Triangle & Square

Scaling up our shape drawings with reusable functions.

Overview

In Day 3 we made our triangle DRYer using a for loop. Today we're going to add the ability to easily **draw many triangles** by creating a triangle function. **Functions** allow us to reuse code that we (& others) have already written. We'll further explore the power of functions by adding **square** and **move** functions.

Video Introduction

Day 4: Functions https://youtu.be/GqHcVdyvk-A

Instructions

1. If you don't have your previous program open, **navigate to trinket.io/turtle** and replace any code that you didn't write with the code from Day 3's activity:

```
1  import turtle
2  turtle.color("red")
3
4  size = 100
5  # Repeat 3 times
6  for i in range(3):
7    turtle.forward(size)
8    turtle.left(120)
```

*What if we wanted to draw a **second** triangle? Is there a **DRY** way to do this **without copying & pasting** lines 4-8 again?*

2. By creating a triangle **function**, we can easily create many triangles! Recall our function template & answer the following:

```
def function_name(parameter_names):
  # function body
```

   - What should our **parameter(s)** be? What value(s) change with every call?
   - What should we **name** the function?
   - **What code** becomes the **function's body**?

3. **Refactor** (or refine) the example triangle drawing code above into its own **function**, using your answers to the above questions as a guide.

4. **Test** your function by calling it with size `100`. **Run** your code. Is it the **same**?

Congratulations! *You've successfully created your first function!*

Triangle & Square

*The **true test** of a function is calling it with **different parameters** within the **same program**.*

5. **Test** your function by calling it **3 times**. Your drawing should look something like:

```
triangle(100)
triangle(50)
triangle(25)
```

6. It's difficult to know if your function is working correctly when all the triangles overlap. Let's **implement a move function** to help us space out the different triangles we want to test. Python's turtle supports `penup` & `pendown` functions that enable us to move the turtle without drawing. I've chosen to **move back** instead of forward because my turtle was running off trinket's canvas:

```
def back(len):
  turtle.penup()
  turtle.backward(len)
  turtle.pendown()
```

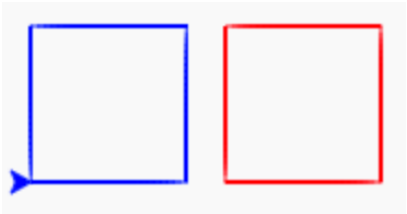
```
triangle(100)
back(75)
triangle(50)
back(50)
triangle(25)
```

*See how DRY & concise the code is? By convention, all functions are defined at the **top** of the file, **after** any **imports**, and **before** any non-indented **calls** are made for testing.*

7. Now let's try drawing a different shape: **square**. It will look **similar** to triangle:
   - How does the code **differ** for drawing a triangle versus a square?
   - What should our **parameter(s)** be? What value(s) change with every call?
   - What should we **name** the function?
   - **What code** becomes the **function's body**?

8. Using your triangle function & your answers to the above questions as a guide, **implement** a square function. **Test** it by calling it with size `100` and **run** it.

*Notice how **similar** the **triangle & square** functions are? We'll fix that missing DRYness **tomorrow**!*



9. **Test** your function **more** by drawing two squares:

Awesome work! *You're learning how to make reusable, DRY, & professional code!*

Video Solution

Day 4: Triangle & Square Walk Thru https://youtu.be/tw9Pum6pelg

Terrific job! *Functions are a challenging concept to master – great work! Join us tomorrow as we put everything we've learned together & explore **opportunities** for your continued learning.*