Python Tuples

- Tuples are used to store multiple items in a single variable
- Tuple is one of 4 built-in data types in Python used to store collections of data, the other 3 are List, Set, and Dictionary, all with different qualities and usage
- Tuples are written with round brackets ( )
- Example
  - Create a Tuple

In [1]:
```
thistuple = ("apple", "banana", "cherry")
print(thistuple)
```

```
('apple', 'banana', 'cherry')
```

Tuple Items

- Tuple items are ordered, unchangeable, and allow duplicate values
- Tuple items are indexed, the first item has index [0], the second item has index [1] etc.

Ordered

- When we say that tuples are ordered, it means that the items have a defined order, and that order will not change

Unchangeable

- Tuples are unchangeable, meaning that we cannot change, add or remove items after the tuple has been created

Allow Duplicates

- Since tuples are indexed, they can have items with the same value
- Example
  - Tuples allow duplicate values

In [2]:
```
thistuple = ("apple", "banana", "cherry", "apple", "cherry")
print(thistuple)
```

```
('apple', 'banana', 'cherry', 'apple', 'cherry')
```

Tuple Length

- To determine how many items a tuple has, use the len() function
- Example
  - Print the number of items in the tuple

In [4]:
```
thistuple = ("apple", "banana", "cherry")
print(len(thistuple))
```

```
3
```

Create Tuple With One Item

- To create a tuple with only one item, you have to add a comma after the item, otherwise Python will not recognize it as a tuple
- Example
  - One item tuple, remember the comma

In [7]:
```
thistuple = ("apple",)
print(type(thistuple))

#NOT a tuple
thistuple = ("apple")
print(type(thistuple))
```

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

Tuple Items - Data Types

- Tuple items can be of any data type
- Example
  - String, int and boolean data types

In [8]:
```
tuple1 = ("apple", "banana", "cherry")
tuple2 = (1, 5, 7, 9, 3)
tuple3 = (True, False, False)

print(tuple1)
print(tuple2)
print(tuple3)
```

```
('apple', 'banana', 'cherry')
(1, 5, 7, 9, 3)
(True, False, False)
```

- A tuple can contain different data types

- Example
  - A tuple with strings, integers and boolean values

In [9]:
```
tuple1 = ("abc", 34, True, 40, "male")

print(tuple1)
```

```
('abc', 34, True, 40, 'male')
```

The tuple() Constructor

- It is also possible to use the tuple() constructor to make a tuple
- Example
  - Using the tuple() method to make a tuple

In [10]:
```
thistuple = tuple(("apple", "banana", "cherry")) # note the double round-
print(thistuple)
```

```
('apple', 'banana', 'cherry')
```

Access Tuple Items

- You can access tuple items by referring to the index number, inside square brackets [ ]
- Example
  - Print the second item in the tuple

In [12]:
```
thistuple = ("apple", "banana", "cherry")
print(thistuple[1])
```

```
banana
```

- Note: The first item has index 0.

Negative Indexing

- Negative indexing means start from the end
- -1 refers to the last item, -2 refers to the second last item etc
- Example
  - Print the last item of the tuple

In [11]:
```
thistuple = ("apple", "banana", "cherry")
print(thistuple[-1])
```

```
cherry
```

Range of Indexes

- You can specify a range of indexes by specifying where to start and where to end the range
- When specifying a range, the return value will be a new tuple with the specified items
- Example
  - Return the third, fourth, and fifth item

In [13]:
```
thistuple = ("apple", "banana", "cherry", "orange", "kiwi", "melon", "man
print(thistuple[2:5])
```

```
('cherry', 'orange', 'kiwi')
```

- Note: The search will start at index 2 (included) and end at index 5 (not included).
- Remember that the first item has index 0

- By leaving out the start value, the range will start at the first item
- Example
  - This example returns the items from the beginning to, but NOT included, "kiwi"

In [14]:
```
thistuple = ("apple", "banana", "cherry", "orange", "kiwi", "melon", "man
print(thistuple[:4])
```

```
('apple', 'banana', 'cherry', 'orange')
```

- By leaving out the end value, the range will go on to the end of the list
- Example
  - This example returns the items from "cherry" and to the end

In [15]:
```
thistuple = ("apple", "banana", "cherry", "orange", "kiwi", "melon", "man
print(thistuple[2:])
```

```
('cherry', 'orange', 'kiwi', 'melon', 'mango')
```

Range of Negative Indexes

- Specify negative indexes if you want to start the search from the end of the tuple
- Example
  - This example returns the items from index -4 (included) to index -1 (excluded)

In [16]:
```
thistuple = ("apple", "banana", "cherry", "orange", "kiwi", "melon", "man
print(thistuple[-4:-1])
```

```
('orange', 'kiwi', 'melon')
```

Check if Item Exists

- To determine if a specified item is present in a tuple use the in keyword
- Example
  - Check if "apple" is present in the tuple

In [18]:
```
thistuple = ("apple", "banana", "cherry")
if "apple" in thistuple:
  print("Yes, 'apple' is in the fruits tuple")
```

```
Yes, 'apple' is in the fruits tuple
```

Update Tuples

- Tuples are unchangeable, meaning that you cannot change, add, or remove items once the tuple is created
- But there are some workarounds

Change Tuple Values

- Once a tuple is created, you cannot change its values. Tuples are **unchangeable**, or **immutable** as it also is called
- But there is a workaround. You can convert the tuple into a list, change the list, and convert the list back into a tuple
- Example
  - Convert the tuple into a list to be able to change it

In [19]:
```
x = ("apple", "banana", "cherry")
y = list(x)
y[1] = "kiwi"
x = tuple(y)

print(x)
```

```
('apple', 'kiwi', 'cherry')
```

Add Items

- Since tuples are immutable, they do not have a built-in append() method, but there are other ways to add items to a tuple

- 1.Convert into a list: Just like the workaround for changing a tuple, you can convert it into a list, add your item(s), and convert it back into a tuple.
- Example
  - Convert the tuple into a list, add "orange", and convert it back into a tuple

In [21]:
```
thistuple = ("apple", "banana", "cherry")
y = list(thistuple)
y.append("orange")
thistuple = tuple(y)

print(thistuple)
```

```
('apple', 'banana', 'cherry', 'orange')
```

- 2.Add tuple to a tuple. You are allowed to add tuples to tuples, so if you want to add one item, (or many), create a new tuple with the item(s), and add it to the existing tuple
- Example
  - Create a new tuple with the value "orange", and add that tuple

In [22]:
```
thistuple = ("apple", "banana", "cherry")
y = ("orange",)
thistuple += y

print(thistuple)
```

```
('apple', 'banana', 'cherry', 'orange')
```

Remove Items

- Tuples are **unchangeable**, so you cannot remove items from it, but you can use the same workaround as we used for changing and adding tuple items
- Example
  - Convert the tuple into a list, remove "apple", and convert it back into a tuple

In [1]:
```
thistuple = ("apple", "banana", "cherry")
y = list(thistuple)
y.remove("apple")
thistuple = tuple(y)
print(thistuple)
```

```
('banana', 'cherry')
```

- Or you can delete the tuple completely
- Example
  - The del keyword can delete the tuple completely

In [24]:
```
thistuple = ("apple", "banana", "cherry")
del thistuple
print(thistuple) #this will raise an error because the tuple no longer ex
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[24], line 3
      1 thistuple = ("apple", "banana", "cherry")
      2 del thistuple
----> 3 print(thistuple) #this will raise an error because the tuple no longer exists

NameError: name 'thistuple' is not defined
```

Unpack Tuples

- When we create a tuple, we normally assign values to it. This is called "packing" a tuple
- Example
  - Packing a tuple

In [25]:
```
fruits = ("apple", "banana", "cherry")
print(fruits)
```

```
('apple', 'banana', 'cherry')
```

- But, in Python, we are also allowed to extract the values back into variables. This is called "unpacking"
- Example
  - Unpacking a tuple

In [26]:
```
fruits = ("apple", "banana", "cherry")

(green, yellow, red) = fruits

print(green)
print(yellow)
print(red)
```

```
apple
banana
cherry
```

- Note: The number of variables must match the number of values in the tuple, if not, you must use an asterisk to collect the remaining values as a list

Using Asterisk *

- If the number of variables is less than the number of values, you can add an * to the variable name and the values will be assigned to the variable as a list

- Example
  - Assign the rest of the values as a list called "red"

In [27]:
```
fruits = ("apple", "banana", "cherry", "strawberry", "raspberry")

(green, yellow, *red) = fruits

print(green)
print(yellow)
print(red)
```

```
apple
banana
['cherry', 'strawberry', 'raspberry']
```

- If the asterisk is added to another variable name than the last, Python will assign values to the variable until the number of values left matches the number of variables left
- Example
  - Add a list of values the "tropic" variable

In [28]:
```
fruits = ("apple", "mango", "papaya", "pineapple", "cherry")

(green, *tropic, red) = fruits

print(green)
print(tropic)
print(red)
```

```
apple
['mango', 'papaya', 'pineapple']
cherry
```

Join Two Tuples

- To join two or more tuples you can use the + operator
- Example
  - Join two tuples

In [29]:
```
tuple1 = ("a", "b" , "c")
tuple2 = (1, 2, 3)

tuple3 = tuple1 + tuple2
print(tuple3)
```

```
('a', 'b', 'c', 1, 2, 3)
```

Multiply Tuples

- If you want to multiply the content of a tuple a given number of times, you can use the * operator

- Example
  - Multiply the fruits tuple by 2

In [30]:
```
fruits = ("apple", "banana", "cherry")
mytuple = fruits * 2

print(mytuple)
```

```
('apple', 'banana', 'cherry', 'apple', 'banana', 'cherry')
```

Tuple Methods

- Python has **two** built-in methods that you can use on tuples

- 1.count()
- The count() method returns the number of times a specified value appears in the tuple
- Syntax - The tuple.count(value)
  - value - Required. The item to search for
- Example
  - Return the number of times the value 5 appears in the tuple

In [31]:
```
thistuple = (1, 3, 7, 8, 7, 5, 4, 6, 8, 5)

x = thistuple.count(5)

print(x)
```

```
2
```

- 2.index()
- The index() method finds the first occurrence of the specified value
- The index() method raises an exception if the value is not found
- Syntax - The tuple.index(value)
  - value - Required. The item to search for
- Example
  - Search for the first occurrence of the value 8, and return its position

In [32]:
```
thistuple = (1, 3, 7, 8, 7, 5, 4, 6, 8, 5)

x = thistuple.index(8)

print(x)
```

```
3
```