___/___/___

PYTHON HANDWRITTEN NOTES

YOUTUBE - Chip Sized
LINKED IN - Chip Sized
INSTAGRAM - chipsized

INTRODUCTION TO PYTHON :

DEFINITION :

- Python is an interpreted, high-level, and general-purpose programming language.

High level - Programs are easily understood by humans.

Interpreted - Uses an interpreter to execute programs.

General Purpose - Used in a variety of applications like web, desktop, ML, AI etc.

HISTORY :

Founder - Guido van Rossum

Year found - 1989

Purpose - Better Readability.

Versions - Python 2 and 3 (Currently Python 3.10)

FEATURES OF PYTHON :

1) Easy to write, read, and learn.
2) Free and open-source.
3) Interpreted
4) Supports modularity.
5) Extensible
6) Dynamic type system.
7) Automatic memory management.
8) Supports third-party packages.
9) Object-oriented.

APPLICATIONS OF PYTHON :

1) Web Development
2) Game Devellopedent
3) Machine Learning and AI.
4) Data Science and Visualization.
5) Desktop GUI
6) Web Scraping Apps.
7) Business Applications (E-Commerce)
8) Audio and Video Applications
9) CAD Applications
10) Embedded Applications.

⊛ Top companies such as Google, Facebook, Instagram, Quora, Dropbox, Udemy and IBM uses Python.

⊛ It is number one in popularity among languages.

__/__/____

INSTALLING PYTHON :

Note : (To check whether Python is already installed, open command prompt (CMD), and type the command "python --version". If it shows a version, you have it already installed. If not you must download the Python installer.)

STEPS :

1) Open your browser and search python.
2) Click the link of python.org (search).
3) Download the latest version.
4) Open the installer.
5) Check the two boxes below (install for all users, add Python to PATH.)
6) Click on Install Now. Wait until it's complete and then close.
7) Again open CMD. Type the command "python --version", and it would display the version.

Note : (There may be different versions of Python and are constantly updated. At the time of writing, the version was Python 3.10.7.

Download the latest version anyway as there may be very little difference among different version.)

__/__/__

INTERACTIVE AND SCRIPT MODE :

⊛ In Python, there are two ways to run our code.

1) Interactive Mode
2) Script Mode.

INTERACTIVE MODE	SCRIPT MODE
1) Python statements are written in CMD and we got the result of each line instantly.	1) Python program is written in a file. The Python interpreter executes the complete file and displays output in CMD.
2) Better suited for writing very short programs!	2) More suitable for long programs.
3) Code can be edited, but it is hard to do.	3) Editing of code is easily done.
4) Code cannot be saved and used in future.	4) Code can be saved and used in future.
5) Suitable for practice and understanding code.	5) Suitable for writing programs and projects.

THE PRINT FUNCTION :

⊛ The print() function is used to display the output in the console.

SYNTAX :

```
print (<value>)
```

EX :	O/P :
print ('Hello World')	Hello World

MULTIPLE PRINT FUNCTIONS :

⊛ We can use multiple print functions. It prints each value in a newline.

EX :	O/P :
print (20)	20
print (3.56)	3.56
print ('Hello')	Hello
print ("Python")	Python

⊛ Text (called strings) in Python must be given within single or double quotes.

PRINT MULTIPLE VALUES :

⊛ Multiple values can be printed using a single print function and separating values with a comma.

SYNTAX :

```
print (<val 1>, <val 2>, ..., <val n>)
```

Ex:

```
print (10, 'Hello', "Python", 3.87)
```

O/P:

```
10 Hello Python 3.87
```

⊛ The values are printed in a single line with a whitespace in between.

USING END ARGUMENT:

⊛ Multiple values can be printed using different print functions and still make them display in a new single line.

⊛ The end argument is used for that within the print function to print a string after all values are printed.

⊛ By default, it holds a '\n' (newline) character and that is why, a new print function starts printing in the newline.

Ex:

```
print ("Hello", end ='#')
print ("Morning")
```

O/P:

```
Hello # Morning
```

USING SEP ARGUMENT :

⊛ The sep argument can be used within the print function to print a string between all values when multiple values are used.

⊛ By default, the sep argument holds a whitespace (" ") and that's why, values printed in the same line leave a whitespace.

EX :

```
print ("Hello", 10, 20.45, sep = "*")
```

O/P :

```
Hello * 10 * 20.45
```

COMMENTS:

⊛ Comments are used for explaining code in plain English.

⊛ The Python interpreter ignores comments and they are not executed.

⊛ A comment starts with a # symbol and is terminated by the end of line.

⊛ Inline comments are written beside a Python statement.

Ex:

```
# This is a single-line comment
print ("My Code") # This prints My Code
```

O/P:

```
My Code
```

⊛ Here, the first comment is a single line comment and the second comment is an inline comment.

VARIABLES:

⊛ Variables are names assigned to memory locations. The values assigned to variables 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 typed (data type automatically detected during runtime).

EX:	O/P:
a = 10 print (a)	10

⊛ Here, `a` is the variable name for the value 10. When you refer to `a` anywhere in the program, that means you are referring to 10.

RULES FOR NAMING VARIABLES:

1) Variable names must start with a letter or an underscore.

```
    x  =  20      #valid
_chip  = "sized"  # valid
   $a  = 13.94    #invalid
  3b   =  44      #invalid
```

2) Variable names must start with letters or underscores, but digits can be used except the start. Special symbols can't be used.

```
a_var  = "chocolate"        #valid
num1   = 3  # valid.
```

3) Names are case-sensitive

```
a = 10          |   O/P :
print (A)       |   Error
```

4) Python keywords are not allowed. These are 35 keywords (as of version 3.10).

The list of keywords can be seen using this code.

```
Import keyword
print ( keyword. kwlist)
```

ASSIGNING ONE VARIABLE TO ANOTHER:

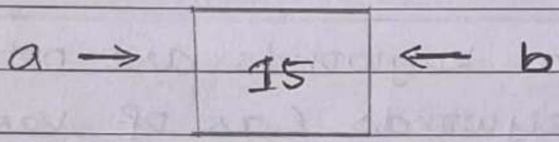
⊛ A variable containing a value can be assigned to another variable.

```
EX :
a = 20
b = a   # Assigning a (which has value 20)
        # to b
print (b)
print (a)
```

```
O/P:
20
20
```

Note: When different variables have the same value, it means that the values has two names. This can be visually displayed.

When a = 15 , b = 15



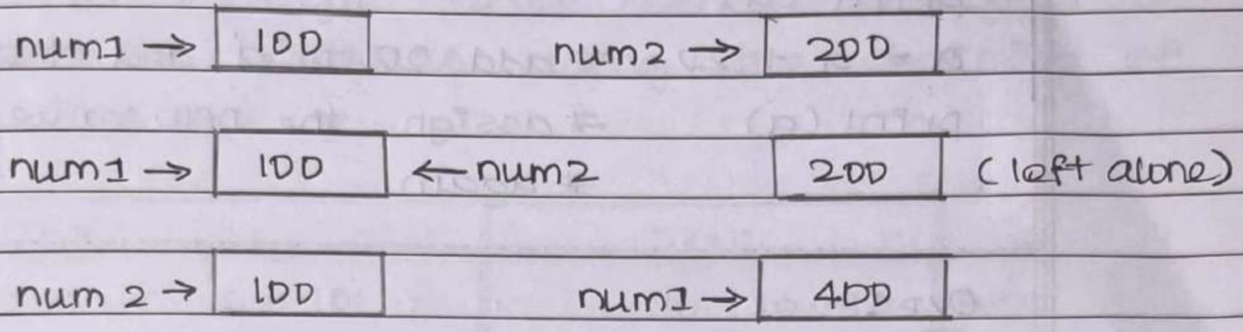
UPDATING VARIABLES:

Variables can be updated to different values within a program.

```
Ex:
num1  = 100
num2  = 200
print (num1, num2)
num2  = num1          # assign num1 -> num2
num1  = 400           # update num1
print (num1, num2)
```

```
O/P:
100 200
400 100
```

VISUAL REPRESENTATION:



⊛ The updated values can be of different data types.

EX:

```
val1 = 10 # an integer
print (val1)
val1 = "Good" # a string
print (val1)
```

O/P:

```
10
Good
```

⊛ You can assign an expression to a variable and it would automatically convert it to a value.

EX:

```
a = 2+4 # automatically added
print (ex)
```

O/P: 6

EX :

```
a = 10
print (a)
a = a + 20   # add 20 to 'a' and then
print (a)    # assign the new value to 'a'
             # again.
```

O/P :

```
10
30
```

ASSIGN MULTIPLE VALUES TO MULTIPLE VARIABLES :

⊛ You can assign multiple values to multiple variables in one line. Make sure that the number of variables are equal to the number of values.

EX :

```
a, b, c = 20, 12, 34
print (a)
print (b)
print (c)
```

O/P :

```
20
12
34
```

⊛ If variables or values become more or less it results in error.

EX :

```
a, b = "Morning", "Night", "Evening"  # Error
a, b, c = 1, 2  # Error
```

ASSIGN A SINGLE VALUE TO MULTIPLE VARIABLES :

⊛ A single value can be assigned to multiple variables using the chaining of assignments.

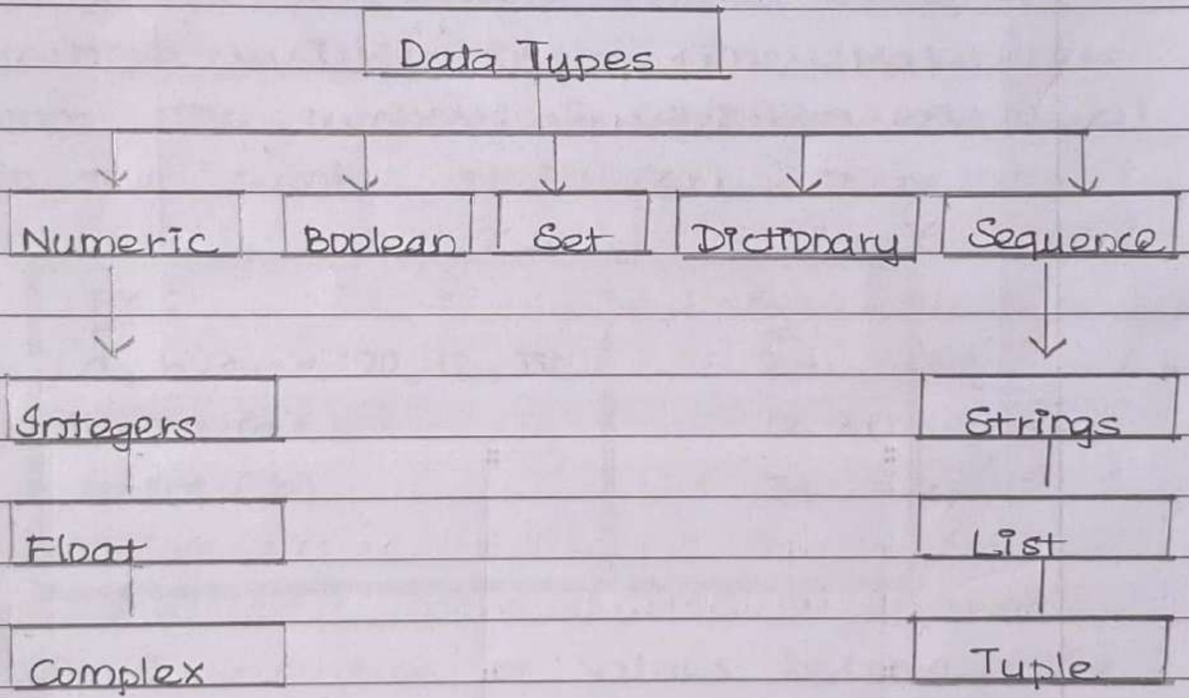
EX :	O/P :
a = b = c = 10 print (a, b, c)	10 10 10

DATA TYPES:

Note: We would discuss about datatypes like string, list, tuple, set and dict in details in upcoming sections.

- A data type is the type of data a value holds. Common data types include Integers, floats, and strings.

- But there are a lot of data types available in Python.



Note: There is also a type called None used for denoting an absent value.

NUMERIC TYPES:

⊛ Numeric data type represents data which contains numeric values (numbers).

⊛ The three types of numeric data types are integers (int), floating point numbers (float) and complex numbers (complex).

INTEGERS:

⊛ Integers contain positive or negative whole numbers (without fraction or decimal).

⊛ In Python, there is no limit to how long an integer value can be. (It just depends on the capacity of your system's memory).

⊛ It is represented as int in Python.

EX: 12, 100, -9784, 49657485

EX:

```
value = 102  # int type
print (value)
```

O/P:

```
102
```

FLOATS :

- Floats contain real numbers with floating point representation (specified by a decimal point).

- The maximum value of a floating point number is 1.8×10^{308}. Any number greater than this will be indicated as inf.(infinity).

- It is represented as float in Python.

- Floats have 16 digits in precision (the maximum). They can also represent scientific notations using the characters E or e.

Ex: 8.35, 4.04, 20.89457.

Ex: 1.3E4 (means 1.3×10^4)
2.86e3 (2.86×10^3)
4.5E-5 (4.5×10^{-5})

Ex:	O/P:

f = 9.101
print (f)
f = 4.3E2
print (f)
	9.101
	430.0

COMPLEX:

⊛ Complex numbers are represented as complex in Python and are rarely used.

⊛ They have a real and imaginary part and are specified as (real) + (imaginary)j.

EX: 7 + 8j, 2 + 5j, 3j (means 0 + 3j).

EX:	O/P:

c = 2+3j
print(c)
	2+3j

SEQUENCE TYPES:

⊛ Sequence data type contains an ordered collection of similar or different data types.

⊛ They allow us to store multiple values in an organized and efficient manner.

⊛ The 3 types of sequences are string (str), lists (list), and typles (tuple).

STRINGS:

⊛ A string is a collection of one or more characters, but in a single quote (or) double quote.

- It is represented as str in Python.

- A string can be defined as arrays of bytes representing Unicode characters.

- Some examples of a string are, 'Hello', "Python", 'A', "40 cookies", "800", '94.86', "$89" etc.

Ex:

```
S = "python $$$ 3.10"
print(S)
```

O/P:

```
Python $$$ 3.10
```

Note:

- The quotes just mark the beginning and end of a string. They do not get printed in the output.

- A numeric value when represented within quotes becomes a string. Normal arithmetic calculations can't be done with those strings.

Ex:	O/P:

num1 = '10'
num2 = '30'
print (num1 + num2)
	1030

⊛ They get concatenated (joined) instead of getting added. When '+' is used between strings, they are concatenated.

LISTS:

⊛ Lists are an ordered collection of data which can contain multiple values of different data types.

⊛ A list can be created by enclosing values, separated by commas, in square brackets.

⊛ The elements of a list can be modified (mutable) and it allows duplicates.

⊛ It is represented as list in Python.

EX:

[1, 2, 3] , ["Hello", 3.45, 8, 100],

[ ] (empty list)

⊛ A list can also contain another list.

EX:

[30, 40, [50, 60, 70], 80, 90] (A list with 4 integers and a list).

[["Hi", 908.0], [50, 40]] (A list with 2 lists)

__/__/__

Ex: lI = [10, 20, 30, 40, 50] print (lI)	O/P: [10, 20, 30, 40, 50]

⊛ A list can contain values of any data type.

TUPLES:

⊛ Tuples are also similar to lists and store multiple values of different types, but the elements within a tuple cannot be modified (immutable).

⊛ A tuple can be created by enclosing values, separated by commas, in paranthesis.

⊛ Tuples can be used for small collection of values, separated by commas, in which the values need not to change.

⊛ It is represented as tuple in Python.

Ex:

```
(30, 40, 50)
('Hello', 45, 0.4789)
(20, (30, 40), [50, 60], 10) # a tuple with
                             # 2 int, a tuple
                             # and a list.
 (3, )   # tuple with a single value
         # (must have a comma after
         # the value).
```