Tuples

- A tuple is an ordered collection of elements/entries.
- Objects in a Tuple are immutable i.e. they cannot be altered once created.
- In a tuple, elements are placed within parentheses, separated by commas.
- The elements in a tuple can be of different data types like integer, list, etc.

How To Create A Tuple?

- A variable is simply assigned to a set of comma-separated values within closed parentheses.

```
>>> a=(1,2)
>>> b=("eat","dinner","man","boy")
>>> print(a)
(1,2)
>>> print(type(a))
<class 'tuple'>
```

- "()" parentheses are optional. If we assign a variable to a set of comma-separated values without parentheses, then by default, Python interprets it as a Tuple; This way of creating a Tuple is called **Tuple Packing**. It is always a good practice to use parentheses. This is shown in the given code snippet:

```
>>> c= 1,2,3,4,5
>>> print(c)
(1,2,3,4,5)
>>> print(type(c))
<class 'tuple'>
```

Note: If we assign a set of comma separated elements (or objects) to a set of comma separated variables, then these variables are assigned the corresponding values. **For Example:**

```
>>> e, f= "boy","man"
>>> print(e)
boy
>>> print(f)
man
```

Creating an Empty Tuple

In order to create an empty Tuple, simply assign a closed parentheses , which doesn't contain any element within, to a variable. This is shown below:

```
a=()
b=()
#Here 'a' and 'b' are empty tuples.
```

Creating A Tuple With A Single Element

- Creating a tuple with a single element is slightly complicated. If you try to create such a tuple by putting a single element within the parentheses, then it would not be a tuple, it would lead to the assignment of a single element to a variable.

```
>>> a=("Hamburger")
>>> print(type(a))
<class 'str'>
```

Here **'a'**, has a type **'string'**.

- If you try to create a tuple using the keyword `tuple`, you will get:

```
>>> tuple("Hamburger")
('H', 'a', 'm', 'b', 'u', 'r', 'g', 'e', 'r')
```

- To create such a tuple, along with a single element inside parentheses, we need a trailing comma. This would tell the system that it is in fact, a tuple.

```
>>> a=("Hamburger",)
>>> print(type(a))
<class 'tuple>
#Here a is indeed a tuple. Thus, the trailing comma is important in such
an assignment.
```

Difference Between A Tuple and A List

Tuple	**List**
A Tuple is immutable	A list is mutable
We cannot change the elements of a Tuple once it is created	We can change the elements of a list once it is created
Enclosed within parentheses “( )”	Enclosed within square brackets “[ ]”

Accessing Elements of a Tuple

Indexing in a Tuple

- Indexing in a Tuple starts from index **0**.
- The highest index is the `NumberOfElementsInTheTuple-1`.
- So a tuple having **10** elements would have indexing from **0-9**.
- This system is just like the one followed in Lists.

Negative Indexing

- Python language allows negative indexing for its tuple elements.
- The last element in the tuple has an index **-1**, the second last element has index **-2**, and so on.

Let us define a Tuple:

```
myTemp=(1,15,13,18,19,23,4,5,2,3) #Number of elements=10
```

Element	1	15	13	18	19	23	4	5	2	3
Index	0	1	2	3	4	5	6	7	8	9

Negative indexing	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

```
>>> print(myTemp[0])
1
>>> print(myTemp[8])
2
>>> print(myTemp[-1])
3
```

If any index out of this range is tried to be accessed, then it would give the following error.

```
>>> print(myTemp[10])
IndexError: tuple index out of range
>>> print(myTemp[20])
IndexError: tuple index out of range
>>> print(myTemp[-100])
IndexError: tuple index out of range
```

Slicing of a Tuple

- We can slice a Tuple and then access any required range of elements. Slicing is done by the means of a slicing operator which is “:”.
- The basic format of slicing is:

```
<Name of Tuple>[start index: end index]
```

- This format by default considers the end index to be `endIndex-1`

```
>>> print(myTemp[1:4])
(15,13,18)
>>> print(myTemp[7:])
(5,2,3)
```

What Changes Can Be Made To A Tuple?

A tuple is immutable. This means that the objects or elements in a tuple cannot be changed once a tuple has been initialized. This is contrary to the case of lists - as a list is mutable, and hence the elements can be changed there.

- Python enables us to reassign the variables to different tuples.
- **For example**, the variable `myTemp` which has been assigned a tuple. To this variable some other tuple can be reassigned.

```
>>> myTemp = ([6,7],"Amity", 2, "boy")
```

- However, you cannot change any single element in the tuple.

```
>>> myTemp[2] = "Amity"
TypeError: 'tuple' object does not support item assignment
```

- Though, you can change items from any mutable object in the tuple.

```
>>> myTemp[0][1] = "Amity"
>>> print(myTemp)
([6,"Amity"],"Amity", 2, "boy")
```

- If you want to delete a tuple entirely, such an operation is possible.
- The keyword used is `del`

```
>>> del(myTemp)
>>> print(myTemp)
NameError: name 'myTemp' is not defined
# The name error shows that the tuple 'myTemp' has been deleted
```