Introduction to Coding (in Python)

Lesson 1 - Variables

Making Variables

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 `=`

Run the code below by clicking on the box and pressing Shift and Enter at the same time (or click the Run button above)

Example 1:

In [7]:
```
1 Var1 = 3.14
2 vaR2 = 2.2
3 c = "speed of light is 3e8 m/s"
```

The equals sign tells the code to take something on the right side of `=` (a number or text or different data type) and store it as a variable, located on the left of the `=` sign.

The variables created in Example 1 are `Var1`, `vaR2`, and `c`.

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. Try it! Python will give you an error for *Invalid Syntax*.

But if that is true, then why is it okay to use letters, since those are another data type (used in variable c)?

> 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.

Example 2:

In [10]:
```
1 myVar = "My Name is Roark."
2 print(myVar)
3
```

```
My Name is Roark.
```

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'

In [ ]:
```
1
```

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.

Operation	Python Symbol
Addition	+
Subtraction	-
Multiplication	*
Division	/
Exponentiation	**

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

Example 3:

In [20]:
```
1 myNumber = 2
2 squareOfMyNumber = myNumber**2
3 newNumber = myNumber + squareOfMyNumber
4 newNumber = newNumber - 1
5 newNumber = newNumber * myNumber
6
7 print(newNumber)
```

```
10
```

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, 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 $5 - 5 \times 2^2$

(you should get -15)

In [ ]:
```
1
```

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:

In [22]:
```
1 n = int(2.10429)
2 print(n)
```

```
2
```

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:

In [23]:
```
a = float(2.10429)
print(a)
```

```
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:

In [24]:
```
a = 2.190394578243096158723985743109857
print(a)
```

```
2.190394578243096
```

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)
```

In [ ]:
```

```

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

In [27]:

```
L1 = ["a", 'b', "c", "d"]
L2 = "abcd"
print(L1[0])
print(L2[0])
print(L1)
print(L2)

```

```
a
a
['a', 'b', 'c', 'd']
abcd
```

Notice the notation of `L1[i]` where `i` is any integer value. This returns the $(i-1)^{\text{th}}$ component of the list (or string).

AN IMPORTANT DISINCTION BETWEEN PYTHON AND OTHER LANGUAGES LIKE MATLAB IS **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:

In [29]:

```
L1 = ["a", 'b', "c", "d"]

L1Len = len(L1)
print(L1Len)
print(L1[L1Len-1])

```

```
4
d
```

Note that if you get rid of the `-1` in the last line of Example 7, then the code will return an IndexError because there is no component in the list with an index of 4.

Indexing is an important part of scientific programming, because data will be indexed into lists or **arrays**. This is another data type we will learn about later.