Objective: This worksheet introduces a compound or collection data type ***List.*** It is called a compound or collection data type because it ***groups the data together*** in a single element. List has several usages in Data Science programming.

Lists

- The features of a list can be summarised as below:
  - The items of a list are specified inside the square brackets [].
  - A list can group heterogeneous items together.
  - A list is mutable.
  - A list can contain duplicate items.
  - A list can contain other lists.
- Let us understand the details of each of above. An empty list ***players*** is created first and then the names of the Indian Cricket Players are added one at a time using the ***insert()*** function that is available for list data type. Please note ***players*** is a variable of type list here.

```
players = []

players.insert(0, "Kapil");
players.insert(1, "Tendulkar")
players.insert(2, "Dravid")

print(type(players))
<class 'list'>

print(players)
['Kapil', 'Tendulkar', 'Dravid']
```

◄------ *An empty list players*

◄------ *Elements are added using insert() function. 0, 1 and 2 are indices.*

◄------ *players is list data type*

◄------ *Print the list elements*

- A list can contain heterogeneous elements. It means the items of a list may not be of the same data type. For example, if there is a need to store the first name, present age, and run rate of test matches for Sachin Tendulkar in a list, it can be done in the following way:

```
tendulkar_details = ["Sachin", 47, 43.78]

print(type(tendulkar_details[0]))
<class 'str'>

print(type(tendulkar_details[1]))
<class 'int'>

print(type(tendulkar_details[2]))
<class 'float'>
```

◄------ *First list item is a string*

◄------ *Second item is an integer*

◄------ *The last item is a float*

- The Python provides a function ***split()*** that splits the constituents of a string into a list. It can take two optional parameters:
  1. **sep**: It is a delimiter(or separator) according to which the string will be splitted. The default value is none that means split according to white space. It discards empty strings from the resulting split.
  2. **maxsplit**: maximum number of splits to be done. The default value is -1, that means no upper limit.

The following examples show the usage of ***split()*** function:

```
str = "Data Science Software Engineering Course"
```
<- *A string*

```
str.split()
['Data', 'Science', 'Software', 'Engineering', 'Course']
```
<- *No parameters. As many parts as possible from white spaces as separator*

```
str.split(" ", 2)
['Data', 'Science', 'Software Engineering Course']
```
<- *2 splits =3 parts from white spaces*

```
str.split(" ", 3)
['Data', 'Science', 'Software', 'Engineering Course']
```
<- *3 splits = 4 parts from white spaces*

```
str = "Data#Science#Software#Engineering#Course"
str.split("#", 4)
['Data', 'Science', 'Software', 'Engineering', 'Course']
```
<- *4 splits = 5 parts from # separator*

- Retrieval of the list parts using individual indices or index-range works as it happens with Strings processing. This is also called ***slicing***. The following examples explain the operations:

```
str = "Deep Learning involves Neural Networks"
words = str.split()
print(words[4])
print(words[2:4])
print(words[-1])
```

```
Networks
['involves', 'Neural']
Networks
```

Exercise:

What will be the output of the following?

```
equipment = ["routers", "switches","modems"]
print(equipment[:2])
print(equipment[1:])
```

- Lists need not contain only strings. They can contain integers, float and even boolean values and the data type will still remain list as shown below:

```
fibonacci = [1, 1, 2, 3, 5, 8, 13, 21]
print(type(fibonacci))
```

```
<class 'list'>
```

- The following example iterates a list using the ***for*** loop. Note that with each iteration ***fibTerm*** variable accesses the next list element.

```
for fibTerm in fibonacci:
    print(fibTerm)
```

```
1
1
2
3
5
8
13
21
```

- The ***len()*** in-built function can be used to get the length of the list data type. It can also be used to iterate the list as shown in the example below:

```
print(len(fibonacci))
```

```
8
```

```
length = len(fibonacci)
for i in range(length):
    print(fibonacci[i])
```

```
1
1
2
3
5
8
13
21
```

Exercise:

A list contains the BMI record of a person that has elements for his Name, Weight, Height in meters and a boolean value if he is obese or not. Using the ***for*** loop, print the value of the each list item and its type in the same line.

- The lists are ***mutable***. It means the content of a list can be altered. For example a list contains the data about an employee of an organization. If there are mistakes, it can be corrected. More details to same list can also be added later using in-built ***append()*** function as shown in the example below:

```
employee = ["Peter Jack", "12345", "Software Engineer", "True"]
print(employee)
employee[0] = "Jack Thomas"
print(employee)
employee.append("jack@email.com")
print(employee)
```

```
['Peter Jack', '12345', 'Software Engineer', 'True']
['Jack Thomas', '12345', 'Software Engineer', 'True']
['Jack Thomas', '12345', 'Software Engineer', 'True', 'jack@email.com']
```

- A list can contain duplicate values. There is NO check for it.

```
funnyList = ["Funny", "False", 12, "False", 12, "Funny"]
print(funnyList)
```

```
['Funny', 'False', 12, 'False', 12, 'Funny']
```

Deep Copy Concept

- Review the following example. Why the value of list Y also changed? How to avoid it so that Y still has the old value of X?

```
X = [1, 2]
Y = X
print(Y)
X[0] = X[0] + X[1]
print(X)
print(Y)
```

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

- This can be achieved using the Deep Copy Concept. Observe the highlighted line in the example below:

```
X = [1, 2]
Y = [item for item in X]
print(Y)
X[0] = X[0] + X[1]
print(X)
print(Y)
```

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

Copy item by item.
Do not just refer to the same object

List of Lists

- The test runs scored by few Indian cricket players are shown below. There is a requirement to save them in a suitable data structure for Python programming.

Sl. No.	Player	Test Runs
1	Dhoni	4876
2	Dravid	13265
3	Ganguly	7212
4	Gavaskar	10122
5	Kapil	5248
6	Tendulkar	15921

- This objective can be achieved using ***list of lists*** as shown below. Observe the usage of square brackets.

```
cricketPlayers = [
    ["Dhoni", 4876],
    ["Dravid", 13265],
    ["Ganguly", 7212],
    ["Gavaskar",10122],
    ["Kapil", 5248],
    ["Tendulkar", 15921]
]
```

- Each list element within the ***cricketPlayers*** list can be accessed using 0, 1, 2 index.

```
print(cricketPlayers[2])
```

```
['Ganguly', 7212]
```

- If there is a need to access the runs scored by Kapil, it can be accessed using the double index mechanism as it is done in two-dimensional arrays in C/C++. Note the element to access Kapil's runs is [4][1]:

```
print(cricketPlayers[4][1])
```

```
5248
```

- Count of elements in a list inside the main list need not be same. For example the run rate for Tendulkar can only be added alone. Note Tendulkar is the 5th list element in the main list.

```
cricketPlayers[5].append(55.8)
```

```
print(cricketPlayers)
```

```
[['Dhoni', 4876], ['Dravid', 13265], ['Ganguly', 7212], ['Gavaskar', 10122], ['Kapil', 5248], ['Tendulkar', 15921, 5
5.8]]
```