GENERATORS AND CONTEXT MANAGERS

Context Manager Pattern

create context manager

enter context (and, optionally, receive an object)

do some work

exit context

```
with open(file_name) as f:
  data = file.readlines()
```

Mimic Pattern using a Generator

```
def open_file(fname, mode):
    f = open(fname, mode)
    try:
        yield f
    finally:
        f.close()
```

```
ctx = open_file('file.txt', 'r')
f = next(ctx)

try:
    # do work with file
finally:
    try:
        next(ctx)
    except StopIteration:
        pass
```

```
ctx = open_file('file.txt', 'r')
```

```
f = next(ctx)
```

opens file, and yields it

```
next(ctx)
```

closes file

→ StopIteration exception

This works in general

```
def gen(args):
    # do set up work here

    try:
        yield object

    finally:
        # clean up object here
```

```
ctx = gen(…)
obj = next(ctx)

try:
    # do work with obj
finally:
    try:
        next(ctx)
    except StopIteration:
        pass
```

This is quite clunky still

but you should see that we can almost create a context manager pattern using a generator function!

Creating a Context Manager from a Generator Function

```
def open_file(fname, mode):
    f = open(fname, mode)
    try:
        yield f
    finally:
        f.close()
```

generator function

generator object → `gen = open_file('test.txt', 'w')`

```
f = next(gen)
    # do work with f
next(f) → closes f
```

```
class GenContext:

    def __init__(self, gen):
        self.gen = gen

    def __enter__(self):
        obj = next(self.gen)
        return obj

    def __exit__(self, exc_type, exc_value, exc_tb):
        next(self.gen)
        return False
```

```
gen = open_file('test.txt', 'w')
with GenContext(gen) as f:
    # do work
```

Code Exercises