Learn Python 3 : Lists

codecademy

Python Lists

In Python, lists are ordered collections of items that allow for easy use of a set of data.

List values are placed in between square brackets `[ ]`, separated by commas. It is good practice to put a space in between the comma and the next value. The values in a list do not need to be unique (the same value can be repeated).

Empty lists do not contain any values within the square brackets.

```
primes = [2, 3, 5, 7, 11]
print(primes)

empty_list = []
```

Python Lists: Data Types

In Python, lists are a versatile data type that can contain multiple different data types within the same square brackets. The possible data types within a list include numbers, strings, other objects, and even other lists.

```
numbers = [1, 2, 3, 4, 10]
names = ['Jenny', 'Sam', 'Alexis']
mixed = ['Jenny', 1, 2]
list_of_lists = [['a', 1], ['b', 2]]
```

Adding Lists Together

In Python, lists can be added to each other using the plus symbol `+`. As shown in the code block, this will result in a new list containing the same items in the same order with the first list's items coming first.

Note: This will not work for adding one item at a time (use `.append()` method). In order to add one item, create a new list with a single value and then use to plus symbol to add the list.

```
items = ['cake', 'cookie', 'bread']
total_items = items + ['biscuit', 'tart']
print(total_items)
# Result: ['cake', 'cookie', 'bread', 'biscuit',
'tart']
```

List Indices

Python list elements are ordered by *index*, a number referring to their placement in the list. List indices start at 0 and increment by one.

To access a list element by index, square bracket notation is used: `list[index]`.

```
berries = ["blueberry", "cranberry", "raspberry"]

berries[0]   # "blueberry"
berries[2]   # "raspberry"
```

Aggregating Iterables Using `zip()`

In Python, data types that can be iterated (called iterables) can be used with the `zip()` function to aggregate data based on the iterables passed in.

As shown in the example, `zip()` is aggregating the data between the owners' names and the dogs' names to match the owner to their dogs. `zip()` returns an iterator containing the data based on what the user passes through and can be printed to visually represent the aggregated data. Empty iterables passed in will result in an empty iterator.

```
owners_names = ['Jenny', 'Sam', 'Alexis']
dogs_names = ['Elphonse', 'Dr. Doggy DDS', 'Carter']
owners_dogs = zip(owners_names, dogs_names)
print(owners_dogs)
# Result: [('Jenny', 'Elphonse'), ('Sam', 'Dr.Doggy
DDS'), ('Alexis', 'Carter')]
```

List Item Ranges Including First or Last Item

In Python, when selecting a range of list items, if the first item to be selected is at index `0`, no index needs to be specified before the `:`. Similarly, if the last item selected is the last item in the list, no index needs to be specified after the `:`.

```
items = [1, 2, 3, 4, 5, 6]

# All items from index `0` to `3`
print(items[:4])

# All items from index `2` to the last item, inclusive
print(items[2:])
```

List Method `.count()`

The `.count()` Python list method searches a list for whatever search term it receives as an argument, then returns the number of matching entries found.

```
backpack = ['pencil', 'pen', 'notebook', 'textbook',
'pen', 'highlighter', 'pen']
numPen = backpack.count('pen')
print(numPen)
# Output: 3
```

List Method `.append()`

In Python, you can add values to the end of a list using the `.append()` method. This will place the object passed in as a new element at the very end of the list. Printing the list afterwards will visually show the appended value. This `.append()` method is *not* to be confused with returning an entirely new list with the passed object.

```
orders = ['daisies', 'periwinkle']
orders.append('tulips')
print(orders)
# Result: ['daisies', 'periwinkle', 'tulips']
```

List Method `.sort()`

The `.sort()` Python list method will sort the contents of whatever list it is called on. Numerical lists will be sorted in ascending order, and lists of Strings will be sorted into alphabetical order. It modifies the original list, and has no return value.

```
exampleList = [4, 2, 1, 3]
exampleList.sort()
print(exampleList)
# Output: [1, 2, 3, 4]
```

Determining List Length with `len()`

The Python `len()` function can be used to determine the number of items found in the list it accepts as an argument.

```
knapsack = [2, 4, 3, 7, 10]
size = len(knapsack)
print(size)
# Output: 5
```

Zero-Indexing

In Python, list index begins at zero and ends at the length of the list minus one. For example, in this list, `'Andy'` is found at index `2`.

```
names = ['Roger', 'Rafael', 'Andy', 'Novak']
```

List Item Ranges Including First or Last Item

In Python, when selecting a range of list items, if the first item to be selected is at index `0`, no index needs to be specified before the `:`. Similarly, if the last item selected is the last item in the list, no index needs to be specified after the `:`.

```
items = [1, 2, 3, 4, 5, 6]

# All items from index `0` to `3`
print(items[:4])

# All items from index `2` to the last item, inclusive
print(items[2:])
```

`sorted()` Function

The Python `sorted()` function accepts a list as an argument, and will return a new, sorted list containing the same elements as the original. Numerical lists will be sorted in ascending order, and lists of Strings will be sorted into alphabetical order. It does not modify the original, unsorted list.

```
unsortedList = [4, 2, 1, 3]
sortedList = sorted(unsortedList)
print(sortedList)
# Output: [1, 2, 3, 4]
```

Negative List Indices

Negative indices for lists in Python can be used to reference elements in relation to the end of a list. This can be used to access single list elements or as part of defining a list range. For instance:

- To select the last element, `my_list[-1]`.
- To select the last three elements, `my_list[-3:]`.
- To select everything except the last two elements, `my_list[:-2]`.

```
soups = ['minestrone', 'lentil', 'pho', 'laksa']
soups[-1]  # 'laksa'
soups[-3:] # 'lentil', 'pho', 'laksa'
soups[-2:] # 'minestrone', 'lentil'
```

List Slicing

A *slice*, or sub-list of Python list elements can be selected from a list using a colon-separated starting and ending point.

The syntax pattern is `myList[START_NUMBER:END_NUMBER]`. The slice will include the `START_NUMBER` index, and everything until but excluding the `END_NUMBER` item.

When slicing a list, a new list is returned, so if the slice is saved and then altered, the original list remains the same.

```
tools = ['pen', 'hammer', 'lever']
tools_slice = tools[1:3] # ['hammer', 'lever']
tools_slice[0] = 'nail'

# Original list is unaltered:
print(tools) # ['pen', 'hammer', 'lever']
```