😊 Tuples with Python

A **tuple** in Python is an ordered, immutable collection of elements. It is similar to a list but cannot be modified after creation. Tuples are commonly used when data should remain unchanged.

Characteristics of Tuples

1. **Ordered** – Elements maintain their insertion order.
2. **Immutable** – Once created, elements cannot be modified, added, or removed.
3. **Heterogeneous** – Can store different data types like integers, strings, and other tuples.
4. **Allow Duplicates** – The same value can appear multiple times.

Basic Tuple Operations

- **Creating a Tuple**: Defined using parentheses () or the tuple() function.
- **Accessing Elements**: Elements are accessed using indexing (zero-based).
- **Tuple Unpacking**: Values can be assigned to multiple variables at once.
- **Concatenation and Repetition**: Tuples can be combined using + and repeated using *.
- **Iterating Through a Tuple**: Loops can be used to traverse elements.

Common Tuple Methods

- count(value) – Counts occurrences of a value.
- index(value) – Returns the first index of a value.

Tuples are useful for fixed collections of data where immutability ensures data integrity.

To begin with the Lab

1. In Python, tuples are very similar to lists; however, unlike lists, they are *immutable*, meaning they cannot be changed. You would use tuples to present things that shouldn't be changed, such as days of the week or dates on a calendar.
2. The construction of a tuple use () with elements separated by commas.

```
[1]: # Create a tuple
     t = (1,2,3)
```

```
[2]: # Check len just like a list
     len(t)
```

```
[2]: 3
```

```
[3]: # Can also mix object types
     t = ('one',2)

     # Show
     t
```

```
[3]: ('one', 2)
```

```
[4]: # Use indexing just like we did in lists
     t[0]
```

```
[4]: 'one'
```

```
[5]: # Slicing just like a list
     t[-1]
```

```
[5]: 2
```

3. Tuples have built-in methods, but not as many as lists do. Below are some examples.

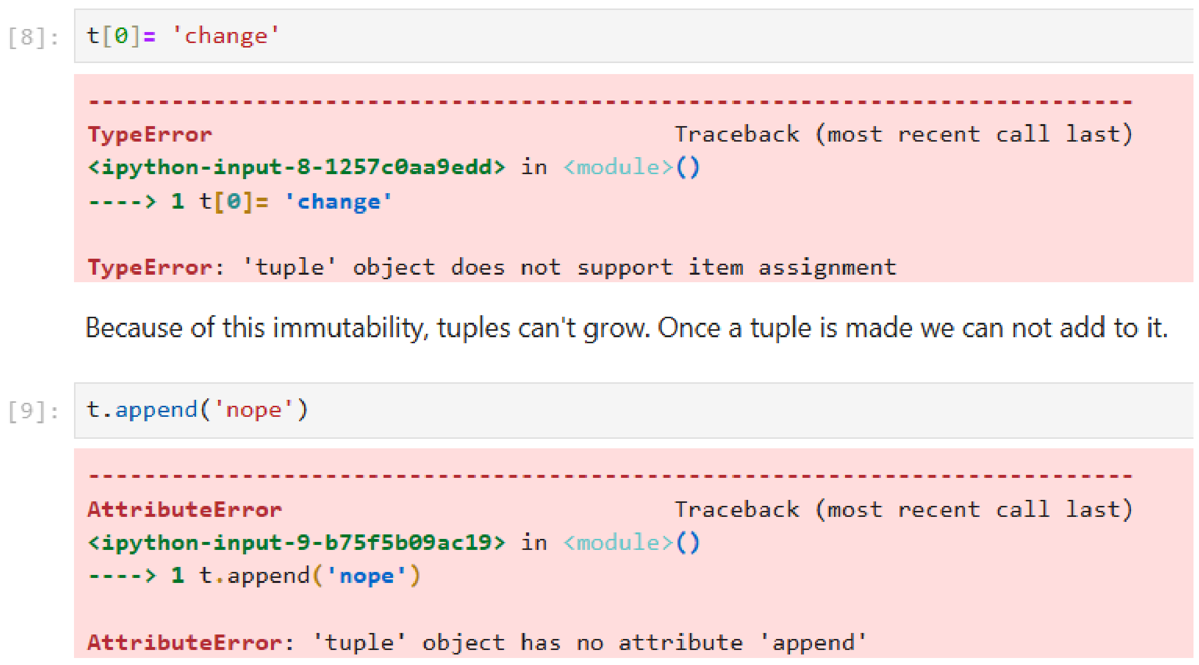
```
[6]: # Use .index to enter a value and return the index
     t.index('one')
```

```
[6]: 0
```

```
[7]: # Use .count to count the number of times a value appears
     t.count('one')
```

```
[7]: 1
```

4. The fact that tuples are unchangeable cannot be emphasized enough. To emphasize that point.



```
[8]: t[0]= 'change'
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-8-1257c0aa9edd> in <module>()
----> 1 t[0]= 'change'

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

Because of this immutability, tuples can't grow. Once a tuple is made we can not add to it.

```
[9]: t.append('nope')
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-9-b75f5b09ac19> in <module>()
----> 1 t.append('nope')

AttributeError: 'tuple' object has no attribute 'append'
```

5. "Why bother using tuples when they have fewer available methods?" you might be asking yourself. Though they are utilized when immutability is required, tuples are really less common in programming than lists. A tuple is your answer if you are passing around an item in your program and need to ensure that it doesn't get altered. It offers a practical way to ensure data integrity.
6. You should now understand the immutability of tuples and be able to generate and use them in your programming.