

As we have already seen try except else, in previous lectures. Now we will see finally. The difference between else and finally is that else block is executed if there is no exception and finally block will execute if there is an exception. So, it is independent of the exception and it guarantees that it will always execute.

#### Program:

##### Input

```
print('before try')
try:
    a=int(input('enter numerator'))
    b=int(input('enter denominator'))
    c=a//b
except ZeroDivisionError as err:
    print(err)
else:
    print('division is ',c)
finally:
    print('finally block')
    print('outside try-except-else')
```

##### Output

```
before try
enter numerator2
enter denominator3
('division is ', 0)
finally block
outside try-except-else
```

```
def div(a,b):#def function
    try:
        c=a//b
        return c
    except ZeroDivisionError as err:
        raise ZeroDivisionError#except block handling error
    finally:
        print('finally block')
    z= div(10, 2)#calling the function and storing the result
    print(z)#printing value of a variable
```

## Program:

Finally is useful inside the function

## User define Exception

- When we use python on real life application . Lets take an example for age . Age can't be enter in negative value .
- What if the user gave negative age as a input .But these types of constraints cannot be applied in the python programs automatically
- To handle this type of exception we use user define exception
- A program automatically terminates after showing which inbuilt exception has occurred while executing the program when it reaches into an undesired state. We can stop it by using user define exception .
- User defined exceptions can be implemented by raising an exception explicitly,

```
class MyError(Exception):
    pass

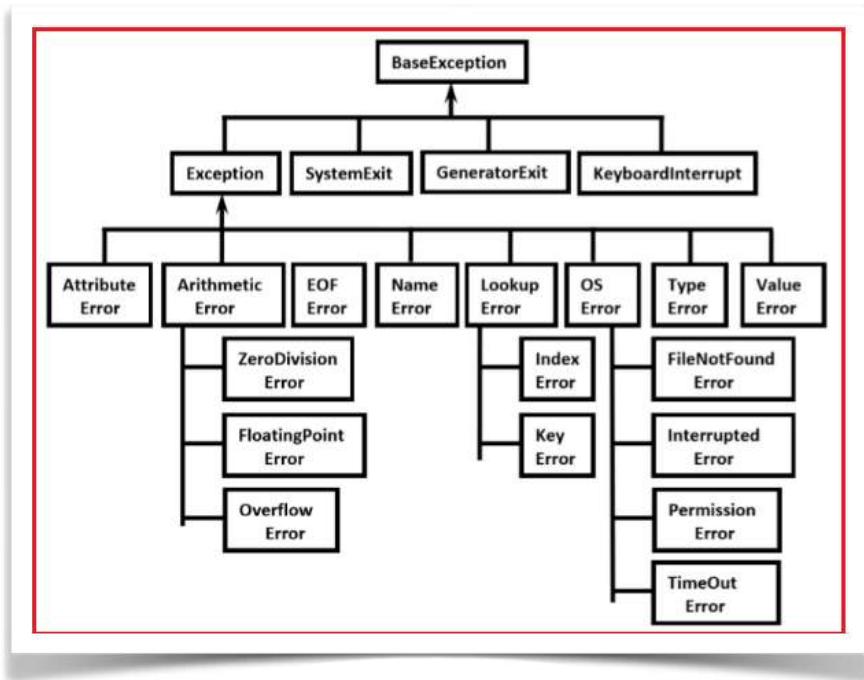
try:
    raise MyError('My Error')
except MyError as e:
    print(e)
```

- The syntax for raise statement is `raise ExceptionName` . When an error is raised, the code in the except block should handle the exception otherwise it will cause error in the program.
- So, we can see that after using raise statement in try except block, the program gives correct output in both the cases. The diagrams show the built in exception.

We have discussed about , exception , athematic error , zero division etc ...

The built-in exceptions listed below can be generated by the interpreter or built-in functions

User code can raise built-in exceptions. This can be used to test an exception handler



A single try statement can have multiple except statements. This is useful when the try block contains statements that may throw different types of exceptions.

```
class MyError(Exception):
    def __init__(self, msg):
        self.msg=msg

    def __str__(self):
        return self.msg

try:
    raise MyError('My Error')
except MyError as e:
    print(e)
```

## Netsted try - except

- One try block can have multiple except block
- Nested try - except is also possible in try block
- Some combinations of nested try - except are

```
try :  
try :  
....  
....  
try :  
try :  
....  
....  
except :  
except :  
....  
....  
except :  
except :  
....  
....  
try :  
....  
....  
except :  
....  
....
```

```
try :  
try :  
try :  
try :  
except :  
except :  
except :  
except :  
....  
....  
....  
....  
....  
....
```

## Q) A program to divide 2 numbers .

# In this program there is a chance that we might get an error when the user is giving input , when the numbers are divided so , we place this risky code in try blocks and if the error actually occurred it should be handled by the particular except block in the program.

```
try:  
    a = int(input('enter first number : ')) # placing the risky code in try  
    try:  
        b = int(input('enter second number : ')) # placing the risky code in try  
        try: # placing the risky code in try  
            c = a // b  
            print(c)  
  
        except ZeroDivisionError as e: # if a number is / by zero this raise an exception for 3rd try block  
            print(e)  
  
    except ValueError: # if the user gives different values as I/P this raise an exception for 2nd try block  
        print('value error inner')  
  
except ValueError: # if the user gives different values as I/P this raise an exception for 1st try block  
    print('value error outer')
```

```
enter first number : 10  
enter second number : 5  
2
```

# output when correct input is given

```
enter first number : 10  
enter second number : abc  
value error inner
```

# output when incorrect value is given exception is raised

```
enter first number : 10  
enter second number : 0  
integer division or modulo by zero
```

# output when zero is divided by an integer value exception is raised

- It is suggested to use multiple try block instead of nested try except ( its completely optional and upto you )
- The same program can be written using multiple except block to a single try block as shown in below

```
try:  
    a = int(input('enter first number : '))  
    b = int(input('enter second number : '))  
    c = a // b  
    print(c)  
  
except ZeroDivisionError as e:  
    print(e)  
  
except ValueError:  
    print('value error outer')
```

```
enter first number : 10  
enter second number : 5  
2
```

# output when correct input is given

```
enter first number : 10  
enter second number : abc  
value error inner
```

# output when incorrect value is given exception is raised

```
enter first number : 10  
enter second number : 0  
integer division or modulo by zero
```

# output when zero is divided by an integer value exception is raised

## File Introduction

file-handler=open(filename, mode)

### What are files ?

- Files are used permanently in our laptop or computer is in hard disk drive(HDD)
- on this hard disk we stored the data in the form of files

### What type of Data is stored in files

- It can be stored in the form of text, image files, audio files or a movie file.
- Files on the hard disk can be identified by their names.
- And located at some place where we required path
- If we know the file. The program can access the file using path.

### How to read the data from file

Program:

Input:

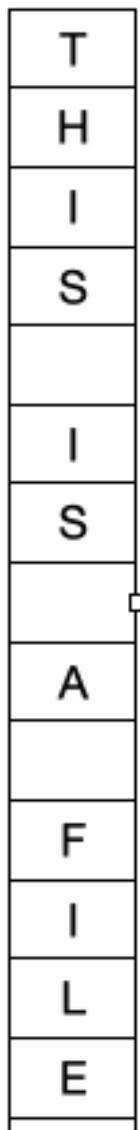
```
This is a test file  
it is stored in hard disk
```

```
file = open('MyData.txt', 'r')  
str1 = file.read()  
print(str1)
```

Output:

This is a test file  
it is stored in hard disk

What is file Pointer:



- Imagine this list as a long strip of characters we can also call it as an array of characters.
- File pointer keeps on proving until the last read and moves to the next character
- File pointer starts at beginning and ends at the end of file.

Program:

Input:

```
file = open('MyData.txt', 'r')
str1 = file.read(6)
print(str1)
str1=file.read(10)
print(str1)
```

Output:

This i  
s a test

- Whenever a person opens a file one should make sure to close it properly.  
`file_handler.close()`
- File is like a resource for a program
- File is an external part of program not a core part of program

## Modes of Opening a File

- The modes in which a file can be opened are r , w , a , r+ , w+ , a+ , x

mode	Description
r	To read data from file
w	To write data from file
a	To append data to the file
r+	To read and write data of a file
w+	To write and read data into a file
a+	To append and read data of a file
x	To open the file in exclusive creation mode

- In **r** i.e, read mode the file should exist only then it can be read if the file doesn't exist then **FileNotFoundException** occurs
- 'w' write mode , in this mode when we open the file it may exist (or) It may not in both cases it works , If file doesn't exist, it creates new file if file exist , it will clear the previous contents of file.

```
file = open('ModeDemo.txt','w')
file.write('Hello!\n')
file.write('How are you\n')
file.write('Do you know python\n')

file.close()
```

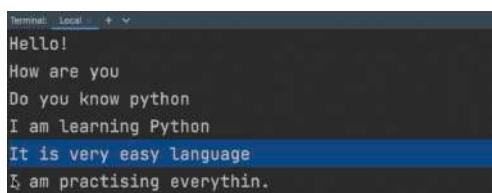
```
Hello!How are youDo you know python(venv) (base) MacBook-Pro-2:FileHandling abdulbari$ cat ModeDemo.txt
Hello!
How are you
Do you know python
```

- ‘**a**’ append mode , its same as write but the file must already exist and when open it will not clear the previous contents of the file , it writes data at the end of existing file

```
file = open('ModeDemo.txt', 'a')
file.write('I am learning Python\n')
file.write('It is very easy language\n')
file.write('I am practising everythin.\n')

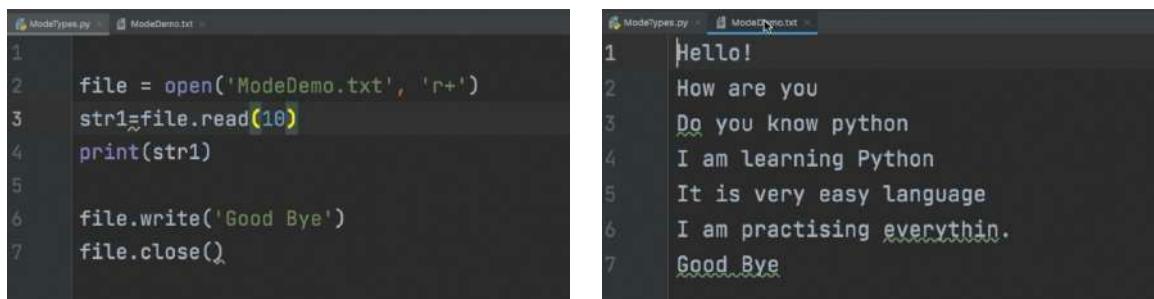
file.close()
```

## Output:



```
Hello!
How are you
Do you know python
I am learning Python
It is very easy language
I am practising everythin.
```

- r+** we can open a file in reading mode and as well as for writing
- w+** we can open a file in writing mode and as well as for reading
- a+** Means append and read
- X** File should not exist already , it will create and write in a file



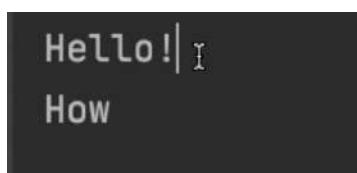
ModeTypes.py

```
1
2 file = open('ModeDemo.txt', 'r+')
3 str1=file.read(10)
4 print(str1)
5
6 file.write('Good Bye')
7 file.close()
```

ModeDemo.txt

```
1 Hello!
2 How are you
3 Do you know python
4 I am learning Python
5 It is very easy language
6 I am practising everythin.
7 Good Bye
```

## Output:



```
Hello!
How
```

## File Methods

### Operations on files

```
file.read([size])  
file.readline([size])  
file.readlines([sizehint])  
file.write(str)  
file.writelines(sequence)  
file.flush()  
file.close()  
readable ()  
writable ()  
file.tell()  
file.seek(offset[, whence])
```

file=open('---')

While opening a file we are getting object of which class ?

Program:

Input

```
file = open('MyData.txt')  
print(type(file))
```

Output:

<type 'file'>

If we don't give anything it is in readymade by default.  
If given 'r' will get the same thing

Program  
Input:

Output:

```
file = open('MyData.txt')
print(type(file))
print(dir(file))# to get members of the class
```

```
['__class__', '__delattr__', '__doc__', '__enter__', '__exit__', '__format__', '__getattribute__',
 '__hash__', '__init__', '__iter__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__setattr__',
 '__sizeof__', '__str__', '__subclasshook__', 'close', 'closed', 'encoding', 'errors', 'fileno', 'flush',
 'isatty', 'mode', 'name', 'newlines', 'next', 'read', 'readinto', 'readline', 'readlines', 'seek',
 'softspace', 'tell', 'truncate', 'write', 'writelines', 'xreadlines']
```

Getting members of the class

### Properties of a file

Using function read

Program:

Input:

```
file = open('MyData.txt', 'r')#asking file to read
file.close()#closing a file
print(file.name)#asking to print the file name
print(file.mode)#asking to print the mode of file
print(file.close)
```

Output:

```
MyData.txt
r
<built-in method close of file object at 0x10752ddb0>
```

Using function read line

Program:

Input:

```
file = open('MyData.txt', 'r')#asking file to read
line = file.readline()#reading a line
print(line)
line = file.readline()# using second time
print(line)
```

Output:

apples are red

grapes are green

To avoid the spaces **write print(line, end=“”)**

The difference between read and readline is read will read the entire content of string whereas readline reads whole line

And readline gives just a string and gives sets of strings.

To print for a loop

For line in lines:

print(line, end=“”)

If it works we will get data line by line

**File write(str): write for a string**

**File.write line (sequences):** Write the line for the content of any sequence

```
file=open('prop.txt', 'w')
```

```
str1= 'python is simple\n it is easy\n everything is object'
```

```
File.write(str1)
```

**Write lines(sequence)**

Program:

```
file = open('Prop.txt', 'w')

list1 = ['python is simple\n', 'it is easy\n', 'everything is an object\n']

file.writelines(list1)
```

**File.flush:** it is used for flushing the content from buffer onto the file

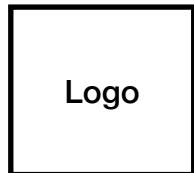
**File.close:** close is a method for closing a file

**readable():** used to know whether a file is readable or not

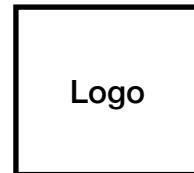
**writeable()**:used to know whether a file is writable or not.

## Copying binary file

- Image , audio , video all this files are in the form of binary



python1.jpg



python2.jpg

- Same content in both images . We will read the file and write in another file . It means we are copying a file
- Search for python logo on google and go to images and save the image on desktop folder ( as mypython)
- Open pycharm and save the prog as Copying.py and save this program at the same place where you have saved the image

```
f = open('Python1.jpg','rb')
data = f.read()

cp = open('Python2.jpg','wb')

cp.write(data)

f.close()
cp.close()
```

- This program shows that first we have the open the program1 file and read it as binary form
- Then open the second one with writing binary mode
- Then write a method and pass the data
- Close both the files

- Then how to run this type of program ??? For that we have to open command prompt ( cmd ) and compile the program and run it
  - C: users\ xxxx \ Desktop \ Mypython>python Copyfile.py
- Go to mypython folder and you can see the logo has been copied

## Random Access of Binary Files

- There are two ways of accessing a file
  1. Sequential Access - Read values in files one by one
  2. Random Access - We can go to any byte of file and read it
- Random access works for binary files as well for example

```
with open('my.data','wb') as f:  
    f.write(b'ABCDEFGHIJ')
```

```
C:\Users\Abdul Bari\Desktop\MyPython>python BinWrite.py  
  
C:\Users\Abdul Bari\Desktop\MyPython>dir  
Volume in drive C is OS  
Volume Serial Number is 3E57-713D  
  
Directory of C:\Users\Abdul Bari\Desktop\MyPython  
  
03/25/2022  01:46 AM    <DIR>      .  
03/25/2022  01:46 AM    <DIR>      ..  
03/25/2022  01:46 AM           63 BinWrite.py  
03/25/2022  01:46 AM          10 my.data  
                2 File(s)       73 bytes  
                2 Dir(s)  418,479,534,080 bytes free  
  
C:\Users\Abdul Bari\Desktop\MyPython>type my.data  
ABCDEFGHIJ  
C:\Users\Abdul Bari\Desktop\MyPython>
```

- Whenever a file is opened its **pointer** will be at index 0
- **tell()** method will tell you the **current position** of file pointer
- **seek()** is the method , which will **move the file pointer** to a certain given position in a file for example

```
with open('my.data','rb') as f:  
    print(f.read(2).decode())
```

```
C:\Users\Abdul Bari\Desktop\MyPython>python BinRead.py  
ab
```

- Moving the pointer

```
with open('my.data','rb') as f:  
    print(f.read(2).decode())  
    f.seek(3,0)  
    print(f.read(1).decode())
```

```
C:\Users\Abdul Bari\Desktop\MyPython>python BinRead.py  
ab  
d
```

- Moving the pointer from current position

```
with open('my.data','rb') as f:  
    print(f.read(2).decode())  
    f.seek(3,1)  
    print(f.read(1).decode())  
    ...
```

```
C:\Users\Abdul Bari\Desktop\MyPython>python BinRead.py  
ab  
f
```

- Moving the pointer from end of file

```
with open('my.data','rb') as f:  
    print(f.read(2).decode())  
    f.seek(-3,2)  
    print(f.read(1).decode())  
    ...
```

```
C:\Users\Abdul Bari\Desktop\MyPython>python BinRead.py  
ab  
h
```

## Pickle and Unpickle

- There is a module available in python that is used for storing and retrieving objects of a class
- The process of **storing multiple objects** of a class in a single file is called **Pickle**
- To **read** the entire object of a class at once is called **Unpickle**
- Therefore, storing an object in a file is **Pickle** , Retrieving an object from a file is called **Unpickle**
- The pickle module provides a function called ‘ **Dump** ’ , which will store the contents in a file
- Pickling is always done in **Binary Mode**
- The function called **Load** is used to read the file as well
- The **display function** is used to display the object in a file

### Example :

// Creating instance of a class named student

```
1
2  class Student:
3      def __init__(self, roll, name, dept):
4          self.roll = roll
5          self.name = name
6          self.dept = dept
7
8      def display(self):
9          print(f'Roll: {self.roll} \nName: {self.name}\nDept: {self.dept}')
10
11
```

- Writing this in separate file and importing this in second file

//Storing student object in second file

```
1 import Student, pickle
2
3 studs = [ Student.Student(10,'John','cs'), Student.Student(20, 'Ajay', 'ec'), Student.Student(30, 'Khan', 'me')]
4
5 with open('students.data', 'wb') as f:
6     for s in studs:
7         pickle.dump(s,f)
8
9
```

- Creating 3 objects in list of students , using for loop to call dump for storing object in files

// running the program in **CMD** , to check for any errors

```
C:\Users\Abdul Bari\Desktop\MyPython>python Storing.py
C:\Users\Abdul Bari\Desktop\MyPython>dir
 Volume in drive C is OS
 Volume Serial Number is 3E57-713D

Directory of C:\Users\Abdul Bari\Desktop\MyPython

03/25/2022  06:39 AM    <DIR>      .
03/25/2022  06:39 AM    <DIR>      ..
03/25/2022  06:37 AM            233 Storing.py
03/25/2022  06:24 AM            243 Student.py
03/25/2022  06:39 AM            207 students.data
03/25/2022  06:39 AM    <DIR>      __pycache__
               3 File(s)       683 bytes
               3 Dir(s)  417,277,509,632 bytes free
```

// Checking the student data , here we can see how the **dump function** works

```
C:\Users\Abdul Bari\Desktop\MyPython>python Storing.py
C:\Users\Abdul Bari\Desktop\MyPython>dir
 Volume in drive C is OS
 Volume Serial Number is 3E57-713D

Directory of C:\Users\Abdul Bari\Desktop\MyPython

03/25/2022  06:39 AM    <DIR>      .
03/25/2022  06:39 AM    <DIR>      ..
03/25/2022  06:37 AM            233 Storing.py
03/25/2022  06:24 AM            243 Student.py
03/25/2022  06:39 AM            207 students.data
03/25/2022  06:39 AM    <DIR>      __pycache__
               3 File(s)       683 bytes
               3 Dir(s)  417,277,509,632 bytes free

C:\Users\Abdul Bari\Desktop\MyPython>type students.data
<01:15student000>id0|i#rollK
<01:15student000>id0|i#name0
<01:15student000>id0|i#dept0
<01:15student000>id0|i#rollK5|i#name1
<01:15student000>id0|i#dept1
<01:15student000>id0|i#rollK6|i#name2
<01:15student000>id0|i#dept2
<01:15student000>id0|i#rollK7|i#name3
<01:15student000>id0|i#dept3
C:\Users\Abdul Bari\Desktop\MyPython>
```

- Using for loop for retrieving 3 objects and displaying them

```
import Student, pickle

with open('students.data','rb') as f:

    for i in range(3):
        s = pickle.load(f)
        s.display()
```

// Reading a file and loading objects from a file

```
C:\Users\Abdul Bari\Desktop\MyPython>python Reading.py
Roll: 10
Name: John
Dept: cs
Roll: 20
Name: Ajay
Dept: ec
Roll: 30
Name: Khan
Dept: me
```

## Zip and Unzip files

### Zip :

- We are compressing the file .
- Create a folder name is as MyPython . In that folder save images in png form.
- Images of csharp , cpp , java , python , javascript
- Open the Vs code . Save the file name as compress and file type python

```
from zipfile import *
f = ZipFile('images.zip','w',ZIP_DEFLATED)
f.write('cpp.png')
f.write('java.png')
f.write('javascript.png')
f.write('csharp.png')
f.write('python.jpg')
f.close()
```

- Here the programs says that we have to import the zip file
- Open the file in writing mode
- ZIP\_DEFLATED correspond to an archive member (a file inside the archive) which is compressed (or deflated)
- Open images in write mode
- Then close the file
- Open command prompt compile that program and run that
- Go to the folder (MyPython) you can see the zip file ( images) if you click on that you can see the images . It is compressed .

### Unzip :

- Lets see how to unzip and extract all the files( images )
- For this we need to write a code where we should open the file in read mode in zip we used write mode but in unzip we use read mode
- There is a method called extractall( ) . By calling this method we can decompress zip files .

```
from zipfile import *

f = ZipFile('images.zip', 'r')

f.extractall()

f.close()
```

- Save this program name as decompress . Open command prompt compile it and run it
- Go to folder ( MyPython ) u can see images .

# Principles of object oriented programming(oop's)

It is a paradigm(oop's is a paradigm)

## What is object?

- Everything in the world is tangible object or intangible object>
- Ex: student, car, washing machine, movie, order
- So we believe that anything in the world can be object.
- By defining in terms of its properties and methods

## Encapsulation:

- means combining all the related things together
- Ex: car, Tv, Washing machine.

## Abstraction:

- means showing required features and hiding internal details.
- To make an object first we design the classes
- Class: a Class is a blueprint of an object

## Inheritance:

- inheriting the features of existing class.
- Ex: parent->child.

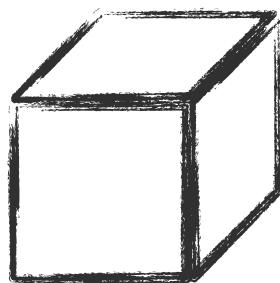
## Polymorphism:

- one name and different actions
- By using a single item we can refer multiple things together.

## Class v/s Object

- In OOP's we consider everything as an object and for every object there is a class
- So, class is a definition of an object . It is also known as blueprint , design , plan etc...
- Every object will have 2 things
  - 1) Properties / Attributes
  - 2) Methods / Operations / Function

Ex :



- The properties of a cuboid are l , b , h
- The methods of a cuboid are lidarea( ) , volume ( ) , totalarea( )
- A sample code for cuboid is

Class cuboid

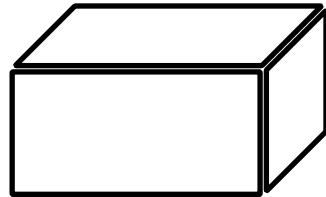
Properties :

Length  
breadth  
height

Methods :

lidarea( )  
totalarea( )  
volume ( )

## How to Write a class



### Methods :

```
lidarea() = l * b  
total() = l * b * b * h * l * h  
volume = l * b * h
```

### Writing a class using java

```
Class cuboid  
{  
int length;  
int breath;  
int height;  
cubiod(int length,int breath,int height)
```

#constructor will have same name as class name . It is use to initialise the object

```
{  
length = l;  
breadth = b;  
height = h;  
}  
int lidarea( ){....}  
int total( ){....}  
int volume ( ) {....}  
}
```

### Writing a class in python

- When we write a variable we also need to initialise it . Initialisation is done in constructor (java) but in python we can write constructor method
- Initialising and declaring both should be done in python

```
class Cuboid:  
    def __init__(self, l, b, h):  
        self.length = l  
        self.breadth = b  
        self.height = h  
  
    def lidarea(self):  
        return self.length * self.breadth  
  
    def total(self):  
        return 2 * (self.length * self.breadth + self.breadth * self.height + self.length * self.height)  
  
    def volume(self):  
        return self.length * self.breadth * self.height  
  
c1 = Cuboid(10, 5, 3)  
print(c1.volume())  
  
c2 = Cuboid(20, 10, 5)  
print(c2.volume())
```

- First parameter should be `self` , it is defined to declare and initialise it and properties are defined inside the function `__init__` and this is standard function that acts as the constructor in class .

Create object in java'

Cuboid c1=new cuboid(10,5,3)

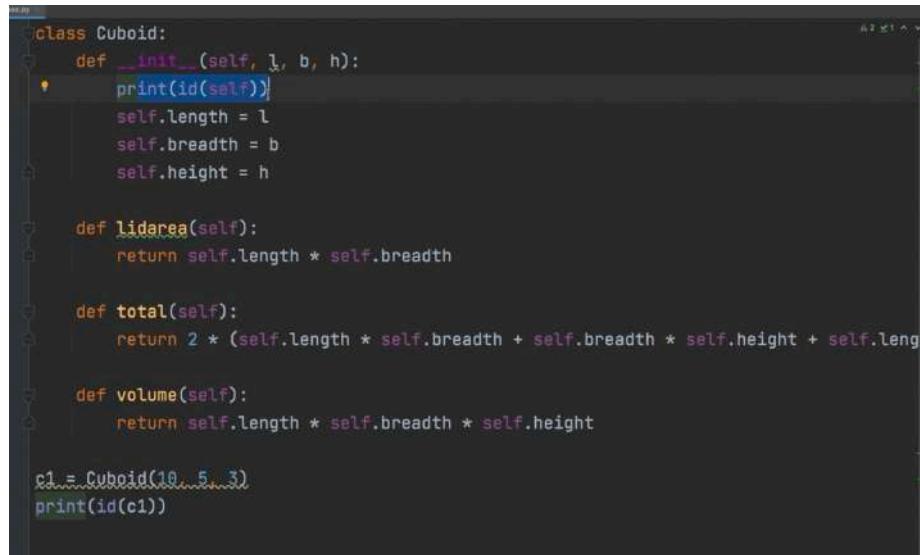
Create object in python

c1=cuboid(10,5,3)

## Self and Construct

### Self

- Self is the **reference** to the current value
- It is the first argument of the constructor
- If self is not given PVM will take the 1st parameter value as self by default
- You can write different names for self but self is recommended
- When you create 2 object the self will refer to the current object that is being called i.e, the self value will be same for these 2 object but it will show values of the method being called (i.e, 1st or 2nd )
- You can verify this by checking the ID of self in both method call



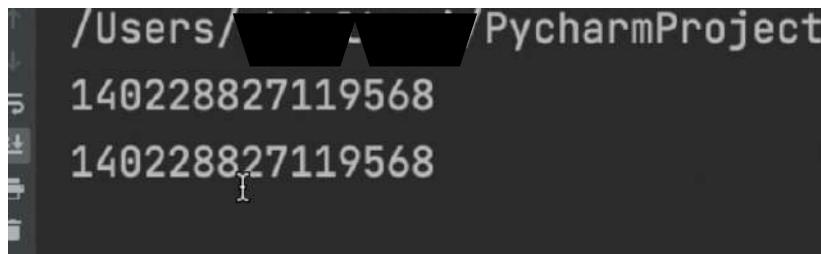
```
cuboid.py
class Cuboid:
    def __init__(self, l, b, h):
        print(id(self))
        self.length = l
        self.breadth = b
        self.height = h

    def lidarea(self):
        return self.length * self.breadth

    def total(self):
        return 2 * (self.length * self.breadth + self.breadth * self.height + self.length * self.height)

    def volume(self):
        return self.length * self.breadth * self.height

c1 = Cuboid(10, 5, 3)
print(id(c1))
```



```
/Users/.../PycharmProject
140228827119568
140228827119568
```

## Constructor

- In a method if `__init__` is used then that method becomes constructor method as `__init__` is a constructor.
- When you create an object of this class then automatically constructor method is called and initialised
- If a class having attributes then its mandatory to have `__init__` method. Therefore every class must have `__init__`
- When `__init__` is not given in a class then PVM will provide default constructor which will not have any property
- You can write more than 1 constructor but only 1 will execute but its not recommended to do so
- You can give default argument in `parameter` and when you call the method you can either give values or it will consider the default values i.e you can make constructor using different Parameters
- Constructors are not overloaded

```
class Cuboid:  
    def __init__(self, l, b, h):  
        print(id(self))  
        self.length = l  
        self.breadth = b  
        self.height = h  
  
    def lidarea(self):  
        return self.length * self.breadth  
  
    def total(self):  
        return 2 * (self.length * self.breadth + self.breadth * self.height + self.length * self.height)  
  
    def volume(self):  
        print(id(self))  
        return self.length * self.breadth * self.height  
  
c1 = Cuboid(10, 5, 3)  
print(id(c1))  
c1.volume()
```

```
140496912371664  
140496912371664  
140496912371664
```

### Program explanation(1 and 2):

- A class of cuboid is created , in method we are writing a constructor it takes 1st parameter as self, and we are initialisation it by writing its properties i.e length, breath and height , we are printing the ID of self as well to show that only one self is used for multiple methods call.
- Then we are defining a method for lidarea , total area and volume and initialisation it with self .
- When printing the results for lidarea , total area and volume we can see that the ID of self is constant while ans for theses 3 methods are different

## Types of Variables and Methods

In python there are different types of variables and methods

### Variables:

- Instance variable
- Static variable

### Methods:

- Instance method
- Class method
- Static method

### Instance :

```
class Rectangle:  
    def __init__(self,length, breadth):  
        self.length=length  
        self.breadth=breadth  
  
    def area(self):  
        return self.length * self.breadth  
  
    def perimeter(self):  
        return 2 * (self.length + self.breadth)
```

Example: Rectangle

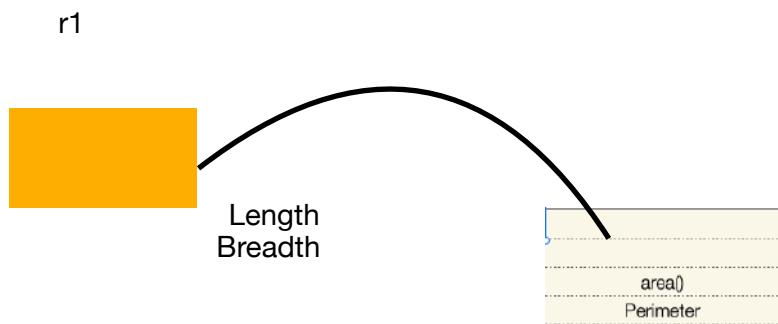


```
area= length*breadth
```

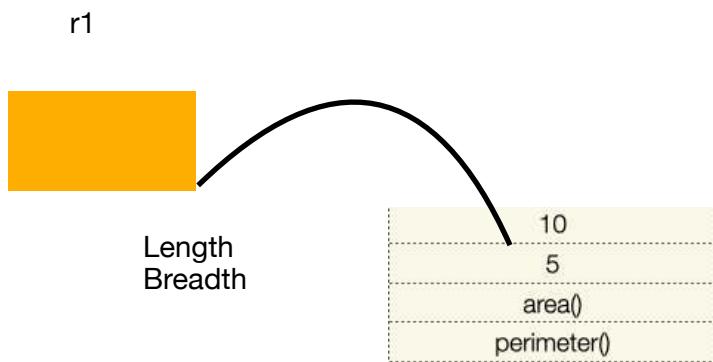
Creating a variable for rectangle

```
r1= rectangle() # automatically calls the constructor of the class
```

In the memory it looks like



It will form like this



```
r2= Rectangle(15,7) # another object
```

So again one more object screamed in memory

These are called instance of class

Each instance will have its own members

The parameter that is always taken by instance method is self and using self they access instance variable and the methods which are accessing them are called as instance methods.

### **Ways of creating instance variables.**

So, there are multiple ways we can declare instance variable.

### **Program:**

Input:

```
class Test:# creating a class
    def __init__(self):# constructor with instance variable
        self.a = 10
t1 = Test()
print(dir(t1))
```

Output:

```
['__doc__', '__init__', '__module__', 'a']
```

So it is having one variable that is a

## Program

Input:

```
class Test:# creating a class
    def __init__(self):# constructor with instance variable
        self.a = 10
        def fun(self):
            self.b=20
t1 = Test()
t1.c = 30
print(dir(t1))
```

Output:

```
['__doc__', '__init__', '__module__', 'a', 'c']
```

It is still having a and function but b is not there for that function should be called  
We have added one more variable to it so, it means we can declare and initialise  
Instance variable in init method also and other method also.

Instead of self we can use the variable name and.

Self is not a keyword it can be anything. The first we are writing is constructor is called as  
(instance variable)

We can give anytime instead of self

If a method is accessing instance variable then that method becomes instance method.

## Class and Static Variable

```
class Rectangle:  
    |  
    |  
    def __init__(self, length, breadth):  
        self.length = length  
        self.breadth = breadth  
    |  
    def perimeter(self):  
        return 2 * (self.length + self.breadth)  
    |  
    def area(self):  
        return self.length * self.breadth
```

When we declare a variable inside a class but outside any method, it is called as class or static variable in python.

- self.length and breath are instance variable.

```
def perimeter(self)
```

- Class variables can be accessed from within the class as well as from outside the class.

class and static method can access class variables. Each method has its own way of accessing static variables.

There are two ways of accessing class variables from inside class method-

- cls.ClassVariable
- ClassName.ClassVaraible

There is only one way to access class variables from static method-

- ClassName.ClassVariable

```
class Rectangle:
    count = 0

    def __init__(self, length, breadth):
        self.length = length
        self.breadth = breadth
        Rectangle.count += 1

    def perimeter(self):
        return 2 * (self.length + self.breadth)

    def area(self):
        return self.length * self.breadth

    @classmethod
    def countRect(cls):
        print(cls.count)

r1= Rectangle(10,5)
r2 = Rectangle(15,7)
```

- Here count= 0
- It is known as class or static cause its not inside the method
- The counter belongs to rectangle r1 and r2 that means count variable is common to both objects ( r1 and r2 )
- Count will belong to class ( Rectangle ) so it is also static( fixed, single copy to both r1,r2 )
- @classmethod ( decorator )

## Static Methods

- Instance method access instance variable, class method access class variable.
- These methods are like global functions. If there is any relation in the class we can put inside the class.
- The `@staticmethod` decorator returns a static method for a function passed as the parameter.
- 

```
class Rectangle:  
  
    def __init__(self, length, breadth):  
        self.length = length  
        self.breadth = breadth  
  
    def perimeter(self):  
        return 2 * (self.length + self.breadth)  
  
    def area(self):  
        return self.length * self.breadth  
  
    @staticmethod  
    def issquare(len, bre):  
        return len==bre  
  
r1= Rectangle(10,5)  
print(r1.issquare(10,10))
```

- Example : Given rectangle is square or not
- If the length and breadth both are equal then it is said as square
- This program will return true or false
- It is not accessing instance variable or class variable such methods are said as static method ( its doesn't access)
- We can use the decorator `@staticmethod`. It is not compulsory . It will just help to read ( readable)

## Accessors and Mutators

These are the types of methods that can be written inside any class

Accessor methods are useful for reading the property of a class or an object.

Mutator is used for writing and updating properties of class and its objects.

So these methods can be called as reading and writing methods

These methods are followed by object oriented programming

**Example :** class Rectangle

**Program:**

**Input:**

```
class Rectangle:
    def __init__(self, l, b):
        self.length = l
        self.breadth = b

    def getlength(self):#get method for length
        return self.length#returning parameter length for changing the length

    def setlength(self, l):
        self.length=l

    def area(self):#calculating area of the rectangle
        return self.length * self.breadth

    def perimeter(self):#this will return perimeter of a rectangle
        return 2*(self.length+ self.breadth)

r=Rectangle(10,20)
print(r.getlength())
r.setlength(15)#calculating the area by 15
print(r.area())#checking the area
```

**Output:**

```
10
300
```

## Introduction to Inheritance

- Inheritance is acquiring features in a class from an existing class
- Inheritance is like **parent - child relation** , a child class can have all of its parent features plus his own features
- The main advantage of Inheritance is **reusability** of code
- Every class in python directly or indirectly is inheriting from **object class**
- Suppose you want to access features of rectangle in cuboid as they are almost same then you can do the following

```
class Rectangle: # rectangle class
    def __init__(self,l,b): # initializing instance method
        self.length = l
        self.breadth = b
    def area(self):      # method on finding area of rectangle
        return self.length * self.breadth
    def perimeter(self): # method on finding perimeter of rectangle
        return 2 *(self.length + self.breadth)

class Cuboid(Rectangle): # creating class Cuboid inheriting features from Rectangle class
    def __init__(Cuboid, self):
        self.height = h

    def volume(self):    # method on finding volume of a cuboid
        return self.length*self.breadth*self.height
```

Rectangle



length

breath

Cuboid



height

length

breath

## Writing Construct Inheritance

- Lets understand the concept of construct with an example

```
class Rectangle:  
    def __init__(self, l, b):  
        self.length = l  
        self.breadth = b  
  
    def area(self):  
        return self.length * self.breadth  
  
    def perimeter(self):  
        return 2 * (self.length + self.breadth)  
  
class Cuboid(Rectangle):  
    def __init__(self, l, b, h):  
        self.height = h  
        super().__init__(l, b)  
  
    def volume(self):  
        return self.length * self.breadth * self.height  
  
c = Cuboid(3)  
print('Volumes is', c.volume())
```

```
Volumes is 150
```

- Our parent class is rectangle , which is having constructor and two methods I.e to find area and perimeter of a rectangle
- we are creating another class (child class) called cuboid which is inheriting all properties of a rectangle class but the cuboid class is also have its own constructor , we are unable to use rectangle method because
- When a child class is having its own construct parent class construct cannot be called in child class
- To overcome the constructor of child class we should use **super( ) method** , so, that we can access parent class method in child class
- Therefore , this process is called constructor overriding in Inheritance .

## Inner/Nested classes

It means we can define class in another class.

Means class can have class as it's members just like a class can have attributes and methods similarly a class can also have a class as it's members.

Let us write class for customer:

```
class Customer:

    def __init__(self, id, name):
        self.custid=id
        self.name=name
        self.baddr=Address()
        self.saddr=Address()
```

These are attributes for the class customer

```
class Customer:
    def __init__(self, id, name, baddr, saddr, city, country, pin):
        self.custid=id
        self.name=name
        self.baddr=baddr
        self.saddr=saddr
        self.city=city
        self.country=country
        self.pin=pin

    class Address:
        def __init__(self, dom, street, city, country, pin):
            self.dom=dom
            self.street=street
            self.city=city
            self.country=country
            self.pin=pin

        def display(self):
            print(self.dom)
            print(self.street)
            print(self.city)
            print(self.country)
            print(self.pin)

c = Customer(10, 'John', 101, 'Delhi', 'India', 110011, 'ijk', 'India', 40001)
c.saddr.display()
```

Output:

```
200
ijk
mumbai
india
40001
```

## Polymorphism

Polymorphism: many functions

Polymorphism means One name different actions

```
def Driver(car):
    car.drive()

class Creta:
    def drive(self):
        print('Creta is driving')

class Mercedes:
    def drive(self):
        print('Mercedes is driving')

c = Mercedes()
Driver(c)
```

Driver is a function who is driving the car and running cars.. so it is the example of polymorphism a driver can use or run multiple cars means different actions and forms but same function

```
def PetLover(pet):
    pet.talk()
    pet.walk()

class Duck: #defining a class duck
    def talk(self):
        print('Duck is Talking')

    def walk(self):
        print('Duck is Walking')

class Dog: #defining class dog
    def talk(self):
        print('Dog is Talking')

    def walk(self):
        print('Dog is Walking')
```

```
d = Dog()
```

```
PetLover(d)
```

## Method Overloading

Method overloading is used for achieving polymorphism

What is polymorphism:

- One name different actions or multiple actions
- Python supports overloading writing one method which can act differently in different situations.

How python supports overloading ?

```
Class Arith:# creating class  
Def sum(x, y)# takes two parameters  
Return x+y # returning parameters by adding
```

```
a= arith()  
print(a.sum(10,5))# calling sum  
A.sum(8.5, 7.6)
```

```
print(a.sum("Hello", "world"))
```

- It is adding int, float , string it is nothing but we are achieving polymorphism
- Python supports overloading implicitly.

Program:

Input:

```
class Arith:  
    def sum(self,a,b):# takes two parameters  
        return a+b  
a= Arith()  
print(a.sum(10, 5))
```

Output:

15

Input:

```
class Arith:  
    def sum(self,a,b):# takes two parameters  
        return a+b  
a= Arith()  
print(a.sum(8.5, 7.6))
```

Output:

16.1

Input:

```
class Arith:  
    def sum(self,a,b):# takes two parameters  
        return a+b  
a= Arith()  
print(a.sum('Hello ', 'world'))
```

Output:

Hello World

- Concatenation means it will just attach first string to the second string

Input:

```
class Arith:  
    def sum(self,a,b):# takes two parameters  
        return a+b  
  
    def sum(self, x, y, z):# takes three parameters  
        return x + y + z  
  
a= Arith()  
print(a.sum('Hello ', 'world'))  
print(a.sum(5, 10, 3))# calling the sum
```

Output:

error

- Python does not allow two methods using same name.
- It will try to call out the second method and shadowed the second method
- We can never call the first method by calling two parameters

How single parameter works for both 2 and 3 parameters

Input:

```
class Arith:  
    def sum(self, a, b, c=None):  
        s = a + b  
        if c==None:  
            return s  
        else:  
            return s + c  
  
a = Arith()  
print(a.sum(5,10,3))  
print(a.sum(8, 9))
```

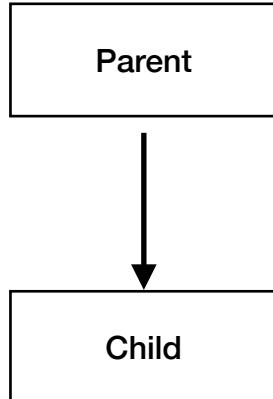
## Output:

18  
17

- So python is having method overloading implicitly
- We don't have to write multiple methods with the same name but the single method can behave in different type of method.

## Method Overriding

- Redefining the method of parent class in child class is said as method overriding
- In other words, the child class has access to the properties and functions of the parent class method while also extending additional functions of its own to the method.



Class **iphone6**:

```
def home(self)
```

Class **iphone x**:

- iphone x is newly designed phone . It is inheriting the features of iphone6 . but it is having new home method . it is not borrowing the method (iphone 6) as it is because they was not physical button
- Both have home method . But the child method will be called parent method will be shadowed we can't call it directly. To override the Parent Class method, you have to create a function in the Child class with the same name and the same number of parameters.
- The child class should have the same name and the same number of parameters as the parent class.
- For self: to call a function firstly we have to create a variable then we can call the function

- Example : ph = iPhone(6) ——function calling
  - ph.home() ——method calling
- 
- You can't use super outside the class .method overriding cannot be performed in the same class, and overriding can only be executed when a child class is derived through inheritance.
  - super() home() —— this is how parent method is called . So it should be called inside the class

## Operator Overloading

- Operator Overloading is where operators **behave differently** in different situations
- Operator Overloading is possible in normal DataType also
- Using operators we can **achieve polymorphism**
- Lets see this with an example
- We can add 2 rational numbers using operator overloading

```
class Rational:  
    def __init__(self, p=1, q=1):  
        self.p=p  
        self.q=q  
  
    def __add__(self, other):  
        s = Rational()  
        s.p = self.p * other.q + self.q * other.p  
        s.q = self.q * other.q  
        return s  
  
r1=Rational(2,3)  
r2=Rational(2,5)  
  
sum = r1+r2  
print(sum.p, sum.q)
```

# we are writing a constructor and method through this way we add 2 rational numbers

```
16 15  
Process finished with exit code 0
```

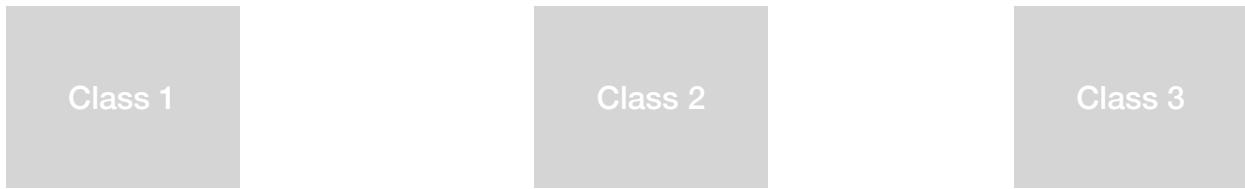
- You can also overload for **subtraction** , **multiplication** , **power** , **division** etc.

Binary Operators	
<code>__add__()</code>	Addition
<code>__sub__()</code>	Subtraction
<code>__mul__()</code>	Multiplication
<code>__pow__()</code>	Power
<code>__truediv__()</code>	Division
<code>__floordiv__()</code>	Floor Division
<code>__mod__()</code>	Remainder (modulo)
<code>__lshift__()</code>	Bitwise Left Shift
<code>__rshift__()</code>	Bitwise Right Shift
<code>__and__()</code>	Bitwise AND
<code>__or__()</code>	Bitwise OR
<code>__xor__()</code>	Bitwise XOR

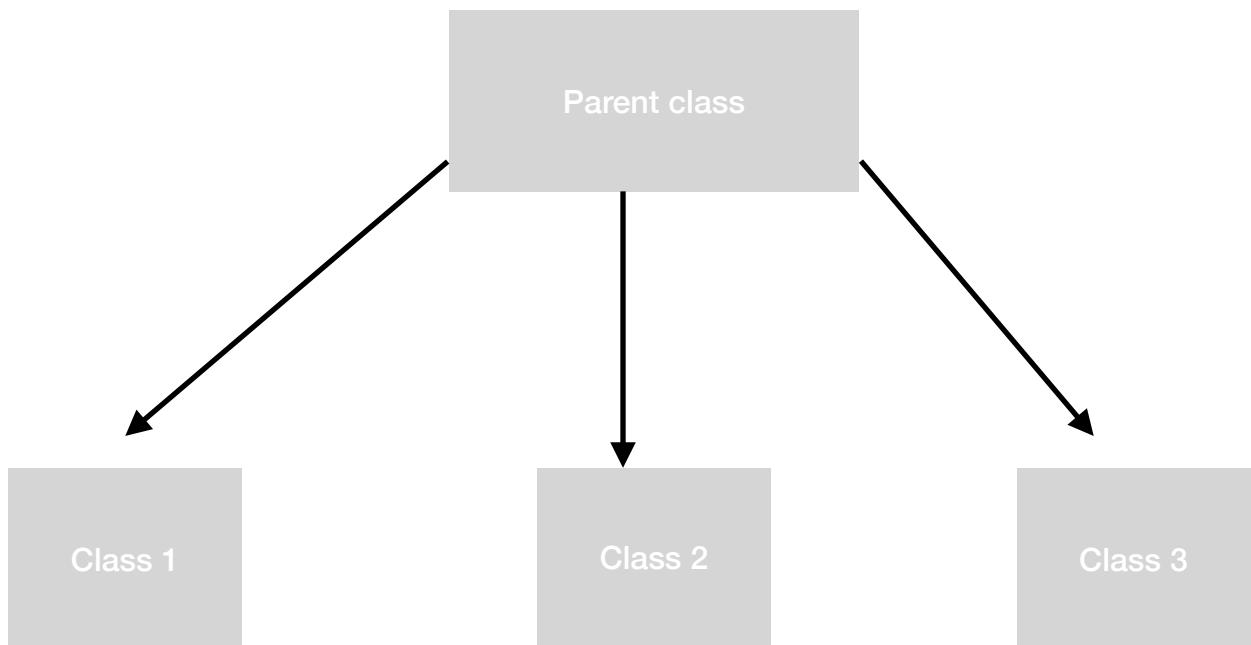
- Operator Overloading also works well for **comparison operators** and **assignment operators**.

Assignment Operators		Comparison Operators	
<code>__iadd__()</code>	equivalent to <code>a += b</code>	<code>__lt__()</code>	Less than
<code>__isub__()</code>	equivalent to <code>a -= b</code>	<code>__le__()</code>	Less than or equal to
<code>__imul__()</code>	equivalent to <code>a *= b</code>	<code>__eq__()</code>	Equal to
<code>__idiv__()</code>	equivalent to <code>a /= b</code>	<code>__ne__()</code>	Not equal to
<code>__ifloordiv__()</code>	equivalent to <code>a // b</code>	<code>__gt__()</code>	Greater than
<code>__imod__()</code>	equivalent to <code>a %= b</code>	<code>__ge__()</code>	Greater than or equal to
<code>__ipow__()</code>	equivalent to <code>a **= b</code>		
<code>__ilshift__()</code>	equivalent to <code>a &lt;&lt;= b</code>		
<code>__irshift__()</code>	equivalent to <code>a &gt;&gt;= b</code>		
<code>__iand__()</code>	equivalent to <code>a &amp;= b</code>		
<code>__ior__()</code>	equivalent to <code>a  = b</code>		
<code>__ixor__()</code>	equivalent to <code>a ^= b</code>		

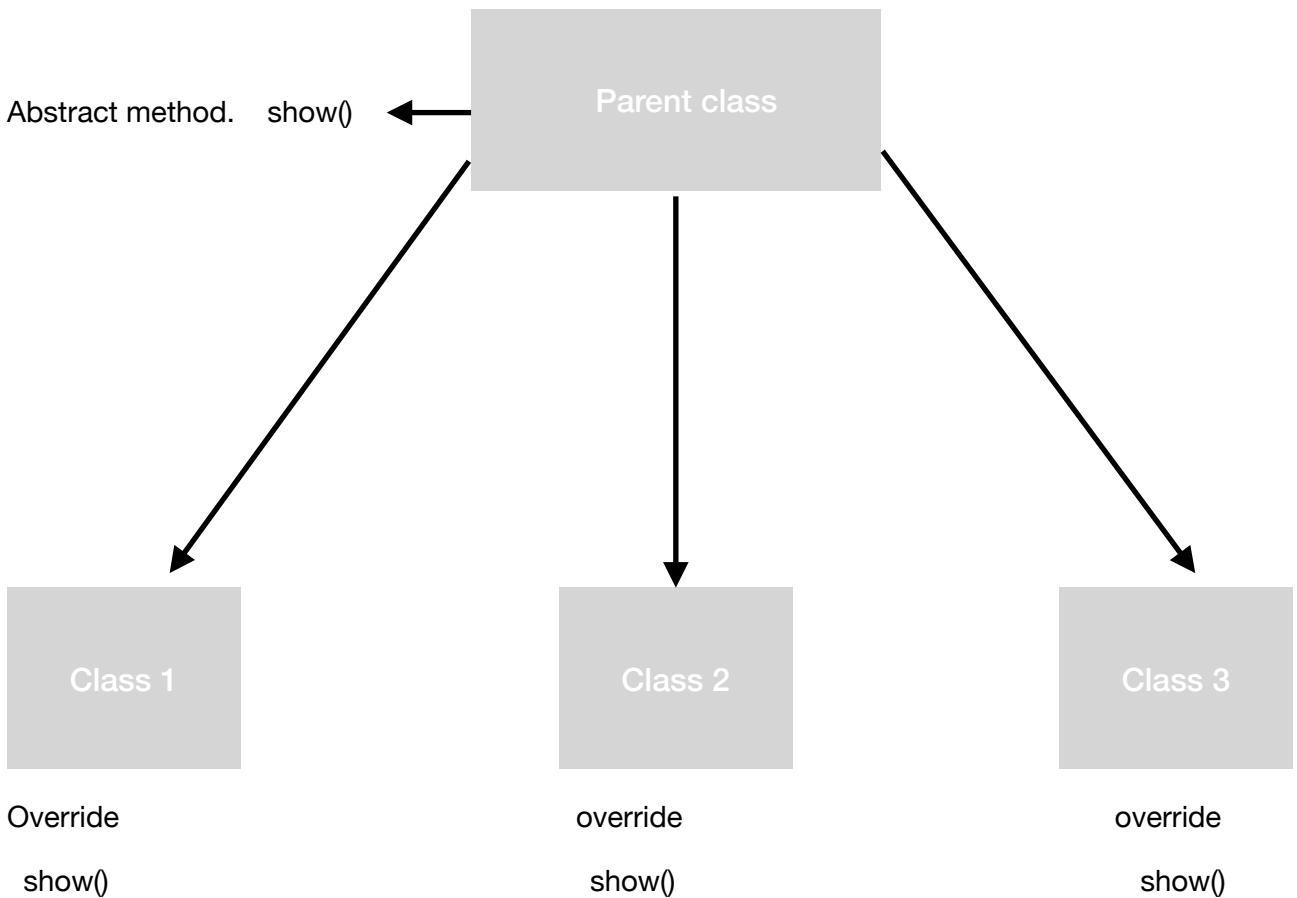
## Abstract classes and interface



- Instead of writing same method in each class and every class we can write a parent class and can inherit from parent class
- So, if we have any common code we can write it in a parent class.
- If we write a parent class then all these three class can inherit that from parent class
- As parent class is not directly used so, we can make it abstract
- Abstract means which cannot be instantiated.



- The whole and sole purpose of this class is to share the content with its child classes this is one reason to write abstract classes
- Abstract classes cannot be instantiated but when we write the child classes for this abstract classes then we can create the instance of these child classes.
- But there is one more reason to write abstract classes.



- Classes must have some set of methods. One or more classes.
- Which is mandatory.
- We can force that classes to inherit them from parent class.
- If the body is not there then this is abstract method
- These classes must override the method in parent class so then that parent class is called as abstract class

**Abstract:** abstract class may have some abstract method and some concrete method

- Concrete method for sharing the code and abstract method for forcing the class to override method

**Interface:** interface means abstract class only but which is having only abstract class. Just for forcing child classes to override the method that is why we call it as an interface.

- Abstract classes interfaces are not directly supported by interpreter of python but it is provided as a module.

```
from abc import ABC, abstractmethod

class Parent(ABC):#abstract class #any class inheriting abc becomes abstract class

    @abstractmethod
    def show(self):
        pass

    def display(self):#concrete method #a method which have body is called concrete
        print('Parent Display')

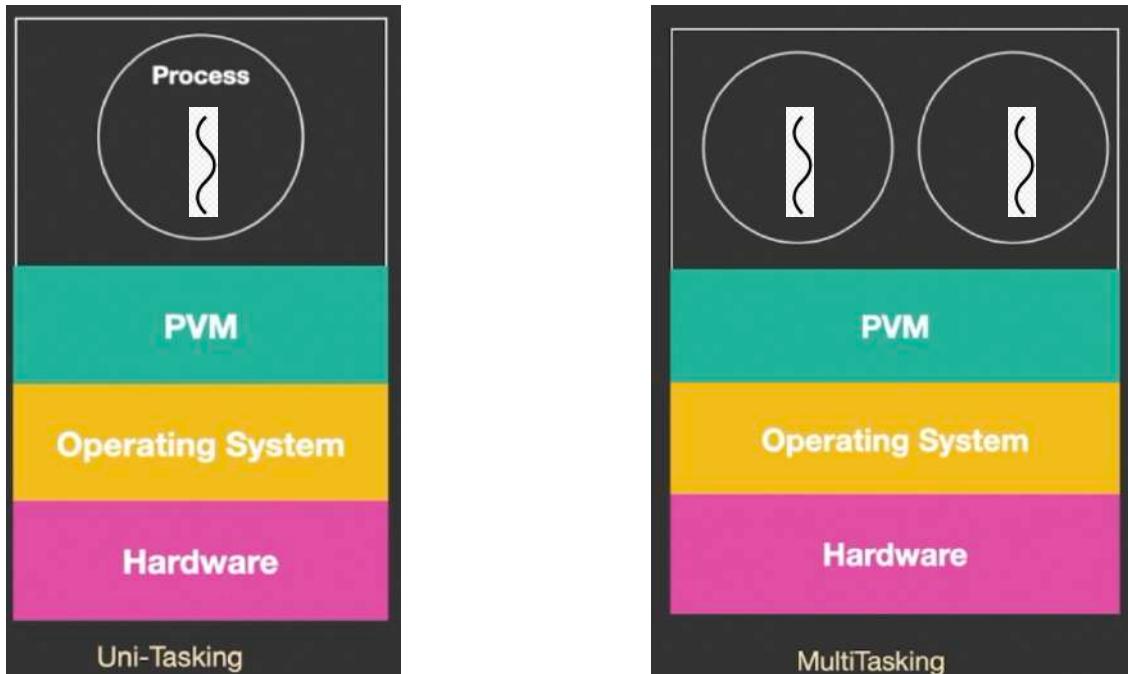
class Child(Parent):
    def show(self):#must override show as show method is declared as abstract in parent
        print('Show from Child')

c = Child
c.show()#calling method
c.display()
```

- Both are abstract it is nothing but interface. One as abstract and one as concrete
- Interface is also written in same way but all the methods are abstract.
- It is directly not available in python it is available through a module ABC

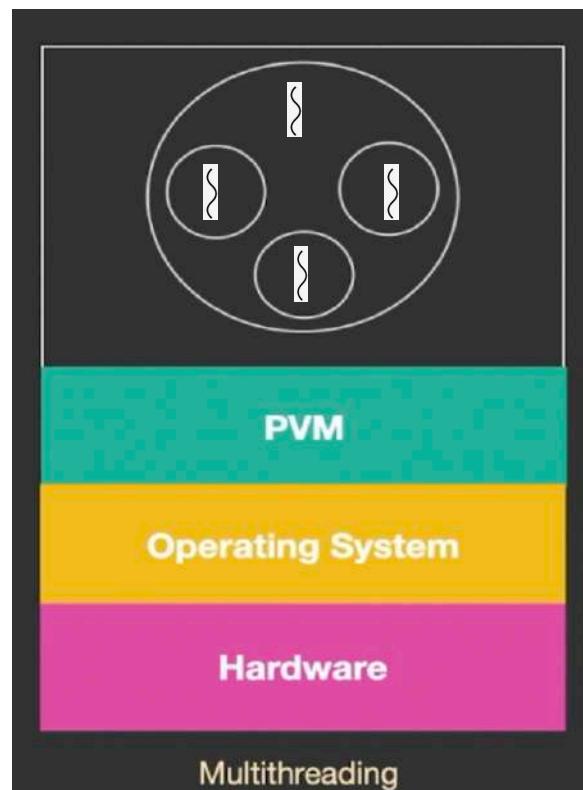
## Introduction to Multithreading

- When we run a program , the running program is called '**Process**'
- A running program(process) will have its own flow of control
- When a **single process** is running then its called "**uni-tasking**"
- When **more than one process** is running then its called "**Multitasking**"



- If two programs are running simultaneously then these two programs will have its own flow of control
- Hardware( CPU) will execute some lines of program 1 then some lines of program 2 then again 1 then 2 so on... this gives us the impression that they are running simultaneously but only one core I.e, CPU will execute these programs at a time

- **Multithreading** is same as multitasking but the difference is , in a main program they are sub independent program that run simultaneously and each sub program is having its own flow control.
- A process is a heavy process , but a thread is a light weighted process because they are children of main program.



## Advantages of Multithreading :

- We can create animations
- We can create games
- Chatting between multiple users
- Providing web services for multiple clients connected to the server at a time

## Creating a thread

There are two ways to create a thread.

1. Using function( method )
2. Using a class.

### Using function

- Suppose we want to print numbers that are 65 , 66 .....90 . These numbers are ascii code of alphabets. We use for loop for this

```
for i in range(65,69):  
    print ( i )
```

- Now we will create a thread for this we have to import threading module

```
from threading import *  
  
def display():  
    for i in range(65,91):  
        print(chr(i))  
  
  
t = Thread(target=display, name='Alphabets')  
t.start()  
for i in range(65,91):  
    print(i)  
t.join()
```

- It will create a thread using the function display to run we use t.start()

First create a thread

then start a thread

- t.join() it will wait for the main program

- Two thread run simultaneously it will give the mixed thread ( Alphabets and numbers )
- The output will not get same always
- If you want them to execute slowly. Then you can import time and use sleep
- It will slow down the program

`sleep()`

## With class

- In this we don't have to create objects of the thread

```
from threading import *
from time import *

class Alphabets(Thread):
    def run(self):
        for i in range(65,91):
            print(chr(i))
            sleep(1)

t = Alphabets()
t.start()
for i in range(65,91):
    print(i)
    sleep(1)

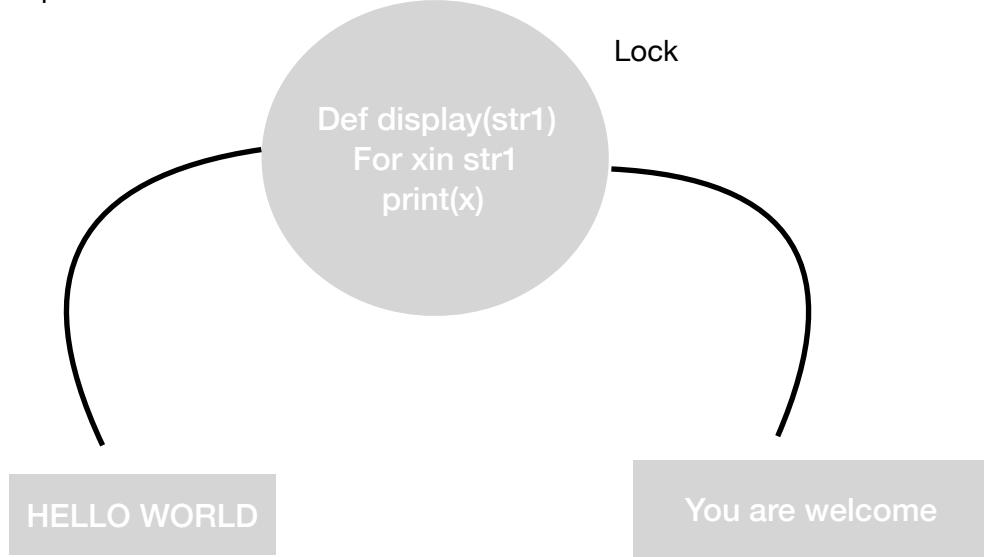
t.join()
```

## Thread Synchronisation(mutex)

If you have more than thread they may access to different resources then they may enter into race condition

And if we want to synchronise then we have to use mutex and semaphore.

Example:



Thread 1

Thread2

This will take HELLO WORLD to str1 and this will be taken as single single alphabets

And sending thread 2 also in the same function

Program:

```
from threading import *
from time import *

def display(str1):
    for x in str1:
        print(x)

t1 = Thread(target=display, args=('HELLO WORLD',))
t2 = Thread(target=display, args=('you are welcome',))

t1.start()
t2.start()

t1.join()
t2.join()
```

HELLO is broken down the output is mixed

- To print Hello world and you are welcome as separate string when this is mixed like this then it is called as race condition

**Mutex:** is a lock or variable that is used as lock and if once the thread is using display function it should put a lock and use display function and other thread is not enter into race condition actual thread should not start start using it

- And second thread use this display function only after first thread has finished it's work.

### So How it is possible?

- We will create a lock called mutex
- So when the first thread has put the lock then the second thread has to wait. When the lock is released by first thread then the second thread can use it.

### Program:

```
from threading import *
from time import *

def display(str1):
    l.acquire()
    for x in str1:
        print(x)
    l.release()

l = Lock()

t1 = Thread(target=display, args=('HELLO WORLD',))
t2 = Thread(target=display, args=('you are welcome',))

t1.start()
t2.start()

t1.join()
t2.join()
```

And the sleep function works slowly in discipline

```
from threading import *
from time import *

def display(str1):
    l.acquire()
    for x in str1:
        print(x)
        sleep(1)
    l.release()

l = Lock()

t1 = Thread(target=display, args=('HELLO WORLD',))
t2 = Thread(target=display, args=('you are welcome',))

t1.start()
t2.start()

t1.join()
t2.join()
```

## Semaphores

- Semaphore is a variable whose value is one , its value decreases on acquire function to zero(lock) , and increases when release
- Semaphores are same as lock but one difference is that in semaphore 2 threads are allowed inside function at a time
- Thus, semaphores allow multiple threads
- If semaphore value is one then one thread will enter the function , if two then 2 values will enter
- If its value is zero no thread is allowed inside function , if the value is non-zero then that no.of threads are allowed inside the function
- If they are many threads then they'll wait in a queue
- Below is the program that allows 2 threads to enter a function at the same time

```
from threading import *
from time import *

def display(str1):
    l.acquire()
    for x in str1:
        print(x)
        sleep(1)
    l.release()

l = Semaphore(2)

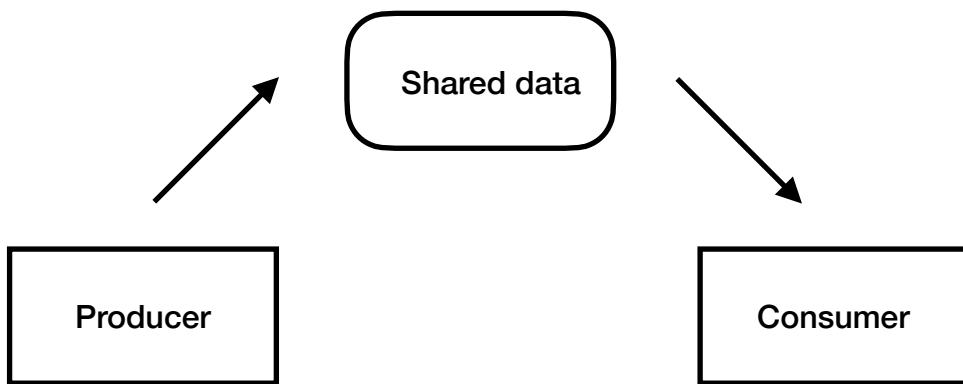
t1 = Thread(target=display, args=('HELLO WORLD',))
t2 = Thread(target=display, args=('you are welcome',))
t3 = Thread(target=display, args=('0123456789',))

t1.start()
t2.start()
t3.start()

t1.join()
t2.join()
t3.join()
```

## InterProcess Communication

- Two or more processors communicating with each other is said inter Process Communication but in this we are having threads so this is inter Process thread Communication
- Lets take example



- Producer will produce the data and consumer will consumes the data
- If you want that one by one then we use flag . Suppose if flag = false then it is producers turn after putting the value the flag will be true that means now its consumers turn .
- Consumer will consume that data and make the flag = false
- Shared data have two methods that is put ( ) and get ( )
- put( ) is for producer and get( ) is for consumer
- They will exchange term by flag value
- Whenever producer is putting the value first it should lock it then put the value and same goes for consumer whenever consumer is consuming the value first it should put a lock and then consumes the value

```
class MyData:
    def __init__(self):
        self.data=0
        self.flag=False
        self.lock= Lock()

    def put(self,d):
        while self.flag==False:
            pass
        self.lock.acquire()
        self.data = d
        self.flag = True
        self.lock.release()

    def get(self):
        while self.flag==True:
            pass
        self.lock.acquire()
        x = self.data
        self.flag = False
        self.lock.release()

def producer(data):
    i = 1
    while True:
        data.put(i)
        print('Producer:',i)
        i += 1

def consumer(data):
    while True:
        x = data.get()
        print('Consumer:',x)

data = MyData()
t1 = Thread(target=lambda:producer(data))
t2 = Thread(target=lambda:consumer(data))

data = MyData()
t1 = Thread(target=lambda:producer(data))
t2 = Thread(target=lambda:consumer(data))

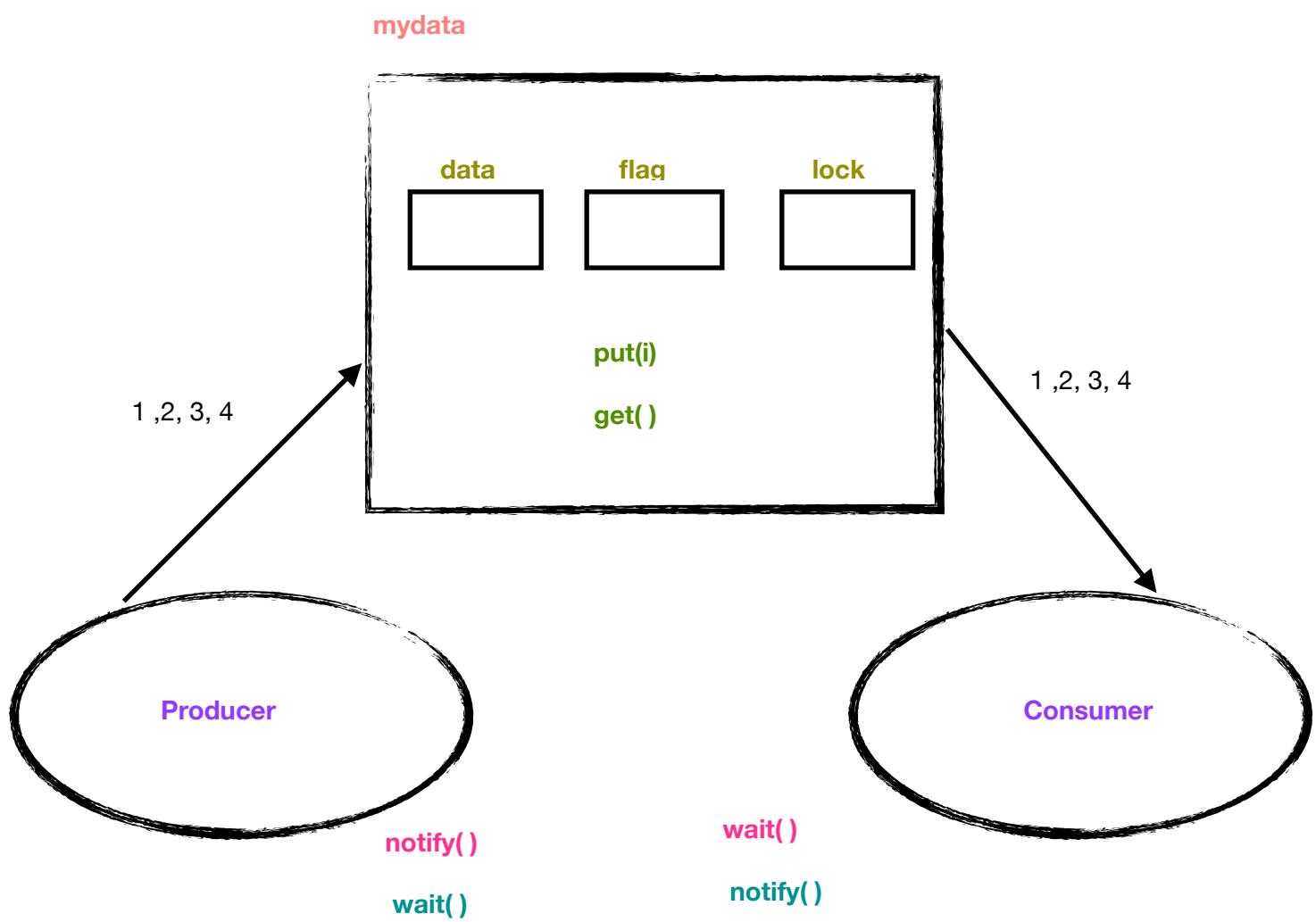
t1.start()
t2.start()

t1.join()
t2.join()
```

## Inter Process Communication using Conditions

- We use to use flag and lock together for synchronisation in mydata but here instead of using flag and lock together we use **conditions** variable for inter process communication.
- Working of condition variable :
  - Producer will **wait** to acquire the lock upon the data
  - Once producer gets the lock it'll lock the data object and write down data inside data object by using **put(i)**
  - Consumer will wait until it get its turn , when it get its turn it'll lock the data object and get the data by using **get( )** method
  - Producer and consumer exchange their turn one by one
  - If producer is producing **consumer** will **wait** when producer is done producing it'll **notify( )** waiting threads ,while consumer is consuming data **producer** will **wait** , once consumer finishes its work it'll **notify producer** saying it's your turn now.

Illustrated using a diagram , here flag and lock will be replaced by conditions



## Program :

```
from threading import *
from time import *

class MyData:
    def __init__(self):
        self.data=0
        self.cv = Condition()

    def put(self,d):
        self.cv.acquire()
        self.cv.wait(timeout=0)
        self.data = d
        self.cv.notify()
        self.cv.release()
        sleep(1)

    def get(self):
        self.cv.acquire()
        self.cv.wait(timeout=0)
        x = self.data
        self.cv.notify()
        self.cv.release()
        sleep(1)
        return x

    def producer(data):
        i = 1
        while True:
            data.put(i)
            print('Producer:',i)
            i += 1

    def consumer(data):
        while True:
            x = data.get()
            print('Consumer:',x)

data = MyData()
t1 = Thread(target=lambda:producer(data))
t2 = Thread(target=lambda:consumer(data))

t1.start()
t2.start()

t1.join()
t2.join()
```

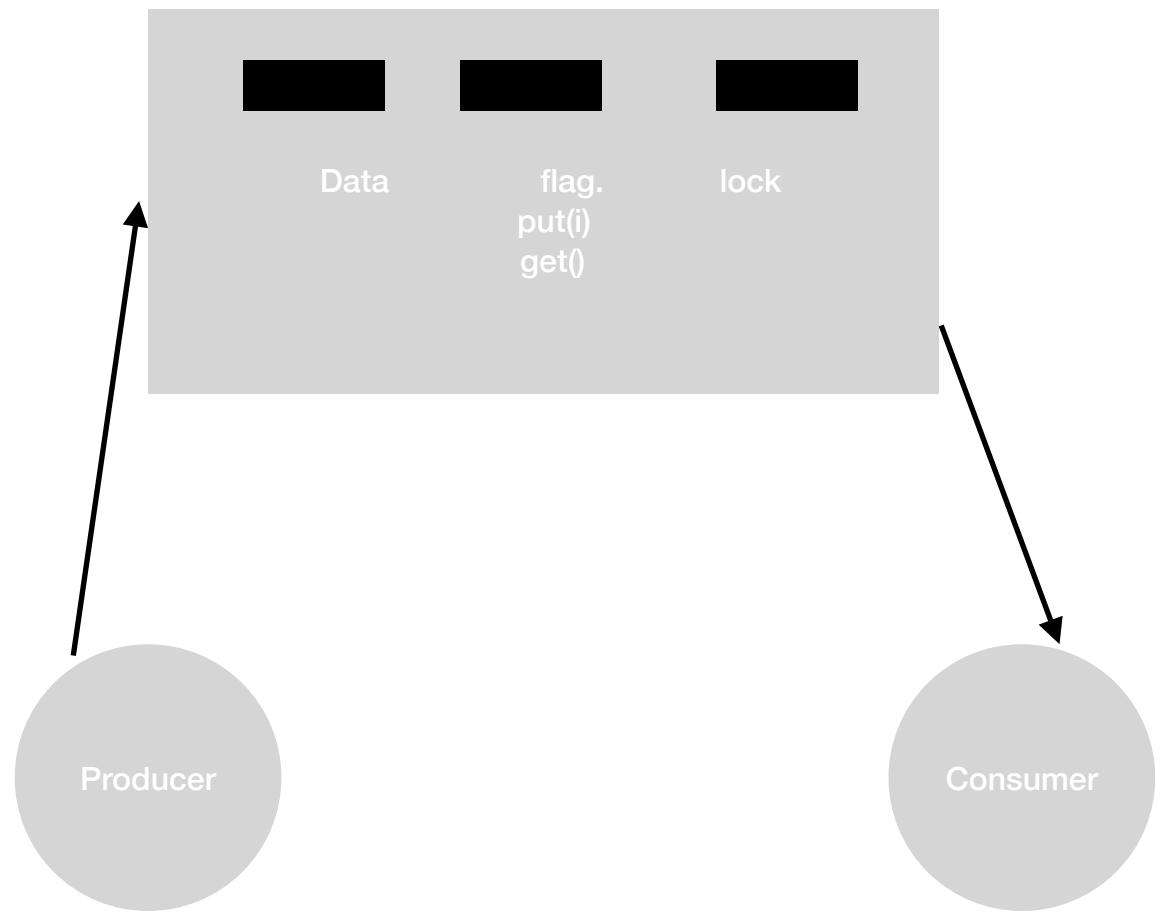
# importing required module and creating a class

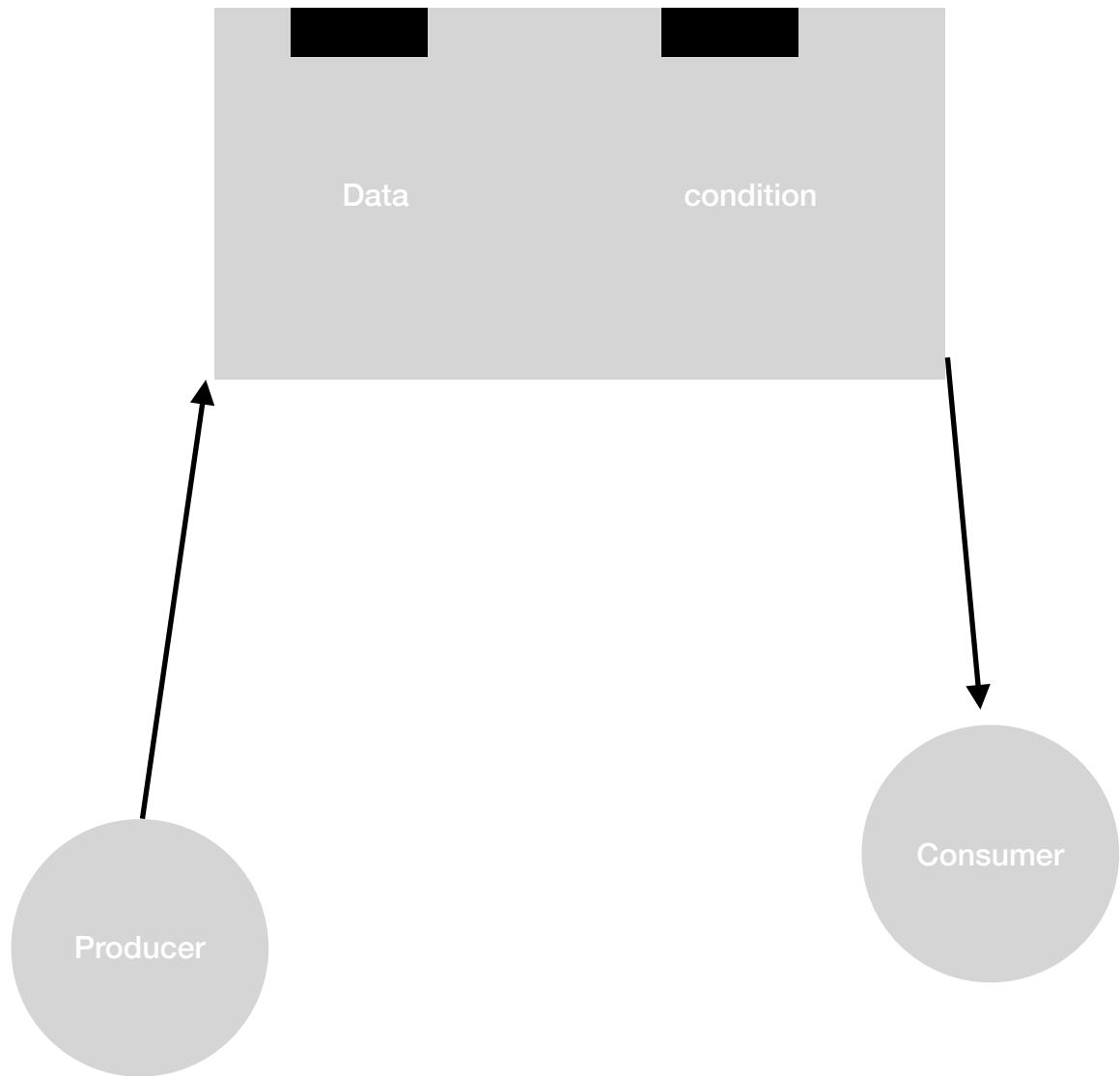
# creating put and get method so, that if producer is producing consumer should wait and vice versa

# initiating threads and returning the result



## Inter process communication using queue





- In data we can keep only value but in queue we can keep multiple values.
- We give producer for producing the data and consumer to consume the data
- The queue has methods like put() and get()

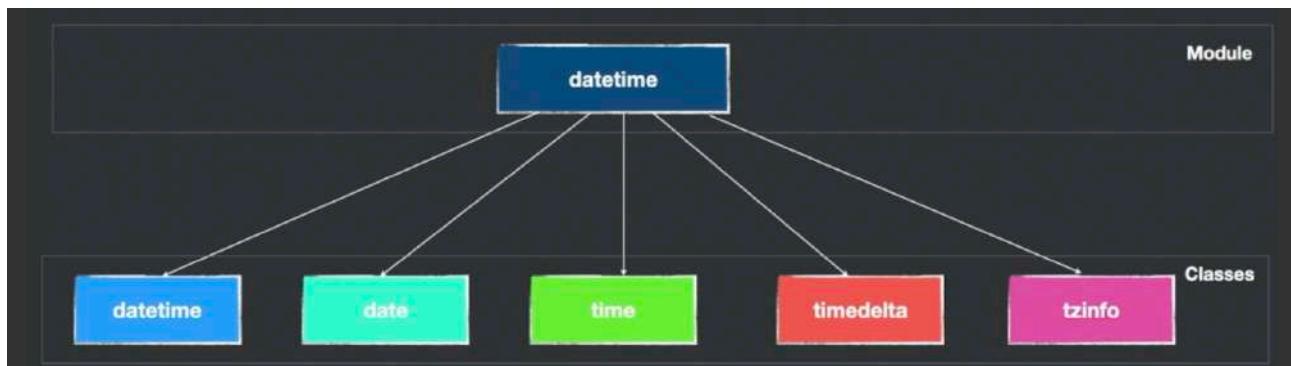
```
class MyData:  
    def __init__(self):  
        self.data=0  
        self.cv = Condition()  
  
    def put(self,d):  
        self.cv.acquire()  
        self.cv.wait(timeout=0)  
        self.data = d  
        self.cv.notify()  
        self.cv.release()  
        sleep(1)  
  
    def get(self):  
        self.cv.acquire()  
        self.cv.wait(timeout=0)  
        x = self.data  
        self.cv.notify()  
        self.cv.release()  
        sleep(1)  
        return x
```

## Date and Time

- Python is having built in modules that are related to date and time they are **datetime, time and calendar**

### [datetime module](#)

- This module contains classes like **datetime , date , time , time delta , tzinfo**



- **datetime** - d / m / y , hours : minutes : seconds : millisecond it also gives timezone
- **date** - date / month / year
- **time** - hours : minutes : seconds : millisecond
- **timedelta** - difference between 2 times , d2 - d1 it also gives day , hour, minutes , second
- **tzinfo**. - IST ,GMT

- If you want know the **current date and time** we have many functions for that
- The starting point for time for any watch is taken as **1 jan 1970** , this is also known as **epoch time** .
- The **time.time( )** will give you the time elapsed from epoch time

```
>>> import time
>>> time.time()
1647342625.926307
>>> |
```

- The **time.localtime( )** will give you all the values of current date and time including year , month , monthly , weekday etc.

```
>>> time.localtime()
time.struct_time(tm_year=2022, tm_mon=3, tm_mday=15, tm_hour=16, tm_min=41, tm_sec=3, tm_wday=1, tm_yday=74, tm_isdst=0)
>>>
>>> lt = time.localtime()
>>> type(lt)
<class 'time.struct_time'>
>>>
>>> lt.tm_year
2022
>>> lt.tm_mday
15
>>> lt.tm_hour
16
>>> |
```

- **lt** is variable for local time , type of **lt** is structured time , you can also get the particular data of your choice like year , day and hour etc...

- Some other function to know current date and time are

```
>>> import time
>>> import datetime
>>>
>>> time.ctime()
'Tue Mar 15 16:50:29 2022'
>>>
>>> datetime.datetime.now()
datetime.datetime(2022, 3, 15, 16, 51, 18, 350843)
>>> dt = datetime.datetime(2022, 3, 15, 16, 51, 18, 350843)
>>> dt.year
2022
>>> dt.month
3
>>> dt.day
15
>>> type(dt)
<class 'datetime.datetime'>
>>>
>>> dt = datetime.datetime.today()
>>> dt
datetime.datetime(2022, 3, 15, 16, 52, 58, 96412)
>>>
```

**time.ctime()** - string form of current date and time

**time.now()** - will give object of date and time only

**time.today()** - same as now()

## Create date object and time object

### datetime

- datetime class
- date class
- time class

### datetime :

```
>>> import datetime
>>>
>>> dt1 = datetime.datetime(2010,1,1,10,10,10)
>>> dt2 = datetime.datetime(day=1,month=1,year=2011,hour=10,minute=10,second=10)
>>> dt1
datetime.datetime(2010, 1, 1, 10, 10, 10)
>>> dt2
datetime.datetime(2011, 1, 1, 10, 10, 10)
>>> dt2 - dt1
datetime.timedelta(days=365)
>>> dt2 > dt1
True
>>> dt2 < dt1
False
>>> |
```

- For this we have to import the datetime module , it contains both date and time object
- We have taken example as dt1 and dt2 we can also give keyword argument so we don't have to give time in ordered form
- In this we can subtract dt2-dt1 , we can also find which is greater( > ) and lesser ( < ) between dt1 and dt2 it will return it in boolean form

### date and time :

- A date object represents a date (year, month and day).

```
>>> import datetime
>>>
>>> dat1 = datetime.date(2015,1,1)
>>> dat2 = datetime.date(2016,1,1)
>>>
>>> dat2 - dat1
datetime.timedelta(days=365)
>>> dat2 > dat1
True
>>>
>>> tim1 = datetime.time(10,10,10)
>>> tim2 = datetime.time(11,11,11)
>>>
>>> tim2 - tim1
Traceback (most recent call last):
File "<pyshell#11>", line 1, in <module>
    tim2 - tim1
TypeError: unsupported operand type(s) for -: 'datetime.time' and 'datetime.time'
>>> tim2 > tim1      ^
True
>>> |
```

- A time object instantiated from the time class represents the local time.
- We can subtract dat2-dat1 but not tim2-tim1
- We can find both greater and less than for both date and time
- By using this classes we can create specific date and time



## Formatting date and time

- We can convert date time information into a string and vice versa
- The function **strftime()** convert **datetime** Information into **string** , strftime() is **instance method**.
- **strptime()** will convert a **string** time into **datetime** , strptime() **static method** inside datetime class
- For conversion they require **control characters** , the following tables contains control characters

%-y	Year without century as a decimal number.
%Y	Year with century as a decimal number.
%H	Hour (24-hour clock) as a zero added decimal number.
%-H	Hour (24-hour clock) as a decimal number.
%I	Hour (12-hour clock) as a zero added decimal number.
%-I	Hour (12-hour clock) as a decimal number.
%p	Locale's AM or PM.
%M	Minute as a zero added decimal number.
%-M	Minute as a decimal number.
%S	Second as a zero added decimal number.

<code>%a</code>	Abbreviated weekday name.
<code>%A</code>	Full weekday name.
<code>%w</code>	Weekday as a decimal number.
<code>%d</code>	Day of the month as a zero added decimal.
<code>%-d</code>	Day of the month as a decimal number.
<code>%b</code>	Abbreviated month name.
<code>%B</code>	Full month name.
<code>%m</code>	Month as a zero added decimal number.
<code>%-m</code>	Month as a decimal number.
<code>%y</code>	Year without century as a zero added decimal number.

<code>%-S</code>	Second as a decimal number.
<code>%f</code>	Microsecond as a decimal number, zero added on the left.
<code>%z</code>	UTC offset in the form +HHMM or -HHMM.
<code>%Z</code>	Time zone name.
<code>%j</code>	Zero padded decimal number
<code>%U</code>	Week number of the year .All days in a new year preceding the first Sunday are considered to be in week 0.
<code>%W</code>	Week number of the year . All days in a new year preceding the first Monday are considered to be in week 0.
<code>%c</code>	Appropriate date and time representation
<code>%x</code>	Appropriate date representation
<code>%X</code>	Appropriate time representation
<code>%%</code>	Single % character

- Lets see this with few examples
- Converting datetime to string ( `strftime()` )

```
>>> from datetime import *
>>>
>>> dt1 = datetime(2010,1,1,10,20,30)
>>> dt1
datetime.datetime(2010, 1, 1, 10, 20, 30)
```

%d - date  
%B - month name  
%A - day name  
%Y - year in 4 digit  
%H - hours  
%M - minutes  
%S - seconds

```
>>> dt1.strftime('%d %B, %A %Y. %H:%M:%S')
'01 January, Friday 2010. 10:20:30'
>>>
```

```
>>> dt1.strftime('%-d %b, %a %y. %H:%M:%S')
'1 Jan, Fri 10. 10:20:30'
>>>
```

- Converting string to datetime ( `strptime( )` )

```
>>> str1 = '10 Jan 2015, 10:15:30'  
>>> type(str1)  
<class 'str'>  
>>> dt2=datetime.strptime(str1,'%d %b %Y, %H:%M:%S')  
>>> dt2  
datetime.datetime(2015, 1, 10, 10, 15, 30)  
>>>
```

## Time delta

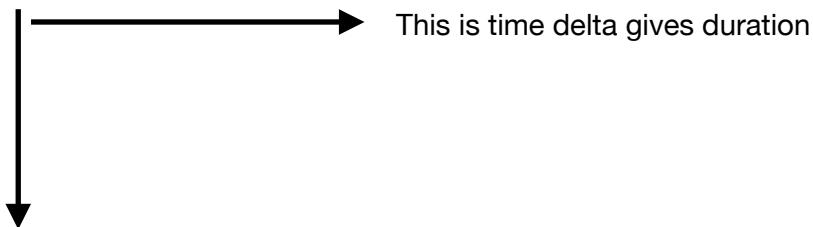
Time delta is the class of datetime package represented for duration /period of time days second microseconds

Creating two date time objects

```
dt1=datetime(2010, 10, 20, 30)
```

```
dt2=datetime(2011, 10, 20, 30)
```

```
td=dt2-dt1
```



365 days

```
>>> from datetime import *
>>>
>>> dt1 = datetime(2010,10,1,10,20,30)
>>> dt2 = datetime(2011,10,1,10,20,30)
>>>
>>> dt2-dt1
datetime.timedelta(days=365)
>>> td = dt2 -dt1
>>> td
datetime.timedelta(days=365)
>>> divmod(td.days,30)
(12, 5)
>>>
>>> divmod(td.days,7)
(52, 1)
>>>
>>> td1 = timedelta(31)
>>> dt3 = datetime.now()
>>> dt3
datetime.datetime(2022, 3, 16, 16, 34, 2, 656220)
>>> dt3 +td1
datetime.datetime(2022, 4, 16, 16, 34, 2, 656220)
>>> dt3+td
datetime.datetime(2023, 3, 16, 16, 34, 2, 656220)
>>> |
```

## Calendar Module

- The methods of **calendar module** are

```
class calendar.LocaleTextCalendar(firstweekday=0, locale=None)
class calendar.LocaleHTMLCalendar(firstweekday=0, locale=None)
calendar.setfirstweekday(weekday)
calendar.firstweekday()
calendar.isleap(year)
calendar.leapdays(y1, y2)
calendar.weekday(year, month, day)
calendar.weekheader(n)
calendar.monthrange(year, month)
calendar.monthcalendar(year, month)
calendar.prmonth(theyear, themonth, w=0, l=0)
calendar.month(theyear, themonth, w=0, l=0)
calendar.timegm(tuple)
```

```
>>> from calendar import *
>>> firstweekday()
0
>>> isleap(2024)
True
>>> leapdays(2000,2022)
6
>>> weekday(2000,1,1)
5
>>> weekday(2022,3,16)
2
>>> prmonth(2022,3)
      March 2022
Mo Tu We Th Fr Sa Su
 1  2  3  4  5  6
 7  8  9 10 11 12 13
14 15 16 17 18 19 20
21 22 23 24 25 26 27
28 29 30 31
```

- The **class calendar** methods are

```
class calendar.Calendar(firstweekday=0)
    ↴
    iterweekdays()
    itermonthdates(year, month)
    itermonthdays(year, month)
    itermonthdays2(year, month)
    itermonthday3(year, month)
    itermonthday4(year, month)
    monthdatescalendar(year, month)
    monthdatescalendar(year, width=3 )
    monthdayscalendar(year, month)
    yeardatescalendar(year, width=3 )
    yeardayscalendar(year, width=3 )
    yeardays2calendar(year, width=3 )
```

```
>>> from calendar import *
>>> c = Calendar()
>>> c
<calendar.Calendar object at 0x7f79b052e8b0>
>>> for i in c.iterweekdays():
    print(i)
```

```
0
1
2
3
4
5
6
```

```
>>> for i in c.itermonthdates(2022,2):
    print(i)
```

```
2022-01-31
2022-02-01
2022-02-02
2022-02-03
2022-02-04
2022-02-05
2022-02-06
2022-02-07
2022-02-08
2022-02-09
2022-02-10
2022-02-11
2022-02-12
2022-02-13
2022-02-14
2022-02-15
2022-02-16
2022-02-17
2022-02-18
2022-02-19
2022-02-20
2022-02-21
```

```
>>> c.monthdatescalendar(2022,3)
[[datetime.date(2022, 2, 28), datetime.date(2022, 3, 1), datetime.date(2022, 3, 2), datetime.date(2022, 3, 3), datetime.date(2022, 3, 4), datetime.dat
e(2022, 3, 5), datetime.date(2022, 3, 6)], [datetime.date(2022, 3, 7), datetime.date(2022, 3, 8), datetime.date(2022, 3, 9), datetime.date(2022, 3, 10
), datetime.date(2022, 3, 11), datetime.date(2022, 3, 12), datetime.date(2022, 3, 13)], [datetime.date(2022, 3, 14), datetime.date(2022, 3, 15), dateti
me.date(2022, 3, 16), datetime.date(2022, 3, 17), datetime.date(2022, 3, 18), datetime.date(2022, 3, 19), datetime.date(2022, 3, 20)], [dateti
me(2022, 3, 21), datetime.date(2022, 3, 22), datetime.date(2022, 3, 23), datetime.date(2022, 3, 24), datetime.date(2022, 3, 25), datetime.date(2022,
3, 26), datetime.date(2022, 3, 27)], [datetime.date(2022, 3, 28), datetime.date(2022, 3, 29), datetime.date(2022, 3, 30), datetime.date(2022, 3, 31),
datetime.date(2022, 4, 1), datetime.date(2022, 4, 2), datetime.date(2022, 4, 3)]]
```

## Text calendar

```
class calendar.TextCalendar(firstweekday=0)

formatmonth(theyear, themonth, w=0, l=0)
prmonth(theyear, themonth, w=0, l=0)
formatyear(theyear, w=2, l=1, c=6, m=3)
pryear(theyear, w=2, l=1, c=6, m=3)
```

```
>>> from calendar import *
>>> c = TextCalendar()
>>> c
<calendar.TextCalendar object at 0x7f978d3ee8b0>
>>> c.prmonth(2022,3)
      March 2022
Mo Tu We Th Fr Sa Su
 1  2  3  4  5  6
 7  8  9 10 11 12 13
14 15 16 17 18 19 20
21 22 23 24 25 26 27
28 29 30 31
>>> |
```

```
>>> c.pryear(2022)
2022
```

I

January							February							March						
Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su
1	2	3	4	5	6		1	2	3	4	5	6		7	8	9	10	11	12	13
3	4	5	6	7	8	9	7	8	9	10	11	12	13	7	8	9	10	11	12	13
10	11	12	13	14	15	16	14	15	16	17	18	19	20	14	15	16	17	18	19	20
17	18	19	20	21	22	23	21	22	23	24	25	26	27	21	22	23	24	25	26	27
24	25	26	27	28	29	30	28							28	29	30	31			
31																				

April							May							June						
Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su
1	2	3			1		1	2	3	4	5			6	7	8	9	10	11	12
4	5	6	7	8	9	10	2	3	4	5	6	7	8	6	7	8	9	10	11	12
11	12	13	14	15	16	17	9	10	11	12	13	14	15	13	14	15	16	17	18	19
18	19	20	21	22	23	24	16	17	18	19	20	21	22	20	21	22	23	24	25	26

```
>>> c.formatmonth(2022,3)
```

```
' March 2022\nMo Tu We Th Fr Sa Su\n 1 2 3 4 5 6\n 7 8 9 10 11 12 13\n14 15 16 17 18 19 20\n21 22 23 24 25 26 27\n28 29 30 31\n'
```

```
>>> for i in c.formatmonth(2022,3).split('\n'):
    print(i)
```

March 2022

Mo	Tu	We	Th	Fr	Sa	Su
1	2	3	4	5	6	
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31			

- **HTML calendar :** it gives Information in html , used mostly in web development

```
class calendar.HTMLCalendar(firstweekday=0)

    formatmonth(theyear, themonth, withyear=True)

    formatyear(theyear, width=3)
        -->
    formatyearpage(theyear, width=3, css='calendar.css')
```

```
>>> c = HTMLCalendar()
>>> c.formatmonth(2022,3)
<table border="0" cellpadding="0" cellspacing="0" class="month">\n<tr><th colspan="7" class="month">March 2022</th></tr>\n<tr><th class="mon">Mon</th><th class="tue">Tue</th><th class="wed">Wed</th><th class="thu">Thu</th><th class="fri">Fri</th><th class="sat">Sat</th><th class="sun">Sun</th></tr>\n<tr><td class="noday">&nbsp;</td><td class="tue">1</td><td class="wed">2</td><td class="thu">3</td><td class="fri">4</td><td class="sat">5</td><td class="sun">6</td></tr>\n<tr><td class="mon">7</td><td class="tue">8</td><td class="wed">9</td><td class="thu">10</td><td class="fri">11</td><td class="sat">12</td><td class="sun">13</td></tr>\n<tr><td class="mon">14</td><td class="tue">15</td><td class="wed">16</td><td class="thu">17</td><td class="fri">18</td><td class="sat">19</td><td class="sun">20</td></tr>\n<tr><td class="mon">21</td><td class="tue">22</td><td class="wed">23</td><td class="thu">24</td><td class="fri">25</td><td class="sat">26</td><td class="sun">27</td></tr>\n<tr><td class="mon">28</td><td class="tue">29</td><td class="wed">30</td><td class="thu">31</td><td class="noday">&nbsp;</td><td class="noday">&nbsp;</td></tr>\n</table>\n'
```

## Database Terminology

- Database is stored in the form of tables

### **Relation :**

- It is also said as RDBMS ( Relational database management system )because data is stored as a form of relation( table)
- Data is entered in the table as per schema
- Firstly we have to design the structure of a table

### **Fields :**

- Columns are called as fields .

### **Record :**

- Row of the table is said as record or also said as tuple

### **Primary Key :**

- It is the unique column with no null value . It will be easy to search using primary key( which will not have any duplicate value )
- If you want to know the name or department you can search it by using primary key .

### **Relationship :**

One table which is referring to the other table

### **Foreign :**

- Two table have the relation of one foreign key and one primary key .
- Foreign key inside the table can be primary key in another table vice versa

### **Constraint :**

- Not null is the constraint ( any row should not be null ) unique is constraint and primary key is also constraint because in table Dept number should be present in another table Deptno. We can only use the table if the values present in the table 1 should be present in table 2.

## Downloading SQLite

<https://www.sqlite.org/download.html>

<https://www.sqlite.org/2022/sqlite-dll-win64-x64-3380100.zip>

# Introduction to SQL(DDL, DML)

## DDL

CREATE  
DROP  
ALTER  
TRUNCATE  
RENAME

## DML

INSERT  
DELETE  
UPDATE

## Query

SELECT  
FROM  
WHERE

### Types of commands available in sql

Divided into three

- 1) **DDL:** data description language
- 2) **DML:** data manipulation language
- 3) **DCL:** data control language

Data can be retrieved using sql commands

### Data type of SQLite

**Null :** means value is not present. If value is not present we write null

**Int :** numeric value without decimal

**Real/float/numeric->** how many digits before and after decimal

**Text:** Text is nothing but just like string

**Var char:** anything can be used as variable or character

**BLOB:** binary large object

Ex : storing image , video, audio in the database

How to create table and insert data

### Steps to create table and insert data:

- 1) create database
- 2) Create tables
- 3) Define keys
- 4) Insert data
- 5) Query data

Students				Dept	
Roll	name	City	Deptno	Deptno.	Name
1	Ajay	Delhi	10		
2	Vijay	Kolkata	10		
3	Ajay	Mumbai	20		
4	Ramesh	Delhi	30		
5	Suneeta	Lucknow	40		
6	Anita	Kolkata	30		
7	Raj	Jaipur	30		
8	Ali	Lucknow	40		
9	Michael	Gochin	10		
10	Pavan	Vijaywada	20		
11	Suraj	Hyderabad	10		
12	Altuf	Bangaluru	40		
13	Ravi	Indore	20		
14	Verma	Delhi	20		
15	Sharma	Vizag	10		

Sql queries to create database

( sqlite Dml query)

.open univ.bd # opening the database

.tables

Create table dept(deptno integer primary key not null unique, name text); .tables #creating a table for department

Create table students(roll integer primary key, name text, deptno integer, foreign key(dept no) reference dept (dept));

.tables # creating a table for students

(Sqlite dml queries)

Insert into dept values(10, 'CSE');# inserting student info dept table

Insert into dept values(20, 'ECE');

Insert into dept values(30, 'CIVIL');

Insert into dept values(40, 'Mech')

Insert into dept values(name, dept no)values('chem', 50);

(Select query)

select\*from dept; here \* brings the info inside the dept table

pragma foreign\_key=ON

....>;

Insert into students values(1, 'ajay', 'delhi', 10);

Insert into students values(2, 'vijay', 'delhi', 10);

Select \* from student where city='delhi' or city='jaipur'

## SQL ( select from and where clause )

- SQL also known as **Data Retrieval Language** ( DRL )
- The clauses used for writing queries and retrieval of data are **select , from , join , where ,order by , group by , having.**
- Lets consider the below DataBase

Students			
Roll	name	City	Deptno
1	Ajay	Delhi	10
2	Vijay	Kolkata	10
3	Ajay	Mumbai	20
4	Ramesh	Delhi	30
5	Suneeta	Lucknow	40
6	Anita	Kolkata	30
7	Raj	Jaipur	30
8	Ali	Lucknow	40
9	Michael	Cochin	10
10	Pavan	Vijaywada	20
11	Suraj	Hyderabad	10
12	Altaf	Bangaluru	40
13	Ravi	Indore	20
14	Verma	Delhi	20
15	Sharma	Vizag	10

Dept	
Deptno	Name
10	CSE
20	ECE
30	Civil
40	Mech

## Select

- Select is used for **filtering Columns**
- Along with select we must use from also without from we cannot create a query
- It is used for **specifying the columns** of a relation that we want to retrieve

## from

- It is used for **specifying a relation name**
- For instance, from the given data we want student Database then we use the following query

```
sqlite> .open univ
sqlite> select name from students;
Ajay
Vijay
Ajay
Ramesh
Suneeta
Anita
Raj
Ali
Michael
Pavan
Suraj
Altaf
Ravi
Verma
Sharma
```

- If we want to retrieve **all the columns** we write **\*** in query

```
sqlite> select * from students;
1|Ajay|Delhi|10
2|Vijay|Kolkata|10
3|Ajay|Mumbai|20
4|Ramesh|Delhi|30
5|Suneeta|Lucknow|40
6|Anita|Kolkata|30
7|Raj|Jaipur|30
8|Ali|Lucknow|40
9|Michael|Cochin|10
10|Pavan|Vijaywada|20
11|Suraj|Hyderabad|10
12|Altaf|Bangaluru|40
13|Ravi|Indore|20
14|Verma|Delhi|20
15|Sharma|Vizag|10
sqlite> _
```

- From dept database we want every thing the following query is used

**Select \* from dept ;**

- If you don't want repetition then we can use distinct in query as follows

**Select distinct deptno from student ;**

```
sqlite> select deptno from students;
10
10
20
30
30
40
30
30
40
10
20
10
40
20
20
10
sqlite> select distinct deptno from students;
10
20
30
40
sqlite>
```

- If you are using **multiple column** then distinct will treat 2 rows as duplicate only if all the values are same and not just one common value .

## Where

- It is used for specifying conditions upon one or more fields / columns
- Basically it is used for **filtering rows**
- We can specify the condition in where clause using **< , <= , > , >= , = , < , >**

```
sqlite> select * from students where city='Delhi';
1|Ajay|Delhi|10
4|Ramesh|Delhi|30
14|Verma|Delhi|20
sqlite> select * from students where city<>'Delhi';
2|Vijay|Kolkata|10
3|Ajay|Mumbai|20
5|Suneeta|Lucknow|40
6|Anita|Kolkata|30
7|Raj|Jaipur|30
8|Ali|Lucknow|40
9|Michael|Cochin|10
10|Pavan|Vijaywada|20
11|Suraj|Hyderabad|10
12|Altaf|Bangaluru|40
13|Ravi|Indore|20
15|Sharma|Vizag|10
```

- If you have multiple conditions you can combine them with **AND , OR , NOT, LIKE , between\_and\_ , IN , NOT IN**

### - AND

```
sqlite> select * from students where deptno>=30 and city='Lucknow';
5|Suneeta|Lucknow|40
8|Ali|Lucknow|40
sqlite> select * from students where deptno>=30 and not (city='Lucknow');
4|Ramesh|Delhi|30
6|Anita|Kolkata|30
7|Raj|Jaipur|30
12|Altaf|Bangaluru|40
```

### - LIKE

```
sqlite> select * from students where name like 'A%';
1|Ajay|Delhi|10
3|Ajay|Mumbai|20
6|Anita|Kolkata|30
8|Ali|Lucknow|40
12|Altaf|Bangaluru|40
sqlite> select * from students where name like '%y';
1|Ajay|Delhi|10
2|Vijay|Kolkata|10
3|Ajay|Mumbai|20
sqlite> select * from students where name like '%m%';
4|Ramesh|Delhi|30
9|Michael|Cochin|10
14|Verma|Delhi|20
15|Sharma|Vizag|10
```

## - between\_and\_

```
sqlite> select * from students where deptno between 20 and 40;
3|Ajay|Mumbai|20
4|Ramesh|Delhi|30
5|Suneeta|Lucknow|40
6|Anita|Kolkata|30
7|Raj|Jaipur|30
8|Ali|Lucknow|40
10|Pavan|Vijaywada|20
12|Altaf|Bangaluru|40
13|Ravi|Indore|20
14|Verma|Delhi|20
sqlite> select * from students where roll between 5 to 10;
Error: near "to": syntax error
sqlite> select * from students where roll between 5 and 10;
5|Suneeta|Lucknow|40
6|Anita|Kolkata|30
7|Raj|Jaipur|30
8|Ali|Lucknow|40
9|Michael|Cochin|10
10|Pavan|Vijaywada|20
sqlite>
```

## - IN , NOT IN

```
sqlite> select * from students where city in ('Delhi','Jaipur');
1|Ajay|Delhi|10
4|Ramesh|Delhi|30
7|Raj|Jaipur|30
14|Verma|Delhi|20
sqlite> select * from students where city not in ('Delhi','Jaipur');
2|Vijay|Kolkata|10
3|Ajay|Mumbai|20
5|Suneeta|Lucknow|40
6|Anita|Kolkata|30
8|Ali|Lucknow|40
9|Michael|Cochin|10
10|Pavan|Vijaywada|20
11|Suraj|Hyderabad|10
12|Altaf|Bangaluru|40
13|Ravi|Indore|20
15|Sharma|Vizag|10
```

## SQL(join, group by and having clause)

**Order by clause:** we can display the content increasing as well as decreasing

- \* shows all the values

```
select * from dept order by dename; #order by clause here * output data
```

```
select * from dept order by dename desc; #here desc means decreasing department will be shown in decreasing order
```

```
select name from students order by name;
```

```
select name from students order by name desc;
```

```
select * from students order by name desc;
```

```
select * from students, dept;
```

```
select * from students, dept where students.deptno=dept.deptno;
```

**Join:** in a query we are retrieving the data from multiple table then we have to join the table.

If joined two tables each row of one table is joined with row of another table

But here it is multiplying a row with each row of another table so, it is nothing but Cartesian product

But join multiplies with only one row with same dept no.

```
select * from students join dept students.deptno=dept.deptno;# joining
```

```
select * from students S join dept D on S.deptno=D.deptno; # this is called renaming short method for joining
```

```
select *
from students S,dept D
where S.deptno=D.deptno;
```

```
select city from students group by city;# from clause
```

```
select count(*),city from students group by city;# grouping rows and also counting the number of students this is called as aggregate function
```

**Having:** shows which are having 2 or more count.

It is grouping the record by city but it can't show whole data

```
select count(*),city from students group by city having count(*)>2;
```

## SQL ( Aggregated Function and Set Operations )

- Aggregated function are used with **select clause** and also used by **group by** clause to work on groups if you don't use group by it'll work on entire collection
- Consider the following database

Students				Dept	
Roll	name	City	Deptno	Deptno	Name
1	Ajay	Delhi	10	10	CSE
2	Vijay	Kolkata	10	20	ECE
3	Ajay	Mumbai	20	30	Civil
4	Ramesh	Delhi	30	40	Mech
5	Suneeta	Lucknow	40		
6	Anita	Kolkata	30		
7	Raj	Jaipur	30		
8	Ali	Lucknow	40		
9	Michael	Cochin	10		
10	Pavan	Vijaywada	20		
11	Suraj	Hyderabad	10		
12	Altaf	Bangaluru	40		
13	Ravi	Indore	20		
14	Verma	Delhi	20		
15	Sharma	Vizag	10		

- From student database we are counting total no.of students , roll numbers , name , and distinct name (without duplicates)

```
sqlite> .open univ
sqlite> select * from students
...> ;
1|Ajay|Delhi|10
2|Vijay|Kolkata|10
3|Ajay|Mumbai|20
4|Ramesh|Delhi|30
5|Suneeta|Lucknow|40
6|Anita|Kolkata|30
7|Raj|Jaipur|30
8|Ali|Lucknow|40
9|Michael|Cochin|10
10|Pavan|Vijaywada|20
11|Suraj|Hyderabad|10
12|Altaf|Bangaluru|40
13|Ravi|Indore|20
14|Verma|Delhi|20
15|Sharma|Vizag|10
sqlite> select count(*) from students;
15
sqlite> select count(roll) from students;
15
sqlite> select count(name) from students;
15
sqlite> select count(distinct name) from students;
14
```

- From student database we are performing maximum, minimum ,sum , average function and we are also finding max rollno from different city's , performing similar operations in the below example

```
sqlite> select max(roll) from students;
15
sqlite> select min(roll) from students;
1
sqlite> select sum(roll) from students;
120
sqlite> select avg(roll) from students;
8.0
sqlite> select max(roll),city from students group by city;
12|Bangaluru
9|Cochin
14|Delhi
11|Hyderabad
13|Indore
7|Jaipur
6|Kolkata
8|Lucknow
3|Mumbai
10|Vijaywada
15|Vizag
sqlite> -
```

```
sqlite> select count(roll),city from students group by city;
1|Bangaluru
1|Cochin
3|Delhi
1|Hyderabad
1|Indore
1|Jaipur
2|Kolkata
2|Lucknow
1|Mumbai
1|Vijaywada
1|Vizag
sqlite> select sum(roll),city from students group by city;
12|Bangaluru
9|Cochin
19|Delhi
11|Hyderabad
13|Indore
7|Jaipur
8|Kolkata
13|Lucknow
3|Mumbai
10|Vijaywada
15|Vizag
sqlite>
```

- Set operations are **Union , Intersection , except**
- The rows or columns taken in set operations must be of **same type** i.e, if one set contains names the other should also contains names only if not well get an error
- Taking 2 database as and performing union , intersection between them

### - Union

```
sqlite> select * from students where city='Delhi';
1|Ajay|Delhi|10
4|Ramesh|Delhi|30
14|Verma|Delhi|20
sqlite> select * from students where city='Kolkata';
2|Vijay|Kolkata|10
6|Anita|Kolkata|30
sqlite> select * from students where city='Delhi' union select * from students where city='Mumbai';
1|Ajay|Delhi|10
3|Ajay|Mumbai|20
4|Ramesh|Delhi|30
14|Verma|Delhi|20
sqlite> select name from students where city='Delhi' union select roll,name from students where city='Mumbai';
Error: SELECTs to the left and right of UNION do not have the same number of result columns
sqlite> select name from students where city='Delhi' union select name from students where city='Mumbai';
Ajay
Ramesh
Verma
sqlite>
```

### - Intersection

```
sqlite> select name from students where city='Delhi' intersect select name from students where city='Mumbai';
Ajay
sqlite> select name from students where city='Delhi' except select name from students where city='Mumbai';
Ramesh
Verma
sqlite>
```

## SQL ( SubQuery and more DML)

Students			
Roll	name	City	Deptno
1	Ajay	Delhi	10
2	Vijay	Kolkata	10
3	Ajay	Mumbai	20
4	Ramesh	Delhi	30
5	Suneeta	Lucknow	40
6	Anita	Kolkata	30
7	Raj	Jaipur	30
8	Ali	Lucknow	40
9	Michael	Cochin	10
10	Pavan	Vijaywada	20
11	Suraj	Hyderabad	10
12	Altaf	Bangaluru	40
13	Ravi	Indore	20
14	Verma	Delhi	20
15	Sharma	Vizag	10

Dept	
Deptno	Name
10	CSE
20	ECE
30	Civil
40	Mech

.open univ

It will open the university

select city from students;

It will select the city from student table

select city from students where name = 'Ajay';

It will select the city whose name is Ajay

select \* from students where city in ( select city from students where name = 'Ajay');

The query in the bracket will execute first .

This query says that find the city names of all students where Ajay is living  
# this is called as subquery

select \* from students where city in ( select city from students where name = 'Ali');

Find the name of the students where Ali is living

#this is also called as inner query and we use in for multiple values

select \* from students where deptno = ( select deptno from students where name = 'Ajay');

Find the students in dept where dept number is matching to the Ajay

`select * from students where roll > ( select roll from students where name = 'Suraj');`

Here we are using greater than . Find all the students whose roll number is greater than Suraj

`select * from students where roll > ( select avg(roll) from students);`

Find all the students whose roll number is greater than average than Suraj

`select * from dept ;`

Select all from dept

`insert into dept values(60, ' EEE');`

It will insert it into dept

`select * from dept;`

Select all from dept

`delete from dept where dname='EEE';`

Delete from whose dept name Is EEE

`select * from dept ;`

Select all from dept

`update dept set dname='Aero' where deptno=50;`

We don't have to say from in this cause we are updating

`select * from dept;`

Select all from dept

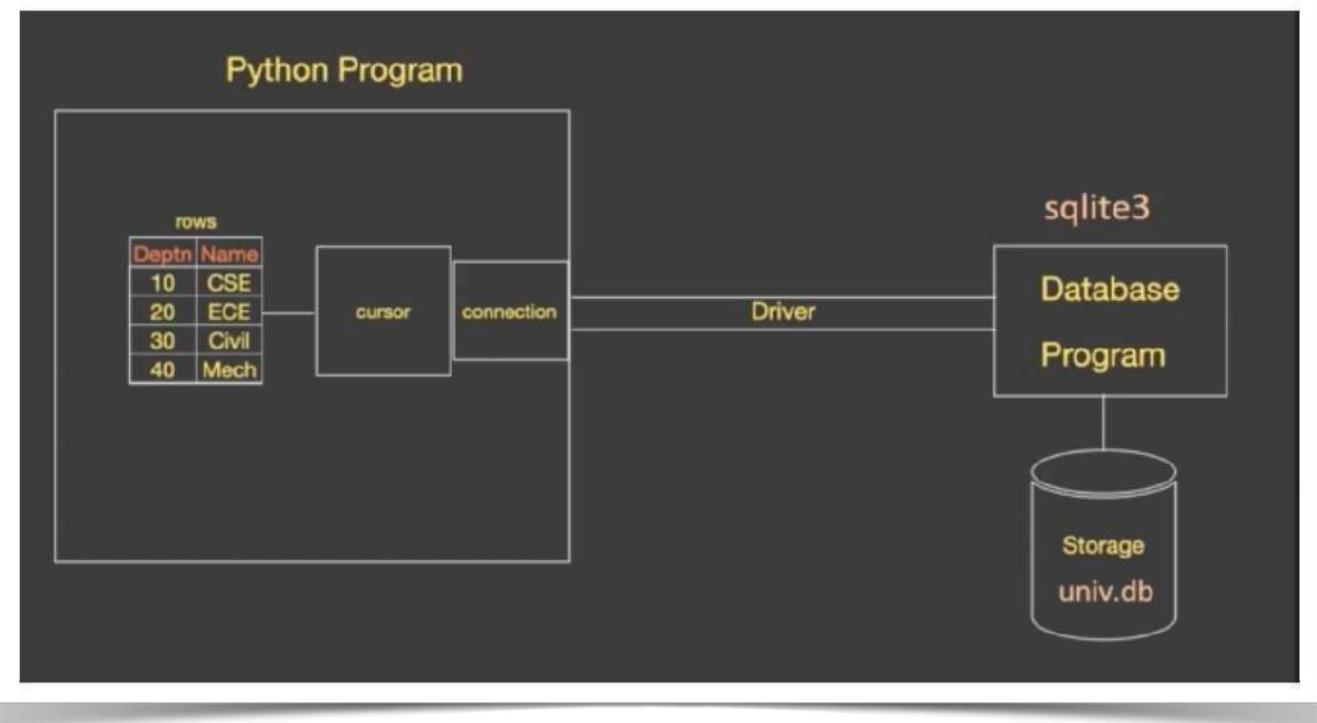
## Creating Database

- Python database can connect to any database such as oracle , Mysql , SQLite
- For connecting this we need a module
- Module will contain a driver
- SQLite is built in module you don't have to install it but if u are using another module than you have to install it
- Accessing and connecting database is same

```
1
2 import sqlite3
3
4 conn = sqlite3.connect('univ.db')
```

- Conn = `sqlite3.connect( ' univ .db')`
- If the database is not present it will create a database . If it is not created than it will create the database
- If you want to if the univ.db is created go to cmd and type dir

## DDL



- DDL stands for Data Definition Language .
- DDL include ALTER TABLE, CREATE TABLE, DROP TABLE, CREATE DATABASE, and TRUNCATE TABLE.
- Python accesses the database through a cursor,
- We will give queries to cursor object and it will pass to connection
- Connection object will connect to the SQLite3

```
import sqlite3

conn = sqlite3.connect('univ.db')

cur = conn.cursor()

cur.execute(['create table Students(roll integer primary key, name text, city text,)'])

conn.commit()

cur.close()

conn.close()
```

- It will enable multiple users to work on the same databases.  
It improved security efficient data access.

```
cur.execute('create table Student(roll integer primary key , name  
text , city text ,deptno integer , foreign key (deptno) references  
Dept(deptno)')
```

- This lines says that create a table name it as Student - where roll is integer type and it primary key ( unique ) , name is text type , city is text type , deptno is integer type , which is foreign key which refers to the deptno table which we have created previously

## DML

Students				Dept	
Roll	name	City	Deptno	Deptno	Name
1	Ajay	Delhi	10	10	CSE
2	Vijay	Kolkata	10	20	ECE
3	Ajay	Mumbai	20	30	Civil
4	Ramesh	Delhi	30	40	Mech
5	Suneeta	Lucknow	40		
6	Anita	Kolkata	30		
7	Raj	Jaipur	30		
8	Ali	Lucknow	40		
9	Michael	Cochin	10		
10	Pavan	Vijaywada	20		
11	Suraj	Hyderabad	10		
12	Altaf	Bangaluru	40		
13	Ravi	Indore	20		
14	Verma	Delhi	20		
15	Sharma	Vizag	10		

- Data manipulation language (DML) statements . which is used to manipulate data itself. For example: insert, update, delete are instructions in SQL

```
cur = conn.cursor()  
  
cur.execute('insert into Dept values(10,"CSE")')  
  
conn.commit()
```

- Using the cursor you have to execute the query . In execute we are saying that insert it into dept values( table ) where deptno is 10 and Name is CSE

- Open cmd .open univ.db  
.tables  
select \* from dept
- 10|CSE

```
cur.execute('insert into Dept(name,deptno) values("ECE",20)')
```

- You can mention it in any order
- We can insert by taking user input also

```
cur = conn.cursor()

dno = int(input('Enter Deptno'))
dname = input('Enter dname')

cur.execute(f'insert into Dept values({dno}, "{dname}")')

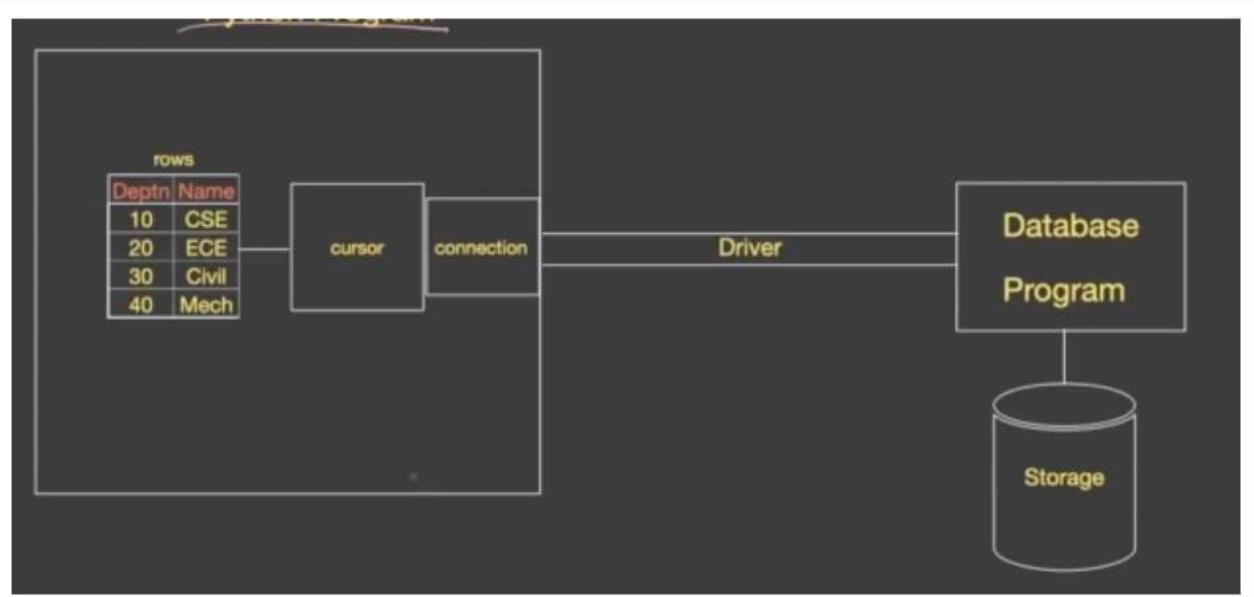
conn.commit()
```

Enter Deptno : 40  
Enter dname: Mech

Output

```
sqlite> select * from Dept;
10|CSE
20|ECE
30|Civil
40|Mech
```

## Select Queries #1



- Connection : This is use to connect
- Cursor :It is use to execute the query

```
import sqlite3

conn = sqlite3.connect('univ.db')

cur = conn.cursor()

rows = cur.execute('select * from Dept')

print(rows.fetchone())

cur.close()

conn.close()
```

```
rows = cur.execute( 'select * from Dept')
print(rows.fetchone( ))
```

- `fetchone()` it will give one row . If you want one more row then again `fetchone()` it will get you the another row  
It will return as one tuple form

- `fetchmany( 3 )` we can fetch as many we want . As the result it will return 3 rows
- `fetchall()` it will return all the rows but in tuple form

```
rows = cur.execute('select name from Students')

allrows = rows.fetchall()

for t in allrows:
    print(t[0])
```

But if we use for loop for printing the rows

```
C:\Users\Abdul Bari\Desktop\MyPython>python Database.py
[('Ajay',), ('Vijay',), ('Ajay',), ('Ramesh',), ('Sunitha',), ('Anitha',), ('Raj',), ('Ali',), ('Michael',), ('Pavan',),
('Suraj',), ('Altaf',), ('Ravi',), ('Verma',), ('Sharma',)]

C:\Users\Abdul Bari\Desktop\MyPython>python Database.py
Ajay
Vijay
Ajay
Ramesh
Sunitha
Anitha
Raj
Ali
Michael
Pavan
Suraj
Altaf
Ravi
Verma
Sharma
```

We can use `fetch all` for where `city = ' Delhi '`

```
rows = cur.execute("select * from Students where city = 'Delhi'")

allrows = rows.fetchall()

print('Roll  Name  City  Deptno')
for t in allrows:
    print(t[0],t[1],t[2],t[3])
```

```
C:\Users\Abdul Bari\Desktop\MyPython>python Database.py  
(1, 'Ajay', 'Delhi', 10)  
(4, 'Ramesh', 'Delhi', 30)  
(14, 'Verma', 'Delhi', 20)
```

```
C:\Users\Abdul Bari\Desktop\MyPython>python Database.py  
Roll Name City Deptno  
1 Ajay Delhi 10  
4 Ramesh Delhi 30  
14 Verma Delhi 20
```

- If we are using fetchall( ) it is returning as a tuple form

## Select Queries #2

- The Data base is as follows

Students				Dept	
Roll	name	City	Deptno	Deptno	Name
1	Ajay	Delhi	10	10	CSE
2	Vijay	Kolkata	10	20	ECE
3	Ajay	Mumbai	20	30	Civil
4	Ramesh	Delhi	30	40	Mech
5	Suneeta	Lucknow	40		
6	Anita	Kolkata	30		
7	Raj	Jaipur	30		
8	Ali	Lucknow	40		
9	Michael	Cochin	10		
10	Pavan	Vijaywada	20		
11	Suraj	Hyderabad	10		
12	Altaf	Bangaluru	40		
13	Ravi	Indore	20		
14	Verma	Delhi	20		
15	Sharma	Vizag	10		

- The queries to solve are

6. select roll, name from Students where city not in ('Delhi', 'Hyderabad')
7. select roll, Students.name, Dept.name from Students, Dept where Students.deptno=Dept.deptno
8. select city, count(\*) from Students group by city
9. select city, count(\*) from Students group by city having count(\*)>2
10. select name from Students where city in (select city from Students where name='Verma')

## Query 6 - solution

```
import sqlite3
conn = sqlite3.connect('univ.db')
cur = conn.cursor()
rows = cur.execute("select roll,name from Students where city not in ('Delhi', 'Hyderabad')")
print(rows.fetchall())
cur.close()
conn.close()
```

```
C:\Users\Abdul Bari\Desktop\MyPython>python Database.py
[(2, 'Vijay'), (3, 'Ajay'), (5, 'Sunitha'), (7, 'Raj'), (8, 'Ali'), (9, 'Michael'), (10, 'Pavan'), (12, 'Altaf'), (13, 'Ravi'), (15, 'Sharma')]
```

## Query 7 - solution

```
import sqlite3
conn = sqlite3.connect('univ.db')
cur = conn.cursor()
rows = cur.execute('''select roll,Students.name,Dept.name
from Students,Dept
where Students.deptno=Dept.deptno ''')
print(rows.fetchall())
cur.close()
conn.close()
```

```
C:\Users\Abdul Bari\Desktop\MyPython>python Database.py
[(1, 'Ajay', 'CSE'), (2, 'Vijay', 'CSE'), (3, 'Ajay', 'ECE'), (4, 'Ramesh', 'Civil'), (5, 'Sunitha', 'Mech'), (6, 'Anitha', 'Civil'), (7, 'Raj', 'Civil'), (8, 'Ali', 'Mech'), (9, 'Michael', 'CSE'), (10, 'Pavan', 'ECE'), (11, 'Suraj', 'CSE'), (12, 'Altaf', 'Mech'), (13, 'Ravi', 'ECE'), (14, 'Verma', 'ECE'), (15, 'Sharma', 'CSE')]
```

## Query 8 - solution

```
import sqlite3

conn = sqlite3.connect('univ.db')

cur = conn.cursor()

rows = cur.execute('''select city,count(*)
                     from Students
                     group by city ''')

print(rows.fetchall())

cur.close()

conn.close()
```

```
[('Bangalore', 1), ('Chennai', 1), ('Delhi', 3), ('Hyderabad', 1), ('Indore', 1), ('Jaipur', 1), ('Kolkata', 2), ('Lucknow', 1), ('Mumbai', 4), ('Vijaywada', 1), ('Vizag', 1)]
```

## Query 9 - solution

```
import sqlite3

conn = sqlite3.connect('univ.db')

cur = conn.cursor()

rows = cur.execute('''select city,count(*)
                     from Students
                     group by city
                     having count(*)>2''')

print(rows.fetchall())

cur.close()

conn.close()
```

```
[(('Delhi', 3))]
```

## Query 10 - solution

```
import sqlite3

conn = sqlite3.connect('univ.db')

cur = conn.cursor()

rows = cur.execute('''select name
                     from Students
                     where city in
                     (select city
                     from Students
                     where name='Verma') ''')

print(rows.fetchall())

cur.close()

conn.close()
```

```
[('Ajay',), ('Ramesh',), ('Verma',)]
```

## DML Update Query

- The following data base is available in our Univ database thus we will be using it

Students				Dept	
Roll	name	City	Deptno	Deptno	Name
1	Ajay	Delhi	10		
2	Vijay	Kolkata	10		
3	Ajay	Mumbai	20		
4	Ramesh	Delhi	30		
5	Suneeta	Lucknow	40		
6	Anita	Kolkata	30		
7	Raj	Jaipur	30		
8	Ali	Lucknow	40		
9	Michael	Cochin	10		
10	Pavan	Vijaywada	20		
11	Suraj	Hyderabad	10		
12	Altaf	Bangaluru	40		
13	Ravi	Indore	20		
14	Verma	Delhi	20		
15	Sharma	Vizag	10		

- We have to perform the following queries on our database using Python

- update Dept set name='Chem' where name='Mech'
- update Students set City='Bhopal' where roll=15
- delete from Students where roll=15
- delete from Dept where deptno=40

- The first query is , In the dept database we need to change name from Mech to Chem

//writing SQL Query in VSCode

```
import sqlite3

conn = sqlite3.connect('univ.db')

cur = conn.cursor()

cur.execute('update Dept set name="Chem" where name="Mech"')

conn.commit()

cur.close()

conn.close()
```

// Running error free program in CMD

```
C:\Users\Abdul Bari\Desktop\MyPython>dir
Volume in drive C is OS
Volume Serial Number is 3E57-713D

Directory of C:\Users\Abdul Bari\Desktop\MyPython

03/26/2022  03:58 AM    <DIR>      .
03/26/2022  03:58 AM    <DIR>      ..
03/27/2022  05:09 AM           190 Database.py
03/26/2022  03:58 AM           12,288 univ.db
                  2 File(s)       12,478 bytes
                  2 Dir(s)   414,603,014,144 bytes free

C:\Users\Abdul Bari\Desktop\MyPython>python Database.py

C:\Users\Abdul Bari\Desktop\MyPython>
```

//before and after update result

```
C:\Users\Abdul Bari\Desktop\MyPython>sqlite3
SQLite version 3.38.1 2022-03-12 13:37:29
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a persistent database.
sqlite> .open univ.db
sqlite> select * from Dept;
10|CSE
20|ECE
30|Civil
40|Mech
sqlite> select * from Dept;
10|CSE
20|ECE
30|Civil
40|Chem
sqlite>
```

- Lets look at the second query

//writing SQL Query in VSCode

```
import sqlite3

conn = sqlite3.connect('univ.db')

cur = conn.cursor()

cur.execute('update Students set city="Bhopal" where roll=15')

conn.commit()

cur.close()

conn.close()
```

// Running error free program in CMD

```
C:\Users\Abdul Bari\Desktop\MyPython>python Database.py

C:\Users\Abdul Bari\Desktop\MyPython>
```

//before and after update result

```
PS C:\Windows\System32\cmd.exe - sqlite3
3|Ajay|Mumbai|20
4|Ramesh|Delhi|30
5|Sunitha|Lucknow|40
6|Anitha|Kolkata|30
7|Raj|Jaipur|30
8|Ali|Lucknow|40
9|Michael|Cohin|10
10|Pavan|Vijaywada|20
11|Suraj|Hyderabad|10
12|Altaf|Bangaluru|40
13|Ravi|Indore|20
14|Verma|Delhi|20
15|Sharma|Vizag|10
sqlite> select * from Students;
1|Ajay|Delhi|10
2|Vijay|Kolkata|10
3|Ajay|Mumbai|20
4|Ramesh|Delhi|30
5|Sunitha|Lucknow|40
6|Anitha|Kolkata|30
7|Raj|Jaipur|30
8|Ali|Lucknow|40
9|Michael|Cohin|10
10|Pavan|Vijaywada|20
11|Suraj|Hyderabad|10
12|Altaf|Bangaluru|40
13|Ravi|Indore|20
14|Verma|Delhi|20
15|Sharma|Bhopal|10
sqlite>
```

- Now , in the third query we'll be deleting roll no 15 from the student database

- It is possible to delete multiple rows , but right now according to our requirement we are deleting a single row

//writing SQL Query in VSCode

```
import sqlite3

conn = sqlite3.connect('univ.db')

cur = conn.cursor()

cur.execute('delete from Students where roll=15')

conn.commit()

cur.close()

conn.close()
```

// Running error free program in CMD

```
C:\Users\Abdul Bari\Desktop\MyPython>python Database.py
C:\Users\Abdul Bari\Desktop\MyPython>python Database.py
C:\Users\Abdul Bari\Desktop\MyPython>
```

//delete result

```
sqlite> .open univ.db
sqlite>
sqlite> select * from Students;
1|Ajay|Delhi|10
2|Vijay|Kolkata|10
3|Ajay|Mumbai|20
4|Ramesh|Delhi|30
5|Sunitha|Lucknow|40
6|Anitha|Kolkata|30
7|Raj|Jaipur|30
8|Ali|Lucknow|40
9|Michael|Cohin|10
10|Pavan|Vijaywada|20
11|Suraj|Hyderabad|10
12|Altaf|Bangaluru|40
13|Ravi|Indore|20
14|Verma|Delhi|20
sqlite> ■
```

- The last query is delete from dept where dept no = 40

- All the keys from student dept **referring** to the key **40** in dept db will also be deleted

//writing SQL Query in VSCode

```
import sqlite3

conn = sqlite3.connect('univ.db')

cur = conn.cursor()

cur.execute('delete from Dept where deptno=40')

conn.commit()

cur.close()

conn.close()
```

// Running error free program in CMD

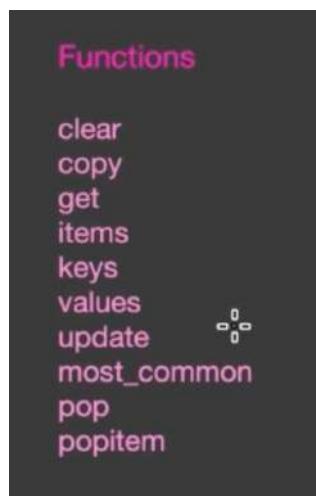
```
C:\Users\Abdul Bari\Desktop\MyPython>python Database.py
C:\Users\Abdul Bari\Desktop\MyPython>
```

//delete result

```
sqlite> select * from Dept;
10|CSE
20|ECE
30|Civil
.
```

## DataStructures : Counter

- It is present inside a package called collections
- It'll take sequence as input and **count number of times** each element is occurring
- It'll also **count duplicate** , it'll give output as **dictionary**
- Methods present inside counter modules are



- Lets see this with examples

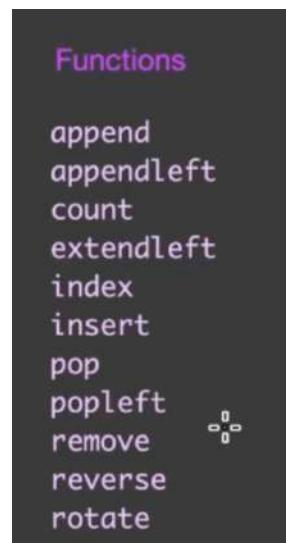
```
>>>
>>> L = [ 'Mark', 'Jonny', 'David', 'Mark', 'Jonny', 'Mark', 'James', 'Mathew' ]
>>>
>>> from collections import Counter
>>>
>>> c = Counter(L)
>>> c
Counter({'Mark': 3, 'Jonny': 2, 'David': 1, 'James': 1, 'Mathew': 1})
>>>
>>> c['Mark']
3
>>> c.get('Mark')
3
>>> c.keys()
dict_keys(['Mark', 'Jonny', 'David', 'James', 'Mathew'])
>>> c.values()
dict_values([3, 2, 1, 1, 1])
>>>
>>> c.update({'Ajay':4})
>>> c
Counter({'Ajay': 4, 'Mark': 3, 'Jonny': 2, 'David': 1, 'James': 1, 'Mathew': 1})
>>> c.elements()
<itertools.chain object at 0x7ff11e5ee190>
```

```
>>> for i in c.elements():
    print(i)

Mark
Mark
Mark
Jonny
Jonny
David
James
Mathew
Ajay
Ajay
Ajay
Ajay
>>> c.pop('Ajay')
4
>>> c
Counter({'Mark': 3, 'Jonny': 2, 'David': 1, 'James': 1, 'Mathew': 1})
>>> c.most_common(1)
[('Mark', 3)]
>>> c.most_common(2)
[('Mark', 3), ('Jonny', 2)]
>>> c.update({'Ajay':4})
>>> c.most_common(2)
[('Ajay', 4), ('Mark', 3)]
>>> |
```

## Data Structure : deque

- deque means **double ended queue** i.e, operations can be performed from both sides of the queue
- It can be used as a **queue** as well as a **stack**
- Queue follows FIFO order , stack follows LIFO order
- Methods available in queue are



- Lets see this with an example

```
>>>
>>> from collections import deque
>>>
>>> L = [1,2,3,4,5]
>>> q = deque(L)
>>> q
deque([1, 2, 3, 4, 5])
>>> q.append(6)
>>> q
deque([1, 2, 3, 4, 5, 6])
>>> q.appendleft(7)
>>> q
deque([7, 1, 2, 3, 4, 5, 6])
>>> q.pop()
6
>>> q.popleft()
7
>>> q.extend([10,20,30])
>>> q
deque([1, 2, 3, 4, 5, 10, 20, 30])
>>> q.extendleft([11,22,33,44])
>>> q
deque([44, 33, 22, 11, 1, 2, 3, 4, 5, 10, 20, 30])
>>> q.rotate(2)
>>> q
deque([20, 30, 44, 33, 22, 11, 1, 2, 3, 4, 5, 10])
>>>
```

## Data Structure : Array

- Array is same as list the difference between them is **array** allows similar type of elements (Datatype) while list takes any type of elements
- Upon array **all list operations** are possible
- Array is **homogenous**
- The Datatypes Array support are

Code	Type	Minimum Size (Bytes)
b	Int	1
B	Int	1
i	Signed int	2
I	Unsigned int	2
l	Signed long	4
L	Unsigned long	4
q	Signed long long	8
Q	Unsigned long long	8
f	float	4
d	Double float	8

- Lets see this with some examples

```
>>> import array
>>> ar1 = array.array('I',[10,20,30,40])
>>> ar1
array('I', [10, 20, 30, 40])
>>>
>>> s1 = b'abcdef'
>>>
>>> ar2 = array.array('b', s1)
>>> ar2
array('b', [97, 98, 99, 100, 101, 102])
>>>
>>> ar2[0]
97
>>> ar2[1:3]
array('b', [98, 99])
>>>
>>> ar2.append(103)
>>> ar2
array('b', [97, 98, 99, 100, 101, 102, 103])
>>> ar2.index(102)
5
>>> ar2.typecode
'b'
>>>
```

## Heap

- It will work as a priority queue . It will gives you the highest priority element

h = [ 10 , 20 , 50 , 40 , 60 , 30 , 70 ]

- Smaller the number will be the highest priority so 10 has the highest priority and 70 will be the least priority . That's means the larger the number will be the least priority

- **heappush** :-

```
>>> import heapq
>>> H = []
>>>
>>> heapq.heappush(H,20)
>>> H
[20]
>>> heapq.heappush(H,50)
>>> H
[20, 50]
>>> heapq.heappush(H,10)
>>> H
[10, 50, 20]
>>> heapq.heappush(H,40)
>>> heapq.heappush(H,30)
>>> heapq.heappush(H,60)
>>> H
[10, 30, 20, 50, 40, 60]
>>>
```

For this you have to import the module heapq . By using heappush we can add the elements .

- **heappop** :

```
>>> heapq.heappop(H)
10
>>> H
[20, 30, 60, 50, 40]
>>> heapq.heappop(H)
20
>>> H
[30, 40, 60, 50]
```

- heappop it will remove the highest priority ( H )

- **heapify** :

- If we have taken a list example  
L = [50,30,60,40,70,20,10]

As you can see that this list is not in the form of heap.

- So heapify will convert the list into heap form

### **heapq.nlargest and heapq.nsmallest :**

- This method is use to find the largest and smallest among the heap

`heapq.nlargest( 2 , L )`

This means two largest number

`heapq.nsmallest( 3,L )`

This means give 3 smallest number present in the List

## Bisect

b = [ 10 , 20 , 20 , 30 , 50 , 60 , 70 , 90 ]

- This will maintain the list in stored order
- It can have duplicate numbers
- If you want to perform intersection operations but it will ensure that the list is a sorted order

### insort :

It is used for inserting the element

### insort\_left and insert\_right :

If you want to enter the number which is already present in the list . Lets take 90 as you can see 90 is already present in the list

- At this situation we can use this . Whether you want 90 at right side of the 90( which is already present in the list ) or left side

```
>>> bisect.insort_left(L, 90)
>>> L
[10, 20, 20, 25, 30, 40, 50, 60, 70, 90, 90]
>>> id(L[9])
140539785970832
>>> id(L[10])
140539785970832
```

### bisect() , bisect\_left() , bisect\_right() :

It will not insert it will only say where is the position of a certain number

## Copy

### copy.copy( x ) :

- It is known as shallow copy . Let us know why it is said as shallow copy

lets take an example

```
L = [ 10 , 20 , 30 , 40 , 50 ]
```

- If you want to copy L in L1 . Then L1 will be holding the same object .
- L1 will create the new list is created but it will not create objects of the L

```
>>> import copy
>>> L = [10,20,30,40,50]
>>> L1 = copy.copy(L)
>>> L1
[10, 20, 30, 40, 50]
>>>
>>> id(L)
140302108233664
>>> id(L1)
140302108131392
>>>
>>> id(L[0])
140302061017680
>>> id(L1[0])
140302061017680
```

- If we check the id for L and L1 ( list ) it is different cause both are different List . But if we check the id of a particular object using indexing the id will be the same for L1 and L
- So it has shallowed copied it

### copy.deepcopy( x )

- This is also said as recursive copy ( copy of list then sublist than sublist than sublist .... So on )
- In this deep copy it will create the L1 ( list ) and it will create an object( numbers) too . This is like real duplicate
- By this deep copy will not work for int , float , string , complex etc .... It will works with classes , objects , functions ..

```
>>> import copy
>>> class Person:
    def __init__(self, name):
        self.name = name

>>> L = [Person('John'), Person('Tim'), Person('Jim')]
>>> L1 = copy.deepcopy(L)
>>> id(L[0])
140479058600576
>>> id(L1[0])
140479053248448
```

## Math Fraction (module)

- Fractions is in the form **p/q** where p [ numerator ] is greater than q [denominator]
- Hence the module math.fraction support operations on numbers in the form p / q
- The constructors for creating fractions are

```
Fraction( num=0, den=1 )
Fraction( float )
Fraction( string )
```

```
limit_denominator(den=10)
numerator
denominator
```

- Lets see this with an example

```
>>> import fractions
>>> f1 = fractions.Fraction(1,2)
>>> f1
Fraction(1, 2)
>>> print('{0}'.format(f1))
1/2
>>> f2 = fractions.Fraction(0.2)
>>> f2
Fraction(3602879701896397, 18014398509481984)
>>> f3 = fractions.Fraction('0.3')
>>> f3
Fraction(3, 10)
>>> f2 = f2.limit_denominator(10)
>>> f2
Fraction(1, 5)
>>> print('{0}'.format(f1 + f2))
7/10
>>> print('{0}'.format(f2 - f1))
-3/10
>>>
```

```
-- 
>>> print('{0}'.format(f2*f1))
1/10
>>> print('{0}'.format(f2/f1))
2/5
>>>
```

- We can also perform addition , subtraction , multiplication and division on fraction
- If you have list of tuples you can convert it into fractions also

## Math

For more math module you can go for

Link → <https://docs.python.org/3/library/math.html>

### math.ceil( ) :

- It will take the float value and return the next integer value ( upper value )

ex — math.ceil( 12.3)  
13

### math.floor( ) :

- It will take the float value and return the previous value ( lower value )

ex — math.floor( 13.9 )  
13

### math.fabs( )

- Whenever we pass negative number it will return a positive number

ex — math.fabs(-5 )  
5

- If we give +ve number it will return positive

### math.fmod( )

- It will return the modules . It will gives the remainder

```
>>> math.fabs(-12.34567)
12.34567
>>> math.fabs(12.34567)
12.34567
>>> math.fmod(11,3)
2.0
>>> math.fmod(12,5)
2.0
```

### math.sqrt( )

It will give you the square root of a number

### math.isqrt( )

It will return the result of square root only in integer form not in float form

```
>>> import math  
>>> math.sqrt(25)  
5.0  
>>> math.sqrt(27)  
5.196152422706632  
>>> math.isqrt(27)  
5
```

### math.pow( )

It will give you the power value

### math.factorial( )

Lets find out factorial of 5

$$\begin{aligned} 5 &\rightarrow 1*2*3*4*5 \\ &\rightarrow 120 \end{aligned}$$

```
>>> math.pow(10,2)
```

100.0

```
>>> math.pow(2,10)
```

1024.0

```
>>> math.factorial(5)
```

120

### math.gcd( )

Greatest common divisor

How to find ?

- Step 1: Write the divisors of positive integer "a".
- Step 2: Write the divisors of positive integer "b".
- Step 3: Enlist the common divisors of "a" and "b".
- Step 4: Now find the divisor which is the highest of both "a" and "b".

```
>>> import math
```

```
>>> math.gcd(35,21)
```

7

### math.perm( )

It will give the result in  $nPr$ . And its formula is  $nPr = n!/ (n-r)!$

ex —>  $5P2 = 5!/(5-2)!$   
—>  $5! / 3!$   
—> ~~$5*4*3*2*1 / 3*2*1$~~   
—> $5*4$   
—> 20

### math.comb( )

$nCr = n! / (n-r)! r !$

```
>>> math.perm(5,2)
```

20

```
>>> math.comb(5,2)
```

10

.

### math.prod( )

It will give the product list of item

ex—> [1,2,3,4,5]

—>  $1*2*3*4*5$

—> 120

It will multiply this values and gives you the product values

### math.fsum( )

It will give the value of sum

ex—> [1,2,3,4,5]

—>  $1+2+3+4+5$

—> 15

```
>>> math.prod([1,2,3,4,5])
```

120

```
>>> math.prod([0,1,2,3,4])
```

0

```
>>> math.fsum([2,2,2,2,2])
```

10.0

```
>>> math.fsum([1,2,3,4,5])
```

15.0

## **math.radians( )**

Convert angle x from degrees to radians.

## **math.degree( ) :**

Convert angle x from radians to degrees.

```
>>> import math  
>>> math.radians(30)  
0.5235987755982988  
>>> math.radians(180)  
3.141592653589793
```

## **math.sin( ) , math.cos( ) , math.tan( ) :**

It will take it as a form of radians

```
>>> math.sin(math.radians(30))  
0.4999999999999994  
>>> math.cos(math.radians(60))  
0.5000000000000001  
>>> math.tan(math.radians(45))  
0.999999999999999
```

## **math.log( ) , math.log10 , math.log2 ( ) :**

```
>>> import math  
>>>  
>>> math.log2(1024)  
10.0  
>>> math.log2(256)  
8.0
```

## math.pi , math.e , math.nan

```
>>> math.pi  
3.141592653589793  
>>> math.e  
2.718281828459045  
>>> math.nan  
nan
```

## Statistics

```
data=[1,2,3,4,5]
```

### mean( )

- It will perform the operation on list ( float and integer ) . Mean will take the sum of values and divide with the number of elements present in the list
- In other words it will take the average

### harmonic\_mean( )

- To find the harmonic mean of a set of n numbers, add the reciprocals of the numbers in the set, divide the sum by n, then take the reciprocal of the result.

```
>>> import statistics as s
>>> s.mean([1,2,3,4,5])
3
>>> s.mean([4,3,5,1,2])
3
>>> s.harmonic_mean([1,2,3,4,5])
2.18978102189781
```

### median( )

- Taking the elements in sorted order and taking the middle element .
- If we are odd elements we can easily take middle number but if we have even number of elements it will take two middle numbers and add them . By adding them we get median. It will take the average

### median\_high( )

- If the number of list values is odd, it returns the exact middle value which is higher .

### median\_low( )

- It will return the lower middle value of the list

```
>>> import statistics as s
>>>
>>> s.median_low([1,2,3,4,5,6])
3
>>> s.median_high([1,2,3,4,5,6])
4
>>> s.median_high([1,2,30,50,51,52])
50
>>> s.median_low([1,2,30,50,51,52])
30
>>> s.median([1,2,30,50,51,52])
```

## **mode( )**

- From the list of elements it will find out most occurring numbers .

```
>>> import statistics as s  
>>> s.mode([1,1,1,2,2,3])  
1  
>>> s.mode([1,1,1,2,2,2,3])  
1  
>>> s.mode([4,4,4,1,1,1,2,2,2,3])  
4
```

## **pvariance ( )**

- It will find out the mean than it will take the whole square

## **pstdev( )**

- Root of the pvariance is pstdev( standard division)

## **stdev( ) and variance ( )**

It will not take the whole list . It will take some sample of data

```
>>> import statistics as s  
>>> s.mode([1,1,1,2,2,3])  
1  
>>> s.mode([1,1,1,2,2,2,3])  
1  
>>> s.mode([4,4,4,1,1,1,2,2,2,3])  
4  
>>> s.pvariance([1,2,3,4,5])  
2  
>>> s.pstdev([1,2,3,4,5])  
1.4142135623730951  
>>> s.variance([1,2,3,4,5])  
2.5  
>>> s.stdev([1,2,3,4,5])  
1.5811388300841898
```

## OS path

- Os.path is the sub module of the main module I.e, os module
- Functions available in os.path are

```
os.path.exists()
os.path.isfile()
os.path.isdir()

os.path.split()
os.path.join()

os.path.basename()
os.path.dirname()

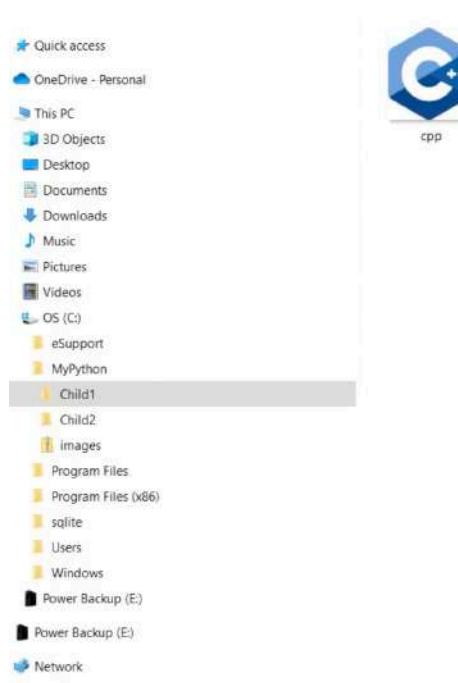
os.path.getmtime()
os.path.getatime()

os.path.relpath()
os.path.abspath()
```

**Os.path.exists** - check if path exist or not

**os.path.isfile** - check if path belongs to file

**os.path.isdir** - given path is file or dir



```
AMD64] on win32
Type "help", "copyright", "credits" or "license()" for
>>> import os
>>>
>>> os.path.exists('C:\\MyPython\\Child1\\cpp.png')
True
>>> os.path.exists('C:\\MyPython\\Child5\\cpp.png')
False
>>>
>>> os.path.isfile('C:\\MyPython\\Child1\\cpp.png')
True
>>>
>>> os.path.isdir('C:\\MyPython\\Child1\\cpp.png')
False
>>> os.path.isdir('C:\\MyPython\\Child1')
True
>>>
>>> os.path.split('C:\\MyPython\\Child1\\cpp.png')
('C:\\MyPython\\Child1', 'cpp.png')
>>> os.path.join('C:\\MyPython\\Child1', 'cpp.png')
'C:\\MyPython\\Child1\\cpp.png'
>>>
>>> os.path.basename('C:\\MyPython\\Child1\\cpp.png')
'cpp.png'
>>> os.path.dirname('C:\\MyPython\\Child1\\cpp.png')
'C:\\MyPython\\Child1'
>>>
```

**os.path.split( )** - split address and file name

**os.path.join( )** - join path and file name together to give single path

**os.path.getmtime( )** - last access time

**os.path.getatime( )** - last modified time

```
>>> import os
>>>
>>> os.path.getmtime('C:\\MyPython\\Compress.py')
1648387173.5853446
>>>
>>> import time
>>>
>>> time.ctime(os.path.getmtime('C:\\MyPython\\Compress.py'))
'Sun Mar 27 18:49:33 2022'
>>>
>>> time.ctime(os.path.getatime('C:\\MyPython\\Compress.py'))
'Thu Apr 14 12:14:02 2022'
>>>
>>> time.ctime(os.path.getctime('C:\\MyPython\\Compress.py'))
'Sun Mar 27 18:46:43 2022'
>>>
```

**os.path.replace( )** - access something from another folder

**os.path.abspath( )** - access something in absolute way

```
>>> import os
>>>
>>> os.chdir('C:\\MyPython\\Child1')
>>> os.getcwd()
'C:\\MyPython\\Child1'
>>>
>>> os.path.relpath('C:\\MyPython\\Child2\\csharp.png')
'..\\Child2\\csharp.png'
>>>
>>> os.path.abspath('..\\Child2\\csharp.png')
'C:\\MyPython\\Child2\\csharp.png'
>>>
```

## OS Module

- OS module or Operating System module
- The OS function that are related to files and directories are

```
os.name  
  
os.getcwd()  
os.chdir()  
os.listdir()  
  
os.mkdir()  
os.makedirs()  
  
os.remove()  
os.rmdir()  
os.removedirs()  
  
os.rename()
```

- Lets look at their functionality

`os.getlogin()` - gives user name which is logged in

`os.getcwd()` - get current working directory

`os.chdir()` - change directory

`os.listdir()` - gives list of files in current directory

`os.mkdir()` - create a folder

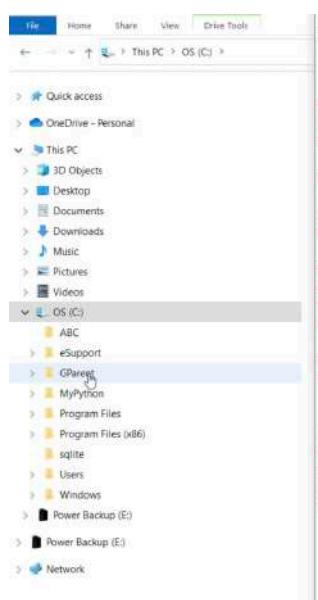
`os.makedirs()` - create multiple folders

`os.remove()` - remove a file

`os.rmdir()` - remove a dir

`os.removedirs()` - remove multiple dirs

`os.rename()` - renames a folder



```

File Edit Shell Debug Options Window Help
Python 3.10.3 (tags/v3.10.3:a342a49, Mar 16 2022, 13:07:40) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> import os
>>>
>>> os.name
'nt'
>>> os.getlogin()
'Abdul Bari'
>>>
>>> os.getcwd()
'C:\\Users\\Abdul Bari\\AppData\\Local\\Programs\\Python\\Python310'
>>>
>>> os.chdir('C:\\MyPython')
>>>
>>> os.getcwd()
'C:\\MyPython'
>>>
>>> os.listdir()
['Compress.py', 'cpp.png', 'csharp.png', 'Database.py', 'Decompress.py', 'images.zip', 'java.png', 'javascript.png', 'python.jpg', 'univ.db']
>>>
>>> os.mkdir('C:\\ABC')
>>>
>>> os.makedirs('C:\\GParent\\Parent\\Child')
>>>

```

## CSV Read

- The data organised in **text files** , values **separated by comma ( , )** new data given in **next line** this type of data arrangement is called **CSV files**
- CSV files will not have images and audios it'll only have text data
- This files are supported by many spread sheets applications like MS-Excel , Numbers , Open Office etc lets see this with an example
- Creating an MS-Excel file with employee data

	A	B	C	D
1	EmpID	Name	Salary	
2	e101	Pramod	1200000	
3	e120	Dinesh	2200000	
4	e205	Sabesta	1500000	
5	e331	Harry	1700000	
6	e421	Avinash	1300000	
7	e231	Joy	2300000	
8	e222	Smith	2100000	
9	e339	Khan	1800000	
10	e150	Dilip	1900000	
11	e131	Kiran	800000	
12				
13				
14				

- Our data is ready now we'll use python program to read this CSV data

```
import csv

f = open('Employees.csv','r')

csv_reader = csv.reader(f)

for row in csv_reader:
    print(row)
```

```
C:\Users\Abdul Bari\Desktop\MyPython>python CSVRead.py
['EmpID', 'Name', 'Salary']
['e101', 'Pramod', '1200000']
['e120', 'Dinesh', '2200000']
['e205', 'Sabesta', '1500000']
['e331', 'Harry', '1700000']
['e421', 'Avinash', '1300000']
['e231', 'Joy', '2300000']
['e222', 'Smith', '2100000']
['e339', 'Khan', '1800000']
['e150', 'Dilip', '1900000']
['e131', 'Kiran', '800000']
```

- To find the minimum and maximum salary of employees we can do

```
import csv

f = open('Employees.csv', 'r')

csv_reader = csv.reader(f)

next(csv_reader)

sals = []
for row in csv_reader:
    sals.append(int(row[2]))

print(sals)
print('Min',min(sals))
print('Max',max(sals))
```

```
C:\Users\Abdul Bari\Desktop\MyPython>python CSVRead.py
[1200000, 2200000, 1500000, 1700000, 1300000, 2300000, 2100000, 18
00000, 1900000, 800000]
Min 800000
Max 2300000
```

## CSV Files

- We can read the CSV data in the form of dictionary as well for example

```
import csv

f = open('Employees.csv', 'r')

rdr = csv.DictReader(f)

for row in rdr:
    print(row)

f.close()
```

# Opening a CSV file in read mode

```
C:\Users\Abdul Bari\Desktop\MyPython>python CSVDictRead.py
{'EmpID': 'e101', 'Name': 'Pramod', 'Salary': '1200000'}
{'EmpID': 'e120', 'Name': 'Dinesh', 'Salary': '2200000'}
{'EmpID': 'e205', 'Name': 'Sabesta', 'Salary': '1500000'}
{'EmpID': 'e331', 'Name': 'Harry', 'Salary': '1700000'}
{'EmpID': 'e421', 'Name': 'Avinash', 'Salary': '1300000'}
{'EmpID': 'e231', 'Name': 'Joy', 'Salary': '2300000'}
{'EmpID': 'e222', 'Name': 'Smith', 'Salary': '2100000'}
{'EmpID': 'e339', 'Name': 'Khan', 'Salary': '1800000'}
{'EmpID': 'e150', 'Name': 'Dilip', 'Salary': '1900000'}
{'EmpID': 'e131', 'Name': 'Kiran', 'Salary': '800000'}
```

# Shows data in dictionary

- You can read a CSV file both in list and dictionary

```
import csv

f = open('Employees.csv', 'r')

rdr = csv.DictReader(f)
l
emps = {}

for row in rdr:
    emps[row['Name']] = row

print(emps)
|
f.close()
```

```
C:\Users\Abdul Bari\Desktop\MyPython>python CSVDictRead.py
{'Pramod': {'EmpID': 'e101', 'Name': 'Pramod', 'Salary': '1200000'}, 'Dinesh': {'EmpID': 'e120', 'Name': 'Dinesh', 'Salary': '2200000'},
'Sabesta': {'EmpID': 'e205', 'Name': 'Sabesta', 'Salary': '1500000'}, 'Harry': {'EmpID': 'e331', 'Name': 'Harry', 'Salary': '1700000'},
'Avinash': {'EmpID': 'e421', 'Name': 'Avinash', 'Salary': '1300000'}, 'Joy': {'EmpID': 'e231', 'Name': 'Joy', 'Salary': '2300000'},
'Smith': {'EmpID': 'e222', 'Name': 'Smith', 'Salary': '2100000'}, 'Khan': {'EmpID': 'e339', 'Name': 'Khan', 'Salary': '1800000'}, 'Dili
p': {'EmpID': 'e150', 'Name': 'Dilip', 'Salary': '1900000'}, 'Kiran': {'EmpID': 'e131', 'Name': 'Kiran', 'Salary': '800000'}}
```

C:\Users\Abdul Bari\Desktop\MyPython>

- If you want details of a particular person then do

```
import csv

f = open('Employees.csv', 'r')

rdr = csv.DictReader(f)

emps = {}

for row in rdr:
    emps[row['Name']] = row

#print(emps)

print('Harry ', emps['Harry'])

f.close()
```

```
C:\Users\Abdul Bari\Desktop\MyPython>python CSVDictRead.py
Harry  {'EmpID': 'e331', 'Name': 'Harry', 'Salary': '1700000'}
```

- To beautify the Output for better readability

```

import csv
import pprint

f = open('Employees.csv','r')

rdr = csv.DictReader(f)

emps = {}

for row in rdr:
    emps[row['Name']] = row

pprint.pprint(emps)

f.close()

```

```

C:\Users\Abdul Bari\Desktop\MyPython>python CSVDictRead.py
{'Avinash': {'EmpID': 'e421', 'Name': 'Avinash', 'Salary': '1300000'}, 
'Dilip': {'EmpID': 'e150', 'Name': 'Dilip', 'Salary': '1900000'}, 
'Dinesh': {'EmpID': 'e120', 'Name': 'Dinesh', 'Salary': '2200000'}, 
'Harry': {'EmpID': 'e331', 'Name': 'Harry', 'Salary': '1700000'}, 
'Joy': {'EmpID': 'e231', 'Name': 'Joy', 'Salary': '2300000'}, 
'Khan': {'EmpID': 'e339', 'Name': 'Khan', 'Salary': '1800000'}, 
'Kiran': {'EmpID': 'e131', 'Name': 'Kiran', 'Salary': '800000'}, 
'Pramod': {'EmpID': 'e101', 'Name': 'Pramod', 'Salary': '1200000'}, 
'Sabesta': {'EmpID': 'e205', 'Name': 'Sabesta', 'Salary': '1500000'}, 
'Smith': {'EmpID': 'e222', 'Name': 'Smith', 'Salary': '2100000'}}

```

## CSV Write 3

- If you have values in [list of tuples](#) you can store them in CSV files

```
import csv

covid = [ ('Country', 'Doses', 'People', 'Percentage'),
          ('India', '186Cr', '84.1Cr', 61),
          ('China', '330Cr', '124Cr', 88.1),
          ('United States', '56.8Cr', '21.9Cr', 66.4),
          ('Brazil', '42.4Cr', '16.2Cr', 76.4),
          ('Indonesia', '39Cr', '16.2Cr', 59.3)]


f = open('Covid.csv', 'w', newline='')

wrtr = csv.writer(f)

for t in covid:
    wrtr.writerow(t)

f.close()
```

	A	B	C	D	E	F
1	Country	Doses	People	Percentage		
2	India	186Cr	84.1Cr	61		
3	China	330Cr	124Cr	88.1		
4	United Sta	56.8Cr	21.9Cr	66.4		
5	Brazil	42.4Cr	16.2Cr	76.4		
6	Indonesia	39Cr	16.2Cr	59.3		
7						
8						

## CSV Dictionary Writer

- We can store the dictionary data into a CSV file

```
import csv

covid = [
{'Country':'India', 'Doses':'186Cr', 'People':'84.1Cr', 'Percentage':61},
{'Country':'China', 'Doses':'330Cr', 'People': '124Cr', 'Percentage':88.1},
{'Country':'United States', 'Doses':'56.8Cr', 'People': '21.9Cr', 'Percentage':66.4},
{'Country':'Brazil', 'Doses':'42.4Cr', 'People': '16.2Cr', 'Percentage':76.4},
{'Country':'Indonesia', 'Doses':'39Cr', 'People': '16.2Cr', 'Percentage':59.3}
]

f = open('CSVDict.csv','w',newline='')

wrtr = csv.writer(f)

for d in covid:
    wrtr.writerow(d)

f.close()
```

	A	B	C	D	E	F
1	Country	Doses	People	Percentage		
2	Country	Doses	People	Percentage		
3	Country	Doses	People	Percentage		
4	Country	Doses	People	Percentage		
5	Country	Doses	People	Percentage		
6						

- When we use a simple writer we get this result
- To get the output as key value we should use **DictWriter( )**

```
['Country': 'China', 'Doses': '330Cr', 'People': '124Cr', 'Percentage': 88.1},
{'Country': 'United States', 'Doses': '56.8Cr', 'People': '21.9Cr', 'Percentage': 66.4},
{'Country': 'Brazil', 'Doses': '42.4Cr', 'People': '16.2Cr', 'Percentage': 76.4},
{'Country': 'Indonesia', 'Doses': '39Cr', 'People': '16.2Cr', 'Percentage': 59.3}

f = open('CSVDict.csv','w',newline='')

fields = ['Country','Doses','People','Percentage']

wrtr = csv.DictWriter(f,fields)

wrtr.writeheader()

for d in covid:
    wrtr.writerow(d)

f.close()
```

	A	B	C	D	E
1	Country	Doses	People	Percentage	
2	India	186Cr	84.1Cr	61	
3	China	330Cr	124Cr	88.1	
4	United Sta	56.8Cr	21.9Cr	66.4	
5	Brazil	42.4Cr	16.2Cr	76.4	
6	Indonesia	39Cr	16.2Cr	59.3	
7					
8					