Lists

Last class

What did we learn about last class?

What is a tuple?

Can individual elements of a tuple be changed?

The List

Today we're going to learn about a new data structure - the list.

Lists are very similar to tuples, but there is one major difference:
Where tuples are **immutable**, lists are **mutable**.

That means that we **are** able to change individual elements within a list, something we cannot do with a tuple.

Makin' Lists

Lists are created in almost the same exact way as tuples - we just use a different symbol. Here's how to define one:

```
my_list = [1, 5, 30]
```

Because lists are **mutable**, we are allowed to do things like this:

```
my_list[0] = "hi"
print(my_list)
```

Makin’ Lists

Lists are created in almost the same exact way as tuples - we just use a different symbol. Here’s how to define one:

```
my_list = [1, 5, 30]
```

Because lists are **mutable**, we are allowed to do things like this:

```
my_list[0] = "hi"
print(my_list)
```

```
["hi", 5, 30]
```

Making the immutable mutable

If we want to turn an **immutable** variable (a tuple or string) into a **mutable** variable, we can use the list() function.

This lets us change single elements within a variable we normally wouldn't be able to!

```python
word = "abc"
word_as_list = list(word)
print(word_as_list)
word_as_list[0] = "A"
print(word_as_list)
```

Making the immutable mutable

If we want to turn an **immutable** variable (a tuple or string) into a **mutable** variable, we can use the list() function.

This lets us change single elements within a variable we normally wouldn't be able to!

```
word = "abc"
word_as_list = list(word)
print(word_as_list)
word_as_list[0] = "A"
print(word_as_list)
```

```
["a", "b", "c"]
```

Making the immutable mutable

If we want to turn an **immutable** variable (a tuple or string) into a **mutable** variable, we can use the list() function.

This lets us change single elements within a variable we normally wouldn't be able to!

```
word = "abc"
word_as_list = list(word)
print(word_as_list)
word_as_list[0] = "A"
print(word_as_list)
```

```
["a", "b", "c"]
["A", "b", "c"]
```

Turning lists into strings

There's a new string function to do this: .join()

The function is called on a string, and uses a list as a parameter. Here's how it looks:

```
my_list = ["A", "b", "c"]
list_as_str = "".join(my_list)
list_as_str_2 = "@".join(my_list)
```

Turning lists into strings

There's a new string function to do this: .join()

The function is called on a string, and uses a list as a parameter. Here's how it looks:

```
my_list = ["A", "b", "c"]
list_as_str = "".join(my_list)
list_as_str_2 = "@".join(my_list)
```

```
Abc
```

Turning lists into strings

There's a new string function to do this: .join()

The function is called on a string, and uses a list as a parameter. Here's how it looks:

```
my_list = ["A", "b", "c"]
list_as_str = "".join(my_list)
list_as_str_2 = "@".join(my_list)
```

```
Abc
A@b@c
```

What .join() does

.join() will take each element in the list parameter, and concatenate them together with the string it was called on.

```
my_list = ["A", "b", "c"]
list_as_str = "".join(my_list)

    "A" + "" + "b" + "" + "c"
```

.join() will *only* work when each element within the list is a string.

What .join() does

.join() will take each element in the list parameter, and concatenate them together with the string it was called on.

```
my_list = ["A", "b", "c"]
list_as_str = "".join(my_list)

    "A" + "" + "b" + "" + "c"
```

.join() will *only* work when each element within the list is a string.

What .join() does

.join() will take each element in the list parameter, and concatenate them together with the string it was called on.

```
my_list = ["A", "b", "c"]
list_as_str = "".join(my_list)

    "A" + "" + "b" + "" + "c"
```

.join() will *only* work when each element within the list is a string.

Strings to Lists 2

There is another string function we can use to turn a string into a list: .split()

.split() takes between 0 and 1 parameters.

When we give 0 parameters, the string will break apart into different elements based on whitespace. When we pass 1 parameter, the string will break apart into different elements based on the string passed.

```
word = "ah be ce"
l1 = word.split()
word2 = "aa@bb@cc"
l2 = word2.split("@")
```

Strings to Lists 2

There is another string function we can use to turn a string into a list: .split()

.split() takes between 0 and 1 parameters.

When we give 0 parameters, the string will break apart into different elements based on whitespace. When we pass 1 parameter, the string will break apart into different elements based on the string passed.

```
word = "ah be ce"
l1 = word.split()
word2 = "aa@bb@cc"
l2 = word2.split("@")
```

```
["ah", "be", "ce"]
```

Strings to Lists 2

There is another string function we can use to turn a string into a list: .split()

.split() takes between 0 and 1 parameters.

When we give 0 parameters, the string will break apart into different elements based on whitespace. When we pass 1 parameter, the string will break apart into different elements based on the string passed.

```
word = "ah be ce"
l1 = word.split()
word2 = "aa@bb@cc"
l2 = word2.split("@")
```

```
["ah", "be", "ce"]
["aa", "bb", "cc"]
```