Data Scientist Codecademy

Python Fundamentals

Lists

Introduction to Lists

What is a List?

A list is one the many built in data structure that allows us to work with a collection of data in sequential order.

- a list begins and ends with square brackets []
- each item in a list is separated by comma ,

What can a List Contain?

Lists can contain any data type in Python! There is no restriction for the data types in a list to be same as with arrays in other programming languages.

Empty Lists

A list doesn’t have to contain anything. We can create an empty list like this:

```
empty_list = []
```

List Methods

Growing a List: Append

We can add a single element to a list using the .append() python method.

When the list already has some elements the new element is added at the end of the existing list.

Growing a List: Plus (+)

When we want to add multiple items to a list, we can use + to combine two lists, this is also known as concatenation.

```
new_list = old_list + [item1, item2, item3]
```

this will add new items to old_list and save it in new_list. old_list remains unaffected.

We can use + only with other lists.

```
list + 4        will generate error

list + [4]      will not generate error
```

Accessing List Elements

In Python, we call the location of an element in a list its index.

Python lists are zero-indexed. This means that the first element in a list has index 0, rather than 1.

We can select a single element from a list by using square brackets ([]) and the index of the list item.

```
print(list[index])
```

When accessing elements of a list, index must be of int type, a float index will generate an error.

Selecting an element that does not exists produces an

```
IndexError: index out of range
```

Accessing List Elements: Negative Index

If we want to select the last element of a list we can use the index -1 to select the last item of a list, even when we don’t know how many elements are in a list.

Negative indexing starts with -1 at the last element and reduces iteratively backward with each element in the list.

Modifying List Elements

We can change a value in the list by reassigning the value using the specific index. Negative indices work as well.

Shrinking a List: Remove

We can remove elements in a list using the .remove() python method.

```
list.remove(element)
```

If the list has duplicate occurrences of element in it then the first instance will be removed.

If the element to be removed does not exist in the list then python throws a error

```
ValueError: list.remove(x) : x not in list
```

Two-Dimensional (2D) Lists

Lists can contain other lists! We refer to these as two-dimensional (2D) lists.

Accessing 2D Lists

Two-dimensional lists can be accessed similar to their one-dimensional counterpart. Instead of providing a single pair of brackets [] we will use an additional set for each dimension past the first.

Adding by index: Insert

The Python list method .insert() allows us to add an element to a specific index in a list. The .insert() method takes in two inputs:

- the index you want to insert into
- the element you want to insert at the specified index

.insert() method handles shifting over elements and can be used with negative indices.

Removing by index: Pop

.pop() method removes elements at a specific index. It takes a single input:

- the index for the element you want to remove.

Using .pop() without an index will remove whatever the last element of the list is.

.pop() method returns the removed element which can be stored for future use.

Passing an index that does not exists or calling .pop() on an empty list will both result in an IndexError.

Consecutive Lists: Range

Often we want to create a list of consecutive numbers in our programs. We can manually create one by typing out each element or we can use a built-in python function called range().

The function range() takes a single input, and generates numbers starting at 0 and ending at the number before the input.

range() function creates a range object. In order to use this object as a list, we have to first convert it using another built-in function called list().

```
my_range = range(10)
print(my_range)
print(list(my_range))
```

```
Output:    range(0, 10)
           [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

The Power of Range!

By default, range() creates a list starting at 0. However, if we call range() with two inputs, we can create a list that starts at a different number.

```
my_range = range(2,9)
print(list(my_range))
```

```
Output:    [2, 3, 4, 5, 6, 7, 8]
```

If we use a third input, we can create a list that “skips” numbers.

```
my_range = range(2,9,2)
print(list(my_range))
```

Output: [2, 4, 6, 8]

Length

The number of elements in a list is called its length. We can find the length of a list using a built-in function called len().

Slicing List 1

In programming we would often want to extract only a portion of a list. Dividing a list in such a manner is referred to as slicing.

Syntax: list_name[start:end]

- start is the index of the first element that we want to include
- end is the index of *one more than* the last index that we want to include

Slicing List 2

If we want to select the first n elements of a list, we could use following syntax:

```
list_name[:n]
```

If we want to slice last n elements we can do:

```
list_name[-n:]
```

If we don’t want last n elements:

```
list_name[:-n]
```

Counting in a List

We could also come across problems when we would like to find the frequency of an item in a list. We can use a built-in method .count().

.count() returns a value which can be stored and takes a single input which is the item to count in the list.

Sorting Lists 1

Often we want to sort our list. We can use .sort() method to do so.

.sort() sorts the list in numerical/alphabetical order.

.sort() also gives the option to sort in reverse.

```
names.sort()

names.sort(reverse=True)
```

.sort() does not returns anything so if it assigned to a variable it will get a value None.

Sorting Lists 2

We can also use a built-in function sorted(). It is different from .sort() as:

- it comes before a list, instead of after as all built-in function do.
- it generates a new list rather than modifying the one that already exists.

zip() function

the zip() function allows us to quickly combine associated data-sets without needing to rely on multi-dimensional lists.

If we had two list like:

```
names = [“Jenny”, “Alexus”, “Sam”, “Grace”]

heights = [61, 70, 67, 64]
```

We can use the zip function as

```
names_and_heights = zip(names, heights)
print(names_and_heights)
```

Output: <zip object at 0x7f1631e86b48>

This zip object contains the location of this variable in our computer's memory.

```
converted_list = list(names_and_heights)
print(converted_list)
```

Output: [("Jenny", 61), ("Alexus", 70), ("Sam", 67), ("Grace", 64)]

Things to notice:

- our inner list don't use [] cause they have been converted into tuples.

Tuples

What are Tuples?

Tuple is a data structure in python which allows us to store multiple pieces of data inside of it. Tuples are immutable which means that once a tuple is created it cannot be changed or modified.

```
tuple_name = (item1, item2, item3, ...)
tuple_name[0]   # outputs item1
```

Unpacking a tuple:

```
item1, item2, item3 = tuple_name
```

as long as number of variables on left is same to number of elements in tuple we can store respective element values in respective variable name.

While creating a one element tuple

```
tuple_name = (item)
```

will not create a tuple instead

```
tuple_name = (item, )
```

will do so as without the trailing comma python treats it as an expression.