18-01-2026

Agenda: Python - II

→ string properties
→ operators - II
→ control flow
→ control flow - II (advance)
→ Non primitive data type.

String properties

type() ← function

"Data Science" ⟶ <class 'str'>

↓ properties

↓ methods

class

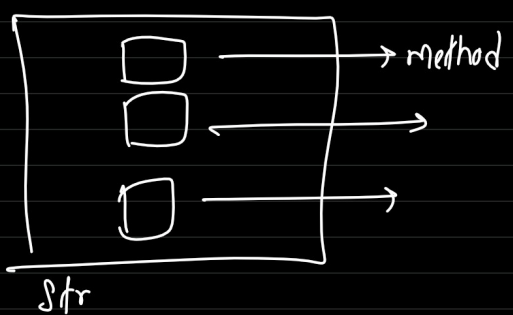
↑

House → door
→ window
→ room
→ floor

glass()→

print()→
id()→
type()→
int() float() str()
} function

str → strip()
→ lstrip()
→ rstrip()



car → call → Drive
→ open door

House → call → ~~Drive~~
→ live
→ room

door ( ) → function

methods → property of something is called a method

function → does some job.

datatype

string → strip()
→ lstrip()
→ rstrip()

int → ~~strip()~~

→ we use . to call its property.

function → not related to datatype
→ does something
→ cannot be used with "."

To remove white space from string we use below methods:

→ lstrip() → removes space from left side
→ rstrip() → removes space from right side
→ strip() → lstrip() + rstrip()

name = " .....Data..... "

name = name.lstrip().rstrip()

" Data ... " .rstrip() → "Data"

name = name.lstrip().rstrip()

print(9) → 12

print(12) →

Developer → google term → 'Data', 'data', 'daTa', '.data'

→ Age '27', '28', '40', '50' → int('27') → 27

'27' →

int('27'.strip())

Operators

Arithmetic operators

Relational operator / comparison operator

Relational operators: → relation of two values & we use operator to check if it is true or not.
True False

12 > 3 greater than

12 == 12 equal

3 < 12 smaller than

greater than or equal to :

12 >= 3

smaller than or equal to :

3 <= 12

not equal to :

3 != 12

12 > 3 → True

12 < 3 → False

12 >= 3 → True

12 <= 3 → False

12 != 12 →

↓

opposit of ==

12 == 12 → True

↓

False

3 == 3 → True

3 >= 3 → True

"Data" > "Data"

"Data" == "Data"

Assignment operator :

$a = 2$

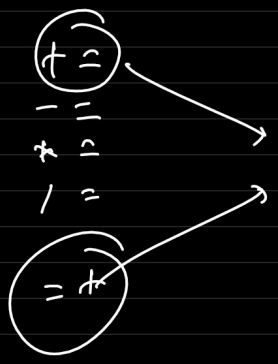
$a \mathrel{+}= 2 \longrightarrow a = a + 2$

$a \mathrel{-}= 2 \Rightarrow a = a - 2$

$a \mathrel{/}= 2 \Rightarrow a = a / 2$

$a \mathrel{*}= 2 \Rightarrow a = a * 2$

Monad



Logical operator

Truth Table :

operator	A	B	output
And	0	0	0
	1	1	1
	0	1	0
	1	0	0

(1) (1)

And → True, when both are True else false (0).

operator	A	B	output
OR	0	0	0
	1	1	1
	0	1	1
	1	0	1

OR → if any one is True → True
else false

operator	A		output
NOT	0		1
	1		0

Not →
True → false
False → True

operator	A	B	output
XOR	0	0	0
	1	1	0
	0	1	1
	1	0	1

XOR → only true when only one of them is true

Bitwise operators :— &, |, ^

2^2 2^1 2^0
4 2 1 → 7 (1-7)

1 →
2 →
3 →
4 →
7 →

0 0 1 →
0 1 0
0 1 1 →
1 0 0
1 1 1

```
  0 1 1   3
  0 0 1   1
& ------
  0 0 1 → 1
```

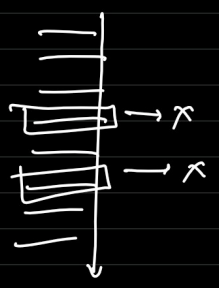
```
  0 1 1   3
  0 0 1   1
| ------
  0 1 1 → 3
```

```
  0 1 1   3
~ 1 0 0 → 4
  ↓
 NOT
```

^ → XOR

Control Flow :-

Control the flow of the program

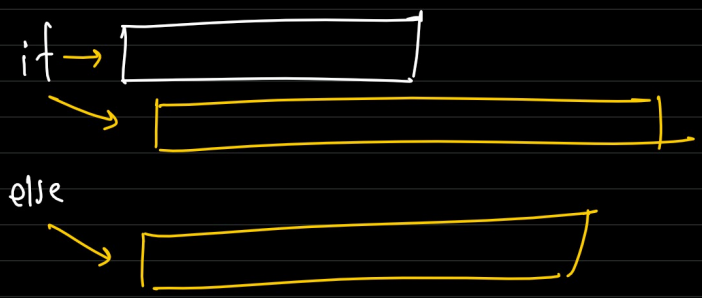


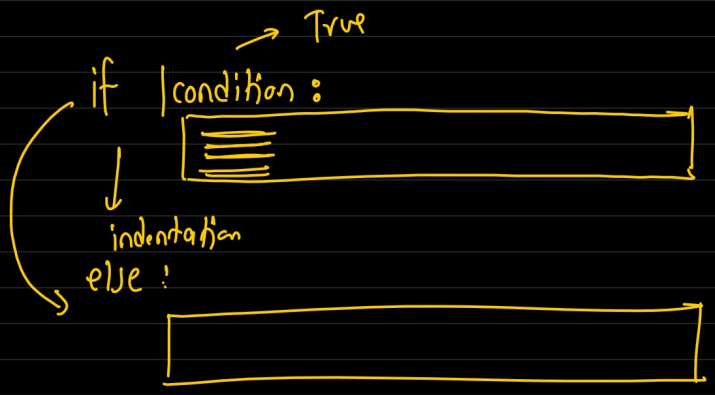
→ used to execute/run blocks of code on specific condition.

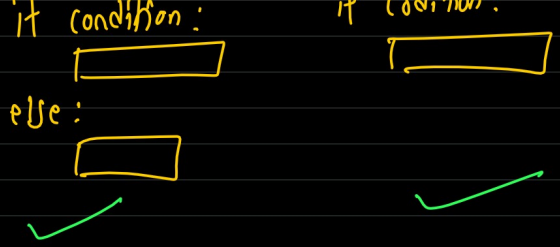
Task 1 : apple : 4

if → whether 4 apples or more are present in the store
we will buy apple

then →
we will not buy apply







1 Tab = 4 spaces

Task : application/tool ⟶ connect to a wifi

①

```
if wifi_avail :
    connect
```

②

```
if wifi_avail :
    connect
else :
    tell user, no wifi available
```

Control flow – II

```
if cond :
    do something
else :
    do something
```

```
if cond :
    do something
```

```
if cond :
    do something
elif cond :
    do something
    .
    .
    .
elif cond :
    do something
else :
    do something
```

```
elif cond :
    do something
```

if cond : ⟶ starts control flow block

if cond : ⟶ start control flow block

if cond : ⟶ start

elif cond :

⟶ end

if condition ⟶ start

primary data types of python → primitive data type.

→ int, float, str, bool

non-primitive data types:

→ list, dict, tuple, set

list = [1, 2, 3, 4, 5] → values are comma seperated

- list → variable name
- [→ square bracket
-] → closing square brack

array → [1, 2, 3, 4, 5] → int
(not python) ["Hello", "world"] → str

list → [1, 2, 3, 3.0, 4.0, "Manali"]

people_register:

↓

building entry registry

- name : ①
- phone : ②
- address : ③
- in : ④
- out : ⑤
- reason : ⑥
- flat : ⑦

→ 1 person

person_1 → [①, ②, ③, ④, ⑤, ⑥, ⑦]

register = [ ]

register.append (person_1)

[1, 2, 3, 4]

[ ]

[1, 4, 5, 8]

[[1, 2, 3, 4]]

[[1,2,3,4], [1,4,5,8]]