07 Prepare: Preparation Material

Overview

We've all seen how a picture or a movie is taken in front of a "green screen" and then editors are able to replace the green background with another picture, making it look like the person was standing in a completely different location. Have you ever wondered how this works?

In this lesson, you'll learn the tools you need to perform your own green screen editing, where you merge two pictures together.

Images are composed of a number of pixels, or small "dots," that have different colors associated with them. In order to accomplish the green screen effect, you will need to have your program look at the value of one pixel and do something with it, then move to the next one, then the the next, etc., until you have gone through every pixel in the picture.

When you go through items one at a time or repeat the same steps over and over again, it is called a *loop*.

I. LOOPS

An important feature of programs is the ability for the computer to repeat certain steps over and over again. This concept of a *loop* is the topic of this week.

As you might expect, there are many variations of loops depending on how many times or under which conditions the program should repeat certain pieces. In Python, we tell the program to loop by using either the **`for`** or the **`while`** keyword, depending on our specific situation.

Preparation Material

There are two types of loops in Python, **`while`** loops and **`for`** loops. This lesson will focus on **`while`** loops and the next lesson will focus on **`for`** loops.

Watch the following videos:

- Introduction to Loops (11 mins)

A **`while`** loop continues *while* something is still true, or *as long as* it's true, or stated another way, "until it's no longer true."

For example, you might keep asking the user for a number as long as, or *while* they keep typing numbers under 10, and stop as soon as they enter one that is larger than that:

```python
number = 0

# Keep looping as long as the number is less than 10
while number < 10:
    number = int(input("What is the number? "))

print("Finished with the loop")
```

The output of this program could be something like the following:

```
What is the number? 3
What is the number? 7
What is the number? 2
What is the number? 24
Finished with the loop
```

This also works when the computer is updating a value, rather than getting input from the user. The following program counts up to 5 and then stops:

```
# Start with the number 1
number = 1

# Keep looping as long as the number is less than or equal to 5
while number <= 5:
    # Display the number
    print(f"The number is: {number}")

    # Update the number to be one more than it was
    number = number + 1

print("Finished with the loop")
```

The output of this program could be something like the following:

```
The number is: 1
The number is: 2
The number is: 3
The number is: 4
The number is: 5
Finished with the loop
```

II. DECLARING VARIABLES BEFORE THEY ARE USED

Just like with **`if`** statements, any variables that will be used in the **`while`** expression, need to be declared and assigned values **before** they are referenced in that condition.

In the following example, the program will cause an error because it tries to check of number is less than 10, but the variable does not exist yet.

```
# BAD EXAMPLE: This code does not define a value for the number before it is used

while number < 10: # ERROR, number is not defined yet
    number = int(input("What is the number? "))
```

```
print("Finished with the loop")
```

The following example corrects the mistake by declaring the variable and setting it equal to a value prior to the loop.

```
# GOOD EXAMPLE: This code correctly sets the variable to a value before it is used

number = 0

while number < 10:
    number = int(input("What is the number? "))

print("Finished with the loop")
```

In an example such as these, what value should you initialize the variable to? The answer is that you can pick any value that allows the program to enter the loop. In the case above, the loop will run as long as `number` is less than 10. So if we started it at something more than 10, it would never run:

```
# BAD EXAMPLE: This code sets the variable to a number that prevents
# the code from ever entering the loop.

number = 25 # ERROR: This number is greater than 10, so the loop will not run

while number < 10: # This body of this loop will NEVER run
    number = int(input("What is the number? "))

print("Finished with the loop")
```

In the following two examples, the mistake is corrected by assigning the variable to a value that allows the loop to run.

```
# OK EXAMPLE: This code sets the variable to a number that allows the loop to run
# But it is not great, because it sets it to a non-standard value of 6.

number = 6 # This number is less than 10, so the loop will run, but it is not standard

while number < 10: # This body of this loop will run just fine
```

```
    number = int(input("What is the number? "))

print("Finished with the loop")
```

```
# GOOD EXAMPLE: This code sets the variable to a number that allows the loop to run
# It uses a standard initialization value of 0.

number = 0 # This number is less than 10, and is a standard value

while number < 10: # This body of this loop will run just fine
    number = int(input("What is the number? "))

print("Finished with the loop")
```

Notice that in the previous two examples, the variable is set to a value less than 10, so the loop will run. Technically, any number less than 10 would work (including negative numbers), but it is common practice to initialize unused integer variables to **`0`** or **`-1`**, to initialize strings to **`""`**, and to initialize boolean variables to **`False`**.

III. VARIABLE SCOPE

Just as you need to declare a variable before the loop if you want to use it in the condition statement, variables that are first declared inside the body of a loop (or an **`if`** statement) should not be used after the loop. Sometimes programming languages like Python will allow this to work, but it is not considered good practice, because it can cause bugs to arise in your code that are difficult to track down.

```
# BAD EXAMPLE: This code uses the variable "name" outside the loop where
# it was declared

number = 0

while number < 10:
    number = int(input("What is the number? "))
    name = input("What is your name?")
```

```
print(f"Your name is: {name}")
```

```
# GOOD EXAMPLE: This code first declares the variable "name" before the loop so
# that it can be used afterward.

number = 0
name = ""

while number < 10:
    number = int(input("What is the number? "))
    name = input("What is your name?")

print(f"Your name is: {name}")
```

Working with Images in Python

For this lesson's Prove assignment, you'll be working with images. In order to do so, you'll need to understand some of the fundamentals of digital images on a computer and also download and install a Python library to help you manipulate them.

IV. DIGITAL IMAGES

A digital image is composed of a number pixels arranged in a grid. Each of these pixels is a color that can be represented as a red, green, and blue (RGB) value. These values range from 0 to 255 for each of the red, green, and blue components. So if a pixel had a red value of 255, and a green value of 0, and a blue value of 255, it would appear as one bright purple dot (mix of red and blue) in the picture.

When each of these individual pixels are shown next to one another, they form a picture or image. In any given image you might have thousands or millions of pixels. If you talk about a 10 Megapixel digital camera, that translates to approximately 10 million pixels.

V. INSTALLING A PYTHON LIBRARY

For your assignment, you will use a pre-built library (Python code written by others that you can use in your program) to help open an image, access the pixel values, and save it out when you are done. A library like the one you'll use for your project is very powerful and can help you do much more than we are using it for, but it will certainly help us.

The library you will use for this project is called Pillow (the name comes from its relation to an earlier library "PIL" or Python Imaging Library). Because it doesn't come with the standard installation of Python, you'll need to download and install it. While this could be done manually, Python has a tool designed to download and install libraries for you called **`pip`**.

Once everything is set up correctly, the pip tool works really well, but sometimes your computer needs help finding the pip program. There are different ways to run pip, but if we know where Python is installed, we can run a single command in our terminal window in VS Code. Please note that the terminal window is where you usually see the output of your program. You are not writing this code in a python program.

Please watch the following video that talks about how to identify the location of Python on your computer and run the pip command:

- Video: Installing Python Packages (15 mins)

As stated in the video, the following are recommended as the first commands to try running in the terminal window:

Windows users:

```
py -m pip install pillow --user
```

Mac users:

```
python3 -m pip install pillow --user
```

If these commands do not work for you, you'll need to follow the other advice in the video to include the full path to python.

VI. VERIFY THE INSTALLATION

Once you have installed the library, you can verify that it was installed correctly and in a place where VS Code will find it. To verify your installation, start a new python program and include the following code:

```
from PIL import Image
print("The library is loaded correctly")
```

Run the program like you would any other Python program. If it runs, and displays, "The library is loaded correctly" without producing any error messages, you'll know that everything is installed and working correctly.

If you see error messages, you'll have to keep working to get it installed. Sometimes this takes a little bit of work to get exactly right, so please try it early and ask for help if you run into problems.