The Object Metaphor in Python

In the beginning of this text, we distinguished between functions and data: functions performed operations and data were operated upon. When we included function values among our data, we acknowledged that data too can have behavior. Functions could be operated upon like data, but could also be called to perform computation.

In this text, *objects* will serve as our central programming metaphor for data values that also have behavior. Objects represent information, but also *behave* like the abstract concepts that they represent. The logic of how an object interacts with other objects is bundled along with the information that encodes the object's value. When an object is printed, it knows how to spell itself out as letters and numerals. If an object is composed of parts, it knows how to reveal those parts on demand. Objects are both information and processes, bundled together to represent the properties, interactions, and behaviors of complex things.

The object metaphor is implemented in Python through specialized object syntax and associated terminology, which we can introduce by example. A date is a kind of simple object.

```
>>> from datetime import date
```

The name `date` is bound to a *class*. A class represents a kind of object. Individual dates are called *instances* of that class, and they can be *constructed* by calling the class as a function on arguments that characterize the instance.

```
>>> today = date(2012, 9, 10)
```

While `today` was constructed from primitive numbers, it behaves like a date. For instance, subtracting it from another date will give a time difference, which we can display as a line of text by calling `str`.

```
>>> str(date(2012, 11, 30) - today)
'81 days, 0:00:00'
```

Objects have *attributes*, which are named values that are part of the object. In Python, like many other programming languages, we use dot notation to designated an attribute of an object.

<expression> . <name>

Above, the `<expression>` evaluates to an object, and `<name>` is the name of an attribute for that object.

Unlike the names that we have considered so far, these attribute names are not available in the general environment. Instead, attribute names are particular to the object instance preceding the dot.

```
>>> today.year
2012
```

Objects also have *methods*, which are function-valued attributes. Metaphorically, we say that the object "knows" how to carry out those methods. By implementation, methods are functions that compute their results from both their arguments and their object. For example, The `strftime` method (a classic function name meant to evoke "string format of time") of `today` takes a single argument that specifies how to display a date (e.g., `%A` means that the day of the week should be spelled out in full).

```
>>> today.strftime('%A, %B %d')
'Monday, September 10'
```

Computing the return value of `strftime` requires two inputs: the string that describes the format of the output and the date information bundled into `today`. Date-specific logic is applied within this method to yield this result. We never stated that the 10th of September, 2012, was a Monday, but knowing one's weekday is part of what it means to be a date. By bundling behavior and information together, this Python object offers us a convincing, self-contained abstraction of a date.

Dot notation provides another form of combined expression in Python. Dot notation also has a well-defined evaluation procedure. However, developing a precise account of how

dot notation is evaluated will have to wait until we introduce the full paradigm of object-oriented programming over the next several sections.

Source : http://inst.eecs.berkeley.edu/~cs61A/book/chapters/objects.html#the-object-metaphor