

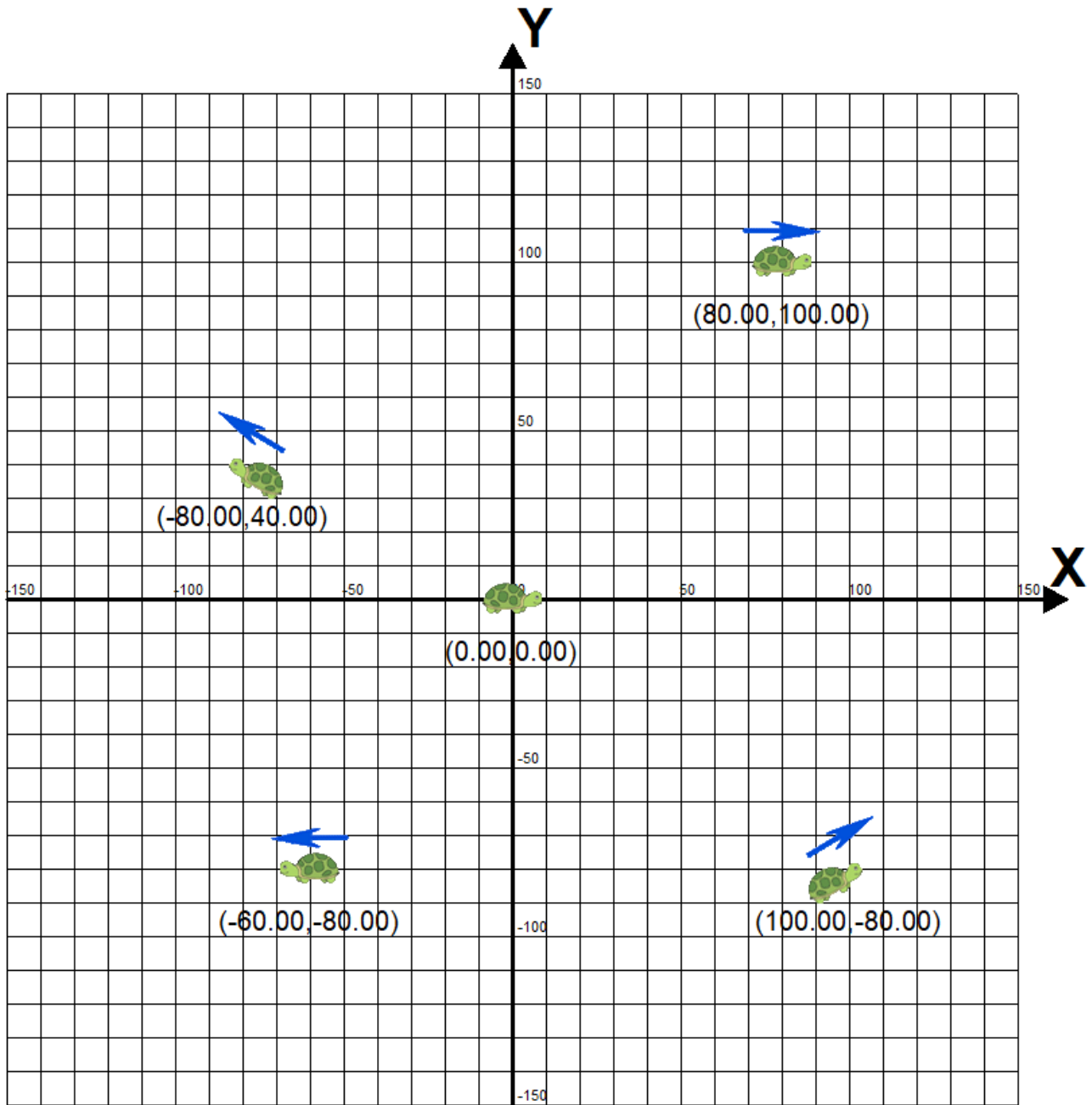
Lesson 1: Simple Geometry Shapes with Python turtle module

There are many different graphical toolkits available for Python. For these lessons we chose one of the simplest: the turtle module. The turtle module provides an environment where turtles move upon a 2-dimensional grid. Turtle has a position and a heading (the direction in which the turtle is facing) as shown below.

Summary: Main Motion Commands

Instruction	What it does
<code>import turtle</code>	Importing a module tells Python that you want to use it
<code>t=turtle.Turtle()</code>	Specifies the name of turtle, could be any name that you want to use
<code>t.goto(x,y)</code> <code>t.setposition(x,y)</code>	Send the turtle to the coordinates x and y
<code>t.forward(value)</code> <code>t.fd(value)</code>	Move the turtle forward by specified distance value, in the direction the turtle is headed
<code>t.right(value)</code> <code>t.rt(value)</code>	Turn turtle right by angle value units. (Units are in degrees, by default)
<code>t.left(value)</code> <code>t.lt(value)</code>	Turn turtle left by angle units. (Units are in degrees, by default)
<code>t.pensize(value)</code>	Set the line thickness value to the number you supply
<code>t.setheading(value)</code>	Set the orientation of the turtle to angle value. (The angle is in degrees, by default.)

Fig 1



To place the turtle on the turtle screen, we need to understand the x- and y-coordinate system used in our Turtle environment. X,Y coordinate system is drawn in Fig 1. In

the graph the dark horizontal line is called the x -axis, and it runs from left to right. The dark vertical line is the y -axis, running from bottom to top. We call the point (initial position of the turtle) where these lines meet, $(0, 0)$, the *origin* because all other points on the grid are labeled with coordinates measured from, or *originating* from, that point. Think of the origin, $(0, 0)$, as the center of your screen. Every other point you want to find can be labeled with an x - and y -coordinate by starting at the origin and moving left or right, down or up. We label points on a graph with this pair of coordinates inside parentheses, separated by a comma: (x, y) . The first number, the x -coordinate, tells us how far to move left or right, while the second number, the y -coordinate, tells us how far to move up or down. Positive x -values tell us to move right from the origin; negative x -values tell us to move left. Positive y -values tell us to move up from the origin, and negative y -values tell us to move down. Look at the points labelled in the Figure. The point in the upper right is labelled with the x and y -coordinates $(80, 100)$. To find the location of this point, we start at the origin $(0, 0)$ and move 80 spaces to the right (because the x -coordinate, 80, is positive) and then 100 spaces up (because the y -coordinate, 100, is positive). To get to the point in the lower right, $(100, -80)$, we go back to the origin and then move right 100 spaces or units. This time, the y -coordinate is -80 , so we move *down* 80 units. Moving right 100 and down -80 puts us at $(100, -80)$. For $(-80, 40)$, we move *left* 80 units from the origin and then up 40 units to the point in the upper left. Finally, for $(-60, -80)$, we move left 60 units and then down 80 units to the lower-left point.

By default, turtle screen occupies 0.5 of full width and 0.75 of full height computer screen.

Code `t.fd(value)` moves the turtle (with a name `t`) forward by specified distance `value`, in the direction the turtle is headed turtle as shown in the Fig 1.

Example #1: Draw a line

One of the simplest things you can do using the turtle module is drawing a line. First, open a new file window to begin the lesson by clicking the File button on the Shell window. (Screenshot of the Shell window is shown below). The new file window appears.

Type the following code that produces a line drawing

```
import turtle  
  
t=turtle.Turtle()  
  
t.forward(200)
```


The first line makes all the “turtle” commands available for us. This line allows us to use Python turtle library.

The second line (command) creates a new turtle named by `t`. We will call it “`t`”. You can give the turtle any name you want.

Using the third line (command) we move the turtle previously named as “`t`” by 200 pixels. You can type any pixel number you want. Pixel is a single point on the screen-the smallest element that can be represented. Everything you see on your computer monitor is made up of pixels, which are tiny, square dots.

After typing these three lines in the new file window, you can run this code using the button “Run”. The editor will ask you to save the file. You need to save the file with your desired name in any folder you please. For example, create a new folder in the documents directory and name this folder “Python Projects” which puts your file to this directory. The result of code execution:

```
1 import turtle  
2 t=turtle.Turtle()  
3 t.forward(200)
```



By default, if you don't specify, turtle starts to move from position (x=0, y=0) the orientation of the turtle to angle value equal 0.

Example #2:

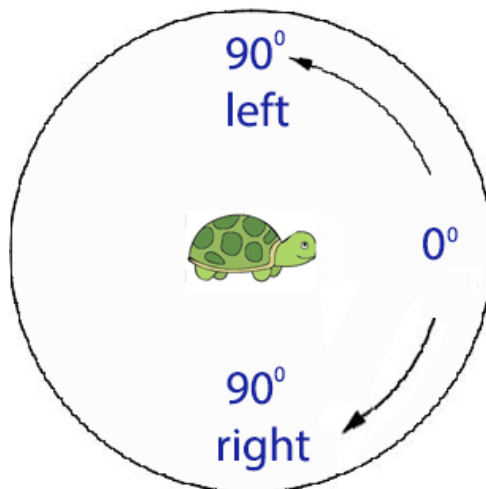
```
import turtle  
  
t=turtle.Turtle()  
  
t.setheading(30) # Set the orientation of the turtle to angle (degrees).  
  
t.forward(300)
```

Here we inserted one additional command that allows us to change the line's trajectory (30 degrees instead of the initial horizontal direction)

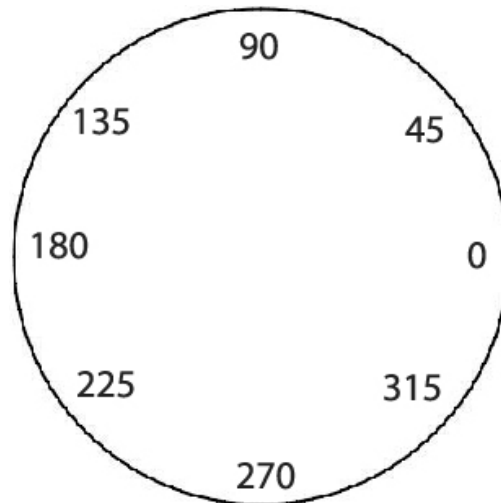
If you haven't learned about degrees yet, here's how to think about them. Imagine that you were standing in the center of a circle.

- The direction you are facing is 0 degrees.
- If you hold out your left arm, that's 90 degrees left.
- If you hold out your right arm, that's 90 degrees right.

You can see this 90-degree turn to the left or right here:

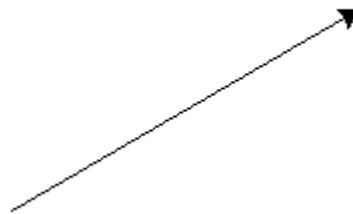


If you continue around the circle to the right from where you right arm positing, 180 degrees is directly behind you, 270 degrees is the direction your left arm is positing, and 360 degrees is back where you started: degrees from 0 to 360. The degrees in a full circle, when turning to the right, can be seen here I 45-degree increments:



So, expected output

```
1 import turtle
2 t=turtle.Turtle()
3 t.setheading(30)
4 t.forward(200)
```



Example #3: (controlling of the line thickness with code <t.pensize(10)>)

Let's try to change the thickness of the line and see what happens.

```
import turtle

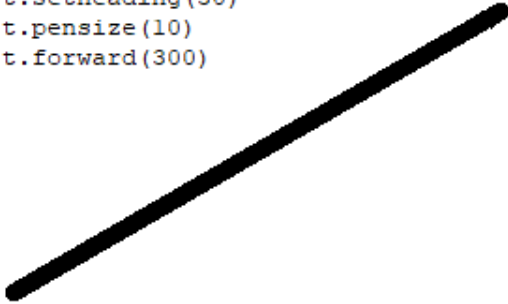
t=turtle.Turtle()

t.setheading(30) #
```

```
t.pensize(10)
t.forward(300)
```

Expected output

```
import turtle
t=turtle.Turtle()
t.setheading(30)
t.pensize(10)
t.forward(300)
```



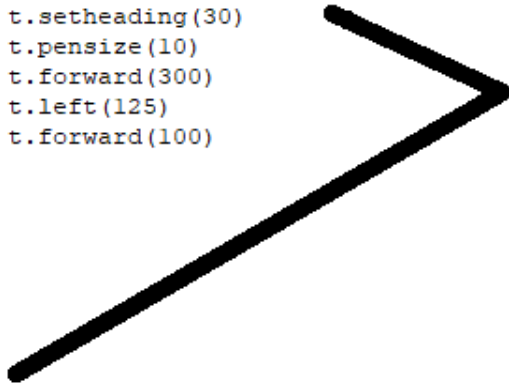
Example #4: (Two lines)



```
import turtle
t=turtle.Turtle()
t.setheading(30)
t.pensize(10)
t.forward(300)
t.right(125)
t.forward(100)
```

This example includes the additional line `<t.right(125)>` that turns the turtle right 125 degrees. You can also turn it left using this variation of the command: `<t.left(125)>`.

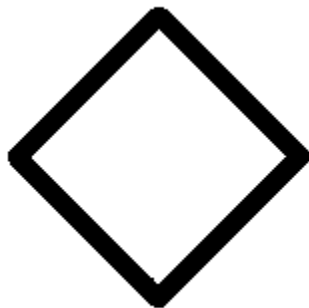
```
import turtle
t=turtle.Turtle()
t.setheading(30)
t.pensize(10)
t.forward(300)
t.left(125)
t.forward(100)
```



Example #5: (Square)



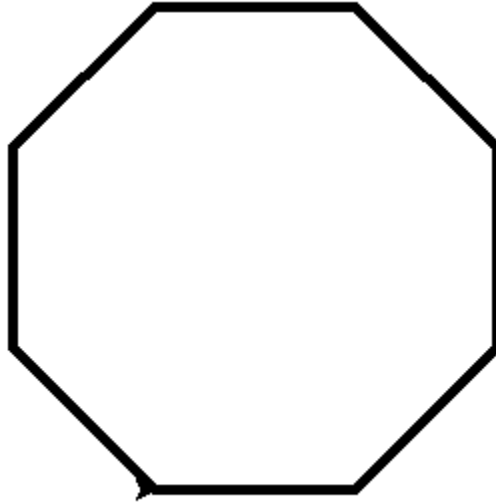
```
import turtle
t=turtle.Turtle()
t.pensize(10)
t.forward(100)
t.left(90)
t.forward(100)
t.left(90)
t.forward(100)
t.left(90)
t.forward(100)
```



```
import turtle
t=turtle.Turtle()
t.pensize(10)
t.setheading(45)
t.forward(100)
t.left(90)
t.forward(100)
t.left(90)
t.forward(100)
t.left(90)
t.forward(100)
```

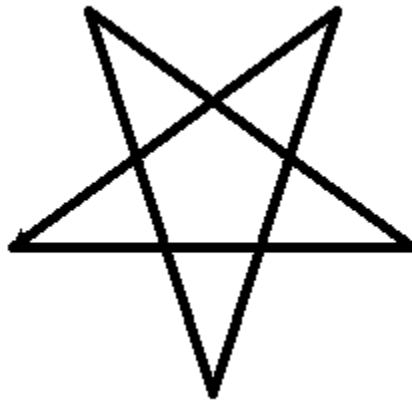

Example #6: (Octagon)

```
import turtle
t=turtle.Turtle()
t.pensize(5)
t.forward(100)
t.left(45)
t.forward(100)
t.left(45)
t.forward(100)
t.left(45)
t.forward(100)
t.left(45)
t.forward(100)
t.left(45)
t.forward(100)
t.left(45)
t.forward(100)
t.left(45)
t.forward(100)
t.left(45)
t.forward(100)
t.left(45)
```



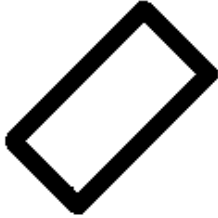
Example #7: (Star)

```
import turtle
t=turtle.Turtle()
t.pensize(5)
t.forward(200)
t.left(144)
t.forward(200)
t.left(144)
t.forward(200)
t.left(144)
t.forward(200)
t.left(144)
t.forward(200)
t.left(144)
t.forward(200)
```



Challenges: write codes to create the following geometry shapes:

1. Expected output



2. Expected output



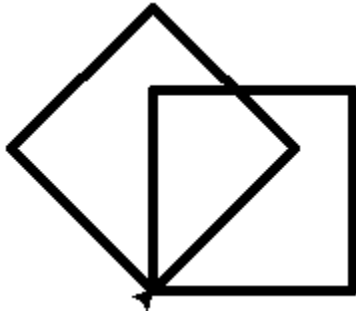
3. Expected output



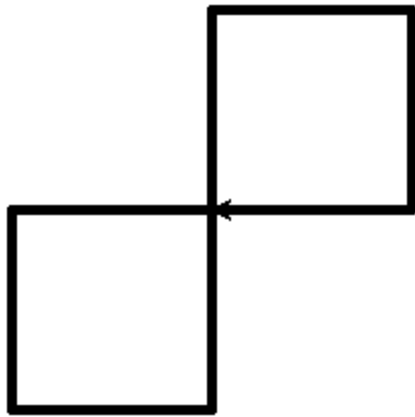
4. Expected output



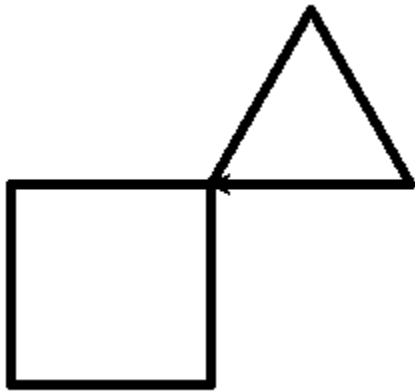
5. Expected output



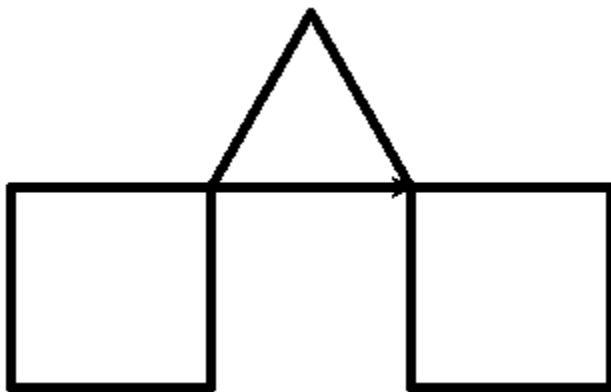
6. Expected output



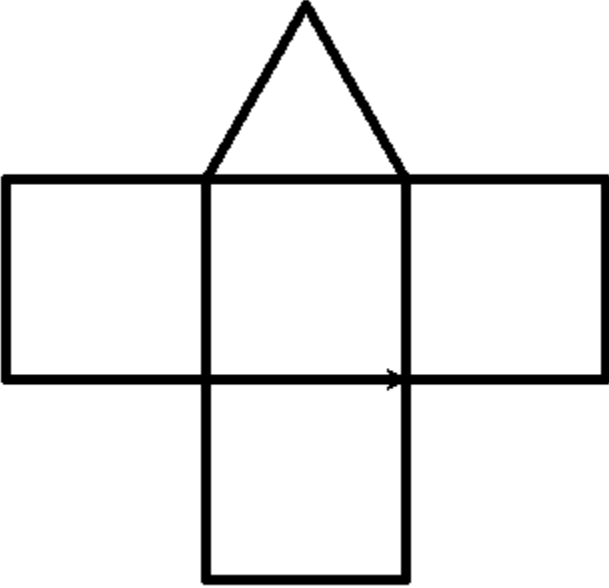
7. Expected output



8. Expected output



9. Expected output



Python + Math



Code

for Kids Output



```
import turtle
t=turtle.Turtle()
t.forward(200)
```



```
import turtle
t=turtle.Turtle()
t.setheading(30)
t.forward(200)
```



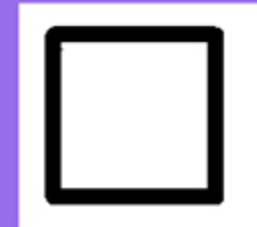
```
import turtle
t=turtle.Turtle()
t.setheading(30)
t.pensize(10)
t.forward(200)
```



```
import turtle
t=turtle.Turtle()
t.setheading(30)
t.pensize(10)
t.forward(200)
t.right(125)
t.forward(100)
```



```
import turtle
t=turtle.Turtle()
t.pensize(10)
t.forward(100)
t.right(90)
t.forward(100)
t.right(90)
t.forward(100)
t.right(90)
t.forward(100)
```



```
import turtle
t=turtle.Turtle()
t.pensize(10)
t.forward(100)
t.right(144)
t.forward(100)
t.right(144)
t.forward(100)
t.right(144)
t.forward(100)
t.right(144)
t.forward(100)
```



```
import turtle
t=turtle.Turtle()
t.pensize(10)
t.forward(100)
t.right(60)
t.forward(100)
t.right(60)
t.forward(100)
t.right(60)
t.forward(100)
t.right(60)
t.forward(100)
t.right(60)
t.forward(100)
```



```
import turtle
t=turtle.Turtle()
t.pensize(10)
t.forward(100)
t.right(45)
t.forward(100)
t.right(45)
t.forward(100)
t.right(45)
t.forward(100)
t.right(45)
t.forward(100)
t.right(45)
t.forward(100)
t.right(45)
t.forward(100)
t.right(45)
t.forward(100)
```



Lesson 1

Simple Geometry Shapes

