Welcome Back Everyone

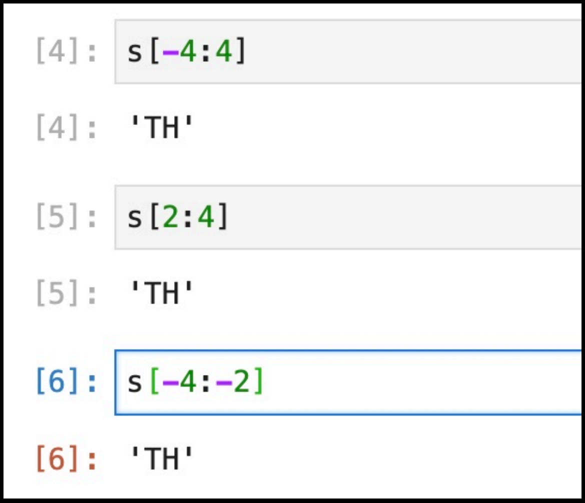
Agenda

→ Slicing ⇒ 2 concepts left

→ Operators
  - → Relational operator
  - → Logical
  - → assignment
  - → Identity
  - → Membership

Slicing

1. When +ve & -ve indexes are used together, just convert to either +ve fully or -ve fully or just set start & end based on provided index



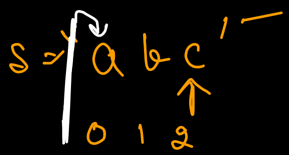
```
[4]: s[-4:4]
[4]: 'TH'

[5]: s[2:4]
[5]: 'TH'

[6]: s[-4:-2]
[6]: 'TH'
```

2. When step is -1

default start = len(string) -1

default end = Start -1

s[::-1] ⇒ cba # reverse of string



start = 2

end = -1 or 1 step before 0

Relational Operator In String

>=
<=
==
!=

>
<

== [checking equal]

!= [not equal to]

★ Always remember that string comparision is case sensitive

'A' ≠ 'a'

⇒ greater than or smaller than

abc x

Lexiographic Comparision

ord ( )

ord('a')= 97
ord('A') = 65
ord('0') = 48

str1 = 'a b c'

str = 'x y z'

1. We take first character of each string

a x

2. Compare their ASCII / Unicode value

97 120

3. Whichever ascii value is greater that string is greater.

4. If equal, keep on repeating steps.

	string	first char	ascii
str1	a b c	a	97
str2	x	x	120

as $120 > 97$

$\Rightarrow$ str2 > str1

$\Rightarrow$ 'x' > 'abc'

why not b or c ?

⇒ Because we move char by char

why not sum 'abc' ASCII ?

⇒ Because we do lexiographic comparision

---

abc

ade

a	a
97	97
b	d
98 <	100

abc < ade

str & int (or other datatypes)

cannot be compared with

> <

but == & != works

We can say apple is not equal or equal
To orange
but comparision requires a criteria
to be specified (size , color,
task)

Nesting of relational ~~operators~~

Chain relational operator

$1 < 2 < 3$

$1 < 2$ and $2 < 3$

It evaluates each expression individually & return true if all true else return false

== and !=

== compares the operands for equality

It checks

1] Type

2] Value

& return true if both equal

else False

~~exce~~

!= compares Type & value and

returns False if equal

true if not equal

exception

True == 1 ✓ [it show true as 1]

False == 0 ✓ False as 0]

$\underset{\text{float}}{\underline{3.0}} == \underset{\text{int}}{3}$ ✓ [______]

Logical Operators

logical operators, they help to combine 2 or more conditions and perform the total operation

and	True when all are true
or	True when either is true
not	(changes to True when false else False) or negate your result

Till now we have composed boolean data types only.

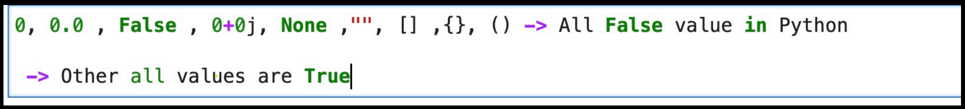
True, False, 3 > 4 , 5 < 3

Logical operator on non bool type

eg. "mayank" ~~and 10~~

3 and 4

None, 0.0 , 0 , '' , False
-0 +0
0+0j 0-0j
} ⇒ False



```
0, 0.0 , False , 0+0j, None ,"", [] ,{}, () -> All False value in Python

 -> Other all values are True
```

return value of logical & and logical OR for non boolean data type is never True or False

eg. $\underbrace{3>4}_{bool}$ and $\underbrace{'a'>'A'}_{bool}$ ⇒ bool

3 and 4 ⇒ ✗ bool

And operator

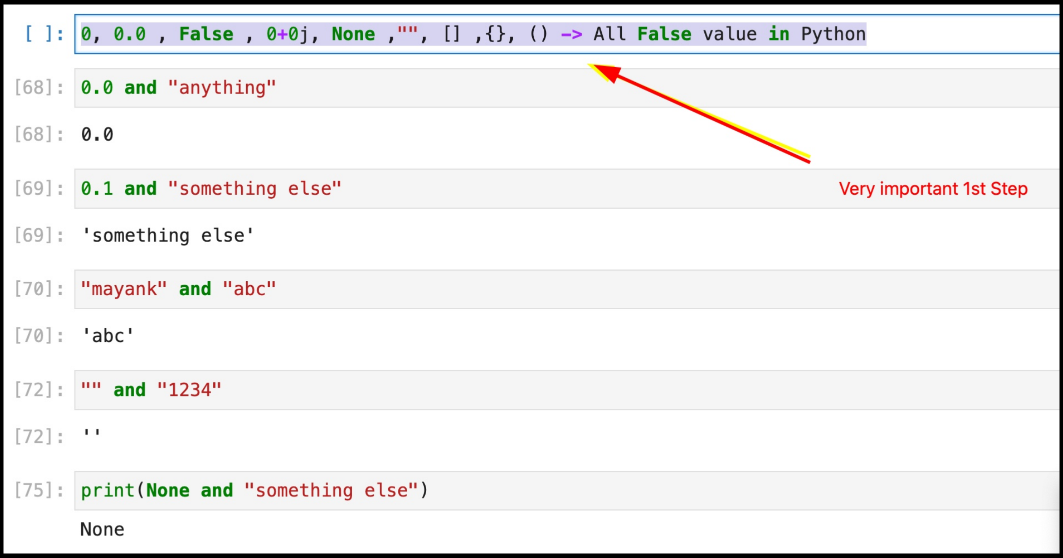
If 1st value is False
output 1st value
else
output 2nd value

eg.

$\underline{0}$ and 5 ⇒ 0
F

$\underline{5}$ and 0 ⇒ 0

"mayank" and "abc" ⇒ abc



```
[ ]: 0, 0.0 , False , 0+0j, None ,"", [] ,{}, () -> All False value in Python

[68]: 0.0 and "anything"
[68]: 0.0

[69]: 0.1 and "something else"
[69]: 'something else'

[70]: "mayank" and "abc"
[70]: 'abc'

[72]: "" and "1234"
[72]: ''

[75]: print(None and "something else")
      None
```

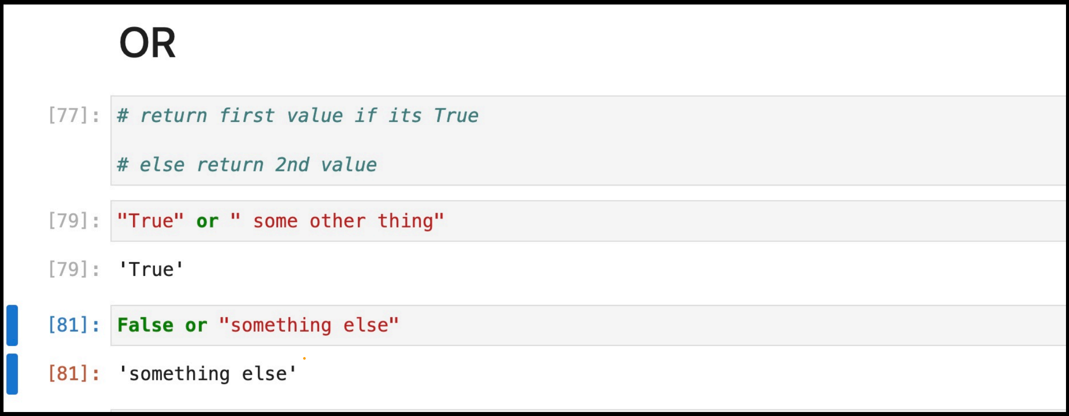
OR operator   If first value is True
return 1st value

else
return 2nd value

Eg

$\underline{5}$ or 0 ⇒ 5

0.0 or "abc" ⇒ abc



OR

```
[77]: # return first value if its True

      # else return 2nd value
```

```
[79]: "True" or " some other thing"

[79]: 'True'
```

```
[81]: False or "something else"

[81]: 'something else'
```

not operator on non-boolean types

If your operand is False (any form)

it returns True

else if operand is True

return False

Assignment Operator (=)

$a = 5$
$b = 6$

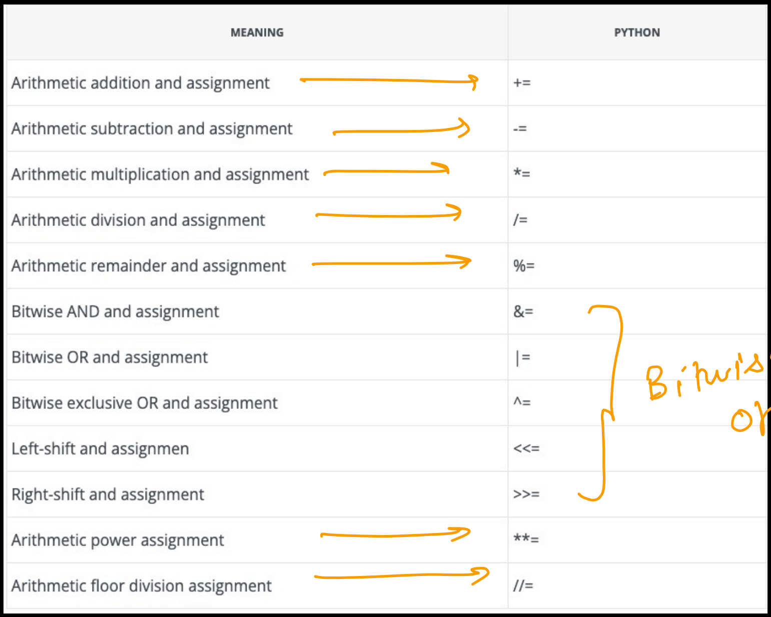
$a, b, c = 5, 6, 7$

$a = b = c = 10$

Compound assignment Operator

Python allows one to combine assignment operator with other arithmatic and bitwise operator.

$+=$ $\quad x += 5 \Rightarrow x = x + 5$



MEANING	PYTHON
Arithmetic addition and assignment	+=
Arithmetic subtraction and assignment	-=
Arithmetic multiplication and assignment	*=
Arithmetic division and assignment	/=
Arithmetic remainder and assignment	%=
Bitwise AND and assignment	&=
Bitwise OR and assignment	|=
Bitwise exclusive OR and assignment	^=
Left-shift and assignmen	<<=
Right-shift and assignment	>>=
Arithmetic power assignment	**=
Arithmetic floor division assignment	//=

Bitwise operation