எங்கள் வாழ்வும் எங்கள் வளமும்
மங்காத தமிழ் என்று சங்கே முழங்கு ... *புரட்சிக்கவி*

NOTICE

➢ We support open-source products to spread Technology to the mass.

➢ This is completely a FREE training course to provide introduction to Python language

➢ All materials / contents / images/ examples and logo used in this document are owned by the respective companies / websites. We use those contents for FREE teaching purposes only.

➢ We take utmost care to provide credits when ever we use materials from external source/s. If we missed to acknowledge any content that we had used here, please feel free to inform us at info@DataScienceInTamil.com.

- All the programing examples in this document are for teaching purposes only.

Thanks to all the open-source community and to the below websites from where we take references / content /code example. definitions, please use these websites for further reading:

Python Notes For Professionals.pdf – this is the book we follow
https://docs.python.org
https://www.w3schools.com/python/python_conditions.asp

WHAT TO LEARN Today?

Python Conditions and If statements

1. if
2. if else
3. if elif
4. if elif else

5. Truthy Values / Falsey values
6. Boolean Logic Expressions
7. Conditionals and 'and' logical operator
8. Conditionals and 'or' logical operator
9. Lazy evaluation
10. How NOT to call a function
11. How 'None' works
12. Testing for multiple conditions using 'and' operator
13. Pythonic way of using 'None'

Conditionals

Conditional expressions, involving keywords such as if, elif, and else, provide Python programs with the ability to perform different actions depending on a boolean condition: True or False. This section covers the use of Python conditionals, boolean logic

Python Conditions and If statements

Python supports the usual logical conditions from mathematics:

- Equals: a == b
- Not Equals: a != b
- Less than: a < b
- Less than or equal to: a <= b
- Greater than: a > b
- Greater than or equal to: a >= b

These conditions can be used in several ways, most commonly in "if statements" and loops.

An "if statement" is written by using the if keyword.

If

```python
a = 33
b = 200
if b > a:
  print("b is greater than a")
```

```
output
b is greater than a
```

If ……else

```
a = 33
b = 20
if b > a:
  print("b is greater than a")
else:
    print("a is greater")
```

----

if, elif, and else

In Python you can define a **series of conditionals** using if for the first one, elif for the rest, up until the final (optional) else for **anything not caught by the other conditionals.**

```
number = 5
```

```
if number > 2:
    print("Number is bigger than 2")
elif number < 2:
    print("number is smaller than 2")
else:
    print("Number is 2")

output
Number is bigger than 2
---------
```

if, elif,elif ….. and else

```
number = 5

if number < 3:
    print("Number is bigger than 2")
elif number < 4:
    print("number is smaller than 2")
elif number > 4:
  print("Number is greater than 4")

else:
    print("Number is 5")
```

```
output
Number is greater than 4
number = 5
```

================

If condition can be given inside else

```
number = 5

if number < 2:
    print("Number is bigger than 2")
elif number < 3:
    print("number is smaller than 2")
else:
    if number > 3.5:
        print("Number greater")
```

```
output
Number greater
```

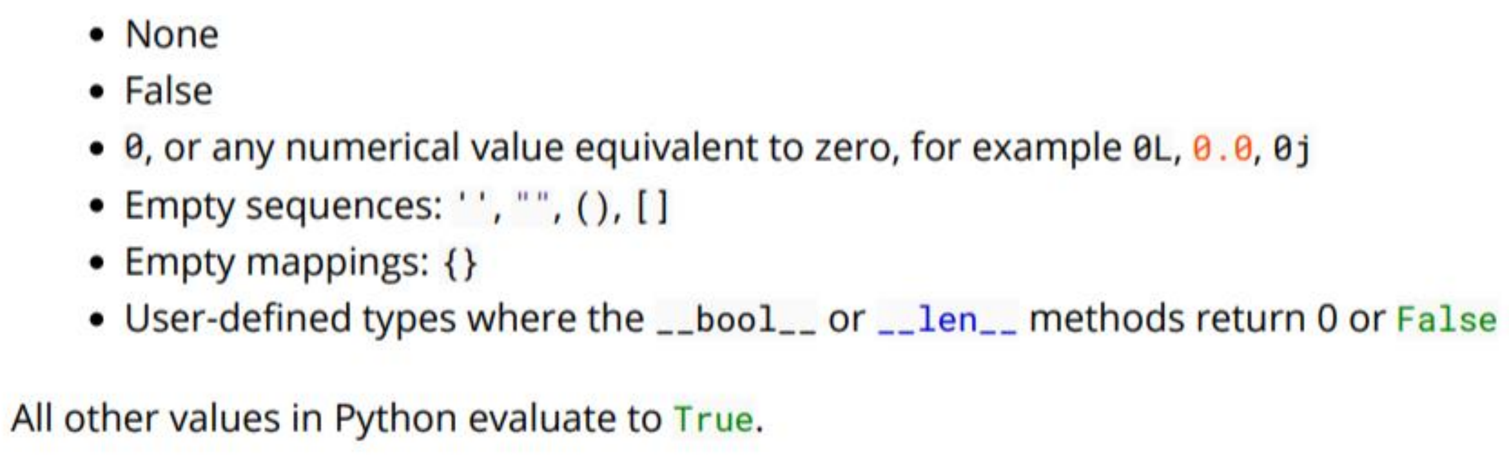
==========

Using else if (in JAVA) instead of elif will trigger a syntax error and is not allowed.

==========

Truth Values

The following values are considered **falsey**, in that they evaluate to False when applied to a boolean operator.



- None
- False
- 0, or any numerical value equivalent to zero, for example 0L, 0.0, 0j
- Empty sequences: '', "", (), []
- Empty mappings: {}
- User-defined types where the __bool__ or __len__ methods return 0 or False

All other values in Python evaluate to True.

```
==========
if  True:
    print("Yes")
if  False:
    print("it does not print and execution does not come to this line")
```

```
output
Yes
=======
if  None:
    print("it does not print and execution does not come to this
line")
=========
if  1:
    print("Yes")
if  0:
    print("it does not print and execution does not come to this
line")
output
Yes
===========
if  [3,4,5]:
    print("Yes")
if  []:
    print("it does not print and execution does not come to this
line")
---------
if  (5,6):
    print("Yes")
if  ():
    print("it does not print and execution does not come to this line")
-----------

if  {7,8,}:
    print("Yes")
if  {}:
    print("it does not print and execution does not come to this line")
=========
if 'abc':
   print("Yes")
if '':
   print("it does not print and execution does not come to this line")

if " ":
   print("It will print, since the string  has a space")

--------


if  "Data Science":
    print("Yes")

if  "":
    print("it does not print and execution does not come to this line")
----------

if  5-5:
    print("it does not print and execution does not come to this line")

===========
if  0.0:
    print("it does not print and execution does not come to this line")

if  1j:
    print("Yes")

if  0j:
    print("it does not print and execution does not come to this line")
==============
```

Note: A common mistake is to simply check for the Falseness of an operation which returns different Falsey values where the difference matters. For example, using if foo() rather than the more explicit if foo() is None

```
def foo():
    return " some string"

if foo():
    print("print somthing")
```

output

print something

=======

```
def foo():
    return ""

if foo():
    print("print somthing")
```

```
def foo():
    return None

if foo():
    print("this line does not execute")#None becomes Falsey
----------
def foo():
    return True

if foo():
    print("this line will execute")
```

```
------------
def foo():
    return False
if foo():
    print("this line does not execute")
```

```
===========
```

Boolean Logic Expressions

Boolean logic expressions, in addition to evaluating to True or False, return the value that was interpreted as True or False. It is Pythonic way to represent logic that might otherwise require an if-else test.

```
def foo():
    return None

if foo():#they calling the foo()in if statement
    print("this line does not execute")
----------
```

```
def foo():
    return True

if foo():
    print("this line will execute")
-------------
def foo():
    return False
if foo():
    print("this line does not execute")
```

And operator

The and operator evaluates all expressions and **returns the last expression if all expressions evaluate to True**. Otherwise it returns the first value that evaluates to False:

```
print(1 and 2 and 3 and 4)  # 4
print(1 and 2 and 3)  # 3
print(1 and 2 and 0 and 4)  # 0
print(1 and 2 and [] and 4)  # []
print(1 and 2 and -5 and 4 and 5 and { } ) # {} , note, -5 is value

print(1 and {} and [] and 4) #{}// it returns the first occurance of the False
=======
print(2 and "Data Science")#Data Science
print(2 and "Data Science" and "AI") # AI
print(" " and "Py" and "")
print(" " and "Py" and " ")

print("" and "Py")#"" // we will not be able to view “”
-----
a = "" and "Py"
print(repr(a))# “”
```

The repr() function **returns a printable representation of the given object.**

```
a = " " and "Linda" and " "
print(a)
print(repr(a))
```

output

```

' '
```

```
a = " " and "Linda" and " " and None and []
print(a)
print(repr(a))

========
```

Or operator

The or operator evaluates the expressions left to right and returns the **first value that evaluates to True** or the last value (if none are True).

```
print(1 or 2)#1
print(0 or 2)  #2
print(None or 5)#5
print(False or 6 )#6
print(True or False)#True
print(False or True)#True
print([] or True)#True
print([] or 0 or {})#{} # takes the last False
print([] or "ABC" or {22})#ABC (Takes first True
value)
```

```
print(None or None)# it takes the last None
print(0 or {} or [])#[] ..it takes the last false

print(0 or (10-10) or "BB") #BB

print( [] or 0 or {})#
print([] or "ABC" or {22})#
print(False or False or True)
print(None or 0j or True)
print(None or {} or 0j)



--------

print(("Tamil" or "English") and ("England" or "Australia"))
print("Tamil" and "England")



------

print("Sir" and {False} and 56)
print("Sir" and False and 56)

print("Sir" and {False} and 56)# Set . this code is equal to ("Sir" and {0} and 56)# Set
print("Sir" and [False] and 56)#List
print("Sir" and (False) and 56)#Tuple (Warning: Tuple is immutable

print(True + True)
print(False + True)
print(False + False)

----------
```

Lazy evaluation

When you use this approach, remember that the **evaluation is lazy. Expressions that are not required to be evaluated to determine the result are not evaluated**. For example:

```
def print_me():
    print("I am here")

0 and print_me()
output
```

Note: it does not call the print_me(). Because the expression starts with 0(zero) that stands for False

What circumstances can we use?

In the above example, **print_me** is never executed because Python can determine the entire expression is False when it encounters the 0 (False). Keep this in mind if print_me needs to execute to serve your program logic.

```
def print_me():
    print("I am here")

1 or print_me() # the fn will not be called
print_me()  or 1# the fn will be called
```

Another example to NOT call the fn. we converted the print_me() as string. Now the last value 5 will be printed

```
def print_me():
    print("I am here")

print(1 and 2 and "print_me()" and 5)
```

>>5

=======

See some more ex(Melcose)

```
def print_me():
    print("I am here")
    return False

print(1 and 2 and print_me() and 10)
```

output

```
I am here
False
```

Note: the fn is called / result is printed. And the return value also printed. Since the return value is false, it gives first False value

----

```
def print_me():
    print("I am here")
    return True

print(1 and 2 and print_me() and 10)
```

output

```
I am here
10
```

Note: the fn is called and printed the result. Since it is True, it goes and check the next value (ie 10). Now all values become True.Therefore, it gives the last value in the expression .

How None works

```
def print_me():
    print("I am here")

print(1 and 2 and print_me() and 10)
output
I am here
None
```

Note: here the fn is called and printed. This fn does not return anything. So by default the return type is None. The None is retured to the expreission. Since is None stands for False, the expression shows the false value, here the false value is None. So we get None

============

Testing for multiple conditions

A common mistake when checking for multiple conditions is to apply the logic incorrectly.

This example is trying to check if two variables are each greater than 2. The statement is evaluated as - if (a) and (b > 2). This produces an unexpected result because bool(a) evaluates as True when a is not zero.

```
a = 1
b = 6

if a and b > 2:
    print("Yes")
else:
    print("No")
```

output

Yes

Note: but this output is NOT right.

==============

Each variable needs to be compared separately. See below

```
a = 1
b = 6

if a > 2 and b > 2:
    print("Yes")
else:
    print("No")
```

output

No #this is the right output

=========

Another, similar, mistake is made when checking if a variable is one of multiple values. The statement in this example is evaluated as - if (a == 3) or (4) or (6). This produces an unexpected result because bool(4) and bool(6) each evaluate to True

```
a = 1
if a ==3 or 4 or 6:
    print("Yes")
else:
    print("No")
Output
Yes
```

Note: but this output is not right.

========

Again each comparison must be made separately

```
a = 1
if a == 3 or a == 4  or a == 6:
    print("Yes")
else:
    print("No")
```

```
output
No
```

Note

The above pgm is written as below

```
a = 1
if False or False or False:
   print("Yes")
else:
   print("No")
```

```
output
No
```

=======

Using the in operator is the canonical way to write this. (using membership operator)

```
a = 1
if a in (3,4,6):
    print("Yes")
else:
    print("No")
>>No
```

================

Else statement

```
if True:
    print("it prints")
else:
    print("it does not print")

if False:
    print("it prints")
else:
    print("it DOES print")
===========
```

Note: Try with ‘not’ (not True / not False). We get the opposite result of the above

Testing if an object is None and assigning it

You'll often want to assign something to an object if it is None, indicating it has not been assigned. We'll use aDate.

The simplest way to do this is to use the **is** None test.

```
marks = None
if marks  is None:
    marks = 35
print(marks)
# --------------------
```

```
import datetime
aDate = None
if aDate is None:
    aDate =datetime.datetime.now()
print(aDate)
```

(Note that it is more Pythonic to say is None instead of == None.)

But this can be optimized slightly by exploiting the notion that not None will evaluate to True in a boolean expression. The following code is equivalent:

```
import datetime
aDate = None
if   not aDate is None:
    aDate =datetime.datetime.now()
    print(aDate)
no output
-----------
```

```
import datetime
aDate = None
if  not  aDate is  not None:
    aDate =datetime.datetime.now()
    print(aDate)
```

There is an output

```
import datetime
aDate = None # None means False
if   not aDate: #  here we convert the None to True by
adding ‘not’
    aDate =datetime.datetime.now()
    print(aDate)
```

output

```
2020-10-31 00:36:23.329926
```

========

But there is a more Pythonic way. The following code is also equivalent:

```
import datetime
aDate = None
aDate = aDate or datetime.date.today()
print(aDate)
```

output

```
2020-10-31
```

========

(the above) This does **a Short Circuit evaluation**. If aDate is initialized and is not None, then it gets assigned to itself with no net effect. If it is None, then the datetime.date.today() gets assigned to aDate

```
import datetime
aDate = not None
aDate = aDate or datetime.date.today()
print(aDate)
```

output

```
True
```

========