Accessing Elements - tuples

Let us see details related to operations on tuples. Unlike other collections (`list`, `set`, `dict`) we have limited functions with `tuple` in Python.

- `tuple` is by definition immutable and hence we will not be able to add elements to a tuple or delete elements from a tuple.
- Only functions that are available are `count` and `index`.
- `count` gives number of times an element is repeated in a tuple.
- `index` returns the position of element in a tuple. `index` can take up to 3 arguments - `element`, `start` and `stop`.

```
t = (1, 2, 3, 4, 4, 6, 1, 2, 3)
```

```
help(t)
```

```
Help on tuple object:

class tuple(object)
 |  tuple() -> empty tuple
 |  tuple(iterable) -> tuple initialized from iterable's items
 |
 |  If the argument is a tuple, the return value is the same object.
 |
 |  Methods defined here:
 |
 |  __add__(self, value, /)
 |      Return self+value.
 |
 |  __contains__(self, key, /)
 |      Return key in self.
 |
 |  __eq__(self, value, /)
 |      Return self==value.
 |
 |  __ge__(self, value, /)
 |      Return self>=value.
 |
 |  __getattribute__(self, name, /)
 |      Return getattr(self, name).
 |
 |  __getitem__(self, key, /)
 |      Return self[key].
 |
 |  __getnewargs__(...)
 |
 |  __gt__(self, value, /)
 |      Return self>value.
 |
 |  __hash__(self, /)
 |      Return hash(self).
 |
 |  __iter__(self, /)
 |      Implement iter(self).
 |
 |  __le__(self, value, /)
 |      Return self<=value.
 |
 |  __len__(self, /)
 |      Return len(self).
 |
 |  __lt__(self, value, /)
 |      Return self<value.
 |
 |  __mul__(self, value, /)
 |      Return self*value.
 |
 |  __ne__(self, value, /)
 |      Return self!=value.
 |
 |  __new__(*args, **kwargs) from builtins.type
 |      Create and return a new object.  See help(type) for accurate signature.
 |
 |  __repr__(self, /)
 |      Return repr(self).
 |
 |  __rmul__(self, value, /)
 |      Return value*self.
 |
 |  count(...)
 |      T.count(value) -> integer -- return number of occurrences of value
 |
 |  index(...)
 |      T.index(value, [start, [stop]]) -> integer -- return first index of value.
 |      Raises ValueError if the value is not present.
```

```
t.count?
```

```
Docstring: T.count(value) -> integer -- return number of occurrences of value
Type:      builtin_function_or_method
```

```
t.count(4)
```

```
2
```

```
t.count(9)
```

```
0
```

```
t.index?
```

```
Docstring:
T.index(value, [start, [stop]]) -> integer -- return first index of value.
Raises ValueError if the value is not present.
Type:      builtin_function_or_method
```

```
t.index(2) # Scans all the elements
```

```
1
```

```
t.index(2, 3) # Scans all the elements starting from 4th
```

```
7
```

```
t.index(2, 3, 5) # throws ValueError, scans from 4th element till 5th element
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-10-abe9594d5c59> in <module>
----> 1 t.index(2, 3, 5) # throws ValueError, scans from 4th element till 5th element

ValueError: tuple.index(x): x not in tuple
```

```
t.index(9)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-11-c3fb968f2496> in <module>
----> 1 t.index(9)

ValueError: tuple.index(x): x not in tuple
```

```
t.index(6, 3, 5) # throws ValueError, scans from 4th element till 5th element
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-12-0d1f64a89a67> in <module>
----> 1 t.index(6, 3, 5) # throws ValueError, scans from 4th element till 5th element

ValueError: tuple.index(x): x not in tuple
```

```
t.index(6, 3, 6)
```

```
5
```