Google

Reading: Reference guide: Lists

You've been learning that lists are important data structures in Python. A list is a data structure that helps store and manipulate an ordered collection of items. These items can be of any data type such as integers, floats, strings, and even other lists. Because they are so versatile, data professionals and all Python programmers use lists every day, so it's important to be familiar with how they work. This reading is a reference guide for lists designed to help you as you learn Python.

Save this course item

You may want to save a copy of this guide for future reference. You can use it as a resource for additional practice or in your future professional projects. To access a downloadable version of this course item, click the following link and select "Use Template."

Reference guide: Lists

OR

If you don't have a Google account, you can download the item directly from the attachment below.

Reference guide: Lists

Create a list

There are two main ways to create lists in Python:

- Square brackets: **[]**
- The list function: **list()**

When instantiating a list using brackets, separate each element with a comma.

For example, the following code creates a list of strings:

```
list_a = ['olive', 'palm', 'coconut']
print(list_a)
```

Output:

```
['olive', 'palm', 'coconut']
```

You can also create a list of integers:

```
list_b = [8, 6, 7, 5, 3, 0, 8]
print(list_b)
```

Output:

```
[8, 6, 7, 5, 3, 0, 8]
```

Or a list of mixed data types:

```
list_c = ['Abidjan', 14.2, [1, 2, None], 'Zagreb']
print(list_c)
```

Output:

```
['Abidjan', 14.2, [1, 2, None], 'Zagreb']
```

To create an empty list, use empty brackets or the **list()** function:

```
empty_list_1 = []
empty_list_2 = list()
```

Output:

```
[]
```

Indexing and slicing

Just as with strings, you can access elements in a list using indexing and slicing. The first element of a list has index zero, the second element has index one, and so on. Use square brackets to index:

```
phrase = ['Astra', 'inclinant', 'sed', 'non', 'obligant']
print(phrase[1])
```

Output:

```
inclinant
```

You can also use negative indices to access items from the end of a list:

```
phrase = ['Astra', 'inclinant', 'sed', 'non', 'obligant']
print(phrase[-1])
```

Output:

```
obligant
```

Use slicing to extract a sublist. To slice, use square brackets containing a range of indices separated by a colon:

```
phrase = ['Astra', 'inclinant', 'sed', 'non', 'obligant']
print(phrase[1:4])
```

Output:

```
['inclinant', 'sed', 'non']
```

Notice that this code returned a sublist containing the elements at indices one, two, and three of phrase. The ending index of the slice is not included.

Omitting the starting index in a slice implies an index of zero, and omitting the ending index implies an index of **`len(my_list)`**:

```
phrase = ['Astra', 'inclinant', 'sed', 'non', 'obligant']
print(phrase[:3])
print(phrase[3:])
```

Output:

```
['Astra', 'inclinant', 'sed']
['non', 'obligant']
```

List mutability

Lists are mutable, which means that you can change their contents after they are created. You can change an individual item in a list by specifying its index and assigning a new value to it. For example:

```
my_list = ['Macduff', 'Malcolm', 'Duncan', 'Banquo']
my_list[2] = 'Macbeth'
print(my_list)
```

Output:

```
['Macduff', 'Malcolm', 'Macbeth', 'Banquo']
```

You can even change a slice of a list using the same logic. The slice can be of any length. The elements in the new list will be inserted in place of the indicated slice:

```
my_list = ['Macduff', 'Malcolm', 'Macbeth', 'Banquo']
my_list[1:3] = [1, 2, 3, 4]
print(my_list)
```

Output:

```
['Macduff', 1, 2, 3, 4, 'Banquo']
```

List operations

Lists can be combined using the addition operator (`+`):

```
num_list = [1, 2, 3]
char_list = ['a', 'b', 'c']
num_list + char_list
```

Output:

```
[1, 2, 3, 'a', 'b', 'c']
```

They can also be multiplied using the multiplication operator (`*`):

```
list_a = ['a', 'b', 'c']
list_a * 2
```

Output:

```
['a', 'b', 'c', 'a', 'b', 'c']
```

But they cannot be subtracted or divided.

You can check whether a value is contained in a list by using the **in** operator:

```
num_list = [2, 4, 6]
print(5 in num_list)
```

```
print(5 not in num_list)
```

Output:

```
False
True
```

List methods

Lists are a core Python class. As you've learned, classes package data together with tools to work with it. Methods are functions that belong to a class. Lists have a number of built-in methods that are very useful.

append()

Add an element to the end of a list:

```
my_list = [0, 1, 1, 2, 3]
variable = 5
my_list.append(variable)
print(my_list)
```

Output:

```
[0, 1, 1, 2, 3, 5]
```

insert()

Insert an element at a given position:

```
my_list = ['a', 'b', 'd']
my_list.insert(2, 'c')
print(my_list)
```

Output:

```
['a', 'b', 'c', 'd']
```

remove()

Remove the first occurrence of an item:

```
my_list = ['a', 'b', 'd', 'a']
my_list.remove('a')
```

```
print(my_list)
```

Output:

```
['b', 'd', 'a']
```

pop()

Remove the item at the given position in the list, and return it. If no index is specified, **pop()** removes and returns the last item in the list:

```
my_list = ['a', 'b', 'c']
print(my_list.pop())
print(my_list)
```

Output:

```
c
['a', 'b']
```

clear()

Remove all items:

```
my_list = ['a', 'b', 'c']
my_list.clear()
print(my_list)
```

Output:

```
[]
```

index()

Return the index of the first occurrence of an item in the list:

```
my_list = ['a', 'b', 'c', 'a']
my_list.index('a')
```

Output:

```
0
```

count()

Return the number of times an item occurs in the list:

```
my_list = ['a', 'b', 'c', 'a']
my_list.count('a')
```

Output:

```
2
```

sort()

Sorts the list ascending by default. You can also make a function to decide the sorting criteria:

```
char_list = ['b', 'c', 'a']
num_list = [2, 3, 1]
char_list.sort()
num_list.sort(reverse=True)
print(char_list)
print(num_list)
```

Output:

```
['a', 'b', 'c']
[3, 2, 1]
```

Additional resources

- For more information about lists, refer to An Informal Introduction to Python: Lists.
- For more list methods, refer to Data Structures: More on Lists.
