some\_dict **=** {}  
some\_dict[5.5] **= "Ruby"**print(some\_dict[5.5])  
some\_dict[5.0] **= "JavaScript"**print(some\_dict[5.0])  
some\_dict[5] **= "Python"**print(some\_dict[5])  
some\_dict[0] **=** int  
print(some\_dict[0])  
some\_dict[0.0] **=** float  
print(some\_dict[0.0])  
some\_dict[**False**] **=** bool  
print(some\_dict[**False**])  
print(some\_dict)  
  
output**:**Ruby  
JavaScript  
Python  
**<class 'int'>  
<class 'float'>  
<class 'bool'>**{5.5**: 'Ruby'**, 5.0**: 'Python'**, 0**: <class 'bool'>**}

print(some\_dict[5.0]) *# Python*print(some\_dict[0]) *# <class 'bool'>*

Python dictionaries check for equality and compare the hash value to determine if two keys are the same.

Immutable objects with same value always have the same hash in python.

0 == 0.0 == False

1 == 1.0 == True == 1+0

**Strings can be tricky sometimes**

a **= 'some\_strong'**print(id(a))  
print(id(**'some'+'\_'+'strong'**))  
  
output**:**2644234643632  
2644234643632

**Return finally**

**def some\_func**()**:  
 try:  
 return 'from\_try'  
 finally:  
 return 'from\_finally'**s **=** some\_func()  
print(s) *# from\_finally*

When a return, break or continue statement is executed in the try of try…finally statements, the finally will always executed.

The return value of a function is determined by the last return statement executed.

**Class**

**class Saisk:  
 pass**print(Saisk() **==** Saisk()) *# False  
# two different instancs can't be equal*print(Saisk() **is** Saisk()) *# False  
# identities are different*print(hash(Saisk()) **==** hash(Saisk())) *# True*print(id(Saisk()) **==** id(Saisk())) *# True*

**class Test**(object)**:  
 def \_\_init\_\_**(self)**:** print(**"I"**)  
 **def \_\_del\_\_**(self)**:** print(**"D"**)  
  
print(Test() **is** Test())  
*# I  
# I  
# D  
# D  
# False*print(id(Test()) **==** id(Test()))  
*# I  
# D  
# I  
# D  
# True*

**for**

some\_string **= 'saisk'**some\_dict **=** {}  
**for** i, some\_dict[i] **in** enumerate(some\_string)**:  
 pass**print(some\_dict) *# {0: 's', 1: 'a', 2: 'i', 3: 's', 4: 'k'}*

**for** i **in** range(4)**:** print(i)  
 i **=** 10  
  
output**:**0  
1  
2  
3

array **=** [1, 8, 15]  
g **=** (x **for** x **in** array **if** array.count(x) **>** 0)  
  
array **=** [1, 8, 15]  
**for** x **in** array**:** print(x)  
  
output**:**1  
8  
15

array **=** [1, 8, 15]  
g **=** (x **for** x **in** array **if** array.count(x) **>** 0)  
*# print(list(g)) # [1, 8, 15]*array **=** [2, 8, 22]  
print(list(g)) *# [8]*

Before runtime, array is re-assigned to the list [2, 8, 22], and since out of 1, 8 and 15, only count of 8 is greater than 0, the generator only yields 8.

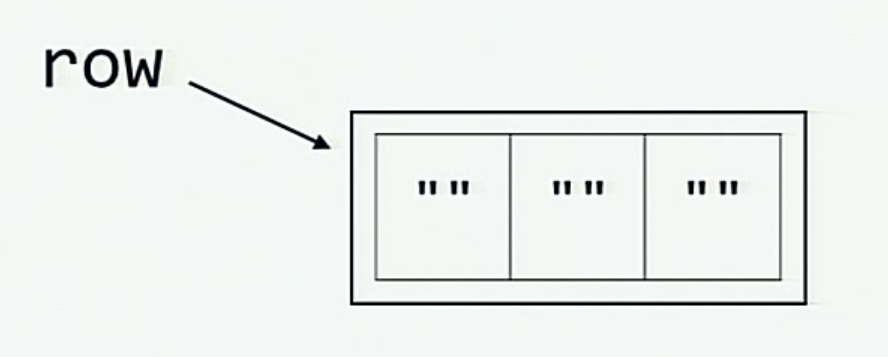
**is**

print(id(256))  
a **=** 256  
b **=** 256  
print(a **is** b)  
print(id(a))  
print(id(b))  
  
print(**"----------------------"**)  
print(**"for int it is True"**)  
print(id(257))  
a **=** 257  
b **=** 257  
print(a **is** b)  
print(id(a))  
print(id(b))  
print(type(a))  
  
print(**"-------------------------------"**)  
print(**"for bytes it is Flase"**)  
print(id(257))  
a **=** bytes(257)  
b **=** bytes(257)  
print(a **is** b)  
print(id(a))  
print(id(b))  
print(type(a))  
  
output**:**1971751920  
**True**1971751920  
1971751920  
**----------------------  
for** int it **is True**1975283789520  
**True**1975283789520  
1975283789520  
**<class 'int'>  
-------------------------------  
for** bytes it **is** Flase  
1975283789520  
**False**1975316519280  
1975320745464  
**<class 'bytes'>**

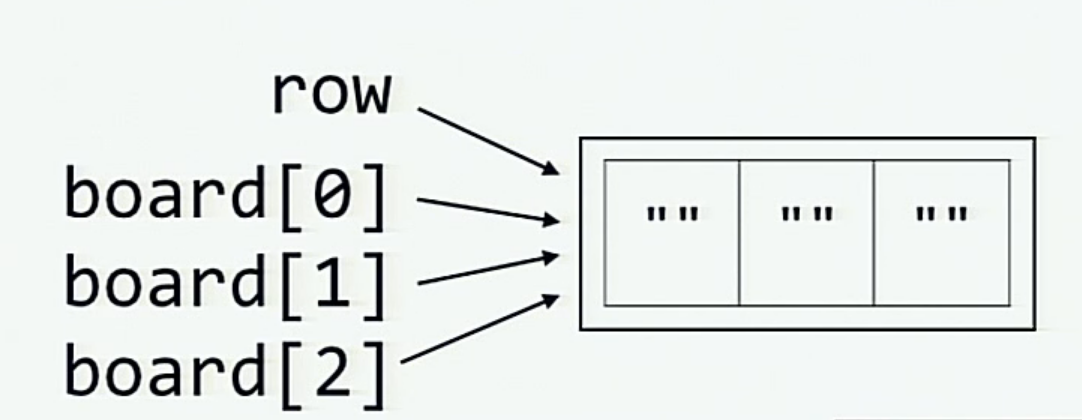
**A tic-tac-toe where x wins in the first attempt**

row **=** [**""**]**\***3  
board **=** [row]**\***3  
print(board)  
board[0][0] **= "x"**print(board)  
print(id(row))  
print(id(board[0]))  
print(id(board[1]))  
print(id(board[2]))  
print(**"----------------"**)  
print(id(board[0][0]))  
print(id(board[1][0]))  
print(id(board[2][0]))  
  
output**:**[[**''**, **''**, **''**], [**''**, **''**, **''**], [**''**, **''**, **''**]]  
[[**'x'**, **''**, **''**], [**'x'**, **''**, **''**], [**'x'**, **''**, **''**]]  
1631801550216  
1631801550216  
1631801550216  
1631801550216  
**----------------**1631762083208  
1631762083208  
1631762083208

When we initialize row variable,

[](https://github.com/satwikkansal/wtfpython/blob/master/images/tic-tac-toe/after_row_initialized.png)

When the board is initialized by multiplying the row, elements of board[0], board[1] and board[2] is referring to the same row.

[](https://github.com/satwikkansal/wtfpython/blob/master/images/tic-tac-toe/after_board_initialized.png)

We can avoid this by using the following

board **=** [[**""**]**\***3 **for** \_ **in** range(3)]  
print(board)  
board[0][0] **= "x"**print(board)  
  
output**:**[[**''**, **''**, **''**], [**''**, **''**, **''**], [**''**, **''**, **''**]]  
[[**'x'**, **''**, **''**], [**''**, **''**, **''**], [**''**, **''**, **''**]]

funcs **=** []  
results **=** []  
**for** x **in** range(7)**:  
 def some\_func**()**:  
 return** x  
 funcs.append(some\_func())  
 results.append(some\_func())  
  
print(funcs) *# [0, 1, 2, 3, 4, 5, 6]*print(resul ts) *# [0, 1, 2, 3, 4, 5, 6]*

**Half triple=quoted strings**

print(**"hello """**) *# hello*print(**'hi'''**) *# hi  
  
# we will get error  
# print("""hi")  
# print('''hello')*

**Counting number of booleans and integers in the given list**

mixed\_list **=** [**False**, 1.0, **"some\_string"**, 3, **True**, [], **False**]  
integers\_found\_so\_far **=** 0  
booleans\_found\_so\_far **=** 0  
  
**for** item **in** mixed\_list**:  
 if** isinstance(item, int)**:** print(item)  
 integers\_found\_so\_far **+=** 1  
 **elif** isinstance(item, bool)**:** print(**"item"**)  
 booleans\_found\_so\_far **+=** 1  
  
print(**"-------------------"**)  
print(**"booleans\_found:"**, booleans\_found\_so\_far)  
print(**"integers\_found:"**, integers\_found\_so\_far)  
  
output**:  
False**3  
**True  
False  
-------------------**booleans\_found**:** 0  
integers\_found**:** 4  
  
Explanation**:**Booleans are subclasses of int  
  
print(isinstance(**True**, int)) *# True*print(isinstance(**False**, int)) *# True*

**Printing the string by multiplying with the Boolean value**

some\_bool **= True**print(**"saisk"\***some\_bool) *# saisk*some\_bool **= False**print(**"saisk"\***some\_bool) *# nothing will display*

**From filled to None in one instruction**

some\_list **=** [1,2,3,4]  
some\_dict **=** {  
 **"key\_1":** 1,  
 **"key\_2":** 2,  
 **"key\_3":** 3  
}  
  
some\_list **=** some\_list.append(4)  
some\_dict **=** some\_dict.update({**"key\_4":** 4})  
  
print(some\_list) *# None*print(some\_dict) *# None*

Methods that modify the items of sequence/mapping objects like list.append, dict.update, list.sort, etc. modifies the objects and return None.

**for loop**

**for** x **in** range(7)**:  
 if** x **==** 6**:** print(x, **": for x inside loop"**)  
print(x, **": x in global"**)  
  
output**:**6 **: for** x inside loop  
6 **:** x **in global**

x**= -**1  
**for** x **in** range(7)**:  
 if** x **==** 6**:** print(x, **": for x inside loop"**)  
print(x, **": x in global"**)  
  
output**:**6 **: for** x inside loop  
6 **:** x **in global**

x**=** 1  
print([x **for** x **in** range(5)]) *# [0, 1, 2, 3, 4]*print(x, **": x in global"**) *# 1 : x in global*

**def some\_func**(*default\_arg***=**[])**:** *default\_arg*.append(**"some\_string"**)  
 **return** *default\_arg*print(some\_func())  
print(some\_func())  
print(some\_func())  
  
output**:**[**'some\_string'**]  
[**'some\_string'**, **'some\_string'**]  
[**'some\_string'**, **'some\_string'**, **'some\_string'**]

*# a = [1,2,3,4]  
# b = a  
# a = a + [5,6,7,8]  
#  
# print(a) # [1,2,3,4,5,6,7,8]  
# print(b) # [1,2,3,4]*print(**"---------------------"**)  
  
a **=** [1,2,3,4]  
b **=** a  
a **+=** [5,6,7,8]  
print(a) *# [1,2,3,4,5,6,7,8]*print(b) *# [1,2,3,4,5,6,7,8]*

**Chained operations**

x **=** (**False == False**) **in** [**False**]  
print(x) *# False*y **= False ==** (**False in** [**False**])  
print(y) *# False*z **= False == False in** [**False**] *# (False == False) and (False in [False])*print(z) *# True*a **= True is False == False** *# (True is False) and (False == False)*print(a) *# False*b **= False is False is False** *# (False is False) and (False is False)*print(b) *# True*print(**"-----------"**)  
print(1 **>** 0 **<** 1) *# True  
# (1 > 0) and (0 < 1)*print(1 **>** (0 **<** 1)) *# False  
# This is like 1 > True  
# 1 > 1*

print(set([**'global'**, **'warming'**]) **==** set([**'warming'**, **'global'**])) *# True*print(set([**'warming'**, **'global'**])) *# order is not preserverd*print(set([**'global'**, **'global'**]) **==** set([**'global'**])) *# True*print(([**'global'**, **'warming'**]) **==** ([**'warming'**, **'global'**])) *# False*print([**'global'**, **'warming'**] **==** [**'warming'**, **'global'**]) *# False*

print(bool(**None**)) *# False***if None:** print(**"hello"**)  
*# output is nothing*

**for** i **in** [1, 0]**:** print(i**+**1)  
  
output**:**2  
1

**def greetPerson**(*\*name*)**:** print(**'Hello'**, *name*)  
  
greetPerson(**'saisk'**) *# Hello ('saisk',)*print(**"-------------------------------------------"**)  
  
**def greetPerson**(*\*name*)**:** print(**'Hello'**, *name*)  
greetPerson(**'saisk'**,**'mouli'**) *# Hello ('saisk', 'mouli')*