Turtle Graphics with Python

```
$ python
Python 2.7.12 (default, Jun 29 2016, 14:04:44)
[GCC 4.2.1 Compatible Apple LLVM 7.0.2 (clang-700.1.81)] on
darwin
Type "help", "copyright", "credits" or "license" for more in
formation.
>>>
```

Let's do some turtle graphics! First fire up a python shell.

```
$ python
Python 2.7.12 (default, Jun 29 2016, 14:04:44)
[GCC 4.2.1 Compatible Apple LLVM 7.0.2 (clang-700.1.81)] on
darwin
Type "help", "copyright", "credits" or "license" for more in
formation.
>>> from turtle import *
>>>
```

Import all the functions from the built-in turtle module

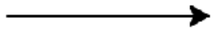

```
$ python
Python 2.7.12 (default, Jun 29 2016, 14:04:44)
[GCC 4.2.1 Compatible Apple LLVM 7.0.2 (clang-700.1.81)] on
darwin
Type "help", "copyright", "credits" or "license" for more in
formation.
>>> from turtle import *
>>> forward(100)
>>>
```

Now you can move the turtle! We'll move it forward by 100 pixels.

```
$ python
Python 2.7.12 (default, Jun 29 2016, 14:04:44)
M 7.0.2 (clang-700.1.81)] on

its" or "license" for more in
```

Python Turtle Graphics



A white window came up behind my terminal...there it is!

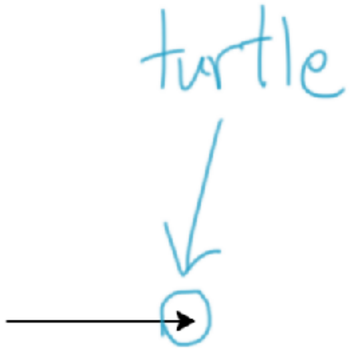
```
$ python
Python 2.7.12 (default, Jun 29 2016, 14:04:44)
[GCC 4.2.1 Compatible Apple LLVM 7.0.2 (clang-700.1.81)] on
darwin
Type "help", "copyright", "credits" or "license" for more in
formation.
>>> from turtle import *
>>> forward(100)
>>>
```

I will tile it next to my terminal.

Terminal Shell Edit View Window Help

Sat 7:45 AM

Python Turtle Graphics



turtle — Python — 60×29

```
$ python
Python 2.7.12 (default, Jun 29 2016, 14:04:44)
[GCC 4.2.1 Compatible Apple LLVM 7.0.2 (clang-700.1.81)] on
darwin
Type "help", "copyright", "credits" or "license" for more in
formation.
>>> from turtle import *
>>> forward(100)
>>>
```

```
$ python
Python 2.7.12 (default, Jun 29 2016, 14:04:44)
[GCC 4.2.1 Compatible Apple LLVM 7.0.2 (clang-700.1.81)] on
darwin
Type "help", "copyright", "credits" or "license" for more in
formation.
>>> from turtle import *
>>> forward(100)
>>> left(90)
>>>
```

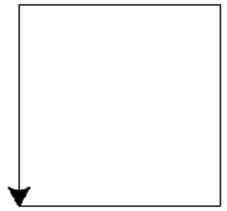
We can turn it to the left, and now it faces north.

```
$ python
Python 2.7.12 (default, Jun 29 2016, 14:04:44)
[GCC 4.2.1 Compatible Apple LLVM 7.0.2 (clang-700.1.81)] on
darwin
Type "help", "copyright", "credits" or "license" for more in
formation.
>>> from turtle import *
>>> forward(100)
>>> left(90)
>>> right(90)
>>>
```

We can turn it to the right, and now it faces east again.

```
$ python
Python 2.7.12 (default, Jun 29 2016, 14:04:44)
[GCC 4.2.1 Compatible Apple LLVM 7.0.2 (clang-700.1.81)] on
darwin
Type "help", "copyright", "credits" or "license" for more in
formation.
>>> from turtle import *
>>> forward(100)
>>> left(90)
>>> right(90)
>>> left(90)
>>>
```

we'll turn to the left again.



```
$ python
Python 2.7.12 (default, Jun 29 2016, 14:04:44)
[GCC 4.2.1 Compatible Apple LLVM 7.0.2 (clang-700.1.81)] on
darwin
Type "help", "copyright", "credits" or "license" for more in
formation.
>>> from turtle import *
>>> forward(100)
>>> left(90)
>>> right(90)
>>> left(90)
>>> forward(100)
>>> left(90)
>>> forward(100)
>>> left(90)
>>> forward(100)
>>>
```

There

```
$ atom draw_a_square.py
```

Now let's rewrite that code to make a square in a .py file.

draw_a_square.py

```
1  from turtle import *
2
3  forward(100)
4  left(90)
5  forward(100)
6  left(90)
7  forward(100)
8  left(90)
9  forward(100)
10
```

That's the code.

draw_a_square.py

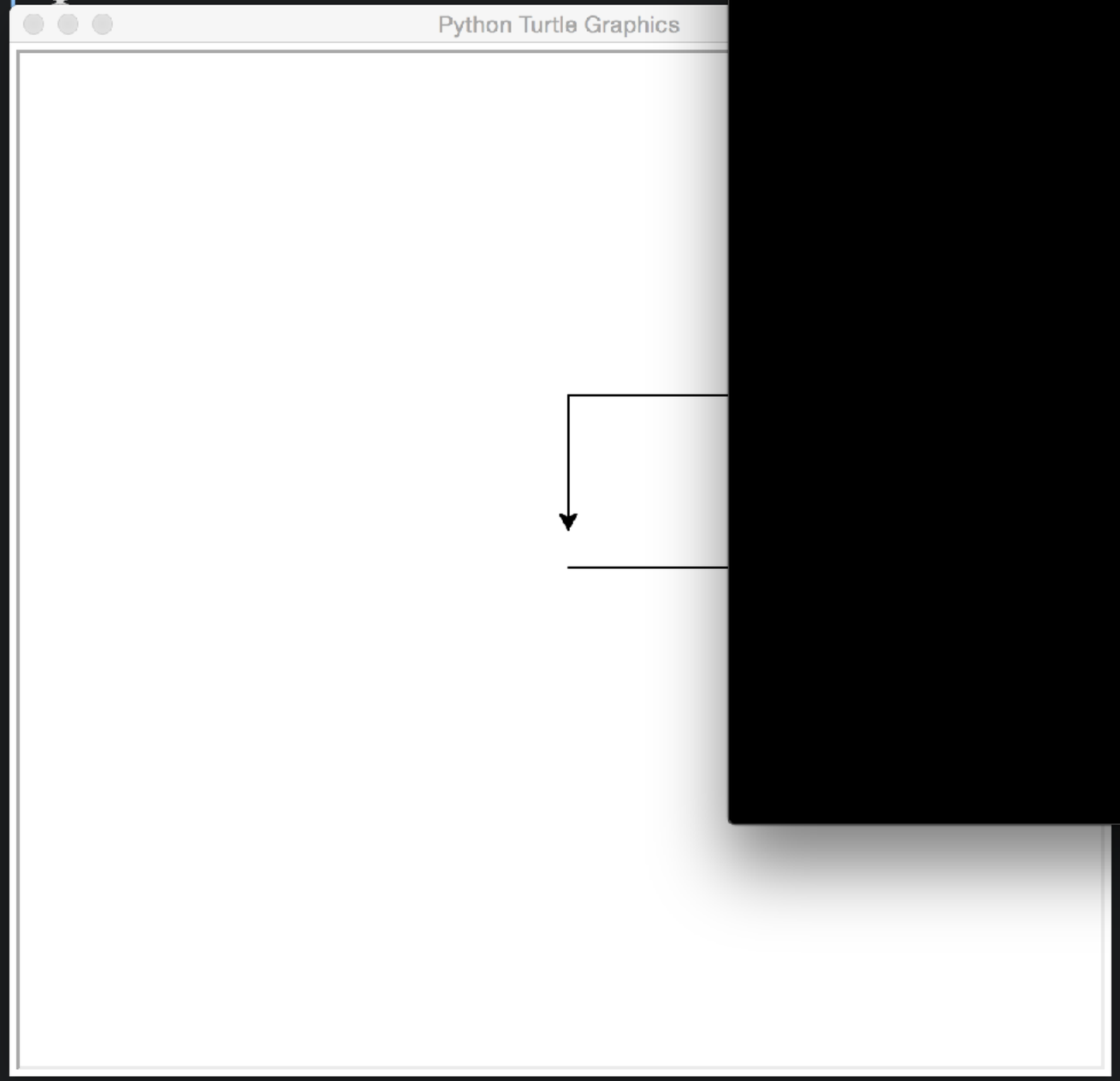
```
1  from turtle import *
2
3  forward(100)
4  left(90)
5  forward(100)
6  left(90)
7  forward(100)
8  left(90)
9  forward(100)
10
```

```
$ python draw_a_square.py
```

Run it!

draw_a_square.py

```
1  from turtle import *
2
3  forward(100)
4  left(90)
5  forward(100)
6  left(90)
7  forward(100)
8  left(90)
9  forward(100)
10
```




```
$ python draw_a_square.py
```

Run it!

draw_a_square.py

```
1  from turtle import *
2
3  forward(100)
4  left(90)
5  forward(100)
6  left(90)
7  forward(100)
8  left(90)
9  forward(100)
10
```

```
$ python draw_a_square.py
$
```

It did run it and draw the square, but the window closed immediately after it was drawn.

draw_a_square.py

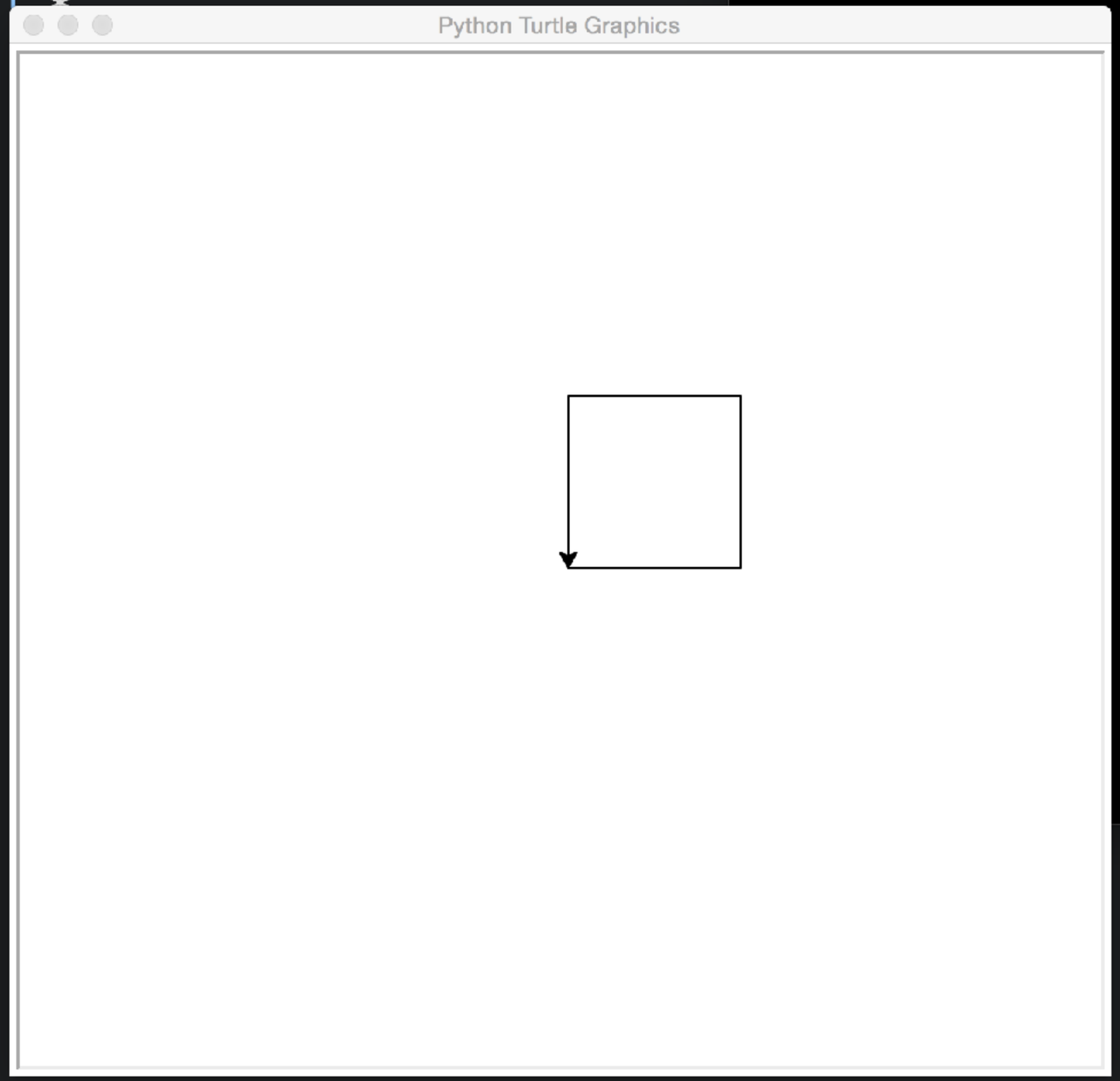
```
1  from turtle import *
2
3  forward(100)
4  left(90)
5  forward(100)
6  left(90)
7  forward(100)
8  left(90)
9  forward(100)
10
11 mainloop()
```

To prevent the window from closing immediately so you can take a better look at the picture, add a call to the mainloop() function at the end of the program.

draw_a_square.py

```
1  from turtle import *
2
3  forward(100)
4  left(90)
5  forward(100)
6  left(90)
7  forward(100)
8  left(90)
9  forward(100)
10
11 mainloop()
12
```

```
$ python draw_a_square.py
```



Now run it and the window stays.

draw_a_square.py

```
1  from turtle import *
2
3  forward(100)
4  left(90)
5  forward(100)
6  left(90)
7  forward(100)
8  left(90)
9  forward(100)
10
11 mainloop()
12
```

Now, we have a lot of code duplication in this code. Can we make this program simpler and less repetitive?

draw_a_square.py

```
1  from turtle import *
2
3  forward(100)
4  left(90)
5  forward(100)
6  left(90)
7  forward(100)
8  left(90)
9  forward(100)
10 left(90)
11
12 mainloop()
13
```

Now, we have a lot of code duplication in this code. Can we make this program simpler and less repetitive?

We are simply moving forward and then turning left 4 times in a row - except not turning left for the final time - although it wouldn't hurt to have turned left the final time anyway.

draw_a_square.py

```
1  from turtle import *
2
3  for i in range(4):
4      forward(100)
5      left(90)
6
7  mainloop()
8
```

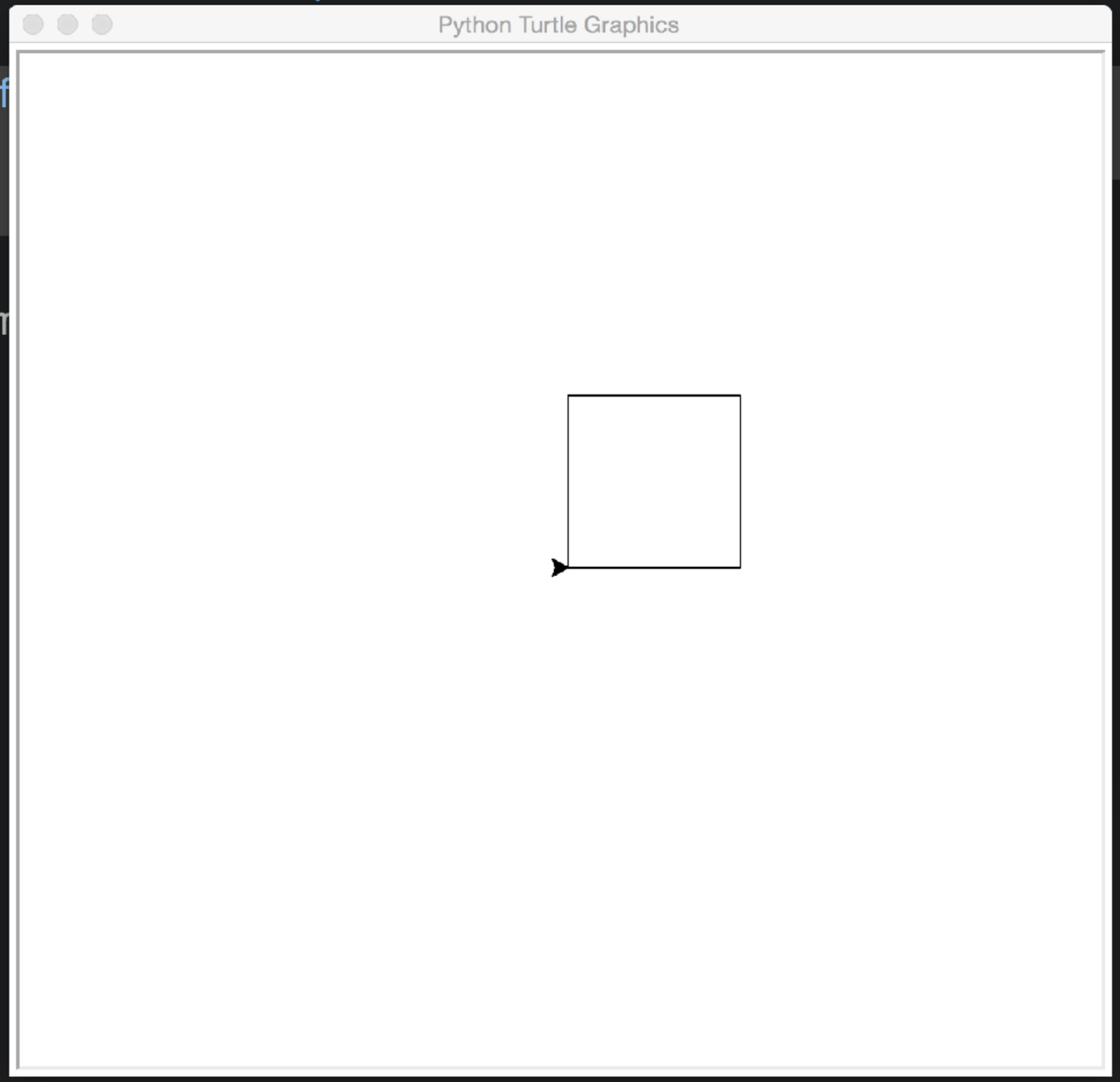
Given this repetition, we can put the statements

forward(100)
left(90)

inside of a loop that executes 4 times, and that would do the equivalent.

draw_a_square.py

```
1  from turtle import *
2
3  f
4
5
6
7  m
8
```

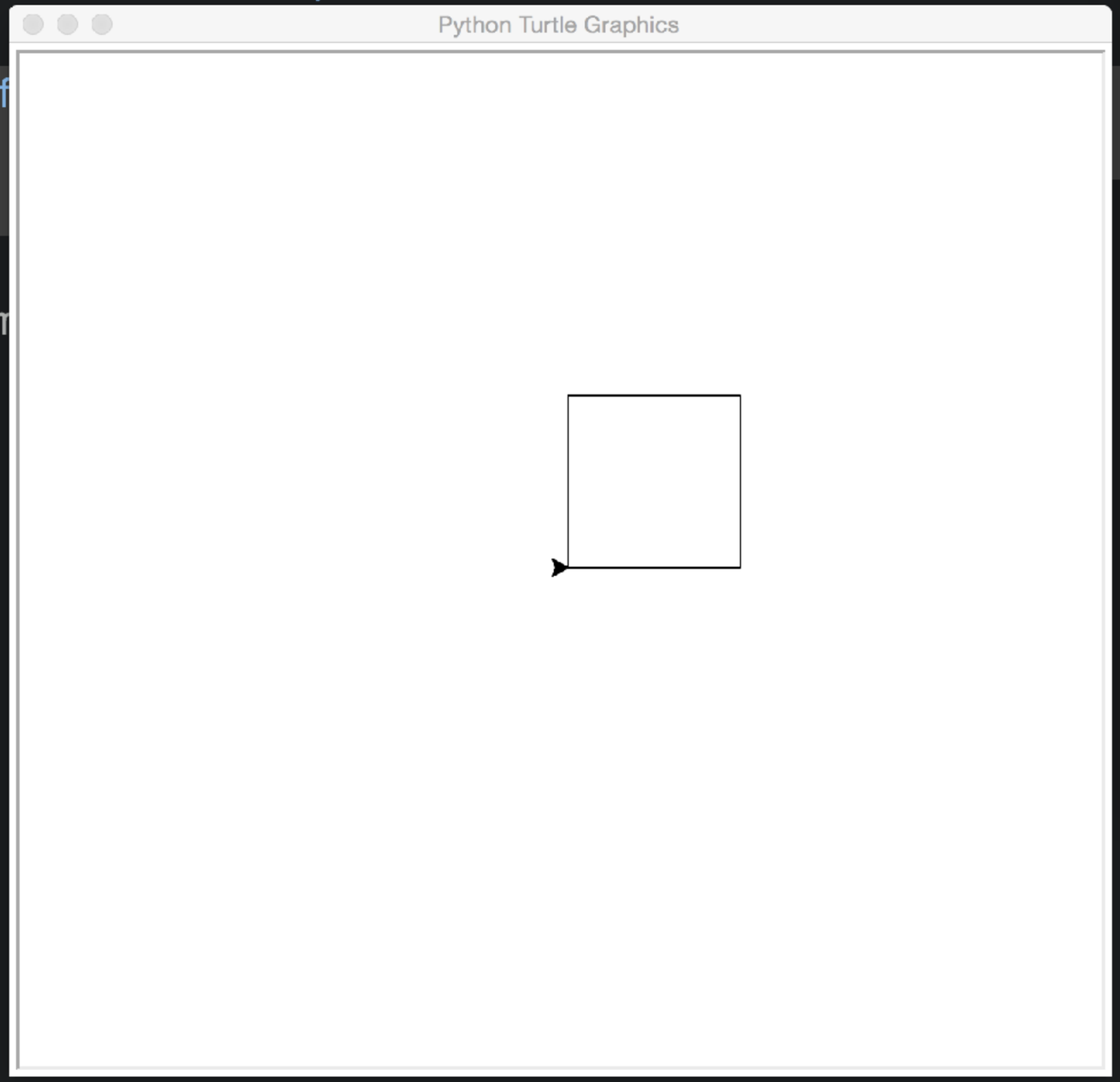



```
$ python draw_a_square.py
```

Now let's re-run it to verify it still works like it used to. It still works!

draw_a_square.py

```
1  from turtle import *
2
3  f
4
5
6
7  m
8
```

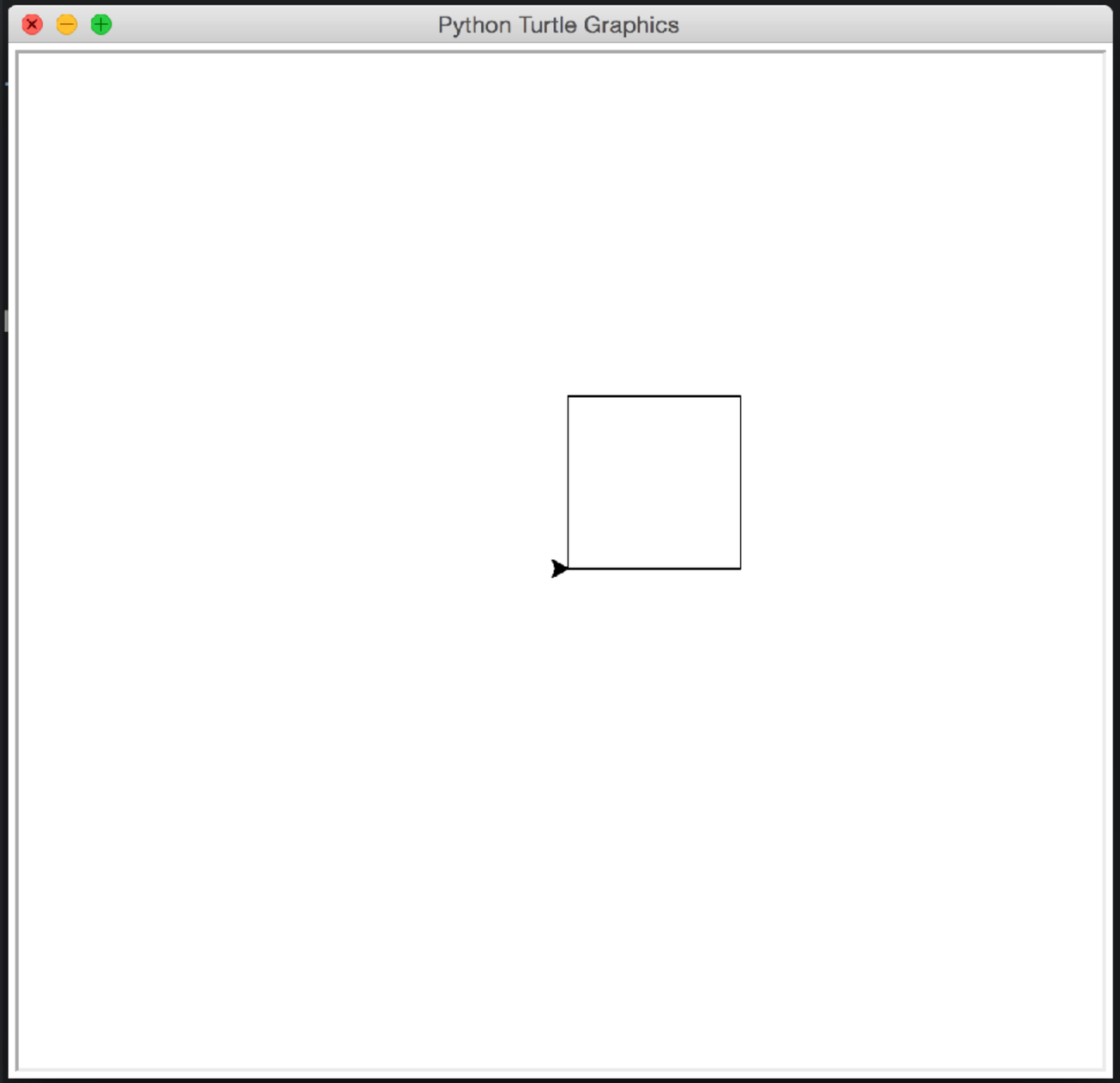



```
$ python draw_a_square.py
```

What we just did is called a refactoring: modifying code to improve the structure and quality of the code without changing the behavior of the code. Refactoring is a very important practice and should be a big part of your software practices. In the long run, it's the #1 thing that will keep you sane as you work with larger and larger projects.

draw_a_square.py

```
1 from turtle import *
2
3
4
5
6
7
8
```




```
$ python draw_a_square.py
```

Click on the red dot on the top left corner to close the program.

draw_a_square.py

```
1  from turtle import *
2
3  for i in range(4):
4      forward(100)
5      left(90)
6
7  mainloop()
8
```

```
$ python draw_a_square.py
$
```

Click on the red dot on the top left corner to close the program.

draw_a_square.py

```
1  from turtle import *
2
3  for i in range(4):
4      forward(100)
5      left(90)
6
7  mainloop()
8
```

```
$
```

Now, what if we wanted to draw more squares?

draw_a_square.py

```
1  from turtle import *
2
3  for i in range(4):
4      forward(100)
5      left(90)
6
7  forward(200)
8
9  mainloop()
10
```

We could move forward after drawing the first square.

draw_a_square.py

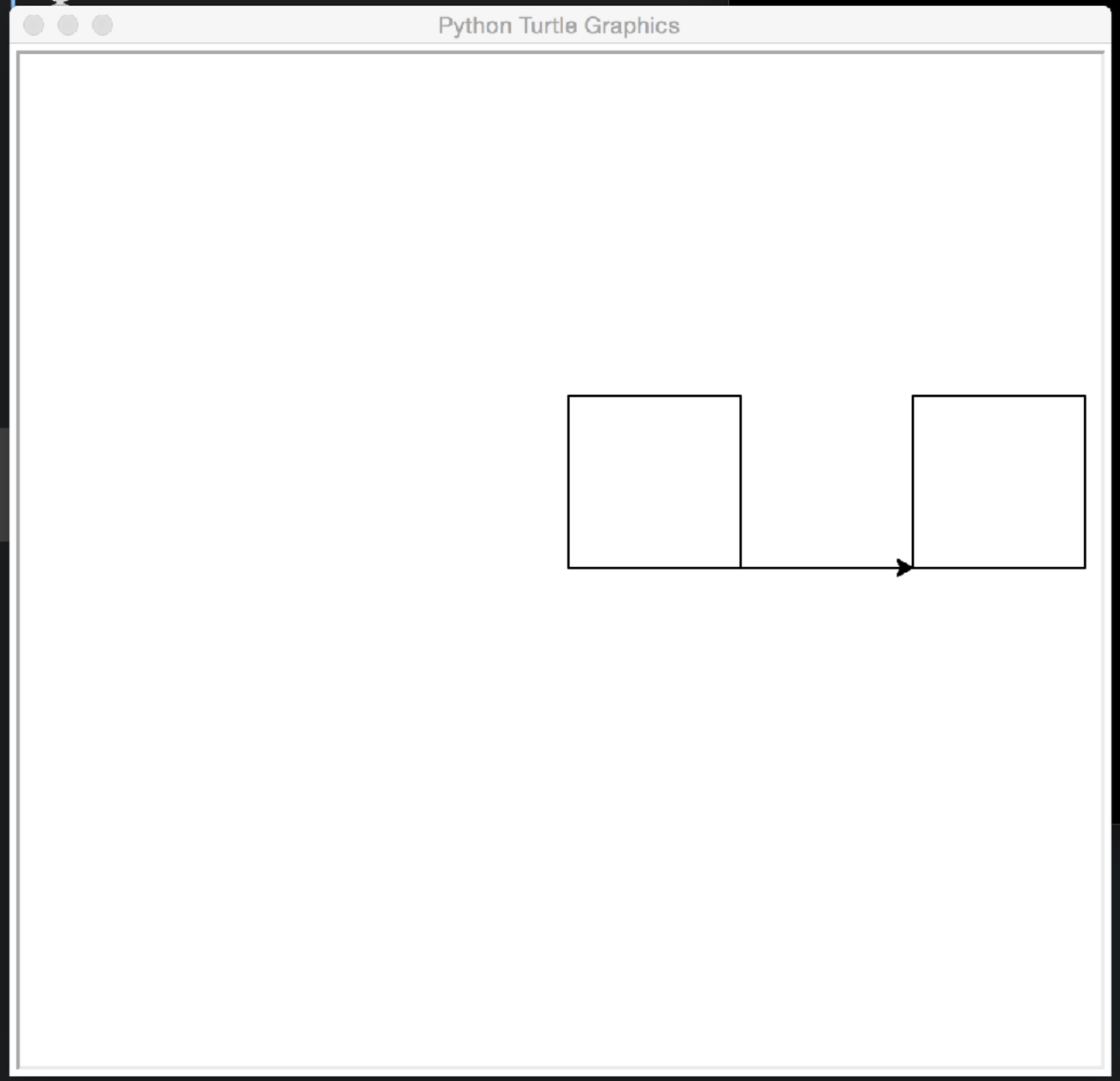
```
1  from turtle import *
2
3  for i in range(4):
4      forward(100)
5      left(90)
6
7  forward(200)
8
9  for i in range(4):
10     forward(100)
11     left(90)
12
13 mainloop()
14
```

Then draw another one using the same code.

draw_a_square.py

```
1  from turtle import *
2
3  for i in range(4)
4      forward(100)
5      left(90)
6
7  forward(200)
8
9  for i in range(4)
10     forward(100)
11     left(90)
12
13 mainloop()
14
```

```
$ python draw_a_square.py
```



That worked. Except that there's a line connecting the two.

draw_a_square.py

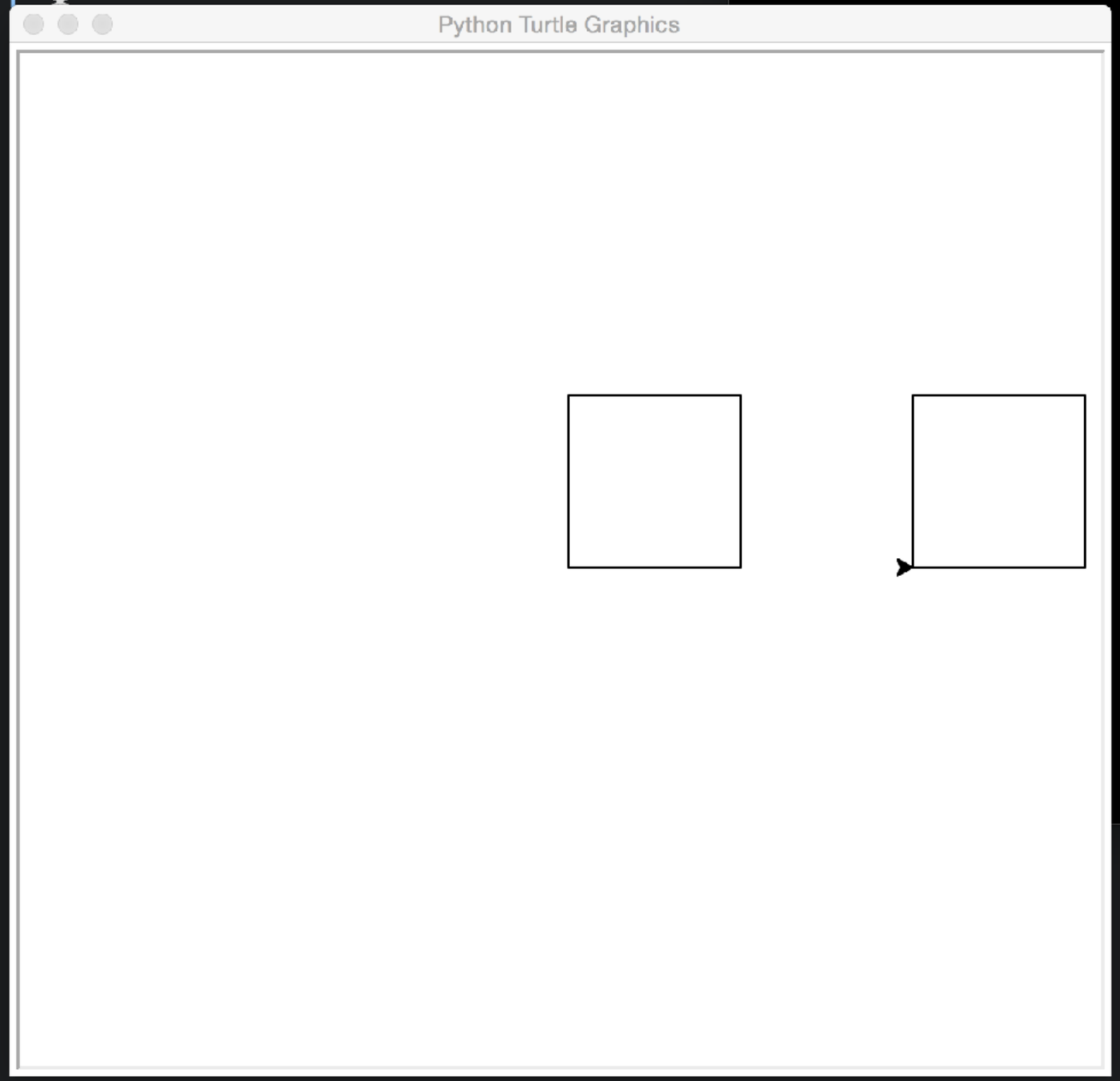
```
1   from turtle import *
2
3   for i in range(4):
4       forward(100)
5       left(90)
6
7   up()
8   forward(200)
9   down()
10
11  for i in range(4):
12      forward(100)
13      left(90)
14
15  mainloop()
16
```

To get rid of that line, we want to lift up the pen before the move forward and then put the pen down when drawing the second square.

draw_a_square.py

```
1  from turtle import *
2
3  for i in range(4):
4      forward(100)
5      left(90)
6
7  up()
8  forward(200)
9  down()
10
11 for i in range(4):
12     forward(100)
13     left(90)
14
15 mainloop()
16
```

```
$ python draw_a_square.py
```



There we go.

draw_a_square.py

```
1  from turtle import *
2
3  for i in range(4):
4      forward(100)
5      left(90)
6
7  up()
8  forward(200)
9  down()
10
11 for i in range(4):
12     forward(100)
13     left(90)
14
15 mainloop()
16
```

Instead repeating these 3 lines of code every time we want to draw a square, we can instead extract it into a function - as a unit of work. Then every time we want to draw a square, we just call that function.

draw_a_square.py

```
1  from turtle import *
2
3  def draw_square():
4      for i in range(4):
5          forward(100)
6          left(90)
7
8  up()
9  forward(200)
10 down()
11
12 for i in range(4):
13     forward(100)
14     left(90)
15
16 mainloop()
17
```

First we create the draw_square function and indent the existing square drawing code within it.

draw_a_square.py

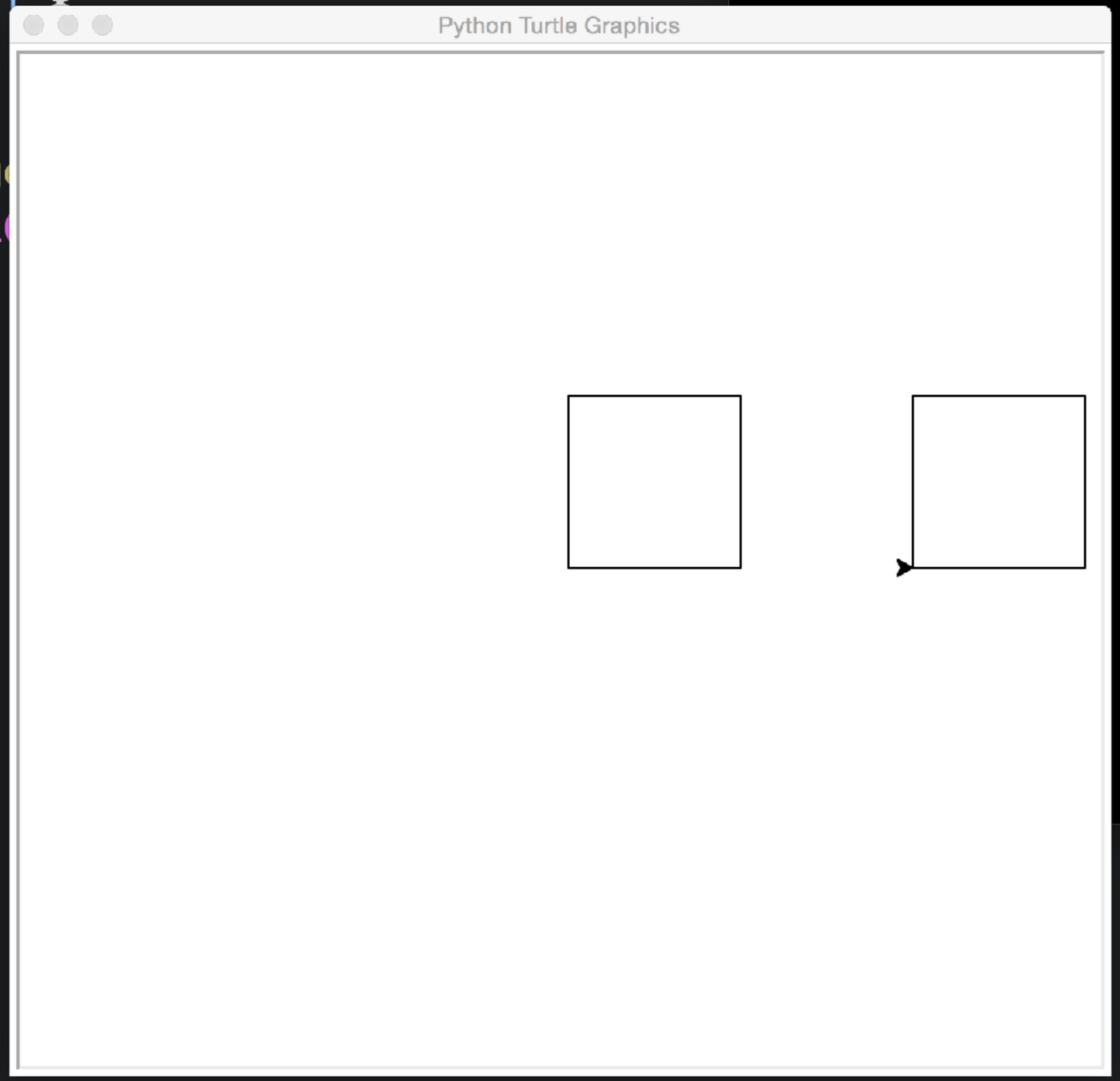
```
1   from turtle import *
2
3   def draw_square():
4       for i in range(4):
5           forward(100)
6           left(90)
7
8   draw_square()
9
10  up()
11  forward(200)
12  down()
13
14  draw_square()
15
16  mainloop()
17
```

Then we call that function in the required places.

draw_a_square.py

```
1  from turtle import *
2
3  def draw_square()
4      for i in rang
5          forward(1
6          left(90)
7
8  draw_square()
9
10 up()
11 forward(200)
12 down()
13
14 draw_square()
15
16 mainloop()
17
```

```
$ python draw_a_square.py
```



We made another refactoring change, and so again, we want to re-test the code to make sure we didn't make a mistake that broke something.

draw_a_square.py

```
1  from turtle import *
2
3  def draw_square():
4      for i in range(4):
5          forward(100)
6          left(90)
7
8  if __name__ == '__main__':
9      draw_square()
10
11     up()
12     forward(200)
13     down()
14
15     draw_square()
16
17     mainloop()
18
```

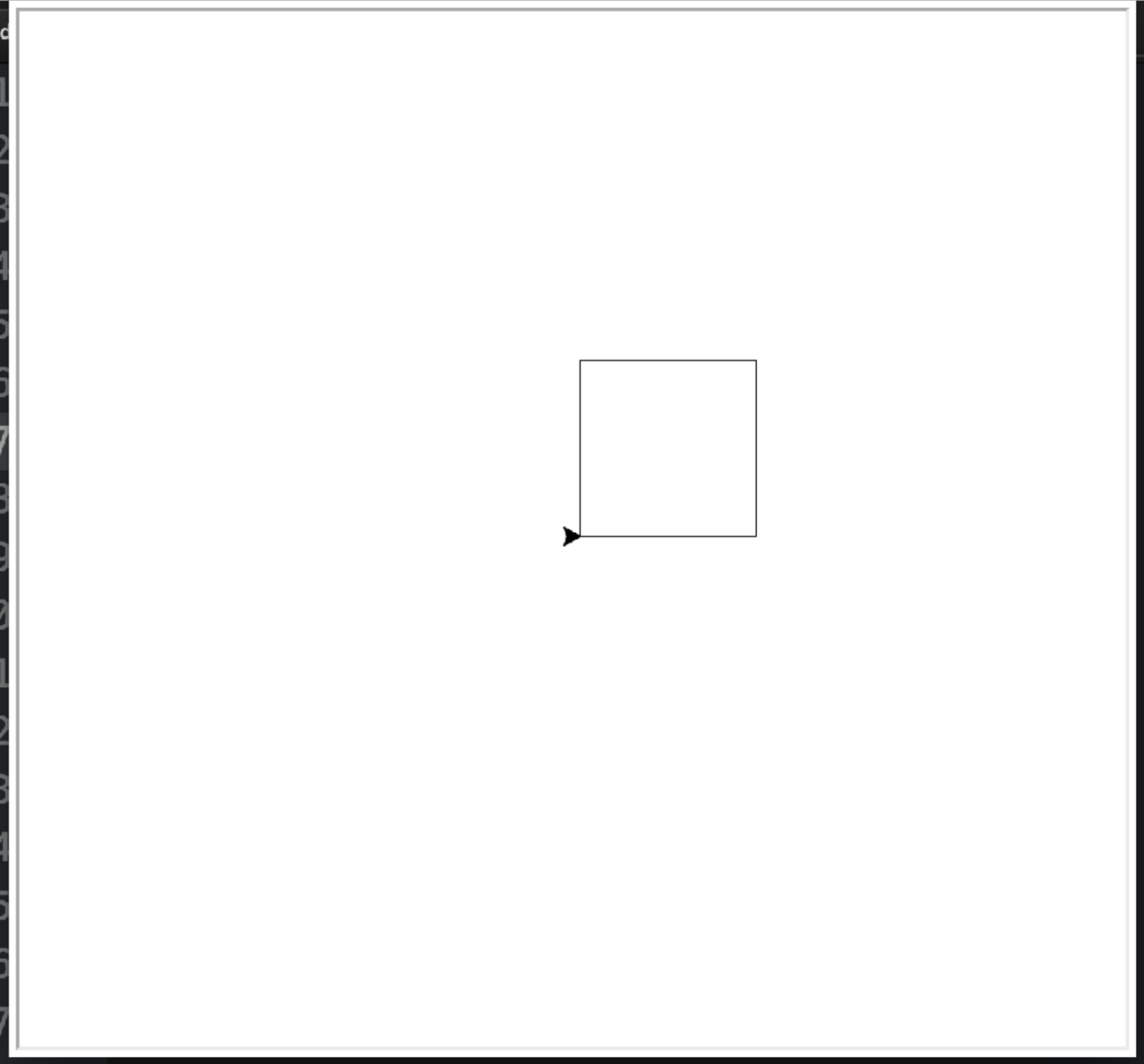
Now, let's draw more squares. I want to do some interactive coding using the python shell, and I want to import my draw_a_square module into the shell so that I can call the draw_square function. To prevent the existing drawing code from running during my interactive coding session, I will add a if statement that says: run this code only if this file is being run as the main program.

draw_a_square.py

```
1  from turtle import *
2
3  def draw_square():
4      for i in range(4):
5          forward(100)
6          left(90)
7
8  if __name__ == '__main__':
9      draw_square()
10
11     up()
12     forward(200)
13     down()
14
15     draw_square()
16
17     mainloop()
18
```

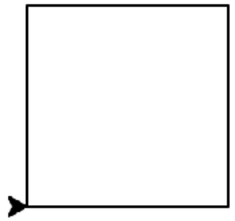
```
$ python
Python 2.7.12 (default, Jun 29 2016, 14:04:44)
[GCC 4.2.1 Compatible Apple LLVM 7.0.2 (clang-70
0.1.81)] on darwin
Type "help", "copyright", "credits" or "license"
 for more information.
>>>
```

Now I'll fire off the interactive shell.



```
$ python
Python 2.7.12 (default, Jun 29 2016, 14:04:44)
[GCC 4.2.1 Compatible Apple LLVM 7.0.2 (clang-70
0.1.81)] on darwin
Type "help", "copyright", "credits" or "license"
 for more information.
>>> import draw_a_square
>>> draw_a_square.draw_square()
>>>
```

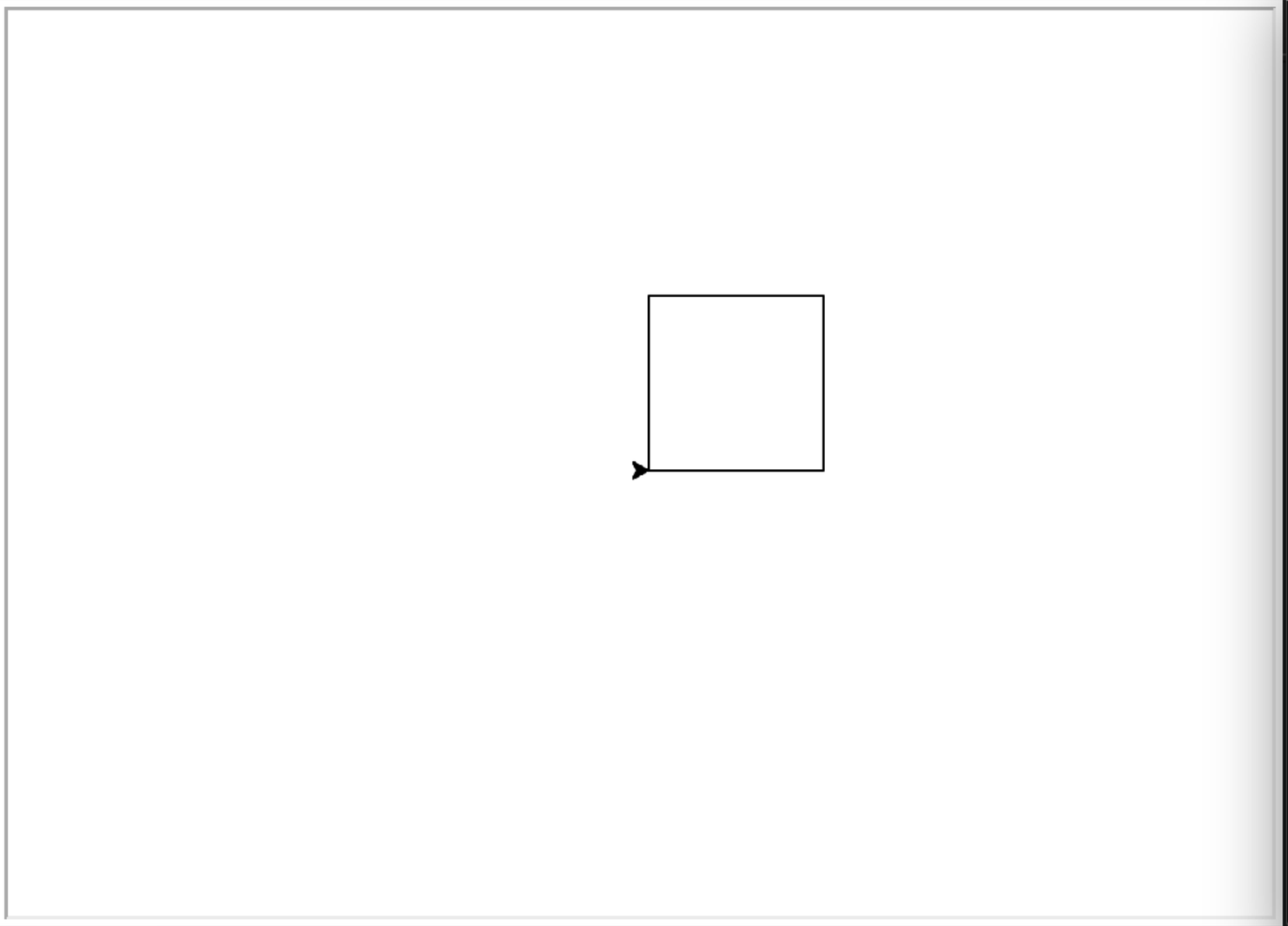
And draw a square using the draw_square function within the module.



```
$ python
Python 2.7.12 (default, Jun 29 2016, 14:04:44)
[GCC 4.2.1 Compatible Apple LLVM 7.0.2 (clang-70
0.1.81)] on darwin
Type "help", "copyright", "credits" or "license"
 for more information.
>>> import draw_a_square
>>> draw_a_square.draw_square()
>>>
```

```
16
17      mainloop()
18
```

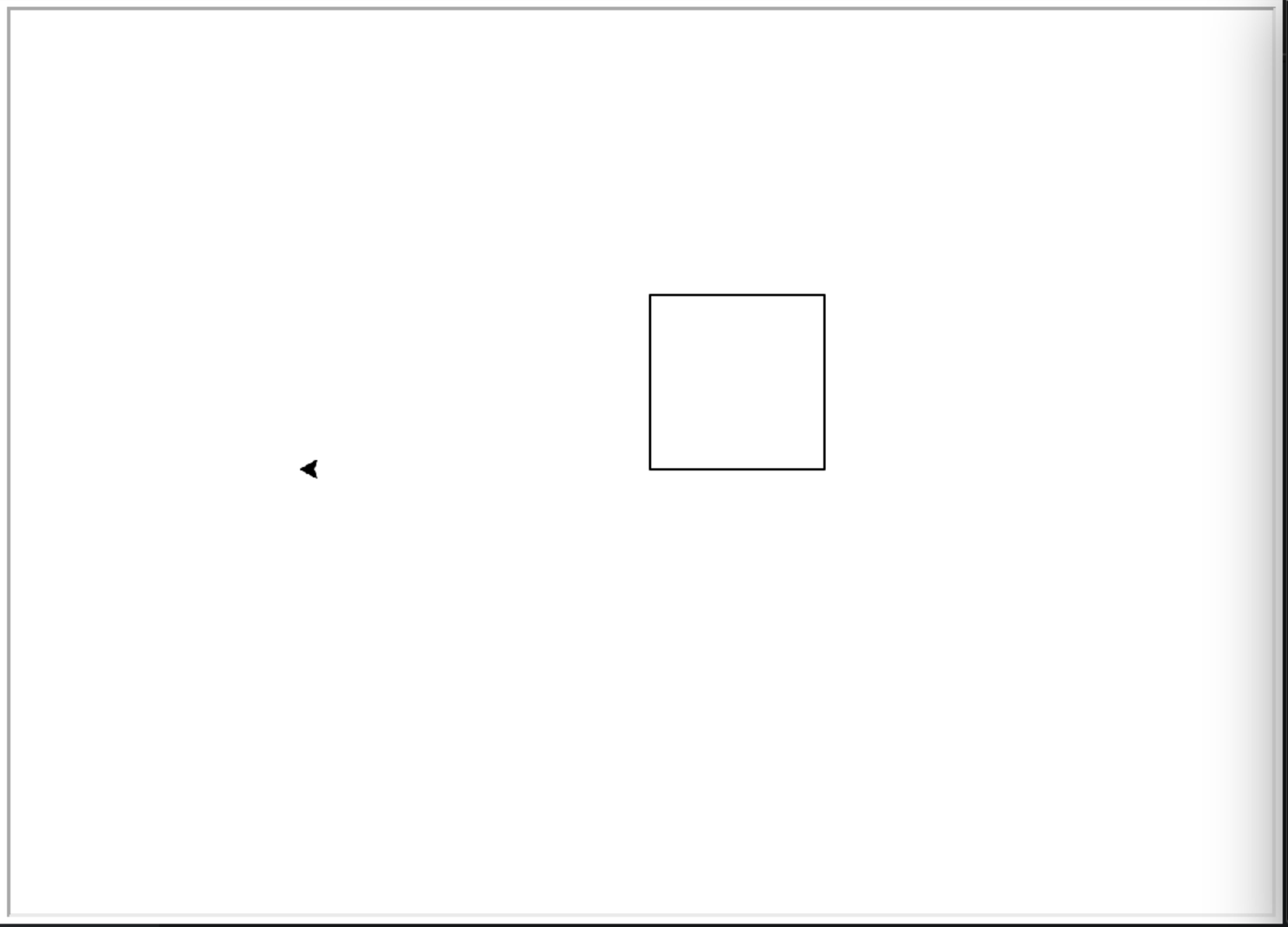
And draw a square using the draw_square function within the module.



```
$ python
Python 2.7.12 (default, Jun 29 2016, 14:04:44)
[GCC 4.2.1 Compatible Apple LLVM 7.0.2 (clang-70
0.1.81)] on darwin
Type "help", "copyright", "credits" or "license"
 for more information.
>>> import draw_a_square
>>> draw_a_square.draw_square()
>>> from turtle import *
>>>
```

```
16
17      mainloop()
18
```

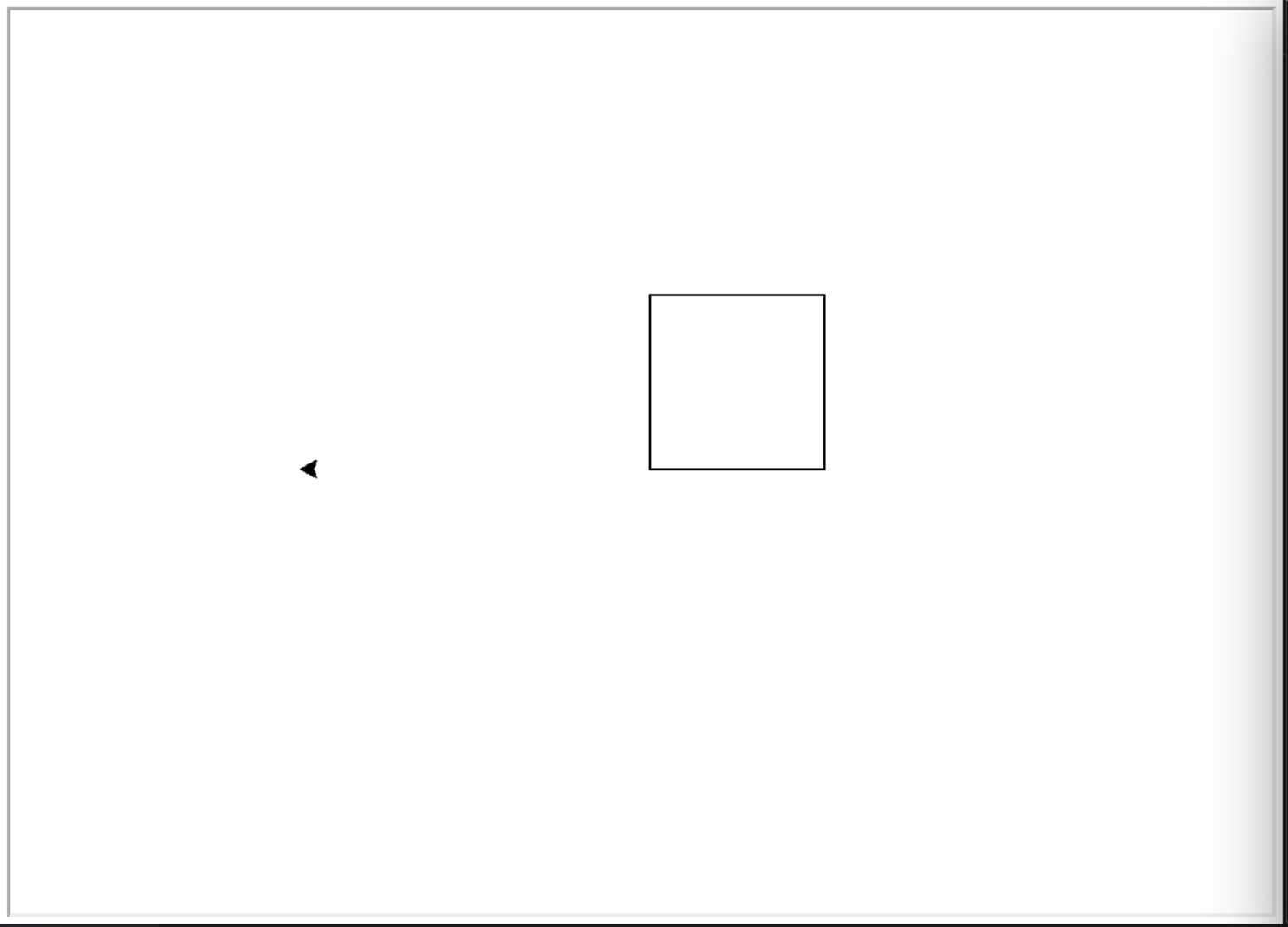
Now I'll also import all the functions from the turtle module.



```
$ python
Python 2.7.12 (default, Jun 29 2016, 14:04:44)
[GCC 4.2.1 Compatible Apple LLVM 7.0.2 (clang-70
0.1.81)] on darwin
Type "help", "copyright", "credits" or "license"
 for more information.
>>> import draw_a_square
>>> draw_a_square.draw_square()
>>> from turtle import *
>>> left(180)
>>> up()
>>> forward(200)
>>>
```

```
16
17      mainloop()
18
```

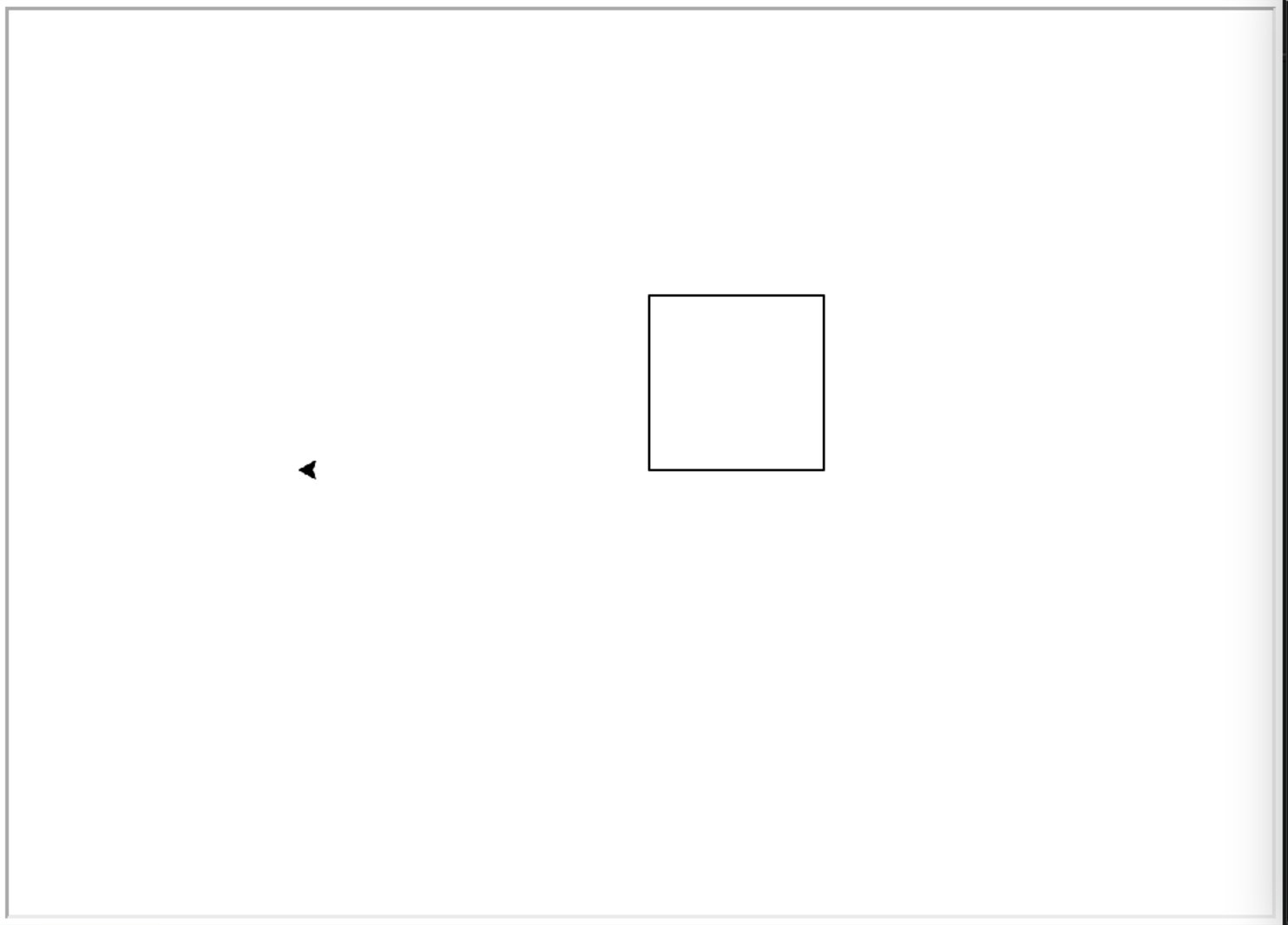
Let's move elsewhere (with the pen up).



```
$ python
Python 2.7.12 (default, Jun 29 2016, 14:04:44)
[GCC 4.2.1 Compatible Apple LLVM 7.0.2 (clang-70
0.1.81)] on darwin
Type "help", "copyright", "credits" or "license"
 for more information.
>>> import draw_a_square
>>> draw_a_square.draw_square()
>>> from turtle import *
>>> left(180)
>>> up()
>>> forward(200)
>>> draw_a_square.draw_square()
>>>
```

```
16
17      mainloop()
18
```

Now draw another square.



```
$ python
Python 2.7.12 (default, Jun 29 2016, 14:04:44)
[GCC 4.2.1 Compatible Apple LLVM 7.0.2 (clang-70
0.1.81)] on darwin
Type "help", "copyright", "credits" or "license"
 for more information.
>>> import draw_a_square
>>> draw_a_square.draw_square()
>>> from turtle import *
>>> left(180)
>>> up()
>>> forward(200)
>>> draw_a_square.draw_square()
>>>
```

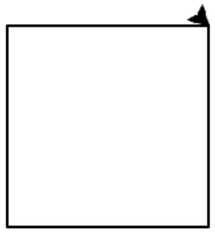
```
16
17      mainloop()
18
```

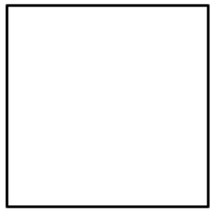
Oops! That didn't work because I forgot to put the pen down.

```
$ python
Python 2.7.12 (default, Jun 29 2016, 14:04:44)
[GCC 4.2.1 Compatible Apple LLVM 7.0.2 (clang-70
0.1.81)] on darwin
Type "help", "copyright", "credits" or "license"
 for more information.
>>> import draw_a_square
>>> draw_a_square.draw_square()
>>> from turtle import *
>>> left(180)
>>> up()
>>> forward(200)
>>> draw_a_square.draw_square()
>>> down()
>>> draw_a_square.draw_square()
>>>
```

```
16
17      mainloop()
18
```

Let's put the pen down and try again. There!





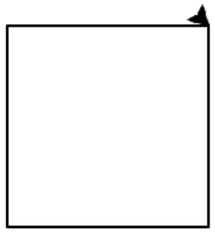
```
$ python
Python 2.7.12 (default, Jun 29 2016, 14:04:44)
[GCC 4.2.1 Compatible Apple LLVM 7.0.2 (clang-70
0.1.81)] on darwin
Type "help", "copyright", "credits" or "license"
 for more information.
>>> import draw_a_square
>>> draw_a_square.draw_square()
>>> from turtle import *
>>> left(180)
>>> up()
>>> forward(200)
>>> draw_a_square.draw_square()
>>> down()
>>> draw_a_square.draw_square()
>>> setheading(315)
>>>
```

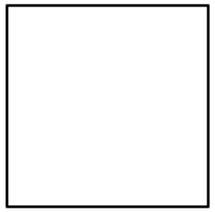

`turtle.`**`home`**`()`

Move turtle to the origin – coordinates (0,0) – and s
orientation (which depends on the mode, see `mode()`).

```
>>> turtle.heading()
90.0
```

We'll move to another open part of the canvas. Instead of turning left or right, this time, we'll use the setheading function, which tells the turtle to face a specific direction in degrees, regardless of which way it is facing at the moment.





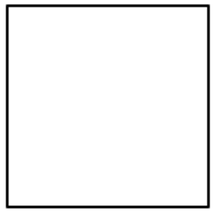
```
$ python
Python 2.7.12 (default, Jun 29 2016, 14:04:44)
[GCC 4.2.1 Compatible Apple LLVM 7.0.2 (clang-70
0.1.81)] on darwin
Type "help", "copyright", "credits" or "license"
 for more information.
>>> import draw_a_square
>>> draw_a_square.draw_square()
>>> from turtle import *
>>> left(180)
>>> up()
>>> forward(200)
>>> draw_a_square.draw_square()
>>> down()
>>> draw_a_square.draw_square()
>>> setheading(315)
>>> up()
>>>
```

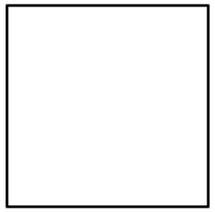

`turtle.`**`home`**`()`

Move turtle to the origin – coordinates (0,0) – and s
orientation (which depends on the mode, see `mode()`

```
>>> turtle.heading()
90.0
```

Lift the pen.





```
$ python
Python 2.7.12 (default, Jun 29 2016, 14:04:44)
[GCC 4.2.1 Compatible Apple LLVM 7.0.2 (clang-70
0.1.81)] on darwin
Type "help", "copyright", "credits" or "license"
 for more information.
>>> import draw_a_square
>>> draw_a_square.draw_square()
>>> from turtle import *
>>> left(180)
>>> up()
>>> forward(200)
>>> draw_a_square.draw_square()
>>> down()
>>> draw_a_square.draw_square()
>>> setheading(315)
>>> up()
>>> forward(300)
>>>
```

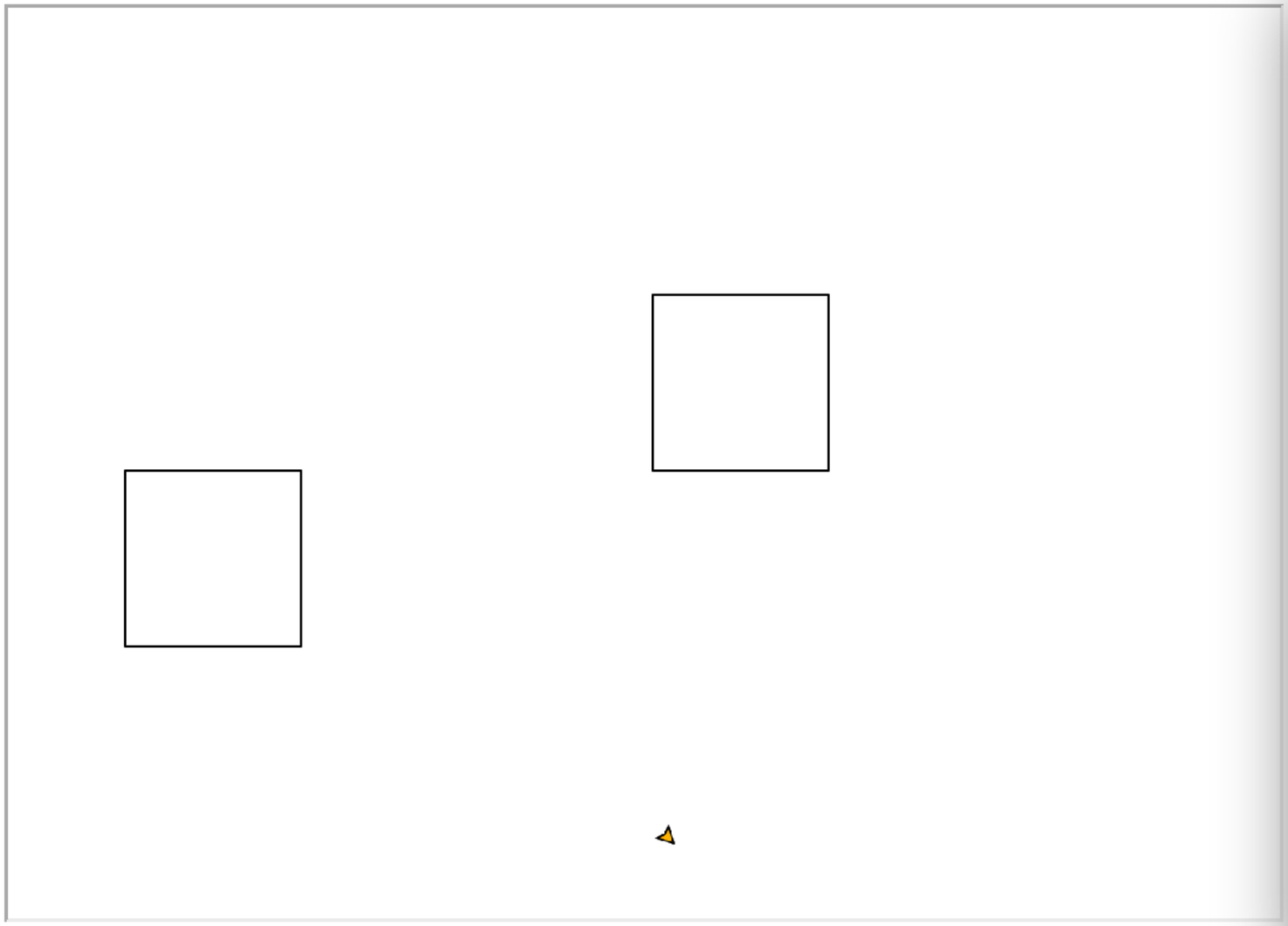
- 24.5.3.1. Turtle motion
- 24.5.3.2. Tell Turtle's state
- 24.5.3.3. Settings for measurement
- 24.5.3.4. Pen

`turtle.`**`home`**`()`

Move turtle to the origin – coordinates (0,0) – and s
orientation (which depends on the mode, see `mode()`

```
>>> turtle.heading()
90.0
```

move forward.



```
$ python
Python 2.7.12 (default, Jun 29 2016, 14:04:44)
[GCC 4.2.1 Compatible Apple LLVM 7.0.2 (clang-70
0.1.81)] on darwin
Type "help", "copyright", "credits" or "license"
 for more information.
>>> import draw_a_square
>>> draw_a_square.draw_square()
>>> from turtle import *
>>> left(180)
>>> up()
>>> forward(200)
>>> draw_a_square.draw_square()
>>> down()
>>> draw_a_square.draw_square()
>>> setheading(315)
>>> up()
>>> forward(300)
>>> fillcolor('orange')
>>>
```

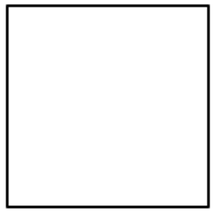
- 24.5.3.1. Turtle motion
- 24.5.3.2. Tell Turtle's state
- 24.5.3.3. Settings for measurement
- 24.5.3.4. Pen

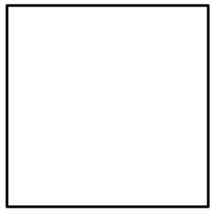
`turtle.`**`home`**`()`

Move turtle to the origin – coordinates (0,0) – and s
orientation (which depends on the mode, see `mode()`

```
>>> turtle.heading()
90.0
```

This time we want to fill a square. To do that, we'll first set the fillcolor - let's set it to orange.





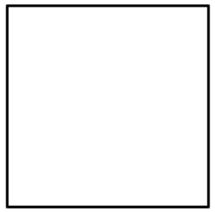
```
$ python
Python 2.7.12 (default, Jun 29 2016, 14:04:44)
[GCC 4.2.1 Compatible Apple LLVM 7.0.2 (clang-70
0.1.81)] on darwin
Type "help", "copyright", "credits" or "license"
 for more information.
>>> import draw_a_square
>>> draw_a_square.draw_square()
>>> from turtle import *
>>> left(180)
>>> up()
>>> forward(200)
>>> draw_a_square.draw_square()
>>> down()
>>> draw_a_square.draw_square()
>>> setheading(315)
>>> up()
>>> forward(300)
>>> fillcolor('orange')
>>> begin_fill()
>>>
```

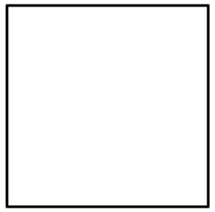

`turtle.`**`home`**`()`

Move turtle to the origin – coordinates (0,0) – and s
orientation (which depends on the mode, see `mode()`

```
>>> turtle.heading()
90.0
```

Then we'll call the begin_fill function.





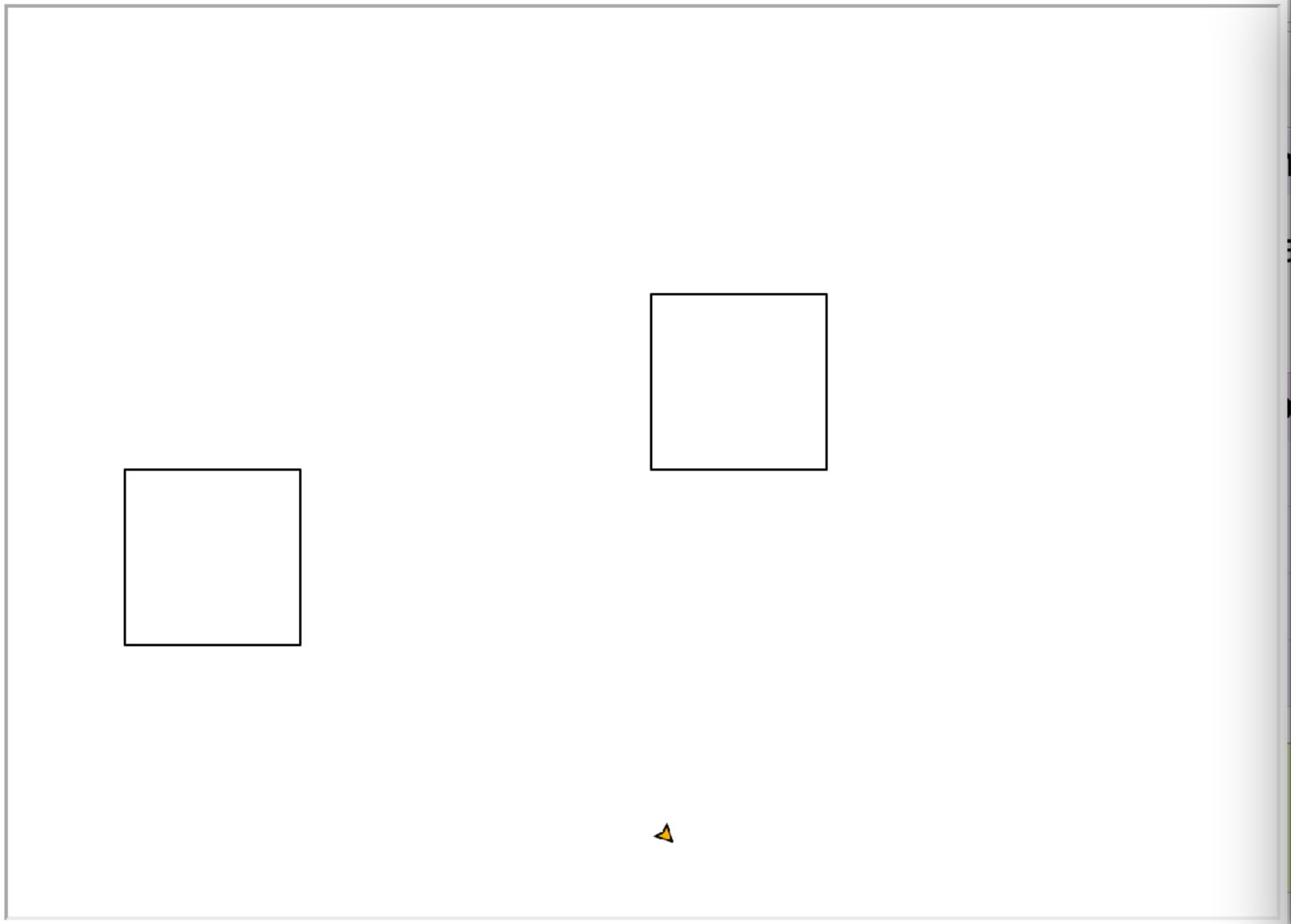
```
$ python
Python 2.7.12 (default, Jun 29 2016, 14:04:44)
[GCC 4.2.1 Compatible Apple LLVM 7.0.2 (clang-70
0.1.81)] on darwin
Type "help", "copyright", "credits" or "license"
 for more information.
>>> import draw_a_square
>>> draw_a_square.draw_square()
>>> from turtle import *
>>> left(180)
>>> up()
>>> forward(200)
>>> draw_a_square.draw_square()
>>> down()
>>> draw_a_square.draw_square()
>>> setheading(315)
>>> up()
>>> forward(300)
>>> fillcolor('orange')
>>> begin_fill()
>>> draw_a_square.draw_square()
```


`turtle.`**`home`**`()`

Move turtle to the origin – coordinates (0,0) – and s
orientation (which depends on the mode, see `mode()`

```
>>> turtle.heading()
90.0
```

When draw a closing shape, in our case, the square.



```
Python 2.7.12 (default, Jun 29 2016, 14:04:44)
[GCC 4.2.1 Compatible Apple LLVM 7.0.2 (clang-70
0.1.81)] on darwin
Type "help", "copyright", "credits" or "license"
 for more information.
>>> import draw_a_square
>>> draw_a_square.draw_square()
>>> from turtle import *
>>> left(180)
>>> up()
>>> forward(200)
>>> draw_a_square.draw_square()
>>> down()
>>> draw_a_square.draw_square()
>>> setheading(315)
>>> up()
>>> forward(300)
>>> fillcolor('orange')
>>> begin_fill()
>>> draw_a_square.draw_square()
>>>
```

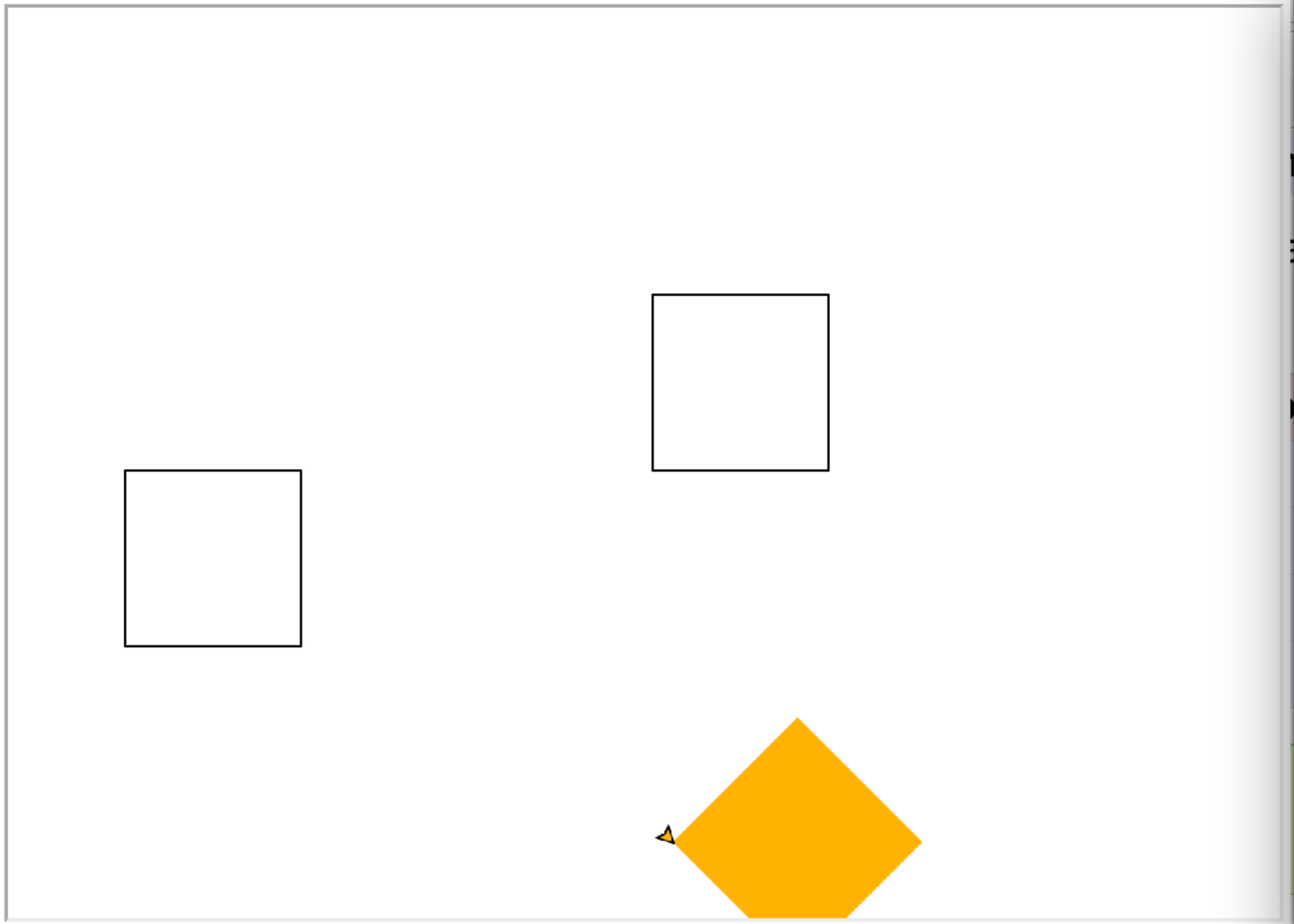
- 24.5.3.1. Turtle motion
- 24.5.3.2. Tell Turtle's state
- 24.5.3.3. Settings for measurement
- 24.5.3.4. Pen

`turtle.`**`home`**`()`

Move turtle to the origin – coordinates (0,0) – and s
orientation (which depends on the mode, see `mode()`

```
>>> turtle.heading()
90.0
```

When draw a closing shape, in our case, the square.



```
[GCC 4.2.1 Compatible Apple LLVM 7.0.2 (clang-70
0.1.81)] on darwin
Type "help", "copyright", "credits" or "license"
 for more information.
>>> import draw_a_square
>>> draw_a_square.draw_square()
>>> from turtle import *
>>> left(180)
>>> up()
>>> forward(200)
>>> draw_a_square.draw_square()
>>> down()
>>> draw_a_square.draw_square()
>>> setheading(315)
>>> up()
>>> forward(300)
>>> fillcolor('orange')
>>> begin_fill()
>>> draw_a_square.draw_square()
>>> end_fill()
>>>
```

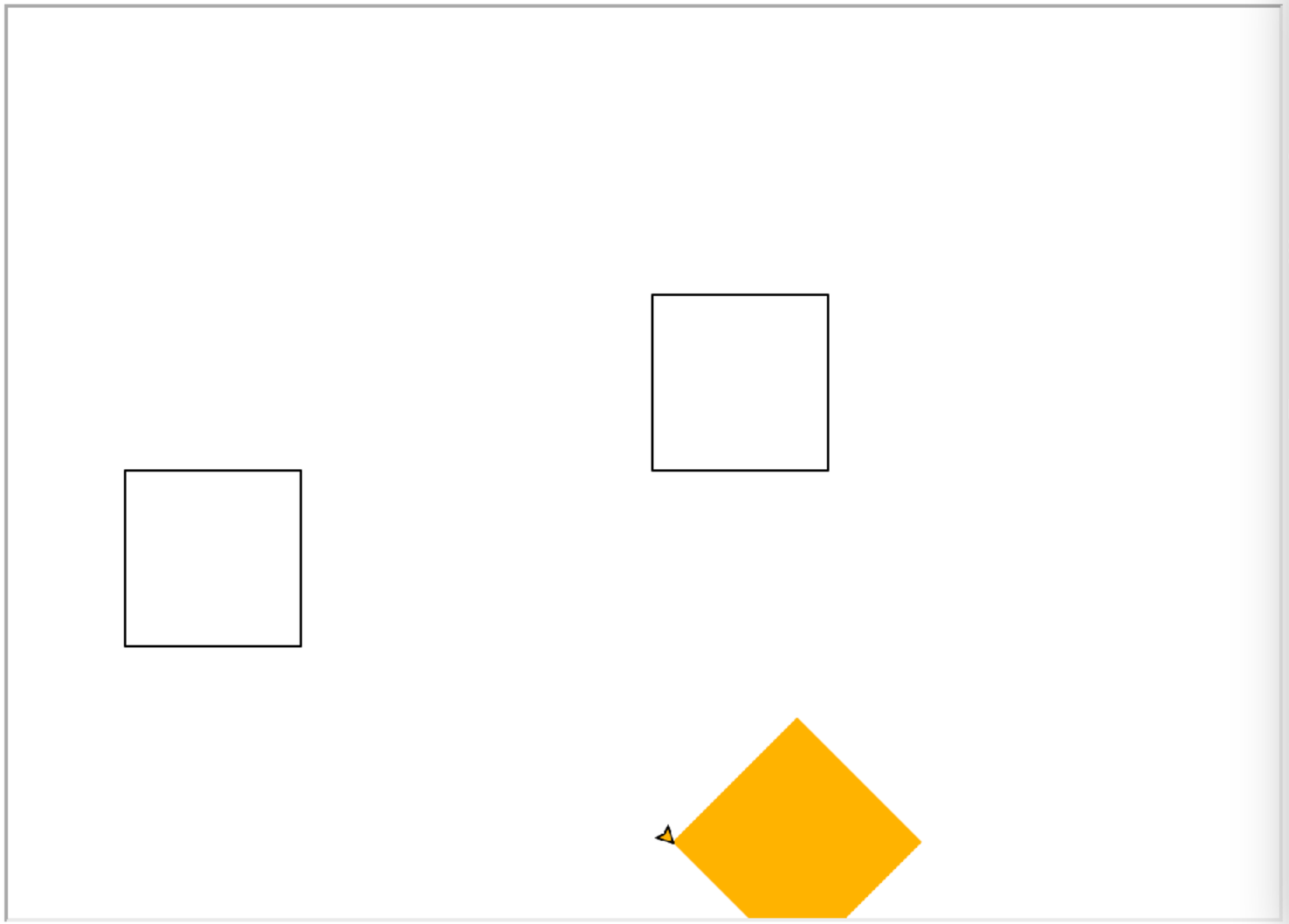
- 24.5.3.1. Turtle motion
- 24.5.3.2. Tell Turtle's state
- 24.5.3.3. Settings for measurement
- 24.5.3.4. Pen

`turtle.`**`home`**`()`

Move turtle to the origin – coordinates (0,0) – and s
orientation (which depends on the mode, see `mode()`

```
>>> turtle.heading()
90.0
```

Then call the end_fill function.



```
[GCC 4.2.1 Compatible Apple LLVM 7.0.2 (clang-70
0.1.81)] on darwin
Type "help", "copyright", "credits" or "license"
 for more information.
>>> import draw_a_square
>>> draw_a_square.draw_square()
>>> from turtle import *
>>> left(180)
>>> up()
>>> forward(200)
>>> draw_a_square.draw_square()
>>> down()
>>> draw_a_square.draw_square()
>>> setheading(315)
>>> up()
>>> forward(300)
>>> fillcolor('orange')
>>> begin_fill()
>>> draw_a_square.draw_square()
>>> end_fill()
>>>
```

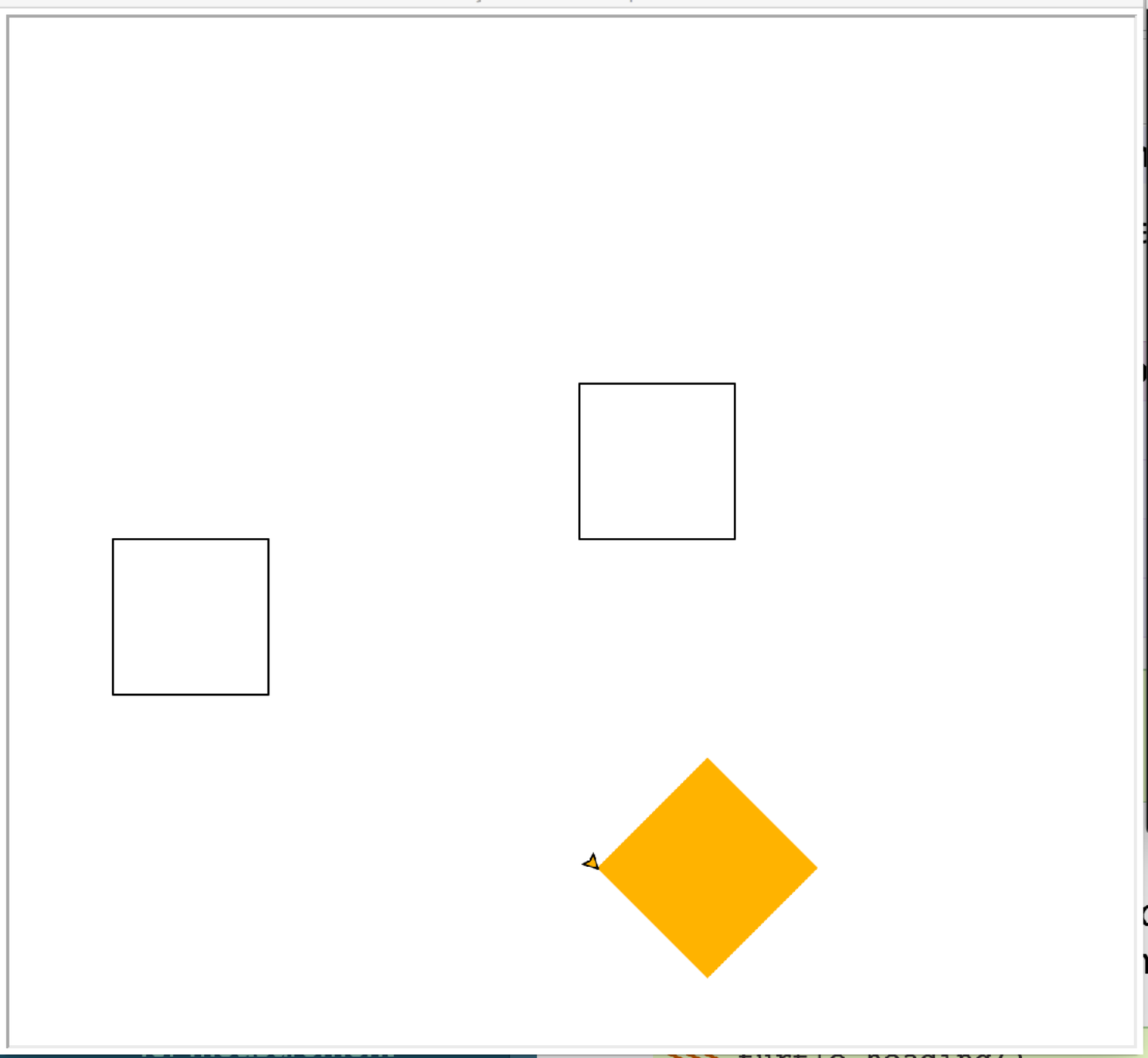
- 24.5.3.1. Turtle motion
- 24.5.3.2. Tell Turtle's state
- 24.5.3.3. Settings for measurement
- 24.5.3.4. Pen

`turtle.`**`home`**`()`

Move turtle to the origin – coordinates (0,0) – and s
orientation (which depends on the mode, see `mode()`)

```
>>> turtle.heading()
90.0
```

And we have an orange-filled square!



```
[GCC 4.2.1 Compatible Apple LLVM 7.0.2 (clang-70
0.1.81)] on darwin
Type "help", "copyright", "credits" or "license"
 for more information.
>>> import draw_a_square
>>> draw_a_square.draw_square()
>>> from turtle import *
>>> left(180)
>>> up()
>>> forward(200)
>>> draw_a_square.draw_square()
>>> down()
>>> draw_a_square.draw_square()
>>> setheading(315)
>>> up()
>>> forward(300)
>>> fillcolor('orange')
>>> begin_fill()
>>> draw_a_square.draw_square()
>>> end_fill()
>>>
```

ordinates (0,0) – and s
the mode, see `mode()`

Stretch the window a little more so that we can see it :)

- 24.5.3.4. Pen

```
>>> turtle.heading()
90.0
```

draw_squares.py

```
1
```

```
$
```

One more thing before we go. What if you want to draw a number of squares in various coordinates? How would you do it?

draw_squares.py

```
1  from turtle import *
2  from draw_a_square import draw_square
3
```

$

We'll create a new draw_squares.py file which makes use of the draw_square function within the draw_a_square.py file

draw_squares.py

```
1  from turtle import *
2  from draw_a_square import draw_square
3
4  coord_list = [(-100, -100), (100,
•  100), (-100, 100), (100, -100)]
5
```

```
$
```

We define a list of coordinates (tuples) where we want to draw a square at.

draw_squares.py

```
1  from turtle import *
2  from draw_a_square import draw_square
3
4  coord_list = [(-100, -100), (100,
•  100), (-100, 100), (100, -100)]
5
```

```
$
```

This is our strategy:

1. for each coordinate:
    0. lift the pen
    1. go back to the (0, 0) coordinate
    2. move to the current coordinate by using setheading and forward twice, once to move to the correct x location, and once to move to the correct y location.
    3. put down the pen
    4. draw the square

draw_squares.py

```
1  from turtle import *
2  from draw_a_square import draw_square
3
4  coord_list = [(-100, -100), (100,
   100), (-100, 100), (100, -100)]
5
6  for coord in coord_list:
7
```

$

So, first we'll use a for loop to loop through coord_list.

This is our strategy:

1. for each coordinate:
    0. lift the pen
    1. go back to the (0, 0) coordinate
    2. move to the current coordinate by using setheading and forward twice, once to move to the correct x location, and once to move to the correct y location.
    3. put down the pen
    4. draw the square

draw_squares.py

```
1  from turtle import *
2  from draw_a_square import draw_square
3
4  coord_list = [(-100, -100), (100,
   100), (-100, 100), (100, -100)]
5
6  for coord in coord_list:
7      up()
8
```

$

For each coordinate, we'll lift the pen.

This is our strategy:

1. for each coordinate:
    0. lift the pen
    1. go back to the (0, 0) coordinate
    2. move to the current coordinate by using setheading and forward twice, once to move to the correct x location, and once to move to the correct y location.
    3. put down the pen
    4. draw the square

draw_squares.py

```
1  from turtle import *
2  from draw_a_square import draw_square
3
4  coord_list = [(-100, -100), (100,
•  100), (-100, 100), (100, -100)]
5
6  for coord in coord_list:
7      up()
8      home()
9
```

```
$
```

Then we'll use the home function to go back to the (0, 0) coordinate.

This is our strategy:

1. for each coordinate:
    0. lift the pen
    1. go back to the (0, 0) coordinate
    2. move to the current coordinate by using setheading and forward twice, once to move to the correct x location, and once to move to the correct y location.
    3. put down the pen
    4. draw the square

draw_squares.py

```
1  from turtle import *
2  from draw_a_square import draw_square
3
4  coord_list = [(-100, -100), (100,
   100), (-100, 100), (100, -100)]
5
6  for coord in coord_list:
7      up()
8      home()
9      x, y = coord
10
```

$

Now, we'll move to the correct position based on coord. We'll first destructure it into its x and y components.

This is our strategy:

1. for each coordinate:
    0. lift the pen
    1. go back to the (0, 0) coordinate
    2. move to the current coordinate by using setheading and forward twice, once to move to the correct x location, and once to move to the correct y location.
    3. put down the pen
    4. draw the square

draw_squares.py

```
1   from turtle import *
2   from draw_a_square import draw_square
3
4   coord_list = [(-100, -100), (100,
•   100), (-100, 100), (100, -100)]
5
6   for coord in coord_list:
7       up()
8       home()
9       x, y = coord
10      
11
```

```
$
```

Now, we'll move to the correct position based on coord. We'll first destructure it into its x and y components.

This is our strategy:

1. for each coordinate:
    0. lift the pen
    1. go back to the (0, 0) coordinate
    2. move to the current coordinate by using setheading and forward twice, once to move to the correct x location, and once to move to the correct y location.
    3. put down the pen
    4. draw the square

draw_squares.py

```
1   from turtle import *
2   from draw_a_square import draw_square
3
4   coord_list = [(-100, -100), (100,
•   100), (-100, 100), (100, -100)]
5
6   for coord in coord_list:
7       up()
8       home()
9       x, y = coord
10      setheading(270)
11
```

```
$
```

We'll set the heading to the west, which corresponds to 270 degrees.

This is our strategy:

1. for each coordinate:
    0. lift the pen
    1. go back to the (0, 0) coordinate
    2. move to the current coordinate by using setheading and forward twice, once to move to the correct x location, and once to move to the correct y location.
    3. put down the pen
    4. draw the square

draw_squares.py

```
1   from turtle import *
2   from draw_a_square import draw_square
3
4   coord_list = [(-100, -100), (100,
•   100), (-100, 100), (100, -100)]
5
6   for coord in coord_list:
7       up()
8       home()
9       x, y = coord
10      setheading(270)
11      forward(x)
12
```

```
$
```

Then we'll move forward by the amount of the x coordinate. If x is negative, the turtle will move backwards.

This is our strategy:

1. for each coordinate:
    0. lift the pen
    1. go back to the (0, 0) coordinate
    2. move to the current coordinate by using setheading and forward twice, once to move to the correct x location, and once to move to the correct y location.
    3. put down the pen
    4. draw the square

draw_squares.py

```
1  from turtle import *
2  from draw_a_square import draw_square
3
4  coord_list = [(-100, -100), (100,
   100), (-100, 100), (100, -100)]
5
6  for coord in coord_list:
7      up()
8      home()
9      x, y = coord
10     setheading(270)
11     forward(x)
12     setheading(0)
13
```

$

Then we'll set the heading to north.

This is our strategy:

1. for each coordinate:
    0. lift the pen
    1. go back to the (0, 0) coordinate
    2. move to the current coordinate by using setheading and forward twice, once to move to the correct x location, and once to move to the correct y location.
    3. put down the pen
    4. draw the square

draw_squares.py

```
1  from turtle import *
2  from draw_a_square import draw_square
3
4  coord_list = [(-100, -100), (100,
   100), (-100, 100), (100, -100)]
5
6  for coord in coord_list:
7      up()
8      home()
9      x, y = coord
10     setheading(270)
11     forward(x)
12     setheading(0)
13     forward(y)
14
```

$

And we'll move forward by the amount of the y coordinate.

This is our strategy:

1. for each coordinate:
    0. lift the pen
    1. go back to the (0, 0) coordinate
    2. move to the current coordinate by using setheading and forward twice, once to move to the correct x location, and once to move to the correct y location.
    3. put down the pen
    4. draw the square

draw_squares.py

```
1   from turtle import *
2   from draw_a_square import draw_square
3
4   coord_list = [(-100, -100), (100,
    100), (-100, 100), (100, -100)]
5
6   for coord in coord_list:
7       up()
8       home()
9       x, y = coord
10      setheading(270)
11      forward(x)
12      setheading(0)
13      forward(y)
14      down()
15
```

```
$
```

Time to put the pen down.

This is our strategy:

1. for each coordinate:
    0. lift the pen
    1. go back to the (0, 0) coordinate
    2. move to the current coordinate by using setheading and forward twice, once to move to the correct x location, and once to move to the correct y location.
    3. put down the pen
    4. draw the square

draw_squares.py

```
1  from turtle import *
2  from draw_a_square import draw_square
3
4  coord_list = [(-100, -100), (100,
   100), (-100, 100), (100, -100)]
5
6  for coord in coord_list:
7      up()
8      home()
9      x, y = coord
10     setheading(270)
11     forward(x)
12     setheading(0)
13     forward(y)
14     down()
15     draw_square()
16
```

$

Now draw the square

This is our strategy:

1. for each coordinate:
    0. lift the pen
    1. go back to the (0, 0) coordinate
    2. move to the current coordinate by using setheading and forward twice, once to move to the correct x location, and once to move to the correct y location.
    3. put down the pen
    4. draw the square

draw_squares.py

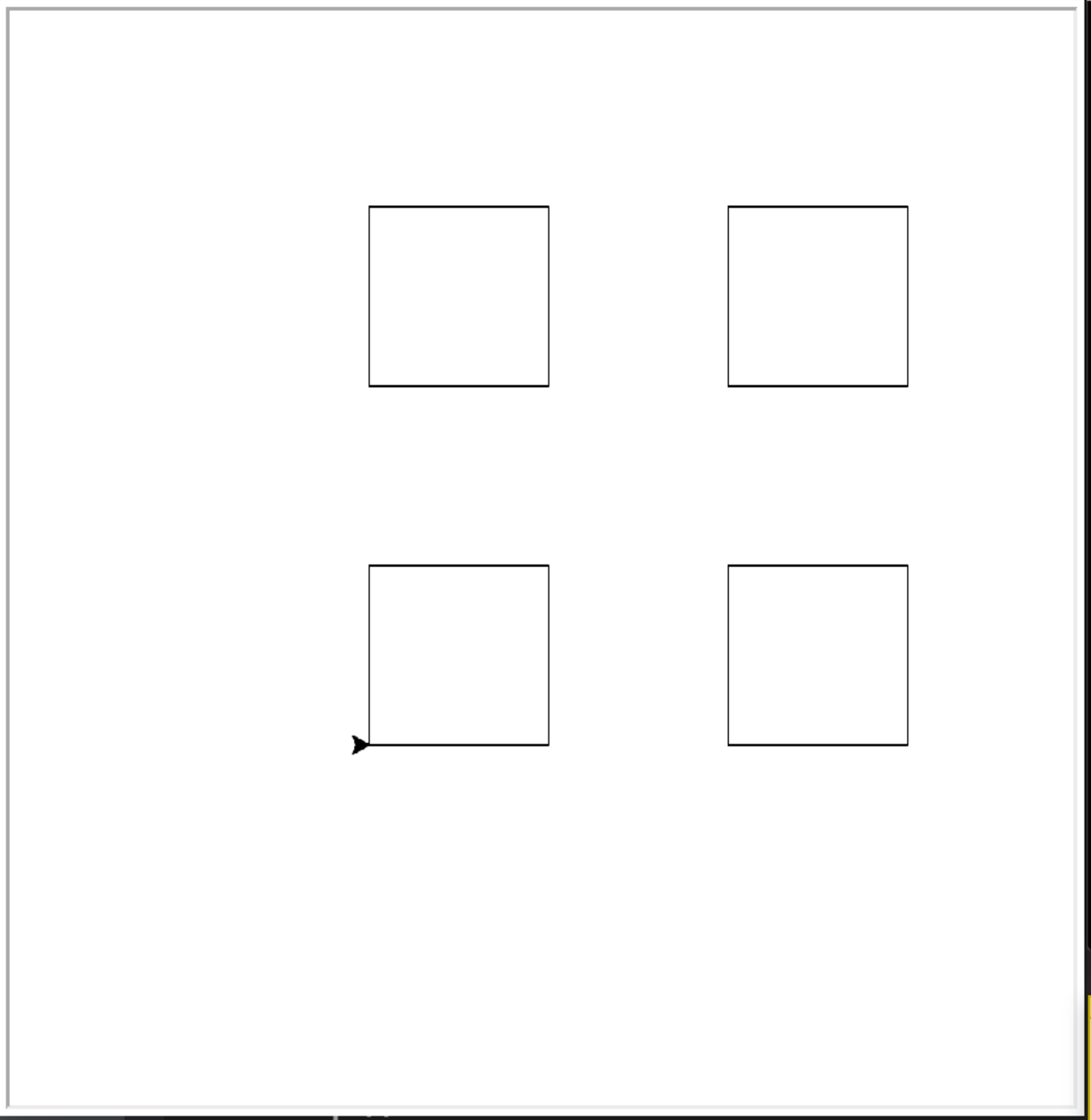
```
1  from turtle import *
2  from draw_a_square import draw_square
3
4  coord_list = [(-100, -100), (100,
•  100), (-100, 100), (100, -100)]
5
6  for coord in coord_list:
7      up()
8      home()
9      x, y = coord
10     setheading(270)
11     forward(x)
12     setheading(0)
13     forward(y)
14     down()
15     draw_square()
16
17 mainloop()
18
```

$

call the mainloop() function at the end so we can see the whole picture for more than a moment.

This is our strategy:

1. for each coordinate:
    0. lift the pen
    1. go back to the (0, 0) coordinate
    2. move to the current coordinate by using setheading and forward twice, once to move to the correct x location, and once to move to the correct y location.
    3. put down the pen
    4. draw the square



```
$ python draw_squares.py
```

There it is.

This is our strategy:

1. for each coordinate:
    0. lift the pen
    1. go back to the (0, 0) coordinate
    2. move to the current coordinate by using setheading and forward twice, once to move to the correct x location, and once to move to the correct y location.
    3. put down the pen
    4. draw the square

draw_squares.py | draw_a_square.py

```
1  from turtle import *
2  from draw_a_square import draw_square
3
4  coord_list = [(-100, -100), (100,
   100), (-100, 100), (100, -100)]
5
6  for coord in coord_list:
7      up()
8      home()
9      x, y = coord
10     setheading(270)
11     forward(x)
12     setheading(0)
13     forward(y)
14     down()
15     draw_square()
16
17 mainloop()
18
```

```
$
```

Now, what if we wanted to draw squares of different sizes? Or of different fill colors? This is where we use function parameters.

draw_squares.py | draw_a_square.py

```
1  from turtle import *
2
3  def draw_square():
4      for i in range(4):
5          forward(100)
6          left(90)
7
8  if __name__ == '__main__':
9      draw_square()
10
11     up()
12     forward(200)
13     down()
14
15     draw_square()
16
17     mainloop()
18
```

```
$
```

Now, what if we wanted to draw squares of different sizes? Or of different fill colors? This is where we use function parameters.

draw_squares.py | draw_a_square.py

```
from turtle import *

def draw_square(size):
    for i in range(4):
        forward(size)
        left(90)

if __name__ == '__main__':
    draw_square()

    up()
    forward(200)
    down()

    draw_square()

    mainloop()

```

```
$
```

I will introduce a size parameter to the draw_square function, and use it to determine how long each size of the square is.

draw_squares.py | draw_a_square.py

```
1  from turtle import *
2  from draw_a_square import draw_square
3
4  coord_list = [(-100, -100), (100,
•  100), (-100, 100), (100, -100)]
5
6  for coord in coord_list:
7      up()
8      home()
9      x, y = coord
10     setheading(270)
11     forward(x)
12     setheading(0)
13     forward(y)
14     down()
15     draw_square()
16
17 mainloop()
18
```

```
$ python draw_squares.py
Traceback (most recent call last):
  File "draw_squares.py", line 15, in <module>
    draw_square()
TypeError: draw_square() takes exactly 1 argument (0 given)
$
```

If I run the draw_squares.py program again now, I get this error.

draw_squares.py | draw_a_square.py

```
1  from turtle import *
2  from draw_a_square import draw_square
3
4  coord_list = [(-100, -100), (100,
•  100), (-100, 100), (100, -100)]
5
6  for coord in coord_list:
7      up()
8      home()
9      x, y = coord
10     setheading(270)
11     forward(x)
12     setheading(0)
13     forward(y)
14     down()
15     draw_square(120)
16
17 mainloop()
18
```

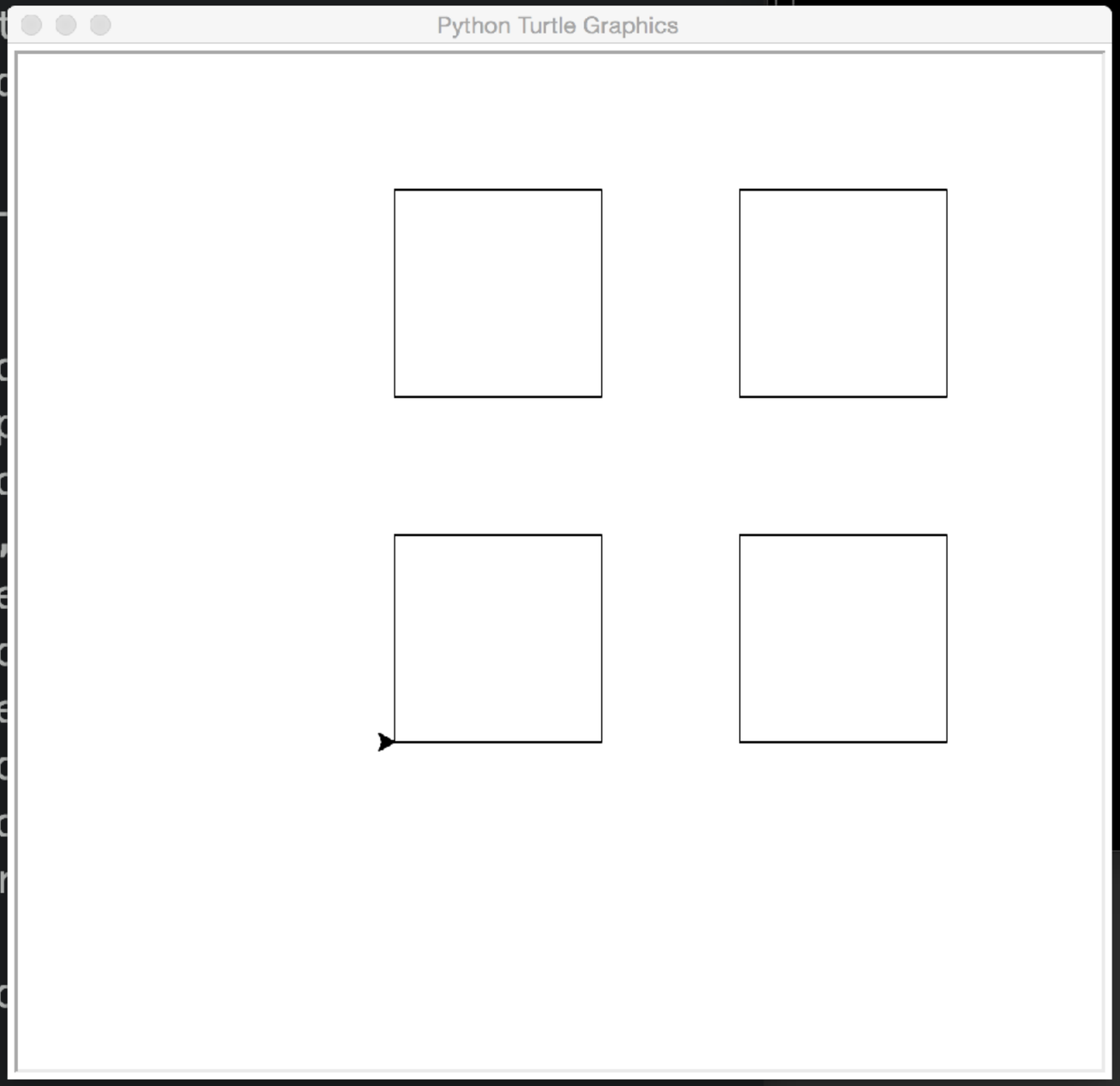
```
$ python draw_squares.py
Traceback (most recent call last):
  File "draw_squares.py", line 15, in <module>
    draw_square()
TypeError: draw_square() takes exactly 1 argument (0 given)
$
```

That's because I haven't provide an argument to the function - required by the function as the parameter: size. I will put one in.

draw_squares.py | draw_a_square.py

```
1   from t
2   from d
3
4   coord_
•   100),
5
6   for co
7       up
8       ho
9       x,
10      se
11      fo
12      se
13      fo
14      do
15      dr
16
17  mainlo
18
```

Python Turtle Graphics



```
$ python draw_squares.py
```

Re-run the program. And now the squares are slightly bigger.

draw_squares.py | draw_a_square.py

```
1  from turtle import *
2  from draw_a_square import draw_square
3
4  coord_list = [(-100, -100), (100,
   100), (-100, 100), (100, -100)]
5
6  for coord in coord_list:
7      up()
8      home()
9      x, y = coord
10     setheading(270)
11     forward(x)
12     setheading(0)
13     forward(y)
14     down()
15     draw_square(120)
16
17 mainloop()
18
```

```
$ python draw_squares.py
$
```

But what if I want each square to have a different size? I can group that information along with my existing tuples in the list.

draw_squares.py | draw_a_square.py

```
1  from turtle import *
2  from draw_a_square import draw_square
3
4  coord_list = [(-100, -100), (100,
   100), (-100, 100), (100, -100)]
5
6  for coord in coord_list:
7      up()
8      home()
9      x, y = coord
10     setheading(270)
11     forward(x)
12     setheading(0)
13     forward(y)
14     down()
15     draw_square(120)
16
17 mainloop()
18
```

```
$ python draw_squares.py
$
```

But what if I want each square to have a different size? I can group that information along with my existing tuples in the list.

draw_squares.py | draw_a_square.py

```
1  from turtle import *
2  from draw_a_square import draw_square
3
4  coord_list = [(-100, -100, 100),
•  (100, 100, 200), (-100, 100, 120),
•  (100, -100, 250)]
5
6  for coord in coord_list:
7      up()
8      home()
9      x, y = coord
10     setheading(270)
11     forward(x)
12     setheading(0)
13     forward(y)
14     down()
15     draw_square(120)
16
17 mainloop()
18
```

```
$ python draw_squares.py
$
```

Here, I have added a third value in each tuple of the list, representing the size of each square.

draw_squares.py | draw_a_square.py

```
1  from turtle import *
2  from draw_a_square import draw_square
3
4  coord_list = [(-100, -100, 100),
•  (100, 100, 200), (-100, 100, 120),
•  (100, -100, 250)]
5
6  for coord in coord_list:
7      up()
8      home()
9      x, y, size = coord
10     setheading(270)
11     forward(x)
12     setheading(0)
13     forward(y)
14     down()
15     draw_square(120)
16
17 mainloop()
18
```

```
$ python draw_squares.py
$
```

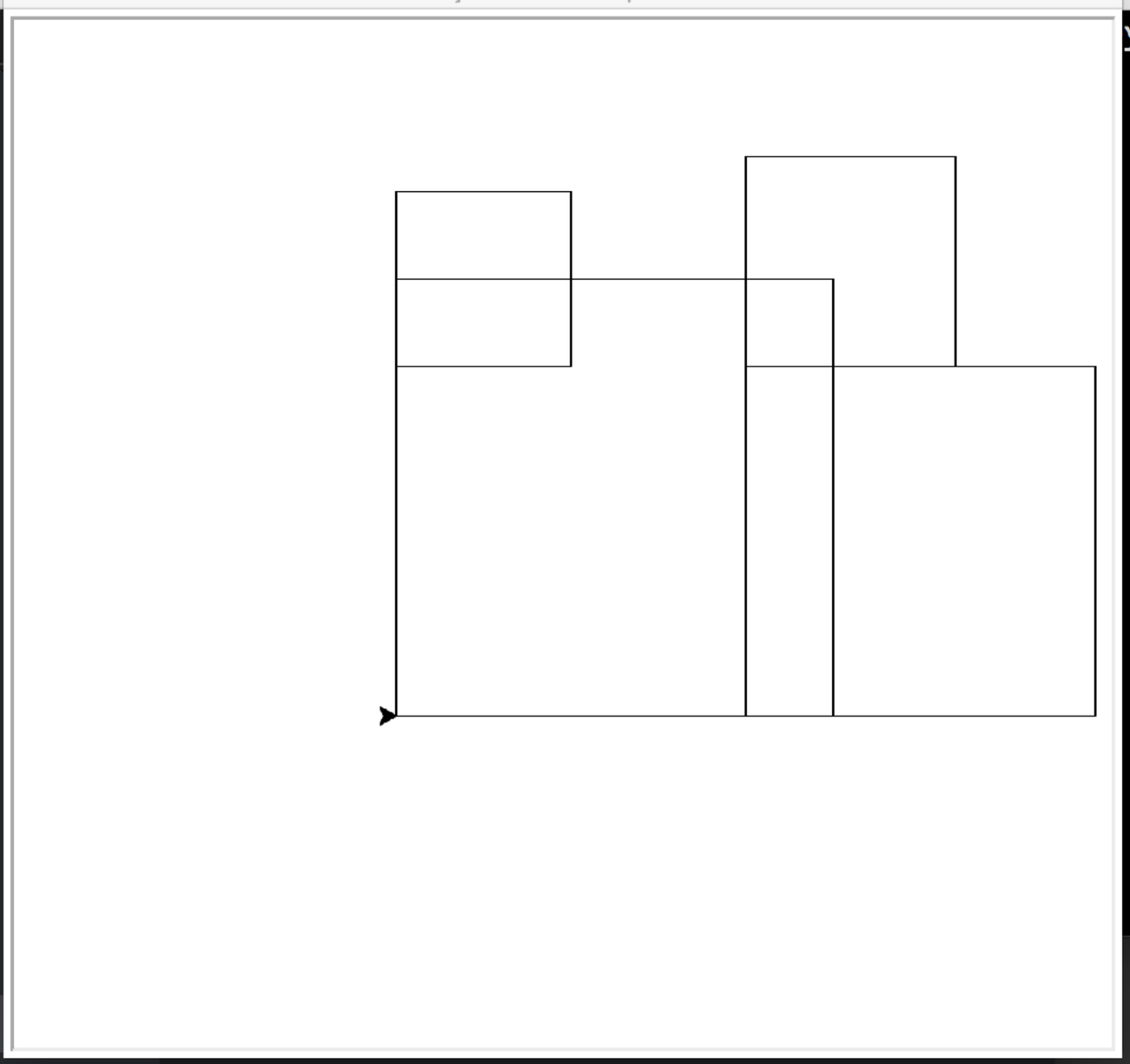
Then in the destructuring assignment I will assign that third value to a size variable.

draw_squares.py | draw_a_square.py

```
1  from turtle import *
2  from draw_a_square import draw_square
3
4  coord_list = [(-100, -100, 100),
•  (100, 100, 200), (-100, 100, 120),
•  (100, -100, 250)]
5
6  for coord in coord_list:
7      up()
8      home()
9      x, y, size = coord
10     setheading(270)
11     forward(x)
12     setheading(0)
13     forward(y)
14     down()
15     draw_square(size)
16
17 mainloop()
18
```

```
$ python draw_squares.py
$
```

Then I will substitute in that size to the draw_square function as the first argument.



```
ython draw_squares.py
```

```
16
17  mainloop()
18
```

Now I have different sized squares!