Python Easy Notes

Notes

Abhishek Bohare

a_bohare@icloud.com

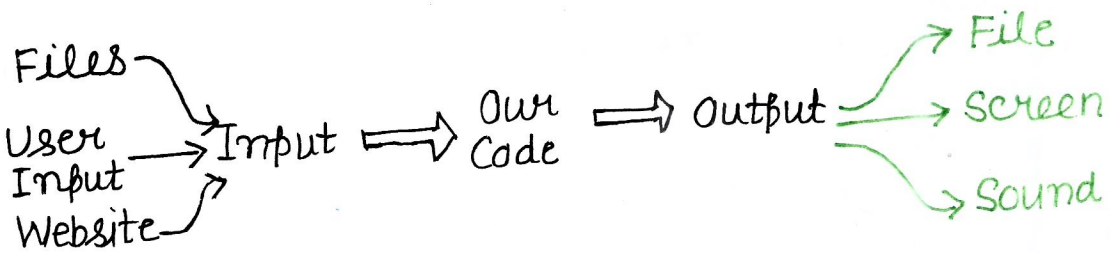
typeset with LaTeX



PROGRAMMING LANGUAGE

- CS1301XL
- Computing
- Programming
- Debugging
- Procedural Programming
- Variables
- Logical Operators
- Mathematical Operators

- Computing is a massive field that touches almost every corner of modern society in some way.
- Programming
  ↓
  Writing the commands
  ↓
  code
- Speak code to communicate with computer.

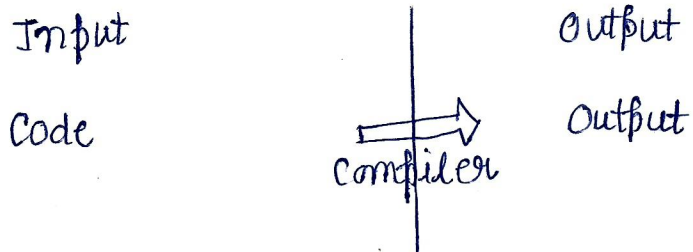


Bricks ⟹ Hard Work ⟹ House/Building

- Compilers ⟹ Find error ⟸ User work
  ⇩ ⇩
  Run ⟺ Error remove

#Static → Language that requires compilation.
#Dynamic → Language that do not require compiling.
#High Level → Great deal of abstraction.
#Low Level → Manual Works.

→ New approaches → Foundations
→ Language
→ domain



COMPUTING

Variables → Blue
Function → Purple
Reserved Words → Red
User Input data → Bold Red TR

"Programming is a highly-procedural skill and it's only by doing it that you never really know."

Course outline →

Foundation
- Unit 1 : The Basics
- Unit 2 : Procedural Programming → Airthmatics, Variables
- Unit 3 : Control Structures → If, Loops
- Unit 4 : Data Structures → Strings

Advanced
- Unit 5 : Objects & Algorithms → Object-oriented
- Unit 6 : .....

Create Concept ⟹ Create Instances → Object-Oriented Programming

→ Python is a high-level dynamic programming Language. It was first created in early 1990s.
→ It is very popular language.
→ Run code line by line.
→ Seprate software for windows, Mac, and Linux.
→ We use it like a calculator.
→ Alternative is Scripting mode.

DOWNLOAD SOFTWARES :

PyCharm
Netbeans
Vocareum
Skulpt
Python.org
Ipython

- Computer have vocubulary and syntaxs which it really understand.
  - > Writing Code
  - > running code
  - > evaluating the result

⇩

Fix error

⇩

Exact result as user want!!

So Voila!!!

Print("___")

↑

Data

```
for i in range(1,9):
    print(i)
```

Example: Building a Table

⇩

Use same analogy

⇩

Building computer programme

PROGRAMMING

- Debugging is trying to find why the code doesn't behave the way you want it to.

TYPES OF ERROR

- Compilation error
- Runtime error

Error inherent in the code
Syntax Error
Name Error
Type Error

Devide by zero error
Null Errors
Memory Errors

Print(5) ✓
Print 5 X ⇒ Error

→ Print debugging
→ Scope debugging
→ Rubber duck debugging

UGGING

→ Functional Programming
→ Methods
→ Procedures
→ Object-oriented programs

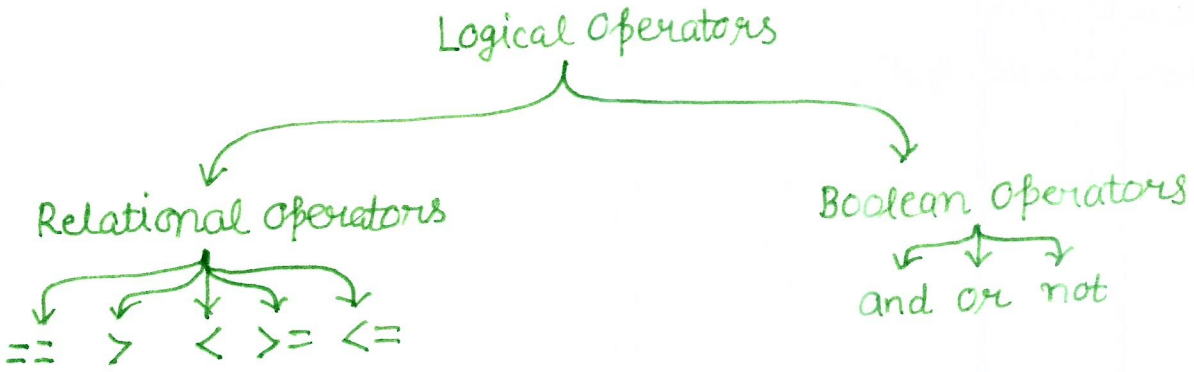
A Student

First Name
Last Name
ID Number
Address
Sex
Blood Group

→ Event driven programming

- data Types and Variables
  myInteger
  myFloat
  myDate

from datetime import date



#Comments & Documentation

Comments
Comments
Comments
Comments
Comments
Comments
Comments

PROCEDURAL PROGRAMM

#Sometime we can write inline comments.

Good Variables

```
age = 30
firstName = "David"
catcount = 20
```

• Variable - A name that hold values.

Variable - Microsoftstockprice
Value - 53.50 %

↑

Changeable.

aNumber = -2
aDecimal = 7.2
aString = Hello
aBoolean = True
aDate = date.today( )

#Reserve Word:
for, None

• Assigning Variables:

x = 5 (✓)
5 = x (X)

• Assigning Python:

Mynumber = 5
Print (myNumber).
.....
5 (✓)

5 = MyNumber (X)

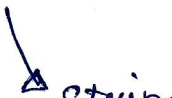
• Data Types:

- Integers $\mathbb{Z} = 21, 4, 0, -2048$
- Real Numbers $\mathbb{R} = -\infty$ to $+\infty$
- Characters a to z and A to Z
- Strings

VARIABLES

myVariable = ?
Print(type(myVariable))

"5554321"



String if password

5,554,321



Number if maths.

```
from datetime import date
myDate = date.today()        myDateasString = Str(myDate)
print("Today's date: " Str(myDate))
```

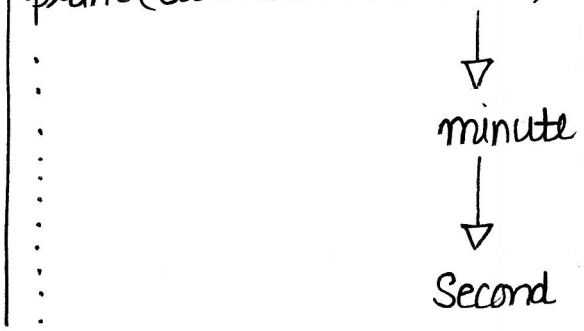
- Reserved Words:

```
Break
except
import
or
and
del
from
None
True
class
False
in
pass
with
as
elif
global
nonlocal
try
continue
finally
is
raise
```

yield
assert
else
if
not
while
def
for
lambda
return

```
print(mydate.year)
print(mydate.month)
print(mydate.day)
```

```
import datetime
currentTime = datetime.datetime.now()
print(currentTime.hour)
```



LOGICAL OPERATORS

$\geq \leq > < \neq =$

multiple conditions are true.
One of the multiple conditions is true.

Logical Operators → Relational operators
Logical Operators → Boolean operators

$==$ ↑ Operator

$=$ ↑ Assigned value already

ATM BANK TRANSACTION EXAMPLE

- What?
  Operators → Integers, Strings, Floats
  ↑ logical
  ↑ Mathematical → Next chapter
  ↑ less Important
  +, −, X, ÷, [%] Modulus

→ and

↓

and is true if all entries are true.

→ or

↓

or is true if any entry is true.

→ not

↓

or is true if any of the entry is present

e.g.

rotten = grapes are rotten

not rotten

onList = grapes are on shopping list.
onSale = grapes are on sale for a discount price.
rotten = grapes are rotten.
inStock = grapes are in stock at store.

(onlist or onsale) and instock and not rotten

Example

It was best example to explain.

```
...
if TransactionAmount < accountBalance :
    if customerName == CardholderName :
        if Vendor in trustedVendors :
            acceptTransaction( )

    else:
        rejectTransaction("Untrusted Vendor and Invalid
                           Cardholder and Insufficient Funds")

...
```

↖ Part of a program

• TRUTH TABLES : Deal with boolean operators are one of the foundation of the computing. To explore this further, we use something we call Truth Tables. Truth Tables are visualization which helps us simulating and understanding the results of different different boolean expressions.

AND

A	B	A and B
T	T	T
T	F	F
F	T	F
F	F	F

OR

A	B	A or B
T	T	T
T	F	T
F	T	T
F	F	F

NOT

A	$\bar{A}$
T	F
F	T
T	F
F	T

COMPLEX E.G.

A	B	C	A or (B and $\bar{C}$)
T	T	T	T
T	T	F	T
T	F	T	T
T	F	F	T
F	T	T	F
F	T	F	T
F	F	T	F
F	F	F	F

COMMUTATIVE PROPERTY (GROUP THEORY):

A	B	A and B	B and A
T	T	T	T
T	F	F	F
F	T	F	F
F	F	F	F

DISTRIBUTIVE PROPERTY

A and (B or C) = (A and B) or (A and C)

$\cup$ $\cap$

$\cdot$ $+$

DE MORGAN'S LAW (AND → OR)

$$\overline{(A \text{ and } B)} = \bar{A} \text{ or } \bar{B}$$

$$\overline{(A \text{ or } B)} = \bar{A} \text{ and } \bar{B}$$

$$\overline{A \cdot B} = \bar{A} + \bar{B}$$

$$\overline{A + B} = \bar{A} \cdot \bar{B}$$

myInteger = 5

+, −, X, ÷, (%)

↑

Remainder

$$3\overline{)7} \quad 2 \text{ r } 1$$

7 modulus 3 = 1

- Additional operators : Some Languages have them.

9/3 = 3.0

9//3 = 3

↑

round configuration

** ← stands for power

- Self Assignment :

newBalance = oldBalance + Paycheck

Balance = Balance + paycheck

→ myNum = myNum + 1

→ myNum += 1

Physics variables

- Integers
- Float
- Complex

MATHEMATICAL OPERATORS

X = 1
Print(X)
Then Result
1

Floats meaning
X = 1.2e34 mean 1.2×10^{34}
X = 1e-12+2.3e45j mean $10^{-12} + 2.3 \times 10^{45} j$

int
float
complex
} Initials

Sep command to add some comma.
" " ← Parentheses

Operations
x+y
x-y
x*y
x/y
x**y
x//y
x%y

- Other Facilities:
  Multiplying matrices
  Calculating a logarithm
  Making a graph

↑
Packages

(math)

from math import log

x = log(2·5)

log, log_{10}, exp, sin, cos, tan, arcsine, arccos, arctan, sinh, cosh, tanh, sqrt

asin acos atan

π = 3·14

from math import *

everything

Linear Algebra Module numpy·linalg

Fourier Transformation Module numpy·fft

from numpy·linalg import inv

Inverse of the matrix

abs(-2) = 2

abs(3+4j) = 5·0

- IF Statement -

```
x = int(input("Enter the whole number not greater than ten:"))
if x>10:
        Print("You entered a number greater than ten")
        Print("Let me fix")
        X=10
        Print(X)
```

or

if

else

elif → else if

- WHILE Statement -

```
while
break
```

→ Check odd even value using divisiblity rules.

- The Fibonacci Sequence -

```
f1,f2 = 1,1
  while f1<=1000:
      print(f1)
      f1,f2 = f2,f1+f2
```

- Lists and Array -

$r = (x, y, z)$

$m \times n$

$A = \{A_i\}$

Situation like these are so common that python provides standard feature called containers, for storing collection of numbers.

type < Lists / Arrays

1. Lists -
The most basic type of container is lists

$r = (x, y, z)$

Container elements

NOTES *

- Module-

A module is a file containing code written by somebody else (usually) which can be imported and used in our programs.

↳ REPL

- Pip-

Terminal

pip Install flask

- Type of Modules-

(1) Built in Pre Install, e.g. os
(2) External Install using pip, e.g. tensorflow, flask

→ Use python as calculator

Open window power shell
Type "Python"

→ Use comments

written but not exacuted

```
#comment 1
#comment 2

'''
comments
'''
```

→ If printing multiline print document then use ("""..... ....")

- Variable-

A variable is the name given to a memory location in a program.
e.g.

```
a = 30
b = 71.2
c = Abhishek
```

→ Variable = cointener to store value
→ Keyword = reserve words in python

• Data Types -

(1) Integers
(2) Floating point numbers
(3) String
(4) Booleans
(5) None

→ A variable name can't be start with numbers
→ Identifier = Class / Function / Variable name.

• Operators -

(1) Airthmatic +, -, *, /
(2) Assigned =, +=, -=
(3) Comparison ==, >, <, >=, <=, !=
(4) Logical and, or, not

Typecasting

<int> ⇩ ⟶ can be converted

<Str>

Str(31) = "31"
int("32") = 32
float(32) = 32.00

and so on

• String - String is a data type in python. It is a sequence of character.

(1) Single quoted
(2) Double quoted
(3) Triple quoted

#String slicing - A string in python can be sliced for getting a part of the string.

e.g.

name = " A B H I S H E K " ⇒ Length = 8

0	1	2	3	4	5	6	7
(-8)	(-7)	(-6)	(-5)	(-4)	(-3)	(-2)	(-1)

→ Negative Indices

The index in a string start from 0 to python.
In order to slice;

[0]
[1]
[2]
:
so on

e.g.

```
b = "Abhishek"
Print(b)                    Abhishek
Print(type(b))              <class 'str'>
Print(b[0])                 A
Print(b[0:4])               Abhi
Print(b[-1])                K
Print(b[:4])                Abhi
Print(b[1:])                bhishek
Print(b[0:])                Abhishek
```

- **Slicing with skip value-**

We can provide a skip value as a part of our slice like this

```
Print(b[0::2])              Ahse
```

- **String Function-**

Function we can use;

e.g.

```
Story = "There was a youTuber once"
Print(len(Story))                          21
Print(Story.endswith("once"))              True
Print(Story.count("e"))                    1
Print(Story.capitalize())                  Capitalize First words
Print(Story.find("once"))
Print(Story.replace(oldword,renew)
```

\n ← new line

\t ← tab

\' ← Single quote

\\ ← backslash

- **Lists-**

Python lists are ~~sequences~~ containers to store a set of values of any data type.

```
friends = ["Apple", "Akash", "Rahul", 7, False]
```

⇓

can store values of any data type

e.g.

```
a = [1,2,4,56,6]
Print(a)                    1,2,4,56,6
Print(a[0])                 1
Print(a[1])                 2
Print(a[2])                 4
Print(a[3])                 56
Print(a[4])                 6
Print(a[5])                 Error
Print(a[-1])                6
Print(a[-2])                56
Print(a[-3])                4

b = [4, False, "Rohan"]
Print(b[-1])                Rohan
```

- List slicing :

```
friends = ["Harry", "Tom", "Rohan", 45, "Divya"]
Print(friends[0:3 ])
Print(friends[-4:])
```

- List methods :

```
L1 = [1,8,7,2,21,50]
L1 X L1_Sorted = L1.sort()
Print(L1_Sorted) X          None
                            [1,2,7,8,21,50]
```

L1

```
L1.reverse( )               [50,21,8,7,2,1]
L1.append(1)                [1,8,7,2,21,50,1]
L1.insert(3,8)              [1,8,7,8,2,21,50]
```

on index

This will add

```
L1.pop(2)                   [1,8, 2,21,50]
```

Remove index 2

```
L1.remove(21)               [1,8,7,2,50]
```

- Tuple -

```
# creating a tuple using ()
t = (1,2,4,5)
print(t[0])
t[0] = 34
```

tuple doesn't support assignment

A tuple is an immtable data type in python.
↳ can't change.

```
t1 = ( )
t2 = (1,)
t3 = (1,7,2)
Print(t3.count(1))
Print(t3.index(7))
```

← empty tuple
← one element tuple
← more than one element tuple
→ count values. 1
→ Index count 1

↓
Subscribe more methods online on python.org

- Dictionary and sets -

Dictionary is a collection of key value pairs.

Syntax

```
dic = {
    "Fast": "In a quick way",
    "Hurry": "quickly, fastly"
    }
print(dic["Fast"])
```

In a quick way

Properties of dictionary:
(1) Unordered
(2) Mutable
(3) Can't contain duplicate keys

Methods:

- keys → keys of dictionary
- values → values of dictionary

↓ you can run these with list command

- items → items of dictionary
- Updates → can update data
- get → get data

- Sets in python -

Set is a collection of non-repitative elements.

```
S = set( )                    -> No repitative items
S.add(1)                      -> set = {1,2}
S.add(2)
```

```
a = {1,2,3,4,1}
print(a)                      1,2,3,4
print(type(a))                <class 'dict'>
```

```
b = set( )
print(type(b))                <class 'set'>
```

→ Properties;

Unordered, Unindexed, no change (Unmutable), no duplicate values.

→ Methods;

- len(s) -> length
- remove( ) -> remove something
- pop( ) -> remove anything by default
- clear( ) -> clear set
- union( ) -> All items of both sets (U)
- intersection( ) -> contains only item in both side (∩)

- Conditional Expressions -

Decisions which depends on the conditions like if, if and else, if and elif in python

In all programming languages, we must be able to execute instructions on a condition(s) being met. This is why conditions are for!

Syntax:

```
if(condition1):
        print("yes")
elif(condition2):
        print("no")
else:
    print("maybe")
```

Code Example:

```
a = 8
if(a>2):
print("True")
else:
    print("False")
```

- Relational Operators :

  == → equal.
  >= → greater than and equal to .
  <= → less than and equal to .

- Logical operators :

  and → both conditions are true.
  or → At least one is true.
  not → inverts the result.

Notes :

(1) If "None" is already used in syntax, than use "is None" while using again.
(2) There can be any number of elif statements.
(3) Last else is executed only if all the conditions inside elif fails.
(4) You can use lots of conditions in one program.

- Loops -

Sometimes we want to repeat a set of statement in our program.
e.g. print 1 to 100
Loops make it easy for a programmer to tell the computer, which set of instruction to repeat and how.

Types -

(i) While
(ii) for → Used to iterate through a sequence of list, tuple or string.

Program for e.g.

```
i=0
while i<10 :
        print("Yes")
        i=i+1
print("Done")
```

Note: If the condition never becomes false, the loop keeps getting executed.

```
l=[1,7,8]
for item in l:
        print(item)
```

```
1
7
8
```

e.g.

```
for i in range(6):
        print(i)
```

```
for i in range(2,6):
        print(i)
```

range

→ For loop with else-
An optional else can be used with a for loop if the code is to be executed when the loop exausts.

→ The break statement -
It instructs the program to exit the loop now.

```
for i in range(9):
        print(i)
        if i==5:
            break
```

else after break will not work.

```
1
2
3
4
5
```

→ The continue statement-
It instructs the program to "skip the iteration".

→ The Pass statement -
It instructs the program to "do nothing".

- Function-
A function is a group of statements performing a specific task.

When a program gets bigger in size and its complexity grows, it gets difficult for a programmer to keep track on which piece of code is doing what.

A function can be reused by the programmer in a given program any number of.

e.g.

```
def func1():
        print("Hello")
```

This function can be called any number of times anywhere in the program.

→ function call-

func1( ) → This is called function call.

```
def greet(name):
    print("Good day," + name)

greet("Harry")
```

→ default parameter value can be given here.

→ Types of function :

A. Built-In function → print(), len()

B. User defined function → func( )

→ Default parameter value :

We can have a value as default argument function.

#Recursion-

Recursion is a function which calls itself. It is used to directly use a mathematical formula as a function.

e.g.

factorial(n) = n x factorial(n-1)

```
n! = 1*2*3*4*....*n
n=4
for i in range(n):
        product = product*(i+1)
```

```
def factorial_iter(n):
    product=1
    for i in range(n):
            product = product*(i+1)
    return product
f = factorial_iter(n)
print(f)
```

Sample program

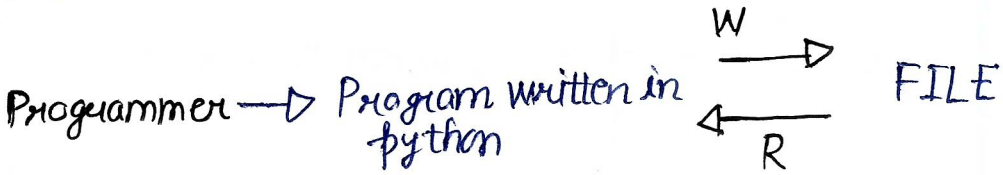
Note : The programmer need to be extremely careful while working with recursion to ensure that the function does not infinitely keep calling itself. Recursion is something the most direct way to code algorithm.

write recursion code only when you know what are you coding.

- File I/O -

The random access memory is valuable and all its content are lost once a program terminates in order to persist the data forever, we use files.

A file is data stored in a storage device. A python program can talk to the file by reading content from it and writing content to it.

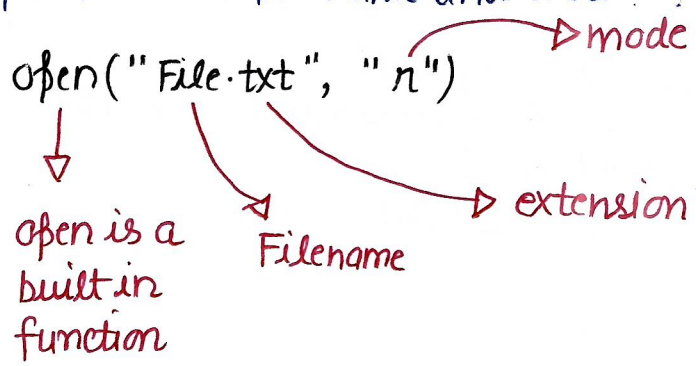


- Types :
  (1) Text File
  (2) Binary files

Python has a lot of functions for reading, updating and deleting files.

- Opening a file :

Python has an open( ) function for opening files. It takes parameters : filename and mode.



e.g.

```
f = open ('sample.txt', 'r')
data = f.read()
print(data)
f.close()
```

Create a file in directory

by default condition

- Syntax:

```
f = open("This.txt", "r")
text = f.read()
print(text)
f.close()
```

We can also specify the number of characters in read() function:

```
f.read(2)
```

↳ read first 2 characters.

Other methods to read are readline() etc.

- Modes:
  - r → read
  - w → write
  - a → append
  - t → updating

"rb" will open for read in binary mode.

"rt" will open for read in text mode.

Write file in python-

In order to write into a file, we first open it in write or append mode after which, we use python's f.write() method to write to a file!

e.g.

```
f = open("This.txt", "w")
f.write("AB")
f.close()
```

#Write file with "with" statement:

```
with open('sample.txt', 'w') as f:
    a = f.write()
print(a)
```

↳ Don't need to close a file using f.close() as we were doing in other modes.

#Object-oriented programming:

Solving a problem by creating objects is one of the most popular approaches in programming. This is called Object-oriented programming.

This concept focuses on using reusable codes.

↳ Implements DRY principle

Do not repeat yourself

e.g.

```
a=12
b=34
c=a+b
print(c)
```

oops

```
class Number:
    def sum(self):
        return self.a + self.b
num= Number()
num.a = 12
num.b = 34
S= num.sum()
print(S)
```

→ Class:
A class is a blueprint for creating objects.





☆ The syntax of the class looks like this:

```
class Employee:
    #methods & variables
```

class name will be written in pascalCase.

↓

EmployeeName

camelCase

↓

employeeName

→ Object:
An object is an instantiation of the class. When class is defined, a template (info) is defined. Memory is allocated only after object Instantiation.
Objects of a given class can invoke the methods available to it without revealing the implementation details to the user.

- Modelling a problem in oops:
We identify the following in our problem.

Noun → class → Employee
Adjective → attributes → name, age, salary
Verbs → methods → getSalary(), Increment()

- **Class Attributes :**
An attribute that belongs to the class rather than a particular object .
e.g.

```
class Employee :
    company = "Google"

harry = Employee()
harry.company
harry.company = "YouTube"
```

→ Changing example

- **Instance Attributes :**
An instance that belongs to the object assuming the class from the previous example.

```
harry.salary = "3000M"
```

Note : Instance attributes take preference over class attributes during assignment & reterival.

- **Self Parameter :**
Self refers to the instance of the class. It is automatically passed with a function call from an object.

```
harry.getsalary()
```

- **Static Method :**
Sometime we need a function that doesn't use self parameter. We can define a static method like this.

☆ **Inheritance–**
Inheritance is a way of creating a new class from an existing class.

Syntax :

```
class Employee:
        #code
        ....

class Programmer(Employee):
        #code
        ....
```

← parent or base class

← child or derived (daughter) class

We can use the methods and attributes of employee in programmer object.

Also, we can overwrite or add new attributes and methods in programmer class.

☆ Type of Inheritance -

(i) Single Inheritance → Only one parent

(ii) Multiple Inheritance → Multiple parent

(iii) Multilevel Inheritance → Multiple child