```
>>> c = tf.matmul(a, b)
>>> c.eval()
array([[ 3.,  3.,  3.,  3.],
       [ 3.,  3.,  3.,  3.]], dtype=float32)
```

You can check that this answer matches the mathematical definition of matrix multiplication we provided earlier.

Tensor Types

You may have noticed the `dtype` notation in the preceding examples. Tensors in TensorFlow come in a variety of types such as `tf.float32`, `tf.float64`, `tf.int32`, `tf.int64`. It's possible to to create tensors of specified types by setting `dtype` in tensor construction functions. Furthermore, given a tensor, it's possible to change its type using casting functions such as `tf.to_double()`, `tf.to_float()`, `tf.to_int32()`, `tf.to_int64()`, and others (Example 2-15).

Example 2-15. Creating tensors of different types

```
>>> a = tf.ones((2,2), dtype=tf.int32)
>>> a.eval()
array([[0, 0],
       [0, 0]], dtype=int32)
>>> b = tf.to_float(a)
>>> b.eval()
array([[ 0.,  0.],
       [ 0.,  0.]], dtype=float32)
```

Tensor Shape Manipulations

Within TensorFlow, tensors are just collections of numbers written in memory. The different shapes are views into the underlying set of numbers that provide different ways of interacting with the set of numbers. At different times, it can be useful to view the same set of numbers as forming tensors with different shapes. `tf.reshape()` allows tensors to be converted into tensors with different shapes (Example 2-16).

Example 2-16. Manipulating tensor shapes

```
>>> a = tf.ones(8)
>>> a.eval()
array([ 1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.], dtype=float32)
>>> b = tf.reshape(a, (4, 2))
>>> b.eval()
array([[ 1.,  1.],
       [ 1.,  1.],
       [ 1.,  1.],
       [ 1.,  1.]], dtype=float32)
```

```
>>> c = tf.reshape(a, (2, 2, 2))
>>> c.eval()
array([[[ 1.,  1.],
        [ 1.,  1.]],

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

Notice how we can turn the original rank-1 tensor into a rank-2 tensor and then into a rank-3 tensor with `tf.reshape`. While all necessary shape manipulations can be performed with `tf.reshape()`, sometimes it can be convenient to perform simpler shape manipulations using functions such as `tf.expand_dims` or `tf.squeeze`. `tf.expand_dims` adds an extra dimension to a tensor of size 1. It's useful for increasing the rank of a tensor by one (for example, when converting a rank-1 vector into a rank-2 row vector or column vector). `tf.squeeze`, on the other hand, removes all dimensions of size 1 from a tensor. It's a useful way to convert a row or column vector into a flat vector.

This is also a convenient opportunity to introduce the `tf.Tensor.get_shape()` method (Example 2-17). This method lets users query the shape of a tensor.

Example 2-17. Getting the shape of a tensor

```
>>> a = tf.ones(2)
>>> a.get_shape()
TensorShape([Dimension(2)])
>>> a.eval()
array([ 1.,  1.], dtype=float32)
>>> b = tf.expand_dims(a, 0)
>>> b.get_shape()
TensorShape([Dimension(1), Dimension(2)])
>>> b.eval()
array([[ 1.,  1.]], dtype=float32)
>>> c = tf.expand_dims(a, 1)
>>> c.get_shape()
TensorShape([Dimension(2), Dimension(1)])
>>> c.eval()
array([[ 1.],
       [ 1.]], dtype=float32)
>>> d = tf.squeeze(b)
>>> d.get_shape()
TensorShape([Dimension(2)])
>>> d.eval()
array([ 1.,  1.], dtype=float32)
```

Introduction to Broadcasting

Broadcasting is a term (introduced by NumPy) for when a tensor system's matrices and vectors of different sizes can be added together. These rules allow for conveniences like adding a vector to every row of a matrix. Broadcasting rules can be quite complex, so we will not dive into a formal discussion of the rules. It's often easier to experiment and see how the broadcasting works (Example 2-18).

Example 2-18. Examples of broadcasting

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

Notice that the vector `b` is added to every row of matrix `a`. Notice another subtlety; we explicitly set the `dtype` for `b`. If the `dtype` isn't set, TensorFlow will report a type error. Let's see what would have happened if we hadn't set the `dtype` (Example 2-19).

Example 2-19. TensorFlow doesn't perform implicit type casting

```
>>> b = tf.range(0, 2, 1)
>>> b.eval()
array([0, 1], dtype=int32)
>>> c = a + b
ValueError: Tensor conversion requested dtype float32 for Tensor with dtype int32:
'Tensor("range_2:0", shape=(2,), dtype=int32)
```

Unlike languages like C, TensorFlow doesn't perform implicit type casting under the hood. It's often necessary to perform explicit type casts when doing arithmetic operations.

Imperative and Declarative Programming

Most situations in computer science involve imperative programming. Consider a simple Python program (Example 2-20).