DSML Intermediate : Python Refresher - 3

Recap

```
a, b = 12, 5
print(a)
print(b)
```

```
12
5
```

```
print(a + b)
```

```
17
```

```
if "":
    print('True')
else:
    print('False')
```

```
False
```

```
if 0:
    print('True')
else:
    print('False')
```

```
False
```

```
if 0.0:
    print('True')
else:
    print('False')
```

```
False
```

```
if False:
    print('True')
else:
    print('False')
```

```
False
```

```
if True:
    print('True')
else:
    print('False')
```

```
True
```

```
if 0.9:
    print('True')
else:
    print('False')
```

```
True
```

```
a, b = 12, 5

if a + b:
    print('True')
else:
    print('False')
```

```
True
```

```
a = [1,2,3,4,5,6,7,8,9,10]

a[3:8:-1]
```

```
[]
```

Up Next

0. HW to be discussed
1. Tuples
2. Sets and Dictionaries
3. Lists => Deep Copy
4. Comprehension
5. Some more important methods in the list, set, tuples, dict

Python Refresher - 3

Iteration: Looping

While Loops

```
x=0

while (x < 100):
    x+=2

print(x)
```

```
100
```

```
x=0

while (x < 10):
    x+=2
    print(x, x < 10)

print(x)
```

```
2 True
4 True
6 True
8 True
10 False
10
```

Nested List HW

```
board = ['' * 3]
print(board)
```

```
['']
```

```
'a' * 3
```

```
'aaa'
```

```
'' * 3
```

```
''
```

```
board = [''] * 3
print(board)
```

```
['', '', '']
```

```
id(board[0]) == id(board[1]) == id (board[2])
```

```
True
```

```
board = [1, 1, 2]

id(board[0]) == id(board[1]) == id(board[2])
```

```
False
```

```
id(board) # id of the list
```

```
140538585904832
```

```
id(board[0])
```

```
140539114613040
```

```
id(board[1])
```

```
140539114613040
```

```
id(board[2])
```

```
140539114613072
```

```
board = [''] * 3
```

```
print(board)
```

```
['', '', '']
```

```
board[0] = '0'
```

```
print(board)
```

```
['0', '', '']
```

```

```

```
a = 'x'
b = 'x'

id(a) == id(b)
```

```
True
```

```
a = '0'

print(a, b)
```

```
0 x
```

```
board = [''] * 3
board[0] = '0'
print(board)
```

```
['0', '', '']
```

```

```

```
board = [['', '', ''], ['', '', ''], ['', '', '']]
print(board)
```

```
[['', '', ''], ['', '', ''], ['', '', '']]
```

```
board[0]
```

```
['', '', '']
```

```
print(id(board[0]))
print(id(board[1]))
print(id(board[2]))
```

```
140538591717248
140538590568384
140538589167488
```

```
id(board[0][0]) == id(board[1][1]) == id(board[2][2])
```

```
True
```

```
R = len(board)
C = len(board[0])

for i in range(R):
    C = len(board[i])
    for j in range(C):
        print(id(board[i][j]))
```

```
140539114800752
140539114800752
140539114800752
140539114800752
140539114800752
140539114800752
140539114800752
140539114800752
140539114800752
```

Magic

```
board = [['', '', '']] * 3
print(board)
```

```
[['', '', ''], ['', '', ''], ['', '', '']]
```

```
id(board[0])
```

```
140539129280320
```

```
id(board[1])
```

```
140539129280320
```

```
id(board[2])
```

```
140539129280320
```

```
board[0][0] = '0'
```

```
print(board)
```

```
[['0', '', ''], ['0', '', ''], ['0', '', '']]
```

```
id(board[0]) == (board[1]) == id(board[2])
```

```
True
```

```
a = [1, 2, 3]
b = [1, 2, 3] # deep copy => creating a new list => different object
print(id(a) == id(b))

b[1] = 5
print(a)
print(b)
```

```
False
[1, 2, 3]
[1, 5, 3]
```

```
a = [1, 2, 3]
b = a   # shallow copy => the same object in the memory
print(id(a) == id(b))

b[1] = 5
print(a)
print(b)
```

```
True
[1, 5, 3]
[1, 5, 3]
```

```
# board = [123252] * 3
# print(board)
```

```
# id(board[0]) == id(board[1]) == id (board[2])
```

```
# l = [3, 3, 3] # small integer caching [-5, 256]

# id(l[0]) == id(l[1]) == id(l[2])
```

```
# l = [123252, 123252, 123252]

# id(l[0]) == id(l[1]) == id(l[2])
```

List Operations

```
l = [4, 6, 7]

l.append(10)
```

```
print(l)
```

```
[4, 6, 7, 10]
```

```
l.append([1, 2, 3])
print(l)
```

```
[4, 6, 7, 10, [1, 2, 3]]
```

```
len(l)
```

```
5
```

```
l.pop() # what does pop return? => last element
```

```
[1, 2, 3]
```

```
print(l)
```

```
[4, 6, 7, 10]
```

```
# l.pop(7) # pop(index)
```

```
l.remove(7) # does not return, only remove a particular element
```

```
print(l)
```

```
[4, 6, 10]
```

```
# l.remove(7)
```

```
4 in l # citizenship / membership operator
```

```
True
```

```
7 in l
```

```
False
```

```
l = [35, 12, 56, 77]
```

Iteration Protocol

```
# how to check whether some thing is iterable
```

```
iter(l)
```

```
<list_iterator at 0x7fd1b0a7f8b0>
```

```
iter(5)
```

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

TypeError                                 Traceback (most recent call last)

Input In [140], in <cell line: 1>()
----> 1 iter(5)


TypeError: 'int' object is not iterable
```

```
iter("hello")
```

```
<str_iterator at 0x7fd1b1062ac0>
```

```
iter(range(1, 5))
```

```
<range_iterator at 0x7fd1b1062990>
```

```
iter(True)
```

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

TypeError                                 Traceback (most recent call last)

Input In [143], in <cell line: 1>()
----> 1 iter(True)


TypeError: 'bool' object is not iterable
```

```
l = [35, 12, 56, 77]

i = iter(l) # is it sufficient to just get the iterator?
```

```
# you also need the next function
```

```
next(i)
```

```
35
```

```
next(i)
```

```
12
```

```
next(i)
```

```
56
```

```
next(i)
```

```
77
```

```
next(i)
```

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

StopIteration                             Traceback (most recent call last)

Input In [162], in <cell line: 1>()
----> 1 next(i)


StopIteration:
```

```
for i in l:
    print(i)
```

```
35
12
56
77
```

```
# dir(l)
```

```
l = [[1, 2, 3, 4], [5, 6, 7, 8]]
```

```
target = 5

for i in l:
    # i is the list
    # print(i)
    for j in i:
        print(j)
        if j == target:
            print('Found the target')
```

```
1
2
3
4
5
Found the target
6
7
8
```

Tuples

```
l = [4, 5, 2, 3, 4]
print(l)
```

```
[4, 5, 2, 3, 4]
```

```
# tuples are frozen or immutable lists
```

```
t = ()
type(t) # empty tuple
```

```
tuple
```

```
d = {}
type(d)
```

```
dict
```

```
s = {1, 2, 3}
type(s)
```

```
set
```

```
t = (1, 2, 3, 4, 5)
type(t)
```

```
tuple
```

```
t = (1,)
type(t)
```

```
tuple
```

```
t[0] = 5 # immutable
```

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

TypeError                                 Traceback (most recent call last)

Input In [195], in <cell line: 1>()
----> 1 t[0] = 5


TypeError: 'tuple' object does not support item assignment
```

```
t = (1, 2, 3, 5, 6)
print(id(t))

t = (5, 6, 7)
print(id(t))
print(t)
```

```
140538590094656
140538588030656
(5, 6, 7)
```

Creation of Tuples

```
t = (30)

type(t)
```

```
int
```

```
t = (50 + 30 - 7*3)
print(t)

type(t)
```

```
59




int
```

```
t = ("ninja hattori!! " * 2)

print(t)
```

```
ninja hattori!! ninja hattori!!
```

```
type(t)
```

```
str
```

single element tuple

```
t = (1,)
print(t)
```

```
(1,)
```

```
type(t)
```

```
tuple
```

```
t = ("ninja", "hattori")
type(t)
```

```
tuple
```

Unpacking

```
t = (1, 2, 3)

print(t)
```

```
(1, 2, 3)
```

```
a, b, c = t
```

```
print(a)
```

```
1
```

```
print(b)
```

```
2
```

```
print(c)
```

```
3
```

```
a, b = [1, 2]
print(a)
print(b)
```

```
1
2
```

```
a, b, c = (5, 6, 7, 8, 9)
```

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

ValueError                                Traceback (most recent call last)

Input In [204], in <cell line: 1>()
----> 1 a, b, c = (5, 6, 7, 8, 9)


ValueError: too many values to unpack (expected 3)
```

```
a, b, c, d, e = (5, 6, 7, 8, 9)
print(a + b + c + d + e)
```

```
35
```

Packing

```
def foo():
    return 100
```

```
foo()
```

```
100
```

```
x = foo()
```

```
type(x)
```

```
int
```

```
# python functions can return any number of values
def foo():
    return 100, 200 # this data is packed in a tuple
```

```
foo()
```

```
(100, 200)
```

```
x = foo()

type(x)
```

```
tuple
```

```
def foo():
    return 100, 200, 300
```

```
x = foo()
x
```

```
(100, 200, 300)
```

```
we, are, together = foo()
```

```
print(we)
```

```
100
```

```
print(are)
```

```
200
```

```
print(together)
```

```
300
```

```
x = 12, 5
```

```
type(x)
```

```
tuple
```

```
print(x)
```

```
(12, 5)
```

```
t = 1, 2, 3
```

```
type(t)
```

```
tuple
```

```
print(t)
```

```
(1, 2, 3)
```

```
x = 1,
```

```
print(x)
```

```
(1,)
```

Swap

```
a, b = 1, 2
```

```
print(a)
```

```
1
```

```
print(b)
```

```
2
```

```
a, b = b, a
```

```
print(a)
```

```
2
```

```
print(b)
```

```
1
```

Up Next

0. HW to be discussed
1. Sets and Dictionaries
2. Lists => Deep Copy
3. Comprehension
4. Some more important methods in the list, set, tuples, dict

Doubts

```
a = 1999
b = 2999
```

```
# step 1: packing into tuple

x = b, a
```

```
print(a, id(a))
```

```
1999 140538590154544
```

```
print(b, id(b))
```

```
2999 140538590154000
```

```
print(x)
```

```
(2999, 1999)
```

```
# step 2: unpacking from tuple

a, b = x
```

```
print(a, id(a))
```

```
2999 140538590154000
```

```
print(b, id(b))
```

```
1999 140538590154544
```

```
t = (1, 2, 3)
b = t

id(b) == id(t)

# think about it => same or different
```

```
True
```

```
b = (3, 4) # new object gets created here
```

```
print(t)
```

```
(1, 2, 3)
```

```
print(b)
```

```
(3, 4)
```

```
id(b) == id(t)
```

```
False
```

```
a = "iraban"
b = "iraban"
print(id(b))

print(id(a) == id(b)) # immutablev => some space optimisations to
# avoid creating new object
```

```
140538591235632
True
```

```
b = "dutta"

print(a)
print(id(b))
print(id(a) == id(b))
```

```
iraban
140538591364976
False
```

```
a = [1, 2, 3]
b = [1, 2, 3]

id(a) == id(b) # mutable

print(id(a))

a[0] = 4

print(id(a))
```

```
140538588074944
140538588074944
```

```
l = [2] * 3

print(l)
```

```
[2, 2, 2]
```

```
l = [[2] * 3] * 3
# [2] * 3 => shallow copy of the list
print(l)
```

```
[[2, 2, 2], [2, 2, 2], [2, 2, 2]]
```

```
l[0][0] = 5
```

```
print(l)
```

```
[[5, 2, 2], [5, 2, 2], [5, 2, 2]]
```

```
id(l[0]) == id(l[1]) == id(l[2])
```

```
True
```

```
l = [[2, 2, 2], [2, 2, 2], [2, 2, 2]]
```

```
id(l[0]) == id(l[1]) == id(l[2])
```

```
False
```

```
nl=[['','',''],['','',''],['','','']]

for i in range(len(nl)):
    for j in range(len(nl[0])):
        print(id(nl[i][j]))
```

```
140539114800752
140539114800752
140539114800752
140539114800752
140539114800752
140539114800752
140539114800752
140539114800752
140539114800752
```

```
l = [[5, 6, 7], [8, 9,  10]]
```

```
id(l)
```

```
140538588078016
```

```
id(l[0])
```

```
140538591574720
```

```
id(l[0][0])
```

```
140539114613168
```

```
lst = [1,2,3,4,5]
lst1 = lst[:] # different objects => deep copy

# list slicing always gives you a new list
id(lst) == id(lst1)
```

```
False
```

```
l = [1, 2, 3]

id(l)
```

```
140538592264256
```

```
l.append(4) # mutable => changed the orignial list without changing
# the object

id(l)
```

```
140538592264256
```

```
x = "abc"
id(x)
```

```
140539383334320
```

```
x = "amit sethi" # immutable => when changed a new object gets created
id(x)
```

```
140538594651568
```

```
a = 3
id(a)
```

```
140539114613104
```

```
a = 5 # immutable => when changed a new object gets created
id(a)
```

```
140539114613168
```

```
bool("False")
```

```
True
```

```
bool("")
```

```
False
```

```
l = [1, 2, 3, 4]

d = l[:] # list slice is a new object => deep copy => different object

id(l) == id(d)
```

```
False
```

```
d2 = l.copy() # creates a deep copy
```

```
print(d2)
```

```
[1, 2, 3, 4]
```

```
id(l) == id(d2)
```

```
False
```

```
l[0] = 5

print(l)
```

```
[5, 2, 3, 4]
```

```
print(d2)
```

```
[1, 2, 3, 4]
```