Exercise 43: Basic Object-Oriented Analysis and Design

I'm going to describe a process to use when you want to build something using Python, specifically with object-oriented programming (OOP). What I mean by a "process" is that I'll give you a set of steps that you do in order, but that you aren't meant to be a slave to or that will totally always work for every problem. They are just a good starting point for many programming problems and shouldn't be considered the *only* way to solve these types of problems. This process is just one way to do it that you can follow.

The process is as follows:

1. Write or draw about the problem.
2. Extract key concepts from 1 and research them.
3. Create a class hierarchy and object map for the concepts.
4. Code the classes and a test to run them.
5. Repeat and refine.

The way to look at this process is that it is "top down," meaning it starts from the very abstract loose idea and then slowly refines it until the idea is solid and something you can code.

I start by just writing about the problem and trying to think up anything I can about it. Maybe I'll even draw a diagram or two, maybe a map of some kind, or even write myself a series of emails describing the problem. This gives me a way to express the key concepts in the problem and also explore what I might already know about it.

Then I go through these notes, drawings, and descriptions and I pull out the key concepts. There's a simple trick to doing this: Simply make a list of all the *nouns* and *verbs* in your writing and drawings, then write out how they're related. This gives me a good list of names for classes, objects, and functions in the next step. I take this list of concepts and then research any that I don't understand so I can refine them further if I needed.

Once I have my list of concepts I create a simple outline/tree of the concepts and how they are related as classes. You can usually take your list of nouns and start asking "Is this one like other concept nouns? That means they have a common parent class, so what is it called?" Keep doing this until you have a class hierarchy that's just a simple tree list or a diagram. Then take the *verbs* you have and see if those are function names for each class and put them in your tree.

With this class hierarchy figured out, I sit down and write some basic skeleton code that has just the classes, their functions, and nothing more. I then write a test that runs this code and makes sure the classes I've made make sense and work right. Sometimes I may write the test first though, and other times I might write a little test, a little code, a little test, etc. until I have the whole thing built.

Finally, I keep cycling over this process repeating it and refining as I go and making it as clear as I can before doing more implementation. If I get stuck at

any particular part because of a concept or problem I haven't anticipated, then I sit down and start the process over on just that part to figure it out more before continuing.

I will now go through this process while coming up with a game engine and a game for this exercise.

The Analysis of a Simple Game Engine

The game I want to make is called "Gothons from Planet Percal #25" and will be a small space adventure game. With nothing more than that concept in my mind I can explore the idea and figure out how to make the game come to life.

Write or Draw About the Problem

I'm going to write a little paragraph for the game:

"Aliens have invaded a space ship and our hero has to go through a maze of rooms defeating them so he can escape into an escape pod to the planet below. The game will be more like a Zork or Adventure type game with text outputs and funny ways to die. The game will involve an engine that runs a map full of rooms or scenes. Each room will print its own description when the player enters it and then tell the engine what room to run next out of the map."

At this point I have a good idea for the game and how it would run, so now I want to describe each scene:

Death
: This is when the player dies and should be something funny.

Central Corridor
: This is the starting point and has a Gothon already standing there they have to defeat with a joke before continuing.

Laser Weapon Armory
: This is where the hero gets a neutron bomb to blow up the ship before getting to the escape pod. It has a keypad the hero has to guess the number for.

The Bridge
: Another battle scene with a Gothon where the hero places the bomb.

Escape Pod
: Where the hero escapes but only after guessing the right escape pod.

At this point I might draw out a map of these, maybe write more descriptions of each room, whatever comes to mind as I explore the problem.

Extract Key Concepts and Research Them

I now have enough information to extract some of the nouns and analyze their class hierarchy. First I make a list of all the nouns:

- Alien
- Player
- Ship
- Maze
- Room
- Scene
- Gothon
- Escape Pod

- Planet
- Map
- Engine
- Death
- Central Corridor
- Laser Weapon Armory
- The Bridge

I would also possibly go through all the verbs and see if they are anything that might be good function names, but I'll skip that for now.

At this point you might also research each of these concepts and anything you don't know right now. For example, I might play a few of these types of games and make sure I know how they work. I might research how ships are designed or how bombs work. Maybe I'll research some technical issue like how to store the game's state in a database. After I've done this research I might start over at step 1 based on new information I have and rewrite my description and extract new concepts.

Create a Class Hierarchy and Object Map for the Concepts

Once I have that I turn it into a class hierarchy by asking "What is similar to other things?" I also ask "What is basically just another word for another thing?"

Right away I see that "Room" and "Scene" are basically the same thing depending on how I want to do things. I'm going to pick "Scene" for this game. Then I see that all the specific rooms like "Central Corridor" are basically just Scenes. I see also that Death is basically a Scene, which confirms my choice of "Scene" over "Room" since you can have a death scene, but a death room is kind of odd. "Maze" and "Map" are basically the same so I'm going to go with "Map" since I used it more often. I don't want to do a battle system so I'm going to ignore "Alien" and "Player" and save that for later. The "Planet" could also just be another scene instead of something specific.

After all of that thought process I start to make a class hierarchy that looks like this in my text editor:

```
* Map
* Engine
* Scene
  * Death
  * Central Corridor
  * Laser Weapon Armory
  * The Bridge
  * Escape Pod
```

I would then go through and figure out what actions are needed on each thing based on verbs in the description. For example, I know from the description I'm going to need a way to "run" the engine, "get the next scene" from the map, get the "opening scene," and "enter" a scene. I'll add those like this:

```
* Map
  - next_scene
  - opening_scene
* Engine
  - play
* Scene
  - enter
  * Death
  * Central Corridor
```

```
* Laser Weapon Armory
* The Bridge
* Escape Pod
```

Notice how I just put `-enter` under `Scene` since I know that all the scenes under it will inherit it and have to override it later.

Code the Classes and a Test to Run Them

Once I have this tree of classes and some of the functions I open up a source file in my editor and try to write the code for it. Usually I'll just copy-paste the tree into the source file and then edit it into classes. Here's a small example of how this might look at first, with a simple little test at the end of the file.

```
1  class Scene(object):
2
3      def enter(self):
4          pass
5
6
7  class Engine(object):
8
9      def __init__(self, scene_map):
10         pass
11
12     def play(self):
13         pass
14
15 class Death(Scene):
16
17     def enter(self):
18         pass
19
20 class CentralCorridor(Scene):
21
22     def enter(self):
23         pass
24
25 class LaserWeaponArmory(Scene):
26
27     def enter(self):
28         pass
29
30 class TheBridge(Scene):
31
32     def enter(self):
33         pass
34
35 class EscapePod(Scene):
36
37     def enter(self):
38         pass
39
40
41 class Map(object):
42
43     def __init__(self, start_scene):
44         pass
45
46     def next_scene(self, scene_name):
47         pass
48
49     def opening_scene(self):
50         pass
51
52
53 a_map = Map('central_corridor')
54 a_game = Engine(a_map)
55 a_game.play()
```

In this file you can see that I simply replicated the hierarchy I wanted and then a little bit of code at the end to run it and see if it all works in this basic structure. In the later sections of this exercise you'll fill in the rest of this code and make it work to match the description of the game.

Repeat and Refine

The last step in my little process isn't so much a step as it is a `while-loop`. You don't ever do this as a one-pass operation. Instead you go back over the whole process again and refine it based on information you've learned from later steps. Sometimes I'll get to step 3 and realize that I need to work on 1 and 2 more, so I'll stop and go back and work on those. Sometimes I'll get a flash of inspiration and jump to the end to code up the solution in my head while I have it there, but then I'll go back and do the previous steps to make sure I cover all the possibilities I have.

The other idea in this process is that it's not just something you do at one single level but something that you can do at every level when you run into a particular problem. Let's say I don't know how to write the `Engine.play` method yet. I can stop and do this whole process on *just* that one function to figure out how to write it.

Top Down vs. Bottom Up

The process is typically labeled "top down" since it starts at the most abstract concepts (the top) and works its way down to actual implementation. I want you to use this process I just described when analyzing problems in the book from now on, but you should know that there's another way to solve problems in programming that starts with code and goes "up" to the abstract concepts. This other way is labeled "bottom up." Here are the general steps you follow to do this:

> 1. Take a small piece of the problem; hack on some code and get it to run barely.
> 2. Refine the code into something more formal with classes and automated tests.
> 3. Extract the key concepts you're using and try to find research for them.
> 4. Write a description of what's really going on.
> 5. Go back and refine the code, possibly throwing it out and starting over.
> 6. Repeat, moving on to some other piece of the problem.

I find this process is better once you're more solid at programming and, are naturally thinking in code about problems. This process is very good when you know small pieces of the overall puzzle, but maybe don't have enough information yet about the overall concept. Breaking it down in little pieces and exploring with code then helps you slowly grind away at the problem until you've solved it. However, remember that your solution will probably be meandering and weird, so that's why my version of this process involves going back and finding research then cleaning things up based on what you've learned.

The Code for "Gothons from Planet Percal #25"

Stop! I'm going to show you my final solution to the above problem but I don't want you to just jump in and type this up. I want *you* to take the rough skeleton code I did and try to make it work based on the description. Once you have your solution then you can come back and see how I did it.

I'm going to break this final file `ex43.py` down into sections and explain each one rather than dump all the code at once.

```
from sys import exit
from random import randint
```

This is just our basic imports for the game, nothing fancy really.

```
class Scene(object):

    def enter(self):
        print "This scene is not yet configured. Subclass it and implement enter().
        exit(1)
```

As you saw in the skeleton code, I have a base class for `Scene` that will have the common things that all scenes do. In this simple program they don't do much so this is more a demonstration of what you would do to make a base class.

```
class Engine(object):

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

    def play(self):
        current_scene = self.scene_map.opening_scene()
        last_scene = self.scene_map.next_scene('finished')

        while current_scene != last_scene:
            next_scene_name = current_scene.enter()
            current_scene = self.scene_map.next_scene(next_scene_name)

        # be sure to print out the last scene
        current_scene.enter()
```

I also have my `Engine` class and you can see how I'm already using the methods for `Map.opening_scene` and `Map.next_scene`. Because I've done a bit of planning I can just assume I'll write those and then use them before I've written the `Map` class.

```
class Death(Scene):

    quips = [
        "You died.  You kinda suck at this.",
         "Your mom would be proud...if she were smarter.",
         "Such a luser.",
         "I have a small puppy that's better at this."
    ]

    def enter(self):
        print Death.quips[randint(0, len(self.quips)-1)]
        exit(1)
```

My first scene is the odd scene named `Death`, which shows you the simplest kind of scene you can write.

```
class CentralCorridor(Scene):

    def enter(self):
        print "The Gothons of Planet Percal #25 have invaded your ship and destroy
        print "your entire crew.  You are the last surviving member and your last"
        print "mission is to get the neutron destruct bomb from the Weapons Armory
        print "put it in the bridge, and blow the ship up after getting into an "
        print "escape pod."
        print "\n"
        print "You're running down the central corridor to the Weapons Armory when
        print "a Gothon jumps out, red scaly skin, dark grimy teeth, and evil clow
ostume"
        print "flowing around his hate filled body.  He's blocking the door to the
        print "Armory and about to pull a weapon to blast you."

        action = raw_input("> ")

        if action == "shoot!":
            print "Quick on the draw you yank out your blaster and fire it at the
hon."
            print "His clown costume is flowing and moving around his body, which
ows"
            print "off your aim.  Your laser hits his costume but misses him entir
.  This"
            print "completely ruins his brand new costume his mother bought him, w
h"
            print "makes him fly into an insane rage and blast you repeatedly in t
face until"
            print "you are dead.  Then he eats you."
            return 'death'

        elif action == "dodge!":
            print "Like a world class boxer you dodge, weave, slip and slide right
            print "as the Gothon's blaster cranks a laser past your head."
            print "In the middle of your artful dodge your foot slips and you"
            print "bang your head on the metal wall and pass out."
            print "You wake up shortly after only to die as the Gothon stomps on"
            print "your head and eats you."
            return 'death'

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

        else:
            print "DOES NOT COMPUTE!"
            return 'central_corridor'
```

After that I've created the `CentralCorridor`, which is the start of the game. I'm doing the scenes for the game before the `Map` because I need to reference them later.

```
class LaserWeaponArmory(Scene):

    def enter(self):
        print "You do a dive roll into the Weapon Armory, crouch and scan the roo
        print "for more Gothons that might be hiding.  It's dead quiet, too quiet
```

```
        print "You stand up and run to the far side of the room and find the"
        print "neutron bomb in its container.  There's a keypad lock on the box"
        print "and you need the code to get the bomb out.  If you get the code"
        print "wrong 10 times then the lock closes forever and you can't"
        print "get the bomb.  The code is 3 digits."
        code = "%d%d%d" % (randint(1,9), randint(1,9), randint(1,9))
        guess = raw_input("[keypad]> ")
        guesses = 0

        while guess != code and guesses < 10:
            print "BZZZZEDDD!"
            guesses += 1
            guess = raw_input("[keypad]> ")

        if guess == code:
            print "The container clicks open and the seal breaks, letting gas out
            print "You grab the neutron bomb and run as fast as you can to the"
            print "bridge where you must place it in the right spot."
            return 'the_bridge'
        else:
            print "The lock buzzes one last time and then you hear a sickening"
            print "melting sound as the mechanism is fused together."
            print "You decide to sit there, and finally the Gothons blow up the"
            print "ship from their ship and you die."
            return 'death'



class TheBridge(Scene):

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

        action = raw_input("> ")

        if action == "throw the bomb":
            print "In a panic you throw the bomb at the group of Gothons"
            print "and make a leap for the door.  Right as you drop it a"
            print "Gothon shoots you right in the back killing you."
            print "As you die you see another Gothon frantically try to disarm"
            print "the bomb. You die knowing they will probably blow up when"
            print "it goes off."
            return 'death'

        elif action == "slowly place the bomb":
            print "You point your blaster at the bomb under your arm"
            print "and the Gothons put their hands up and start to sweat."
            print "You inch backward to the door, open it, and then carefully"
            print "place the bomb on the floor, pointing your blaster at it."
            print "You then jump back through the door, punch the close button"
            print "and blast the lock so the Gothons can't get out."
            print "Now that the bomb is placed you run to the escape pod to"
            print "get off this tin can."
            return 'escape_pod'
        else:
            print "DOES NOT COMPUTE!"
            return "the_bridge"


class EscapePod(Scene):

    def enter(self):
        print "You rush through the ship desperately trying to make it to"
        print "the escape pod before the whole ship explodes.  It seems like"
        print "hardly any Gothons are on the ship, so your run is clear of"
        print "interference.  You get to the chamber with the escape pods, and"
        print "now need to pick one to take.  Some of them could be damaged"
```

```
        print "but you don't have time to look.  There's 5 pods, which one"
        print "do you take?"

        good_pod = randint(1,5)
        guess = raw_input("[pod #]> ")


        if int(guess) != good_pod:
            print "You jump into pod %s and hit the eject button." % guess
            print "The pod escapes out into the void of space, then"
            print "implodes as the hull ruptures, crushing your body"
            print "into jam jelly."
            return 'death'
        else:
            print "You jump into pod %s and hit the eject button." % guess
            print "The pod easily slides out into space heading to"
            print "the planet below.  As it flies to the planet, you look"
            print "back and see your ship implode then explode like a"
            print "bright star, taking out the Gothon ship at the same"
            print "time.  You won!"

            return 'finished'

class Finished(Scene):

    def enter(self):
        print "You won! Good job."
        return 'finished'
```

This is the rest of the game's scenes, and since I know I need them and have thought about how they'll flow together I'm able to code them up directly.

Incidentally, I wouldn't just type all this code in. Remember I said to try and build this incrementally, one little bit at a time. I'm just showing you the final result.

```
class Map(object):

    scenes = {
        'central_corridor': CentralCorridor(),
        'laser_weapon_armory': LaserWeaponArmory(),
        'the_bridge': TheBridge(),
        'escape_pod': EscapePod(),
        'death': Death(),
        'finished': Finished(),
    }

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

    def next_scene(self, scene_name):
        val = Map.scenes.get(scene_name)
        return val

    def opening_scene(self):
        return self.next_scene(self.start_scene)
```

After that I have my `Map` class, and you can see it is storing each scene by name in a dictionary, and then I refer to that dict with `Map.scenes`. This is also why the map comes after the scenes because the dictionary has to refer to the scenes so they have to exist.

```
a_map = Map('central_corridor')
a_game = Engine(a_map)
a_game.play()
```

Finally I've got my code that runs the game by making a `Map` then handing that map to an `Engine` before calling `play` to make the game work.

What You Should See

Make sure you understand the game and that you tried to solve it yourself first. One thing to do is if you're stumped, is cheat a little by reading my code, then continue trying to solve it yourself.

When I run my game it looks like this:

```
$ python ex43.py
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.
> dodge!
Like a world class boxer you dodge, weave, slip and slide right
as the Gothon's blaster cranks a laser past your head.
In the middle of your artful dodge your foot slips and you
bang your head on the metal wall and pass out.
You wake up shortly after only to die as the Gothon stomps on
your head and eats you.
Your mom would be proud...if she were smarter.
```

Study Drills

1. Change it! Maybe you hate this game. Could be to violent, you aren't into sci-fi. Get the game working, then change it to what you like. This is your computer, you make it do what you want.
2. I have a bug in this code. Why is the door lock guessing 11 times?
3. Explain how returning the next room works.
4. Add cheat codes to the game so you can get past the more difficult rooms. I can do this with two words on one line.
5. Go back to my description and analysis, then try to build a small combat system for the hero and the various Gothons he encounters.
6. This is actually a small version of something called a "finite state machine." Read about them. They might not make sense but try anyway.

Common Student Questions

Q: Where can I find stories for my own games?

> You can make them up, just like you would tell a story to a friend. Or you can take simple scenes from a book or movie you like.

Video