Exposure CS 2021 for CS1-Honors

Chapter 18 Slides

More Graphics: Events & Computer Animation

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

Section 18.1

Introduction

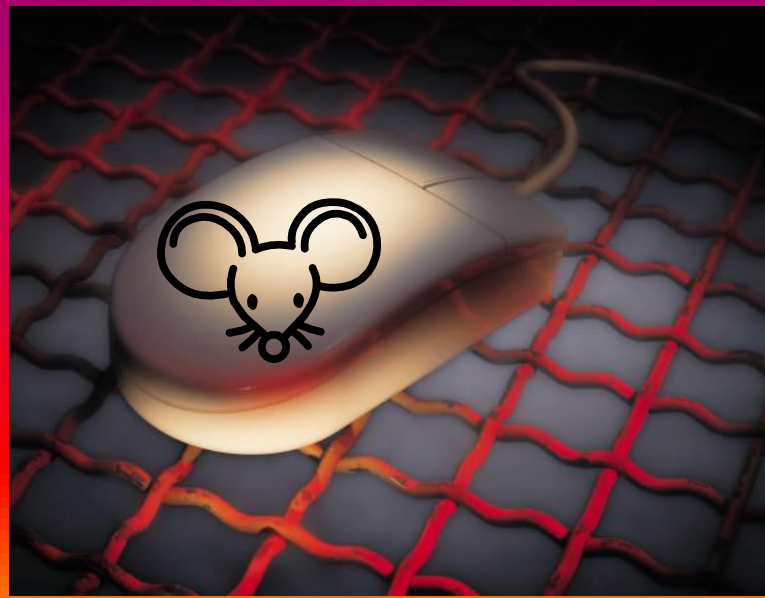
The Last Chapter is Special!

In this chapter you will learn more advanced graphics features. These features will not be as advanced as those taught in AP® Computer Science-A or Advanced Graphics, but they still will allow you to create more impressive graphics programs.

There will be no homework exercises for this chapter. There will also be no quizzes and no test. The information covered in this chapter will not even be part of the Final Exam. However, there are 2 lab assignments – 1 minor lab and 1 major lab – that are based on the information in this chapter. At John Paul II High School the major lab counts as the 4th Quarter Project.

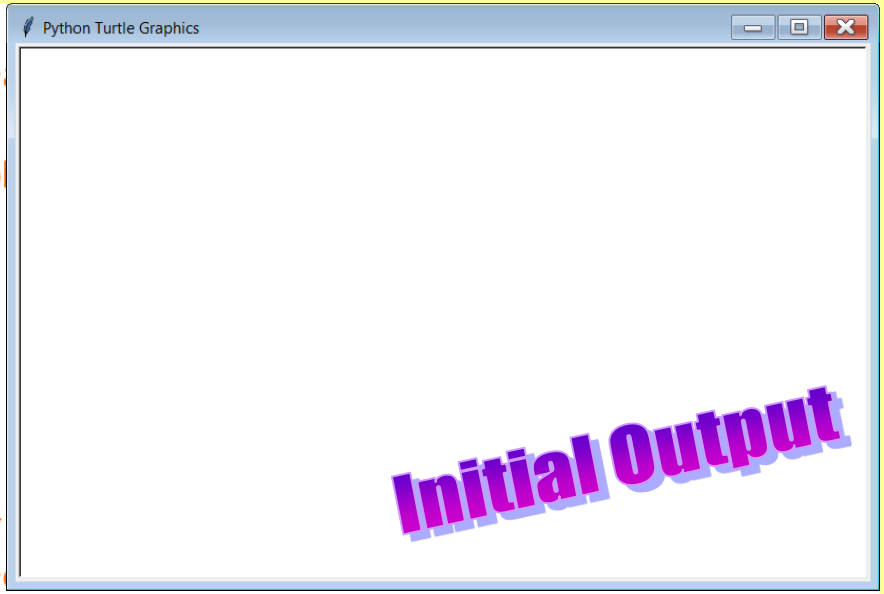
Section 18.2

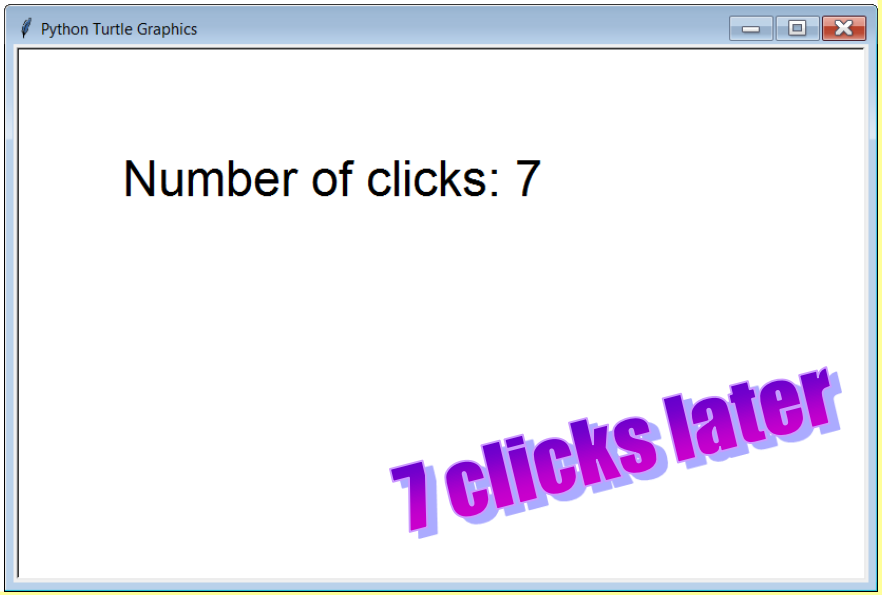
Mouse Events



```python
# MouseEvents01.py
# This program introduces "Mouse Interactivity"
# with the <onscreenclick> event which is triggered
# any time the user clicks on the graphics screen
# with the left mouse button.


from Graphics import *


def display(x,y):    # Parameters (x,y) are required
   global numClicks  # even if they are not used.
   clear()
   numClicks += 1
   drawString("Number of clicks: "+str(numClicks),100,150,"Arial",28,"normal")
   update()



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

numClicks = 0
beginGrfx(800,500)
onscreenclick(display)
endGrfx()
```

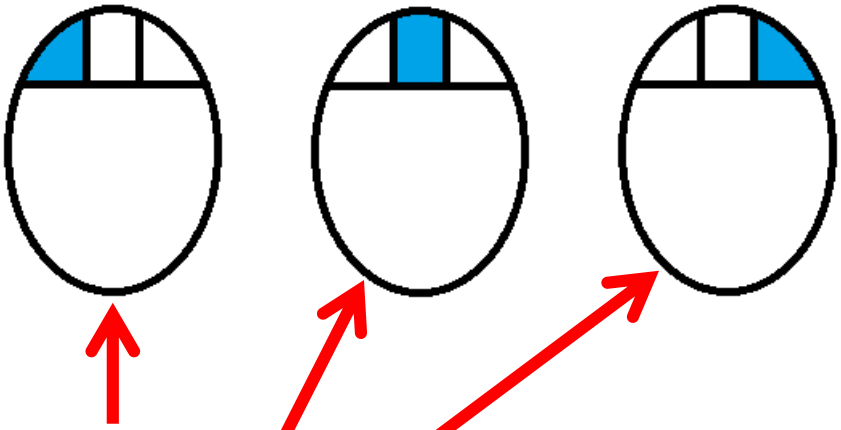
```
1  # MouseEvents01.py
2  # This program introduces "Mouse Inter
3  # with the <onscreenclick> event which
4  # any time the user clicks on the grap
5  # with the left mouse button.
6
7
8  from Graphics import *
9
10
11 def display(x,y):     # Parameters (x,y
12     global numClicks  # even if they ar
13     clear()
14     numClicks += 1
15     drawString("Number of clicks: "+str(numClicks),100,150,"Arial",28,"normal")
16     update()
17
18
19
20 ##########
21 #  MAIN  #
22 ##########
23
24 numClicks = 0
25 beginGrfx(800,500)
26 onscreenclick(display)
27 endGrfx()
```



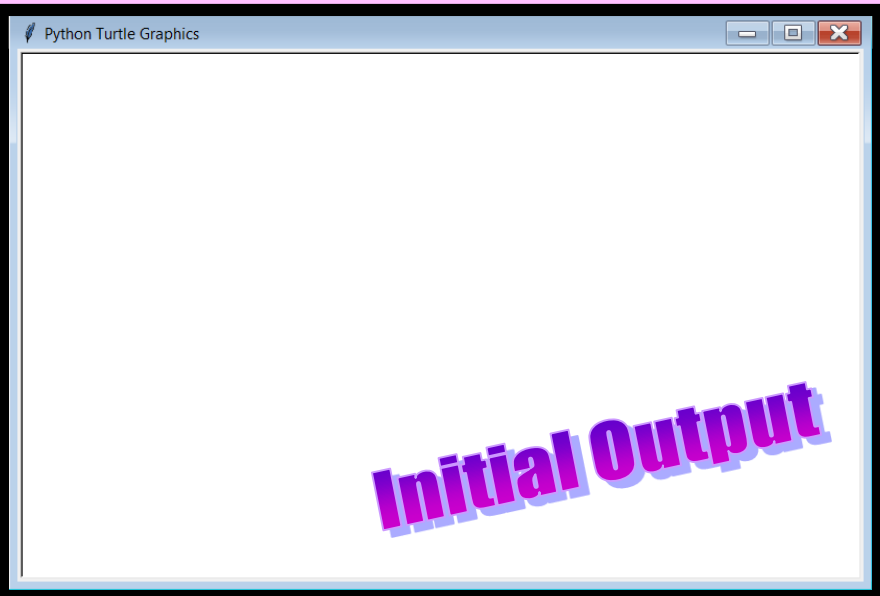


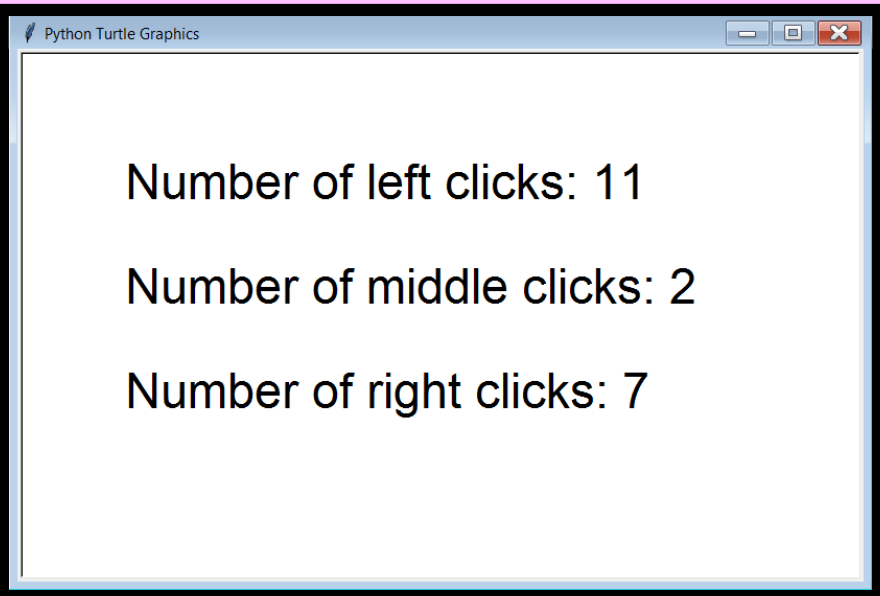
```
1 # MouseEvents02.py
2 # This program demonstrates how to distinguish between
3 # the left, middle & right mouse buttons on a Windows PC.
4 # NOTE: Some mice have a "middle" button; most do not.
5 # ALSO: On some mice, the "wheel" is clickable and is
6 #       actually interpreted as the middle button.
7
8
9 from Graphics import *
10
11
12 def leftDisplay(x,y):
13    global numLeftClicks
14    clear()
15    numLeftClicks += 1
16    drawString("Number of left clicks: "+str(numLeftClicks),100,150,"Arial",28,"normal")
17    drawString("Number of middle clicks: "+str(numMiddleClicks),100,250,"Arial",28,"normal")
18    drawString("Number of right clicks: "+str(numRightClicks),100,350,"Arial",28,"normal")
19
20
21 def middleDisplay(x,y):
22    global numMiddleClicks
23    clear()
24    numMiddleClicks += 1
25    drawString("Number of left clicks: "+str(numLeftClicks),100,150,"Arial",28,"normal")
26    drawString("Number of middle clicks: "+str(numMiddleClicks),100,250,"Arial",28,"normal")
27    drawString("Number of right clicks: "+str(numRightClicks),100,350,"Arial",28,"normal")
28
29
30 def rightDisplay(x,y):
31    global numRightClicks
32    clear()
33    numRightClicks += 1
34    drawString("Number of left clicks: "+str(numLeftClicks),100,150,"Arial",28,"normal")
35    drawString("Number of middle clicks: "+str(numMiddleClicks),100,250,"Arial",28,"normal")
36    drawString("Number of right clicks: "+str(numRightClicks),100,350,"Arial",28,"normal")
```

```
39
40 ##########
41 #  MAIN  #
42 ##########
43
44 numLeftClicks = 0
45 numMiddleClicks = 0
46 numRightClicks = 0
47 beginGrfx(800,500)
48 onscreenclick(leftDisplay,btn=1)
49 onscreenclick(middleDisplay,btn=2)
50 onscreenclick(rightDisplay,btn=3)
51 endGrfx()
```

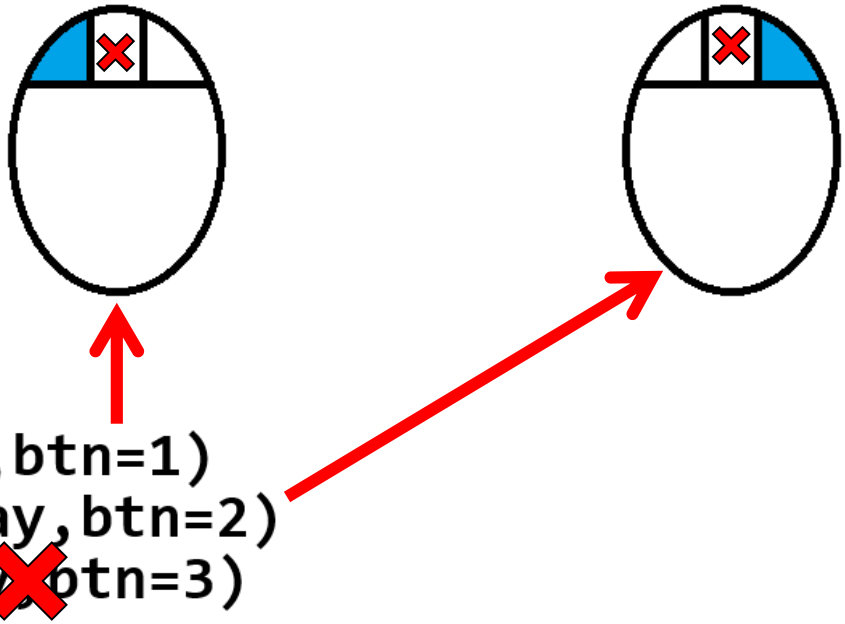


NOTE: On some mice the *wheel* is *clickable* and acts as the *middle button*.

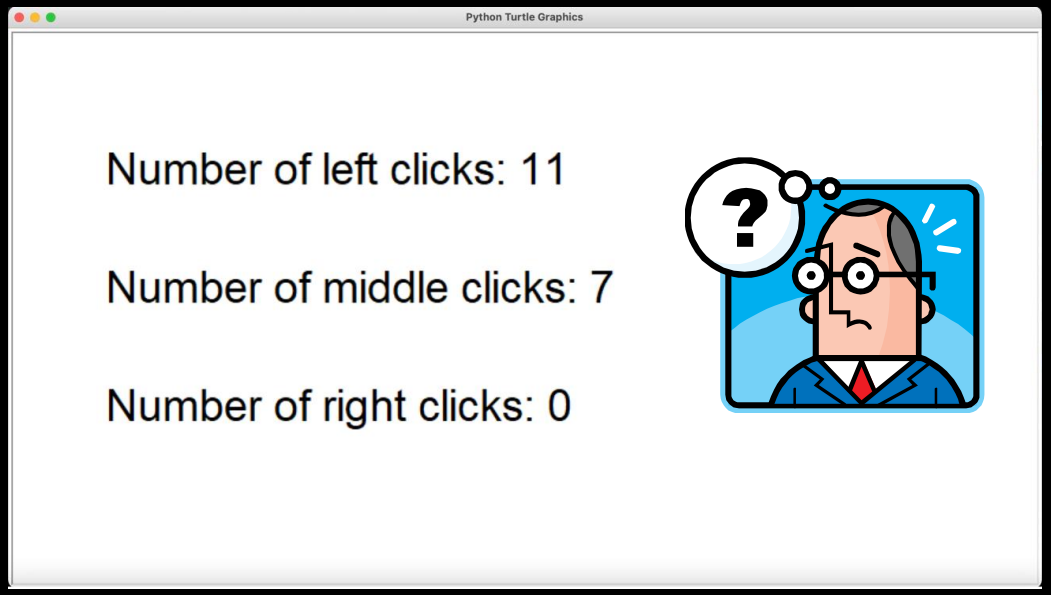




```
##########
#  MAIN  #
##########

numLeftClicks = 0
numMiddleClicks = 0
numRightClicks = 0
beginGrfx(800,500)
onscreenclick(leftDisplay,btn=1)
onscreenclick(middleDisplay,btn=2)
onscreenclick(rightDisplay,btn=3)
endGrfx()
```



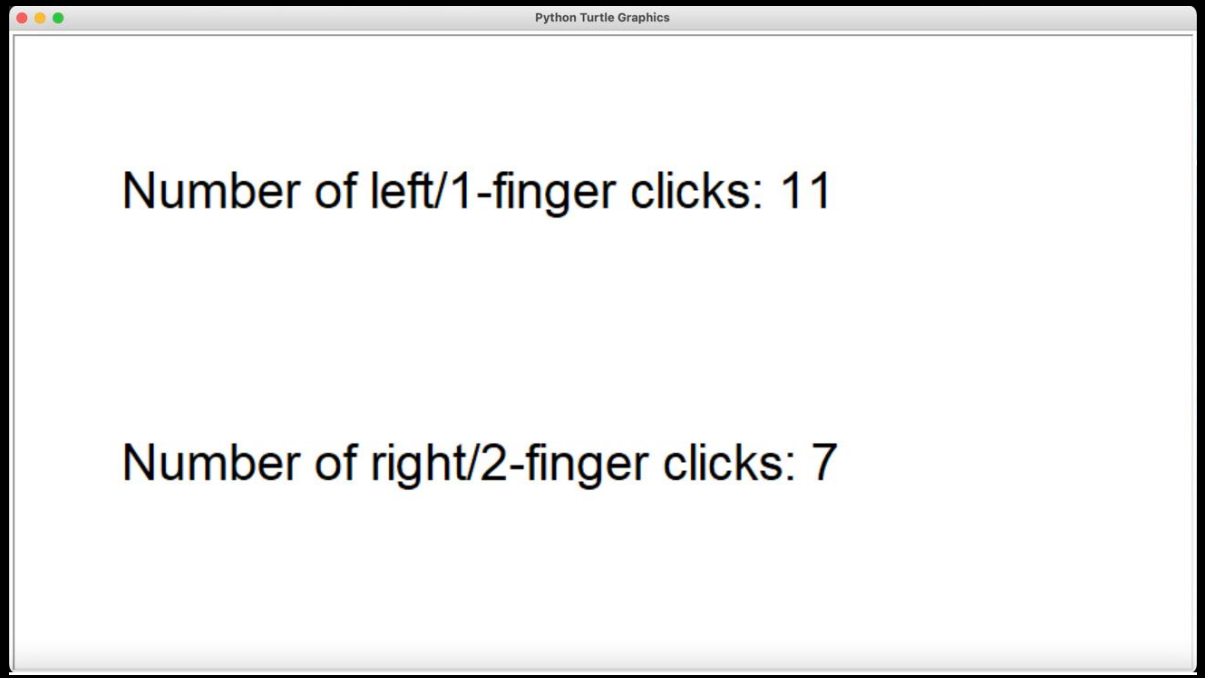
ALSO: If you run this program on a Mac, and right/2-finger click 7 times, you get this output. Since Macs don't have any type of "middle-click" they use a button value of 2 for right/2-finger click.



```
1 # MouseEvents03.py
2 # This program demonstrates how to distinguish between a
3 # left/1-finger click & a right/2-finger click on a Mac.
4 # NOTE: On a Mac, button 2 is for right/2-finger click.
5 # ALSO: There is no middle-click on a Mac.
6
7
8 from Graphics import *
9
10
11 def leftDisplay(x,y):
12     global numLeftClicks
13     clear()
14     numLeftClicks += 1
15     drawString("Number of left/1-finger clicks: "+str(numLeftClicks),100,150,"Arial",28,"normal")
16     drawString("Number of right/2-finger clicks: "+str(numRightClicks),100,350,"Arial",28,"normal")
17
18
19 def rightDisplay(x,y):
20     global numRightClicks
21     clear()
22     numRightClicks += 1
23     drawString("Number of left/1-finger clicks: " +str(numLeftClicks),100,150,"Arial",28,"normal")
24     drawString("Number of right/2-finger clicks: " +str(numRightClicks),100,350,"Arial",28,"normal")
25
26
```

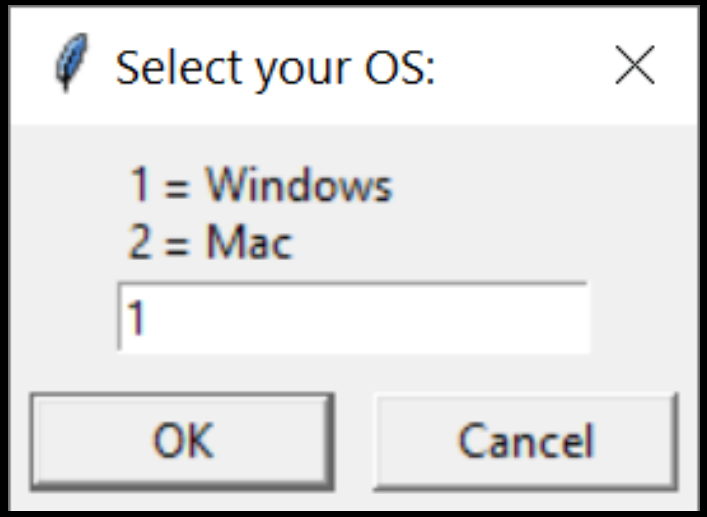
```
##########
#  MAIN  #
##########

numLeftClicks = 0
numRightClicks = 0
beginGrfx(800,500)
onscreenclick(leftDisplay,btn=1)
onscreenclick(rightDisplay,btn=2)
endGrfx()
```

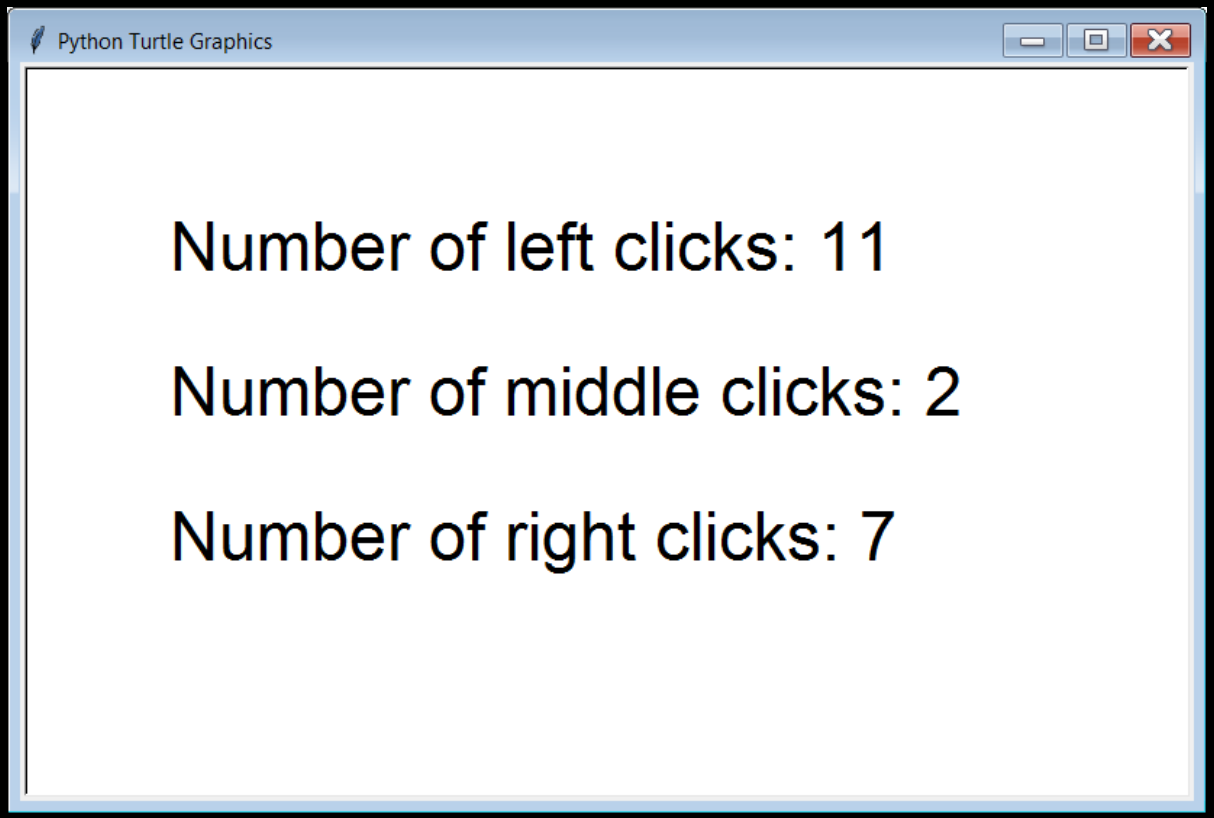
This Photo by Unknown Author is licensed under CC BY-SA

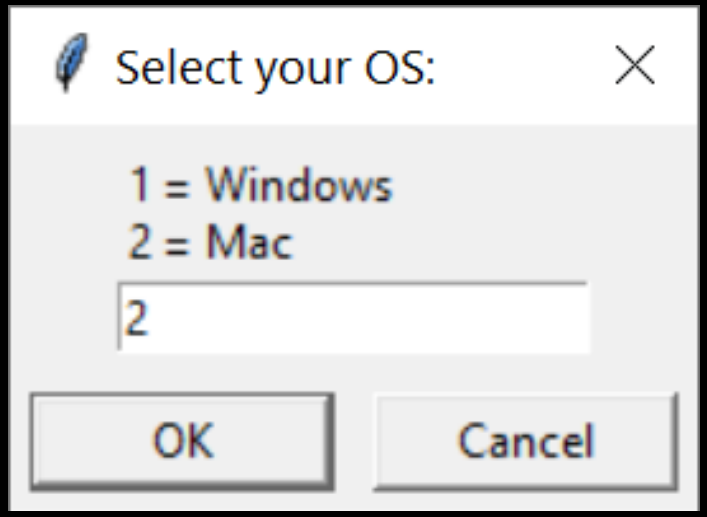


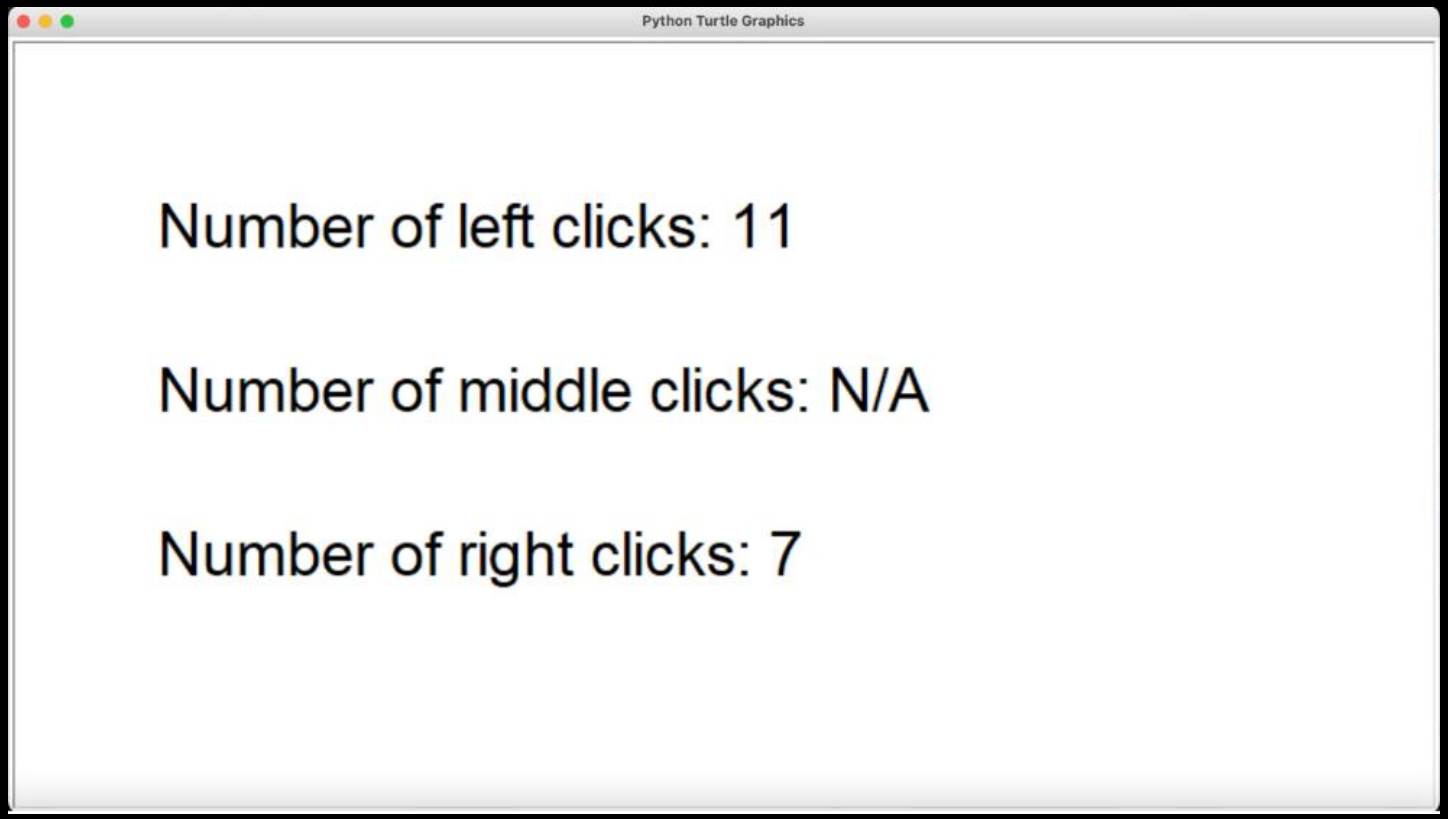
Determining the Operating System

```
 1 # MouseEvents04.py
 2 # This program will allow the user to select the computer's
 3 # operating system and use this information to properly
 4 # determine the button value of the right/2-finger click
 5 # and whether or not a middle-click exists.
 6 # NOTE: There is no guarantee that the user will select
 7 #       the correct operating system.
:
43 numLeftClicks = 0
44 numMiddleClicks = 0
45 numRightClicks = 0
46 right = 4
47 middle = 4
48
49 os = numinput("Select your OS:","1 = Windows\n2 = Mac")
50 if os == 1 : # Windows
51    middle = 2
52    right = 3
53 if os == 2 : # Mac
54    right = 2
55    numMiddleClicks = "N/A"
56
57 beginGrfx(800,500)
58 onscreenclick(leftDisplay,btn=1)
59 onscreenclick(middleDisplay,btn=middle)
60 onscreenclick(rightDisplay,btn=right)
61 endGrfx()
```


Select your OS:
1 = Windows
2 = Mac
1
OK
Cancel

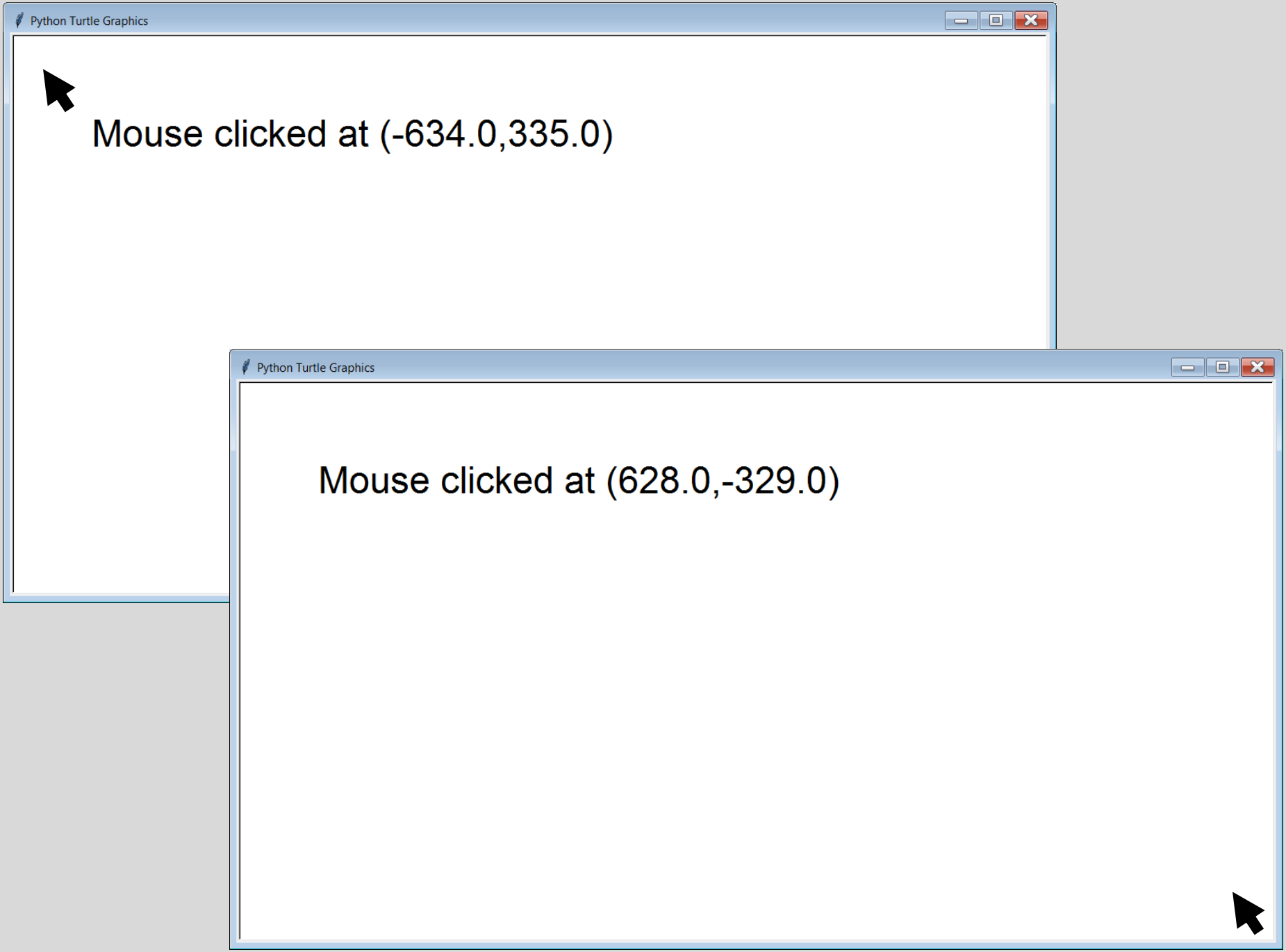

Python Turtle Graphics
Number of left clicks: 11
Number of middle clicks: 2
Number of right clicks: 7


Select your OS:
1 = Windows
2 = Mac
2
OK
Cancel


Python Turtle Graphics
Number of left clicks: 11
Number of middle clicks: N/A
Number of right clicks: 7

```
# MouseEvents05.py
# This program is more efficient and reliable that the
# previous program because it can actually detect the
# computer's operating system and use this information
# to properly determine the button value of the right/
# 2-finger click and whether or not a middle-click exists.


from Graphics import *
import os
:
numLeftClicks = 0
numMiddleClicks = 0
numRightClicks = 0
right = 4
middle = 4
if os.name == "nt": # Windows
   middle = 2
   right = 3
if os.name == "posix": # Mac
   right = 2
   numMiddleClicks = "N/A"

beginGrfx(800,500)
onscreenclick(leftDisplay,btn=1)
onscreenclick(middleDisplay,btn=middle)
onscreenclick(rightDisplay,btn=right)
endGrfx()
```

```
# MouseEvents06.py
# This program demonstrates that the <x> and <y>
# parameters store the location where the user
# clicked on the screen.  The problem is, the
# coordinate displayed is based on "Turtle Graphics"
# where (0,0) is in the center of the screen.
# This results in some negative coordinates.


from Graphics import *


def display(x,y):
   clear()
   drawString("Mouse clicked at ("+str(x)+","+str(y)+")",
100,150,"Arial",28,"normal")



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

beginGrfx(1300,700)
onscreenclick(display)
endGrfx()
```


Python Turtle Graphics
Mouse clicked at (-634.0,335.0)
Python Turtle Graphics
Mouse clicked at (628.0,-329.0)

Python Turtle Graphics

Mouse clicked at (-634.0,335.0)

Python Turtle Graphics

Mouse clicked at (628.0,-329.0)

Python Turtle Graphics

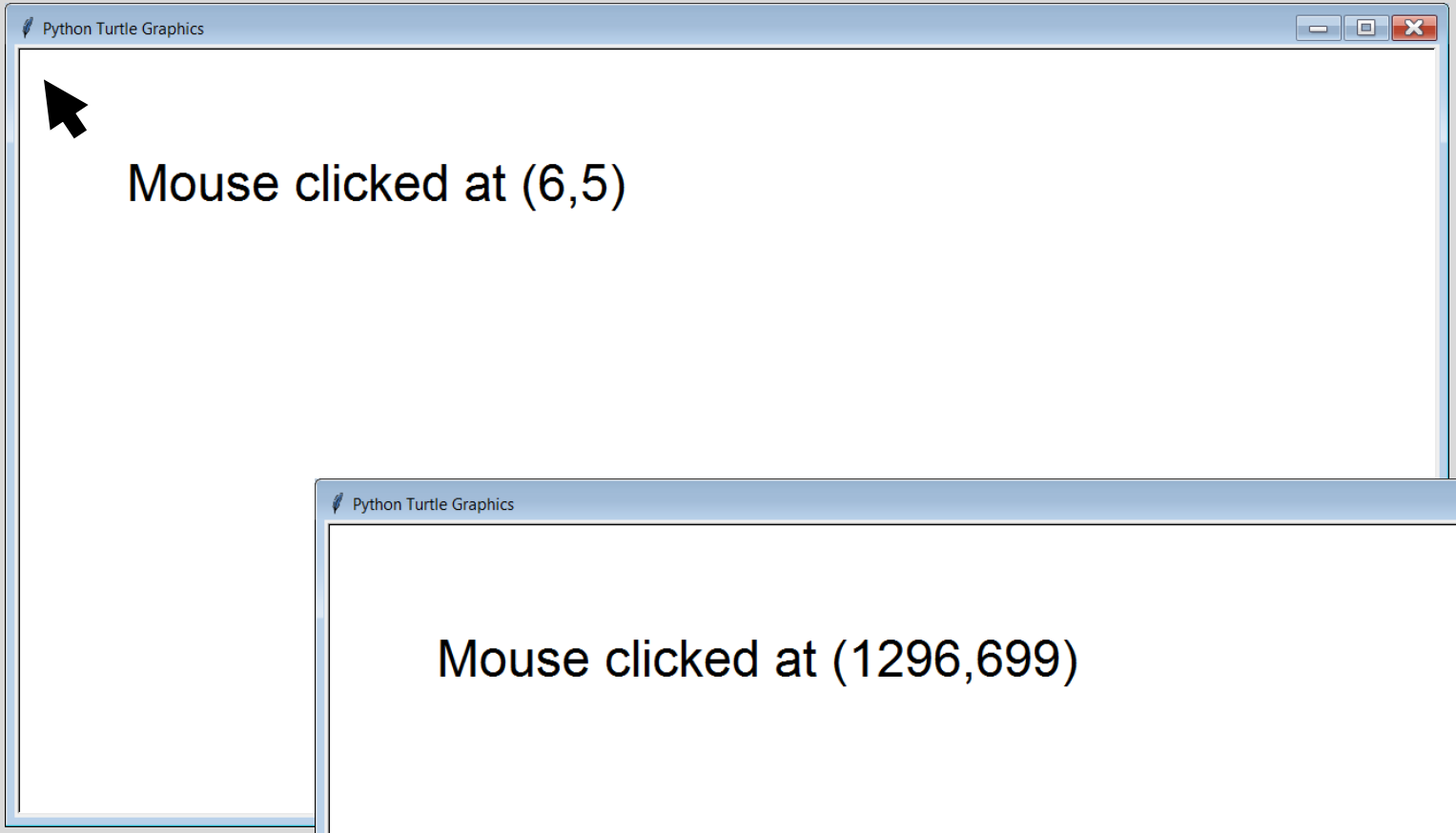
Mouse clicked at (-634.0,335.0)

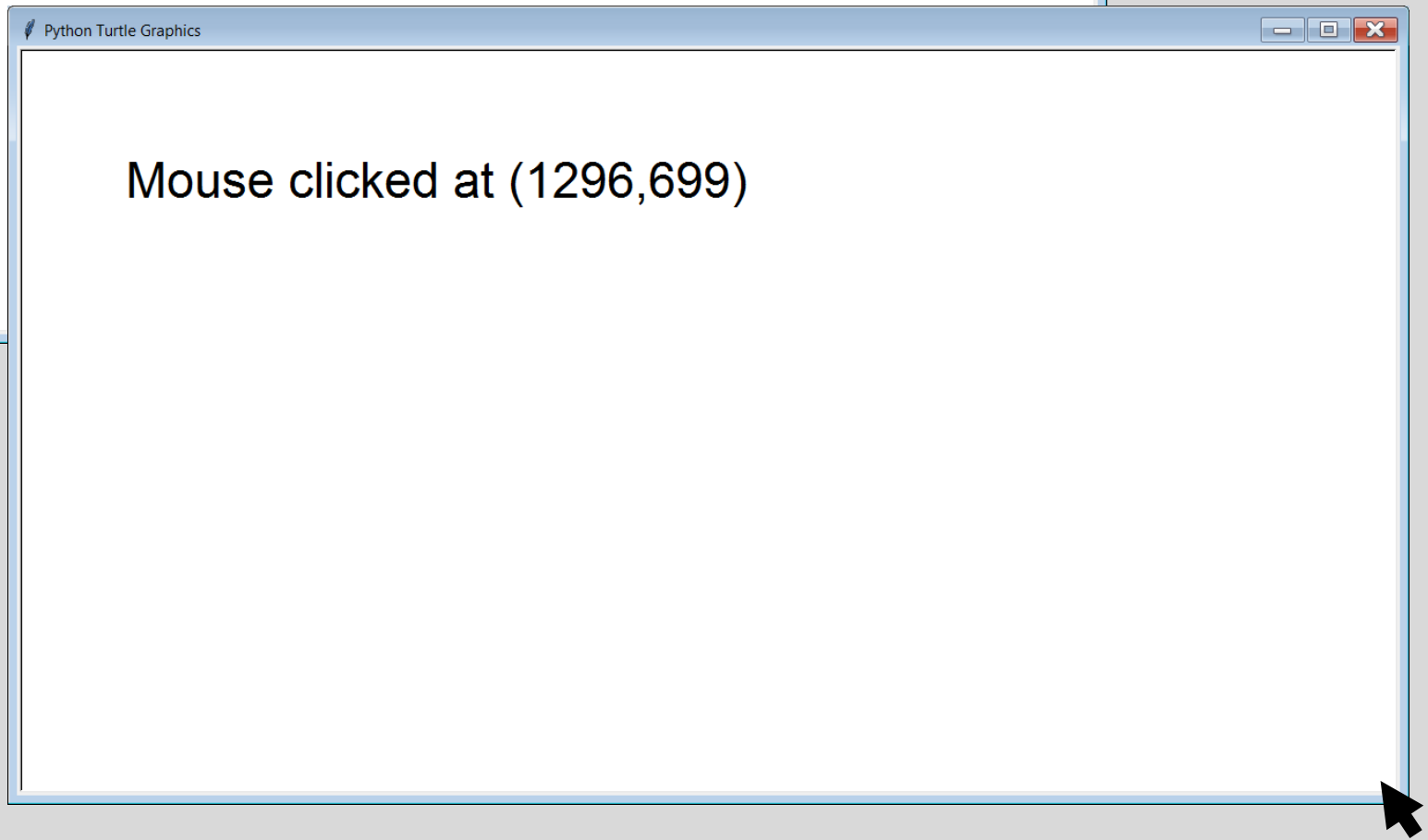
Python Turtle Graphics

Mouse clicked at (628.0,-329.0)

The coordinate returned is based on the coordinate system used by *Turtle Graphics* where (0,0) is in the center of the window. Any y values below and x values to the left will be negative as the outputs show.

```
# MouseEvents07.py
# This program fixes the issue of the previous program
# by using the <traditionalXY> function of the <Graphics>
# library.  The result is the coordinate is converted to
# "Traditional Graphics" where (0,0) is in the top-left
# corner and all coordinate values are positive.


from Graphics import *


def display(x,y):
   x,y = traditionalXY(x,y)
   clear()
   drawString("Mouse clicked at ("+str(x)+","+str(y)+")",
100,150,"Arial",28,"normal")



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

beginGrfx(1300,700)
onscreenclick(display)
endGrfx()
```


Python Turtle Graphics
Mouse clicked at (6,5)


Python Turtle Graphics
Mouse clicked at (1296,699)

Section 18.3

Creating Simple Paint Program Tools

Paint Program Tools vs. Entire Paint Program

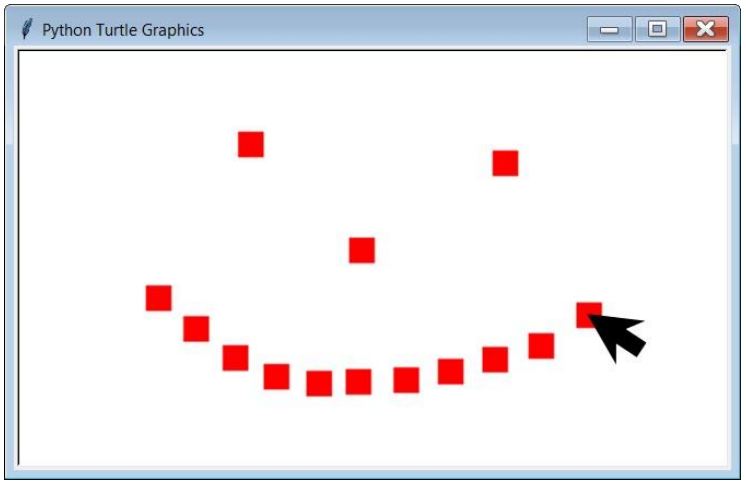
We are not creating an entire paint program. That assignment will have to wait for the *Advanced Graphics* class.

In this class, we are merely introducing some concepts by creating some of the simpler tools from a paint program.

This will lead into Lab 18A.

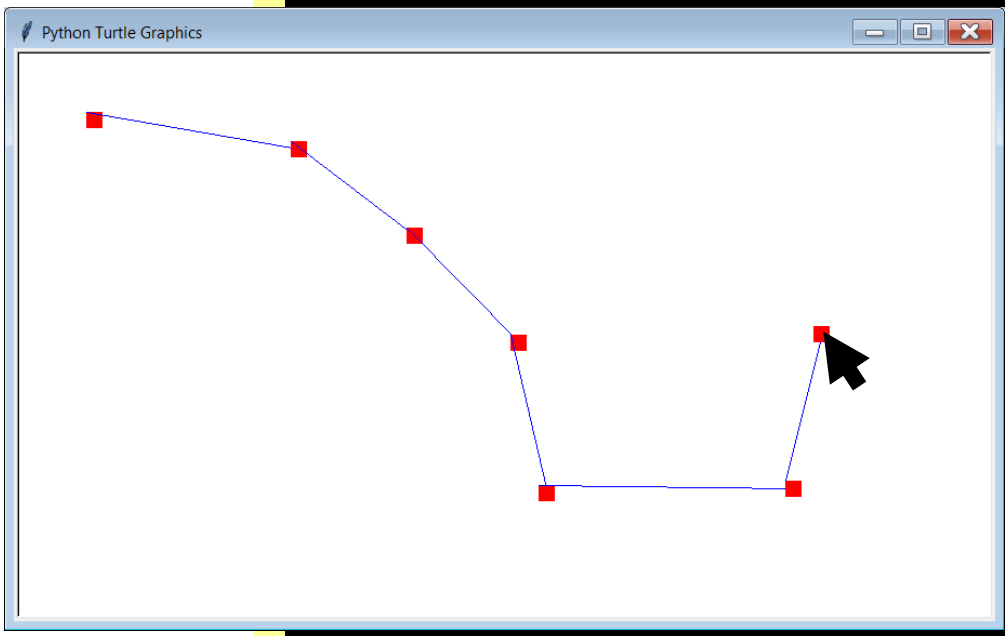
```
# MouseEvents08.py
# This program draws a small red
# square at every click location.


from Graphics import *


def display(x,y):
    x,y = traditionalXY(x,y)
    fillRectangle(x-7,y-7,x+7,y+7)
    update()



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

beginGrfx(1300,700)
setColor("red")
onscreenclick(display)
endGrfx()
```

```python
# MouseEvents08.py
# This program draws a small red
# square at every click location.


from Graphics import *


def display(x,y):
    x,y = traditionalXY(x,y)
    fillRectangle(x-7,y-7,x+7,y+7)
    update()



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

beginGrfx(1300,700)
setColor("red")
onscreenclick(display)
endGrfx()
```

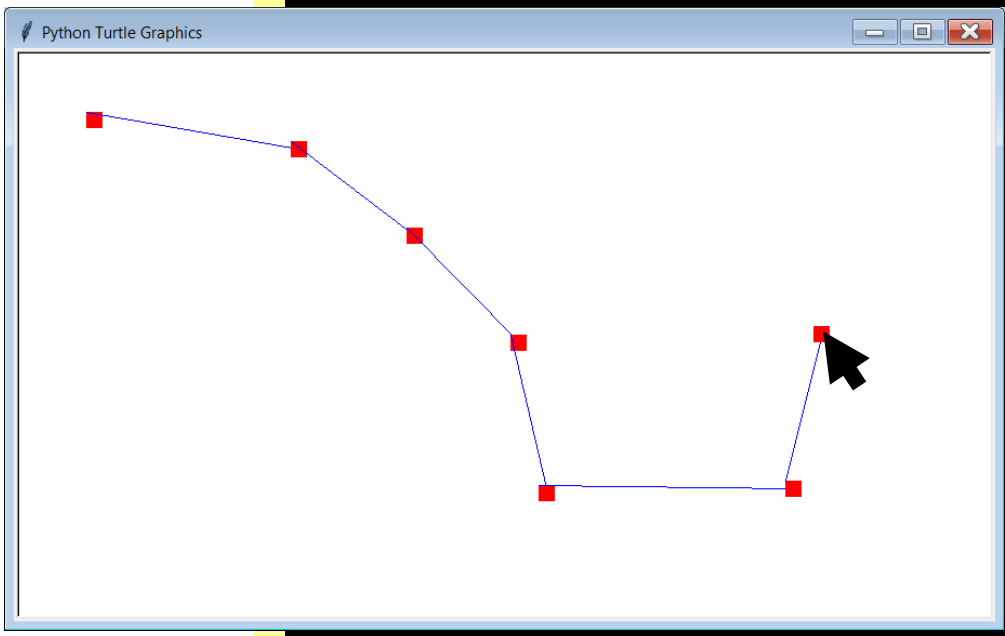


```python
# MouseEvents09.py
# This program draws a small red square when
# the mouse is left-clicked and a blue line
# when the mouse in right-clicked.


from Graphics import *


def displayDot(x,y):
   setColor("red")
   x,y = traditionalXY(x,y)
   fillRectangle(x-7,y-7,x+7,y+7)
   update()

def displayLine(x,y):
   setColor("blue")
   goto(x,y)
   update()



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

beginGrfx(1300,700)
onscreenclick(displayDot,btn=1)
onscreenclick(displayLine,btn=3)
endGrfx()
```

```python
# MouseEvents09.py
# This program draws a small red square when
# the mouse is left-clicked and a blue line
# when the mouse in right-clicked.


from Graphics import *


def displayDot(x,y):
    setColor("red")
    x,y = traditionalXY(x,y)
    fillRectangle(x-7,y-7,x+7,y+7)
    update()

def displayLine(x,y):
    setColor("blue")
    goto(x,y)
    update()



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

beginGrfx(1300,700)
onscreenclick(displayDot,btn=1)
onscreenclick(displayLine,btn=3)
endGrfx()
```



```python
# MouseEvents09.py
# This program draws a small red square when
# the mouse is left-clicked and a blue line
# when the mouse in right-clicked.


from Graphics import *


def displayDot(x,y):
    setColor("red")
    x,y = traditionalXY(x,y)
    fillRectangle(x-7,y-7,x+7,y+7)
    update()

def displayLine(x,y):
    setColor("blue")
    goto(x,y)
    update()



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

beginGrfx(1300,700)
onscreenclick(displayDot,btn=1)
onscreenclick(displayLine,btn=3)
endGrfx()
```



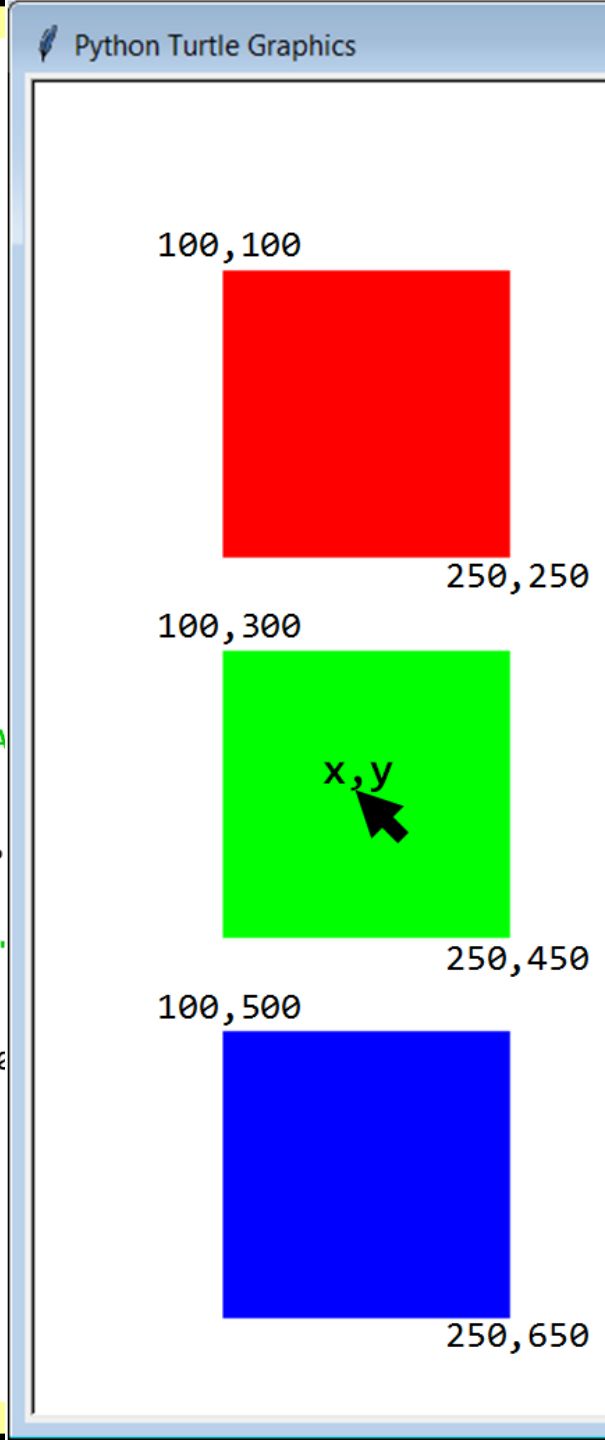
Unlike drawLine, with goto, you only specify the *end point*. The *start point* is simply the last coordinate that was used. If the program has just started, the *start point* is (0,0). Also, goto uses the coordinates from *Turtle Graphics*, so no conversion is necessary.

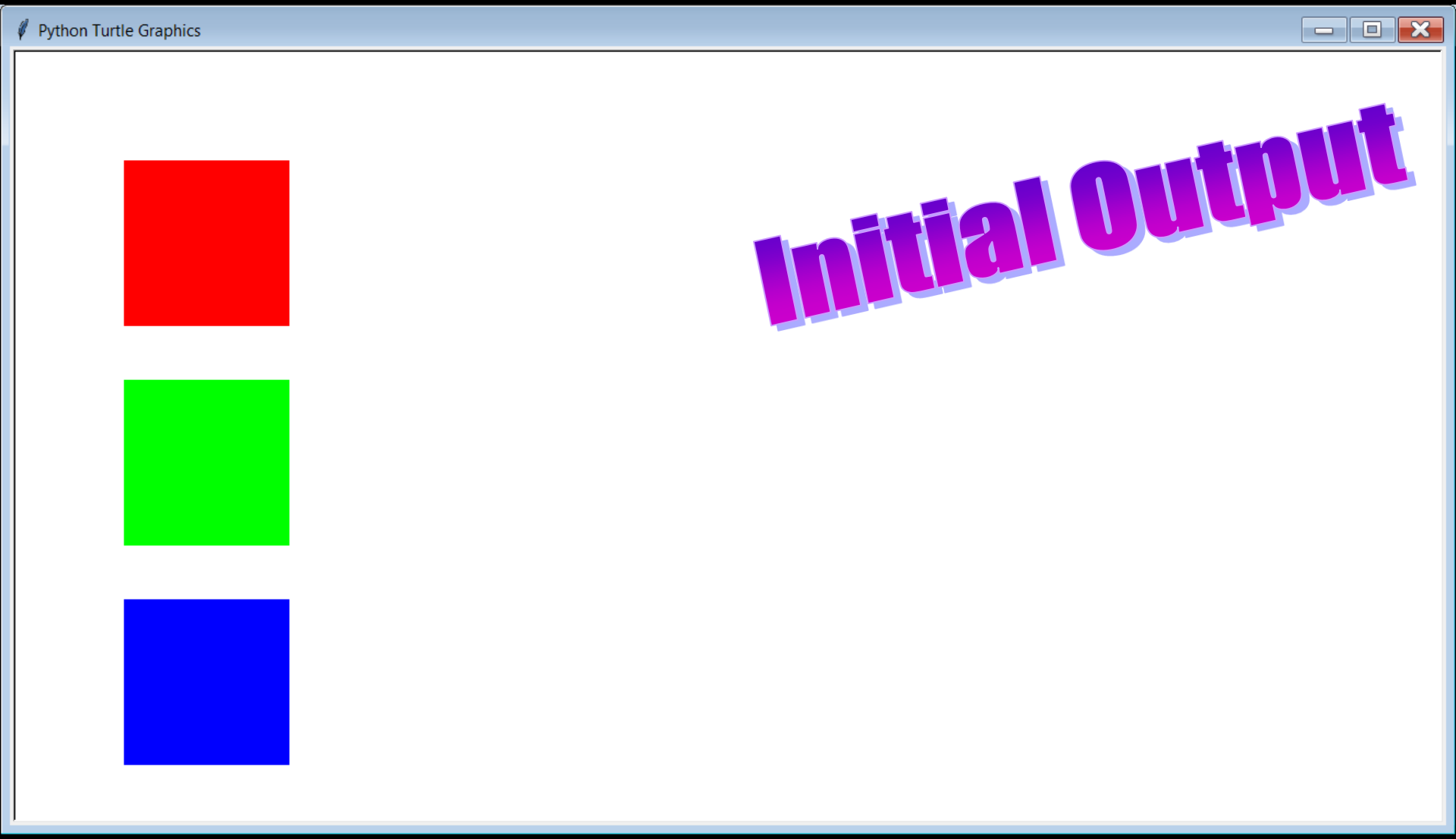
Section 18.4

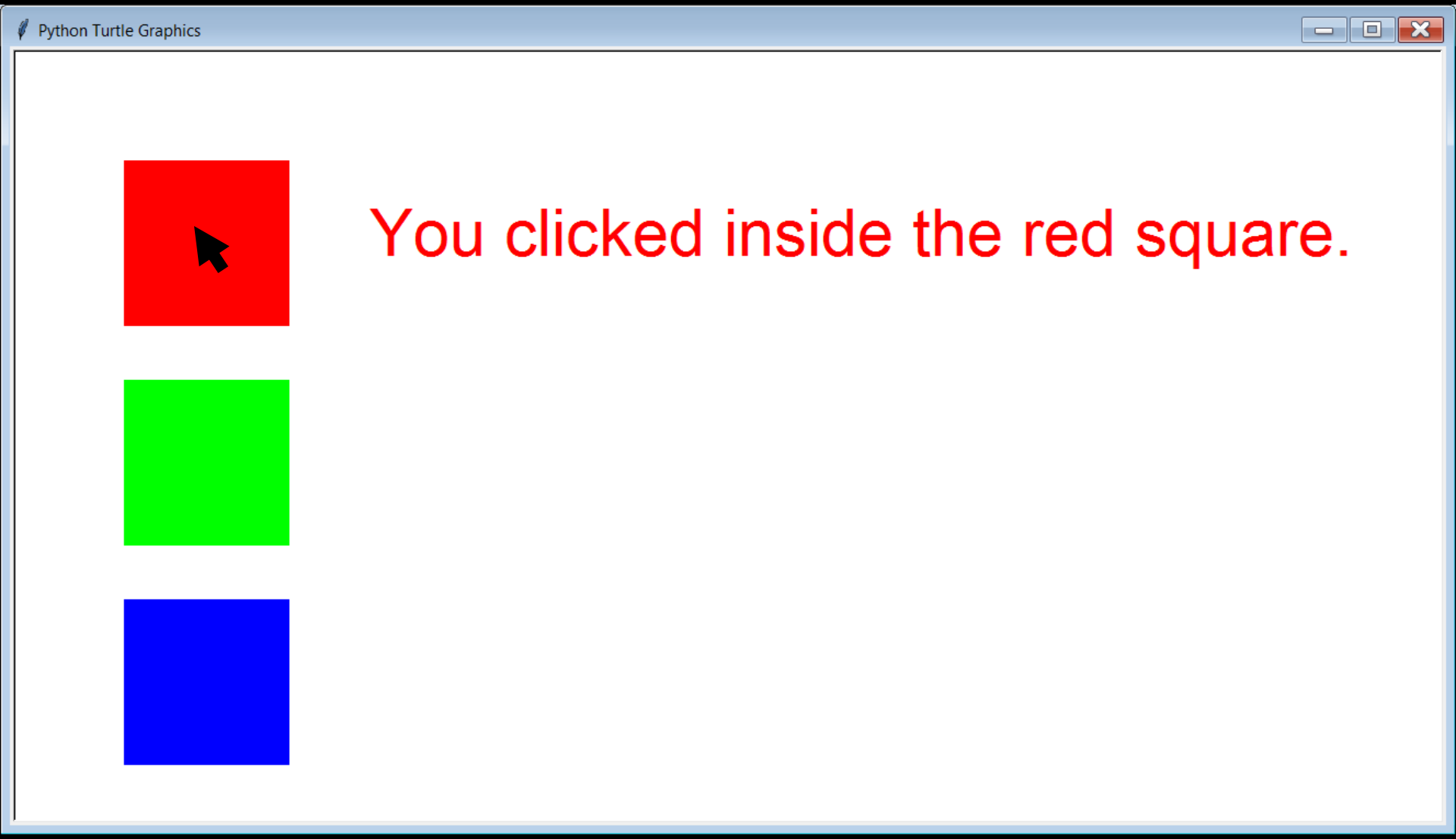
Creating Clickable Areas

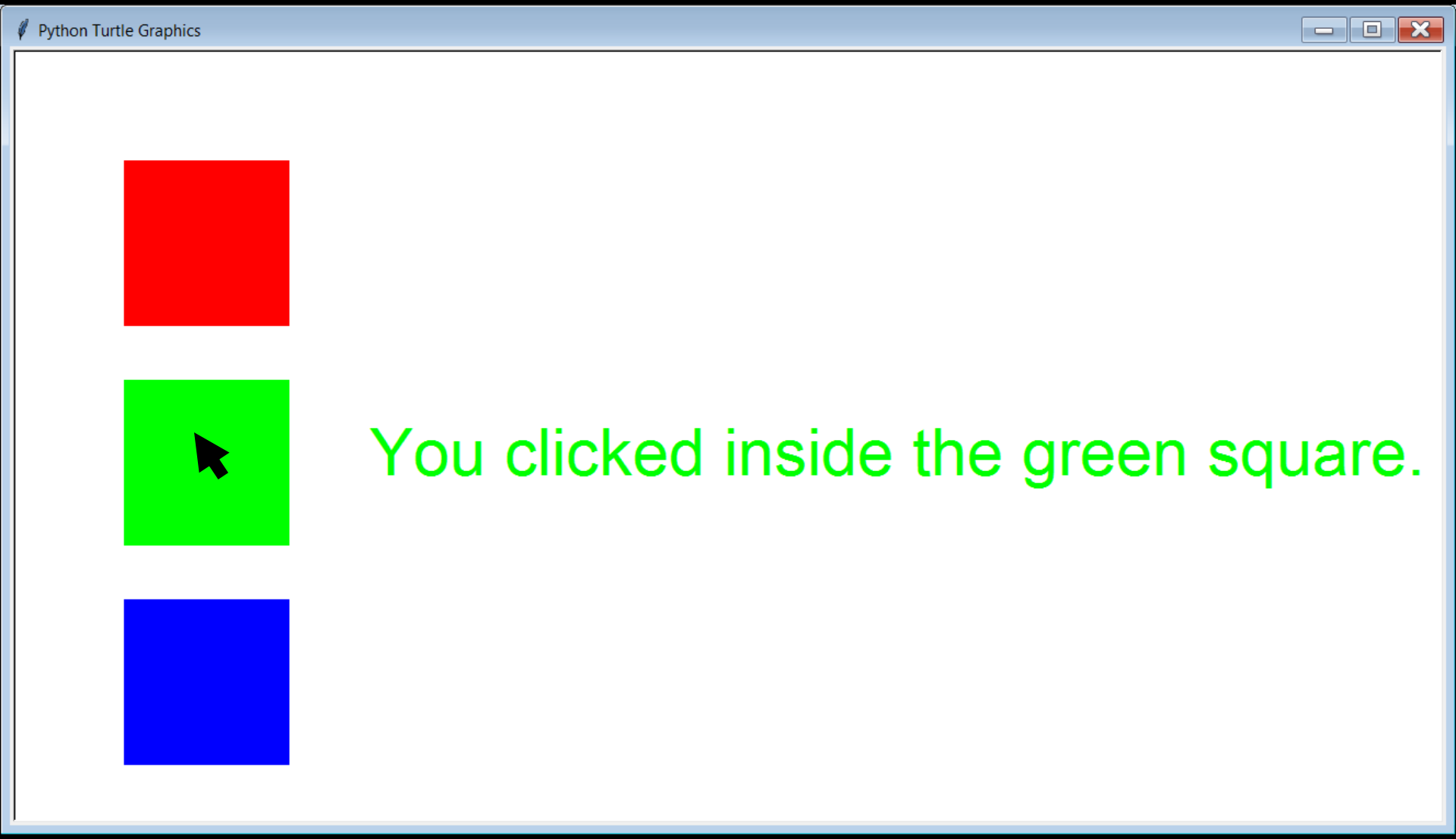
```python
# MouseEvents10.py
# This program uses "ranges" to determine
# if the user clicked inside a certain
# rectangular (or square) area of the screen.

from Graphics import *

def displayBoxes():
   setColor("red")
   fillRectangle(100,100,250,250)
   setColor("green")
   fillRectangle(100,300,250,450)
   setColor("blue")
   fillRectangle(100,500,250,650)

def locate(x,y):
   clear()
   displayBoxes()
   x,y = traditionalXY(x,y)
   if 100 <= x <= 250 and 100 <= y <= 250:
      setColor("red")
      drawString("You clicked inside the red square.",325,200,"Arial",36,"normal")
   elif 100 <= x <= 250 and 300 <= y <= 450:
      setColor("green")
      drawString("You clicked inside the green square.",325,400,"Arial",36,"normal")
   elif 100 <= x <= 250 and 500 <= y <= 650:
      setColor("blue")
      drawString("You clicked inside the blue square.",325,600,"Arial",36,"normal")
   else:
      setColor("black")
      drawString("You did not click inside any of the squares.",150,80,"Arial",36,"normal")


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

beginGrfx(1300,700)
displayBoxes()
onscreenclick(locate)
endGrfx()
```

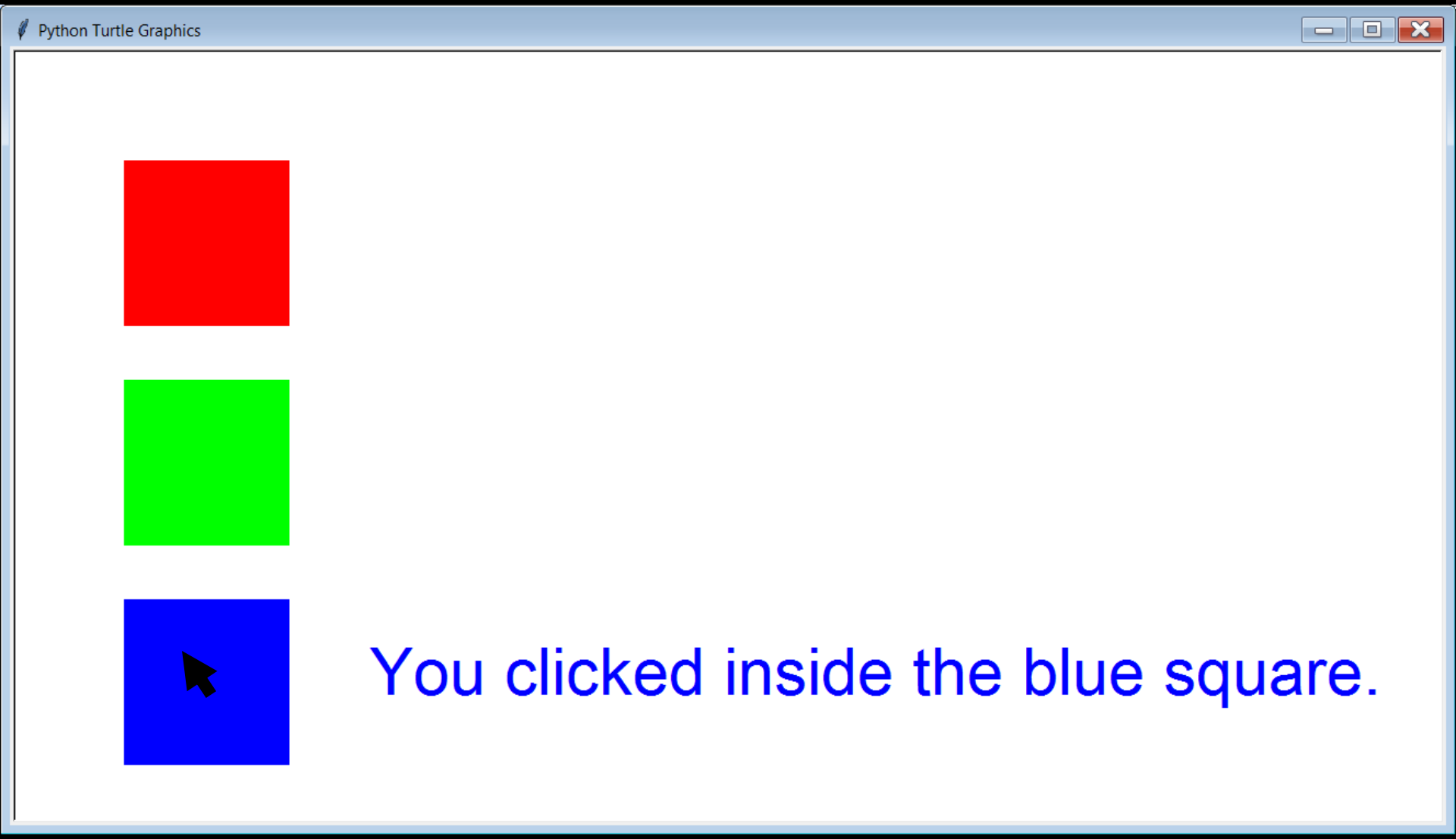
```python
# MouseEvents10.py
# This program uses "ranges" to determine
# if the user clicked inside a certain
# rectangular (or square) area of the screen.

from Graphics import *

def displayBoxes():
   setColor("red")
   fillRectangle(100,100,250,250)
   setColor("green")
   fillRectangle(100,300,250,450)
   setColor("blue")
   fillRectangle(100,500,250,650)

def locate(x,y):
   clear()
   displayBoxes()
   x,y = traditionalXY(x,y)
   if 100 <= x <= 250 and 100 <= y <= 250:
      setColor("red")
      drawString("You clicked inside the red square.",325,200,"A
   elif 100 <= x <= 250 and 300 <= y <= 450:
      setColor("green")
      drawString("You clicked inside the green square.",325,400,
   elif 100 <= x <= 250 and 500 <= y <= 650:
      setColor("blue")
      drawString("You clicked inside the blue square.",325,600,"
   else:
      setColor("black")
      drawString("You did not click inside any of the squares.",150,86


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

beginGrfx(1300,700)
displayBoxes()
onscreenclick(locate)
endGrfx()
```

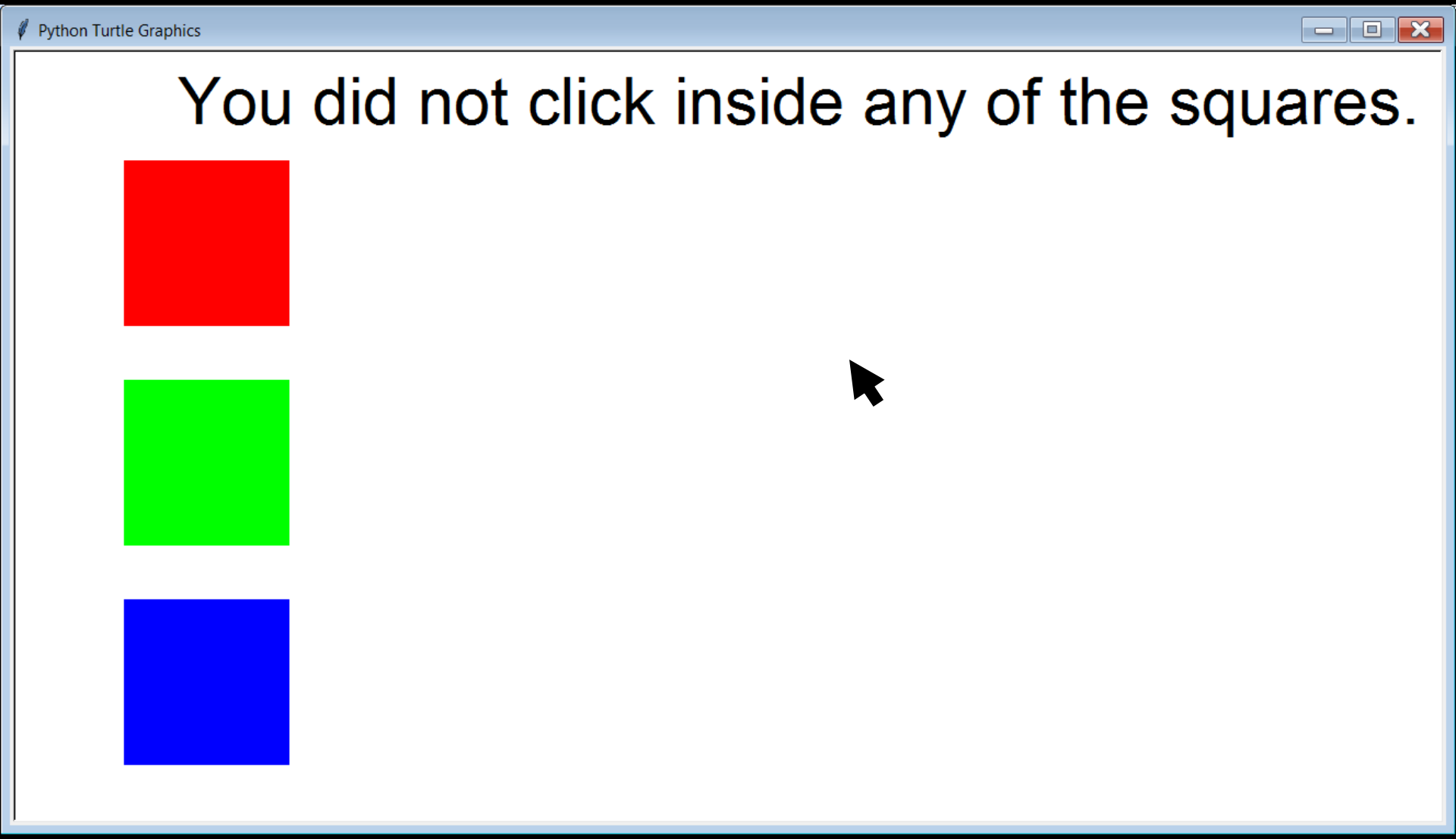



Python Turtle Graphics
Initial Output


Python Turtle Graphics
You clicked inside the red square.

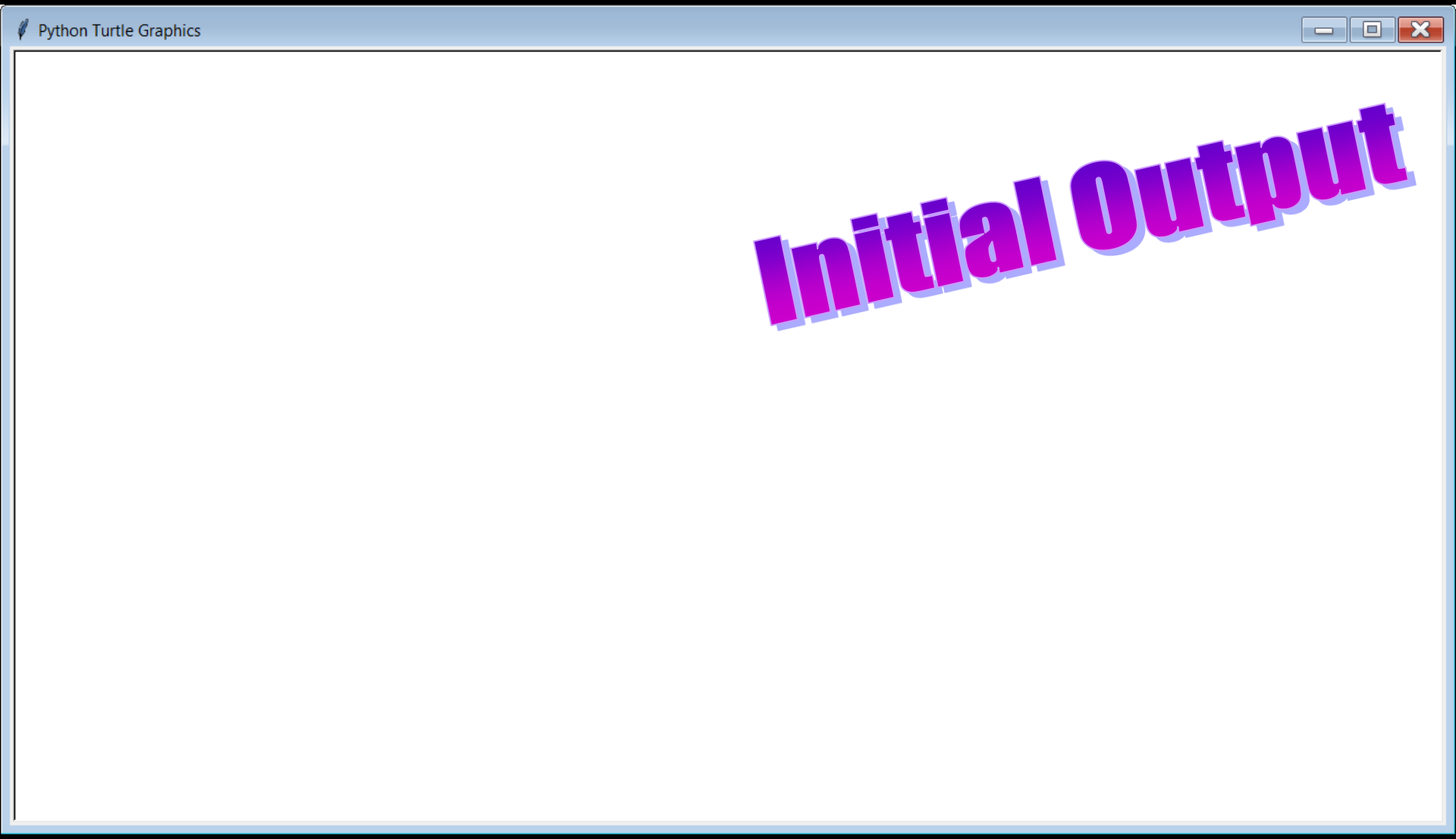

Python Turtle Graphics
You clicked inside the green square.


Python Turtle Graphics
You clicked inside the blue square.


Python Turtle Graphics
You did not click inside any of the squares.

```
# MouseEvents11.py
# This program does the exact same thing as the previous program.
# The code is now a little more efficient because it uses the <inside>
# function of the <Graphics> library.

from Graphics import *

def displayBoxes():
   setColor("red")
   fillRectangle(100,100,250,250)
   setColor("green")
   fillRectangle(100,300,250,450)
   setColor("blue")
   fillRectangle(100,500,250,650)

def locate(x,y):
   clear()
   displayBoxes()
   if inside(x,y,100,100,250,250):
      setColor("red")
      drawString("You clicked inside the red square.",325,200,"Arial",36,"normal")
   elif inside(x,y,100,300,250,450):
      setColor("green")
      drawString("You clicked inside the green square.",325,400,"Arial",36,"normal")
   elif inside(x,y,100,500,250,650):
      setColor("blue")
      drawString("You clicked inside the blue square.",325,600,"Arial",36,"normal")
   else:
      setColor("black")
      drawString("You did not click inside any of the squares.",150,80,"Arial",36,"normal")


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

beginGrfx(1300,700)
displayBoxes()
onscreenclick(locate)
endGrfx()
```

```
1  # MouseEvents12.py
2  # The only difference between this program and the previous program is that
3  # in this one the calls to procedure <displayBoxes> have been commented out.
4  # Note that even though the boxes are not displayed, you can still click on
5  # them... or at least, you can click on the area they occupied.
6
7  from Graphics import *
8
9  def displayBoxes():
10     setColor("red")
11     fillRectangle(100,100,250,250)
12     setColor("green")
13     fillRectangle(100,300,250,450)
14     setColor("blue")
15     fillRectangle(100,500,250,650)
16
17 def locate(x,y):
18     clear()
19     #displayBoxes()
20     if inside(x,y,100,100,250,250):
21         setColor("red")
22         drawString("You clicked inside the red square.",325,200,"Arial",36,"normal")
23     elif inside(x,y,100,300,250,450):
24         setColor("green")
25         drawString("You clicked inside the green square.",325,400,"Arial",36,"normal")
26     elif inside(x,y,100,500,250,650):
27         setColor("blue")
28         drawString("You clicked inside the blue square.",325,600,"Arial",36,"normal")
29     else:
30         setColor("black")
31         drawString("You did not click inside any of the squares.",150,80,"Arial",36,"normal")
32
33
34 ##########
35 #  MAIN  #
36 ##########
37
38 beginGrfx(1300,700)
39 #displayBoxes()
40 onscreenclick(locate)
41 endGrfx()
```


Python Turtle Graphics
Initial Output

Python Turtle Graphics

You clicked inside the red square.

Python Turtle Graphics

You clicked inside the green square.

Python Turtle Graphics

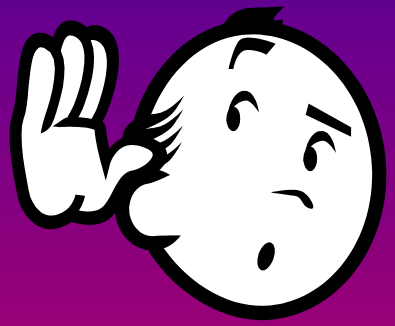
You clicked inside the blue square.

Python Turtle Graphics

You did not click inside any of the squares.

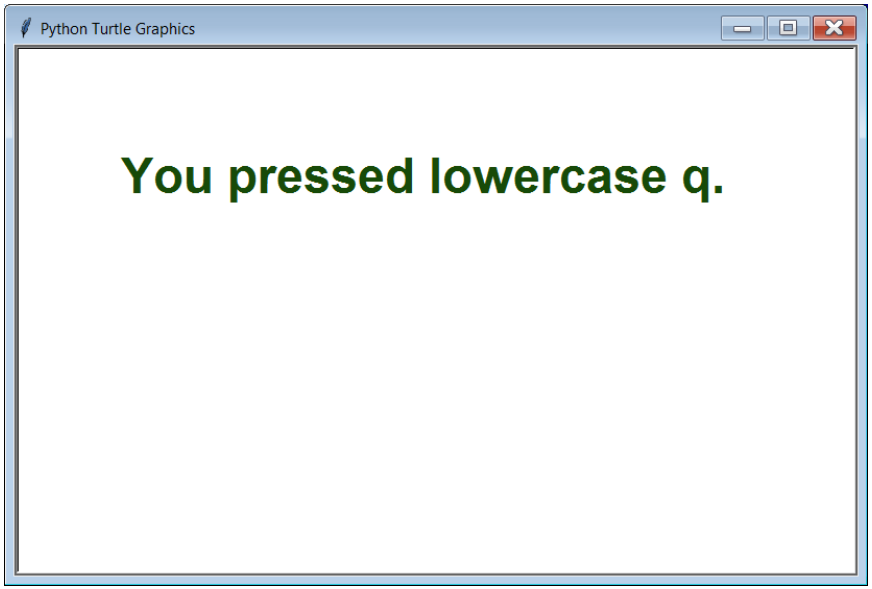
Section 18.5

Key Events



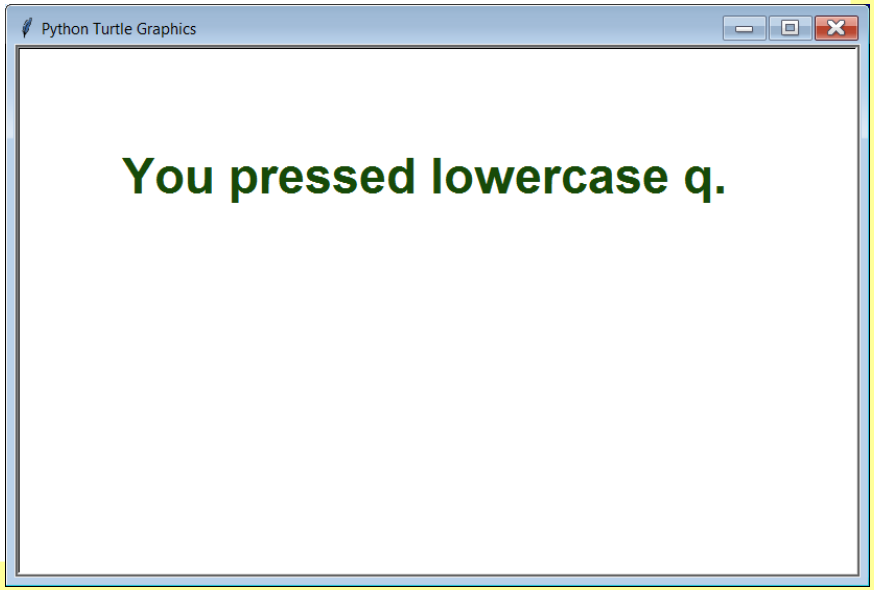
```
# KeyEvents01.py
# This program introduces "Key Interactivity"
# by displaying a randomly colored message
# anytime lowercase 'q' is typed.
# NOTE: This not the same as "Keyboard Input".
# ALSO: The <listen> command is necessary for
# the <onkey> command to function properly.


from Graphics import *


def displayq():
   clear()
   setRandomColor()
   drawString("You pressed lowercase q.",100,150,"Arial",28,"bold")
   update()



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


beginGrfx(800,500)
listen()
onkey(displayq,"q")
endGrfx()
```

```
# KeyEvents01.py
# This program introduces "Key Interactivity"
# by displaying a randomly colored message
# anytime lowercase 'q' is typed.
# NOTE: This not the same as "Keyboard Input".
# ALSO: The <listen> command is necessary for
# the <onkey> command to function properly.


from Graphics import *


def displayq():
   clear()
   setRandomColor()
   drawString("You pressed lowercase q.",100,150,"Arial",28,"bold")
   update()



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


beginGrfx(800,500)
listen()
onkey(displayq,"q")
endGrfx()
```

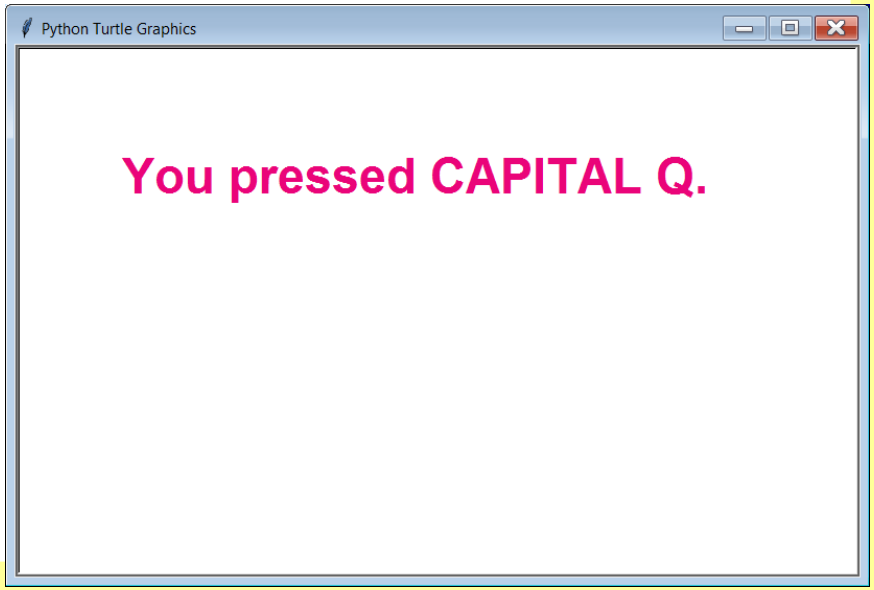


```
1 # KeyEvents02.py
2 # This program demonstrates that the computer
3 # can "listen" for multiple different keys.
4 # It also shows that CAPITAL 'Q' and lowercase 'q'
5 # are treated as different keys.
6
7
8 from Graphics import *
9
10
11 def displayq():
12    clear()
13    setRandomColor()
14    drawString("You pressed lowercase q.",100,150,"Arial",28,"bold")
15    update()
16
17 def displayQ():
18    clear()
19    setRandomColor()
20    drawString("You pressed CAPITAL Q.",100,150,"Arial",28,"bold")
21    update()
22
23
24
25 ##########
26 #  MAIN  #
27 ##########
28
29
30 beginGrfx(800,500)
31 listen()
32 onkey(displayq,"q")
33 onkey(displayQ,"Q")
34 endGrfx()
```

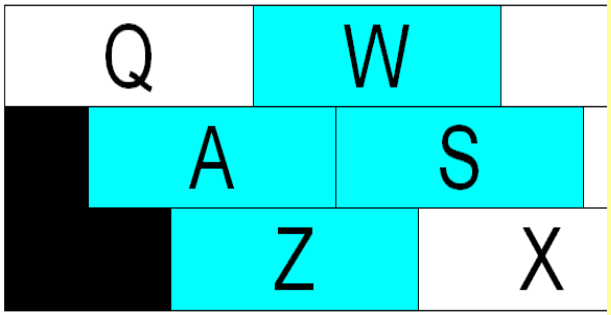
```python
# KeyEvents02.py
# This program demonstrates that the computer
# can "listen" for multiple different keys.
# It also shows that CAPITAL 'Q' and lowercase 'q'
# are treated as different keys.


from Graphics import *


def displayq():
   clear()
   setRandomColor()
   drawString("You pressed lowercase q.",100,150,"Arial",28,"bold")
   update()

def displayQ():
   clear()
   setRandomColor()
   drawString("You pressed CAPITAL Q.",100,150,"Arial",28,"bold")
   update()



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


beginGrfx(800,500)
listen()
onkey(displayq,"q")
onkey(displayQ,"Q")
endGrfx()
```



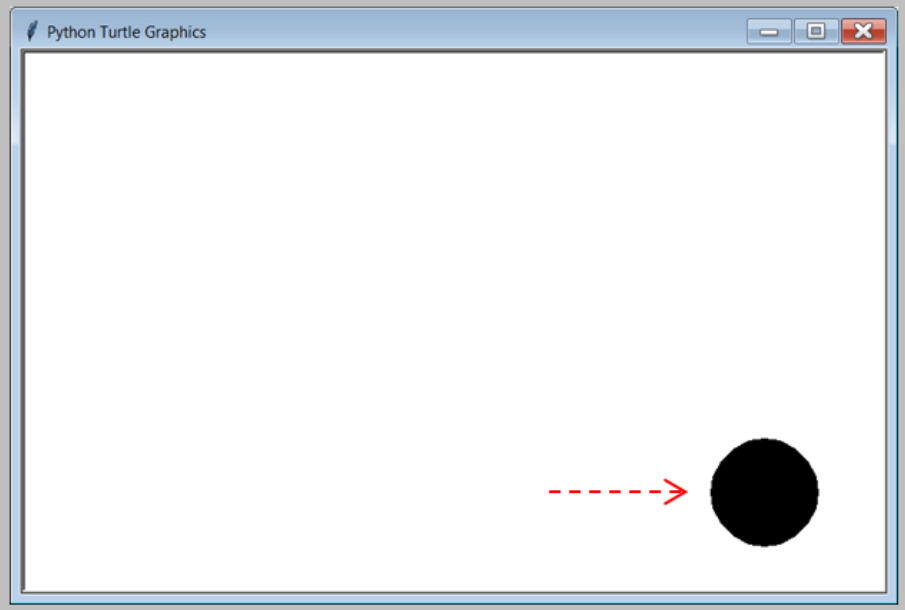
```python
# KeyEvents02.py
# This program demonstrates that the computer
# can "listen" for multiple different keys.
# It also shows that CAPITAL 'Q' and lowercase 'q'
# are treated as different keys.


from Graphics import *


def displayq():
   clear()
   setRandomColor()
   drawString("You pressed lowercase q.",100,150,"Arial",28,"bold")
   update()

def displayQ():
   clear()
   setRandomColor()
   drawString("You pressed CAPITAL Q.",100,150,"Arial",28,"bold")
   update()



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


beginGrfx(800,500)
listen()
onkey(displayq,"q")
onkey(displayQ,"Q")
endGrfx()
```



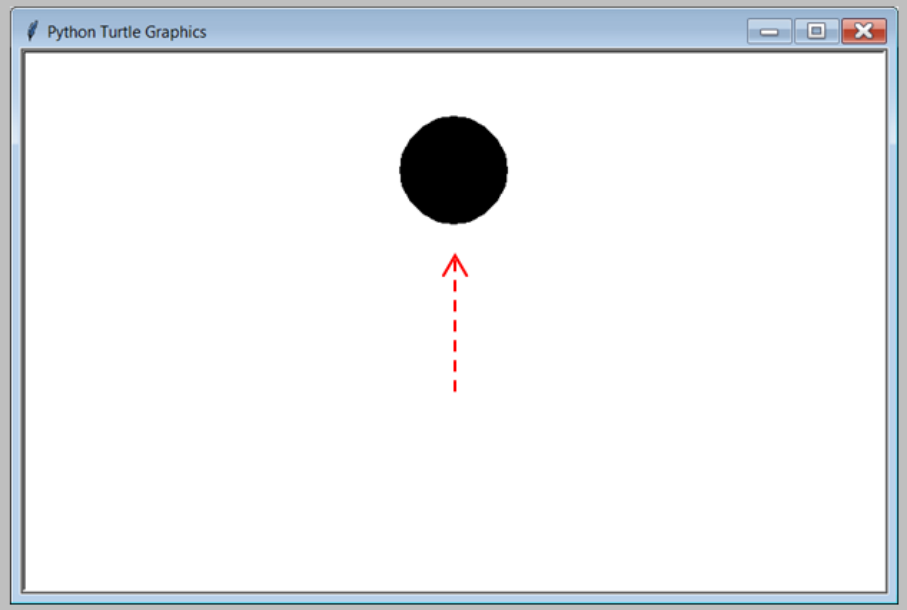
```python
# KeyEvents03.py
# This program controls the movement of a
# circle on the screen using the letters:
#     w
#   a s
#     z


from Graphics import *


def moveUp():
    global y
    y -= 10
    clear()
    fillCircle(x,y,50)
    update()

def moveDown():
    global y
    y += 10
    clear()
    fillCircle(x,y,50)
    update()

def moveLeft():
    global x
    x -= 10
    clear()
    fillCircle(x,y,50)
    update()

def moveRight():
    global x
    x += 10
    clear()
    fillCircle(x,y,50)
    update()



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


x = 400
y = 250

beginGrfx(800,500)
listen()
onkey(moveUp,"w")
onkey(moveDown,"z")
onkey(moveLeft,"a")
onkey(moveRight,"s")
fillCircle(x,y,50)
endGrfx()
```



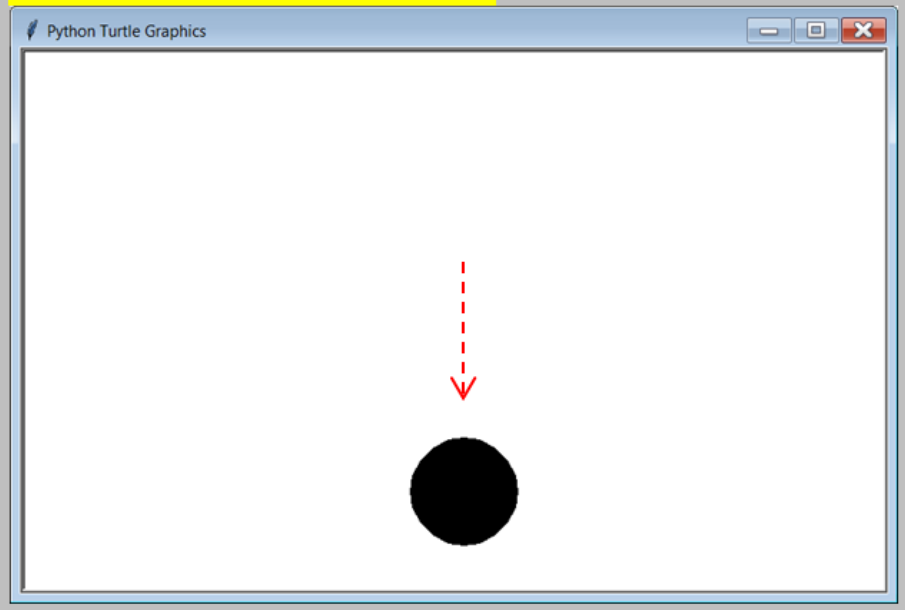
Press '**s**' to move right.



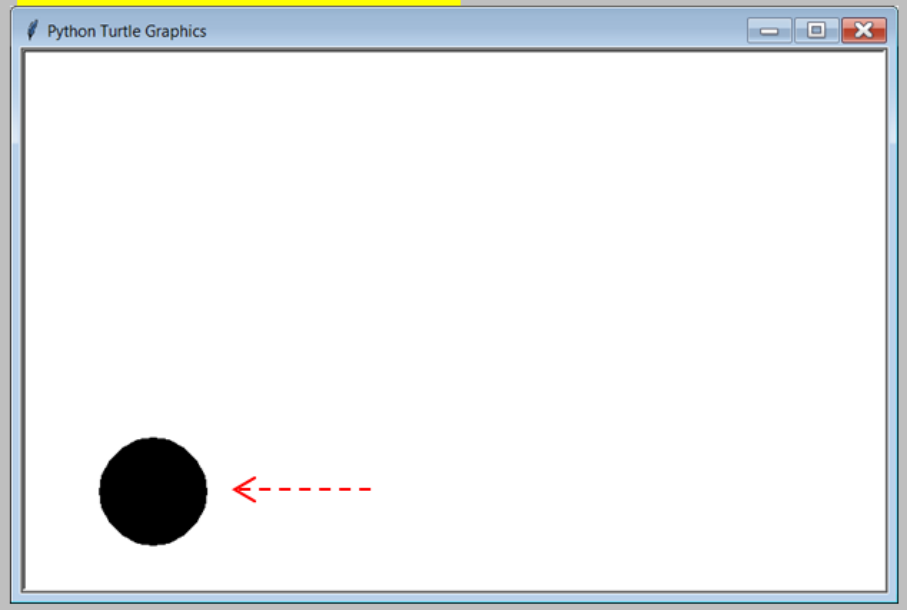
Press '**w**' to move up.



Press '**z**' to move down.



Press '**a**' to move left.

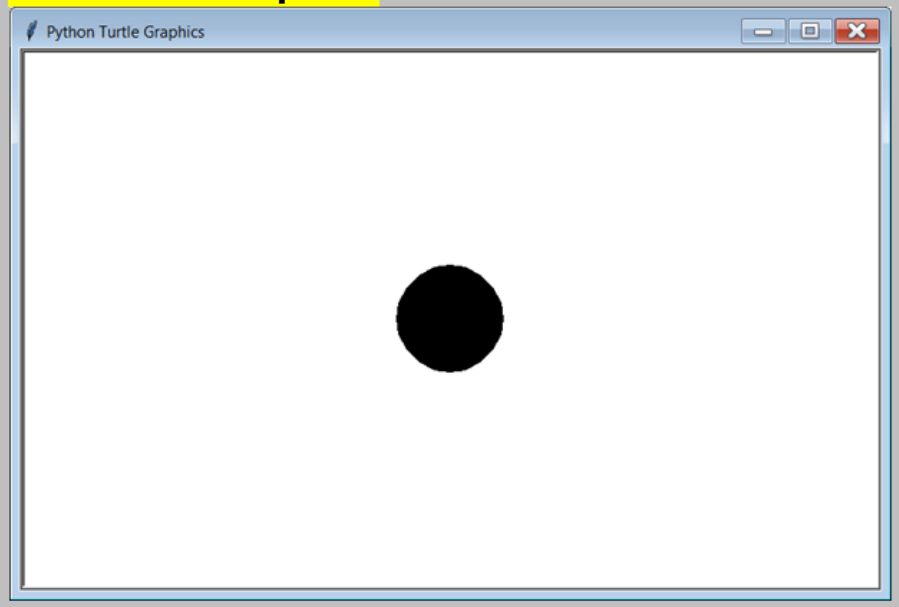


```python
# KeyEvents04.py
# This program shows that <onkey> works with
# "Special Keys" as well like the arrow keys,
# as well as "Insert", "Delete" and "Home".


from Graphics import *


def moveUp():
    global y
    y -= 10
    clear()
    fillCircle(x,y,r)
    update()

def moveDown():
    global y
    y += 10
    clear()
    fillCircle(x,y,r)
    update()

def moveLeft():
    global x
    x -= 10
    clear()
    fillCircle(x,y,r)
    update()

def moveRight():
    global x
    x += 10
    clear()
    fillCircle(x,y,r)
    update()
```

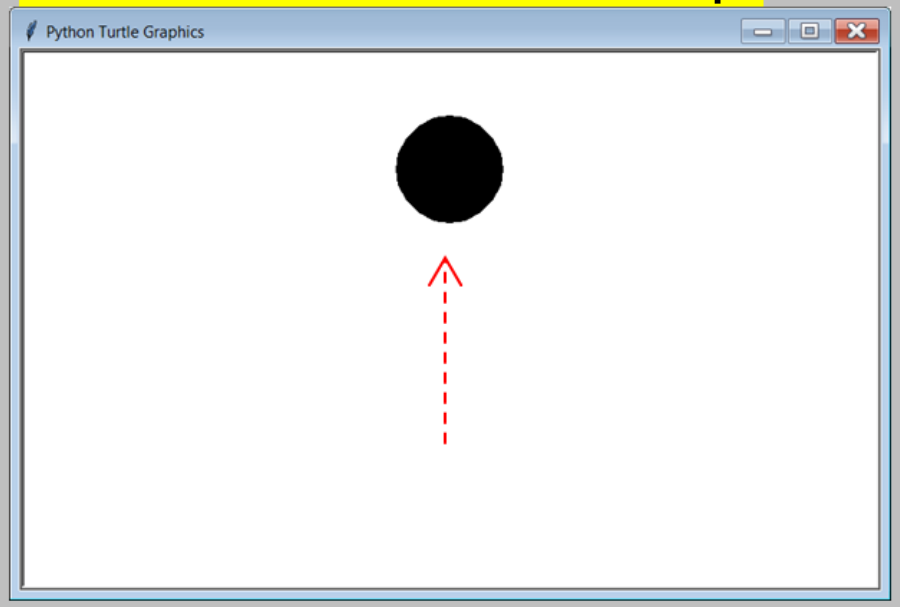
```python
def bigger():
    global r
    r += 10
    clear()
    fillCircle(x,y,r)
    update()

def smaller():
    global r
    r -= 10
    if r < 0:
        r = 0
    clear()
    fillCircle(x,y,r)
    update()

def center():
    global x,y,r
    x = 400
    y = 250
    r = 50
    clear()
    fillCircle(x,y,r)
    update()
```

```python
##########
#  MAIN  #
##########


x = 400
y = 250
r = 50

beginGrfx(800,500)
listen()
onkey(moveUp,"Up")
onkey(moveDown,"Down")
onkey(moveLeft,"Left")
onkey(moveRight,"Right")
onkey(bigger,"Insert")
onkey(smaller,"Delete")
onkey(center,"Home")
fillCircle(x,y,r)
endGrfx()
```

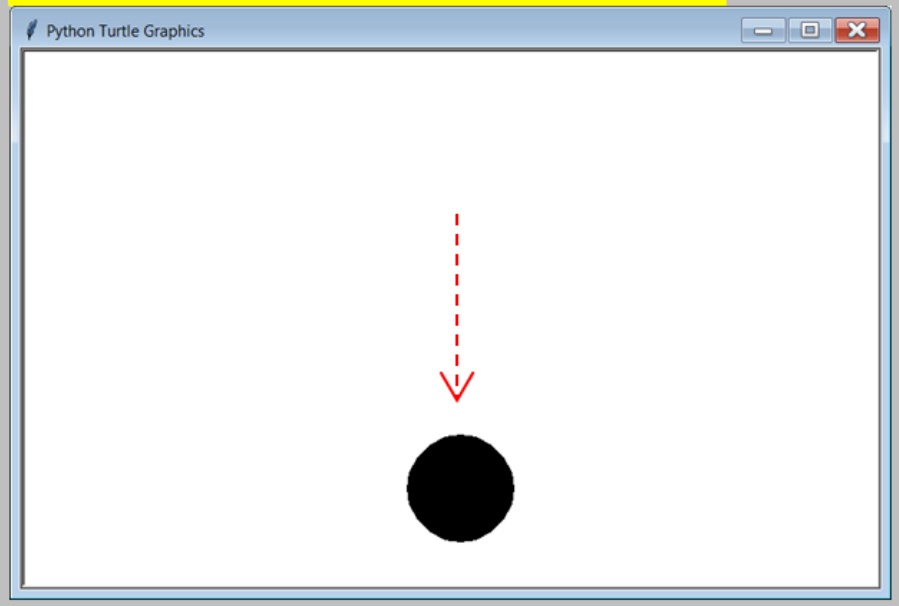
Initial Output:



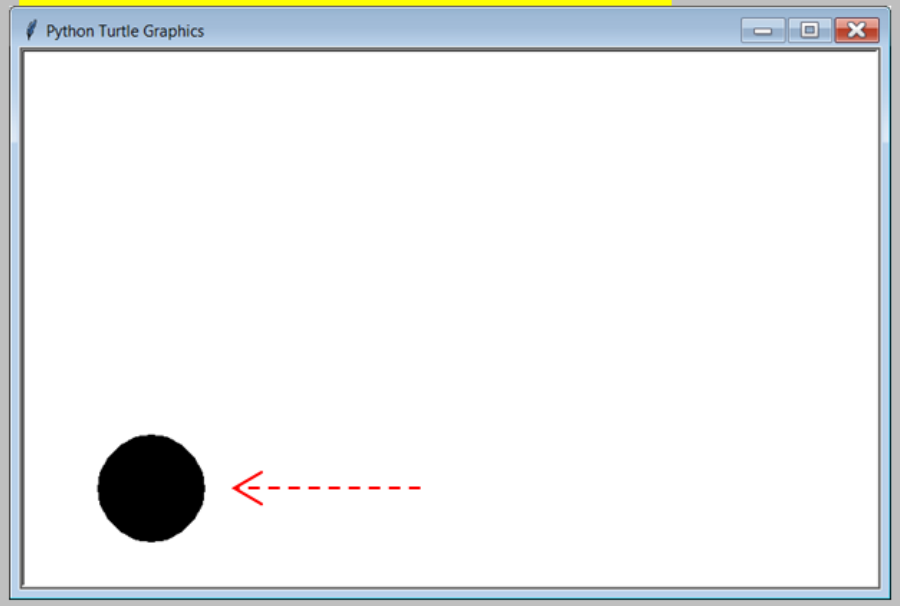
Press the <↑> to move up.



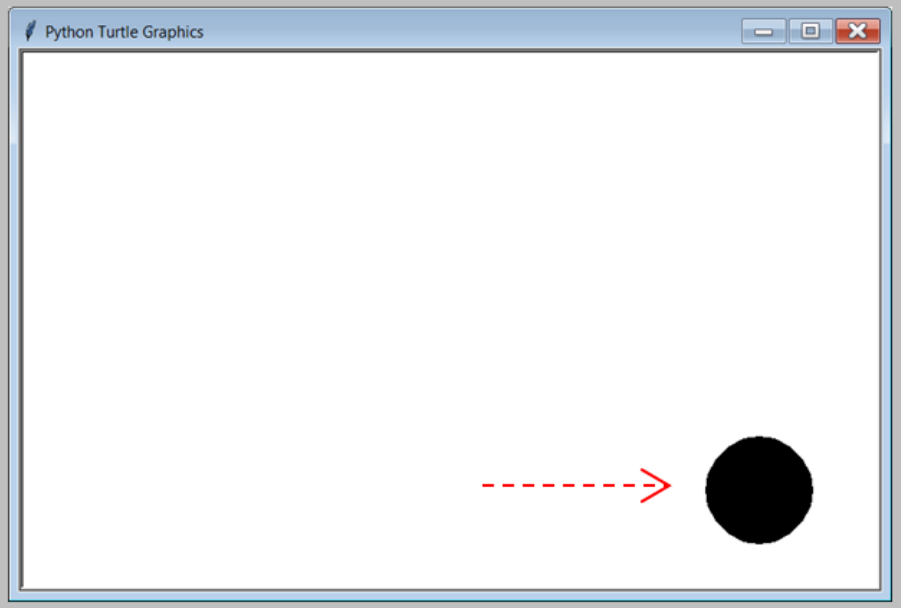
Press <↓> to move down.



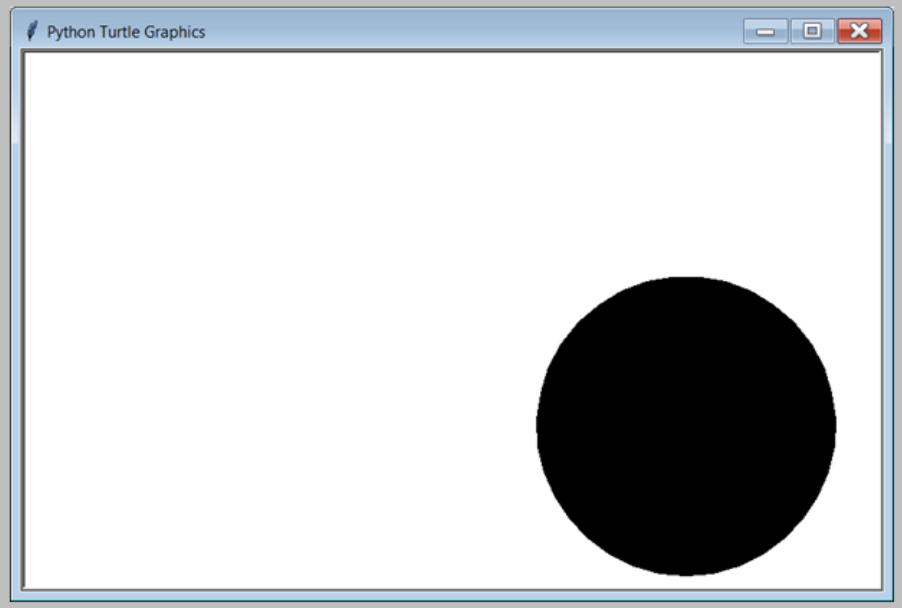
Press <←> to move left.



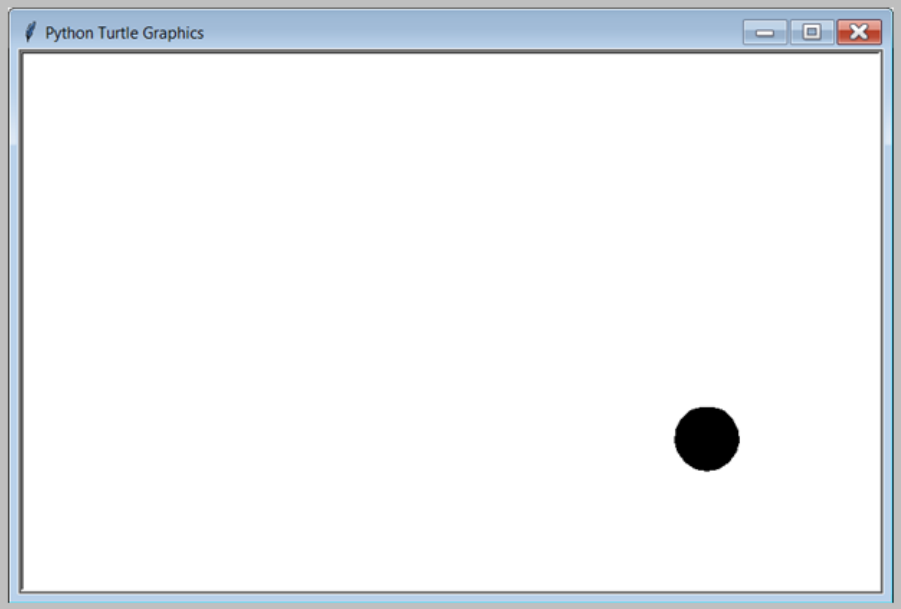
Press <→> to move right.



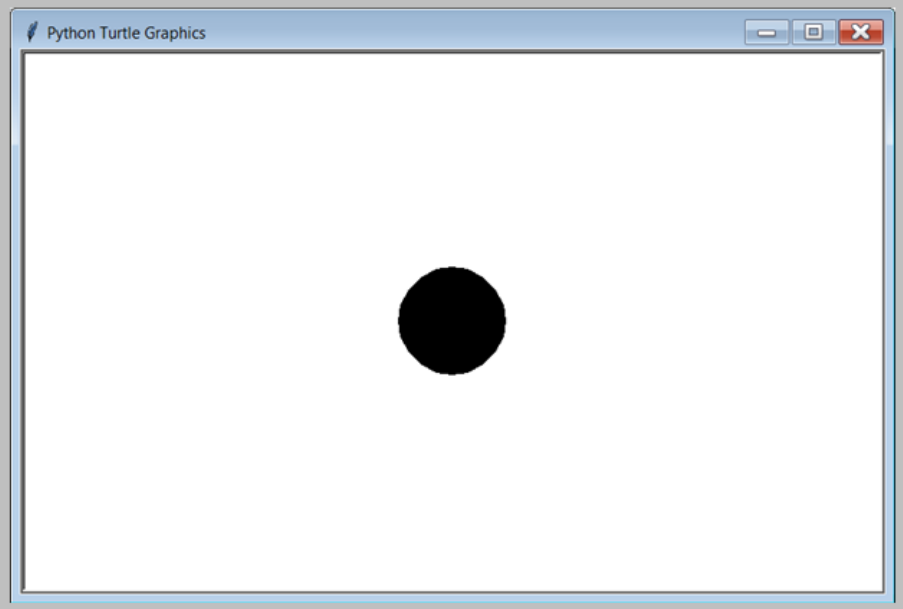
<Insert> makes circle bigger.



<Delete> makes circle smaller.



<Home> resets the screen.



```
1 # KeyEvents05.py
2 # This program replaces most of the <onkey>
3 # commands with <onkeypress>.  This allows
4 # the user to simply hold the key down rather
5 # than having to type it repeatedly.
6
```

⋮ ⋮ ⋮ ⋮ ⋮ ⋮ ⋮ ⋮

```
74
75 beginGrfx(800,500)
76 listen()
77 onkeypress(moveUp,"Up")
78 onkeypress(moveDown,"Down")
79 onkeypress(moveLeft,"Left")
80 onkeypress(moveRight,"Right")
81 onkeypress(bigger,"Insert")
82 onkeypress(smaller,"Delete")
83 onkey(center,"Home")
84 fillCircle(x,y,r)
85 endGrfx()
```

```
# KeyEvents06.py
# This program demonstrates that the string literal
# "key symbols" used for the "Special Keys" are
# "Case-Sensitive".
```

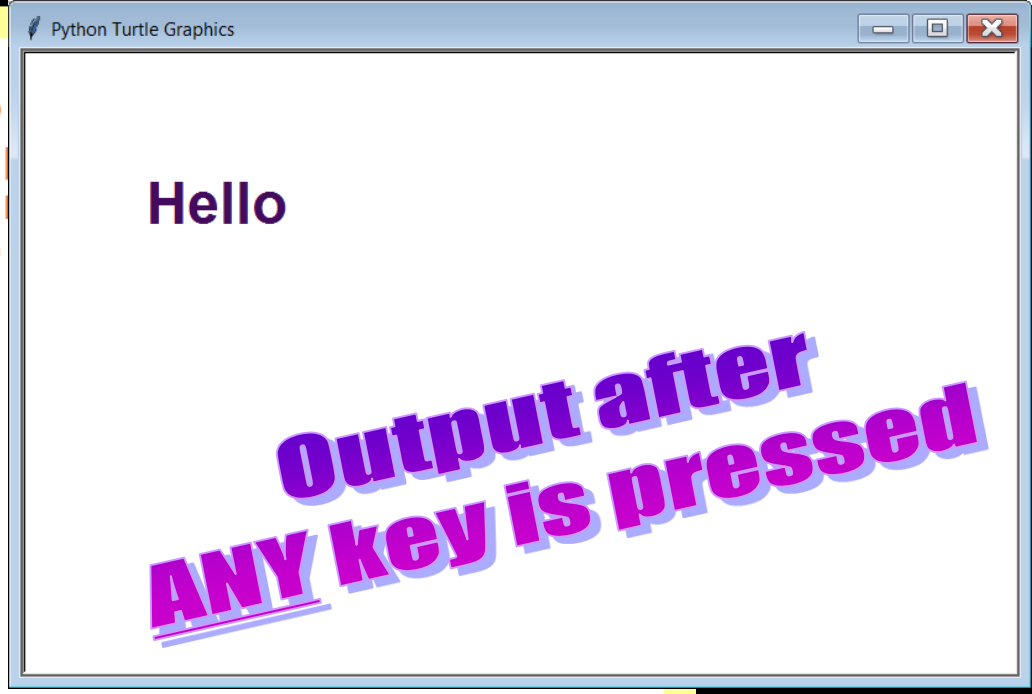
```
:       :       :       :       :       :       :       :
```

```
beginGrfx(800,500)
listen()
onkeypress(moveUp,"up")  # should be "Up"
onkeypress(moveDown,"Down")
onkeypress(moveLeft,"Left")
onkeypress(moveRight,"Right")
onkeypress(bigger,"Insert")
onkeypress(smaller,"Delete")
onkey(center,"Home")
fillCircle(x,y,r)
endGrfx()
```

```
 ----jGRASP exec: python KeyEvents06.py
Traceback (most recent call last):
  File "KeyEvents06.py", line 76, in <module>
    onkeypress(moveUp,"up")
  File "<string>", line 8, in onkeypress
  File "turtle.py", line 1426, in onkeypress
    self._onkeypress(fun, key)
  File "turtle.py", line 705, in _onkeypress
    self.cv.bind("<KeyPress-%s>" % key, eventfun)
  File "turtle.py", line 416, in bind
    self._canvas.bind(*args, **kwargs)
  File "__init__.py", line 1245, in bind
    return self._bind(('bind', self._w), sequence, func, add)
  File "__init__.py", line 1200, in _bind
    self.tk.call(what + (sequence, cmd))
_tkinter.TclError: bad event type or keysym "up"

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

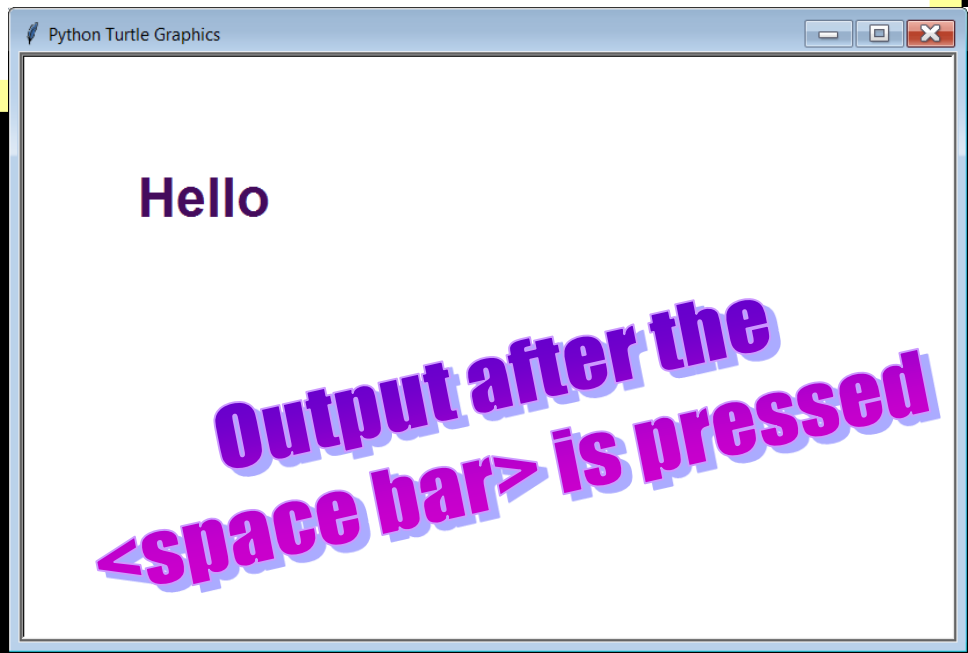
```
# KeyEvents07.py
# This program attempts to use the "space bar"
# to make the computer display "Hello".  It may
# seem to work at first, until you realize that
# typing ANY key will make the computer display
# "Hello".


from Graphics import *


def display():
   clear()
   setRandomColor()
   drawString("Hello",100,150,"Arial",28,"bold")
   update()



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


beginGrfx(800,500)
listen()
onkeypress(display," ")
endGrfx()
```

```python
# KeyEvents07.py
# This program attempts to
# to make the computer dis
# seem to work at first, u
# typing ANY key will make
# "Hello".


from Graphics import *


def display():
    clear()
    setRandomColor()
    drawString("Hello",100,150,"Arial",28,"bold")
    update()



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


beginGrfx(800,500)
listen()
onkeypress(display," ")
endGrfx()
```



```
1 # KeyEvents07.py
2 # This program corrects the issue of the previous
3 # program by using the string literal key symbol
4 # "space" instead of " ".
```

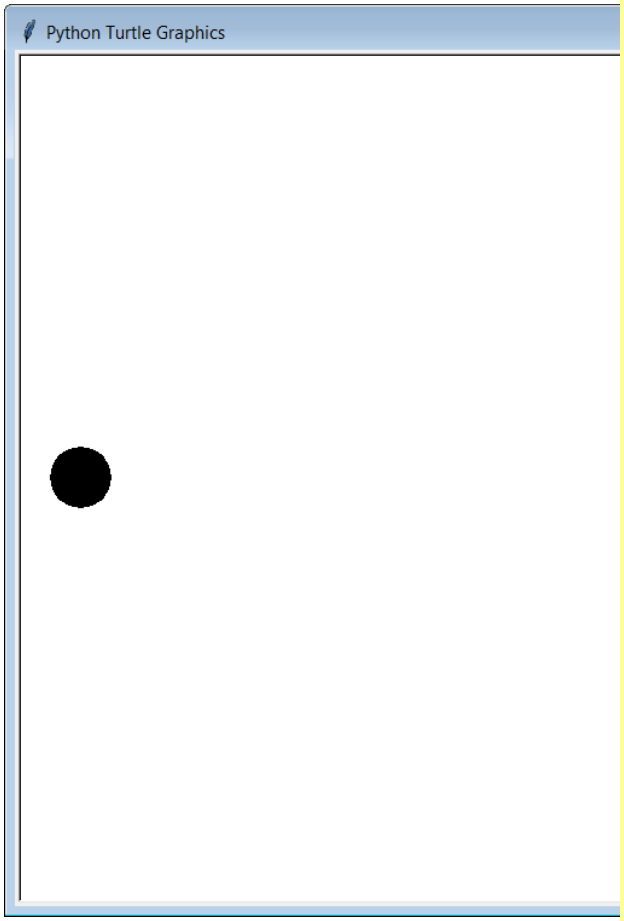
```
23 beginGrfx(800,500)
24 listen()
25 onkeypress(display,"space")
26 endGrfx()
```



Section 18.6

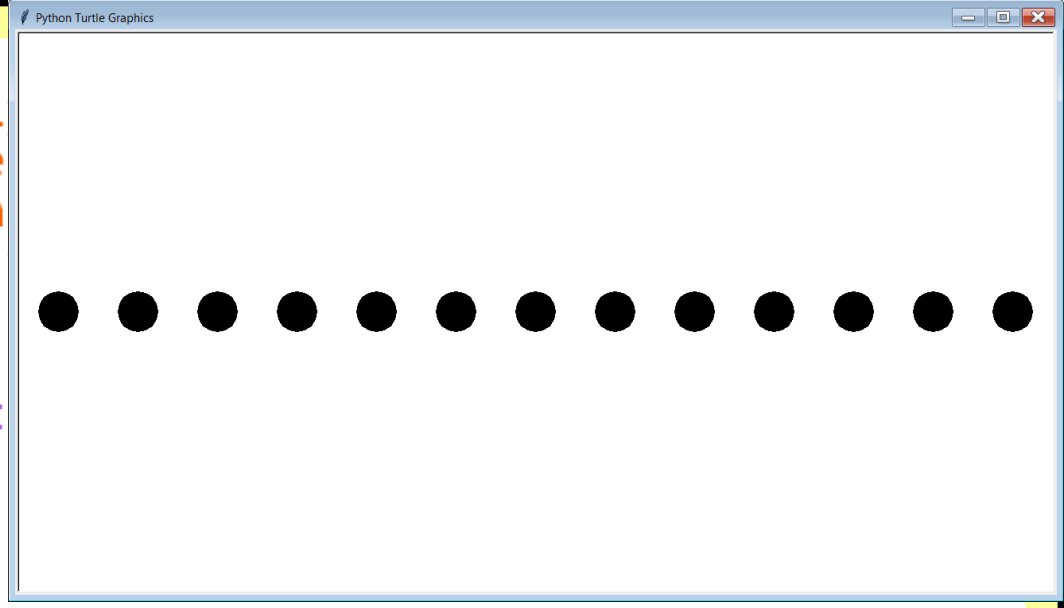
COMPUTER ANIMATION

```python
# Animation01.py
# This program simply draws a solid black circle.
# This will eventually be "animated" as we go
# through the next few programs.


from Graphics import *


beginGrfx(1300,700)

fillCircle(50,350,25)

endGrfx()

```

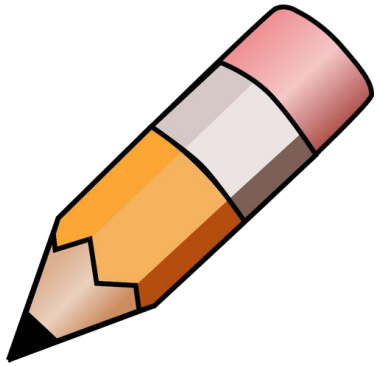


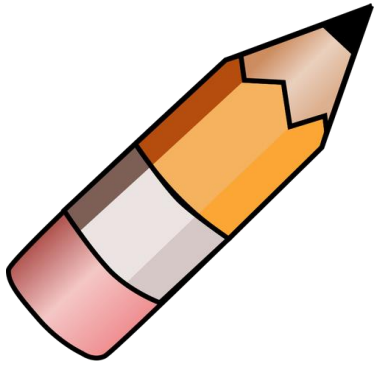
```python
# Animation02.py
# This program simply draws the solid black
# circle at 13 different locations.
# This is not yet animation because we see
# all 13 circles at the same time.


from Graphics import *


beginGrfx(1300,700)

fillCircle(50,350,25)
fillCircle(150,350,25)
fillCircle(250,350,25)
fillCircle(350,350,25)
fillCircle(450,350,25)
fillCircle(550,350,25)
fillCircle(650,350,25)
fillCircle(750,350,25)
fillCircle(850,350,25)
fillCircle(950,350,25)
fillCircle(1050,350,25)
fillCircle(1150,350,25)
fillCircle(1250,350,25)

endGrfx()
```

```
1  # Animation02.py
2  # This program simpl
3  # circle at 13 diffe
4  # This is not yet an
5  # all 13 circles at
6
7
8  from Graphics import
9
10
11 beginGrfx(1300,700)
12
13 fillCircle(50,350,25)
14 fillCircle(150,350,25)
15 fillCircle(250,350,25)
16 fillCircle(350,350,25)
17 fillCircle(450,350,25)
18 fillCircle(550,350,25)
19 fillCircle(650,350,25)
20 fillCircle(750,350,25)
21 fillCircle(850,350,25)
22 fillCircle(950,350,25)
23 fillCircle(1050,350,25)
24 fillCircle(1150,350,25)
25 fillCircle(1250,350,25)
26
27 endGrfx()
```

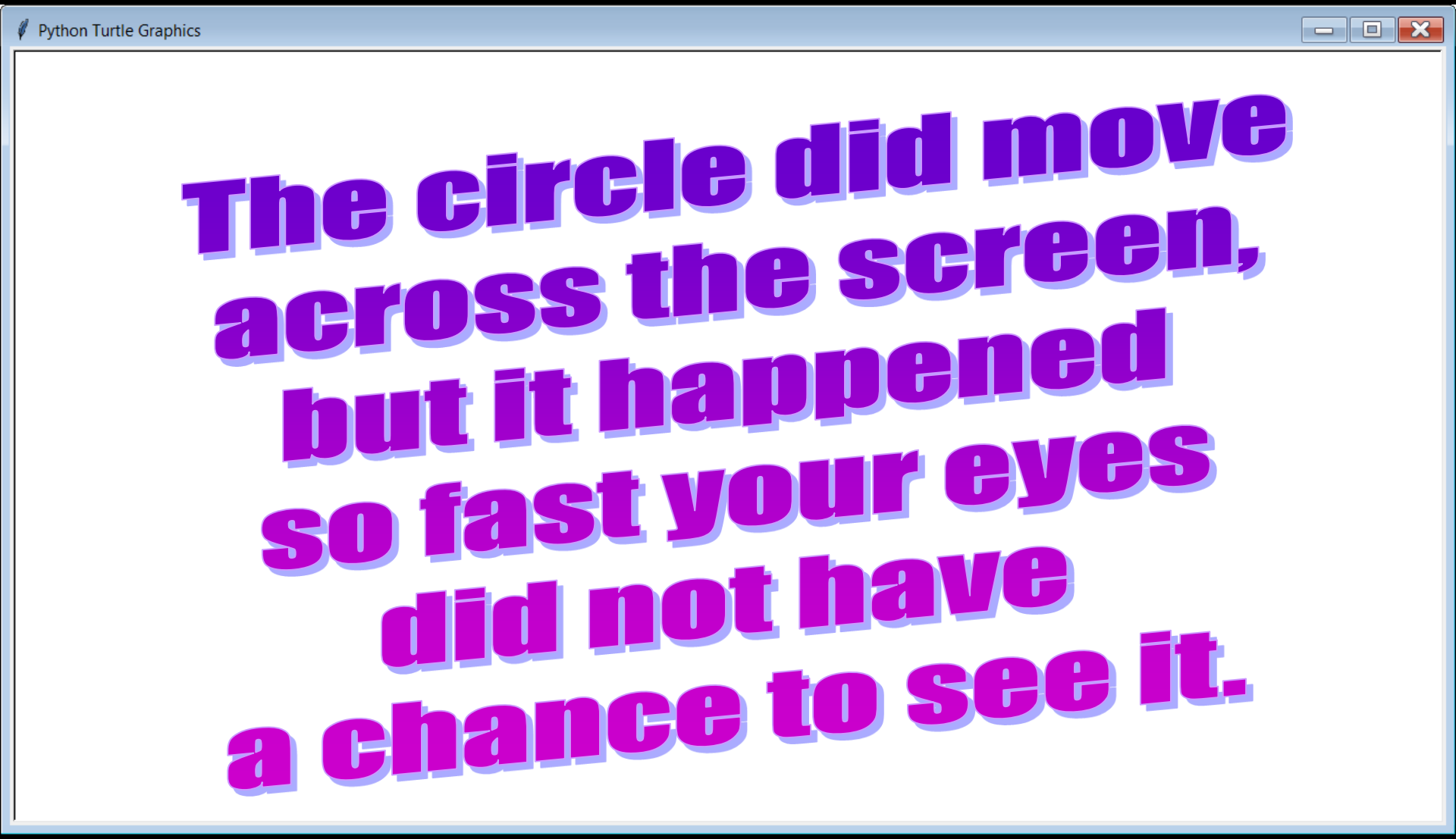


```python
# Animation03.py
# This program erases every circle before it draws the
# next one.  This is necessary for animation; however,
# the process happens so fast that you cannot see the
# circle move.
# NOTE: On a Mac, you may see some circle outlines left
# behind because it did not completely erase the circles.


from Graphics import *


beginGrfx(1300,700)

setColor("black")
fillCircle(50,350,25)
setColor("white")
fillCircle(50,350,25)

setColor("black")
fillCircle(150,350,25)
setColor("white")
fillCircle(150,350,25)

setColor("black")
fillCircle(250,350,25)
setColor("white")
fillCircle(250,350,25)

setColor("black")
fillCircle(350,350,25)
setColor("white")
fillCircle(350,350,25)
setColor("black")
fillCircle(450,350,25)
setColor("white")
fillCircle(450,350,25)

setColor("black")
fillCircle(550,350,25)
setColor("white")
fillCircle(550,350,25)

setColor("black")
fillCircle(650,350,25)
setColor("white")
fillCircle(650,350,25)

setColor("black")
fillCircle(750,350,25)
setColor("white")
fillCircle(750,350,25)

setColor("black")
fillCircle(850,350,25)
setColor("white")
fillCircle(850,350,25)

setColor("black")
fillCircle(950,350,25)
setColor("white")
fillCircle(950,350,25)

setColor("black")
fillCircle(1050,350,25)
setColor("white")
fillCircle(1050,350,25)

setColor("black")
fillCircle(1150,350,25)
setColor("white")
fillCircle(1150,350,25)

setColor("black")
fillCircle(1250,350,25)
setColor("white")
fillCircle(1250,350,25)

endGrfx()
```

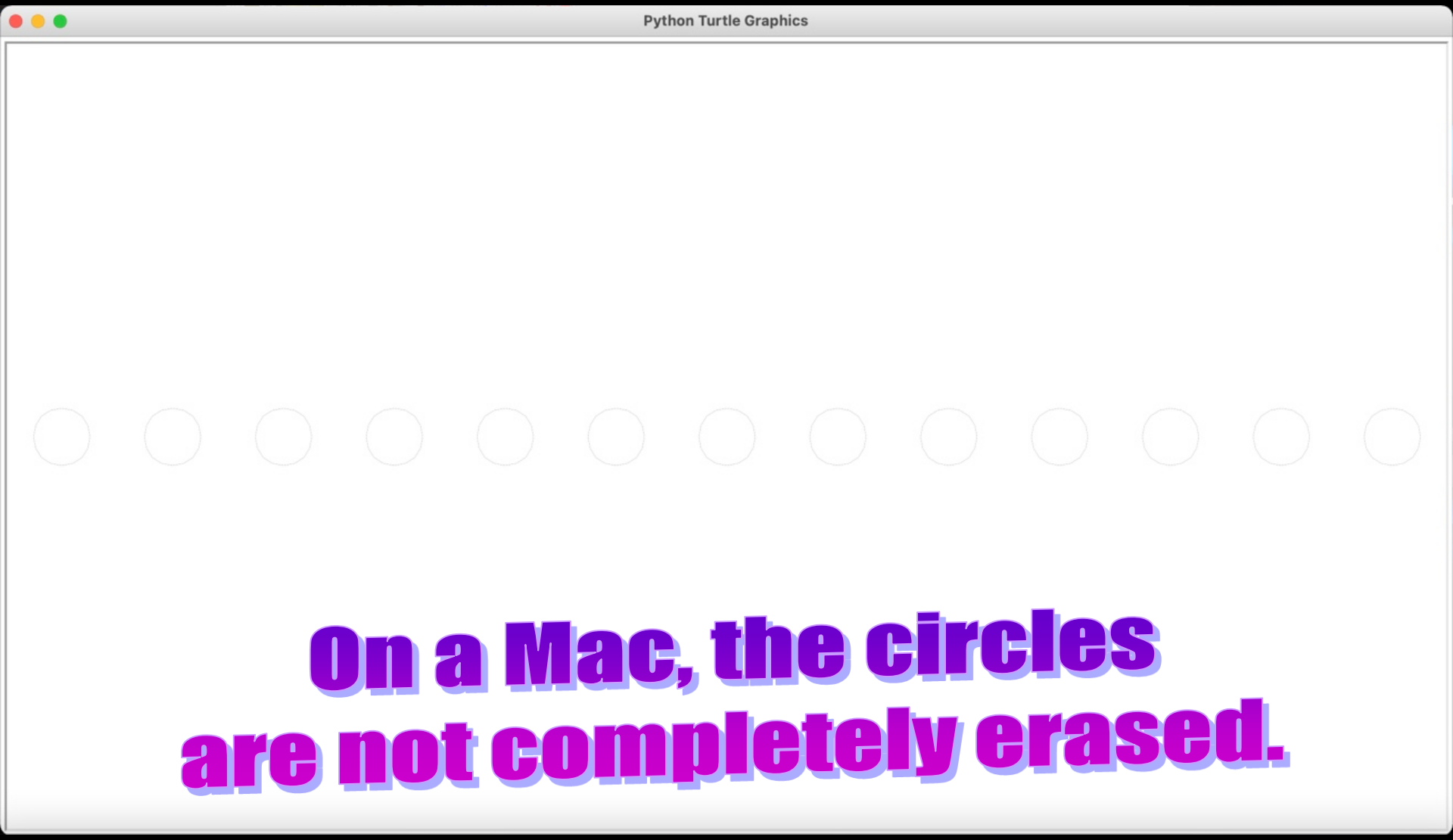




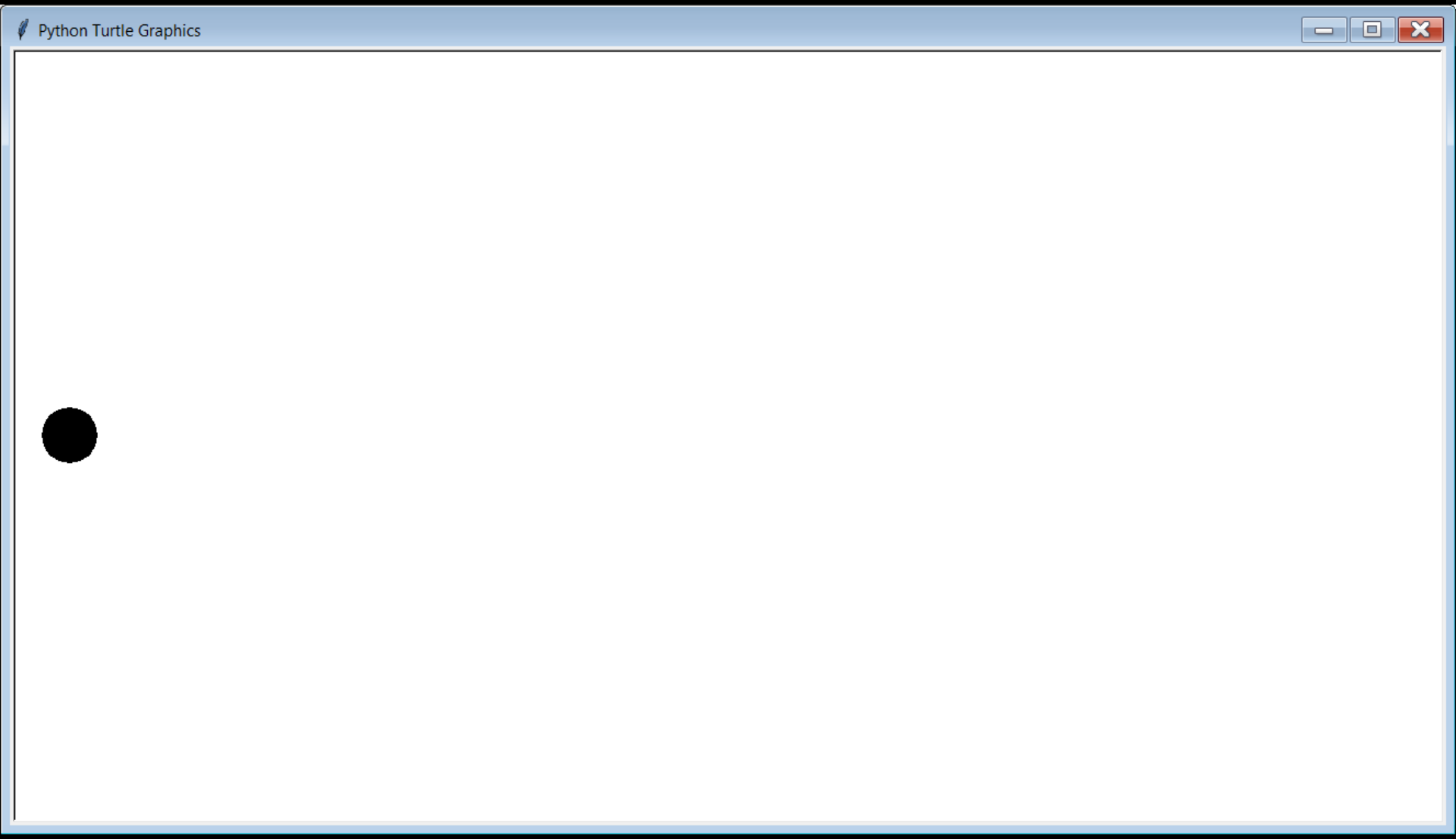
Windows PC Output:

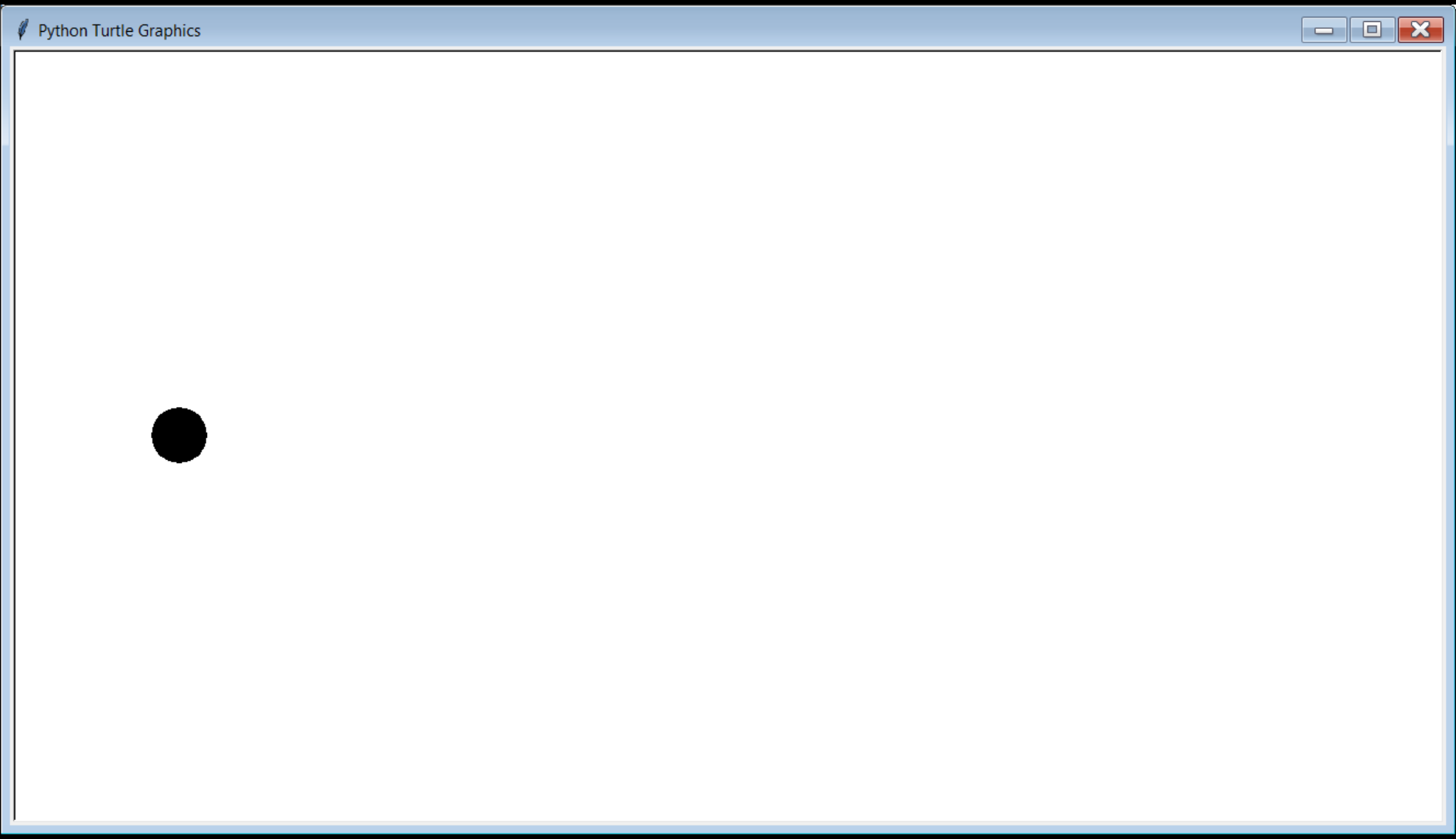


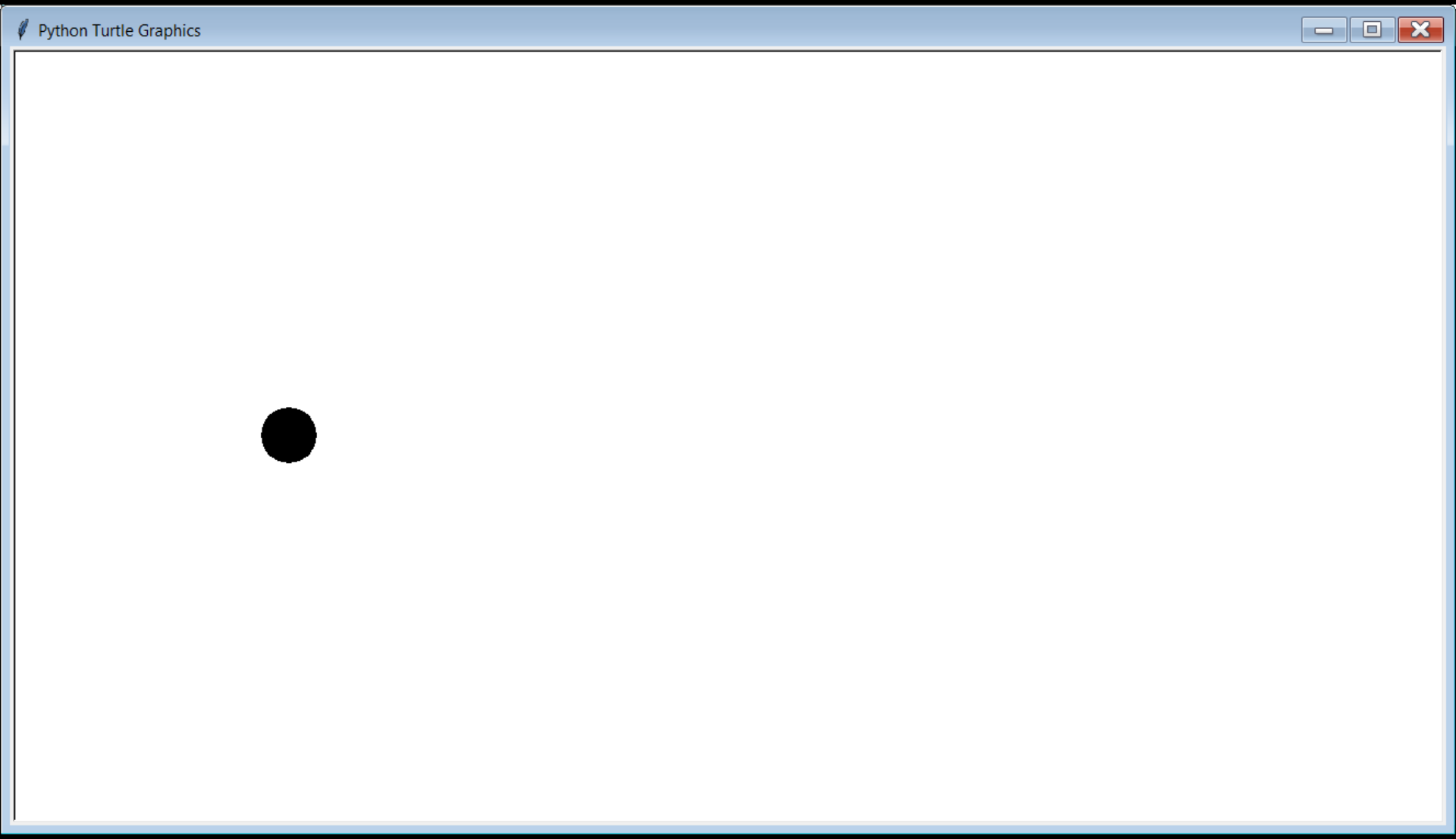
Mac Output:

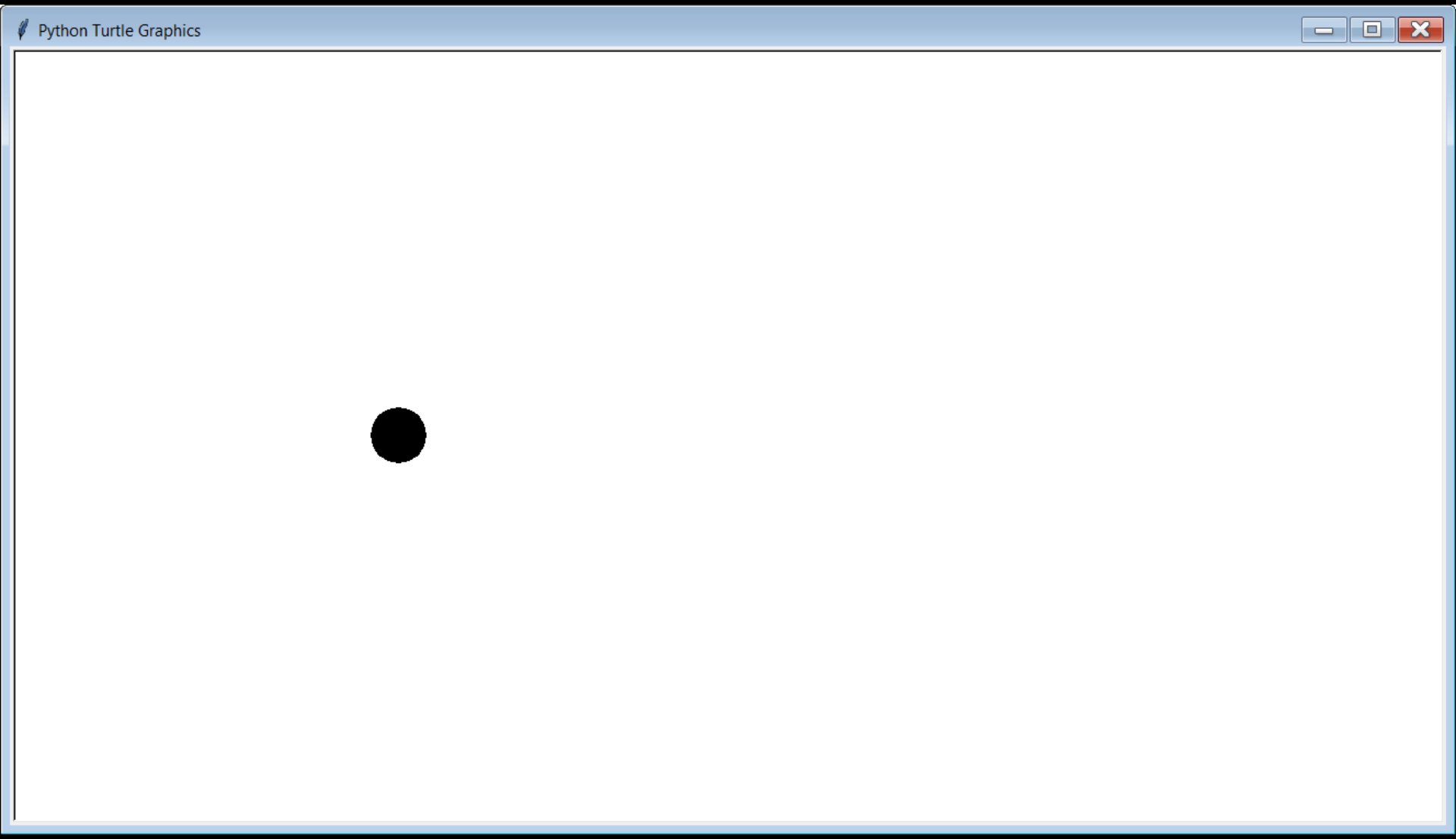

Python Turtle Graphics
On a Mac, the circles
are not completely erased.

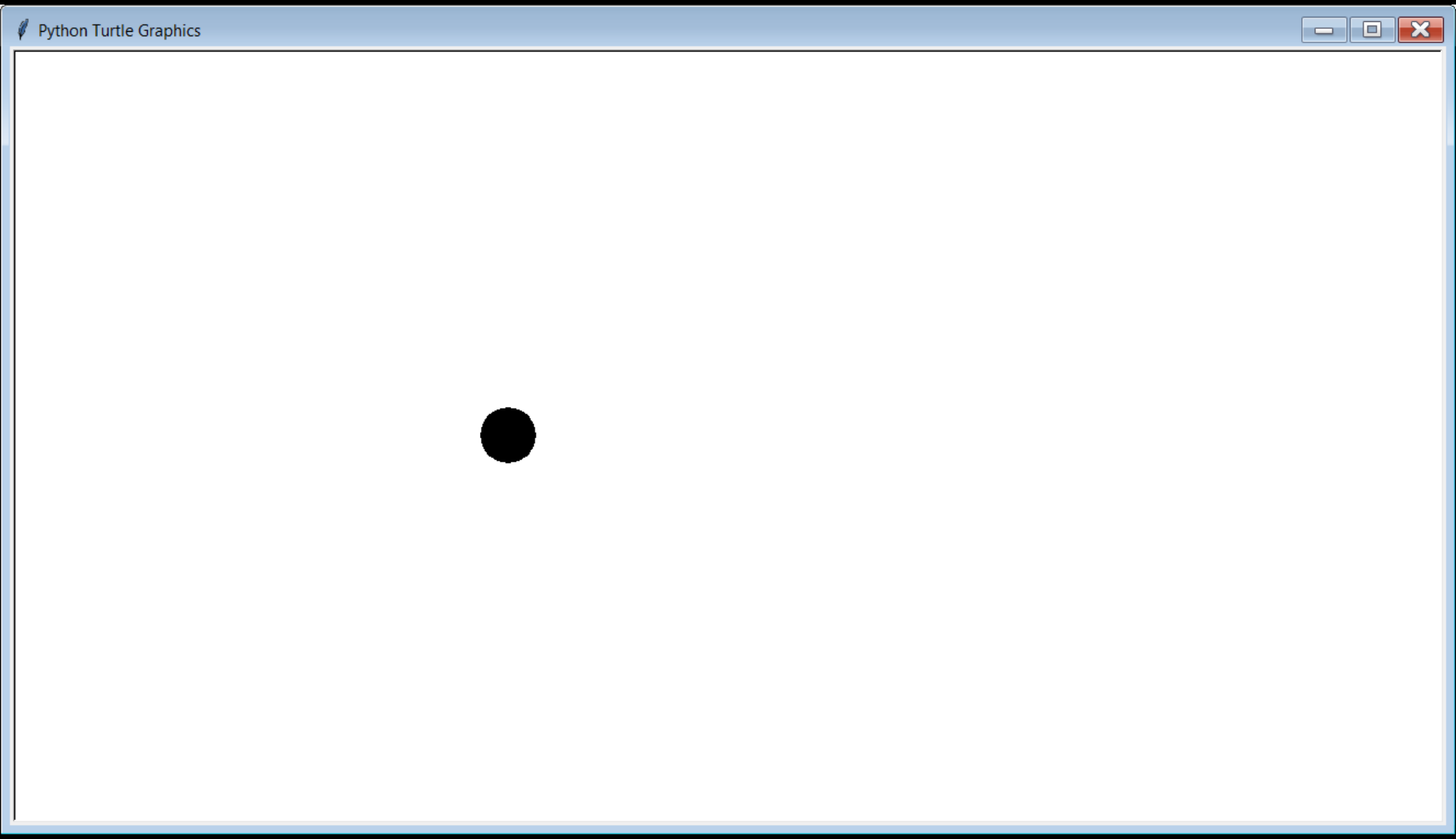
```
# Animation04.py
# This program adds a short 1 second delay
# after each circle is drawn to give you a
# chance to see it before it is erased.
# NOTE: This is called "Draw-And-Erase Animation".
# ALSO: The issue of the circle outlines on a Mac
#       was fixed by making the radii of the
#       erasing circles a little bigger.


from Graphics import *


beginGrfx(1300,700)

setColor("black")
fillCircle(50,350,25)
delay(1000)
setColor("white")
fillCircle(50,350,27)

setColor("black")
fillCircle(150,350,25)
delay(1000)
setColor("white")
fillCircle(150,350,27)

setColor("black")
fillCircle(250,350,25)
delay(1000)
setColor("white")
fillCircle(250,350,27)

setColor("black")
fillCircle(350,350,25)
delay(1000)
setColor("white")
fillCircle(350,350,27)

setColor("black")
fillCircle(450,350,25)
delay(1000)
setColor("white")
fillCircle(450,350,27)

setColor("black")
fillCircle(550,350,25)
delay(1000)
setColor("white")
fillCircle(550,350,27)

setColor("black")
fillCircle(650,350,25)
delay(1000)
setColor("white")
fillCircle(650,350,27)

setColor("black")
fillCircle(750,350,25)
delay(1000)
setColor("white")
fillCircle(750,350,27)

setColor("black")
fillCircle(850,350,25)
delay(1000)
setColor("white")
fillCircle(850,350,27)

setColor("black")
fillCircle(950,350,25)
delay(1000)
setColor("white")
fillCircle(950,350,27)

setColor("black")
fillCircle(1050,350,25)
delay(1000)
setColor("white")
fillCircle(1050,350,27)

setColor("black")
fillCircle(1150,350,25)
delay(1000)
setColor("white")
fillCircle(1150,350,27)

setColor("black")
fillCircle(1250,350,25)
delay(1000)
setColor("white")
fillCircle(1250,350,27)

endGrfx()
```

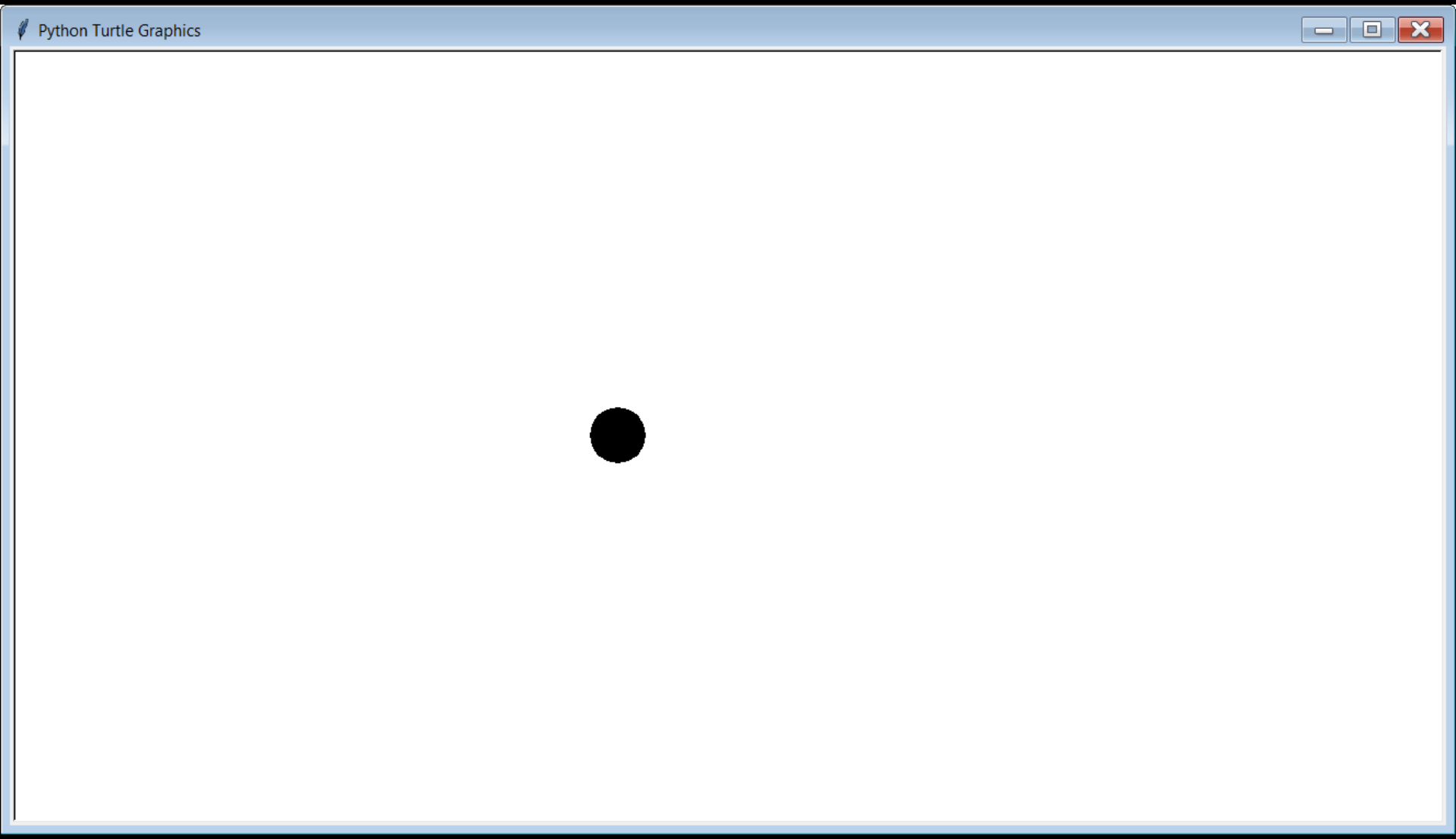

Python Turtle Graphics

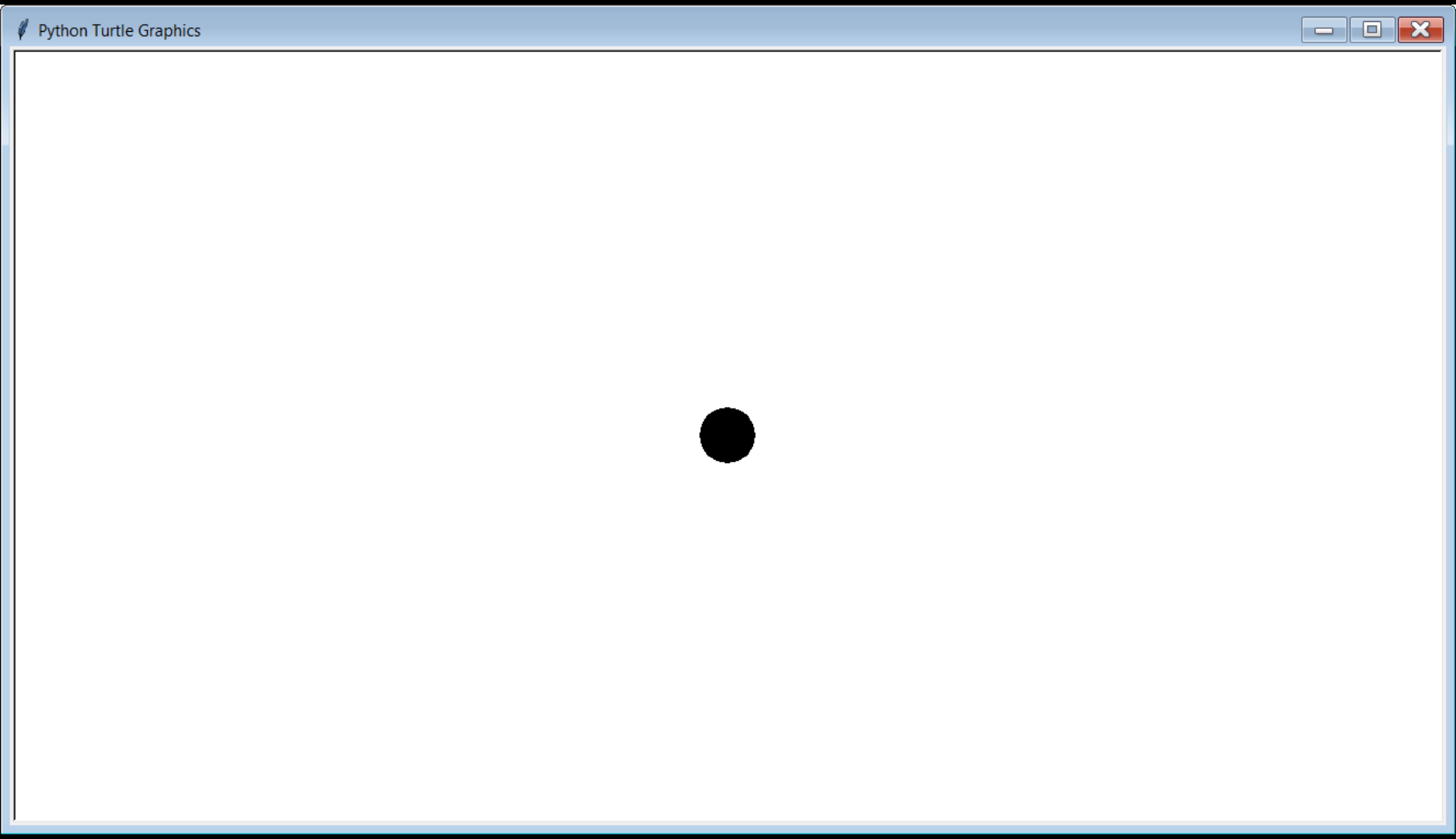

Python Turtle Graphics

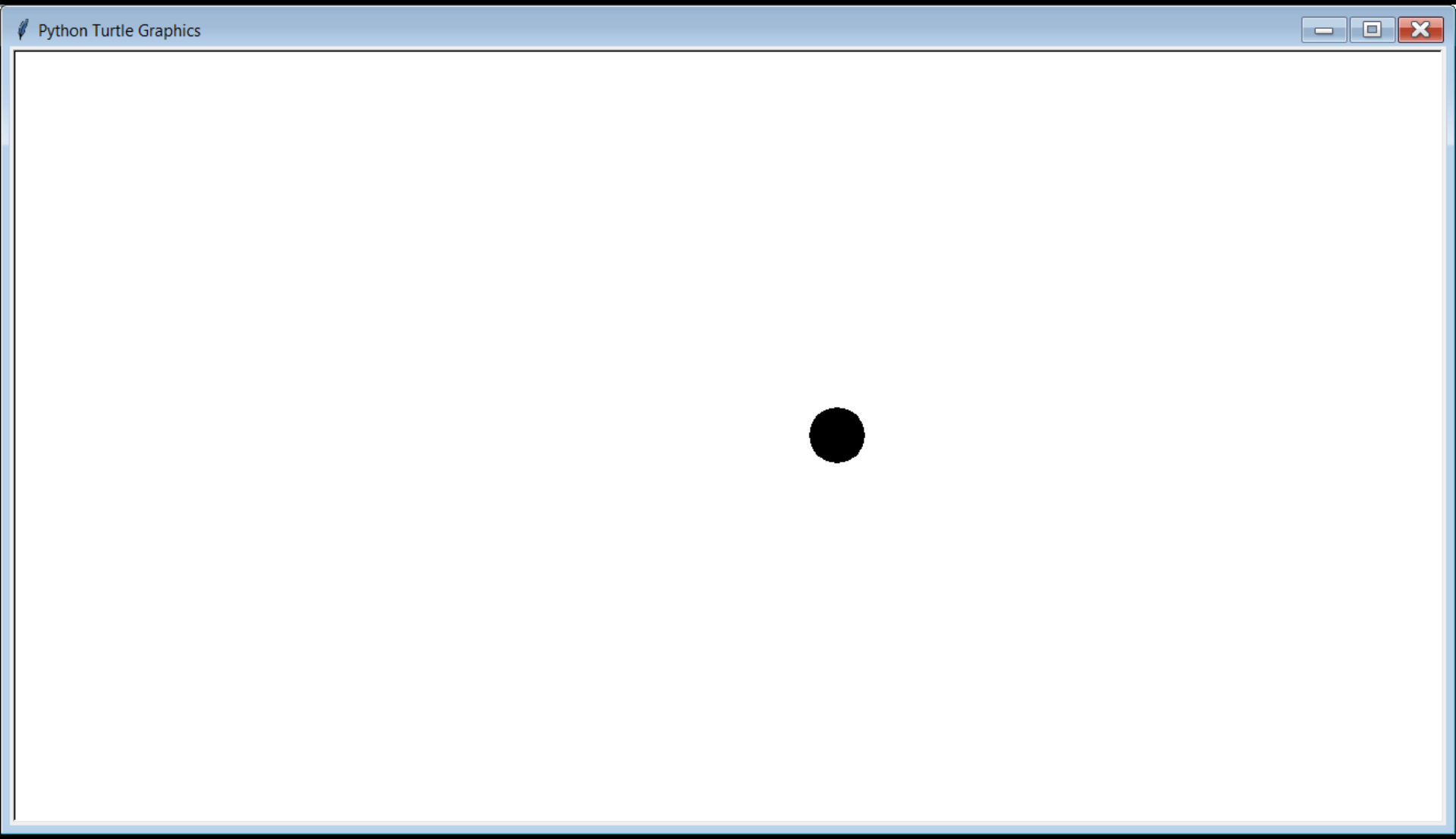

Python Turtle Graphics

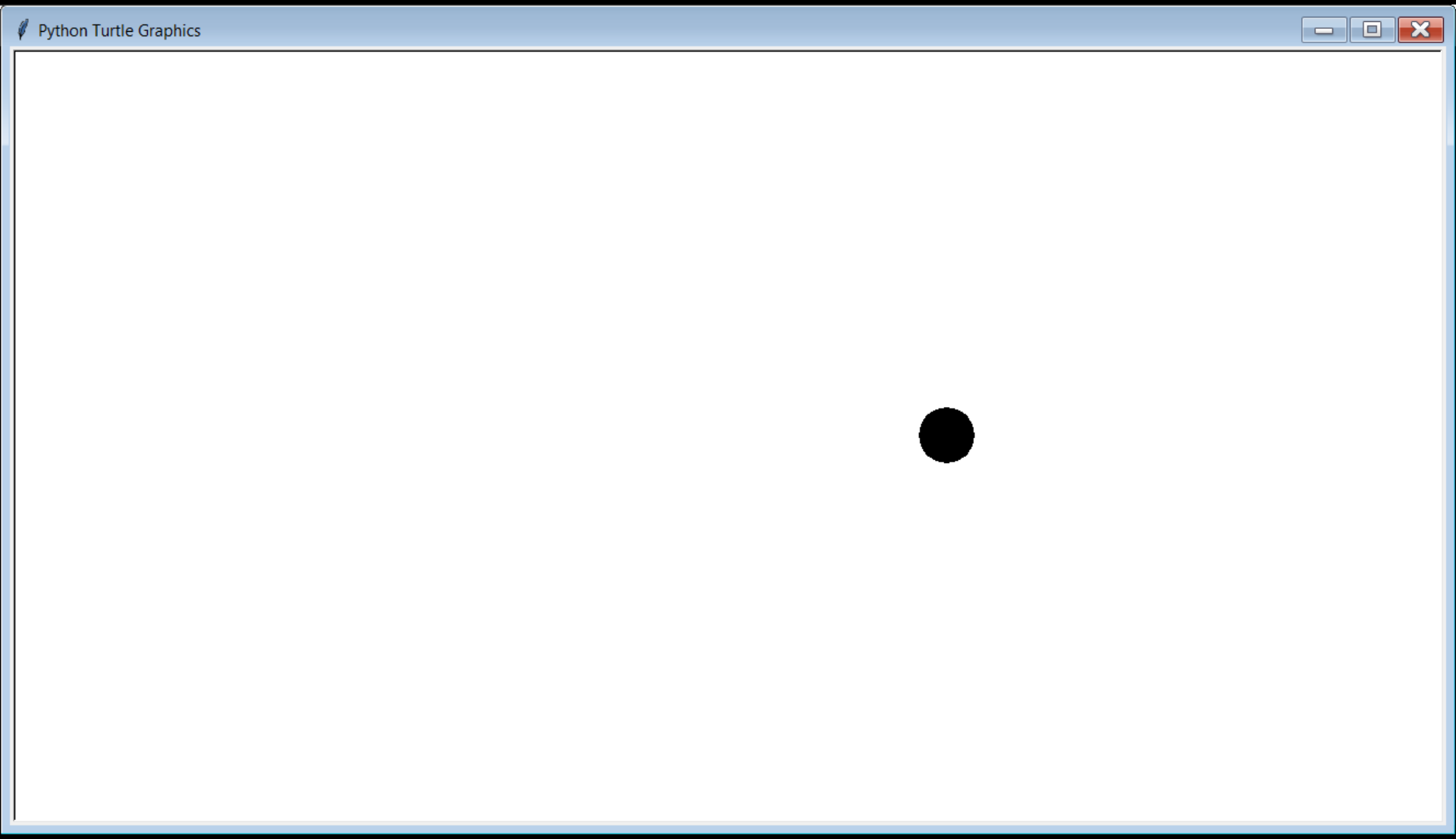

Python Turtle Graphics

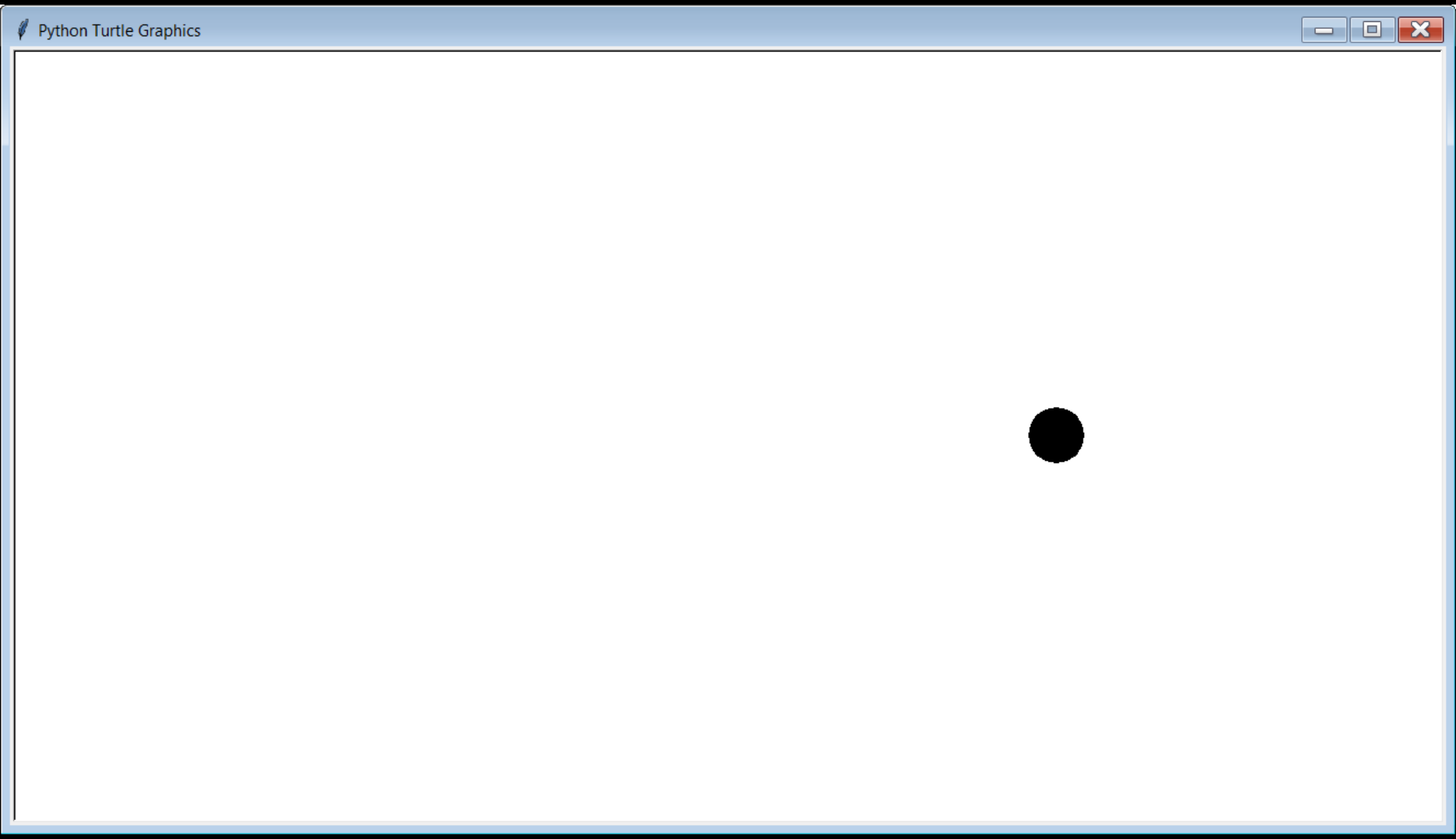

Python Turtle Graphics


Python Turtle Graphics

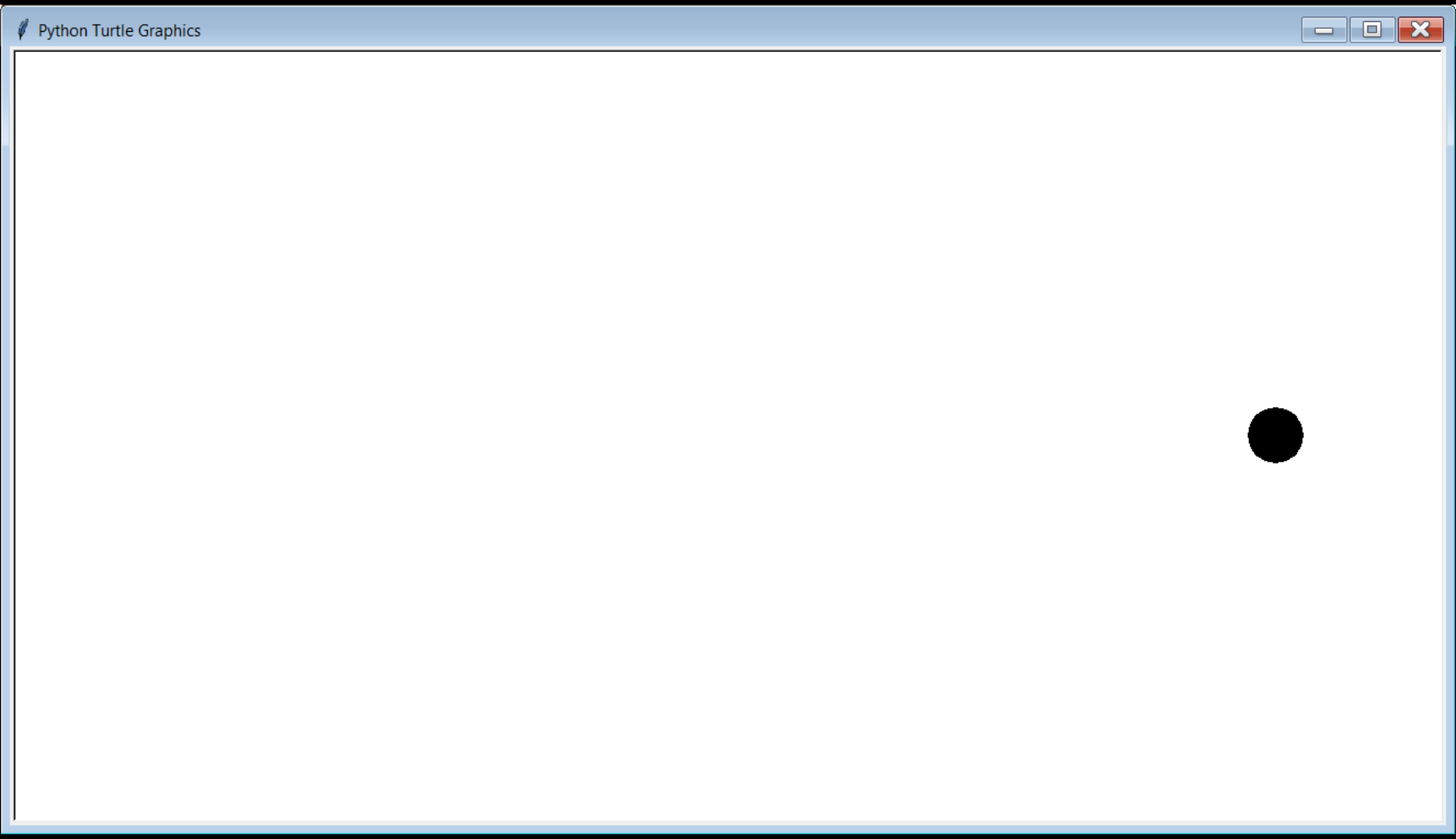

Python Turtle Graphics

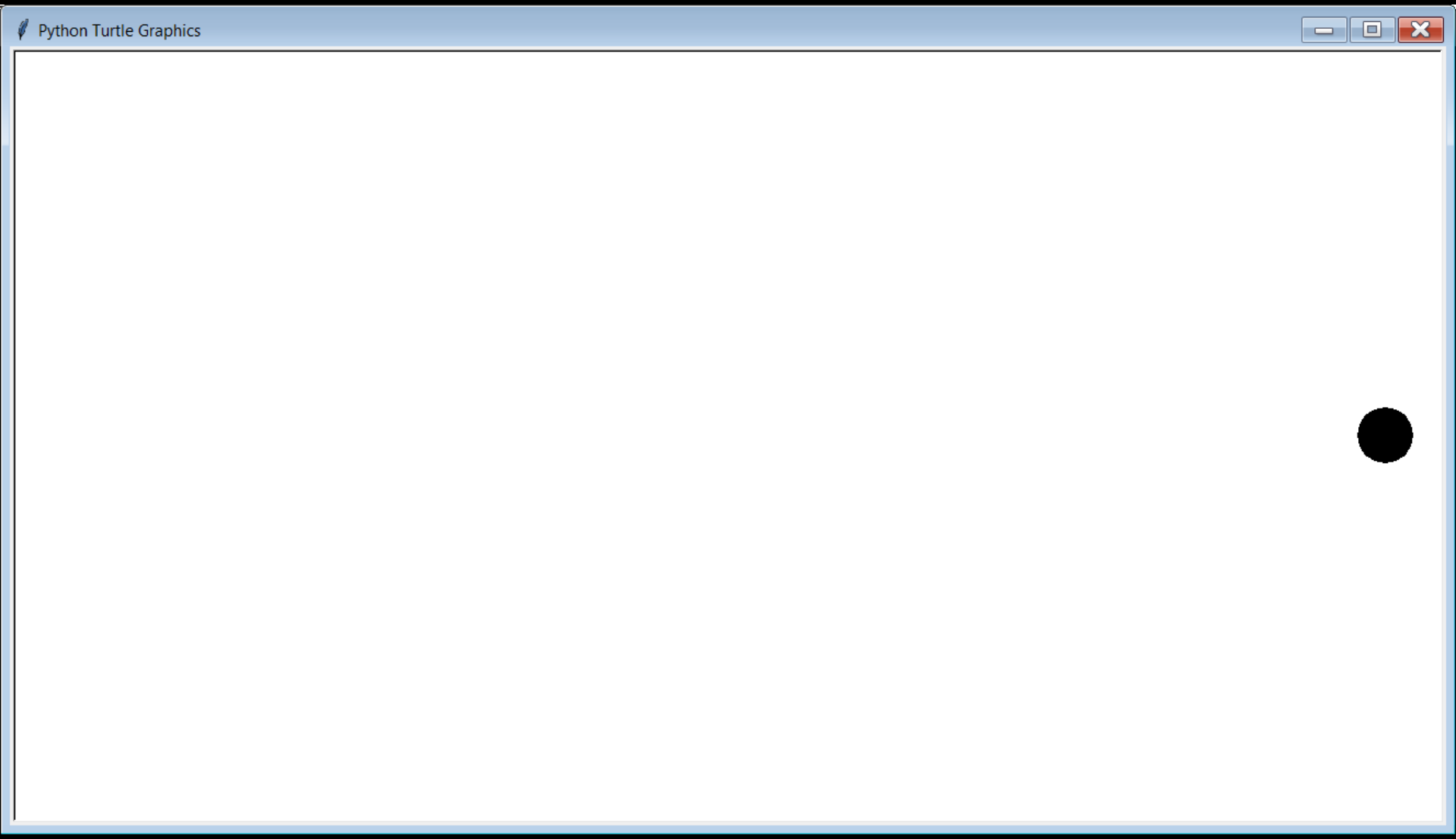

Python Turtle Graphics


Python Turtle Graphics


Python Turtle Graphics

Python Turtle Graphics


Python Turtle Graphics


Python Turtle Graphics

Python Turtle Graphics

Program Note

The output of program **Animation04.py** was simulated with PowerPoint.

It will not be possible to simulate the output of the remaining programs.

You need to execute them on your own computer to truly see what they are doing.

Do you see a pattern here?

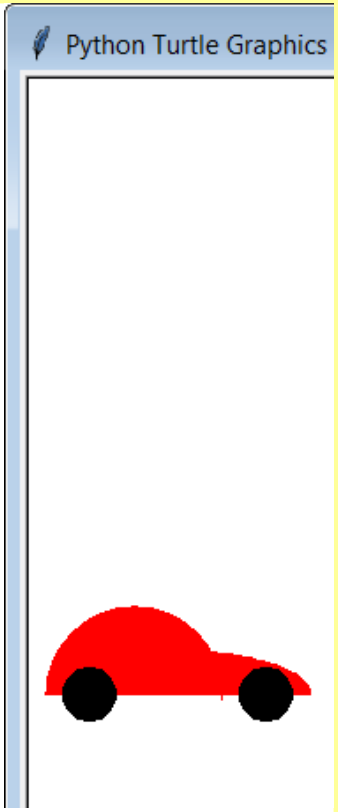
```python
# Animation04.py
#
#
#
#
#
#
#


from Graphics import *


beginGrfx(1300,700)

setColor("black")
fillCircle(50,350,25)
delay(1000)
setColor("white")
fillCircle(50,350,27)

setColor("black")
fillCircle(150,350,25)
delay(1000)
setColor("white")
fillCircle(150,350,27)

setColor("black")
fillCircle(250,350,25)
delay(1000)
setColor("white")
fillCircle(250,350,27)

setColor("black")
fillCircle(350,350,25)
delay(1000)
setColor("white")
fillCircle(350,350,27)

setColor("black")
fillCircle(450,350,25)
delay(1000)
setColor("white")
fillCircle(450,350,27)

setColor("black")
fillCircle(550,350,25)
delay(1000)
setColor("white")
fillCircle(550,350,27)

setColor("black")
fillCircle(650,350,25)
delay(1000)
setColor("white")
fillCircle(650,350,27)

setColor("black")
fillCircle(750,350,25)
delay(1000)
setColor("white")
fillCircle(750,350,27)

setColor("black")
fillCircle(850,350,25)
delay(1000)
setColor("white")
fillCircle(850,350,27)

setColor("black")
fillCircle(950,350,25)
delay(1000)
setColor("white")
fillCircle(950,350,27)

setColor("black")
fillCircle(1050,350,25)
delay(1000)
setColor("white")
fillCircle(1050,350,27)

setColor("black")
fillCircle(1150,350,25)
delay(1000)
setColor("white")
fillCircle(1150,350,27)

setColor("black")
fillCircle(1250,350,25)
delay(1000)
setColor("white")
fillCircle(1250,350,27)

endGrfx()
```

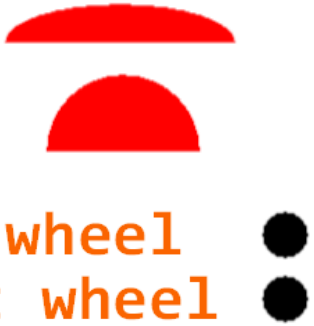
```
1  # Animation05.py
2  # This program has the same output as the
3  # previous program.  The program is now much
4  # shorter because it uses a <for> loop.
5
6
7  from Graphics import *
8
9
10 beginGrfx(1300,700)
11
12 for x in range(50,1251,100):
13     setColor("black")
14     fillCircle(x,350,25)
15     delay(1000)
16     setColor("white")
17     fillCircle(x,350,27)
18
19 endGrfx()
```

```
1  # Animation06.py
2  # This program makes the animation smoother
3  # by using a smaller delay and a smaller
4  # increment in the <for> loop.
5
6
7  from Graphics import *
8
9
10 beginGrfx(1300,700)
11
12 for x in range(50,1251,10):
13     setColor("black")
14     fillCircle(x,350,25)
15     delay(100)
16     setColor("white")
17     fillCircle(x,350,27)
18
19 endGrfx()
```

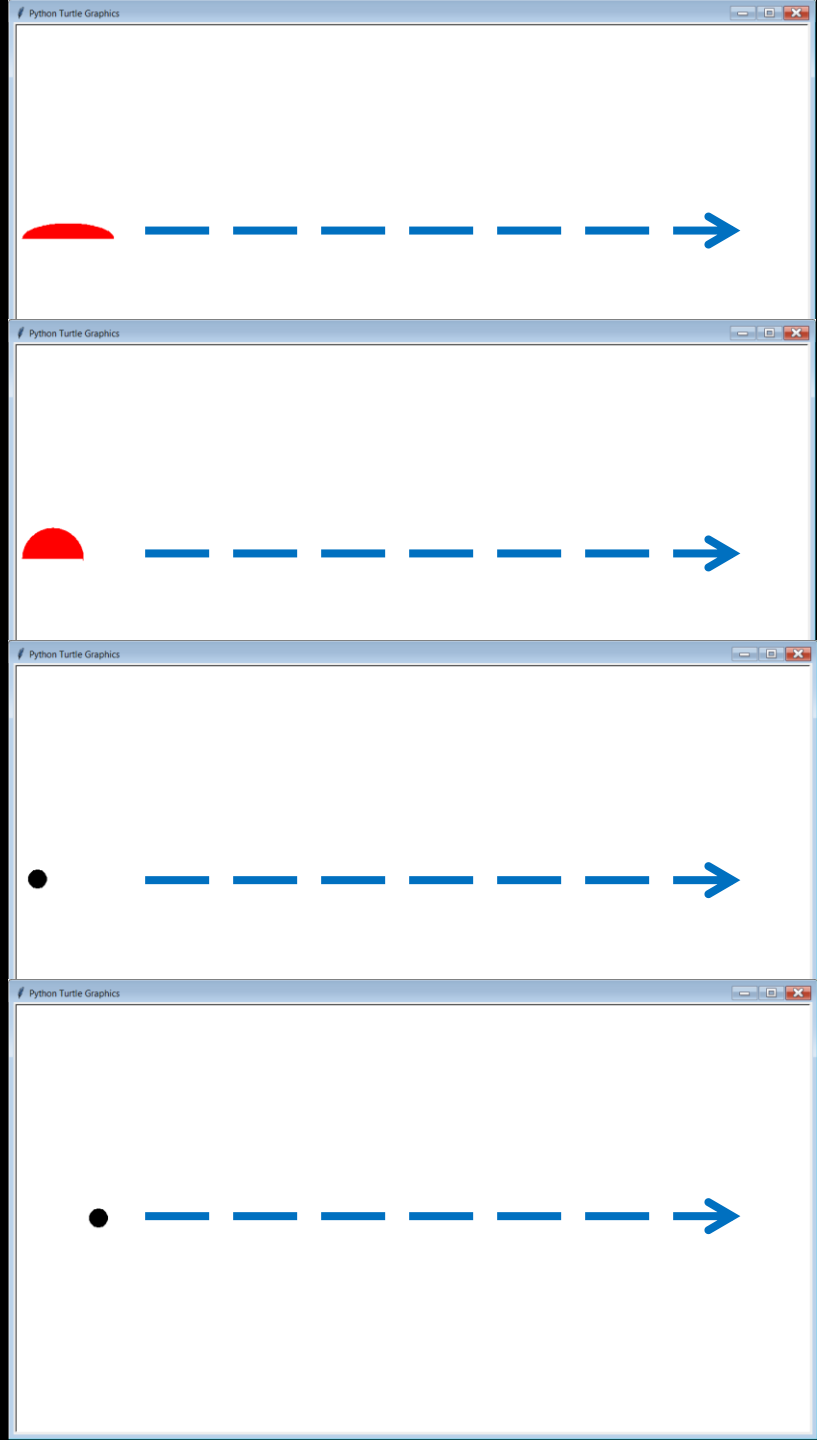
```
# Animation07.py
# This program makes the animation as smooth
# as possible by having an increment of just
# 1 pixel in the <for> loop.
# The delay is also made smaller.


from Graphics import *


beginGrfx(1300,700)

for x in range(50,1251,1):
    setColor("black")
    fillCircle(x,350,25)
    delay(10)
    setColor("white")
    fillCircle(x,350,27)

endGrfx()
```

```python
# Animation08.py
# This program draws the car that will
# be animated in the next few programs.
# The car is an overlapping composition
# of 4 filled shapes, specifically the
# 2 arcs and 2 circles.


from Graphics import *


beginGrfx(1300,700)

setColor("red")
fillArc(85,350,75,25,270,90)  # body
fillArc(60,350,50,50,270,90)  # hood
setColor("black")
fillCircle(35,350,15)         # rear wheel
fillCircle(135,350,15)        # front wheel

endGrfx()
```



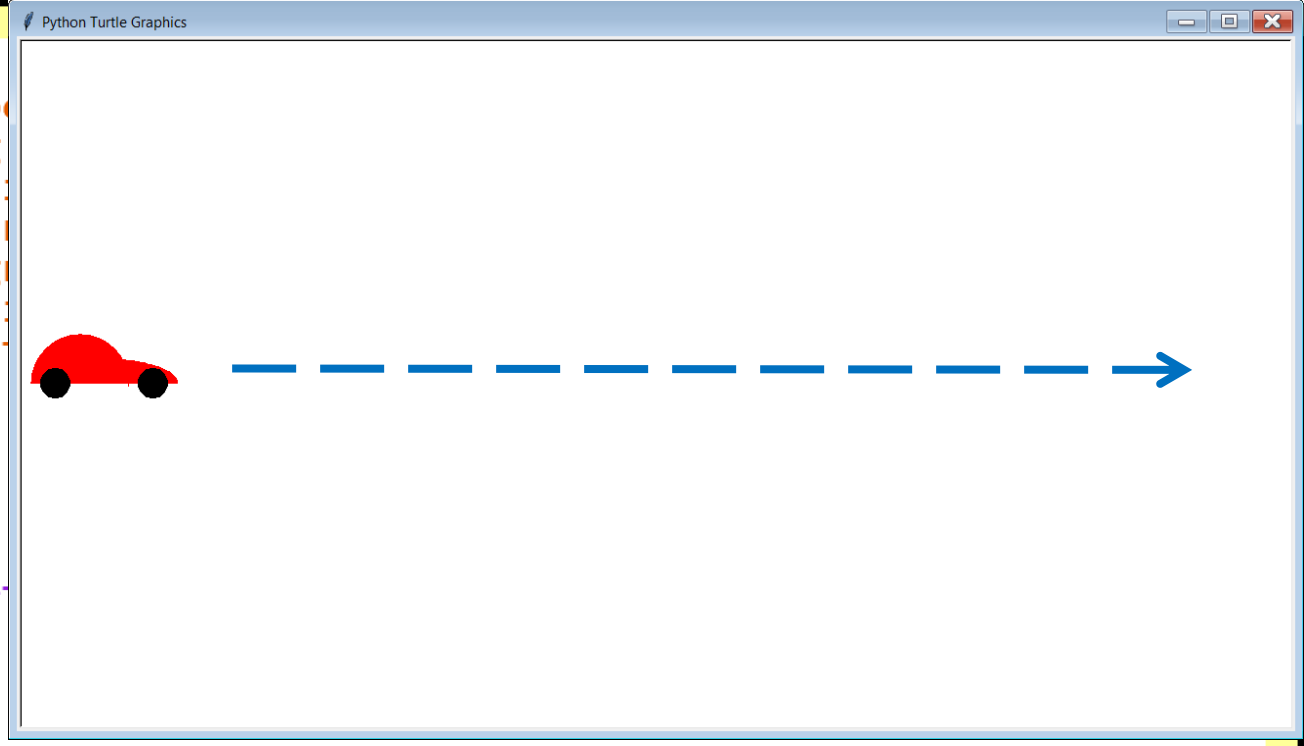


```python
# Animation09.py
# This program demonstrates a common logic error
# when students try to move complicated images
# across the screen. The problem is each part of
# the car is in its own separate loop.  This means
# each part of the car moves across the screen one
# at a time instead of all together.  On top of this,
# on a Mac each piece may leave behind a trail.


from Graphics import *


beginGrfx(1300,700)

for x in range(85,1226,10):        # body
   setColor("red")
   fillArc(x,350,75,25,270,90)
   delay(50)
   setColor("white")
   fillArc(x,350,75,25,270,90)

for x in range(60,1201,10):        # hood
   setColor("red")
   fillArc(x,350,50,50,270,90)
   delay(50)
   setColor("white")
   fillArc(x,350,50,50,270,90)

for x in range(35,1176,10):        # rear wheel
   setColor("black")
   fillCircle(x,350,15)
   delay(50)
   setColor("white")
   fillCircle(x,350,15)

for x in range(135,1276,10):       # front wheel
   setColor("black")
   fillCircle(x,350,15)
   delay(50)
   setColor("white")
   fillCircle(x,350,15)

endGrfx()
```



```
# Animation10.py
# This program properly animates the car across the
# screen by placing all 4 shapes in the same <for> loop.
# A little algebra is required since the location of 3
# of the shapes is relative to the location of the rear
# wheel.  This program also introduces another issue.
# As more and more images are drawn/erased on/from the
# screen, the output gets slower and slower and eventually
# drags to a crawl.  And on top of all of this, on a Mac
# the car may leave behind a trail.



from Graphics import *


beginGrfx(1300,700)

for x in range(35,1176,5):
   setColor("red")
   fillArc(x+50,350,75,25,270,90) # draw body
   fillArc(x+25,350,50,50,270,90) # draw hood
   setColor("black")
   fillCircle(x,350,15)           # draw rear wheel
   fillCircle(x+100,350,15)       # draw front wheel
   delay(10)
   setColor("white")
   fillArc(x+50,350,75,25,270,90) # erase body
   fillArc(x+25,350,50,50,270,90) # erase hood
   fillCircle(x,350,15)           # erase rear wheel
   fillCircle(x+100,350,15)       # erase front wheel

endGrfx()
```

```python
# Animation10.py
# This program prop
# screen by placing
# A little algebra
# of the shapes is
# wheel.  This prog
# As more and more
# screen, the outpu
# drags to a crawl.
# the car may leave



from Graphics impor


beginGrfx(1300,700)

for x in range(35,1176,5):
    setColor("red")
    fillArc(x+50,350,75,25,270,90) # draw body
    fillArc(x+25,350,50,50,270,90) # draw hood
    setColor("black")
    fillCircle(x,350,15)           # draw rear wheel
    fillCircle(x+100,350,15)       # draw front wheel
    delay(10)
    setColor("white")
    fillArc(x+50,350,75,25,270,90) # erase body
    fillArc(x+25,350,50,50,270,90) # erase hood
    fillCircle(x,350,15)           # erase rear wheel
    fillCircle(x+100,350,15)       # erase front wheel

endGrfx()
```

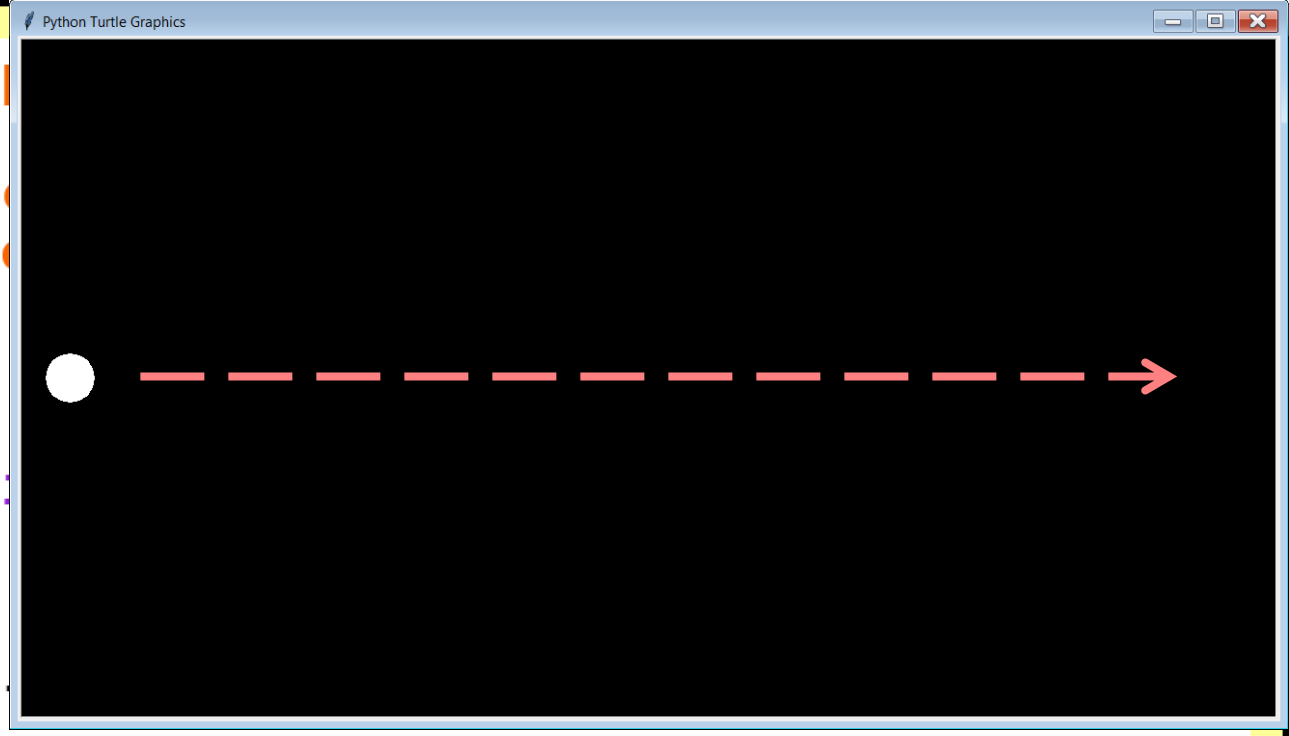


```
1  # Animation11.py
2  # This program fixes all of the issues of the
3  # previous program by using the <clear> command
4  # to completely erase everything on the screen.
5
6
7  from Graphics import *
8
9
10 beginGrfx(1300,700)
11
12 for x in range(35,1176,5):
13     setColor("red")
14     fillArc(x+50,350,75,25,270,90) # draw body
15     fillArc(x+25,350,50,50,270,90) # draw hood
16     setColor("black")
17     fillCircle(x,350,15)           # draw rear wheel
18     fillCircle(x+100,350,15)       # draw front wheel
19     delay(10)
20     clear()                        # erase everything
21
22 endGrfx()
```

```python
# Animation12.py
# This program shows an alternate way to
# animate the car across the screen.
# While this way does not require Algebra
# and is less complicated, the code is much
# longer, and the output is not as smooth.


from Graphics import *


beginGrfx(1300,700)

setColor("red")
fillArc(85,350,75,25,270,90)
fillArc(60,350,50,50,270,90)
setColor("black")
fillCircle(35,350,15)
fillCircle(135,350,15)
delay(1000)
clear()

setColor("red")
fillArc(185,350,75,25,270,90)
fillArc(160,350,50,50,270,90)
setColor("black")
fillCircle(135,350,15)
fillCircle(235,350,15)
delay(1000)
clear()

setColor("red")
fillArc(285,350,75,25,270,90)
fillArc(260,350,50,50,270,90)
setColor("black")
fillCircle(235,350,15)
fillCircle(335,350,15)
delay(1000)
clear()

setColor("red")
fillArc(385,350,75,25,270,90)
fillArc(360,350,50,50,270,90)
setColor("black")
fillCircle(335,350,15)
fillCircle(435,350,15)
delay(1000)
clear()

setColor("red")
fillArc(485,350,75,25,270,90)
fillArc(460,350,50,50,270,90)
setColor("black")
fillCircle(435,350,15)
fillCircle(535,350,15)
delay(1000)
clear()

setColor("red")
fillArc(585,350,75,25,270,90)
fillArc(560,350,50,50,270,90)
setColor("black")
fillCircle(535,350,15)
fillCircle(635,350,15)
delay(1000)
clear()

setColor("red")
fillArc(685,350,75,25,270,90)
fillArc(660,350,50,50,270,90)
setColor("black")
fillCircle(635,350,15)
fillCircle(735,350,15)
delay(1000)
clear()

setColor("red")
fillArc(785,350,75,25,270,90)
fillArc(760,350,50,50,270,90)
setColor("black")
fillCircle(735,350,15)
fillCircle(835,350,15)
delay(1000)
clear()

setColor("red")
fillArc(885,350,75,25,270,90)
fillArc(860,350,50,50,270,90)
setColor("black")
fillCircle(835,350,15)
fillCircle(935,350,15)
delay(1000)
clear()

setColor("red")
fillArc(985,350,75,25,270,90)
fillArc(960,350,50,50,270,90)
setColor("black")
fillCircle(935,350,15)
fillCircle(1035,350,15)
delay(1000)
clear()

setColor("red")
fillArc(1085,350,75,25,270,90)
fillArc(1060,350,50,50,270,90)
setColor("black")
fillCircle(1035,350,15)
fillCircle(1135,350,15)
delay(1000)
clear()

setColor("red")
fillArc(1185,350,75,25,270,90)
fillArc(1160,350,50,50,270,90)
setColor("black")
fillCircle(1135,350,15)
fillCircle(1235,350,15)
delay(1000)
clear()

endGrfx()
```

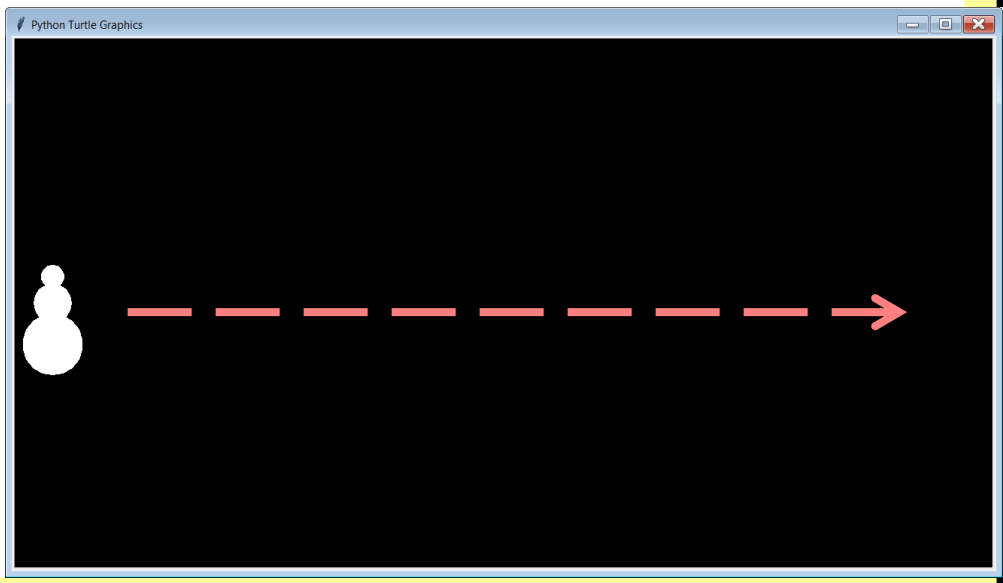
```
1  # Animation13.py
2  # This program is similar to Animation07.py,
3  # but now the circle is white and the
4  # background color is black.  This means
5  # <"black"> is now the "erasing color".
6
7
8  from Graphics import *
9
10
11 beginGrfx(1300,700)
12
13 setBackground("black")
14
15 for x in range(50,1251,1):
16     setColor("white")
17     fillCircle(x,350,25)
18     delay(10)
19     setColor("black")
20     fillCircle(x,350,27)
21
22 endGrfx()
```

```
1  # Animation13.p
2  # This program
3  # but now the
4  # background co
5  # <"black"> is
6
7
8  from Graphics
9
10
11 beginGrfx(1300,
12
13 setBackground("black")
14
15 for x in range(50,1251,1):
16     setColor("white")
17     fillCircle(x,350,25)
18     delay(10)
19     setColor("black")
20     fillCircle(x,350,27)
21
22 endGrfx()
```



```python
# Animation14.py
# This program animates a snowman across the screen
# by placing 3 circles in the same <for> loop.
# Note how the "slowing down" problem returns.


from Graphics import *


beginGrfx(1300,700)

setBackground("black")

for x in range(50,1251,1):
    setColor("white")
    fillCircle(x,315,15)
    fillCircle(x,350,25)
    fillCircle(x,405,40)
    delay(10)
    setColor("black")
    fillCircle(x,315,17)
    fillCircle(x,350,27)
    fillCircle(x,405,42)

endGrfx()
```

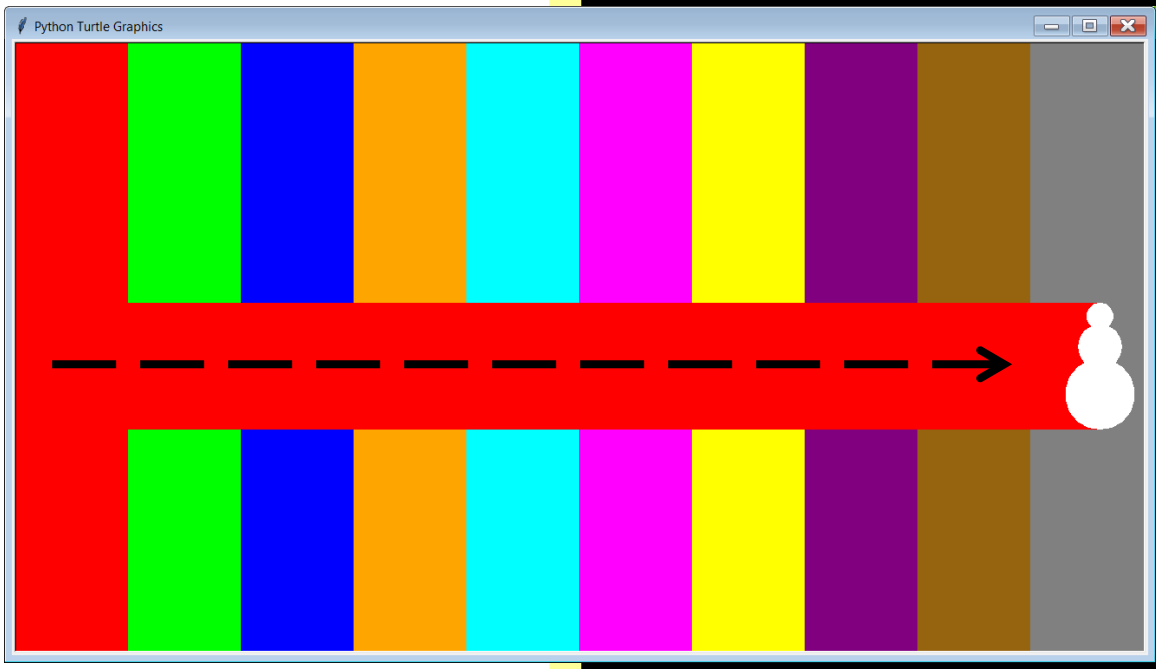
```python
# Animation14.py
# This program animates a snowman across the screen
# by placing 3 circles in the same <for> loop.
# Note how the "slowing down" problem returns.


from Graphics import *


beginGrfx(1300,700)

setBackground("black")

for x in range(50,1251,1):
    setColor("white")
    fillCircle(x,315,15)
    fillCircle(x,350,25)
    fillCircle(x,405,40)
    delay(10)
    setColor("black")
    fillCircle(x,315,17)
    fillCircle(x,350,27)
    fillCircle(x,405,42)

endGrfx()
```



```python
# Animation15.py
# This program fixes the "slowing down" problem
# of the previous program by using <clear>.
# This does require that the background color
# is set immediately after the <clear> command.
# NOTE: This is called "Draw-And-Redraw Animation".


from Graphics import *


beginGrfx(1300,700)

for x in range(50,51,3):
    clear()
    setBackground("black")
    setColor("white")
    fillCircle(x,315,15)
    fillCircle(x,350,25)
    fillCircle(x,405,40)
    delay(10)

endGrfx()
```

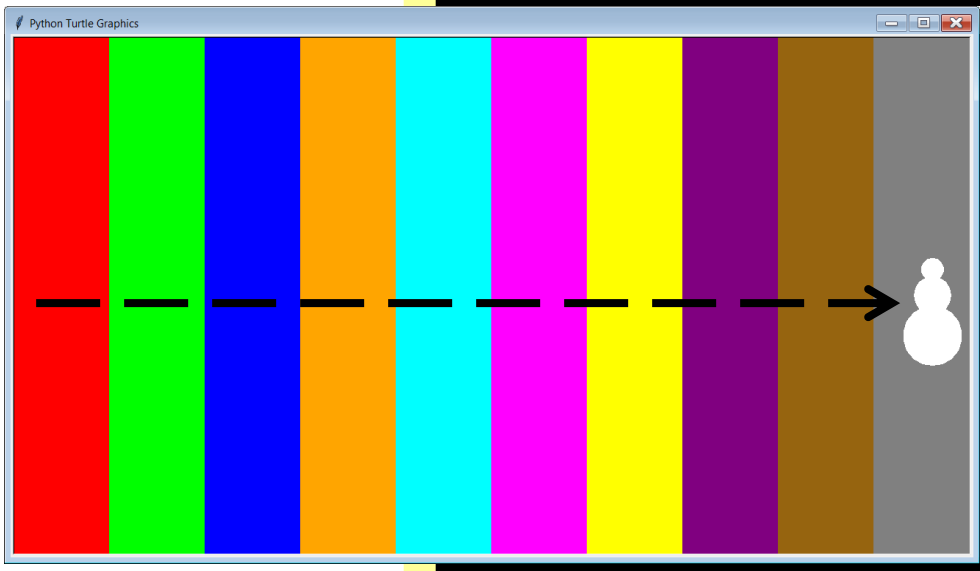
```python
# Animation16.py
# This program shows another major problem with
# "Draw-And-Erase Animation."  It falls flat on
# its face when you have a multi-colored background.


from Graphics import *


def drawColorfulBackground():
    for c in range(10):
        setColor(c+1)
        fillRectangle(c*130, 0, (c+1)*130, 700)



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

beginGrfx(1300,700)

drawColorfulBackground()

for x in range(50,1251,2):
    setColor("white")
    fillCircle(x,315,15)
    fillCircle(x,350,25)
    fillCircle(x,405,40)
    delay(10)
    setColor("red")
    fillCircle(x,315,17)
    fillCircle(x,350,27)
    fillCircle(x,405,42)

endGrfx()
```

```python
# Animation16.py
# This program shows another major problem with
# "Draw-And-Erase Animation."  It falls flat on
# its face when you have a multi-colored background.


from Graphics import *


def drawColorfulBackground():
    for c in range(10):
        setColor(c+1)
        fillRectangle(c*130, 0, (c+1)*130, 700)



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

beginGrfx(1300,700)

drawColorfulBackground()

for x in range(50,1251,2):
    setColor("white")
    fillCircle(x,315,15)
    fillCircle(x,350,25)
    fillCircle(x,405,40)
    delay(10)
    setColor("red")
    fillCircle(x,315,17)
    fillCircle(x,350,27)
    fillCircle(x,405,42)

endGrfx()
```



```python
# Animation17.py
# This program shows how to handle a multi-colored
# background. By using "Draw-And-Redraw Animation"
# instead of "Draw-And-Erase" the entire background
# is redrawn every time the object moves.


from Graphics import *


def drawColorfulBackground():
   for c in range(10):
      setColor(c+1)
      fillRectangle(c*130, 0, (c+1)*130, 700)



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

beginGrfx(1300,700)

for x in range(50,1251,2):
   clear()
   drawColorfulBackground()
   setColor("white")
   fillCircle(x,315,15)
   fillCircle(x,350,25)
   fillCircle(x,405,40)
   delay(10)

endGrfx()
```

```python
# Animation17.py
# This program shows how to handle a multi-colored
# background. By using "Draw-And-Redraw Animation"
# instead of "Draw-And-Erase" the entire background
# is redrawn every time the object moves.


from Graphics import *


def drawColorfulBackground():
    for c in range(10):
        setColor(c+1)
        fillRectangle(c*130, 0, (c+1)*130, 700)



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

beginGrfx(1300,700)

for x in range(50,1251,2):
    clear()
    drawColorfulBackground()
    setColor("white")
    fillCircle(x,315,15)
    fillCircle(x,350,25)
    fillCircle(x,405,40)
    delay(10)

endGrfx()
```



Draw-And-Redraw Animation Step-By-Step Review

1. Clear the screen.
2. Draw the entire background (preferably by using a procedure).
3. Draw the foreground image.
4. Delay for a fraction of a second.
5. Repeat this process over and over each time drawing the foreground image in a slightly different location.

This is the recommended approach for Computer Animation in our class because it solves BOTH of the problems of *Draw and Erase Animation*:

1. It works even if your background is not one solid color.
2. The animation will not slow down, even for complicated images.

Section 18.7

The Last Word

Programming Knowledge <u>Never</u> Becomes Obsolete

Some people may be concerned that with technology changing so rapidly, the programming knowledge that they have acquired in this course will soon be out of date. While new programming languages pop up from time to time, the logic of programming has never changed.

The next 6 programs all do the exact same thing:
The 1st is written in Python. The 2nd is in Java. The 3rd is in C++. The 4th is in Pascal; the 5th is in BASIC; and the 6th is written in VEXcode VR Blocks. Even though you may have never seen some of these languages before, you may be surprised how familiar these programs will look to you now that you know Python.

```python
# LastWord01.py
# This program will count backwards from 20 to 0.
# For each number, it will display whether it has
# 1 or 2 digits. Compare this Python code to the
# code from other languages.


print()
for k in range (20, -1, -1):
    if k >= 10:
        print (k,"is a 2-digit number.")
    else:
        print (k,"is a 1-digit number.")

```

```python
# LastWord01.py
# This program will count backw
# For each number, it will disp
# 1 or 2 digits. Compare this P
# code from other languages.


print()
for k in range (20, -1, -1):
    if k >= 10:
        print (k,"is a 2-digit
    else:
        print (k,"is a 1-digit

```

```
----jGRASP exec: python

20 is a 2-digit number.
19 is a 2-digit number.
18 is a 2-digit number.
17 is a 2-digit number.
16 is a 2-digit number.
15 is a 2-digit number.
14 is a 2-digit number.
13 is a 2-digit number.
12 is a 2-digit number.
11 is a 2-digit number.
10 is a 2-digit number.
9 is a 1-digit number.
8 is a 1-digit number.
7 is a 1-digit number.
6 is a 1-digit number.
5 is a 1-digit number.
4 is a 1-digit number.
3 is a 1-digit number.
2 is a 1-digit number.
1 is a 1-digit number.
0 is a 1-digit number.

----jGRASP: operation c
```

```
1 // LastWord02.java
2 // This program will count backwards from 20 to 0.
3 // For each number, it will display whether it has
4 // 1 or 2 digits. Compare this Java code to the
5 // code from other languages.
6
7 public class LastWord02
8 {
9    public static void main (String args[])
10   {
11      System.out.println();
12      for (int k = 20; k >= 0; k--)
13      {
14         if (k >= 10)
15         {
16            System.out.println(k + " is a 2-digit number.");
17         }
18         else
19         {
20            System.out.println(k + " is a 1-digit number.");
21         }
22      }
23   }
24 }
```

```java
// LastWord02.java
// This program will count backwards from 2
// For each number, it will display whether
// 1 or 2 digits. Compare this Java code to
// code from other languages.

public class LastWord02
{
   public static void main (String args[])
   {
      System.out.println();
      for (int k = 20; k >= 0; k--)
      {
         if (k >= 10)
         {
            System.out.println(k + " is a 2
         }
         else
         {
            System.out.println(k + " is a 1
         }
      }
   }
}
```

```
----jGRASP exec: java

20 is a 2-digit number.
19 is a 2-digit number.
18 is a 2-digit number.
17 is a 2-digit number.
16 is a 2-digit number.
15 is a 2-digit number.
14 is a 2-digit number.
13 is a 2-digit number.
12 is a 2-digit number.
11 is a 2-digit number.
10 is a 2-digit number.
9 is a 1-digit number.
8 is a 1-digit number.
7 is a 1-digit number.
6 is a 1-digit number.
5 is a 1-digit number.
4 is a 1-digit number.
3 is a 1-digit number.
2 is a 1-digit number.
1 is a 1-digit number.
0 is a 1-digit number.

----jGRASP: operation
```

Turbo C++ IDE

File Edit Search Run Compile Debug Project Options Help

LASTWRD3.CPP 2

```
// LASTWRD3.CPP
// This program will count backwards from 20 to 0.
// For each number, it will display whether it has 1 or 2 digits.
// Compare this C++ code to the code from other languages.

#include <iostream.h>
#include <conio.h>

void main()
{
   for (int k = 20; k >= 0; k--) {
      if (k >= 10) {
         cout << k << " is a 2-digit number." << endl;
      }
      else {
         cout << k << " is a 1-digit number." << endl;
      }
   }
   cout << endl;
   getch();  // makes the computer wait until a character is pressed
}
```

16:19

F1 Help F2 Save F3 Open Alt-F9 Compile F9 Make F10 Menu

Turbo C++ IDE

File Edit Search Run Compile

```
// LASTWRD3.CPP
// This program will count backwards
// For each number, it will display
// Compare this C++ code to the code

#include <iostream.h>
#include <conio.h>

void main()
{
   for (int k = 20; k >= 0; k--) {
      if (k >= 10) {
         cout << k << " is a 2-digit
      }
      else {
         cout << k << " is a 1-digit
      }
   }
   cout << endl;
   getch();  // makes the computer w
}
```

16:19

F1 Help F2 Save F3 Open Alt-F9 Co

Turbo C++ IDE

```
20 is a 2-digit number.
19 is a 2-digit number.
18 is a 2-digit number.
17 is a 2-digit number.
16 is a 2-digit number.
15 is a 2-digit number.
14 is a 2-digit number.
13 is a 2-digit number.
12 is a 2-digit number.
11 is a 2-digit number.
10 is a 2-digit number.
9 is a 1-digit number.
8 is a 1-digit number.
7 is a 1-digit number.
6 is a 1-digit number.
5 is a 1-digit number.
4 is a 1-digit number.
3 is a 1-digit number.
2 is a 1-digit number.
1 is a 1-digit number.
0 is a 1-digit number.
```

Turbo Pascal

File Edit Run Compile Options Debug Break/watch

Line 1 Col 7 Insert Indent Unindent C:LASTWRD4.PAS

```
{ LASTWRD4.PAS
  This program will count backwards from 20 to 0.
  For each number, it will display whether it has 1 or 2 digits.
  Compare this Pascal code to the code from other languages. }


PROGRAM My_Pascal_Program;

VAR
   K : INTEGER;  { Loop Counter }

BEGIN
   FOR K := 20 DOWNTO 0 DO BEGIN
      IF K >= 10 THEN BEGIN
         WRITELN(K,' is a 2-digit number.')
      END
      ELSE BEGIN
         WRITELN(K,' is a 1-digit number.')
      END
   END
END.
```

Alt: F1-Last help F3-Pick F6-Swap F9-Compile

Turbo Pascal

```
20 is a 2-digit number.
19 is a 2-digit number.
18 is a 2-digit number.
17 is a 2-digit number.
16 is a 2-digit number.
15 is a 2-digit number.
14 is a 2-digit number.
13 is a 2-digit number.
12 is a 2-digit number.
11 is a 2-digit number.
10 is a 2-digit number.
9 is a 1-digit number.
8 is a 1-digit number.
7 is a 1-digit number.
6 is a 1-digit number.
5 is a 1-digit number.
4 is a 1-digit number.
3 is a 1-digit number.
2 is a 1-digit number.
1 is a 1-digit number.
0 is a 1-digit number.
```

http://www.quitebasic.com/
Quite BASIC — fun, learnin...
File Edit View Favorites Tools Help
Google Apple BASIC Search Share More »
Sign In
Suggested Sites Web Slice Gallery
Page Safety Tools

BASIC Program

```
100 REM LASTWORD05.BAS
110 REM This program will count backwards from 20 to 0.
120 REM For each number, it will display whether it has
130 REM 1 or 2 digits. Compare this BASIC code to the
140 REM code from other languages.
150 REM
160 REM
170 FOR K = 20 TO 0 STEP -1
180 PRINT K;
190 IF K >= 10 THEN PRINT " is a 2-digit number. " ELSE
PRINT " is a 1-digit number."
200 NEXT K
```

Output

```
20 is a 2-digit number.
19 is a 2-digit number.
18 is a 2-digit number.
17 is a 2-digit number.
16 is a 2-digit number.
15 is a 2-digit number.
14 is a 2-digit number.
13 is a 2-digit number.
12 is a 2-digit number.
11 is a 2-digit number.
10 is a 2-digit number.
9 is a 1-digit number.
8 is a 1-digit number.
7 is a 1-digit number.
6 is a 1-digit number.
5 is a 1-digit number.
4 is a 1-digit number.
3 is a 1-digit number.
2 is a 1-digit number.
1 is a 1-digit number.
0 is a 1-digit number.
```

VR File TUTORIALS ACTIVITIES LEARN LastWord06 PLAYGROUND START STEP STOP SHARE FEEDBACK

Code

Drivetrain
Magnet
Looks
Control
Sensing
Operators
Variables
My Blocks
Comments

```
when started
    LastWord06.vrblocks
    This program will count backwards from 20 to 0.
    For each number, it will display whether it has 1 or 2 digits.
    Compare this VEXcode VR Blocks program to the code from other languages.
clear all rows
set k to 20
repeat 21
    print k
    if k > 9 then
        print is a 2-digit number.
    else
        print is a 1-digit number.
    set cursor to next row
    change k by -1
```

Monitor

Sensors	

Variables	

```
20 is a 2-digit number.
19 is a 2-digit number.
18 is a 2-digit number.
17 is a 2-digit number.
16 is a 2-digit number.
15 is a 2-digit number.
14 is a 2-digit number.
13 is a 2-digit number.
12 is a 2-digit number.
11 is a 2-digit number.
10 is a 2-digit number.
9 is a 1-digit number.
8 is a 1-digit number.
7 is a 1-digit number.
6 is a 1-digit number.
5 is a 1-digit number.
4 is a 1-digit number.
3 is a 1-digit number.
2 is a 1-digit number.
1 is a 1-digit number.
0 is a 1-digit number.
```

CLEAR SAVE COPY TO CLIPBOARD