Lecture 02

Programming Fundamentals: Values, Types, and Variables

January 26th, 2022

Song of the day: ***Hell of a Woman*** *by Papooz (2022).*

Part 1: *Parts of a Program*

Before we start with the programming material, check out the following tour of the IDLE IDE, which I will be using for examples during class and what not.

The moment you open IDLE up. You will be met with the following window:

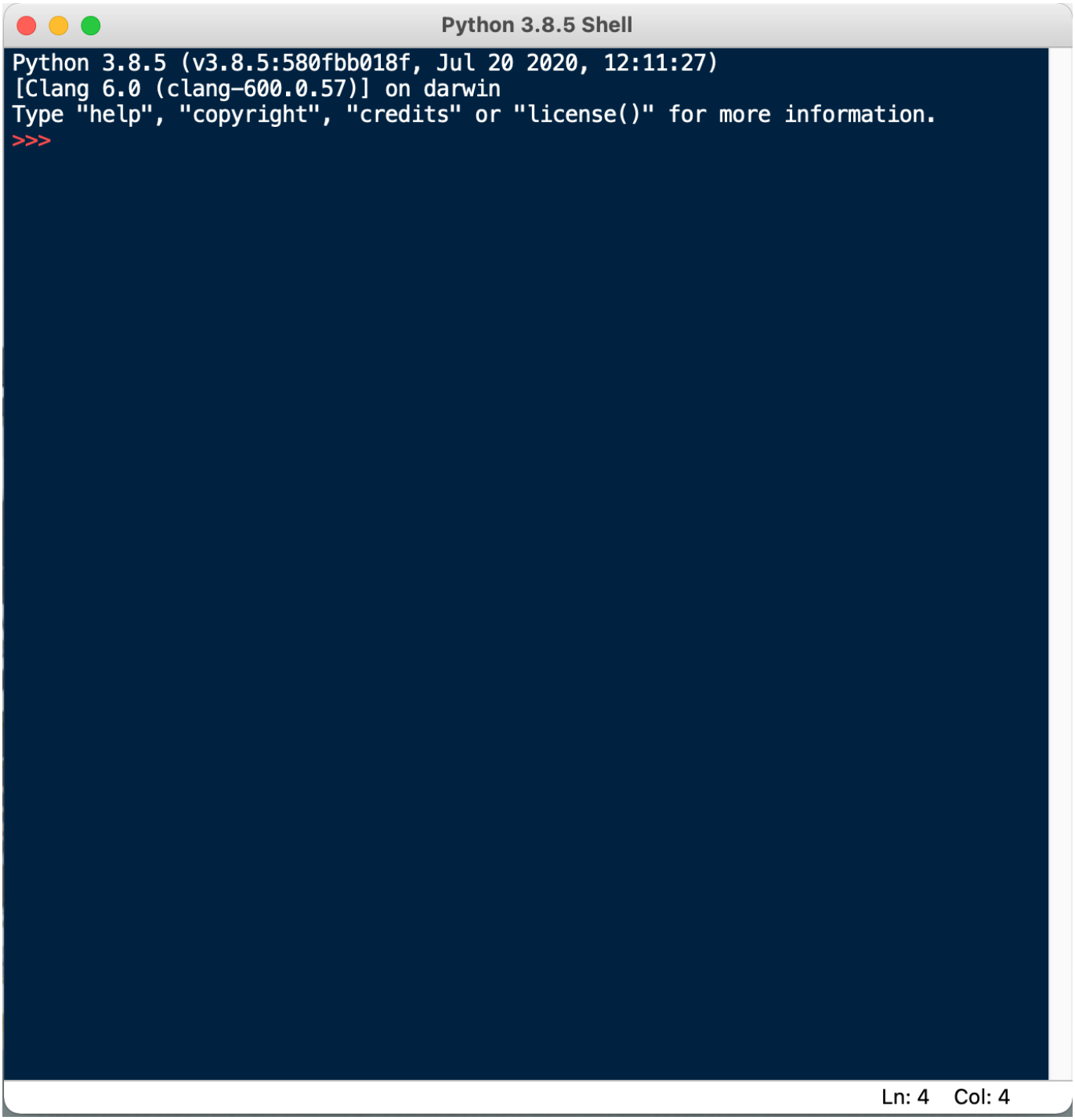

Figure 1: Your console, the IDLE Shell.

In order to create a new Python file (i.e. a file with the `.py` extension), you should navigate to the following menu:

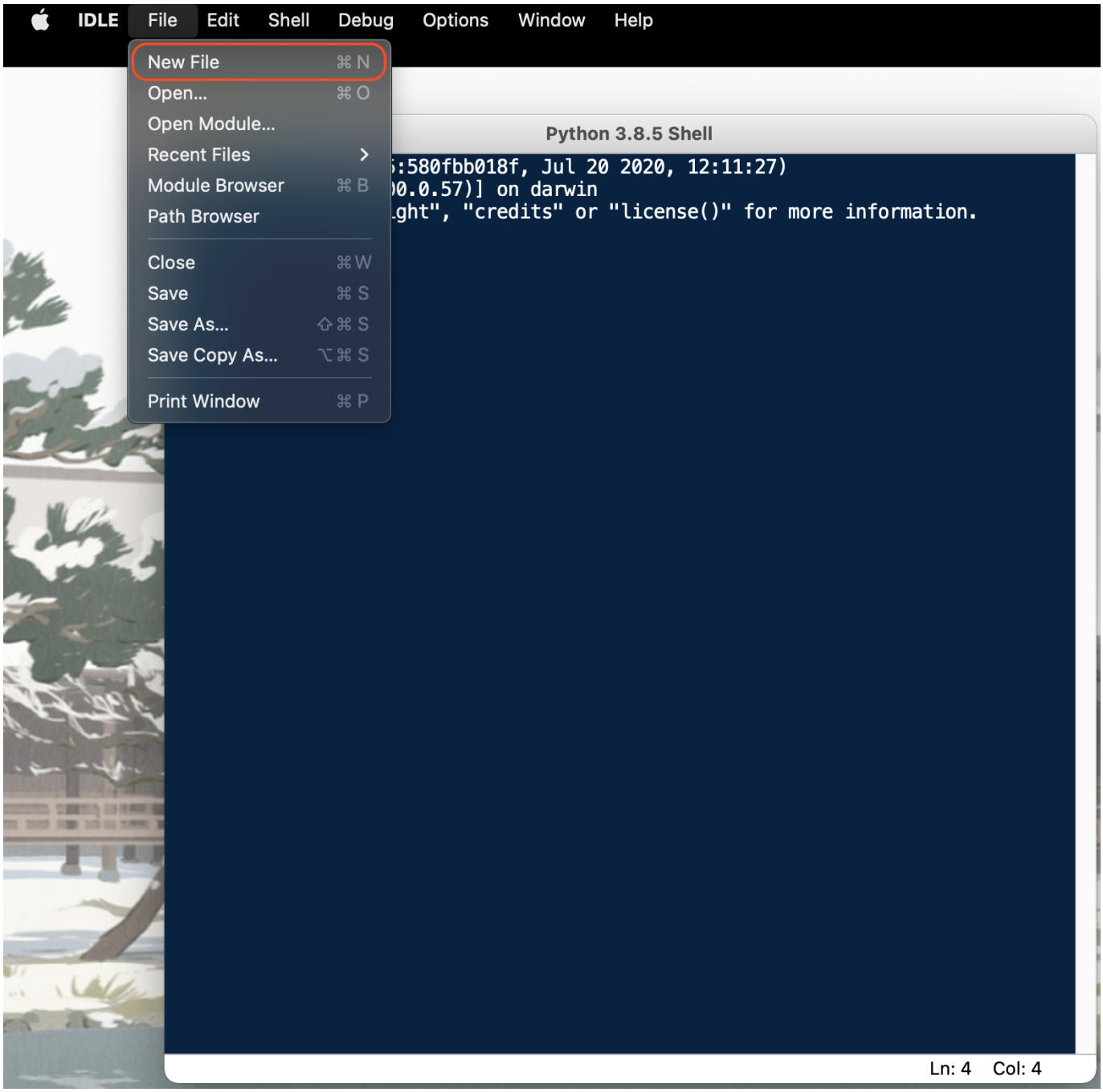

Figure 2: Creating a new `py` file.

Clicking on `New File` will open a blank `py` file close to your console:

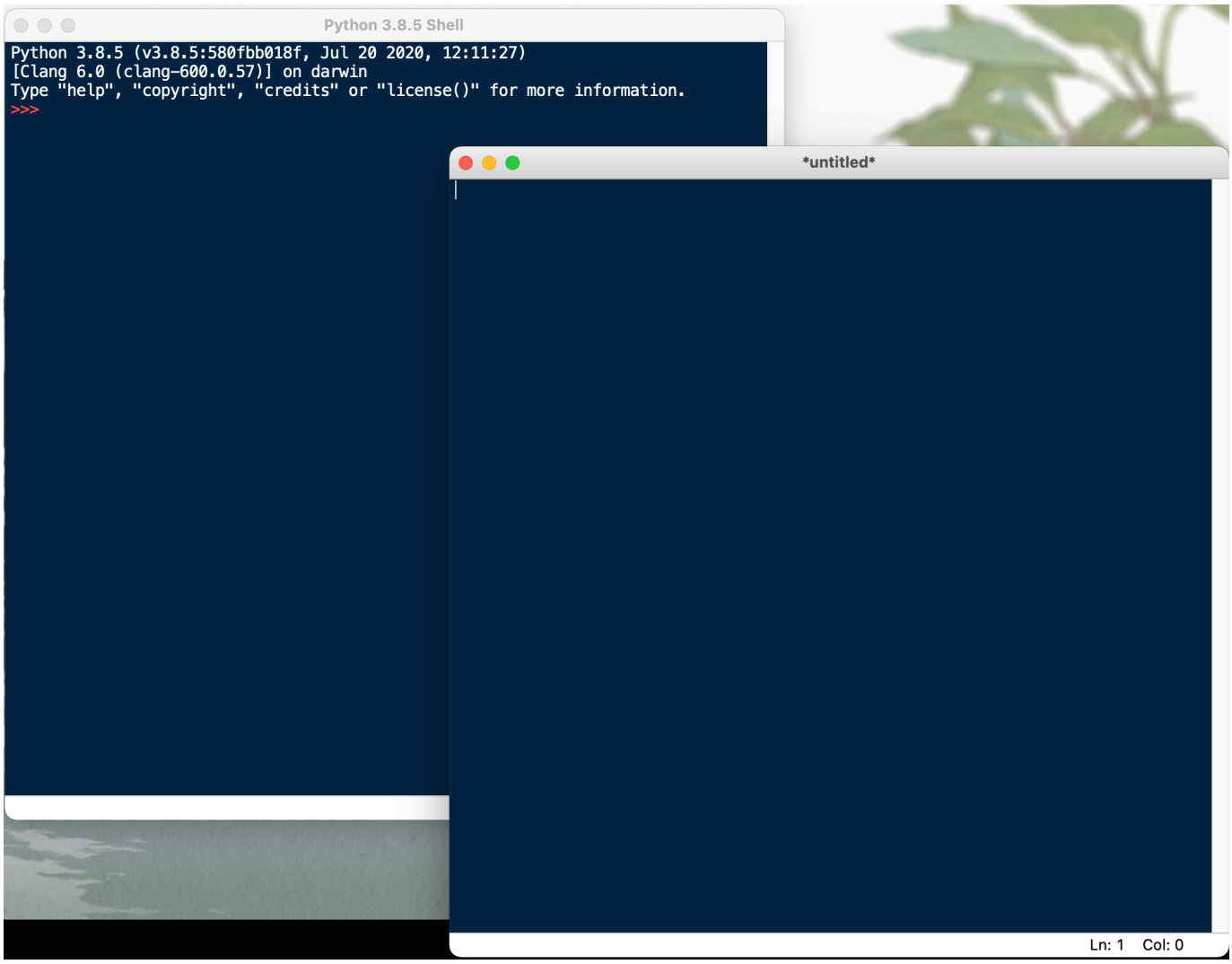

Figure 3: A blank `.py` file (right).

We can then write a program, such as this simple *Hello, World!*, and **save the file** anywhere on your computer. In my case, I saved it on my Desktop, but I ***highly*** recommend creating a dedicated folder for this class.

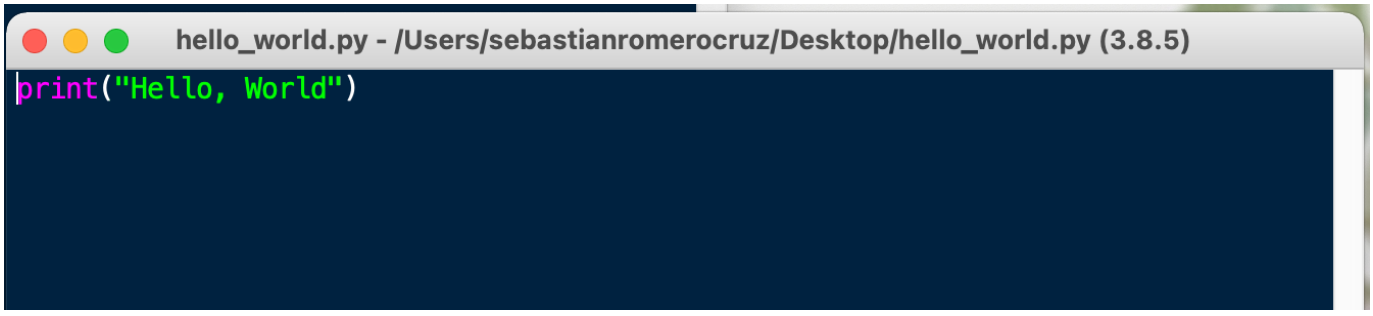

Figure 4: The canonical *Hello, World!* program in Python.

To run our program, we have to navigate to the following top menu option:

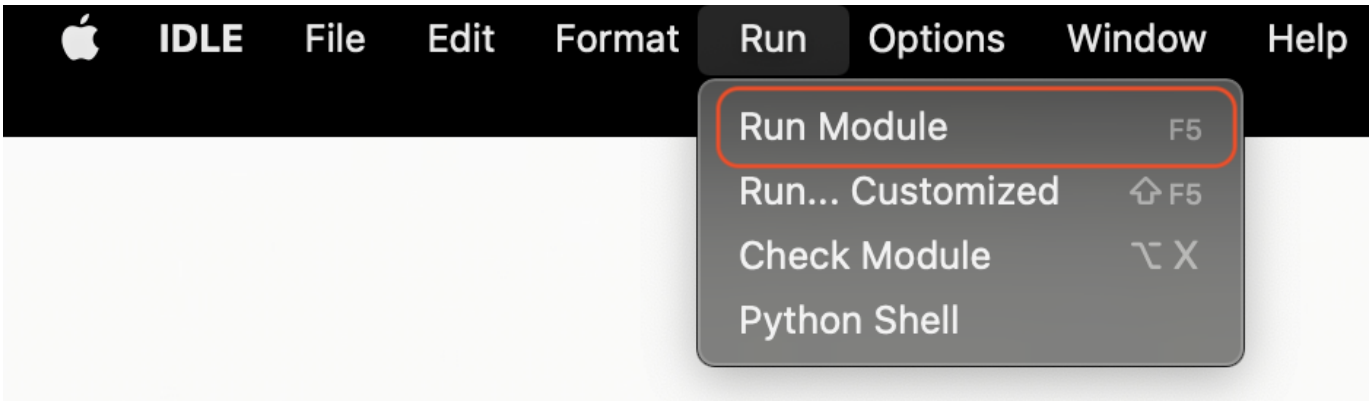

Figure 5: Click on `Run Module` to run your code.

The window should switch over to the console, showing you the result of running your program:

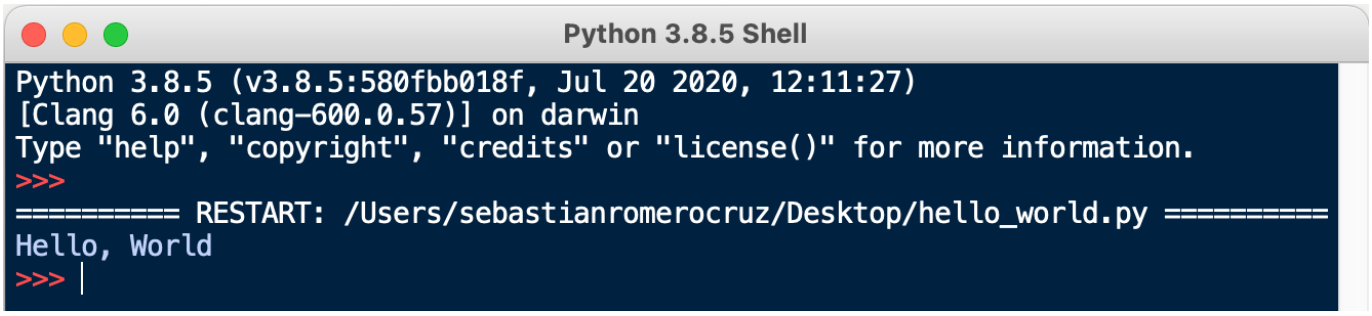

Figure 6: The output of our `.py` file being displayed on our console.

Part 2: *Values and Types*

The very first thing we will learn about is quite literally the reason why computer science exists: data—things like our ages, our grades, our names, etc..

The formal definition of a ***value*** is as follows:

> **Value (a.k.a. Objects)**: A number, string, or other kinds of data that can be stored in a variable or computed in an expression.

There's a couple of words in that sentence that you might have not seen before, but we'll get to them in due time. Just know that a value in Python is basically just a piece of data or information.

One quirk of Python is that ***all values are instances of an object***. You'll learn the specifics of objects near the end of the semester, but for now we can be introduced to the most basic object types of the language:

Type	Examples	Description

Type	Examples	Description
`int`	`1` , `42` , `-101` , `0`	A data type representing a **whole number (integer)** value, positive or negative
`float`	`3.1416` , `22.7` , `-4.0` , `1.0`	A data type representing a **floating-point (decimal-valued)** number value, positive or negative, and is only an approximation. Be careful using them in calculations.
`str`	`"Cardcaptor Sakura"` , `'Viva la Revolución'` , `'''Comments'''` , `""""""""""`	A data type representing a **sequence of characters (string)** characters. Can be enclosed using `'` , `"` , `'''` (or `"""`)

***Figure 7**: Three of the most common types in Python.*

The keywords `int` , `float` , and `str` , aside from representing these three types, also serve as **conversion functions**:

```
>>> int(4.5)
4

>>> float(7)
7.0

>>> str(1.2)
'1.2'

>>> int('42')
42

>>> float('Liz and the Blue Bird')
Traceback (most recent call last):
  File "<input>", line 1, in <module>
ValueError: could not convert string to float: 'Liz and the Blue Bird'
```

***Code Block 1**: Examples of valid and invalid type conversions in Python.*

The process of converting values from one type to the other is often called **type casting**. So, code block 2, in order, reads as follows:

> The value of the float value `4.5` casted as an **integer** is `4` .
>
> The value of the integer value `7` casted as a **float** is `7.0` .

> The value of the float value `1.2` casted as a **string** is `'1.2'`.
>
> The value of the string value `"42"` casted as an **integer** is `42`.
>
> The value of the string value `"Liz and the Blue Bird"` casted as a float is **invalid**.

As you can see, casting to either an integer or a float from a string requires your string to contain a numeric value, and nothing else.

This is often a point of confusion for students. They will get, say, the string `"3.15"` as the answer for an operation. However, if the rest of the program operates on `3.15` assuming that it is a float number, your program will very likely crash. Being able to catch and recognize these errors takes some practice, but it is something you should be consciously watching out for from the beginning.

Part 2: *Variables*

Okay, so we have a way of representing data in the form of types, but how do we store this data so that we can use it in our programs? This is the job of ***variables***.

A good way of thinking of variables is as boxes that store our belongings when we are moving. Usually, we store things in boxes to keep them safe and organized so that we can easily find and use them later on. Moreover, the best way to know which box holds what is to label them—like putting a piece of tape with the contents written on it.

That's basically the exact same process we use in programming to ensure that our data is stored and easily accessible to us.

For example, if we wanted to store the current year, we'd do something like this in Python:

```
current_year = 2022
```

In this statement, `current_year` is the name of the variable, `=` is the **assignment operator**, and `2021` is the value.

If I try doing this in IDLE's console, you'll see this:

```
Python 3.8.5 (v3.8.5:580fbb018f, Jul 20 2020, 12:11:27)
[Clang 6.0 (clang-600.0.57)] on darwin
Type "help", "copyright", "credits" or "license()" for more information.
>>> current_year = 2022
>>> current_year
2022
>>>
```

***Figure 8**: A visualization of Python storing the integer value `2021` inside a variable called `current_year`.*

Now, I could have called this variable anything I wanted. As long as your variable names start with an alphabetic character or an underscore (`_`), you are not restricted in any way:

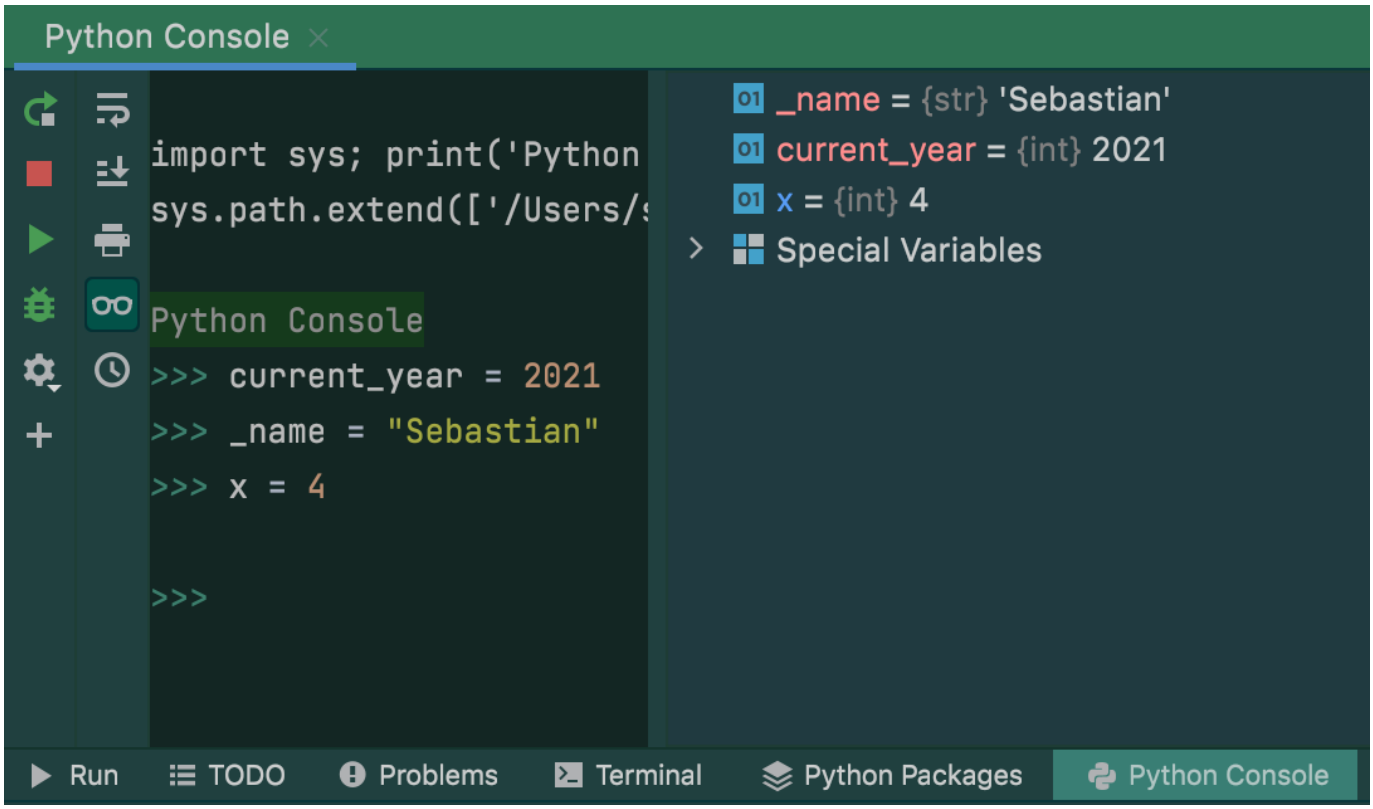

***Figure 9**: A visualization of Python storing integer and string values inside variables `_name`, `current_year`, `x`.*

Now, of course, not all variable names are understood equally. Just like labes on boxes, giving your variables relevant, explicative names is the way to go. In this class, in particular, make sure to follow these rules in order to not get points taken off:

1. Make sure your variables have useful names (i.e. favor `acceleration_of_gravity = 9.81` over `aog = 9.81`).
2. Do not, and I repeat, do ***not*** give your variables single-letter variable names. This will always be penalized (with a single exception that we won't get into for a while).
3. [redacted] follow either snake-case (`sound_euphonium_2`), or camel-case (`soundEuphonium2`); this, of course, means that variables are case-sensitive (i.e. `hello_world` and `HELLO_WORLD` are two different, completely unrelated variable names).
4. They cannot be a Python keyword (`if`, `def`, `while`, etc.).

A couple of technical terms that you should be aware of are **namespace** and **object space**. Simply put, the namespace is where the names of your variables are stored, and the object space is where

the values of your variables are stored:

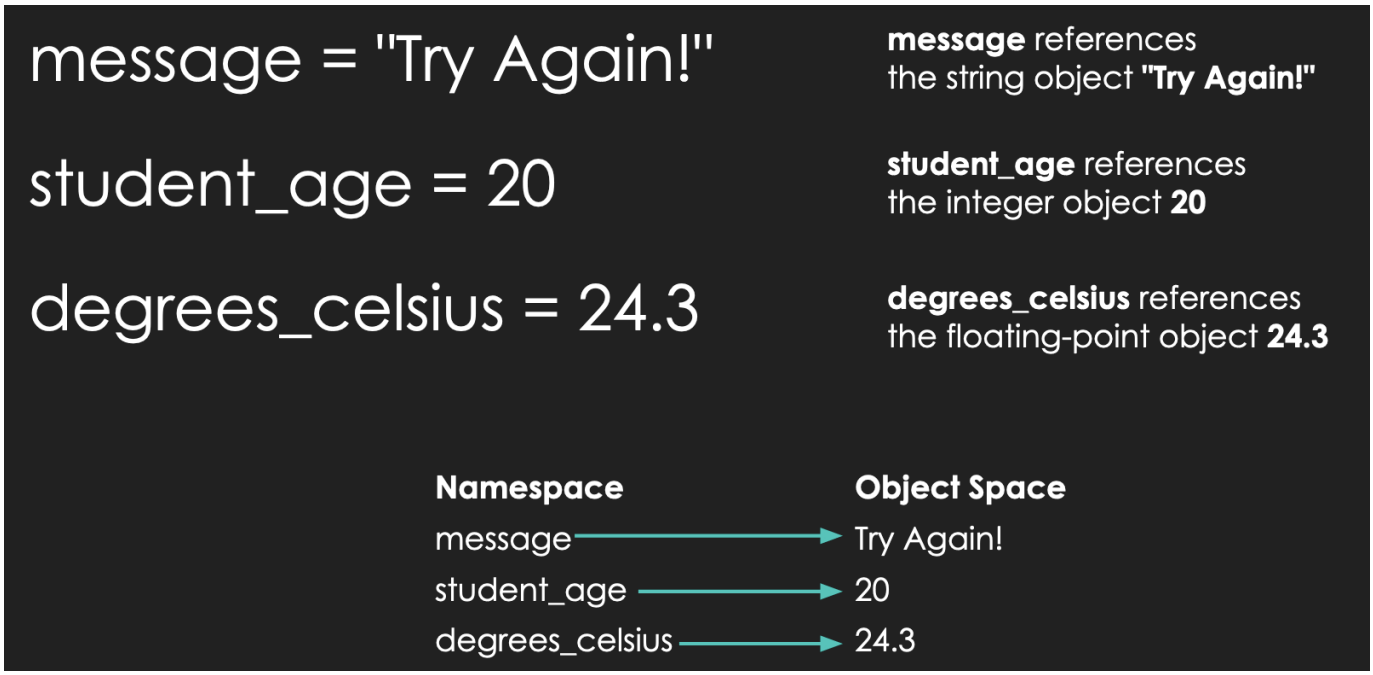

***Figure 10**: Examples of namespace vs. object space connections.*

These basically refer to the way values are stored and referenced in your computer's memory. You don't have to know much more about this; you'll learn more about memory than you'll ever want to during the rest of your four years as a CS/CE/EE major.

That's it for this week. [redacted]

Have a great weekend <3