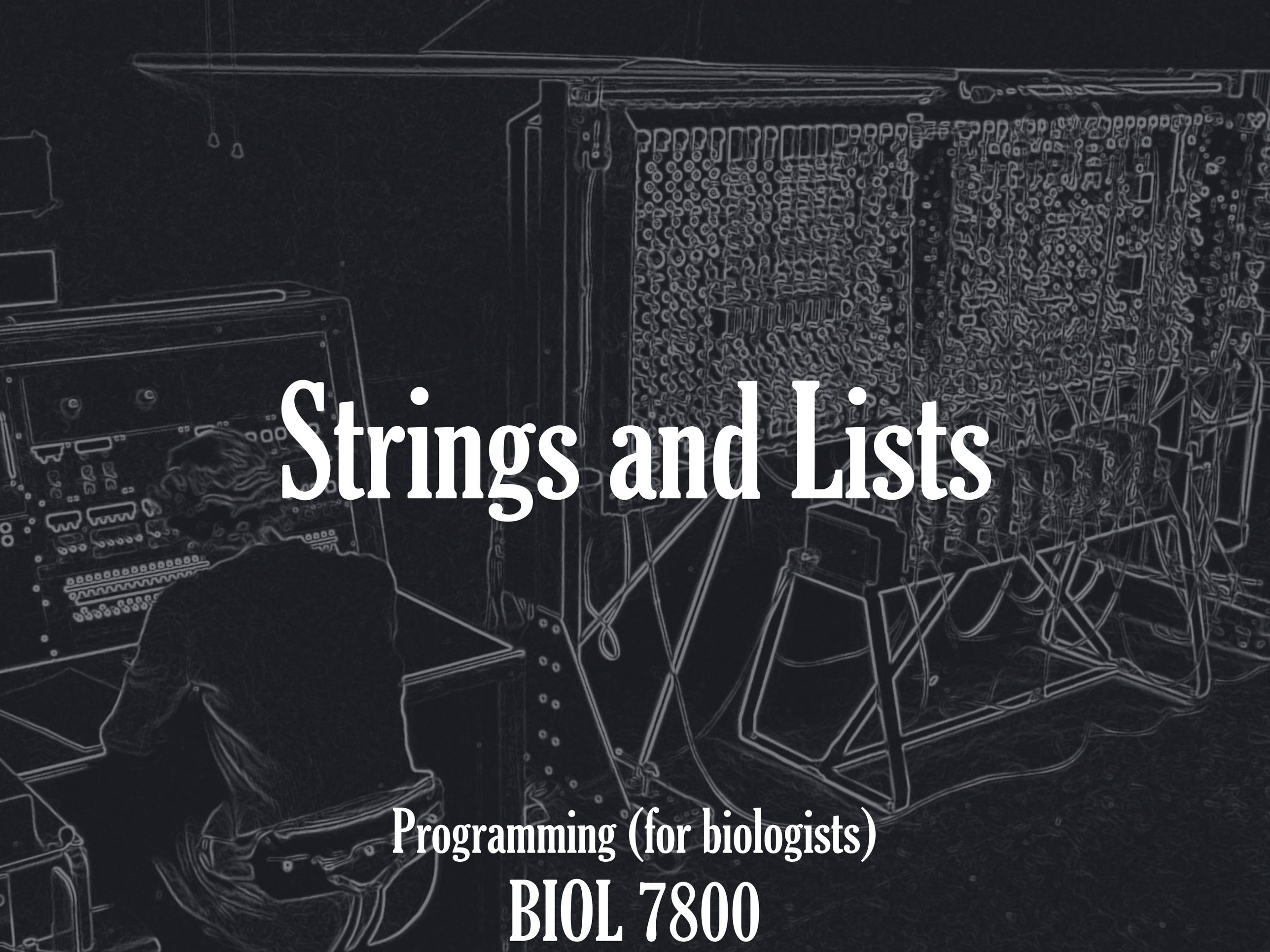


Strings and Lists

Programming (for biologists)

BIOL 7800

Strings

A **string** is a **sequence** of characters

‘Able was I ere I saw Elba’

These are both “strings”

‘1 2 3 4 5 6 7’

Strings

A **string** is a **sequence** of characters

Because a string is a sequence you can:

my_string = ‘Able was I ere I saw Elba’

Iterate over the sequence

```
In:   for letter in my_string:
          print(letter)
Out: ‘A’
     ‘b’
     ‘l’
     ‘e’
```

Strings

A **string** is a **sequence** of characters

Because a string is a sequence you can:

my_string = ‘Able was I ere I saw Elba’

Reverse the sequence

```
In:  print(my_string[::-1])
Out: ‘ablE was I ere I saw elbA’
```

Strings

A **string** is a **sequence** of characters

Because a string is a sequence you can:

my_string = 'Able was I ere I saw Elba'

Find the length of the sequence

```
In:  len(my_string)
Out: 25
```

Strings

A **string** is a **sequence** of characters

Because a string is a sequence you can:

my_string = ‘gorilla’

Access elements of the sequence

In: **my_string[1]**
Out: ‘o’

Accessing an element

g	o	r	i	l	l	a
0	1	2	3	4	5	6
-7	-6	-5	-4	-3	-2	-1

Strings

Because a string is a sequence you can also **slice** the sequence

my_string = 'gorilla'

Slicing goes from the m^{th} character (inclusive) to the n^{th} character (not inclusive)

Think of slicing as operating on indices **between** letters

Accessing an element

g	o	r	i	l	l	a
0	1	2	3	4	5	6
-7	-6	-5	-4	-3	-2	-1

Slicing a sequence

g	o	r	i	l	l	a	
0	1	2	3	4	5	6	7
-8	-7	-6	-5	-4	-3	-2	-1

Strings

Slicing goes from the m^{th} character (inclusive)
to the n^{th} character (not inclusive)

Think of slicing as operating on indices **between** letters

Slicing a
sequence

my_string = | g | o | r | i | l | l | a |

0	1	2	3	4	5	6	7
-8	-7	-6	-5	-4	-3	-2	-1

What do we get with:

In: **my_string**[1:5]
Out: ????

In: **my_string**[-6:-2]
Out: ????

Strings

Slicing goes from the m^{th} character (inclusive)
to the n^{th} character (not inclusive)

Think of slicing as operating on indices **between** letters

Slicing a
sequence

my_string = | g | o | r | i | l | l | a |

0	1	2	3	4	5	6	7
-8	-7	-6	-5	-4	-3	-2	-1

What do we get with:

In: my_string[-2:-6]
Out: ????

In: my_string[::-2]
Out: ????

In: my_string[::-3]
Out: ????

Strings

Strings are immutable - meaning, we **cannot** change them directly

my_string = ‘gorillas like bananas’

What is:

my_string[0:8]

Strings

Strings are immutable - meaning, we **cannot** change them directly

my_string = ‘gorillas like bananas’

So,

my_string[0:8] == ‘gorillas’

Let’s just change that one word,

my_string[0:8] = ‘monkeys’

TypeError: 'str' object does not support item assignment

How might we change ‘gorillas’ to ‘monkeys’?

Strings

Strings are immutable - meaning, we **cannot** change them directly

How might we change ‘gorillas’ to ‘monkeys’?

```
my_string = ‘gorillas like bananas’
```

```
my_new_string = ‘monkeys’ + my_string[8:]
```

Strings

We can do lots of stuff with strings….

We can count letters

example8.py — /Users/bcf/Dropbox (faircloth-lab)/Classes/BIOL...

example8.py

```
1  word = 'banana'
2  count = 0
3  for letter in word:
4      if letter == 'a':
5          count = count + 1
6  print(count)
7
```

File 0 Project 0 ✔ No Issues example8.py* 7:1 LF UTF-8 Python

We can find words

example8.py — /Users/bcf/Dropbox (faircloth-lab)/Classes/BIOL...

example8.py

```
1  def find(word, letter):
2      index = 0
3      while index < len(word):
4          if word[index] == letter:
5              return index
6          index = index + 1
7      return False
8
```

File 0 Project 0 ✔ No Issues example8.py* 8:1 LF UTF-8 Python

Strings

But **strings** are also what are known as "**objects**"

object - a defined *class* of a *certain type*

And, as **objects** have their own "**methods**"

where **methods** are basically *functions* that operate

only on an object of the string class

Strings

But **strings** are also what are known as "**objects**"

object - a defined *class* of a *certain type*

And, as **objects** have their own "**methods**"

where **methods** are basically *functions* that operate *only on an object of the string class*

my_string = 'gorilla'

my_string.upper() = 'GORILLA'

The **.upper()** method

We say we "invoke" **upper()** on **my_string**.

Strings

As **objects**, **strings** have lots of **methods**

How do we show the **attributes** of an object?

(**attributes** of an object include the **methods**)

my_string = 'gorilla'

dir(my_string)

```
[...
'capitalize',
 'casefold',
 'center',
 'count',
 'encode',
 'endswith',
 'expandtabs',
 'find',
 'format',
...
]
```

Strings

We can do lots of stuff with strings….

my_string = '**this is my string**'

We can count letters

example8.py — /Users/bcf/Dropbox (faircloth-lab)/Classes/BIOL...

example8.py

```
1  word = 'banana'
2  count = 0
3  for letter in word:
4      if letter == 'a':
5          count = count + 1
6  print(count)
7
```

File 0 Project 0 ✔ No Issues example8.py* 7:1 LF UTF-8 Python

We can find words

example8.py — /Users/bcf/Dropbox (faircloth-lab)/Classes/BIOL...

example8.py

```
1  def find(word, letter):
2      index = 0
3      while index < len(word):
4          if word[index] == letter:
5              return index
6          index = index + 1
7      return False
8
```

File 0 Project 0 ✔ No Issues example8.py* 8:1 LF UTF-8 Python

Strings

We can do lots of stuff with strings….

my_string = '**this is my string**'

We can count letters

example8.py — /Users/bcf/Dropbox (faircloth-lab)/Classes/BIOL...

example8.py

```
1  word = 'banana'
2  count = 0
3  for letter in word:
4      if letter == 'a':
5          count = count + 1
6  print(count)
7
```

File 0 Project 0 ✔ No Issues example8.py* 7:1 LF UTF-8 Python

We can find words

example8.py — /Users/bcf/Dropbox (faircloth-lab)/Classes/BIOL...

example8.py

```
1  def find(word, letter):
2      index = 0
3      while index < len(word):
4          if word[index] == letter:
5              return index
6          index = index + 1
7      return False
8
```

File 0 Project 0 ✔ No Issues example8.py* 8:1 LF UTF-8 Python

But, there are already **methods** to do the above…

In: **my_string.count('i')**
Out: **3**

In: **my_string.find('my')**
Out: **8**

Strings

We can do lots of stuff with strings….

my_string = '**this is my string**'

One of the most useful string methods is **.split()**

There's a space in here…

In: my_string.split(' ')
Out: ['this', 'is', 'my', 'string']

Why would **this** be **useful**?

Strings

Why would **this** be **useful**?

Delimited files!

my_line_1 = ‘**index,name,species,quantity**’
my_line_2 = ‘**A2455,bob,canis,2**’
my_line_3 = ‘**A2456,steve,felis,1**’

my_line_1.split(‘,’)

['index', 'name', 'species', 'quantity']

my_line_2.split(‘,’)

['A2455', 'bob', 'canis', '2']

my_line_3.split(‘,’)

['A2456', 'steve', 'felis', '1']

Strings

We can do lots of stuff with strings….

my_string = ‘**this is my string\n**’

Another *very* useful string methods is **.strip()**

Removes newlines (of all types)

In: **my_string.strip()**

Out: ‘this is my string’

There’s a space in here…

Removes other chars

In: **my_string.strip(‘ ’)**

Out: ‘thisismystring’

Strings

We can test for "membership" in a string using **in** operator

my_string = '**this is a good string**'

In: 'good' in my_string
Out: True

And we can compare strings using **==** operator

In: **my_string** **==** '**this is a good string**'
Out: True

In: **my_string** **==** '**dogs**'
Out: False

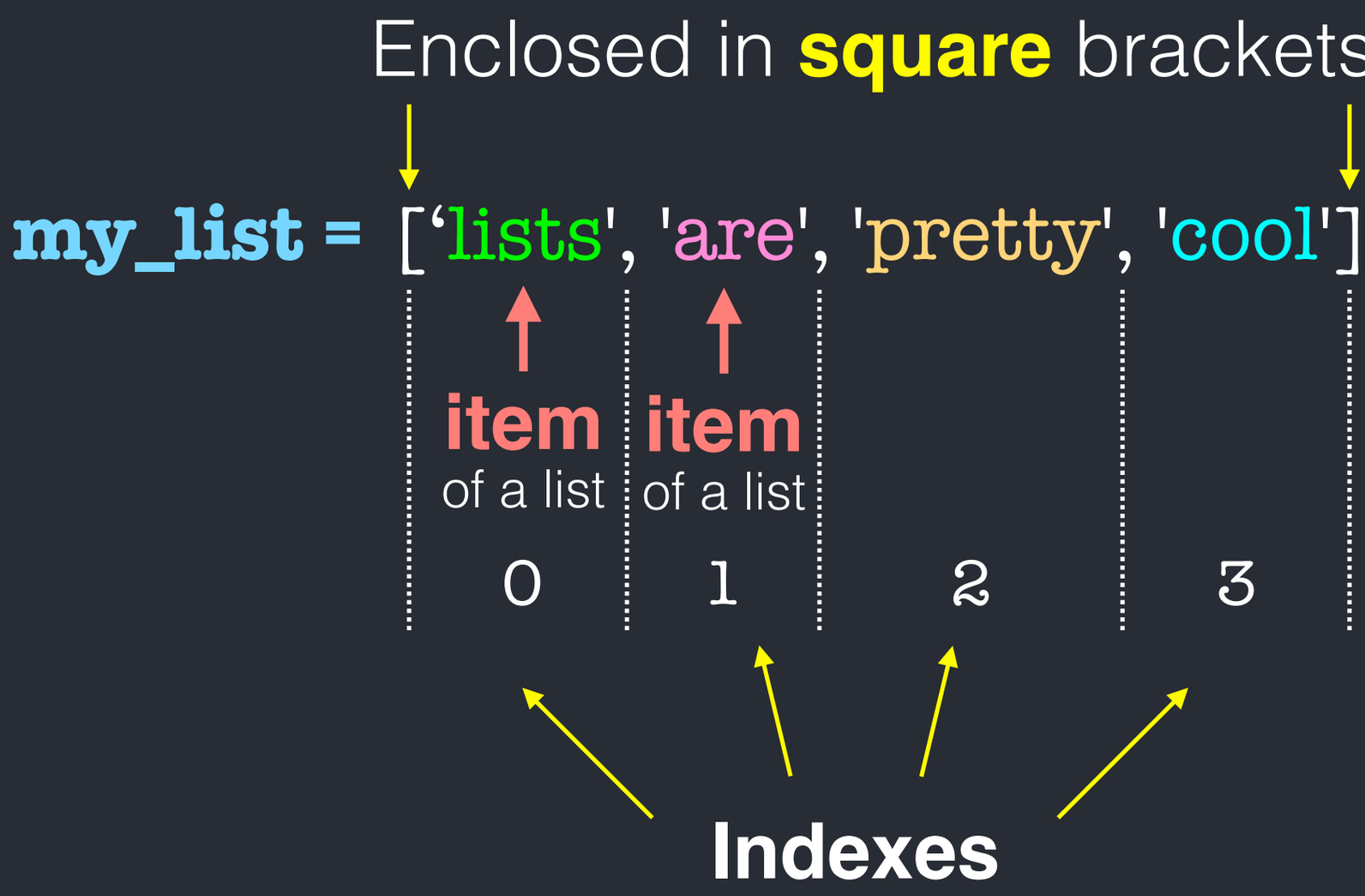
Lists

In: **my_string.split(‘ ’)**
Out: [‘this', 'is', 'my', 'string']

List

Like a string, a **list** is also a sequence of values, but values can be of any **type** (not just characters)

Anatomy of a List



You can access an element

my_list[3] = 'cool'

Anatomy of a List

Enclosed in **square** brackets

You can slice a list

```
my_list[2:4] = ['pretty', 'cool']
```

Note: this returns another list

Creating a List

my_list = []

Set a variable to opposing square brackets
to create empty list

my_list = [‘dog’, ‘cat’, ‘mouse’, ‘rat’]

Type in list entries between square brackets

my_list = list(‘dog’)

“listify” a string

['d', 'o', 'g']

Creating a List

Lists can also be “nested”

And may contain different **types**

```
my_list = [[1,2,3], [‘cat’,‘dog’], [4.6]]
```

```
my_list[0] = [1,2,3]
```

What does **this** return?

```
my_list[0][0] = ??
```

Lists are mutable

Unlike strings, you can modify list elements

```
In:   my_list = ['dog', 'cat', 'mouse', 'rat']
In:   my_list[1] = 'hamster'
In:   print(my_list)
Out: ['dog', 'hamster', 'mouse', 'rat']

In:   my_list.pop(2)
In:   print(my_list)
```

What does this print?

Lists are mutable

Unlike strings, you can modify list elements

```
In:   my_list = ['dog', 'cat', 'mouse', 'rat']
In:   my_list[1] = 'hamster'
In:   print(my_list)
Out: ['dog', 'hamster', 'mouse', 'rat']

In:   my_list.pop(2)
Out: 'mouse'
In:   print(my_list)
Out: ['dog', 'hamster', 'rat']
```

What does this print?

List operations

You can **add** (concatenate) lists

In: `['a','b','c'] + ['1','2','3']`
Out: `['a','b','c', '1','2','3']`

You can **multiply** (repeat) lists

In: `['a'] * 6`
Out: `['a','a','a','a','a','a']`

Lists

Lists are also "**objects**"
object - a defined *class* of a *certain type*

And, as **objects,** lists have their own **"methods"**
where **methods** are basically *functions* that operate
only on an object of the string class

Creating a List

One of the most useful list method is **.append()**

In: my_list = []

In: my_list.append(‘dog’)
In: my_list
Out: [‘dog’]

In: my_list.append(‘cat’)
In: my_list
Out: [‘dog’, ‘cat’]

In: my_list.append(‘rat’)
In: my_list
Out: [‘dog’, ‘cat’, ‘rat’]

Creating a List

Another useful list method is **.extend()**

```
In: list_1 = [1, 2, 3]
In: list_2 = [4, 5, 6]
In: list_1.append(list_2)
Out: [1, 2, 3, [4, 5, 6]]
```

Not quite what we wanted…

```
In: list_1 = [1, 2, 3]
In: list_2 = [4, 5, 6]
In: list_1.extend(list_2)
Out: [1, 2, 3, 4, 5, 6]
```

Functions just like concatenation (**+**)

Sorting a List

Another useful list method is **.sort()**

sort() does exactly what you think it will

```
In:   list_1 = ['z', 'b', 'a', 'n', 'm']
In:   list_1.sort()
In:   print(list_1)
Out: ['a', 'b', 'm', 'n', 'z']
```

Joining a List

Another useful list method is **join()**

join() is the opposite of the string method split()
join() is also a string method

```
In:   list_1 = ['z', 'b', 'a', 'n', 'm']
In:   ''.join(list_1)
Out: 'zbanm'

In:   ':'.join(list_1)
Out: 'z:b:a:n:m'
```