

# PYTHON

|          |            |
|----------|------------|
| Page No. |            |
| Date     | 22/10/2021 |

\* Python Pluses (Pros) :- (features/characteristics)

i) Easy to Use

• As it supports OOP (Object Oriented Programming)

• Syntax are very simple.

ii) Expressive Language

Fewer lines of code

Swapping of two numbers :

| C/C++                            | Python         |
|----------------------------------|----------------|
| ( $24 \rightarrow 42$ )          | $a, b = 2, 4$  |
| <code>int a=2, b=4, temp;</code> | <del>def</del> |
| <code>temp=a;</code>             | $a, b = b, a$  |
| <code>a=b;</code>                | <hr/>          |
| <code>b=temp;</code>             | $(a=4, b=2)$   |
| (Now,<br>$a=4$<br>$b=2$ )        |                |

iii) Interpreted language

↳ Executes code line by line

iv) Completeness - (Battery Included)  $\rightarrow$  known as

↳ Libraries (built-in)

(Python Standard Libraries)

v) Cross Platform / Portable language

Ex :- Ubuntu / Windows / Linux / Unix / Mac / DOS etc ..

• Python can work on any of these \* .

vi) Free & Open Source

vii) Variety of usage / Application

- Scripting
- Web Application
- Game Development
- Rapid Prototype
- GUI Programs
- etc.

• System Administration  
• Database Applications

viii) Good Memory utilization .

\* Python Minuses (Cons) / Disadvantages / Limitations / short coming / lacking :-

i) Not the fastest language

ii) Less libraries than C , Java & Perl .

\* iii) Not strong on type-binding .  
↳ (No type declaration is there)  
in Python

iv) Not easily convertible .

↳ because of simple syntax .

\* for programming in Python -

→ Python IDLE , pip (Package Installer  
Anaconda Package)

↳ Numpy , Scipy , Panda Libraries .

else → Spyder IDE , Pycharm IDE .

## Modes of Python

|          |  |
|----------|--|
| Page No. |  |
| Date     |  |

- i) Interactive mode / Immediate mode / Interpreter
- ii) Scripting Mode

# PYTHON

|          |            |
|----------|------------|
| Page No. |            |
| Date     | 27/10/2021 |

## Building Blocks

### \* Character Sets -

Supports Unicode encoding standard.  
Encoding Standards -



- Alphabet A-Z, a-z.

- 0-9

- Special Symbols

- White Spaces (Includes - Blank space, Tab, carriage return(↵), new line, etc.)

- Other characters - ASCII, unicode, etc.

### \* Tokens -

Smallest individual units in a program is known as a token or a ~~flexible unit~~ lexical unit.

Includes →

Keywords, Identifiers,  
literals, operators, punctuation.

(of ~~any~~ one type)

### \* Keywords -

false, assert, del, for, in, etc.

↳ are the words having special meaning reserved by the programming language.

### \* Identifier -

Names (used to identify).

|      |  |
|------|--|
| Name |  |
| Date |  |

## Identifier naming conventions -

- i) length can be any long that is supported by your system.
- ii) first character should be a letter.
- iii) It is a case-sensitive language - i.e. remember the casing of names.
- iv) 0-9 are valid except at first location.
- v) Must not be a keyword.
- vi) No special characters allowed in names except underscore (-).

## \* Literals (values) -

$$x = \textcircled{1} \text{ a}$$

Literal

↳ Literal often referred to as constant or any value.

Includes -

- i) String literal
- ii) Numeric
- iii) Boolean
- iv) Special Literal → (NULL)
- v) Literal Collection

### i) String Literals -

character sequences surrounded by ' ' / " " . Ex - ( "Akash") ,  
Graphic characters

Punctuators → . , " " etc.

|          |  |  |
|----------|--|--|
| Page No. |  |  |
| Date     |  |  |

Non-Graphic Characters - (It is also string literals)

- Escape Sequences

These are ~~to~~ started by a \ (backslash).

Ex- \t, \n, \l, \", etc.

\* → These are counted as only one character in memory.

Types of String Literals -

- Single Line String (Also called in-line)
- Multi-line String

Ex- 'Hello

Sir'

X

'Hello'  
Sir'

multi-line  
String

Ex- 'Hello World!'

↓

Single Line String (In-Line)

OR

'Hello'  
world'

OR

(Using " " quotes)

"Hello"

World""

# Python

## String Literals

### \* Size of String -

"abc" = 3 bytes

"\\" = 1 byte

"ab" = 2 bytes

(new line, size = 0 bytes)

str-3 = "b\n" byte = 3

a  
b

c  
"

(Enter = 1 byte / 1 character)

str-2 = "b\na  
c  
"  
byte = 5

a  
b

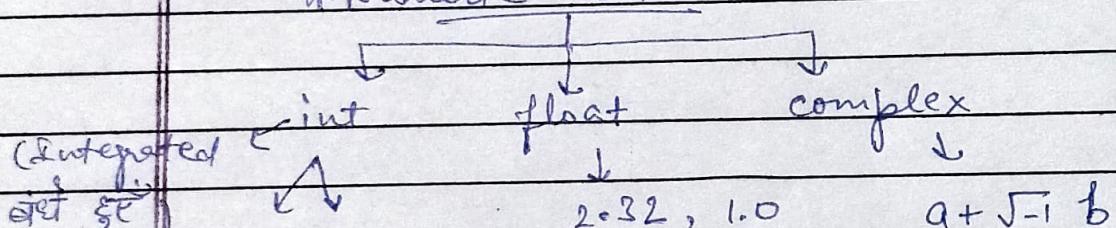
c  
"

\* Size function  $\Rightarrow$  size("string")  $\rightarrow$  returns size of string

On calling  
size(str-3) will return 3.

### \* Literals -

#### \* Numeric Literals



Signed Unsigned

1, 2, -1, -2

$\Rightarrow$  Comma cannot be used in numeric literals

### \* Types of Integer literals

- Decimal int literal (0-9)

123, 414  
✓

012  
✗

→ 0 must not be the first digit.

- Octal integer literal (0-7)

→ 0 must be used at first place.

012, 011  
✓

$$(10)_8 = (8)_{10}$$

$10 \atop 1$        $15 \atop 1$

- Hexadecimal integer literal (0-9 and A-F)

$$(XA)_{16} = (10)_{10}$$

### \* Floating point literal

exponent form

$$2.0 \rightarrow 0.2 \times 10^1 \text{ or } 0.2E01$$

④  $av = 14, 67$

↳ 67 is stored in tuple.

### \* Boolean Literal

→ T/F

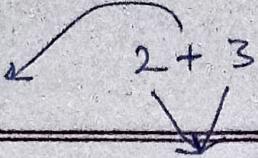
### \* Special Literal → "NONE"

$$x = \text{NONE}$$

↳ It means / represents absence of value.

## Operators

↳ used on operands.



### i) Unary Operator -

+ → Unary plus (increments) +a

- → minus

~ → Bitwise complement  $-01 = 10$

not → logical negation

### ii) Binary operator -

+, -, /, \*% → Reminder Modulus, // Floor

Division.

$3//2$  gives

1 instead  
of 1.

### iii) Bitwise -

& → Bitwise AND

^ → Bitwise exclusive OR / XOR

| → bitwise OR

### iv) Shift operator -

<<, >>

left shift

<< 0110 → 1100

### v) Identity operator -

is, is not.

### vi) Relational operator -

<, >, ==, !=, <=, >=,

vii) Assignment Operator -

=, /=, \*=, +=, -=, //=

a = 3 ; (3 is assigned to a)

viii) Logical Operator -

and, or

ix) Membership Operator -

in, not in

\* Punctuators - ' . " # \ , . [ ] { } @ =

# Python

Page No.

Date 11 11 21

## \* Python string format methods -

```
txt = "For only {price:.3f} dollars!"  
print(txt.format(price = 49))
```

Output -

For only 49.000 dollars!

{ } → Placeholder  
Holder

format() -

Syntax:

string.format(values, values, ...)

add %c %f  
format specifiers

In python,  
percentage

specifier.

Placeholder → { }

Named Indexes, Number indexes, empty

txt1 = "My name is {fname}, I am {age}."  
format (fname = "John", age = 36)

Output - (txt1)

My name is John, I am 36

txt2 = "My name is {0}, I am {1}.".format  
( "John", 36 )

txt3 = "My name is {3}, I am {2}.".format  
( "John", 36 )

Output -

My name is John, I am 36

( Same output for txt2, txt3 )

i) `:<` → left align the result

$\rightarrow$  upto 2 no change in output  
`txt = "we have {:<8} chicken."`  
`print(txt.format(49))`.

Output -

we have 49 ----- chicken.  
 -----  
 ↴ whitespace  
 (8 unit places)

ii) `:>` → right align the result

iii) `:^` → centered align

iv) `:=`

`txt = "the temp is {:=8} degree celcius."`  
`print(txt.format(-5))`

Output -

the temp is -----5 degree celcius.

v) `:b` → binary format

`txt = "the binary version of {0} is {0:b}."`  
`print(txt.format(5))`

Output -

The binary version of 5 is 101.

vi) `:d` → decimal format

vii) `:%` → percentage format

`txt = "You scored {0}%."`

`print(txt.format(0.25))`

Output -

You scored 25,000000%.

## \* String Operators -

i) + → concatenation operator

↳ concatenate two strings,

`str0 = "Hello"`

`str1 = "World"`

`print(str0 + str1)` → Output

Hello World

ii) \* → repetition operator

`print(str0 * 3)` → Output

Hello Hello Hello

iii) [:] → Range ~~slice~~ Operator

`print(str0[2:4])` → Output

↓ ↓ ll

From To(exclusive)  
(inclusive)

iv) [] → slice operator

`print(str0[2])` → Output

↓ l  
at index

v) in → includes (returns True or False)

vi) not in → opposite to in operator

vii) N/R → to specify new string

`print(r"\n\t;/ python37")` → Output  
(;/python37

viii) % → percentage method

|       |  |
|-------|--|
| Topic |  |
| Date  |  |

print ("The string stro: %s " % (stro))

output →

The string stro: hello

## \* Types of statements

i) Empty statement

→ pass

• null operation statement

ii) Simple statement

→ print ("Hello World")

iii) Compound statement (In blocks)

→

header

→ [ If  $a > b$  :

—  
—

] body

## \* Statement Flow Control -

Part  
IV