:MLP:

Python :

Python is a Highlevel proqramming laung -uqe known for its simple code and beginner friendly Syntax. its called an interpreted langua -qe because, when python code runs, it is executed line by line by a proquam called the python interpreter. This means errors can be found as the proquam runs and make testing and learning Easier.

* features of python:

(i) Easy to write, read and learn
(ii) Free and open source
(iii) Interpreted
(iv) Simple Syntax
(v) Indentation
(vi) In built libraries
(vii) Supports all kind of developement
(viii) Less memory consumption
(ix) Dynamic typed

* Application of python :

(i) Web developement
(ii) Game developement
(iii) Machine learning and AI
(iv) Data Science and visualization
(v) Audio & Vidio Application
(vi) CAD Application
(vii) Business Application
(viii) Software developement
(ix) Web Scrapping Apps
(x) Automation

* History of python:

python was developed by Guido van Possom in 1980s(1987) and released in 1991

* Meaning of "dynamically typed":

Dynamically typed means variables in python don't have fixed types. The python Interpreter figures out the type of variable (number, test, List, etc) when the code runs, for Example, a variable can start as a number and then become a string in the same program, and python will Handle it automatically

* Difference between python 2 and python 3

* Print statement / function:

* In python 2, print is a statement, used without parantheses (Ex: print "Hello").

* In python 3, print() is a built-in function and requires paranthesis (Ex: print("Hello"))

* Integer Division:

* Python 2 performs integer division by default when dividing 2 integers, resulting in a truncated integer (Ex. 5/2 yields 2)

* In python 3's / operator performs float division, returning a float (Ex 5/2 yields 2.5). Integer division in python 3 requires the // operator.

* In python 2, input() Evaluates the input as python code, while raw_input() reads as a str.

* In python 3, input() directly reads the input as a string, and raw_input is removed

* PEP 8 and importance of PEP8 in python:

PEP8 :

PEP 8 is stands for python Enhancement Proposal 8, it is the official style guide for python code. PEP8 provides a set of guidelines & best practices for writting clean, readable and consistent python code and it will reduce a errors

* Importance of PEP 8:

(i) Readability and maintanability
(ii) Consistency
(iii) collabration
(iv) Professionalism
(v) Reduced Error

* Difference b/w Compiler and interpreter:

* A compiler translates all source code into Computer code before running like a program, like in C or Java. It finds all Errors before running

* An Interpreter (like python) runs and checks code line by line so you can see the results and Errors instantly, which is greate for learning
* we can't proceed further before solving the Error

* Saving Extensions :

* for python files the saving extensions is (.py).
these files can be opened in all python IDEs including jupyter notebook.

* for jupyter notebook the saving extension is (.ipynb), these files can be opened in jupyter notebook, VS code, google colab

* Comments :

it is a part of code but it won't take any place in execution

1) Single line comment : (#)
2) multiple line comment : ''' ''' or """ """

* Variables :

variables are names assigned to memory locations. The value assigned to variable can be used within the program.

* Syntax :

<variable_name> = <value>

Note : No need to declare a variable or specify the data type python is dynamically types (means automatically detect)

Ex:

```
a = 10
print(a)
```

O/P:

```
10
```

* Here 'a' is a variable name for value '10', when you refer 'a' anywhere in the python, that means you are refering '10'

* Rules for variable;

1) it must start with letter or an underscore

```
x = 20          # valid
_sana = "size"  # valid
$a = 13.9       # invalid
3b = 44         # invalid
num1 = 3        # valid   (num use in last or middle)
```

* Assigning variable to another;

Ex:

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

O/P:

```
20
20
```

* when diff variable have the same value, it means the value has 2 names

when a = 15 b = 15

a → [15] ← b

Ex: a = 2+4 print(a)	O/P: 6

Ex: a = 10 print(a) a = a+20 print(a)	O/P: 10 30

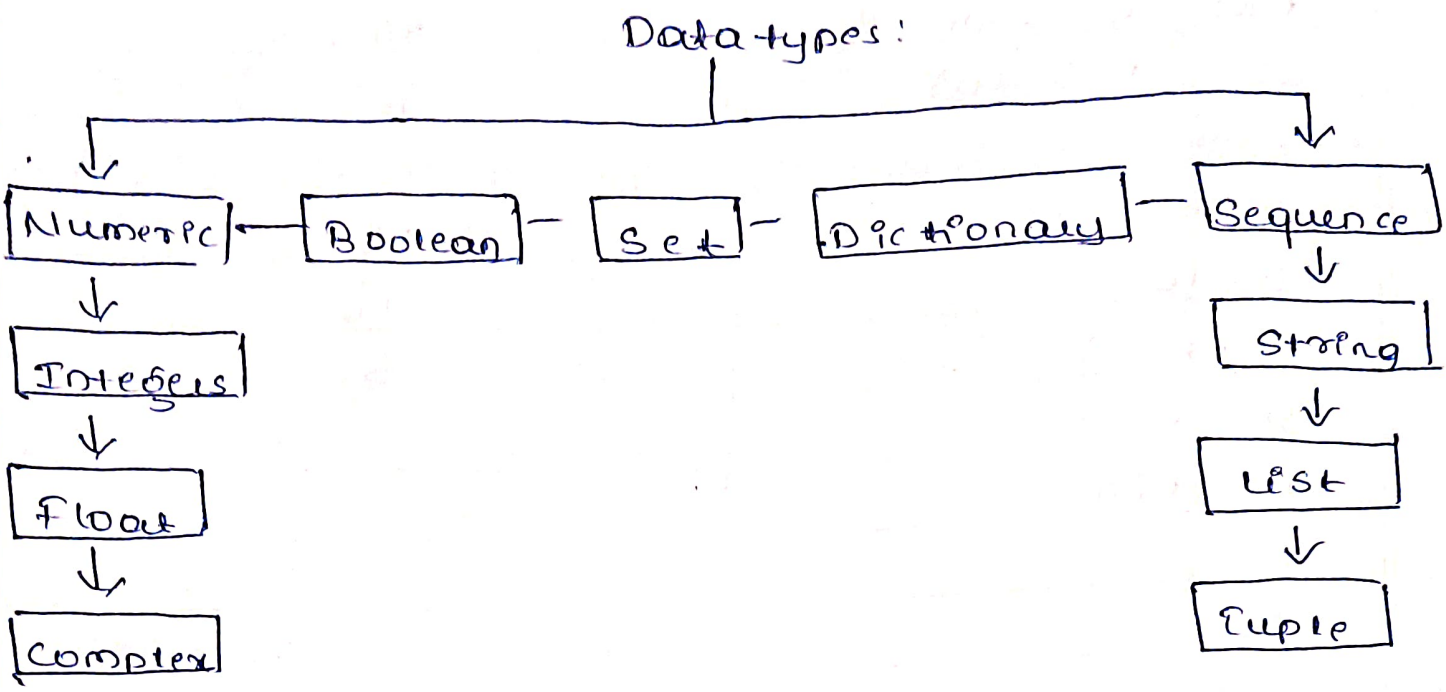
Ex; a, b, c = 20, 12, 34 print(a) print(b) print(c)	O/P; 20 12 34

Ex: a, b, c = 10, 13, 16 print(a, b, c)	O/P; ~~(10, 13, 16)~~ (10 13 16)

Ex; a=b=c=10 print(a, b, c)	O/P; (10 10 10)

Ex; a, b = "good", "bad", "ugly" print(a, b, c)	O/P: , # Error

Data types :



*) Numeric : it contains numeric value.

* Integers : it contain +ve or -ve and whole number (int)

* float : it contain real number with floating point / decimal point

Ex : 8.35 , 4.04 , ($1.3E4 = 1.3 \times 10^4$) , ($4.3E2 = 4.3 \times 10^2$) = 430

* complex : They contain real number & imaginary part

Ex : 7+8j , 2+5j , 3J (0+3j)

* String : collection of one or more character but in single or double quote

Ex : "Hello" , "pytho" , "4" , "94.8" , "$89" Etc

List : Lists are an ordered collection of data which contain multiple values of different data types

* It is created by enclosing value with square bracket [ ] & separated by comma.
* This values are modified (mutable)

Ex :

[1, 2, 3] , ["Hello", 3.45, 8, 100], [] (Empty list)

Ex : [30, 40, [50, 60], 70] (A list with 4 integers)

Ex : [["Hi", 908], [50, 40]] (A list with 2 integer)

* Mutable and Immutable Datatype:

* Mutable data type include list, set, dict (modify)

* Immutable data type include (int, float, complex, str and Tuple)

* Testing the Datatype of variables:

In python we check data type by Built in function

Ex:

```
a = 10
print (type(a))
```

o/p:

```
<class 'int'>
10
```

Ex:

```
a = '247'
print (type(a))
```

o/p:

```
<class 'str'>
```

TYPE CONVERSION:

(i) Implicit type conversion

(ii) Explicit type conversion

(i) Implicit type conversion:

Python interpreter automatically converts one data type to another without any user involvement

* This is also called as promotion

Ex:

```
num1 = 3        # int
num2 = 4.5      # float
num3 = num1 + num2
print (type (num3))
```

O/P:

<class 'float'>

(ii) Explicit Type Conversion:

In this, The data type is manually changed by users as per requirement

* Here there is risk of data loss, since we manually force an expression to be converted to another datatype

Ex:

```
s = "80"
a = 40
b = a + int (s)  (# convert s to int)
print (b)
```

O/P:

120

Ex:

```
age = 24
print ("my age is" + str(age)
```

O/P:

my age is 24

Ex;

```
s = 'Hello'
Print (List(S))
```

O/P;

['H', 'E', 'L', 'L', 'O']

* Conversion with invalid value

Ex:

```
S = 'Hello'
print (int(S))
```

O/P;

ValueError: invalid literal for int()

* Getting user INPUT:

* To get a input from the user, python uses built in function called input()
* The input typed is returned as 'string', if you need of other type, you must explicitly convert it

Ex:

```
name = input ("Enter your name:")
print ("Hello", name)
```

O/P:

Enter a word: chip
Hello chip

Ex:

```
num1 = int(input("Enter a no:"))
num2 = int(input("Enter an other no":"))
Print(num1+num2)
```

O/P:

```
Enter a no : 20
Enter an other no : 30
50
```

Ex:

```
print("Good morning")
name = str(input("Enter your name:"))
age = int(input("Enter your age:"))
print(name, type(name))
print(age, type(age))
```

O/P:

```
Good morning
Enter your name : Sana
Enter your age : 22
Sana <class 'str'>
22 <class 'int'>
```

#1) Write a program to read Employee data
Emp ID , Emp name , Emp salary

* Expected output format :

Employee ID = 123
Employee name = Sana
Employee Salary = 45000 Rs

Code :-

```
id = int (input (" Enter Employee id :")
name = str (input (" Enter Employee name :")
sal = int (input (" Enter Employee Salary :")
print (" Employee id =", id, "\n employee name =", name, "\n Employee
                Salary =", sal, "Rs")
```

O/P :

Enter Employee id = 123
Enter Employee name = sana
Enter Employee Salary = 45000
Employee id = 123
Employee Name = Sana
Employee Salary = 45000 Rs

print by .format method :

```
print (" Employee id = { } \n Employee name = { } \n Employee sal = {sal} {}
                Rs" . format (id, name, sal))
```

Print using f" method:

```
print(f"employee id = {id} \n employee salary = {name}
        \n employee salary = {sal} Rs")
```

#2 write a program to read student ID, name and 3 subject marks and display in details

```
Student id :
Student name :
Sub1 :
Sub2 :
Sub3 :
Total marks :
Percentage :

id = int(input("Enter your id :"))
name = str(input("Enter your name :"))
Sub1 = float(input("Enter Sub1 marks :"))
Sub2 = float(input("Enter Sub2 marks :"))
Sub3 = float(input("Enter Sub3 marks :"))
total = Sub1 + Sub2 + Sub3
Per = (total/300) * 100
print(f"student id = {id} \n student name = {name} \n
    Sub1 marks = {Sub1} \n Sub2 marks = {Sub2} \n Sub3 marks
    = {Sub3} \n total marks = {total} \n Percentage = {Per}")
```

(Assignment 2)

* Operators :

it is a special symbol perform certain operation between operands

Ex: z = x + y

=+ are operator , xyz are operands

Types of operators :

1) arithmetic operator : (+, -, *, /, %, //, **)
2) comparsion operator : (<, >, =<, =>, ==, !=)
3) logical operator : (And, or, not)
4) assignment operator : (=, +=, -=, *=, /=, //=, **=)
5) Bitwise operator : (|, ^, <<, >>, ~)
6) identity operator : (is, isnot)
7) membership operator : (in, notin)

1) Arithmetic :

#1 take a 2 inputs from users & perform all A. operation

```
a = float (input ("Enter the number:")
b = float (input ("Enter the number:")
c = a+b
d = a-b
e = a*b
f = a/b
g = a//b
h = a**b
i = a%b
```

```
print = (f" sum = {a}\n sub = {d}\n multi = {e}\n float division = {f}\n floor div = {g}\n exp power = {h}\n modulus = {i}")
```

O/P :

Enter the number = 5

Enter the number = 3

Sum : 8.0

Sub = 2.0

multi° = 15.0

f.div = 1.6666

floor div = 1.0

exp power = 125.0

modulus = 2.0

* Comparison operator: (Relational operator)

Ex:

```
k = 9
l = 14
print (k > l)
print (k < l)
print (k == l)
print (k != l)
print (k >= l)
print (k <= l)
```

O/P :

False

True

false

True

False

~~False~~ True

* Logical operator : (and, or, not)

and = all condition should pass

or = atleast one condition should pass

not = vice versa

Operators	Definition	Example
and	True if both operands are true	x & y
or	True if either one of operands are true	x or y
not	True if operands is false	not x

fx :

a	b	and	or	xor	xnor
f	f	f	f	T	f
f	T	f	T	F	T
T	f	f	T	f	T
T	T	T	f	T	f

* Not

a	not
T	F
F	T

Ex:1) a = 7 — O/P: False

b = 10

#1) print (a>10 and b<10)

#2 Print (a!=10 and b==10) — True

#3 print (a==10 or b==10) — ~~False~~ True

Ex: a = 10 — O/P

print(not(a==10)) — False

* Assignment operator : (=, +=, -=, x=, /=, //=, **=)

Assignment operator assign a value to the variables

Ex:

```
a = 10          # Assignment
b = a           # Assignment
print (b)
b += a          # Add & Assign (b = b+a)
print(b)
b*=a            # b = b*a
```

O/P

10 — b = a

20 — b = b + a => 10+10

200 — b * a = a => 20 * 10

Identity Operator :

Identity Operators are used to check if 2 variables or values share the same memory location & are of same datatype

Operator	Definition	Example
is	Evaluates to True if both values are same	x is y
is not	Evaluates to True if both values are not same	x is not y

Ex :

```
x = 4
y = 4
z = 5
print (x is y)
print (x is not z)
print (y is z)
```

O/P:

```
True
True
False
```

* Membership Operator:

Membership Op// are used to check or validate the membership of value. It tests the membership in a sequence, such as Str, List or Tuple

Operator	Definition	Example
in	Evaluates to true if an Element Exist in a sequence	x in y
not in	Evaluates to true if an Elements does not Exist in Sequence	x is not y

Ex:

```
S = "Hello"
Li = [1, 2, 3, 4, 5]                    O/P:
print('H' in S)                         True
print (3 in Li)                         True
print (7 not in Li)                     True
print ('e' not in S)                    false
```

Ex:

```
num = (1, 2, 3, 4)                      O/P:
1 in num                                True
4 not in num                            false
```