Open in Colab

Policy:

- You are encouraged to discuss with each other, but the final submitted script must be your own workings.
- You may use any IDE for python coding.
- All submission should be on-line by the deadline, and not extension is allowed.
- Submit both pdf and .jpynb (or .py file).

Question-1 Understand lists

Instance creation,list creation addition (concatenation), and length query.

1. Create, respectively, two objects: x and y.
2. Create two lists called x_list and y_list each contains, respectively, 10 instances of variables x and y.
3. Print the lengths of these two lists.
4. Create next a longer list called big_list, which contains all the variables x and y, by concatenation.
5. Count the number of the objects in all these three lists.

```
x = 4
y = 5
x_list = [x]*10
y_list = [y]*10
print(f'length of the x list is {len(x_list)}')
print(f'length of the y list is {len(y_list)}')
big_list = x_list+y_list
num_ob = len(x_list+y_list+big_list)
```

```
length of the x list is 10
10
```

Question-2 Understand memory addresses

1. Assign 3 different numbers, respectively, to three variables n1, n2, and n3
2. Create a list called n_list, and pass these three numbers to the n_list.
3. Assign 2 different numbers, respectively, to two variables m1, and m2.
4. Create a list called m_list, and pass m1 and m2 to the m_list.
5. Print the numbers of the elements in the n_list, and lengths of the n_list.
6. Print the memory addresses of n1 and that of the first numbers in the n_list.
7. Print the memory addresses of m2 and that of the last numbers in the m_list.
8. Change the value for n1 and n_list[0], respectively, to another number, and print out n1,n_list[0] and m_list[0].
9. Create a longer list using n_list and m_list. Print its length of this newly created list. Print the addresses of n1,first element in n_list, the fist element in nm_list, m2, and the last element in nm_list.
10. Change the value of n1 and the first element in nm_list. Print the addresses of n1,first element in n_list, first element in nm_list.

Discuss about the results obtained.

```
n1 = 3
n2 = 5
n3 = 7
n_list = [n1,n2,n3]
m1 = 11
m2 = 13
m_list = [m1,m2]
print(f'the elements in n_list are {n_list} and the length is {len(n_list)}')
print(f'the id of n1 {id(n1)} and of first in n_list {id(n_list[0])}')
print(f'the id of m2 {id(m2)} and of last in m_list {id(m_list[-1])}')
n1 = 17
n_list[0] = 19
print(f'n1 is {n1}, nlist[0] is {n_list[0]} and mlist 0 is {m_list[0]}')
```

```
nm_list = n_list + m_list
print(f'length of nm_list {len(nm_list)}, addr n1 {id(n1)}, addr n_list 0 {id(n_list[0])}, nm 0 {id(nm_list[0])}, addr m2 {id(m2)}, addr last
```

```
the elements in n_list are [3, 5, 7] and the length is 3
the id of n1 9793152 and of first in n_list 9793152
the id of m2 9793472 and of last in m_list 9793472
n1 is 17, nlist[0] is 19 and mlist 0 is 11
length of nm_list 5, addr n1 9793600, addr n_list 0 9793664, nm 0 9793664, addr m2 9793472, addr last in nm_list 9793472
```

Notably the ids in memory of the n1 and the first element in n_list are the same. When the values are updated later the addresses also change to new unique numbers. Referencing the new combined list also points to the same place in memory as before.

Question-3 Understand lists: creation, appending, concatenation

1. Create two empty lists: "numbers" and "strings".
2. Use the "append" list method to add three members of 5,7, and 4 to the "numbers" list, in that order.
3. Then, create a list named "names", and assign it with three names: 'John', 'Eric', and 'Jessica', in that order
4. Next, add words 'John', 'Eric', and 'Jessica' to the "strings" list.
5. Assign a variable "second_name" with the second name in the names list, using the brackets operator [], and the member list.

Note that the index is zero-based, and hence the index the second item in the list should be 1, and so on.

```
numbers = []
strings = []
numbers.append(5)
numbers.append(7)
numbers.append(4)
names = ["John", "Eric", "Jessica"]
strings.append(names)
second_name = names[1]
```

Question-4 Understand more list operations

1. Create first a list "materials", and assign it with three names: 'Plastic', 'Wood', 'Steel', and 'Aluminum', in that order.
2. Use the "append" method to add the 1st and 3rd members to a new list "selected_materials".
3. Generate a list containing the unselected materials.
4. Print out the selected materials and the unselected materials.

```
materials = ["Plastic", "Wood", "Steel", "Aluminum"]
selected_materials = []
selected_materials.append(materials[0])
selected_materials.append(materials[2])
unselected_materials = materials[1:4:2]
print(f'selected materials are {selected_materials} and unselected are {unselected_materials}')
```

```
selected materials are ['Plastic', 'Steel'] and unselected are ['Wood', 'Aluminum']
```

Question-5 Lists and Numpy Array for scientific computations

1. Create a list that contain 10 values for 10 hollow balls with different outer diameters.
2. Create another list that contain 10 different values of inner diameters for the same set of 10 hollow balls.
3. Create two corresponding Numpy arrays using these two lists.
4. Write a code to compute the volumes of these 10 balls using these Numpy arrays.
5. Write a code to compute the volumes of these 10 balls using these two lists.
6. print out only the volume values that are above or equal the average value.

```
import numpy as np
dia1 = [1,2,3,4,5,6,7,8,9,10]
dia2 = [11,12,13,14,15,16,17,18,19,20]
d1 = np.asarray(dia1)
d2 = np.asarray(dia2)
vol_np = [4*np.pi*(y**3 - x**3)/3 for x,y in zip(d1, d2)]
vol_l = [4*np.pi*(y**3 - x**3)/3 for x,y in zip(dia1, dia2)]
```

```
idx = np.argmax(vol_np>np.average(vol_np))
print(vol_np[idx:])
```

```
[16252.103994771196, 19142.771235873806, 22284.3638894636, 25677.283955340576, 29321.531433504733]
```

Question-6 Use of list comprehension for list and Numpy arrays

Using list comprehension, complete the following taks:

1. Create an initial list with 10 different values of float type with both positive and negative.
2. Create a "newlist" using the initial list. The "newlist" that contains only the positive integer values in the initial list.
3. Create a Numpy array using the initial list "values", and then create a "newlist" that only contains the positive integers in the initial list. Finally, create a numpy array with only these positive integers.

```
values = [-1.0, -0.7, 0.7, 0.3, 0.5, -0.2, -0.6, 0.9, 1.1, -3.2]
newlist = [x for x in values if x > 0.0]
np_new = np.asarray(values)
np_newlist = np_new[np_new>0.0]
```

Question-7 Understanding conditions

Change the variables in Section 1 of the codes given below, so that each if statement resolves as True, and hence all integers from 1 to 6 are printed.

```
#----- Section 1 ------
number1 = 16
number2 = 0
list1 = [1]
list2 = [1,3,4,5]
#----------------------

if number1 > 15:
    print("1: True")

if list1:
    print("2: True")

if len(list2) == 2:
    print("3: True")

if len(list1) + len(list2) == 5:
    print("4: True")

if list1 and list1[0] == 1:
    print("5: True")

if not number2:
    print("6: True")
```

```
1: True
2: True
4: True
5: True
6: True
```

Question-8 Controlling print outs

Write a shortest possible code to print out only all odd numbers bellow 100.

```
print(*range(1,100,2))
```

```
1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45 47 49 51 53 55 57 59 61 63 65 67 69 71 73 75 77 79 81 83 85 87 89 91 93
```

Question-9 Practicing loops

Loop through and print out all the odd numbers in the given list of numbers in the same order. Skip all the numbers that come after the first 183 in the sequence of the list.

```
n_list = [
   851, 301, 883, 651, 360, 68, 308, 318, 601, 385, 880, 505, 515, 535, 533, 545,
   625, 183, 165, 131, 501, 163, 615, 865, 555, 118, 380, 883, 581, 136, 105, 541,
   486, 361, 35, 318, 805, 333, 136, 355, 813, 566, 585, 858, 318, 615, 853,704,
   388, 161, 558, 118, 818, 135, 311, 566, 816, 138, 866, 850, 616, 838, 685,108,
   515, 65, 103, 58, 511, 13, 881, 883, 565, 553, 81, 358, 833, 831, 335, 531,332
   ]
for val in n_list:
  if val % 2 == 1:
    print(val)
  if val == 183:
    break
```

```
851
301
883
651
601
385
505
515
535
533
545
625
183
```