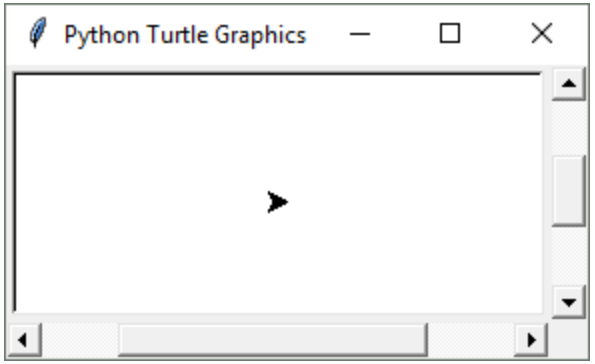
CN101

Turtle Graphics

Introduction to Turtle Graphics

- Python's turtle graphics system displays a small cursor known as a *turtle*.



- You can use Python statements to move the turtle around the screen, drawing lines and shapes.

Introduction to Turtle Graphics

- To use the turtle graphics system, you must import the turtle module with this statement:

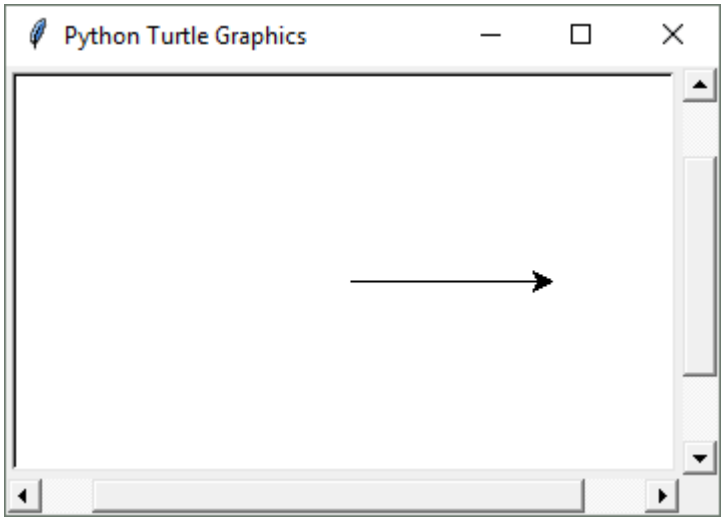
```
import turtle
```

This loads the turtle module into memory

Moving the Turtle Forward

- Use the turtle.forward(n) statement to move the turtle forward n pixels.

```
>>> import turtle
>>> turtle.forward(100)
>>>
```

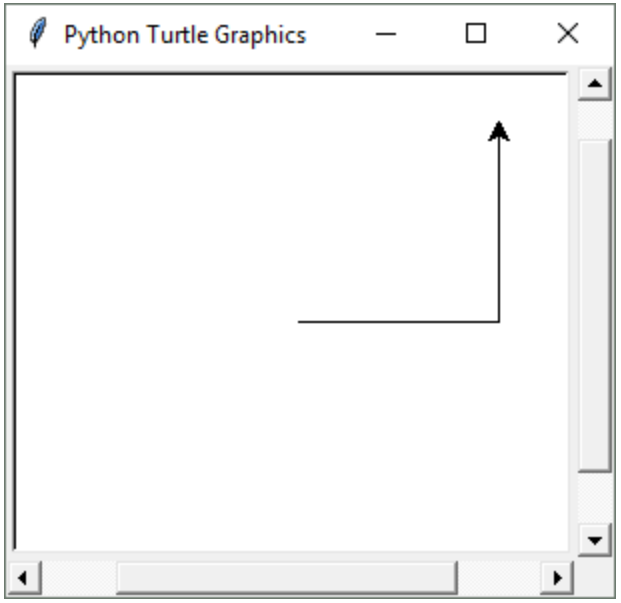


Turning the Turtle

- The turtle's initial heading is 0 degrees (east)
- Use the `turtle.right(`*`angle`*`)` statement to turn the turtle right by *`angle`* degrees.
- Use the `turtle.left(`*`angle`*`)` statement to turn the turtle left by *`angle`* degrees.

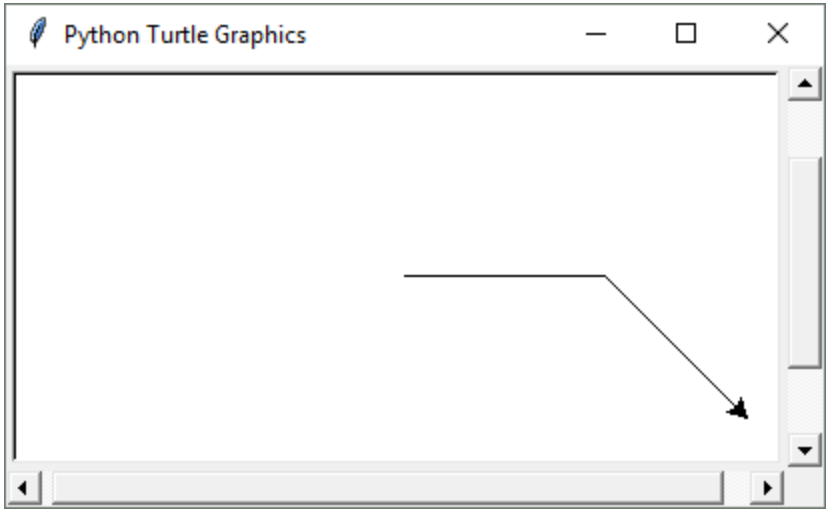
Turning the Turtle

```
>>> import turtle
>>> turtle.forward(100)
>>> turtle.left(90)
>>> turtle.forward(100)
>>>
```



Turning the Turtle

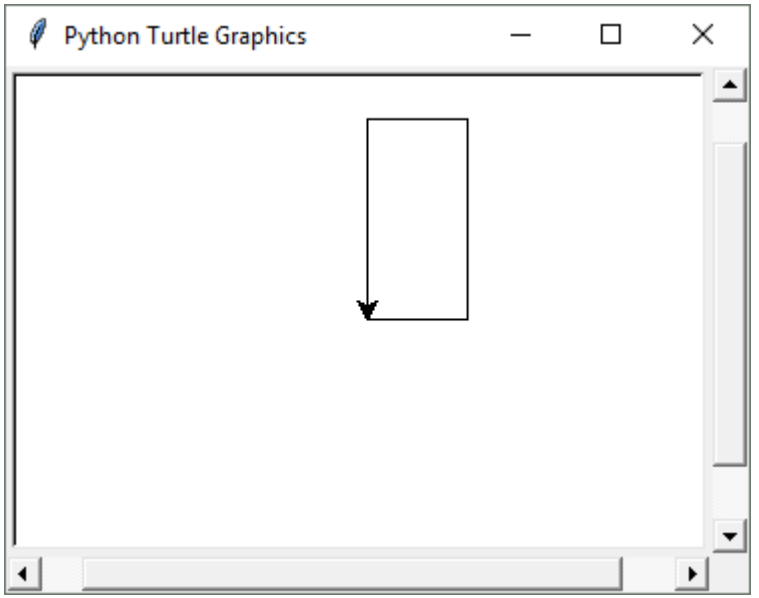
```
>>> import turtle
>>> turtle.forward(100)
>>> turtle.right(45)
>>> turtle.forward(100)
>>>
```



Setting the Turtle's Heading

- Use the `turtle.setheading(`*`angle`*`)` statement to set the turtle's heading to a specific angle.

```
>>> import turtle
>>> turtle.forward(50)
>>> turtle.setheading(90)
>>> turtle.forward(100)
>>> turtle.setheading(180)
>>> turtle.forward(50)
>>> turtle.setheading(270)
>>> turtle.forward(100)
>>>
```

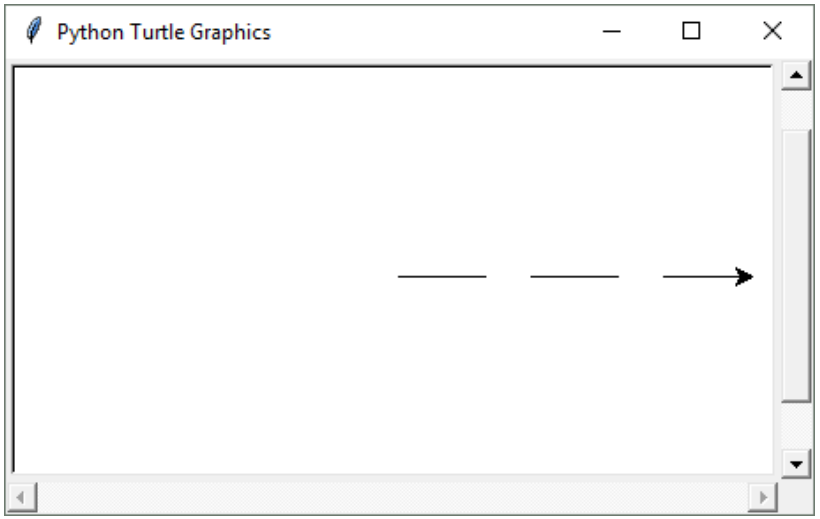


Setting the Pen Up or Down

- When the turtle's pen is down, the turtle draws a line as it moves. By default, the pen is down.
- When the turtle's pen is up, the turtle does not draw as it moves.
- Use the turtle.penup() statement to raise the pen.
- Use the turtle.pendown() statement to lower the pen.

Setting the Pen Up or Down

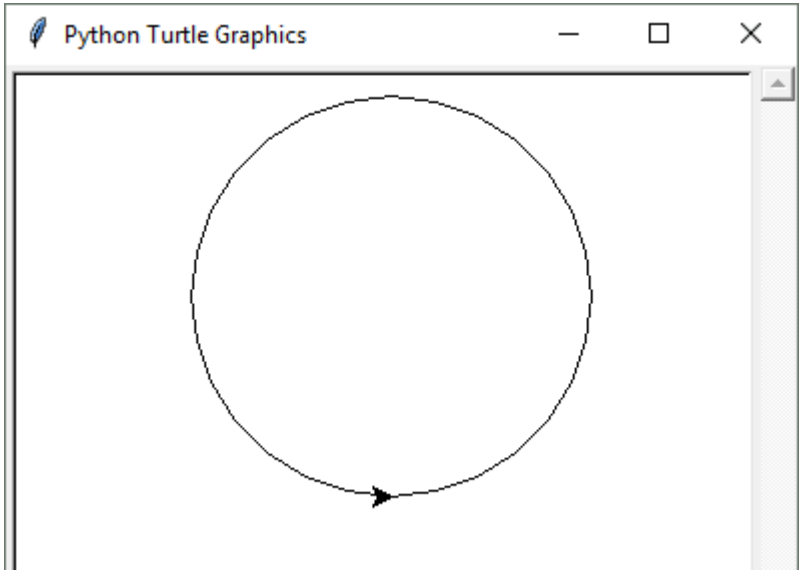
```
>>> import turtle
>>> turtle.forward(50)
>>> turtle.penup()
>>> turtle.forward(25)
>>> turtle.pendown()
>>> turtle.forward(50)
>>> turtle.penup()
>>> turtle.forward(25)
>>> turtle.pendown()
>>> turtle.forward(50)
>>>
```



Drawing Circles

- Use the `turtle.circle(`*`radius`*`)` statement to draw a circle with a specified radius.

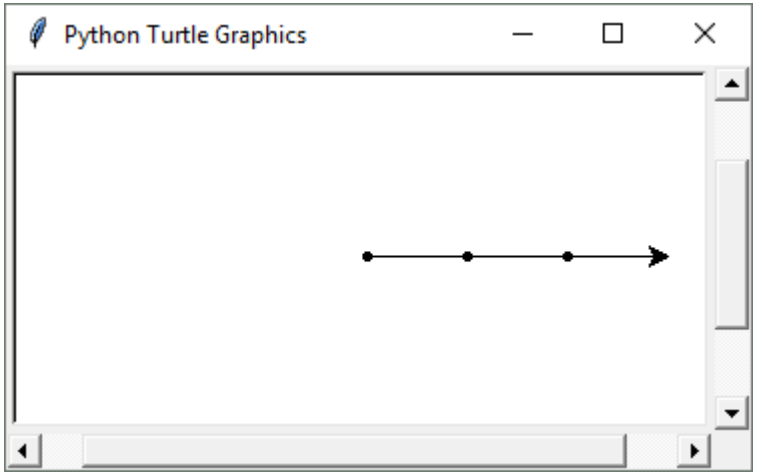
```
>>> import turtle
>>> turtle.circle(100)
>>>
```



Drawing Dots

- Use the `turtle.dot()` statement to draw a simple dot at the turtle's current location.

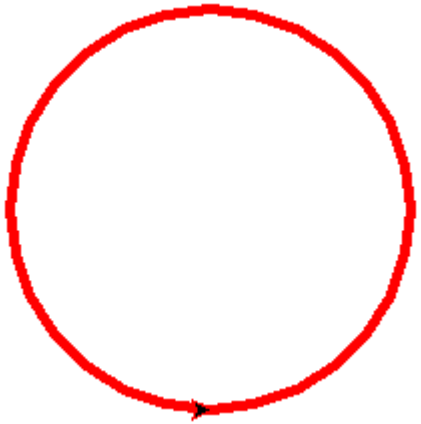
```
>>> import turtle
>>> turtle.dot()
>>> turtle.forward(50)
>>> turtle.dot()
>>> turtle.forward(50)
>>> turtle.dot()
>>> turtle.forward(50)
>>>
```



Changing the Pen Size and Drawing Color

- Use the turtle.pensize(width) statement to change the width of the turtle's pen, in pixels.
- Use the turtle.pencolor(color) statement to change the turtle's drawing color.
  - See Appendix D in your textbook for a complete list of colors.

```
>>> import turtle
>>> turtle.pensize(5)
>>> turtle.pencolor('red')
>>> turtle.circle(100)
>>>
```



Working with the Turtle's Window

- Use the `turtle.bgcolor(`*`color`*`)` statement to set the window's background color.
  - *See Appendix D in your textbook for a complete list of colors*.

- Use the `turtle.setup(`*`width`*`, `*`height`*`)` statement to set the size of the turtle's window, in pixels.
  - The *`width`* and *`height`* arguments are the width and height, in pixels.
  - For example, the following interactive session creates a graphics window that is 640 pixels wide and 480 pixels high:

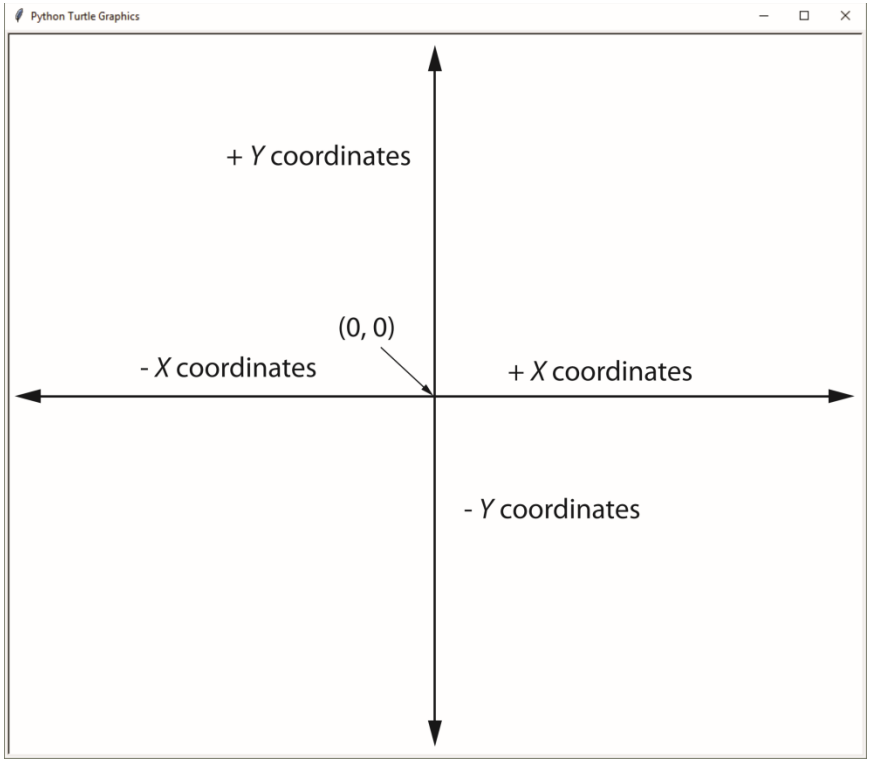
```
>>> import turtle
>>> turtle.setup(640, 480)
>>>
```

Resetting the Turtle's Window

- The `turtle.reset()` statement:
  - Erases all drawings that currently appear in the graphics window.
  - Resets the drawing color to black.
  - Resets the turtle to its original position in the center of the screen.
  - Does *not* reset the graphics window's background color.
- The `turtle.clear()` statement:
  - Erases all drawings that currently appear in the graphics window.
  - Does *not* change the turtle's position.
  - Does *not* change the drawing color.
  - Does *not* change the graphics window's background color.
- The `turtle.clearscreen()` statement:
  - Erases all drawings that currently appear in the graphics window.
  - Resets the drawing color to black.
  - Resets the turtle to its original position in the center of the screen.
  - Resets the graphics window's background color to white.

Working with Coordinates

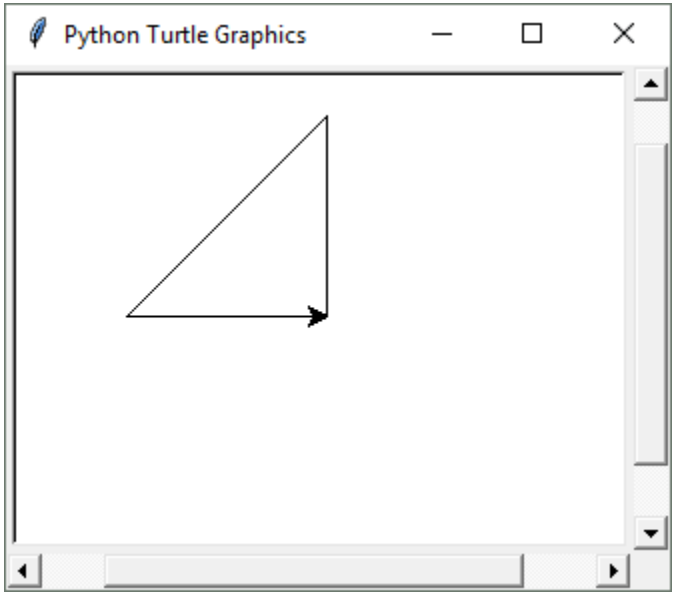
- The turtle uses Cartesian Coordinates



Moving the Turtle to a Specific Location

- Use the `turtle.goto(x, y)` statement to move the turtle to a specific location.

```
>>> import turtle
>>> turtle.goto(0, 100)
>>> turtle.goto(-100, 0)
>>> turtle.goto(0, 0)
>>>
```



- The `turtle.pos()` statement displays the turtle's current *X,Y* coordinates.
- The `turtle.xcor()` statement displays the turtle's current *X* coordinate and the `turtle.ycor()` statement displays the turtle's current *Y* coordinate.

Animation Speed

- Use the `turtle.speed(speed)` command to change the speed at which the turtle moves.
  - The *`speed`* argument is a number in the range of 0 through 10.
  - If you specify 0, then the turtle will make all of its moves instantly (animation is disabled).

Hiding and Displaying the Turtle

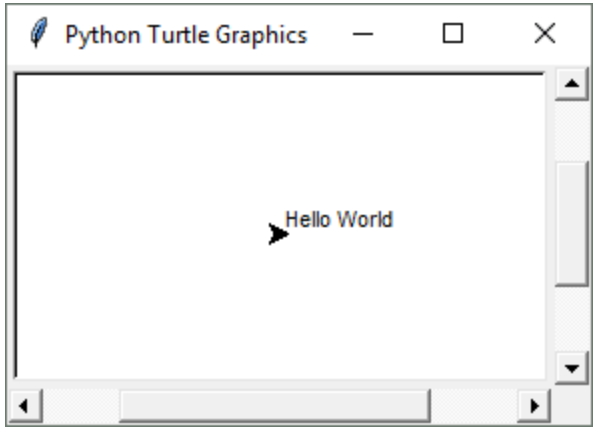
- Use the `turtle.hideturtle()` command to hide the turtle.
  - This command does not change the way graphics are drawn, it simply hides the turtle icon.
- Use the `turtle.showturtle()` command to display the turtle.

Displaying Text

- Use the `turtle.write(`*`text`*`)` statement to display text in the turtle's graphics window.
  - The *`text`* argument is a string that you want to display.
  - The lower-left corner of the first character will be positioned at the turtle's *X* and *Y* coordinates.

Displaying Text

```
>>> import turtle
>>> turtle.write('Hello World')
>>>
```

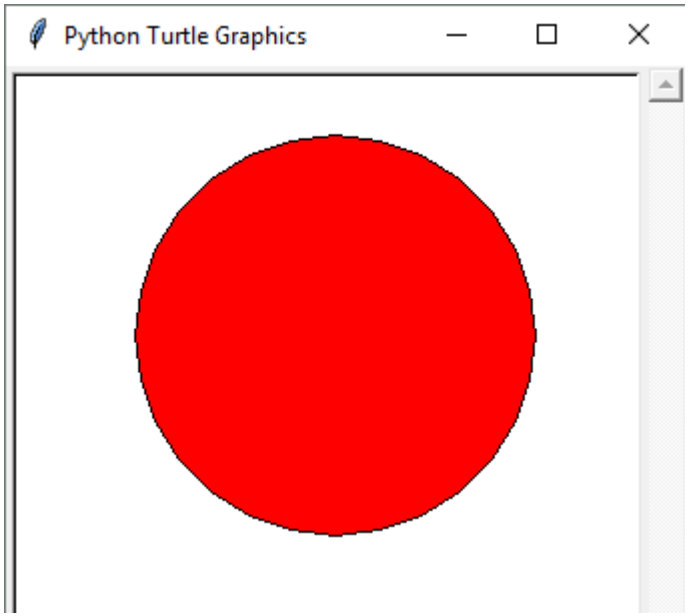


Filling Shapes

- To fill a shape with a color:
  - Use the `turtle.begin_fill()` command before drawing the shape
  - Then use the `turtle.end_fill()` command after the shape is drawn.
  - When the `turtle.end_fill()` command executes, the shape will be filled with the current fill color

Filling Shapes

```
>>> import turtle
>>> turtle.hideturtle()
>>> turtle.fillcolor('red')
>>> turtle.begin_fill()
>>> turtle.circle(100)
>>> turtle.end_fill()
>>>
```



Keeping the Graphics Window Open

- When running a turtle graphics program outside IDLE, the graphics window closes immediately when the program is done.

- To prevent this, add the `turtle.done()` statement to the very end of your turtle graphics programs.
  - This will cause the graphics window to remain open, so you can see its contents after the program finishes executing.

Turtle Graphics: Determining the State of the Turtle

- The `turtle.xcor()` and `turtle.ycor()` functions return the turtle's *X* and *Y* coordinates
- Examples of calling these functions in an `if` statement:

```
if turtle.ycor() < 0:
    turtle.goto(0, 0)
```

```
if turtle.xcor() > 100 and turtle.xcor() < 200:
    turtle.goto(0, 0)
```

Turtle Graphics: Determining the State of the Turtle

- The `turtle.heading()` function returns the turtle's heading. (By default, the heading is returned in degrees.)
- Example of calling the function in an `if` statement:

```
if turtle.heading() >= 90 and turtle.heading() <= 270:
    turtle.setheading(180)
```

Turtle Graphics: Determining the State of the Turtle

- The `turtle.isdown()` function returns `True` if the pen is down, or `False` otherwise.
- Example of calling the function in an `if` statement:

```
if turtle.isdown():
    turtle.penup()

if not(turtle.isdown()):
    turtle.pendown()
```

Turtle Graphics: Determining the State of the Turtle

- The `turtle.isvisible()` function returns `True` if the turtle is visible, or `False` otherwise.
- Example of calling the function in an `if` statement:

```
if turtle.isvisible():
    turtle.hideturtle()
```

Turtle Graphics: Determining the State of the Turtle

- When you call `turtle.pencolor()` without passing an argument, the function returns the pen's current color as a string. Example of calling the function in an `if` statement:

```
if turtle.pencolor() == 'red':
    turtle.pencolor('blue')
```

- When you call `turtle.fillcolor()` without passing an argument, the function returns the current fill color as a string. Example of calling the function in an `if` statement:

```
if turtle.fillcolor() == 'blue':
    turtle.fillcolor('white')
```

Turtle Graphics: Determining the State of the Turtle

- When you call `turtle.bgcolor()` without passing an argument, the function returns the current background color as a string. Example of calling the function in an `if` statement:

```
if turtle.bgcolor() == 'white':
    turtle.bgcolor('gray')
```

Turtle Graphics: Determining the State of the Turtle

- When you call `turtle.pensize()` without passing an argument, the function returns the pen's current size as a string. Example of calling the function in an `if` statement:

```
if turtle.pensize() < 3:
    turtle.pensize(3)
```

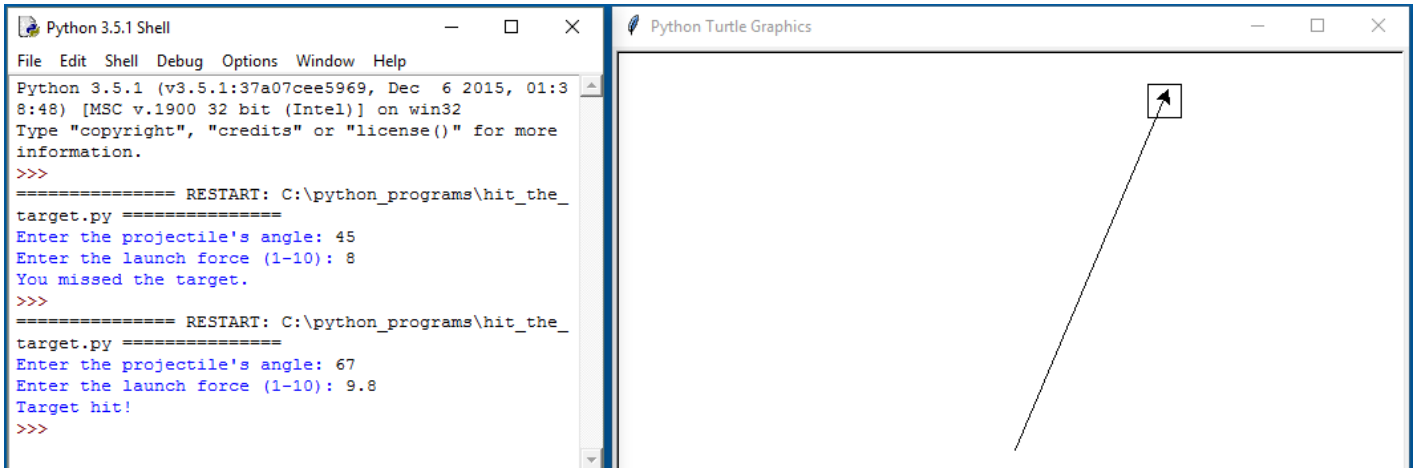
Turtle Graphics: Determining the State of the Turtle

- When you call `turtle.speed()` without passing an argument, the function returns the current animation speed. Example of calling the function in an `if` statement:

```
if turtle.speed() > 0:
    turtle.speed(0)
```

Turtle Graphics: Determining the State of the Turtle

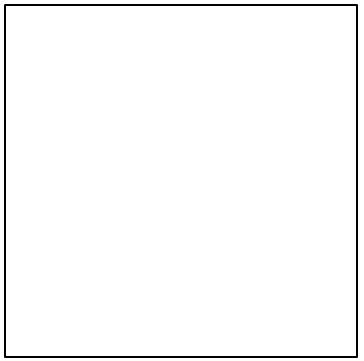
- See *In the Spotlight: The Hit the Target Game* in your textbook for numerous examples of determining the state of the turtle.



Turtle Graphics: Using Loops to Draw Designs

- You can use loops with the turtle to draw both simple shapes and elaborate designs. For example, the following for loop iterates four times to draw a square that is 100 pixels wide:

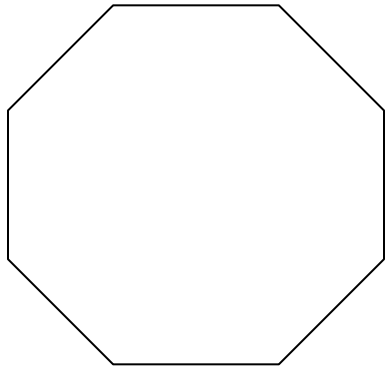
```
for x in range(4):
    turtle.forward(100)
    turtle.right(90)
```



Turtle Graphics: Using Loops to Draw Designs

- This `for` loop iterates eight times to draw the octagon:

```
for x in range(8):
    turtle.forward(100)
    turtle.right(45)
```

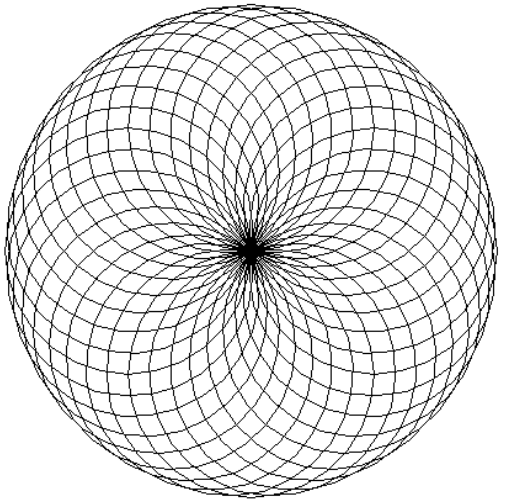


Turtle Graphics: Using Loops to Draw Designs

- You can create interesting designs by repeatedly drawing a simple shape, with the turtle tilted at a slightly different angle each time it draws the shape.

```
NUM_CIRCLES = 36     # Number of circles to draw
RADIUS = 100         # Radius of each circle
ANGLE = 10           # Angle to turn

for x in range(NUM_CIRCLES):
    turtle.circle(RADIUS)
    turtle.left(ANGLE)
```



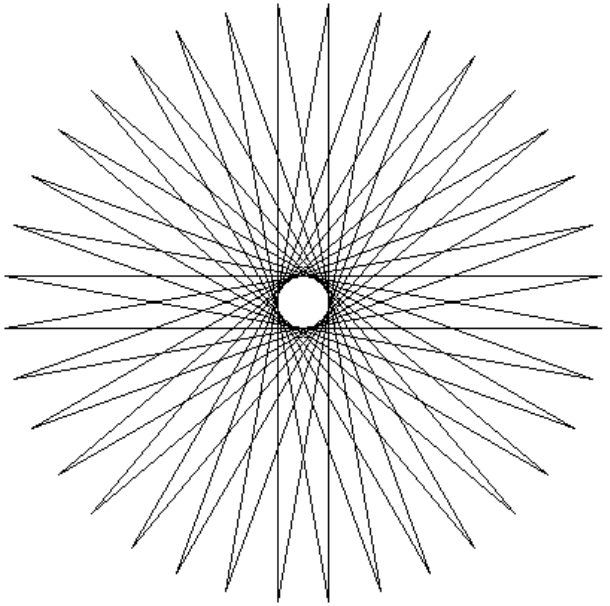
Turtle Graphics: Using Loops to Draw Designs

- This code draws a sequence of 36 straight lines to make a "starburst" design.

```
START_X = -200      # Starting X coordinate
START_Y = 0         # Starting Y coordinate
NUM_LINES = 36      # Number of lines to draw
LINE_LENGTH = 400   # Length of each line
ANGLE = 170         # Angle to turn

turtle.hideturtle()
turtle.penup()
turtle.goto(START_X, START_Y)
turtle.pendown()

for x in range(NUM_LINES):
    turtle.forward(LINE_LENGTH)
    turtle.left(ANGLE)
```



Turtle Graphics: Modularizing Code with Functions

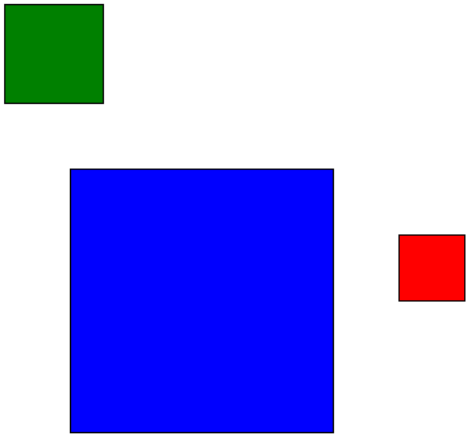
- Commonly needed turtle graphics operations can be stored in functions and then called whenever needed.
- For example, the following function draws a square. The parameters specify the location, width, and color.

```
def square(x, y, width, color):
    turtle.penup()              # Raise the pen
    turtle.goto(x, y)           # Move to (X,Y)
    turtle.fillcolor(color)     # Set the fill color
    turtle.pendown()            # Lower the pen
    turtle.begin_fill()         # Start filling
    for count in range(4):      # Draw a square
        turtle.forward(width)
        turtle.left(90)
    turtle.end_fill()           # End filling
```

Turtle Graphics: Modularizing Code with Functions

- The following code calls the previously shown `square` function to draw three squares:

```
square(100, 0, 50, 'red')
square(-150, -100, 200, 'blue')
square(-200, 150, 75, 'green')
```



Turtle Graphics: Modularizing Code with Functions

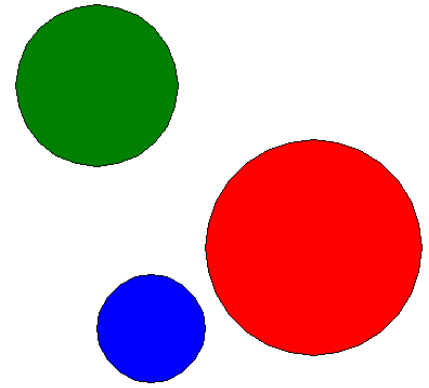
- The following function draws a circle. The parameters specify the location, radius, and color.

```
def circle(x, y, radius, color):
    turtle.penup()                # Raise the pen
    turtle.goto(x, y - radius)    # Position the turtle
    turtle.fillcolor(color)       # Set the fill color
    turtle.pendown()              # Lower the pen
    turtle.begin_fill()           # Start filling
    turtle.circle(radius)         # Draw a circle
    turtle.end_fill()             # End filling
```

Turtle Graphics: Modularizing Code with Functions

- The following code calls the previously shown `circle` function to draw three circles:

```
circle(0, 0, 100, 'red')
circle(-150, -75, 50, 'blue')
circle(-200, 150, 75, 'green')
```



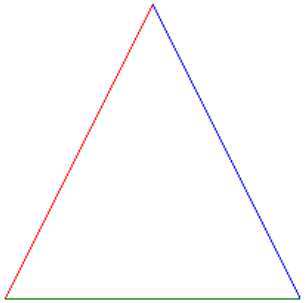
Turtle Graphics: Modularizing Code with Functions

- The following function draws a line. The parameters specify the starting and ending locations, and color.

```
def line(startX, startY, endX, endY, color):
    turtle.penup()                # Raise the pen
    turtle.goto(startX, startY)   # Move to the starting point
    turtle.pendown()              # Lower the pen
    turtle.pencolor(color)        # Set the pen color
    turtle.goto(endX, endY)       # Draw a square
```

Turtle Graphics: Modularizing Code with Functions

- The following code calls the previously shown `line` function to draw a triangle:



```
TOP_X = 0
TOP_Y = 100
BASE_LEFT_X = -100
BASE_LEFT_Y = -100
BASE_RIGHT_X = 100
BASE_RIGHT_Y = -100
line(TOP_X, TOP_Y, BASE_LEFT_X, BASE_LEFT_Y, 'red')
line(TOP_X, TOP_Y, BASE_RIGHT_X, BASE_RIGHT_Y, 'blue')
line(BASE_LEFT_X, BASE_LEFT_Y, BASE_RIGHT_X, BASE_RIGHT_Y, 'green')
```