Introduction to Coding (in Python)
Lesson 1 - Variables

Variables

This initial introduction to coding uses .py files written for Python 3. Hopefully you already have a way to copy, create, edit, and execute these files (see the Install folder if not).

The first, fundamental concept in coding is the idea of a *variable*, or a piece of information stored by the code.
In Python, you can create a variable using =
Example 1:

```
Var1 = 3.14
vaR2 = 2.2
c = 'speed of light is 3e8 m/s'
```

The equals sign tells the code to take something on the right side of the =, (a number or text or different *data type*) and store it as a variable, located on the left of the =. The variables created in Example 1 are Var1, vaR2, and c. As you can see, you can use any combination of letters and numbers (without spaces) as a variable. You cannot use a number as the first character of a variable, since the code would try to interpret it as a number.

But if that is true, then why is it okay to use letters, since those are another data type?
>The key is in the quotation marks we used when saving the text to the variable c. Putting a series of letters and digits inside single or double quotations in python creates a *string*. That is the data type of the variable c.

Printing

Printing in python does not mean the code attaches to a printer and prints out a sheet of paper. The print *function* in python writes down the *argument* you give the function[1] into the same place where you ran your code.
Example 2:

```
myVar = "My Name is Roark."
print(myVar)
```

This example, if the lines are in a file that your run, will output "My Name is Roark." to the place where you ran the file.

[1] We will talk more about functions later; just focus on a function being something followed by parentheses, where anything inside the parentheses is the 'argument' of the function.

PROBLEM 1:

Print "Hello World!" followed by the number Pi (3.14…) to two decimal places.

Note: The number Pi should not have quotation marks (single or double) around it. 3.14, not '3.14'

Addition, Division, and all that Jazz

Since we can save numbers, it would be nice if we could perform all the mathematical operations we want on those saved numbers. In python, the characters used for basic mathematic operations are shown in the table below.

Addition	+
Subtraction	-
Multiplication	*
Division	/
Exponentiation	**

Each of these operations can be applied to any variable that is a number!

Example 3:

```
     myNumber = 2
     squareOfMyNumber = myNumber**2
     newNumber = myNumber + squareOfMyNumber
     newNumber = newNumber – 1
     newNumber = newNumber * myNumber
```

Now, the last two lines of Example 3 can be simplified, since they end up being very common in coding. In both cases, we are doing something to the variable newNumber to get something new and then saving that new thing as the variable newNumber. So another way to think of newNumber = newNumber – 1 is "removing 1 from newNumber."

The simpler way to write this is newNumber -= 1

Similarly, for the last line, newNumber *= myNumber

This also works for addition using the operation +=

PROBLEM 2:

Test python's order of operations by printing out the value of (5 – 5 * 2 ** 2)

(Unless your computer is broken, you should get -15)

Now we need to look at division. Ideally, if we divide 1 by 2, we should get 0.5. Usually this works well in python! However, in certain cases the code might return 0 or 1 instead of 0.5 because of the *Data Type* of the variable.

Data Types

Every variable contains some value of a certain data type. Some examples of data types are integers, floats, and characters. A computer's memory is finite, and any given variable can only hold a certain amount of information, regardless of data type. Without getting into a discussion about bits, bytes, and memory, let's just focus on the difference between integers and floats.

Numerical data

An integer is a number such as 1, 2, 5, -23, etc.
In python, we can make anything an integer with the function int

Example 4:

```
     n = int(2.10429)
```

In this example, n will equal 2 since the int function rounds its argument to the nearest integer.

Similarly, the float function stores any number as its decimal interpretation.

Example 5:

```
     a = float(2.10429)
```

In this example a will be exactly 2.10429, without any rounding.

However, python does have limited *precision* and can only store floats and integers with a certain number of digits. Try to run the code in Bad Example 1.

Bad Example 1:

```
     a = 2.190394578243096158723985743109857
```

In this case, a is actually stored as 2.190394578243096, which contains 16 digits. There are different data types than the basic float and int with more memory. But the default precision for a single number is 16 significant digits.

PROBLEM 3:

Using your new understanding of data types, predict what the following code will produce as output:

```
     b = 2.0
     c = 5.0
     print(int(b/c))
```

```
    print(b/c)
```

Non-numerical data

Most of the time, you will not store a single character in a variable. Instead, you will store a string of characters. A string is a special kind of *list*, one specifically made of characters and formatted differently.

Lists in python are denoted with brackets [] and the elements of the list are separated by commas.

Example 6:

```
    L1 = [“a”, ‘b’, “c”, “d”]
    L2 = “abcd”
    print(L1[0])
    print(L2[0])
    print(L1)
    print(L2)
```

Notice the notation of L1[i] where i is any integer value. This returns the (i-1)th component of the list (or string).

AN IMPORTANT DISINCTION BETWEEN PYTHON AND OTHER LANGUAGES LIKE MATLAB IS THAT THE INDEXING OF LISTS STARTS AT 0, NOT 1.

Indexing is how a coding language labels the different components of a list. In python, the indexing begins at 0, meaning that the first component in the list is given an index of 0. This also means that the final component is given an index equivalent to the length of the list *minus 1.*

To better understand this, let’s learn about the len function.

Example 7:

```
    L1 = [“a”, ‘b’, “c”, “d”]
    L1Len = len(L1)
    print(L1Len)
    print(L1[L1Len-1])
```

Note that if you get rid of the minus 1 in the last line of Example 7, then the code will return an IndexError because there is no component in the list with that index.

Indexing is an important part of scientific programming, because any data will be indexed into lists or *arrays*, which is a different data type we will talk about when learning about the NumPy library.