learn-co-curriculum / dsc-lists Public

View license

0 stars 85 forks

Star Watch

Code Issues Pull requests Actions Projects Security Insights

master

hoffm386 fix objectives spacing ... on Aug 1 9

View code

README.md

# Working with Lists

## Introduction

So far, we have worked with individual pieces of data like the string 'hello'. In this lesson, we'll see how we can group pieces of data together using lists.

## Objectives

You will be able to:

- Use indexing to access elements in a list
- Apply list methods to make changes to a list
- Change elements of a list

## What Are Lists?

A list is our first form of a collection. A collection is just a way of grouping multiple pieces of data together. For example, let's consider the top cities for travel according to the magazine Travel and Leisure. Here is how we usually see a list of travel locations in a document or on a website.

**Travel Locations**

1. Solta
2. Greenville
3. Buenos Aires
4. Los Cabos
5. Walla Walla Valley
6. Marakesh
7. Albuquerque
8. Archipelago Sea
9. Iguazu Falls
10. Salina Island
11. Toronto
12. Pyeongchang

Here is what that list looks like as a Python `list` :

```
['Solta', 'Greenville', 'Buenos Aires', 'Los Cabos', 'Walla Walla Valley', 'Marakesh
```

```
['Solta',
 'Greenville',
 'Buenos Aires',
 'Los Cabos',
 'Walla Walla Valley',
 'Marakesh',
 'Albuquerque',
 'Archipelago Sea',
 'Iguazu Falls',
 'Salina Island',
 'Toronto',
 'Pyeongchang']
```

We indicate that we are initializing a `list` with an opening bracket, `[`, and we end the list with a closing bracket `]`. We separate each list item, also called an element, with a comma.

```
['Croatia', 'USA', 'Argentina', 'Mexico', 'USA', 'Morocco', 'New Mexico', 'Finland',
```

```
['Croatia',
 'USA',
 'Argentina',
 'Mexico',
 'USA',
 'Morocco',
 'New Mexico',
 'Finland',
 'Argentina',
 'Italy',
 'Canada',
 'South Korea']
```

We can, of course, assign lists to variables and later retrieve the elements of lists using the variable names.

```
top_travel_cities = ['Solta', 'Greenville', 'Buenos Aires', 'Los Cabos', 'Walla Wall
```

```
top_travel_cities
```

```
['Solta',
 'Greenville',
 'Buenos Aires',
 'Los Cabos',
 'Walla Walla Valley',
 'Marakesh',
 'Albuquerque',
 'Archipelago Sea',
 'Iguazu Falls',
 'Salina Island',
 'Toronto',
 'Pyeongchang']
```

```
countries_of_top_cities = ['Croatia', 'USA', 'Argentina', 'Mexico', 'USA', 'Morocco'
```

## Accessing Elements of Lists

Now our `top_travel_cities` list contains multiple elements, and just like we are used to list elements having a rank or number associated with them...

1. Solta
2. Greenville
3. Buenos Aires

...a list in Python also assigns a number to each element.

```
top_travel_cities
```

```
['Solta',
 'Greenville',
 'Buenos Aires',
 'Los Cabos',
 'Walla Walla Valley',
 'Marakesh',
 'Albuquerque',
 'Archipelago Sea',
 'Iguazu Falls',
 'Salina Island',
 'Toronto',
 'Pyeongchang']
```

```
top_travel_cities[0]
```

```
'Solta'
```

In the above line we are referencing a list and then using the brackets to access a specific element of our list, the first element. We access elements in a list with the `index` , and there is a separate index for each element in the list. It begins at the number **zero** (not the number 1 as you might expect). Like many modern programming languages , Python uses a "zero-indexed" numbering scheme for collections like lists. The value then increases by 1 for every element thereafter.

So to access the second element we write `top_travel_cities[1]` , and the third element is `top_travel_cities[2]` .

```
top_travel_cities[2]
```

```
'Buenos Aires'
```

How would we access the last element? Well, we could count all of the elements in the list, and `Pyeongchang` would just be one less than that. Or we can ask Python to start from the end and move back one:

```
top_travel_cities[-1]
```

```
'Pyeongchang'
```

And we can move back as many as we want.

```
top_travel_cities[-2]
```

```
'Toronto'
```

Each element in our list is a string, so, we can always set an element of our string equal to a variable.

```
top_canadian_city = top_travel_cities[-2]
top_canadian_city
```

```
'Toronto'
```

```
type(top_canadian_city)
```

```
str
```

Now we have a variable of `top_canadian_city`, equal to the string 'Toronto', and a variable of `top_travel_cities` equal to the list of cities.

```
top_travel_cities
```

```
['Solta',
 'Greenville',
 'Buenos Aires',
 'Los Cabos',
 'Walla Walla Valley',
 'Marakesh',
 'Albuquerque',
 'Archipelago Sea',
 'Iguazu Falls',
 'Salina Island',
 'Toronto',
 'Pyeongchang']
```

```
type(top_travel_cities)
```

```
list
```

## Accessing Multiple Elements

Now imagine that we don't want to access just one element of a list, but multiple elements at once. Python allows us to do that as well:

```
top_travel_cities[0:2]
```

```
['Solta', 'Greenville']
```

As we can see from the above example, we can access elements of a list by placing two numbers separated by a colon inside of our brackets. The first number indicates the index of the first element we wish to retrieve.

The second number could represent the number of elements we want to retrieve, or maybe it represents the stopping index of the elements that we are retrieving. Looking at our `top_travel_cities` it could be either.

```
top_travel_cities
```

```
['Solta',
 'Greenville',
 'Buenos Aires',
 'Los Cabos',
 'Walla Walla Valley',
 'Marakesh',
 'Albuquerque',
 'Archipelago Sea',
 'Iguazu Falls',
 'Salina Island',
 'Toronto',
 'Pyeongchang']
```

Let's try a different experiment to answer our question.

```
top_travel_cities[4:5]
```

```
['Walla Walla Valley']
```

Ok, so that second number is not representing the number of elements we want retrieved. Instead it must be the index at which we stop our selection of elements.

```
top_travel_cities[4:6]
```

```
['Walla Walla Valley', 'Marakesh']
```

This operation is called `slice` . So, we can say we are `slicing` the elements with indices 4 and 5 in the line above. Note that even though we are `slicing` elements, our list remains intact.

```
top_travel_cities
```

```
['Solta',
 'Greenville',
 'Buenos Aires',
 'Los Cabos',
 'Walla Walla Valley',
 'Marakesh',
 'Albuquerque',
 'Archipelago Sea',
 'Iguazu Falls',
 'Salina Island',
 'Toronto',
 'Pyeongchang']
```

In programming terms, we would say that slicing elements is non-destructive, because it does not change the underlying data structure. We can do it as many times as we like, and our `top_travel_cities` array remains unchanged. If we wish to store that slice of elements, we can store it in another variable.

```
top_two = top_travel_cities[0:2]
top_two
```

```
['Solta', 'Greenville']
```

Now we have another variable called `top_two` that points to an array which contains an array of elements equal to the first two elements of `top_travel_cities` .

## Changing elements with destructive methods

Now that we can read and select certain elements from lists, let's work on changing these lists. To add a new element to a list, we can use the `append` method.

```
top_travel_cities.append('San Antonio')
```

Now let's take another look at `top_travel_cities` .

```
top_travel_cities
```

```
['Solta',
 'Greenville',
 'Buenos Aires',
 'Los Cabos',
 'Walla Walla Valley',
 'Marakesh',
 'Albuquerque',
 'Archipelago Sea',
 'Iguazu Falls',
 'Salina Island',
 'Toronto',
 'Pyeongchang',
 'San Antonio']
```

You will see that 'San Antonio' has been added to the list. Note that unlike slice, `append` is destructive. That is, it changes our underlying data structure. Every time we execute the `append` method, another element is added to our list. Now what if we accidentally add 'San Antonio' a second time to our list.

```
top_travel_cities.append('San Antonio')
top_travel_cities
```

```
['Solta',
 'Greenville',
 'Buenos Aires',
 'Los Cabos',
 'Walla Walla Valley',
 'Marakesh',
 'Albuquerque',
 'Archipelago Sea',
 'Iguazu Falls',
 'Salina Island',
 'Toronto',
 'Pyeongchang',
 'San Antonio',
 'San Antonio']
```

If you press shift+enter on the above line of code, we will have `'San Antonio'` as the last two elements of the list. Luckily, we have the `pop` method to remove one of them. The `pop` method is available to call on any list and removes the last element from the list. As you can see below, calling `pop` removed our last element.

```
top_travel_cities.pop()
```

```
'San Antonio'
```

Now if we want to change an element from the middle of the list, we can access and then reassign that element. For example, let's change 'Walla Walla Valley' to the number 5.

```
top_travel_cities[4]
```

```
'Walla Walla Valley'
```

```
top_travel_cities[4] = 5
```

```
top_travel_cities
```

```
['Solta',
 'Greenville',
 'Buenos Aires',
 'Los Cabos',
 5,
 'Marakesh',
 'Albuquerque',
 'Archipelago Sea',
 'Iguazu Falls',
 'Salina Island',
 'Toronto',
 'Pyeongchang',
 'San Antonio']
```

Our list is changed, but now it's not as sensible, so let's change it back.

```
top_travel_cities[4] = 'Walla Walla Valley'
```

With that, our list is back to the way we like it.

```
top_travel_cities
```

```
['Solta',
 'Greenville',
 'Buenos Aires',
 'Los Cabos',
 'Walla Walla Valley',
 'Marakesh',
 'Albuquerque',
 'Archipelago Sea',
 'Iguazu Falls',
 'Salina Island',
 'Toronto',
 'Pyeongchang',
 'San Antonio']
```

## Finding Unique elements and length of lists

If we are not sure whether there are repeated elements, we can use Python to get a unique list.

```
top_travel_cities.append('Solta')
top_travel_cities
```

```
['Solta',
 'Greenville',
 'Buenos Aires',
 'Los Cabos',
 'Walla Walla Valley',
 'Marakesh',
 'Albuquerque',
 'Archipelago Sea',
 'Iguazu Falls',
 'Salina Island',
 'Toronto',
 'Pyeongchang',
 'San Antonio',
 'Solta']
```

For example, now that we have added Solta to the end of our list, Solta appears twice.

Well to see a unique list of the elements, we can call the `set` function. A set is a different type collection in Python. A set is just like a list, except elements do not have order and each element appears just once.

```
unique_travel_cities = set(top_travel_cities)
unique_travel_cities
```

```
{'Albuquerque',
 'Archipelago Sea',
 'Buenos Aires',
 'Greenville',
 'Iguazu Falls',
 'Los Cabos',
 'Marakesh',
 'Pyeongchang',
 'Salina Island',
 'San Antonio',
 'Solta',
 'Toronto',
 'Walla Walla Valley'}
```

The `set` function is non-destructive on our list.

```
top_travel_cities
```

```
['Solta',
 'Greenville',
 'Buenos Aires',
 'Los Cabos',
 'Walla Walla Valley',
 'Marakesh',
 'Albuquerque',
 'Archipelago Sea',
 'Iguazu Falls',
 'Salina Island',
 'Toronto',
 'Pyeongchang',
 'San Antonio',
 'Solta']
```

So here, when we convert our list into a set, our set just consists of the unique elements. But unfortunately this structure is a set, not a list.

```
type(unique_travel_cities)
```

```
set
```

So let's convert this set, which has a unique list of our travel cities, into a list.

```python
unique_travel_cities = list(unique_travel_cities)
```

```python
type(unique_travel_cities)
```

```
list
```

So the array of `unique_travel_cities` is now a unique list.

```python
unique_travel_cities
```

```
['Archipelago Sea',
 'Salina Island',
 'Toronto',
 'Marakesh',
 'Greenville',
 'San Antonio',
 'Los Cabos',
 'Pyeongchang',
 'Walla Walla Valley',
 'Albuquerque',
 'Solta',
 'Iguazu Falls',
 'Buenos Aires']
```

And you can see quickly that it differs from the list of top travel cities by checking the length.

```python
len(unique_travel_cities)
```

```
13
```

```python
len(top_travel_cities)
```

```
14
```

> **Note:** *For most purposes, Python developers prefer to work with `lists` as opposed to sets, as `lists` are generally easier to manipulate, as you will see in future lessons.*

## Summary

In this section we saw how to associate data together in a collection, called a list. A list is similar to a list in the real world - it implies the data has some connection, and that it has an order to it. We initialize a list with the brackets, `[]`, and separate each element by a comma. To access elements from a list, we use the bracket accessor followed by the index of the element we want to retrieve, and our indices begin at zero and increase by 1 from there. To add a new element to the end of the list we use the `append` method, and to remove an element from the end of a list we use the `pop` method. We can change elements anywhere between by first accessing the elements and then reassigning them.

### Releases

No releases published

### Packages

No packages published

### Contributors 5

### Languages

● **Jupyter Notebook** 100.0%