Diving In

Convention dictates that I should bore you with the fundamental building blocks of programming, so we can slowly work up to building something useful. Let's skip all that. Here is a complete, working Python program. It probably makes absolutely no sense to you. Don't worry about that, because you're going to dissect it line by line. But read through it first and see what, if anything, you can make of it.

```
SUFFIXES = {1000: ['KB', 'MB', 'GB', 'TB', 'PB', 'EB', 'ZB', 'YB'],
            1024: ['KiB', 'MiB', 'GiB', 'TiB', 'PiB', 'EiB', 'ZiB', 'YiB']}

def approximate_size(size, a_kilobyte_is_1024_bytes=True):
    '''Convert a file size to human-readable form.

    Keyword arguments:
    size -- file size in bytes
    a_kilobyte_is_1024_bytes -- if True (default), use multiples of 1024
                                if False, use multiples of 1000

    Returns: string

    '''
    if size < 0:
        raise ValueError('number must be non-negative')

    multiple = 1024 if a_kilobyte_is_1024_bytes else 1000
    for suffix in SUFFIXES[multiple]:
        size /= multiple
        if size < multiple:
            return '{0:.1f} {1}'.format(size, suffix)

    raise ValueError('number too large')

if __name__ == '__main__':
    print(approximate_size(1000000000000, False))
    print(approximate_size(1000000000000))
```

Now let's run this program. The output will look something like this:

```
1.0 TB
931.3 GiB
```

What just happened? You executed your first Python program. The Python interpreter executed the script you wrote. The script defines a single function, the `approximate_size()` function, which takes an exact file size in bytes and calculates a "pretty" (but approximate) size. (You've probably seen this in Windows Explorer, or the Mac OS X Finder, or Nautilus or Dolphin or Thunar on Linux. If you display a folder of documents as a multi-column list, it will display a table with the document icon, the document name, the size, type, last-modified date, and so on. If the folder contains a 1093-byte file named `TODO`, your file manager won't display `TODO 1093 bytes`; it'll say something like `TODO 1 KB` instead. That's what the `approximate_size()` function does.)

Look at the bottom of the script, and you'll see two calls to `print(approximate_size(arguments))`. These are function calls — first calling the `approximate_size()` function and passing a number of arguments, then taking the return value and passing it straight on to the `print()` function. The `print()` function is built-in; you'll never see an explicit declaration of it. You can just use it, anytime, anywhere. (There are lots of built-in functions, and lots more functions that are separated into `modules`. Patience, grasshopper.)

So why does running the script on the command line give you the same output every time? We'll get to that. First, let's look at that `approximate_size()` function.