Set 11: Working With Lists

Skill 11.1: Apply the *insert()* method to add an element at a specific index in a list

Skill 11.1 Concepts

Now that we know how to create and access list data, we can start to explore additional ways of working with lists with Python's built-in functions.

Previously we learned how to add items to the end of a list using *append()*. The built-in *insert()* method allows us to add an element to a specific location in a list.

The *insert()* method takes in two inputs:

1. The index you want to insert into.
2. The element you want to insert at the specified index.

The *insert()* method will handle shifting over elements and can be used with negative indices.

To see it in action let's imagine we have a list representing a line at a store.

"Maxium" saved a spot for his friend "Vikor" and we need to adjust the list to add him into the line right behind "Maxium".

For this example, we can assume that "Karla" is the front of the line and the rest of the elements are behind her.

Here is how we would use the *insert()* method to insert "Vikor" :

Code

```
store_line = ["Karla", "Maxium", "Martim", "Isabella"]
store_line.insert(2, "Vikor")
print(store_line)
```

Output

```
['Karla', 'Maxium', 'Vikor', 'Martim', 'Isabella']
```

Some important things to note,

1. The order and number of the inputs is important. The *insert()* method expects two inputs, the first being a numerical index, followed by any value as the second input.

2. When we insert an element into a list, all elements from the specified index and up to the last index are shifted one index to the right. This does not apply to inserting an element to the very end of a list as it will simply add an additional index and no other elements will need to shift.

Skill 11.1 Exercise

Skill 11.2: Apply the *pop()* method to remove an element from a list

Skill 11.2 Concepts

Just as we learned to insert elements at specific indices, we can use the *pop()* method to remove an element at a specific index.

To see it in action, let's consider a list called *cs_topics* that stores a collection of topics one might study in a computer science program.

Two of these topics don't look like they belong, let's see how we remove them using *pop().*

Code

```
cs_topics = ["Python", "Data Structures", "Balloon Making", "Algorithms", "Clowns 101"]
removed_element = cs_topics.pop()
print(cs_topics)
print(removed_element)
```

Output

```
['Python', 'Data Structures', 'Balloon Making', 'Algorithms']
Clowns 101
```

Notice two things about this example,

1. *pop()* can be called without a specific index. Using *pop()* without an parameter will remove whatever the last element of the list is. In our case "Clowns 101" gets removed.

2. *pop()* is unique in that it will *return* the value that was removed. If we wanted to know what element was deleted, simply assign a variable to the call of the *pop()* method. In this case, we assigned it to *removed_element*.

Next, let's remove "Balloon Making":

Code

```
cs_topics = ["Python", "Data Structures", "Algorithms"]
cs_topics.pop(2)
print(cs_topics)
```

Output

```
['Python', 'Data Structures', 'Algorithms']
```

Notice two things about this example,

1. *pop()* can be called with an optional specific index to remove. In our case, the index 2 removes the value of "Balloon Making".

2. We don't have to save the removed value to any variable if we don't care to use it later.

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

Code

```
cs_topics = ["Python", "Data Structures", "Algorithms"]
cs_topics.pop(3)
print(cs_topics)
```

Output

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

Skill 11.2 Exercise 1

Skill 11.3: Create a list with the *range()* funciton

Skill 11.3 Concepts

Often, we want to create a list of consecutive numbers in our programs. For example, suppose we want a list containing the numbers 0 through 9,

```
my_list = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Typing out all of those numbers takes time and the more numbers we type, the more likely it is that we have a typo that can cause an error.

Python gives us an easy way of creating these types of lists using a built-in function called *range()*.

The function *range()* takes a single input, and generates numbers starting at 0 and ending at the number **before** the input. So, if we want the numbers from 0 through 9, we use range(10) because 10 is 1 greater than 9,

Code	Output
`my_range = range(10)` `print(my_range)`	range(0, 10)

Notice something different? The *range()* function is unique in that it creates a *range object*. It is not a typical list like the ones we have been working with.

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

.

The *list()* function takes in a single input for the object you want to convert. We use the *list()* function on our range object like this,

Code

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

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

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.

For example, range(2, 9) would generate numbers starting at 2 and ending at 8 (just before 9):

Code
`my_list = range(2, 9)` `print(list(my_list))`
Output
[2, 3, 4, 5, 6, 7, 8]

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

For example, range(2, 9, 2) will give us a list where each number is 2 greater than the previous number:

Code
`my_range2 = range(2, 9, 2)` `print(list(my_range2))`
Output
[2, 4, 6, 8]

We can skip as many numbers as we want!

For example, we'll start at 1 and skip in increments of 10 between each number until we get to 99 (one before 100):

Code

```
my_range3 = range(1, 100, 10)
print(list(my_range3))
```

Output

```
[1, 11, 21, 31, 41, 51, 61, 71, 81, 91]
```

Our list stops at 91 because the next number in the sequence would be 101, which is greater than or equal to 100 (our stopping point).

Skill 11.3 Exercise 1

Skill 11.4: Apply slicing to lists

Skill 11.4 Concepts

In Python, often we want to extract only a portion of a list. Dividing a list in such a manner is referred to as *slicing*.
Let's assume we have a list of letters,

```
letters = ["a", "b", "c", "d", "e", "f", "g"]
```

Suppose we want to select from "b" through "f".

We can do this using the following syntax: letters[start:end], where,

- start is the index of the first element that we want to include in our selection. In this case, we want to start at "b", which has index 1.
- end is the index of *one more than* the last index that we want to include. The last element we want is "f", which has index 5, so end needs to be 6.

Code

```
letters = ["a", "b", "c", "d", "e", "f", "g"]


sliced_list = letters[1:6]
print(sliced_list)
```

The second number is not inclusive, so b thru f is printed

Output

```
['b', 'c', 'd', 'e', 'f']
```

Slicing syntax in Python is very flexible. Let's look at a few more problems we can tackle with slicing.

Take the list fruits as our example.

```
fruits = ["apple", "cherry", "pineapple", "orange", "mango"]
```

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

```
fruits[:n]
```

The following code starts slicing from index 0 and up to index 3. Note that the fruit at index 3 (orange) is not included in the results.

Code

```
fruits = ["apple", "cherry", "pineapple", "orange", "mango"]
first_three = fruits[:3]
print(first_three)
```

If no value is provided, slicing starts at index 0

Output

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

We can do something similar when we want to slice the *last n elements* in a list,

```
fruits[-n:]
```

For example, suppose we wanted to slice the last two elements. The following example slices from the element at index -2 up through the last index.

Code

```
fruits = ["apple", "cherry", "pineapple", "orange", "mango"]
last_two = fruits[-2:]
print(last_two)
```

If no value is provided, slicing continues to the end of the list

Output

```
['orange', 'mango']
```

Negative indices can also accomplish taking *all but n last elements* of a list.

```
fruits[:-n]
```

For our fruits example, suppose we wanted to slice all but the last element from the list.

Code

```
fruits = ["apple", "cherry", "pineapple", "orange", "mango"]
all_but_last = fruits[:-1]
print(all_but_last)
```

Output

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

Skill 11.4 Exercise 1

Skill 11.5: Apply the *count()* method to count the instances of an element in a list

Skill 11.5 Concepts

In Python, it is common to want to count occurrences of an item in a list.

Suppose we have a list called letters that represents the letters in the word “Mississippi”,

Code

```
letters = ["m", "i", "s", "s", "i", "s", "s", "i", "p", "p", "i"]
num_i = letters.count("i")
print(num_i)
```

Output

```
4
```

We can even use *count()* to count element appearances in a two-dimensional list.

Let’s use the list number_collection as an example,

Code

```
number_collection = [[100, 200], [100, 200], [475, 29], [34, 34]]
num_pairs = number_collection.count([100, 200])
print(num_pairs)
```

Output

```
2
```

Skill 11.5 Exercise 1

Skill 11.6: Apply the *index()* method to locate the index of an element

Skill 11.6 Concepts

The *index()* method in Python is used to find the position (index) of a specified element within a sequence, such as a list. It returns the index of the first occurrence of the element.

Below is an example,

Code

```
secret_word = "EARTH"
word_list = list(secret_word)
result = ["_"] * len(word_list)

letter_loc = secret_word.index("T")
result[letter_loc] = "T"

print(letter_loc)
print(result)
```

Output

```
3
['_', '_', '_', 'T', '_']
```

If a list contains more than one instance of the element, only the index of the first element is returned.

Code

```
secret_word = "TIMBERLINE"
word_list = list(secret_word)
result = ["_"] * len(word_list)

letter_loc = secret_word.index("I")
result[letter_loc] = "I"

print(letter_loc)
print(result)
```

Output

```
1
['_', 'I', '_', '_', '_', '_', '_', '_', '_', '_']
```

Skill 11.6 Exercise 1

Skill 11.7: Sort a list

Skill 11.7 Concepts

Often, we will want to sort a list in either numerical (1, 2, 3, …) or alphabetical (a, b, c, …) order.
We can sort a list using the *sort()* method.

Below is an example,

Code

```
names = ["Xander", "Buffy", "Angel", "Willow", "Giles"]
names.sort()
print(names)
```

Output

```
['Angel', 'Buffy', 'Giles', 'Willow', 'Xander']
```

As we can see above, the *sort()* method sorted our list of names in alphabetical order.

sort() also provides us the option to go in reverse. Instead of sorting in ascending order like we just saw, we can do so in descending order.

Code

```
names = ["Xander", "Buffy", "Angel", "Willow", "Giles"]
names.sort(reverse=True)
print(names)
```

Output

```
['Xander', 'Willow', 'Giles', 'Buffy', 'Angel']
```

Note: The *sort()* method does not return any value and thus does not need to be assigned to a variable since it modifies the list directly. If we do assign the result of the method, it would assign the value of None to the variable.

Code

```
names = ["Xander", "Buffy", "Angel", "Willow", "Giles"]
names_reversed = names.sort(reverse=True)
print(names_reversed)
```

Output

```
None
```

A second way of sorting a list in Python is to use the built-in function *sorted().*

sorted() is different from *sort()* in two ways:

1. It comes *before* a list, instead of after as all built-in functions do.
2. It generates a new list rather than modifying the one that already exists.

Let's return to our list of names,

Code

```
names = ["Xander", "Buffy", "Angel", "Willow", "Giles"]
sorted_names = sorted(names)
print(sorted_names)
```

Output

```
['Angel', 'Buffy', 'Giles', 'Willow', 'Xander']
```

Note that using sorted did not change names,

Code

```
names = ["Xander", "Buffy", "Angel", "Willow", "Giles"]
sorted_names = sorted(names)
print(sorted_names)
print(names)
```

Output

```
['Xander', 'Buffy', 'Angel', 'Willow', 'Giles']
['Angel', 'Buffy', 'Giles', 'Willow', 'Xander']
```

Skill 11.7 Exercises 1 and 2

Skill 11.8: Create a list from a String

Skill 11.8 Concepts

We have already seen how we can convert a range object into a list. It is also possible to turn a String into a list,

Code

```
secret_word = "EARTH"
word_list = list(secret_word)
print(secret_word)
print(word_list)
```

Output

```
EARTH
['E', 'A', 'R', 'T', 'H']
```

Skill 11.8 Exercise 1