CS61A Notes – Week 7: OOP Continued

This week we will be continuing our discussion of Object Oriented Programming. Last week we learned that OOP consists of data abstraction in the form of Classes, or blueprints for creating types of objects, and Objects, an instance of a certain class created according to its blueprint.

We learned how to define classes along with class variables, instance variables and methods, how to create objects and how to inherit from a parent class.

We also saw the use of self in most of our class methods. Let's take a close look at what self is actually referring to.

1. Self and String Representations

Define the __init__ method of a class called Time

```
class Time(object):
    """ Represents the time of day. Has a constructor which
    takes the hour, minute, and second and stores them as
    attributes. The attributes default to 0 if not provided
    to init """
    # Your Code Here
```

Now define an independent function in the global scope called `print_time` that takes a Time object and prints out the hour minute and second on one line.

```
def print_time(time):
    """ time - an instance of the class Time
    print out the hour minute and second of time on one
    line - hh:mm:ss """
    # Your Code Here
```

Once we have defined our Time and class and `print_time` function, what would Python print?

```
>>> start = Time(12,15,50)
>>> start.hour
```

```
>>> print_time(start)
```

```
>>> end = Time(1,30)
>>> print_time(end)
```

```
>>> midnight = Time()
>>> print_time(midnight)
```

Notice, to call `print_time` we have to pass it a Time object as an argument. We can make this look more like OOP by making `print_time` a method which is done by moving it inside the class definition

```
class Time(object):
    def print_time(time):
        """ Your Code Here """
```

Now what does python print?

```
>>> start = Time(4,15,30)
>>> start.print_time(start)
```

```
>>> start.print_time()
```

Note that for our function `print_time(time)` we had to pass it an object to assigning it to the parameter time for it to print, now the we have turned `print_time` into a method, we get an error when we try to pass it an object to assign time but we get the right output when we don't pass it anything even though the definition calls for one argument. What is going on here?

When we call `start.print_time()` we are invoking the `print_time` method on start - an instance of the class Time. When a method is invoked on an object, that object is implicitly passed as the first argument to the method. That is, the object that is the value of the <expression> to the left of the dot is passed automatically as the first argument to the method named on the right side of the dot expression. As a result, the object is bound to the parameter `self`.

When defining methods in a class we use `self` as the first parameter to denote that this object this method is invoked on will implicitly assigns it**self** to the first parameter. So using `self`, what should our `print_time` method should look like now?

```
class Time(object):
    def print_time(self):
        """ Your Code Here """
```

Lastly what would python print here:

```
>>> t1 = Time(3,45,10)
>>> print(t1)
```

We shouldn't have to call print_time every time we want a Time object to print itself out in a readable manner. Luckily Python provides classes with the __str__ method which should return a string representation of the object. So change your `print_time(self)` method to `__str__(self)` making sure to return the string instead of printing it.

```
class Time(object):
    def __str__(self):
        """ Return a string representation of the object
```

And now try:

```
>>> t1 = Time(3,45,10)
>>> print(t1)
```

---

2. Operator Overloading

Now we are going to add two methods to your Time class. One called `time_to_int` that returns an integer representation of the time in total seconds from midnight, and a static method called `int_to_time` that takes an amount in seconds and returns a Time object with the corresponding time. `int_to_time` is given to you.

```
class Time(object):
      . . .
      def time_to_int(self):
            """ Returns the integer representation the time in
            total seconds """
            # Your Code Here
```

```
    @staticmethod
    def int_to_time(seconds):
        """ Takes an argument seconds, and returns a Time
        object representing the corresponding time """
        time = Time()
        minutes, time.second = divmod(seconds, 60)
        time.hour, time.minute = divmod(minutes, 60)
        return time
```

Why is `int_to_time` a static method?

_____

What does `divmod` do?

_____

So we have a time class that can print itself. What would Python print if we try:

```
>>> time1 = Time(1,45,10)
>>> time2 = Time(2,10,10)
>>> time1 + time2
```

_____

So far Python doesn't know how to add to two objects of type Time. We can use the __add__ method here to tell Python how to add two Time objects Hint: use `time_to_int` and `int_to_time`

```
class Time(object):
    """. . ."""
    def __add__(self,other):
        # Your Code Here
```

Now what would Python print?

```
>>> time1 = Time(1,45,10)
>>> time2 = Time(2,10,10)
>>> time1 + time2
```

_____

3. Type Based Dispatch

So now Python knows how to add two time objects together, but what if we wanted to simply increment a time object by a number of seconds using something like:

```
>>> time1 = Time(1,45,10)
>>> time1 + 65
01:46:15
```

We would have to be able to know if we are adding two Time objects or a Time object and an integer.

Hint: you can use `type(object)` to check the type of other.

Define methods `add_time(self, other)` and `increment(self, seconds)` and change your `__add__(self, other)` method to check the type of other and call the correct method to add it.

```
class Time(object):
    """. . ."""
    def __add__(self,other):
        """ Checks the type of other
        If other is a time object, call add_time
        If not call increment """
        # Your Code Here



    def add_time(self,other):
        """ Add self and other together and return the resulting
        Time object """
        # Your Code Here




    def increment(self,seconds):
        """ Increment self by seconds and return the resulting
        Time object """
        # Your Code Here
```

The last thing we need to do is make sure that we can add seconds to a time object from the right hand side of an add operation. It would look something like this:

```
>>> time1 = Time(1,45,10)
>>> 65 + time1
01:46:15
```

This is done by implementing the __radd__(self, other) method, which works similarly to __add__ except that other comes from the right hand side of the add operator.

```
class Time(object):
    """. . ."""
    def __radd__(self,other):
        """ Does the same thing as add pretty much but other
        refers to the value on the right of the + operator """
        # Your Code Here
```

4. Inheritance

Now that we've implemented a Time class that corresponds to a 24 hour clock, let's write another class to represent a twelve hour clock that displays time like this:

```
>>> start = Time12(14,15,50)
>>> start.hour
2
>>> start.period
PM
>>> print(start)
02:15:50 PM
```

Fortunately we don't have to rewrite all the methods for Time12, we can have it inherit common operations from the Time class. Think about which methods you would have to overwrite in order to give the Time12 class full functionality.

We should also make sure that Time12 objects are initialized with valid arguments for time. What is a valid range for hour, second and minutes in our new format? What exceptions would you raise for incorrect values?

```
class Time12(Time):
      """ Time12 is a Time object which in a 12 hour time format
      followed by the period (am/pm). The class inherits from
      time and should only add attributes and methods that are
      particular to our new format. It should also check to make
      sure that inputs are valid and raise an exception if not.
      """
      # Your Code Here
```