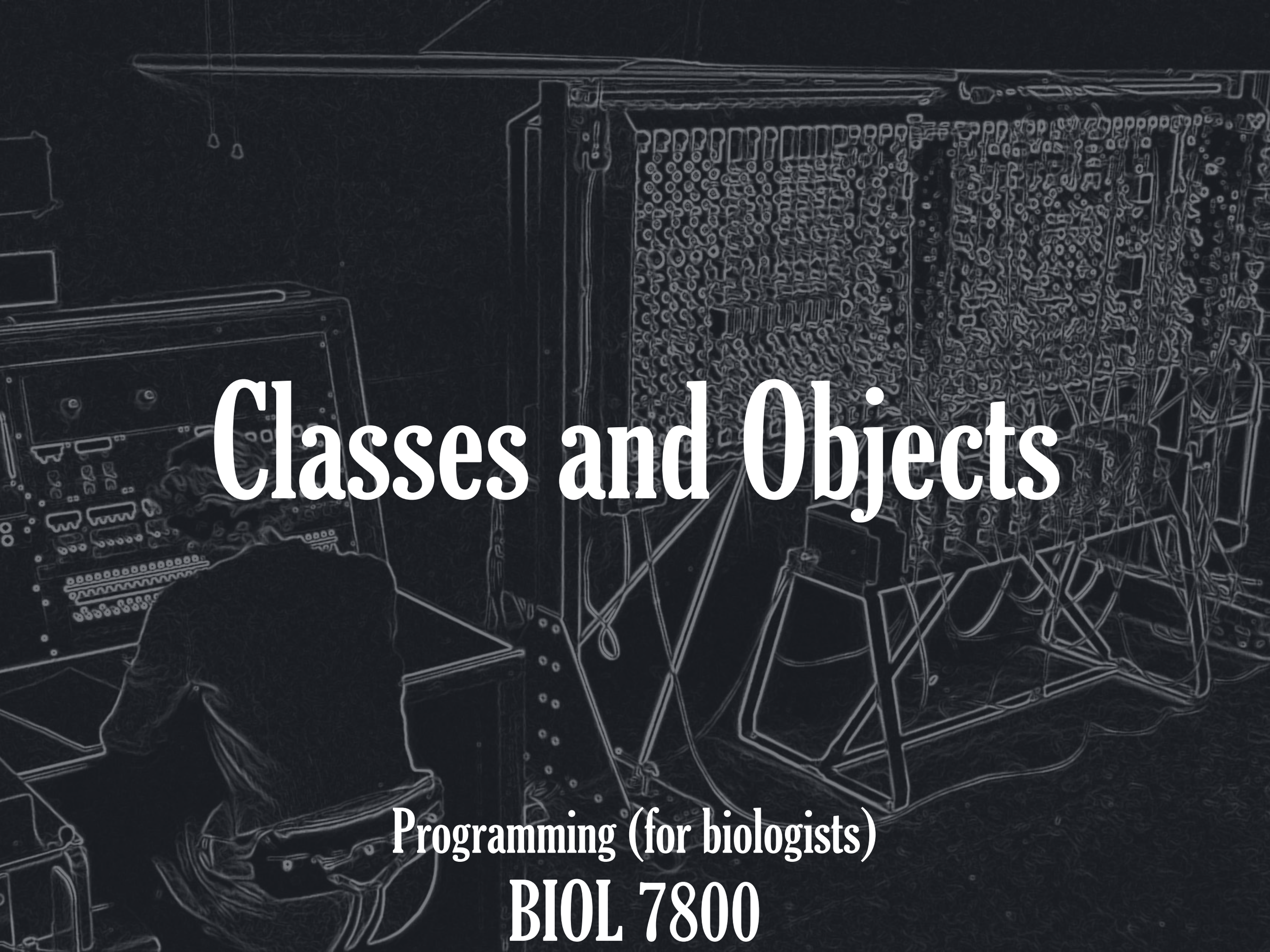


Classes and Objects

Programming (for biologists)

BIOL 7800

Strings

Strings are what are known as "**objects**"

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

And, as **objects, strings** have their own
"methods" and "attributes"

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

and **attributes** are *values* associated with *named elements*
of an object of the string class

Strings

But

object

And, as

where

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

```
my_string = ‘gorilla’
dir(my_string)
```

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

Lists

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

And, as **objects, lists** have their own
"methods" and "attributes"

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

and **attributes** are *values* associated with *named elements*
of an object of the list class

Dictionary

```
my_pets = {'dogs': 2, 'cats':3, 'hamsters':1}
```

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

And, as **objects, dictionaries** have their own
"methods" and "attributes"

where **methods** are basically *functions* that operate
only on an object of the dict class
and **attributes** are *values* associated with *named elements*
of an object of the dict class

Tuple

my_tuple = (‘this’, 'is', 'my', ‘tuple’)

Tuples are also“**objects**”
object - a defined *class* of a *certain type*

And, as **objects, tuples** have their own
“methods” and “attributes”

where **methods** are basically *functions* that operate
only on an object of the tuple class
and **attributes** are *values* associated with *named elements*
of an object of the tuple class

But what is a damn “object”?

It is a way of encapsulating a certain **type** or **class** of data

…along with attributes that make up that type of data

(you could think of these as “metadata”)

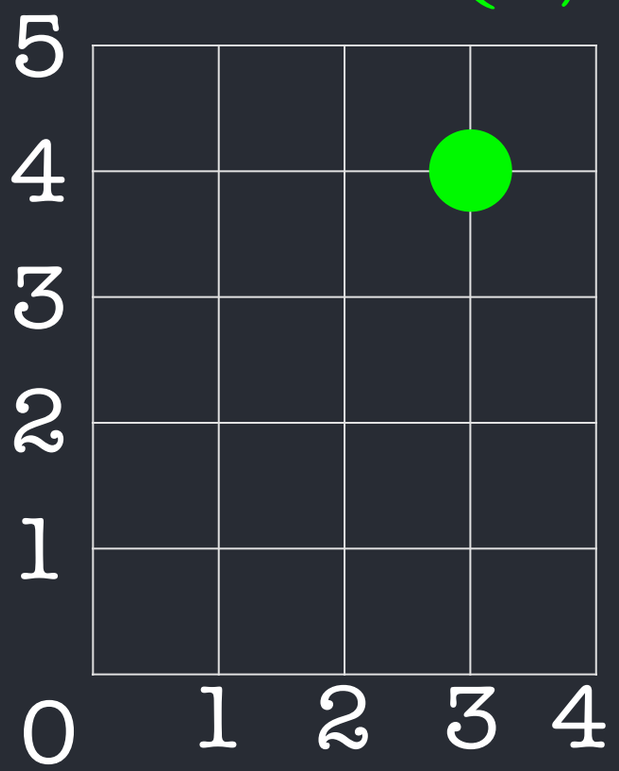
…and methods that operate on that type of data

(you could think of these as object-specific functions)

And, what is a damn "class"?

A `class` is essentially a user-defined object (or type)
We can create new `Class`es to define new types

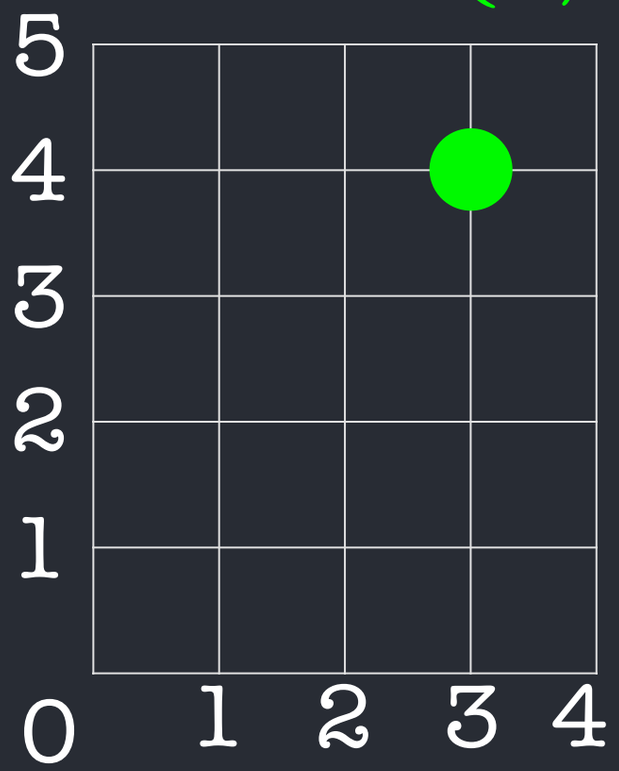
(3, 4) ← A point



How might you represent this in Python?

And, what is a damn "class"?

How might you represent this in Python?



(3, 4) ← A point

variables { x = 3, y = 4

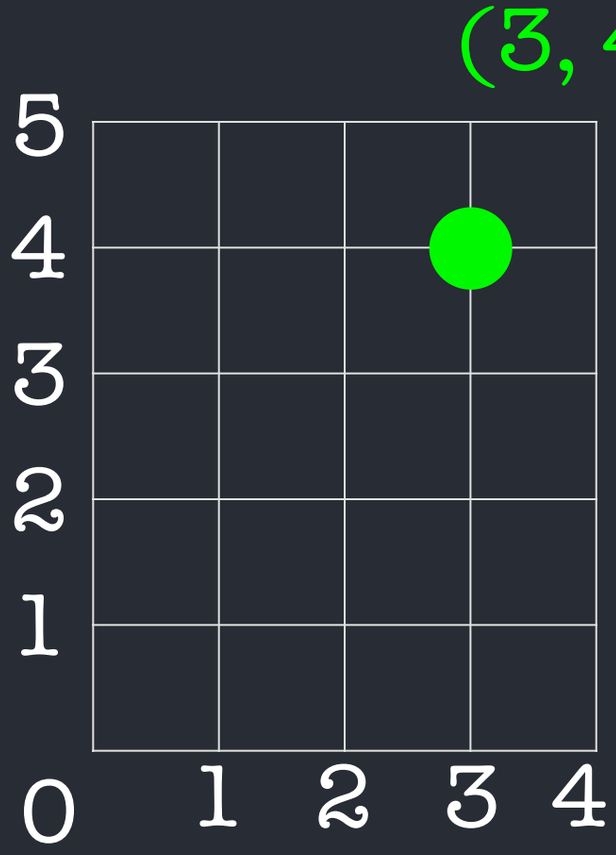
list `point = [3, 4]`

tuple `point = (3, 4)`

dictionary `point = {'x':3, 'y':4)`

But there is another option…

We can define a `class` to represent points



`class` keyword

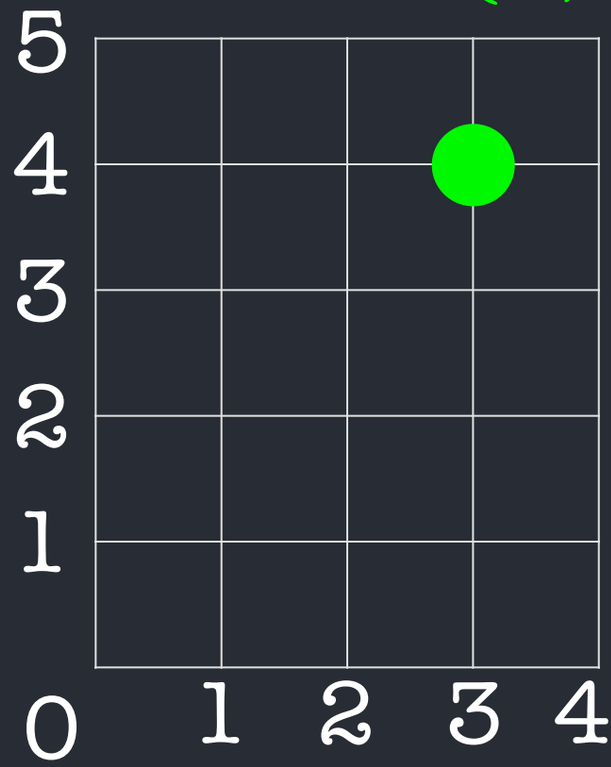
`class` name

```
class Point():
    '''A class to hold point data'''
    # other stuff to do w/ class goes below
```

`class` docstring

`class` body (attributes, methods)

We can define a **class** to represent points




```
class Point():
    '''A class to hold point data'''
    # other stuff to do w/ class goes below
```

This `Point()` class allows us to create `Point()` objects that have their own **"methods" and "attributes"**

where **methods** are basically *functions* that operate only on an object of the `Point()` class

…and **attributes** are *values* assoc. with *named elements of an object of the* `Point()` *class*

Defining and using Point() class

and creating a Point() object

example.py — /Users/bcf/Dropbox (faircloth-lab)/Classes/BIOL7800/temp

example.py

```
1   #!/usr/bin/env python
2   # encoding: utf-8
3
4
5   class Point():
6       '''A class to hold point data'''
7       # nothing here yet
8
9
10  def main():
11      my_point = Point()
12      print("Output of type(my_point): ", type(my_point))
13
14
15  if __name__ == '__main__':
16      main()
17
```

define Point() class

create a Point object

aka "instantiate" a Point object

my_point object is an "instance" of the Point() class

$ python example.py
Output of type(my_point): <class '__main__.Point'>

Defining and using Point() class

and creating a Point() object

example.py — /Users/bcf/Dropbox (faircloth-lab)/Classes/BIOL7800/temp

example.py

```
1   #!/usr/bin/env python
2   # encoding: utf-8
3
4
5   class Point():
6       '''A class to hold point data'''
7       # nothing here yet
8
9
10  def main():
11      my_point = Point()
12      print("Output of print(my_point): ", my_point)
13
14
15  if __name__ == '__main__':
16      main()
17
```

define Point() class

create a Point object

aka "instantiate" a Point object

my_point object is an "instance" of the Point() class

$ python example.py

Output of print(my_point): <__main__.Point object at 0x1021d8dd8>

Defining and using Point() class

and creating a Point() object

example.py — /Users/bcf/Dropbox (faircloth-lab)/Classes/BIOL7800/temp

example.py

```
1   #!/usr/bin/env python
2   # encoding: utf-8
3
4
5   class Point():
6       '''A class to hold point data'''
7       # nothing here yet
8
9
10  def main():
11      my_point = Point()
12      print("Output of dir(my_point): ", dir(my_point))
13
14
15  if __name__ == '__main__':
16      main()
```

Where'd all that come from?

$ python example.py

Output of dir(my_point): ['__class__', '__delattr__', '__dict__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '__gt__', '__hash__', '__init__', '__le__', '__lt__', '__module__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__setattr__', '__sizeof__', '__str__', '__subclasshook__', '__weakref__']

Defining and using Point() class

and creating several Point() objects

example.py — /Users/bcf/Dropbox (faircloth-lab)/Classes/BIOL7800/temp

example.py

```
1   #!/usr/bin/env python
2   # encoding: utf-8
3
4
5   class Point():
6       '''A class to hold point data'''
7       # nothing here yet
8
9
10  def main():
11      my_point_1 = Point()
12      my_point_2 = Point()
13      my_point_3 = Point()
14      for item in [my_point_1, my_point_2, my_point_3]:
15          print("Output of print(my_point): ", item)
16
17
18  if __name__ == '__main__':
19      main()
```

instantiate several Point objects

each my_point object is a differen5 "instance" of the Point() class

$ python example.py

each instance of Point() has a diff. RAM location

```
Output of print(my_point): <__main__.Point object at 0x1019d8c88>
Output of print(my_point): <__main__.Point object at 0x1019d8cf8>
Output of print(my_point): <__main__.Point object at 0x1019eada0>
```

Attributes

We can "assign" attributes after we create an instance of Point() using dot notation

example.py — /Users/bcf/Dropbox (faircloth-lab)/Classes/BIOL7800/temp

example.py

```
1   #!/usr/bin/env python
2   # encoding: utf-8
3
4
5   class Point():
6       '''A class to hold point data'''
7       # nothing here yet
8
9
10  def main():
11      my_point = Point()
12      my_point.x = 3
13      my_point.y = 4
14      print("x = ", my_point.x)
15      print("y = ", my_point.y)
16
17
18  if __name__ == '__main__':
19      main()
```

the x attribute of my_point

the y attribute of my_point

where **attributes** are *values* assoc. with *named elements of an object of the* Point() *class*

```
$ python example.py
x = 3
y = 4
```

Attributes

We can "assign" attributes after we create an instance of `Point()` using dot notation

example.py — /Users/bcf/Dropbox (faircloth-lab)/Classes/BIOL7800/temp

example.py

```
1   #!/usr/bin/env python
2   # encoding: utf-8
3
4
5   class Point():
6       '''A class to hold point data'''
7       # nothing here yet
8
9
10  def main():
11      my_point = Point()
12      my_point.x = 3
13      my_point.y = 4
14      print("x = ", my_point.x)
15      print("y = ", my_point.y)
16
17
18  if __name__ == '__main__':
19      main()
```

Now, `my_point` encapsulates our point values in its `x` and `y` attributes

```
$ python example.py
x = 3
y = 4
```

Attributes

We can "assign" attributes to different instances of the `Point()` class

example.py — /Users/bcf/Dropbox (faircloth-lab)/Classes/BIOL7800/temp

example.py

```
1   #!/usr/bin/env python
2   # encoding: utf-8
3
4
5   class Point():
6       '''A class to hold point data'''
7       # nothing here yet
8
9
10  def main():
11      my_point = Point()
12      my_point.x, my_point.y = 3, 4
13      print("x = {}; y = {}".format(my_point.x, my_point.y))
14      my_other_point = Point()
15      my_other_point.x, my_other_point.y = 100, 300
16      print("x = {}; y = {}".format(my_other_point.x, my_other_point.y))
17
18
19  if __name__ == '__main__':
20      main()
```

One instance of `Point()`

Encapsulates the values (3, 4)

Another instance of `Point()`

Encapsulates different values of (100, 300)

```
$ python example.py
x = 3; y = 4
x = 100; y = 300
```

Objects

You can pass around an instance of the Point() object as you would expect

example.py — /Users/bcf/Dropbox (faircloth-lab)/Classes/BIOL7800/temp

example.py

```
1   #!/usr/bin/env python
2   # encoding: utf-8
3
4
5   class Point():
6       '''A class to hold point data'''
7       # nothing here yet
8
9
10  def print_point(p):
11      print("x = {}; y = {}".format(p.x, p.y))
12
13
14  def main():
15      my_point = Point()
16      my_point.x, my_point.y = 3, 4
17      print_point(my_point)
18
19  if __name__ == '__main__':
20      main()
```

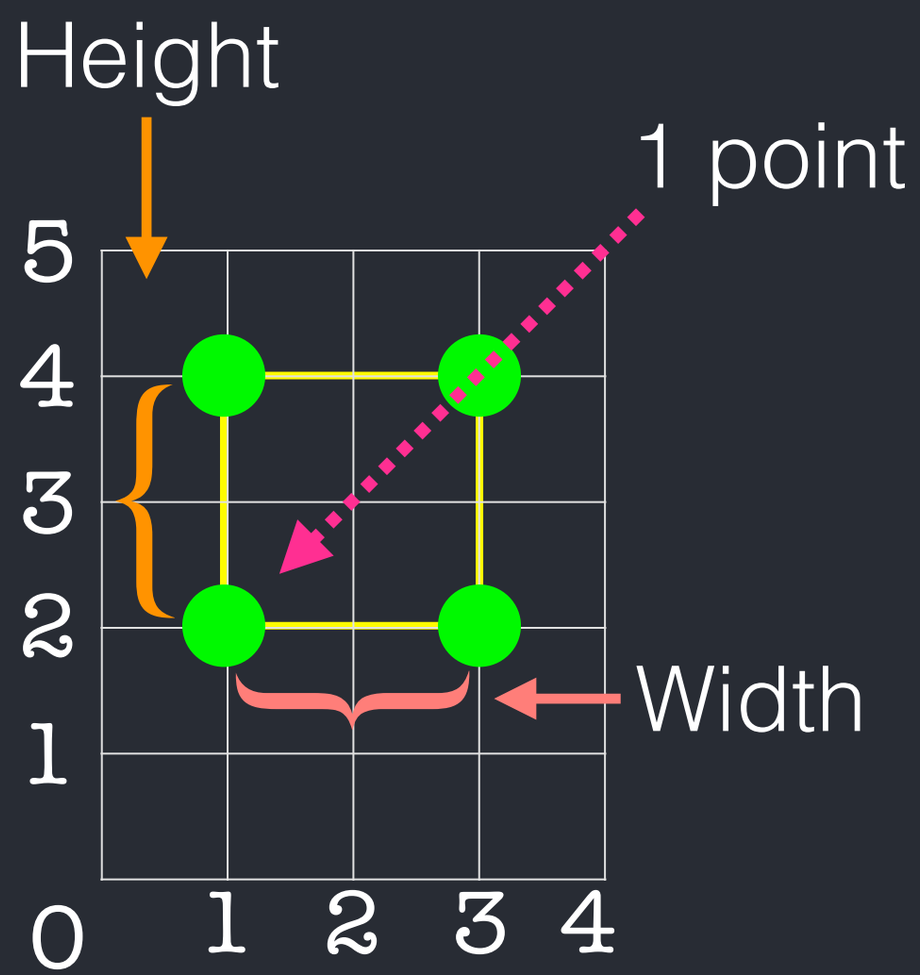
$ python example.py

x = 3; y = 4

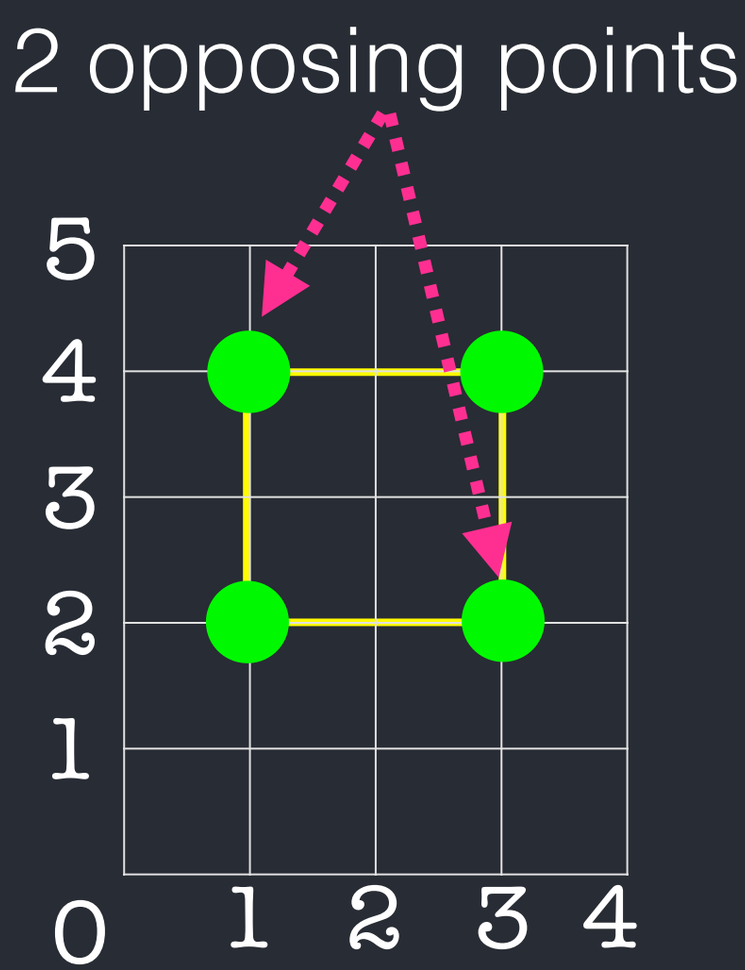
Rectangle Class

Let's also create a `Rectangle()` class
One of the tricky things about this is that
we can specify a rectangle by:

Approach #1



Approach #2

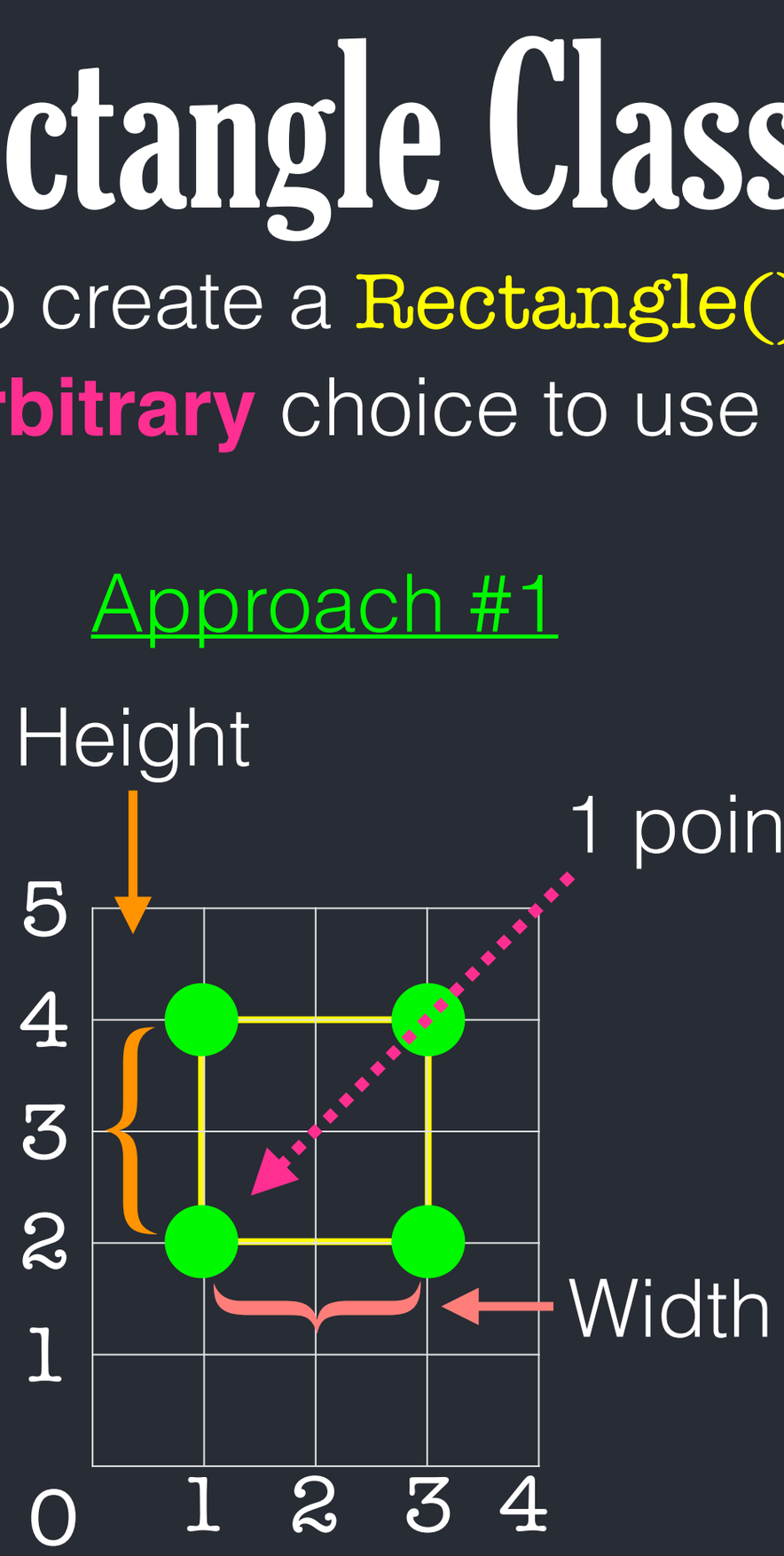


Rectangle Class

Let's also create a `Rectangle()` class

We make the **arbitrary** choice to use Approach 1

Approach #1



Rectangle Class

example2.py — /Users/bcf/Dropbox (faircloth-lab)/Classes/BIOL7800/temp

example2.py

```
1   #!/usr/bin/env python
2   # encoding: utf-8
3
4
5   class Point():
6       '''A class to hold point data'''
7       # nothing here yet
8
9
10  class Rectangle():
11      '''Reps a rect.  has width, height, corner'''
12      # nothing here yet
13
14
15  def main():
16      my_rect = Rectangle()
17      my_rect.width = 2
18      my_rect.height = 2
19      my_rect.corner = Point()
20      my_rect.corner.x = 1
21      my_rect.corner.y = 2
22
23  if __name__ == '__main__':
24      main()
```

Instantiate my_rect

assign the width attribute of my_rect

assign the height attribute of my_rect

assign the corner attribute of my_rect to Point()

assign the x and y attributes of my_rect.corner (which is a Point() object)

Rectangle Class

example2.py — /Users/bcf/Dropbox (faircloth-lab)/Classes/BIOL7800/temp

example2.py

```
#!/usr/bin/env python
# encoding: utf-8


class Point():
    '''A class to hold point data'''
    # nothing here yet


class Rectangle():
    '''Reps a rect.  has width, height, corner'''
    # nothing here yet

def main():
    my_rect = Rectangle()
    my_rect.width = 2
    my_rect.height = 2
    my_rect.corner = Point()
    my_rect.corner.x = 1
    my_rect.corner.y = 2
    print("this is dir(my_rect)", dir(my_rect))
```

```
$ python example2.py
this is dir(my_rect) ['__class__', '__delattr__', '__dict__', '__dir__', '__doc__', '__eq__
'__format__', '__ge__', '__getattribute__', '__gt__', '__hash__', '__init__', '__le__',
'__lt__', '__module__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__',
'__setattr__', '__sizeof__', '__str__', '__subclasshook__', '__weakref__', 'corner',
'height', 'width']
```

A function can return an instance

example2.py — /Users/bcf/Dropbox (faircloth-lab)/Classes/BIOL7800/temp

example2.py

```
#!/usr/bin/env python


class Point():
    '''A class to hold point data'''
    # nothing here yet


class Rectangle():
    '''Reps a rect.  has width, height, corner'''
    # nothing here yet


def find_center(rect):
    p = Point()
    p.x = rect.corner.x + rect.width/2
    p.y = rect.corner.y + rect.height/2
    return p


def main():
    my_rect = Rectangle()
    my_rect.width, my_rect.height = 2, 2
    my_rect.corner = Point()
    my_rect.corner.x, my_rect.corner.y = 1, 2
    result = find_center(my_rect)
    print(result)
    print(result.x, result.y)


if __name__ == '__main__':
    main()
```

```
$ python example2.py
<__main__.Point object at 0x10181e3c8>
2.0 3.0
```

This seems inefficient…

example2.py — /Users/bcf/Dropbox (faircloth-lab)/Classes/BIOL7800/temp

example2.py

```
#!/usr/bin/env python


class Point():
    '''A class to hold point data'''
    # nothing here yet


class Rectangle():
    '''Reps a rect.  has width, height, corner'''
    # nothing here yet


def find_center(rect):
    p = Point()
    p.x = rect.corner.x + rect.width/2
    p.y = rect.corner.y + rect.height/2
    return p


def main():
    my_rect = Rectangle()
    my_rect.width, my_rect.height = 2, 2
    my_rect.corner = Point()
    my_rect.corner.x, my_rect.corner.y = 1, 2
    result = find_center(my_rect)
    print(result)
    print(result.x, result.y)


if __name__ == '__main__':
    main()
```

But, all of this is sort of **cumbersome**, and seems **inefficient**…

And, really, there are more **efficient** and **clear** ways of working with objects

1. take control of object initialization

example2.py — /Users/bcf/Dropbox (faircloth-lab)/Classes/BIOL7800/temp

example2.py

```
1   #!/usr/bin/env python
2
3
4   class Point():
5       '''A class to hold point data'''
6       # nothing here yet
7
8
9   class Rectangle():
10      '''Reps a rect.  has width, height, corner'''
11      def __init__(self, width, height, x, y):
12          self.width = width
13          self.height = height
14          self.corner = Point()
15          self.corner.x = x
16          self.corner.y = y
17
18
19  def main():
20      my_rect = Rectangle(2, 2, 1, 2)
21      print(my_rect)
22      print("this is dir(my_rect)", dir(my_rect))
23      print("my_rect.width = {}, my_rect.height = {}".format(
24          my_rect.width, my_rect.height)
25      )
26
27
28  if __name__ == '__main__':
29      main()
```

__init__ is a "special" method that is called when an object is instantiated

It takes arguments, which we assign to attributes

1. take control of object initialization

But, what's all this self. stuff?

example2.py — /Users/bcf/Dropbox (faircloth-lab)/Classes/BIOL7800/temp

example2.py

```
1   #!/usr/bin/env python
2
3
4   class Point():
5       '''A class to hold point data'''
6       # nothing here yet
7
8
9   class Rectangle():
10      '''Reps a rect.  has width, height, corner'''
11      def __init__(self, width, height, x, y):
12          self.width = width
13          self.height = height
14          self.corner = Point()
15          self.corner.x = x
16          self.corner.y = y
17
18
19  def main():
20      my_rect = Rectangle(2, 2, 1, 2)
21      print(my_rect)
22      print("this is dir(my_rect)", dir(my_rect))
23      print("my_rect.width = {}, my_rect.height = {}".format(
24          my_rect.width, my_rect.height)
25      )
26
27
28  if __name__ == '__main__':
29      main()
```

1. take control of object initialization

But, what's all this self. stuff?

The first self. denotes that __init__ is a method of the Rectangle() class

example2.py — /Users/bcf/Dropbox (faircloth-lab)/Classes/BIOL7800/temp

example2.py

```
1  #!/usr/bin/env python
2
3
4  class Point():
5      '''A class to hold point data'''
6      # nothing here yet
7
8
9  class Rectangle():
10     '''Reps a rect.  has width, height, corner'''
11     def __init__(self, width, height, x, y):
12         self.width = width
13         self.height = height
14         self.corner = Point()
15         self.corner.x = x
16         self.corner.y = y
17
18
19 def main():
20     my_rect = Rectangle(2, 2, 1, 2)
21     print(my_rect)
22     print("this is dir(my_rect)", dir(my_rect))
23     print("my_rect.width = {}, my_rect.height = {}".format(
24         my_rect.width, my_rect.height)
25     )
26
27
28 if __name__ == '__main__':
29     main()
```

1. take control of object initialization

But, what's all this self. stuff?

The second self. denotes that each variable is an attribute of the Rectangle() class

example2.py — /Users/bcf/Dropbox (faircloth-lab)/Classes/BIOL7800/temp

example2.py

```
1  #!/usr/bin/env python
2
3
4  class Point():
5      '''A class to hold point data'''
6      # nothing here yet
7
8
9  class Rectangle():
10     '''Reps a rect.  has width, height, corner'''
11     def __init__(self, width, height, x, y):
12         self.width = width
13         self.height = height
14         self.corner = Point()
15         self.corner.x = x
16         self.corner.y = y
17
18
19 def main():
20     my_rect = Rectangle(2, 2, 1, 2)
21     print(my_rect)
22     print("this is dir(my_rect)", dir(my_rect))
23     print("my_rect.width = {}, my_rect.height = {}".format(
24         my_rect.width, my_rect.height)
25     )
26
27
28 if __name__ == '__main__':
29     main()
```

1. take control of object initialization

But, what's all this self. stuff?

The second self. denotes that each variable is an attribute of the Rectangle() class

We can also have normal variables within methods that are not attributes

example2.py — /Users/bcf/Dropbox (faircloth-lab)/Classes/BIOL7800/temp

example2.py

```
1  #!/usr/bin/env python
2
3
4  class Point():
5      '''A class to hold point data'''
6      # nothing here yet
7
8
9  class Rectangle():
10     '''Reps a rect.  has width, height, corner'''
11     def __init__(self, width, height, x, y):
12         self.width = width
13         self.height = height
14         self.corner = Point()
15         self.corner.x = x
16         self.corner.y = y
17         random_variable = "my dog is stupid"
18
19
20 def main():
21     my_rect = Rectangle(2, 2, 1, 2)
22     print(my_rect)
23     print("this is dir(my_rect)", dir(my_rect))
24     print("my_rect.width = {}, my_rect.height = {}".format(
25         my_rect.width, my_rect.height)
26     )
27
28
29 if __name__ == '__main__':
```

1. take control of object initialization

We can clean this up by adding `__init__` method to `Point()` class

example2.py — /Users/bcf/Dropbox (faircloth-lab)/Classes/BIOL7800/temp

example2.py

```
#!/usr/bin/env python


class Point():
    '''A class to hold point data'''
    def __init__(self, x, y):
        self.x = x
        self.y = y


class Rectangle():
    '''Reps a rect.  has width, height, corner'''
    def __init__(self, width, height, x, y):
        self.width = width
        self.height = height
        self.corner = Point(x, y)


def main():
    my_rect = Rectangle(2, 2, 1, 2)
    print(my_rect)
    print("this is dir(my_rect)", dir(my_rect))
    print("my_rect.width = {}, my_rect.height = {}".format(
        my_rect.width, my_rect.height)
    )


if __name__ == '__main__':
    main()

```

This seems inefficient…

Let's return to our center finding exercise…

We've cleaned up inefficient attribute stuff

example2.py — /Users/bcf/Dropbox (faircloth-lab)/Classes/BIOL7800/temp

example2.py

```
#!/usr/bin/env python


class Point():
    '''A class to hold point data'''
    def __init__(self, x, y):
        self.x = x
        self.y = y


class Rectangle():
    '''Reps a rect.  has width, height, corner'''
    def __init__(self, width, height, x, y):
        self.width = width
        self.height = height
        self.corner = Point(x, y)


def find_center(rect):
    x = rect.corner.x + rect.width / 2
    y = rect.corner.y + rect.height / 2
    return Point(x, y)


def main():
    my_rect = Rectangle(2, 2, 1, 2)
    result = find_center(my_rect)
    print(result)
    print(result.x, result.y)


if __name__ == '__main__':
```

This seems inefficient…

Let's return to our center finding exercise…

But we still have a function here that really applies **only** to `Rectangle()` objects

example2.py — /Users/bcf/Dropbox (faircloth-lab)/Classes/BIOL7800/temp

example2.py

```
1   #!/usr/bin/env python
2
3
4   class Point():
5       '''A class to hold point data'''
6       def __init__(self, x, y):
7           self.x = x
8           self.y = y
9
10
11  class Rectangle():
12      '''Reps a rect.  has width, height, corner'''
13      def __init__(self, width, height, x, y):
14          self.width = width
15          self.height = height
16          self.corner = Point(x, y)
17
18
19  def find_center(rect):
20      x = rect.corner.x + rect.width / 2
21      y = rect.corner.y + rect.height / 2
22      return Point(x, y)
23
24
25  def main():
26      my_rect = Rectangle(2, 2, 1, 2)
27      result = find_center(my_rect)
28      print(result)
29      print(result.x, result.y)
30
31
32  if __name__ == '__main__':
```

2. take control of object methods

We can make this function a **method** of all **Rectangle()** objects

example2.py — /Users/bcf/Dropbox (faircloth-lab)/Classes/BIOL7800/temp

example2.py

```
#!/usr/bin/env python


class Point():
    '''A class to hold point data'''
    def __init__(self, x, y):
        self.x = x
        self.y = y


class Rectangle():
    '''Reps a rect.  has width, height, corner'''
    def __init__(self, width, height, x, y):
        self.width = width
        self.height = height
        self.corner = Point(x, y)

    def find_center(self):
        x = self.corner.x + self.width / 2
        y = self.corner.y + self.height / 2
        return Point(x, y)


def main():
    my_rect = Rectangle(2, 2, 1, 2)
    result = my_rect.find_center()
    print(result)
    print(result.x, result.y)


if __name__ == '__main__':
    main()
```

2. take control of object methods

And we can call that method whenever we want

(after we instantiate the object)

example2.py — /Users/bcf/Dropbox (faircloth-lab)/Classes/BIOL7800/temp

example2.py

```
1   #!/usr/bin/env python
2
3
4   class Point():
5       '''A class to hold point data'''
6       def __init__(self, x, y):
7           self.x = x
8           self.y = y
9
10
11  class Rectangle():
12      '''Reps a rect.  has width, height, corner'''
13      def __init__(self, width, height, x, y):
14          self.width = width
15          self.height = height
16          self.corner = Point(x, y)
17
18      def find_center(self):
19          x = self.corner.x + self.width / 2
20          y = self.corner.y + self.height / 2
21          return Point(x, y)
22
23
24  def main():
25      my_rect = Rectangle(2, 2, 1, 2)
26      result = my_rect.find_center()
27      print(result)
28      print(result.x, result.y)
29
30
31  if __name__ == '__main__':
32      main()
```

2. take control of object methods

```
$ python example2.py
<__main__.Point object at 0x101a19f98>
2.0 3.0
```

example2.py — /Users/bcf/Dropbox (faircloth-lab)/Classes/BIOL7800/te

example2.py

```
1   #!/usr/bin/env python
2
3
4   class Point():
5       '''A class to hold point data'''
6       def __init__(self, x, y):
7           self.x = x
8           self.y = y
9
10
11  class Rectangle():
12      '''Reps a rect.  has width, height, corner'''
13      def __init__(self, width, height, x, y):
14          self.width = width
15          self.height = height
16          self.corner = Point(x, y)
17
18      def find_center(self):
19          x = self.corner.x + self.width / 2
20          y = self.corner.y + self.height / 2
21          return Point(x, y)
22
23
24  def main():
25      my_rect = Rectangle(2, 2, 1, 2)
26      result = my_rect.find_center()
27      print(result)
28      print(result.x, result.y)
29
30
31  if __name__ == '__main__':
32      main()
```

Now, let's look at argparse