**Festive Coding Handout**

**BCC 12/12/19**

**By, Liz Goldstein**

Email for class [JPGs—codecandy123@gmail.com](mailto:JPGs—codecandy123@gmail.com)

Password: Ilovecode123

(capital I; the rest lower case)

Personal Email--[lizgoldstein15@gmail.com](mailto:lizgoldstein15@gmail.com)

Find Programs at:

https://github.com/onceuponpython

<https://repl.it/@lizgoldstein15>

The below code is for Python 3.x.

**Combining Two Images**



This program uses the PIL library, which does not always

work in repl.it, but will work in a regular IDE such as Visual

Studio, PyCharm, or Spyder. It combines two images

named image1 and image2 to create two different

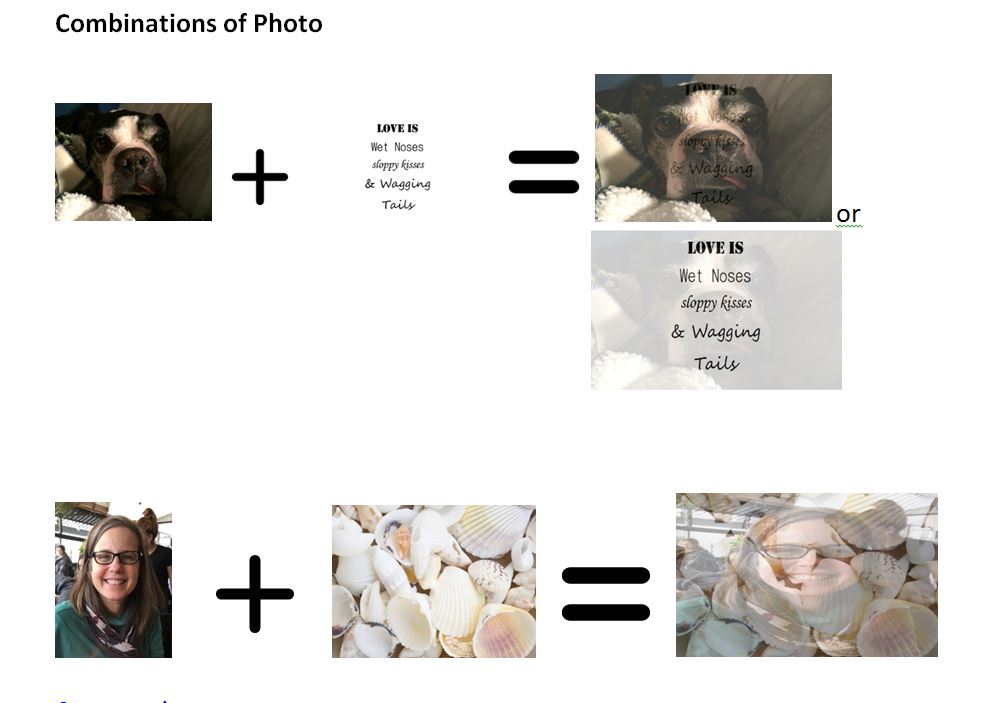
combined photos. The difference between the two

combined photos is how dominant each image is. The

program cannot produce a JPG. However, in Windows

you can use the Snipping tool to create a JPG of the

image.

****

**The Code below is also located at:** <https://repl.it/@lizgoldstein15/photo-blending-simple> However, it may not work in repl.it.

from PIL import Image

# Take two images for blending them together

image1 = Image.open("jackie.jpg")

image2 = Image.open("smallest.jpg")

# Make the images of uniform size

image1 = image1.resize((400, 400))

image2=image2.resize((400, 400))

# Make sure images got an alpha channel

image1 = image1.convert("RGBA")

image2 = image2.convert("RGBA")

# alpha-blend the images with varying values of alpha

alphaBlended1 = Image.blend(image1, image2, alpha=.9)

alphaBlended2 = Image.blend(image1, image2, alpha=.5)

# Display the alpha-blended images

alphaBlended1.show()

alphaBlended2.show()

#save image

alphaBlended2.save("jackiecombo1.png")

alphaBlended2.save("jackiecombo2.png")

**Drawing with Turtle**

The first line of code tells Pythanna to start-up the turtle library. You can change **50** to **100**, and run the program again. The second line of code names our turtle “t”. You can give your turtle another name like Martha, which in code would be **martha=turtle.Turtle()** or tom, which in code would be **tom=turtle.Turtle()**. In the third line of code, we use the **forward** command, which tells the turtle to move in the direction it is facing. The number in the parentheses **()** tells the turtle how many steps it should move forward. The bigger the number we give the turtle the further it will move.

**Changing the Turtle’s Direction**

**The turtle always starts off facing to the right and in the center of the page.** You can change the turtle’s direction by telling the turtle to move turn **left** or turn **right**. Left commands the turtle to stay in place and to turn a certain number of degrees counter-clockwise. The right command does the same, but the turtle will turn clockwise. Add the last two lines of code listed below, and see what happens.

**The Code**

import turtle

t=turtle.Turtle()

t.shape("turtle")

t.forward(50)

**t.right(90)**

**t.forward(50)**

At the end of the program, your turtle is facing down like it is about to crawl down the monitor. Can you figure out how to get the turtle to two make two more lines and two more turns to complete a square? The correct code is below. The lines with the hashes are not executed, but instead it gives us information about the code.

**The Code** is located at: <https://repl.it/@lizgoldstein15/Square-longways>

import turtle

t=turtle.Turtle()

t.shape("turtle")

#square

#first side

t.forward(50)

t.right(90)

#second side

t.forward(50)

t.right(90)

#third side

t.forward(50)

t.right(90)

#fourth side

t.forward(50)

t.right(90)

Since the four sides have the same two commands, we can shorten the code by telling the turtle to do the same things four times. The code is below and located at: <https://repl.it/@lizgoldstein15/squareez>

import turtle

t=turtle.Turtle()

t.shape("turtle")

#for loop

for i in range(4):

t.forward(50)

t.right(90)

The line after the hash (which is the first line of code) tells the turtle to execute the next two lines of code four times. This first line is a called a **for loop**. Note that the code lines that are within the loop are indented.

If we want the turtle, to create more than one square, then we can create an **outer loop** and an **inner loop**. The outer loop will tell the turtle how many squares to make, and the inner loop will tell it how to build each square. Below is the code creating 8 squares each 45 degrees apart. The code is located at: <https://repl.it/@lizgoldstein15/squaresinround>

import turtle

t=turtle.Turtle()

t.shape("turtle")

#outer loop; makes 8 squares

for i in range(8):

t.right(45)

#inner loop to build each square; will execute the two lines of code four times

for i in range(4):

t.forward(50)

t.right(90)

Here is the same program, but we have changed pen colors. The new lines are highlighted in yellow. This code is located at: <https://repl.it/@lizgoldstein15/squaresinroundcolor>

import turtle

t=turtle.Turtle()

t.shape("turtle")

#outer loop

for i in range(8):

t.right(45)

#square with for loop

for i in range(4):

if i == 2:

t.pencolor("blue")

else:

t.pencolor("hot pink")

t.forward(50)

t.right(90)

The lines related to the pen color, tell the turtle that on the **third side** of the square, make the pen color blue. For all other sides, make the pen color hot pink. You would think the line of code if i == 2: would mean if the turtle was working on the second side of the square, but the turtle starts counting at zero. This is why when the turtle gets to the number two loop, it is actually drawing the third side.

0=first side

1=second side

2=third side

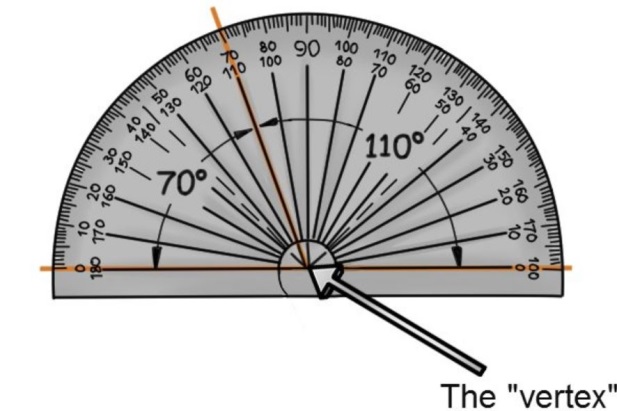
3=fourth side

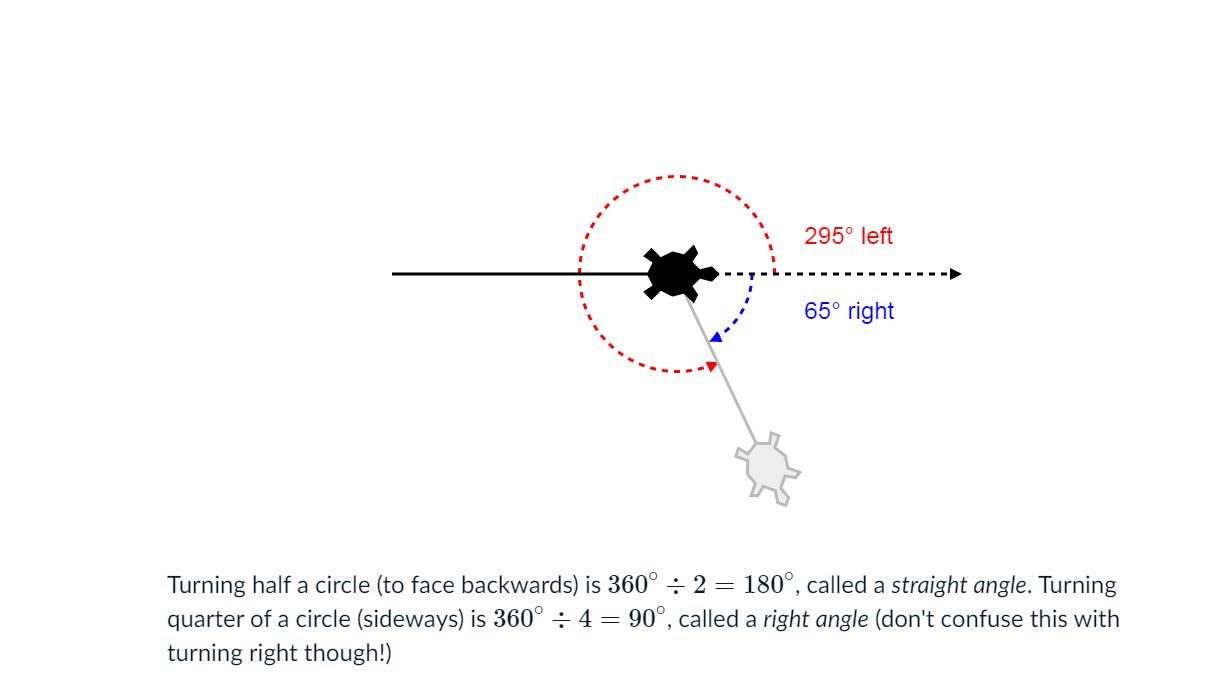
Note that you need to put the names of the colors in quotes (some people call these speech tags). Also to write an equal sign, we needed to write == rather than =. This is because a single equal sign is to assign a name to something. For instance, t=turtle.Turtle() assigns the name “t” to our turtle.

**Understanding How to Get the Turtle to Turn**

The turtle can turn 360 degrees to the right and left. **For the turtle, its turns correspond to the external angles at the vertex.**

While in geometry class, you measured a shape’s internal angles. If the turtle was facing to the left, and we wanted it to stay at the same location, but face where the 70 degree angle is marked on the protractor, we would tell the turtle to turn right 70 degrees—t.right(70) or left 110 degrees—t.left(110). Remember that “t” is the name of your turtle. If you name your turtle “earl”, then the command for turn right 70 degrees would be earl.right(70) and not t.right(70).

****

The below image shows that if you have the turtle turn 295 degrees to the left, it is the same thing as the turtle turning 65 degrees to the right. 

**Drawing an Equilateral Triangle**

import turtle

t=turtle.Turtle()

t.shape("turtle")

#first side

t.forward(70)

t.right(120)

#second side

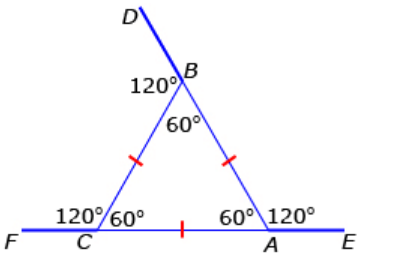
t.forward(70)

t.right(120)

#third side

t.forward(70)

t.right(120)



The above code is at: https://repl.it/@lizgoldstein15/trianglefull

The above figure shows the internal and external angles of an equilateral triangle. The code above shows you that you need to give the turtle the **external angle** in order to have the turtle draw the equilateral triangle.

Here is the same program, but we added a **for loop** and changed pen colors using an **if-else statement**. The code is located at: <https://repl.it/@lizgoldstein15/triangleforloop>

import turtle

t=turtle.Turtle()

t.shape("turtle")

#for loop

for i in range(3):

#if-else statement

if i==0:

t.pencolor("red")

else:

t.pencolor("green")

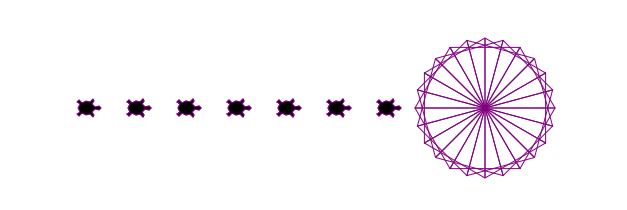
t.forward(70)

t.right(120)

Here is another program drawing a series of 24 equilibrium triangles all 15 degrees apart and adds six turtle stamps, which uses the **stamp()** command. Note that the **penup** command tells the turtle to stop drawing. The **pendown** command tells the turtle to start drawing again. This code is located at: <https://repl.it/@lizgoldstein15/spyrowithstamp>

import turtle

t=turtle.Turtle()



t.shape("turtle")

t.speed("fastest")

t.pencolor("purple")

for i in range(24):

t.right(15)

for i in range(3):

t.forward(70)

t.right(120)

t.penup()

t.backward(400)

for i in range(6):

t.stamp()

t.forward(50)

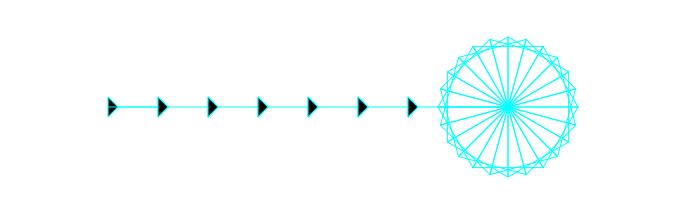
t.stamp()

t.penup()

Here are just a couple of changes to the code (highlighted) makes a pretty different picture. The hash in front of the command **t.penup()** means that the turtle will ignore this line of code. The code is located at: <https://repl.it/@lizgoldstein15/spyrostamp2>

import turtle

t=turtle.Turtle()



t.shape("arrow")

t.speed("fastest")

t.pencolor("aqua")

for i in range(24):

t.right(15)

for i in range(3):

t.forward(70)

t.right(120)

#t.penup()

t.backward(400)

for i in range(6):

t.stamp()

t.forward(50)

t.stamp()

t.penup()

Now that you know how to drive the turtle, the below code shows you how to use the turtle to draw a tree with a ball on top. The code is located at: <https://repl.it/@lizgoldstein15/Tree-with-Ball>

**Code**

import turtle

t=turtle.Turtle()

t.shape("turtle")

t.pendown()

t.width(2)

#tree

t.color("dark olive", "green")

t.begin\_fill()

#first 1/2 of tree

t.forward(200)

t.left(135)

t.forward(200)

t.right(135)

t.forward(100)

t.left(135)

t.forward(150)

t.right(135)

t.forward(60)

t.left(135)

t.forward(100)

t.left(90)

#Second 1/2 of tree (reverse)

t.forward(100)

t.left(135)

t.forward(60)

t.right(135)

t.forward(150)

t.left(135)

t.forward(100)

t.right(135)

t.forward(200)

t.left(135)

t.forward(200)

t.end\_fill()

#trunk

t.color("dark brown", "tan")

t.begin\_fill()

t.penup()

t.backward(75)

t.pendown()

t.color("brown")

t.width(4)

t.right(90)

t.forward(100)

t.left(90)

t.forward(75)

t.right(90)

t.backward(100)

t.end\_fill()

#create ball on top of tree

t.penup()

t.right(90)

t.backward(200)

t.right(90)

t.forward(325)

t.left(90)

t.forward(210)

t.right(90)

t.color("yellow")

t.begin\_fill()

t.circle(30)

t.end\_fill()

t.penup()

t.forward(300)

**Turtle Holiday Art Program**

Another program, which you can cut and paste is at:

https://repl.it/@lizgoldstein15/Turtle-Holiday-Art