SIMATS

School of Engineering

PYTHON PROGRAMMING

Computer Science Engineering

Saveetha Institute of Medical And Technical Sciences,Chennai.

UNIT-I

INTRODUCTION TO PYTHON

Def: Python is a general-purpose interpreted, interactive, object-oriented, and high-level Programming language

* Interpreted - Processed at runtime by interpreter
* Interactive - interact with the interpreter directly through a python prompt.
* object-oriented - Encapsulates code within objects.

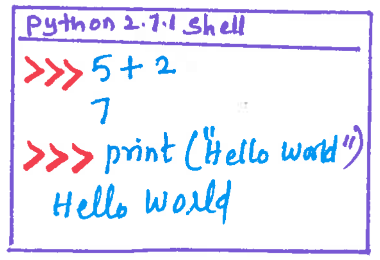
Features of Python

* easy-to-learn
* easy-to-maintain
* Portable
* extensible
* Free and open Source.
* High-level language
* Scalable.
* interface to database
* GUI Programming.

Modes of python Interpreter

Interactive Mode

Interpreter displays the result immediately.

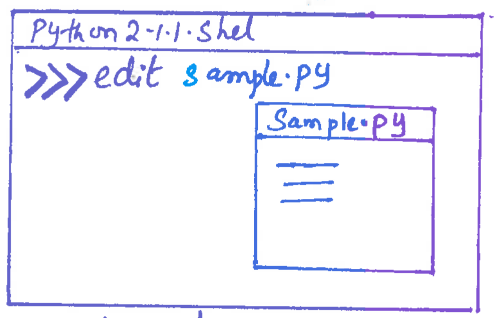


```
Python 2.7.1 Shell
>>> 5+2
7
>>> print ("Hello world")
Hello World
```

* commands and expressions are directly executed at prompt
* Cant save and edit the code
* can see the results immediately

Script Mode

Type the program in a file with (.py) extension and then use interpreter to execute.



```
Python 2.1.1 Shel
>>>edit sample.py
```

* Read and execute statement in a script
* Can save and edit the code
* cannot see the results immediately,

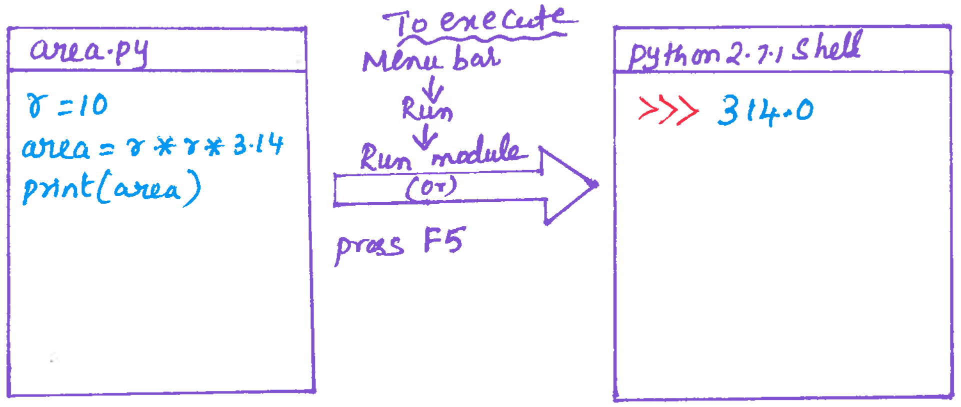
IDLE

* Integrated Development Learning Environment.
* Graphical user interface written in python
* Bundled with default implementation of the python language.

Features of IDLE

* Multi-window text editor with syntax highlighting.
* Auto completion with Smart indentation

To create a python file → Filename.py



```
area.py
r=10
area=r*r*3.14
print(area)
```

```
python 2.7.1 Shell
>>> 314.0
```

Input Statements: → inbuilt function to read the input from user

Addition of 2 numbers

```
a = int(input("Enter the first value"))
b = int(input("Enter the second value"))
c = a+b
print(c)
```

O/P

```
Enter the first value 2
Enter the Second value 3
Enter the
5
```

* input function reads the input as string value by default. Explicit conversion is required to read as int

Values:

* Value can be any letter, number or string

Examples: 2, 2.5, 'Hello world'

Variables:

* Named place in the memory in which the values can be stored and can be retrieved for later use.
* Name of the variable is user-defined
* Values of Variables can be changed.

Example: x = 12.2
y = 14
x = 10

x ~~12.2~~ 100

y 14

Reserved words:

* Reserved words / Keywords cannot be used as variable names / identifiers.

and, del, for, is, raise, assert, elif, from, lamda, return, break, else, global, not, try, class, if, while, for, def, print, import . . .

Identifiers:

* Names that identify the elements such as variables and functions in a program.
* a sequence of characters that consists of letters, digits and underscores (_)
* must start with a letter or an underscore.
* cannot start with a digit.
* cannot be a keyword.
* can be of any length.

Assignment Statements:

* The statement for assigning a value to a variable is called an assignment (operator) statement. operator → Equal sign $\boxed{=}$

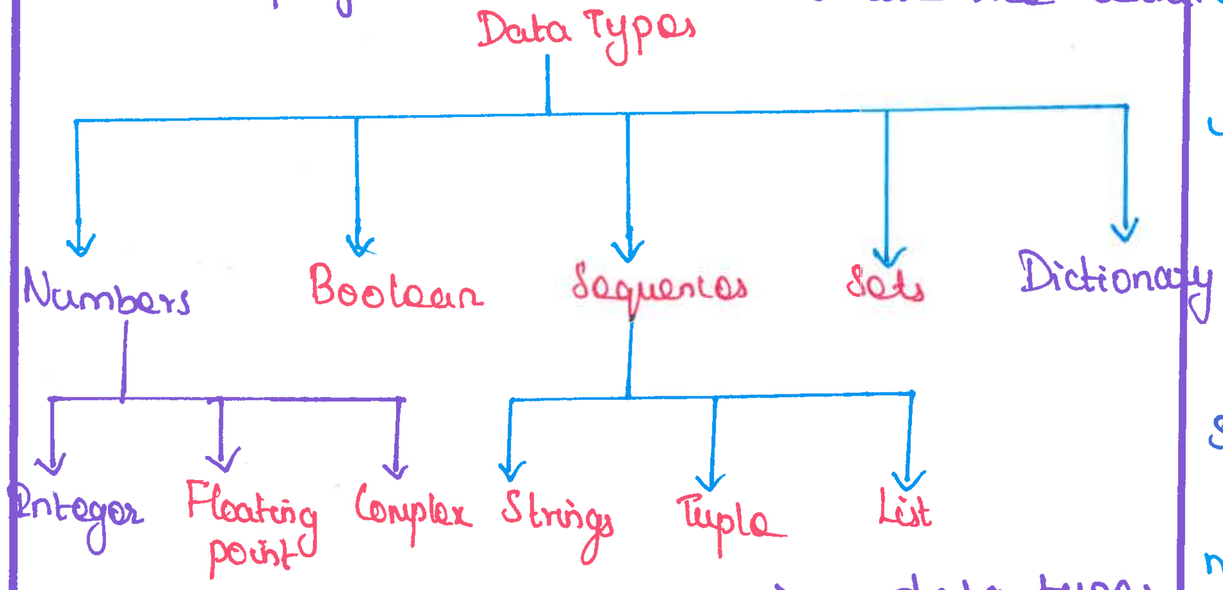
Syntax: Variable = expression.

* multiple names can be assigned at same time & chained. Ex: >>>x, y = 2, 3
>>>a = b = x = 2

DATA TYPES

DATA TYPES

* A datatype tells the compiler or interpreter how the programmer intends to use the data.



* In Python programming, data types are classes and variables are instances of those classes.

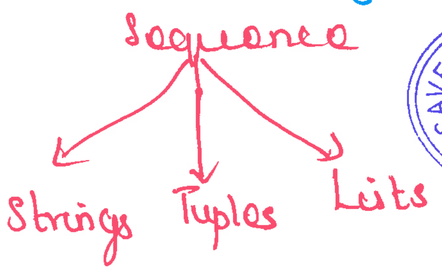
Numbers:

* stores numerical values.
* immutables [ie, values /items cannot be changed]

Integers	Float	Complex
⇒ represented as 'int' ⇒ positive or negative whole numbers with no decimal point. Eg: 56	⇒ Written with a decimal point dividing the integer and the fractional parts Eg : 56.778	⇒ They are of the form a+bj, where a + b are floats and j represents the square root of -1. Eg: square root of -1 is a complex number

Sequences:

* Ordered collection of items indexed by positive integers.
* combination of mutable and immutable data types.



Strings:

* series or sequence of letters, numbers and special characters.

⇒ single quotes (' ')

Eg: 'This is a string'

⇒ double quotes (" ")

Eg: "This is a string"

⇒ triple quotes (""" """)

Eg: """ This is a paragraph. It is made up of multiple lines and sentences."""

Tuple:

* consist of collection of values separated by commas.
* enclosed in parenthesis ()
* immutable

Eg: >>> a = (1,2,3,4,5,6)

List:

* contains items separated by commas
* mutable
* enclosed within square brackets []

Eg: >>> a = [1,2,3,4,5,6]

Boolean:

* has two values → 0 + 1
* 0 represents False
* 1 represents True

Eg:
```
>>> 3 == 5
False
```

Sets:

* collection of items that are unordered and unindexed.
* written with curly brackets

Eg: >>> a = {'apple', 'orange', 'grape'}

Dictionary:

* used to store data values in key : value pairs.
* ordered, mutable
* do not allow duplicates

Eg: car = { "brand": "Ford", "model": "Mustang", "year": "1964" }

Expressions:

* Combination of operators and operands that is interpreted to produce some other value.

Types:

1. Constant expressions ⇒ Eg: $x = 15 + 1.3$ (15 → operand1, + → Operator, 1.3 → operand2)
2. Arithmetic expressions ⇒ Eg: x = 40, y = 12, add = x + y
3. Integral expressions ⇒ Eg: b = 12.5, c = x + int(b)
4. Floating expressions ⇒ Eg: c = x / y
5. Relational expressions ⇒ Eg: c = x > y
6. Logical expressions ⇒ Eg: P = (10 == 9), Q = (7 > 5), R = P and Q

FUNCTIONS

FUNCTIONS:

* block of statements that return the specific task.

* Functions helps to break the program into smaller and modular chunks.

* As the programs grows larger and larger, functions make it more organized and manageable.

* avoids repetition and makes the code reusable.

Syntax:

```
  Keyword   Function Name      Parameter
def   function_name(parameters):
      # statement  } ──→ Body of statement
      return expression
             ↓
       Function return
```

Example:1

```
def fun():
    print ("welcome to AI&DS")

fun()
```

Output: welcome to AI&DS

Example:2

```
def add (num1, num2):          ↑ Function Name  ──→ Parameters
    print ("Number1:", num1)     ┐
    print ("Number2:", num2)     │ Function
    addition = num1+num2         │ Body
    return addition ──→ return value ┘
res = add (2,4) ──→ Function call
print (res)
```

FLOW OF EXECUTION

* top to bottom ie, execution always begins at the first statement of the program

* function is not executed untill the function is called

```
def add (a, b):              a   b       step2
step3 ──→ c=a+b
    return c
a=5 // Execution starts here
b=4                          step4
d=add (a,b)
print d
step1
```

PARAMETERS

* The variables that are defined when the function is declared ──→ a, b are parameters

Eg:

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

sum (1,2)
```

ARGUMENTS

* a value that is passed to a function when it is called.

* It might be a variable, value or object passed to a function.

Eg:

```
def mul (a,b):
    print (a*b)
mul (5,2) // 5,2 are arguments
```

MODULES

* refers a file containing Python definitions and statements

* defines functions, classes and variables.

* contains executable code

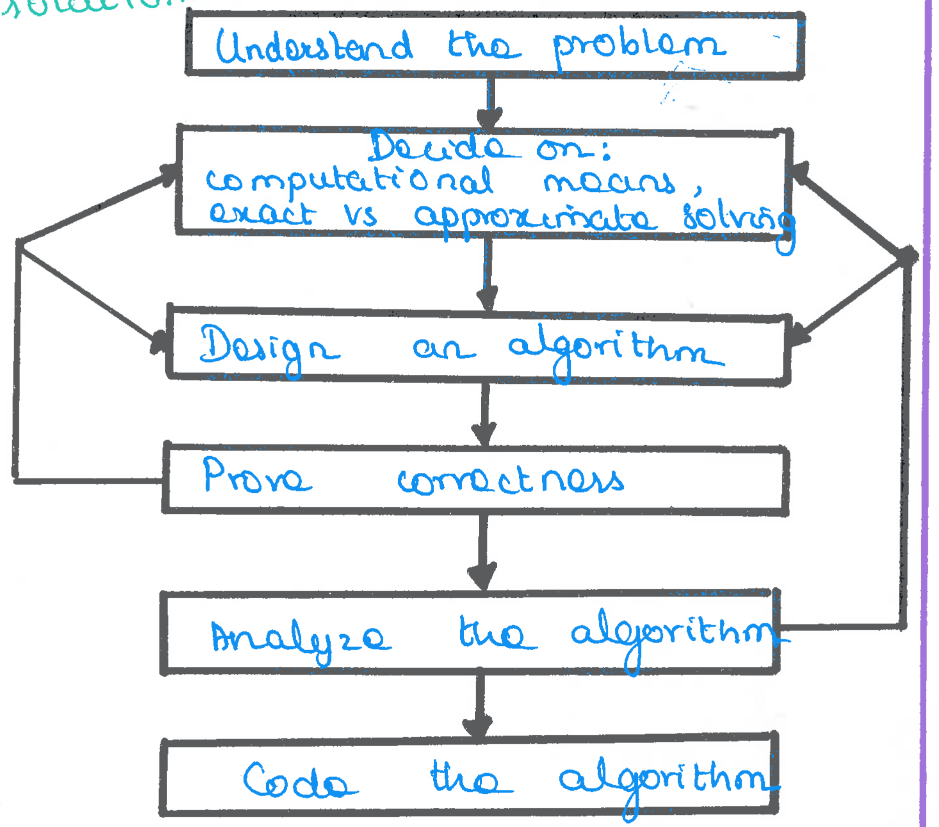
* makes the code easier to understand and use.

```
# A simple module, calc.py
def add(x,y):
    return (x+y)                import calc

def sub (x,y):                  print (calc.add
    return (x-y)                        (10,2))
```

ALGORITHMIC PROBLEM SOLVING

* solving problem that require the formulation of an algorithm for the solution.



Understand the problem

Decide on: computational means, exact vs approximate solving

Design an algorithm

Prove correctness

Analyze the algorithm

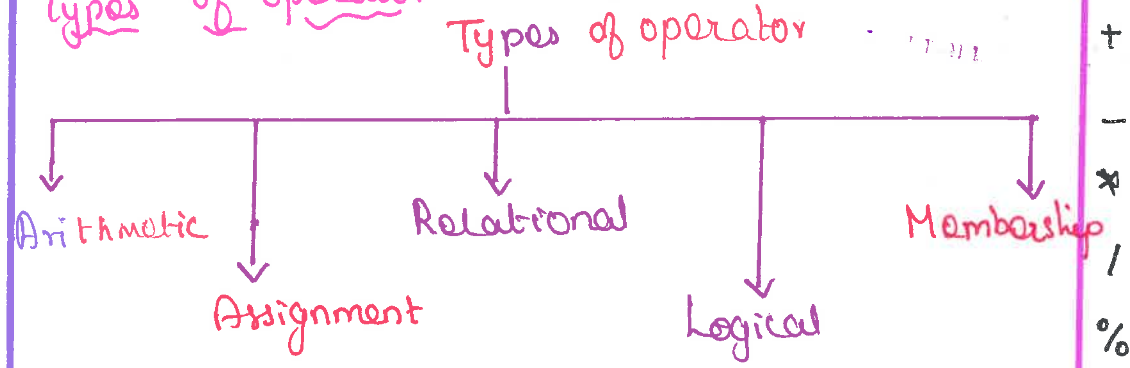
Code the algorithm

OPERATORS IN PYTHON

OPERATOR:

* a symbol that tells the compiler to perform specific mathematical or logical functions.

Types of Operator



Arithmetic operator: a=17 , b=2

Operator	Description	Example	Output
+	Addition	print (a+b)	19
-	Subtraction	print (a-b)	15
*	Multiplication	print (a* b)	32
/	Division	print (a/b)	8
%	Modulus ↳returns remainder	print (a%b)	1
//	Floor Division ↳ returns whole number quotient	print (a//b)	8
**	Exponent ↳ power	print (a**b)	289

Logical operator: AND, OR , NOT

A	B	A AND B	A OR B	not A
F	F	F	F	T
F	T	F	T	T
T	F	F	T	F
T	T	T	T	F

Assignment Operators:

Operator	Description	Examples
=	Assigns values from right to left	a=17
+=	Add AND	c+=a ⇒ c=c+a
-=	Subtract AND	c-=a ⇒ c=c-a
*=	Multiply AND	c*=a ⇒ c=c*a
/=	Divide AND	c/=a ⇒ c=c/a
%=	Modulus AND	c%=a ⇒ c=c%a
**=	Exponent AND	c**=a ⇒ c=c**a
//=	Floor Division	c//=a ⇒ c=c//a

Relational Operator : a=5 , b=2

Operator	Description	Example	Output
==	equal to	print (a==b)	False
>	Greater than	print (a>b)	True
<	Less than	print (a<b)	False
>=	greater than equal to	print (a>=b)	False
<=	less than equal to	print (a<=b)	False
!=	Not equal to	print (a!=b)	True

Membership Operator:

* Operator used to validate the membership of a value.

Eg: a= [5,1,8,7]

* Types

1. in operator ⇒ print (8 in a)
   output : True
2. not in operator ⇒ print (0 not in a)
   output : True

OPERATOR PRECEDENCE

Parenthesis
Power
Division
Multiplication
Addition
Subtraction
Left to Right

Example:

$3+4*4+5*\boxed{(4+3)}-1$

$3+\boxed{4*4}+5*7-1$

$3+16+\boxed{5*7}-1$

$\boxed{3+16}+35-1$

$\boxed{19+35}-1$

$\boxed{54-1}$

53

COMMENTS IN PYTHON

* not executed by compiler
* used for documentation of code.

Example

```
# This is a comment
print ("Hello, world!")
"""
This is a comment
written in more than just one line
"""
print ("Hello, World")
```

2.a

TOPIC: CONDITIONAL STATEMENTS

* Performs different computations depending on specific boolean constraint (True or False)

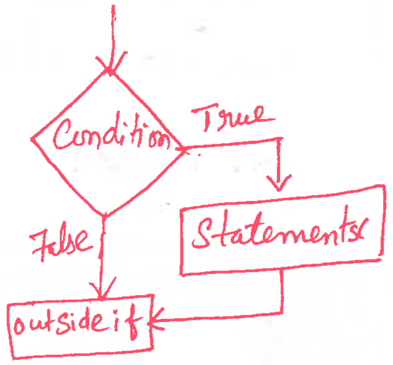
Conditional - if

* used to test a condition

* if the condition is true statements inside if will be executed.

Syntax:

```
if (condition 1):
    statement 1
    statement 2
        :
    statement 3
```



Example:

Program to provide flat Rs.500, if the purchase amount is greater than 2000

```
a=~~int~~(input("Enter the purchase amt"))
if(a >= 2000):
    a = a-500
print ("Amt to pay", a)
```

o/p

```
Enter the purchase amount : 2500
Amt to pay                : 2000
```

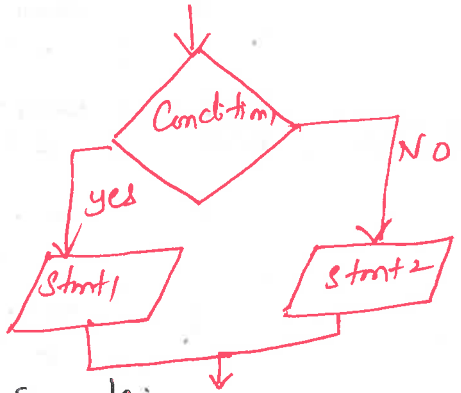
if .. else

* Used to test a condition, when the alternative is present.

* if the condition is true, statements inside the if gets executed otherwise statements inside else part gets executed.

Syntax:

```
if (condition1):
    statement 1
else:
    statement 2
```



Example:

Program to find the given number is odd or even

```
n=~~int~~(input("Enter the number"))
if(n%2 ==0):
    print ("even number")
else:
    print ("odd number")
```

o/p

```
Enter a number : 4
even number
```

SAVEETHA SCHOOL OF ENGINEERING
SAVEETHA INSTITUTE OF MEDICAL AND TECHNICAL SCIENCES
★ CHENNAI-602105 ★

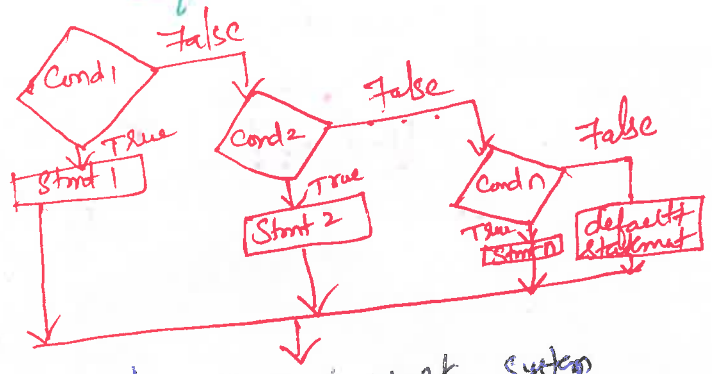
if .. elif .. else

* Used to check more than one condition.

* If condition1 is false, it checks the condition 2 of the elif block. If all the conditions are false, then the else part is executed.

Syntax:

```
if (condition 1):
    statement1
elif (condition 2):
    statement 2
elif (condition 3):
    statement 3
else: default statement
```



Example: Student Mark System

```
mark=~~int~~(input("enter ur mark"))
if(mark >=90):
    print("grade:S")
elif (mark >=80):
    print ("grade: A")
elif (mark >=70):
    print ("grade: B")
elif (mark >=50):
    print ("grade : C")
else: print ("fail")
```

o/p

```
Enter ur mark:78
  grade : B
```

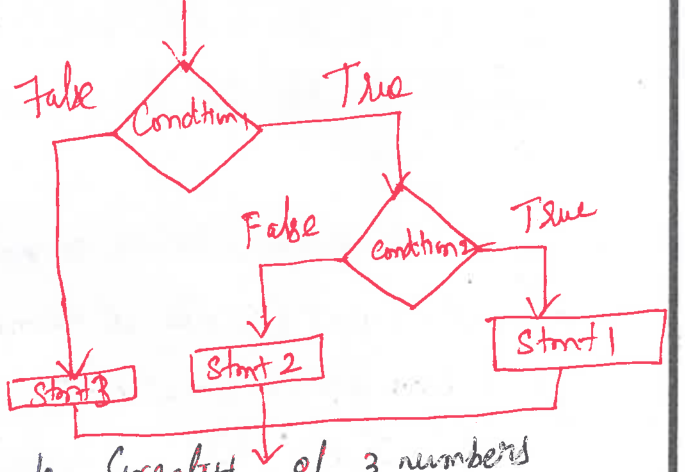
Nested if .. else

* Any number of condition can be nested inside one another

* If condition1 is true, it checks another if condition2. If both the conditions are true statement1 get executed otherwise statement 2 gets executed.

Syntax:

```
if (condition):
    if (condition 1):
        statement 1
    else:
        statement 2
else:
    statement 3
```



Example: Greatest of 3 numbers

```
a = input(" Enter the value of a")
b = input ("Enter the value of b")
c = input ("Enter the value of c")
if(a>b):
    if(a>c):
        print("The greatest ",a)
    else:
        print ("The greatest",c)
else:
    if(b>c):
        print("The greatest",b)
    else: print("The greatest",c)
```

o/p

```
Enter the value of a: 9
Enter the value of b: 1
Enter the value of c:8
The greatest 9
```

LOOPING STATEMENTS

Ref: allows to execute a statement or group of statements multiple times

FOR LOOP

- ✱ used to iterate over a sequence (list, tuple, string)
- ✱ Loop continues until the last element in the sequence is reached.

Syntax:

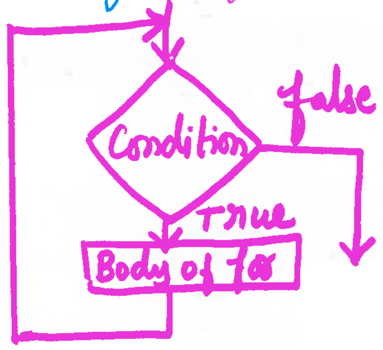
```
for i in sequence :
    print(i)
```

String	List	Tuple
Eg: for i in "Ramu": print(i) o/p: R a m u	Eg: for i in [2, 3, 5, 6, 9]: print(i) o/p 2 3 5 6 9	Eg: for i in (2, 3, 1): print(i) o/p 2 3 1

✱ Sequence of numbers can be generated using range() function.

Syntax:

```
for i in range(start, stop, steps):
    body of for loop
```



Example Program: Prime or not

```
n = int(input("Enter a number"))
for i in range(2,n,1):
    if(n % i = =0):
        print("The num is not a prime")
        break
    else:
        print ("The num is a prime number")
        break
```

o/p

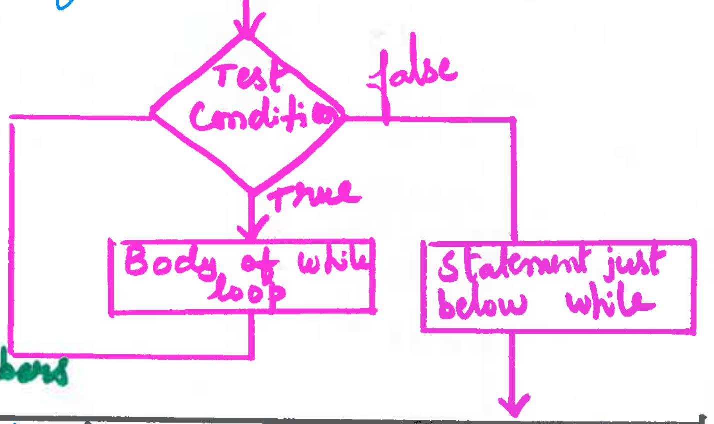
```
Enter a number  7
The num is a prime number
```

WHILE LOOP

✱ used to repeatedly execute set of statements as long as the given condition is true.

Syntax:

```
initial value
while(condition):
    body of while loop
    increment.
```



Example Program: Sum of n numbers

```
n=int(input("Enter n"))
i =1
Sum=0
while (i<=n):
    Sum = Sum+i
    i=i+1
print(Sum)
```

o/p: Enter n 10
55

o/p: Enter n 5
15

Iteration	Variable	i<=num	Body of the loop

1	num =5
	i =1
		True	Sum =1

2	num = 5
	i = 2
		True	Sum = 1+2=3

3	num =5
	i = 3
		True	Sum = 3+3=6

4	num = 5
	i =4
		True	Sum = 6 +4=10

5	num =5
	i=5
		True	Sum = 10+5 =15

6	num =5
	i =6
		False	exit the loop

Break

✱ It terminates the current loop and executes the remaining statement outside the loop

Example:

```
for i in "welcome":
    if(i = ="c"):
        break
    print(i)
```

o/p

```
w
e
l
```

Continue

✱ It terminates the current iteration and transfers the control to the next iteration in the loop.

Example:

```
for i in "welcome":
    if (i = = "c"):
        continue
    print(i)
```

o/p

```
w
e
l
o
m
e
```

Types of function

Built-in function (or) Pre-defined function [Library functions]

Eg: len() function

```
x=[1,2,3,4,5]
print(len(x))
```

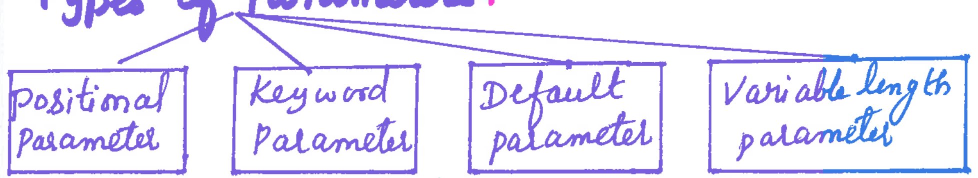
↳ returns length of list

User-defined functions [Defined by the programmer to reduce the complexity of big problems]

Eg:

```
x= 3
y = 4
def add():
    print(x+y)
add()
```

Types of Parameters:



1. Positional parameter

*Number of parameter in the function definition Should match exactly with number of arguments in the function call.

Eg:

```
def student(name,roll):
    print(name, roll)
student("Ram", 98)
```

o/p

```
Ram  98
```

2. Keyword parameter

* During the function call, the calling function identifies the parameters by the function's parameter name.

*order of the arguments can be changed.

Eg:

```
def student(name,roll, mark):
    print(name, roll, mark)
student(mark=90, roll=11078, name="Ram")
```

o/p

```
Ram, 11078, 90
```

3. Default parameter

*If the function is called without the argument, the argument gets its default value in function definition.

Eg:

```
def student(name, age=17):
    print(name, age)
student("Kumar")
student("ajay")
```

o/p

```
Kumar  17
ajay   17
```

FUNCTION

Function is a group of related statements that performs a specific task

4. Variable length parameter

*If the number of arguments to be passed, is not known in advance, asterisk (*) can be used before the parameter name to denote the variable length of parameter.

Eg:

```
def student(name, *mark):
    print(name, mark)
student("Ram", 98, 88)
```

o/p

```
Ram(98, 88)
```

Local and Global Scope

Scope: Refers to the places where it is declared, used and can be modified. It is the lifetime of the variable in the program.

Local scope:

*variable created inside a function belongs to the local scope of that function.

Global scope:

* Variable with global scope can be used anywhere in the program.

* variable defined outside the function.

Eg:

```
a=50          ⟶ Global Variable
def add():
    b=20      ⟶ Local Variable
    c=a+b
    print(c)
```

Return Values:

"return" keyword is used to return the values from the function.

Eg:

```
return a     - return 1 variable
return a,b   - return 2 variable
return a+b   - return expression
return 8     - return value.
```

Fruitful function:

* function that returns a value

Eg:

```
def add():
    a=10
    b=20
    c=a+b
    return c
c=add()
print(c)
```

o/p : 30

Void function:

* Function that performs action but don't return any value

Eg:

```
def add():
    a=10
    b=20
    c=a+b
    print(c)
add()
```

o/p: 30

Function Composition

* Ability of a function to call from within another function.

* Result of each function is passed as the argument of next function.

* output of one function is given as input of another.

Eg:

```
def add(a,b):
    c=a+b
    return c
def mul(c,d):
    e=c*d
    return e
c=add(10,20)
e=mul(c,30)
print(e)
```

o/p : 900

Recursion

* A function calling itself till it reaches the base value (stop point) of function call.

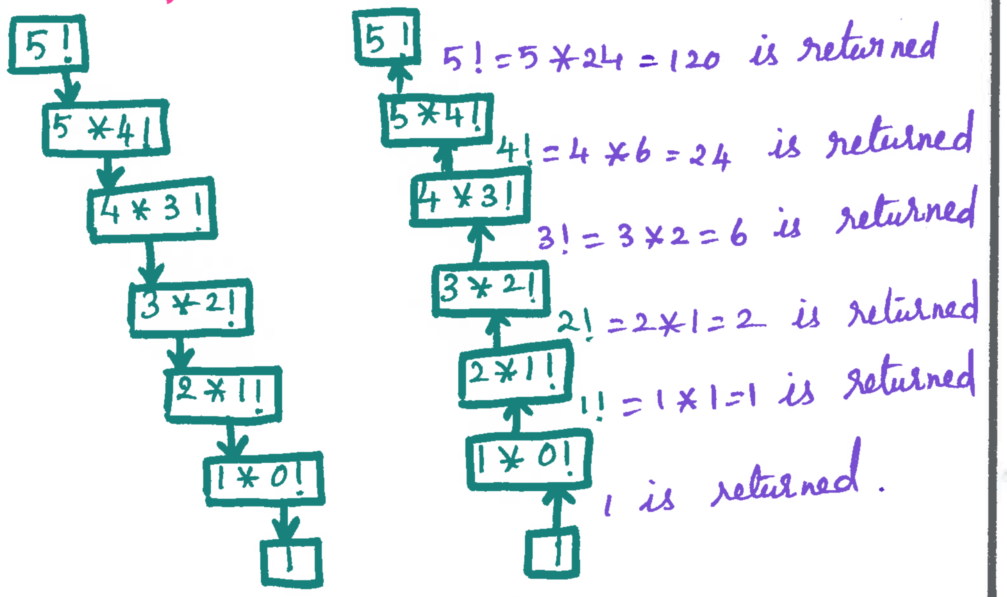
Eg: Factorial of n

```
def fact(n):
    if(n==1):
        return 1
    else:
        return n*fact(n-1)
n=int(input("Enter the number"))
fact=fact(n)
print(fact)
```

o/p

```
Enter the number 5
120
```

Final value = 120



STRING

Def : Sequence of characters represented in quotation marks Single quotes, double quotes

* immutable → contents of the string cannot be changed after creation.

* Python will get the input at runtime by default as a string .

Operations on string

1. Indexing :

* Individual character in a string is accessed using a index.

* Index must be an integer (positive or negative), and starts from 0 to n-1

String A	H	E	L	L	O
positive index	0	1	2	3	4
negative index	-5	-4	-3	-2	-1

Eg a = "HELLO"
print(a[0])
o/p : H

*Access the string from beginning

Eg print(a[-1])
o/p : O

* Access the string from end

2. Slicing :

* Extracting substring from a string.

* operator [start : stop] (or) [start : stop : steps]

Eg a = "HELLO"   o/p
print [0 : 4] – HELL
print[: 3] – HEL
print [0 :] – HELLO

3. Concatenation :

* operator '+' joins the text on both sides of operator

Eg : a = "save"
b = "earth"
print(a+b)
o/p saveearth

4. Repetition

* operator '*' repeats the string on the left hand side for the number of times given on the right hand side

Eg : a = "Saveetha"
print (2 * a)
o/p SaveethaSaveetha

5. Membership

* "in" operator check a particular character in string.

* "not in" operator check character is not in string

Eg : S = "good morning"
"m" in S   o/p True
"a" not in S   True .

Built-in-methods : a = "happy birthday"

1) a. capitalize() o/p 'Happy Birthday'

2) a. upper() o/p 'HAPPY BIRTHDAY'

3) a. lower() o/p 'happy birthday'

4) a. title() o/p 'Happy Birthday'

5) a. Swapcase() o/p 'HAPPY BIRTHDAY'

6) a. split() o/p ['happy', 'birthday']

7) a. count(substring) a.count('happy')
o/p 1

8) a. replace(old,new) a. replace('happy', 'wish you happy')
o/p wish you happy birthday'

9) a. join(b) b = 'happy'
a = "-"
a. join(b)
o/p = 'h-a-p-p-y'

10) a. isalpha() o/p = False

11) a. isdigit() o/p = False

12) a. startswith (substring) a. startswith("h")
o/p True

13) a. endswith (substring) a. endswith("y")
o/p True

14) a. find (substring) a. find("happy")
o/p 0
→ returns index if found
→ returns -1 if not found

15) len(a) o/p 14

16) min(a) o/p ' '

17) max(a) o/p 'y'

LIST

Def : Ordered sequences of items that can be different data types

* values in the list are called elements / items.

* Notation : [ ]

* Mutable → elements in the list can be changed.

Operations on list

1. Indexing : Eg: a = [2, 3, 4, 5, 6, 7, 8, 9, 10]
print(a[0])   o/p 2
print (a[-1])   o/p 10

2. Slicing : Eg: print(a[0: 3])   o/p [2, 3, 4]

3. concatenation : Eg: b = [20, 30]
print(a+b)   o/p [2, 3, 4, 5, 6, 7, 8, 9, 10, 20, 30]

4. Repetition : Eg: Print(b * 3)   o/p [20, 30, 20, 30, 20, 30]

5. Membership : Eg: 5 in a   o/p True
100 in a   o/p False
2 not in a   o/p False

6. Updating : Eg: a [2] = 100
print(a)   o/p [2, 3, 100 5, 6, 7, 8, 9, 10]

7. comparison : Eg: b = [2, 3, 4]
a == b   o/p False
a != b   o/p True

Built-in-methods : a = [1, 2, 3, 4, 5]

1) a. append(element) Eg: a. append(6)
print(a)
o/p : [1, 2, 3, 4, 5, 6]

2) a. insert(index, element) Eg: a. insert(0,0)
print(a)
o/p : [0, 1, 2, 3, 4, 5, 6]

3) a. extend(b) Eg : b = [7, 8, 9]
a. extend(b)
print(a)
o/p [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

4) a. sort() Eg : a. sort()
print(a)
o/p [0, 1, 2, 3, 4, 5, 6, 7, 8]

5) a. index(element) Eg : a. index(8)
o/p : 8

6) a. reverse() Eg : a. reverse()
print(a)
o/p : [8, 7, 6, 5, 4, 3, 2, 1, 0]

7) a. remove(element) Eg: a. remove(1)
print(a)
[8, 7, 6, 5, 4, 3, 2, 0]

8) a. pop() Eg: a. pop()
o/p 0

9) a. pop(index) Eg : a. pop(0)
o/p 8

10) a. Count(element) Eg: a. Count(6)
o/p 1

11) a. Copy() Eg: b = a. copy()
print(b)
o/p [7, 6, 5, 4, 3, 2]

12) len(list) Eg: len(a)
o/p : 6

13) min (list) Eg: min(a)
o/p : 2

14) max (list) Eg: max(a)
o/p : 7

15) a. clear() Eg: a. clear()
print(a)
o/p : [ ]

16) del (a) Eg : del (a)
print (a)
o/p : Error : name 'a' is not defined

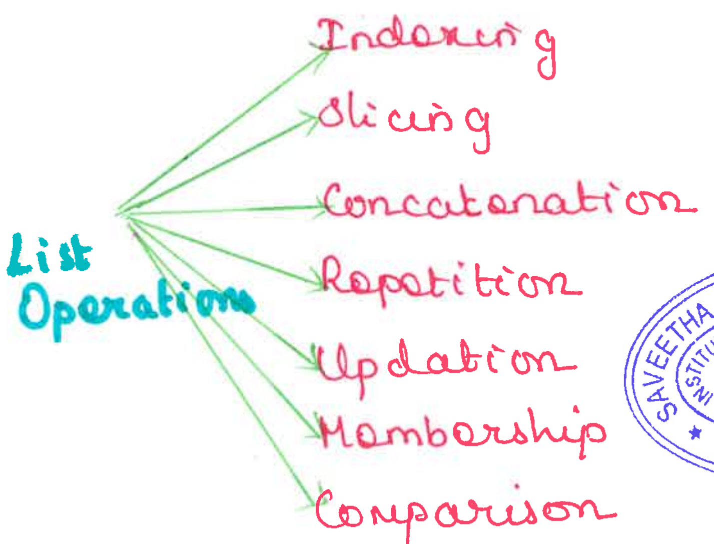
LIST

LISTS:

*ordered sequence of elements

*comma seperated values between squared brackets [ ]

Operations on Lists:



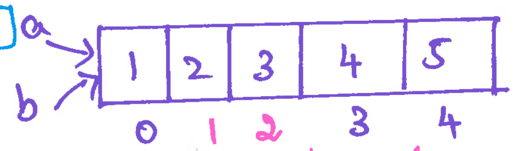
Working with Lists:

Aliasing (copying):

* Creating a copy of a list
* Same memory location
* Aliasing refers to having different names for same list values

Example:

```
>>> a = [1,2,3,4,5]
>>> b = a
>>> print (b)
    [1,2,3,4,5]
>>> a is b
    True
>>> a[0] = 100
>>> print (a)
    [100 ,2,3,4,5]
>>> print (b)
    [100,2,3,4,5]
```



* If the first element of "a" is replaced, then the first element of the "b" is also replaced.

CLONING:

* creating a copy of a same list of elements with two different memory locations.

* Changes in one list will not affect locations of another list.

cloning using slicing

```
>>> a= [1,2,3,4,5]
>>> b= a[:]
>>> print(b)  ⇒[1,2,3,4,5]
>>> a is b    ⇒ False
```

cloning using copy() method

```
>>> a = [1,2,3,4,5]
>>> b = a.copy()
>>> print(b)  ⇒ [1,2,3,4,5]
>>> a is b    ⇒ False
```

LIST LOOPS: "FOR"

a = [10,20,30,40,50]

Element	Index	Range
for i in a: print(i)	for i in range(0, len(a), 1): print (i)	for i in range (0, len(a),1): print (a[i])
o/p: 10 20 30 40 50	o/p: 0 1 2 3 4	o/p: 10 20 30 40 50

List using while loop:

* to iterate over a block of code as long as the test expression (condition) is true.

sum of elements in list

```
a=[1,2,3,4,5]
i=0
sum=0
while i<len(a):
    sum=sum+a[i]
    i=i+1
print(sum)
```

o/p: 15

LIST COMPREHENSION

* An elegant and concise way to create a new list or list from an existing list.

Example

```
>>> pow2 = [2**x for x in range(10)]
>>> print(pow2)
o/p ⇒ [1,2,4,8,16,32,64,128,256,512]
```

Multidemsional Lists

* Multi-dimensional lists are the lists within lists.

Example

```
>>> a = [[2,4,6,8,10],[3,6,9,12,15],
         [4,8,12,16,20]]
>>> print(a)
o/p ⇒ [[2,4,6,8,10], [3,6,9,12,15],
       [4,8,12,16,20]]
```

Accessing using square brackets

Eg:1

```
a = [[2,4,6,8], [1,3,5,7],
     [8,6,4,2], [7,5,3,1]]
for i in range(len(a)):
    for j in range(len(a[i])):
        print (a[i][j], end=" ")
    print()
```

```
O/p:⇒  2 4 6 8
       1 3 5 7
       8 6 4 2
       7 5 3 1
```

Eg:2

```
a=[[2,4,6,8],[1,3,5,7]]
b=[[8,6,4,2],[7,5,3,1]]
c=[[0,0,0,0],[0,0,0,0]]
for i in range(len(a)):
    for j in range(len(a[i])):
        c[i][j]=a[i][j]+b[i][j]
for i in range(len(a)):
    for j in range(len(a[i])):
        print(c[i][j],end=" ")
    print()
```

```
O/p:
[10 10 10 10
 8  8  8  8
```

TUPLE

Defn: * A tuple is same as list, except that the set of elements is enclosed in parenthesis instead of square brackets

* tuple is an immutable list

Conversion: list to tuple

```
list → a=[1,2,3,4,5]
      a = tuple(a)
      print(a)
tuple → (1,2,3,4,5)
```

tuple to list

```
tuple → a =(1,2,3,4,5)
        a= list(a)
        print (a)
list → [1, 2, 3, 4, 5]
```

Operations in tuple

Creating a tuple	a=(20,40,60,"apple","ball")
Indexing	print(a[0]) o/p: 20
slicing	print(a[1:3]) o/p: (40,60)
Concatenation	b=(2,4) print (a+b) o/p:(20,40,60,"apple","ball",2,4)
Repetition	print (b*2) o/p:(2,4,2,4)
Membership	a =(2,3,4,5,6,7,8,9,10) 5 in a o/p: True 2 not in a o/p: false
Comparison	a = = b o/p: false a != b o/p: True

Methods in tuple

a.index(tuple)	a=(1,2,3,4,5)
	a.index(5)
	o/p: 4

a.count(tuple)	a.count(3) o/p: 1
len(tuple)	len(a) o/p: 5
min (tuple)	min(a) o/p: 1
max(tuple)	max(a) o/p: 5
del (tuple)	del (a)

Tuple Assignment: Tuple assignment allows variable on the left of an assignment operator and values of tuple on the right of the assignment operator

(var1, var2,....) = (12,15, ...)

number of variables (LHS) must be equal to numbers of values (RHS)

Example: Swapping using tuple assignment

```
a =20
b=50                 o/p:
print(a,b)    → (20, 50)
(a,b) = (b, a)
print (a,b) → (50, 20)
```

DICTIONARY

* An unordered collection of elements. An element in a dictionary has a Key : pair value
* All elements in dictionary are placed inside the curly braces { }
* Elements in a dictionary are accessed via Keys and not by their position.
* The value of a dictionary can be any datatype.
* Keys must be immutable data type (numbers, strings, tuple)

operation	Create	Accessing an element	Update	add element	membership

example:	a={1:"one",
	2:"two"}
	print (a)
	o/p:{1:"one",
	2:"two"}
		a[1]
		o/p: "one"
		a[0]
		o/p: KeyError:0
			a[1]="ONE"
			print(a)
			o/p: {1:"ONE",
			2:"two"}
				a[3]="three"
				print (a)
				o/p: {1:"ONE",
				2:"two",
				3:"three"}
					a={1:"ONE",
					2:"two",
					3:"three"}
					1 in a
					o/p: True

Method	Example

a.copy( )	a={1:"one", 2:"two", 3:"three"}
	b= a.copy()
	print (b)
	{1:"one", 2:"two", 3:"three"}

a.items( )	a.items() dict_items([(1,"one"), (2,"two"), (3, "three")])
a.keys( )	dict_keys ([1,2,3])
a.values( )	dict_values (['one', 'two', 'three'])
a.pop(key)	a.pop(3) "three" print(a) {1:"one", 2:"two"}

setdefault(key, values)	a.setdefault(3, "three")
	print(a)
	{1: "one", 2:"two", 3:"three"}

fromkeys( )	key = {"apple", "ball"}
	value = "for kids"
	d= dict.fromkeys(key, value)
	print(d)
	{"apple": 'for kids', "ball": "for kids"}

len(a) o/p: 3

Example Program:

```
Marks = {'Ravi': [97,85,85,67],
         'Rahul': [92,91,94,85]}
tot = 0
Tot_marks = Marks.copy()
for key, val in Marks.items():
        tot = sum(val)
        Tot_Marks[key] = tot
print (Tot_Marks)
          {'Ravi': 334, 'Rahul': 362}
max =0
Topper = ''
for key, val in Tot_Marks.items():
     if (val > max):
          max = val
          Topper = key
print ("Topper is:", Topper, "with marks =", max)
Topper is : Rahul
```

Features of files

- * Permanently store data in memory
- * use files for future use of the data.

In python, file operation takes place in the following order:

1. Open a file
2. Read or write (perform operation)
3. close the file

Working of open() function

* open() function is to open a file in read or write mode.

* open() will return a file object.

* It accepts 2 arguments - filename & mode

Syntax:

```
open (filename, mode)
```

Basic modes of operation

- * r - reading
- * w - writing
- * a - appending
- * r+ - both reading and writing

* By default 'r' mode is used in python

Ex:

```
f = open("test.txt")
```

↳ opens file in the current directory

```
f = open("C:/python 33/Readme.txt")
```

↳ specifying full path.

Creating a file using write() mode

Eg:

```
file = open ("text.txt","w")
file.write ("Example program")
file.close()
```

FILES

File is a named location on disk to store related information. It is used to permanently store data in a non-volatile memory (eg: Hard disk)

close() - command terminates all the resources in use and frees the system of this program

read() - To extract a string that contains all characters in the file then we have to use read()

Eg1:

```
file = open("file.txt", "r")
    print(file.read())
```

Eg2: To read certain number of characters

```
file = open ("file.txt","r")
print file.read(5)
```

append() - To add few more data's to the existing file

```
file = open ('file.txt',"a")
file.write ("This will add this line")
file.close()
```

write() with "with()" function

```
with open ("file.txt", "w") as f:
    f.write ("Hello world!!!")
```

File Modes of operation

- *'x' → open a file for exclusive creation
- *'t' → open in text mode
- *'b' → open in binary mode
- *'+' → open a file for updating.

Format Operator

* The argument of write() function has to be a string, so if any other value has to be entered into a file, Conversion of these values to string is required. This is performed using "str"

```
>>> x = 52
>>> fout.write(str(x))
```

* Another way to implement is format operator %. This mod operator works as a format operator when the first operand is a string.

* First operand - format string, that consists of one or more format sequences, which specify how the second operand is formatted

Eg:

```
>>> camels = 42
>>>'%d' % camels
>>>'42'
```

$\underline{\text{'\%d'}}$ → formatted as integer

$\underline{\text{camels}}$ ↳ second operand

Eg: To count the number of lines in a text file

```
f = open ("sample.txt","r")
c = 0
for x in f:
    print(x)
    c = c+1
print ("The number of lines", c)
```

MODULES AND PACKAGES

MODULE

* File containing python statements and functions
* Module name is → File name . Py extension
* import

Eg: Sample.py - Filename

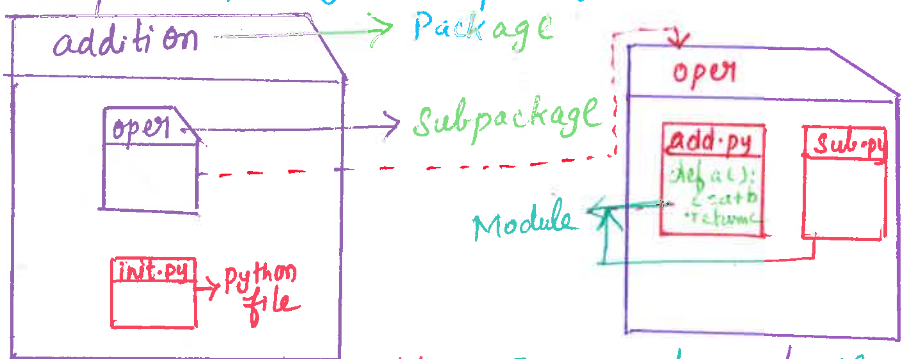
```
def linecount(filename):
        count =0
        for line in open(filename):
            count +=1
        return count
print linecount('Sample.py')
```

```
>>import sample
>>>print ~~Sample~~ sample
<module 'sample' from
  'Sample.py'>
>>>Sample.linecount
       ('Sample.py')
```

PACKAGES

* Collection of modules
* have subpackages and modules.
* Directoly with _init_.py file

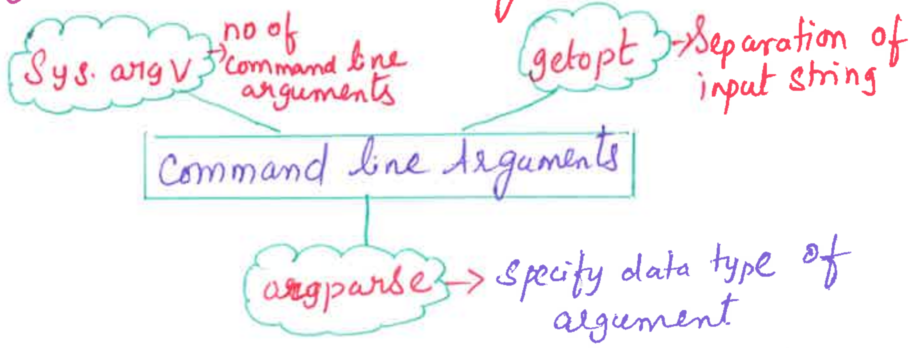
Eg: import package. Subpackage. module



import addition.oper.add ← To import package

COMMAND LINE ARGUMENTS

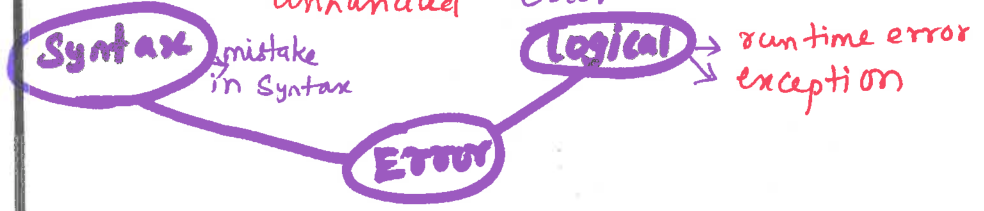
* Tent inteface
* Program that passes directly to the operating System

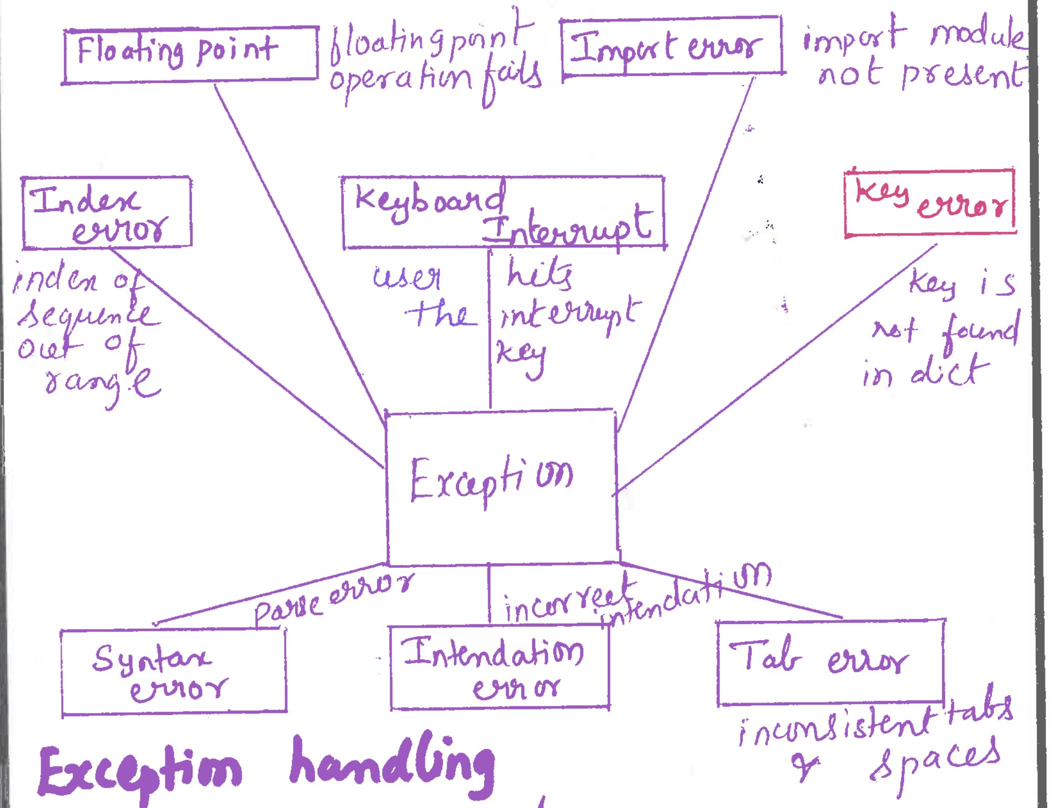


Sys.argV
no of command line arguments
getopt
Separation of input string
Command line Arguments
argparse
Specify data type of argument

EXCEPTION HANDLING

Error - critical problem occurs due to scarcity of system resources.
- unhandled error





Exception handling

↳ does not stop execution of program

→ changes normal flow

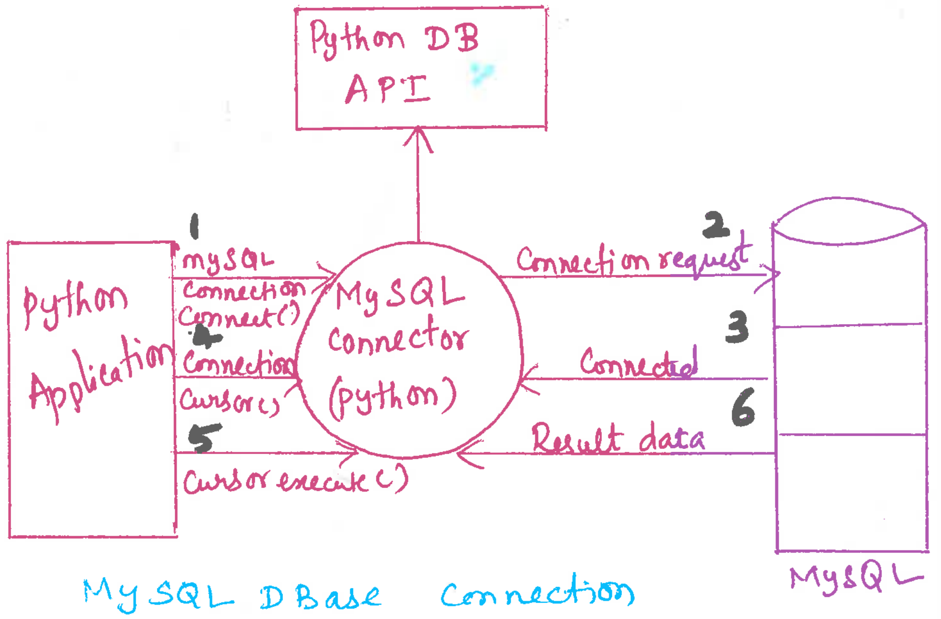
Try - clause , except clause

Syntax :

```
try :
    write the suspicious code here
except :
    Code that handles the exception
...
else :
    No exception, the statements here
    will get executed
```

MYSQL DATABASE ACCESS

Pip install Mysql connector ← installation



MySQL DBase Connection

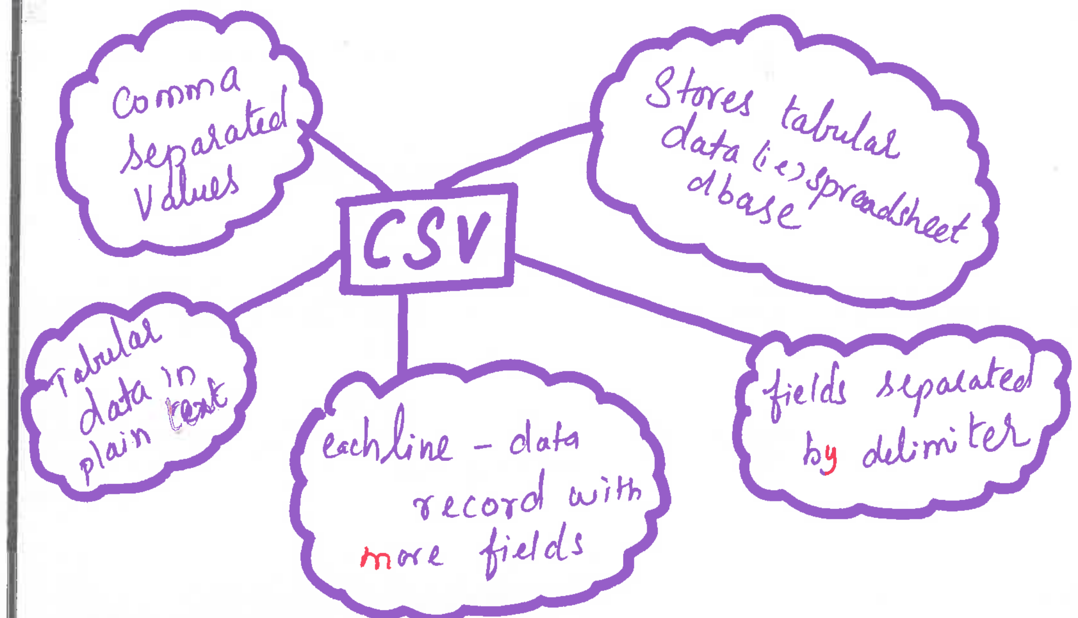
* Import API module
* Acquire connection with database
* Issue SQL statements
* close the connection

Create database:

```
import mysql.connector
mydb =mysql.connector.connect( host="local
        host", user ='root', password = 1234)
mycursor = mydb.cursor()
mycursor.execute ("Create database sample")
mycursor.execute ("show databases")
for i in mycursor :
        print(i)
# create table
mycursor.execute("create table customer(
     name  varchar(255),address varchar(255))
# insert into table
mycursor.execute ("insert into customers(
 name, address) value[("AAA", chennai),
 ("BBB", Kanchipuram)"]
mydb.commit()
```

CHENNAI-602105 * SAVEETHA INSTITUTE OF MEDICAL AND TECHNICAL SCIENCES * SAVEETHA SCHOOL OF ENGINEERING

CSV FILES


CSV
Comma separated Values
Stores tabular data (i.e) spreadsheet / dbase
Tabular data in plain text
eachline - data record with more fields
fields separated by delimiter

```
pip install csv
```
← Installation

Delimiter :

*) Separator character → (,) comma

*) in use → tab (\t), colon(:) and semicolon(;)

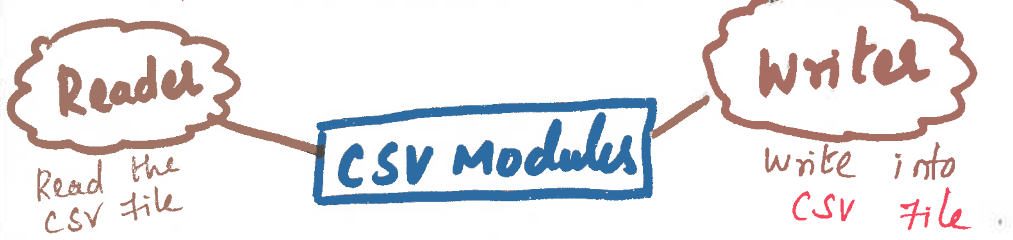
Example:

S.No.	Name	City
1	Michael	New Jersy
2	Jack	California
3	Donald	Texas

← Table

↓ CSV

```
S.No, Name, City
1, Michael, New Jersy
2, Jack, California
3, Donald, Texas
```



Example: Write into a csv file

```
import csv
csvData = [['Name of person', 'Age'], ['John', 22]]
with open('person.csv', 'w') as csvFile:
    writer = csv.writer(csvFiles)
    writer.write rows (csvData)
```

Output:

```
Name of Person, Age
John, 22
```

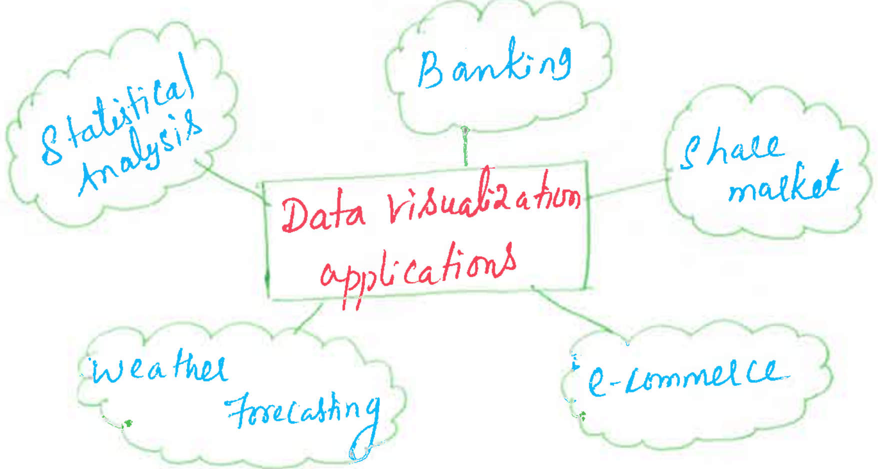
Read a csv file

```
import csv
with open ('person.csv','r') as csvFiles:
    reader = csv.reader (csvFile)
    for rows in reader
        print(row)
```

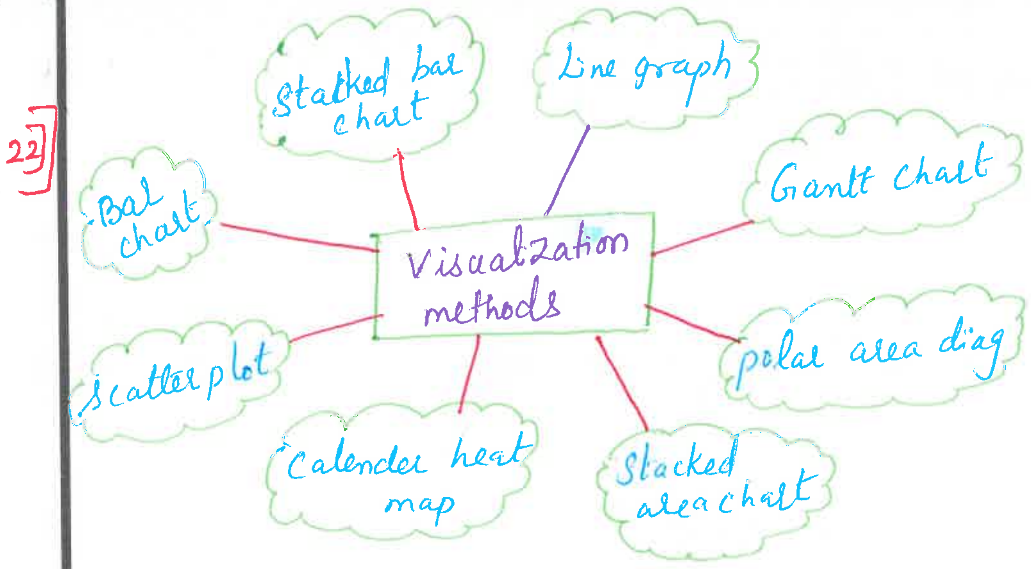
output: ['Name of person', 'Age']
['peter', '22']

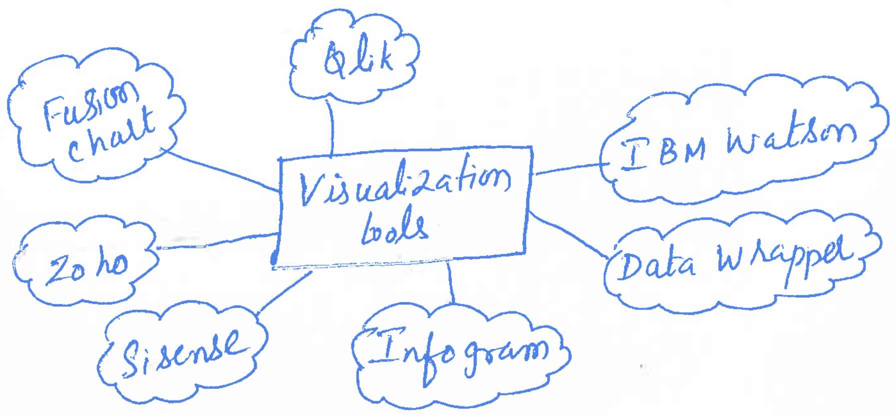
PLOTTING

* Graphical representation of information & data — Data visualization.



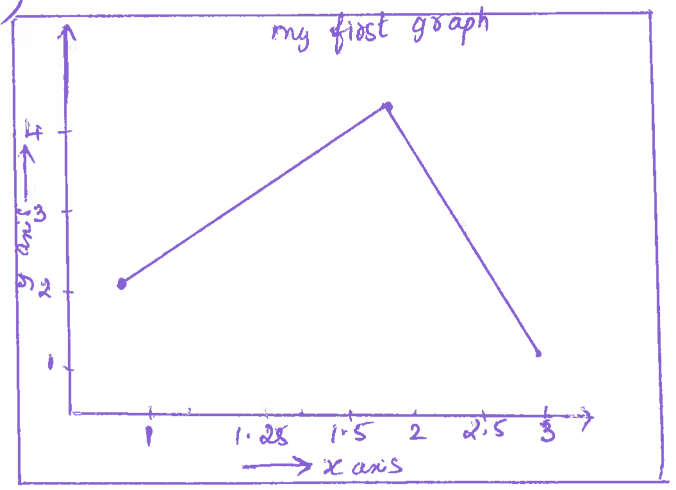
```
pip install matplotlib
```
→ Installation





Example : Plotting a line

```
import matplotlib.pyplot as plt
x = [1,2,3]
y = [2,4,1]
plt.plot(x, y)
plt.xlabel (x-axis)
plt.ylabel( Y- axis)
plt.title ("my first graph")
plt.show()
```



SAVEETHA SCHOOL OF ENGINEERING
SAVEETHA INSTITUTE OF MEDICAL AND TECHNICAL SCIENCES
CHENNAI-602105