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

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:

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:

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:

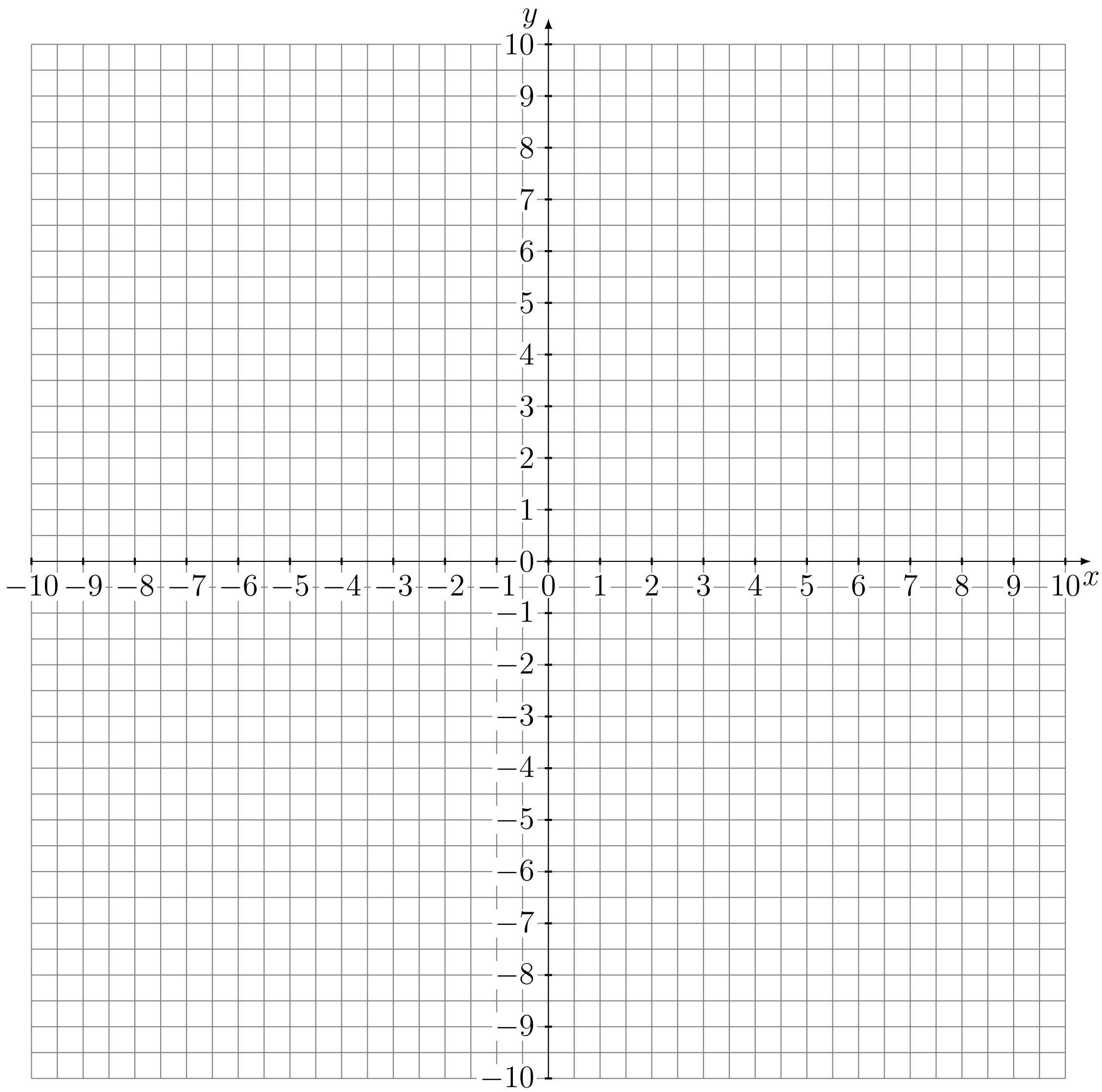
If you want, write a totally different kind of Mad Libs program (make it funny!), or move on to the next section.

(optional) Write your code here:

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()
```

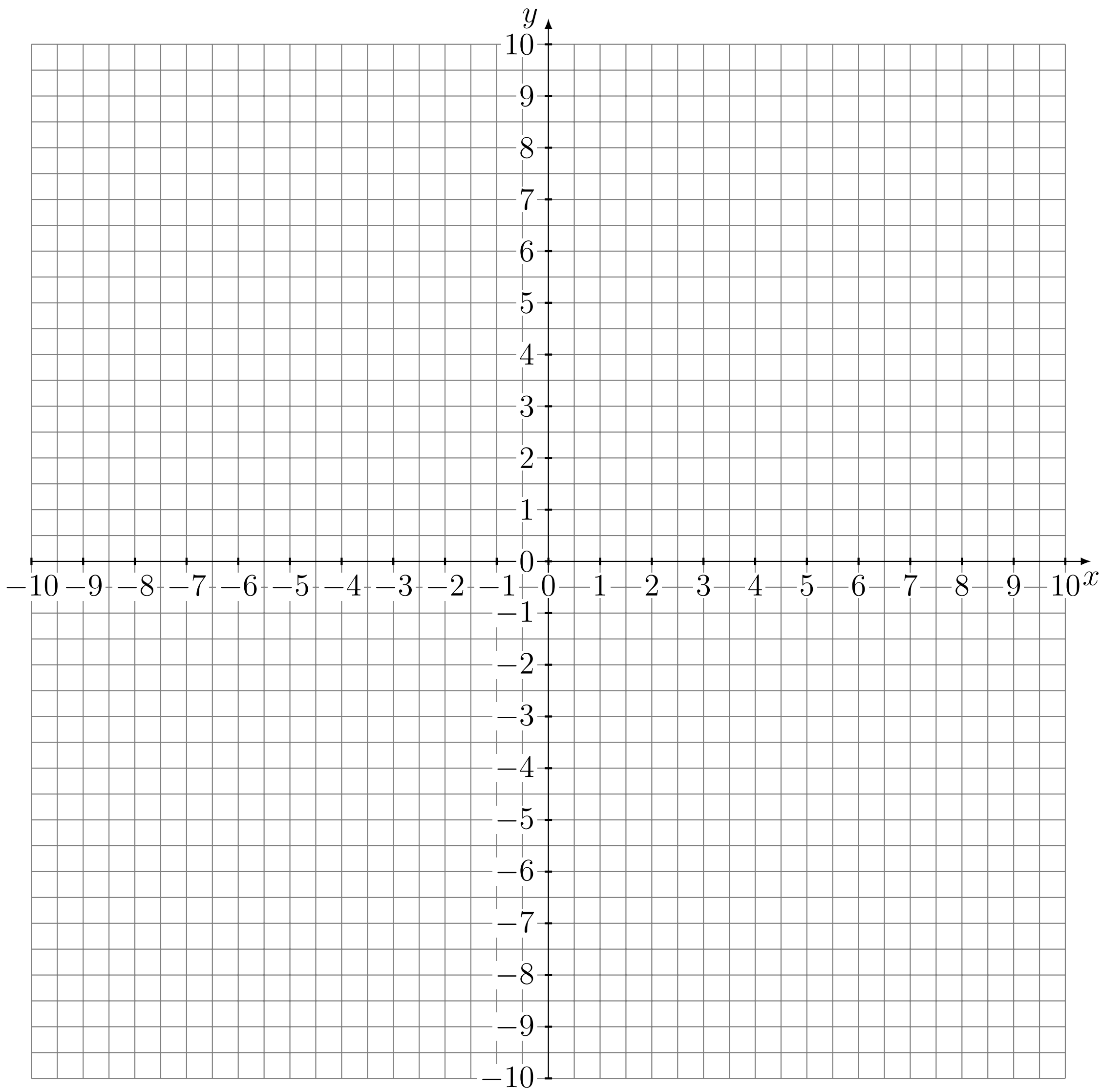


1 unit = 10 pixels

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()
```

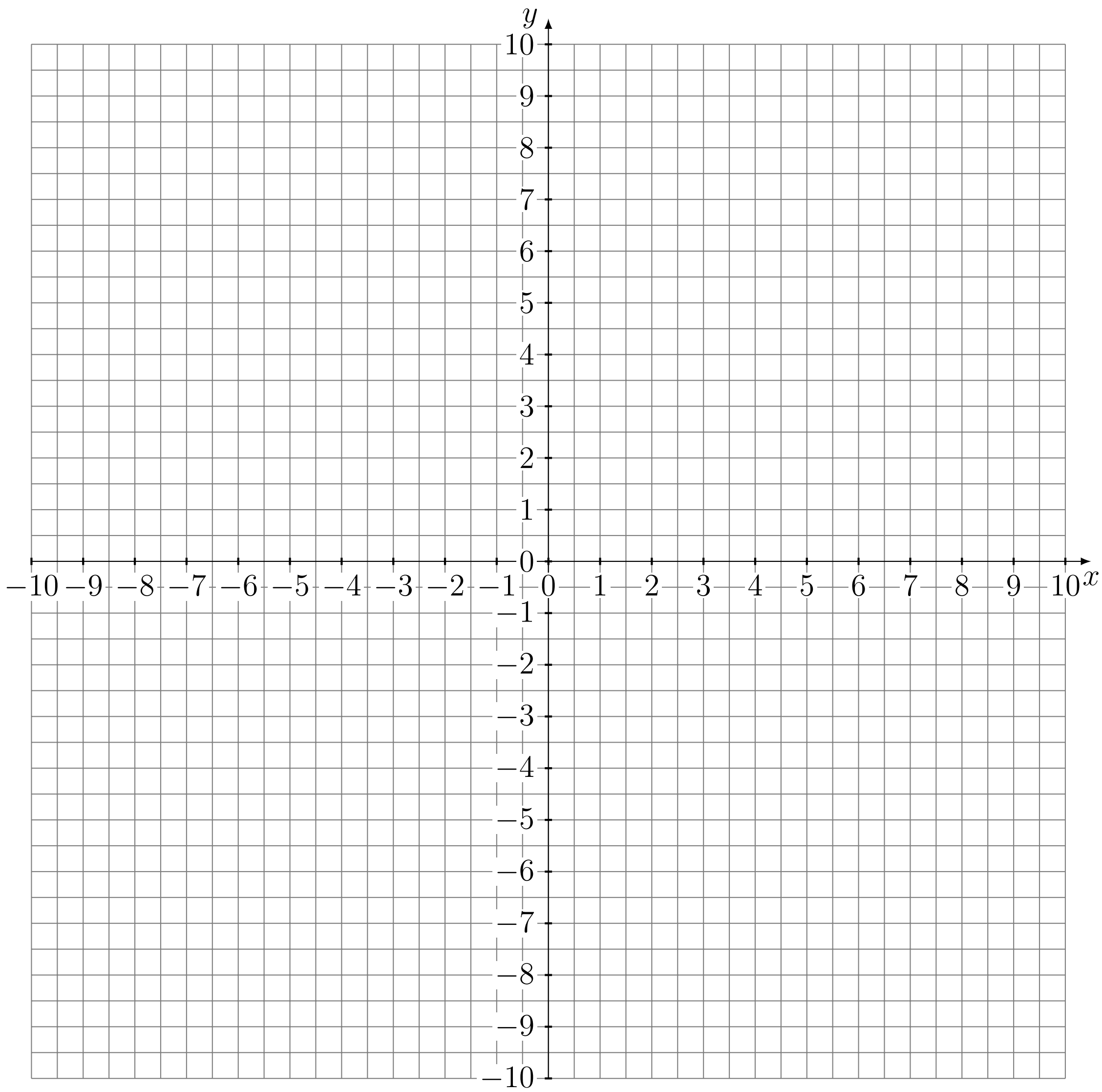


1 unit = 10 pixels

Python Turtle - drawing.1.1.py

Try to draw the output of the Python code in the graphing area below. Use a protractor, 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.backward(100)
turtle.done()
```

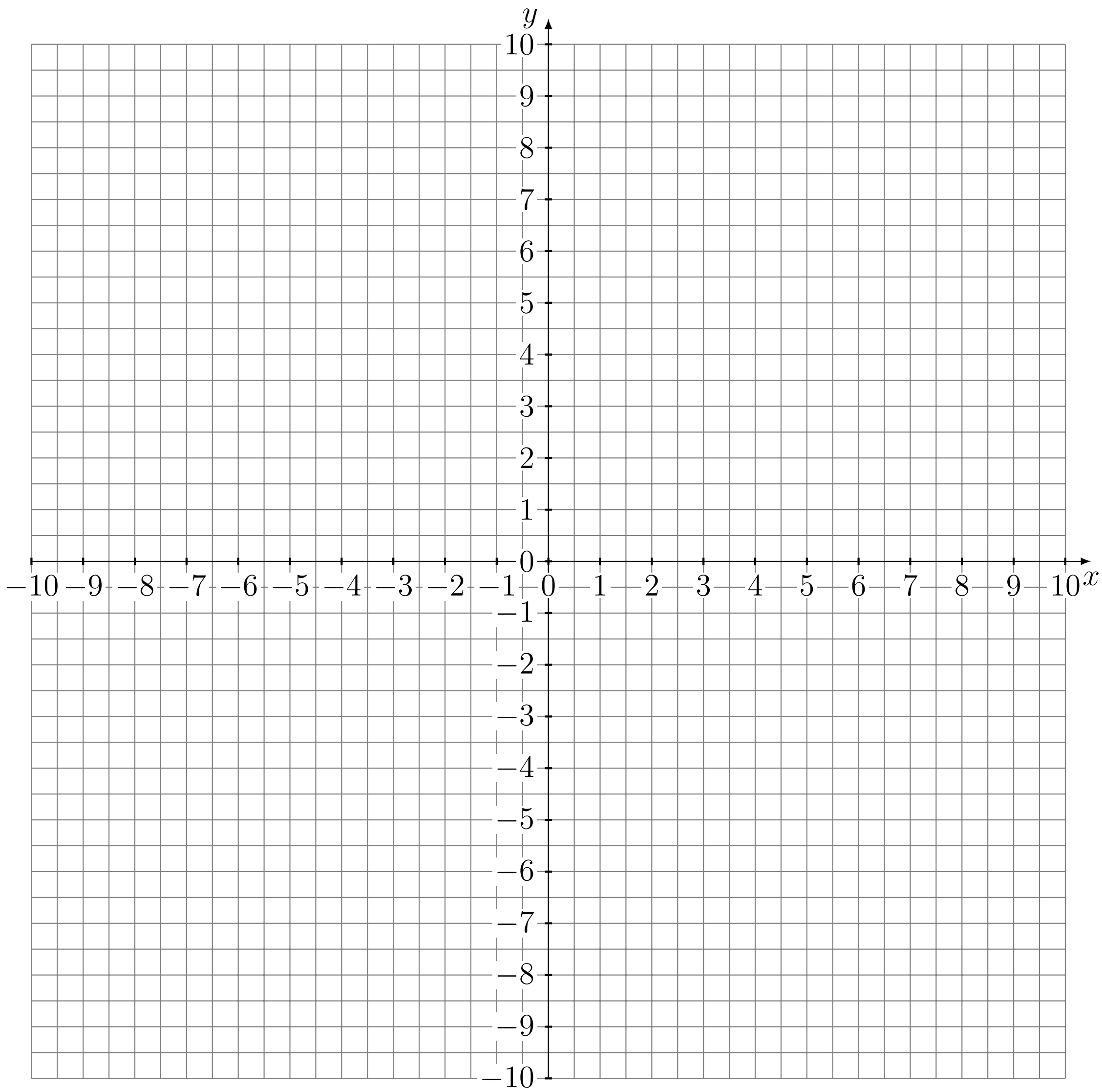


1 unit = 10 pixels

Python Turtle - drawing.1.2.py

Try to draw the output of the Python code in the graphing area below. Use a protractor, 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.forward(-100)
turtle.done()
```

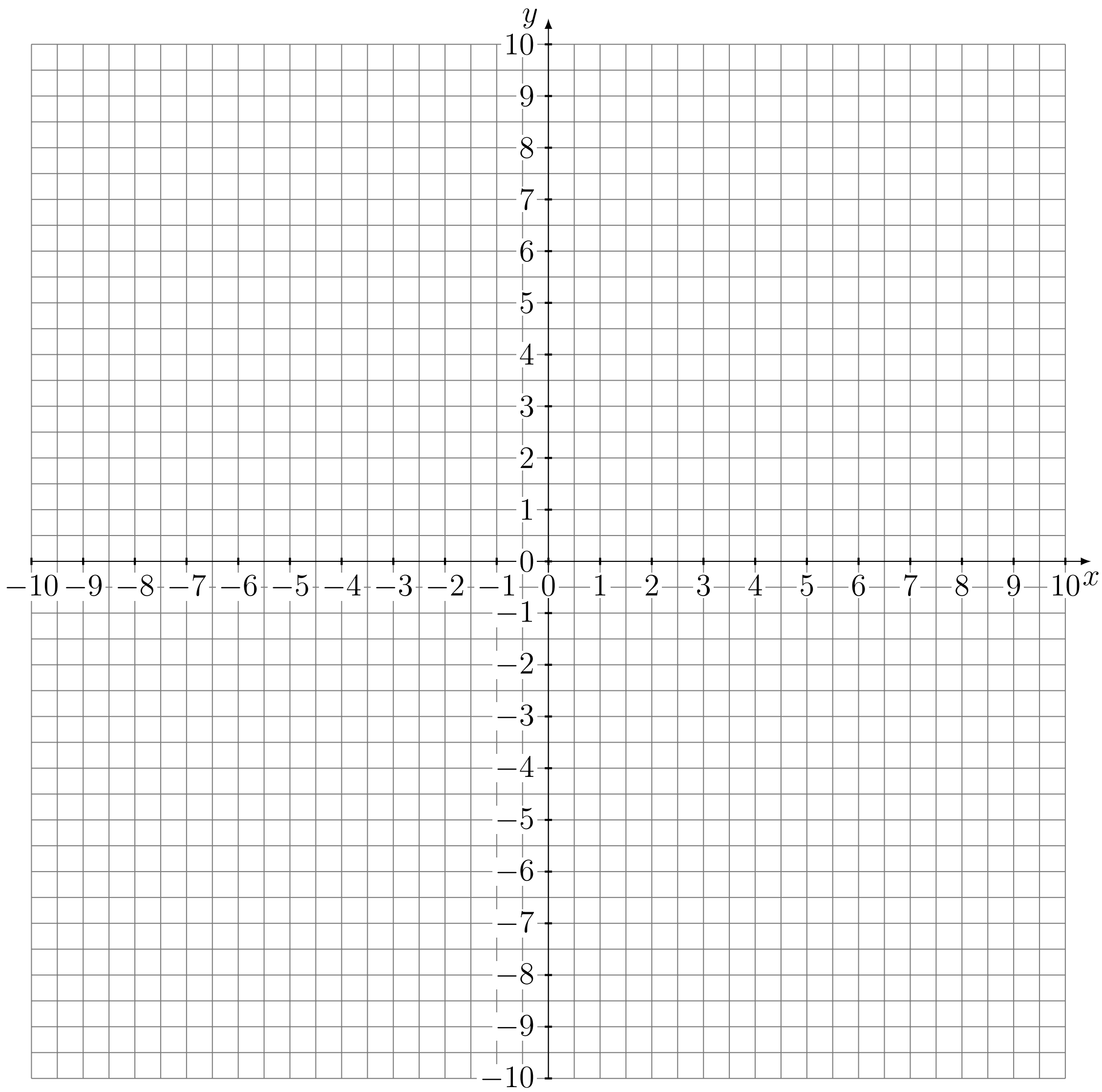


1 unit = 10 pixels

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()
```

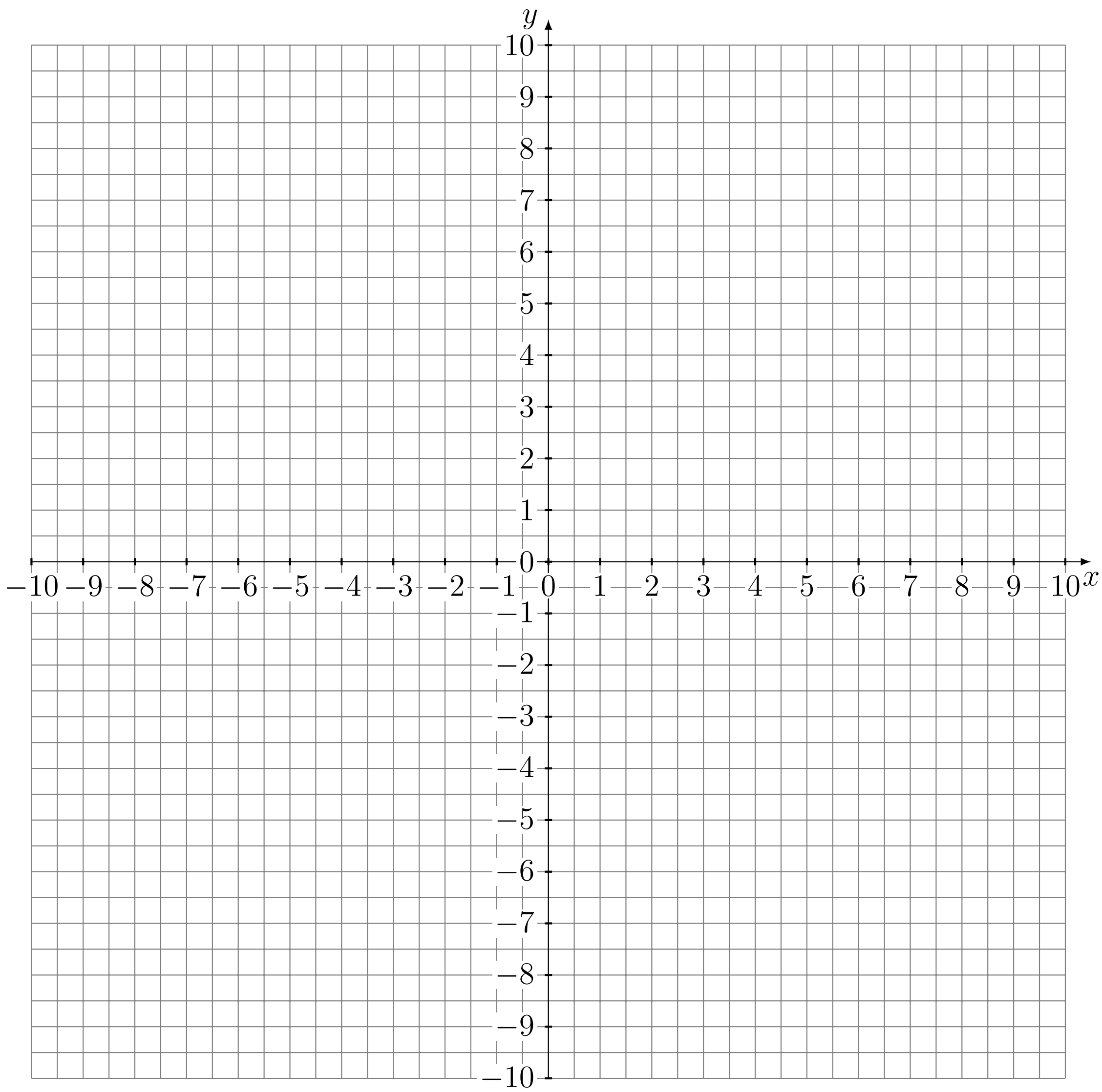


1 unit = 10 pixels

Python Turtle - drawing.2.0.py

Try to draw the output of the Python code in the graphing area below. Use a protractor, 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
mysteryshape = turtle.Pen()
mysteryshape.forward(100)
mysteryshape.left(90)
mysteryshape.forward(100)
mysteryshape.left(90)
mysteryshape.left(45)
mysteryshape.forward(141)
turtle.done()
```

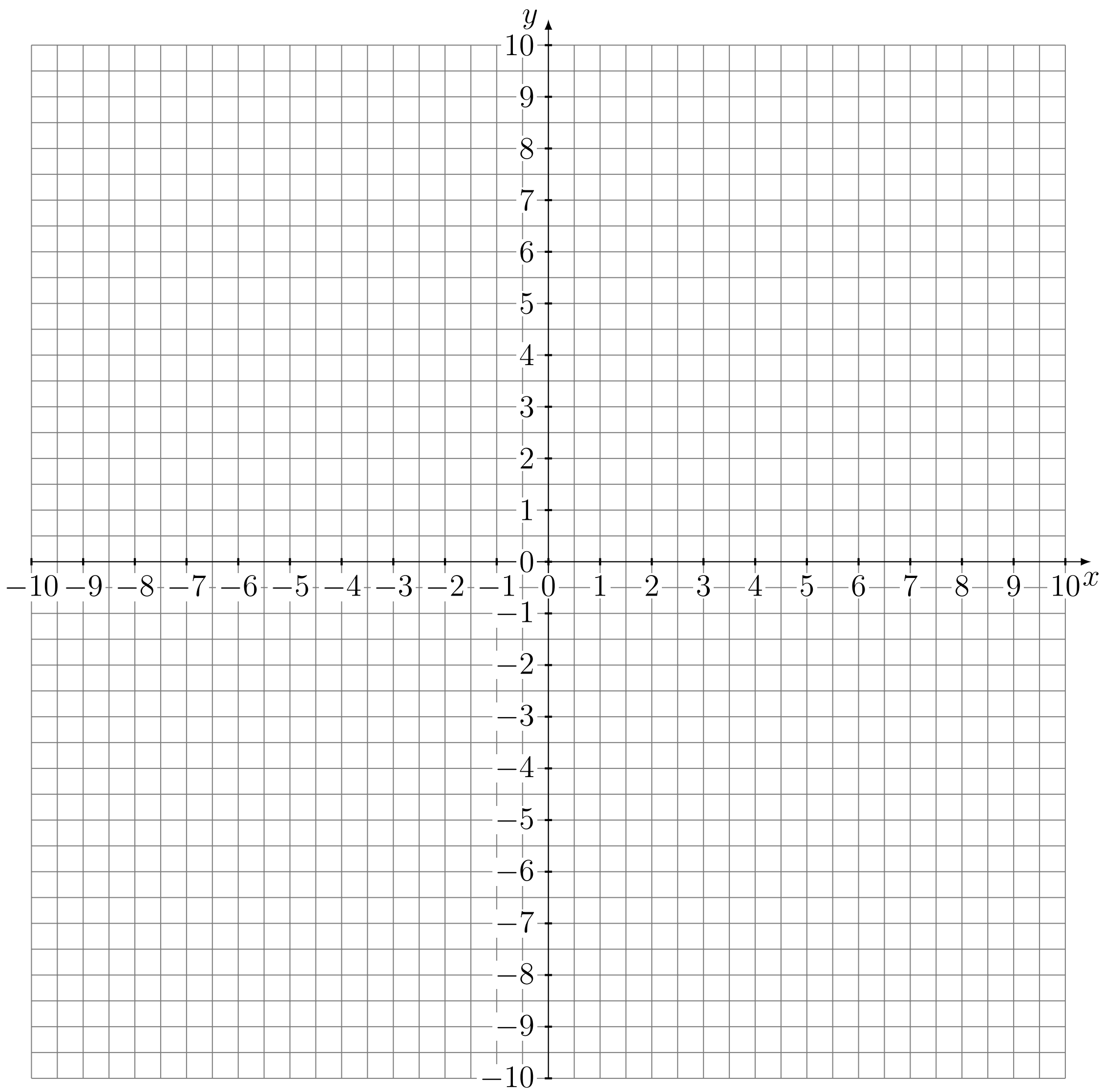


1 unit = 10 pixels

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()
```

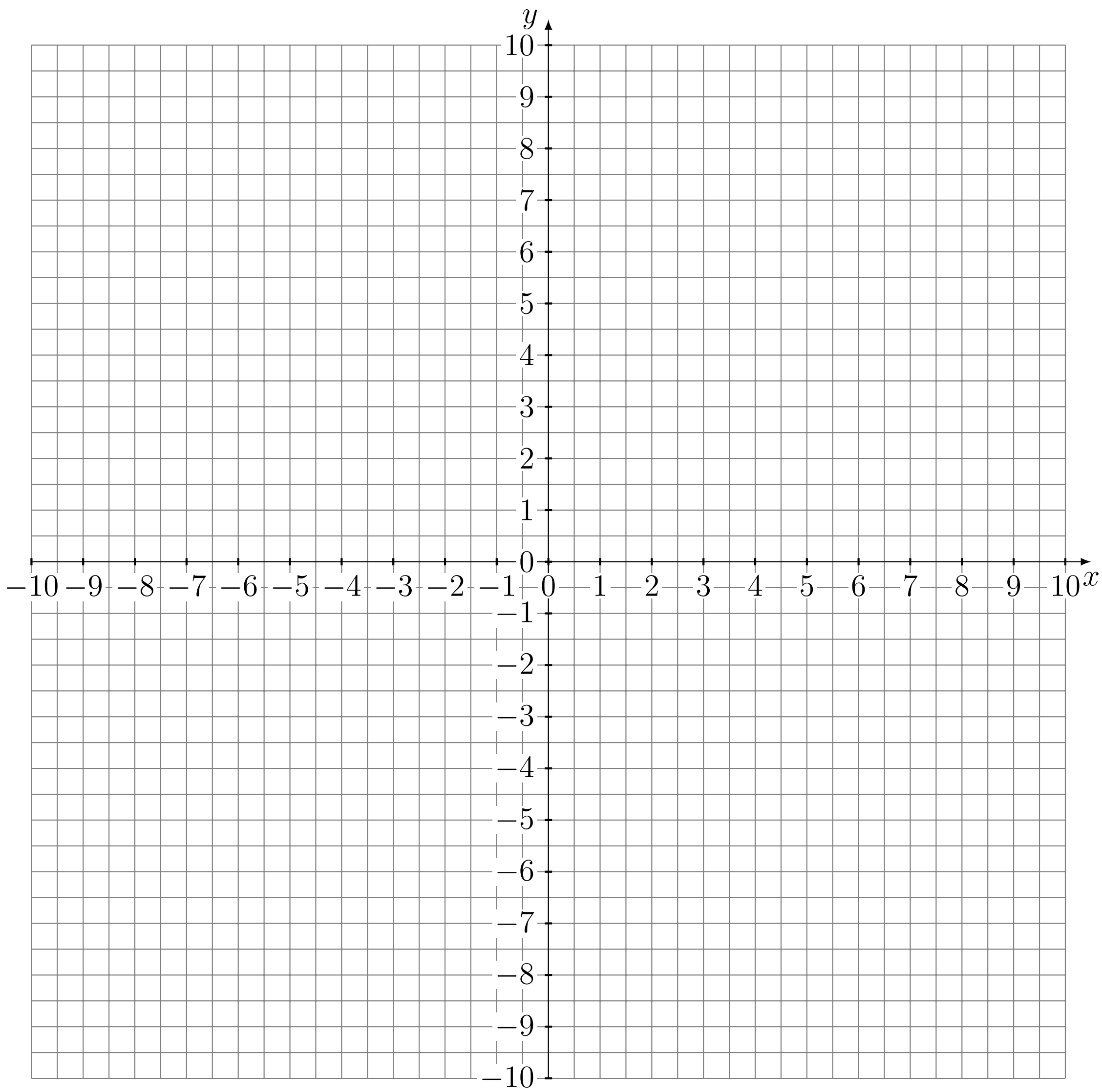


1 unit = 10 pixels

Python Turtle - drawing.2.2.py

Try to draw the output of the Python code in the graphing area below. Use a protractor, 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
s = turtle.Pen()
s.forward(100)
s.right(90)
s.forward(100)
s.right(90)
s.right(45)
s.forward(141)
turtle.done()
```

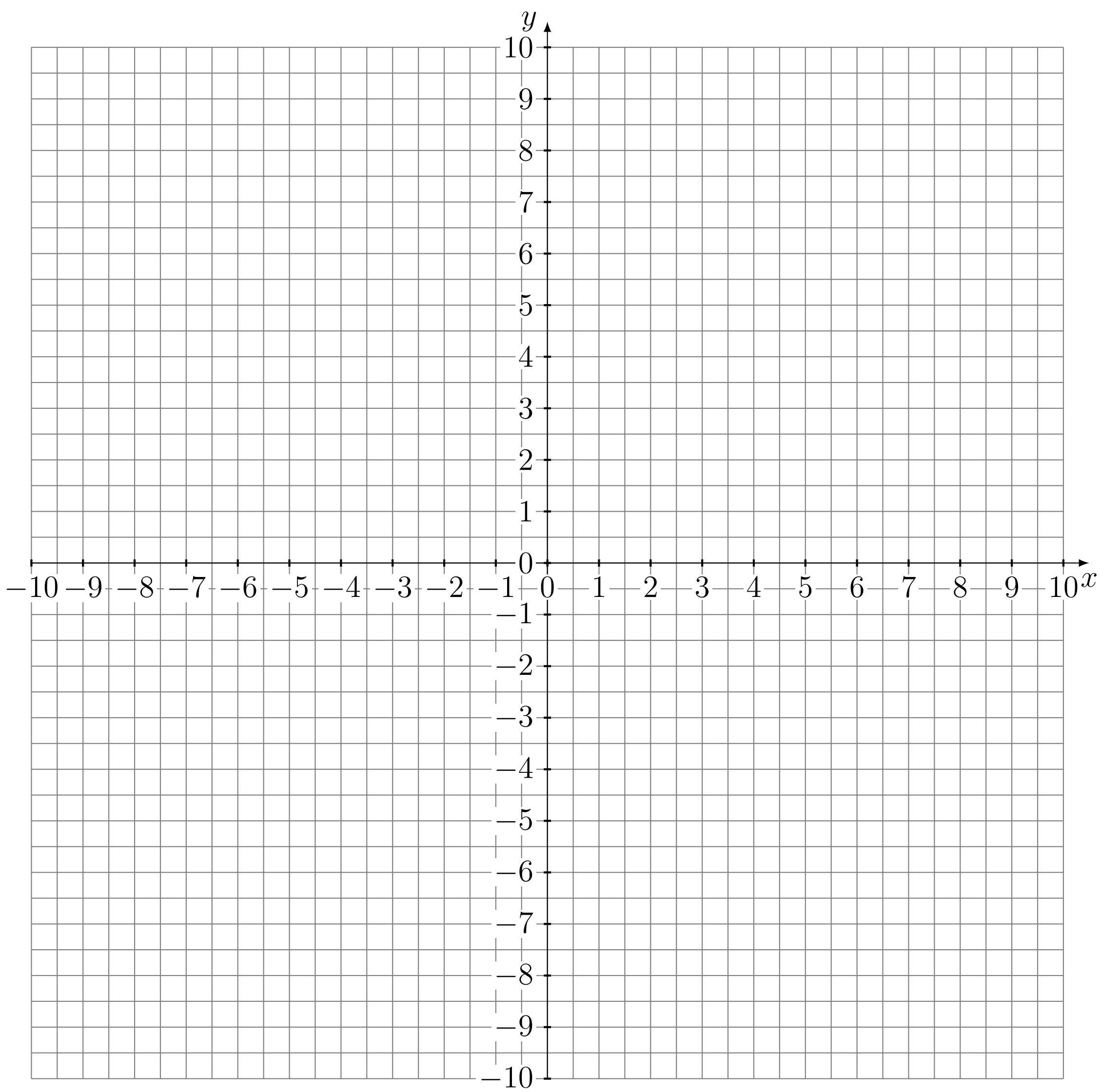


1 unit = 10 pixels

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.

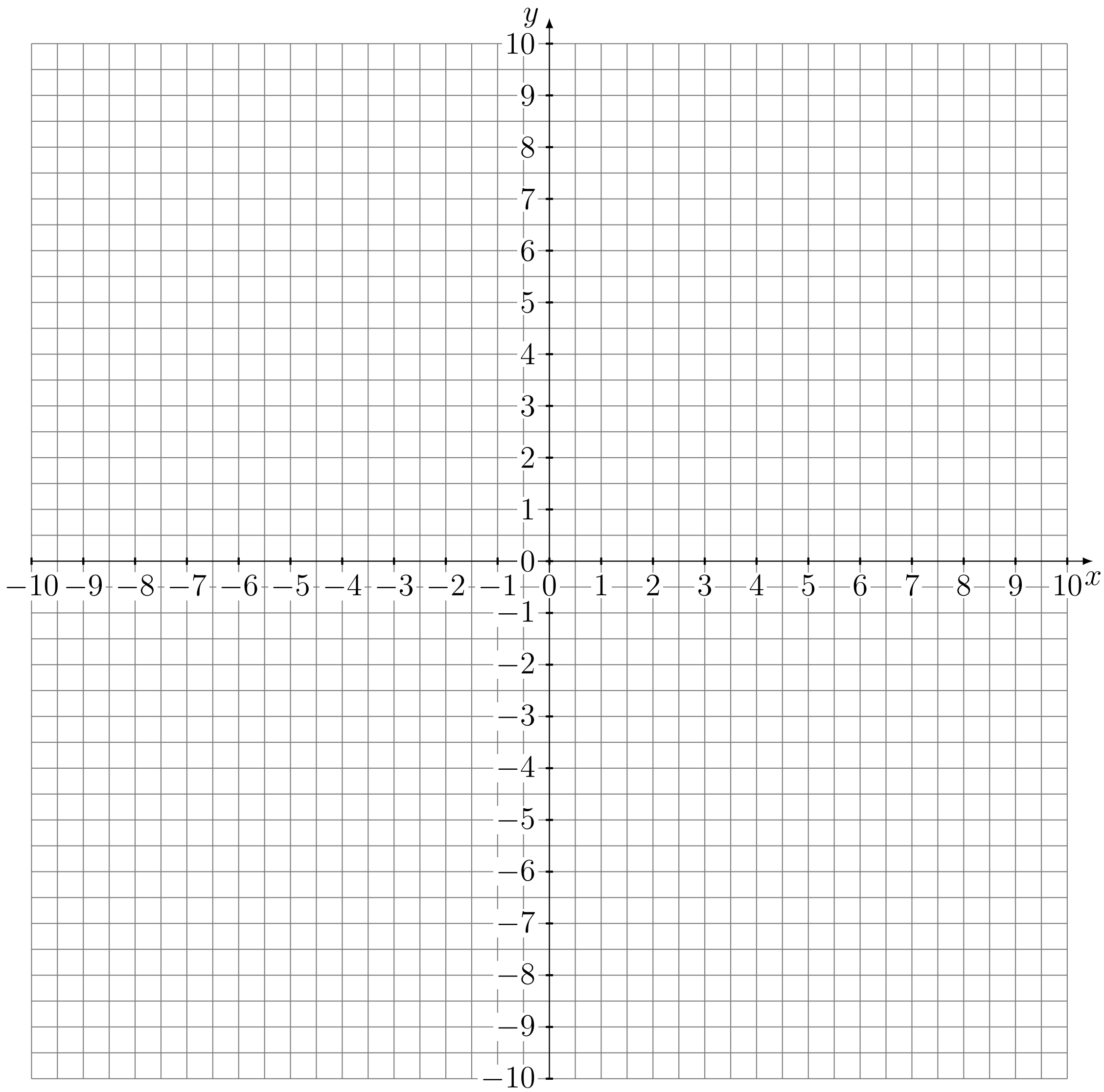
```
import turtle
q = turtle.Pen()
q.left(45)
q.forward(141.2)
q.left(135)
q.forward(200)
q.left(135)
q.forward(2*141.2)
q.right(135)
q.forward(200)
q.right(135)
q.forward(141.2)
turtle.done()
```



Python Turtle - drawing.3.0.py

Try to draw the output of the Python code in the graphing area below. Use a protractor, 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
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()
```

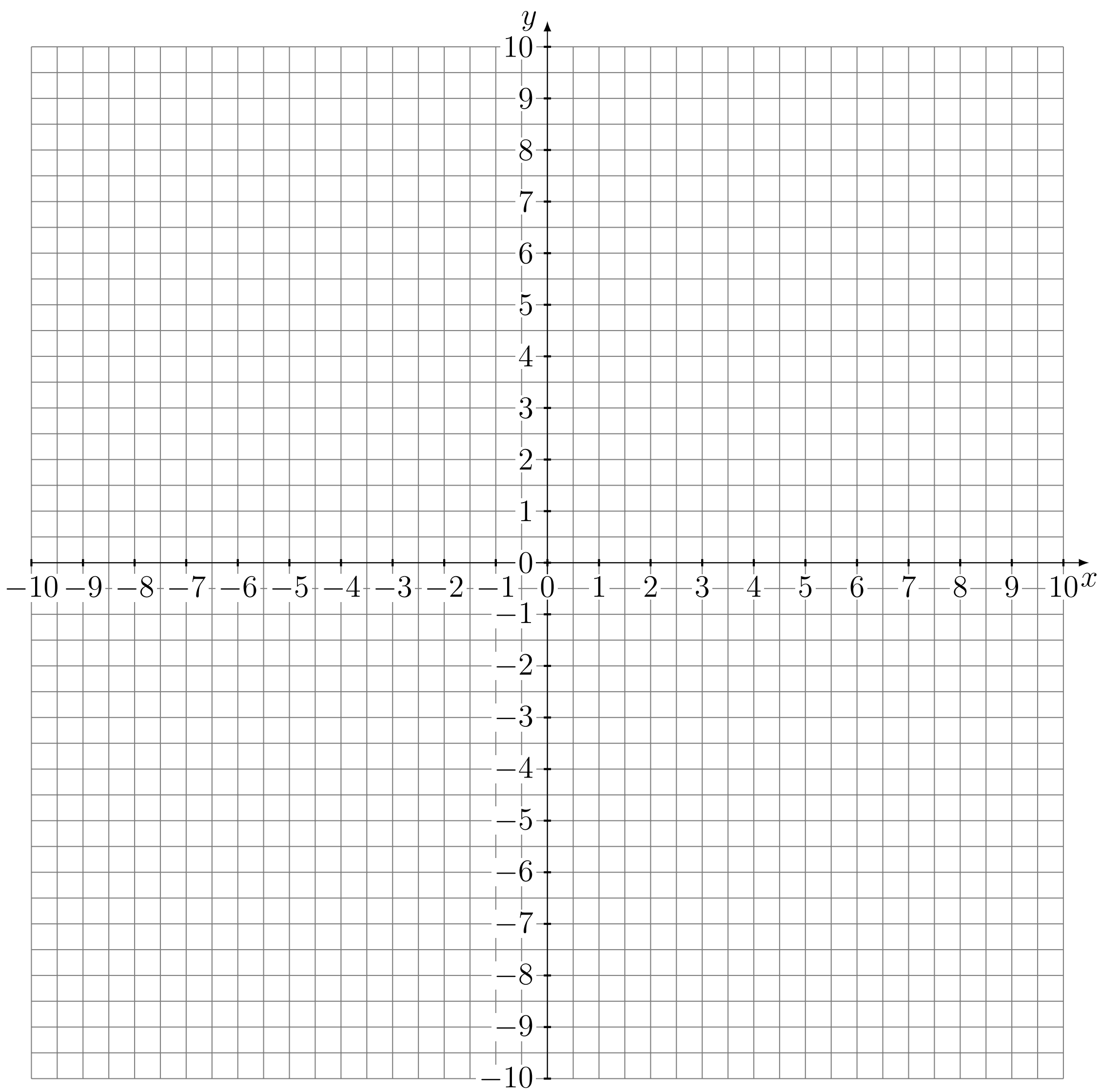


1 unit = 10 pixels

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()
```

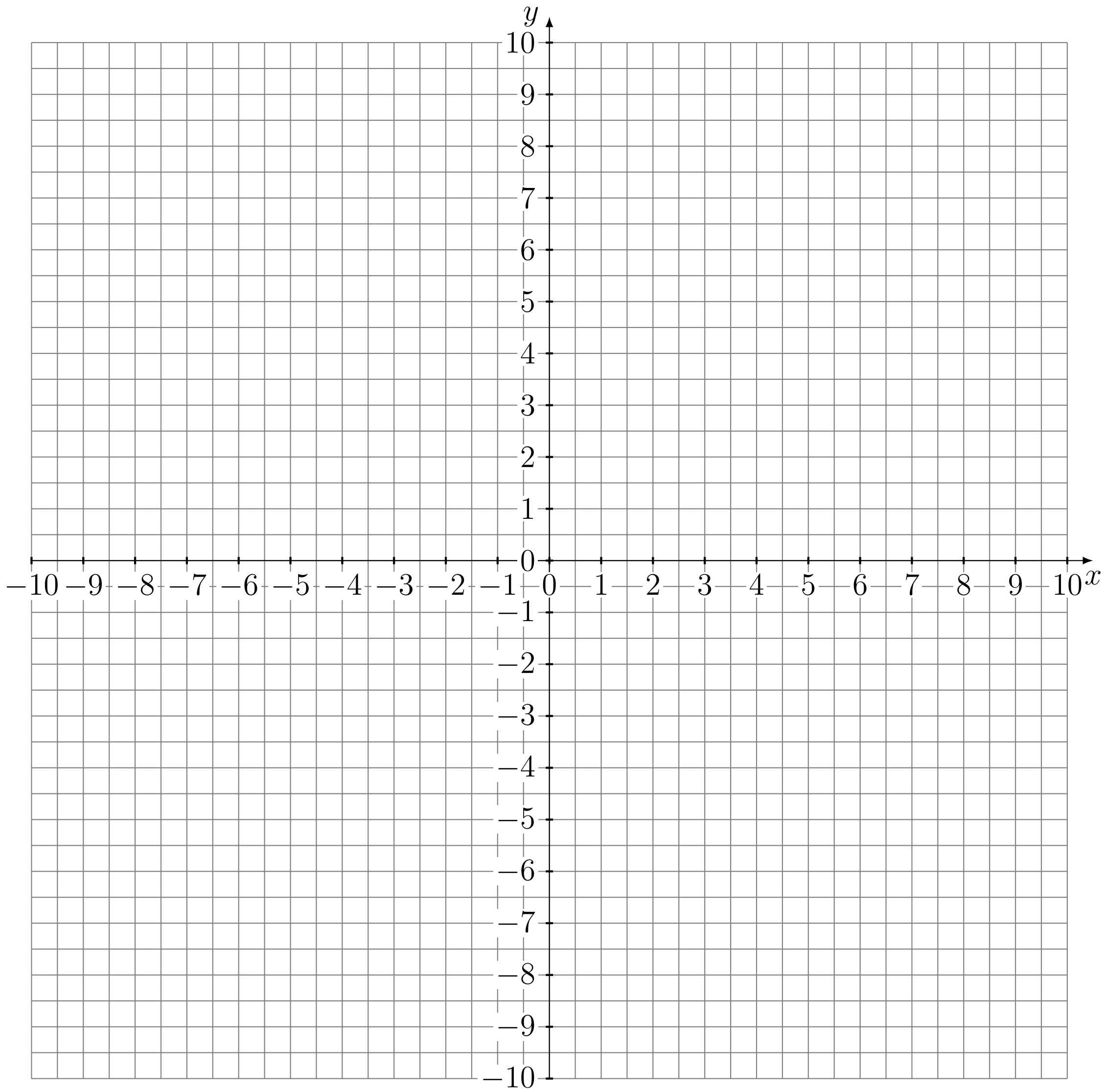


1 unit = 10 pixels

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)?

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:

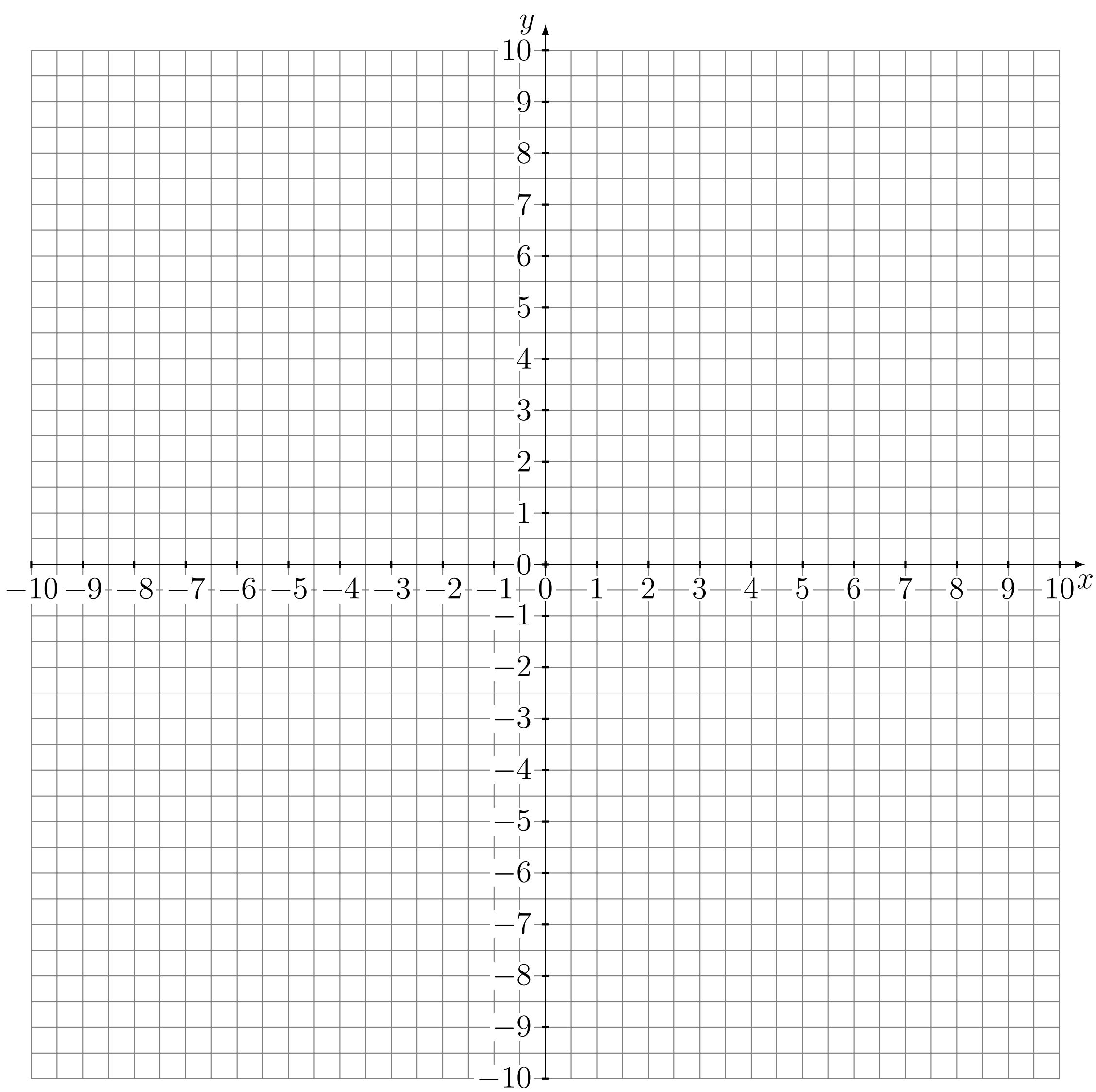
Does Python let you do arithmetic (addition, subtraction, etc.) inside the "print()" function?

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)
```

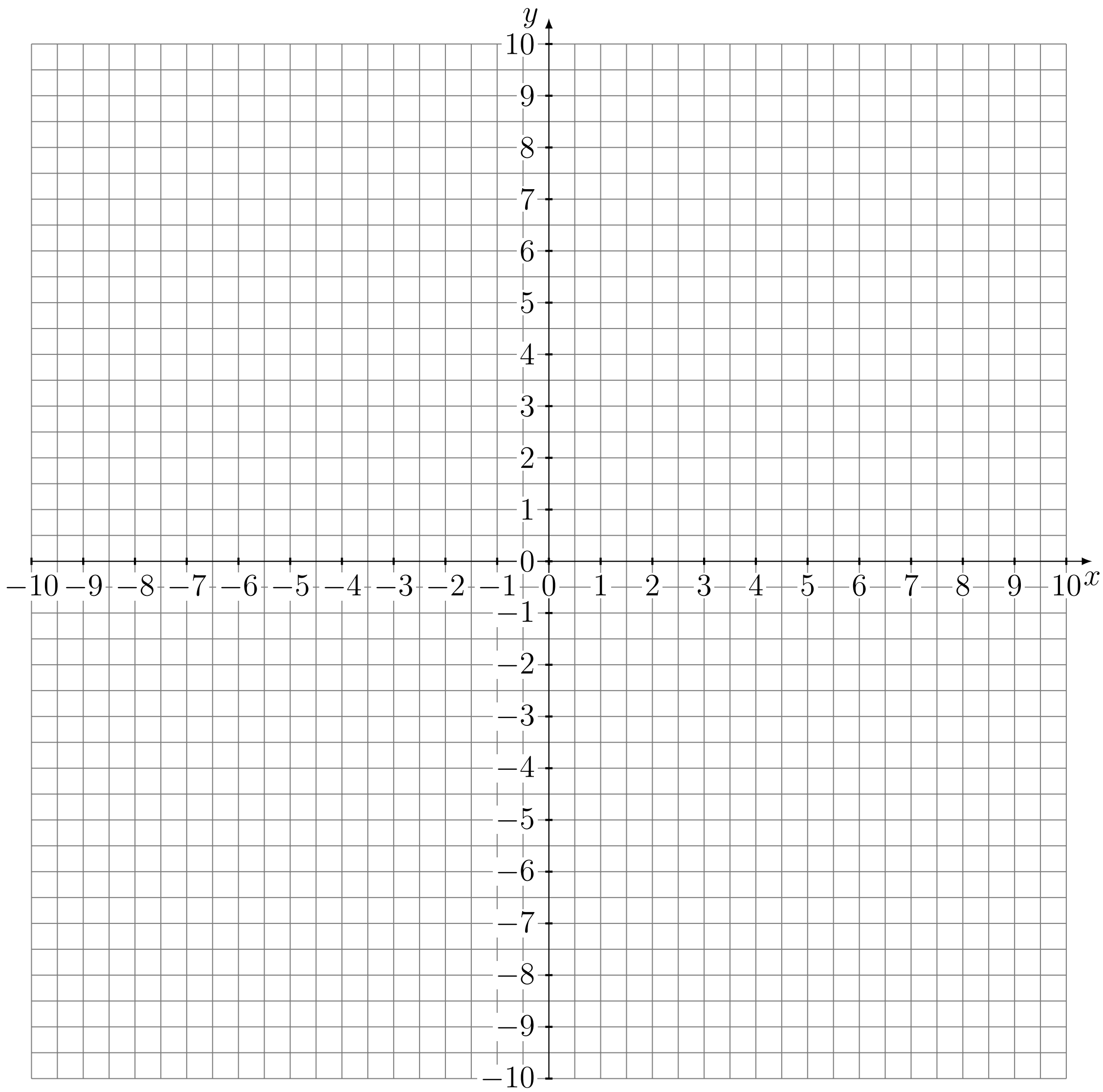
Output:

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

What is the highest number in range(10)?

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()
```

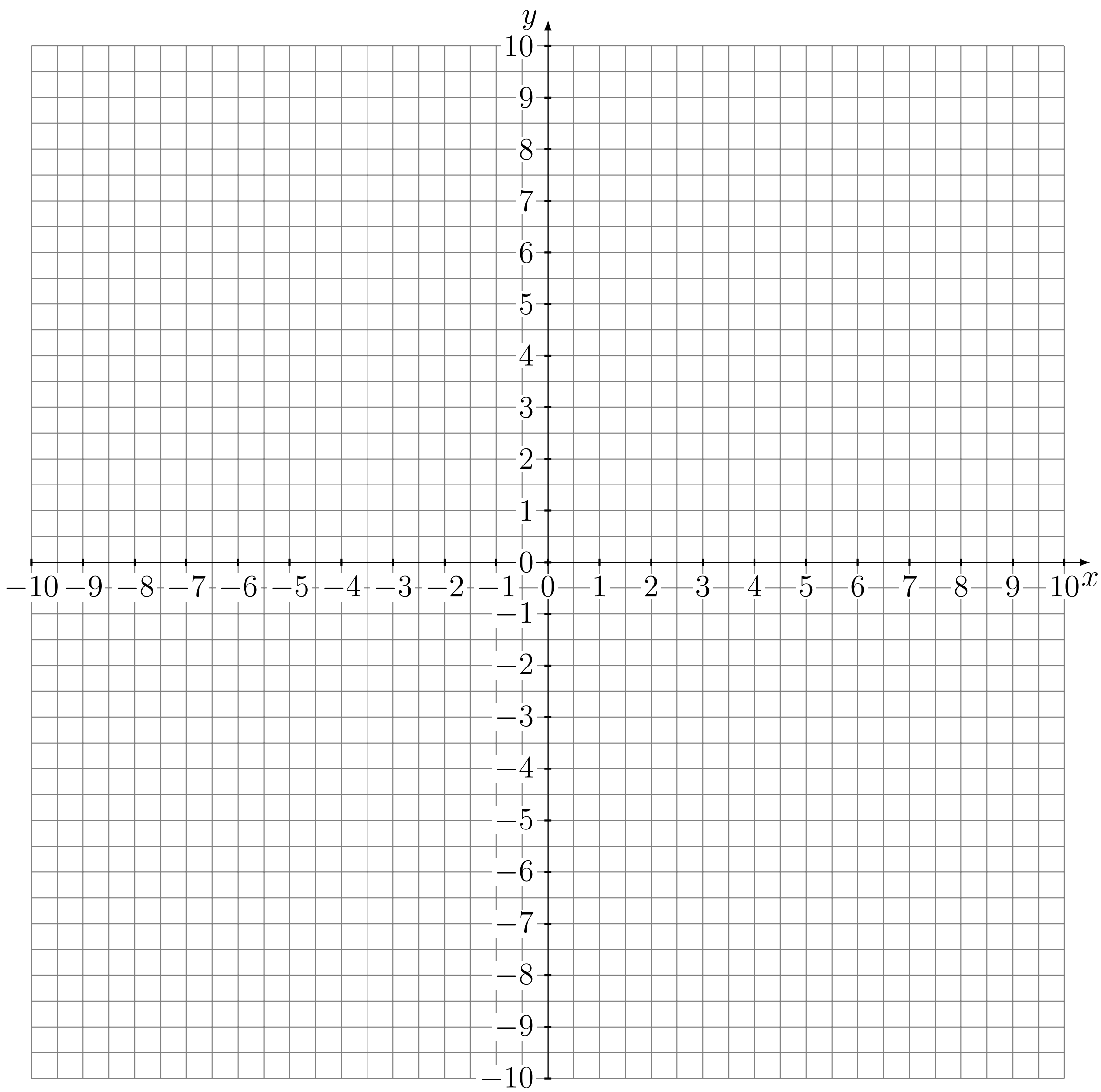


1 unit = 10 pixels

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()
```

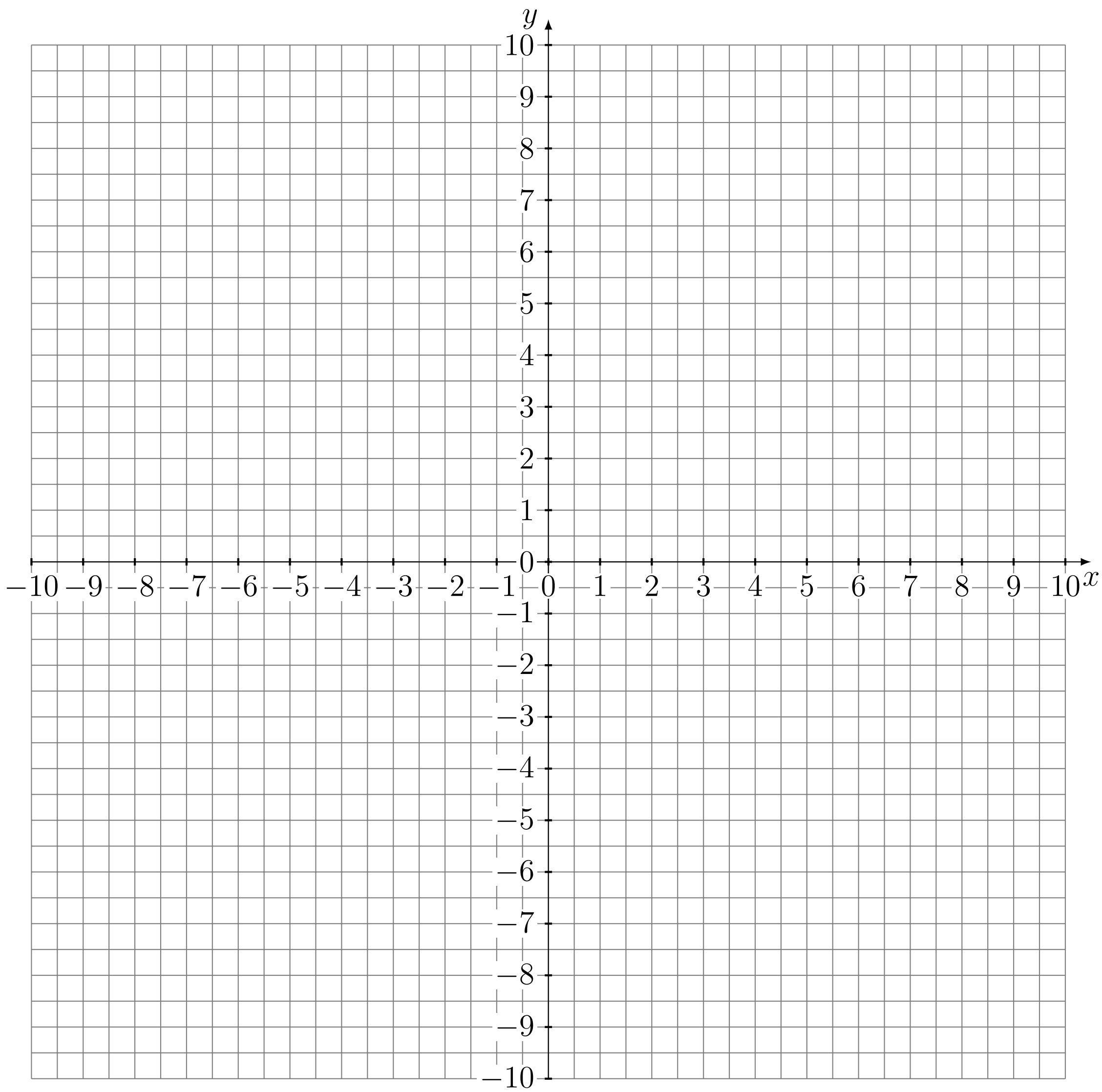


1 unit = 1 pixel

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:

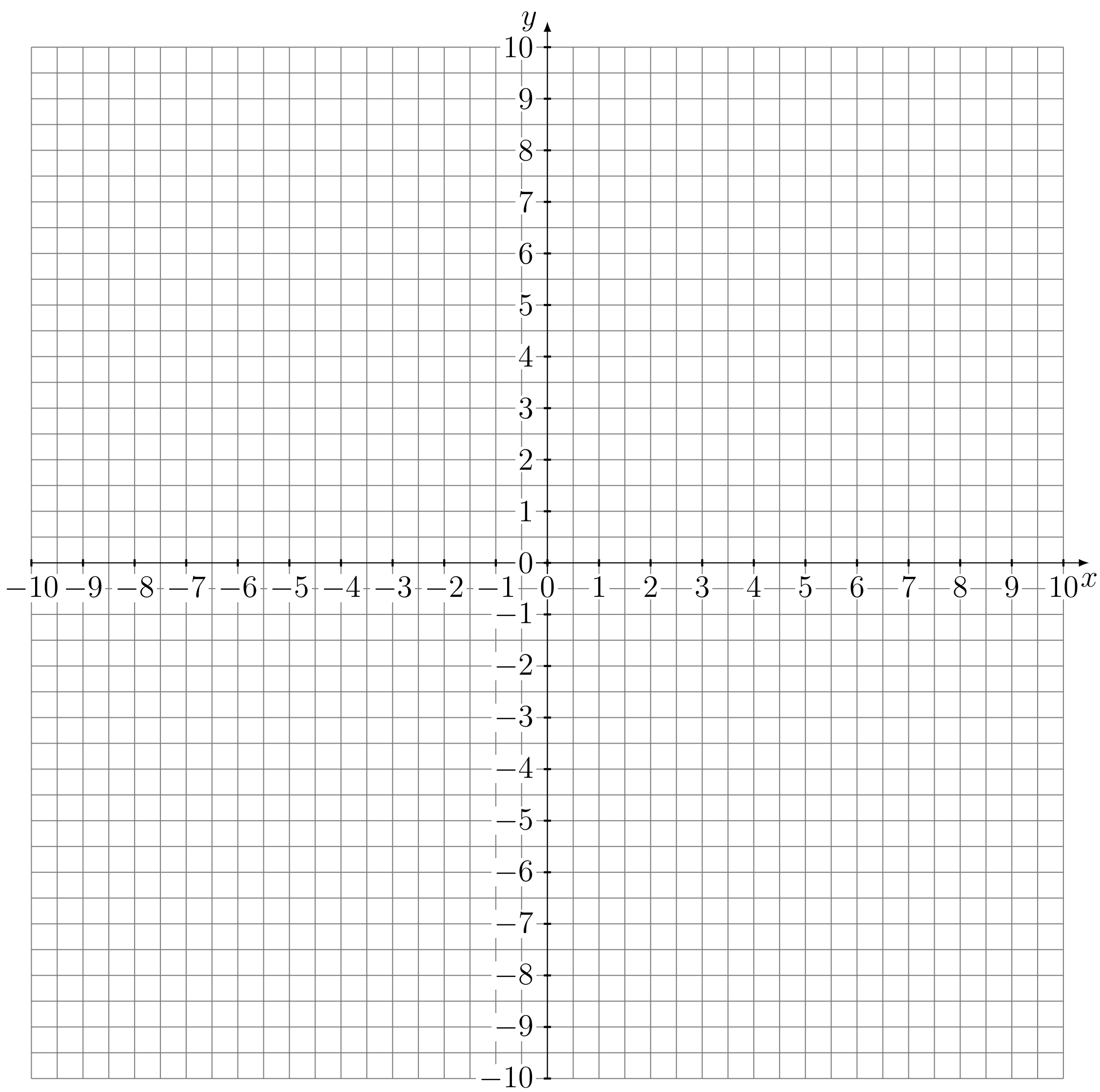
How many elements does the colors list have?

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()
```

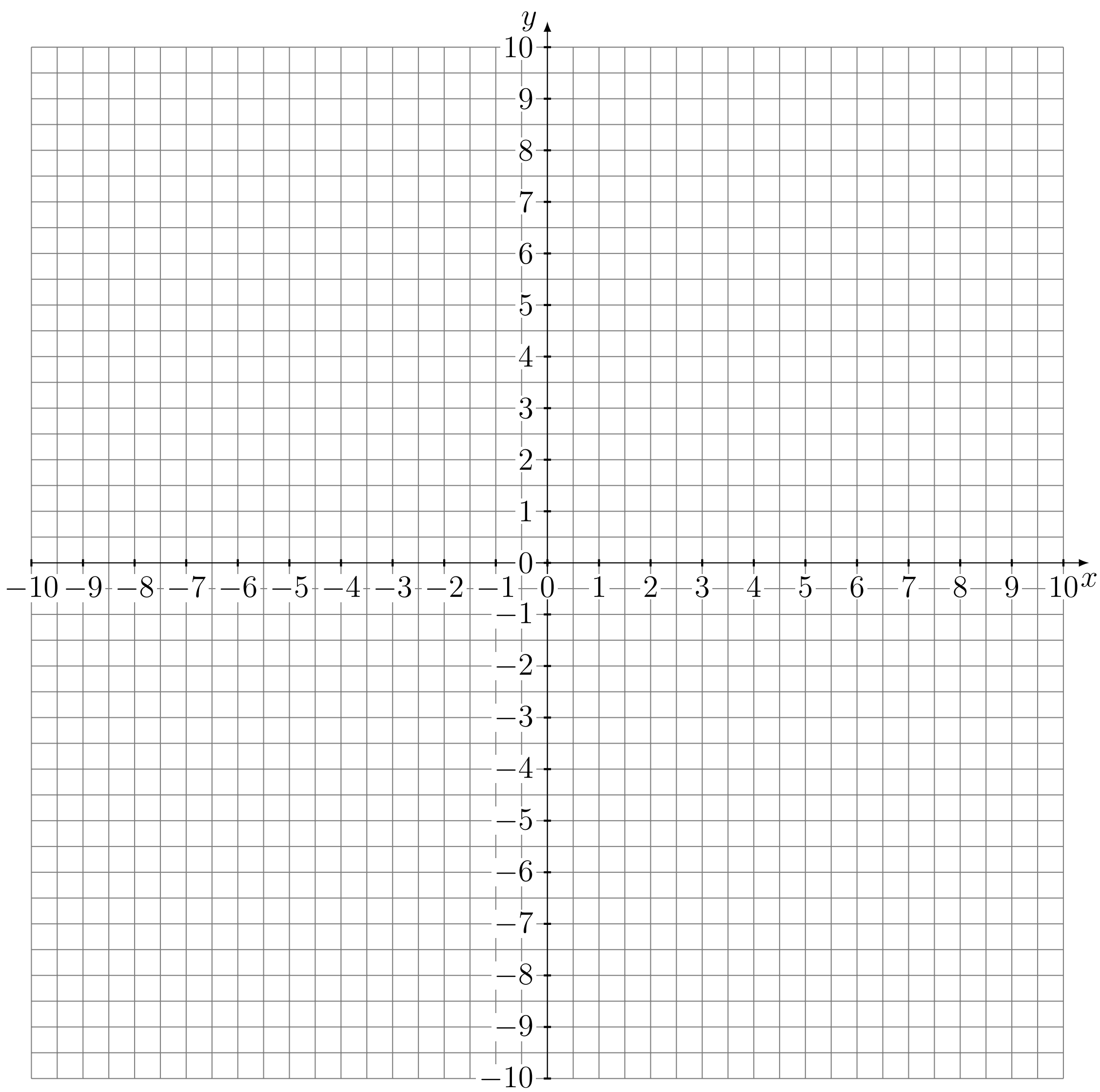


1 unit = 10 pixels

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()
```

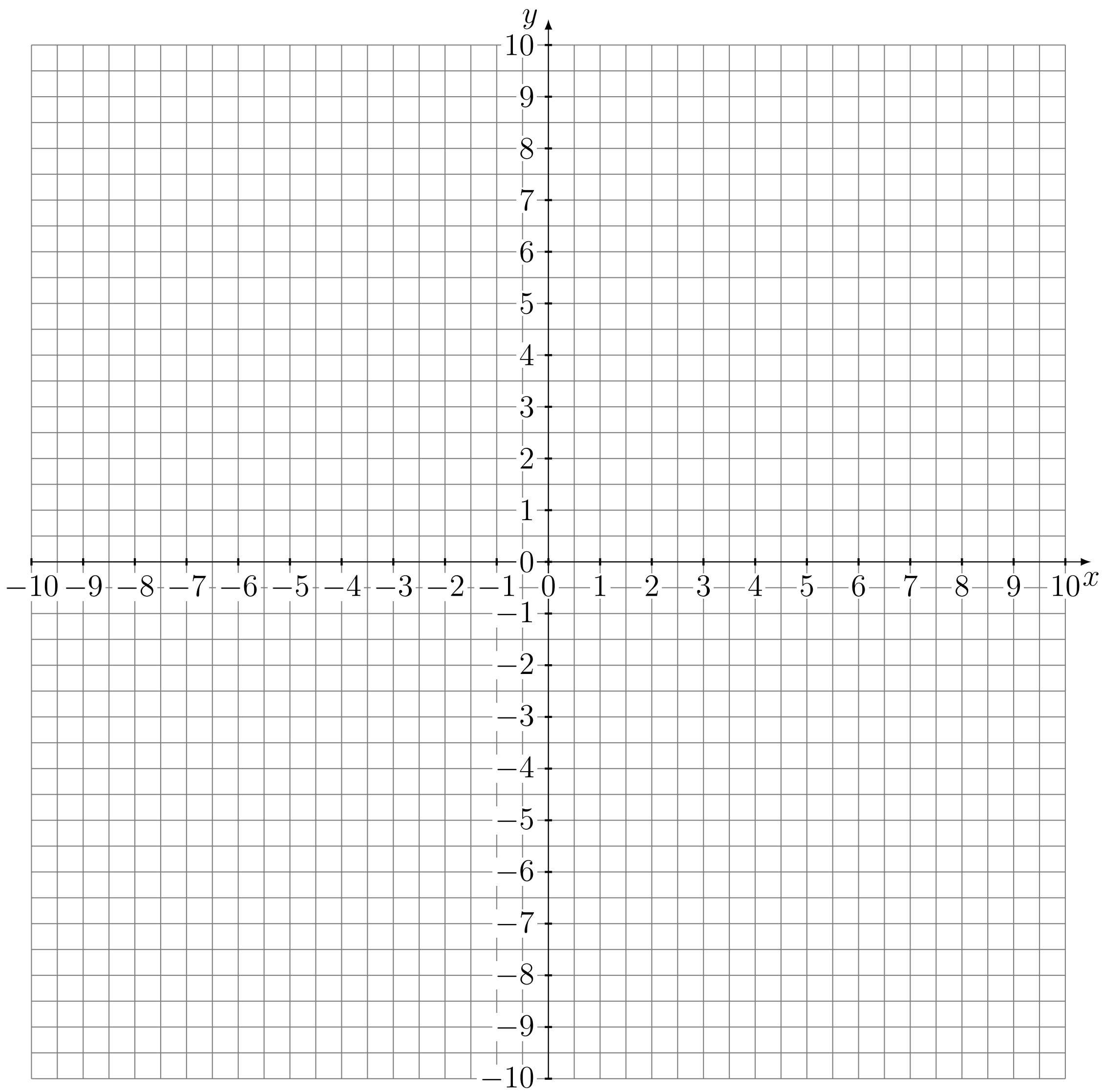


1 unit = 10 pixels

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?

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:

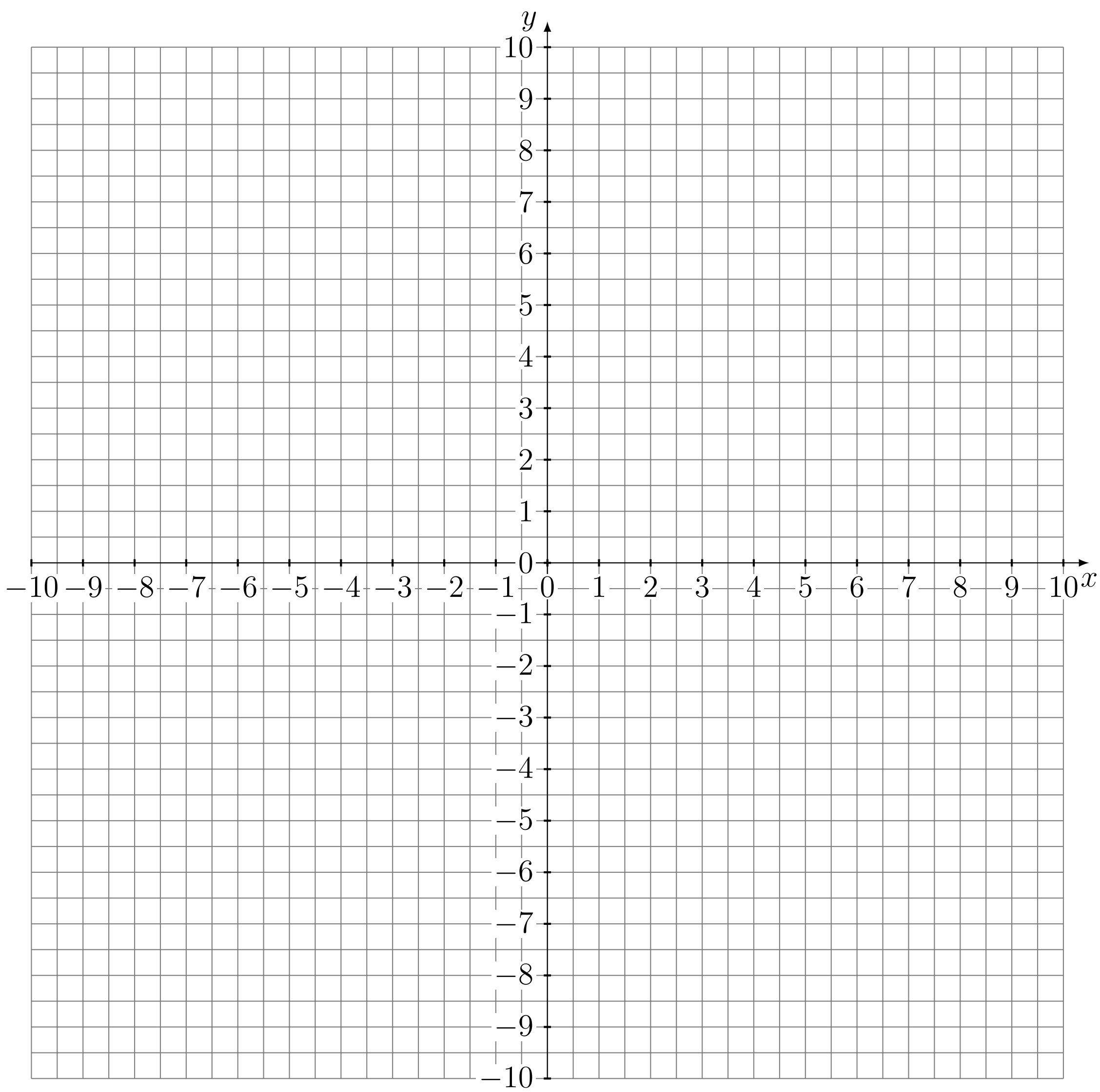
```
colors = ["blue", "red", "purple", "yellow", "black"]
for x in range(11):
    print(colors[x%5])
```

Output:

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

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!

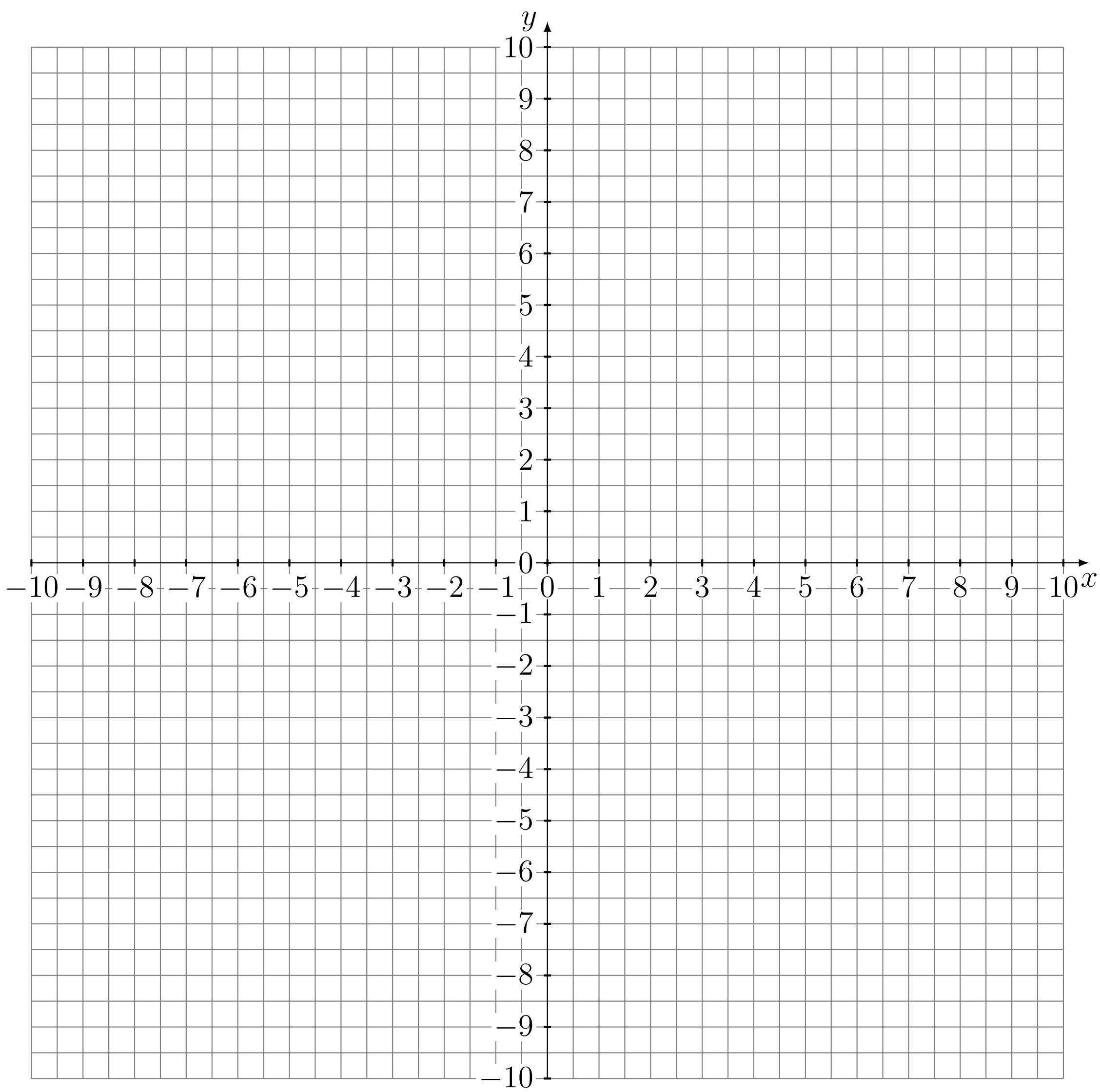
```
import turtle
t = turtle.Pen()
colors = ["green", "brown", "purple", "red", "blue"]
for x in range(10):
     t.pencolor(colors[x%5])
     t.forward(60)
     t.left(76)
turtle.done()
```



1 unit = 10 pixels

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!

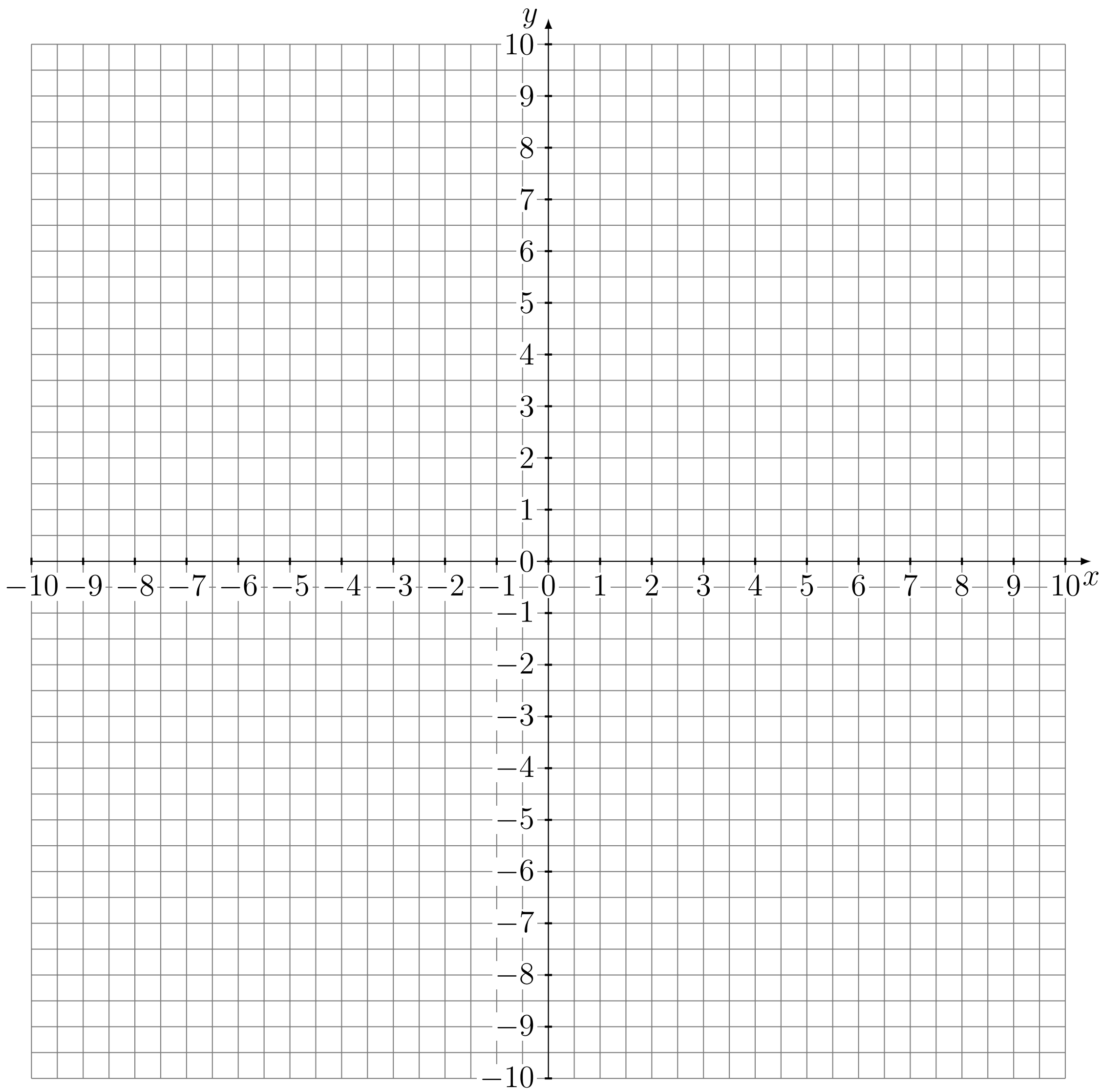
```
import turtle
t = turtle.Pen()
colors = ["green", "brown", "purple", "red", "blue"]
for x in range(10):
     t.pencolor(colors[x%5])
     t.forward(60)
     t.left(76)
turtle.done()
```



1 unit = 10 pixels

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()
```

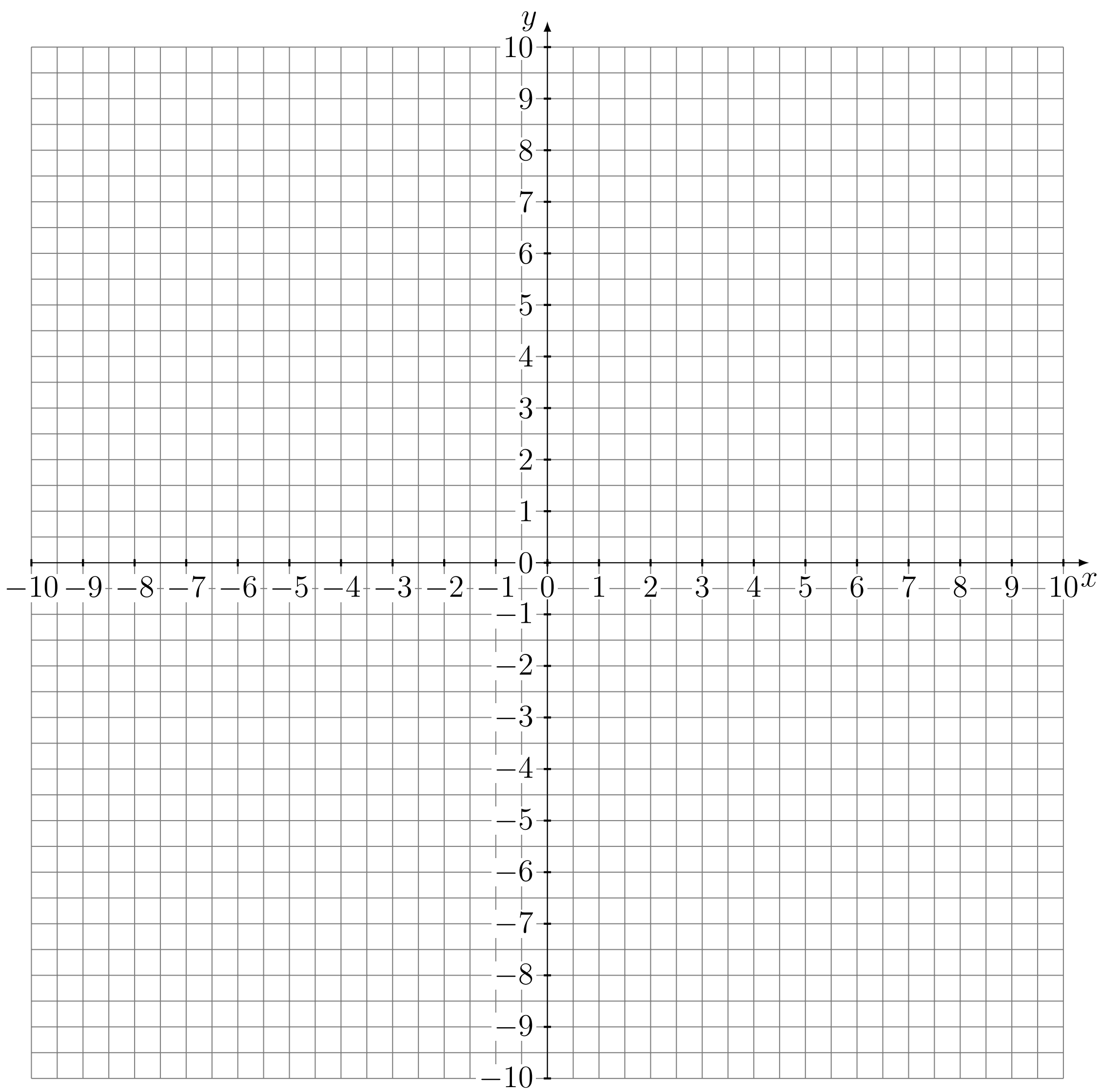


1 unit = 10 pixels

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!

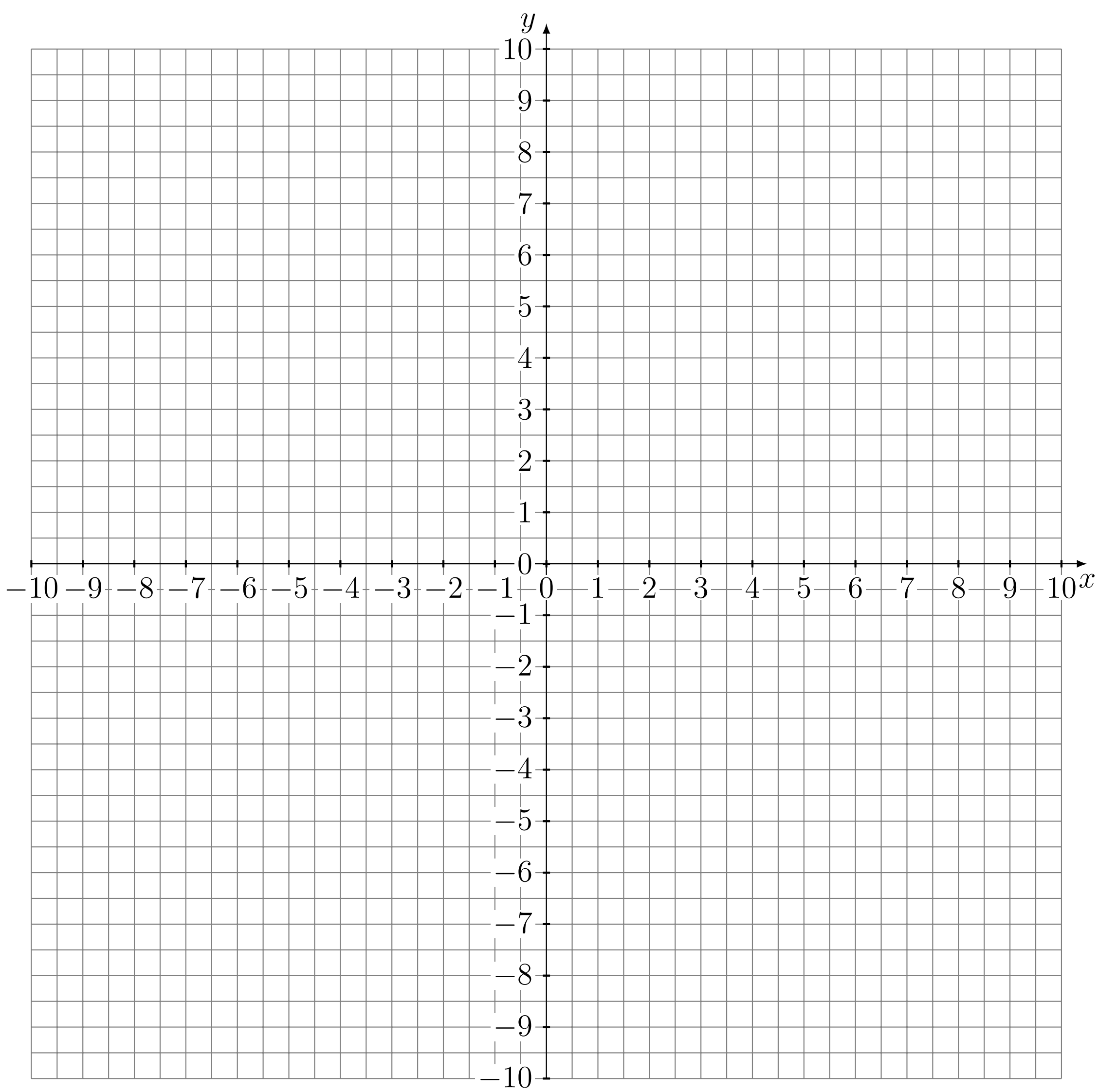
```
import turtle
# <-- the hash symbol is one way to write comments in Python.
# Anything on the same line as the hash is ignored!
t = turtle.Pen()
colors = ["green", "brown", "purple", "red", "blue"]
for x in range(20):
     t.pencolor(colors[x%5])
     t.forward(2*x+10)
     t.right(29)
turtle.done()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



1 unit = 10 pixels

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?

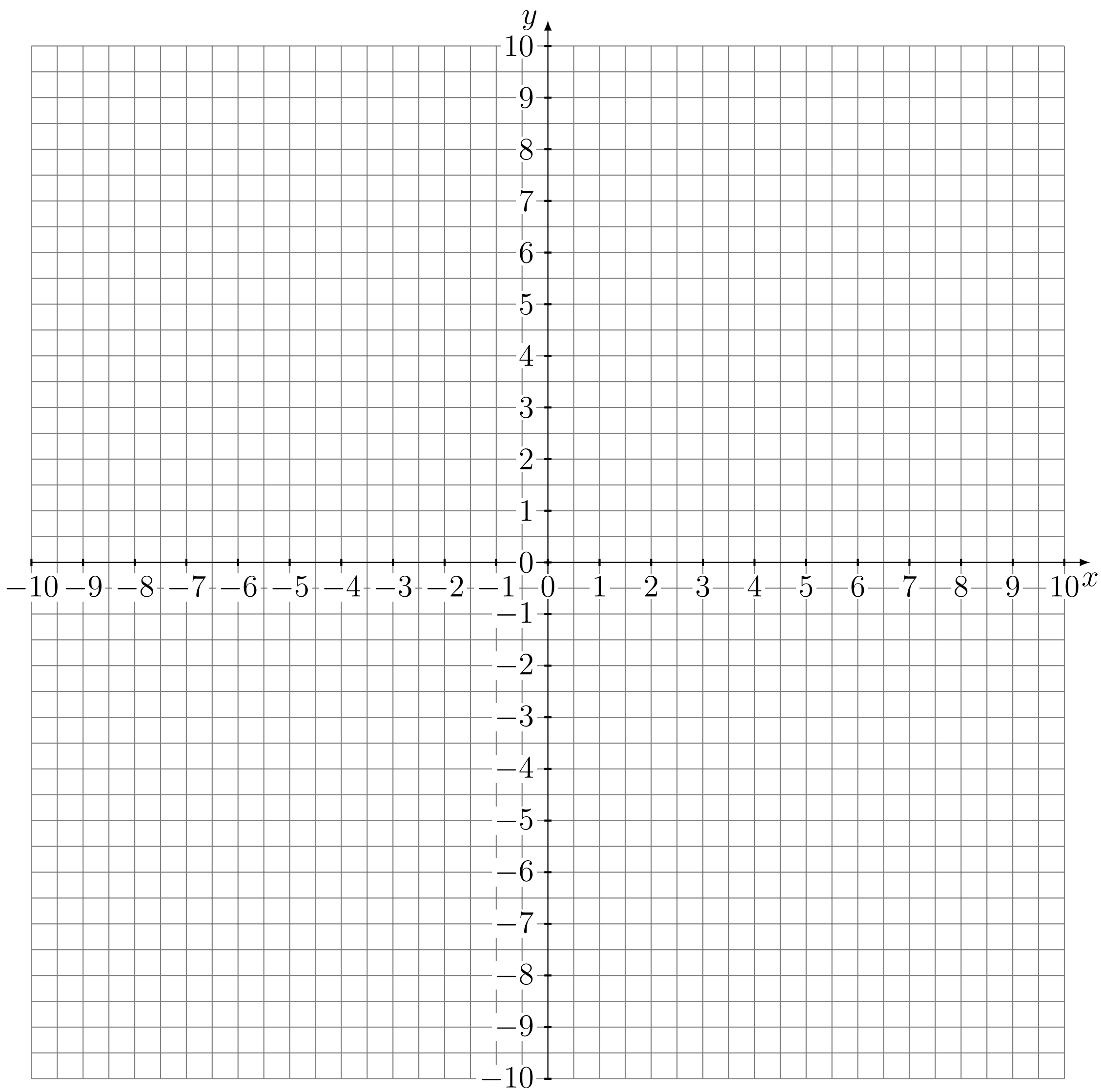
```
import turtle
t = turtle.Pen()
colors = ["green", "blue", "purple", "red"]
for x in range(100):
     t.pencolor(colors[x%4])
     t.forward(x)
     t.right(91)
turtle.done()
```

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!

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

Python Turtle - drawing.5.1.py

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()
```

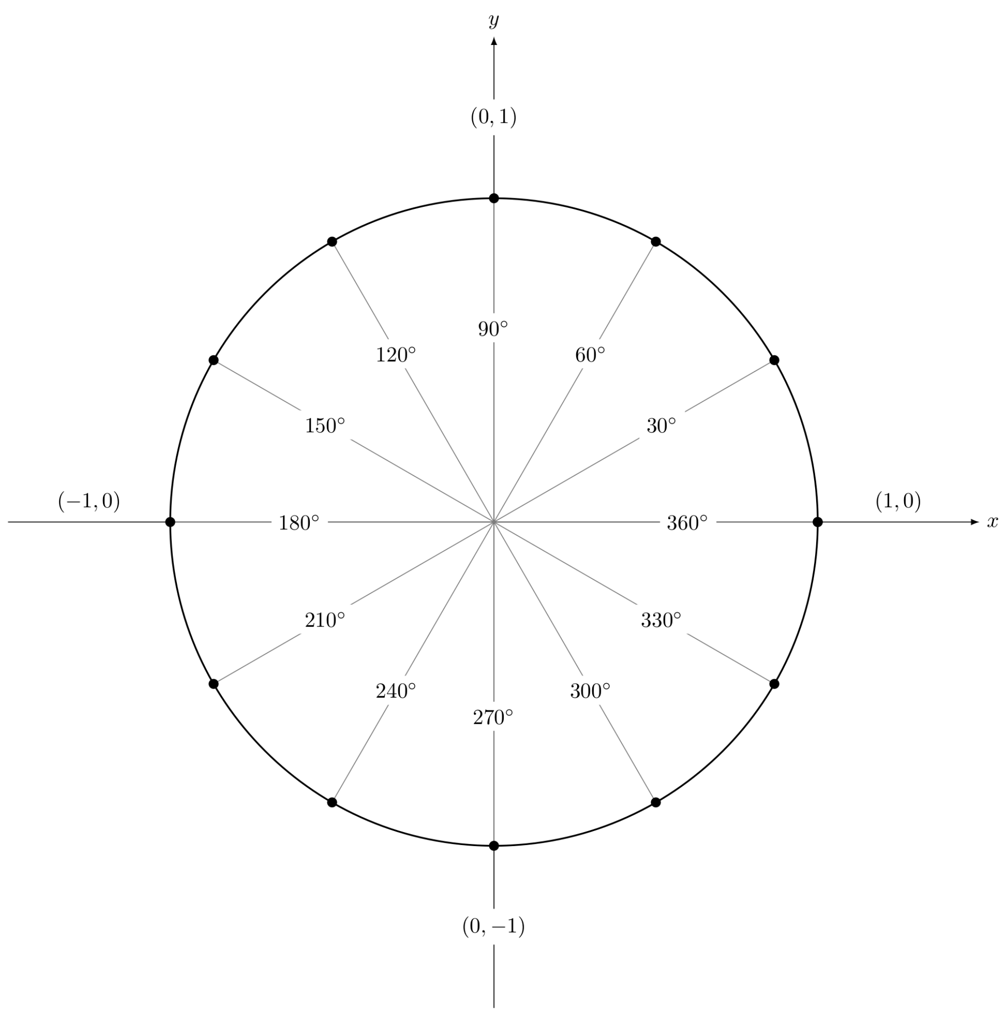
Describe what you see. What angle does the pen turn at each step?

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 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()
```