Exposure CS 2021 for CS1

Chapter 9 Section 1-6 Slides

Modular Programming: Simple Procedures, Graphics Programs and Creating Libraries

PowerPoint Presentation
created by:
Mr. John L. M. Schram
and Mr. Leon Schram
Authors of Exposure
Computer Science

Section 9.1

Introduction

Creating Tools

It is in the nature of people to create things that make our lives simpler and more enjoyable.

To that end, it is also in the nature of people to create *tools* to make this process easier and more efficient.

Program Subroutines

A *subroutine* is a series of programming commands that performs a specific task (like a *tool*).

Once a subroutine is created, it is no longer necessary to use all the individual commands.

Subroutines have many names in different programming languages.

For example, Java calls their subroutines *methods*.

In Python:
A subroutine that returns a value is called a *function*.
A subroutine that does not return a value is called a *procedure*.

Section 9.2

What is Modular Programming?

Modular Programming

Modular programming is the process of placing statements that achieve a common purpose into its own module or subroutine.

An old programming saying says it well

One Task, One Module

Section 9.3

Creating Simple Procedures

```
 1 # SimpleProcedures01.py
 2 # This program displays a simple mailing address.
 3 # It will be used to demonstrate how to divide a
 4 # program into multiple user-created procedures.
 5
 6
 7 print()
 8 print("Kathy Smith")
 9 print("7003 Orleans Court")
10 print("Kensington, MD 20795")
11
```

```
----jGRASP exec: python SimpleProcedures01.py


Kathy Smith
7003 Orleans Court
Kensington, MD 20795


----jGRASP: operation complete.
```

```
 1 # SimpleProcedures02.py
 2 # This program introduces user-created procedures.
 3 # The 3 program statements of the SimpleProcedures01.py
 4 # program are now divided into 3 user-created procedures.
 5 # Each procedure is simply called by using its name.
 6
 7
 8 ############################
 9 #  Procedure Definitions   #
10 ############################
11
12 def displayName():
13     print()
14     print("Kathy Smith")
15
16
17 def displayStreetAddress():
18     print("7003 Orleans Court")
19
20
21 def displayCityStateZip():
22     print("Kensington, MD 20795")
23
24
25
26 ############################
27 #  MAIN: Procedure Calls   #
28 ############################
29
30 displayName()
31 displayStreetAddress()
32 displayCityStateZip()
```

```
----jGRASP exec: pyth


Kathy Smith
7003 Orleans Court
Kensington, MD 20795


----jGRASP: operation
```

User-Created Procedure Format

A user-defined procedure requires:

- The Python keyword **def**
- A heading, which includes the procedure name
- A set of parentheses ( )
- Just like conditional statements, procedure headings end with a colon **:** .
- An indented body of program statements beneath the heading

```
def example():
    print("This is an example of a")
    print("user-defined procedure.")
```

Potential Procedure Problems

```
1  # SimpleProcedures03.py
2  # This program demonstrates that, as with
3  # control structures, in Python indenting
4  # subroutine commands is required.
5
6
7  ############################
8  #  Procedure Definitions   #
9  ############################
10
11 def displayName():
12 print()
13 print("Kathy Smith")
14
15
16 def displayStreetAddress():
17 print("7003 Orleans Court")
18
19
20 def displayCityStateZip():
21 print("Kensington, MD 20795")
22
23
24
25 ############################
26 #  MAIN: Procedure Calls   #
27 ############################
28
29 displayName()
30 displayStreetAddress()
31 displayCityStateZip()
```

```
1  # Simpl
2  # This
3  # contr
4  # subro
5
6
7  #######
8  #  Proc
9  #######
10
11 def dis
12 print()
13 print("
14
15
16 def displayStreetAddress():
17 print("7003 Orleans Court")
18
19
20 def displayCityStateZip():
21 print("Kensington, MD 20795")
22
23
24
25 ############################
26 #  MAIN: Procedure Calls  #
27 ############################
28
29 displayName()
30 displayStreetAddress()
31 displayCityStateZip()
```

```
----jGRASP exec: python SimpleProcedures03.py
 File "SimpleProcedures03.py", line 12
   print()
       ^
IndentationError: expected an indented block

----jGRASP wedge2: exit code for process is 1.
----jGRASP: operation complete.
```

Updated Indentation Rule

In most languages, indenting the program statements that are "controlled" by control structures *or part of a subroutine* is recommended.

In Python, both are required.

Python programs that do not use proper and consistent indentation will not execute.

```
1  # SimpleProcedures04.py
2  # This program demonstrates what happens
3  # when indentation is inconsistent.
4
5
6  ##############################
7  #  Procedure Definitions  #
8  ##############################
9
10 def displayName():
11     print()
12      print("Kathy Smith")
13
14
15 def displayStreetAddress():
16    print("7003 Orleans Court")
17
18
19 def displayCityStateZip():
20    print("Kensington, MD 20795")
21
22
23
24 ##############################
25 #  MAIN: Procedure Calls  #
26 ##############################
27
28 displayName()
29 displayStreetAddress()
30 displayCityStateZip()
```

```
1  # SimpleProcedures04.py
2  # This program demonstrates what happens
3  # when indentation is inconsistent.
4
5
6  ############################
7  #  Procedure Definitions   #
8  ############################
9
10 def displayName():
11     print()
12      print("Kathy Smith")
13
14   ----jGRASP exec: python SimpleProcedures04.py
15
16    File "SimpleProcedures04.py", line 12
17
18      print("Kathy Smith")
19      ^
20
21  IndentationError: unexpected indent
22
23
24   ----jGRASP wedge2: exit code for process is 1.
25
26   ----jGRASP: operation complete.
27
28 displayName()
29 displayStreetAddress()
30 displayCityStateZip()
```

```
 1 # SimpleProcedures05.py
 2 # "Invisibly Inconsistent Indentation Error"
 3 # You can indent with a <tab> or with some
 4 # spaces, but you need to be consistent.
 5
 6
 7 ##############################
 8 #  Procedure Definitions  #
 9 ##############################
10
11 def displayName():
12     print()                 # Indented with a <tab>
13     print("Kathy Smith") # Indented with 3 spaces
14
15
16 def displayStreetAddress():
17     print("7003 Orleans Court")
18
19
20 def displayCityStateZip():
21     print("Kensington, MD 20795")
22
23
24
25 ##############################
26 #  MAIN: Procedure Calls  #
27 ##############################
28
29 displayName()
30 displayStreetAddress()
31 displayCityStateZip()
```

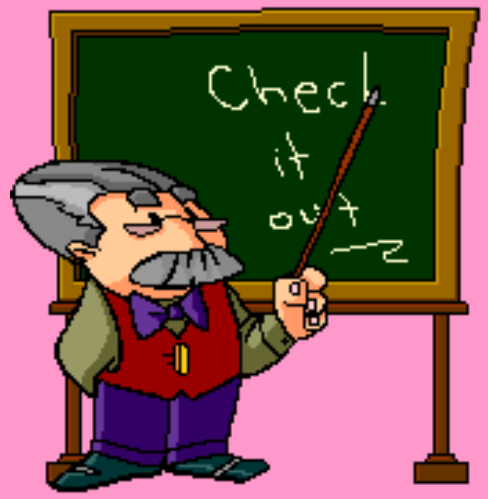
```
 1 # SimpleProcedures05.py
 2 # "Invisibly Inconsistent Indentation Error"
 3 # You can indent with a <tab> or with some
 4 # spaces, but you need to be consistent.
 5
 6
 7 ##############################
 8 #  Procedure Definitions  #
 9 ##############################
10
11 def displayName():
12     print()                 # Indented with a <tab>
13     print("Kathy Smith") # Indented with 3 spaces
```

```
----jGRASP exec: python SimpleProcedures05.py
 File "SimpleProcedures05.py", line 13
   print("Kathy Smith") # Indented with 3 spaces
                                                  ^
IndentationError: unindent does not match any outer indentation level

----jGRASP wedge2: exit code for process is 1.
----jGRASP: operation complete.
```

```
25 ##############################
26 #  MAIN: Procedure Calls  #
27 ##############################
28
29 displayName()
30 displayStreetAddress()
31 displayCityStateZip()
```

TRY THIS:

Click at the beginning of lines 12 and 13. Notice how the cursor appears different on line 12. This is because that line is indented with a <tab>.




```
11 def displayName():
12     print()                  # Indented with a <tab>
13     print("Kathy Smith") # Indented with 3 spaces
```

```
11 def displayName():
12     print()                  # Indented with a <tab>
13     print("Kathy Smith") # Indented with 3 spaces
```

```
 1 # SimpleProcedures06.py
 2 # This program shows that in Python,
 3 # the "Procedure Definitions" must
 4 # come before the "Procedure Calls".
 5
 6
 7 ##############################
 8 #  MAIN: Procedure Calls  #
 9 ##############################
10
11 displayName()
12 displayStreetAddress()
13 displayCityStateZip()
14
15
16
17 ##############################
18 #  Procedure Definitions  #
19 ##############################
20
21 def displayName():
22     print()
23     print("Kathy Smith")
24
25
26 def displayStreetAddress():
27     print("7003 Orleans Court")
28
29
30 def displayCityStateZip():
31     print("Kensington, MD 20795")
```

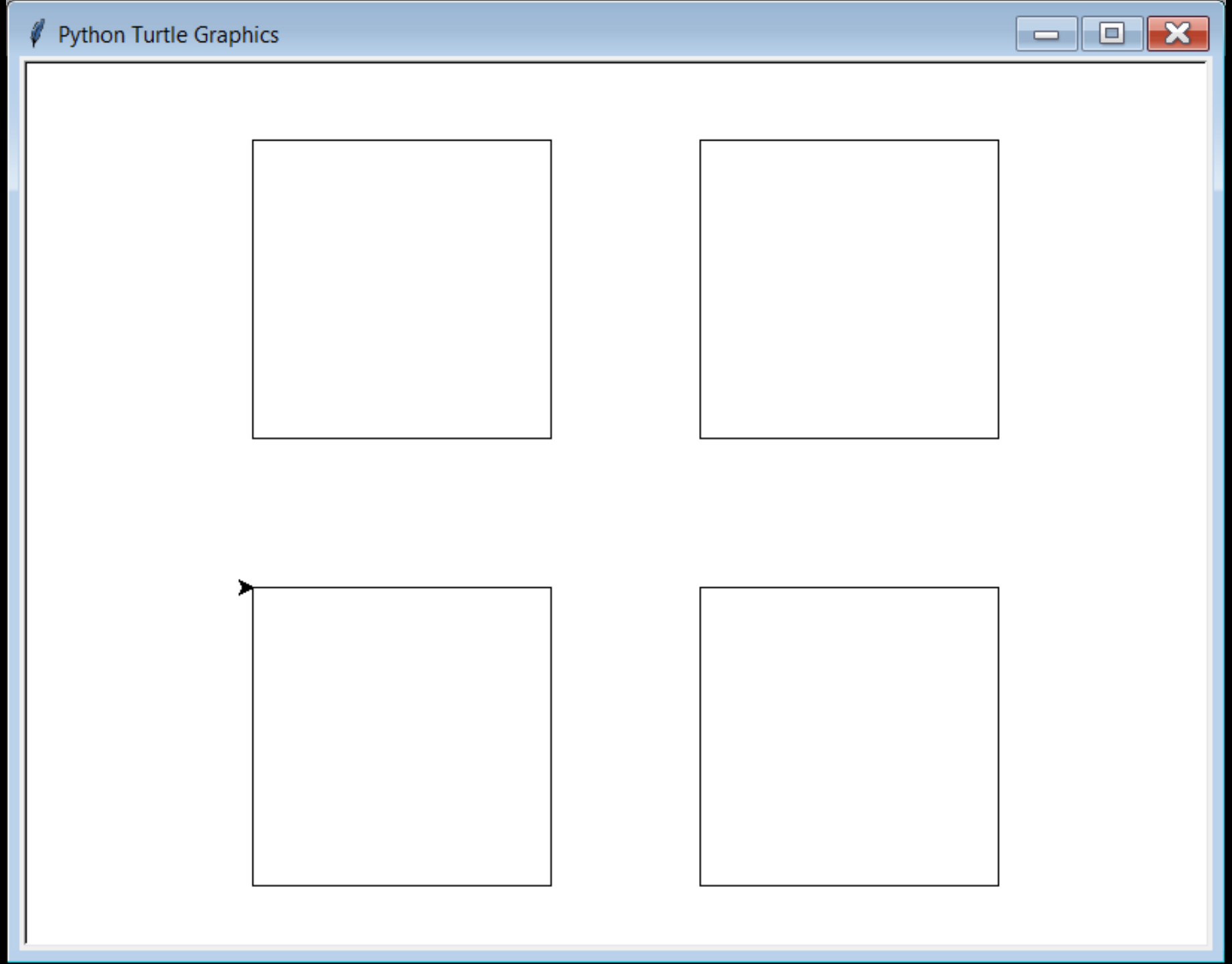
```
 1 # SimpleProced
 2 # This program
 3 # the "Procedu
 4 # come before
 5
 6
 7 ##############
 8 #  MAIN: Proce
 9 ##############
10
11 displayName()
12 displayStreetAddress()
13 displayCityStateZip()
14
15
16
17 ############################
18 #  Procedure Definitions  #
19 ############################
20
21 def displayName():
22     print()
23     print("Kathy Smith")
24
25
26 def displayStreetAddress():
27     print("7003 Orleans Court")
28
29
30 def displayCityStateZip():
31     print("Kensington, MD 20795")
```

```
 ----jGRASP exec: python SimpleProcedures06.py
Traceback (most recent call last):
File "SimpleProcedures06.py", line 11, in <module>
    displayName()
NameError: name 'displayName' is not defined

 ----jGRASP wedge2: exit code for process is 1.
 ----jGRASP: operation complete.
```

Section 9.4

Graphics Programs with Procedures

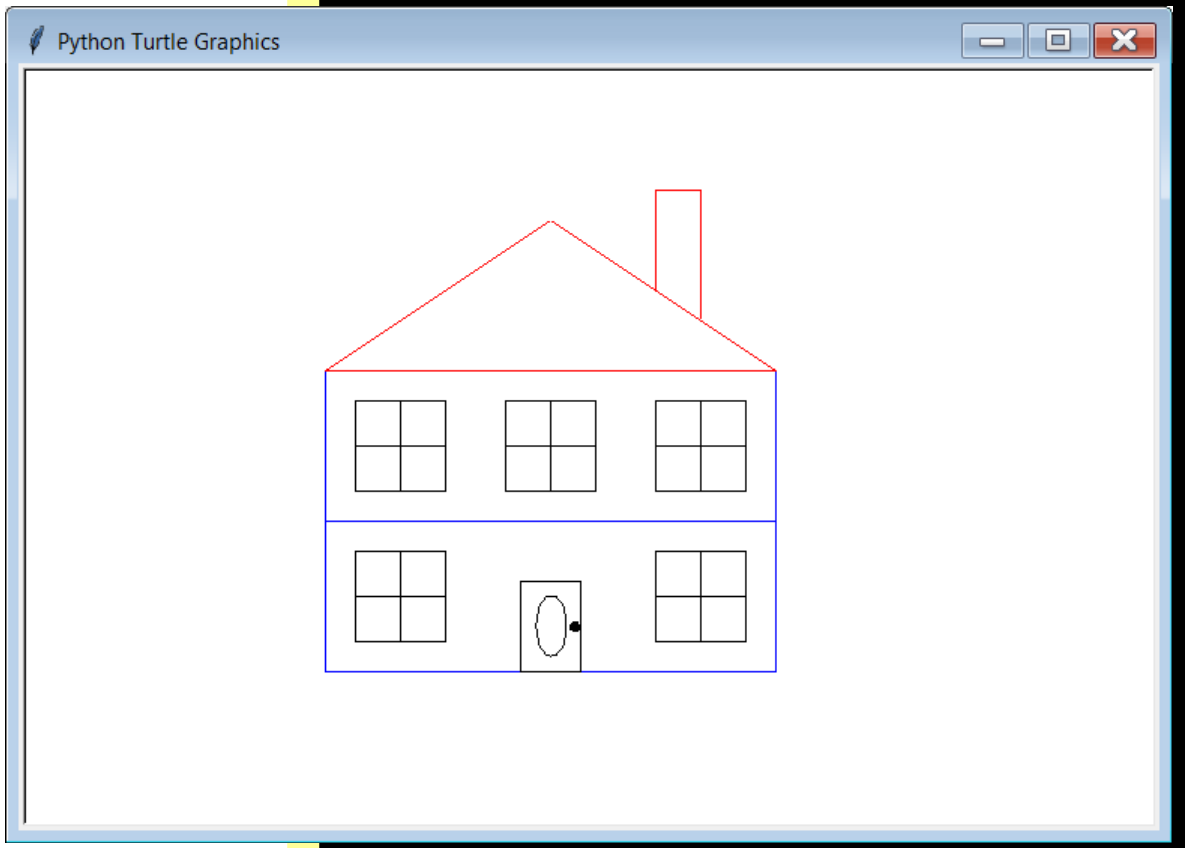
```
# GraphicsProcedures01.py
# This is a copy of TurtleGraphics03.py from Chapter 5.
# Note the repetitive code needed to draw the 4 squares.


from turtle import *

setup(800,600)

penup()
left(135)      # face North-West
forward(350)   # Move to top-left corner
right(135)     # face East again
pendown()

# draw square
forward(200)
right(90)
forward(200)
right(90)
forward(200)
right(90)
forward(200)
right(90)

penup()
forward(300)   # Move to top-right corner
pendown()

# draw square
forward(200)
right(90)
forward(200)
right(90)
forward(200)
right(90)
forward(200)
right(90)

penup()
right(90)      # face South
forward(300)   # Move to bottom-right corner
left(90)       # face East again
pendown()

# draw square
forward(200)
right(90)
forward(200)
right(90)
forward(200)
right(90)
forward(200)
right(90)

penup()
backward(300) # Move to bottom-left corner
pendown()

# draw square
forward(200)
right(90)
forward(200)
right(90)
forward(200)
right(90)
forward(200)
right(90)

update()
done()
```


Python Turtle Graphics

```
 1 # GraphicsProcedures02.py
 2 # This program has the same output as the previous
 3 # program but is more efficient because a special
 4 # <drawSquare> procedure has been created which
 5 # eliminates the repetitive code.
 6
 7
 8 from turtle import *
 9
10
11 def drawSquare():
12     pendown()
13     forward(200)
14     right(90)
15     forward(200)
16     right(90)
17     forward(200)
18     right(90)
19     forward(200)
20     right(90)
21     penup()
22
23
24
```

```
25 ##########
26 #  MAIN  #
27 ##########
28
29 setup(800,600)
30
31 penup()
32 left(135)      # face North-West
33 forward(350)   # Move to top-left corner
34 right(135)     # face East again
35
36 drawSquare()
37
38 forward(300)  # Move to top-right corner
39
40 drawSquare()
41
42 right(90)      # face South
43 forward(300)   # Move to bottom-right corner
44 left(90)       # face East again
45
46 drawSquare()
47
48 backward(300) # Move to bottom-left corner
49
50 drawSquare()
51
52 update()
53 done()
```

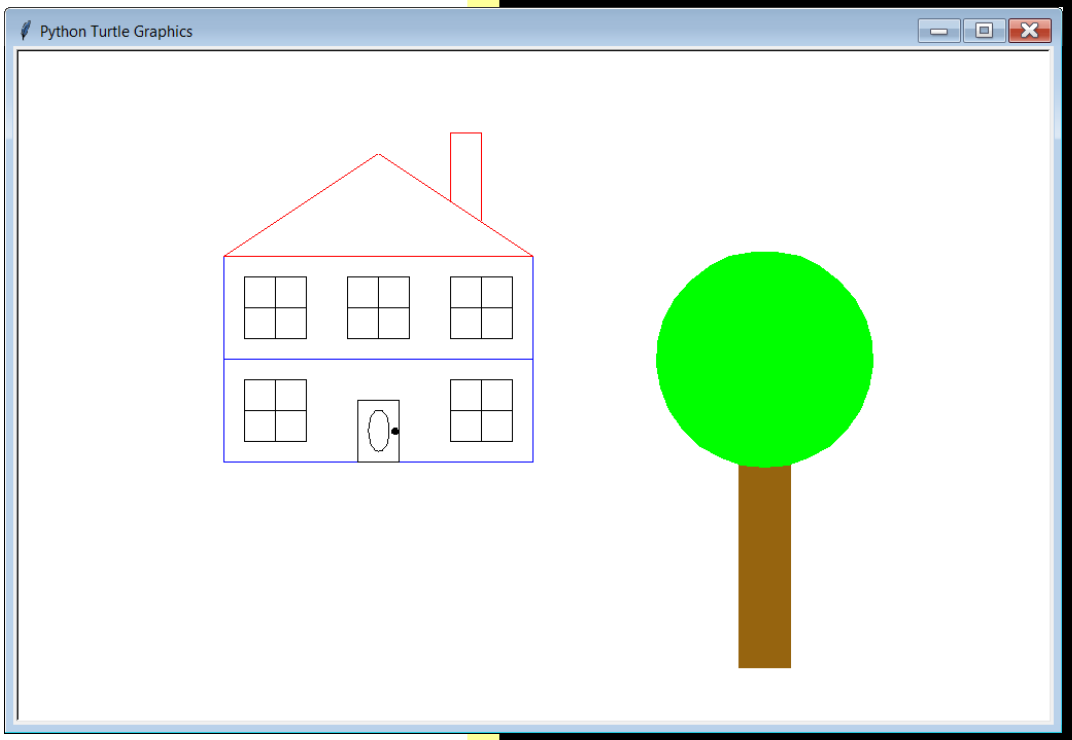
```
1  # GraphicsProcedures03.py
2  # This program demonstrates drawing a house.
3  # Imagine that you want to change the appearance
4  # of the chimney.  What code would you change?
5
6
7  from Graphics import *
8
9
10 beginGrfx(750,500)
11
12 setColor("blue")
13 drawRectangle(200,200,500,300)
14 drawRectangle(200,300,500,400)
15 setColor("red")
16 drawLine(200,200,350,100)
17 drawLine(500,200,350,100)
18 drawLine(200,200,500,200)
19 drawLine(420,146,420,80)
20 drawLine(420,80,450,80)
21 drawLine(450,80,450,166)
22 setColor("black")
23 drawRectangle(330,340,370,400)
24 drawOval(350,370,10,20)
25 fillCircle(366,370,3)
26 drawRectangle(220,220,280,280)
27 drawLine(220,250,280,250)
28 drawLine(250,220,250,280)
29 drawRectangle(420,220,480,280)
30 drawLine(420,250,480,250)
31 drawLine(450,220,450,280)
32 drawRectangle(320,220,380,280)
33 drawLine(320,250,380,250)
34 drawLine(350,220,350,280)
35 drawRectangle(220,320,280,380)
36 drawLine(220,350,280,350)
37 drawLine(250,320,250,380)
38 drawRectangle(420,320,480,380)
39 drawLine(420,350,480,350)
40 drawLine(450,320,450,380)
41
42 endGrfx()
```



NOTE: This is <u>NOT</u>
***Good Program Design*.**

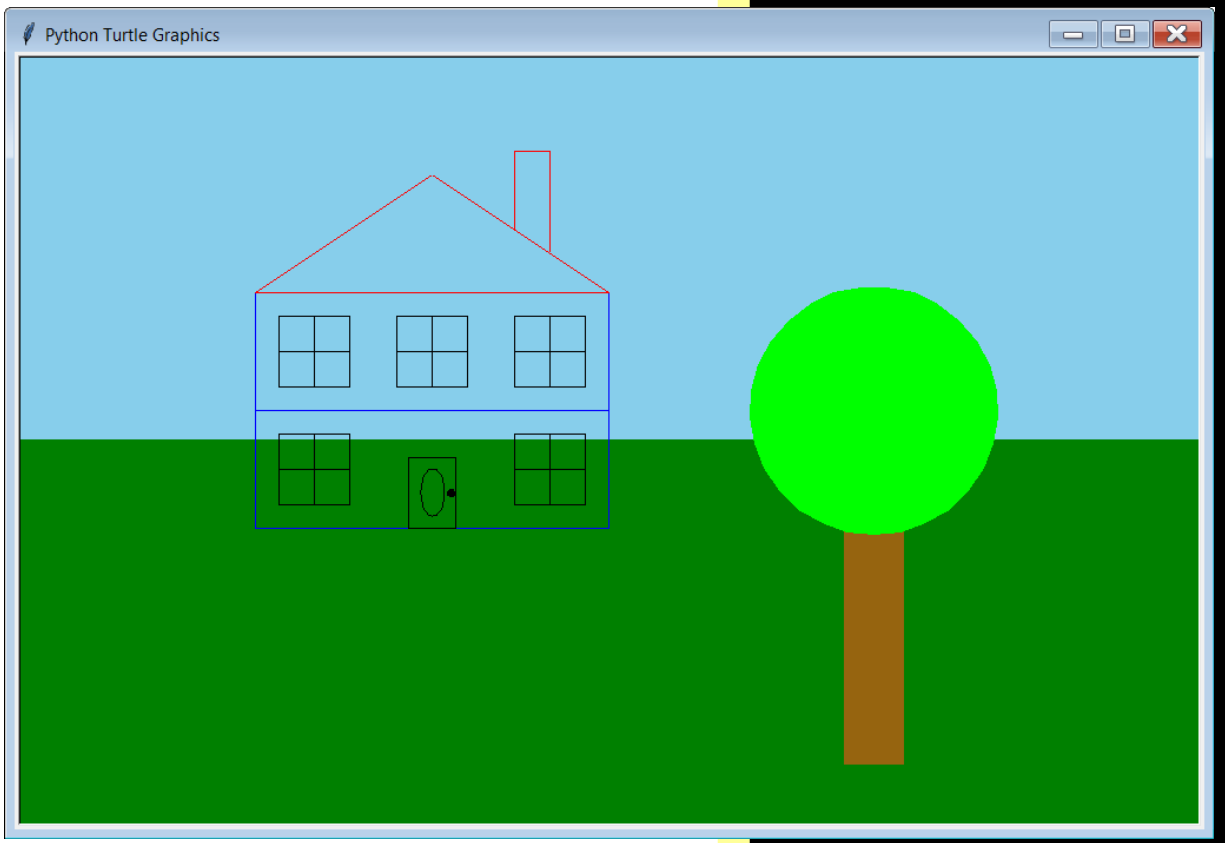
```
# GraphicsProcedures04.py
# This program divides the house up into several
# different procedures.  Imagine again that you
# want to change the appearance of the chimney.
# Finding the appropriate code is much easier now.
# NOTE: The "MAIN" section of the program now looks
#       like an outline.


from Graphics import *


def drawFloors():
    setColor("blue")
    drawRectangle(200,200,500,300)
    drawRectangle(200,300,500,400)


def drawRoof():
    setColor("red")
    drawLine(200,200,350,100)
    drawLine(500,200,350,100)
    drawLine(200,200,500,200)


def drawChimney():
    drawLine(420,146,420,80)
    drawLine(420,80,450,80)
    drawLine(450,80,450,166)


def drawDoor():
    setColor("black")
    drawRectangle(330,340,370,400)
    drawOval(350,370,10,20)
    fillCircle(366,370,3)

def drawWindows():
    drawRectangle(220,220,280,280)
    drawLine(220,250,280,250)
    drawLine(250,220,250,280)
    drawRectangle(420,220,480,280)
    drawLine(420,250,480,250)
    drawLine(450,220,450,280)
    drawRectangle(320,220,380,280)
    drawLine(320,250,380,250)
    drawLine(350,220,350,280)
    drawRectangle(220,320,280,380)
    drawLine(220,350,280,350)
    drawLine(250,320,250,380)
    drawRectangle(420,320,480,380)
    drawLine(420,350,480,350)
    drawLine(450,320,450,380)



##########
#  MAIN  #
##########

beginGrfx(750,500)

drawFloors()
drawRoof()
drawChimney()
drawDoor()
drawWindows()

endGrfx()
```

```
 1 # GraphicsProcedures05.py
 2 # This program adds 2 more procedures for drawing a tree.
 3
 4
 5 from Graphics import *
 6
 7
 8 # House procedures
   (same as before)

54 # Tree procedures
55
56 def drawTrunk():
57     setColor("brown")
58     fillRectangle(700,400,750,600)
59
60
61 def drawLeaves():
62     setColor("green")
63     fillCircle(725,300,105)
64
65
66
67 ##########
68 #  MAIN  #
69 ##########
70
71 beginGrfx(1000,650)
72
73 drawFloors()
74 drawRoof()
75 drawChimney()
76 drawDoor()
77 drawWindows()
78
79 drawTrunk()
80 drawLeaves()
81
82 endGrfx()
```

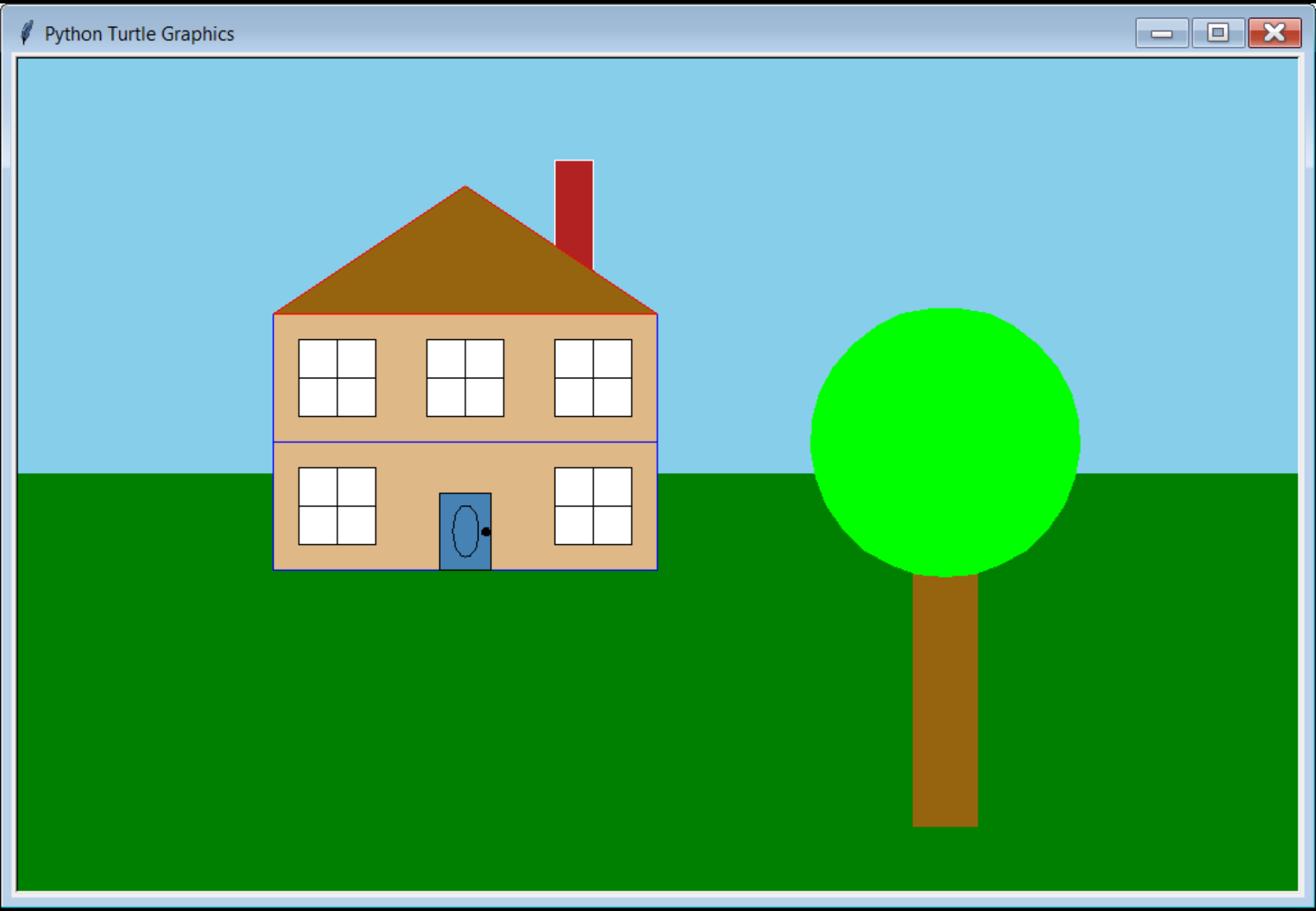


```
1  # GraphicsProcedures06.py
2  # This program adds 2 more procedures for drawing a
3  # background.  When you run the program, all you see is
4  # the background.  This is because the background was
5  # drawn after, and therefore on top of, everything else.
6
7
8  from Graphics import *
9
10
11 # Background procedures
12
13 def drawSky():
14     setColor("sky blue")
15     fillRectangle(0,0,1000,325)
16
17
18 def drawGrass():
19     setColor("dark green")
20     fillRectangle(0,325,1000,650)
21
22 # House and Tree procedures
   (same as before)

82 ##########
83 #  MAIN  #
84 ##########
85
86 beginGrfx(1000,650)
87
88 drawFloors()
89 drawRoof()
90 drawChimney()
91 drawDoor()
92 drawWindows()
93 drawTrunk()
94 drawLeaves()
95
96 drawSky()
97 drawGrass()
98
99 endGrfx()
```

```
 1 # GraphicsProcedures07.py
 2 # This program fixes the issue of the previous
 3 # program by drawing the background first.
 4 # The house does not look right because we
 5 # can see the background through it.
 6 # Unlike the tree, the house is not solid.
 7
 8 # Background, House and Tree procedures
   (same as before)


82
83 ##########
84 #  MAIN  #
85 ##########
86
87 beginGrfx(1000,650)
88
89 drawSky()
90 drawGrass()
91
92 drawFloors()
93 drawRoof()
94 drawChimney()
95 drawDoor()
96 drawWindows()
97 drawTrunk()
98 drawLeaves()
99
100 endGrfx()
```



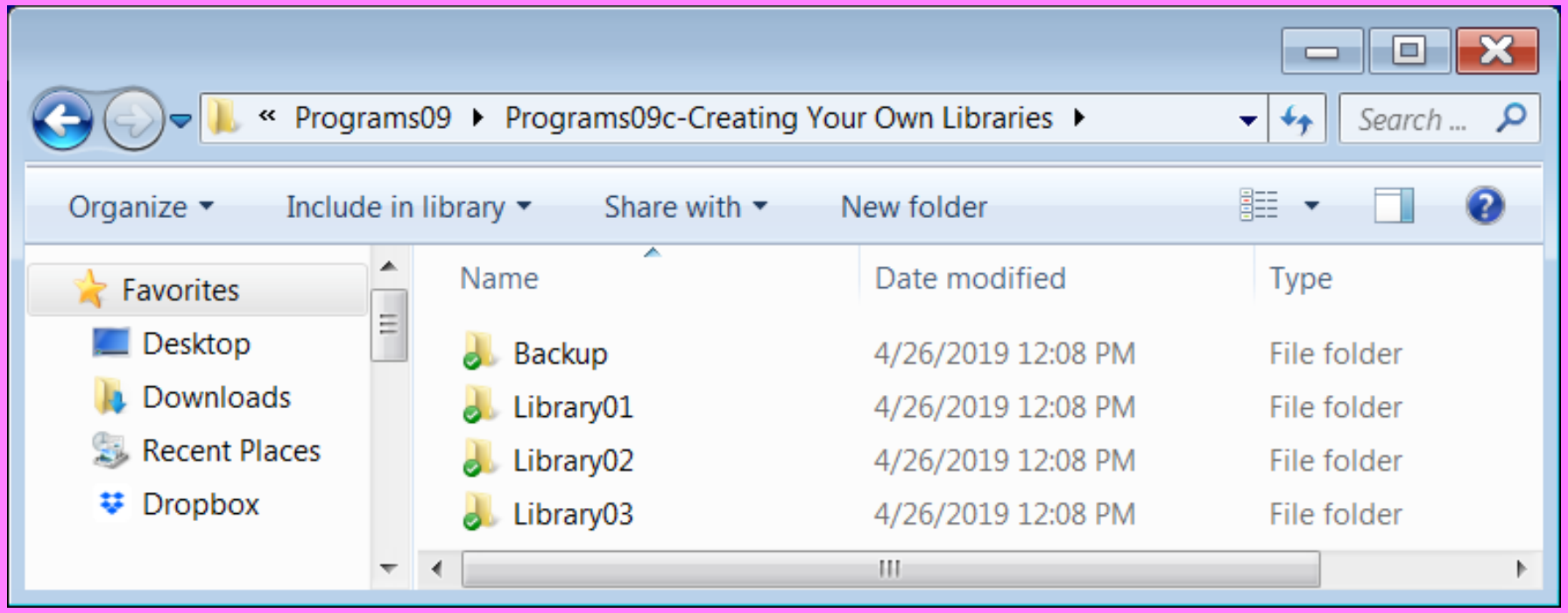
```
# GraphicsProcedures08.py
# This program makes the house solid by replacing
# several "draw" commands with "fill" commands.

# Background and Tree procedures
(same as before)


# House procedures

def drawFloors():
    setColor("burlywood")
    fillRectangle(200,200,500,400)
    setColor("blue")
    drawRectangle(200,200,500,300)
    drawRectangle(200,300,500,400)

def drawRoof():
    setColor("brown")
    fillPolygon([200,200,350,100,500,200])
    setColor("red")
    drawLine(200,200,350,100)
    drawLine(500,200,350,100)
    drawLine(200,200,500,200)

def drawChimney():
    setColor("firebrick")
    fillPolygon([420,146,420,80,450,80,450,166])
    setColor("white")
    drawLine(420,146,420,80)
    drawLine(420,80,450,80)
    drawLine(450,80,450,166)

def drawDoor():
    setColor("steel blue")
    fillRectangle(330,340,370,400)
    setColor("black")
    drawRectangle(330,340,370,400)
    drawOval(350,370,10,20)
    fillCircle(366,370,3)

def drawWindows():
    setColor("white")
    fillRectangle(220,220,280,280)
    fillRectangle(420,220,480,280)
    fillRectangle(320,220,380,280)
    fillRectangle(220,320,280,380)
    fillRectangle(420,320,480,380)
    setColor("black")
    drawRectangle(220,220,280,280)
    drawLine(220,250,280,250)
    drawLine(250,220,250,280)
    drawRectangle(420,220,480,280)
    drawLine(420,250,480,250)
    drawLine(450,220,450,280)
    drawRectangle(320,220,380,280)
    drawLine(320,250,380,250)
    drawLine(350,220,350,280)
    drawRectangle(220,320,280,380)
    drawLine(220,350,280,350)
    drawLine(250,320,250,380)
    drawRectangle(420,320,480,380)
    drawLine(420,350,480,350)
    drawLine(450,320,450,380)



##########
#  MAIN  #
##########

beginGrfx(1000,650)

drawSky()
drawGrass()
drawFloors()
drawRoof()
drawChimney()
drawDoor()
drawWindows()
drawTrunk()
drawLeaves()

endGrfx()
```


Python Turtle Graphics

Section 9.5

Creating Your Own Libraries

When you open the **Programs09c-Creating Your Own Libraries** folder, you will see that it is different from the other subfolders in **Programs09**. Besides **Backup**, this subfolder simply contains 3 other subfolders:



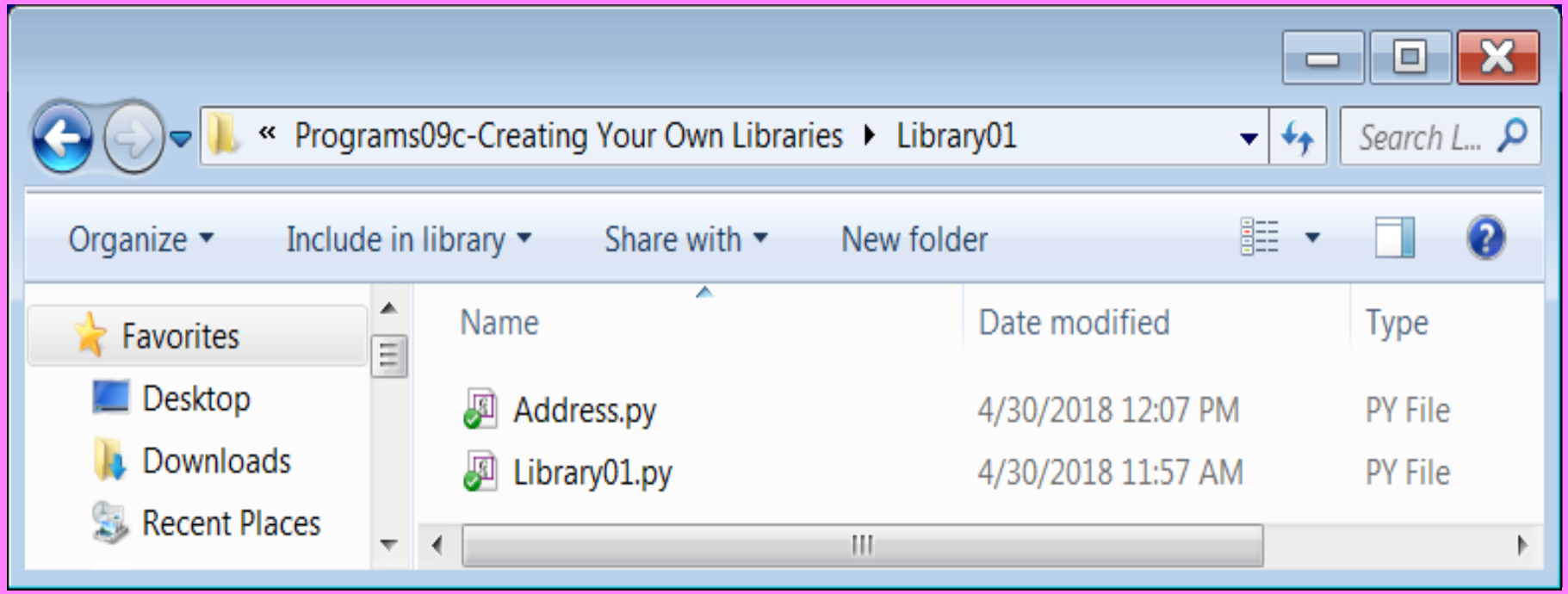
Since this section is on creating your own libraries, it means that each program example will actually consist of multiple files.

NOTE: Whenever a program consists of more than one file, all of the necessary files for that program should be placed into a single folder.

ALSO: Whenever a program consists of more than one file, the file that contains the "**MAIN**" section of the program is called the "Driving File". It is this file that needs to be executed.

Subfolder **Library01** has 2 files: **Library01.py** and **Address.py**.

The *driving file* is **Library01.py** so we will load that file first.



```
1 # Library01.py
2 # This program has the same output as SimpleProcedures02.py
3 # The difference is all of the procedures are now in a
4 # user-created library called <Address> which is located
5 # in the file Address.py
6
7
8 from Address import *
9
10
11 displayName()
12 displayStreetAddress()
13 displayCityStateZip()
14
```

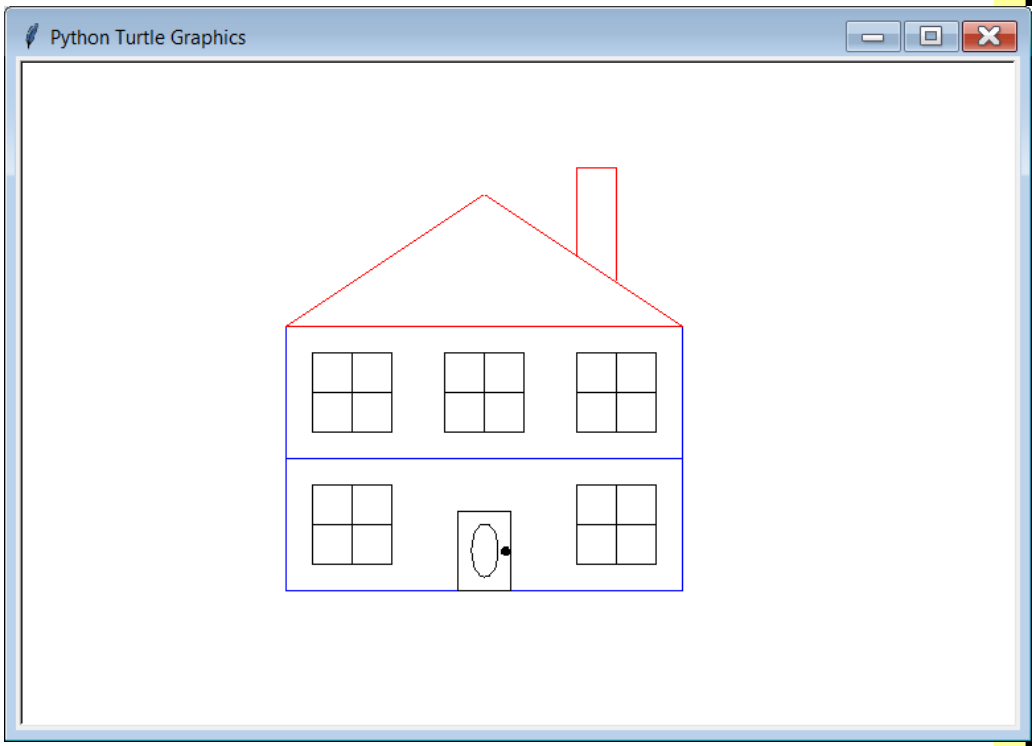
```
1  # Library01.py
2  # This program has the same output as SimpleProcedures02.py
3  # The difference is all of the procedures are now in a
4  # user-created library called <Address> which is located
5  # in the file Address.py
6
7
8  from Address import *
9
10
11 displayName()
12 displayStreetAddress()
13 displayCityStateZip()
14
```

```
----jGRASP exec: pyth

Kathy Smith
7003 Orleans Court
Kensington, MD 20795

----jGRASP: operation
```

How does this work without the three procedure definitions?

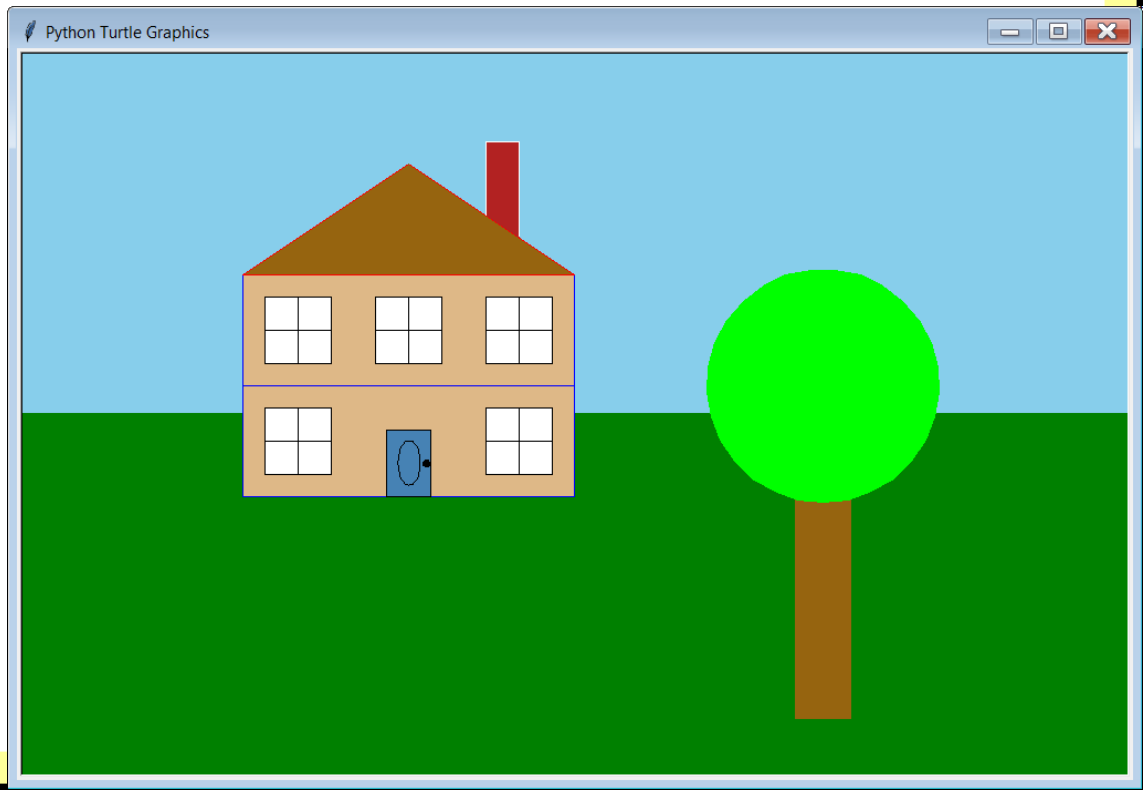
```
1  # Address.py
2  # This is the <Address> library used by Library01.py
3  # NOTE: When creating a library, you can click "Run"
4  # to see if there are any syntax errors.  Keep in mind,
5  # the program will not actually execute because it has
6  # no "MAIN" section and is not the "Driving File".
7
8
9  def displayName():
10     print()
11     print("Kathy Smith")
12
13
14 def displayStreetAddress():
15     print("7003 Orleans Court")
16
17
18 def displayCityStateZip():
19     print("Kensington, MD 20795")
20
```

```
1  # Library02.py
2  # This program has the same output as GraphicsProcedures04.py
3  # The difference is all of the procedures are now in a
4  # user-created library called <House> which is located
5  # in the file House.py
6
7  # NOTE: Technically, we do not need to import <Graphics>
8  # because it is already imported by the <House> library.
9
10
11 from Graphics import *
12 from House import *
13
14
15 beginGrfx(750,500)
16
17 drawFloors()
18 drawRoof()
19 drawChimney()
20 drawDoor()
21 drawWindows()
22
23 endGrfx()
```



```python
# House.py
# This is the <House> library used by Library02.py

from Graphics import *

def drawFloors():
    setColor("blue")
    drawRectangle(200,200,500,300)
    drawRectangle(200,300,500,400)

def drawRoof():
    setColor("red")
    drawLine(200,200,350,100)
    drawLine(500,200,350,100)
    drawLine(200,200,500,200)

def drawChimney():
    drawLine(420,146,420,80)
    drawLine(420,80,450,80)
    drawLine(450,80,450,166)

def drawDoor():
    setColor("black")
    drawRectangle(330,340,370,400)
    drawOval(350,370,10,20)
    fillCircle(366,370,3)

def drawWindows():
    drawRectangle(220,220,280,280)
    drawLine(220,250,280,250)
    drawLine(250,220,250,280)
    drawRectangle(420,220,480,280)
    drawLine(420,250,480,250)
    drawLine(450,220,450,280)
    drawRectangle(320,220,380,280)
    drawLine(320,250,380,250)
    drawLine(350,220,350,280)
    drawRectangle(220,320,280,380)
    drawLine(220,350,280,350)
    drawLine(250,320,250,380)
    drawRectangle(420,320,480,380)
    drawLine(420,350,480,350)
    drawLine(450,320,450,380)
```

```
1  # Library03.py
2  # This program has the same output as GraphicsProcedures08.py
3  # It demonstrates that multiple user-created libraries can
4  # be imported from the same file.  Breaking up a program in
5  # this manner makes things more organized and less cluttered.
6
7
8  from Graphics import *
9  from House import *
10 from Tree import *
11 from Background import *
12
13
14 beginGrfx(1000,650)
15
16 drawSky()
17 drawGrass()
18 drawFloors()
19 drawRoof()
20 drawChimney()
21 drawDoor()
22 drawWindows()
23 drawTrunk()
24 drawLeaves()
25
26 endGrfx()
```



```python
# House.py
# This is the <House> library used by Library03.py


from Graphics import *


def drawFloors():
    setColor("burlywood")
    fillRectangle(200,200,500,400)
    setColor("blue")
    drawRectangle(200,200,500,300)
    drawRectangle(200,300,500,400)


def drawRoof():
    setColor("brown")
    fillPolygon([200,200,350,100,500,200])
    setColor("red")
    drawLine(200,200,350,100)
    drawLine(500,200,350,100)
    drawLine(200,200,500,200)


def drawChimney():
    setColor("firebrick")
    fillPolygon([420,146,420,80,450,80,450,166])
    setColor("white")
    drawLine(420,146,420,80)
    drawLine(420,80,450,80)
    drawLine(450,80,450,166)


def drawDoor():
    setColor("steel blue")
    fillRectangle(330,340,370,400)
    setColor("black")
    drawRectangle(330,340,370,400)
    drawOval(350,370,10,20)
    fillCircle(366,370,3)


def drawWindows():
    setColor("white")
    fillRectangle(220,220,280,280)
    fillRectangle(420,220,480,280)
    fillRectangle(320,220,380,280)
    fillRectangle(220,320,280,380)
    fillRectangle(420,320,480,380)
    setColor("black")
    drawRectangle(220,220,280,280)
    drawLine(220,250,280,250)
    drawLine(250,220,250,280)
    drawRectangle(420,220,480,280)
    drawLine(420,250,480,250)
    drawLine(450,220,450,280)
    drawRectangle(320,220,380,280)
    drawLine(320,250,380,250)
    drawLine(350,220,350,280)
    drawRectangle(220,320,280,380)
    drawLine(220,350,280,350)
    drawLine(250,320,250,380)
    drawRectangle(420,320,480,380)
    drawLine(420,350,480,350)
    drawLine(450,320,450,380)
```

```
 1 # Background.py
 2 # This is the <Background> library
used by Library03.py
 3
 4
 5 from Graphics import *
 6
 7
 8 def drawSky():
 9     setColor("sky blue")
10     fillRectangle(0,0,1000,325)
11
12
13 def drawGrass():
14     setColor("dark green")
15     fillRectangle(0,325,1000,650)
16
```

```
 1 # Tree.py
 2 # This is the <Tree> library
used by Library03.py
 3
 4
 5 from Graphics import *
 6
 7
 8 def drawTrunk():
 9     setColor("brown")
10     fillRectangle(700,400,750,600)
11
12
13 def drawLeaves():
14     setColor("green")
15     fillCircle(725,300,105)
16
```

Program Note

While it is proper programming style to put subroutines of a common nature in their own file, thereby creating a *library*, most program examples in this textbook will continue to put all subroutines in the same file for simplicity.

Some Program Design Notes

Programs should not be written by placing all the program statements in the "**MAIN**" section of the program.

Program statements that perform a specific purpose should be placed inside their own module.

These modules are called *procedures* or *functions* in Python.

Good Program Design continues by placing subroutines of a common nature in a separate file, which creates a *library* that can be imported by multiple programs.

Section 9.6

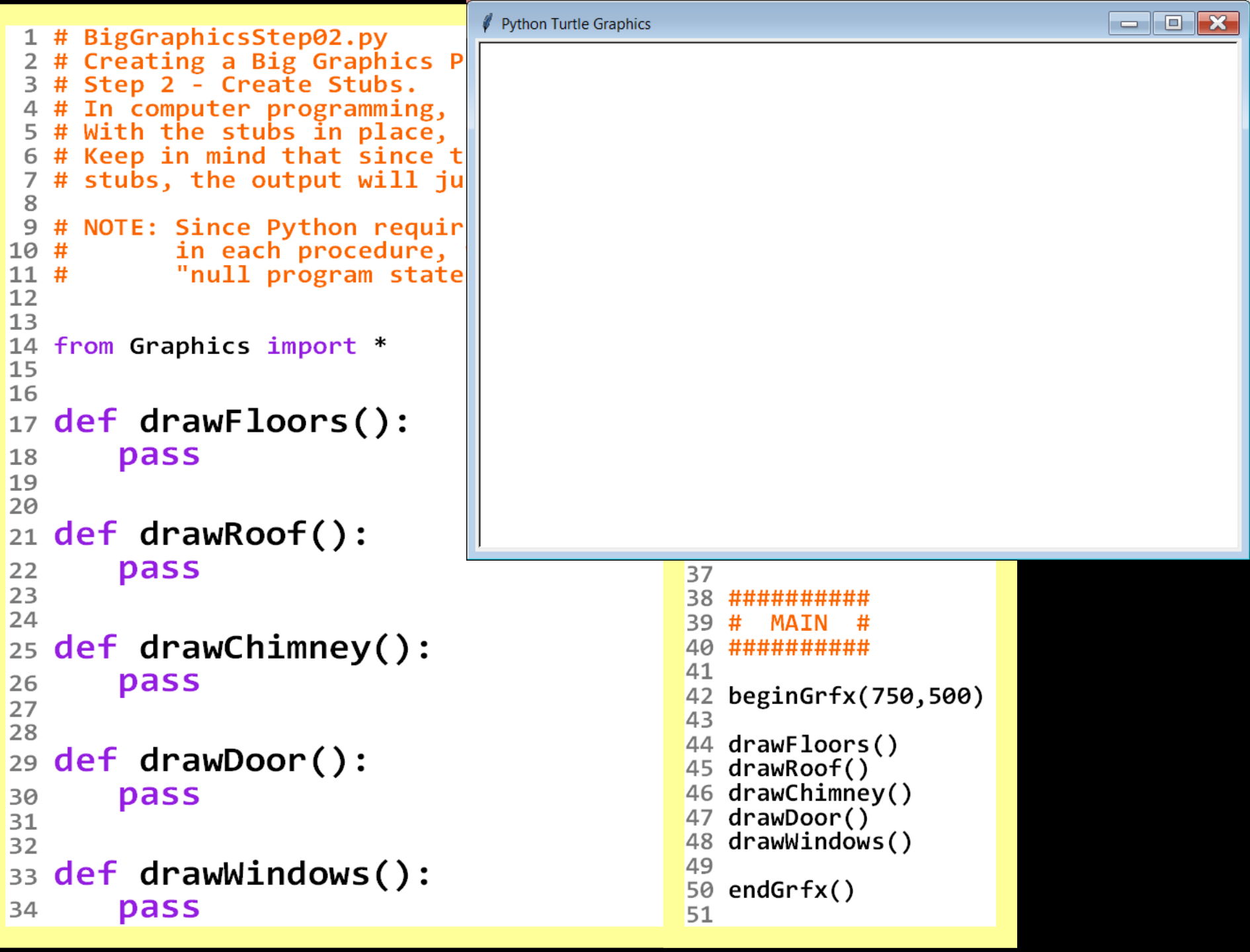
Creating a Big Graphics Program Step-By-Step

```
 1 # BigGraphicsStep01.py
 2 # Creating a Big Graphics Program
 3 # Step 1 - Create the "MAIN" section of the program.
 4 # This should simply be a list of procedure calls
 5 # and look like an outline.
 6
 7 # NOTE: This first step will not execute because
 8 #       the program is attempting to call procedures
 9 #       which do not exist yet.
10
11
12 from Graphics import *
13
14
15 ##########
16 #  MAIN  #
17 ##########
18
19 beginGrfx(750,500)
20
21 drawFloors()
22 drawRoof()
23 drawChimney()
24 drawDoor()
25 drawWindows()
26
27 endGrfx()
```

```
 1 # BigGraphicsStep01.py
 2 # Creating a Big Graphics Program
 3 # Step 1 - Create the "MAIN" section of the program.
 4 # This should simply be a list of procedure calls
 5 # and look like an outline.
 6
 7 # NOTE: This first step will not execute because
 8 #       the program is attempting to call procedures
 9 #       which do not exist yet.
10
11
12 from Graphics import *
13
14
15 ##########
16 #  MAIN  #
17 ##########
18
19 beginGrfx(750,500)
20
21 drawFloors()
22 drawRoof()
23 drawChimney()
24 drawDoor()
25 drawWindows()
26
27 endGrfx()
```

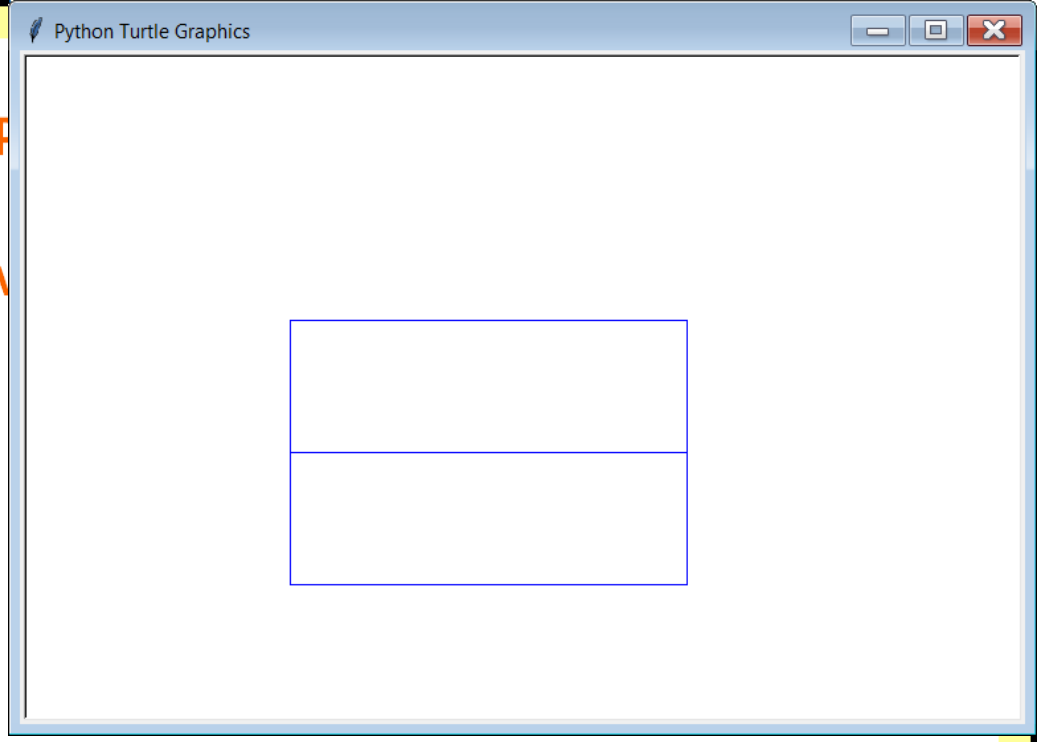
```
----jGRASP exec: python BigGraphicsStep01.py
Traceback (most recent call last):
  File "BigGraphicsStep01.py", line 21, in <module>
    drawFloors()
NameError: name 'drawFloors' is not defined


----jGRASP wedge2: exit code for process is 1.
----jGRASP: operation complete.
```

```
# BigGraphicsStep02.py
# Creating a Big Graphics Program
# Step 2 - Create Stubs.
# In computer programming, a "stub" is an empty procedure.
# With the stubs in place, the program can now execute.
# Keep in mind that since the program contains nothing but
# stubs, the output will just be a blank white window.

# NOTE: Since Python requires at least one program command
#       in each procedure, we will use <pass>.  This is a
#       "null program statement" which simply does nothing.


from Graphics import *


def drawFloors():
    pass


def drawRoof():
    pass


def drawChimney():
    pass


def drawDoor():
    pass


def drawWindows():
    pass



##########
#  MAIN  #
##########

beginGrfx(750,500)

drawFloors()
drawRoof()
drawChimney()
drawDoor()
drawWindows()

endGrfx()
```


Python Turtle Graphics
1 # BigGraphicsStep02.py
2 # Creating a Big Graphics P
3 # Step 2 - Create Stubs.
4 # In computer programming,
5 # With the stubs in place,
6 # Keep in mind that since t
7 # stubs, the output will ju
8
9 # NOTE: Since Python requir
10 # in each procedure,
11 # "null program state
12
13
14 from Graphics import *
15
16
17 def drawFloors():
18 pass
19
20
21 def drawRoof():
22 pass
23
24
25 def drawChimney():
26 pass
27
28
29 def drawDoor():
30 pass
31
32
33 def drawWindows():
34 pass
37
38 ##########
39 # MAIN #
40 ##########
41
42 beginGrfx(750,500)
43
44 drawFloors()
45 drawRoof()
46 drawChimney()
47 drawDoor()
48 drawWindows()
49
50 endGrfx()
51

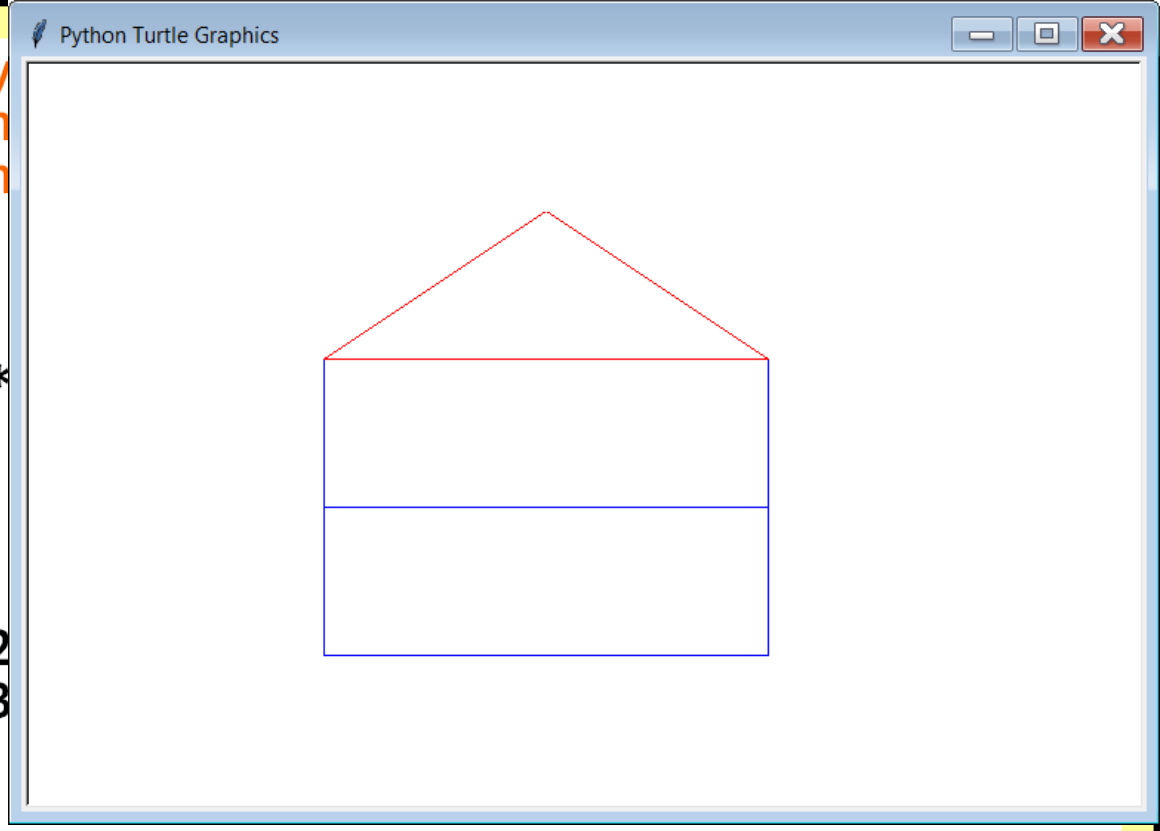
```
1  # BigGraphicsStep03.py
2  # Creating a Big Graphics Program
3  # Step 3 - Write the first procedure, and make sure it works.
4  # NOTE: Make sure you remove <pass> from this procedure.
5
6
7  from Graphics import *
8
9
10 def drawFloors():
11     setColor("blue")
12     drawRectangle(200,200,500,300)
13     drawRectangle(200,300,500,400)
14
15
16 # The rest of the program is the same as before.
17
18
```

```
1  # BigGraphicsStep03.py
2  # Creating a Big Graphics P
3  # Step 3 - Write the first
4  # NOTE: Make sure you remov
5
6
7  from Graphics import *
8
9
10 def drawFloors():
11     setColor("blue")
12     drawRectangle(200,200,500,300)
13     drawRectangle(200,300,500,400)
14
15
16 # The rest of the program is the same as before.
17
18
```



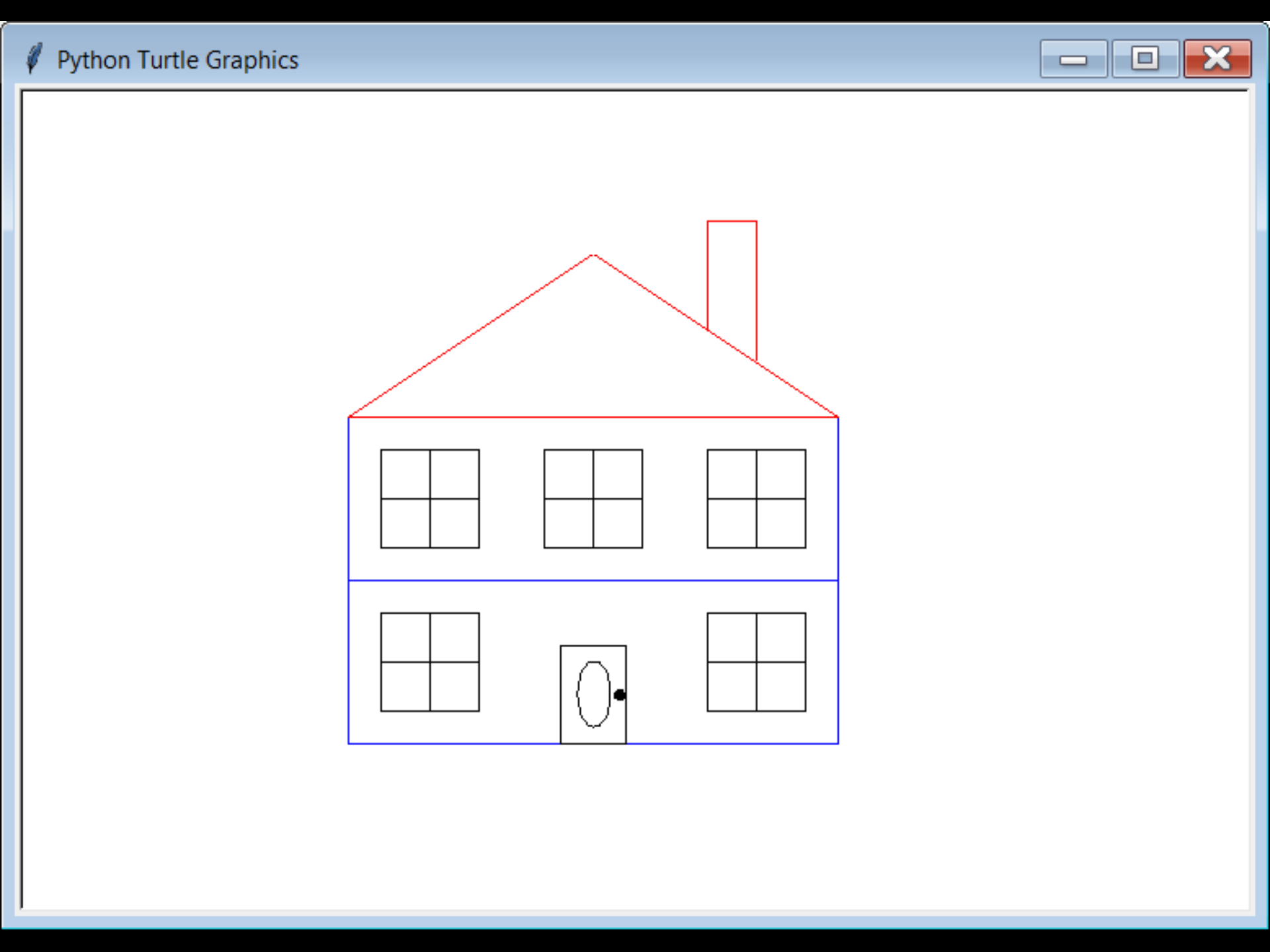
```
1  # BigGraphicsStep04.py
2  # Creating a Big Graphics Program
3  # Step 4 - Write the next procedure, and make sure it works.
4  # NOTE: Make sure you remove <pass> from this procedure.
5
6
7  from Graphics import *
8
9
10 def drawFloors():
11     setColor("blue")
12     drawRectangle(200,200,500,300)
13     drawRectangle(200,300,500,400)
14
15
16 def drawRoof():
17     setColor("red")
18     drawLine(200,200,350,100)
19     drawLine(500,200,350,100)
20     drawLine(200,200,500,200)
21
22
23 # The rest of the program is the same as before.
24
```

```
 1 # BigGraphicsStep04.py
 2 # Creating a Big Graph
 3 # Step 4 - Write the m
 4 # NOTE: Make sure you
 5
 6
 7 from Graphics import *
 8
 9
10 def drawFloors():
11     setColor("blue")
12     drawRectangle(200,2
13     drawRectangle(200,3
14
15
16 def drawRoof():
17     setColor("red")
18     drawLine(200,200,350,100)
19     drawLine(500,200,350,100)
20     drawLine(200,200,500,200)
21
22
23 # The rest of the program is the same as before.
24
```



```
# BigGraphicsStep05.py
# Creating a Big Graphics Program
# Step 5 - Repeat step 4 until the entire program is done.
# Remember remove <pass> from each procedure as you go.


from Graphics import *


def drawFloors():
    setColor("blue")
    drawRectangle(200,200,500,300)
    drawRectangle(200,300,500,400)


def drawRoof():
    setColor("red")
    drawLine(200,200,350,100)
    drawLine(500,200,350,100)
    drawLine(200,200,500,200)


def drawChimney():
    drawLine(420,146,420,80)
    drawLine(420,80,450,80)
    drawLine(450,80,450,166)


def drawDoor():
    setColor("black")
    drawRectangle(330,340,370,400)
    drawOval(350,370,10,20)
    fillCircle(366,370,3)
```

```
def drawWindows():
    drawRectangle(220,220,280,280)
    drawLine(220,250,280,250)
    drawLine(250,220,250,280)
    drawRectangle(420,220,480,280)
    drawLine(420,250,480,250)
    drawLine(450,220,450,280)
    drawRectangle(320,220,380,280)
    drawLine(320,250,380,250)
    drawLine(350,220,350,280)
    drawRectangle(220,320,280,380)
    drawLine(220,350,280,350)
    drawLine(250,320,250,380)
    drawRectangle(420,320,480,380)
    drawLine(420,350,480,350)
    drawLine(450,320,450,380)



##########
#  MAIN  #
##########

beginGrfx(750,500)

drawFloors()
drawRoof()
drawChimney()
drawDoor()
drawWindows()

endGrfx()
```


Python Turtle Graphics