Classes and Methods

Classes have methods

- A *method* is a function that is associated with a particular *class.*
- Methods are *defined* inside the class definition.
- *Method calls* use dot notation.

```
print math.sqrt(25)
```


The Time Class

- Consider the Time class.
- To print a Time object we can write a function.

```
class Time(object):
    """represents the time of day.
       attributes: hour, minute, second"""

def print_time(time):
    print '%.2d:%.2d:%.2d' % (time.hour,
time.minute, time.second)
```


Calling the print functions

- We pass a Time object to the print function.

```
>>> start = Time()
>>> start.hour = 9
>>> start.minute = 45
>>> start.second = 00
>>> print_time(start)
09:45:00
```

Calling the Time.print Method

• We can call the print method as we do a function, invoking it on the Time class and passing it a Time object:

```
>>> Time.print_time(start)
09:45:00
```


The Time.print Method

• It makes more sense to define *print* as a method because it only applies to Time objects.

```
class Time(object):
  def print_time(time):
    print '%.2d:%.2d:%.2d' % (time.hour,
time.minute, time.second)
```


Self Reference

• By convention, the first parameter of a method is called *self*, so it would be more common to write print_time like this:

```
class Time(object):
  def print_time(self):
    print '%.2d:%.2d:%.2d' % (self.hour,
self.minute, self.second)
```


Calling the Time.print Method

• We can call the print method as we do a function, invoking it on the Time class and passing it a Time object:

```
>>> Time.print_time(start)
09:45:00
```

• Or we can use the *self convention,* invoking it on a Time object (its *subject*):

```
>>> start.print_time()
09:45:00
```

Example

• Define a *time_to_int()* method that converts a Time object into an int representing the total number of seconds.

Self Reference

• By convention, the first parameter of a method is called *self*, so it would be more common to write print_time like this:

```
class Time(object):
  def print_time(self):
    print '%.2d:%.2d:%.2d' % (self.hour,
self.minute, self.second)
```

•This is like saying: "Hey start, print yourself!"

```
>>> start.print_time()
09:45:00
```


Example

• Define a *time_to_int()* method that converts a Time object into an int representing the total number of seconds.

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


Example

• Define a *time_to_int()* method that converts a Time object into an int representing the total number of seconds.

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

```
>>> start.print_time()
09:45:00
>>> print start.time_to_int()
3500
```

Another Example

• Define a *int_to_time()* **function** that converts a int representing the total number of seconds into a Time object and returns the object.

```
def int_to_time(seconds):
    time = Time()
    minutes = seconds / 60
    time.second = seconds % 60
    time.hour = minutes / 60
    time.minute = minutes % 60
    return time
```


Another Example

• Write a method that will increment the time by a given number of seconds.

Another Example

• Define a *int_to_time()* **function** that converts a int representing the total number of seconds into a Time object and returns the object.

Another Example

• Define a *int_to_time()* **function** that converts a int representing the total number of seconds into a Time object and returns the object.

```
>>> start.print_time()
09:45:00

>>> end = int_to_time(start.time_to_int())
>>> end.print_time()
09:45:00
```

Another Example

• Write a method that will increment the time by a given number of seconds.

```
# inside class Time:

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


Calling the Increment Method

• When we call the increment() method on the start object, start gets assigned to the ***self*** parameter.

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

```
>>> start.print_time()
09:45:00
>>> end = start.increment(1337)
>>> end.print_time()
10:07:17
```


Calling the Increment Method

• Note that the increment method does not change its object.

• It is a *pure function.*

```
>>> start.print_time()
09:45:00
>>> end = start.increment(1337)
>>> end.print_time()
10:07:17
>>> start.print_time()
09:45:00
```


Syntax Error

• But look what happens when you don't pass an object as the first parameter of a method.

```
>>> end = start.increment(1337, 460)
TypeError: increment() takes exactly 2 arguments
(3 given)
```

An Example with Two Objects

• Write an *is_after()* method that returns True iff one object (self) is later than another object.

```
class Time(object):
    def is_after(self, other):
        return self.time_to_int() > other.time_to_int()

>>> end.is_after(start)
True
```


An Example with Two Objects

• Write an *is_after()* method that returns True iff one object (self) is later than another object.

```
class Time(object):
    def is_after(self, other):
        return self.time_to_int() > other.time_to_int()
```


The special __init__ method

• The __init__ method is a special *initialization* method that is called when an object is *instantiated.*

```
# inside class Time:
    def __init__(self, hour=0, minute=0, second=0):
        self.hour = hour
        self.minute = minute
        self.second = second
```


Optional Parameters

• The __init__ parameters are optional. If you call it with no arguments, you get the default values for each attribute.

```
>>> time = Time()
>>> time.print_time()
00:00:00
```

• If you call it with one argument it applies to the hour attribute, etc.

```
>>> time = Time(9)
>>> time.print_time()
09:00:00
```

The special __str__ method

- The special method __str__ method converts the object to a string.

```
# inside class Time:
    def __str__(self):
        return '%.2d:%.2d:%.2d' % (self.hour,
self.minute, self.second)
```

- When you print an object, Python automatically invokes its __str__ method.

```
>>> time = Time(9, 45)
>>> print time
09:45:00
```


Operator Overloading

- By defining other special methods, you can overload operators, such as __add__ for +.

```
# inside class Time:
    def __add__(self, other):
        seconds = self.time_to_int() + other.time_to_int()
        return int_to_time(seconds)
```

- Sample usage:

```
>>> start = Time(9, 45)
>>> duration = Time(1, 35)
>>> print start + duration
11:20:00
```


The special __str__ method

- The special method __str__ method converts the object to a string.

```
# inside class Time:
    def __str__(self):
        return '%.2d:%.2d:%.2d' % (self.hour,
self.minute, self.second)
```


Operator Overloading

- By defining other special methods, you can overload operators, such as __add__ for +.

```
# inside class Time:
    def __add__(self, other):
        seconds = self.time_to_int() + other.time_to_int()
        return int_to_time(seconds)
```

Inclass Exercises

- Write an __init__ method for the Point class.
- Write a __str__ method for the Point class.
- Write a __add__ method for the Point class.