**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**  
**Topic 20 - Tuples**  
**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**What Is a Tuple?**

A tuple (pronounced either "toople" or "tupple") in Python is similar to a list but with a key difference: **tuples are immutable**. This means that once you create a tuple, you can’t change, add, or remove any elements from it without redefining the entire tuple.

Tuples are ideal for storing collections of data that should remain constant. For example, the order of the first U.S. states founded is fixed and doesn’t need modifications, so it’s an excellent candidate for a tuple.

**Example of a Tuple for State Order**

states\_in\_order\_of\_founding = ("Delaware", "Pennsylvania", "New Jersey", "Georgia")

In this tuple:

* states\_in\_order\_of\_founding[0] is "Delaware"
* states\_in\_order\_of\_founding[1] is "Pennsylvania"
* states\_in\_order\_of\_founding[2] is "New Jersey"
* states\_in\_order\_of\_founding[3] is "Georgia"

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Why Use a Tuple Instead of a List?**

1. **Immutability**:  
   Tuples are fixed and can’t be altered, so they’re perfect for data that should remain constant.
2. **Simplicity**:  
   For collections where you know the data won’t change, tuples help communicate that to anyone reading your code.
3. **Efficiency**:  
   Python optimizes tuples for faster processing since they’re immutable, making them slightly faster than lists in certain cases.
4. **Safety**:  
   Because tuples can’t be changed, they protect data from accidental modification, which is helpful in larger programs.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**How to Use Tuples**

**1. Defining a Tuple**

The syntax for a tuple is almost the same as for a list, except you use parentheses () instead of square brackets [].

**Example of Defining a Tuple**:

states\_in\_order\_of\_founding = ("Delaware", "Pennsylvania", "New Jersey", "Georgia")

**2. Accessing Tuple Elements**

Access elements in a tuple by using the index, just like with lists. Remember that indexing starts at 0.

**Example**:

second\_state\_founded = states\_in\_order\_of\_founding[1]

print("The second state founded was " + second\_state\_founded)

# Output: The second state founded was Pennsylvania

**3. Characteristics of Tuples**

* **Fixed Elements**:  
  Once defined, you can’t change the elements in a tuple. If the data needs modification, redefine the entire tuple.
* **Index-Based Access**:  
  Like lists, tuples support indexing to access specific elements.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Limitations of Tuples**

Unlike lists, you **cannot**:

* Add elements to a tuple.
* Remove elements from a tuple.
* Modify elements within a tuple.
* Delete or pop elements.

If you need to change any data in a tuple, you must redefine it entirely.

**Example of Redefining a Tuple**

Suppose Pennsylvania changes its name to “Taylorswiftsylvania.” You’ll need to create a new tuple to reflect this change:

states\_in\_order\_of\_founding = ("Delaware", "Taylorswiftsylvania formerly known as Pennsylvania", "New Jersey", "Georgia")

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Important Points to Remember**

1. **Fixed Structure**:  
   Tuples are fixed collections of elements, which makes them useful when data should not change.
2. **Defining Syntax**:  
   Use parentheses () instead of square brackets [] for tuples.
3. **Re-indexing Requirement**:  
   If you redefine a tuple, remember that the new tuple will restart its indexing from 0.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Examples of Using Tuples**

1. **Defining a Tuple**:

colors = ("red", "green", "blue")

1. **Accessing Tuple Elements**:

first\_color = colors[0]

# Output: first\_color = "red"

1. **Redefining a Tuple**:

colors = ("red", "green", "blue", "yellow") # Adds "yellow" by redefining the tuple

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Summary**

Tuples are immutable collections, ideal for holding data that doesn’t change. They share many similarities with lists but restrict operations to ensure the data remains constant. Tuples are fast, reliable, and a good choice for storing fixed sequences or unchanging datasets in Python.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**