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 and that might not 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: I make a list of all the *nouns* and *verbs* in my 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 needed.

Once I have my list of concepts I create a simple outline/tree of the concepts and how they are related as classes. I can usually take my 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?" I keep doing this until I have a class hierarchy that's just a simple tree list or a diagram. Then I take the *verbs* I have and see if those are function names for each class and put them in my 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, and other times I might write a little test, a little code, a little test, and so on 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 it 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 that the players 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 This is another battle scene with a Gothon where the hero places the bomb.

Escape Pod This is 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 any of them might be good function names, but I'll skip that for now.

At this point I might also research each of these concepts and anything I 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 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.

ex43_classes.py

```
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 added 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 versus 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 research 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 preceding 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.

ex43.py

```
from sys import exit
from random import randint
from textwrap import dedent
```

This is just our basic imports for the game. The only new thing is the import of the `dedent` function from the `textwrap` module. This little function will help us write our room descriptions using `"""` (triple-quote) strings. It simply strips leading white-space from the beginnings of lines in a string. Without this function, using `"""` style strings fails because they are indented on the *screen* the same level as in the *Python* code.

ex43.py

```
class Scene(object):

    def enter(self):
        print("This scene is not yet configured.")
        print("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.

ex43.py

```
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.

ex43.py

```
1   class Death(Scene):
2
3       quips = [
4           "You died.  You kinda suck at this.",
5           "Your Mom would be proud...if she were smarter.",
6           "Such a luser.",
7           "I have a small puppy that's better at this.",
8           "You're worse than your Dad's jokes."
9
10      ]
11
12      def enter(self):
13          print(Death.quips[randint(0, len(self.quips)-1)])
14          exit(1)
```

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

ex43.py

```
1   class CentralCorridor(Scene):
2
3       def enter(self):
4           print(dedent("""
5                The Gothons of Planet Percal #25 have invaded your ship and
6                destroyed your entire crew.  You are the last surviving
7                member and your last mission is to get the neutron destruct
8                bomb from the Weapons Armory, put it in the bridge, and
9                blow the ship up after getting into an escape pod.
10
11               You're running down the central corridor to the Weapons
12               Armory when a Gothon jumps out, red scaly skin, dark grimy
13               teeth, and evil clown costume flowing around his hate
14               filled body.  He's blocking the door to the Armory and
15               about to pull a weapon to blast you.
16               """))
17
18          action = input("> ")
19
20          if action == "shoot!":
21              print(dedent("""
22                    Quick on the draw you yank out your blaster and fire
23                    it at the Gothon.  His clown costume is flowing and
24                    moving around his body, which throws off your aim.
25                    Your laser hits his costume but misses him entirely.
26                    This completely ruins his brand new costume his mother
```

```
27                  bought him, which makes him fly into an insane rage
28                  and blast you repeatedly in the face until you are
29                  dead.  Then he eats you.
30                  """))
31              return 'death'
32
33          elif action == "dodge!":
34              print(dedent("""
35                  Like a world class boxer you dodge, weave, slip and
36                  slide right as the Gothon's blaster cranks a laser
37                  past your head.  In the middle of your artful dodge
38                  your foot slips and you bang your head on the metal
39                  wall and pass out.  You wake up shortly after only to
40                  die as the Gothon stomps on your head and eats you.
41                  """))
42              return 'death'
43
44          elif action == "tell a joke":
45              print(dedent("""
46                  Lucky for you they made you learn Gothon insults in
47                  the academy.  You tell the one Gothon joke you know:
48                  Lbhe zbgure vf fb sng, jura fur fvgf nebhaq gur ubhfr,
49                  fur fvgf nebhaq gur ubhfr.  The Gothon stops, tries
50                  not to laugh, then busts out laughing and can't move.
51                  While he's laughing you run up and shoot him square in
52                  the head putting him down, then jump through the
53                  Weapon Armory door.
54                  """))
55              return 'laser_weapon_armory'
56
57          else:
58              print("DOES NOT COMPUTE!")
59              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. You should also see how I use the `dedent` function on line 4. Try removing it later to see what it does.

ex43.py

```
1   class LaserWeaponArmory(Scene):
2
3       def enter(self):
4           print(dedent("""
5                 You do a dive roll into the Weapon Armory, crouch and scan
6                 the room for more Gothons that might be hiding.  It's dead
7                 quiet, too quiet.  You stand up and run to the far side of
8                 the room and find the neutron bomb in its container.
9                 There's a keypad lock on the box and you need the code to
10                get the bomb out.  If you get the code wrong 10 times then
11                the lock closes forever and you can't get the bomb.  The
```

```
12                  code is 3 digits.
13                  """))
14
15          code = f"{randint(1,9)}{randint(1,9)}{randint(1,9)}"
16          guess = input("[keypad]> ")
17          guesses = 0
18
19          while guess != code and guesses < 10:
20              print("BZZZZEDDD!")
21              guesses += 1
22              guess = input("[keypad]> ")
23
24          if guess == code:
25              print(dedent("""
26                    The container clicks open and the seal breaks, letting
27                    gas out.  You grab the neutron bomb and run as fast as
28                    you can to the bridge where you must place it in the
29                    right spot.
30                    """))
31              return 'the_bridge'
32          else:
33              print(dedent("""
34                    The lock buzzes one last time and then you hear a
35                    sickening melting sound as the mechanism is fused
36                    together.  You decide to sit there, and finally the
37                    Gothons blow up the ship from their ship and you die.
38                    """))
39              return 'death'
40
41
42
43  class TheBridge(Scene):
44
45      def enter(self):
46          print(dedent("""
47                You burst onto the Bridge with the netron destruct bomb
48                under your arm and surprise 5 Gothons who are trying to
49                take control of the ship.  Each of them has an even uglier
50                clown costume than the last.  They haven't pulled their
51                weapons out yet, as they see the active bomb under your
52                arm and don't want to set it off.
53                """))
54
55          action = input("> ")
56
57          if action == "throw the bomb":
58              print(dedent("""
59                    In a panic you throw the bomb at the group of Gothons
60                    and make a leap for the door.  Right as you drop it a
61                    Gothon shoots you right in the back killing you.  As
```

```
62                  you die you see another Gothon frantically try to
63                  disarm the bomb. You die knowing they will probably
64                  blow up when it goes off.
65                  """))
66              return 'death'
67
68          elif action == "slowly place the bomb":
69              print(dedent("""
70                  You point your blaster at the bomb under your arm and
71                  the Gothons put their hands up and start to sweat.
72                  You inch backward to the door, open it, and then
73                  carefully place the bomb on the floor, pointing your
74                  blaster at it.  You then jump back through the door,
75                  punch the close button and blast the lock so the
76                  Gothons can't get out.  Now that the bomb is placed
77                  you run to the escape pod to get off this tin can.
78                  """))
79
80              return 'escape_pod'
81          else:
82              print("DOES NOT COMPUTE!")
83              return "the_bridge"
84
85
86  class EscapePod(Scene):
87
88      def enter(self):
89          print(dedent("""
90                You rush through the ship desperately trying to make it to
91                the escape pod before the whole ship explodes.  It seems
92                like hardly any Gothons are on the ship, so your run is
93                clear of interference.  You get to the chamber with the
94                escape pods, and now need to pick one to take.  Some of
95                them could be damaged but you don't have time to look.
96                There's 5 pods, which one do you take?
97                """))
98
99          good_pod = randint(1,5)
100         guess = input("[pod #]> ")
101
102
103         if int(guess) != good_pod:
104             print(dedent("""
105                 You jump into pod {guess} and hit the eject button.
106                 The pod escapes out into the void of space, then
107                 implodes as the hull ruptures, crushing your body into
108                 jam jelly.
109                 """))
110             return 'death'
111         else:
```

```
            print(dedent("""
                  You jump into pod {guess} 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!
                  """))

            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.

ex43.py

```
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.

ex43.py

```
1    a_map = Map('central_corridor')
2    a_game = Engine(a_map)
3    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 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:

Exercise 43 Session

```
$ python3.6 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.

You're worse than your Dad's jokes.
```

Study Drills

1. Change it! Maybe you hate this game. It could be too violent, or maybe 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

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.