Exercise 52: The Start Of Your Web Game

We're coming to the end of the book, and in this exercise I'm going to really challenge you. When you're done, you'll be a reasonably competent Python beginner. You'll still need to go through a few more books and write a couple more projects, but you'll have the skills to complete them. The only thing in your way will be time, motivation, and resources.

In this exercise, we won't make a complete game, but instead we'll make an "engine" that can run the game from Exercise 47 in the browser. This will involve refactoring Exercise 43, mixing in the structure from Exercise 47, adding automated tests, and finally creating a web engine that can run the games.

This exercise will be *huge*, and I predict you could spend anywhere from a week to months on it before moving on. It's best to attack it in little chunks and do a bit a night, taking your time to make everything work before moving on.

Refactoring the Exercise 43 Game

You've been altering the `gothonweb` project for two exercises and you'll do it one more time in this exercise. The skill you're learning is called "refactoring," or as I like to call it, "fixing stuff." Refactoring is a term programmers use to describe the process of taking old code, and changing it to have new features or just to clean it up. You've been doing this without even knowing it, as it's second nature to building software.

What you'll do in this part is take the ideas from Exercise 47 of a testable "map" of Rooms, and the game from Exercise 43, and combine them together to create a new game structure. It will have the same content, just "refactored" to have a better structure.

The first step is to grab the code from `ex47/game.py` and copy it to `gothonweb/map.py` and copy the `tests/ex47_tests.py` file to `tests/map_tests.py` and run `nosetests` again to make sure it keeps working.

> **NOTE**
>
> From now on I won't show you the output of a test run just assume that you should be doing it and it'll look like the above unless you have an error.

Once you have the code from Exercise 47 copied over, it's time to refactor it to have the Exercise 43 map in it. I'm going to start off by laying down the basic structure, and then you'll have an assignment to make the `map.py` file and the `map_tests.py` file complete.

Lay out the basic structure of the map using the `Room` class as it is now:

```
class Room(object):

    def __init__(self, name, description):
        self.name = name
        self.description = description
        self.paths = {}

    def go(self, direction):
        return self.paths.get(direction, None)

    def add_paths(self, paths):
        self.paths.update(paths)


central_corridor = Room("Central Corridor",
"""
The Gothons of Planet Percal #25 have invaded your ship and destroyed
your entire crew.  You are the last surviving member and your last
mission is to get the neutron destruct bomb from the Weapons Armory,
put it in the bridge, and blow the ship up after getting into an
escape pod.

You're running down the central corridor to the Weapons Armory when
a Gothon jumps out, red scaly skin, dark grimy teeth, and evil clown costume
flowing around his hate filled body.  He's blocking the door to the
Armory and about to pull a weapon to blast you.
""")


laser_weapon_armory = Room("Laser Weapon Armory",
"""
Lucky for you they made you learn Gothon insults in the academy.
You tell the one Gothon joke you know:
Lbhe zbgure vf fb sng, jura fur fvgf nebhaq gur ubhfr, fur fvgf nebhaq gur ubhfr.
The Gothon stops, tries not to laugh, then busts out laughing and can't move.
While he's laughing you run up and shoot him square in the head
putting him down, then jump through the Weapon Armory door.

You do a dive roll into the Weapon Armory, crouch and scan the room
for more Gothons that might be hiding.  It's dead quiet, too quiet.
You stand up and run to the far side of the room and find the
neutron bomb in its container.  There's a keypad lock on the box
and you need the code to get the bomb out.  If you get the code
wrong 10 times then the lock closes forever and you can't
get the bomb.  The code is 3 digits.
""")


the_bridge = Room("The Bridge",
"""
The container clicks open and the seal breaks, letting gas out.
You grab the neutron bomb and run as fast as you can to the
bridge where you must place it in the right spot.

You burst onto the Bridge with the netron destruct bomb
under your arm and surprise 5 Gothons who are trying to
take control of the ship.  Each of them has an even uglier
clown costume than the last.  They haven't pulled their
weapons out yet, as they see the active bomb under your
arm and don't want to set it off.
""")


escape_pod = Room("Escape Pod",
"""
You point your blaster at the bomb under your arm
and the Gothons put their hands up and start to sweat.
You inch backward to the door, open it, and then carefully
place the bomb on the floor, pointing your blaster at it.
```

```
You then jump back through the door, punch the close button
and blast the lock so the Gothons can't get out.
Now that the bomb is placed you run to the escape pod to
get off this tin can.

You rush through the ship desperately trying to make it to
the escape pod before the whole ship explodes.  It seems like
hardly any Gothons are on the ship, so your run is clear of
interference.  You get to the chamber with the escape pods, and
now need to pick one to take.  Some of them could be damaged
but you don't have time to look.  There's 5 pods, which one
do you take?
""")


the_end_winner = Room("The End",
"""
You jump into pod 2 and hit the eject button.
The pod easily slides out into space heading to
the planet below.  As it flies to the planet, you look
back and see your ship implode then explode like a
bright star, taking out the Gothon ship at the same
time.  You won!
""")


the_end_loser = Room("The End",
"""
You jump into a random pod and hit the eject button.
The pod escapes out into the void of space, then
implodes as the hull ruptures, crushing your body
into jam jelly.
"""
)

escape_pod.add_paths({
    '2': the_end_winner,
    '*': the_end_loser
})

generic_death = Room("death", "You died.")

the_bridge.add_paths({
    'throw the bomb': generic_death,
    'slowly place the bomb': escape_pod
})

laser_weapon_armory.add_paths({
    '0132': the_bridge,
    '*': generic_death
})

central_corridor.add_paths({
    'shoot!': generic_death,
    'dodge!': generic_death,
    'tell a joke': laser_weapon_armory
})

START = central_corridor
```

You'll notice that there are a couple of problems with our `Room` class and this map:

1. We have to put the text that was in the `if-else` clauses that got printed *before* entering a room as part of each room. This means you can't shuffle the map around, which would be nice. You'll be fixing that up in this exercise.
2. There are parts in the original game where we ran code that determined things like the bomb's keypad code, or the right

pod. In this game we just pick some defaults and go with it, but later you'll be given Study Drills to make this work again.
3. I've just made a `generic_death` ending for all of the bad decisions, which you'll have to finish for me. You'll need to go back through and add in all the original endings and make sure they work.
4. I've got a new kind of transition labeled `"*"` that will be used for a "catch-all" action in the engine.

Once you've got that basically written out, here's the new automated test `tests/map_test.py` that you should have to get yourself started:

```
from nose.tools import *
from gothonweb.map import *

def test_room():
    gold = Room("GoldRoom",
                """This room has gold in it you can grab. There's a
                door to the north.""")
    assert_equal(gold.name, "GoldRoom")
    assert_equal(gold.paths, {})

def test_room_paths():
    center = Room("Center", "Test room in the center.")
    north = Room("North", "Test room in the north.")
    south = Room("South", "Test room in the south.")

    center.add_paths({'north': north, 'south': south})
    assert_equal(center.go('north'), north)
    assert_equal(center.go('south'), south)

def test_map():
    start = Room("Start", "You can go west and down a hole.")
    west = Room("Trees", "There are trees here, you can go east.")
    down = Room("Dungeon", "It's dark down here, you can go up.")

    start.add_paths({'west': west, 'down': down})
    west.add_paths({'east': start})
    down.add_paths({'up': start})

    assert_equal(start.go('west'), west)
    assert_equal(start.go('west').go('east'), start)
    assert_equal(start.go('down').go('up'), start)

def test_gothon_game_map():
    assert_equal(START.go('shoot!'), generic_death)
    assert_equal(START.go('dodge!'), generic_death)

    room = START.go('tell a joke')
    assert_equal(room, laser_weapon_armory)
```

Your task in this part of the exercise is to complete the map, and make the automated test completely validate the whole map. This includes fixing all the `generic_death` objects to be real endings. Make sure this works really well and that your test is as complete as possible because we'll be changing this map later and you'll use the tests to make sure it keeps working.

Sessions and Tracking Users

At a certain point in your web application you'll need to keep track of some information and associate it with the user's browser. The web (because of HTTP) is what we like to call "stateless," which means each request you make is independent of any other requests being made. If you request page A, put in some data, and click a link to page B, all the data you sent to page A just

disappears.

The solution to this is to create a little data store (usually in a database or on the disk) that uses a number unique to each browser to keep track of what that browser was doing. This is called "session tracking" and uses Cookies in the browser to maintain the state of the user through the application. In the little `lpthw.web` framework it's fairly easy, and there's an example showing how it's done:

```
import web

web.config.debug = False

urls = (
    "/count", "count",
    "/reset", "reset"
)
app = web.application(urls, locals())
store = web.session.DiskStore('sessions')
session = web.session.Session(app, store, initializer={'count': 0})

class count:
    def GET(self):
        session.count += 1
        return str(session.count)

class reset:
    def GET(self):
        session.kill()
        return ""

if __name__ == "__main__":
    app.run()
```

To make this work, you need to create a `sessions/` directory where the application can put session storage. Do that, run this application, and go to `/count`. Hit refresh and watch the counter go up. Close the browser and it *forgets* who you are, which is what we want for the game. There's a way to make the browser remember forever, but that makes testing and development harder. If you then go to `/reset` and back to `/count` you can see your counter reset because you've killed the session.

Take the time to understand this code so you can see how the session starts off with the `count` equal to 0. Also try looking at the files in `sessions/` to see if you can open them up. Here's a Python session where I open up one and decode it:

```
>>> import pickle
>>> import base64
>>> base64.b64decode(open("sessions/XXXXX").read())
"
(dp1\nS'count'\np2\nI1\nsS'ip'\np3\nV127.0.0.1\np4\nsS'session
_id'\np5\nS'XXXX'\np6\ns."
>>>
>>> x = base64.b64decode(open("sessions/XXXXX").read())
 >>>
>>> pickle.loads(x)
 {'count': 1, 'ip': u'127.0.0.1', 'session_id': 'XXXXX'}
```

The sessions are really just dictionaries that get written to disk using `pickle` and `base64` libraries. There are probably as many ways to store and manage sessions as there are web frameworks, so it's not too important to know how these work. It does help if you need to debug the session or potentially clean it out.

Creating an Engine

You should have your game map working and a good unit test for it. I now want you to make a simple little game engine that will run the rooms, collect input from the player, and keep track of where a player is in the game. We'll be using the sessions you just learned to make a simple game engine that will:

1. Start a new game for new users.
2. Present the room to the user.
3. Take input from the user.
4. Run user input through the game.
5. Display the results and keep going until the user dies.

To do this, you're going to take the trusty `bin/app.py` you've been hacking on and create a fully working, session-based game engine. The catch is I'm going to make a very simple one with *basic HTML* files, and it'll be up to you to complete it. Here's the base engine:

```
import web
from gothonweb import map

urls = (
  '/game', 'GameEngine',
  '/', 'Index',
)

app = web.application(urls, globals())

# little hack so that debug mode works with sessions
if web.config.get('_session') is None:
    store = web.session.DiskStore('sessions')
    session = web.session.Session(app, store,
                                  initializer={'room': None})
    web.config._session = session
else:
    session = web.config._session

render = web.template.render('templates/', base="layout")


class Index(object):
    def GET(self):
        # this is used to "setup" the session with starting values
        session.room = map.START
        web.seeother("/game")


class GameEngine(object):

    def GET(self):
        if session.room:
            return render.show_room(room=session.room)
        else:
            # why is there here? do you need it?
            return render.you_died()

    def POST(self):
        form = web.input(action=None)

        # there is a bug here, can you fix it?
        if session.room and form.action:
            session.room = session.room.go(form.action)

        web.seeother("/game")

```

```
48 if __name__ == "__main__":
49     app.run()
```

There are even more new things in this script, but amazingly it's an entire web-based game engine in a small file. The biggest "hack" in the script are the lines that bring the sessions back, which is needed so that debug mode reloading works. Otherwise, each time you hit refresh the sessions will disappear and the game won't work.

Before you run bin/app.py you need to change your PYTHONPATH environment variable. Don't know what that is? I know, it's kind of dumb, but you have to learn what this is to run even basic Python programs, but that's how Python people like things.

In your Terminal, type:

```
export PYTHONPATH=$PYTHONPATH:.
```

On Windows PowerShell do:

```
$env:PYTHONPATH = "$env:PYTHONPATH;."
```

You should only have to do it once per shell session, but if you get an import error, then you probably need to do this or you did it wrong.

You should next delete `templates/hello_form.html` and `templates/index.html` and create the two templates mentioned in the above code. Here's a *very* simple `templates/show_room.html`:

```
$def with (room)

<h1> $room.name </h1>

<pre>
$room.description
</pre>

$if room.name == "death":
    <p><a href="/">Play Again?</a></p>
$else:
    <p>
    <form action="/game" method="POST">
        - <input type="text" name="action"> <input type="SUBMIT">
    </form>
    </p>
```

That is the template to show a room as you travel through the game. Next you need one to tell someone they died in the case that they got to the end of the map on accident, which is `templates/you_died.html`:

```
<h1>You Died!</h1>

<p>Looks like you bit the dust.</p>
<p><a href="/">Play Again</a></p>
```

With those in place, you should now be able to do the following:

1. Get the test `tests/app_tests.py` working again so that you are testing the game. You won't be able to do much more than a few clicks in the game because of sessions, but you should be able to do some basics.
2. Remove the `sessions/*` files and make sure you've started over.
3. Run the `python bin/app.py` script and test out the game.

You should be able to refresh and fix the game like normal, and work with the game HTML and engine until it does all the things you want it to do.

Your Final Exam

Do you feel like this was a huge amount of information thrown at you all at once? Good, I want you to have something to tinker with while you build your skills. To complete this exercise, I'm going to give you a final set of exercises for you to complete on your own. You'll notice that what you've written so far isn't very well built; it is just a first version of the code. Your task now is to make the game more complete by doing these things:

1. Fix all the bugs I mention in the code, and any that I didn't mention. If you find new bugs, let me know.
2. Improve all of the automated tests so that you test more of the application and get to a point where you use a test rather than your browser to check the application while you work.
3. Make the HTML look better.
4. Research logins and create a signup system for the application, so people can have logins and high scores.
5. Complete the game map, making it as large and feature-complete as possible.
6. Give people a "help" system that lets them ask what they can do at each room in the game.
7. Add any other features you can think of to the game.
8. Create several "maps" and let people choose a game they want to run. Your `bin/app.py` engine should be able to run any map of rooms you give it, so you can support multiple games.
9. Finally, use what you learned in Exercises 48 and 49 to create a better input processor. You have most of the code necessary; you just need to improve the grammar and hook it up to your input form and the `GameEngine`.

Good luck!

Common Student Questions

Q: I'm using `sessions` in my game and I can't test it with `nosetests`.

> You need to read about sessions in the reloader http://webpy.org/cookbook/session_with_reloader .

Q: I get an `ImportError`.

> Wrong directory. Wrong Python version. `PYTHONPATH` not set. No `__init__.py` file. Spelling mistake in import.

Video

- All 52 videos, 1 per exercise, almost 2G of video.
- A PDF of the book.
- Email help from the author.
- See a list of everything you get before you buy.

When you buy the videos they will immediately show up **right here** without any hassles.

Already Paid? Reactivate Your Purchase Right Now!

Full Name

Email Address

Email Address

Pay With Credit Card (by Stripe™)

Use your PayPal™ account.

Buy Learn Python The Hard Way, 3rd Edition

Zed Shaw
PDF + Videos + Updates
$29.95

Amazon
Paper + DVD
$22.74

InformIT
eBook + Paper
$43.19

Amazon
Kindle
$17.27

B&N
Paper + DVD
$22.96

B&N
Nook (No Video)
$17.27

Interested In Ruby?
Ruby is also a great language.
Learn Ruby The Hard Way