# PYTHON

0 : INTRO PYTHON
- INTRO

1 : MODULUS , COMMENTS & PIP
- MODULUS/libries

2 : VARIABLES & DATA TYPES
- Variables , keywords
- data-types
- operators , Typecasting

3 : STRINGS
- String function
- fString & DOC String

4 : CONDITIONAL STATEMENT
- If - Else
- match case / switch case

5 : LOOPS
- LOOPS , for , while , for with else
- Break , Continue & pass

6 : LISTS & TUPLES
- Lists , List methods
- Tuples , Tuple methods

7 : DICTIONARY & SETS
- Sets , Set Properties & functions
- Dictionary , Dictionary function

8 : FUNCTIONS & RECURSION
- functions , types of functions
- Recursion

9 : OOPS
- INTRO

Date
Page

# CHAPTER 0 : INTRODUCTION

## * Programming :

- Just like we use hind english to communicate with each other, we use a programming language like python, c++ to communicate with computer

- Programming is a way to instruct the computer to perform various tasks.

## * Python :

Python is a simple and easy to understand language which feels like reading simple english. This Pseudo code nature of python makes it easy to learn and understand by beginneres

Features of python

- Easy to understand
- Free and open source
- High level language
- Portable → works on Linux/windows/Mac
- Fun to work with

- Python was made in 1991 by "Guido Van Rossum"

# CHAPTER 04 : MODULUS, COMMENTS & PIP

print ("Hello world") ⟶ print is a function

## * MODULUS / libraries

- module is a file which contain codes written by somebody else, which can be imported and used in our programs.
- module is a pre-written program or code
- we can install module by writing
  pip install module name ⟶ by this we can install external module
- Some examples of module are
  - flask
  -

MODULUS

| BUILT IN MODULE | EXTERNAL MODULE |
| --- | --- |
| • Import OS : is a built in module which compes with python istallation | • Import flask : is a external module which we have to download from interent |
| • pre installed module | • need to install using pip |
| eg OS, apc etc | eg : flask, panda |

- We can use module by writing
  Import module name ⟶ and we can use

* **Pip :**

- Pip is a package manger for python, which helps to install modules.
- We can use pip to install external module
  → pip install flask

* **PYTOH AS A CALCULATOR**

- We can use python as calculator by directly typering 5+7 = in the terminal OR
- We can type "python+enter" on the terminal
  This will open a REPL → Read Evalute Print Loop

* **COMMENTS :**

Comments are used to write something which the programmer does not want to execute.

Type of comment :

| Single line comment | Multi line comment |
| --- | --- |
| • Written using '#' | Written using |
| • shortcut = cntrl + '/' slash | ''' coment ''' |

* String should be point with 3 code if they are more then one line

ctrl + forward slash → we can comment selected area and uncomment also

eg of a external module

* From playsound import playsound
playsound ('mp3 song path')
→ this path edit it by typing one more slash \
two slash (\\) should be their

* print ("Subscribe now", end = " ")
print ("Bhai video bhi like kar do")
→ subscribe now Bhai video bhi like kar do
→ एक ही line मे print होगा
(end =) से

* Forward slash OR ESCAPE sequences
black slash n = \n → new line
black slash t = \t → one tab space

Print ("C:\Harry")
→ C:\Harry

Print ("C:\\harry")
→ C:\harry

Print ("C:\" Harry")
→ C:" Harry

# CHAPTER 02: VARIABLES AND DATATYPES

## * Variable :

variable is the name given to a memory location in a pgm

eg
a = 30 → it means 30 no. is going to save in our memory location, and that memory locations name is 'a'
b = "Harry"
c = 71.22
identifier

we can access that memory location by 'a' name

- in python their is no need to give variable type like we give in C eg int a = 39;
- strings are to be written in double cote

## * Keywords :

keywords are the reserved words in python which have their own meaning

| | | | | |
|---|---|---|---|---|
| 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 | yeild |

* Data type :

Primary there are following data types in Python

1) integer := eg 3, 4
2) float = eg 3.77
3) Strings = "abc", "a", 'b'
4) Booleans = such as jhooth, true or flase
5) None = eg d = none

- python is a fantastic language that automatically identifies the type of data for us.

How to type or print type of datatype

eg a = '''harry'''
print (type (a)) ⟶ will print datatype of 'a'
= <class 'str'>

* Rules for defining a variable name

→ variable are case sensetive
→ A variable name can contain alphabets, digit and underscores
→ A variable name can only start with an ~~object~~ alphabet and underscore
→ A variable name can't start with a digit
→ No white space is allowed to be used inside a variable name

* Identifiers

Identifiers are the names of any class, function or variable name

## * OPRATORS IN Python

oprators are special symbol that carry out arithematic or logical computation

eg $2+3=5$ → here 2 and 3 → are called oprand

+ → is called oprator

5 → output of operation

### 1) Arithmetic operators :

+, -, *, /, %, **, //

| oprator | Name | eg | |
|---|---|---|---|
| + | Addition | x+y | x+y |
| - | Subtraction | 2-3 | -1 |
| * | multiplication | 2*3 | 6 |
| / | division | 47/5 | 9.4 |
| % | Modulus | 47%5 | 2 → remainder |
| ** | Exponentiation | $5^{**}2 = 5^2$ | power = 25 |
| // | Floor division | 47//5 | 9 — quetoind |

### 2) Assignment operators :

| operator | defination | eg. |
|---|---|---|
| = | Assigns value from right to left | eg c=a+b. |
| += | it adds Right operend to left | eg a=34, a+=2 a will have 36 value |
| -= | it subtract Right operend to left | eg a=34, a-=2 a will have 32 |
| **= | it perform exponential power | eg a=2, a**=2 a will have 4 |

similarly we have =, +=, -=, *=, **=, /=, //=, %=,

3) Comparision operator

Comparision operators return boolean datatype that is true or false

| Oprator | Name | eg | Output |
|---|---|---|---|
| == | equal | 3==4 | false |
| != | Not equal | 3!=4 | True |
| > | greater than | 3>4 | false |
| < | less than | 3<4 | True |
| >= | Greater than or equal to | 3>=4 | False |
| <= | less then or equal to | 3<=4 | True |

4) logical oprators :

and , or , not are 3 logical operators and they return boolean datatype

| Operator | definition | eg | output |
|---|---|---|---|
| and | Returns true if both statements are true otherwise false | 1) 3<5 and 10<5 | F |
| | | 2) 3<5 and 3<10 | T |
| or | Returns true if one of the statement is true | 1) 3<5 or 10>3 | T |
| | | 2) 3>5 or 10>3 | F |
| not | Reverse the Results Return false if the result is true | 1) not 3>5 | T |
| | | 2) not 3<5 | F |

* Python follows indentation. Unlike many other programming langues that uses curly brackets { } to define code blocks, but python uses indentation to group statements together

## * Typecasting

Typecasting is a way to convert one datatype to another

converte a (string) to integer

eg
```
a = "355"          → is a string data type
a = int(a)         → got converted to integer
print(type(a))     → <class int>
print(a+5)         → 360
```

eg
```
a = 31             → int type
a = str(a)         → converted int to string
print(type(a))     → "31"
t = type(a)
print(t)           → <class string>
```

## * INPUT function

'Input' is a function. it takes input from user in the form of string.

∴ 'input' always take input in form of string

eg
```
a = input("Enter ur name")   → Enter ur name
print(type(a))               → <class 'string'>
print(a)                     → vishu
```
→ //input will take the input in string format

eg
```
a = input("Enter")           → enter
a = int(a)                   → converting a from str to int
print(type(a))               → <class 'int'>
```

* : it is imp to note that the output of input is always a string even if numbers are entered.

# CH 3 : STRINGS

## #1 INTRO

### * STRINGS

- strings is a sequence of character
- string is a datatype in python
- you can write string in ' ' or " " or ''' ''' or """ """
- string characters can be accessed like indices of array
- you can write multi line string using ''' '''

eg:

```
str = ''' hello, hey
          I am Vishwajeet
          Gaikwad '''
```

- string in python are immutable
- immutable : मतलब existing string को हम change नही कर सकते
- len() : using len() function you can find length of string
- eg:

```
str = ''' hello World '''
length = len(str)
print(length)  // Output : 11
               // Length of "Hello World" string will be printed
```

- name = "Vishu" // length = 5

vishu =

| v | i | s | h | u |
|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 |

index: 0 1 2 3 4

-ve index: -5 -4 -3 -2 -1

# #2 STRING SLICING

## * String slicing

- A string in python can be sliced for getting a part of the string
- slicing मतलब string को cut करना
- Syntax :

variable_name [ index_starting : index_ending ]

Start index से end index तक slice करके string देगा & string immutable है ∴ they cannot be changed once created, ∴ store the sliced string in new variable

- eg :

```
fruit = "mango and apple"
length = len(fruit)
mango = fruit[0:5]   // including 0th character but not 5th
apple = fruit[10:15] // including 10th character but not 15th
xyz = fruit[-3:-1]
abc = fruit[len(fruit)-3 : len(fruit)-1]
print(mango)
print(apple)
```

→ both are same, python automatically puts len(fruit) in -ve numbering

Output : mango
apple

- negative indicies can also be used as shown in below

-1 corresponds to length -1

-2 corresponds to length -2

## * SLICING WITH SKIP VALUE

- We can provide a skip value as a part of our slice
- syntax:

```
variable_name = [start_index : end_index : skip_indexes]
```

Starting index — ending index — Jump_index or no. of characters to be skip

- eg:

```
word = "amazing"
new_word = word[1: 6: 2]   // output = "mzn"
print(new_word)
```

Output:

"mzn"

## * Advance Slicing techniques

- if start index is not mention, then assum it as 0
- if end index is not mention, then assum it as 0

- eg:

```
word = "amazing"
newword = [ :7]   // [0:7] = amazing
oldword = [1: ]   // [1:7] = mazing
newword = word[ :]   // it means [0:len(word)]
```

by default python assums this

# #3 STRING FUNCTION

## * STRING FUNCTION

- There are many built-in fun() of string in python some of them are

### * len()

- This len() function returns the length of string.
- eg:

```
str = "vishu"
print (len(str))   // output = 5
```

### * str.endswith ( )

- This fun() tells wether the variable (passed to it) string ends with the string passed as parameter or not
- it returns true if the string ends with passed string value
- eg:

```
name = "vishwajeet"
ans = name.endswith("jeet")
print (ans)
```

Output = true

### * str.startswith ( )

- This fun() tells wether the parameter passed to it, starts with the string or not
- it is similar to endswith()
- returns true or false

### * str.capitalize ()

- This method will capiatalize the first letter of the sentence and returns a new string
- eg:

```
name = "vishu gaikwad"
newname = name.capitalize ()
print (newname)
```

Output :

Vishu gaikwad

## * str.upper/lower()

- it will change the string into upper/lower case & return a new string, it won't change the exsicting string bcoz strings are immutable

- eg:

```
str1 = "AabcdefFGH"
str2 = str1.upper()
print (str2)
```

Output = AABCDEFFGH

NOTE

- strings are immutable, मतलब exisisting string को string methods change नही करेंगे
- str1 को str1.upper() uppercase मे change नही करेगा, but str1.upper() एक new string return करेगा जो uppercase मे होगी

## * str.rstrip()

- it will strip the exclamentary mark only occuring at the end

- eg:

```
name = "Vishwajeet !!!!!"
newname = name.rstrip("!")
print (newname)
```

Output = Vishwajeet

## * str.replace()

- The replace method will replace all occurances of a string with another string

- eg:

```
name = "Vishwajeet Gaikwad Vishu"
newname = name.replace("Vish", "Gaikwad")
print (newname)
```

Output:

Gaikwadwajeet Gaikwad Gaikwadu

* str.count()

- Counts the total no. of occurances of any character
- eg:

```
name = "Vishwajeet"
no = str.count(e)
print (no)
```

output:
2

* str.center()

- The center() method aligns the string to the center as per the parameter given by the user
- eg:

```
str1 = "welcome"
print(len(str1))
str2 = (str1.center(50)) //it will add 50 spaces
print (len(str2))
```

Output = 7
57

* str.find (word)

- This fun() finds a word and returns the index of first occurance of that word in the string
- eg:

```
str = "Fun Coding is best"
index = str.find ("is")
print (index)    // Output = 12
```

* str.replace (oldword, newword)

- This fun() will replace old word with new word & returns a new string becoz. string are immuteable

# #4 FSTRING & DOCSTRING

## * F-STRING

- F-string is used in string formating
- String formating : मतलब हम variables को string के बिच मे रख सकते है
- you can use F-string & you can put the variables in b/w the string like we do in JS using backticks (``)
- sytax :

```
Print(F" {variable-1} string. {variable-2} string ")
```

These are variables already defined in pgm

you can also use no. and do arthemetic operation on them here

- eg :

```
country = "India"
name = "Vishu"
print(F" Hey my name is {name} and I am
        from {country} ")
Print(F" my marks are {2*30} percent ")
```

output = 60 percent

- if you want to show the "{name}" string as it is in curly Brackets then use {{ }} double curly brackets for this
- eg :

```
country = "India"
name = "vishu"
print(F" Hey we can use F-string as {{name}} and
        country is {{country}} ")
```

Output :

Hey we can use F-string as {name} and country is {country}

// :. when you want curly brackets use double curly bracket to print them

# * DOC - STRINGS

- python DOC strings are the string literals that appear right after the defination of a fun(), method, class or module
- ~~[illegible]~~ you can write additional information inside fun(), somewhat like comments, but doc-string are different from comments
- doc-string are not ingnored by compiler in python
- Syntax

```
print(variable_fun_name.__doc__)
```

- eg:

```
def square(n):
    '''Takes in a no. n, returns the square.'''
    print(n**2)

square(5)
print(square.__doc__)
```

This is doc string

→ // this will print the doc string

→ // if you don't print the doc string, then it won't be shown in terminal

- doc-string should be written just below the fun, as the fun start
- doc-string should be in the first line of function / class
- if you write it below some code which is inside fun then the doc-string will be none and if there is something in ''' ''' cotes then that will act as comments
- you can also write doc-string in class
- if you do

```
import this
```

then Zen of python will get printed, which is a poem by "Tim Peters"

# CH : 4 CONDITIONAL STATEMENT

## (iii) IF - ELSE

### * if - else

- if - else statement is used to perform a task based on some condition
- syntax :

```
if (condition) :
        #code that should be executed
        # if condition is true
elif (condition 2) :
        # code that should get execute
          if condition 2 is true
else :
        # code if both the conditions
        # are not true
```

- you will either enter in if, elif or inside else
- you can't get inside all the 3 blocks
- the code should be written inside the if, elif or else block, by giving one tab space (indentation)
- only any one of the condition will become true, if you enter in if block then other block (elif, else) will not get execute
- elif in python means else if
- अगर if के अंदर गया code तो baki else को ingnor करेगा, becoz if कि condition true हो गई
- eg:

```
a = input("Enter your age :")
if (a > 18) :
        print("you are above age of consent")
elif (a < 0) :
        print("entered an invalid age")
else :
        print("below age of consent")
```

- if you have only if statements and that too multiple if statements then pgm will enter in each of the if ; if that condition is true inside if

- eg:

```
a= input("Enter your age")

# if statement no: 1
if (a%2==0):
        print("a is even")
# end of if statement no : 1

# if statement no : 2
if(a>=18) :
        print("you are 18 ")

elif ( a<0):
        print("-ve age Entered")

else :
        print (" you are not 18 ")

# end of if statement no : 2
```

→ // it will go inside this if and this if is independent of other ∴ both if will get execute

→ // this if will also get execute

→ but if the 'if' condition above is true then code will not enter in elif and neither else will get execute

NOTE

- if there are only if statements one below the other all will get execute
- if there there is ladder then only one of 'if' or 'elif' or 'else' will get execute

# #2 MATCH CASE / SWITCH CASE

## * MATCH CASE

- match case is like switch case in c/c++
- you can pass a variable and according to the value of that variable you can perform which case you want to execute
- **NOTE** :- in C/C++ you have to write break after each case so that other case won't execute
  - but in python no need to write "break" after each case
  - मतलब in python जो case match होगा वही case सीधे Run होगा (no need of "break" key word)
- match case is noth but a other way of if-else
- syntax :

```
match variable :        → // you can pass any datatype variable here which will have values

    case value :
        // code

    case value :
        // code

    case value
        // code

    case _ if ( ) :     → // default case with if block
        // code

    case _ :            ← if none of the case gets execute then this default case will get execute
        // default case if none of the case is true
```

# CH: 5 LOOPS

## (III) LOOPS

### * LOOPS

- Sometimes we want to do Repetative work, ∴ we can use loop to loop some task
- There are following types of loop in python
  1) while loop
  2) for loop

### * while loop

- syntax

```
while (condition):
    //code
    //task to do repetatively
```

- The while loop will loop the code inside it if the condition is true
- it will loop until the condition becomes false
- eg:

```
i = 0
while (i<6):
    print(i)
    i+=2
```

- In while loop the condition is checked first. If it evaluates to true, the Body of the loop is executed.
- If the loop is entered, the process of condition check & execution is continued until the condition becomes false

## * FOR LOOP

- A for loop is used to iterate through a sequence like list, tuples, or string
- Syntax:

```
for i in range(start_iterator, end_iterator)
```

OR

```
str = "vishwajeet"
for i in str:
    print(i)
```

## * range()

- range(): fun takes 3 parameter same as string slicing
- The first parameter means from where to start loop
- The second parameter means ~~for~~ on which no, or after how many count end the loop
- We have to specify start, stop & step_size in range

  range(start, stop, step_size)

- eg:

```
list1 = [1, 4, 6, 234, 6, 764]
for i in list1:
    print(i)  ---> # 1, 4, 6, 234, 6, 764

tup = (6, 231, 75, 122)
for i in tup
    print(i)  ---> # 6, 231, 75, 122

str = "vishwajeet")
for i in str:
    print(i)  ---> # v i s h w a j e e t
```

## * For loop with else

- An optional else can be used with a for loop if the code is to be executed when loop exhausts
- मतलब जब code for loop से बाहर आएगा, exit करेगा तब तो else को execute करेगा
- मतलब for loop खतम होने के बाद else वाला part execute होगा

- eg:

```
for i in range (7):  ---> // automatically this loop
    print(i)              will run from rang(0,7)
                          that is it will print 0 to 6
else :
    print (" loop over")
```

## #2 BREAK, CONTINUE & PASS

### * BREAK

- 'Break' is used to break a loop, or come out of the loop
- मतलब "Break" अगर loop मे मिला तो loop खतम हो जाएगा

- eg:

```
for i in range(100):  ---> // for i in range (0,100)
    Print (i)  ---> //will print only 0 to 30 & then
                      loop will break
    if (i==30):
        break  ---> # Exit the loop right now

print (" loop over")
```

## * CONTINUE

- Continue is used to stop the current iteration of loop & continue with the next one.
- using "Continue" you can skip the current iteration
- eg:

```
for i in range(100):
    print("printing")
    if(i%2 == 0):        → // check for even no
        continue
    print(i)             → // only odd no will get print
print("out of loop")
```

## * PASS

- pass is a null statement in python, it instruct to do nothing
- मतलब अगर तुम्हे for loop मे कोई code नही लिखना तो तुम्हे "pass" लिखना पडेगा, without writing pass you will get indentation Error
- eg:

```
i = 1
str = "vishwajeet"
for i in str:
    pass                 → // without writing pass you will get Error
while (i<11):
    print(i)
    i = 2*i
```

- pass means, when you don't have to do anything in for loop, then write "pass"

# CH: 6 LISTS & TUPLES

## #1 LISTS

- list is a data type in python and list are mutable
- we use list to store multiple data values in single variable
- list can be iterate like array indexes
- python list are container to store a set of values of any data type
- you can create list ~~of~~ containing different datatypes values
- syntax:

```
list_variable = ['value1', 'value2', 'value3' ...]
```

- eg:

```
marks = [98, 99, 23, 48, "fail", "pass", 91, 10]
// list can contain diff datatype values
print(len(marks))
```

Output : 8

- list is like container, which can store multiple values
- eg:

```
marks = [3, 5, 6, "Vishu", True]
if 7 in marks:
    print("Yes")
else:
    print("No")    //Output : No
```

- you can use "in" keyword to see wether any value is present in list or not
- you can use "in" in strings also to check wether any string or word is present in string or not
- eg:

```
print(marks)        → // print all marks list
print(marks[:])     → // print marks like string sliding
```

- eg: lst = [i for i in range(4)] // Loop will run for 4 time
  print (lst)
  Output = [0, 1, 2, 3]

- List can contain another lists

## * List Comprehensen / for / IF

- you can use for loop & if statements inside list, so that elements are inserted in list using or based on some condition

- eg:
  lst = [i*i for i in range (4)]
  print (lst)
  Output = [0, 1, 4, 9]

- eg:
  lst_2 = [i*i for i in range(10) if (i%2==0)]
  print (lst_2)
  Output = [0, 4, 16, 36, 64]

  // taking values based on if () & using loop

## * List methods

- List methods are inbuilt fun used to manupulate list
- List are mutable मतलब जब हम list methods को list पर apply करेंगे तो वो list (जो existing list है) उसे manupulate/change करेगे

## * lst. append (value)

- append () methods adds the value in list at the end
- eg:

```
lst = [11, 15, 17, 19]
lst.append (21)
print (lst)
```

Output: [11, 15, 17, 19, 21]

## * lst. sort ()

- sort() fun will sort the list in accending order
- eg:

```
lst = [11, 15, 21, 5, 100, 2]
lst. sort()
print (lst)
```

Output: [02, 05, 11, 15, 21, 100]

## * l.reverse ()

- reverse () will reverse the list
- eg:

```
lst.reverse ()
Print (lst)
```

output: [2, 100, 5, 21, 15, 11]

## * l.sort (reverse == true)

- this will sort in desending order

* l.index(element)

- index() fun returns the index of the element you pass if present
- it will give index of first occurance of that element
- eg:

```
        0  1  2  3  4   = indexes
l = [1, 5, 25, 0, 2]
ind = l.index(2)
print(ind)
```

Output:

4

* l.count(element)

- count() fun will count the no. of occurance of the element you pass & return the count
- eg:

```
c = l.count(1)
print(c)
```

Output: 1

* l.insert(index, Element)

- insert() will add the Element specified at the index passed
- eg:

```
l = [2, 4, 0, 3]
l.insert(2, 10)
print(l)
```

Output:

[2, 4, 10, 0, 3]

* l.pop(index)

- pop() will ~~delement~~ delete element at passed index & return that element

* l.remove(ele)

- remove will remove the passed/parameter element from list

## * Refrencing IN python

- if you copy the list by creating Another variable/element then, the List is not coppied
- Only the address is copied or you can say that you have 2 variables now pointing to same memory location or address
- ∴ if you manupulate any one variable then both the variables will be manupulated becoz you are changing values stored at that memory address which is pointed by 2 variables
- eg:

```
l = [10, 2, 3, 4]
print(l)          → // output : [10, 2, 3, 4]
m = l
m[0] = 0
print(l)          → // output : [0, 2, 3, 4]
print(m)          → // output : [0, 2, 3, 4]
```

## * l.copy()

- copy() fun will copy the list from one list variable to a diff list variable which is pointing diff memory locations
- eg:

```
l = [10, 20, 30, 40]
m = l.copy()
m[0] = 0
print(l)          → // [10, 20, 30, 40]
print(m)          → // [0, 20, 30, 40]
```

# #2 TUPLES

## * Tuples

- A tuple is an immutable data type in python
- tuple is similar to list but list are mutable
- tuple stores multiple data items in single variables
- tuple cannot be changed
- tuple can store data items of diff data types in single variable
- tuples items are seperated by ',' and are enclosed by round brackt
- Syntax:

  tuple_variable = (item1, item2, item3 . . .)

- eg:

```
a1 = () ——> Empty tuple
a2 = (1,) ——> // tuple with single element should have ',' at end
a3 = (1, 7, "vishu") ——>
tup = (1, 7, "vishwajeet")
tup [0] = 0 ——> // gives error becoz you can't change tuple (immutable)
```

- you can print tuples using index also like strings, list, array
- you can use "if in" in ~~tu~~ tuples also like list to check wether some element is present in tuple or not
- eg:

```
tup = [1, 7, 6, "vishwajeet"]
if 6 in tup :
    print ("Yes")
else :
    print ("No")
```

- you can do slicing in tuples like list or string

## * tuple Methods

- you can convert tuple to list by following code
- eg:

```
tup = [1, 4, 69, 71, "Vishu"]
lst = list (tup)
lst.append ("Ruvi'a")
tup = tuple (lst)
print (tup)
```

- so you can convert tuple to list, apply & manipulate all list methods on that converted tuple (to list) & again you can convert that list back to tuple
- tuple method are same as string methods, they return new tuple

## * t.count (element)

- count () method counts the no. of occ of the element passed
- eg:

```
tup = (0, 1, 2, 2, 2, 3, 4)
no = tup.count (2)
print (no)
```

output : 3

## * t. index (element)

- index () returns the index of the element passed
- index (element, start_index, end index)
- eg:

```
no = tup. index ( 2, 3, 6)
print (no)
```

output : 4

# CH : 7 Dictionary & Sets

## #1 SETS

### * SETS

- Set is a data type which contain only unique items
- Set is a collection of non-repetive elements
- if you want to collect names from array which has repetative elements & we just want unique entries then we can store it in set
- you can do everything as you can do in List
- Sets are mutable, you can change the elements of set
- you can store ~~dat~~ multiple data types within a set inside curly bracks
- dictionary & set have same syntax, they start with curly bracks
- Syntax :

```
s = {2, 4, 2, 6}  ---> // set contains only unique element
print (s)
Output
    = {2, 4 6}
```

- eg :

```
info = {"Carlo", 19, 69, false}
print(info)  ---> // {Carlo, 19, 69, false}

st = {}  ---> // This is emty dictionary not set
st_2 = set()  ---> // This is set
print (type(st))  ---> //dictomary
print (type(st_2))  ---> // set
```

- you can't access set elements using indices like string, you can use "for in" loop to access them

## * Properties of SETS

- Sets are unordered, Elements order does'nt mater
- Sets are unindexd, meaning, you cannot access elements of sets by indexes like we do in string
- There is no way to change item in sets
- Sets cannot contain duplicate value

## * Methods of SET

### * s1.union(s2)

- you can do union of two sets
- union() fun Returns a new set with all unique items from both sets
- eg:

```
s1 = {1, 2, 3, 3}
s2 = {1, 5, 6, 2, 2}
s11 = s1.union(s2)
print (s11)
```

Output:
{1, 2, 3, 5, 6}

### * s1.update(s2)

- update() fun will update s1
- मतलब s1 मे s2 के values भी आ जाएंगे, Basically union होगा & s1 मे store हो जाएगा
- update() fun will update the set on which it is called

### * s1.intersection(s2)

- intersection() fun will do intersection of two set & return a new set with all intersection values of two set
- eg:

```
s3 = s1.intersection(s2)
print (s3)
```

Output:
{1, 2}

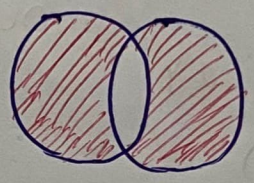
## * s1.intersection_update (s2)

- This intersection_update() fun will find intersection & update the s1 set on which it is called
- eg:

```
s1 = {1, 5, 6, 7}
s2 = {1, 1, 6, 8, 9, 9}
s1.intersection_update (s2)
print (s1)  ---> // output: {1, 6}
print (s2)       {1, 6, 8, 9}
```

## * s1.symmetric_difference (s2)

- symmetric_difference () fun finds the elements which are not common to both sets and returns a new set
- s1 ∪ s2 − s1 ∩ s2



semantic_diff

eg:

```
s3=s1.symantic_difference (s2)
print (s3)
```

Output: {5, 7, 8, 9}

## * s.remove (element)

- remove () takes an element and removes the element from sets
- eg:

```
s1 = {1, 2, 4, 4}
s1.remove (4)
print (s1)
```

Output: {1, 2}

- This remove() method updates the OG set bcoz sets are mutable

* s1.pop()

- pop() fun removes an random arbitrary element from the sets and returns the element removed

* s1.clear()

- it will Empty the s1 set

* s1.isdisjoint (s2)

- isdisjoint() fun returns boolean value if s1 set and s2 set are disjoint
- disjoint sets means, they don't have any element in common

* s1.issuperset (s2)

- it returns true/false, if the set s1 is superset of s2 or not

* s1.add (element)

- you can add () a element in s1 set using add fun
- it will add element in OG set

# #2 DICTIONARIES

## * DICTIONARIES

- Dictionary is a collection of key value pairs
- Like a dictionary (oxford) has words as key and value as the meaning, you can also create a key/value pair
- dictionary is mutable in python (you can change the key value pairs in OG dictionary)

- Syntax :

```
dictionary_variable = {
        "key" : "value",
        "key2" : "value",
        "list" : [1, 2, 3, 4]
}
```

- you can store multiple data types inside same dictionary
- eg :

```
dic = {
        344 : "Vishu",
        023 : "shubham",
        "list" : [1, 2, 3, 4]
}
print(dic[344])
print(dic)
```

- dictionary is ordered collection of data items
- you can access the value of dictionary as :

dic_name[key] ——→ error occurs if key is not present

dic_name.get(key) ——→ returns none if key is not present

- get( ) is a fun which takes key as parameter and returns its value if present

- you can get all keys of dictionary by
- eg:

```
info = {
    'name' = "vishu",
    'age' = 20,
    'Id' = 99
}
print (info.keys())
for key in info.keys():
        print(info[key])
print (info.values())
```

print keys

Output:

dict_keys(['name', 'age', 'Id'])

vishu
20
99

print values

// vishu.
20
99

- you can get the key value pair using items()
- eg:

```
print ( info.items ()) // print all key value pairs
for key, value in info.items ():
        print ("value corresponding to key {key} is
            {info[key]} or {value} ")
```

output:

name = 'vishu'
age = 20
Id = 99

# * Methods in Dictionary

## * d1.update(d2)

- This update method will take a dictionary as parameter and adds that dictionary to d1
- As dictionaries are mutable ∴ the OG dictionary d1 will get updated
- eg:

```
d1 = {122 : 45 , 123 : 89 , 169 : 670}
d2 = {222 : 77 , 333 : 90}

d1.update(d2)   // Output:
print(d1)          122 : 45
                   123 : 89
                   169 : 670
                   222 : 77
                   333 : 90
```

## * d1.clear()

- it will delete all the elements from dictionary & an empty dictionary will be there

## * d1.pop(key)

- pop() method will take out the key/value pair & return it
- you have to pass key as parameter
- eg:

```
d1 = {
    122 : 45
    123 : 89
    169 : 69
}
d1.pop(122)
print(d1)
```

Output:

```
123 : 45
169 : 69
```

## * d1.popitem()

- This method will remove last key/pair value from dictionary, no need to pass any parameter, & return it

## * del d1[key]

- del keyword is used to delete something in python
- del d1[key] : will delete the key/value pair, passed as key parameter will get deleted
- if you don't mention any 'key' then whole dictionary will be deleted
- eg:

```
d1 = {
    122 : 45, 699 : 69, 123 : 45
}
del d1[122]  --> // this will delete 122 : 45 Entry
del d1       --> // This will delete the Entire dictionary d1
print(d1)
```

## * d1.items()

- items() returns a list of key/value pairs

## * d1.key()

- returns all keys of dictionary d1

## * d1.get(key)

- get() fun returns the value of a specific key passed as parameter.

# CH : 8 function & RECURSION

## #1) FUNCTION

### * Function

- A function is a group of statements performing a specific task
- A fun can be used in a program any no. of times
- Syntax :

```
def fun_name ():
        //code to be executed
```

OR

```
def fun_name (parameters) :
        //code to be executed
```

- Function call : whenever we want to call (execute) a fun we need to call it using name of fun followed by parenthesis ()
- function defination : it is the main body or logic of fun.
  - The part containing the exact set of instruction which are executed during fun call.
- eg :

```
def avg():
    a = input("Enter a no")
    b = input ("Enter b no")
    average = (a+b)/2
    print (average)
```

//function defination

```
avg() ---> // function call
```

## * Types of function

There are two types of fun in Python

1) Built in fun: These are fun which are already present in python by default
   - eg: print(), len(), range() etc.

2) User defined fun: function made by user are called User defined fun
   - eg: avg() etc

## * USER DEFINED fun ()

1) fun with Argument/parameter

→ A fun can accept some values it can work with. We can put these values/parameters in the parenthesis

eg:

```
def greet (name):
    gr = "Hello" + name
    print (gr)
greet ("vishu")
```

//output:
Hello vishu

2) fun with Return value

→ A fun can also return value after performing some operation

eg:

```
def avg (a, b, c):
    average = (a+b+c)/3
    return average
ans = avg (25, 10, 25)
print (ans)
```

//output:
20

- you can return string, dictionary, list, set etc as a return value, even you can send all this as parameter also

## * fun with DEFAULT PARAMETER/ARGUMENT

- We can have a value as default argument/parameter in a fun
- मतलब अगर हमने fun call मे parameter values pass नही की तो हम by default parameters की value set कर सकते है
- eg:

```
def greet (name = " stranger")
        print (" Hello " + name )

greet ()                    // output: Hello stranger
greet ("vishu")                     Hello vishu
```

# #2 RECURSION

## * RECURSION

- Recursion means when a fun calls itself
- we can use Recursion to make factorial fun
- मतलब जब एक fun, अपने defination मे खुद को ही call करे तो उसे Recursion बोलते है
- factorial(n) = n x factorial(n-1)
- eg : factorial(5)

```
factorial(5)
↓
5 x factorial(4)
↓
4 x factorial(3)
↓
3x factorial(2)
↓
2 x factorial(1)
↓
1 x 1
```

eg:

```
def factorial(n):
    if(n==0 or n==1):
        return 1

    return n * factorial(n-1)

n = input("Enter a no")
ans = factorial(n)
print("factorial of", n, "is", ans)
```

- Ensure that you don't stuck in finfinite loop of calling fun

# #3 LAMBDA Function

## * Lambda function

- lambda function is a way of of writing functions without name
- lambda functions are anonymous functions (without name) & it is defined using "lambda" keyword
- Syntax :

lambda argument_variables : expression

for calling fun you need to use this variable & pass parameters to this variable it will behave like function name ← variable = lambda arguments : expression

↳ This argument is noth but the function parameters

- use case : lambda functions are commonly used as arguments (parameter) to pass to high order function such as map, filter, & reduce
- eg1:

```
def square_fun (x):
    return x*x
```

or

```
ans = lambda x : x*x
print (ans(2)) → // in this ans is noth but name of function
ans(4) // calling of lambda fun, this "ans(4)" will get evalute as 16
```

or

```
m = ans(4) // you called the lambda fun & pass 4 to it, it returns 16
print (m) // it will print 16
```

- eg2:

```
def fun_1 ( fx , val1, val2, val3 )
    return 10 + fx(val1, val2, val3)
```

→ fx is a function, we are using lambda function to pass function as parameter

```
avg = lambda x,y,z : (x+y+z)/3
final_ans = fun_1(avg , 5, 10, 15)
print(final_ans)
```

→ we are passing function as parameter & its parameters also

or

```
print (fun_1 (avg , 5,10, 15))
```

**NOTE:**

- you can pass function as a parameters to other functions
- high order function : These are function which take function as parameter

# (#4) MAP, FILTER & REDUCE

~~*1) MAP~~

## * map, filter & Reduce

- map, filter, & Reduce functions are built in function & are also present in JS
- These functions allows you to apply a function to a sequence of elements (list, sets etc) & return a new sequence from that OG sequence
- These functions are known as high order functions bcoz they take functions as parameters (These parameter functions are noth but lambda functions)

## * map

- map function applies a function to each element of a sequence (like list, set, string etc) & returns new sequence
- This function returns a new sequence (like list, set, string etc) containing the transformed elements.
- Syntax:

```
map (function , iterable_sequence)
```

function argument, This is a function that is applied on each element of the iterable_sequence argument

This argument can be list, Tuple, string

```
eg1: def cube (i):
        return i*i*i
     l = [1, 2, 4, 6, 4, 3]
     newl = map (cube , l)
     print (newl)
//output : <map object>
     print (list (newl))
//output: 1, 8, 64, 216, 64, 27
```

- you can also send lambda fun to it
- This is the function cube send as parameter
- The parameters of cube will be the elements of list & it will automatically send to cube

l is list, on each element of list "l" that function will get apply

* filter

- filter functions filters a sequence of elements based on the function passed to it.
- The function which is passed as parameter to filter function is called predicate
- Predicate : function passed as parameter should return a boolean value T/F, so that filter function can decide wether to add that value/element or not
- filter function returns a new sequence (list, set etc) which includes the filtered elements
- Syntax:

filter ( function , iterable)

This passed function should return a boolean

This is noth but a sequence like list, & each element of this sequence is passed as parameter to the lambda function send as parameter to filter

eg:
```
l = [1, 2, 4, 6, 4, 3]
def filter_fun (a) :
        return  a>2 //returns T or F

newl = ~~list~~ filter( filter_fun , l)
print (newl) // out : filter object
print (list (newl)) // output :[4, 6, 4, 3]
```

output is all the elements from list which are greater then the function condition

or

```
l = [1, 2, 4, 6, 4, 3]
fun_1 = lambda a : a>2 // This lambda function should return T or F only
newl = filter (fun_1, l)
print (newl)  // output : lambda fun
print (list (newl)) // output : [4, 6, 4, 3]
```

## *reduce

- reduce function is a high order function that takes a function & sequence as parameter & then apply that function on the sequence elements & returns a single value
- "reduce" function is a part of "functools" modules, which is a built in module & you need to import it
- Syntax :

reduce (lambda_function , iterable)

it will apply this fun on each element & return a single value

↳ Can be list, tuple, string

eg: from functools import reduce

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

sum = lambda x, y : x+y

it will be the previous return value for reduce function & y will be the next element

~~ans = sum~~

ans = reduce (sum , l)

print (ans) // output = 15

// working

[1, 2, 3, 4, 5] ⟹ sum (1, 2)

[3, 3, 4, 5] ⟹ sum (3, 3)

[6, 4, 5] ⟹ sum (6, 4)

[10, 5] ⟹ sum (10, 5)

15 → return 15

# #5 ENUMERATE FUNCTION

## * Enumerate function

- Enumerate function is a built in function that allows you to loop over a sequence (like list, tuple, string) and get the value/element and index of that element both at same time
- मतलब अगर हमे एक list के elements के साथ साथ उस element का index भी देना है directly. Just हमे "enumerate" keyword use करना है.
  Generally C++ मे हमे map use करके ऐसे behaviour मिलता है
- Syntax :

```
for i, ele in enumerate(sequence)
```

this will help us to get both element & index of that element

this "i" is variable for index

ele is variable for elements of that sequence

sequence can be name of list, string or tuple etc

eg:
```
marks = [10, 20, 32, 45, 65, 95]
for index, m in enumerate(marks):
    print(index, " = ", m)
    if (index == 5 or m == 95)
        print("Congratulation topper")
```

in this for loop, you can access the index of the elements of list using ~~index~~ "index" variable

// you can also specify diff that ~~yo~~ start indexing from a custom no like:

eg:
```
for i, m in enumerator(marks, start=1):
    print(i, " = ", m)
```

## 46 LOCAL V/S GLOBAL VARIABLE

* **Local variable**: this is the variable which is defined inside a specific function
  - you can only access this variable inside the function & its scope is only till inside the function
  - if you have local & global variable with same name & you want to access the global variable inside the the function where local variable is also present with same name, then make use of "Global" keyword

* **Global variables**: The variables which can be accessed from anywhere in the pgm

* **Global keyword**: if you use "global" keyword then you can access global variable inside function who has same name local variable

- eg:

```
x = 10 ---> // global variable
def fun_1():
    x = 20 ---> // local variable
    print (x)
    global x = 100
    print (global x)   ] // not allowed, you will get Error
fun_1()
print (x)
```

- eg 2:

```
x = 10 ---> // global variable
def fun_1()
    global x ---> // this tells that "x" is the global x
    x = 100
    print (x)
fun_1()
print (x) //output : 100
```

~~* Destructor~~

## #7 *args & **kwargs

***args** : it is a tuple, & its used inside a function to take in dynamic no. of arguments or parameters

- *args lets your function or constructor to accept any no. of positional arguments without explicitly defining each one
- it is useful for function overloading or constructor overloading

```
eg: def funargs ( x , *args):
        print ("msg is : ", x)
        for i in args :
            print (i)
```

this "*" means its a tuple & this should always come at the end or as last parameter.

```
har_variable = [ "Hari", "Vicher", "Gaikwad" ]
x = "Hello this is args demo"
funargs (x ,*har_variable)
```

store it as tuple or list & pass it while calling

while calling also you need to pass it using "*"

- syntax :

```
def fun_name (parameter1, para2 , *args):
    // your code

fun_name (parameters , *tuple)
```

- you can pass tuple or list while calling the function
- & using elif you can design constructor overloading

****kwargs** : this is noth but dictionary, it works same as *args but it takes key value pair

- **kwargs lets your function accept any no. of key-value pair arguments
- when you don't know Exact no. of arguments, then you can use this for dynamic arguments accepting inside a function or Constructor

eg:
```
def demo(a, b, **kwargs):
    print(a, b)
    for key, val in kwargs.items():
        print(key, ":", value)
```

you need to use "**" it tells that, it is a dictionary variable

```
a = 69
b = "Hello 69"
dic = (name = "vishu", age = 23, country = "India")
demo(a, b, **dic)
```

while calling also, you need to pass dictionary using "**"

# CH : 09 OOPs

## (#i) INTRODUCTION

### * Procedural Programming

- Procedural Programming means program that uses a step-by step approach to solve problems
- Programs written (without OOPs concept) by using fun, for loops, dictionary etc is called procedural Programming
- programs written till now, without mapping real world Entity & without using class - object concept are called procedural programming

### * Object Oriented Programming (OOPs)

- In OOPs we will map variables/data with real world entities
- eg: Enmies in game
- We will have classes & object using which you can create fun() & its data
- This way of writting program using classes - object (inheritance, Abstraction, Encapsulation, polymorphism) is called OOPs
- FEATURES OF OOPs :

  1) class - objects : blue print having data & methods
  2) Abstraction : data is private
  3) Encapsulation : implementation details are hidden
  4) inheritance : phone में drone के features add कर दिये
  5) polymorphism : one thing having many forms