2.1: VALUES AND TYPES

Contributed by Chuck Severance
Clinical Associate Professor (School of Information) at University of Michigan

A *value* is one of the basic things a program works with, like a letter or a number. The values we have seen so far are `1`, `2`, and "Hello, World!"

These values belong to different *types*: `2` is an integer, and "Hello, World!" is a *string*, so called because it contains a "string" of letters. You (and the interpreter) can identify strings because they are enclosed in quotation marks.

The `print` statement also works for integers. We use the `python` command to start the interpreter.

```
python
>>> print(4)
4
```

If you are not sure what type a value has, the interpreter can tell you.

```
>>> type('Hello, World!')
<class 'str'>
>>> type(17)
<class 'int'>
```

Not surprisingly, strings belong to the type `str` and integers belong to the type `int`. Less obviously, numbers with a decimal point belong to a type called `float`, because these numbers are represented in a format called *floating point*.

```
>>> type(3.2)
<class 'float'>
```

What about values like "17" and "3.2"? They look like numbers, but they are in quotation marks like strings.

```
>>> type('17')
<class 'str'>
>>> type('3.2')
<class 'str'>
```

They're strings.

When you type a large integer, you might be tempted to use commas between groups of three digits, as in `1,000,000`. This is not a legal integer in Python, but it is legal:

```
>>> print(1,000,000)
1 0 0
```

Well, that's not what we expected at all! Python interprets `1,000,000` as a comma-separated sequence of integers, which it prints with spaces between.

This is the first example we have seen of a semantic error: the code runs without producing an error message, but it doesn't do the "right" thing.