Chapter 1 - Variables

Objectives:

- Learn about how to create and use Python variables for calculations
- Become comfortable using Jupyter Notebooks and Google Colab
- Learn how to use to `input()` statement to prompt the user for information

1.0 Use variables to store values

- Variables are names for values
- In Python, the `=` symbol assigns a value on the right to the name on the left
- The variable is created when a value is assigned to it
- In the following example, we create two variables to store the name and age of a person. Execute the code by mousing over the code cell and pressing the play button. Once you run the code cell, a green checkmark should appear next to the cell. This indicates the cell has been executed and the variables are stored in the computer memory.

```
name = "Marie"
age = 20
```

- Variable names can **only** contain letters, digits and the underscore `_`.
- They cannot contain spaces
- Variable names cannot start with a digit.
- Variable names are case sensitive. The variable `velocity` is not the same as the variable `Velocity`.

1.1 Use `print()` to display variables

- Python has a built-in function called `print()` that prints things as text.
- For example, run the following code cell to display the value of the `name` variable. You should see "Marie" displayed under the code cell.

```
print(name)
```

If you see an error message, it might be because the initial code cell defining the `age` and `name` variables was never run. If so, go back, run it and then re-run the print statement.

You can also include descriptive text and multiple variables in the print statement. For example, run the following code cell:

```
print(name, 'is', age, 'years old')
```

```
Marie is 20 years old
```

You should see the output displayed below the code cell. Notice the following:

- descriptive text is enclosed in quotes (either single or double)
- the text and variables are separated by commas
- `print()` automatically puts a single space between the items to separate them and wraps to a new line at the end.

BTW, the the print statement is the first example of something called a **function**. The function takes **arguments**, which are the variable names and the strings enclosed in quotes. We'll talk more about functions later.

1.2 Variables persist between cells

The previous examples show that variables defined in one cell can be used in cells farther down the page. In the above example, the variables `age` and `name` were defined in one cell and used in a print statement in a different cell.

One consequence of this "memory" is that after the notebook is run once, all the variables will be remembered the next time the code is run. While this normally isn't an issue for well-writted code, it can cause unexpected results if variables aren't properly defined the first time they are used. If you suspect that this effect may be causing problems, you have an option:

- Click on the down triangle next to "Run all" and select "Restart session and run all". This will clear all variables before running the code cells.

1.3 Variables must be created before they are used

- If a variable doesn't exist yet, or if the name has been misspelled, Python reports an error.

Run the following code block to see what happens when you try to print out a variable that hasn't been defined

```
print(first_name)
```

✅ Skill Check 1

To correct this error, double-click on the code cell and replace "first_name" with "name". Then run the cell. The error message should disappear and you should see "Maria" displayed. 🟩

1.4 Order of execution

Be aware that if you run code cells out of order in a Jupyter document, strange things can happen. I recommend the following:

- **Always execute code cells in order as you go down the document**. Running code cells out of order can lead to unexpected results, confusion and frustration!
- Sometimes it is better to just click **Run all** on the menu bar at the top of this page to run all code cells in order, rather than running them individually. This will ensure that you don't miss a cell.
- If you want to be really safe, click on the down triangle next to "Run all" and select "**Restart session and run all**". This will clear all variables before running the code cells.

1.5 Variables can be used in calculations

Consider the following code that defines the width and height of a rectangle, calculates the area and then displays the result. Run the following code cell:

```
width = 10              # in meters
height = 5              # in meters
area = width * height   # calculate area of rectangle in meters^2
print("area =",area)    # display result
```

- You should see the result appear below the code cell.
- Note, we added comments to describe each line of code. As mentioned before, comments are to help make the code easier to read and are highly encouraged.
- When commenting your own code, we recommend lining the hash tags # up whenever possible to keep your code need and easy to read.

Suppose you received revised information that the triangle is actually 5 meters wider than you thought. You could update the width variable as follows (run the following code cell):

```
width = width + 5
```

This can be a confusing statement to non-programmers. If interpreted as a math equation, one could "cancel" the `width` variable from both sides of the equation and be left with `0 = 5` 😕. But, remember:

- In Python and other programming languages the "=" symbol is an **assignment operator** and and not a mathematical equals sign.
- In this example, it means take the previously defined value of the `width` variable and add 5 to it. Then store that new value back in the `width` variable.

If you want to know the area of the revised rectangle, you will need to recalculate the area (run the following):

```
area = width * height           # calculate area of revised rectangle in meters^2
print("revised area =",area)    # display result
```

Remember that variables are updated in the order that they are executed (i.e. down the page). Thus, if we change the value of a variable, we would need to redo any calculations that depend on it.

1.6 Variable names should be descriptive

Variable names should be descriptive, i.e. `mass`, `velocity`, etc. or the should correspond to typical mathematical variables, i.e. `m`, `v`, etc. Variable names cannot contain spaces. If you want a complex variable name, you can use the underscore `_` in place of a space. Another option is "Camel" notation, where capitalization can be used. See the following examples:

```
seconds_per_minute = 60   # use underscore
secondsPerMinute = 60     # "Camel" notation
```

Sometimes it is useful to define fundamental constants in MKS units when doing physics calculations. In Python, scientific notation is written as `1.23e4` for 1.23×10^4. The `e` proceeds the power of ten. Here are some standard fundamental constants. Run this code cell to create these variables.

🔆 List of fundamental physical constants

```
c = 299792458            # definition of the speed of light in m/s
h = 6.626e-34            # Planck's constant (J s)
hbar = 1.0546e-34        # "h bar" = h / 2*pi  (J s)
G = 6.6743e-11           # Gravitational constant (m^3/kg/s^2
e = 1.602177e-19         # fundamental charge (C)
me = 9.10938e-31         # mass of electron (kg)
epsilon0 = 8.854188e-12  # vacuum permittivity (F/m)
```

After running the above code cell, you can now use these constants in calculations.

🔆 Example: Rest energy

For example, let's use Einstein's famous $E = mc^2$ to calculate the energy (in Joules) contained in my cat ($m = 5$ kg). Of course this energy can ony be released if my cat meets an anti-cat made purely of antimatter, but...

```
m = 5               # mass of my cat in kg
E = m * c**2        # Einstein's formula. We use the value of c defined above
print(m,"kg is equivalent to",E,"Joules")  # display the result
```

✅ Skill Check 2

A light year (ly) is defined as the distance light can travel in one year: $ly = c \cdot year$, where $c =$ the speed of light and $year =$ length of a year. Follow these steps to calculate the length of a light year in meters:

- Create a code cell below this text cell (Hover your mouse near the end of this text cell. Click on the `+Code` button that appears)
- Define variables to store (1) the number of seconds per minute, (2) number of minutes per hour, (3) hours per day, (4) ave days per year (= 365.24)
- Calculate the number of seconds per year and store it in a variable (maybe call it `year'):
  $year = \left(\frac{seconds}{minute}\right)\left(\frac{minutes}{hour}\right)\left(\frac{hours}{day}\right)\left(\frac{days}{year}\right)$
- Use the variable `c` defined above for the speed of light and the variable containing the length of a year in seconds to calculate a light year. $ly = c \cdot year$
- Use a print statement to display your answer including a short label, for example: "light year in meters = ".
- Run your code. If you get errors, don't panic!! This is normal. You'll need to debug your code and see if you can find and correct the errors.

1.7 Prompting the user for input

Sometimes, it is convenient to have the program prompt the user for input, rather than defining variables by hand. Python provides the `input()` command to do this. Here's how the `input()` function works:

- `my_variable` = input("message to user")
- This line of code will prompt the user to type in a value and save it in the variable `my_variable`
- replace `my_variable` with a variable name of your choice
- replace "`message to user`" with your own message

The following prompts the user to enter their favorite number and then displays a message back to the user:

```
number = input("Enter your favorite number: ")
print(number, "is my favorite number too!")
```

- Run the above code cell
- An input box should appear under the code cell
- Type your favorite number into the input box and press Enter or Return
- The code should display your number back ot you.

✅ Skill Check 3

Write some code to prompt the user to enter the name of their favorite physicist. Print a message to the screen that says "Your favotite physicist is ___", where ___is the name the user entered.

1.8 Possible Errors

Error messages are a normal part of coding. Even though they can be quite vexing at times, think of them as the computer's way of helping you succeed! Overcoming anxiety over error messages is a big step on the path to coding enlightenment.

Here are some common errors that you might encounter and ways to fix them:

Variable is not defined

- We talked about this one earlier.
- This error arises if you try to use a variable in print statement or calculation before defining it

```
print(hot_dog)
```

- Because we haven't defined a variable named "hot_dog", we get this error. Make sure to define variables before using them.
- When using Jupyter notebooks, you might have forgotten to run a cell with the variable definition. To check if this is the case, you can simply click "**Run all**" from the top menu bar to ensure all cells are executed
- Often, this error arises if you have a typo in the variable name. For example, if your variable was spelled "Hot_Dog" instead of "hot_dog" you would get this error

Missing quote

Google Colab has auto-completion and often tries to help you remember to include closing quotes and parentheses. But sometimes errors still arise.

For example, if you forget to include a closing quote in tha print statement, you can get the cryptic message "unterminated string literal". Run the following statement as is:

```
print("hello)
```

- To fix this message, simply include the closign quote after "hello"

Missing parenthesis

- If you have unequal left and right parentheses, you can get the message (run the code to see the error):

```
print("hello"
```

- To fix this message, simply include the missing parenthesis

Key Points

- Use variables to store values.
- Use `print()` to display variable values.
- Variables persist between cells.
- Variables must be created before they are used.
- Always run cells in order from top to bottom of the notebook
- Python is case-sensitive.
- Use meaningful variable names.
- Use the `input()` statement if you want to prompt the user for input
- Learn to overcome anxiety caused by error messages and accept them as helpful tools

This tutorial is a modified adaptation of "Python for Physicists" © Software Carpentry