Harry Netzer's Comps Project 3

The ?purpose of this project was to further explore how functions, i.e. bits of code that assign complex algorithms to single words for later use, can reduce your labor as a programmer. In this project we put functions inside of functions inside of functions, successively streamlining my program to the point where I could draw an underwater scene of any size, in any place, with a single line of code.

This project was also meant to acquaint us with for loops, which can serve to automate the execution of a single or similar commands any number of times. Unfortunately my pictures didn't lend themselves to for loops because most of what I was drawing was pretty arbitrary and without much pattern. I did manage to shove a for loop into one place, where in drawing the mojave desert i had to draw six mountains in a row.

This week I also used if-then statements. If statements are helpful if you dont want a section of code to execute unless a condition is fulfilled. In my program I wanted the shapes to fill in only if the function parameter "fill" has the boolean value true. I used the following code:

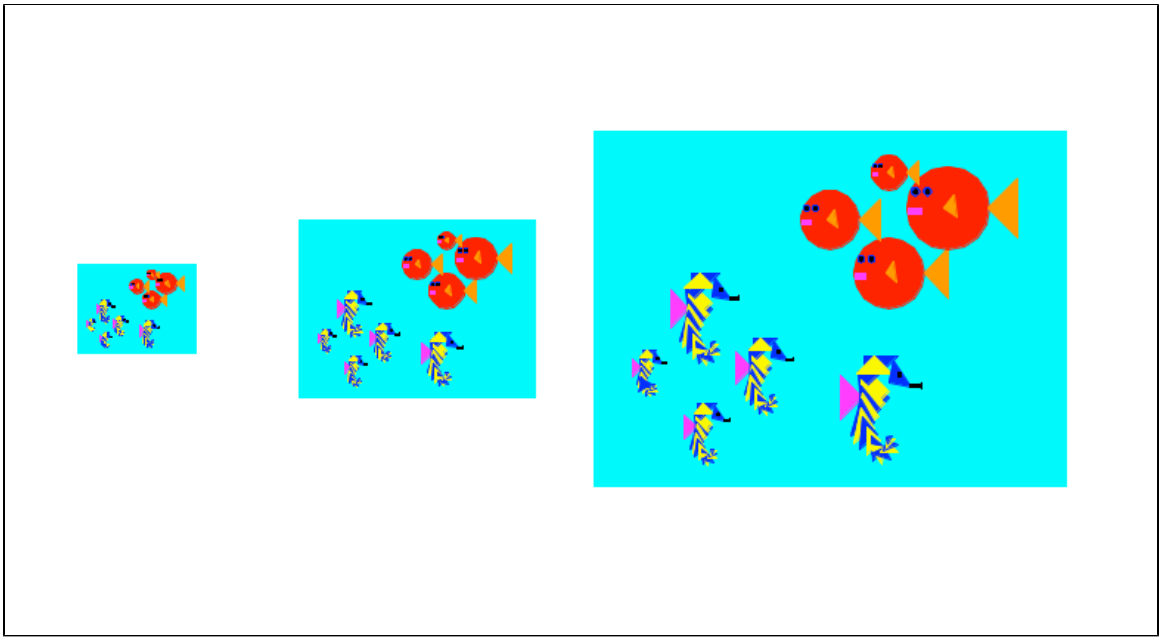
```
if fill == True:
 turtle.fill(True)if fill == True:

 turtle.fill(True)
```

I don't need a then statement because the implied then in this case is to do nothing.

My first task was to make my underwater scene moveable and scaleable. I would need to modify my fishTank() function to take three new input parameters: x, y, and scale. The function fishTank() already called several other functions, namely fish() and seaHoss(), which take parameters for x position, y position, and scale. While before I gave the fish and seaHorse functions explicit values, now I gave them variables. Let me illustrate with two code snippets, first from the static fishTank and then from the improved, movable and scaleable fishTank:

```
def fishTank():

 fish(0, 0, .6 )

 fish(100, 100, .7)

 fish(-100, 100, .5)

 seaHoss(-200, -100, .5)

def fishTank(x, y, scale):

 fish(x, y, .6 * scale)

 fish(x + 100 * scale, y + 100 * scale, .7 * scale)

 fish(x - 100 * scale, y + 100 * scale, .5 * scale)

 seaHoss(x - 200 * scale, y - 100 * scale, .5 * scale)
```

As you can see, the new fishTank takes parameters, which go inside the code where fishTank calls other shape functions. Now I can make fishTanks of any size and position I want, see:

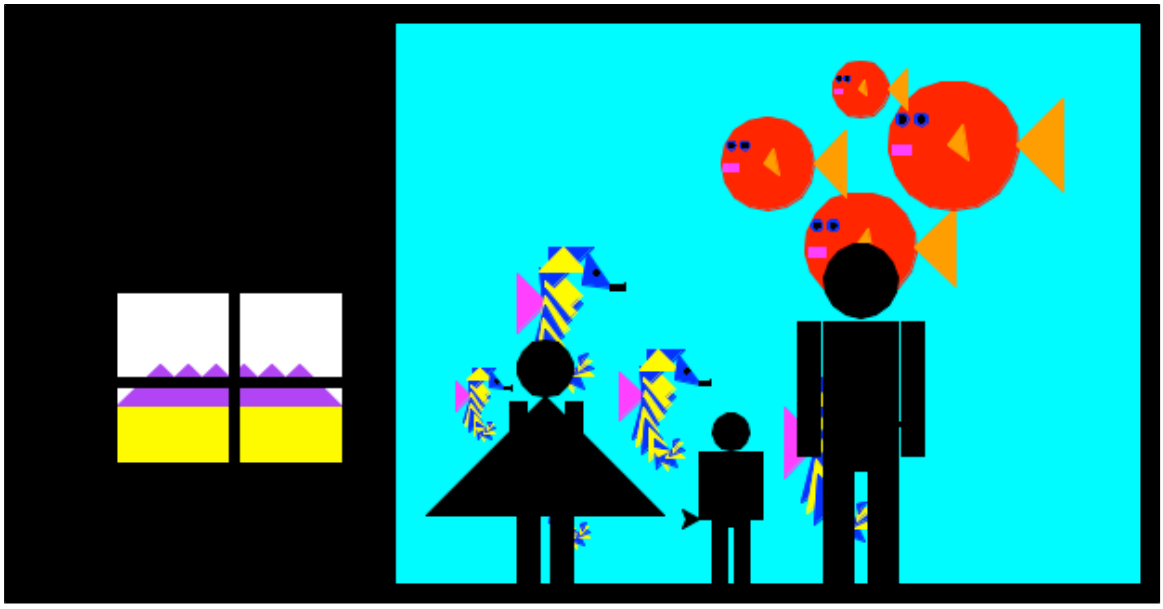


I should note that I didn't call the fishTank function three times inside the shapes.py file where I defined fishTank. Instead I did so in a different file which imported the code from shapes.py. But when python imports a file it does so by executing it, and I had some test code at the bottom of shapes.py which I didn't want executed. To avoid this I inserted a special if statement:

```
def main():
 woman(0, 0, 1)
 raw_input("Press enter when ready. ")
if _name_ == "_main_":
 main()
```

main() is my top level code, and the if statement makes sure main() only gets executed when the file is being executed on its own.

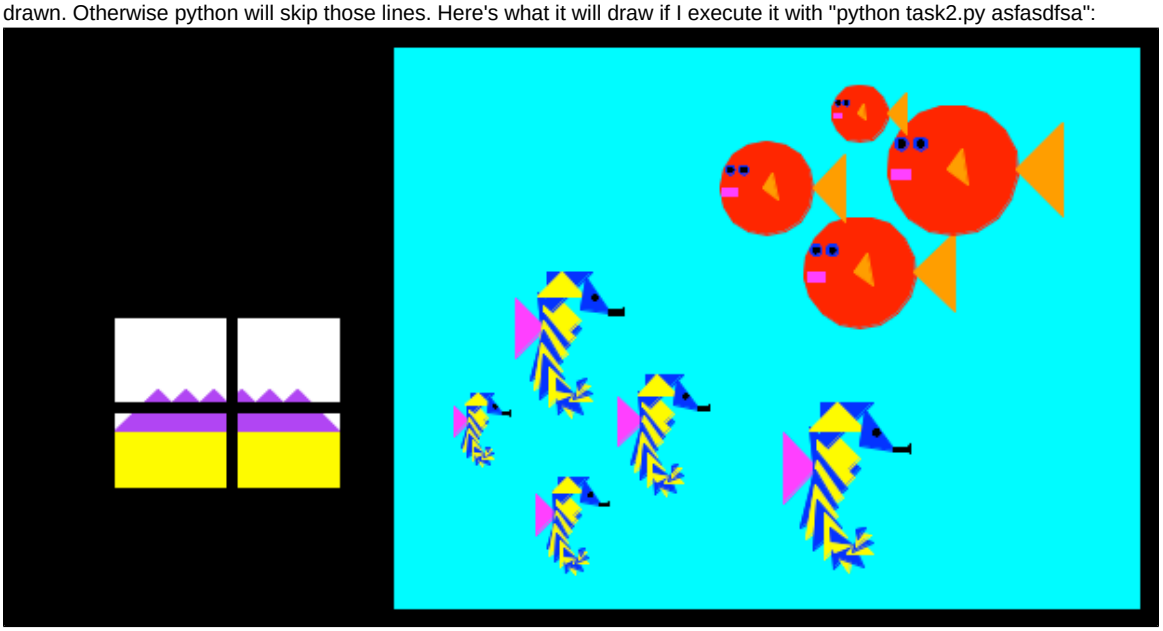
Next I made an public aquarium scene with a fish tank and a few other shapes, some people, and the mojave desert outside. My aquarium function contained only calls to moveable, scaleable shapes from my shapes.py file, that is the mojave, 2 men, a woman, a fishtank, and some

blocks. picture: 

Really makes you think about the improbability of life. The last thing I did was program in the ability for the user toggle whether the people show or not. I did this with a library called sys which reads the user's input from terminal and inserts every word after "python" into a list. Then I can read the entries from that list with a command like sys.argv[x], where x is the number of words after python I want it to read from. To toggle the people in that scene I inserted the following code into my aquarium function:

```
if sys.argv[1] == "people":
  things.man(120,-50,.4)
  things.woman(-50,-80,.3)
  things.man(50, -100, .2)
```

With this code python will check to see if the user typed "people" after "python task2.py" to execute the file. If he or she did, the people will be drawn. Otherwise python will skip those lines. Here's what it will draw if I execute it with "python task2.py asfasdfsa":



No people.

In this project I learned the importance of precision and consistency when defining shapes. It was a constant challenge to make my shapes draw in the right place, which could have been avoided if every shape took its x and y position parameters for the location of, say, its bottom left corner. As I did it, the x and y parameters could be anywhere in the shape. A simply goto statement at the beginning of each shape would have helped, but its too late to change that now.

The sys library seems like a powerful feature and I look forward to programming with more user input in the future.

I did this project entirely on my own.