1.4 Designing Functions

Video: Show Hide

Functions are an essential ingredient of all programs, large and small, and serve as our primary medium to express computational processes in a programming language. So far, we have discussed the formal properties of functions and how they are applied. We now turn to the topic of what makes a good function. Fundamentally, the qualities of good functions all reinforce the idea that functions are abstractions.

- Each function should have exactly one job. That job should be identifiable with a short name and characterizable in a single line of text. Functions that perform multiple jobs in sequence should be divided into multiple functions.
- *Don't repeat yourself* is a central tenet of software engineering. The so-called DRY principle states that multiple fragments of code should not describe redundant logic. Instead, that logic should be implemented once, given a name, and applied multiple times. If you find yourself copying and pasting a block of code, you have probably found an opportunity for functional abstraction.
- Functions should be defined generally. Squaring is not in the Python Library precisely because it is a special case of the `pow` function, which raises numbers to arbitrary powers.

These guidelines improve the readability of code, reduce the number of errors, and often minimize the total amount of code written. Decomposing a complex task into concise functions is a skill that takes experience to master. Fortunately, Python provides several features to support your efforts.

1.4.1 Documentation

A function definition will often include documentation describing the function, called a *docstring*, which must be indented along with the function body. Docstrings are conventionally triple quoted. The first line describes the job of the function in one line. The following lines can describe arguments and clarify the behavior of the function:

```
>>> def pressure(v, t, n):
        """Compute the pressure in pascals of an ideal gas.

        Applies the ideal gas law: http://en.wikipedia.org/wiki/Ideal_gas_law

        v -- volume of gas, in cubic meters
        t -- absolute temperature in degrees kelvin
        n -- particles of gas
        """
        k = 1.38e-23  # Boltzmann's constant
        return n * k * t / v
```

When you call `help` with the name of a function as an argument, you see its docstring (type `q` to quit Python help).

```
>>> help(pressure)
```

When writing Python programs, include docstrings for all but the simplest functions. Remember, code is written only once, but often read many times. The Python docs include docstring guidelines that maintain consistency across different Python projects.

Comments. Comments in Python can be attached to the end of a line following the `#` symbol. For example, the comment `Boltzmann's constant` above describes `k`. These comments don't ever appear in Python's `help`, and they are ignored by the interpreter. They exist for humans alone.

1.4.2 Default Argument Values

A consequence of defining general functions is the introduction of additional arguments. Functions with many arguments can be awkward to call and difficult to read.

In Python, we can provide default values for the arguments of a function. When calling that function, arguments with default values are optional. If they are not provided, then the default value is bound to the formal parameter name instead. For instance, if an application commonly computes pressure for one mole of particles, this value can be provided as a default:

```
>>> def pressure(v, t, n=6.022e23):
        """Compute the pressure in pascals of an ideal gas.

        v -- volume of gas, in cubic meters
        t -- absolute temperature in degrees kelvin
        n -- particles of gas (default: one mole)
        """
        k = 1.38e-23  # Boltzmann's constant
        return n * k * t / v
```

The = symbol means two different things in this example, depending on the context in which it is used. In the `def` statement header, = does not perform assignment, but instead indicates a default value to use when the `pressure` function is called. By contrast, the assignment statement to `k` in the body of the function binds the name `k` to an approximation of Boltzmann's constant.

```
>>> pressure(1, 273.15)
2269.974834
>>> pressure(1, 273.15, 3 * 6.022e23)
6809.924502
```

The `pressure` function is defined to take three arguments, but only two are provided in the first call expression above. In this case, the value for `n` is taken from the `def` statement default. If a third argument is provided, the default is ignored.

As a guideline, most data values used in a function's body should be expressed as default values to named arguments, so that they are easy to inspect and can be changed by the function caller. Some values that never change, such as the fundamental constant `k`, can be bound in the function body or in the global frame.

Continue: 1.5 Control