1 Lists

One of the most important data types in Python is the list. A list is an ordered collection of other values. For example, we might create a list of numbers representing data points on a graph or create a list of strings representing students in a class. Creating a list is simple. We simply place comma-separated values inside square brackets.

```
>>> values = [1, 2, 3, 4, 5]
>>> print(values)
[1, 2, 3, 4, 5]
```

1.1 Indices

Once we put a value in a list, we can treat the list as a single entity or access individual values. In order to access an individual element, we need to use the index of the value we want to access. The index is the position of the element in the list if we start numbering the elements from 0. When we are accessing list elements by index we can use them in any way we might use a normal variable.

```
>>> values = [23, 7, 18, 0.23, 91]
>>> print(values[0])
23
>>> print(values[2])
18
>>> print(values[3])
0.23
>>> values[3] = 7.5
>>> print(values[3])
7.5
>>> print(values)
[23, 7, 18, 7.5, 91]
>>> values[0] = values[0] + values[1]
>>> print(values)
[30, 7, 18, 7.5, 91]
```

If we wish, we can use negative array indices to reference elements starting at the end of the list. For example, -1 is the last element, -2 is the second to last element, and so on.

```
>>> values = [32, 1, 54, -3, 6]
>>> print(values[-1])
6
>>> print(values[-2])
-3
```

1.2 Values in Lists

We can use an existing variable when creating a list. If we change the value of the variable, the value in the list will stay the same.

```
>>> x = 35
>>> k = 19
>>> y = 5
>>> values = [x, k, y]
>>> print(values)
[35, 19, 5]
>>> x = 1
>>> print(values)
[35, 19, 5]
```

All values in a list do not need to be the same type. If we want, we can create a list with both numbers and strings, although this is usually a poor idea.

```
>>> values = [2, 'hello', 5.3]
>>> print(values)
[2, 'hello', 5.3]
```

If you want, you can even put a list inside a list!

```
>>> values = [1, 5, 2]
>>> more_values = [7, 'test', values]
>>> print(more_values)
[7, 'test', [1, 5, 2]]
>>> print(more_values[2])
[1, 5, 2]
```

Note that, unlike with other variables, changing an element of a list inside another list will change both values.

```
>>> values = [1, 5, 2]
>>> more_values = [7, 'test', values]
>>> print(more_values)
[7, 'test', [1, 5, 2]]
>>> values[2] = 7
>>> print(more_values)
[7, 'test', [1, 5, 7]]
>>> values = [1, 2]
>>> print(more_values)
[7, 'test', [1, 5, 7]]
```

You generally will not have to worry about this, though the reason is because we are modifying the existing value rather than create a new list. When we reassign `values`, it no longer changes the values inside inside `more_values`. This is because we are creating a new list for `values` rather than modifying the existing list.

1.3 Testing List Contents

A common operation is to test if a list contains a given value. We can do this using the `in` keyword. We can also test if a list does not contain a value using `not in`.

```
>>> values = [1, 'test', 30, 20]
>>> print(1 in values)
True
>>> print('test' in values)
True
>>> print(2 in values)
False
>>> print(2 not in values)
True
```

This could be used to simplify the example from Lab 2 involving checking user input against multiple valid passwords.

```
passwords = ['hunter2', 'hunter3', 'hunter4']

user_in = input('Please enter your password: ')

if user_in in passwords:
    print('Correct password. Welcome!')
else:
    print('Incorrect password.')
```

1.4 Slicing

Often we will want to make a new list out of part of a larger list. We do this using slicing. To do this, we specify the first and last indices we want to include in our new list, separated by a colon. The last index is used as a bookend – that is, values up to but not including that index are included in the new list. If one of the values is omitted, then Python will act as if the most extreme index on that side was enterred. Omitting both values will use the full list.

Thus, for a list `L` and two positions (indices) `B` and `E` within that list, the expression `L[ B : E ]` produces a new list containing the elements in `L` between those two positions not including `E`. Notice that `B` and `E` must be indices – whole numbers.

```
>>> L = [1, 2, 3, 4, 5]
>>> print(L[1:3])
[2, 3]
>>> print(L[3])
4
```

Note that `L[1:3]` did *not* include `L[3]`.

You may omit the starting position to slice elements from the beginning of the list up to the specified position. You may similarly omit the ending position to specify that a slice extends to the end of the list. You can even omit both of them to just get a copy of the whole list.

```
>>> L[:3]
[1, 2, 3]
>>> L[1:]
[2, 3, 4, 5]
>>> L[:]
[1, 2, 3, 4, 5]
```

Using slicing allows us to add or delete elements from lists:

```
>>> L = [1, 2, 3, 4, 5]
>>> L[2:4] = [93, 94, 95, 96]
>>> L
[1, 2, 93, 94, 95, 96, 5]
>>> L[3:6] = []
>>> L
[1, 2, 93, 5]
```

If we include an additional colon and value, we can include a step size. For example, a step size of two will create the list from every other value in the original list. A negative step size allows the list to be reversed.

Then, given a list `L` and two positions (indices) `B` and `E` as well as a step size `S`, you can use the expression `L[ B : E : S]` to obtain the elements of `L` between `B` and `E` positions in increments of `S` positions.

```
>>> values = [1, 2, 3, 4, 5]
>>> values[::2]
[1, 3, 5]
>>> values[1:4:2]
[2, 4]
>>> values[4:1:-1]
[5, 4, 3]
```

1.5 List Methods and Functions

Many functions exist to help manipulate lists. The first of these is `.append()`. This adds a new value onto the end of an existing list.

```
>>> values = [1, 2, 3]
>>> values.append(12)
>>> values.append(123)
>>> print(values)
[1, 2, 3, 12, 123]
```

`.insert()` also inserts a value, but it allows you to choose where it goes. The first argument of the function is the index you want your new value to be located. Other values will be moved to make room.

```
>>> values = [1, 2, 3, 4]
>>> values.insert(2, 10)
>>> print(values)
[1, 2, 10, 3, 4]
```

The `.pop()` function works similarly to accessing list values by index, but also removes the element from a list. If given a value, then `.pop()` will remove the value at that index. If not, it will remove the last value in the list.

```
>>> values = [1, 2, 3, 12, 123]
>>> print(values.pop())
123
>>> print(values.pop(0))
1
>>> print(values)
[2, 3, 12]
```

Using `len` returns the length of the list passed to it.

```
>>> values = [1, 2, 3, 12, 123]
>>> print(len(values))
5
```

`sum` allows the easy summing of every value in a list, so long as every value in the list is a number. If any values are not, an error will occur.

```
>>> values = [1, 2, 3, 12, 123]
>>> print(sum(values))
141
>>> values.append('test')
>>> print(sum(values))
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

1.6 Combining Lists

Two lists can be added together in order to combine them into one list.

```
>>> values = [1, 2, 3]
>>> more_values = [3, 2, 1]
>>> combined = values + more_values
>>> print(combined)
[1, 2, 3, 3, 2, 1]
```

Function/method	What it does
`lst.append(x)`	appends `x` to `lst`
`lst.insert(i, x)`	inserts `x` at index `i` in `lst` (moves other elements)
`x = lst.pop()`	removes last element of `lst` and places it in `x`
`lst.reverse()`	reverses the order of elements in list `lst` in place
`lst.sort()`	sort `lst` in place
`len(lst)`	number of elements in `lst`
`sum(lst)`	sum the elements of `lst` (only works when `+` works between the elements)

Table 1: List methods and functions, where `lst` is a variable that holds a list

1.7 Summary

- A list is created from a series of comma-separated values inside square brackets.
- An element in an array can be referenced and manipulated using its index. The first element has an index of 0. Negative indices can be used to reference elements from the end of the list, with the last element having an index of -1.
- The `in` keyword can be used to test if a value exists in a list, while `not in` can be used to test if a value does not exist in a list.
- Slicing can be used to create a new list from an existing one.
- See Table 1 for functions on lists.