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 → 42)	a, b = 2, 4
int a=2, b=4, temp;	a, b = b, a
temp = a;	
a = b;	(a=4, b=2)
b = temp;	
(Now, a=4 b=2)	

iii) Interpreted Language
↳ (Executes code line by line)

iv) Completeness (Battery Included) → known as
↳ libraries (built-in)
(Python Standard libraries)

v) Cross Platform / Portable language

Ex :- Ubuntu / Window / 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 Misuses (Cons) / Disadvantages / Limitations / short coming / Lacking :-

i) Not the fastest language

ii) Less libraries than C, Java & Perl.

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

iv) Not easily convertible.
↳ because of simple syntax.

---

* For programming in Python -

→ Python IDLE, pip (Package Installer Package)

Anaconda
↳ 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~~ lexical unit.
  ~~(of only one type)~~

Includes →
keywords, identifiers, literals, operators, punctuation.

* Keywords -
  False, assert, del, for, in, etc.
  ↳ are the words having special meaning reserved by the programming language.

* Identifier -
  Names (used to identify).

Page No.
Date

Identifier naming conventions -

i) Length can be any long that is supported by your system.
ii) Front 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 = (1)
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 started by a \ (backslash).

Ex- \t , \n , \\ , \" , 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' ✗

'Hello Sir' ✓ ↑ Multi-line String

Ex- 'Hello World!'

↓

Single Line String (In-Line)

↓ OR

'Hello\ world'

OR

(using """ quotes)

""" Hello World """

Page No.
Date 9 | 11 | 2021

Python

String Literals

* Size of String –

"abc" = 3 bytes
"11" = 1 byte
"\ab" = 2 bytes

str_3 = "" b (new line, size = 0 byte)
a
c"" byte = 3

str_2 = "" b ↵ (Enter = 1 byte / 1 character)
a ↵ byte = 5
c""

- Size function ⇒ Size ("string") → returns size of string

On calling ↑ size(str_3) will return 3.

* Literals –

* Numeric Literals

int (Integrated अर्थ इंट) — Signed, unsigned — 1, 2, -1, -2
float — 2.32, 1.0
complex — $a + \sqrt{-1}\, b$

* Comma cannot be used in numeric literals

Page No.
Date

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

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

$(XA)_{16} = (10)_{10}$ (10 and 15)

* Floating point literal

exponent form

2.0 → 0.2×10^1 OR 0.2E01

(a) av = 14.67

↳ 67 is stored in tuple.

* Boolean Literal

→ T/F

* Special Literal → "NONE"

x = NONE

↳ It means / represents absence of value.

Operators

2 + 3

↳ 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 1.5

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 -

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

Page No.
Date

viii) 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))
```

{} → Place Holder

Output -

For only 49.000 dollars!

format() -

Syntax: string.format(values, values, ---)

%d %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 {}, I am {}.format
    ("John", 36)
```

Output -

My name is John, I am 36

(Same output for txt2, txt3)

i) :< → left align the result
→ (upto 2 no change in output)

```
txt = "we have {:<8} chicken."
print(txt.format(49))
```

Output —

we have 49 _____ chicken.
(8 unit places)
whitespace

ii) :> → right align the result
iii) :^ → centred aligned
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 {:.2%}."
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) * → repeatation operator

```
print(str0*3) → Output
```

HelloHelloHello

iii) [:] → Range Slice Operator

```
print(str0[2:4]) → Output
```

ll

(2 → From (inclusive), 4 → To (exclusive))

iv) [] → slice operator

```
print(str0[2]) → Output
```

l

(2 → at index)

v) in → includes (returns True or false)

vi) not in → opposite to in operator

vii) r/R → to specify new string

```
print(r"://python37") → output
```

://python37

viii) % → percentage method

print ("The string str0: %s " % (str0))

output →

The string str0 : hello

* Types of statements

i) Empty statement

→ pass

• null operation statement

ii) Simple statement

→ print ("Hello World")

iii) Compound statement (in blocks)

→

→ If a > b :

header

body

* ~~Statement Flow Control~~ -