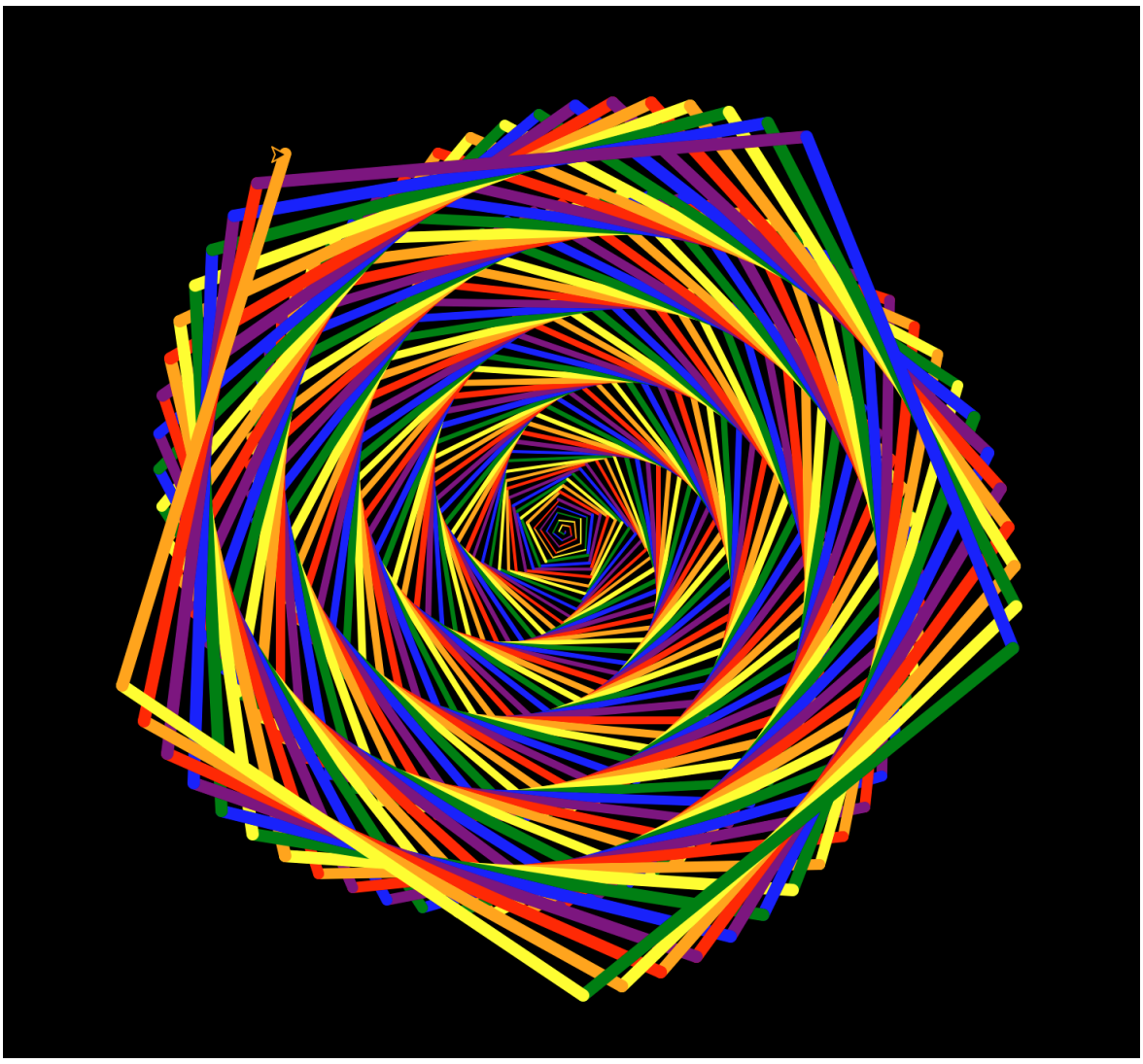
Mark A. Miller

Coding with Python for Absolute Beginners: Workbook 1



Version 0.1 DRAFT February 2019

Introduction

This experimental workbook has been written primarily as an aid to the students in my introductory programming classes. It is largely based off of the ideas in the excellent book Teach Your Kids To Code: A Parent-Friendly Guide To Python Programming by Bryson Payne.

In teaching my first students, it quickly became evident that the learning curve in Mr. Payne's fantastic introduction to programming may still be too steep for most people. I believe that most students with no programming or command line experience require much more scaffolding and guided discovery than most computer scientists seem to think. If this isn't the case, then why are there so many millions of unfilled, high-paid jobs that involve programming?

Real programming is hard, but I believe that if enough of the right seeds are planted, and enough learning aids are provided, practically anybody can learn basic programming, particularly in the easy-to-read Python language.

I'll just have to see how well my ideas for these exercises actually work in the classroom. Expect them to change and improve as time goes along!

Mark Miller

Somewhere in the Sierra Nevada, California

Unit 1: Tell Python to say Hello, World!

```
print("Hello, World!")
```

With Python, all it takes is one line of *code* to create a computer program! *Code* is just written instructions that tell a computer what to do. Code used to be stored as holes punched into thick paper *punchcards* that were then read by a computer that might have been as big as an entire room. Now things are much easier, and we are able to simply save code on our laptop computers in text files and share code almost instantly around the world, and even out to space, via the internet.

Python is a *programming language* that is growing in popularity because for one thing, Python code reads a lot more like English than most other computer languages. This makes it easy to use. Another reason for its popularity is, well, its popularity. The more Python users there are, the better the language gets. This is because as they do their work, people take the time to develop and share useful *libraries* that anybody can use. In a lot of ways Python is a community effort. People are inventing new words in Python all the time and sharing them as libraries, making the language easier to use and more powerful as time goes along. In Unit 2, we will learn to use a decades-old library called `turtle` to draw pretty pictures.

The "Hello, World!" program uses just one command, the `print()` function from Python's *standard library*. The standard library comes prepackaged as part of the Python language, and Python always understands commands from its standard library.

The `print()` function has an interesting history. Until the 1970s, the most inexpensive way to look at a computer's output was often through the use of electronic typewriters and printers, and so as people invented programming languages to operate computers, the results were often literally printed onto paper. Computer screens and monitors were too expensive for most users! One result of this history is that most computer languages have some sort of basic `print` function. Today things are much easier, and the computer just "prints" its output to the screen.

In this unit we will get used to programming in Python by using the standard library functions `print()` and `input()` to write simple "Hello!" programs.

Python - helloworld.py

The traditional first program in any computer language is one that simply prints "Hello, World!" to the standard output. Try it below!

Write the output of the Python code. If you aren't sure what the code will do, you can just run the code to see what it does and then write its output here.

```
print("Hello, World!")
```

Output:

```
print("Hello, Sarah!")
```

Output:

Using the pattern above, try to write a one-line Python program that says: "Hello, everybody!" Test out your code by running it on your computer.

Write your code here:

Using the pattern above, try to write a one-line Python program that calls you by name: "Hello, <name>" where <name> is your own first name. Test out your code by running it on your computer.

Write your code here:

Using the pattern above, try to write a one-line Python program that calls you by name: "<name> is so smart and cool!" where <name> is your own first name. Test out your code by running it on your computer.

Write your code here:

Python - YourName.py

The next step is to write a Python program with two lines, one that accepts input from the keyboard, and a second line that prints something out to the standard output. We can use the `input()` function along with the `print()` function that we already used for our "Hello, World!" program.

Run this code and describe what it does below:

```
name = input("Hi, what's your name? ")
print("Nice to meet you,", name)
```

Description:

Run this code and describe what it does below:

```
firstname = input("Hi, what's your name? ")
print("Nice to meet you,", firstname)
```

Description:

Using the pattern above, write a two-line Python program that asks for your name and then says "Good night, `<name>`!" where `<name>` is your name. Test out your code by running it on your computer.

Write your code here:

Using the pattern above, write a two-line Python program that asks for your name and then says "`<name>` is so smart and cool!" where `<name>` is your name. Test out your code by running it on your computer.

Write your code here:

Using the pattern above, write a two-line Python program that asks for your name and then says something nice or funny about you. Test out your code by running it on your computer.

Write your code here:

Python - MadLibs.py

With just two standard Python functions, `input()` and `print()`, we can make a very simple game called Mad Libs!

Run this code and describe what it does below:

```
noun = input("Please enter a noun: ")
verb = input("Please enter a verb ending in -ed: ")
print("Your Madlib:")
print("The", noun, verb, "over the lazy brown dog.")
```

Description:

Run this code and describe what it does below:

```
adjective = input("Please enter an adjective: ")
noun = input("Please enter a noun: ")
verb = input("Please enter a verb ending in -ed: ")
print("Your Madlib:")
print("The", adjective, noun, verb, "over the lazy brown dog.")
```

Description:

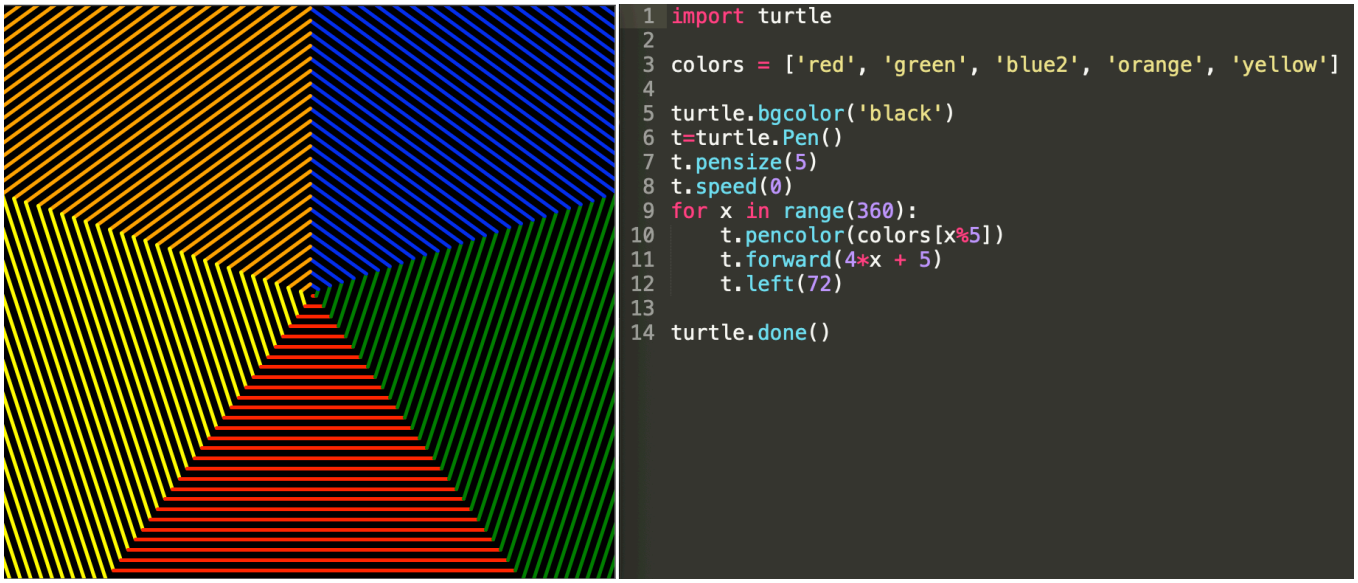
Using the pattern above, write a Python program that asks for the user to name an animal and then uses that information to make a sentence that says something like "The `<adjective>` `<noun>` `<verb>` over the lazy brown `<animal>`". To do this, change the `print()` statement by removing the word `dog` and adding a new `animal` variable after the end of the quoted sentence. Test out your code by running it on your computer.

Write your code here:

If you want, write a totally different kind of Mad Libs program (make it funny!), or move on to the next section.

(optional) Write your code here:

Unit 2: Introducing our new friend, Turtle



```
1  import turtle
2
3  colors = ['red', 'green', 'blue2', 'orange', 'yellow']
4
5  turtle.bgcolor('black')
6  t=turtle.Pen()
7  t.pensize(5)
8  t.speed(0)
9  for x in range(360):
10     t.pencolor(colors[x%5])
11     t.forward(4*x + 5)
12     t.left(72)
13
14 turtle.done()
```

Figure 1. A pentagon-like spiral drawn with 11 lines of Python code using the Turtle library.

The fascinating picture above can be drawn with only 11 lines of code. You're soon going to understand every one of them, and in the process learn a few of the basics of computer programming with the Python language.

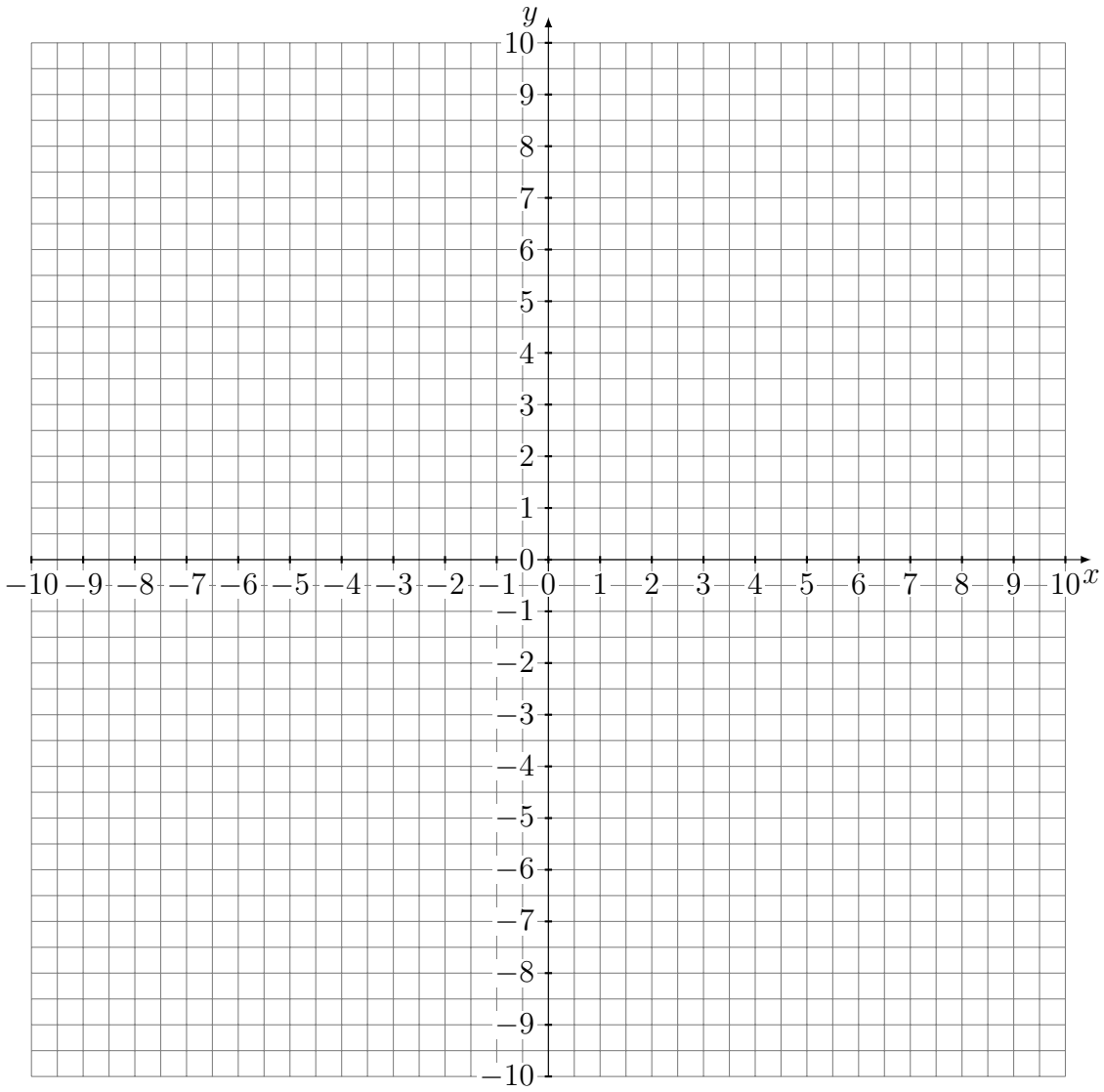
`Turtle` is a *library* of the Python *language* that does all the behind-the-scenes work of drawing things on the screen. This means that we don't have to worry about a lot of complicated details; we can just tell Turtle's "pen" to do what we want it to do. We can create a Turtle pen with the command `t = turtle.Pen()`, and Turtle lets us change the background color of our picture window with `turtle.bgcolor()`. It lets us set the pen color with `t.pencolor()`, set the pen's width with `t.pensize()`, draw forward a certain number of pixels with `t.forward()`, and turn left a certain number of degrees with `t.left()`.

In this unit we will build up our understanding of the code in Figure 1 step by step by actually drawing simpler code on paper with colored pencils. Have fun!

Python Turtle - drawing.1.0.py

Try to draw the output of the Python code in the graphing area below. Use a protractor, straightedge, and/or compass as appropriate. Run the code on your computer and compare the results. If you aren't sure what the code will draw, you can just run the code to see what it does, then draw what you see on the graph paper. By default, the Turtle pen starts out pointing to the right.

```
import turtle
t = turtle.Pen()
t.forward(100)
turtle.done()
```

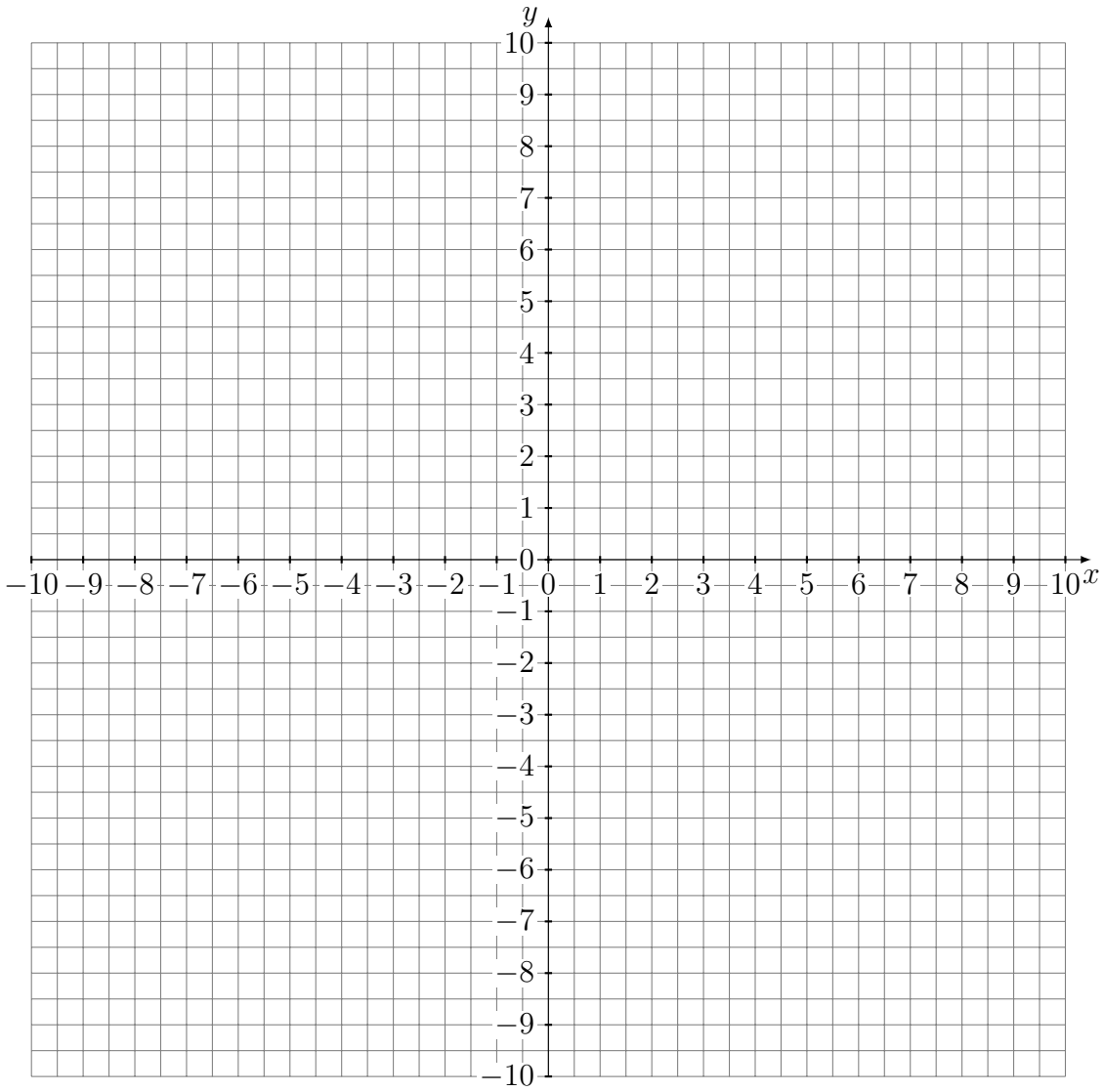


1 unit = 10 pixels

Python Turtle - drawing.1.0.5.py

Try to draw the output of the Python code in the graphing area below. Use a protractor, straightedge, and/or compass as appropriate. Run the code on your computer and compare the results. If you aren't sure what the code will draw, you can just run the code to see what it does, then draw what you see on the graph paper. By default, the Turtle pen starts out pointing to the right.

```
import turtle
giantwhiterabbit456 = turtle.Pen()
giantwhiterabbit456.forward(50)
turtle.done()
```

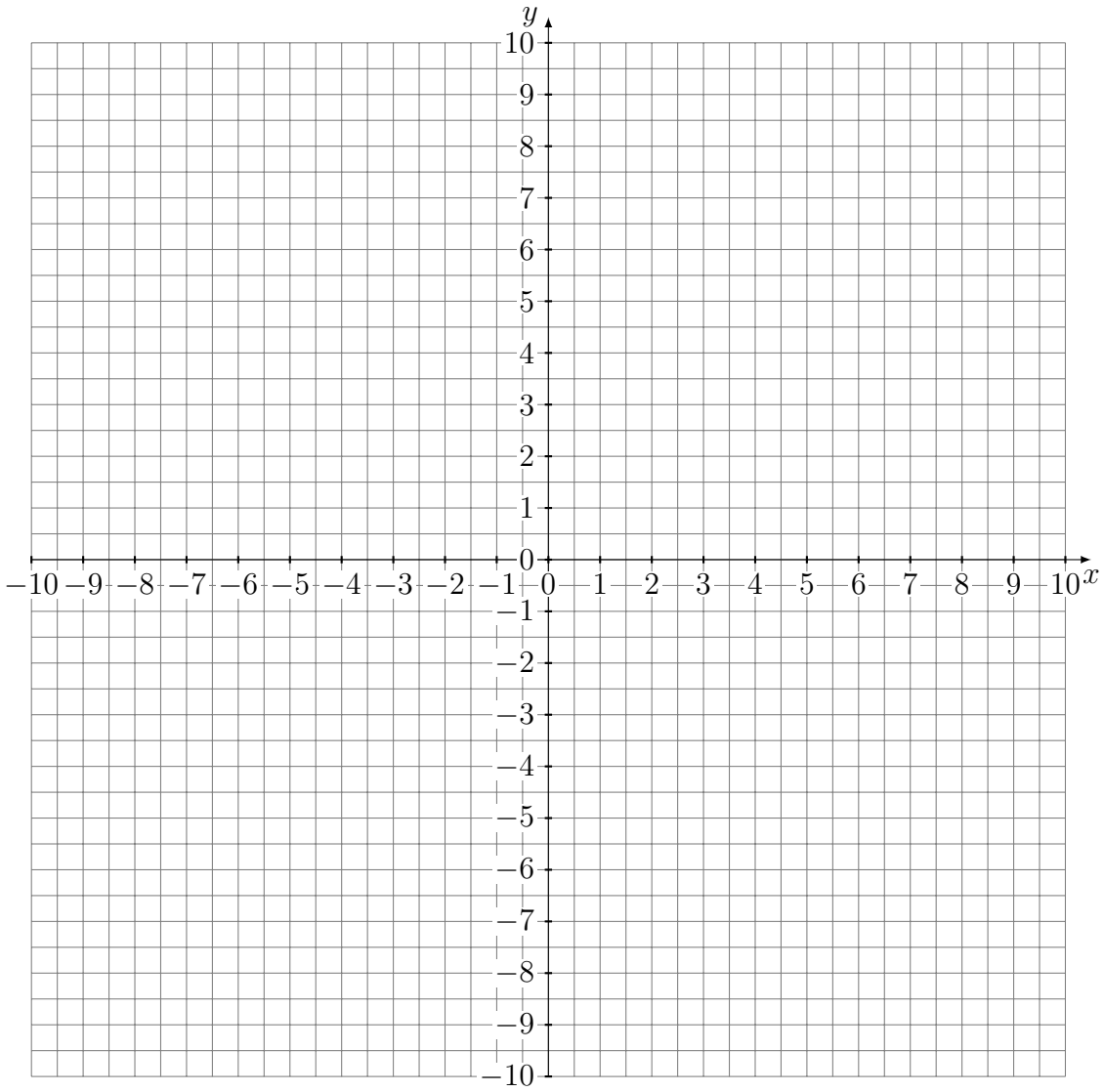


1 unit = 10 pixels

Python Turtle - drawing.1.1.py

Try to draw the output of the Python code in the graphing area below. Use a protractor, straightedge, and/or compass as appropriate. Run the code on your computer and compare the results. If you aren't sure what the code will draw, you can just run the code to see what it does, then draw what you see on the graph paper. By default, the Turtle pen starts out pointing to the right.

```
import turtle
t = turtle.Pen()
t.backward(100)
turtle.done()
```

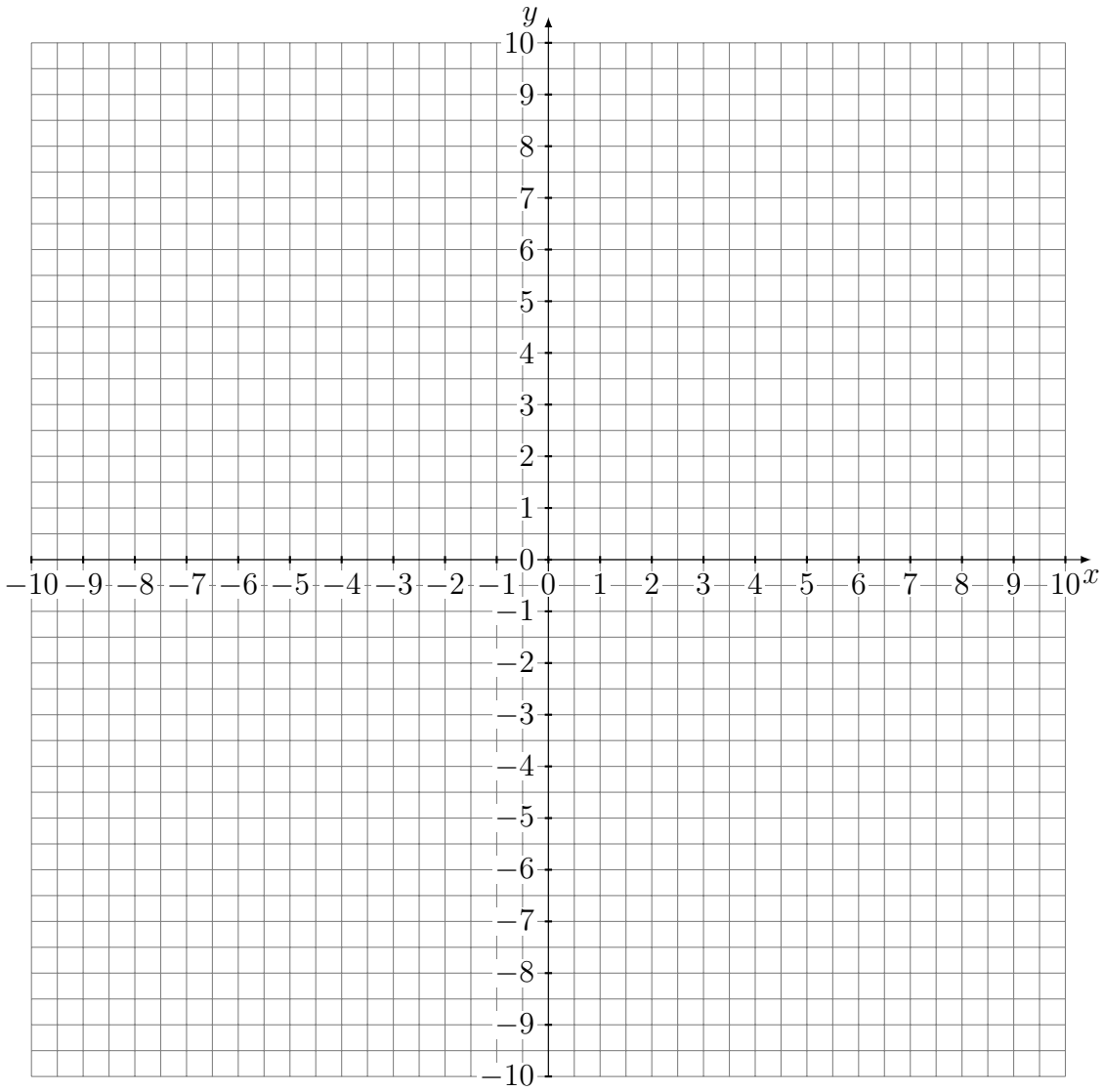


1 unit = 10 pixels

Python Turtle - drawing.1.2.py

Try to draw the output of the Python code in the graphing area below. Use a protractor, straightedge, and/or compass as appropriate. Run the code on your computer and compare the results. If you aren't sure what the code will draw, you can just run the code to see what it does, then draw what you see on the graph paper. By default, the Turtle pen starts out pointing to the right.

```
import turtle
t = turtle.Pen()
t.forward(-100)
turtle.done()
```

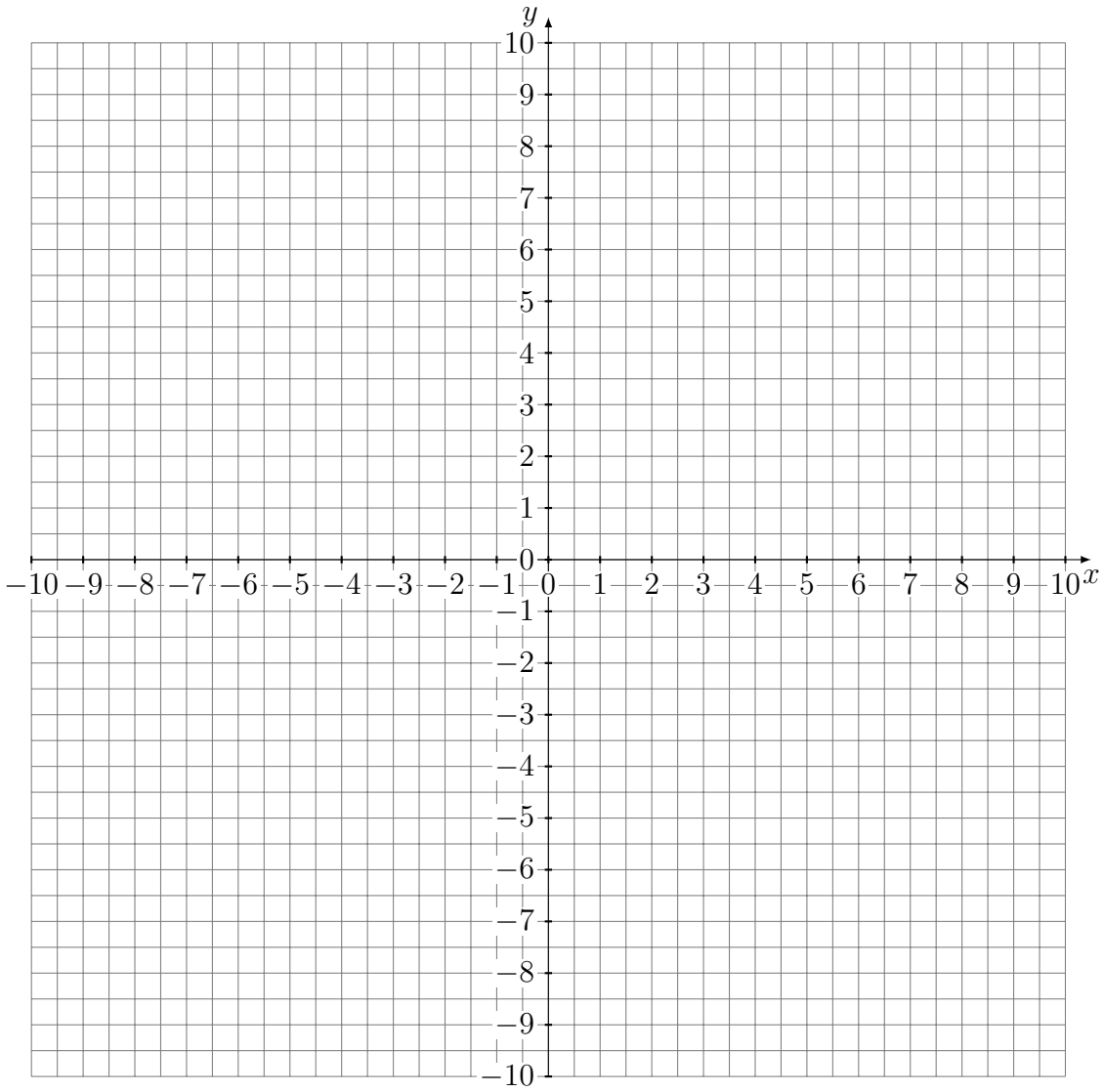


1 unit = 10 pixels

Python Turtle - drawing.1.3.py

Try to draw the output of the Python code in the graphing area below. Use a protractor, straightedge, and/or compass as appropriate. Run the code on your computer and compare the results. If you aren't sure what the code will draw, you can just run the code to see what it does, then draw what you see on the graph paper. By default, the Turtle pen starts out pointing to the right.

```
import turtle
p = turtle.Pen()
p.backward(-100)
turtle.done()
```

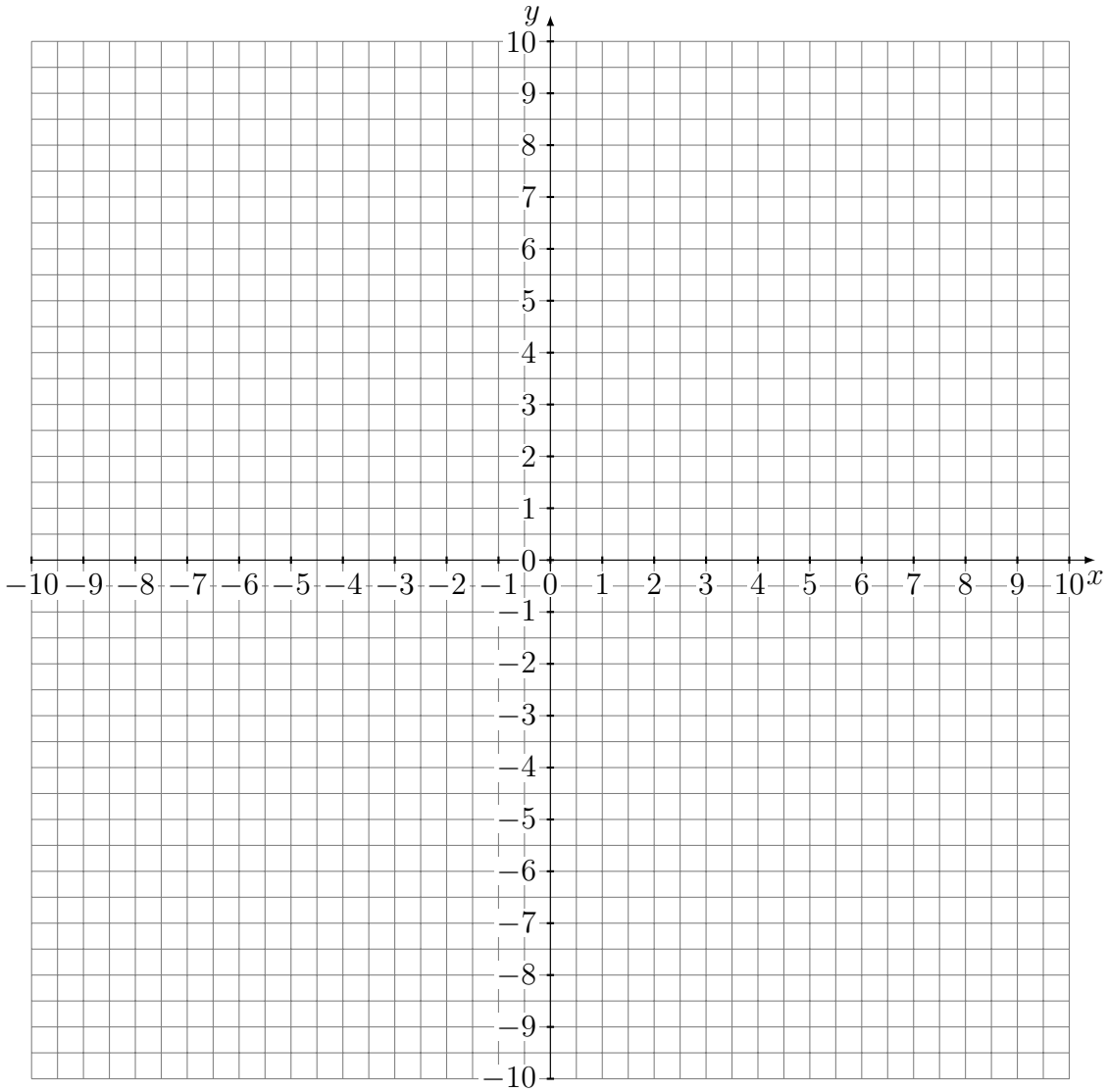


1 unit = 10 pixels

Python Turtle - drawing.2.0.py

Try to draw the output of the Python code in the graphing area below. Use a protractor, straightedge, and/or compass as appropriate. Run the code on your computer and compare the results. If you aren't sure what the code will draw, you can just run the code to see what it does, then draw what you see on the graph paper. By default, the Turtle pen starts out pointing to the right.

```
import turtle
mysteryshape = turtle.Pen()
mysteryshape.forward(100)
mysteryshape.left(90)
mysteryshape.forward(100)
mysteryshape.left(90)
mysteryshape.left(45)
mysteryshape.forward(141)
turtle.done()
```

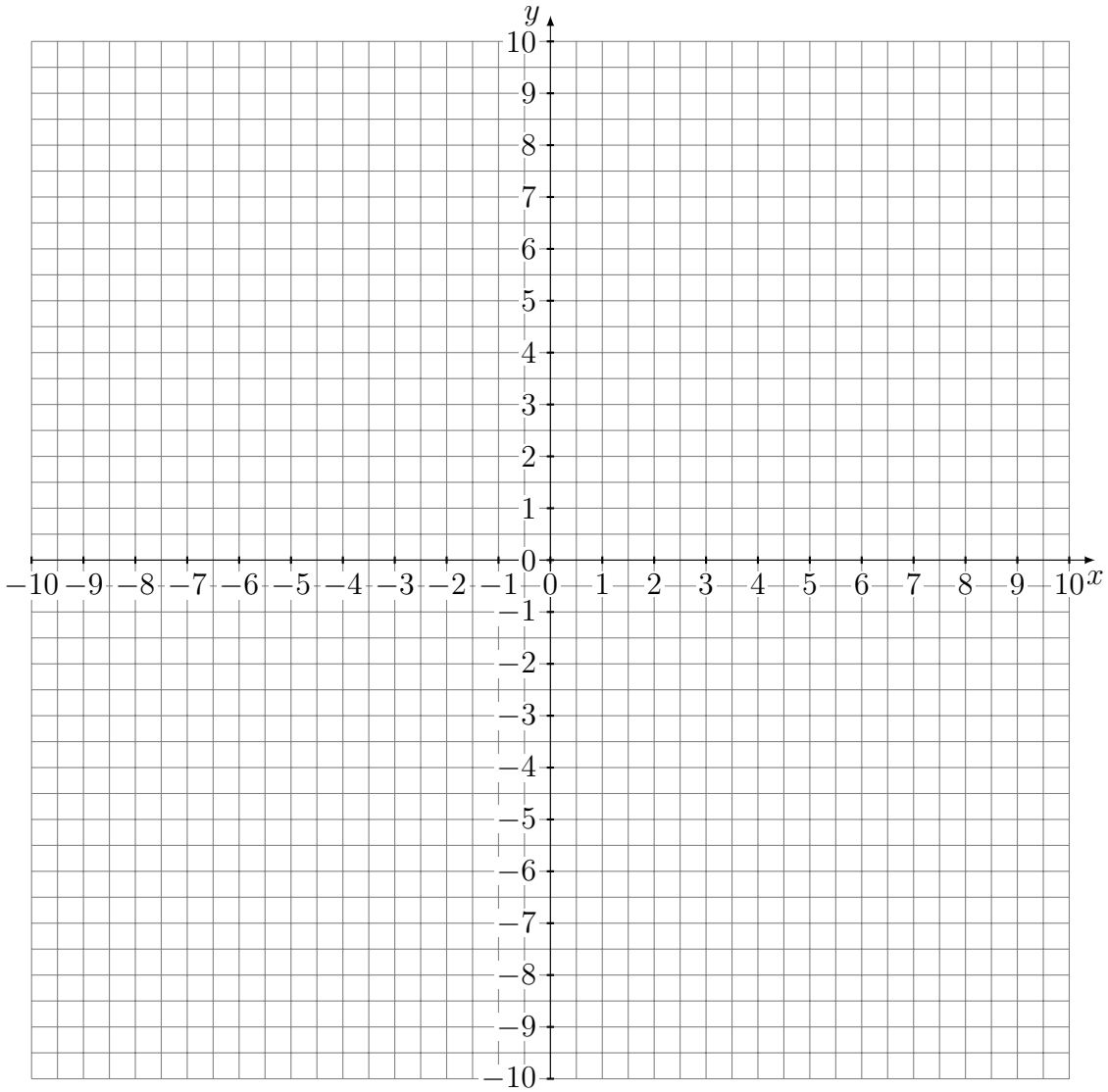


1 unit = 10 pixels

Python Turtle - drawing.2.1.py

Try to draw the output of the Python code in the graphing area below. Use a protractor, straightedge, and/or compass as appropriate. Run the code on your computer and compare the results. If you aren't sure what the code will draw, you can just run the code to see what it does, then draw what you see on the graph paper. By default, the Turtle pen starts out pointing to the right.

```
import turtle
m = turtle.Pen()
m.backward(100)
m.left(90)
m.forward(100)
m.left(90)
m.left(45)
m.forward(141)
turtle.done()
```

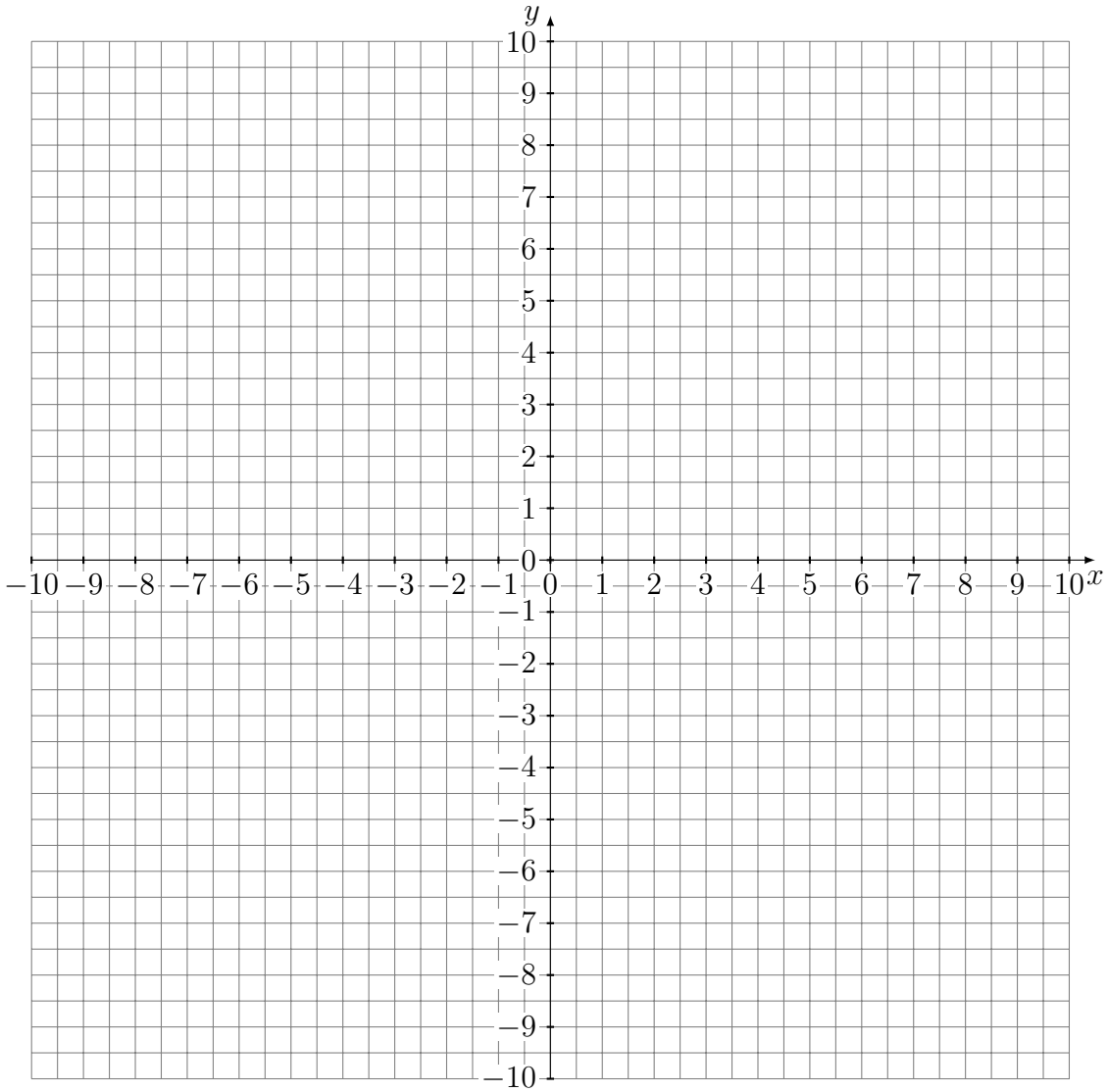


1 unit = 10 pixels

Python Turtle - drawing.2.2.py

Try to draw the output of the Python code in the graphing area below. Use a protractor, straightedge, and/or compass as appropriate. Run the code on your computer and compare the results. If you aren't sure what the code will draw, you can just run the code to see what it does, then draw what you see on the graph paper. By default, the Turtle pen starts out pointing to the right.

```
import turtle
s = turtle.Pen()
s.forward(100)
s.right(90)
s.forward(100)
s.right(90)
s.right(45)
s.forward(141)
turtle.done()
```

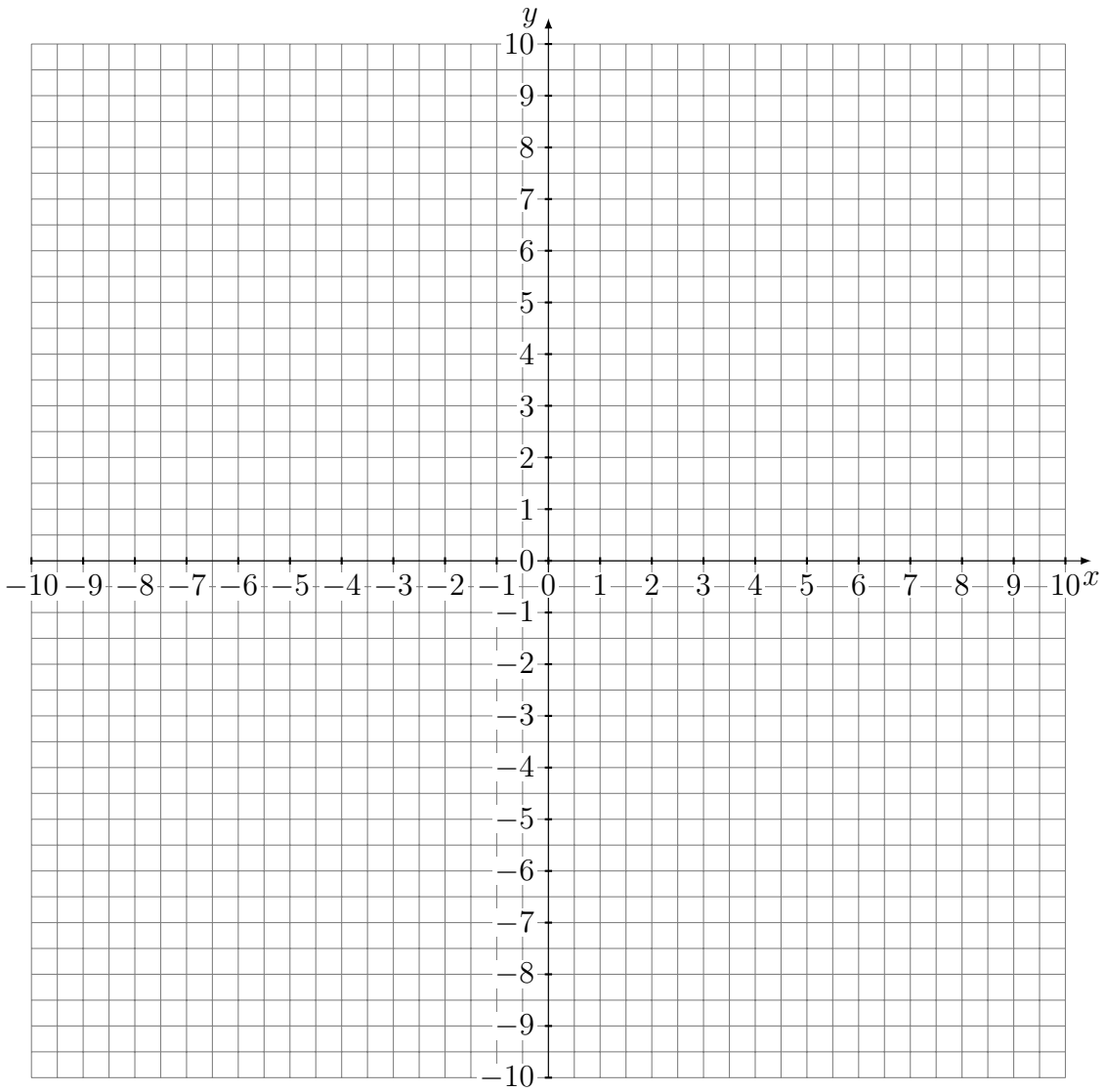


1 unit = 10 pixels

Python Turtle - drawing.2.3.py

Try to draw the output of the Python code in the graphing area below. Use a protractor, straightedge, and/or compass as appropriate. Run the code on your computer and compare the results. If you aren't sure what the code will draw, you can just run the code to see what it does, then draw what you see on the graph paper. By default, the Turtle pen starts out pointing to the right.

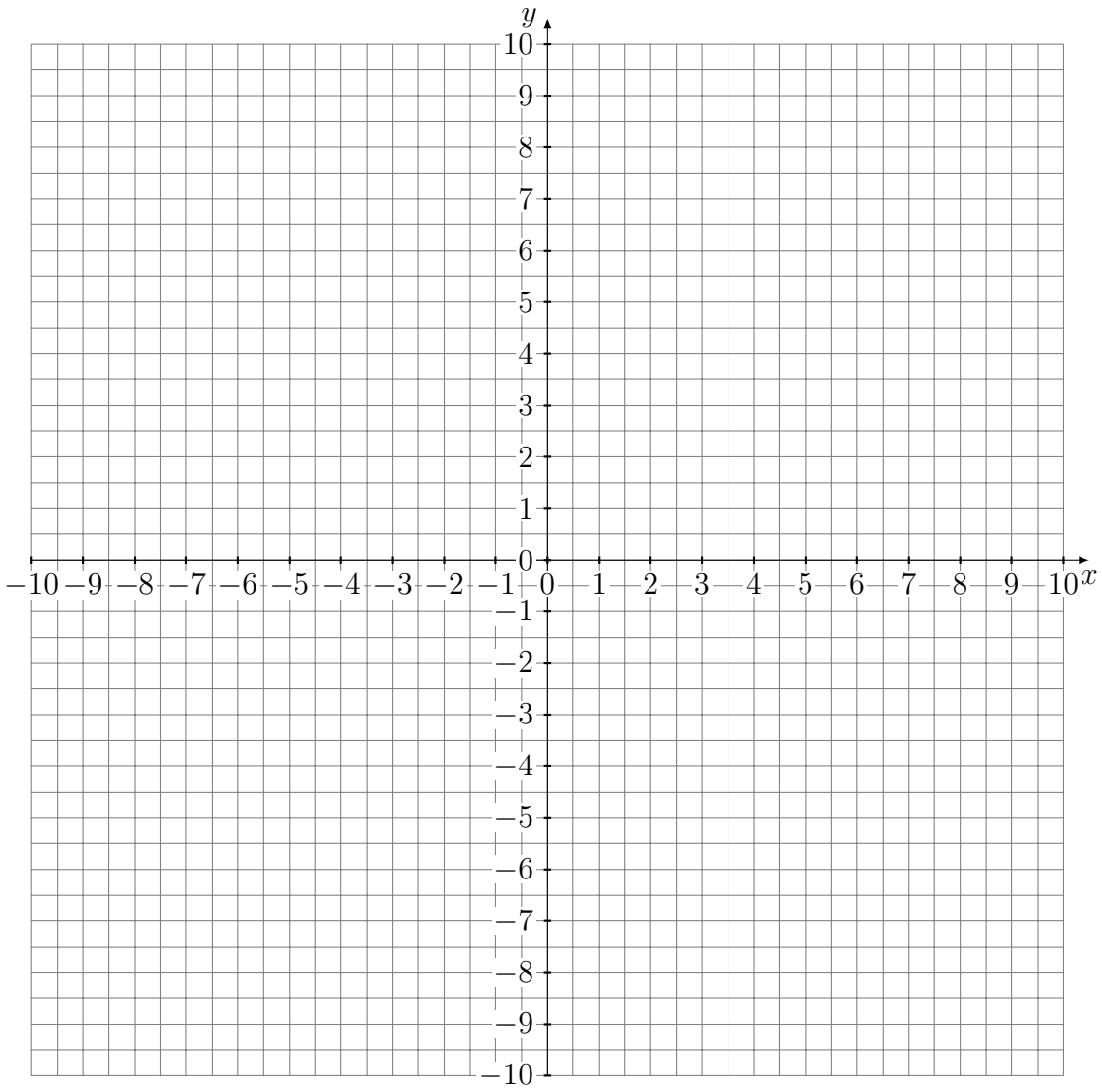
```
import turtle
q = turtle.Pen()
q.left(45)
q.forward(141.2)
q.left(135)
q.forward(200)
q.left(135)
q.forward(2*141.2)
q.right(135)
q.forward(200)
q.right(135)
q.forward(141.2)
turtle.done()
```



Python Turtle - drawing.3.0.py

Try to draw the output of the Python code in the graphing area below. Use a protractor, straightedge, and/or compass as appropriate. Run the code on your computer and compare the results. If you aren't sure what the code will draw, you can just run the code to see what it does, then draw what you see on the graph paper. By default, the Turtle pen starts out pointing to the right.

```
import turtle
reallycoolshape = turtle.Pen()
reallycoolshape.forward(100)
reallycoolshape.left(90)
reallycoolshape.forward(100)
reallycoolshape.left(90)
reallycoolshape.forward(100)
reallycoolshape.left(90)
reallycoolshape.forward(100)
turtle.done()
```

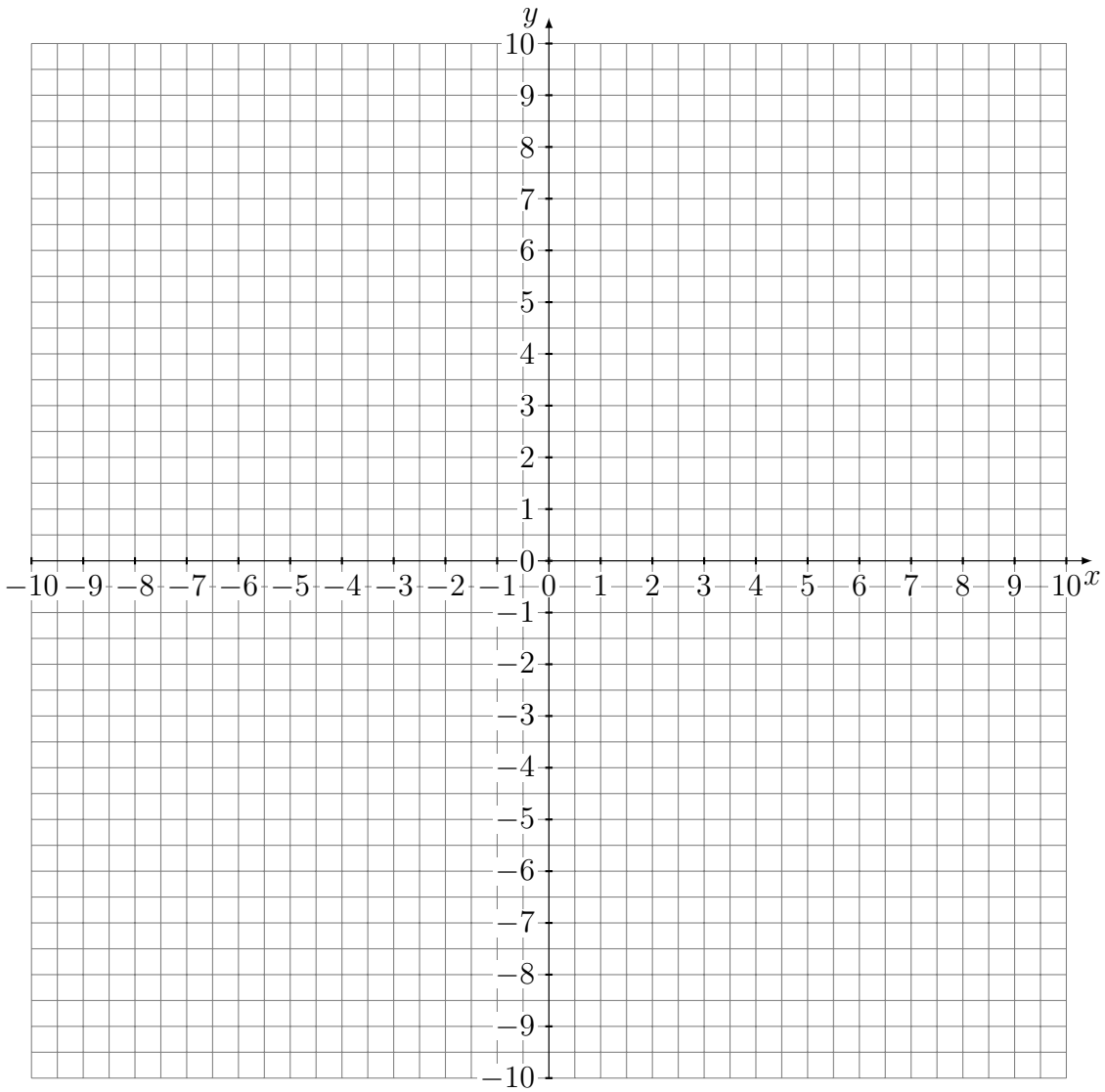


1 unit = 10 pixels

Python Turtle - drawing.3.1.py

Try to draw the output of the Python code in the graphing area below. Use a protractor, straightedge, and/or compass as appropriate. Run the code on your computer and compare the results. If you aren't sure what the code will draw, you can just run the code to see what it does, then draw what you see on the graph paper. By default, the Turtle pen starts out pointing to the right.

```
import turtle
t = turtle.Pen()
t.left(90)
t.forward(50)
t.left(90)
t.forward(50)
t.left(90)
t.forward(50)
t.left(90)
t.forward(50)
turtle.done()
```

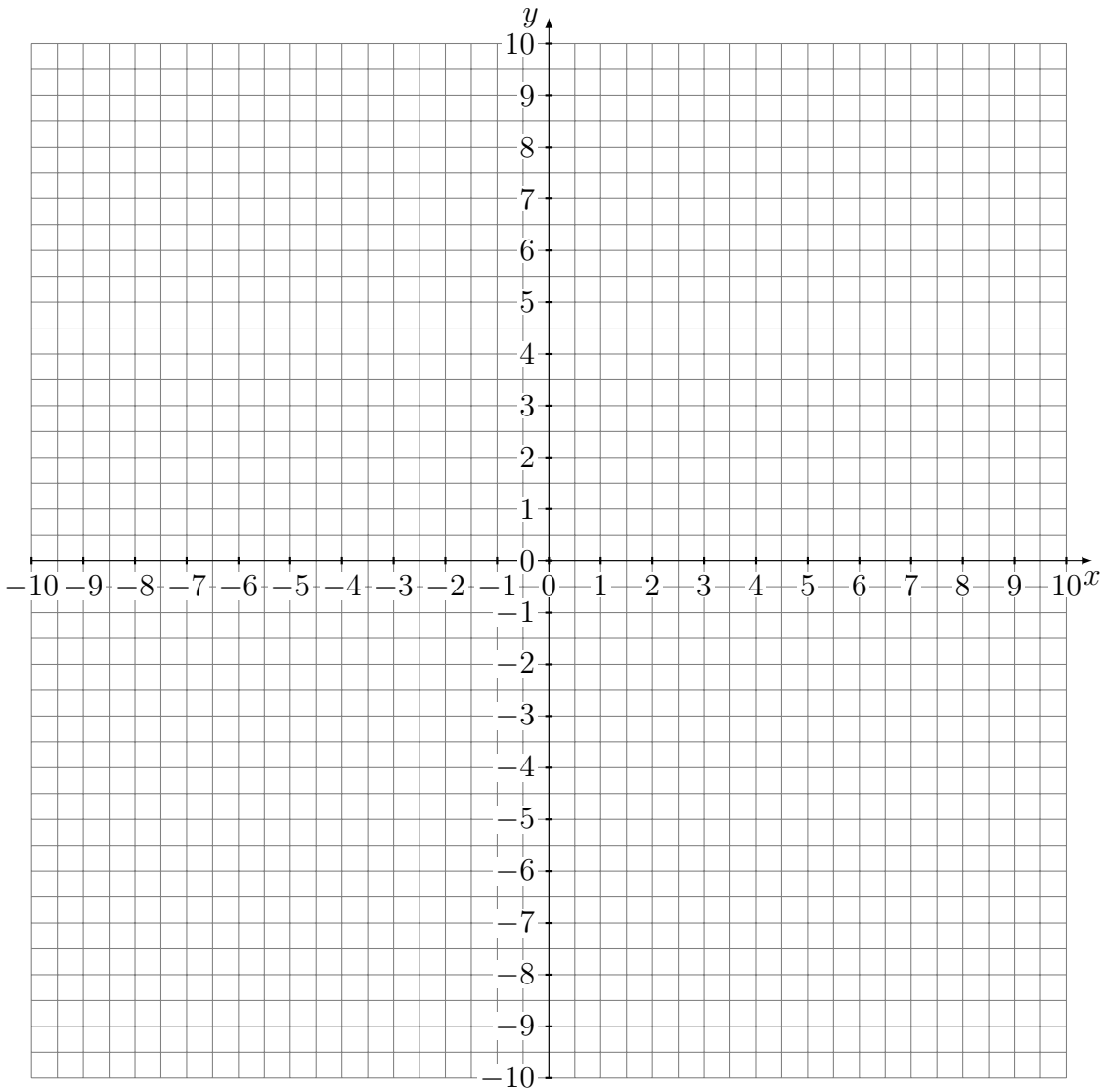


1 unit = 10 pixels

Python Turtle - drawing.3.2.py

Try to draw the output of the Python code in the graphing area below. Use a protractor, straight-edge, and/or compass as appropriate. Run the code on your computer and compare the results. If you aren't sure what the code will draw, you can just run the code to see what it does, then draw what you see on the graph paper. By default, the Turtle pen starts out pointing to the right.

```
import turtle
t = turtle.Pen()
t.left(45+45)
t.forward(2*50)
t.left(2*45)
t.forward(2*50)
t.left(180/2)
t.forward(2*50)
t.left(70+20)
t.forward(2*50)
turtle.done()
```



1 unit = 10 pixels

Write the output of the following Python code. If you aren't sure what the code will do, you can just run the code to see what it does and then write its output here.

```
for x in range(0,4):
    print(x)
```

Output:

```
for number in range(0,5):
    print(number)
```

Output:

```
for digit in range(4):
    print(digit)
```

Output:

```
for william in range(5):
    print(william)
```

Output:

What number does "range" start counting on in Python?

What is the highest number in range(5)?

Python Turtle - drawing.3.3.1.py

Write the output of the following Python code. If you aren't sure what the code will do, you can just run the code to see what it does and then write its output here.

```
for x in range(0,4):
    print(x+5)
```

Output:

```
for number in range(0,5):
    print(number - 3)
```

Output:

```
for digit in range(4):
    print(digit + 5)
```

Output:

```
for william in range(5):
    print(william - 3)
```

Output:

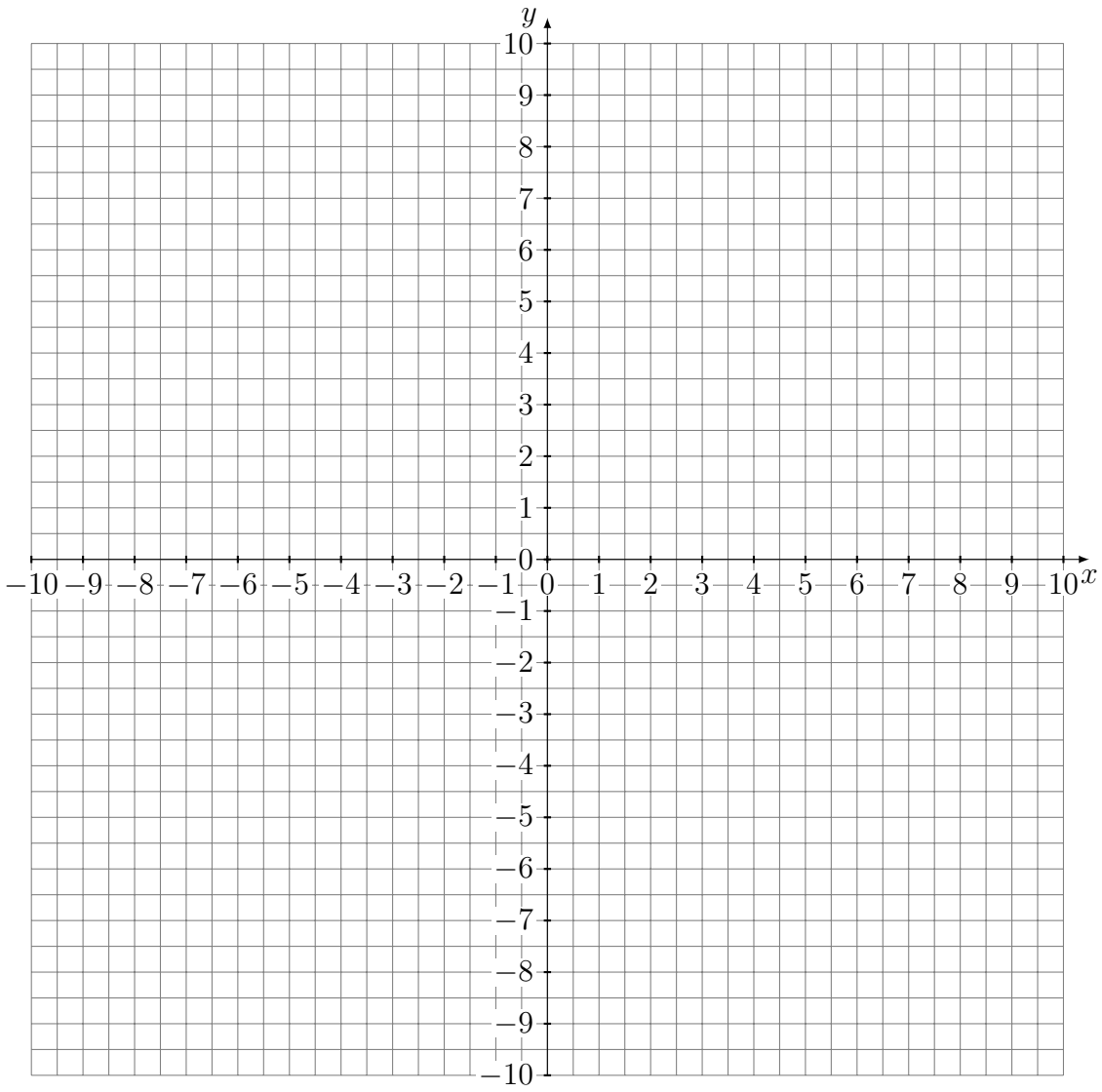
Does Python let you do arithmetic (addition, subtraction, etc.) inside the "print()" function?

In the last code snippet above, "william" is a variable. In Python, almost any word or string of characters can be used as a variable name. What are some words that you can't use as variable names because they are "reserved"? Hint: Some of these words can be seen in the code above.

Python Turtle - drawing.3.3.py

Try to draw the output of the Python code in the graphing area below. Use a protractor, straightedge, and/or compass as appropriate. Run the code on your computer and compare the results. If you aren't sure what the code will draw, you can just run the code to see what it does, then draw what you see on the graph paper. By default, the Turtle pen starts out pointing to the right.

```
import turtle
t = turtle.Pen()
for x in range(4):
     t.forward(100)
     t.left(90)
turtle.done()
```



1 unit = 10 pixels

Write the output of the following Python code. If you aren't sure what the code will do, you can just run the code to see what it does and then write its output here.

```
for x in range(4):
    print(x)
```

Output:

```
for x in range(10):
    print(x)
```

Output:

```
for x in range(5):
    print(2*x)
```

Output:

```
for x in range(5):
    print("x =", x)
```

Output:

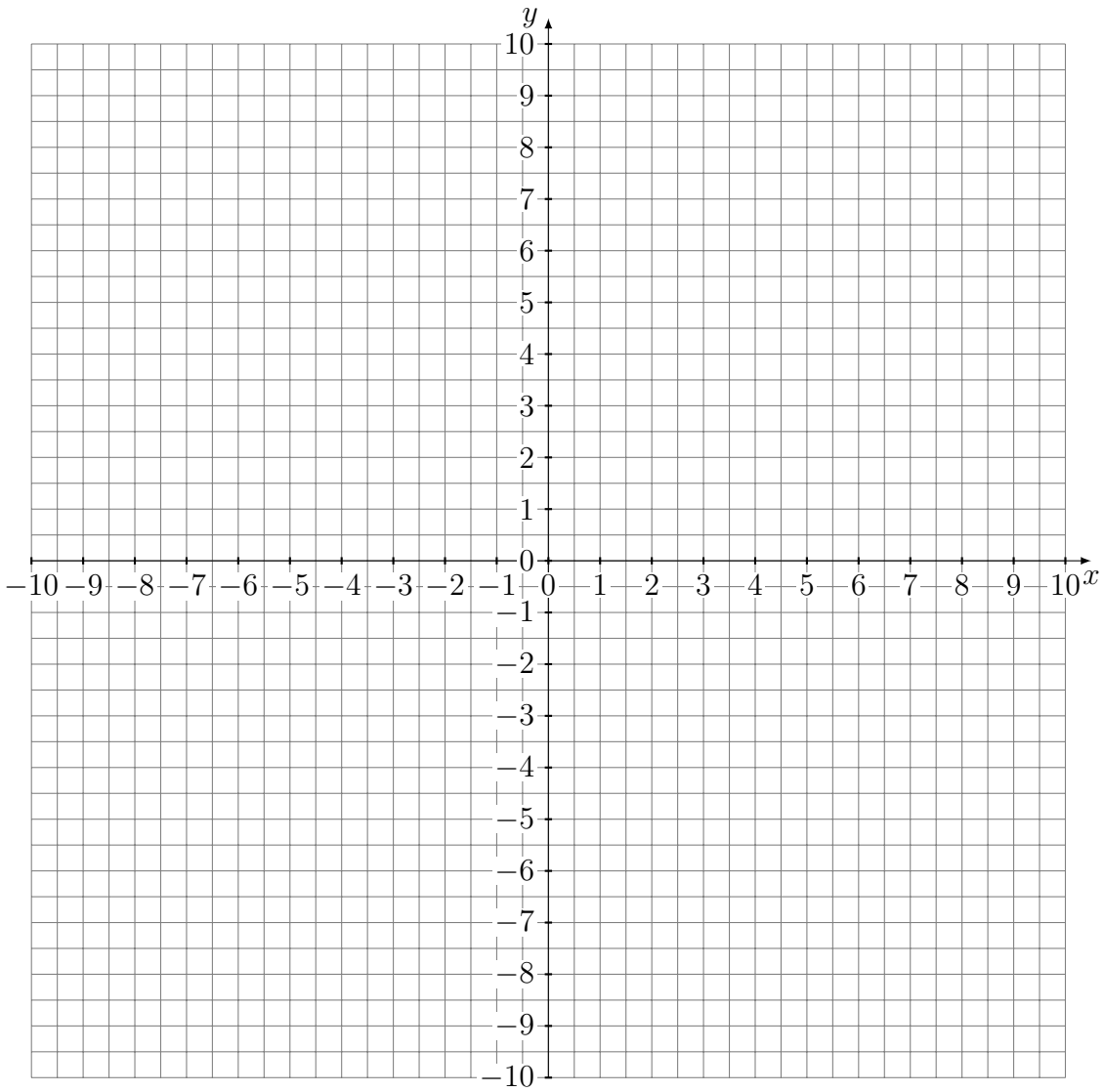
What number does "range" finish counting on in Python?

What is the highest number in range(10)?

Python Turtle - drawing.3.5.py

Try to draw the output of the Python code in the graphing area below. Use a protractor, colored pencils, straightedge, and/or compass as appropriate. Run the code on your computer and compare the results. If you aren't sure what the code will draw, you can just run the code to see what it does, then draw what you see on the graph paper. By default, the Turtle pen starts out pointing to the right.

```
import turtle
t = turtle.Pen()
t.pencolor("blue")
for x in range(8):
     t.forward(100)
     t.left(90)
turtle.done()
```

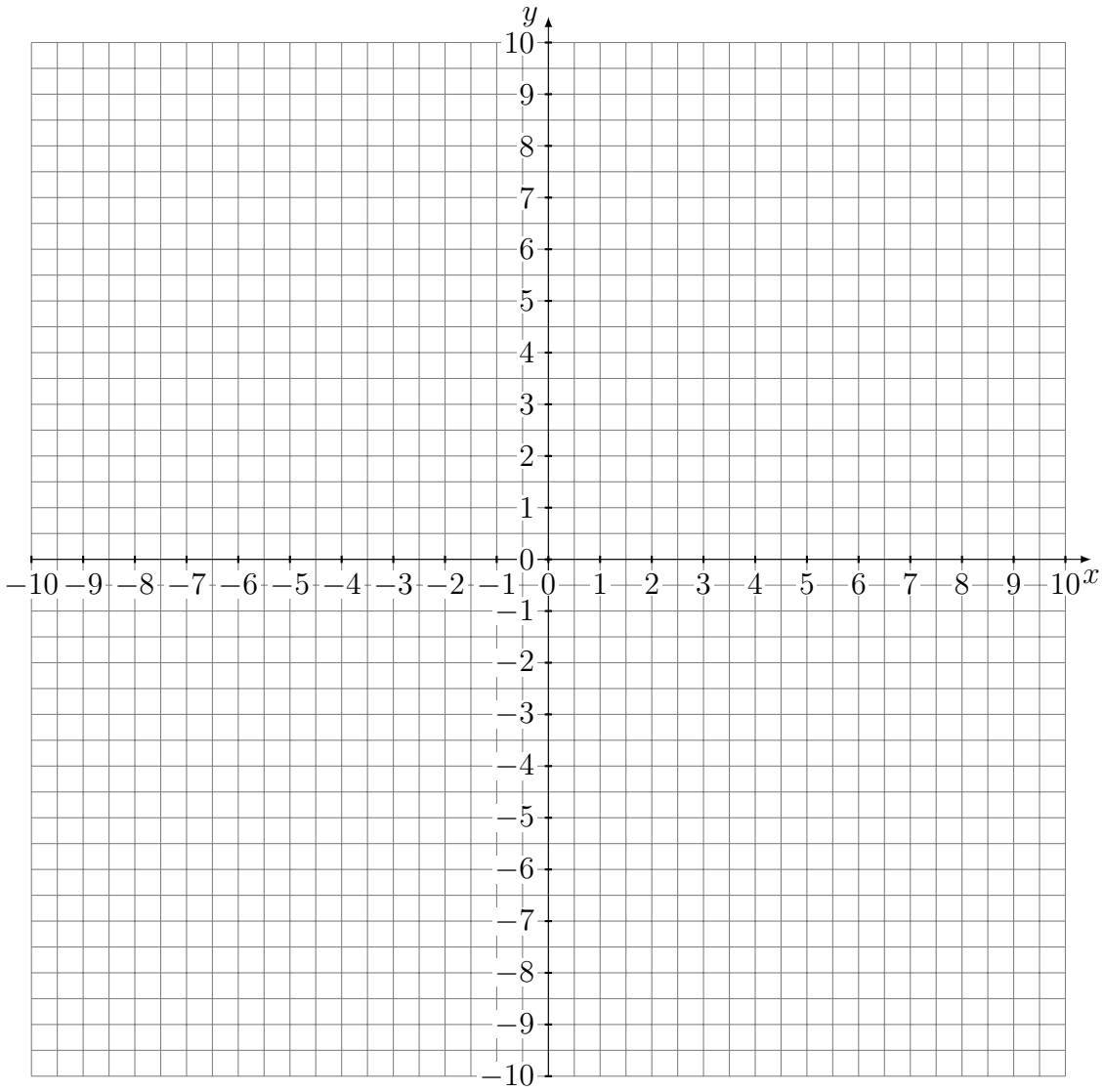


1 unit = 10 pixels

Python Turtle - drawing.3.6.py

Try to draw the output of the Python code in the graphing area below. Use a protractor, colored pencils, straightedge, and/or compass as appropriate. Run the code on your computer and compare the results. If you aren't sure what the code will draw, you can just run the code to see what it does, then draw what you see on the graph paper. By default, the Turtle pen starts out pointing to the right. NOTE: On this graph, 1 unit = 1 pixel. For this problem, it may help you to draw a table of x values!

```
import turtle
t = turtle.Pen()
t.pencolor("red")
for x in range(8):
     t.forward(x)
     t.left(90)
turtle.done()
```

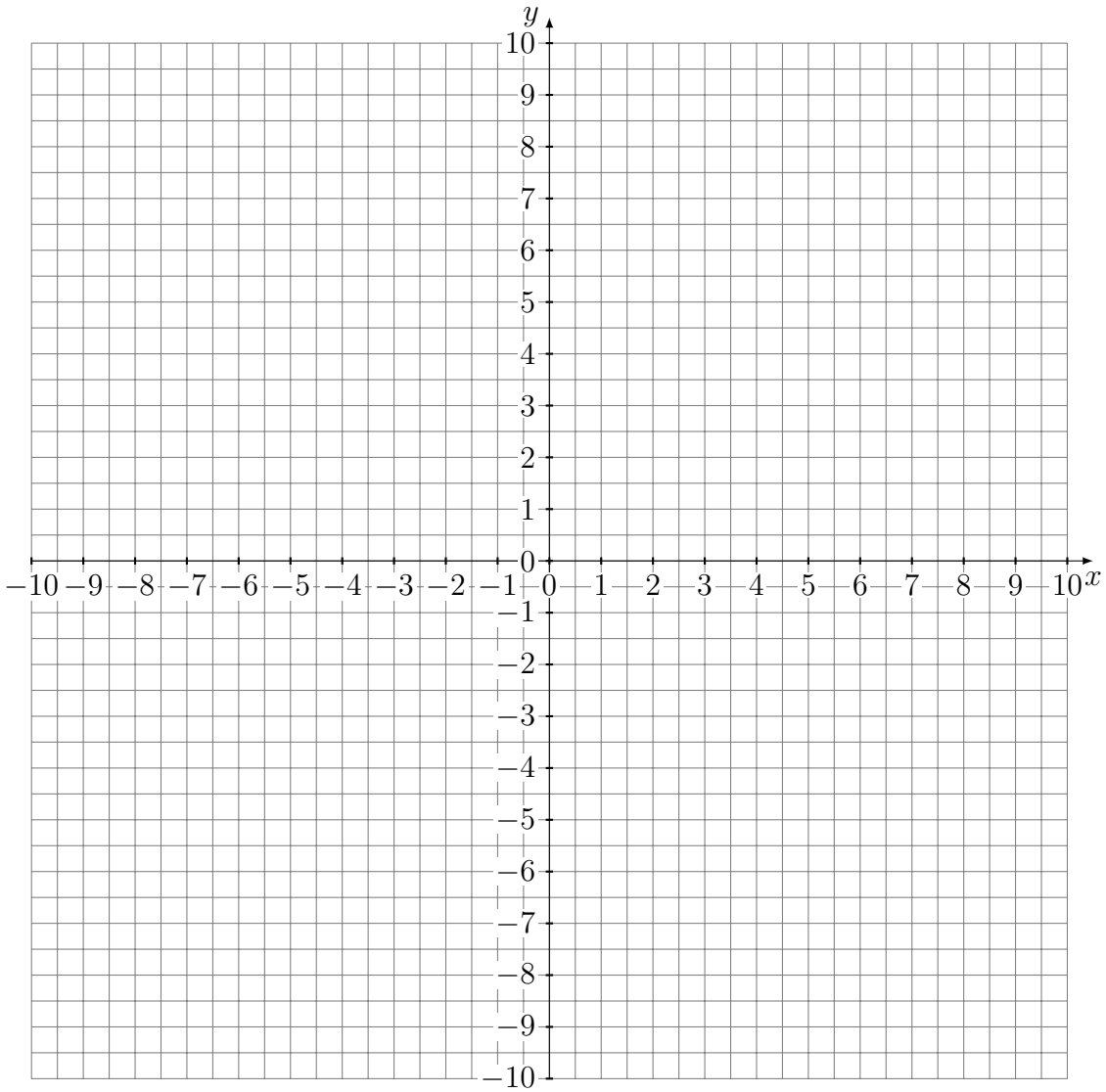


1 unit = 1 pixel

Python Turtle - drawing.3.7.py

Try to draw the output of the Python code in the graphing area below. Use a protractor, colored pencils, straightedge, and/or compass as appropriate. Run the code on your computer and compare the results. If you aren't sure what the code will draw, you can just run the code to see what it does, then draw what you see on the graph paper. By default, the Turtle pen starts out pointing to the right. NOTE: On this graph, 1 unit = 1 pixel. For this problem, it may help you to draw a table of x values!

```
import turtle
t = turtle.Pen()
colors = ["red", "purple", "blue", "green"]
t.speed(1)
for x in range(4):
     t.pencolor(colors[x])
     t.forward(x)
     t.right(90)
turtle.done()
```



1 unit = 1 pixel

Write the output of the following Python code or answer the question, as appropriate. If you aren't sure what the code will do, you can just run the code to see what it does and then write its output here.

```
colors = ["blue", "green", "red", "yellow", "brown"]
print(colors[0])
```

Output:

```
colors = ["blue", "green", "red", "yellow", "brown"]
print(colors[1])
```

Output:

```
colors = ["blue", "green", "red", "yellow", "brown"]
print(colors)
```

Output:

```
colors = ["blue", "green", "red", "yellow", "brown"]
for x in range(5):
    print(colors[x])
```

Output:

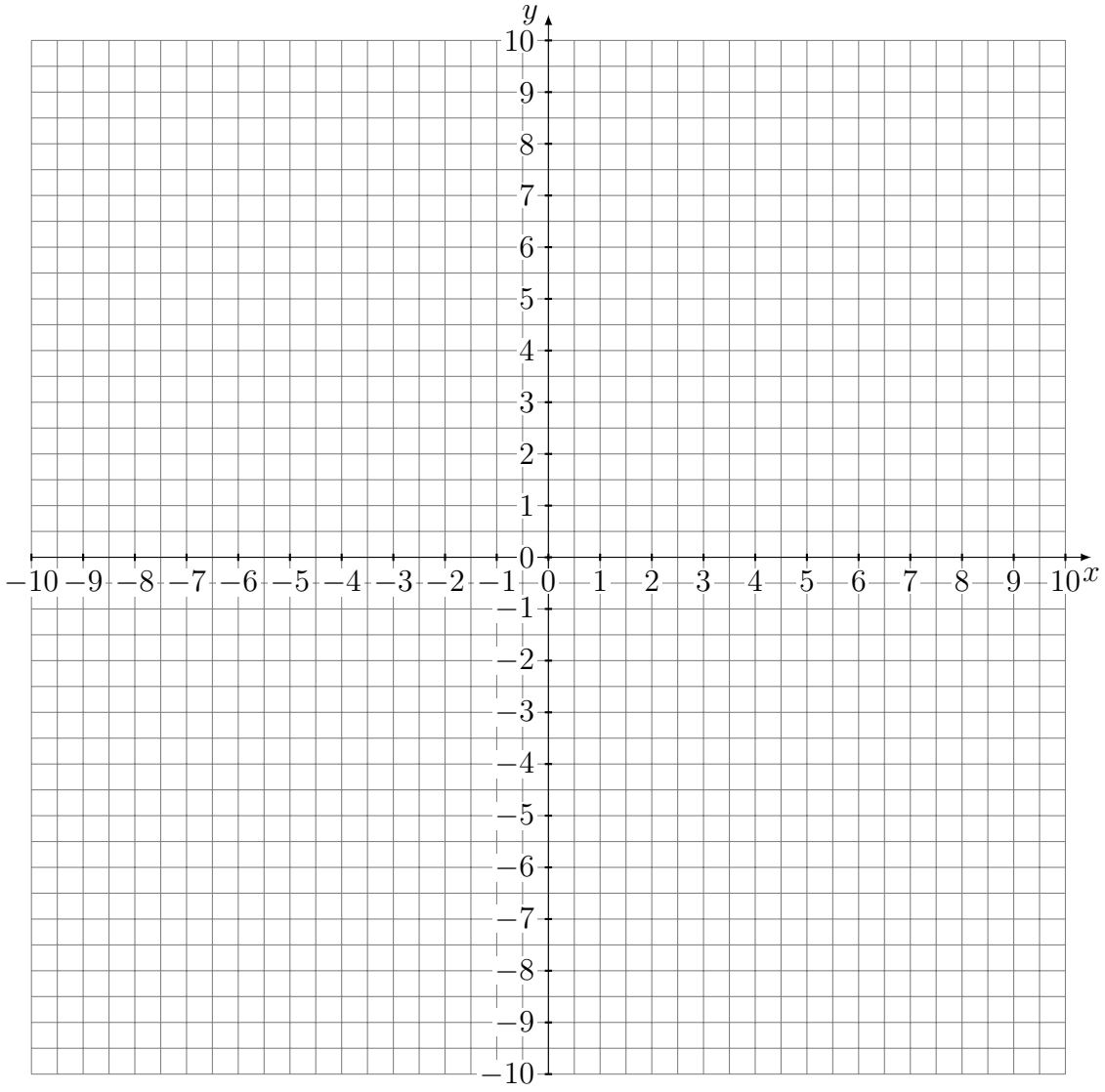
How many elements does the colors list have?

What is the remainder when 9 is divided by 4?

Python Turtle - drawing.3.8.1.py

Try to draw the output of the Python code in the graphing area below. Use a protractor, colored pencils, straightedge, and/or compass as appropriate. Run the code on your computer and compare the results. If you aren't sure what the code will draw, you can just run the code to see what it does, then draw what you see on the graph paper. By default, the Turtle pen starts out pointing to the right. NOTE: On this graph, 1 unit = 10 pixels. For this problem, it may help you to draw a table of x values!

```
import turtle
t = turtle.Pen()
colors = ["green", "brown", "purple", "red", "blue"]
for x in range(5):
     t.pencolor(colors[x])
     t.forward(60)
     t.left(72)
turtle.done()
```

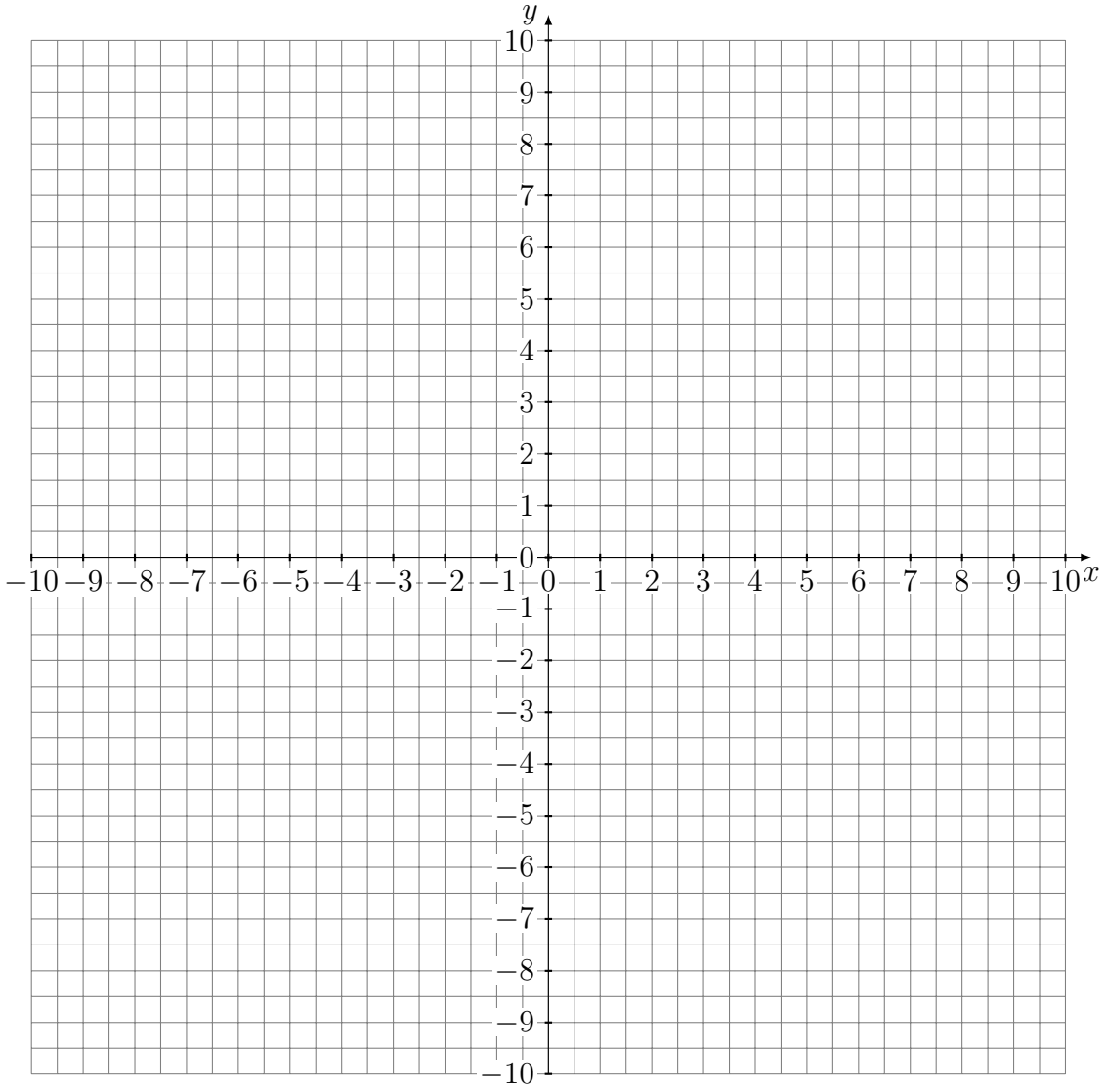


1 unit = 10 pixels

Python Turtle - drawing.3.8.2.py

Try to draw the output of the Python code in the graphing area below. Use a protractor, colored pencils, straightedge, and/or compass as appropriate. Run the code on your computer and compare the results. If you aren't sure what the code will draw, you can just run the code to see what it does, then draw what you see on the graph paper. By default, the Turtle pen starts out pointing to the right. NOTE: On this graph, 1 unit = 10 pixels. For this problem, it may help you to draw a table of x values!

```
import turtle
t = turtle.Pen()
colors = ["green", "brown", "purple", "red", "blue", "yellow"]
for x in range(6):
     t.pencolor(colors[x])
     t.forward(60)
     t.left(60)
turtle.done()
```

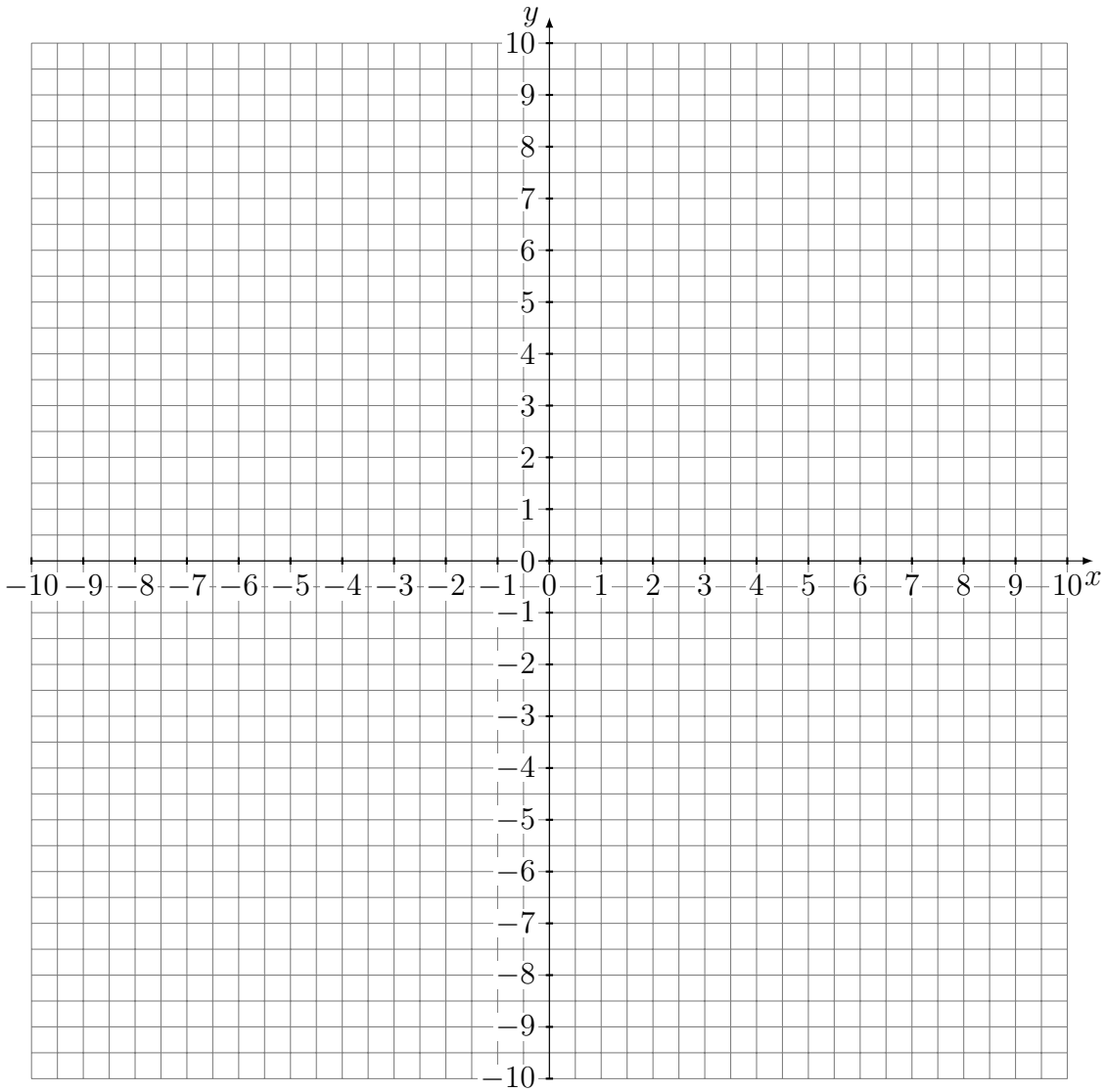


1 unit = 10 pixels

Python Turtle - drawing.3.8.3.py

Try to draw the output of the Python code in the graphing area below. Use a protractor, colored pencils, straightedge, and/or compass as appropriate. Run the code on your computer and compare the results. If you aren't sure what the code will draw, you can just run the code to see what it does, then draw what you see on the graph paper. By default, the Turtle pen starts out pointing to the right. NOTE: On this graph, 1 unit = 10 pixels. For this problem, it may help you to draw a table of x values!

```
import turtle
t = turtle.Pen()
colors = ["green", "brown", "purple", "red", "blue", "yellow", "orange red", "sky
    blue"]
for x in range(8):
     t.pencolor(colors[x])
     t.forward(60)
     t.right(45)
turtle.done()
```



1 unit = 10 pixels

Write the output of the following Python code or answer the question, as appropriate. If you aren't sure what the code will do, you can just run the code to see what it does and then write its output here. If the code will produce an error, then write "error" for the output.

```
colors = ["blue", "green", "red", "yellow", "brown"]
print(colors[2])
```

Output:

```
colors = ["blue", "green", "red", "yellow", "brown"]
print(colors[4])
```

Output:

```
colors = ["blue", "green", "red", "yellow", "brown"]
print(colors[5])
```

Output:

```
print(8/4)
```

Output:

```
print(20/5)
```

Output:

What is the remainder when 10 is divided by 4?

Write the output of the following Python code or answer the question, as appropriate. If you aren't sure what the code will do, you can just run the code to see what it does and then write its output here. If the code will produce an error, then write "error" for the output.

```
print(4/4)
```

Output:

```
print(10%4)
```

Output:

```
print(5%4)
```

Output:

```
print(6%4)
```

Output:

```
print(7%4)
```

Output:

What is the remainder when 8 is divided by 4?

Write the output of the following Python code or answer the question, as appropriate. If you aren't sure what the code will do, you can just run the code to see what it does and then write its output here. If the code will produce an error, then write "error" for the output.

```
print(0%4)
```

Output:

```
print(11%5)
```

Output:

```
print(16%3)
```

Output:

```
print(4%3)
```

Output:

```
print(100%10)
```

Output:

What is the remainder when 200 is divided by 50?

Python Turtle - drawing.3.12.py

Write the output of the following Python code or answer the question, as appropriate. If you aren't sure what the code will do, you can just run the code to see what it does and then write its output here. If the code will produce an error, then write "error" for the output.

```
for y in range(8):
    print(y%4)
```

Output:

```
for z in range(5):
    print(z*2)
```

Output:

```
for s in range(6):
    print(s%4)
```

Output:

```
colors = ["red", "blue", "green", "purple", "yellow"]
for x in range(11):
    print(colors[x%5])
```

Output:

```
colors = ["blue", "red", "purple", "yellow", "black"]
for x in range(11):
    print(colors[x%5])
```

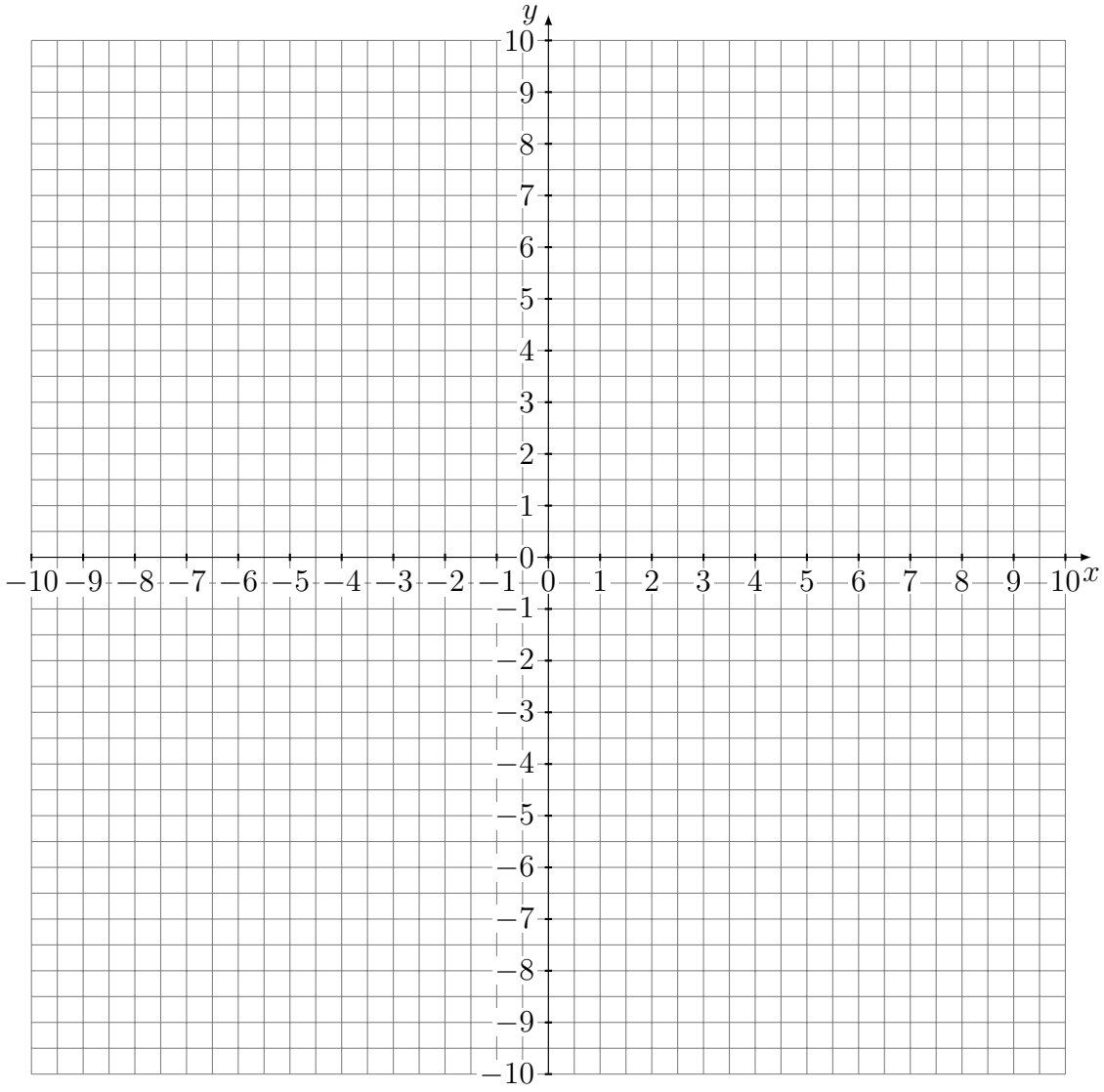
Output:

How do you say "%" when reading "10%5"?

Python Turtle - drawing.4.0.py

Try to draw the output of the Python code in the graphing area below. Use a protractor, colored pencils, straightedge, and/or compass as appropriate. Run the code on your computer and compare the results. If you aren't sure what the code will draw, you can just run the code to see what it does, then draw what you see on the graph paper. By default, the Turtle pen starts out pointing to the right. NOTE: On this graph, 1 unit = 10 pixels. For this problem, it may help you to draw a table of x values!

```
import turtle
t = turtle.Pen()
colors = ["green", "brown", "purple", "red", "blue"]
for x in range(10):
     t.pencolor(colors[x%5])
     t.forward(60)
     t.left(76)
turtle.done()
```

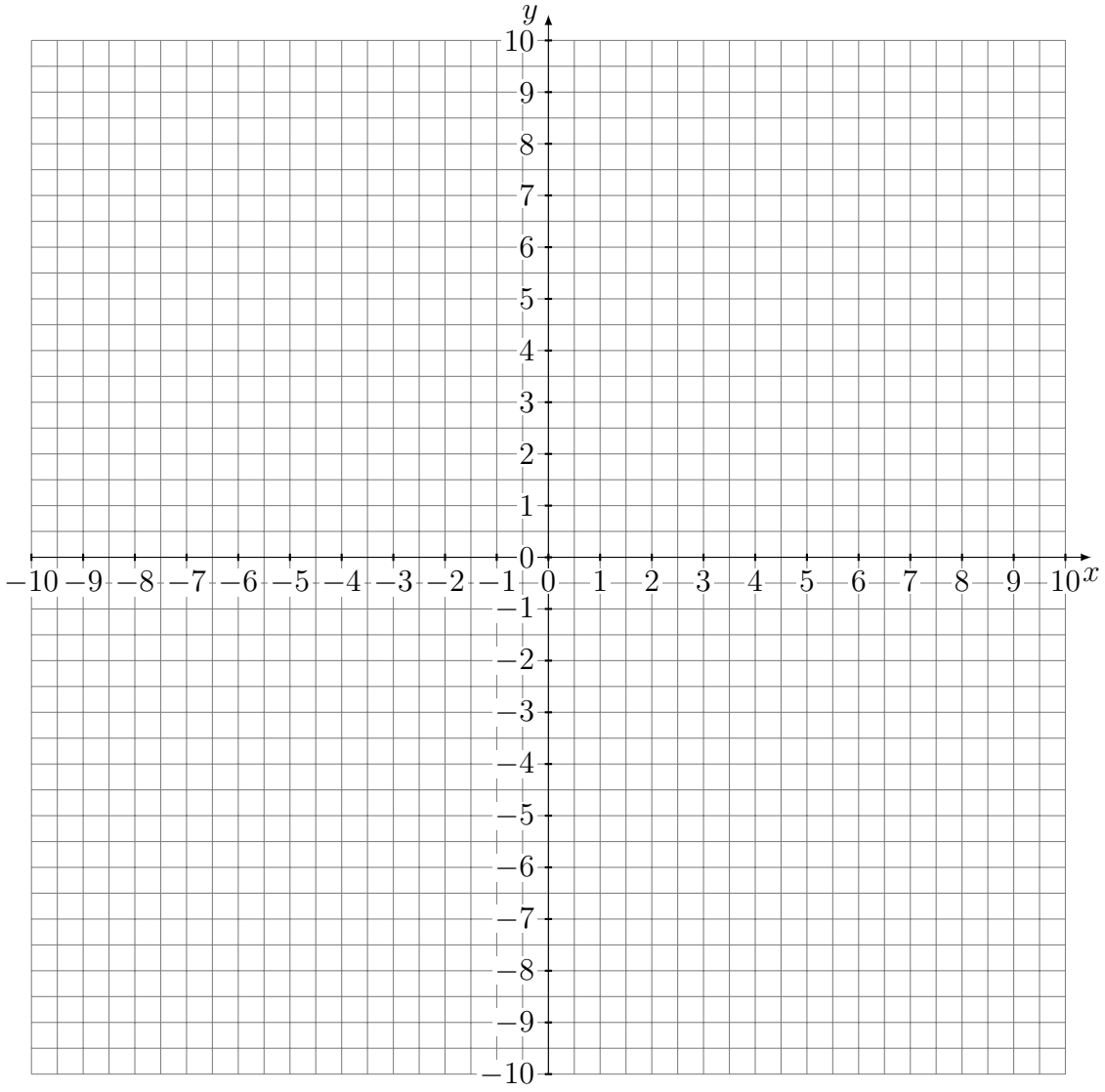


1 unit = 10 pixels

Python Turtle - drawing.4.1.py

Try to draw the output of the Python code in the graphing area below. Use a protractor, colored pencils, straightedge, and/or compass as appropriate. Run the code on your computer and compare the results. If you aren't sure what the code will draw, you can just run the code to see what it does, then draw what you see on the graph paper. By default, the Turtle pen starts out pointing to the right. NOTE: On this graph, 1 unit = 10 pixels. For this problem, it may help you to draw a table of x values!

```
import turtle
t = turtle.Pen()
colors = ["green", "brown", "purple", "red", "blue"]
for x in range(10):
     t.pencolor(colors[x%5])
     t.forward(60)
     t.left(76)
turtle.done()
```

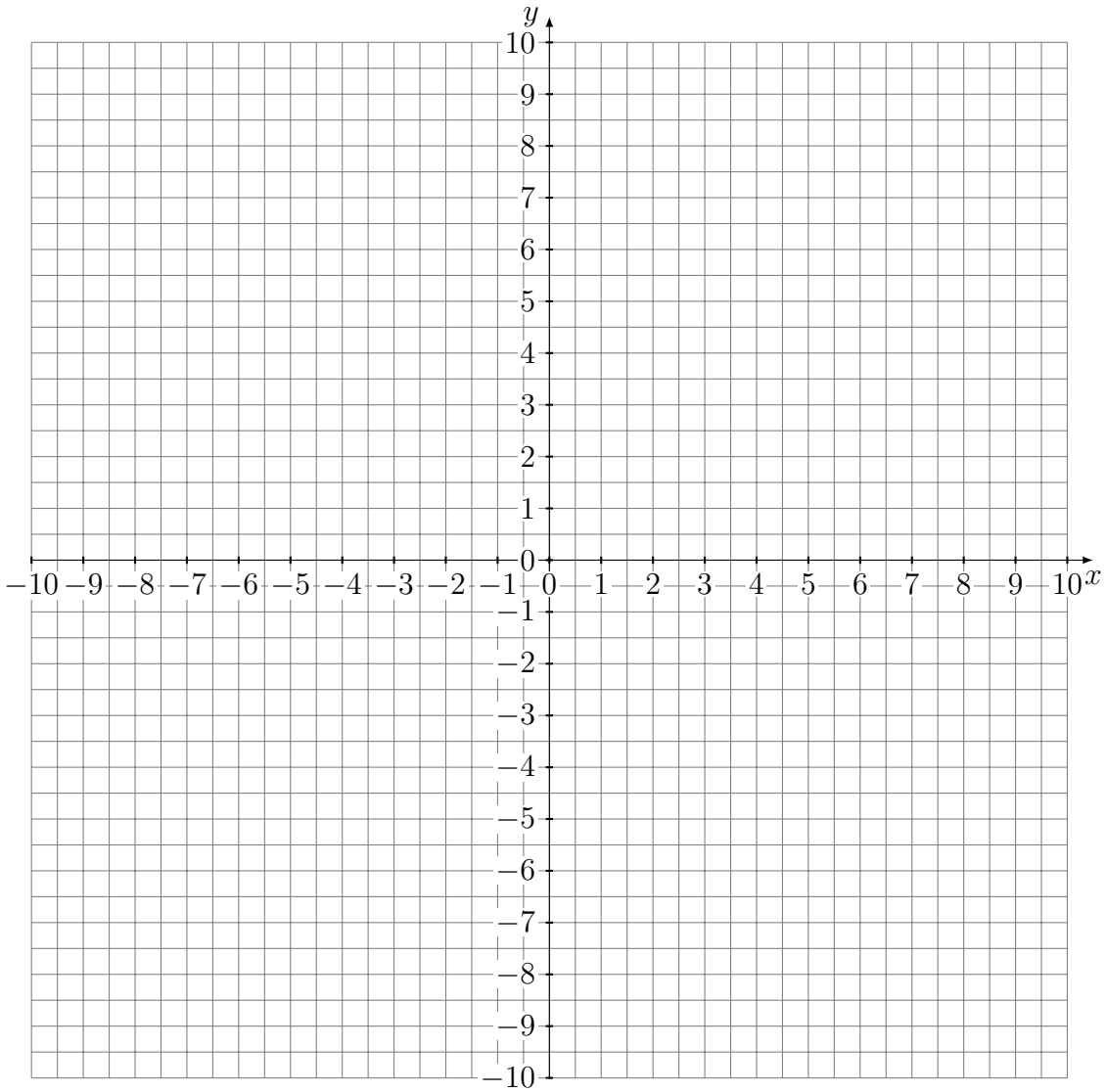


1 unit = 10 pixels

Python Turtle - drawing.4.2.py

Try to draw the output of the Python code in the graphing area below. Use a protractor, colored pencils, straightedge, and/or compass as appropriate. Run the code on your computer and compare the results. If you aren't sure what the code will draw, you can just run the code to see what it does, then draw what you see on the graph paper. By default, the Turtle pen starts out pointing to the right. NOTE: On this graph, 1 unit = 10 pixels. For this problem, it may help you to draw a table of x values!

```
import turtle
# <-- the hash symbol is one way to write comments in Python. Anything on the same
    line is ignored!
t = turtle.Pen()
#colors = ["green", "brown", "purple", "red", "blue"]
for x in range(20):
     #t.pencolor(colors[x%5])
     t.forward(2*x+10)
     t.left(29)
turtle.done()
```

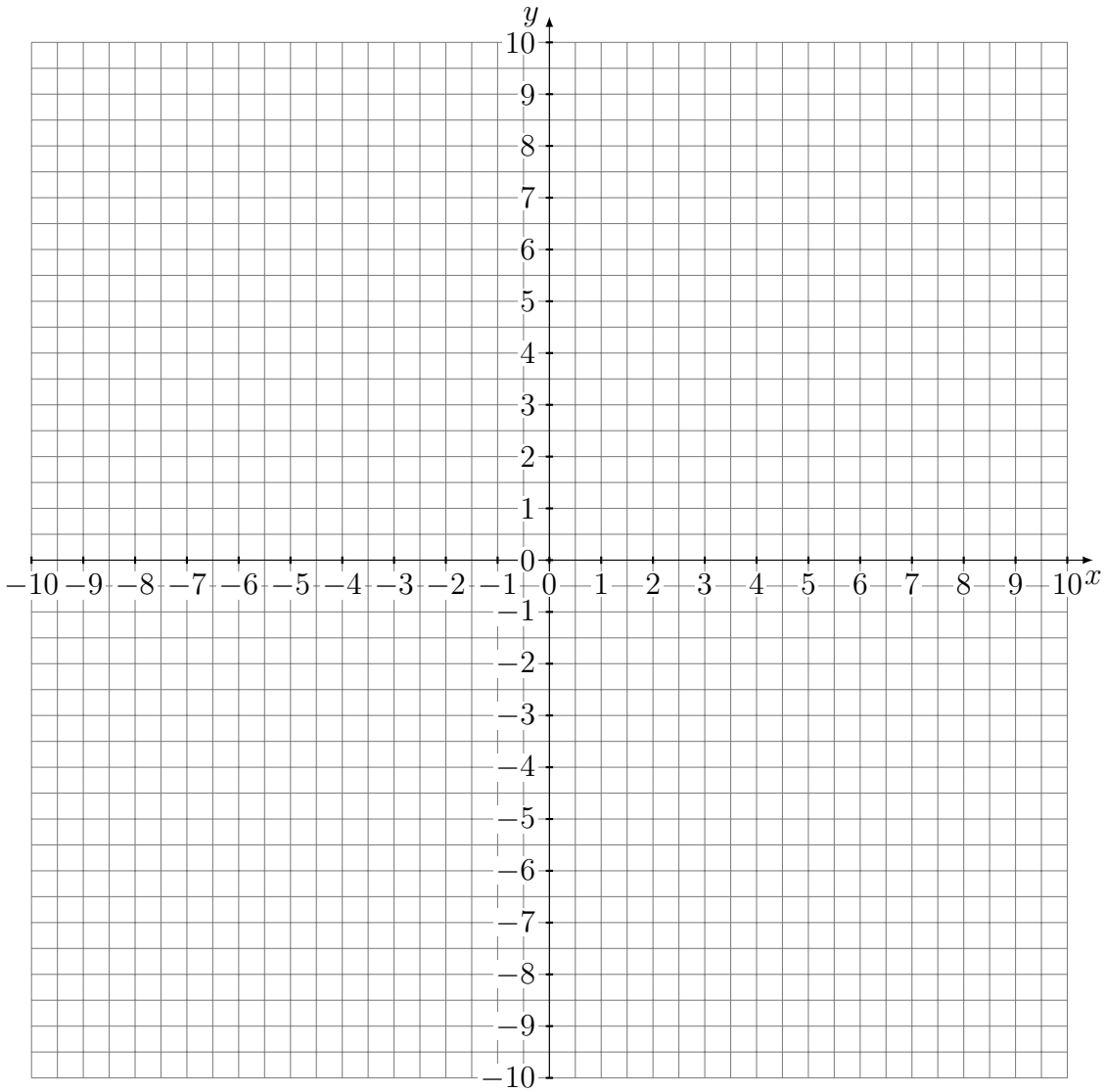


1 unit = 10 pixels

Python Turtle - drawing.4.3.py

Try to draw the output of the Python code in the graphing area below. Use a protractor, colored pencils, straightedge, and/or compass as appropriate. Run the code on your computer and compare the results. If you aren't sure what the code will draw, you can just run the code to see what it does, then draw what you see on the graph paper. By default, the Turtle pen starts out pointing to the right. NOTE: On this graph, 1 unit = 10 pixels. For this problem, it may help you to draw a table of x values!

```
import turtle
# <-- the hash symbol is one way to write comments in Python.
# Anything on the same line as the hash is ignored!
t = turtle.Pen()
colors = ["green", "brown", "purple", "red", "blue"]
for x in range(20):
     t.pencolor(colors[x%5])
     t.forward(2*x+10)
     t.right(29)
turtle.done()
```

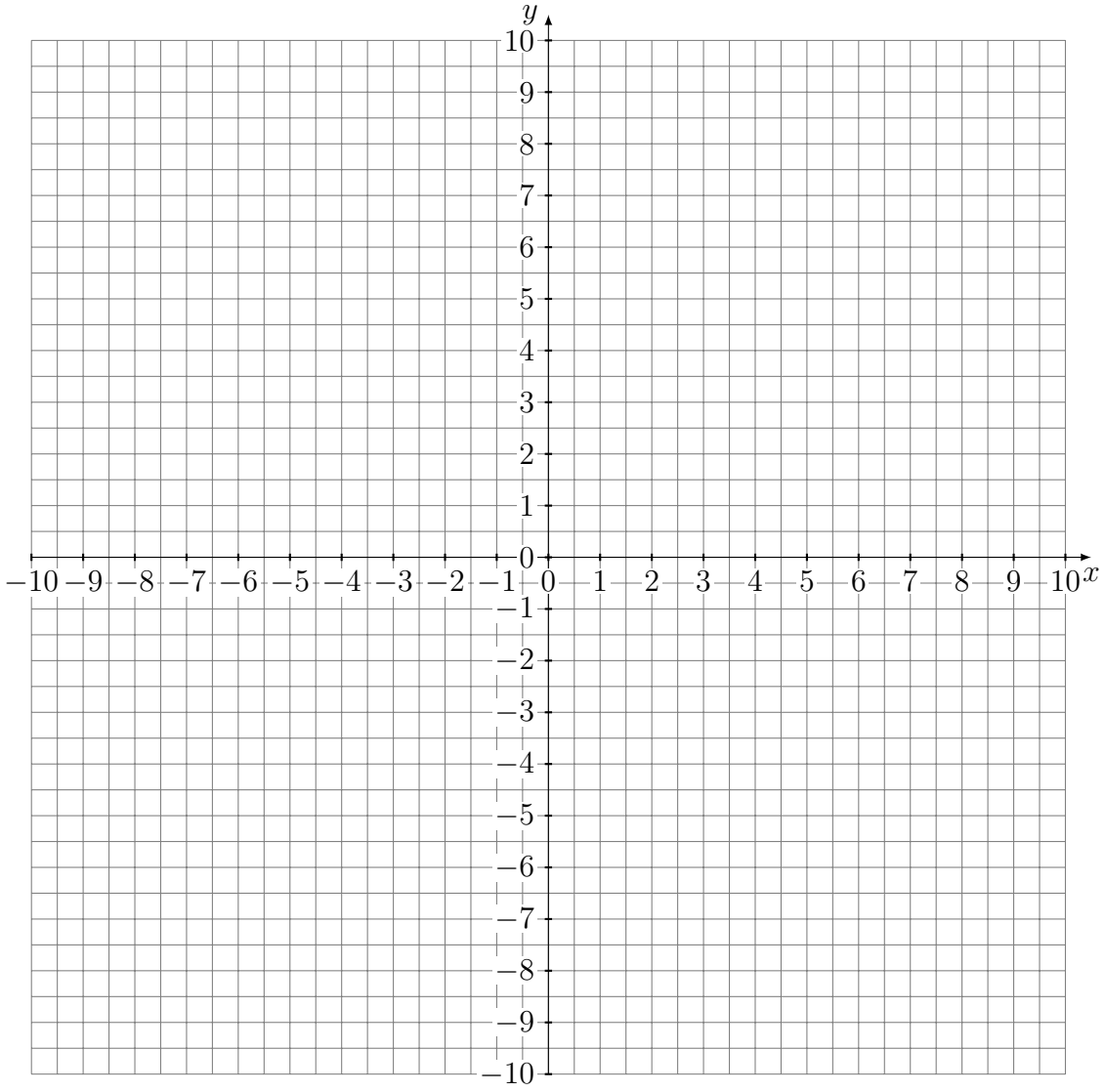


1 unit = 10 pixels

Python Turtle - drawing.4.4.py

Try to draw the output of the Python code in the graphing area below. Use a protractor, colored pencils, straightedge, and/or compass as appropriate. Run the code on your computer and compare the results. If you aren't sure what the code will draw, you can just run the code to see what it does, then draw what you see on the graph paper. By default, the Turtle pen starts out pointing to the right. NOTE: On this graph, 1 unit = 10 pixels. For this problem, it may help you to draw a table of x values!

```
import turtle
t = turtle.Pen()
colors = ["green", "blue", "purple", "red"]
for x in range(20):
     t.pencolor(colors[x%4])
     t.forward(2*x+10)
     t.left(92)
turtle.done()
```



1 unit = 10 pixels

How would you describe what appears when you run the following code?

```
import turtle
t = turtle.Pen()
colors = ["green", "blue", "purple", "red"]
for x in range(100):
     t.pencolor(colors[x%4])
     t.forward(x)
     t.left(91)
turtle.done()
```

Description:

How is this picture different?

```
import turtle
t = turtle.Pen()
colors = ["green", "blue", "purple", "red"]
for x in range(100):
     t.pencolor(colors[x%4])
     t.forward(x)
     t.right(91)
turtle.done()
```

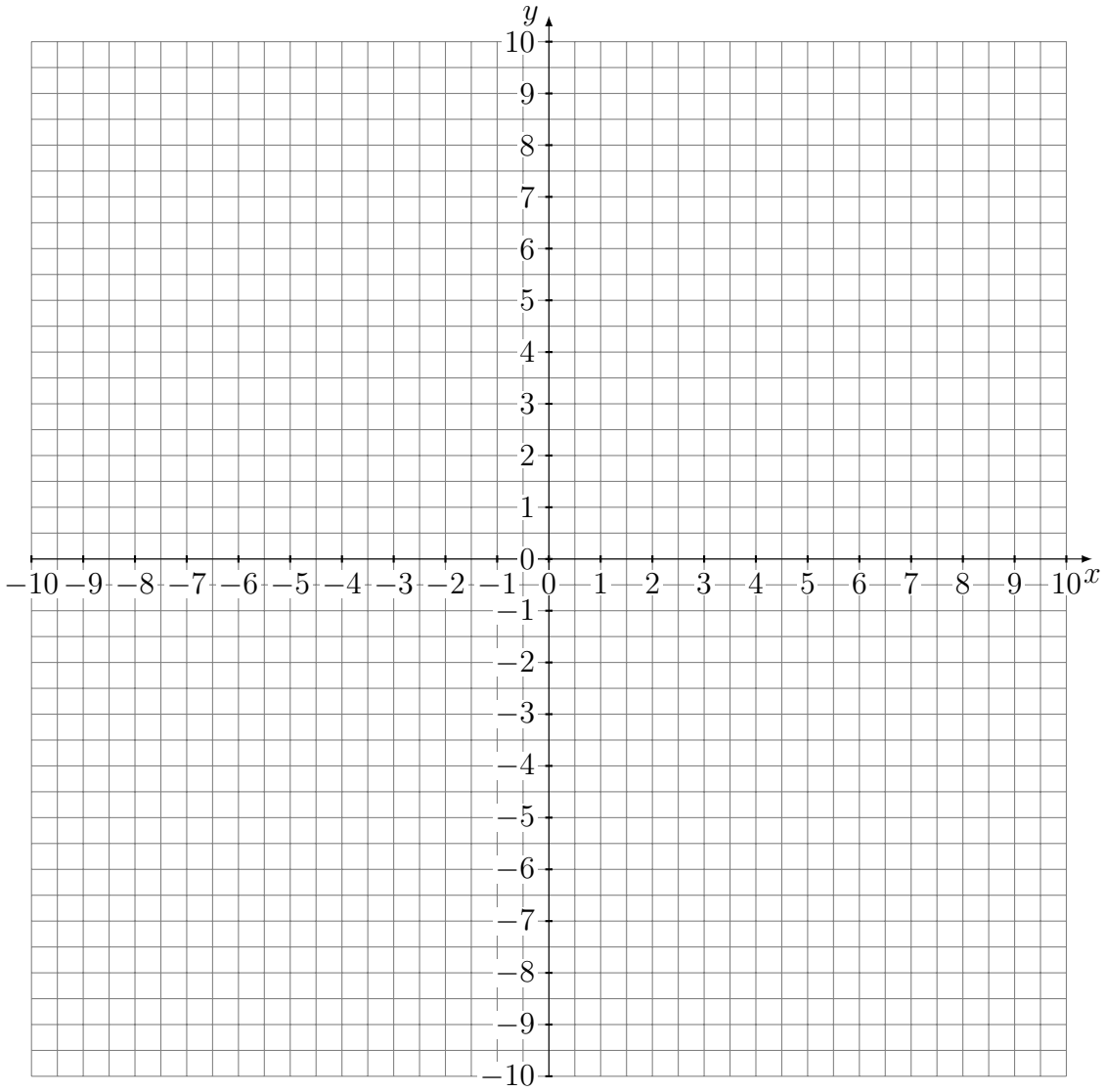
Description:

Create at least four different variations of this picture by changing the colors, the number of steps or iterations in the loop (the number in `range()`), the distance that the pen moves forward (the number in `t.forward()`, or the angle that the pen turns in each step (the number in `t.right()`). Take screenshots of your artistic creations to save and share them!

Python Turtle - drawing.5.0.py

Try to draw the output of the Python code in the graphing area below. Use a protractor, colored pencils, straightedge, and/or compass as appropriate. Run the code on your computer and compare the results. If you aren't sure what the code will draw, you can just run the code to see what it does, then draw what you see on the graph paper. By default, the Turtle pen starts out pointing to the right. NOTE: On this graph, 1 unit = 1 pixel. For this problem, it may help you to draw a table of x values!

```
import turtle
t = turtle.Pen()
for x in range(4):
     t.circle(2*x+10)
     t.left(91)
turtle.done()
```



1 unit = 1 pixel

How would you describe what appears when you run the following code?

```
import turtle
t = turtle.Pen()
t.speed(0)
colors = ["green", "blue", "purple", "red"]
for x in range(100):
     t.pencolor(colors[x%4])
     t.circle(x)
     t.left(91)
turtle.done()
```

Description:

How is this picture different?

```
import turtle
t = turtle.Pen()
t.speed(0)
colors = ["green", "blue", "purple", "red"]
for x in range(400):
     t.pencolor(colors[x%4])
     t.circle(x)
     t.left(91)
turtle.done()
```

Description:

Create at least four different variations of this picture by changing the colors, the number of steps or iterations in the loop (the number in `range()`), the radius of the circles (the expression in `t.circle()`, or the angle that the pen turns in each step (the number in `t.right()`). Take screenshots of your artistic creations to save and share them!

How would you describe what appears when you run the following code?

```
import turtle
t = turtle.Pen()
t.speed(0)
colors = ["green", "blue", "purple", "red"]
for x in range(200):
     t.pencolor(colors[x%4])
     t.forward(x)
     t.left(91)
turtle.done()
```

Description:

How is this picture different?

```
import turtle
t = turtle.Pen()
t.speed(0)
colors = ["green", "blue", "purple", "red"]
for x in range(200):
     t.pencolor(colors[x%4])
     t.forward(x)
     t.right(91)
turtle.done()
```

Description:

Create at least four different variations of this picture by changing the colors, the number of steps or iterations in the loop (the number in `range()`), the number of pixels that the pen moves forward (the expression in `t.forward()`, or the angle that the pen turns in each step (the number in `t.right()` or `t.left()`). Take screenshots of your artistic creations to save and share them!

Circle the line of code that you didn't see in the previous example.

```
import turtle
t = turtle.Pen()
turtle.bgcolor("black")
t.speed(0)
colors = ["green", "blue", "purple", "red"]
for x in range(200):
     t.pencolor(colors[x%4])
     t.forward(x)
     t.left(91)
turtle.done()
```

What did that single line of code do?

How is this picture different?

```
import turtle
t = turtle.Pen()
turtle.bgcolor("green")
t.speed(0)
colors = ["green", "blue", "purple", "red"]
for x in range(200):
     t.pencolor(colors[x%4])
     t.forward(x)
     t.left(91)
turtle.done()
```

What's strange about this picture? Why do you think that happens?

Run the following code.

```
import turtle
t = turtle.Pen()
t.speed(0)
turtle.bgcolor("black")
# You can choose between 2 and 6 sides for some cool shapes!
sides = 6
colors = ["red", "yellow", "blue", "orange", "green", "purple"]
for x in range(360):
     t.pencolor(colors[x%sides])
     t.forward(3/sides * x + x)
     t.left(360/sides + 1)
     t.width(x*sides/200)
turtle.done()
```

Describe what you see. What angle does the pen turn at each step?

How is this picture different?

```
import turtle
t = turtle.Pen()
t.speed(0)
turtle.bgcolor("black")
# You can choose between 2 and 6 sides for some cool shapes!
sides = 6
colors = ["red", "yellow", "blue", "orange", "green", "purple"]
for x in range(360):
     t.pencolor(colors[x%sides])
     t.forward(3/sides * x + x)
     t.right(360/sides + 1)
     t.width(x*sides/200)
turtle.done()
```

Create at least four different variations of this picture by changing the number of sizes, the colors, the number of steps or iterations in the loop (the number in `range()`), the number of pixels that the pen moves forward (the expression in `t.forward()`, the width of the pen (the expression in `t.width()`), or the angle that the pen turns in each step (the expression in `t.right()` or `t.left()`). Take screenshots of your artistic creations to save and share them! Now it's your turn to just play around with this code. See if you can create something beautiful that nobody has ever seen before!

Appendix A: Python Turtle Color Palette

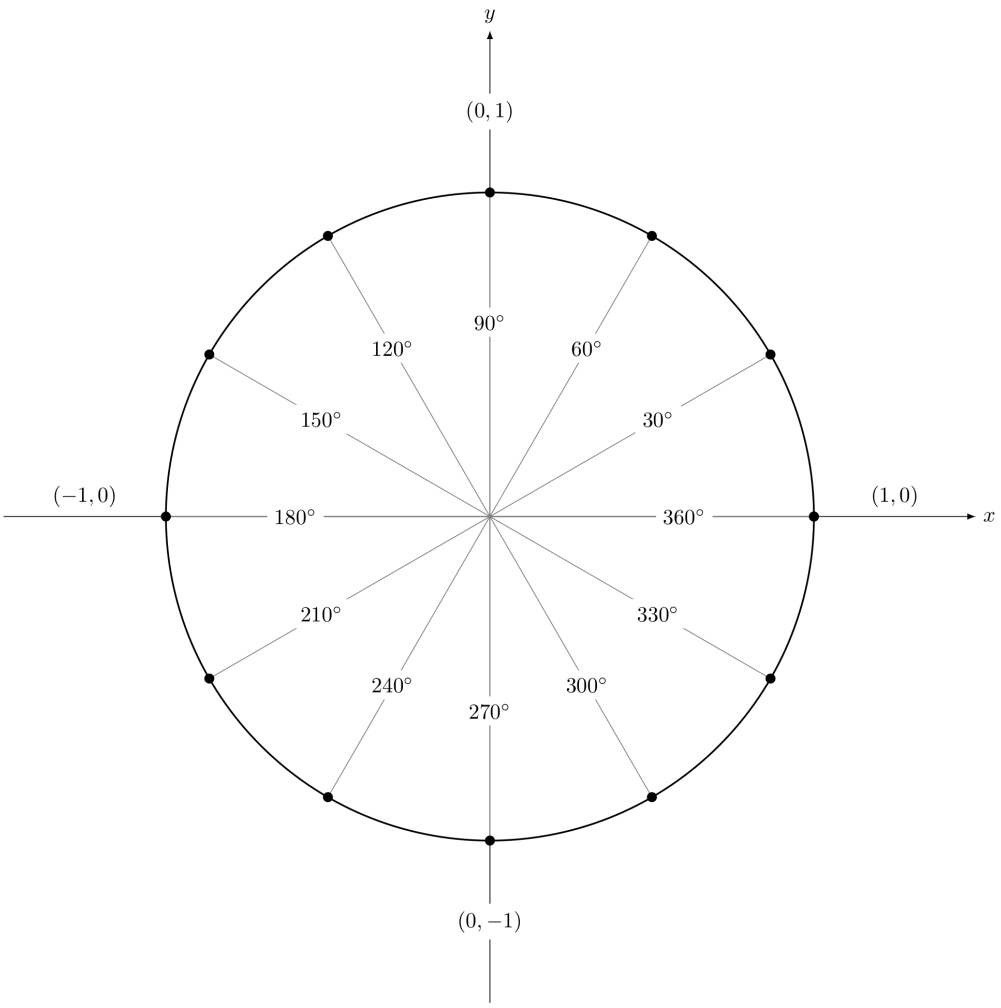
snow	deep sky blue	gold	seashell3	SlateBlue2	LightBlue3	SpringGreen2	DarkGoldenrod1	brown4	pink3	purple1	gray26	gray64
ghost white	sky blue	light goldenrod	seashell4	SlateBlue3	LightBlue4	SpringGreen3	DarkGoldenrod2	salmon1	pink4	purple2	gray27	gray65
white smoke	light sky blue	goldenrod	AntiqueWhite1	SlateBlue4	LightCyan2	SpringGreen4	DarkGoldenrod3	salmon2	LightPink1	purple3	gray28	gray66
gainsboro	steel blue	dark goldenrod	AntiqueWhite2	RoyalBlue1	LightCyan3	green2	DarkGoldenrod4	salmon3	LightPink2	purple4	gray29	gray67
floral white	light steel blue	rosy brown	AntiqueWhite3	RoyalBlue2	LightCyan4	green3	RosyBrown1	salmon4	LightPink3	MediumPurple1	gray30	gray68
old lace	light blue	indian red	AntiqueWhite4	RoyalBlue3	PaleTurquoise1	green4	RosyBrown2	LightSalmon2	LightPink4	MediumPurple2	gray31	gray69
linen	powder blue	saddle brown	bisque2	RoyalBlue4	PaleTurquoise2	chartreuse2	RosyBrown3	LightSalmon3	PaleVioletRed1	MediumPurple3	gray32	gray70
antique white	pale turquoise	sandy brown	bisque3	blue2	PaleTurquoise3	chartreuse3	RosyBrown4	LightSalmon4	PaleVioletRed2	MediumPurple4	gray33	gray71
papaya whip	dark turquoise	dark salmon	bisque4	blue4	PaleTurquoise4	chartreuse4	IndianRed1	orange2	PaleVioletRed3	thistle1	gray34	gray72
blanched almond	medium turquoise	salmon	PeachPuff2	DodgerBlue2	CadetBlue1	OliveDrab1	IndianRed2	orange3	PaleVioletRed4	thistle2	gray35	gray73
bisque	turquoise	light salmon	PeachPuff3	DodgerBlue3	CadetBlue2	OliveDrab2	IndianRed3	orange4	maroon1	thistle3	gray36	gray74
peach puff	cyan	orange	PeachPuff4	DodgerBlue4	CadetBlue3	OliveDrab4	IndianRed4	DarkOrange1	maroon2	thistle4	gray37	gray75
navajo white	light cyan	dark orange	NavajoWhite2	SteelBlue1	CadetBlue4	DarkOliveGreen1	sienna1	DarkOrange2	maroon3		gray38	gray76
lemon chiffon	cadet blue	coral	NavajoWhite3	SteelBlue2	turquoise1	DarkOliveGreen2	sienna2	DarkOrange3	maroon4		gray39	gray77
mint cream	medium aquamarine	light coral	NavajoWhite4	SteelBlue3	turquoise2	DarkOliveGreen3	sienna3	DarkOrange4	VioletRed1		gray40	gray78
azure	aquamarine	tomato	LemonChiffon2	SteelBlue4	turquoise3	DarkOliveGreen4	sienna4	coral1	VioletRed2		gray42	gray79
alice blue	dark green	orange red	LemonChiffon3	DeepSkyBlue2	turquoise4	khaki1	burlywood1	coral2	VioletRed3		gray43	gray80
lavender	dark olive green	red	LemonChiffon4	DeepSkyBlue3	cyan2	khaki2	burlywood2	coral3	VioletRed4		gray44	gray81
lavender blush	dark sea green	hot pink	cornsilk2	DeepSkyBlue4	cyan3	khaki3	burlywood3	coral4	magenta2		gray45	gray82
misty rose	sea green	deep pink	cornsilk3	SkyBlue1	cyan4	khaki4	burlywood4	tomato2	magenta3		gray46	gray83
dark slate gray	medium sea green	pink	cornsilk4	SkyBlue2	DarkSlateGray1	LightGoldenrod1	wheat1	tomato3	magenta4		gray47	gray84
dim gray	light sea green	light pink	ivory2	SkyBlue3	DarkSlateGray2	LightGoldenrod2	wheat2	tomato4	orchid1		gray48	gray85
slate gray	pale green	pale violet red	ivory3	SkyBlue4	DarkSlateGray3	LightGoldenrod3	wheat3	OrangeRed2	orchid2		gray49	gray86
light slate gray	spring green	maroon	ivory4	LightSkyBlue1	DarkSlateGray4	LightGoldenrod4	wheat4	OrangeRed3	orchid3		gray50	gray87
gray	lawn green	medium violet red	honeydew2	LightSkyBlue2	aquamarine2	LightYellow2	tan1	OrangeRed4	orchid4		gray51	gray88
light grey	medium spring green	violet red	honeydew3	LightSkyBlue3	aquamarine4	LightYellow3	tan2	red2	plum1		gray52	gray89
midnight blue	green yellow	medium orchid	honeydew4	LightSkyBlue4	DarkSeaGreen1	LightYellow4	tan4	red3	plum2		gray53	gray90
navy	lime green	dark orchid	LavenderBlush2	SlateGray1	DarkSeaGreen2	yellow2	chocolate1	red4	plum3		gray54	gray91
cornflower blue	yellow green	dark violet	LavenderBlush3	SlateGray2	DarkSeaGreen3	yellow3	chocolate2	DeepPink2	plum4		gray55	gray92
dark slate blue	forest green	blue violet	LavenderBlush4	SlateGray3	DarkSeaGreen4	yellow4	chocolate3	DeepPink3	MediumOrchid1		gray56	gray93
slate blue	olive drab	purple	MistyRose2	SlateGray4	SeaGreen1	gold2	firebrick1	DeepPink4	MediumOrchid2		gray57	gray94
medium slate blue	dark khaki	medium purple	MistyRose3	LightSteelBlue1	SeaGreen2	gold3	firebrick2	HotPink1	MediumOrchid3		gray58	gray95
light slate blue	khaki	thistle	MistyRose4	LightSteelBlue2	SeaGreen3	gold4	firebrick3	HotPink2	MediumOrchid4		gray59	gray97
medium blue	pale goldenrod	snow2	azure2	LightSteelBlue3	PaleGreen1	goldenrod1	firebrick4	HotPink3	DarkOrchid1		gray60	gray98
royal blue	light goldenrod yellow	snow3	azure3	LightSteelBlue4	PaleGreen2	goldenrod2	brown1	HotPink4	DarkOrchid2		gray61	gray99
blue	light yellow	snow4	azure4	LightBlue1	PaleGreen3	goldenrod3	brown2	pink1	DarkOrchid3		gray62	
dodger blue	yellow	seashell2	SlateBlue1	LightBlue2	PaleGreen4	goldenrod4	brown3	pink2	DarkOrchid4		gray63	

The table above lists all of the colors that turtle uses to draw with. If you would like the turtle pen to draw something in "midnight blue", for example, you can use the command `pencolor("midnight blue")`.

The Python code below will draw a straight line to the right that is 200 pixels long and is "orange red" in color:

```
import turtle
t = turtle.Pen()
t.pencolor("orange red")
t.forward(200)
turtle.done()
```

Appendix B: Angles on the Unit Circle (degrees)



The unit circle above (a circle with radius one, plotted on the x-y coordinate plane) is intended to help you get a better feel for the angles that Turtle uses. In mathematics, the x-axis is by definition 0°. After one revolution counter-clockwise, the x-axis is 360°.

Appendix C: List of Turtle Methods (Functions)

From the Python 3.3.7 documentation at https://docs.python.org/3.3/library/turtle.html

Turtle motion

Move and draw

```
forward() | fd()
backward() | bk() | back()
right() | rt()
left() | lt()
goto() | setpos() | setposition()
setx()
sety()
setheading() | seth()
home()
circle()
dot()
stamp()
clearstamp()
clearstamps()
undo()
speed()
```

Tell Turtle's state

```
position() | pos()
towards()
xcor()
ycor()
heading()
distance()
```

Setting and measurement

```
degrees()
radians()
```

Pen control

Drawing state

```
pendown() | pd() | down()
penup() | pu() | up()
pensize() | width()
pen()
isdown()
```

Color control

```
color()
pencolor()
fillcolor()
```

Filling

```
filling()
begin_fill()
end_fill()
```

More drawing control

```
reset()
clear()
write()
```

Turtle state

Visibility

```
showturtle() | st()
hideturtle() | ht()
isvisible()
```

Appendix C: List of Turtle Methods (Functions)

From the Python 3.3.7 documentation at https://docs.python.org/3.3/library/turtle.html

Appearance

```
shape()
resizemode()
shapesize() | turtlesize()
shearfactor()
settiltangle()
tiltangle()
tilt()
shapetransform()
get_shapepoly()
```

Using events

```
onclick()
onrelease()
ondrag()
```

Methods of TurtleScreen/ Screen

Window control

```
bgcolor()
bgpic()
clear() | clearscreen()
reset() | resetscreen()
screensize()
setworldcoordinates()
```

Animation control

```
delay()
tracer()
update()
```

Using screen events

```
listen()
onkey() | onkeyrelease()
onkeypress()
onclick() | onscreenclick()
ontimer()
mainloop() | done()
```

Settings and special methods

```
mode()
colormode()
getcanvas()
getshapes()
register_shape() | addshape()
turtles()
window_height()
window_width()
```

Input methods

```
textinput()
numinput()
```

Methods specific to Screen

```
bye()
exitonclick()
setup()
title()
```