Lecture 7.3 : Object-oriented programming: Instance methods

Adding an instance method

- Let's add another method to our `Time` class.
- The new method is called `is_later_than()` and returns `True` if one time is later than another.
- How will we go about writing this method?
- Let's start with its signature i.e. how many parameters will the method need to specify?
- Since we are comparing two times it seems clear that the method will require two times be passed to it, `t1` and `t2`, both of which will be objects of the `Time` class.
- Next, what will our method return?
- It also seems sensible that our method should return a boolean, `True` if `t1` is later than `t2` and `False` otherwise.
- How will our method be invoked?
- Well, our method operates directly on an *instance* of the `Time` class so it will be an *instance method*.
- It compares one instance of the class with another instance of the same class.
- Thus we will invoke it like this: `t1.is_later_than(t2)` in order to ask "Is `t1` later than `t2`?"
- Here is our updated `Time` class.

```
class Time(object):

    def __init__(self, hour=0, minute=0, second=0):
        self.hour = hour
        self.minute = minute
        self.second = second

    def is_later_than(self, other):
        # Compare hours
        if self.hour > other.hour:
            return True
        if self.hour < other.hour:
            return False

        # Hours are equal so compare minutes
        if self.minute > other.minute:
            return True
        if self.minute < other.minute:
            return False

        # Hours and minutes are equal so compare seconds
        if self.second > other.second:
            return True

        return False

    def __str__(self):
        return 'The time is {:02d}:{:02d}:{:02d}'.format(
            self.hour, self.minute, self.second)

t1 = Time(13, 43, 6)
t2 = Time(14, 52, 7)
t3 = Time(14, 43, 7)
```

```
t4 = Time(13, 43, 7)

print(t2.is_later_than(t1))
print(t1.is_later_than(t2))
print(t3.is_later_than(t2))
print(t4.is_later_than(t1))
```

```
True
False
False
True
```

- When we call `t1.is_later_than(t2)` the `t1` argument becomes the `self` parameter in the method while the `t2` argument becomes the `other` parameter.
- (Remember `t1.is_later_than(t2)` is really just shorthand for `Time.is_later_than(t1, t2)` and in the latter it is obvious that `t1` becomes `self` and `t2` becomes `other` inside the method.)
- Study the method to ensure you understand how it works.
- It begins by comparing hours, then minutes and finally seconds.
- Note how it returns `True` or `False` immediately it has a decision.
- *We can return from a method any time*. We do not have to wait until the end of its code has been reached. (This technique can help keep your code succinct.)

Adding another instance method

- Looking again at our `is_later_than()` method, it could require making many comparisons (through `if` statements) before coming to a conclusion.
- The problem is we have potentially many attributes to compare (`hour`, `minute` and `second` from each of `self` and `other`).
- It might help if we could could convert all of the attributes of `self` and `other` into a single number and compare them instead.
- Any ideas on how to proceed?
- Well, if we convert each `Time` instance's attributes to a total number of seconds since midnight (`00:00:00`) then comparing two times can be done simply with `>`, `<`, `==`, etc.
- So we need to add another method to our class called `time_to_seconds()` that specifies a single `Time` object parameter and returns a single number representing the corresponding number of seconds since midnight.
- The method will be another of our `Time` class's instance methods.
- We will call this helper method from our updated `is_later_than()` method. Putting everything together we get the following.

```
class Time(object):

    def __init__(self, hour=0, minute=0, second=0):
        self.hour = hour
        self.minute = minute
        self.second = second

    def time_to_seconds(self):
        return self.hour*60*60 + self.minute*60 + self.second
```

```python
    def is_later_than(self, other):
        return self.time_to_seconds() > other.time_to_seconds()

    def show_time(self):
        print('The time is {:02d}:{:02d}:{:02d}'.format(
            self.hour, self.minute, self.second))

t1 = Time(13, 43, 6)
t2 = Time(14, 52, 7)
t3 = Time(14, 43, 7)
t4 = Time(13, 43, 7)

print(t2.is_later_than(t1))
print(t1.is_later_than(t2))
print(t3.is_later_than(t2))
print(t4.is_later_than(t1))
```

```
True
False
False
True
```

Adding another instance method

- Let's extend our `Time` class with a more complex instance method. This one will take two `Time` objects and add them to produce and return a *new* `Time` object.
- We want the new `Time` object to be a valid time in the 24-hour format.
- Again, when writing a new method we start with its signature i.e. how will we invoke the method?
- It seems it ought to work as follows: `t3 = t1.plus(t2)` where `t3` is the result of adding time `t2` to `t1`.
- The most straightforward approach to coding our new `plus()` method would seem to be to take the two `Time` instances passed to it and firstly convert each to an equivalent number of seconds.
- We can then add the seconds in each to produce a total number of seconds.
- Finally we need to convert this total number of seconds back into a valid `Time` object to be returned to the caller.
- To to the conversion we will have to add another helper method `seconds_to_time()`.
- Where will we put the helper method `seconds_to_time()`? This is an interesting question. Is it an instance method?
- If it were an instance method we would add it to the class definition as we have done with all of our methods so far.
- It is *not* an instance method however.
- How do we know it is not an instance method?
- *Because it is a method that it makes sense to call in the absence of an instance of the Time class.*
- In other words we should not be required to have an instance of `Time` in order to invoke the method `seconds_to_time()`.
- All we should require is a number of seconds from which we want the method to derive an instance of the class `Time`.
- Given it is not an instance method, for now we will simply add `seconds_to_time()` as a *function* to the *module* containing the definition of our `Time` class.

- The `seconds_to_time()` function makes use of `divmod()` to help us avoid generating `Times` such as `26:78:91`
- What does `divmod()` do? Well `minute, second = divmod(s, 60)` divides `s` by `60` and puts the resulting whole number of minutes in `minute` with any remainder going in `second`.
- So `1, 20 == divmod(80, 60)` or "80 seconds is equal to 1 minute 20 seconds".
- We apply similar logic to working out the final number of minutes and hours in our new `Time` object.
- Our updated `Time` class looks like this.

```python
class Time(object):

    def __init__(self, hour=0, minute=0, second=0):
        self.hour = hour
        self.minute = minute
        self.second = second

    def time_to_seconds(self):
        return self.hour*60*60 + self.minute*60 + self.second

    def is_later_than(self, other):
        return self.time_to_seconds() > other.time_to_seconds()

    def plus(self, other):
        return seconds_to_time(self.time_to_seconds() +
                               other.time_to_seconds())

    def __str__(self):
        return 'The time is {:02d}:{:02d}:{:02d}'.format(
            self.hour, self.minute, self.second)

def seconds_to_time(s):
    minute, second = divmod(s, 60)
    hour, minute = divmod(minute, 60)
    overflow, hour = divmod(hour, 24)
    return Time(hour, minute, second)
```

- Let's see our new method in action.

```python
t1 = Time(13, 58, 23)
t2 = Time(0, 10, 0)
t3 = t1.plus(t2)
print(t3)
t4 = Time(16, 18, 36)
t5 = Time(12, 10, 19)
t6 = t4.plus(t5)
print(t6)
```

```
The time is 14:08:23
The time is 04:28:55
```

- Note we do *not* want `t3` to be `13:68:23` as that would not be a a valid time in the 24-hour format. For similar reasons `t6` should not be `28:28:55`.
- So we have correctly handled wraparound in our new method thanks to `divmod()`.