Chapter_4_Case_study_interface_design

March 6, 2024

This chapter presents a case study that demonstrates a process for designing functions that work together. It introduces the turtle module, which allows you to create images using turtle graphics. The turtle module is included in most Python installations, but if you are running Python using PythonAnywhere, you won't be able to run the turtle examples (at least you couldn't when I wrote this).

0.1 4.1 The turtle module

[1]:
```
import turtle
bob = turtle.Turtle()
```

[2]:
```
print(bob)
```

```
<turtle.Turtle object at 0x00000186162554C0>
```

When you run this code, it should create a new window with small arrow that represents the turtle. Close the window

Create a file named mypolygon.py and type in the following code:

The turtle module (with a lowercase 't') provides a function called Turtle (with an uppercase 'T') that creates a Turtle object, which we assign to a variable named bob. Printing bob displays something like: <turtle.Turtle object at 0xb7bfbf4c>

This means that bob refers to an object with type Turtle as defined in module turtle. mainloop tells the window to wait for the user to do something, although in this case there's not much for the user to do except close the window.

Once you create a Turtle, you can call a method to move it around the window. A method is similar to a function, but it uses slightly different syntax. For example, to move the turtle forward:

bob.fd(100) The method, fd, is associated with the turtle object we're calling bob. Calling a method is like making a request: you are asking bob to move forward. The argument of fd is a distance in pixels, so the actual size depends on your display.

Other methods you can call on a Turtle are bk to move backward, lt for left turn, and rt right turn. The argument for lt and rt is an angle in degrees

Also, each Turtle is holding a pen, which is either down or up; if the pen is down, the Turtle leaves a trail when it moves. The methods pu and pd stand for "pen up" and "pen down". To draw a right angle, add these lines to the program (after creating bob and before calling mainloop):

[3]:
```
bob.fd(100)
bob.lt(90)
bob.fd(100)
```

Wrapping a piece of code up in a function is called encapsulation. One of the benefits of encapsulation is that it attaches a name to the code, which serves as a kind of documentation. Another advantage is that if you re-use the code, it is more concise to call a function twice than to copy and paste the body

Adding a parameter to a function is called generalization because it makes the function more general: in the previous version, the square is always the same size; in this version it can be any size.

By convention, import statements are usually at the beginning of the script.

This process—rearranging a program to improve interfaces and facilitate code re-use—is called refactoring

0.2 4.9 docstring

A docstring is a string at the beginning of a function that explains the interface ("doc" is short for "documentation")

[4]:
```
def polyline(t, n, length, angle):
    """Draws n line segments with the given length and
    angle (in degrees) between them. t is a turtle.
    """
    for i in range(n):
        t.fd(length)
        t.lt(angle)
```

By convention, all docstrings are triple-quoted strings, also known as multiline strings because the triple quotes allow the string to span more than one line

[ ]:
```

```