Generators and OS

Generators

We can create a generator as follows:

```
def build_squares(n):
    for i in range(n):
        yield i**2
```

The corresponding iterable looks like this:

```
def build_squares(n):
    result = []
    for i in range(n):
        result.append(i**2)
    return result
```

With a generator, we can suspend our function call halfway through running it and do something with the current item. This prevents us from holding a lot of data in memory at one point.

```
for i in build_squares(3):
    print(i)
```

Here, when `build_squares()` hits the `yield` statement, control is returned to the for loop above, along with a value for i, and when it's called again, `build_squares()` will resume from the next valid statement after the `yield` statement. In this case, that means each call will yield the next square.

OS

```
import os

output = os.open("cp ~/file1.txt ~/file2.txt").read()
```

The `open()` command allows us to run commands on the command line from Python. We can then use `.read()` to pull in the output from the command.

You could also use string manipulation, e.g.:

```
output = os.open("cp {0} {1}".format(filename1,
filename2)).read()
```

This can be useful for calling programs in other languages from within Python.