Boolean Operators

```
a =10
b =100

print(a==10 and b == 10)
print(a == 10 or b == 1009)
```

output

```
False

True

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

How… “or” works in bool expression

MEL Note: when the first argument of the function evaluates to `true`, the

O
R

overall value must be `true`. See the code here

```
a = 10
b = 20
if a == 10 or b == 200:
 print ("condition is satisfied")
```

output

It is true, and the value a is 10

Note: it checks only the first expression (a ==10). If it is true, it ASSUMES the other expression (b == 200) also true. But that is NOT

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

MEL note: second argument is executed or evaluated only if the first argument does not suffice to determine the value of the expression – see the code below

```
a = 10
b = 20
if a == “DD” or b == 20:
 print ("condition is satisfied")
```

output

It is true, and the value a is 10

Note: now the first expression (a==”DD) is evalatued first and it does not have enough information to determine true or false, so it goes second expression (b==20) and try to evaluate and return true or false
=======

How…”and” works in bool expression

Note: when the first argument of the function evaluates to `false`, the `AND` overall expression must be `false`; - see below

```
a = 10
b = 20
if a == "DD" and b == 20:
 print ("It is not true, and the value a is 10")
```

output

No output

Note: now the first expression (a==”DD) is evalatued first and it it is false/ not suffice information , so it DOES NOT go second expression (b==20) even though the second expression is right

=======

```
a = 10
b = 20
if a == 10 and b == 20:
 print ("It is true, and the value a is 10")
```

output

```
It is true, and the value a is 10
```

Note: now the first expression (a==10) becomes true. So it goes to the second exrpression and check is if it is true. If true it gives result, else it does not gives the outout

======

How to use ‘or’ and ‘and” when calling functions

Example for "or"

```
def true_func():
 print("true_function")
 return True

def false_func():
 print("false_function")
 return False

a = true_func() or false_func()
print(a)
```

output

```
true_function

True
```

Note: since we call true_func() **or** false_func(), once the true_func() becomes returns True, it ASSUEMS the second function false_func() also true (But that is not right, because the the second function false_func() is false) ---------

```
def true_func():
 print("true_function")
 return True

def false_func():
 print("false_function")
 return False


b = false_func() or true_func()
print(b)
```

output

```
false_function
true_function
True
```

Note: now we call false_func() **or** true_func(). Since it is false(), it becomes False, so it goes and check the second function

=========

Example for "and"

```
def true_func():
 print("true_function")
 return True

def false_func():
 print("false_function")
 return False

b = true_func() and false_func()
print(b)
output
true_function
false_function
False
=======
def true_func():
 print("true_function")
 return True

def false_func():
 print("false_function")
 return False
```

```
b = false_func() and true_func()
print(b)
```

output

```
false_function
False
============
From Data Science
From Latha
```

and

Evaluates from left to right.
Stops execution when first encounter a false value and return that value
Executes till last if all are true and return last value

or

Evaluates from left to right.
Stops execution when first encounter a true value and return that value
Executes till last if all are false and return false

following are considered false

0
none
false
{}
()
[]

From Raja Gopal
Note: It just checks if the variable is empty or not ----------
From Sankar

Logical/ Boolean Operator Precedence

a. The following is the precedence order
   1. NOT
   2. AND
   3. OR

b. Left to right will be followed for same precedence **Statement:**

```
print(True or False and not False and True)
```

Output:

```
True
```

Explanation:

```
print(True or False and not False and True) # NOT

print(True or False and True and True) # AND / Left to

Right print(True or False and True) # AND

print(True or False)# OR

print(True)
```

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

Section 11.4: and

Evaluates to the second argument if and only if both of the arguments are truthy. Otherwise evaluates to the first falsey argument

```
x = True
y =True
print(x and y)
```

# True

```
x = 12
y = 10
print( y and x )
print(x and y)
```

o/p:

12

10

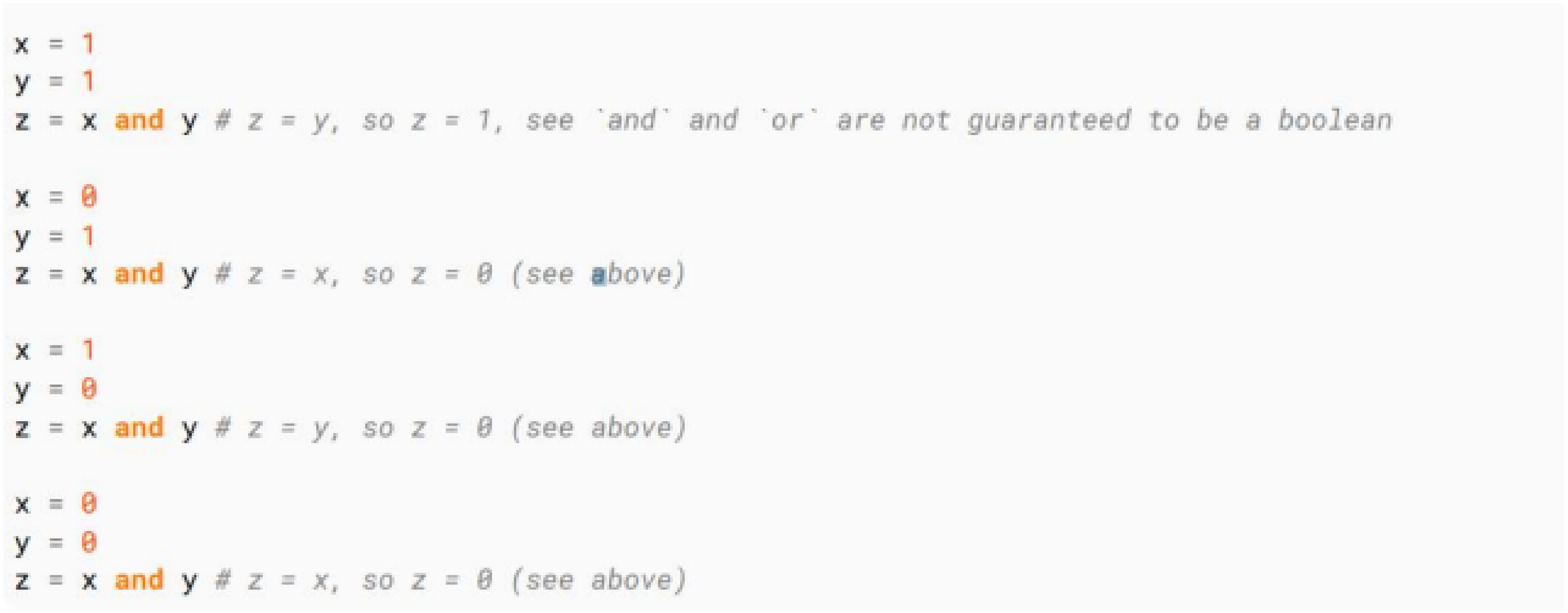
```
x = 12
y = None
print( y and x
) print(x and
y)
```

```
o/p:

None

None

-----------

x = 12
y = []
print( y and x
) print(x and
y) o/p:

 []

 []

 ------
x = 12
y = False
print( y and x
) print(x and
y) o/p:

False

False

-------

x = 12

y = ''


print((y) and (x))

print((x) and (y))

no output
```

repr(), it is a printable representation of a given object. If the object is empty string, then repr(), treats the empty has value

```
x = 12
y = ''

print(repr(y) and repr(x))
print(repr(x) and repr(y))

o/p:
12
''
-----


x = True
y =False
print(x and y)
# False

# both expression will be evaluated,because of x is True
x = False
y =True
print(x and y)
# False
# only x expression will be evaluated,because of x is False

x = False
y =False
print(x and y)
# False
```

```
# only x expression will be evaluated,because of x is

False ========
```



```
x = 1
y = 1
z = x and y # z = y, so z = 1, see `and` and `or` are not guaranteed to be a boolean

x = 0
y = 1
z = x and y # z = x, so z = 0 (see above)

x = 1
y = 0
z = x and y # z = y, so z = 0 (see above)

x = 0
y = 0
z = x and y # z = x, so z = 0 (see above)
```

The 1's in the above example can be changed to any truthy value, and the 0's can be changed to any falsey value.

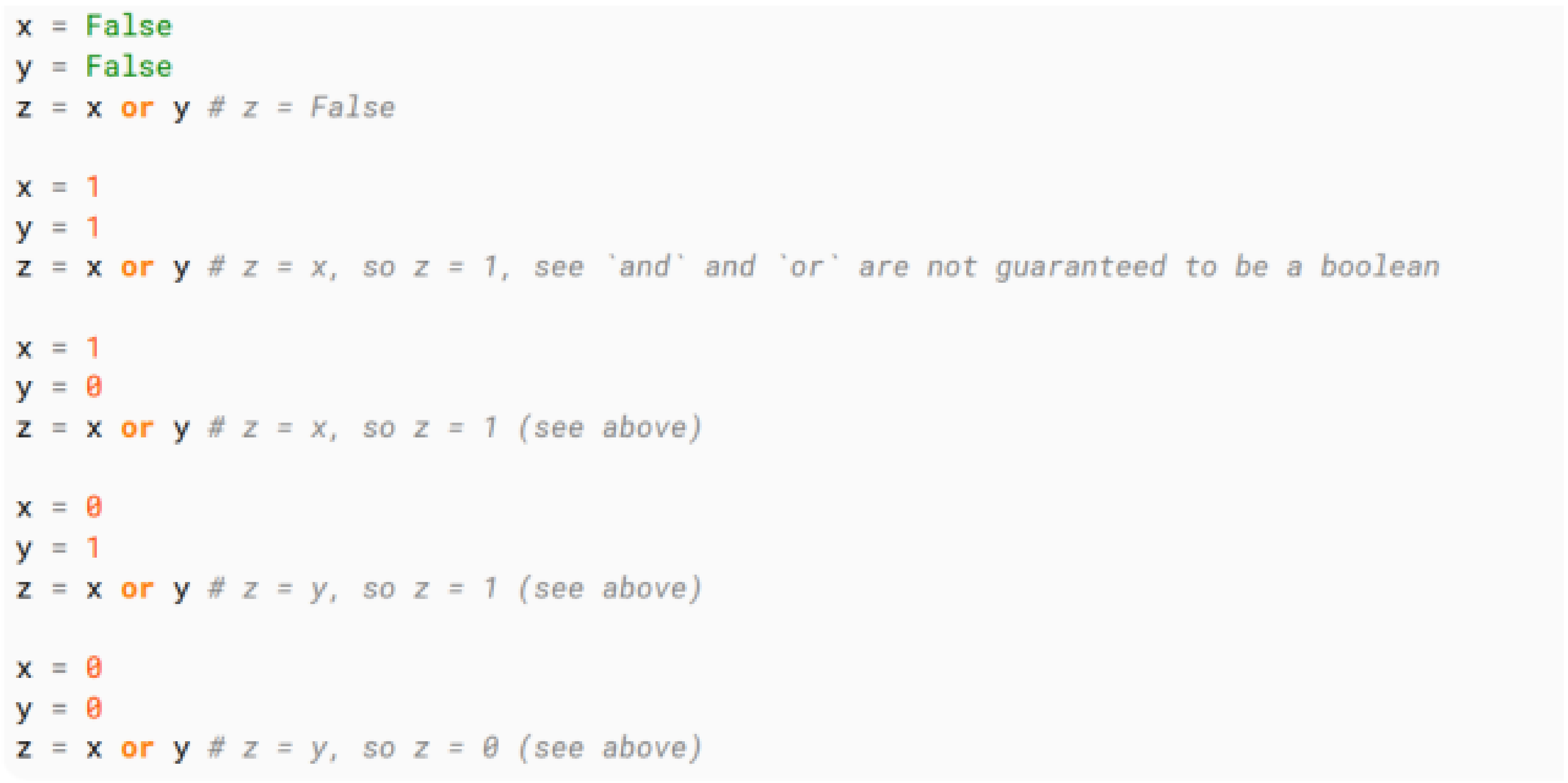
See the below example

```
from operator import truth

print(True+True)  #2
print(True == True)  #True
print(True)  # True
print(truth(2))  #True
print(truth(0))  #False
print(truth(''))#False
print(False - True) #-1
```

Note: True + True gives us 2. It does not mean 1 is True(Boolean)

Section 11.5: or

Evaluates to the first truthy argument if either one of the arguments is truthy. If both arguments are falsey, evaluates to the second argument



```python
x = False
y = False
z = x or y # z = False

x = 1
y = 1
z = x or y # z = x, so z = 1, see `and` and `or` are not guaranteed to be a boolean

x = 1
y = 0
z = x or y # z = x, so z = 1 (see above)

x = 0
y = 1
z = x or y # z = y, so z = 1 (see above)

x = 0
y = 0
z = x or y # z = y, so z = 0 (see above)
```

The 1's in the above example can be changed to any truthy value, and the 0's can be changed to any falsey value.

---

x = 12
y = ''

```
print(repr(y)              or
repr(x))      print(repr(x)
or repr(y)) o/p:
''

 12

------

x = 12
y = None

print((y) or (x))
print((x) or (y))

o/p:

 12

 12

----
x = 12
y = False
```

print((y) or (x)) print((x) or (y)) o/p:

12

12

```
x = 12
y = 20
```

print((y) or (x)) print((x) or (y)) o/p:

20

12

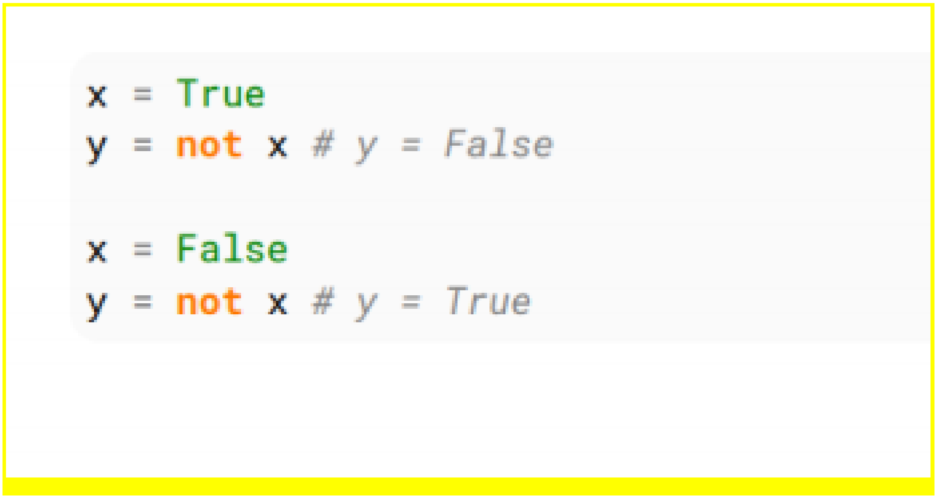
```
x = 12
y = []
```

```
print((y) or (x))
print((x) or (y))

# from operator import truth #
from operator import truth as t
# import operator as op
# print(truth(y))
# print(t(y))
```

o/p:

```
12

12

-----
```

Section 11.6: not



```
x = True
y = not x # y = False

x = False
y = not x # y = True
```

```
x = 12
y = []

print( x and not y)
```

o/p

True

Note: If we use ‘not’ in the Boolean operators (and / or), it gives either True or False

----

```
x = 12
```

```
y = None

print( x and not y)
```

o/p:

True

```
x = 12
y = False

print( x and not
```

```
y)
```
o/p:

True

```
x = 12
y = ''

print( x and not y)
```

o/p:

True

```
x = 12
y = ""
print( not x and y)
```

outout

False

```
x = 12
y = 30

print( x and not

y) o/p:

False

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