behave almost imperatively, allowing beginners to play with tensors much more easily. You will learn about imperative versus declarative style in greater depth later in this chapter.

Example 2-1. Initialize an interactive TensorFlow session

```
>>> import tensorflow as tf
>>> tf.InteractiveSession()
<tensorflow.python.client.session.InteractiveSession>
```

The rest of the code in this section will assume that an interactive session has been loaded.

Initializing Constant Tensors

Until now, we've discussed tensors as abstract mathematical entities. However, a system like TensorFlow must run on a real computer, so any tensors must live on computer memory in order to be useful to computer programmers. TensorFlow provides a number of functions that instantiate basic tensors in memory. The simplest of these are `tf.zeros()` and `tf.ones()`. `tf.zeros()` takes a tensor shape (represented as a Python tuple) and returns a tensor of that shape filled with zeros. Let's try invoking this command in the shell (Example 2-2).

Example 2-2. Create a zeros tensor

```
>>> tf.zeros(2)
<tf.Tensor 'zeros:0' shape=(2,) dtype=float32>
```

TensorFlow returns a reference to the desired tensor rather than the value of the tensor itself. To force the value of the tensor to be returned, we will use the method `tf.Tensor.eval()` of tensor objects (Example 2-3). Since we have initialized `tf.InteractiveSession()`, this method will return the value of the zeros tensor to us.

Example 2-3. Evaluate the value of a tensor

```
>>> a = tf.zeros(2)
>>> a.eval()
array([ 0.,  0.], dtype=float32)
```

Note that the evaluated value of the TensorFlow tensor is itself a Python object. In particular, `a.eval()` is a `numpy.ndarray` object. NumPy is a sophisticated numerical system for Python. We won't attempt an in-depth discussion of NumPy here beyond noting that TensorFlow is designed to be compatible with NumPy conventions to a large degree.

We can call `tf.zeros()` and `tf.ones()` to create and display tensors of various sizes (Example 2-4).

Example 2-4. Evaluate and display tensors

```
>>> a = tf.zeros((2, 3))
>>> a.eval()
array([[ 0.,  0.,  0.],
       [ 0.,  0.,  0.]], dtype=float32)
>>> b = tf.ones((2,2,2))
>>> b.eval()
array([[[ 1.,  1.],
        [ 1.,  1.]],

       [[ 1.,  1.],
        [ 1.,  1.]]], dtype=float32)
```

What if we'd like a tensor filled with some quantity besides 0/1? The `tf.fill()` method provides a nice shortcut for doing so (Example 2-5).

Example 2-5. Filling tensors with arbitrary values

```
>>> b = tf.fill((2, 2), value=5.)
>>> b.eval()
array([[ 5.,  5.],
       [ 5.,  5.]], dtype=float32)
```

`tf.constant` is another function, similar to `tf.fill`, which allows for construction of tensors that shouldn't change during the program execution (Example 2-6).

Example 2-6. Creating constant tensors

```
>>> a = tf.constant(3)
>>> a.eval()
3
```

Sampling Random Tensors

Although working with constant tensors is convenient for testing ideas, it's much more common to initialize tensors with random values. The most common way to do this is to sample each entry in the tensor from a random distribution. `tf.random_normal` allows for each entry in a tensor of specified shape to be sampled from a Normal distribution of specified mean and standard deviation (Example 2-7).