In [343]:

```
# LIST  --- collection of certain items, separated by coma between square brackets
```

In [344]:

```
L = [1, '11', "1da", [1, 3], ["X", 10, (11, 7)]] # lists can have different types of el
ements
```

In [345]:

```
type(L) # return the type of a variable
```

Out[345]:

```
list
```

In [346]:

```
empty = [] # empty list has no elements
empty
```

Out[346]:

```
[]
```

In [347]:

```
len(empty) # number of elements in a list
```

Out[347]:

```
0
```

In [348]:

```
f = [11, 12, 13, 14] # create a list of elements 11, 12, 13, 14
f
```

Out[348]:

```
[11, 12, 13, 14]
```

In [349]:

```
f += [15, 16] # add 15 and 16 right after 14 to f
f
```

Out[349]:

```
[11, 12, 13, 14, 15, 16]
```

In [350]:

```
# to add a single element 17 to the end of the list f
f.append(17) #, more likely append is the same as the incrementation, but takes exactly
1 element.
f
```

Out[350]:

```
[11, 12, 13, 14, 15, 16, 17]
```

In [33]:

```
# f.append(18, 19, 20) # is not correct
```

```
---------------------------------------------------------------------------
-
TypeError                                 Traceback (most recent call las
t)
<ipython-input-33-5bdd47fbd634> in <module>
----> 1 f.append(18, 19, 20)

TypeError: append() takes exactly one argument (3 given)
```

In [351]:

```
f.extend([18, 19, 20])
f
```

Out[351]:

```
[11, 12, 13, 14, 15, 16, 17, 18, 19, 20]
```

In [352]:

```
# insert element at a particular location
f.insert(0,8) # insert 8 to the first element (which is indexed by 0)
f
```

Out[352]:

```
[8, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]
```

In [353]:

```
f.insert(1,9)
f
```

Out[353]:

```
[8, 9, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]
```

In [194]:

```
f.index(100) # returns the first index of a value 9 in a list f if the value already ex
ists in a list
```

```
--------------------------------------------------------------------------
-
ValueError                                Traceback (most recent call las
t)
<ipython-input-194-8dc70a7a16a2> in <module>
----> 1 f.index(100) # returns the first index of a value 9 in a list f

ValueError: 100 is not in list
```

In [354]:

```
9 in f #decide simply if an element appears in a list
```

Out[354]:

```
True
```

In [355]:

```
#if an element appears in a list we can return its index
```

In [356]:

```
b = -1
if 100 in f:
    b = f.index(100)
b
```

Out[356]:

```
-1
```

In [357]:

```
b = -1
if 9 in f:
    b = f.index(9)
b
```

Out[357]:

```
1
```

In [358]:

```
# if an element exists in a list we can insert something directly after that.
f.insert(f.index(9)+1, 10)
f
```

Out[358]:

```
[8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]
```

In [359]:

```
f.remove(8) # remove element 8 from list f
f
```

Out[359]:

```
[9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]
```

In [360]:

```
f.remove(8) # remove element 8 from list f (only if the element exists in the list)
f
```

```
---------------------------------------------------------------------------
-
ValueError                                Traceback (most recent call las
t)
<ipython-input-360-30ffdd4888a9> in <module>
----> 1 f.remove(8) # remove element 8 from list f (only if the element ex
ists in the list)
      2 f

ValueError: list.remove(x): x not in list
```

In [361]:

```
p = f.pop() # remove and return the last element from the list
f
```

Out[361]:

```
[9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
```

In [362]:

```
p
```

Out[362]:

```
20
```

In [363]:

```
f.sort() # sort elements (by default in ascending order)
f
```

Out[363]:

```
[9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
```

In [364]:

```
f.sort(reverse=True) # sort elements in descending order
f
```

Out[364]:

```
[19, 18, 17, 16, 15, 14, 13, 12, 11, 10, 9]
```

In [365]:

```
f.reverse() # reverse the list
f
```

Out[365]:

```
[9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
```

In [366]:

```
max(f)
```

Out[366]:

```
19
```

In [367]:

```
min(f)
```

Out[367]:

```
9
```

In [368]:

```
f.count(10) # how many times an item appears in a list
```

Out[368]:

```
1
```

In [369]:

```
f.count(5)
```

Out[369]:

```
0
```

In [370]:

```
f.clear() # remove all the elements from a list
f == []
```

Out[370]:

```
True
```

In [ ]:

```

```

In [371]:

```
L = list(range(10))
L
```

Out[371]:

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

In [372]:

```
L[0] # get the first element from the list
```

Out[372]:

```
0
```

In [373]:

```
L[1] # get the second element from the list
```

Out[373]:

```
1
```

In [374]:

```
len(L) # number of elements in the list, last element is indexed by len(L)-1
```

Out[374]:

```
10
```

In [375]:

```
L[len(L)-1] # the last element in the list
```

Out[375]:

```
9
```

In [376]:

```
L[len(L)-2] # second element from the back
```

Out[376]:

```
8
```

In [377]:

```
L[-1] # different way of indexing elements last element is indexed by -1
```

Out[377]:

```
9
```

In [378]:

```
L[-2] == L[len(L)-2]
```

Out[378]:

```
True
```

In [379]:

```
list(range(10)) # list of numbers from 0 to 10 (including 0, but excluding 10, range(1
0)=[0,10))
```

Out[379]:

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

In [380]:

```
g = []
k = 0
while k < 10:
    if (k % 2 == 0):
        g.append(k)
    k+=1
del k
g
```

Out[380]:

```
[0, 2, 4, 6, 8]
```

In [381]:

```
g = []
for k in range(10):
    if (k % 2 == 0):
        g.append(k)
g
```

Out[381]:

```
[0, 2, 4, 6, 8]
```

In [431]:

```
[k for k in range(10) if k % 2 == 0] # create a list of even numbers using list compreh
ension
```

Out[431]:

```
[0, 2, 4, 6, 8]
```

In [ ]:

```

```

In [383]:

```
[L[0], L[1], L[2]] # create a new list of the first three element from the list
```

Out[383]:

```
[0, 1, 2]
```

In [384]:

```
[L[k] for k in range(3)]
```

Out[384]:

```
[0, 1, 2]
```

In [385]:

```
L[0:3] # first 3 elements in the list L[0:3] = [L[0], L[1], L[2]]
```

Out[385]:

```
[0, 1, 2]
```

In [386]:

```
L[:3] # if the first argument is not specified, it means it is zero (L[:3]==L[0:3])
```

Out[386]:

```
[0, 1, 2]
```

In [387]:

```
L[3:] # if the second argument is not specified it means it is the last element
```

Out[387]:

```
[3, 4, 5, 6, 7, 8, 9]
```

In [388]:

```
L[:] # from the first to the last (both arguments neither specified) which means it is
 the whole list
```

Out[388]:

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

In [432]:

```
list(range(0,10,2)) # exactly the same, but the most efficient way (create a range from
0 to 10 but exclude every second item)
```

Out[432]:

```
[0, 2, 4, 6, 8]
```

In [389]:

```
L[1:4] # 3 elements after the first
```

Out[389]:

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

In [390]:

```
[L[4], L[5], L[6]]
```

Out[390]:

```
[4, 5, 6]
```

In [391]:

```
[L[k] for k in range(4,7)]
```

Out[391]:

```
[4, 5, 6]
```

In [392]:

```
L[4:7]
```

Out[392]:

```
[4, 5, 6]
```

In [393]:

```
L[4:7:1] # specify a skipping (1 means every element included between 4th and 7th (7th
 still excluded))
```

Out[393]:

```
[4, 5, 6]
```

In [394]:

```
L[4:7:2] # hold every second element
```

Out[394]:

```
[4, 6]
```

In [395]:

```
L[-3:-1] # -1 is the last element (which is excluded)
```

Out[395]:

```
[7, 8]
```

In [396]:

```
L[-3:] # we can include the last element as well
```

Out[396]:

```
[7, 8, 9]
```

In [397]:

```
L[-3:-1:1] # number of steps can be specified
```

Out[397]:

```
[7, 8]
```

In [398]:

```
L[:-4:-1] # if the number of steps is negative it means the order of the elements will
 be reversed
```

Out[398]:

```
[9, 8, 7]
```

In [399]:

```
L[::-1] # all the elements in reversed order
```

Out[399]:

```
[9, 8, 7, 6, 5, 4, 3, 2, 1, 0]
```

In [400]:

```
L.reverse()
L
```

Out[400]:

```
[9, 8, 7, 6, 5, 4, 3, 2, 1, 0]
```

In [401]:

```
reversed(L)
```

Out[401]:

```
<list_reverseiterator at 0x1c5a36de4a8>
```

In [299]:

```
list(reversed(L))
```

Out[299]:

```
[9, 8, 7, 6, 5, 4, 3, 2, 1, 0]
```

In [402]:

```
L[1:3] = [11, 12] # slice assignment
L
```

Out[402]:

```
[9, 11, 12, 6, 5, 4, 3, 2, 1, 0]
```

In [403]:

```
L[1:3]=[] # remove elements using slicing assignment
L
```

Out[403]:

```
[9, 6, 5, 4, 3, 2, 1, 0]
```

In [404]:

```
LL = L # looks like this assignment create a copy of list L, but it is just a reference
LL
```

Out[404]:

```
[9, 6, 5, 4, 3, 2, 1, 0]
```

In [405]:

```
LL[0]=100 # we change the first element of LL
LL
```

Out[405]:

```
[100, 6, 5, 4, 3, 2, 1, 0]
```

In [406]:

```
L # as LL is a reference to L, L[0] changed to 100 as well
```

Out[406]:

```
[100, 6, 5, 4, 3, 2, 1, 0]
```

In [407]:

```
LLL=list(LL) # now we copied the LL list, LLL is not a reference to LL
LLL
```

Out[407]:

```
[100, 6, 5, 4, 3, 2, 1, 0]
```

In [408]:

```
LLL[0]=1
LLL
```

Out[408]:

```
[1, 6, 5, 4, 3, 2, 1, 0]
```

In [409]:

```
LL # LL list remains the same
```

Out[409]:

```
[100, 6, 5, 4, 3, 2, 1, 0]
```

In [410]:

```
# this approach fails when the list has list elements
```

In [411]:

```
X = [[10, 2], [20, 3], [30, 4]]
X
```

Out[411]:

```
[[10, 2], [20, 3], [30, 4]]
```

In [412]:

```
Y = list(X)
Y
```

Out[412]:

```
[[10, 2], [20, 3], [30, 4]]
```

In [413]:

```
Y[0][0] = 100
Y
```

Out[413]:

```
[[100, 2], [20, 3], [30, 4]]
```

In [414]:

```
X
```

Out[414]:

```
[[100, 2], [20, 3], [30, 4]]
```

In [415]:

```
import copy
```

In [416]:

```
Y = copy.deepcopy(X)
```

In [417]:

```
X[0][0] = 10
X
```

Out[417]:

```
[[10, 2], [20, 3], [30, 4]]
```

In [418]:

```
Y
```

Out[418]:

```
[[100, 2], [20, 3], [30, 4]]
```

In [419]:

```
import itertools
```

In [420]:

```
flatY=list(itertools.chain(*Y))
```

In [421]:

```
flatY
```

Out[421]:

```
[100, 2, 20, 3, 30, 4]
```

In [422]:

```
X = [1, 1, 2, 3, 3, 4, 5, 3, 2, 3]
X
```

Out[422]:

```
[1, 1, 2, 3, 3, 4, 5, 3, 2, 3]
```

In [423]:

```
[k for k in X if X.count(k)>1] # find all the repeated elements
```

Out[423]:

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

In [425]:

```
list(set([k for k in X if X.count(k)>1])) # these are the duplicated elements in X
```

Out[425]:

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

In [426]:

```
from collections import deque
```

In [427]:

```
Z=deque(X)
```

In [428]:

```
Z.rotate(1)
```

In [430]:

```
list(Z)
```

Out[430]:

```
[3, 1, 1, 2, 3, 3, 4, 5, 3, 2]
```

In [ ]:

```

```

In [ ]:

```
import numpy as np
```

In [ ]:

```
list(np.arange(0,10, 2))
```

In [ ]:

```
np.linspace(0, 8, 5, dtype="int")  # by default, np.linspace returns an array filled wi
th doubles
```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```
#  STRING
```

In [433]:

```
sentence = "I got this feeling on the summer day when you were gone"
sentence
```

Out[433]:

```
'I got this feeling on the summer day when you were gone'
```

In [440]:

```
type(sentence)
```

Out[440]:

```
str
```

In [441]:

```
sentence[0] # we can access the characters in the string the same way as in lists
```

Out[441]:

```
'I'
```

In [442]:

```
sentence[-1]
```

Out[442]:

```
'e'
```

In [ ]:

```

```

In [435]:

```
sentence_list = sentence.split(" ")
sentence_list
```

Out[435]:

```
['I',
 'got',
 'this',
 'feeling',
 'on',
 'the',
 'summer',
 'day',
 'when',
 'you',
 'were',
 'gone']
```

In [436]:

```
type(sentence_list)
```

Out[436]:

```
list
```

In [437]:

```
s = " ".join(sentence_list)
s
```

Out[437]:

```
'I got this feeling on the summer day when you were gone'
```

In [154]:

```
word = "hello"
```

In [155]:

```
word_list = list(word) # create a list from a string
```

In [438]:

```
integer = 12345
integer
```

Out[438]:

```
12345
```

In [159]:

```
integer_list = list(integer) # this not works
```

```
---------------------------------------------------------------------------
-
TypeError                                 Traceback (most recent call las
t)
<ipython-input-159-b0bc34acfc73> in <module>
----> 1 integer_list = list(integer) # this not works

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

In [439]:

```
integer_list = list(str(integer)) # is there any way to specify datatype ???
integer_list
```

Out[439]:

```
['1', '2', '3', '4', '5']
```

In [174]:

```
np.array(integer, dtype="int")
```

Out[174]:

```
array(12345)
```

In [176]:

```
[k for k in str(integer)]
```

Out[176]:

```
['1', '2', '3', '4', '5']
```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```
# DICTIONARY
```

In [177]:

```
d={'A': 1, 'B' : 2, 'C' : 3, 'D' : 4}
```

In [178]:

```
d
```

Out[178]:

```
{'A': 1, 'B': 2, 'C': 3, 'D': 4}
```

In [179]:

```
d.keys()
```

Out[179]:

```
dict_keys(['A', 'B', 'C', 'D'])
```

In [180]:

```
type(d.keys())
```

Out[180]:

```
dict_keys
```

In [181]:

```
d.values()
```

Out[181]:

```
dict_values([1, 2, 3, 4])
```

In [182]:

```
type(d.values())
```

Out[182]:

```
dict_values
```

In [ ]:

In [ ]:

In [ ]: