Objective: This worksheet introduces another compound or collection data types ***Tuples*** and ***Sets.*** Participants are expected to practice and observe the differences among Lists, Tuples and Sets.

Tuples

- The features of a tuple can be summarised as below:
  - The items of a list are specified inside the parentheses ().
  - A tuple can also group heterogeneous items together.
  - A tuple is immutable.
- To understand the basics of tuples let us create a simple tuple of human weights as ***bmiCategories.*** Note that the tuple is created using parentheses, but the individual tuple elements are still accessed using the index inside the square brackets. All slicing concepts are same as they are in lists.

```
bmiCategories = ("Underweight", "Normal", "Overweight", "Obese")
```

```
print(type(bmiCategories))
```

```
<class 'tuple'>
```

```
print(bmiCategories[2])
```

```
Overweight
```

```
print(bmiCategories[-1])
```

```
Obese
```

- A built-in function ***index()*** can be used with tuples to get the index of an element of a tuple. Note: some of these functions and methods can also work with other collection data types. Use online help along with Jupyter Notebook to see the details.

```
print(bmiCategories.index("Overweight"))
```

```
2
```

- We can check if an element is a part of a tuple or not as shown below. The return result will be a boolean.

```
print("Thin" in bmiCategories)
```

```
False
```

- Unlike lists, tuples are immutable. The content of a tuple cannot be changed once it is declared. If an attempt is made to do so, it throws an error that the item assignment is not supported for tuple object.

```
bmiCategories[0] = "Thin"
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-7-8c584f234054> in <module>
----> 1 bmiCategories[0] = "Thin"

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

- Immutable property makes tuples suitable to store ***constant*** data that should not be changed intentionally or unintentionally throughout the code.
- Tuple creation is faster than lists. Jupyter Notebook and few other development environments like Google Colab supports magic commands that can perform useful operations for the programmers. The details of magic commands are not discussed here, but a magic command ***%%timeit*** will be utilized to measure the performance of tuple creation and compare it against the list creation. Other magic commands can be seen typing %% in the cell followed by a TAB.

```
%%timeit -n1 -r5
for i in range(1000000):
    myList = ["Check", "who", "is", "faster"]
```

```
151 ms ± 5.59 ms per loop (mean ± std. dev. of 5 runs, 1 loop each)
```

```
%%timeit -n1 -r5
for i in range(1000000):
    myTuple = ("Check", "who", "is", "faster")
```

```
62 ms ± 7.12 ms per loop (mean ± std. dev. of 5 runs, 1 loop each)
```

- In the above example a list and a tuple is created 10^6 times in a ***for*** loop. The operation is repeated only once (-n1) and the best value is picked up out of five (-r5). We can observe that the list creation is almost 2.5 times slower than the tuple creation. Python achieves this performance improvement on the tuples because of its immutable property.

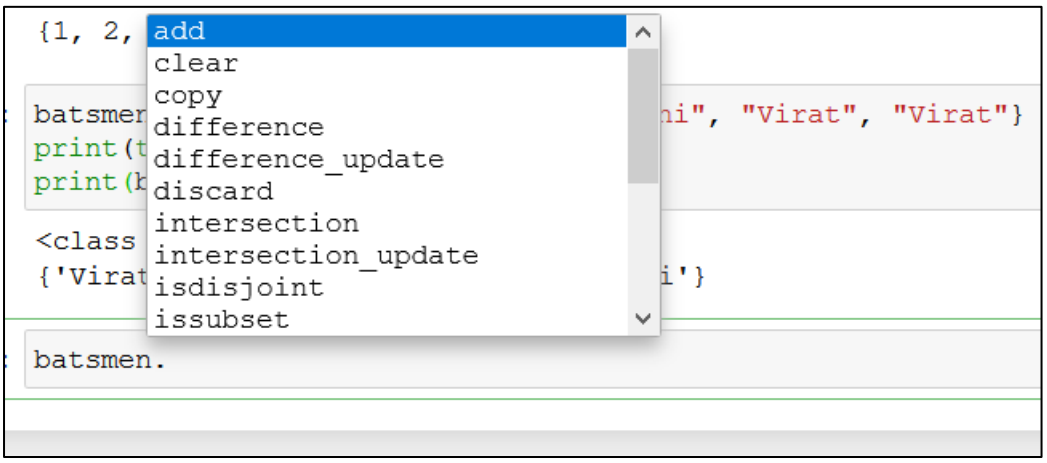
Sets

- A set is a collection of unique elements. The duplicate elements cannot be present in a set.
- A set can be initialized with items enclosed within the curly brackets {}.

```
batsmen = {"Tendulkar", "Rohit", "Dhoni", "Virat", "Virat"}
print(type(batsmen))
print(batsmen)
```

```
<class 'set'>
{'Rohit', 'Tendulkar', 'Virat', 'Dhoni'}
```

- The above example creates a set ***batsmen***. It was attempted to place duplicate elements (Virat), but when it is printed, all duplicate elements were removed. The data type is set.
- Many set theory related functions are available with set data type. A screenshot shows the glimpse of it, when the previous set ***batsmen*** is typed with a dot and the TAB is pressed.



- The presence of elements can be checked as it was done for tuples.

```
print("Rohit" in batsmen)
```
```
True
```
```
print ("Dravid" in batsmen)
```
```
False
```

- Unlike tuples, ***index()*** function is not available in sets because sets do not store elements in an order. So they do not have indices.

```
batsmen.index("Rohit")
```
```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-29-90ac87f4e015> in <module>
----> 1 batsmen.index("Rohit")

AttributeError: 'set' object has no attribute 'index'
```

- Set theory related functions can be performed. The example below shows the usage of a function ***intersection()*** when two sets batsmen and bowlers are created:

```
batsmen = {"Jadeja", "Tendulkar", "Rohit", "Dhoni", "Virat"}
bowlers = {"Bumrah", "Jadeja", "Ishant"}
allRounders = batsmen.intersection(bowlers)
print(type(allRounders), allRounders)
```
```
<class 'set'> {'Jadeja'}
```

- The example below shows that searching is faster (order of ms to µs) in cases of sets because Python does some optimization on sets when it comes to searching – no storage of duplicate elements etc.

```
%%timeit -n1 -r5
myList = list(range(1000))
for i in range(1000):
    i in myList
```

```
10 ms ± 742 µs per loop (mean ± std. dev. of 5 runs, 1 loop each)
```

```
%%timeit -n1 -r5
myTuple = tuple(range(1000))
for i in range(1000):
    i in myTuple
```

```
8.54 ms ± 351 µs per loop (mean ± std. dev. of 5 runs, 1 loop each)
```

```
%%timeit -n1 -r5
mySet = set(range(1000))
for i in range(1000):
    i in mySet
```

```
76.6 µs ± 8.56 µs per loop (mean ± std. dev. of 5 runs, 1 loop each)
```

- Observe that in the above example, functions ***set(), tuples()*** and ***set()*** are used to create corresponding data types that contain a range of elements from 0 to 999. Moreover, inside the ***for*** loop, each value of i is searched in the corresponding data type. It is not printed whether it is found or not because it will flood the screen with output.

Exercise:

Attempt different functions avilable for sets and findout the output of the following:

```
A = {1, 2, 3, 4, 5, 6}
B = {4, 5, 6, 7, 8, 9}
A.intersection_update(B)
print(A)
A.union(B)
print(A)
```