Getting Started with DrawBot

Created by Andy Clymer

https://learn.adafruit.com/getting-started-with-drawbot

Last updated on 2024-03-08 03:15:40 PM EST

Table of Contents

What is DrawBot 3

- Download

Hello World! 4

Python, the very basics 6

- String
- Numbers
- Variable names
- More Basic Math
- Lists
- And more

Basic Shapes and Colors 10

Python in DrawBot vs CircuitPython 14

Looping to make a more complex image 15

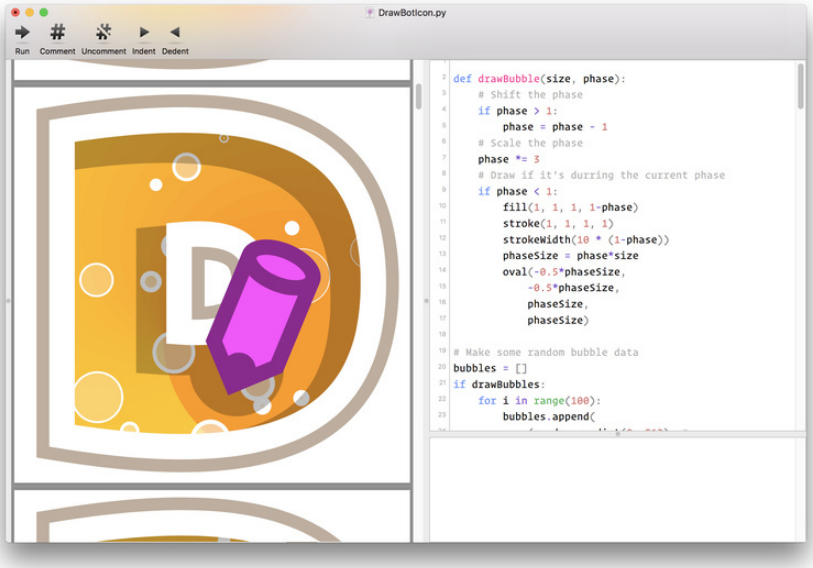
Saving Images 25

Text and multi-page PDFs 26

Animation 29

Learning More 34

What is DrawBot



DrawBot (https://adafru.it/Dkv) is a free MacOS application that makes it easy to use the Python programming language to draw two-dimensional graphics. DrawBot was conceived by typeface designer Just van Rossum as a tool to teach the Python programming language to his design students, and in the last five years or so DrawBot's development has been pushed forward by Frederik Berlaen, an alumnus of Just's course. If Just van Rossum's name sounds familiar, it's no coincidence — his brother Guido van Rossum authored the Python programming language!

Three are several popular environments for writing code to make images, Processing (https://adafru.it/ddm) being a prime example. But as a graphic designer, something that really excites me about DrawBot is that it puts a real emphasis into enabling high quality typography. That and it natively draws vector art that can be saved out as a PDF.

While it's not particularly suited for responding to physical interaction or interfacing with your hardware project, it really excels as a pro-quality design tool. With DrawBot, you can write simple Python scripts to draw vector art into a PDF, save bitmap images in several formats, and even make animated GIFs and movies.

And still, even with its powerful capabilities, DrawBot was still created to be one of the easiest ways to get started with the Python programming language.

If you aim is to learn Python to write code for one of the many CircuitPython boards, the Python language fundamentals that you'll learn by drawing in DrawBot can be applied later to your CircuitPython hardware projects.

Download

DrawBot can be downloaded for free at www.drawbot.com (https://adafru.it/Dkv).

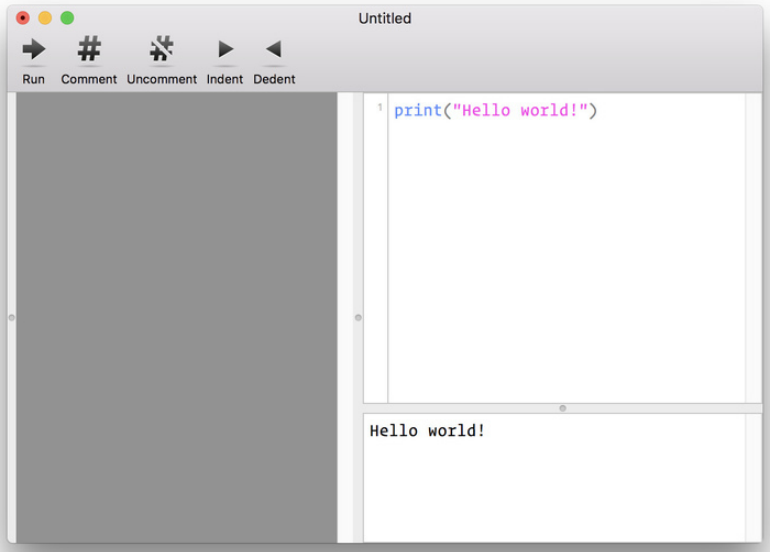
Hello World!

No programming lesson is complete without starting with a "Hello World" example.

Once you've downloaded and launched DrawBot, you'll be welcomed with a three-paned window. The top right side of the window is where your code will be written, and the gray left half of the window is where your drawing will show up. The bottom right of the window is reserved for any text output from your code — the text won't make it into the final drawing, but it's always useful to print messages to yourself in the code as you go. Feel free to resize the window or the individual panes to give your code and image all the room it needs.

In the code editing area of DrawBot's window, let's start programming:

```
print("Hello world!")
```



You'll notice that the code editor helpfully applies color to your code — the `print` function is in blue, and the text within quotes (otherwise known as a **string** in Python) shows up in magenta.

To run this program, either click the **Run** button at the top left of the window, or type `Command-R` from the keyboard. If all goes well, you should see some text show up in the output area below the code editor.

DrawBot does so much more than simply print out text, let's see if we can make an image. The image drawing area is solid gray — that's because we haven't even specified the size of the canvas that we want to draw into. Add this line of code to your program and run it:

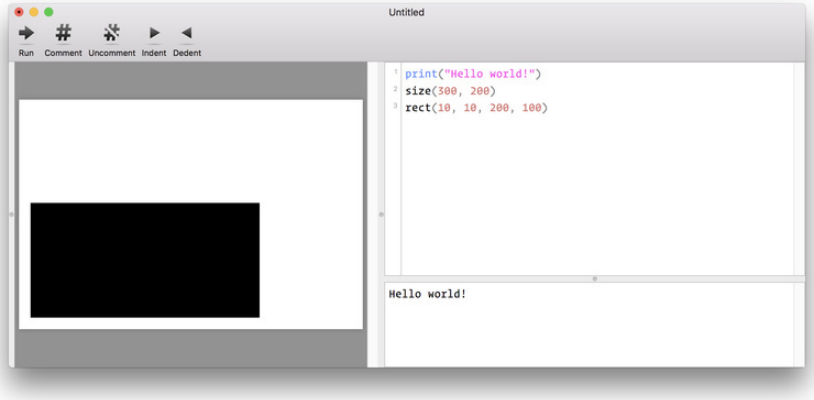
```
print("Hello world!")
size(300, 200)
```

And the canvas appears, 300 points wide by 200 points tall.

If it looks larger than 300 by 200 pixels, that's because you'll start out by drawing vector art into an infinitely scalable canvas without the limitations for screen resolution — right now the canvas is set to scale as large as it can be in the window, but when you save the image later you'll see it at its final size in pixels.

But now that we have a canvas to draw into let's add one more building block for now —

```
print("Hello world!")
size(300, 200)
rect(10, 10, 200, 100)
```



A rectangle appears! So what's really happening here?

Your first program in DrawBot is three lines long, each line containing what is known in Python as a **function**. A function is a command, which sometimes needs some additional information from us (but not always), which we provide as **arguments** in between a set of parenthesis `()`.

In the first line, you called the `print()` function, and provided it with the text that you wanted to print in quotes.

You then called the `size()` function in the second line to set the canvas size, followed by a width and height for the canvas. `size(width, height)`

Finally, you gave DrawBot the command to draw a `rect`, or a **rectangle**, along with four numbers for positioning: the horizontal starting position from the left side of the canvas, the vertical starting position starting from the bottom of the canvas, and then the width and height of the rectangle. `rect(x, y, width, height)`

Go ahead and try changing the numbers and run the script again — as long as you always set two numbers for `size()` and four numbers for `rect()` it should work just fine and you will see the rectangle move around and change size.

Before we get *too* far, let's cover some real Python basics in the next section.

Python, the very basics

You already learned what a function is in the previous section, it's a command that you call by name, which sometimes needs some additional information to be provided between its parenthesis `()`.

Let's step back even further and talk about some of the Python language basic data types and a few fundamentals of how the Python language works.

This page will only serve as an extremely limited introduction to the Python language, and really just enough to be able to finish getting an introduction to DrawBot. I would encourage you to check out the resources for getting started with Python (https://adafru.it/Dkz) from Python's own documentation.

String

In your `print()` statement, the phrase "Hello World" was enclosed in quotes. This text within quotes is known in Python as a **String**.

Strings can be surrounded by double quotes or single quotes

```
print("Double quotes")
print('or single quotes are okay')
```

This is useful if you ever need to have quote marks within your string, this code will give you an error:

```
print("I think "DrawBot" is great")
```

You can even see by the syntax coloring in the code editor that something is wrong here — it really thinks there's a string `"I think "` and another string `" is great"` but the word `DrawBot` is left out of the two strings. So, just switch the outside quotes to be single quotes and you'll be fine —

```
print('I think "DrawBot" is great')
```

Numbers

When setting the size of the canvas, you gave the `size()` function two numbers separated by a comma, with no quote marks. Numbers need to work differently than strings, even though sometimes they might look the same. Try this example:

```
print("300")
print(300)
```

The first line printed a string of the characters `300`, the second line printed the number. The quote marks from the string were removed for printing which made it

look like a number, but it might be more clear what's going on under the hood if we try some basic math:

```
print("300 + 200")
print(300 + 200)
```

Maybe this helps make things more clear, the first line printed `300 + 200` because the text of the string represents those 9 characters, but the second line actually added the two numbers together before printing. Very handy! And a little bit more on this later.

Variable names

Aside from the function names, the other data that we're providing (the strings and the numbers) might need to be reused in several places within your script, or their values might need to changes as the script runs. To make this easy, you can assign a **variable name** of your choosing to these pieces of data. Try this:

```
myName = "Andy"
age = 38
```

Go ahead and run this script — nothing should be output. All we've done is assigned `myName` to be equal to `"Andy"` and `age` to be equal to `38`. Now, any time in the script that we use these names their equivalent values will be used instead:

```
myName = "Andy"
age = 38
print(myName)
print(age)
```

Let's make things more complicated, add more text to a `print` statement by separating the items with commas:

```
myName = "Andy"
age = 38
print("My name is", myName)
print("and I am", age)
```

Remember though, we can do some basic math with that number. Add one more line:

```
myName = "Andy"
age = 38
print("My name is", myName)
print("and I am", age)
print("Next year I will be", age + 1)
```

Nice!

More Basic Math

So, numbers can be added, but all of the other basic math you would expect can be applied to them as well:

```
print(2 + 5)
print(100 - 1)
print(3 * 2)
print(10 / 2)
```

It's worth noting that in all of my examples so far, I've added spaces around the mathematical operators, but in Python this white space is optional, or you can use as much as you like. So really, these two lines are the same:

```
print(2 + 5)
print(2+5)
print(2   +   5)
```

You can do some basic math with strings too, which might sound crazy at first but kind of makes sense:

```
print("Draw" + "Bot")
print("Hello!" * 5)
```

It's no problem to add two strings together to combine them into one, or even multiply a string to repeat it. However, it's not possible to divide or subtract strings (because that's just crazy).

Lists

Two more basic data types for now, along with some useful things you can do with them, and then we'll get back to the drawing.

In Python, a **List** is an ordered sequence of items that you would call out by their index order, and a **Dictionary** is an unordered group of items that you would call out by their **keyword** name.

This example list is full of names:

```
classNames = ["Limor", "Frank", "Erica"]
prit(classNames)
```

Printing the list gives all of the items back to you. It might be useful to ask for the items in order:

```
className = ["Limor", "Frank", "Erica"]
print(className[1])
```

The important thing here is within the print statement, `className[1]`. In this case, we asked for only item `1` in the `className` list. You might be surprised to see `Frank` print out instead of `Limor`, that's because counting in Python (and most programming languages) starts from zero.

Are you ready for a big jump? If we have a list, we can use a `for` statement to fetch one item at a time out of the list and do something with it.

```
className = ["Limor", "Frank", "Erica"]
for name in classNames:
    print(name)
    print(name * 5)
```

Like the command says, for each name in `classNames`, take the name and do something. And that something is really every line that's tabbed in after the `for` loop starts.

First, it prints the name one time, then it prints the name 5 times before moving on to the next name in the list. In this example, `name` is another variable name that we made up, we could have called it `person` and this code would run identically as long as we update all of the tabbed in lines with the new variable name.

```
classNames = ["Limor", "Frank", "Erica"]
for person in classNames:
    print(person)
    print(person * 5)
```

And more

There's a lot more to Python than just what I've covered here, but if you're comfortable with this as a starting point we'll take some new concepts as they come in the next few sections.

Basic Shapes and Colors

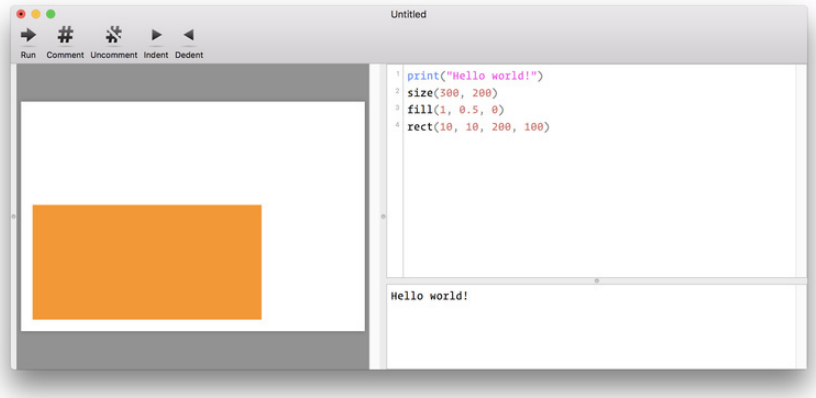
Now that a very basic overview of Python is out of the way, let's get back to some drawing!

I'll return to the code that we started earlier, our rectangle on a 300 x 200 point canvas:

```
print("Hello world!")
size(300, 200)
rect(10, 10, 200, 100)
```

Let's continue where this left off. It could be nice to add a little bit of color. Before issuing the command to draw a rectangle, first set the `fill` color.

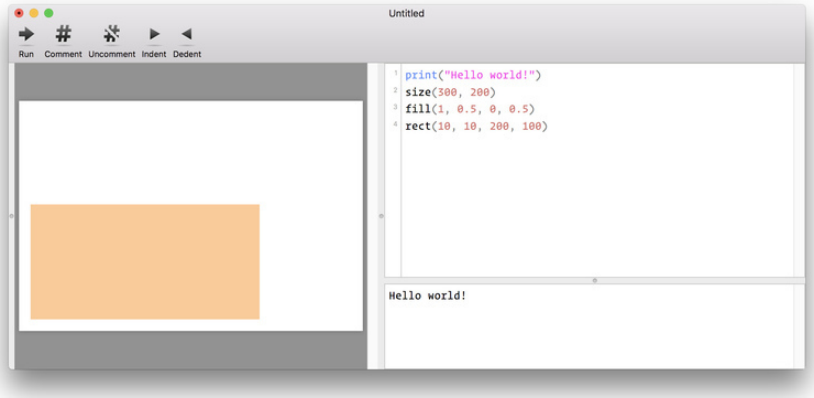
```
print("Hello world!")
size(300, 200)
fill(1, 0.5, 0)
rect(10, 10, 200, 100)
```



Run it and see what happens — the rectangle is now orange. The `fill` color took three arguments, in this case the values for red, green and blue. Setting red to `1` means red is 100%, green is `0.5` which really means 50%, and blue is at 0%. Go ahead and try some more color values, always keeping between `0` and `1`.

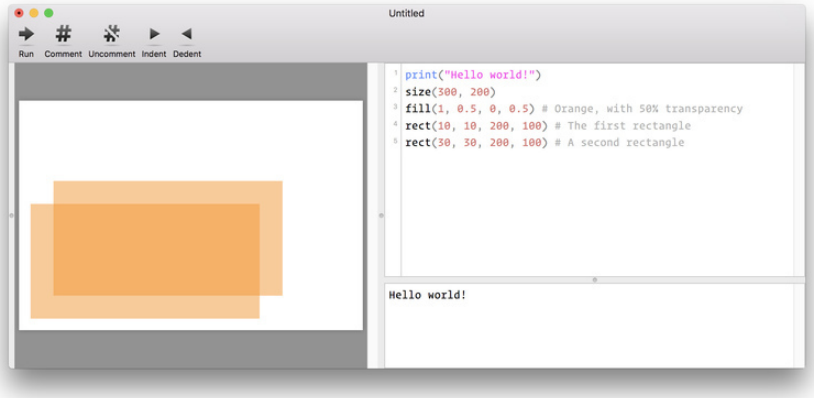
There's still more that `fill` can do, as an example, adding a fourth value also sets the transparency of the color:

```
print("Hello world!")
size(300, 200)
fill(1, 0.5, 0, 0.5)
rect(10, 10, 200, 100)
```



It might be hard to tell that the rectangle is 50% transparent, until you draw another rectangle that overlaps this one:

```
print("Hello world!")
size(300, 200)
fill(1, 0.5, 0, 0.5) # Orange, with 50% transparency
rect(10, 10, 200, 100) # The first rectangle
rect(30, 30, 200, 100) # A second rectangle
```



I'm also introducing another helpful Python concept here, leave yourself some comments! Anything written in a line after the # will be ignored by Python and is only there to help you understand what your code is doing. From here on out, feel free to leave notes to yourself right there in your code.

If you want the color of the second rectangle to be different, just be sure to set the color before drawing the rectangle

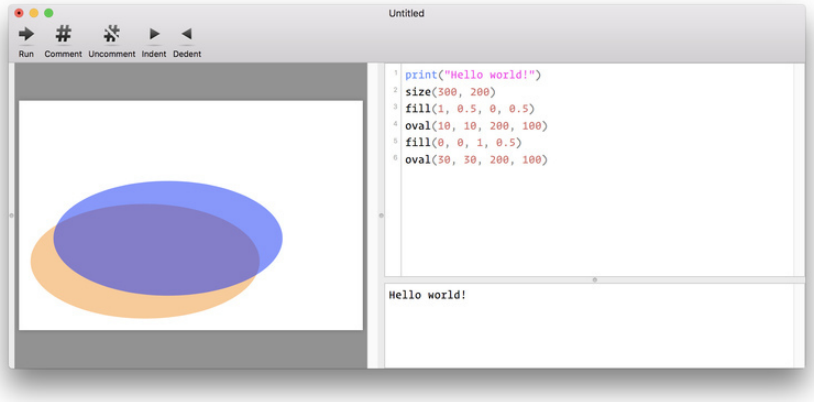
```
print("Hello world!")
size(300, 200)
```

```
fill(1, 0.5, 0, 0.5) # Orange, with 50% transparency
rect(10, 10, 200, 100) # The first rectangle
fill(0, 0, 1, 0.5) # Blue, with 50% transparency
rect(30, 30, 200, 100) # A second rectangle
```

As a refresher, the `rect` needed four values from us, two that specify where the lower left corner of the rectangle starts, and two more for the width and height of the rectangle.

Try changing `rect` to `oval` and run the script again:

```
print("Hello world!")
size(300, 200)
fill(1, 0.5, 0, 0.5)
oval(10, 10, 200, 100)
fill(0, 0, 1, 0.5)
oval(30, 30, 200, 100)
```



Nice!

Instead of going through all of DrawBot's shapes and color options here in this guide, I'd like for you to be familiar with DrawBot's own documentation. Continue experimenting with the shapes, and `fill` and `stroke` settings that you find in DrawBot's documentation and Quick Reference Guide.

Python in DrawBot vs CircuitPython

If you're reading this, guide you're probably already very familiar with the products Adafruit offers for CircuitPython. I think it might be good to take a moment to explain what the difference is between the Python that you're writing in DrawBot and the Python that you would write for your CircuitPython hardware project.

In a nutshell — the language is the very same! But, the vocabulary can be a little bit different.

Think of it this way, whether you're speaking with your tailor or your butcher you might be using the English language, but you'll need to know to ask for a `Filet Mignon` from one and ask for a `French Cuff` from the other. The language for the rest of the sentence you use is still the same, but some of the terms will need be a specific to the trade. Mix the two up and neither will know what you're asking for!

The same goes for using Python in DrawBot and using CircuitPython, the language is the same but DrawBot won't understand the kinds of function names that are specific to CircuitPython (DrawBot has never heard of `DigitalInOut` or a `pin.direction`), and the same goes the other way (CircuitPython would have no way to draw a `rect` or set a `fill` color).

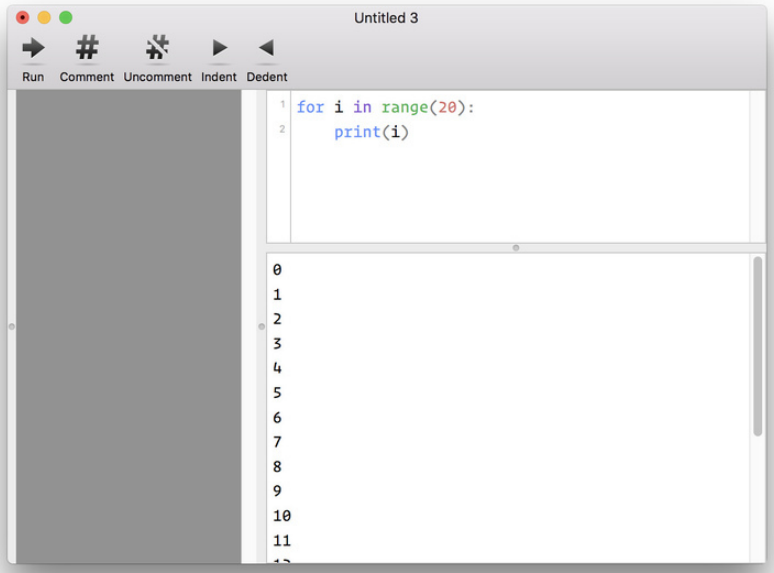
However, being comfortable with how Python works right here in DrawBot will put you many steps ahead for getting started with using Python in a completely different context. The fact that they both use `variable names`, `strings`, `lists`, `numbers`, `for` loops and all the other mechanics of the language will be essentially the same in both environments.

Looping to make a more complex image

The thing about computers is they love repetitive tasks. If you wanted to draw a grid of 100 rectangles you could write 100 lines of code, each line drawing a `rect()` with a slightly different position. But that's the hard way, we can get by with only a few lines of code if we have Python repeat the command for us.

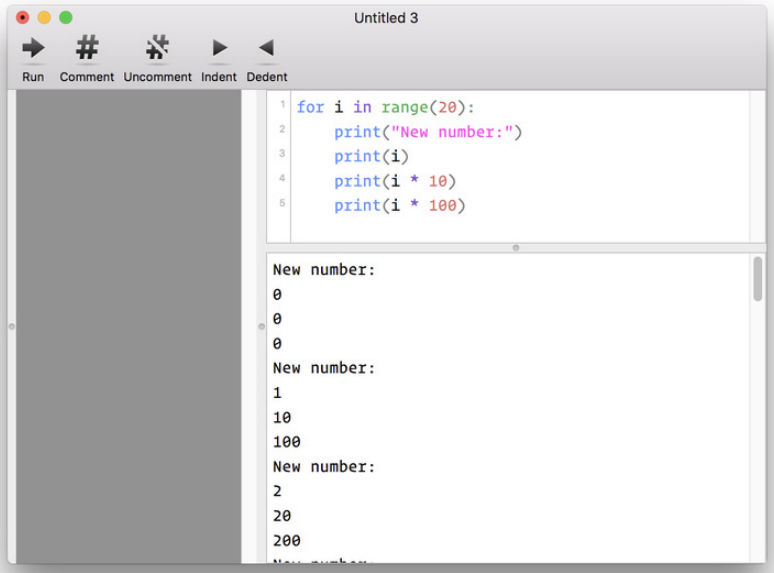
The easiest way to have Python repeat itself is to first get it counting. Python has a `range()` function that only exists to make a list of numbers in a particular range. Let's ask Python to bulid a range of 20 numbers, `range(20)`, which we'll pick through one at a time.

```
for i in range(20):
  print(i)
```



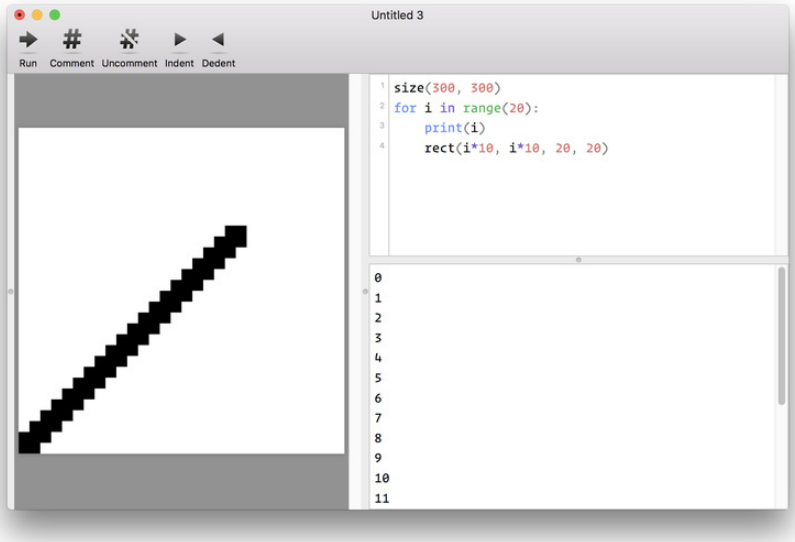
You'll remember that we did something similar with a `for` loop in a previous section of this guide, where we had a list of names and picked one name out at a time. This example is really similar, Python built a range of numbers, and we fetched and printed each one, one at a time. Go ahead and add a few more lines each time it fetches a number from the range:

```
for i in range(20):
  print("New number:")
  print(i)
  print(i * 20)
  print(i * 100)
```



If you can print a sequential number 20 times, try to draw a `rect()` instead using this number as a variable

```
size(300, 300)
for i in range(20):
  print(i)
  rect(i*10, i*10, 20, 20)
```



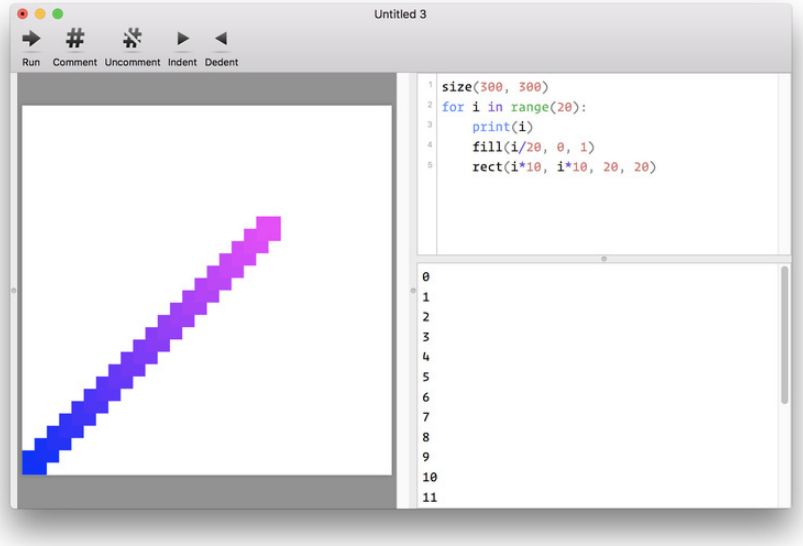
With each new number, the same line of code is being drawn, `rect(i*10, i*10, 20, 20)`, but the variable `i` changes each time.

Remember that the four numbers in the `rect()` function are for horizontal position, vertical position, width and height. So, the first rectangle is drawn at position `0*10,`

`0*10`, which computes to `0, 0`. The second rectangle is drawn starting at `1*10, 1*10`, or `10, 10`, and so on.

Try adding a `fill()` color that uses the variable `i` to see the color change with each new step:

```
size(300, 300)
for i in range(20):
  print(i)
  fill(i/20, 0, 1)
  rect(i*10, i*10, 20, 20)
```

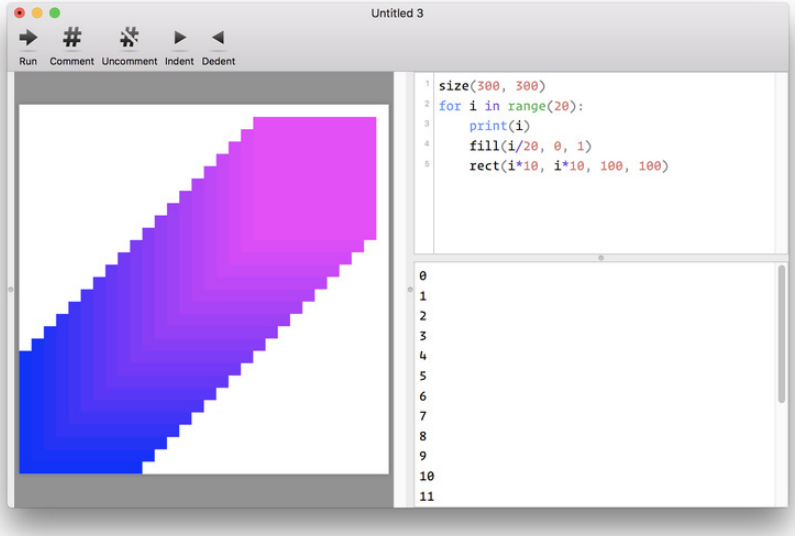


Each rectangle is 100% of the blue component, 0% of the green component, but has some fractional percentage of red each step of the way.

The fun part about having this code written is you can make small changes to the variables to get quick variations on the same drawing.

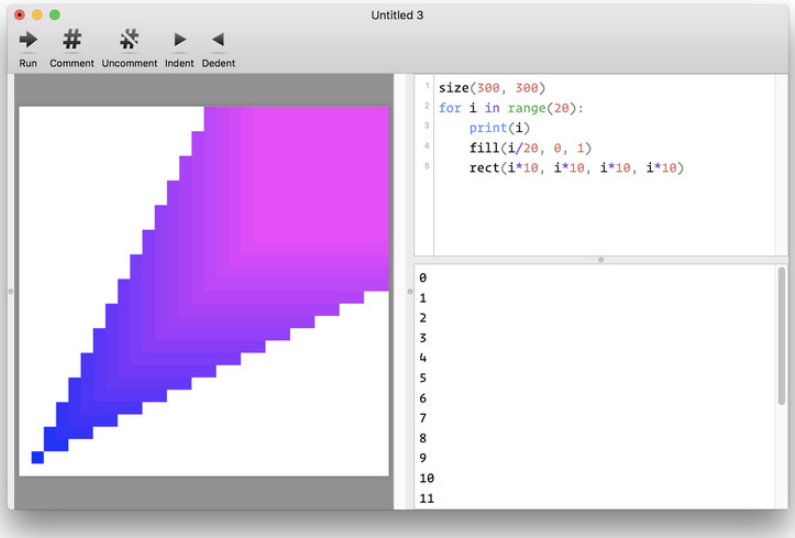
For example, changing the width and height values of the rectangle to `100, 100`:

```
size(300, 300)
for i in range(20):
  print(i)
  fill(i/20, 0, 1)
  rect(i*10, i*10, 100, 100)
```



Or, make the width and height be based on the number taken out of the range:

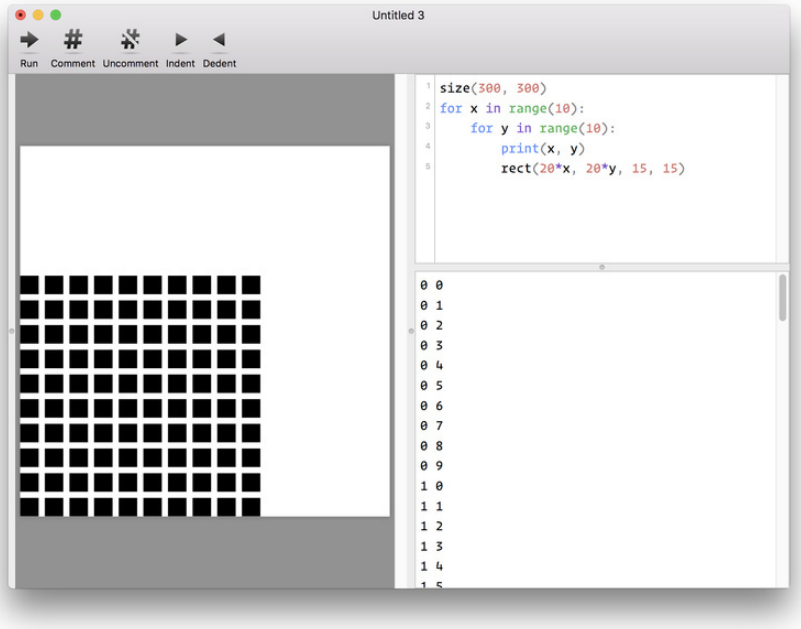
```
size(300, 300)
for i in range(20):
  print(i)
  fill(i/20, 0, 1)
  rect(i*10, i*10, i*10, i*10)
```



The first rectangle is actually `0, 0` in width and height, so it didn't even show up! But You can see where this is going, it can be a lot of fun to build ot a very simple program of only a few lines, and then spend time tweaking the numbers to get something surprising and different than what you had in mind.

Let's go one step more complex, I'll use one `for` loop within another loop to get two numbers for the horizontal and vertical positioning of the rectangles.

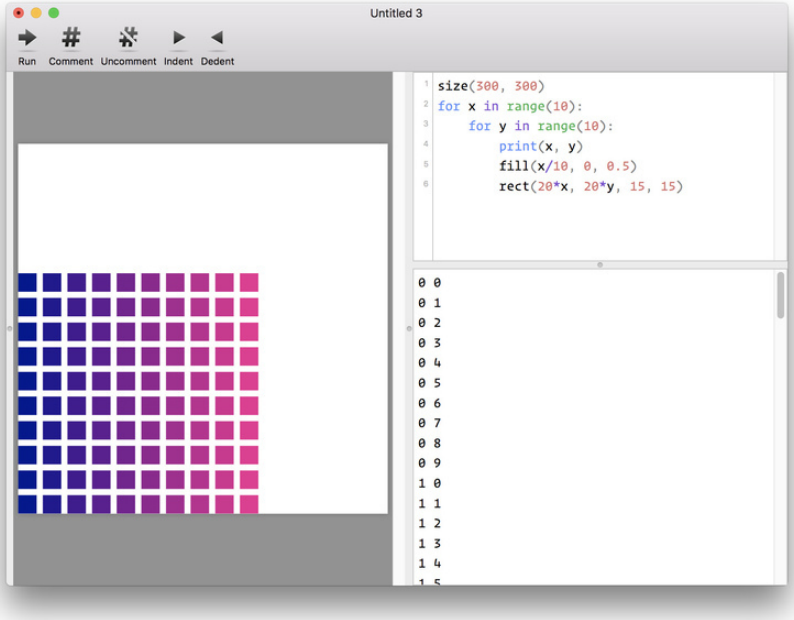
```
size(300, 300)
for x in range(10):
  for y in range(10):
    print(x, y)
    rect(20*x, 20*y, 15, 15)
```



We have two number generators, each making numbers in a range of `0` to `9` (for a total of 10 numbers in the range). You can see the numbers it's choosing with the `print` statement — the first time through `x` is equal to `0` for as long as it takes the second counter to count through all ten numbers. Then, the `x` steps up to `1` and the loop continues.

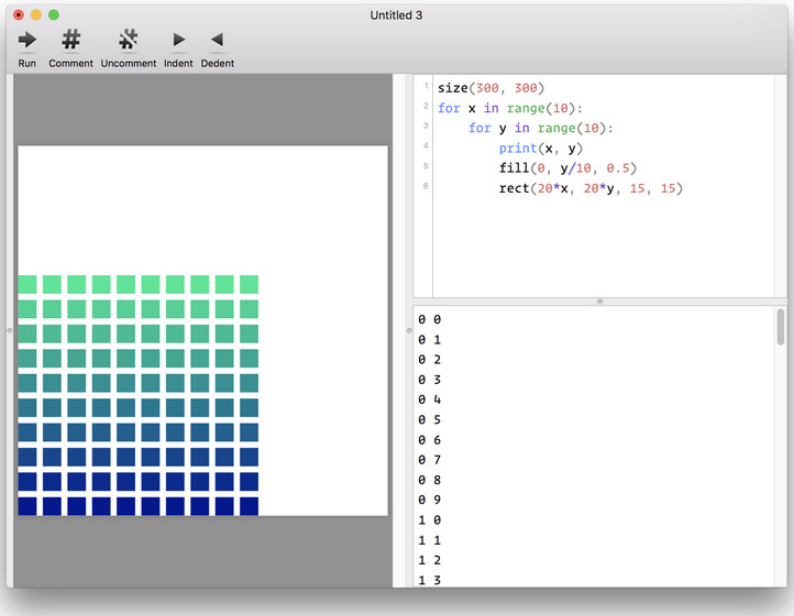
Go ahead and set a `fill()` color that also uses these `x` and `y` values, for example make the rectangles more red as the `x` value goes up:

```
size(300, 300)
for x in range(10):
  for y in range(10):
    print(x, y)
    fill(x/10, 0, 0.5)
    rect(20*x, 20*y, 15, 15)
```



...or more green as the y value goes up, which might help illustrate what's happening:

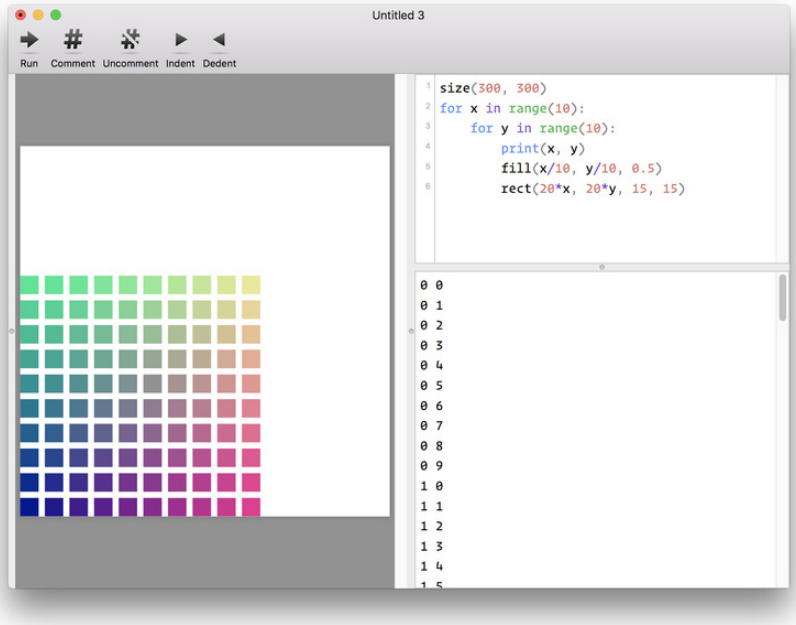
```
size(300, 300)
for x in range(10):
  for y in range(10):
    print(x, y)
    fill(0, y/10, 0.5)
    rect(20*x, 20*y, 15, 15)
```



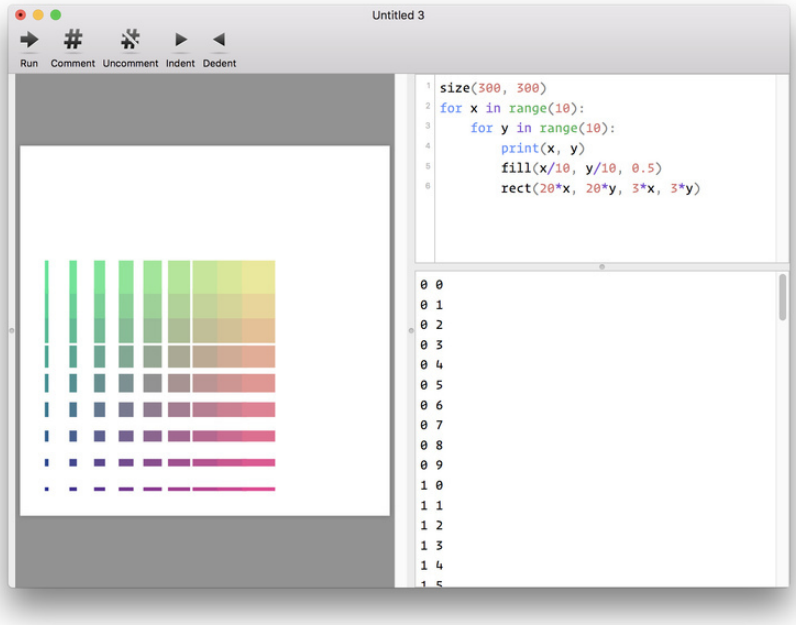
Or a little bit of both!

```
size(300, 300)
for x in range(10):
```

```
for y in range(10):
  print(x, y)
  fill(x/10, y/10, 0.5)
  rect(20*x, 20*y, 15, 15)
```



Nice! But don't stop there, keep messing around with the numbers in your code and see what happens:

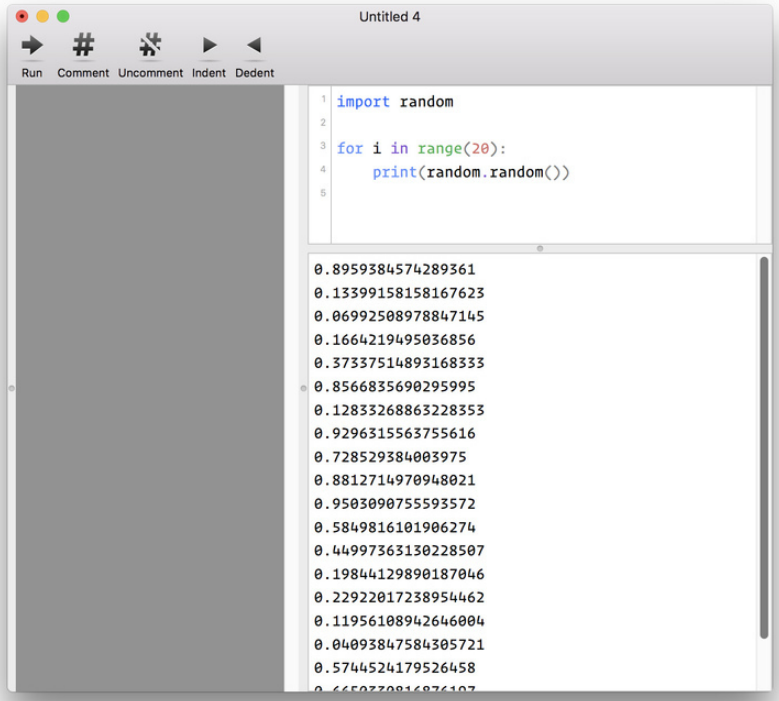


It can be fun to introduce some randomness into your code.

To have Python generate random numbers, first `import random` at the top of your script, which enables you to use several different random functions. We'll use the

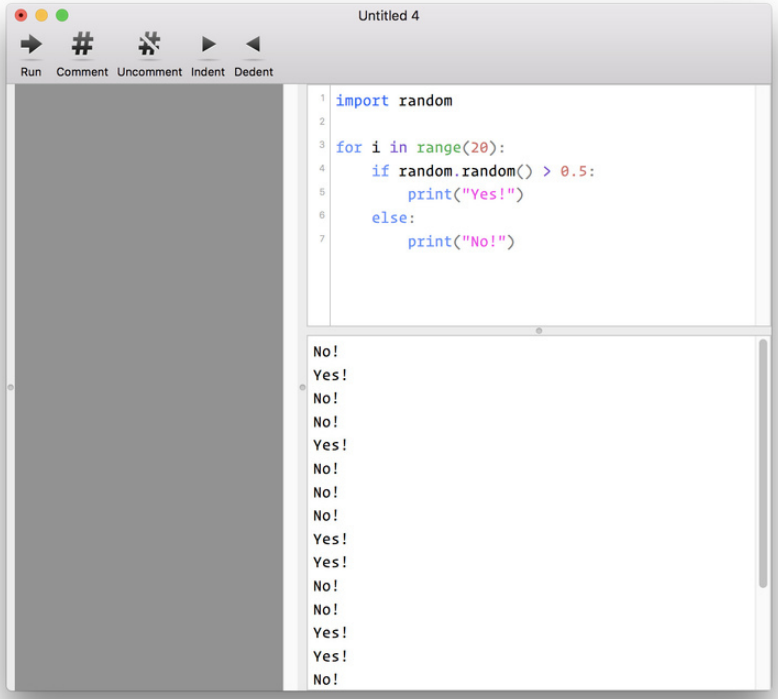
`random.random()` function which generates a random number between `0` and `1`. Try printing some random numbers:

```
import random

for i in range(20):
  print(random.random())
```



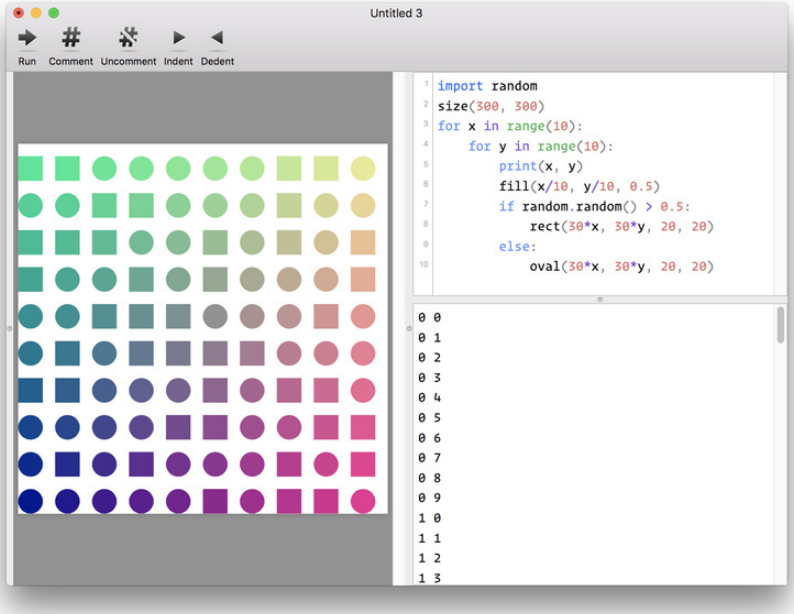
Each time through, the random number is never less than `0` and never greater than `1`. This can be really useful, for instance if we introduce a new concept of doing one thing when this random value is greater than `0.5` and doing something else when the value was less:

```
import random

for i in range(20):
  if random.random() &gt; 0.5:
    print("Yes!")
  else:
    print("No!")
```



Half of the time one thing happens, and the other half of the time another thing happens. Let's bring this concept back into our drawing code, try drawing a `rect()` half of the time and an `oval()` the other half:

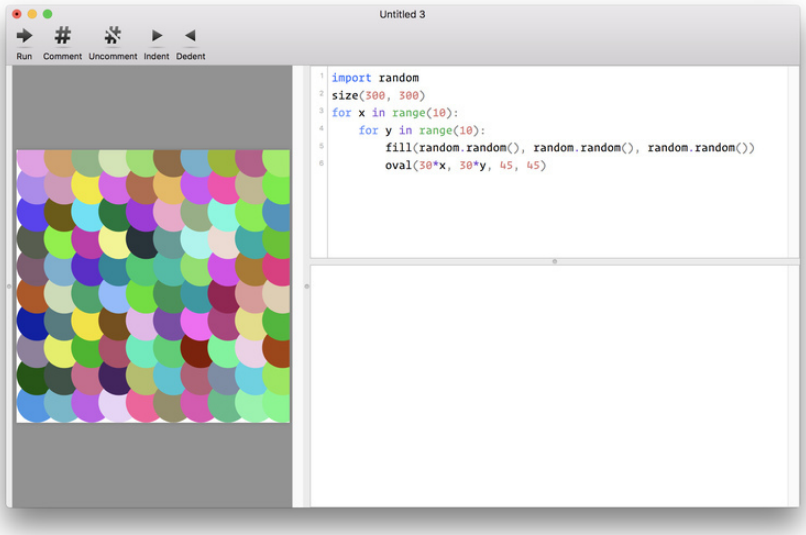
```
import random
size(300, 300)
for x in range(10):
  for y in range(10):
    print(x, y)
    fill(x/10, y/10, 0.5)
    if random.random() &gt; 0.5:
      rect(30*x, 30*y, 20, 20)
    else:
      oval(30*x, 30*y, 20, 20)
```



Keep running the code over and over to see it change, since the random numbers will be different each time you run the script.

These random numbers between `0` and `1` can be very useful, remember that the color values also need to be between `0` and `1`. Try asking for three unique random numbers for each color component:

```
import random
size(300, 300)
for x in range(10):
  for y in range(10):
    fill(random.random(), random.random(), random.random())
    oval(30*x, 30*y, 45, 45)
```



Saving Images

If you haven't saved anything, it could be a good idea to go ahead and do it now. From the **File** menu you'll find **Save As**, where you'll be prompted to find a location to save the Python script that you're writing. There's a second menu option for **Save PDF** to save the image you're drawing in the PDF format.

If you'd like to save your image in a different format, you can really easily do this directly in your code. Adding a line with a call to the function `saveImage()` with a path name will let you save your image in one of 10 formats.

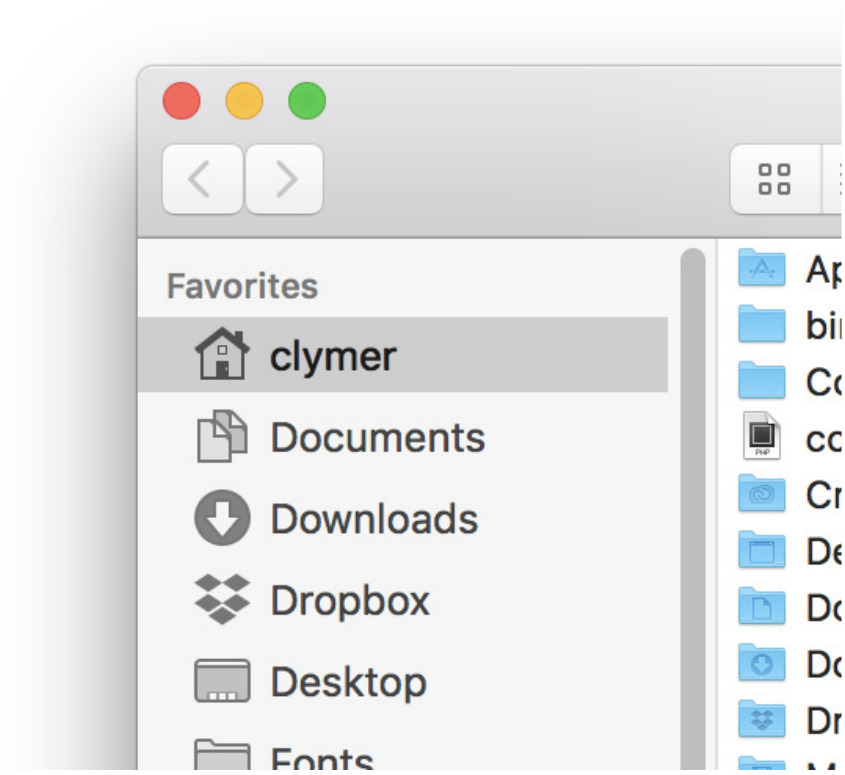
```
saveImage("/path/to/image.jpg")
```

Be sure that the path name in quotes doesn't already exist (or DrawBot will overwrite the file that's there!)

Since the folder names on your computer and my computer are different you might need to pay attention to the code examples that I provide from here on out — I'll be saving images to the Desktop folder with this path name:

```
saveImage("/Users/USERNAME/Desktop/myImage.pdf")
```

Simply replace the `USERNAME` with your own user name, and the image will save as a file directly to your Desktop each time you run the script. You can find your user name at the top of the sidebar in a new Finder window, for example my user name is `clymer`.



If you copy one of my code examples and forget to change the `USERNAME`, DrawBot will give you an error.

Text and multi-page PDFs

As mentioned in the introduction, DrawBot has some pro-level features for fine typography including supporting advanced OpenType features, but it also doesn't take much to get started with drawing text.

A single line of text is as quick as setting a few attributes (color, font and its size) and then setting a line of `text`.

```
size(300, 300) # Document size

font("Times-Bold", 100) # Times-Italic at 100 points in size
fill(0.5, 0, 1) # Purple: 50% red, 100% blue
text("Hello!", (10, 10)) # Type set a string, at a location
```

The order is still important — set the `font` and the `fill` color before drawing the text, otherwise the text wouldn't have been set with these attributes.

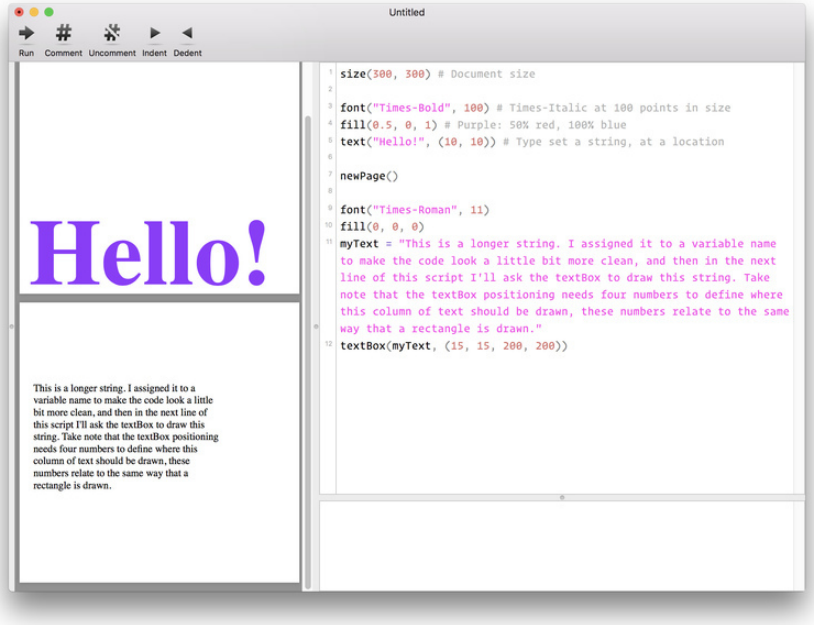
The `text()` function will draw a string starting at a particular location, in this example the bottom left corner of the text starts 10 pixels from the left side of the canvas and 10 units from the bottom of the canvas.

For longer stretches of text that you would rather have drawn into a column you can use the `textBox()` function. In this case, since the string of the text is so long, I've assigned it to a variable name to make the code look a little bit neater. Then, the `textBox()` will draw the string assigned to the variable name.

```
size(300, 300) # Document size

font("Times-Bold", 100) # Times-Italic at 100 points in size
fill(0.5, 0, 1) # Purple: 50% red, 100% blue
text("Hello!", (10, 10)) # Type set a string, at a location

font("Times-Roman", 11)
fill(0, 0, 0)
myText = "This is a longer string. I assigned it to a variable name to make the
code look a little bit more clean, and then in the next line of this script I'll
ask the textBox to draw this string. Take note that the textBox positioning needs
four numbers to define where this column of text should be drawn, these numbers
relate to the same way that a rectangle is drawn."
textBox(myText, (15, 15, 200, 200))
```

You're not limited to drawing into a single canvas, I'll add one more line half way through the code to tell DrawBot to start a new page before drawing the paragraph of text.

```
    size(300, 300) # Document size

font("Times-Bold", 100) # Times-Italic at 100 points in size
fill(0.5, 0, 1) # Purple: 50% red, 100% blue
text("Hello!", (10, 10)) # Type set a string, at a location

newPage()

font("Times-Roman", 11)
fill(0, 0, 0)
myText = "This is a longer string. I assigned it to a variable name to make the
code look a little bit more clean, and then in the next line of this script I'll
ask the textBox to draw this string. Take note that the textBox positioning needs
four numbers to define where this column of text should be drawn, these numbers
relate to the same way that a rectangle is drawn."
textBox(myText, (15, 15, 200, 200))
```



If you were to save this as a PDF you'll find that it will save as a single document with two pages. This `newPage()` function is a small but powerful building block, with a few more elements (okay, many more elements) it could be possible to design and lay out something even as complex as a book or magazine.

Animation

The `newPage()` function that we used in the previous example is used to create a new blank page of a PDF, but the great thing about DrawBot is that each new page of a PDF is really a new frame of an animation if you save your image is a `gif` or `mov` file!

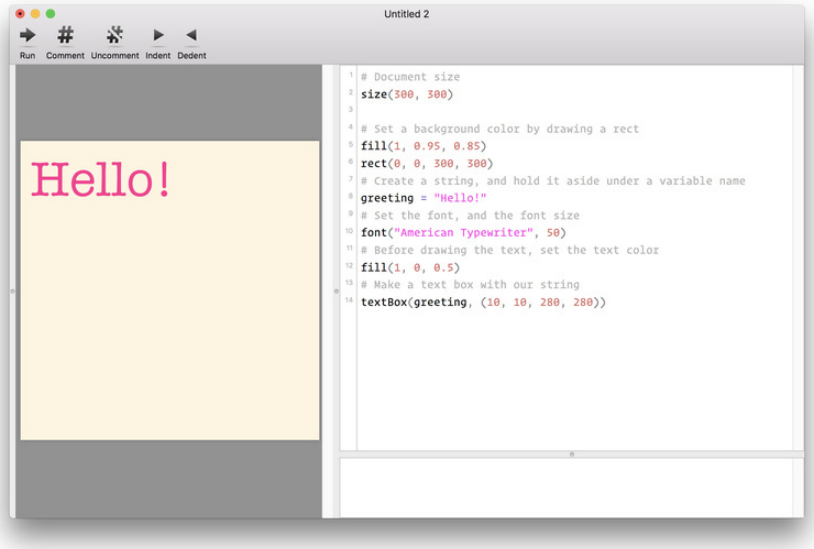
DrawBot will show each frame of an animation as a separte page in its drawing area, to see the animation move you will need to tell DrawBot to save the image.

We'll jump ahead into some parts of Python that a developer with a little bit more experience would already be comfortable with (and if you're not yet, go through those resources linked to at the end of the Python Basics page!)

Building off of the previous text example, let's start by choosing a font, and filling a text box with a string.

```
# Document size
size(300, 300)

# Set a background color by drawing a rect
fill(1, 0.95, 0.85)
rect(0, 0, 300, 300)
# Create a string, and hold it aside under a variable name
greeting = "Hello!"
```

```
# Set the font, and the font size
font("American Typewriter", 50)
# Before drawing the text, set the text color
fill(1, 0, 0.5)
# Make a text box with our string
textBox(greeting, (10, 10, 280, 280))
```



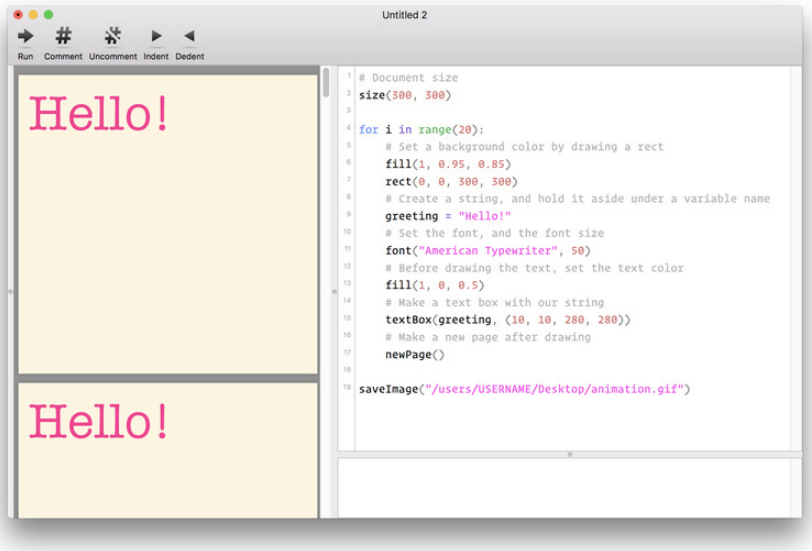
Nothing all that exciting yet, but it's a start.

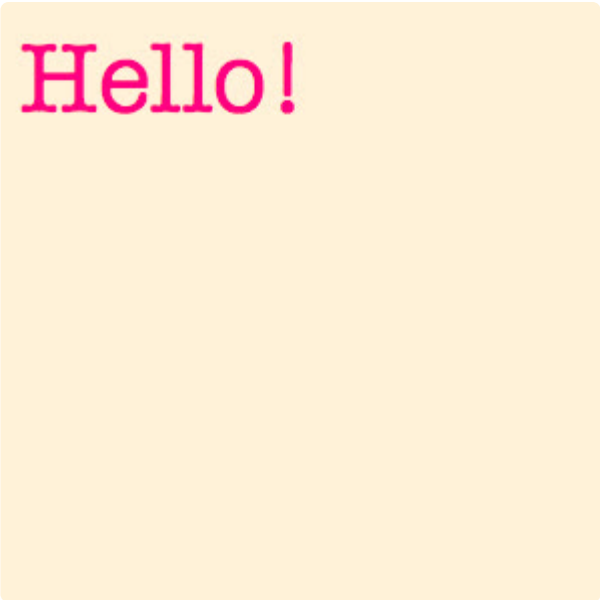
Let's take another lesson from a previous example, using a `for` loop to repeat some code.

We'll have the loop count through `range(20)`, meaning that the same code will be repeated 20 times, and with each time through we'll end our instructions with `newPage()` to tell DrawBot to prepare a new page for the next loop to draw into.

```
# Document size
size(300, 300)

for i in range(20):
    # Set a background color by drawing a rect
    fill(1, 0.95, 0.85)
    rect(0, 0, 300, 300)
    # Create a string, and hold it aside under a variable name
    greeting = "Hello!"
    # Set the font, and the font size
    font("American Typewriter", 50)
    # Before drawing the text, set the text color
    fill(1, 0, 0.5)
    # Make a text box with our string
    textBox(greeting, (10, 10, 280, 280))
    # Make a new page after drawing
    newPage()

saveImage("/users/USERNAME/Desktop/animation.gif")
```

Just remember to change the USERNAME to your own user name, otherwise DrawBot will give you an error.





Great! If all went well, you'll see a sequence of the frames of the animation off to the left of the window, and an animated gif should have saved to your desktop. You can open the gif in a web browser, or select it and tap the space bar to open it in **Quick Look** to see it in motion.

But there's no motion yet! We've drawn 20 of the same frame. You will however notice that there's one completely blank frame at the end of the animation — our code set up a new blank page at the end of each loop, which means that the last time through it made a blank page but drew nothing into it.

I'll make a small change in two places, first I'll write a line of code to only make a `newPage()` when the condition `if i < 19` is True. Remember, each time the `for` loop fetches a number out of the `range(20)` it gets assigned to the variable name `i` so we can have the code keep track of which frame it is on by watching this variable.

To make things move, how about we also subtract `i` from the font size. This means that in the first frame the font will be `50 - 0` points in size (since counting starts from zero), the next frame will be `50 - 1` in size, and so on.

```
# Document size
size(300, 300)

for i in range(20):
    # Set a background color by drawing a rect
    fill(1, 0.95, 0.85)
    rect(0, 0, 300, 300)
    # Create a string, and hold it aside under a variable name
    greeting = "Hello!"
    # Set the font, and the font size
    font("American Typewriter", 50 - i)
    # Before drawing the text, set the text color
    fill(1, 0, 0.5)
    # Make a text box with our string
    textBox(greeting, (10, 10, 280, 280))
    # Make a new page after drawing
    # unless if this is the last frame
    if i &lt; 19:
        newPage()

saveImage("/users/USERNAME/Desktop/animation.gif")
```

Now it's moving!

Keep changing other variables, remember that it's fun and easy to add, subtract, multiply and divide numbers by `i` so that they move according to the frame number.

Let's multiply the string by `i` so that it repeats the greeting:

```
# Document size
size(300, 300)

for i in range(20):
    # Set a background color by drawing a rect
    fill(1, 0.95, 0.85)
    rect(0, 0, 300, 300)
    # Create a string, and hold it aside under a variable name
    greeting = "Hello!" * i
    # Set the font, and the font size
    font("American Typewriter", 50 - i)
    # Before drawing the text, set the text color
    fill(1, 0, 0.5)
    # Make a text box with our string
    textBox(greeting, (10, 10, 280, 280))
    # Make a new page after drawing
    # unless if this is the last frame
    if i &lt; 19:
        newPage()

saveImage("/users/USERNAME/Desktop/animation.gif")
```

With what you learned here, go back to an earlier example and see if you can get it to move. Keep messing with the variables and see what happens!

```
size(300, 300)

for frame in range(20):
    # Background color
    fill(0.9, 1, 1)
    rect(0, 0, 300, 300)
    # Draw a bunch of ovals
    for i in range(frame):
      fill(i/20, frame/40, 1)
      size = i/ frame * 100
      oval(i*10, i*10, size, size)
    newPage()
```

```
saveImage("/users/USERNAME/Desktop/animation.gif")
```

Learning More

Now that you've had an introduction to writing Python in DrawBot, you're ready to continue on with the official examples from DrawBot's website.

Start by copying and pasting lines from the DrawBot Quick Reference into your code and see what happens. Then, dig deeper into the other sections that you find along the left side of the website, such as the documentation for drawing shapes, using color, and formatting text.

In the mean time, here's a little bit more inspiration to keep you going:

- I've written a small overview about DrawBot's icon, the full source code for the icon is available too
- Keep an eye on the @DrawBotApp on Twitter to see what people have been making
- Just van Rossum shares his animation studies on his Daily DrawBot blog
- David Jonathan Ross used DrawBot to make conference badges
- The amazing design for the Lowlands Festival by Hansje van Halem was created using DrawBot, uncluding this series of animations
- And it's not just for animation, Roberto Arista designs entire books with DrawBot and has his own guide for drawing with DrawBot

Have fun!
— Andy