Learning Python with Turtle Graphics

Lesson Goal: There are many ways to learn anything; today, we're going to learn via tinkering. Before we begin, let's talk about what tinkering is: Inspiration-oriented learning. At the moment, you may or may not have an explicit reason to learn programming; maybe there's something specific you want to do now, or maybe you just have always have wanted to learn programming and have finally decided ot do it. Either way, programming is a big field, and even if your goals are specific, you'll likely have to learn some general skills before you can accomplish your goals and feel satisfied with the result. At the beginning of this course, we'll supply everyone with a few goals to start out with. This is, however, a *tinkering*-focused course, which means that we want you to be always questioning, and always exploring. It won't be long before you feel the itch to try something out on your own! In this course, that is absolutely encouraged! At some point, it won't be surprising if everyone is doing something different, and we will shift from a teacher-student relationship to a supporter-maker relationship. Let's get started!

Learning the Language

1. Open a terminal
    - Mac: Open Spotlight Search (apple+space) and type "Terminal". This program will launch.
    - Linux: Use the Keyboard shortcut Ctrl+Alt+T, or open the program "Terminal"
    - Windows: Open the program called "Command Window" or "Powershell Terminal"
2. Run the iPython Interactive REPL (Command-Line Interface), by typing: **ipython**

That's it! You're now in Python, one of the most popular general-purpose proramming languages in the world. Before we start making our 2D graphical designs, let's try doing the first thing everyone does when using computers: making math easier by using them as calculators!

Programming-Language-As-Calculator

- Do some math:
    - What is 8 * 32?
    - What is 1 + 1?

- Let's make x = 7. What is x * 5? x * 32? x / 3.2? x + 555?

- Let's make x = 8, y = 10, z = 12. What is y * x? x + z? z + x? x * 3 + y?
- let's say Anna has 7 apples, Joe has 2 apples, and Nick as 22 apples. Assign the correct number of apples to each person (like you did with x), and give the total number of apples (anna + joe + nick).

Okay, programming languages have even more advanced math features. To activate those features, we must **import** the package that contains them. To do this, type::

```
import math
```

To get access to the abilities inside the math package, you need to use the dot (.)::

```
math.sqrt(x)  # the square root of x
math.sin(x)  # The sine of x
math.exp(x)  # the exponent of x (e to the x'th power)
```

Notice the pattern above: **PackageName.FunctionName(InputName)**. Now, you may be wondering, how do I know what abilities (called "Functions" from now on) are available in the Math package? Try typing "Math.", and then pressing the "Tab" key. You'll see a list of options, which should give you some idea of what's avaliable. If you'd like to know more about what the function does, type "help(functionname)", without the parentheses after the function's name. For example::

```
help(math.sqrt)
```

To exit the help text, simply type the letter "**q**"

Use this pattern to answer the following:

- What is the square root of 32?
- What is the cosine of 1.72?
- what is the 5th digist of pi?
- What is the log of 18271?
- What is the log of the cosine of the square root of pi?

Turtle Graphics

Now you know how to use Python as a calculator! Isn't that great!? Okay, if you don't feel so excited at the moment, at least feel excited that you've learned an important programming pattern: To get access to more abilities, you can **import** packages that contain **functions** that run code for you! And you can see what is available in a package with the "Tab" key! And you can learn how a function works with the **help()** function! This pattern will come up time and time again, which is why we practiced it now.

Okay, we're all ready to go with graphics! The **turtle** package is a drawing application where you move Alex the **Turtle** (yes, you can name him anything you like!) around a **Screen**, and the turtle, who has a pen tied to his tail, leaves a trail behind him. To use this package, first, import it, then use these two lines to make a screen and make a turtle::

```
import turtle

screen = turtle.Screen()
alex = turtle.Turtle()
```

Alex can do lots of things. He can move **forward** some distance, for example::

```
alex.forward(100)
```

He can do other things, to! Let's take some time now and use the skills you've learned so far to figure out what kinds of things you can do with Alex!

(10-minute Tinkering Session, followed by discussion and review)

Exercises

- Figure out how to Clear the screen.
- Make a Triangle
- Make a Bigger Triangle.
- Make a Square.
- Change the color of the turtle's pen.
- Make a second turtle, and set his starting position to (10, 20), where he should draw a triangle, too.

Saving Your Code: Writing Scripts

When you quit Python, all your hard work will be lost! This is terribly sad, but if you put your code in a text file (named my_turtle_art.py), you can always run the code and the computer will make your beautiful artwork again! Let's make a couple scripts together.

(10 Minutes: Live, Joint coding session)

Making Your Own Functions.

Have you noticed how you keep writing the same thing, over and over again? Have you gotten tired of it yet? Well, Python provides a way to make your own functions, which will automatically do several steps for you–once you've made it, all you have to do is tell it to do **MyFunction()**!

Here's how to write a function in Python:

```
def function_name():
    step1
    step2
    step3
```

That's it! Note each part of the above code: the **def** statement, the colon (**:**), and spaces before each line under the **def** are all important. Let's make a couple of functions:

- Make a **triangle()** function.
- Make a **square()** function.
- Make a **triangle(size)** function that takes a **size** input, which makes bigger triangles when bigger numbers are put in them! This hasn't been introduced, but I bet you can figure out how to do this!

Making Loops

While Loops

(Lesson here)

For Loops

(Lesson here)

Tinker!

Okay, now you're free from this guided lesson! Take the next 20 minutes to make something you like!

Control the RoboTurtle

In the Erfindergarden, we have our very own RoboTurtle–he is a wonderful artist, whose life purpose is to draw your Turtle Graphics art on his canvas! He is controlled via a tiny computer called the **Raspberry Pi**. We'll make a Python script that controls him, so he can paint our scripts!

To make your script work with the RoboTurtle, first make a copy of your script. Then add the following at the top:

```
import roboturtle
```

This won't work on your computer, but our Raspberry Pi has the **roboturtle** package, and it will work there. Finally, you need to change your **alex = turtle.Turtle()** line to:

```
alex = roboturtle.Roboturtle()
```

That's it! Now your turtle has been Robo-fied! Give your new robot script to your instructor, and he'll see how our RoboTurtle does with your art!