Lists

can code communities

Lists

Lists store a group of objects (things).

In a list, we can have any type of object.

Unlike strings, lists are mutable.

- This means we can change an individual object in a list using index.
- We can also add and remove objects from a list.

We can also use lists in a for loop.

Index Review

Indexing is how we access an individual thing inside a group.

We use square brackets([]) after the variable name to denote indexing.

Indexes start at 0 and go to the length of the object minus 1.

You can use indexes to access the individual items in a list, just like with strings.

```
planets = ['mercury', 'venus', 'earth', 'mars', 'jupiter']
```

`planets[3]` is `'mars'`

List Indexing

Because lists are mutable, you can use indexes to modify items in a list.

Let's say we have a list of planets, but we misspelled one of them:

```
planets = ['mercury', 'venus', erath, 'mars']
```

We can update it using indexing:

```
planets[2] = 'earth'
```

Now the value of planets is: `['mercury', 'venus', earth, 'mars']`

We just changed one item in the list without changing the entire thing.

Lists in a For Loop

You can iterate through a list in a for loop, just like with a string, because a list is an iterable - a collection of objects.

Iterating through a string goes through each letter:

```
for i in "hello":
```

Iteration 1: `i = "h"`

Iteration 2: `i = "e"`

Iteration 3: `i = "l"` etc.

Iterating through a list goes through each item:

```
for i in [1, 2, 3, 4]:
```

Iteration 1: `i = 1`

Iteration 2: `i = 2`

Iteration 3: `i = 3` etc.

Similarly, the len() function returns the number of letters in a string, and the number of items in a list.

Discussion: Why are lists useful?

Exercise

Create a for loop that goes through a list and prints each item in the list, along with its index. (Hint: Create a separate counter variable to keep track of the index.)

Example:

```
planets = ["mercury", "venus", "earth", "mars"]
0: mercury, 1: venus, 2: earth, 3: mars
```

Exercise

Write some code that takes one list and creates a second list that has the **type** for each entry in the list. Hint: Use the **type()** function and a **for** loop

Optional:

Make sure you filter out any repeats.

List Methods

`append()` Adds an element at the end of the list

`clear()` Removes all the elements from the list

`copy()` Returns a copy of the list

`count()` Returns the number of elements with the specified value

`extend()` Add the elements of a list (or any iterable), to the end of the current list

`index()` Returns the index of the first element with the specified value

`insert()` Adds an element at the specified position

`pop()` Removes the element at the specified position

`remove()` Removes the first item with the specified value

`reverse()` Reverses the order of the list

`sort()` Sorts the list

Adding Elements

append() adds an item to the end of a list:

```
my_list = ['dog', 'cat']
```

```
my_list.append['turtle']
```

Now, the value of `my_list` is `['dog', 'cat', 'turtle']`

Exercise: List of Pets

You want to make a list containing the types of pets that the user has. Keep prompting the user for a pet until they enter "stop". If it's a new pet, add it to the list. If the list already has that pet, don't add it.

Removing Elements

pop() and **remove()** both remove an item from a list.

```
my_list = ['cat', 'dog', 'cat']
```

- **pop()** removes an item based on its index.

  After calling `my_list.pop(2)`, the value of `my_list` is `['cat', 'dog']`

- **remove()** removes an item based on its value. If there are multiple, it removes the first item with that values.

  After calling `my_list.remove('cat')`, the value of `my_list` is `['dog', 'cat']`

List Methods

Since lists are mutable, the list methods directly modify the list - you don't have to set it equal to anything.

```
my_list = [3,4,2,1]

my_list.sort()
print(my_list)

my_list = my_list.sort()
print(my_list)
```

Discussion: What is the output of this code and why?

Sorting Lists

sort() is a list method. You call `list.sort()`, which directly modifies the list, so you don't have to set it equal to anything.

sorted() is a function. You call `sorted(list)`, which gives you a new, sorted list, so you have to set it equal to something. It doesn't modify the original list.

```
my_list = [3,6,2,4,1]
my_list.sort()
print(my_list)
```

Correctly sorting a list with sort()

```
my_list = [3,6,2,4,1]
my_list = sorted(my_list)
print(my_list)
```

Correctly sorting a list with sorted()

Copying Lists

When you make a copy of a list, it's good practice to use the .copy() method, otherwise modifying the copy will modify the original list.

```
planets = ['mercury', 'venus', 'earth', 'mars']

planets2 = planets
planets2.append('jupiter')
print(planets)

planets3 = planets.copy()
planets3.append('saturn')
print(planets)
```

Discussion: What is the output of this code and why?

After running this code, what is stored at `planets`, `planets2`, and `planets3`?

Example: Removing Values

You have a list of numbers, but it contains multiple of the number 2. Remove the number 2 until it only appears in the list once.

Exercise: Removing All Duplicates

You have a list storing important data for your company, but it contains some duplicate entries. Go through your list and remove all the duplicates. When you're done, each item should appear in the list exactly once.

Hint: How would you expand our previous example, which removed duplicates of one value, to remove duplicates of all values?

Hint 2: You might want to make a copy of the original list to use as reference. You may want to use more than one loop.

Resources

https://www.w3schools.com/python/python_lists.asp

https://docs.python.org/3/tutorial/datastructures.html