Think Python 2e, Chapter 16 Notes

Classes and Methods

October 21, 2022

Methods

- Need tighter relationship between classes and the functions that deal with them.
- Methods are semantically the same as functions.
- The syntax for methods is different from functions.
- Methods are defined inside a class definition.
- This makes the relation between class and method explicit.

NON object-oriented way

```
class Time:
    """Represents the time of day."""

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

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

Only way to call the function:

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

Object-oriented way

```
class Time:
    """Represents the time of day."""
    def print_time(time):
        print('%.2d:%.2d:%.2d' %
              (time.hour, time.minute, time.second))
```

There are now two ways to call the function:

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

- The second is more concise.
- `start` is the actual parameter bound to `time`
- `start` is called the **subject**

self

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

- By convention, the formal parameter is usually called `self`

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

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

Function-oriented *vs.* object-oriented programming

Function is focus:

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

Object is focus:

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

Function-oriented *vs.* object-oriented programming

Function is focus:

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

Object is focus:

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

- Notice you can write `time_to_int` as a method, but not `timt_to_time`.
- Why not?

increment

```
# inside class Time:

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

- This is a pure function

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

- `increment` is defined with two formal parameters
- `increment` is called with one subject and one actual parameter

Error message can be confusing

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

- But I only gave two parameters!

Error message can be confusing

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

- But I only gave two parameters!
- Wrong! You gave the subject and two parameters.
- That's three

Positional arguments

- A **positional argument** is an argument that doesn't have a parameter name; that is, it is not a keyword argument.

```
1 sketch(parrot, cage, dead=True)
```

- `parrot` and `cage` are positional, and `dead` is a keyword argument.

Methods with two objects

```
# inside class Time:

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

- `self` and `other` are conventional names.

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

__init__

```
# inside class Time:

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

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

__str__

```
# inside class Time:

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

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

Operator overloading

- Every operator in Python has a dunder method to overload it.
- Here we overload addition, *i.e.* the + operator.

```
# inside class Time:

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

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