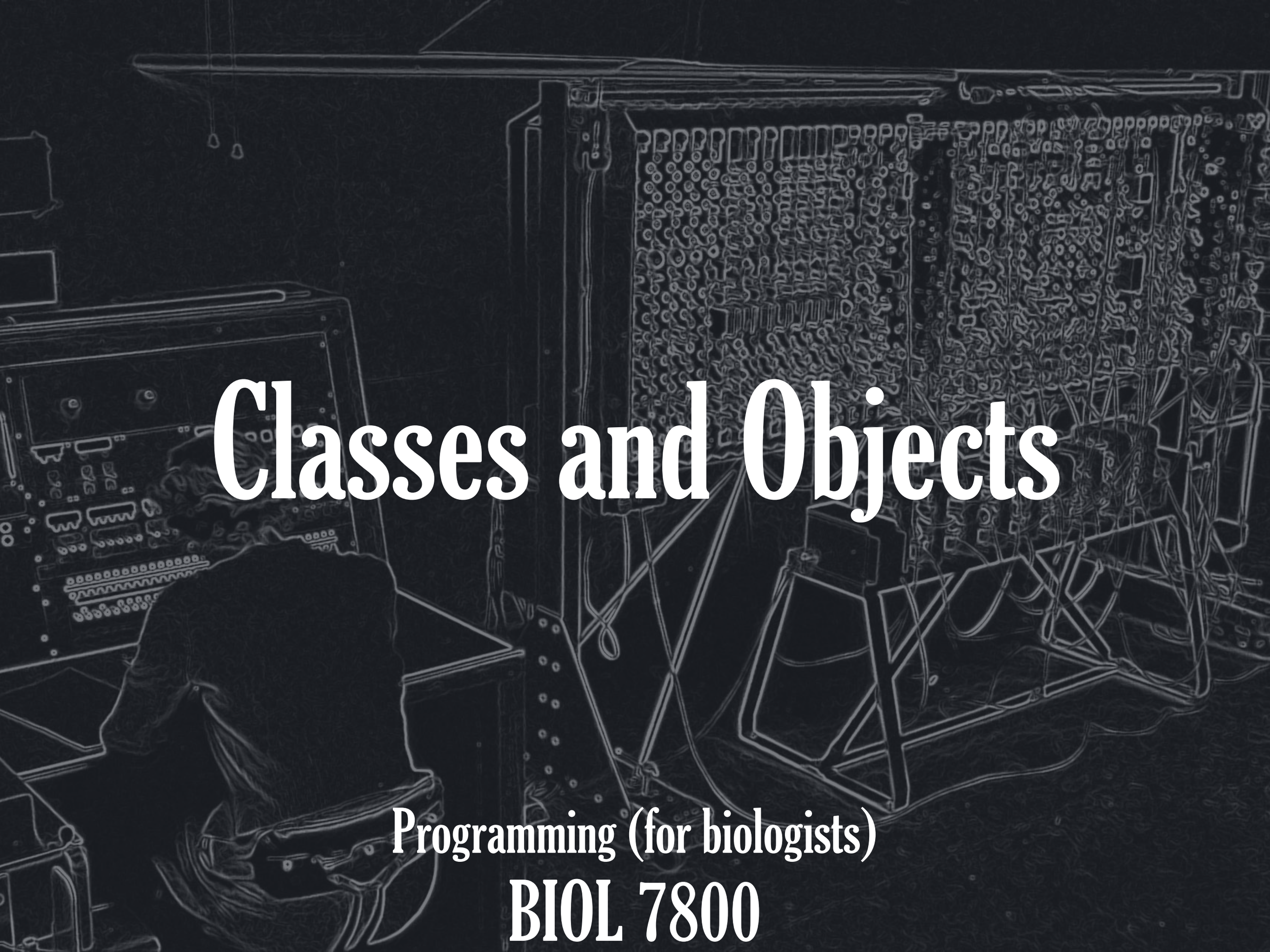
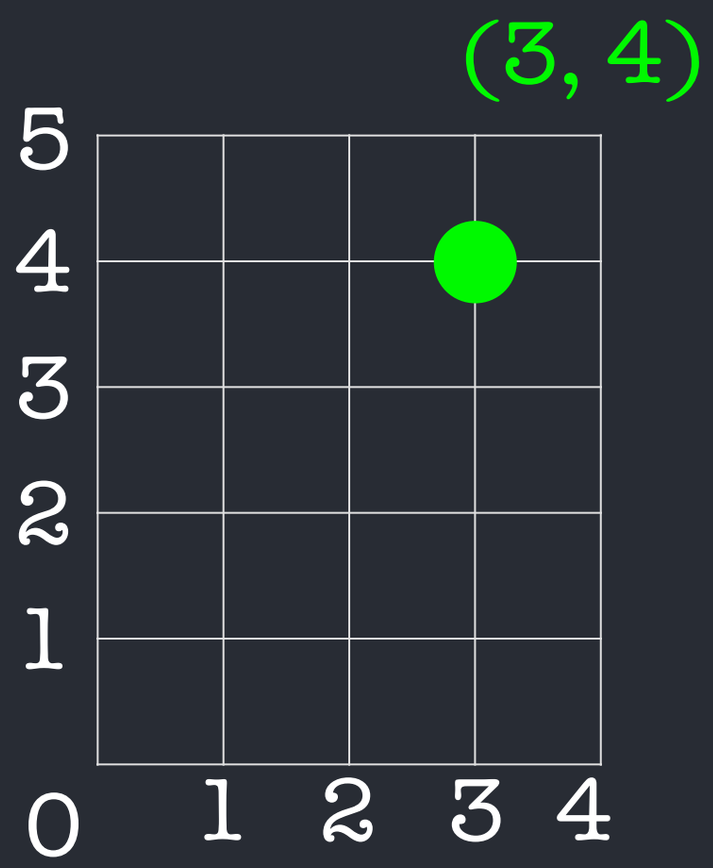


Classes and Objects

Programming (for biologists)

BIOL 7800

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

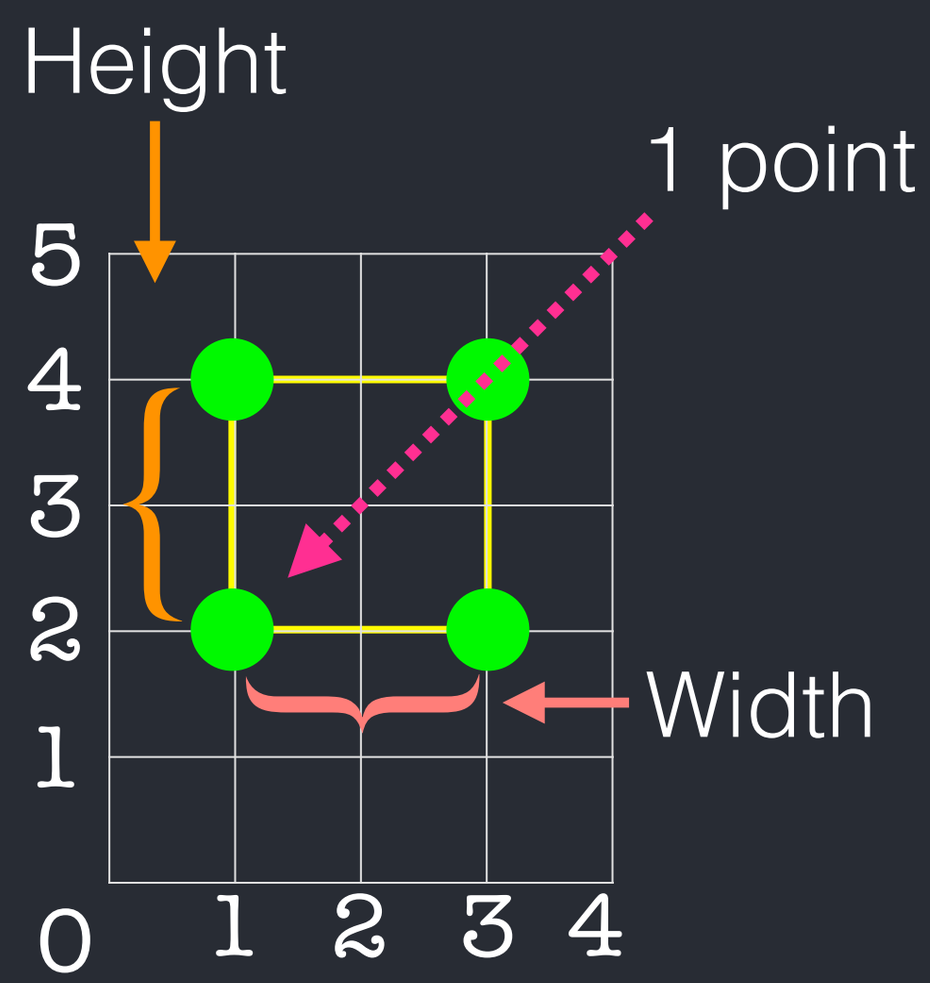
…and **attributes** are *values* assoc. with *named elements of an object of the* Point() *class*

Rectangle Class

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

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

Approach #1



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

```python
#!/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

```python
#!/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

```python
#!/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__':
```

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

```python
#!/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

```python
#!/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()
```

Another example: Time()

example12.py — /Users/bcf/Dropbox (faircloth-lab)/Classes/BIOL7800/temp

example12.py

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


class Time():
    '''A class for time-related data'''
    def __init__(self, hour=0, minute=0, second=0):
        self.hour = hour
        self.minute = minute
        self.second = second


def main():
    start = Time(9, 45)
    print(start)


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

File 0 Project 0 ✔ No Issues example12.py 14:17 LF UTF-8 Python 1 update

As far as you know*, there is no **class** for representing Time in Python

So let's make one…

*(datetime is the standard python module for classes dealing w/ times & dates)

Printing Time()

We can create a print_time function to pretty print the time of each Time() instance

example12.py — /Users/bcf/Dropbox (faircloth-lab)/Classes/BIOL7800/temp

example12.py

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


class Time():
    '''A class for time-related data'''
    def __init__(self, hour=0, minute=0, second=0):
        self.hour = hour
        self.minute = minute
        self.second = second


def print_time(time):
    print("{:0=2d}:{:0=2d}:{:0=2d}".format(
        time.hour,
        time.minute,
        time.second)
    )


def main():
    start = Time(9, 45)
    print_time(start)


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

```
$ python example12.py
09:45:00
```

Printing Time()

We can create a print_time function to pretty print the time of each instance

What is a **better way** to implement this function?

```
$ python example12.py
09:45:00
```

example12.py — /Users/bcf/Dropbox (faircloth-lab)/Classes/BIOL7800/temp

example12.py

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


class Time():
    '''A class for time-related data'''
    def __init__(self, hour=0, minute=0, second=0):
        self.hour = hour
        self.minute = minute
        self.second = second


def print_time(time):
    print("{:0=2d}:{:0=2d}:{:0=2d}".format(
        time.hour,
        time.minute,
        time.second)
    )


def main():
    start = Time(9, 45)
    print_time(start)


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

Printing Time()

What is a **better way** to implement this function?

As a method of the Time() class

Create Time() instance

Call pretty_print() method

example12.py — /Users/bcf/Dropbox (faircloth-lab)/Classes/BIOL7800/temp

example12.py

```
1   class Time():
2       '''A class for time-related data'''
3       def __init__(self, hour=0, minute=0, second=0):
4           self.hour = hour
5           self.minute = minute
6           self.second = second
7
8       def pretty_print(self):
9           print("{:0=2d}:{:0=2d}:{:0=2d}".format(
10              self.hour,
11              self.minute,
12              self.second)
13          )
14
15
16  def main():
17      start = Time(9, 45)
18      start.pretty_print()
19
20
21  if __name__ == '__main__':
```

Adding Time()

We want to add **two** time objects…

add_time() is a method of the Time() class

example12.py — /Users/bcf/Dropbox (faircloth-lab)/Classes/BIOL7800/temp

example12.py

```
5            self.minute = minute
6            self.second = second
7
8        def pretty_print(self):
9            print("{:0=2d}:{:0=2d}:{:0=2d}".format(self.hour, sel
10
11       def add_time(self, t2):
12           hour = self.hour + t2.hour
13           minute = self.minute + t2.minute
14           second = self.second + t2.second
15           if second >= 60:
16               second -= 60
17               minute += 1
18           if minute >= 60:
19               minute -= 60
20               hour += 1
21           return Time(hour, minute, second)
22
23
24   def main():
25       start = Time(9, 45)
26       start.pretty_print()
27       another_time = Time(1, 10)
28       new_time = start.add_time(another_time)
29       start.pretty_print()
30       new_time.pretty_print()
31
32
33   if __name__ == '__main__':
```

```
$ python example12.py
09:45:00

09:45:00
10:55:00
```

Adding Time()

We want to add

add_time()
of the
Notice that this method ***does not change***
the start instance of Time()

You might call this a **"pure"** method
(it does not modify object attributes)

example12.py — /Users/bcf/Dropbox (faircloth-lab)/Classes/BIOL7800/temp

example12.py

```
5            self.minute = minute
6            self.second = second
7
8        def pretty_print(self):
9            print("{:0=2d}:{:0=2d}:{:0=2d}".format(self.hour, self
10
11       def add_time(self, t2):
12           hour = self.hour + t2.hour
13           minute = self.minute + t2.minute
14           second = self.second + t2.second
15           if second >= 60:
16               second -= 60
17               minute += 1
18           if minute >= 60:
19               minute -= 60
20               hour += 1
21           return Time(hour, minute, second)
22
23   |
24   def main():
25       start = Time(9, 45)
26       start.pretty_print()
27       another_time = Time(1, 10)
28       new_time = start.add_time(another_time)
29       start.pretty_print()
30       new_time.pretty_print()
31
32
33   if __name__ == '__main__':
```

```
$ python example12.py
09:45:00

09:45:00
10:55:00
```

Adding Time()

We want to add **some amount of** time to an object…

increment_time() is a method of the Time() class

example12.py — /Users/bcf/Dropbox (faircloth-lab)/Classes/BIOL7800/temp

example12.py

```
class Time():
    '''A class for time-related data'''
    def __init__(self, hour=0, minute=0, second=0):
        self.hour = hour
        self.minute = minute
        self.second = second

    def pretty_print(self):

    def add_time(self, t2):

    def increment_time(self, seconds):
        self.second += seconds
        if self.second >= 60:
            self.second -= 60
            self.minute += 1
        if self.minute >= 60:
            self.minute -= 60
            self.hour += 1

def main():
    start = Time(9, 45)
    start.pretty_print()
    start.increment_time(60)
    start.pretty_print()


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

```
$ python example12.py
09:45:00

09:46:00
```

Adding Time()

We want to add **some amount of** time to an object…

example12.py — /Users/bcf/Dropbox (faircloth-lab)/Classes/BIOL7800/temp

example12.py

```
class Time():
    '''A class for time-related data'''
    def __init__(self, hour=0, minute=0, second=0):
        self.hour = hour
        self.minute = minute
        self.second = second

    def pretty_print(self):

    def increment_time(self, seconds):
        self.second += seconds
        if self.second >= 60:
            self.second -= 60
            self.minute += 1
        if self.minute >= 60:
            self.minute -= 60
            self.hour += 1


def main():
    start = Time(9, 45)
    start.pretty_print()
    start.increment_time(60)
    start.pretty_print()


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

`increment_time()` method of the class

Notice that this method **changes** the `start` instance of `Time()`

You might call this a **"modifier"** method

(it **does** modify object attributes)

```
$ python example12.py
09:45:00

09:46:00
```

Adding Time()

What's a **remaining problem** with both of these methods?

How might we fix them?

add_time()

increment_time()

example12.py — /Users/bcf/Dropbox (faircloth-lab)/Classes/BIOL7800/temp

example12.py

```python
class Time():
    '''A class for time-related data'''
    def __init__(self, hour=0, minute=0, second=0):
        self.hour = hour
        self.minute = minute
        self.second = second

    def pretty_print(self):

    def add_time(self, t2):
        hour = self.hour + t2.hour
        minute = self.minute + t2.minute
        second = self.second + t2.second
        if second >= 60:
            second -= 60
            minute += 1
        if minute >= 60:
            minute -= 60
            hour += 1
        return Time(hour, minute, second)

    def increment_time(self, seconds):
        self.second += seconds
        if self.second >= 60:
            self.second -= 60
            self.minute += 1
```

Adding Time()

example12.py — /Users/bcf/Dropbox (faircloth-lab)/Classes/BIOL7800/temp

example12.py

```python
class Time():
    '''A class for time-related data'''
    def __init__(self, hour=0, minute=0, second=0):
        self.hour = hour
        self.minute = minute
        self.second = second

    def pretty_print(self):

    def time_to_int(self):
        minutes = self.hour * 60 + self.minute
        seconds = minutes * 60 + self.second
        return seconds

    def int_to_time(self, seconds):
        minutes, second = divmod(seconds, 60)
        hour, minute = divmod(minutes, 60)
        return Time(hour, minute, second)

    def increment_time(self, seconds):
        new_time = self.time_to_int() + seconds
        temp = self.int_to_time(new_time)
        self.hour = temp.hour
        self.minute = temp.minute
        self.second = temp.second


def main():
    start = Time(9, 45, 1)
    start.pretty_print()
    start.increment_time(90)
    start.pretty_print()

if __name__ == '__main__':
```

Method to convert time to seconds

Method to convert seconds to time

Method to add seconds to time

```
$ python example12.py
09:45:01
09:46:31
```

Checking Time()

Method to check our time

We can check_time after any method that alters it

example12.py — /Users/bcf/Dropbox (faircloth-lab)/Classes/BIOL7800/temp

example12.py

```
1   class Time():
2       '''A class for time-related data'''
3       def __init__(self, hour=0, minute=0, second=0):
4           self.hour = hour
5           self.minute = minute
6           self.second = second
7
8       def check_time(self):
9           if self.hour < 0 or self.minute < 0 or self.second < 0:
10              return False
11          if self.minute >= 60 or self.second >= 60:
12              return False
13          return True
14
15      def pretty_print(self):
21
22      def time_to_int(self):
26
27      def int_to_time(self, seconds):
31
32      def increment_time(self, seconds):
33          new_time = self.time_to_int() + seconds
34          temp = self.int_to_time(new_time)
35          self.hour = temp.hour
36          self.minute = temp.minute
37          self.second = temp.second
38          self.check_time()
39
40
41  def main():
42      start = Time(9, 45, 1)
43      start.pretty_print()
44      start.increment_time(90)
45      start.pretty_print()
46
```

```
$ python example12.py
09:45:01
09:46:31
```

__str__ method of Time()

We have to call pretty_print to see anything about objects of this class

example12.py — /Users/bcf/Dropbox (faircloth-lab)/Classes/BIOL7800/temp

example12.py

```
1   class Time():
2       '''A class for time-related data'''
3       def __init__(self, hour=0, minute=0, second=0):
4           self.hour = hour
5           self.minute = minute
6           self.second = second
7
8       def check_time(self):
14
15      def pretty_print(self):
16          print("{:0=2d}:{:0=2d}:{:0=2d}".format(
17              self.hour,
18              self.minute,
19              self.second)
20          )
21
22      def time_to_int(self):
26
27      def int_to_time(self, seconds):
31
32      def increment_time(self, seconds):
39
40
41  def main():
42      start = Time(9, 45, 1)
43      start.pretty_print()
44      start.increment_time(90)
45      start.pretty_print()
46
47  if __name__ == '__main__':
48      main()
49
```

$ python example12.py

09:45:01

09:46:31

__str__ method of Time()

We have to call
see anything about objects of
this class

example12.py — /Users/bcf/Dropbox (faircloth-lab)/Classes/BIOL7800/temp

example12.py

```
1   class Time():
2       '''A class for time-related data'''
3       def __init__(self, hour=0, minute=0, second=0):
4           self.hour = hour
5           self.minute = minute
6           self.second = second
7
8 ›     def check_time(self):
14
15      def pretty_print(self):
16          print("{:0=2d}:{:0=2d}:{:0=2d}".format(
17              self.hour,
18              self.minute,
19              self.second)
22 ›    def time_to_int(self):
26
27 ›    def int_to_time(self, seconds):
31
32 ›    def increment_time(self, seconds):
39
40
41  def main():
42      start = Time(9, 45, 1)
43      start.pretty_print()
44      start.increment_time(90)
45      start.pretty_print()
46
47  if __name__ == '__main__':
48      main()
49
```

Wouldn't it be easier to get this information by default?

```
$ python example12.py
09:4
09:4
```

__str__ method of Time()

We convert pretty_print to the special __str__ method, and change print() to return.

The __str__ method returns an objects "string representation"

Now, every time we simply print() an instance of the Time() class, we see its "string representation"

```
$ python example12.py
09:45:01
09:46:31
```

example12.py — /Users/bcf/Dropbox (faircloth-lab)/Classes/BIOL7800/temp

example12.py

```python
class Time():
    '''A class for time-related data'''
    def __init__(self, hour=0, minute=0, second=0):
        self.hour = hour
        self.minute = minute
        self.second = second

    def __str__(self):
        return "{:0=2d}:{:0=2d}:{:0=2d}".format(
            self.hour,
            self.minute,
            self.second
        )

    def check_time(self):…

    def time_to_int(self):…

    def int_to_time(self, seconds):…

    def increment_time(self, seconds):…


def main():
    start = Time(9, 45, 1)
    print(start)
    start.increment_time(90)
    print(start)

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

__str__ method of Time()

We can make the special __str__ method say whatever we want

$ python example12.py
The time is: 09:45:01

example12.py — /Users/bcf/Dropbox (faircloth-lab)/Classes/BIOL7800/temp

example12.py

```
1   class Time():
2       '''A class for time-related data'''
3       def __init__(self, hour=0, minute=0, second=0):
4           self.hour = hour
5           self.minute = minute
6           self.second = second
7
8       def __str__(self):
9           return "The time is: {:0=2d}:{:0=2d}:{:0=2d}".format(
10              self.hour,
11              self.minute,
12              self.second
13          )
14
15 ›    def check_time(self):⋯
21
22 ›    def time_to_int(self):⋯
26
27 ›    def int_to_time(self, seconds):⋯
31
32 ›    def increment_time(self, seconds):⋯
39
40
41  def main():
42      start = Time(9, 45, 1)
43      print(start)
44
45  if __name__ == '__main__':
46      main()
47
```

__add__ method of Time()

Here, we define __add__ special method, which let's us use the + operator with our class

example12.py — /Users/bcf/Dropbox (faircloth-lab)/Classes/BIOL7800/temp

example12.py

```
1   class Time():
2       '''A class for time-related data'''
3       def __init__(self, hour=0, minute=0, second=0):
4           self.hour = hour
5           self.minute = minute
6           self.second = second
7
8 ›     def __str__(self):⋯
14
15      def __add__(self, other):
16          seconds = self.time_to_int() + other.time_to_int()
17          return self.int_to_time(seconds)
18
19 ›    def check_time(self):⋯
25
26 ›    def time_to_int(self):⋯
30
31 ›    def int_to_time(self, seconds):⋯
35
36 ›    def increment_time(self, seconds):⋯
43
44
45  def main():
46      start = Time(9, 45)
47      print("Start time is {}\n".format(start))
48      duration = Time(1, 45)
49      print("End time is {}".format(start + duration))
50
51  if __name__ == '__main__':
52      main()
53
```

instance 1

instance 2

instance 1 + instance 2

$ python example12.py

Start time is 09:45:00

End time is 11:30:00

__add__ method of Time()

Here, we define __add__ special method, which let's us use the + operator with our class

Changing the behavior of + to work with our new Time() type is called operator overloading.

For *every* operator in Python, there is a corresponding special method.

example12.py — /Users/bcf/Dropbox (faircloth-lab)/Classes/BIOL7800/temp

example12.py

```
1   class Time():
2       '''A class for time-related data'''
3       def __init__(self, hour=0, minute=0, second=0):
4           self.hour = hour
5           self.minute = minute
6           self.second = second
7
8 ›     def __str__(self):⋯
14
15      def __add__(self, other):
16          seconds = self.time_to_int() + other.time_to_int()
17          return self.int_to_time(seconds)
18
19 ›    def check_time(self):⋯
25
26 ›    def time_to_int(self):⋯
30
31 ›    def int_to_time(self, seconds):⋯
35
36 ›    def increment_time(self, seconds):⋯
43
44
45  def main():
46      start = Time(9, 45)
47      print("Start time is {}\n".format(start))
48      duration = Time(1, 45)
49      print("End time is {}".format(start + duration))
50
51  if __name__ == '__main__':
52      main()
53
```

instance 1

instance 2

instance 1 + instance 2

Type-based Dispatch

Here, we use isinstance() function to see if we are adding a Time() object or some other object, then we return one answer or another based on that result.

We we can use + to add (1) another Time() object or (2) a bunch of seconds

Because computation is based on argument type, this is an example of type-based dispatch to diff methods

example12.py — /Users/bcf/Dropbox (faircloth-lab)/Classes/BIOL7800/temp

example12.py

```
1   class Time():
2       '''A class for time-related data'''
3       def __init__(self, hour=0, minute=0, second=0):
4           self.hour = hour
5           self.minute = minute
6           self.second = second
7
8 ›     def __str__(self):⋯
14
15      def __add__(self, other):
16          if isinstance(other, Time):
17              return self.add_time(other)
18          else:
19              return self.increment(other)
20
21      def add_time(self, other):
22          seconds = self.time_to_int() + other.time_to_int()
23          return self.int_to_time(seconds)
24
25      def increment(self, seconds):
26          seconds += self.time_to_int()
27          return self.int_to_time(seconds)
28
29 ›     def check_time(self):⋯
35
36 ›     def time_to_int(self):⋯
40
41 ›     def int_to_time(self, seconds):⋯
45
46
47  def main():
48      start = Time(9, 45)
49      duration = Time(1, 45)
50      print("End time is {}".format(start + duration))
51      print("---")
52      print("Incremented time is {}".format(start + 1000))
53
```

Inheritance

Our original Time() class

example12.py — /Users/bcf/Dropbox (faircloth-lab)/Classes/BIOL7800/temp

example12.py

```python
class Time():
    '''A class for time-related data'''
    def __init__(self, hour=0, minute=0, second=0):
        self.hour = hour
        self.minute = minute
        self.second = second

    def __str__(self):...

    def __add__(self, other):
        if isinstance(other, Time):
            return self.add_time(other)
        else:
            return self.increment(other)

    def add_time(self, other):
        seconds = self.time_to_int() + other.time_to_int()
        return self.int_to_time(seconds)

    def increment(self, seconds):
        seconds += self.time_to_int()
        return self.int_to_time(seconds)

    def check_time(self):...

    def time_to_int(self):...

    def int_to_time(self, seconds):...
```

Our new MyTime() class

example13.py — /Users/bcf/Dropbox (faircloth-lab)/Classes/BIOL7800/temp

example13.py

```python
from example12 import Time


class MyTime(Time):
    pass


def main():
    time = MyTime(9, 45, 15)
    print(dir(time))
    print("---")
    print(time)

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

Where did all this come from?

```
$ python example13.py
['__add__', '__class__', ..., 'add_time',
'check_time', 'hour', 'increment',
'int_to_time', 'minute', 'second',
'time_to_int']
---
09:45:15
```

Inheritance

Where did all this come from?

Our new MyTime() class

example13.py — /Users/bcf/Dropbox (faircloth-lab)/Classes/BIOL7800/temp

example13.py

```
1   from example12 import Time
2
3
4   class MyTime(Time):
5       pass
6
7
8   def main():
9       time = MyTime(9, 45, 15)
10      print(dir(time))
11      print("---")
12      print(time)
13
14  if __name__ == '__main__':
15      main()
16
```

Imported our old Time() class

Subclassed our old Time() class, as part of our new MyTime() class

So we get **all** the attributes and methods of Time()

Inheritance

Our new MyTime() class

Let's say we don't like the way that Time() displayed the objects string representation (__str__)

We can "override" that method by defining a new __str__ method for our MyTime() object

example13.py — /Users/bcf/Dropbox (faircloth-lab)/Classes/BIOL7800/temp

example13.py

```
from example12 import Time


class MyTime(Time):
    def __str__(self):
        s = "The hour is {:0=2d}\n" + \
            "The minutes are {:0=2d}\n" + \
            "The seconds are {:0=2d}"
        return s.format(
            self.hour,
            self.minute,
            self.second)


def main():
    time = MyTime(9, 45, 15)
    print(time)

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

```
$ python example13.py
The hour is 09
The minutes are 45
The seconds are 15
```

Inheritance

Our new MyTime() class

Or, we can define entirely new functions for our MyTime() class.

example13.py — /Users/bcf/Dropbox (faircloth-lab)/Classes/BIOL7800/temp

example13.py

```python
from example12 import Time


class MyTime(Time):
    def print_time_backwards(self):
        print("{:0=2d}:{:0=2d}:{:0=2d}".format(
            self.second,
            self.minute,
            self.hour
        ))


def main():
    time = MyTime(9, 45, 15)
    print(dir(time))
    print("---")
    time.print_time_backwards()
    print("---")
    print(time)

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

```
$ python example13.py
['__add__', '__class__', ..., 'add_time', 'check_time', 'hour', 'increment', 'int_to_time', 'minute',
'print_time_backwards', 'second', 'time_to_int']
---
15:45:09
---
09:45:15
```

Homework #13...