Goals

• Know how to use the assignment operator on list elements and slices

• Know how to use the list methods `append`, and `extend`

• Know the definition of `mutability`, and which sequence types are `mutable` (lists) and `immutable` (strings, tuples)
Lists vs Strings: What's the difference?

1. Strings hold only characters, while lists can hold values of any type(s).

...haven't we seen this before?

**Tuples** are also objects that hold a sequence of values of any type(s).

("alpaca", 14, 27.6)
Lists vs Tuples: What's the difference?

Tuples are also objects that hold a sequence of values of any type(s).

Tuples are immutable: their contents cannot be changed.

Lists are mutable: their contents can be changed.

```python
a_tuple = ("a", 14, 27.6)
a_list = ["a", 14, 27.6]

a_tuple[1] # => 14
a_list[1] # => 14

a_tuple[1] = 0 # causes an error
a_list[1] = 0 # a_list is now ["a", 0, 27.6]
```
Lists are mutable

```python
a_list = ["a", 14, 27.6]
```

```
a_list  ➔ ["a", 14, 27.6]
```
Lists are mutable

```python
a_list = ['a', 14, 27.6]
a_list[0] = 'b'
a_list
```

```python
[ 'b', 14, 27.6]
```
Lists are mutable

```python
a_list = ["a", 14, 27.6]

a_list[0] = "b"

a_list.append(19)

append takes a single value and adds it to the end of the list.

a_list -> ["b", 14, 27.6, 19]```
Lists are mutable

```python
a_list = ["a", 14, 27.6]

a_list[0] = "b"

a_list.append(19)

a_list.append(["12", 2])

notice: still a single argument (happens to be a list)
```

```python
a_list = ["b", 14, 27.6, 19, ["12", 2]]
```
Lists are mutable

```python
a_list = ['a', 14, 27.6]

a_list[0] = 'b'

a_list.append(19)

a_list.append(['12', 2])

a_list.extend([22, 33])

extend takes a sequence and adds each value to the list.
```

```
a_list -> ['b', 14, 27.6, 19, ['12', 2], 22, 23]
```
Lists are mutable

Notice the difference between string methods and list methods:

```python
a_list.append(19)  # a_list = [19]
new_string = a_string.lower()  # new_string = "Jon"
```

```python
a_list  # [19]
new_string  # "Jon"
```
Lists are mutable

Notice the difference between string methods and list methods:

```python
a_list.append(19)  # a_list -> ['b', 19]

• **modifies** the list in-place
• has no return value
```

```python
new_string = a_string.lower()

a_string -> "JON"
```
Lists are mutable

Notice the difference between string methods and list methods:

```python
a_list.append(19)  # a_list = ["b", 19]
• modifies the list in-place
• has no return value

new_string = a_string.lower()  # a_string = "JON"
• does not modify a_string
• returns a lower-case copy
new_string = "jon"
```
Slicing, Revisited

\[ a = [5, 6, 7, 8] \]

Unlike list methods, slicing yields a new list. It does not modify the list.

\[ a[0:3] \# \Rightarrow [5, 6, 7] \]

\[ a \# \Rightarrow [5, 6, 7, 8] \]

Indexing yields a list element; slicing yields a sublist:

\[ a[1] \# \Rightarrow 6 \quad \text{← indexing yields a list element} \]

\[ a[1:2] \# \Rightarrow [6] \quad \text{← a list of length 1!} \]

\[ a[1:1] \# \Rightarrow [] \quad \text{← a list of length 0!} \]
List assignment + slicing

We can **assign** to indices:

\[
a = [5, 6, 7, 8]
\]

\[
a[0] = 10
\]

We can **slice** out sublists:

\[
a[0:3] \ # \Rightarrow [10, 6, 7]
\]

Can we **assign** to slices?

You betcha! (demo)
List assignment + slicing: Demo

a = [5, 6, 7, 8]
a[:2] = [3, 4]

a = [5, 6, 7, 8]
a[:3] = a[1:]

a = [5, 6, 7, 8]
a[:2] = a[1:]
Demo: What are lists good for?

• Generate a list of the fibonacci sequence
  • fib_list.py

• Make a deck of cards and deal a blackjack hand
  • blackjack.py

• Make a bale of turtles do some crazy stuff.
  • bale.py
Demo: a bale of turtles

• bale.py