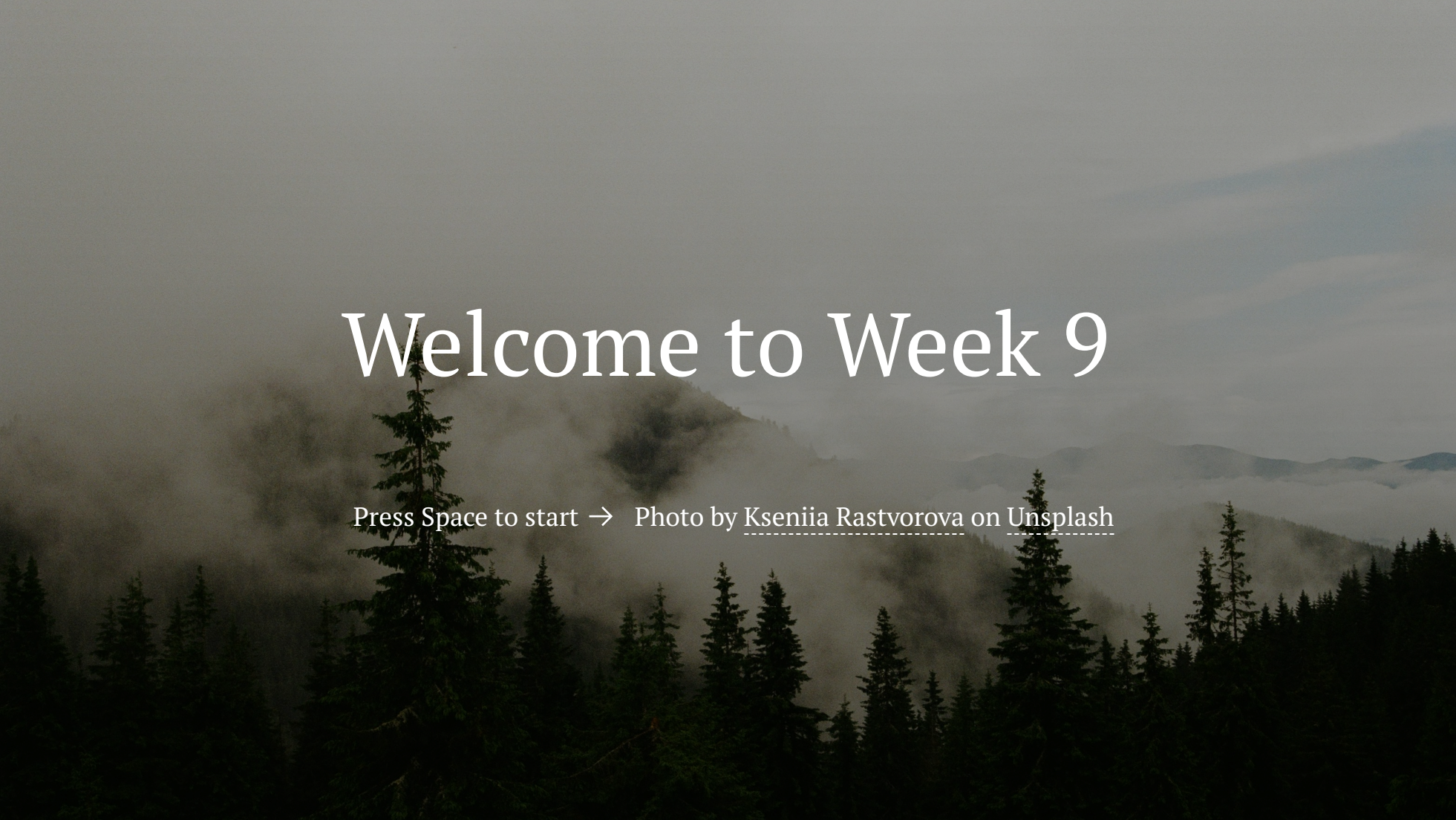

Welcome to Week 9

Press Space to start → Photo by Kseniia Rastvorova on Unsplash

Agenda for Today

- Blast through functions one more time (important addition: talk about mental model)
- Turtle: Drawing in Python
- Review While Loops
- your questions and/or extra practice test-style questions

Functions (again)

We're going to review functions and get some more details on their usage.

Functions are very important in programming!

Functions allow us to reuse code, make it modular, and give structure to our programs. Here, we'll cover the essentials of defining functions in Python.

Defining a Function

From the Python docs:

> "The keyword `def` introduces a function definition. It must be followed by the function name and the parenthesized list of formal parameters. The statements that form the body of the function start at the next line and must be indented."

Example of a Simple Function

```
def my_math_function(a, b):
    return a * 2 + b * 3
```

This function takes two arguments, `a` and `b`, multiplies `a` by 2, and `b` by 3, and then returns the sum.

Recipe for a Function

To create a function, follow these steps:

1. Use `def` + `function_name` + arguments in parentheses + a colon.
2. Indent all code within the function block.
3. Use `return` to send back a value.

Example:

```
def greet(name):
    message = f"Hello, {name}!"
    return message
```

Calling `greet("Alice")` will return `"Hello, Alice!"`.

Scope in Functions

Functions have their own scope. Variables defined inside functions aren’t accessible outside them unless returned.

Tip: Imagine your function as if it were in a separate file.

Slides break: Go to Python visualizer and run some functions!

Default Arguments

You can set default values for arguments:

```
def power(base, exponent=2):
    return base ** exponent
```

Using `power(3)` defaults to `exponent=2`, giving `9`, while `power(3, 3)` returns `27`.

Challenge: What’s an example of a function with a default argument that we’ve used?

```python
# Try to get this code to run by ONLY ADDING things

# give this a default argument
def f1(x):
    return x * 2

# takes multiple arguments: pass then below
def f2(a, b):
    return a + b

# f3 takes no arguments
def f3():  # No arguments
    return "Hello!"

def f4(value):  # Single argument
    if value > 0:
        return "Positive"
    else:
        return "Non-positive"


successes = []
try:
    r1 = f1()
    successes.append("f1 works")
except Exception as e:
    print(e)
```

Summary

- Define functions with `def` and indent all code inside.
- Functions have their own scope; use arguments and returns.
- Use default arguments to make functions flexible.
- Troubleshoot by testing each function individually.

One useful mental model: Buckets and Robots

- Variables are buckets (different shapes and sizes)
- Functions are little robots that do things
- Variables hold a value
- Functions act on those values, change things, and bring things back to you!

Turtle TLDR Overview

Now, let’s do a "Too Long, Didn’t Read" tutorial on Python’s Turtle graphics.

- aim to complement other learning materials.

How to Use This Deck

Here's the suggested approach for learning:

1. Glance through the slides for an overview.
2. Uncomment and run each code line in `turtle_tldr.py` incrementally to see how Turtle behaves.
3. Experiment by changing function arguments to see their effects on the output.

Turtle TLDR Code Snippets

Here are key code snippets from `turtle_tldr.py` to help you get started:

```
import turtle

# Create a turtle object named pen
pen = turtle.Turtle()
pen.speed(0)  # Fastest speed

# Draw a triangle with stamps at each vertex
for i in range(3):
    pen.forward(50)
    pen.stamp()
    pen.left(120)

# Re-position pen for the next part of the drawing
pen.penup()
pen.goto(100, -100)
pen.pendown()

# Draw a square with blue color
pen.color("blue")
for _ in range(4):
    pen.forward(100)
    pen.right(90)
```

Turtle Practice Prompt

Assignment: Create a Unique Abstract Art Piece

Use Python's Turtle module and the `random` module to generate different abstract artwork with each script run.

Suggested Steps

Try out the following approaches:

- Draw random lines
- Draw random circles and lines
- Draw random squares
- **Advanced:** Use a math function like `sin` (via `import math -> math.sin()`)

Be creative and experiment with Turtle methods like `forward()`, `right()`, `left()`, `penup()`, `pendown()`, `color()`, and `speed()`.

Coding Tips

To organize your code:

- Create a `main` function for the main body of code, and call it at the end with `main()`.

Use this snippet as a starter kit:

```
# Starter kit 5-line snippet for turtle art
import turtle as t
import random
def main():
    t.forward(100)
main()
```

Hints for Getting Started

If you're stuck, try drawing random lines with this function:

```
import random

def draw_random_lines(num_lines):
    for _ in range(num_lines):
        t.color(random.random(), random.random(), random.random())
        t.penup()
        t.goto(random.randint(-200, 200), random.randint(-200, 200))
        t.pendown()
        t.setheading(random.randint(0, 360))
        t.forward(random.randint(50, 200))
```

Call `draw_random_lines(5)` under `main()` to draw five random lines. Experiment with circles or squares from here!

Advanced Color and Positioning

```
# Use RGB colors for more variety
turtle.colormode(255)
mycolor = (255, 0, 120)
pen.color(mycolor)
for _ in range(4):
    pen.forward(100)
    pen.right(90)

# Hide the turtle at the end
pen.hideturtle()
turtle.done()
```

Summary

- Experiment with different shapes and functions in Turtle.
- Use randomness to create unique outputs.
- Explore color and positioning to make visually appealing patterns.