CHAPTER – 7: PYTHON FUNDAMENTALS

INTRODUCTION

1. Python character sets
2. Tokens
3. Barebones of a Python Program
4. Variables and Assignments
5. Simple input and output

A program is a set of instructions that govern the processing
A programs make IPO (Input-Process-Output) cycle.
Python is case sensitive language.

PYTHON CHARACTER SET

Definition : Character set is a set of valid characters that a language can recognize.

Python has the following character set:

1. Letters : Upper case (A B C …. Up to Z) and Lower case (a b c ….. Up to z)
2. Digits(0-9) : 0 1 2 3 4 5 6 7 8 9
3. Special symbols : ~ ` @ # $ % ^ & * + = [{] } \ | / ; : ’ _ < >
4. Whitespaces : Blank space (space bar key), tabs, carriage return , newline, formfeed
5. Other characters : All ASCII (American Standard Code for Information Interchange) and Unicode.

TOKENS (or LEXICAL Unit)

Definition : The smallest individual unit in a program is known as a Token or a Lexical unit.
Example 1: ***“ Hurray!”, he claimed, “We are the world champions.”***
In the above sentence, there are sixteen tokens and they are of two types:

1. Eight numbers of word (String literals) token: Hurray he claimed We are the world champions
2. Eight number of punctuation tokens (Punctuators) : **“ ! “ , , “ . “**

Example 2: Considering this Python statement : ***print (“ Total is “ , a + b)***

1. One keyword token : print
2. Two number of words (string literals) token : Total is
3. Five number of punctuators token : (“ “ ,)
4. Two number of identifier token : a b
5. One operator token : +

DIFFERENT TOKENS IN PYTHON

Python has the following tokens :

1. Keyword
2. Identifiers (names)
3. Literals

a) String literals
    i) Single Line Stings (Basics strings)
    ii) Multiline Strings
        - By adding a backslash
        - By using triple quotation mark
b) Numeric literals
    i) Integer Literals
        - Decimal Integer Literals
        - Octal Integer Literals
        - Hexadecimal Literals

    ii) Floating Point Literals
        - Fractional Form
        - Exponential Form
c) Boolean Literals
    - True
    - False
d) Special Literals – None

4. Operators
    a) Unary Operators
    b) Binary Operator
        i) Arithmetic operator
        ii) Bitwise Operators
        iii) Shift Operators
        iv) Identity Operators
        v) Relational Operators
        vi) Assignment Operators
        vii) Logical Operators
        viii) Membership operators
5. Punctuators

KEYWORDS

Definition : A keyword is a word having special meaning reserved by programming language. It convey a special meaning to the language compiler or interpreter and must no be used as normal identifier names.

Example :

False	True	def	for	in
while	from	is	class	print
continue	else	if	raise	break

IDENTIFIERS (NAMES)

Definition : Identifiers are fundamental building blocks of a program and are used as the general terminology for the names given to different parts of the program viz. variables, object, classes, functions, lists, dictionaries etc.

Rules to be followed while making an identifier in python:

- Identifier is a long sequence of letters and digits without space
- First character must be a letter(A-Z or a-z) or underscore (_)
- Upper and lower case are treated differently.
- Digits (0-9) can be part of the identifier except for the first character.
- Identifier is unlimited in length.
- Identifier must not be a keyword of python.
- Identifier cannot contain any special character (#$%^ etc) except for underscore (_)

Example :

Valid	Myfile	_myfile	My_File	My5File123	FOR
Invalid	My File	2Myfile	My.File	My-File123	for

LITERALS / VALUES

Definition : Literals (often referred to as constant value) are data items that have a fixed value.

Several kind of literals are :

A. String Literals :

A string literals is a sequence of characters surrounded by quotes.
The text enclosed in quotes (single or double) forms a string literal in python.
Example : 'b' "b" "JOHN" 'Mary' "This is my car" 'House no. D-91' "3214"

String Literals also have certain Non-graphic character string that cannot be typed directly (except for \ , ' , ") from keyboard which are represented by using escape sequence as follows:
Escape Sequence is treated as single character and hence one byte in ASCII representation.

Escape Sequence is always preceded by back slash (\)

Sl. No.	Escape Sequence	What it does
1	\\	Baskslash (\)
2	\'	Single quote (')
3	\"	Double quote (")
4	\a	ASCII Bell (BEL)
5	\b	ASCII Backspace (BS)
6	\f	ASCII Formfeed (FF)
7	\n	New line character
8	\t	Horizontal Tab (TAB)
9	\r	Carriage Return (CR)

10	\N(name)	Character named name in the Unicode database (Chapter – 2)
11	\uxxxx	Character with 16-bits hex value xxxx (Unicode only)
12	\v	ASCII Vertical Tab (VT)
13	\ooo	Character with octal value ooo
14	\xhh	Character with hex value hh

Note : Single quote can be directly type inside double quote or vice versa.

For example :

	Python print statement	Output/result
Valid	>>> print (“ John’s “)	>>> John’s
Valid	>>> print (‘ John”s ‘)	>>>John”s
Invalid	>>> print (‘ John’s ‘)	>>>ERROR
Invalid	>>> print (“ John”s “)	>>> ERROR

String Types in Python :

Python allows two types of strings :

1. Single-line Strings (Basic strings) : String enclosed within single quotes (i.e ‘ ‘) or double quotes (i.e. “ “) and terminate in one line.

   Example :

Valid	>>>print(“Hello World”)	>>>Hello World
Invalid	>>>print(“Hello World”)	>>>ERROR

2. Multiline Strings :

   i) By adding a backslash at the end of normal single-quote / double-quote.

   Example :

Valid	>>>print(“Hello \ World”)	>>>Hello World

   ii) By typing the text in triple quotation marks (single or double).

   Example :

Valid	>>>print(“ “ “Hello World. Here I come.” ” ”)	>>>Hello World. Here I come.
Valid	>>>print(‘ ’ ’Hello World. Here I come.’ ’ ’)	>>>Hello World. Here I come.

Size of string :

- len(<object name>) is a built-in function use for returning size of the object.
- One character : One byte.

  Eg.

>>>a = “X” >>>len(a)	 Means ‘X’ is one byte

1	

- Escape sequence : Treated as one character so One byte.
  Example:

>>>a = "\n" >>>len(a) 1	Means New Line character '\n' is one byte
>>>a = "A\nB\tC" >>>len (a) 5	Output 5 means that string "A\nB\tC" is 5 bytes =A is 1 byte =\n is 1 byte =B is 1 byte =\n is 1byte =C is also 1 byte Total = 5 bytes

- For multiline string EOL (End of Line) is also counted as 1 byte)

>>>str = """A B C""" >>>len(str) 5	Output 5 means that string str is 5 bytes as there are 3 character A B C and 2 EOL
>>>atr = """John is coming""" >>>len(atr) 14	Output 14 means that string atr is 14 bytes 2 EOL and 12 characters Total 14

- By adding backslash (\ ) for multiline string zero byte is counted.
  Example:

>>>a = "A\ B\ C" >>>len(a) 3	**Output 3 means str is 3 bytes as it does'nt count **

B. Numeric Literals:

Different types of numeric literals:

1. ***Integer literals :*** Integer literals are whole numbers without any fractional part.
   Three types of integer literals:
   a) *Decimal Integer Literals (Base 10) :* Any numeric integer consisting of digits 0-9 and begins with digit 1-9 except 0(zero)

b) *Octal Integer Literals (Base 8):* A sequence of digits 0-7 starting with 0o (digit 0-zero follow by letter 'o').

c) *Hexadecimal Integer Literals (Base 16):* A sequence of digits 0-F starting with 0x or 0X (digit 0-zero follow by letter 'x' or 'X').

Binary(Base 2)	Octal(Base 8)	Decimal(Base10)	Hexadecimal(Base 16)
0000 0000	0o0	0	0x0
0000 0001	0o1	1	0x1
0000 0010	0o2	2	0x2
0000 0011	0o3	3	0x3
0000 0100	0o4	4	0x4
0000 0101	0o5	5	0x5
0000 0110	0o6	6	0x6
0000 0111	0o7	7	0x7
0000 1000	0o10	8	0x8
0000 1001	0o11	9	0x9
0000 1010	0o12	10	0xA
0000 1011	0o13	11	0xB
0000 1100	0o14	12	0xC
0000 1101	0o15	13	0xD
0000 1110	0o16	14	0xE
0000 1111	0o17	15	0xF
	0o20	***16***	***0x10***
	0o21	***17***	***0x11***

2. ***Floating Point Literals :*** Floating point literals are real literals having fractional parts or decimal point.

   It can be written in two forms :

   1. Fractional form : Having atleast one decimal point. E.g. : 0.2 , 23.2 etc.
   2. Exponential form : It has a mantissa followed by exponent (letter E).

      E.g. :

6E03	$6 * 10^3$	6000.0
6E-3	$6 * 10^{-3}$	0.006
6.5E11	$6.5 * 10^{11}$	650000000000
6.5E-12	$6.5 * 10^{-12}$	0.0000000000065
6.5E	Invalid	
6.5E3.2	Invalid	

C. Boolean Literals :

In python there are two Boolean Literals, True to represent Boolean True and False to represent Boolean False.

D. Special Literals 'None' :

The None literal is used to represent absence of value and also used to indicate the end of lists in Python which signifies "There is no useful information" or "There's nothing here".

>>> a = None >>>print(a)	 >>>None

OPERATORS IN PYTHON

Operators : Operators are tokens that trigger some computation when applied to variables and other objects in an expression.

Operands : Any objects(variable/literals etc.) to which the computation is applied are called Operands.

So, both cannot work alone.

Example : >>>area_of_triangle = 0.5 * base * height

In the above statement:

Operator : = *

Operand: area_of_triangle base height 0.5

Different types of operators:

1) Unary Operators
   - Unary plus **+**
   - Unary Minus **-**
   - Bitwise complement **~**
   - Logical negation **not**

2) Binary Operators
   a) Arithmetic operators
      - Addition **+**
      - Subtraction **-**
      - Multiplication *****
      - Division **/**
      - Remainder/Modulus **%**
      - Exponent(raise to power) ******
      - Floor Division **//**

   b) Bitwise operators
      - Bitwise AND **&**
      - Bitwise exclusive OR or **(XOR)** **^**
      - Bitwise OR !

   c) Shift operators
      - Shift Left **<<**
      - Shift Right **>>**

   d) Identity operators
      - Is the identity same ? **is**
      - Is the identity not same ? **is not**

   e) Relational operators

- Less than **<**
- Greater than **>**
- Less than or equal to **>=**
- Greater than or equal to **<=**
- Equal to **==**
- Not equal to **!=**

f) Assignment operators

- Assignment **=**
- Assign quotient **/=**
- Assign sum **+=**
- Assign product ***=**
- Assign remainder **%=**
- Assign different **-=**
- Assign exponent ****=**
- Assign floor division **//=**

g) Logical Operators

- Logical AND **and**
- Logical OR **or**

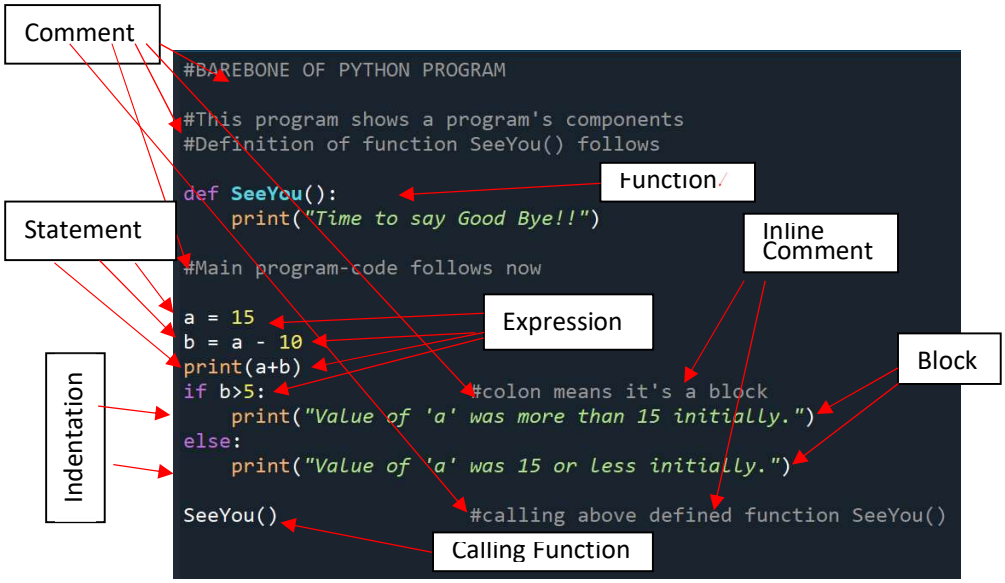
h) Membership operators

- Whether variable in sequence **in**
- Whether variable not in sequence **not in**

PUNCTUATORS

- Punctuators are symbols that are used in programming languages to organize programming-sentence structures, and indicate the rhythm and emphasis of ecpressions, statements, and program structure.
- Most Common Punctuators are : **‘ “ # \ ( ) { } { } @ , ; . =**

BAREBONE OF A PYTHON PROGRAM



COMPONENTS OF PYTHON PROGRAMMING:

1. Expressions: Any Legal combination of symbols that represent a value which python work upon and produce another value/output.
   Example : 15, a-15, a+b, b>5

2. Statements : A statement is a program instruction that perform some action. A statement may or may not yield a value
   Example :
   a = 15
   b = a - 10
   print(a+b)
   if b>5:
   etc.

3. Comments : Comments are the additional readable information to clarify the source code for documentation. It begins the symbol #. Comment is not executable during the program execution and use only during coding (while writing the program).

   Different ways of making comment in python coding are:
   a) Full line comment : A comment begins with # a complete in one line.
      Example : #This program shows a program's components

   b) Inline comment : This is also a full line comment but appear in the middle of a physical line after Python code.
      Example : if b>5: #colon means it's a block

c) Multiline comment with symbol # : We can make a multiline comment by using a number of full line comment adding symbol # at the beginning of every comment line.
Example :
```
#This program shows a program's components
#Definition of function SeeYou() follows
```

d) Docstring : A multiline comment with a pair of triple single quote (‘) or double quote (“).
Example :
```
‘ ’ ’This program shows a program's components
    Definition of function SeeYou() follows’ ‘ ‘
            OR
“ ” ”This program shows a program's components
    Definition of function SeeYou() follows ” “ “
```

4. Functions : A function is sub-routine that has a name which can be reused (call again to executive) by specifying its name in the program when needed.
Example :
```
def SeeYou():
print("Time to say Good Bye!!")

SeeYou()    #a function call which trigger the above function SeeYou()
```

5. Blocks and Indentation : A group of statements which are part of another statement or a function are called block or code-block or suite in Python. Starting of the block is identify by colon (:) and it uses indentation to identify the block in python.
Example :
```
if b>5:
        print("Value of 'a' was more than 15 initially.")
else:
        print("Value of 'a' was 15 or less initially.")
```

PYTHON STYLE RULES AND CONVENTION

1. Statement termination : By pressing Enter Key which is considered as terminated by default.
2. Maximum Line Length : 79 characters.
3. Lines and indentation : Indentation is enforced through 4 spaces (using spacebar key but not tab key) per indentation Level.
4. Blank Lines : Functions and Methods should be separated by two blank lines and Class definition by three blank lines.
5. Whitespace : Insert whitespace round operators and punctuators but not with parenthesis ().
6. Case Sensitive
7. Docstring
8. Identifier Naming : Use underscore between words or Capitalizing the first letter.

VARIABLE in PYTHON

A variable in python represents named location that refers to a value and whose values can be used and processed during program execution or run. Python variable are not storage container and do not have a fixed location.

ASSIGNMENT OF PYTHON VARIABLE

Assignment means giving value to a variable.

Creating a Variable and assignment of value to variable:

Python variable is created only after some value is assign to it. Python create a variable of type similar to the type of value assigned.

Consider all the statements below

Statement	Comment
Mark = 32	This means the variable name is Mark and it is integer type (numeric)
Mark = 34.2	This means the variable name is Mark and it is float type (numeric)
Mark = "Fouty Five"	This means the variable name is Mark and it is String type

Name Error : Use of undefined variables in an expression/statement cause an error called Name Error.

```
>>>print(x)   #NameError
```

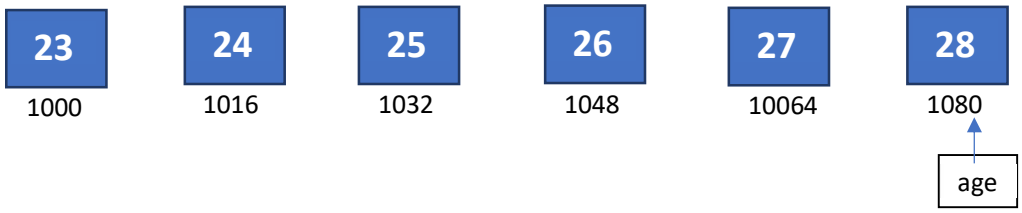
Python variable are not a storage container and do not have a fixed location. Explain.

Front-loaded dataspace is a preloads of some commonly used value even before any identifier is created. So, variables in python program refers to the location of front-loaded dataspace instead of holding data.

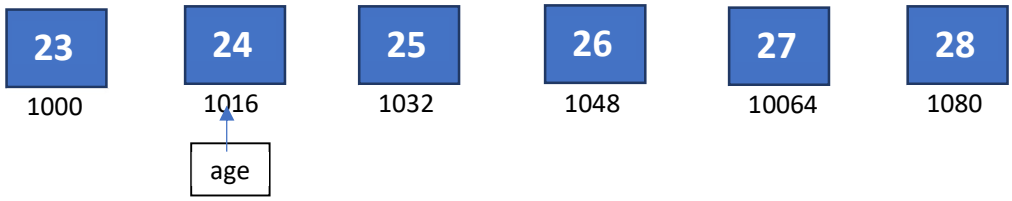
Example :

```
>>> age = 28
>>> age = 24
```

First assignment statement >>> age = 28 will make variable age to refer to location 1080, so value of age will equal to 28.



Second assignment statement >>> age = 24 will make variable age to refer to location 1016, so value of age will equal to 24.



Lvalues and Rvalues:

Lvalue : Expression that can come on the left hand side of an assignment statement. Lvalue can be only one variable (not literals) and can appear on the right hand side as well.

Rvalue : Expression that can come on the right hand side of an assignment statement. Rvalue object can be a variable and literal or both together.

>>> age = 23	Valid
>>> a = 3 >>> age = a	Valid
>>> age = 2 >>> age = age + 21	
>>> 2 = age	Invalid
>>> age + 3 = 4	Invalid

Individual assignment of variable:

```
>>> a = 3
```

Multiple assignment of variable:

Two ways :

1. *Assigning of same value to multiple variables at one time:*

```
>>> a = b = c = 19    # all the value of variable of a, b, c are 19
>>> print (a,b,c)
19 19 19
```

2. *Assigning of different values to multiple variables in one line:*

```
>>> a,b,c = 1, 2 , 3
>>> print (a,b,c)
1 2 3
```

Some more example :

a) *Swapping the value of two variables :*

(Swapping means interchange of value between 2 variables)

```
>>> a, b = 1, 2
>>> print(a, b)
1 2
>>> a, b = b, a
>>> print(a, b)
2 1
```

b) *Python first evaluates all the expression on the RHS and then assign to LHS variables.*

```
>>> a, b, c = 1, 2 , 3
>>> b, c, a = a + 5, b – a, c * 3
>>> print(a, b, c)
9  6 1
```

c) *Expression are evaluated from Left to Right on the RHS.*

```
>>> x = 10
```

```
>>> y, y = x + 5, x + 10
>>> print (y)
20
```

DYNAMIC TYPING:

A variable pointing to a value of a certain type can be made to point to a value/object of different type is Dynamic Typing.

Example :

```
x = 2                  # here variable x is integer type
print(x)               # interger number 2 will be printed
x = "Hello World"      # now variable x is string type
print(x)               # Hello world will be printed without any error

Output will be:
2
Hello World
```

Caution with to be taken:

```
X = "hello world"
Y = X / 3
ERROR!! Because string cannot be divided.
```

DETERMINING OF TYPES OF VARIABLE:

Type of any variable can be determined by using command

type (<object name>)

Where, type is a keyword and *<object name>* could be literal, variable or expression.

INPUT FUNCTION

To get input from user we use built-in function ***input( )***

Syntax:

For String value : ***Variable_name = input (< promt to be display >)***

For Interger value : ***Variable_name = int (input (< prompt string >))***

For Float value : ***Variable_name = float (input (< prompt string >))***

Example:

```
>>> name = input ("Enter City : ")
>>> Enter City : Aizawl
>>> name
Aizawl

>>> Roll_No = int ( input ("Enter Your Roll Number :"))
>>> Enter Your Roll Number : 34
>>> Roll_No
34

>>> Sang_Zawng = int ( input ("Enter Your Height in cm :"))
>>> Enter Your Height in cm :170.3
>>> print ( Sang_Zawng)
170.3
```

Note : input( ) function always return string type by default.
Example :

```
>>> kum_zat = input ("Enter Age : ")
>>> Enter Age : 18
>>> kum_zat / 3
ERROR!
```

ERROR!, because as the python rules say Python create a variable of type similar to the type of value assigned to it and also says that input( ) function is string type by default. Dividing string value is not permitted so it raise an ERROR! report.

OUTPUT FUNCTION

To display output on the screen/monitor we use built-in function ***print ( )***
Syntax :

print (*objects)

Example :

print ("Python Program")	Python program
print (23.21)	23.21
print (2 + 45)	47
A = 3 B = 32 print(A – B) print ("A leh B belh chu : ", A + B) C = A + B print (A, "leh", B , "belh chu :" , C)	 -29 A leh B belh chu : 35 3 leh 32 belh chu : 35
Print ( )	Emty print function will print blank line

Features of print ( ) built-in function :

1. It auto-converts the item to strings type.
2. It insert space between items.
   Example : print ("My","Home)
   Will insert space between 'My' and 'Home', thus output will be My Home instead of MyHome.
3. It append a newline character '\n' at the end.
   Example : *print("My name is John")*
   print("I am 19 years old")

   Output will be:
   My name is John
   I am 19 years old
   Instead of printing in one line :
   My name is John I am 19 years old

4. We can explicitly give an **end** argument.

Example : *print("My name is John.", end = '$')*
print("I am 19 years old.")

Output will be:
My name is John. $I am 19 years old.

Instead of printing in one line :
My name is John.
I am 19 years old

5. print ( ) can be break down by using backslash (\ ).
Example : *print ("Sum of 2 and 3 is",* \
5)
Output : Sum of 2 and 3 is 5