LECTURE 2

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.

A tuple is a collection which is ordered and unchangeable.

Tuples are written with round brackets.

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.

Tuple Length

To determine how many items a tuple has, use the len() function:

In [2]:
```
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.

In [3]:
```
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:

In [4]:
```
tuple1 = ("apple", "banana", "cherry")
tuple2 = (1, 5, 7, 9, 3)
tuple3 = (True, False, False)
```

A tuple can contain different data types:

In [5]:
```
tuple1 = ("abc", 34, True, 40, "male")
```

The tuple() Constructor

It is also possible to use the tuple() constructor to make a tuple.

In [6]:
```
thistuple = tuple(("apple", "banana", "cherry")) # note the double round-brackets
print(thistuple)
```

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

Access Tuple Items

You can access tuple items by referring to the index number, inside square brackets:

In [7]:
```
thistuple = ("apple", "banana", "cherry")
print(thistuple[1])
```

```
banana
```

Negative indexing means start from the end.

-1 refers to the last item, -2 refers to the second last item etc.

In [8]:
```
thistuple = ("apple", "banana", "cherry")
print(thistuple[-1])
```

```
cherry
```

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.

In [9]:
```
thistuple = ("apple", "banana", "cherry", "orange", "kiwi", "melon", "mango")
print(thistuple[2:5])
```

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

In [10]:
```
thistuple = ("apple", "banana", "cherry", "orange", "kiwi", "melon", "mango")
print(thistuple[:4])
```

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

In [11]:
```
thistuple = ("apple", "banana", "cherry", "orange", "kiwi", "melon", "mango")
print(thistuple[2:])
```

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

Check if Item Exists

To determine if a specified item is present in a tuple use the in keyword:

In [12]:
```
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.

In [14]:
```
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.

In [16]:
```
thistuple = ("apple", "banana", "cherry")
y = list(thistuple)
y.append("orange")
thistuple = tuple(y)
print(thistuple)
```

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

1. 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:

In [17]:
```
thistuple = ("apple", "banana", "cherry")
y = ("orange",)
thistuple += y

print(thistuple)
```

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

Unpacking a Tuple

When we create a tuple, we normally assign values to it. This is called "packing" a tuple. But, in Python, we are also allowed to extract the values back into variables. This is called "unpacking":

In [19]:
```
fruits = ("apple", "banana", "cherry")

(g, y, r) = fruits

print(g)
print(y)
print(r)
```

```
apple
banana
cherry
```

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:

In [20]:
```
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.

In [21]:
```
fruits = ("apple", "mango", "papaya", "pineapple", "cherry")

(green, *tropic, red) = fruits

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

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

Loop Through a Tuple

You can loop through the tuple items by using a for loop.

In [22]:
```
thistuple = ("apple", "banana", "cherry")
for x in thistuple:
  print(x)
```

```
apple
banana
cherry
```

Loop Through the Index Numbers

You can also loop through the tuple items by referring to their index number.

Use the range() and len() functions to create a suitable iterable.

In [23]:
```
thistuple = ("apple", "banana", "cherry")
for i in range(len(thistuple)):
  print(thistuple[i])
```

```
apple
banana
cherry
```

Python Sets

Sets are used to store multiple items in a single variable.

Set is one of 4 built-in data types in Python used to store collections of data, the other 3 are List, Tuple, and Dictionary, all with different qualities and usage.

A set is a collection which is unordered, unchangeable, and unindexed.

Sets are written with curly brackets.

In [24]:
```
thisset = {"apple", "banana", "cherry"}
print(thisset)
```

```
{'apple', 'banana', 'cherry'}
```

Set Items

Set items are unordered, unchangeable, and do not allow duplicate values.

Unordered

Unordered means that the items in a set do not have a defined order.

Set items can appear in a different order every time you use them, and cannot be referred to by index or key.

Unchangeable

Set items are unchangeable, meaning that we cannot change the items after the set has been created.

Duplicates Not Allowed

Sets cannot have two items with the same value.

In [25]:
```
thisset = {"apple", "banana", "cherry", "apple"}
print(thisset)
```

```
{'apple', 'banana', 'cherry'}
```

True and 1 is considered the same value (False and 0 is considered the same value):

In [27]:
```
thisset = {"apple", "banana", "cherry", True, 1, 2}
print(thisset)

thisset = {"apple", "banana", "cherry", False, True, 0}
print(thisset)
```

```
{True, 2, 'banana', 'apple', 'cherry'}
{False, True, 'banana', 'apple', 'cherry'}
```

Get the Length of a Set

To determine how many items a set has, use the len() function.

In [28]:
```
thisset = {"apple", "banana", "cherry"}
print(len(thisset))
```

```
3
```

Access Items

You cannot access items in a set by referring to an index or a key.

But you can loop through the set items using a for loop, or ask if a specified value is present in a set, by using the in keyword.

In [29]:
```
thisset = {"apple", "banana", "cherry"}

for x in thisset:
  print(x)
```

```
apple
banana
cherry
```

In [30]:
```
thisset = {"apple", "banana", "cherry"}

print("banana" in thisset)
```

```
True
```

Add Items

To add one item to a set use the add() method.

In [31]:
```
thisset = {"apple", "banana", "cherry"}

thisset.add("orange")

print(thisset)
```

```
{'apple', 'banana', 'cherry', 'orange'}
```

To add items from another set into the current set, use the update() method.

In [32]:
```
thisset = {"apple", "banana", "cherry"}
tropical = {"pineapple", "mango", "papaya"}

thisset.update(tropical)

print(thisset)
```

```
{'banana', 'papaya', 'mango', 'apple', 'cherry', 'pineapple'}
```

The object in the update() method does not have to be a set, it can be any iterable object (tuples, lists, dictionaries etc.).

In [33]:
```
thisset = {"apple", "banana", "cherry"}
mylist = ["kiwi", "orange"]

thisset.update(mylist)

print(thisset)
```

```
{'orange', 'cherry', 'banana', 'apple', 'kiwi'}
```

Remove Item

To remove an item in a set, use the remove(), or the discard() method.

If the item to remove does not exist, remove() will raise an error.

If the item to remove does not exist, discard() will NOT raise an error.

In [34]:
```
thisset = {"apple", "banana", "cherry"}

thisset.remove("banana")
```

```
print(thisset)
```

```
{'apple', 'cherry'}
```

In [35]:
```
thisset = {"apple", "banana", "cherry"}

thisset.discard("banana")

print(thisset)
```

```
{'apple', 'cherry'}
```

You can also use the pop() method to remove an item, but this method will remove a random item, so you cannot be sure what item that gets removed.

The return value of the pop() method is the removed item.

In [39]:
```
thisset = {"apple", "banana", "cherry"}

x = thisset.pop()

print(x)

print(thisset)
```

```
apple
{'banana', 'cherry'}
```

The clear() method empties the set:

In [40]:
```
thisset = {"apple", "banana", "cherry"}

thisset.clear()

print(thisset)
```

```
set()
```

The del keyword will delete the set completely:

In [41]:
```
thisset = {"apple", "banana", "cherry"}

del thisset

print(thisset)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[41], line 5
      1 thisset = {"apple", "banana", "cherry"}
      3 del thisset
----> 5 print(thisset)

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

Loop Items

You can loop through the set items by using a for loop:

In [43]:
```
thisset = {"apple", "banana", "cherry"}

for x in thisset:
  print(x)
```

```
apple
banana
cherry
```

In [ ]:
```
```