Python Lists

- Lists are used to store multiple items in a single variable
- Lists are one of 4 built-in data types in Python used to store collections of data, the other 3 are Tuple, Set, and Dictionary, all with different qualities and usage
- Lists are created using square brackets [ ]
- Example
  - Create a List

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

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

List Items

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

Ordered

- When we say that lists are ordered, it means that the items have a defined order, and that order will not change.
- If you add new items to a list, the new items will be placed at the end of the list

Changeable

- The list is changeable, meaning that we can change, add, and remove items in a list after it has been created

Allow Duplicates

- Since lists are indexed, lists can have items with the same value
- Example

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

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

List Length

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

In [3]:
```
thislist = ["apple", "banana", "cherry"]
print(len(thislist))
```

```
3
```

List Items - Data Types

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

In [4]:
```
list1 = ["apple", "banana", "cherry"]
list2 = [1, 5, 7, 9, 3]
list3 = [True, False, False]

print(list1)
print(list2)
print(list3)
```

```
['apple', 'banana', 'cherry']
[1, 5, 7, 9, 3]
[True, False, False]
```

- A list can contain different data types
- Example
  - A list with strings, integers and boolean values

In [5]:
```
list1 = ["abc", 34, True, 40, "male"]
print(list1)
```

```
['abc', 34, True, 40, 'male']
```

The list() Constructor

- It is also possible to use the list() constructor when creating a new list
- Example
  - Using the list() constructor to make a List

In [6]:
```
thislist = list(("apple", "banana", "cherry")) # note the double round-br
print(thislist)
```

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

Python Collections (Arrays)

- There are four collection data types in the Python programming language:
- List is a collection which is ordered and changeable. Allows duplicate members.
- Tuple is a collection which is ordered and unchangeable. Allows duplicate members.
- Set is a collection which is unordered, unchangeable*, and unindexed. No duplicate members.
- Dictionary is a collection which is ordered** and changeable. No duplicate members

- Note: Set items are unchangeable, but you can remove and/or add items whenever you like
- Note: As of Python version 3.7, dictionaries are ordered. In Python 3.6 and earlier, dictionaries are unordered

Access Items

- List items are indexed and you can access them by referring to the index number
- Example
  - Print the second item of the list

In [7]:
```
thislist = ["apple", "banana", "cherry"]
print(thislist[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 list

In [8]:
```
thislist = ["apple", "banana", "cherry"]
print(thislist[-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 list with the specified items
- Example
  - Return the third, fourth, and fifth item

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

```
['cherry', 'orange', 'kiwi']
```

- Note: The search will start at index 2 (included) and end at index 5 (not included)

- 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 including, "kiwi"

In [10]:
```
thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mang
print(thislist[: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" to the end

In [11]:
```
thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mang
print(thislist[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 list
- Example
  - This example returns the items from "orange" (-4) to, but NOT including "mango" (-1)

In [12]:
```
thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mang
print(thislist[-4:-1])
```

```
['orange', 'kiwi', 'melon']
```

Check if Item Exists

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

In [13]:
```
thislist = ["apple", "banana", "cherry"]
if "apple" in thislist:
  print("Yes, 'apple' is in the fruits list")
```

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

Change List Items

- To change the value of a specific item, refer to the index number
- Example
  - Change the second item

In [14]:
```
thislist = ["apple", "banana", "cherry"]
thislist[1] = "blackcurrant"
print(thislist)
```

```
['apple', 'blackcurrant', 'cherry']
```

Change a Range of Item Values

- To change the value of items within a specific range, define a list with the new values, and refer to the range of index numbers where you want to insert the new values
- Example
  - Change the values "banana" and "cherry" with the values "blackcurrant" and "watermelon"

In [1]:
```
thislist = ["apple", "banana", "cherry", "orange", "kiwi", "mango"]
thislist[1:3] = ["blackcurrant", "watermelon"]
print(thislist)
```

```
['apple', 'blackcurrant', 'watermelon', 'orange', 'kiwi', 'mango']
```

- If you insert more items than you replace, the new items will be inserted where you specified, and the remaining items will move accordingly
- Example
  - Change the second value by replacing it with two new values

In [15]:
```
thislist = ["apple", "banana", "cherry"]
thislist[1:2] = ["blackcurrant", "watermelon"]
```

```
print(thislist)
```

```
['apple', 'blackcurrant', 'watermelon', 'cherry']
```

- Note: The length of the list will change when the number of items inserted does not match the number of items replaced

- If you insert less items than you replace, the new items will be inserted where you specified, and the remaining items will move accordingly
- Example
  - Change the second and third value by replacing it with one value

In [16]:
```
thislist = ["apple", "banana", "cherry"]
thislist[1:3] = ["watermelon"]
print(thislist)
```

```
['apple', 'watermelon']
```

List Methods

- Python has a set of built-in methods that you can use on lists

- 1.append()
- The append() method appends an element to the end of the list
- Syntax - list.append(elmnt)
  - elmnt - Required. An element of any type (string, number, object etc.)
- Example
  - Add an element to the fruits list

In [1]:
```
fruits = ["apple", "banana", "cherry"]

fruits.append("orange")

print(fruits)
```

```
['apple', 'banana', 'cherry', 'orange']
```

- Example
  - Add a list to a list

In [2]:
```
a = ["apple", "banana", "cherry"]

b = ["Ford", "BMW", "Volvo"]

a.append(b)

print(a)
```

```
['apple', 'banana', 'cherry', ['Ford', 'BMW', 'Volvo']]
```

- 2.clear()
- The clear() method removes all the elements from a list
- Syntax - list.clear()
- No parameters
- Example
  - Remove all elements from the fruits list

In [5]:
```
fruits = ["apple", "banana", "cherry"]

fruits.clear()

print(fruits)
```

```
[]
```

- 3.copy()
- The copy() method returns a copy of the specified list
- Syntax - list.copy()
- No parameters
- Example
  - Copy the fruits list

In [1]:
```
fruits = ["apple", "banana", "cherry"]

x = fruits.copy()

print(x)
```

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

- 4.count()
- The count() method returns the number of elements with the specified value
- Syntax - list.count(value)
  - value - Required. Any type (string, number, list, tuple, etc.). The value to search for
- Example
  - Return the number of times the value "cherry" appears in the fruits list

In [2]:
```
fruits = ["apple", "banana", "cherry"]

x = fruits.count("cherry")

print(x)
```

```
1
```

- Example
  - Return the number of times the value 9 appears in the list

In [3]:
```
fruits = [1, 4, 2, 9, 7, 8, 9, 3, 1]

x = fruits.count(9)

print(x)
```

```
2
```

- 5.extend()
- The extend() method adds the specified list elements (or any iterable) to the end of the current list
- Syntax - list.extend(iterable)
  - iterable - Required. Any iterable (list, set, tuple, etc.)
- Example
  - Add the elements of cars to the fruits list

In [4]:
```
fruits = ['apple', 'banana', 'cherry']

cars = ['Ford', 'BMW', 'Volvo']

fruits.extend(cars)

print(fruits)
```

```
['apple', 'banana', 'cherry', 'Ford', 'BMW', 'Volvo']
```

- Example
  - Add a tuple to the fruits list

In [5]:
```
fruits = ['apple', 'banana', 'cherry']

points = (1, 4, 5, 9)

fruits.extend(points)

print(fruits)
```

```
['apple', 'banana', 'cherry', 1, 4, 5, 9]
```

- 6.index()
- The index() method returns the position at the first occurrence of the specified value
- Syntax - list.index(elmnt)
  - elmnt Required. Any type (string, number, list, etc.). The element to search for
- Example
  - What is the position of the value "cherry"

In [6]:
```
fruits = ['apple', 'banana', 'cherry']

x = fruits.index("cherry")
```

```
print(x)
```

```
2
```

- Example
  - What is the position of the value 32

In [7]:
```
fruits = [4, 55, 64, 32, 16, 32]

x = fruits.index(32)

print(x)
```

```
3
```

- 7.insert()
- The insert() method inserts the specified value at the specified position
- Syntax - list.insert(pos, elmnt)
  - pos - Required. A number specifying in which position to insert the value
  - elmnt - Required. An element of any type (string, number, object etc.)
- Example
  - Insert the value "orange" as the second element of the fruit list

In [8]:
```
fruits = ['apple', 'banana', 'cherry']

fruits.insert(1, "orange")

print(fruits)
```

```
['apple', 'orange', 'banana', 'cherry']
```

- 8.pop()
- The pop() method removes the element at the specified position
- Syntax - list.pop(pos)
  - pos Optional. A number specifying the position of the element you want to remove, default value is -1, which returns the last item
- Example
  - Remove the second element of the fruit list

In [9]:
```
fruits = ['apple', 'banana', 'cherry']

fruits.pop(1)

print(fruits)
```

```
['apple', 'cherry']
```

- Example
  - Return the removed element

In [10]:
```
fruits = ['apple', 'banana', 'cherry']

x = fruits.pop(1)

print(x)
```

```
banana
```

- Note: The pop() method returns removed value

- 9.remove()
- The remove() method removes the first occurrence of the element with the specified value
- Syntax - list.remove(elmnt)
  - elmnt - Required. Any type (string, number, list etc.) The element you want to remove
- Example
  - Remove the "banana" element of the fruit list

In [11]:
```
fruits = ['apple', 'banana', 'cherry']

fruits.remove("banana")

print(fruits)
```

```
['apple', 'cherry']
```

- 10.reverse()
- The reverse() method reverses the sorting order of the elements
- Syntax - list.reverse()
- No parameters
- Example
  - Reverse the order of the fruit list

In [12]:
```
fruits = ['apple', 'banana', 'cherry']

fruits.reverse()

print(fruits)
```

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

- 11.sort()
- The sort() method sorts the list ascending by default
- You can also make a function to decide the sorting criteria(s)
- Syntax - list.sort(reverse=True|False, key=myFunc)
  - reverse - Optional. reverse=True will sort the list descending. Default is reverse=False
  - key - Optional. A function to specify the sorting criteria(s)

- Example
  - Sort the list alphabetically

In [13]:
```
cars = ['Ford', 'BMW', 'Volvo']

cars.sort()

print(cars)
```

```
['BMW', 'Ford', 'Volvo']
```

- Example
  - Sort the list descending

In [14]:
```
cars = ['Ford', 'BMW', 'Volvo']

cars.sort(reverse=True)

print(cars)
```

```
['Volvo', 'Ford', 'BMW']
```

- Example
  - Sort the list by the length of the values

In [15]:
```
# A function that returns the length of the value:
def myFunc(e):
  return len(e)

cars = ['Ford', 'Mitsubishi', 'BMW', 'VW']

cars.sort(key=myFunc)

print(cars)
```

```
['VW', 'BMW', 'Ford', 'Mitsubishi']
```

- Example
  - Sort a list of dictionaries based on the "year" value of the dictionaries

In [16]:
```
def myFunc(e):
    return e['year']

cars = [
    {'car': 'Ford', 'year': 2005},
    {'car': 'Mitsubishi', 'year': 2000},
    {'car': 'BMW', 'year': 2019},
    {'car': 'VW', 'year': 2011}
]

cars.sort(key=myFunc)

print(cars)
```

```
[{'car': 'Mitsubishi', 'year': 2000}, {'car': 'Ford', 'year': 2005}, {'c
ar': 'VW', 'year': 2011}, {'car': 'BMW', 'year': 2019}]
```

- Example
  - Sort the list by the length of the values and reversed

In [17]:
```
# A function that returns the length of the value:
def myFunc(e):
  return len(e)

cars = ['Ford', 'Mitsubishi', 'BMW', 'VW']

cars.sort(reverse=True, key=myFunc)

print(cars)
```

```
['Mitsubishi', 'Ford', 'BMW', 'VW']
```