Python Turtle Tutorial

David Wales, 2015

Introduction

This is a quick Python tutorial. By the end of this tutorial, you should understand the basics of variables, functions and loops.

We are going to use the Python turtle module, to make things a little bit more fun.

Python Interpreter

To run your Python programs, you're going to need a Python interpreter. If you are using Mac or Linux, you probably already have Python on your computer. To use Python, open a Terminal, and run the following command:

```
python
```

To run a command, type it, then press 'Enter' or 'Return'.

If everything is working correctly, you should now have a Python Interactive Prompt running in your terminal.

It should look something like this, although some of the numbers may be different:

```
Python 3.4.3 (default, May 17 2015, 18:27:22)
[GCC 4.2.1 Compatible Apple LLVM 6.1.0 (clang-602.0.49)] on darwin
Type "help", "copyright", "credits" or "license" for more
information.
>>>
```

The last line, `>>>` is called the prompt. This is where you type Python commands.

Now that you have Python running, test it out by typing the following into the interactive prompt:

```
2 + 2
```

Always remember to run Python before you start typing Python commands!

Setting up the turtle module

Before you can use the turtle module, you have to import it. There are several different ways to import modules, but in this tutorial, we are going to import it as follows:

```
from turtle import *
```

Go ahead and type that into the Python prompt!

Playing with our turtle

Before we learn about variables and functions and loops, let's just play with our turtle to get comfortable with it.

I'm going to assume you have already opened a Terminal, run Python, and imported the turtle module. If you haven't, go back to the previous sections, and come back to this section when you are ready!

OK. Let's get started with our turtle. You probably can't see your turtle yet, but it should appear after you tell it what to do.

The first step is to tell our turtle to move somewhere. We can do this using the `forward()` or `fd()` function. To make the turtle move 100 pixels, just type

```
fd(100)
```

You should see another window open. There should be a black line, with a triangle at one end. The triangle is your turtle.

You can make your turtle turn right or left by using the `rt()` and `lt()` functions.

Try drawing a square with the following code:

```
fd(100)
rt(90)
fd(100)
rt(90)
fd(100)
rt(90)
fd(100)
rt(90)
```

Here are a few more turtle functions that you might find useful.

The `home()` function moves the turtle back to the middle of the window.

The `clear()` function erases all the lines the turtle has drawn.

The `penup()` function stops the turtle from drawing lines when it moves.

The `pendown()` function makes the turtle start drawing lines again when it moves.

Play around with the turtle, until you feel that you are ready to move onto the next section.

Have fun!

Using variables

If you are going to use the same number more than once, it is often useful to use variables.

This will become particularly useful when we learn about loops and functions.

Let's create a variable called `distance`, which contains the value `100`:

```
distance = 100
```

That was easy!

Now, we can start using `distance` in our turtle code:

```
fd(distance)
```

The turtle should move 100 pixels.

But what if we change our mind? What if we want distance to mean `50`, instead of `100`?

To change the value of a variable, just assign it a different value:

```
distance = 50
```

Now, we can use the exact same code as before, but the turtle will only move 50 pixels!

```
fd(distance)
```

Loops

Remember how we wrote some code to draw a square?

It looked like this:

```
fd(100)
rt(90)
fd(100)
rt(90)
fd(100)
rt(90)
fd(100)
rt(90)
```

Notice how we keep typing the same thing over and over?

Loops let you type something once, and then repeat it as many times as you like.

To make a square, we have to do the following four times:

```
fd(100)
rt(90)
```

We can do this using a `for` loop:

```
for i in range(4):
    fd(100)
    rt(100)
```

Notice how the functions we are looping have been indented four spaces? Indentation is very important in Python. When you are typing this into the interactive prompt, be very careful to type exactly four spaces before `fd(100)` and `rt(100)`. This tells Python that you want those two functions to be inside the loop.

Also, notice the colon (`:`) at then end of the first line. This tells Python that we are starting a special block of code, like a loop or a function. Everything after a colon, which is indented four spaces is part of that block.

If you are using the interactive prompt, you will notice that the prompt changes from `>>>` to `...` when you are typing the indented loop code. To run a loop, you have to press enter twice, once you have finished typing it.

Functions

Now that we know how to make a square, let's make a function so that we can make easily make as many squares as we like.

We define a function like this:

```
def square(size):
    for i in range(4):
        fd(size)
        rt(90)
```

Let's break that down. Every function starts with `def`, which is short for 'define'. Then, we have the name of the function. In this case, the function's name is `square`. After this, we have some brackets. Every function has brackets. Then, we have the function's arguments. In this case, the function has one argument, which is the `size` we want our square to be.

When we run our function, it works just like the other turtle functions.

```
square(50)
```

This runs all the code in the `square` function, and it makes `size = 50`.

Notice that everything inside the function is indented four spaces. Everything inside the `for` loop is indented another four spaces, which means that everything inside the four loop is indented a total of 8 spaces.

Try making some squares of different sizes.

Just for fun, try running the following code. Try to guess what it does before you run it!

```
for i in range(10):
    square(100)
    rt(36)
```

Modules

It would be painful to have to type all our functions in every time we wanted to use them. Fortunately, Python lets us write them down in a text file, and `import` them when we want to use them.

Exit Python, by pressing `control-D`. Then navigate to where you want to store your Python modules. Finally, create a folder called 'python-modules' and navigate to it. You might try the following:

```
cd ~/Documents
mkdir python-modules
cd python-modules
```

Now, create a new text file using `nano`:

```
nano shape.py
```

You can write your programs here, then save them by pressing `control-O` You can exit by pressing `control-X`

Try typing the following program into shape.py:

```
from turtle import *

def square(size):
    for i in range(4):
        fd(size)
        rt(90)
```

Now, save the file and exit `nano` by typing `control-O` then `control-X`.

Now, run `python` again, in the same Terminal window.

Into the interactive prompt, type

```
from shape import *
square(50)
```

Hooray! Now you can make squares whenever you like, without needing to do all that boring typing every time!

Remember that whenever you want to use the shapes in this file, you need to navigate to the correct location in the terminal before you run Python. Otherwise the `import` statement won't be able to find your code to import.

e.g. For this shape module, you have to do the following in the terminal before running Python:

```
cd ~/Documents/python-modules
```

You can check to see what files are in a directory with the `ls` command. You should always be able to see your module before running `python`

Challenge

Make functions to draw triangles, pentagons and hexagons, and add them to the `shape.py` module.

HINT 1: The angle for a triangle is 60, for a pentagon is 72, and for a hexagon is 60.
HINT 2: Remember — You can edit the `shape.py` module by navigating to `~/Documents/python-modules`, and editing the `shape.py` file with `nano`:

```
cd ~/Documents/python-modules
nano shape.py
```

HINT 3: Remember you can save and exit `nano` by typing `control-O` then `control-X`
HINT 4: You can exit Python at any time by pressing `control-D`

Challenge 2

Make another function that uses the `square` and `triangle` functions to draw a house.

Challenge 3

Make a function that draws a fun pattern, by drawing a shape, and then turning slightly, and drawing the shape again, and looping lots of times.

Challenge 4

Make up your own challenge!