Turtle Spiral Python Tutorial (Beginner)

Table of Contents

Overview

Turtle projects in Python are a great way to introduce some basic Python commands in a more visual way, while also offering a lot of room for experimentation. It also allows for some really cool looking designs while only taking a minimal amount of code and time to allow for the student to have something to show to their parents really early on.

Turtle Spiral is more of a general concept and not so much a project that has a set ending point. It focuses more on experimentation and introducing general concepts, that being said, this isn't something that you want to stick in for a super long time (it's recommended no more than three lessons before starting to cover other concepts). The drawing elements in Python can also be revisited in combination with new concepts that you have covered later on down the line to make some even more interesting projects as well.

Platform and languages

Platform	Optional Platform	Language
Repl.it	Trinket.io	Python

Project length

Two to three hours (no more than three sessions).

Skills

The main skills this project trains are:

- Importing packages
- Basic object creation
- Turtle commands
- Loops
- Basic math operators

Customization ideas

This project allows for the students to create their own shapes, designs, and even fractals if you get advanced enough. Give them a base line, but try to push them away from doing the exact designs that you are making and even see if they can make some sort of image. Give them a challenge to make a scene, use specific shapes, or even specific math operators to see what they can come up with.

Project

This project is separated into three segments: basic turtle commands, loops, and math operators (and how to combine all of that).

Part 1: Basic turtle commands

This section focuses on importing packages, creating a turtle object, and going over some basic turtle commands

Skills:

- Importing packages
- Basic object creation
- Turtle commands

Package setup

Before you can do anything with turtle, you will need to import the turtle package that gives you access to the commands. Make sure you explain what it means to import a package (It's recommended to use a tool box analogy, if you imported a hammer, you now have access to using a hammer on this project, etc. Find something that works for you).

You can find a list of turtle commands here: https://docs.python.org/3/library/turtle.html

```
1   # import turtle
2   import turtle
```

Creating the turtle object

Object creation is a big part of most languages (think of Unity for example, basically everything you work with is objects, at least starting out), so make sure to spend some time explaining the importance and what it really means to create an object. Show that you can create more than one object and what you can do with that.

```
7   # setup turtle pen
8   t= turtle.Turtle()
```

Note: you may also note that: t = turtle.Pen() would also work in this scenario, and do the same thing.

Basic turtle commands

Here you just want to do a quick overview of some of the basic commands so you can actually get something moving on screen and then give them a couple minutes (3-5) to just play around with these commands.

```
10  # changes the speed of the turtle
11  t.speed(100)
12  # changes the background color
13  turtle.bgcolor("black")
14  #moves the turtle in the direction it's facing by 50
15  turtle.forward(50)
16  #turns the turtle 90 degrees to the right
17  turtle.right(90)
```

Note:You may also want to talk about t.penup() or t.pendown() depending on what you are planning to teach, or you can save it for later (but definitely bring it up at some point).

Part 2: Loops in Python

This section focuses on indentation in Python, basic variable creation, and obviously loops as a whole.

Skills:

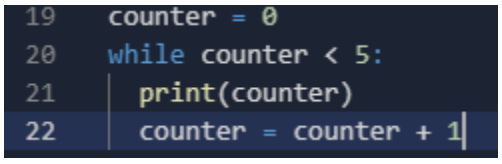
- Indentation
- Variable creation
- Loops

Data types

This may not be the most interesting section, and it's recommended to skip it if they are not in middle school yet, but going over different data types, primarily the really important ones, will help set them up in the long run. It's recommended to talk mostly about: int, double, char, string, bool and float (because floats are a big deal in game design), as you will see these in basically every language you work in. One of the biggest issues with Python is that they make it too easy to create and work with variables and it tends to make transitioning to other languages a bit bumpy if you haven't prepared them for it.

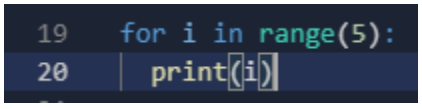
While loops and indentation

Starting with while loops is a good way to build up to for loops as it is broken up and shown explicitly what is happening. It also is a good way to start seeing how indentation works in Python. With this section, really focus on the condition in the loop and what it means, as well as how a difference in indentation can make a huge difference. Make sure to use variable names that make sense and don't start the student in the habit of just naming it different letters. Also very briefly cover what print does, but don't make it the focus of this lesson.



For loops

Make sure to not delete the while you have already created, that way you can show how similar they are. Also, make sure to talk about why the variable name: 'i' is so common in for loops (iteration) and bring up the concept of exclusiveness vs inclusiveness briefly. Also make sure to talk about what the 5 represents in the code (in this case, it runs 0,1,2,3,4).



Loops continued

Now that the general idea of loops is covered, start showing other examples: how would you make a loop go down, can we use the loop variable for other things, what happens if we have two values as parameters in the for loop? What about if we had three? There are a lot of things

to cover with for loops, but just go over the basics. Save the and/or operators for later projects as well.
Here is an example of a for loop with three values that goes down (5,4,3,2,1)

```
19  for i in range(5,0,-1):
20      print(i)
```

Part 3:Basic math operators in Python

This section focuses on expanding upon the math operators you have used for the loops and how to combine what has been gone over so far into some cool designs

Skills:

- Basic math operators
- Combining chunks of code

Basic math operators

This concept is very similar to the data types category as it isn't the most interesting at times, but is necessary to go over to prepare for future projects. It will also give access to the ability to make cooler designs. It's recommended to go over just the very basic math operators for now: + or -, * or /, sometimes even modulus (%) if they seem like they can handle it (as you can make some really cool patterns with it). Make sure to give examples and ask them questions to make sure they are following along.

Combining covered concepts (fun part)

This is where the student(s) will really start to see what these concepts can do. Get through creating a basic shape, and maybe creating a circle with them, then end by showing them some other cool images you can make without showing the code just yet. Once you've covered that, give them some time (5, maybe 10 minutes tops) to play around with it. One of the biggest issues I've found while teaching is not giving them enough time to play around with things on their own often results in a student who relies too much on the teacher later on down the line.
Here is an example of code for a square:

```
18  for i in range(4):
19      t.forward(100)
20      t.right(90)
```

Code for a circle:

```
18  for i in range(180):
19      t.forward(2)
20      t.right(2)
```

Note:Make sure to talk about why 180 and 2 and 2 work here, and other ways you could draw a circle.

Combining covered concepts Pt.2 Spirals

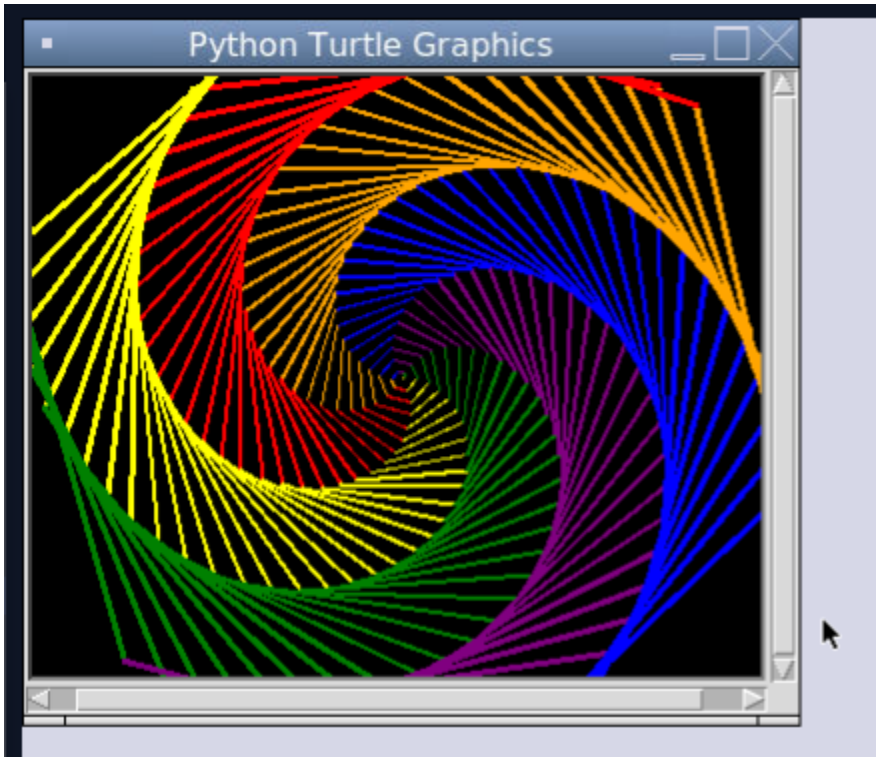
Below is some attached code for one of many ways you can make a spiral, but don't feel boxed into doing a spiral, you could make any other cool designs as long as they fit the level of the student you are teaching. Remember, show them the design, don't show them the code. Give them a chance to come up with their own designs first and if you do end up going through your code with them, make sure to only go line by line explaining each line. You may even want to do a little mini pop quiz at the end to see what they retained.

Spiral code:

```
# import turtle
import turtle

# defining colors
colors = ['red', 'yellow', 'green', 'purple', 'blue', 'orange']

# setup turtle pen
t= turtle.Turtle()

# changes the speed of the turtle
t.speed(100)
# changes the background color
turtle.bgcolor("black")
#moves the turtle in the direction it's facing by 50

# make spiral_web
for x in range(200):
    t.pencolor(colors[x%6]) # setting color
    t.width(x/100 + 1) # setting width
    t.forward(x) # moving forward
    t.left(59) # moving left

turtle.done()
t.speed(10)

turtle.bgcolor("black") # changes the background color

# make spiral_web
for x in range(200):
    t.pencolor(colors[x%6]) # setting color
    t.width(x/100 + 1) # setting width
    t.forward(x) # moving forward
    t.left(59) # moving left
```

Conclusion

The end product should resemble something like this:
Finished



Final notes

- Try to get your student to make their own images and designs by experimenting with the concepts you have covered so far
- Consider going over simple functions and you could use those to make the code cleaner
- Find out what your students like about the project and what your student found the most confusing, try to take note of that to review more next time
- Consider starting to make a quizlet with the student with concepts like data types or math operators (etc.) on it so they can review during the week. Think of this quizlet as a living doc that will grow each couple of sessions.

Version	Author	Reviewed by	Date
0.1	Adam Carter		10/5/21