Python

Date
Page 1

youtube : Apna college

(Python Tutorial for beginners)

Python :

- Python is simple & easy
- Free & open source
- High level language
- Developed by "Guido van Rossum"
- Portable

- Print ("Hello world")
  ↓
  (Function)

- Python character set :
  - Letters : A to z, a to z
  - Digits : 0 to 9
  - special symbols : +, -, *, / etc
  - whitespaces : Blank space, tab, carriage return, newline, Formfeed
  - other character : Python can Process all Ascii & unicode character as part of data or literals

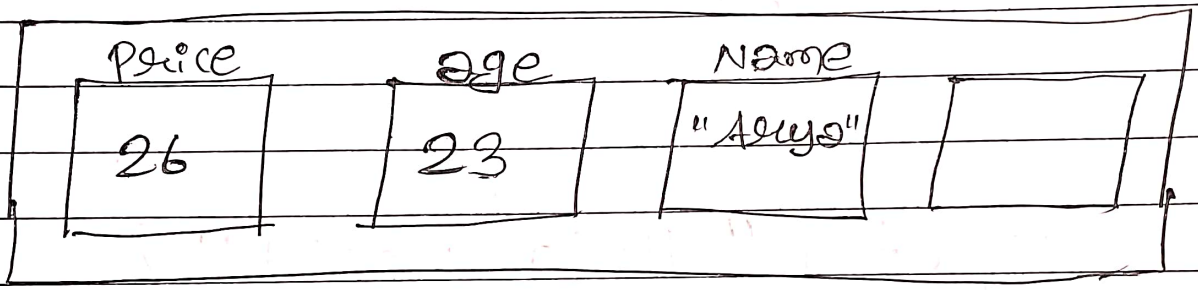
- **Variables :**

A variables is a name given to a memory location in a Program.

↱ variables ↱ value

name = "Arya"

- **Memory :**



name = "Arya"
age = 23
Price = 26

"= (Assignment operator)"

- **Data Types :**
  - integers
  - string
  - float
  - Boolean (True / False)
  - None

Types of Tokens :
(), {}, @, [], # etc
Punctuators are symbols to organize sentence structure in Programming.

Typed language

Implicit (Eg : Python)

Explicit (Eg : Java, C++)

- Name = "Arya" (Python)
  string

- age = 23 (Python)
  int

- int age = 23 (Java)
  int

Expression Execution

- String & Numeric values can operate together with "*".

```
A, B = 2, 3
Txt = "@" (string)
Print (2* txt * 3)
```

```
output :
@@ @@@@
```

- string & string can operate with "+".

```
A, B = "2", 3
txt = "@"
Print ((A + txt) x B)
```

```
output :
2@ 2@ 2@
```

- Numeric values can operate with all arithmetic operators.

```
A ; B. = 2, 3
c = 4
```

```
Print (A+B*C)

Output :
    14
```

- Arithmetic expression with Integer and Float will result in Float

```
A, B = 10, 5.0
C = A * B
Print (C)

Output :
    50.0
```

- Result of division operator with two integers will be Float

```
A, B = 1, 2
C = A / B
Print (C)

Output :
    0.5
```

- Integer division with float and int will give int displayed as float.

```
A, B = 1.5, 3
C = A//B
Print (C, A/B)
```

Output
0.0 0.5

- Floor gives closest integer, which is lesser than or equal to the float value.

Result of (A//B) is same as Floor (A/B)

Floor(Number) = closest integer no
Finding

```
A, B = 12, 5
C = A//B
Print(C)
```

Output : 2

- A, B = -12, 5
  C = A // B
  Print (C)

  Output
  -3

- A, B = 12, -5
  C = A // B
  Print (C)

  Output
  -3

- Remainder is Negative when denominator is Negative.

A, B = -5, 2
C = A % B
Print (C)

Output :-
1

A = -5 (-ve)
B = 2 (+ve)
C = -5/2

2) -5 (2
   -4

    1

A, B = 5, 2
C = A % B
Print (C)

output
1

A, B = 5, -2
C = A % B
Print (C)

output
-1

% (Modulo (Reminder))

N	+	−	+	−
D	+	−	−	+
=	+	+	-ve	+

Comments in Python :

- #single line comments

- """ This is
  a multi-line
  comments """

Print ("Arya") → won't be printed
Print ("Ay") → will be Printed

Input in Python :

- Input () statement is used to accept values (using keyboard) From user

  - String Input
    name = Input ("Name : ")

  - Int Input
    age = Int (Input "age"))

  - Float Input
    Price = Float (Input ("Price : "))

- Types of operators :
  - Arithmetic operators
    (+, -, *, /, //, %, **)
  - Relational / comparision operator
    ~~(==, --, *=, /=~~
    = (==, !=, >, <, >=, <=)
  - Assignment operator.
    (=, -=, *=, /=, %=, //=, **=)
  - Logical operators
    (Not, and, or)
  - Membership operator
    (in, not in)
  - Identity operator
    (is, is not)
  - Bitwise operator
    (&, |, ^)

- Conditional statement

if - elif - else (syntax)

```
if (condition) :
    statement 1
elif (condition) :
    statement 2
else :
    statement N
```

4 space

Intentation (Proper spacing)

Eg :- (Traffic Lights code)

```
Light = input ("Light : ")
if (light == "red") :
    Print ("stop")
elif (light == "yellow") :
    Print ("look")
elif (light == "green") :
    Print ("go")
else :
    Print ("Light is broken")
```

- Single Line if / Ternary operator :

<var> = <val1> if <condition> else <val2>

Eg :

- Food = input ("Food :")
  eat = "yes" if Food == "cake" else "no"

  Print (eat)

- <stat1> if <condition> else <stat2>

  Food = input ("Food")
  Print ("sweet") if Food == "cake" or Food == "Jalebi" else Print ("Not sweet")

- Clever if / Ternary operator :

- <var> = (false_val, true_val) [<condition>]

  age = int (input ("age"))
  vote = ("yes!", "no") [age >= 18]

```
sal = float(input("salary : "))
tax = sal * (0.1, 0.2)[sal <= 5000]
print(tax)
```

- Arithmetic operators

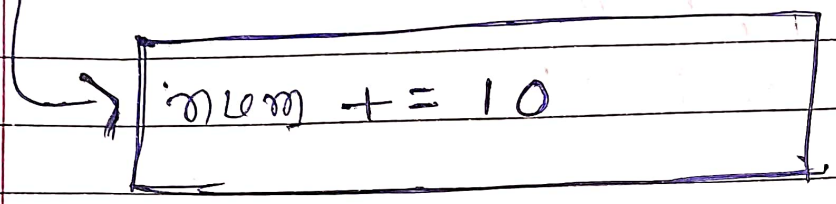
```
a = 5
b = 2
print(a+b)
print(a-b)
print(a*b)
print(a%b) # remainder
print(a**b) # a^b (Power)
```

- Relational operator

```
a = 50
b = 20
print(a==b) # False
print(a!=b) # True
print(a>=b) # True
print(a>b) # True
print(a<=b) # False
print(a<b) # False
```

- Assignment operator

```
num = 10
num = num +10 # 10+0 => 20
Print (num)
```



- Logical operator

```
a = 50
b = 30
Print (not False)
Print (not (a>b))

val1 = False
val2 = False
Print ("And operator :", val1 and val2)

Print ("OR operator :", (a==b) or
                        (a>b))
```

Type conversion

```
a, b = 1, 2.0
sum = a+b  # 3.0
```

error

```
a, b = 1, "2"
sum = a+b  # error
```

Input in Python

Input() statement is used to accept values (using keyboard) from user.

- input() # result for input() is always a str

- int(input()) # int
- float(input()) # float

String :

String is data type that stores a sequence of characters.

- concatenation

"hello" + "world" = helloworld

- Length of str
len(str)

\n = Next Line

- Indexing :

A	P	N	A	_	C	O	L	L	E	G	E
0	1	2	3	4	5	6	7	8	9	10	11

str = "Apna_college"
str[0] is 'A', str[1] is 'P'...
str[0] = 'B' # not allowed

- Slicing :
Accessing Parts of a string

str[starting_idx : ending_idx]
ending idx is not included

str = "Apna college"
str = [1:4] is "pna"

str[1:4] is "pna"
str[:4] is same as str[0:4]

str[1:] is same as str[1:len(str)]

- Negative index

A	P	P	L	E
-5	-4	-3	-2	-1

str = "Apple"
str = [-3:-1]

- String function :

str = "I am a coder."

str.endswith("er.")
# return true if string ends with substr

str.capitalize()
# capitalizes 1st char

str.replace(old, new)
# replace all occurrencess of old with new

str.find(word)
# returns 1st index of 1st occurrencess

str.count("am")
# counts the occurrencess of substr in string

Lists in Python

A built-in data type that stores set of values.
It can store elements of different types (integer, float, string, etc.)

Marks = [87, 64, 33, 95, 76]
# marks[0], marks[1] ...

student = ["Karan", 85, "Delhi"]
# student[0], student[1] ...

student[0] = "Arjun"
# allowed in Python

len(student)
# returns length

- List slicing :
similar to string slicing

list_name[starting_idx : ending_idx]
ending idx is not included

marks = [87, 64, 33, 95, 76]

marks[1:4] is [64, 33, 95]

marks[:4] is same as marks[0:4]

marks[1:] is same as marks[1:len(marks)]

marks[-3:-1] is [33, 95]

List methods :

```
list = [2, 1, 3]

list.append(4)
  # adds one element at the end

list.sort()
  # sorts in ascending order

list.sort(reverse = True)
  # sorts in descending order

list.reverse()
  # reverse list

list.insert(idx, el)
  # Insert element at index

list.remove(1)
  # removes First occurrence of
    element

list.pop(idx)
  # removes element at idx
```

Tuples in Python :

A built-in data type that lets us create immutable sequence of values.

```
tup = (87, 64, 33, 95, 76)
  # tup[0], tup[1] ...

tup[0] = 43
      # Not allowed in Python

tup1 = ()
tup2 = (1)
tup3 = (1, 2, 3)
```

Tuple methods :

```
tup = (2, 1, 3, 1)
tup.index(el)
# return index of first occurrence
  tup.index(1) is 1

tup.count(el)
 # counts total occurrence
  tup.count(1) is 2
```

File I/O in Python :

Python can be used to perform operations on a file. (read & write data).

- Types of all files

1. Text files : .txt, .docx, .log etc.
2. binary files : .mp4, .mov, .png, .jpeg etc.

- Open, read & close file :

We have to open a file before reading or writing.

```
f = open("filename", "mode")
```

mode :

- sample.txt : r : read mode
- demo.docx : w : write mode

```
data = f.read()
f.close()
```

character	Meaning
'r'	: open for reading (default)
'w'	: open for writing, truncating the file first
'x'	: create a new file and open it for writing
'a'	: open for writing, appending to the end of the end of the file if it exists
'b'	: binary mode
't'	: text mode (default)
'+'	: open a disk for updating (reading and writing)

- Reading a file :

```
data = F.read()
    # reads entire file

data = F.readline()
    # read one line at one
      a time.
```

- Writing to a file :

- f = open("demo.txt", "w")

- f.write("this is a new line")
  # overwrites the entire file

- f.open("demo.txt", "a")

- f.write("this is a new line")
  # adds to the file

- with syntax :

with open("demo.txt", "a") as f:
    data = f.read()

- Deleting a file :
  using the os module

Module (like a code library) is a file written by another Programmer that generally has a Functions we can use.

- Keywords are reserved words in Python :

• and	• else	• in
• return	• as	• except
• is	• True	• False
• assert	• finally	• lambda
• try	• break	• nonlocal
• with	• class	• For
• None	• while	• continue
• From	• Not	• yield
• def	• global	• or
• del	• If	• Pass
• elif	• Import	• raise

- case sensitive = "A & a is different"

- Print sum :

```
a = 2
b = 6
sum = a+b
Print (sum)
```

(Program)

Dictionary in Python :

Dictionaries are used to store data values in key : value pairs they are unordered, mutable (changeable) & don't allow duplicate keys.

```
dic = {
  "Name" : "Arya",
  "cgpa" : 8.6,
  "marks" : [98,97,95],
  }
```

```
dict["name"], dict["cgpa"],
dict["marks"]
```

```
dict["key"] = "value" #to assign
or add new
```

- Nested Dictionary :

```
Student = {
  "Name" : "Arya",
  "Score" : {
    "che" : 98,
    "Py" : 82,
    }
  }
```

Dictionary Methods :

```
MyDict.key()
        # return all keys

MyDict.values()
        # return all values

MyDict.items()
    # returns all (key, val) pairs
      as tuples

MyDict.get("key")
    # return the key according
    to value

MyDict.update(newDict)
    # inserts the specified items
to the dictionary
```

Date 25.
Page

- **Set In Python :**

Set is the collection of the unordered items. each element in the set must be unique & Immutable.

```
nums = {1,2,3,4}
set2 = {1,2,2,2,3}
    #repeated elements stored only once, so it resolved to {1,2,3}

null_set = set()
    #empty set syntax
```

- **set Methods :**

```
set.add(el) #adds an element

set.remove(el) #removes the elem

set.clear() #empties the set

set.pop() # removes a random value

set.union(set2)
    #combines both set values & return new
```

set.intersection(set2)
# combines common values & returns new

Loops in Python :
Loop are used to repeat instructions.

- while loop
- for loop

while loop

```
count = 1
while count <=5 :
    print("hello!")
    count += 1
```

- Print num from 1 to 5

```
i = 1
while i <=5
print(i)
  i += 1
print("Loop ended")
```

- Break & continue :

- Break :
  used to terminate the loop when encountered.

- continue :
  Terminates execution in the current iteration & continues execution of the loop with the next iteration.

```
i = 1
while i <= 5 :
    Print (i)
    if (i == 3) :
        break
    i += 1
Print ("end of loop")
```

```
i = 0
while i <= 5 :
    if (i == 3) :
        i += 1
        continue
    Print (i)
    i += 1
```

- For loop :

For loop are used for sequential traversal. For traversing list, string, tuples etc.

```
nums = [1, 2, 3, 4, 5]
for val in nums:
    print(val)
```

```
str = "apna college"
for char in str:
    if (char == 'o'):
        print("o found")
        break
    print(char)
print("End")
```

- Range() :

Range Functions returns a sequence of numbers, starting from 0 by default & increments by 1 (by default) and stops before a specified number.

```
range(start ?, stop, step ?)
```

```
for i in range(10):
    # range(stop)
    print(i)
```

```
for i in range(2,10):
# range(start, stop)
    print(i)
```

```
for i in range(2,10,2):
# range(start, stop, step)
    print(i)
```

Pass statement :

Pass is a null statement that does nothing. it is used as a placeholder for future code.

```
for i in range(5):
    pass
    print("some useful work")
```

Functions in Python :

" Block of statements that Perform a specific task.

```
def func_name(Param1, Param2 --):
    # Some work
    return val
Func_name(arg1, arg2 --)
    # Function call
```

Eg :

```
def calc_sum (a, b) : #Parameters
    sum = a+b
    Print (sum)
    return sum
calc_sum (5, 10) # Function call, Argum.
calc_sum (2, 10) #  "        "
```

- Built-in Functions :
  - Print ()
  - len ()
  - type ()
  - range ()

- User define Function :

- Recursion :

when a function calls itself repeatedly.

Prints n to 1 backwards

```
def show(n):
    if (n == 0):        (Base case)
        return
    print(n)
    show(n-1)
```

```
import os
os.remove (file name)
```

OOP in Python :

To map with real-world scenarios, we started using objects in code. This is called OOP (object oriented Programming).

- class & object in Python :

class is a blueprint for creating objects.

```
# creating class

class student :
    name = "Karan Kumar"

# creating object (instance)
s1 = student()
Print (s1.name)
```

• __init__ Function :

All classes have a function called __init__(), which is always executed when the class is being initiated.

constructor = object creating

creating class

```
class student :
    def __init__ (self, full.name);
    self.name = full name
```

creating object

```
s1 = student ("Koron")
print (s1.name)
```

- class & instance Attributes :

- class. attr
- obj. attr

- Methods :
Methods are functions that belong to objects.

creating class

```
class student :
    def __init__(self, full name) :
        self.name = full_name

    def hello (self) :
        Print("hello ", self. name)

# creating object

    s1 = student ("Karan")
    s1. hello ()
```

- Static methods :

"Methods that don't use the self Parameter (work at class level :)

```
class student :
    @staticmethod # decorator
    def college() :
        Print ("ABC college")
```

- Important :

- Abstraction :

Hiding the implementation details of a class & only showing the essential features to the user.

- Encapsulation :

Wrapping data & functions into a single unit (object).

- del keyword :

" Used to delete Object Properties or object itself.

```
del s1.name
del s
```

- Inheritance :

When one class (child/ derived) derives the Properties & Methods of another class (Parent / base).

```
class car :
  - -

class Toyotacar (car) :
  - - -
```

- Inheritance type :
  - single Inheritance
  - Multi-level inheritance
  - Multiple inheritance

- Polymorphism : (operator overloading)

" when the same operator is allowed to have different meaning according to the context. "

operators & Dunder functions :

- a+b # addition · a--add--(b)
- a-b # sub · a--sub--(b)
- a*b # multiple · a--mul----(b)
- a/b # division a·--truediv---(b)
- a%b # mod a·--mod--(b)