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.

First 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 he 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 out 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 go research how ships are designed or how bombs work. Maybe I'll go 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 I can say "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. And 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 also then go through and figure out what actions are needed on each thing based on verbs in the description. For example, I know that from the above description I'm going to need a way to "run" the engine, "get the next scene" from the map, get the "openining 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 above 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
10      def __init__(self, scene_map):
11          pass
12
13      def play(self):
14          pass
15
16  class Death(Scene):
17
18      def enter(self):
19          pass
20
21  class CentralCorridor(Scene):
22
23      def enter(self):
24          pass
25
26  class LaserWeaponArmory(Scene):
27
28      def enter(self):
29          pass
30
31  class TheBridge(Scene):
32
33      def enter(self):
34          pass
35
36  class EscapePod(Scene):
37
38      def enter(self):
39          pass
40
41
42  class Map(object):
43
44      def __init__(self, start_scene):
45          pass
46
47      def next_scene(self, scene_name):
48          pass
49
50      def opening_scene(self):
51          pass
52
53
54  a_map = Map('central_corridor')
55  a_game = Engine(a_map)
    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 I just described 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 up 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.

This process I find 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 above and then 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.

```
1   from sys import exit
2   from random import randint
```

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

```
1   class Scene(object):
2
3       def enter(self):
4           print "This scene is not yet configured. Subclass
    it and implement enter()."
5           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.

```
1    class Engine(object):
2
3        def __init__(self, scene_map):
4            self.scene_map = scene_map
5
6        def play(self):
7            current_scene = self.scene_map.opening_scene()
8
9            while True:
10               print "\n--------"
11               next_scene_name = current_scene.enter()
12               current_scene =
     self.scene_map.next_scene(next_scene_name)
```

I also have my `Engine` class and you can see how I'm already just 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.

```
1   class Death(Scene):
2
3       quips = [
4           "You died.  You kinda suck at this.",
5            "Your mom would be proud...if she were
6   smarter.",
7            "Such a luser.",
8            "I have a small puppy that's better at this."
9       ]
10
11      def enter(self):
12          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.

```python
class CentralCorridor(Scene):

    def enter(self):
        print "The Gothons of Planet Percal #25 have invaded your ship and destroyed"
        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 clown costume"
        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 Gothon."
            print "His clown costume is flowing and moving around his body, which throws"
            print "off your aim.  Your laser hits his costume but misses him entirely.  This"
            print "completely ruins his brand new costume his mother bought him, which"
            print "makes him fly into an insane rage and blast you repeatedly in the 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"
```

```
38                print "your head and eats you."
39                return 'death'
40
41          elif action == "tell a joke":
42                print "Lucky for you they made you learn
43   Gothon insults in the academy."
44                print "You tell the one Gothon joke you
45   know:"
46                print "Lbhe zbgure vf fb sng, jura fur fvgf
     nebhaq gur ubhfr, fur fvgf nebhaq gur ubhfr."
                  print "The Gothon stops, tries not to laugh,
     then busts out laughing and 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 room"
            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:
1                   print "BZZZZEDDD!"
2                   guesses += 1
3                   guess = raw_input("[keypad]> ")
```

```
 4
 5            if guess == code:
 6                print "The container clicks open and the
 7     seal breaks, letting gas out."
 8                print "You grab the neutron bomb and run as
 9     fast as you can to the"
10                print "bridge where you must place it in the
11     right spot."
12                return 'the_bridge'
13            else:
14                print "The lock buzzes one last time and
15     then you hear a sickening"
16                print "melting sound as the mechanism is
17     fused together."
18                print "You decide to sit there, and finally
19     the Gothons blow up the"
20                print "ship from their ship and you die."
21                return 'death'
22
23
24
25    class TheBridge(Scene):
26
27        def enter(self):
28            print "You burst onto the Bridge with the netron
29     destruct bomb"
30            print "under your arm and surprise 5 Gothons who
31     are trying to"
32            print "take control of the ship.  Each of them
33     has an even uglier"
34            print "clown costume than the last.  They
35     haven't pulled their"
36            print "weapons out yet, as they see the active
37     bomb under your"
38            print "arm and don't want to set it off."
39
40            action = raw_input("> ")
41
42            if action == "throw the bomb":
43                print "In a panic you throw the bomb at the
44     group of Gothons"
45                print "and make a leap for the door.  Right
46     as you drop it a"
47                print "Gothon shoots you right in the back
48     killing you."
49                print "As you die you see another Gothon
50     frantically try to disarm"
51                print "the bomb. You die knowing they will
52     probably blow up when"
53                print "it goes off."
54                return 'death'
55
56            elif action == "slowly place the bomb":
57                print "You point your blaster at the bomb
58     under your arm"
```

```
59              print "and the Gothons put their hands up
60   and start to sweat."
61              print "You inch backward to the door, open
62   it, and then carefully"
63              print "place the bomb on the floor, pointing
64   your blaster at it."
65              print "You then jump back through the door,
66   punch the close button"
67              print "and blast the lock so the Gothons
68   can't get out."
69              print "Now that the bomb is placed you run
70   to the escape pod to"
71              print "get off this tin can."
72              return 'escape_pod'
73          else:
74              print "DOES NOT COMPUTE!"
75              return "the_bridge"
76
77
78   class EscapePod(Scene):
79
80       def enter(self):
81           print "You rush through the ship desperately
82   trying to make it to"
83           print "the escape pod before the whole ship
84   explodes.  It seems like"
85           print "hardly any Gothons are on the ship, so
86   your run is clear of"
87           print "interference.  You get to the chamber
88   with the escape pods, and"
89           print "now need to pick one to take.  Some of
90   them could be damaged"
91           print "but you don't have time to look.  There's
92   5 pods, which one"
93           print "do you take?"
94
95           good_pod = randint(1,5)
96           guess = raw_input("[pod #]> ")
97
98
99           if int(guess) != good_pod:
100              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'
```

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.

```
 1  class Map(object):
 2
 3      scenes = {
 4          'central_corridor': CentralCorridor(),
 5          'laser_weapon_armory': LaserWeaponArmory(),
 6          'the_bridge': TheBridge(),
 7          'escape_pod': EscapePod(),
 8          'death': Death()
 9      }
10
11      def __init__(self, start_scene):
12          self.start_scene = start_scene
13
14      def next_scene(self, scene_name):
15          return Map.scenes.get(scene_name)
16
17      def opening_scene(self):
18          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 them so they have to exist.

```
 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 is if you're stumped, just go cheat a little bit. Take a look real quick in the book, then get your "Aha!" realization from my code, and go back to working on yours. Just try as hard as you can to solve it yourself first.

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. I have a bug in this code. Why is the door lock guessing 11 times?
2. Explain how returning the next room works.
3. 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.
4. Go back to my description and analysis, then try to build a small combat system for the hero and the various Gothons he encounters.
5. 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 also take simple scenes from a book or movie you like.

Purchase The Videos For $29.59

For just $29.59 you can get access to all the videos for Learn Python The Hard Way, **plus** a PDF of the book and no more popups all in this one location. For $29.59 you get:

- 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!
Buying Is Easy

Buying is easy. Just fill out the form below and we'll get started.

Full Name

Email Address

Pay With Credit Card (by Stripe™)

PayPal Use your PayPal™ account.

Buy Learn Python The Hard Way, 3rd Edition