**Strating of the DataScience Journey**

**Solving a mystery with data**

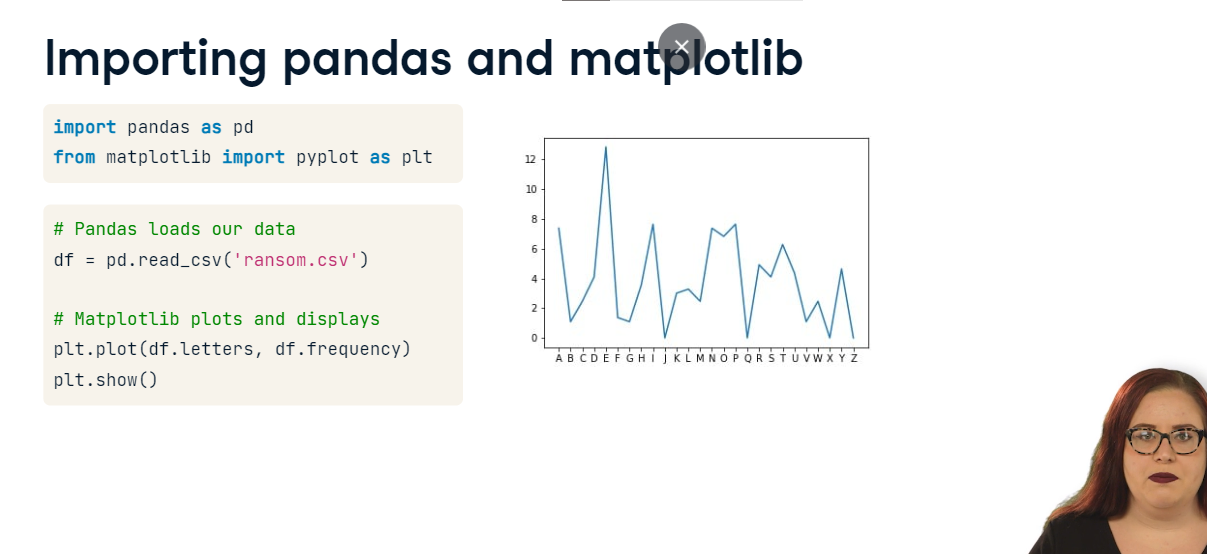
While we learn, we'll be solving a mystery using data. Someone has kidnapped Bayes, DataCamp's prize-winning Golden Retriever. The kidnapper has left clues that we can analyze. We'll use techniques like chemical analysis and letter frequency to pick the correct suspect.

**What is a module?**

Now that we know where to write code, let's dive into our first concept: modules. Modules help group together related tools in Python. For example, we might want to group together all of the tools that make different types of charts: bar charts, line charts, and histograms. Some common examples of modules are matplotlib (which creates charts), pandas (which loads tabular data), scikit-learn (which performs machine learning), scipy (which contains statistics functions), and nltk (which works with text data).

**Importing pandas and matplotlib**

We must import any modules that we plan on using before we can write any other code. We do this at the top of the script editor. If we don't import modules, we can't use the tools that they contain. In this example, by importing the modules pandas and matplotlib, we're able to unbox the tools necessary to create a graph. In this case, pandas gives us the tools to read data from a file, and matplotlib gives us the tools to plot the data.



**Importing a module**

To import a module, simply type "import" followed by a space and then the module name. Oftentimes, module names are long, so we can shorten them by using an alias. To give your module an alias, just add "as" and a shorter name to your original import statement. This statement will alias "pandas" as "pd".

**Creating variables**

Previously, you started writing Python code in the script editor and the console. You learned what a module is, and how to import it. You also learned to simplify module names using an alias. In this lesson, you'll learn about variables, which help us reference a piece of data for later use.

**Filing a missing puppy report**

In the previous lesson, we told you about our kidnapped Golden Retriever, Bayes. To solve the mystery, let's start by filling out a Missing Puppy Report. In order to file the report, we'll need to record some information about Bayes, such as his height and weight. In Python, we will represent each line from the missing puppy report with a variable. A variable gives us an easy-to-use shortcut to a piece of data. Whenever we use the variable name in our code, it will be replaced with the original piece of data. In this case, one of our variables is "name" and its value is "Bayes". Another variable is "height" and its value is "24". We define variables using an equals sign.

**Rules for variable names**

When defining variables, we need to follow a few rules. Variables must start with a letter. You can use a capital letter, but we usually use lowercase. After the first letter, we can use letters, numbers, and underscores in our variable name. We can't use special characters like exclamation points or dashes. Variable names are case sensitive, so these two different ways of typing "my\_var" would be different variables. On the left are some examples of valid variable names, and on the right are some examples of invalid variable names.

**Error messages**

Let's see what happens when we try to use an invalid variable name. The variable bayes-height is invalid because of the hyphen. When we try to enter it, we will receive a SyntaxError. Above the Syntax Error, we see the line of code that caused the problem and a caret that indicates approximately where the problem occurred.

**Floats and strings**

Variables come in many "flavors". Two important flavors are floats and strings. Floats represent either integers or decimals. Strings represent text and can contain letters, numbers, spaces, and special characters. We define a string by putting either single or double quotes around a piece of text.

**Common string mistakes**

It's easy to get errors when working with strings. If you get one, there are two likely causes. You might have forgotten to put quotation marks around your string. If you do this, Python will think that your string is a variable, and if that variable wasn't previously defined, you'll get a "name error". Alternatively, you might have started your string with one type of quote and finished with a different type. If you mix single and double quotes, you'll get a syntax error.

**Displaying variables**

If we want to know the current value of one of our variables, we can use "print". We simply type the word "print" and put our variable name inside of the parentheses. When we execute the code, the value will appear in the console. Remember: the variable name is not a string, so we don't put it in quotes.

**What is a function?**

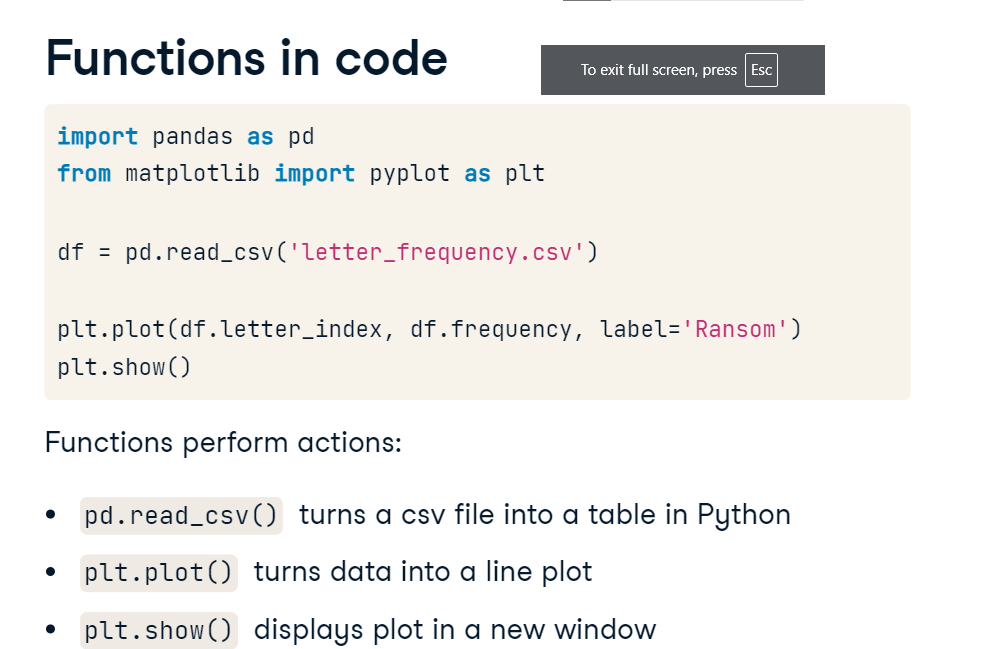
Previously, we learned about variables (which allow us to store information for later use), strings (which represent text), and floats (which represent numbers). Now that we know the basic data types in Python, we'll start using them with functions.

**A function is an action**

A function is an action. It turns one or more inputs into an output. In this example, the function is called "turn\_orange". The input is a shape (such as a blue square) and the "action" is to color the input shape orange.

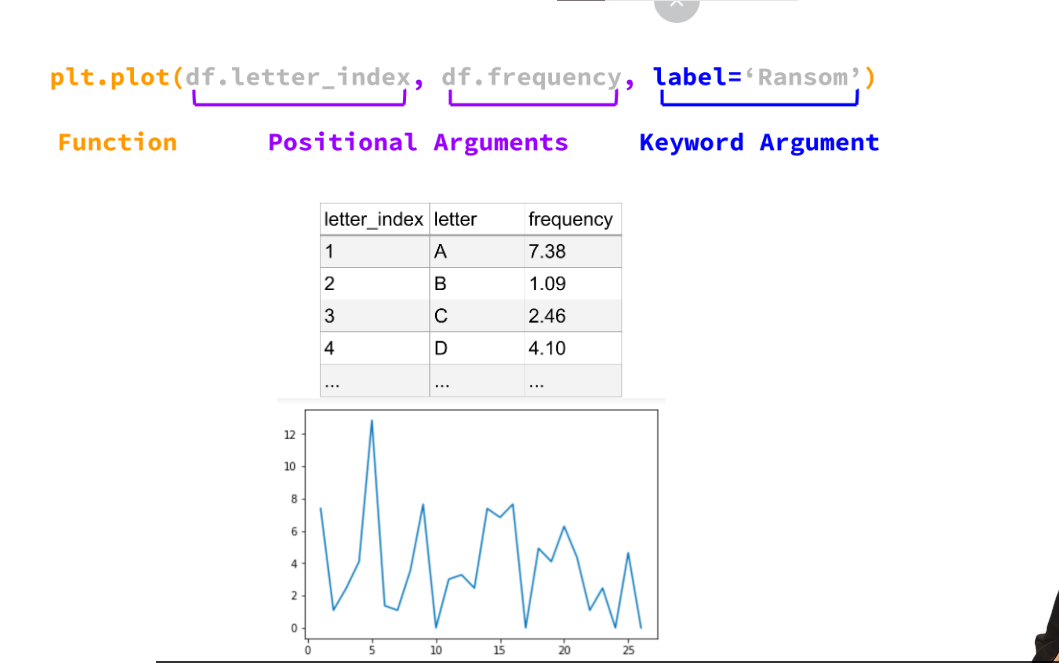
**Functions in code**

Consider the following code snippet. In a previous lesson, we learned that this code would read some data and produce a graph. In this snippet there are three functions: pd-dot-read\_csv turns a csv file into a table in Python, plt-dot-plot turns data into a line plot, and plt-dot-show displays the plot in a new window. Let's learn about functions by examining one from our code snippets.



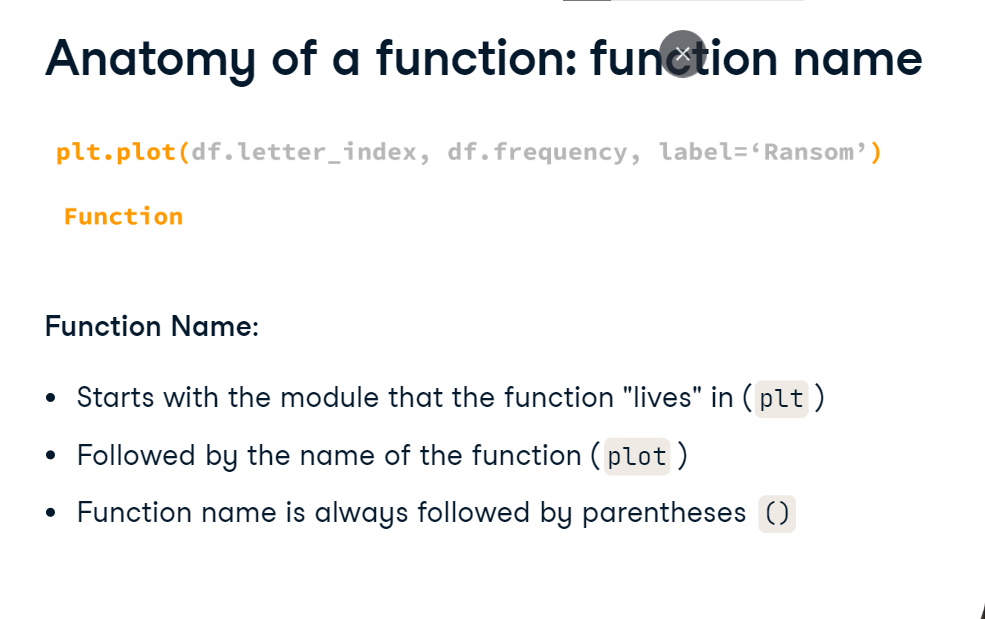
**Anatomy of a function**

This function takes the data from the table in the bottom-left and plots letter\_index on the x-axis and frequency on the y-axis, resulting in the graph on the bottom-right.



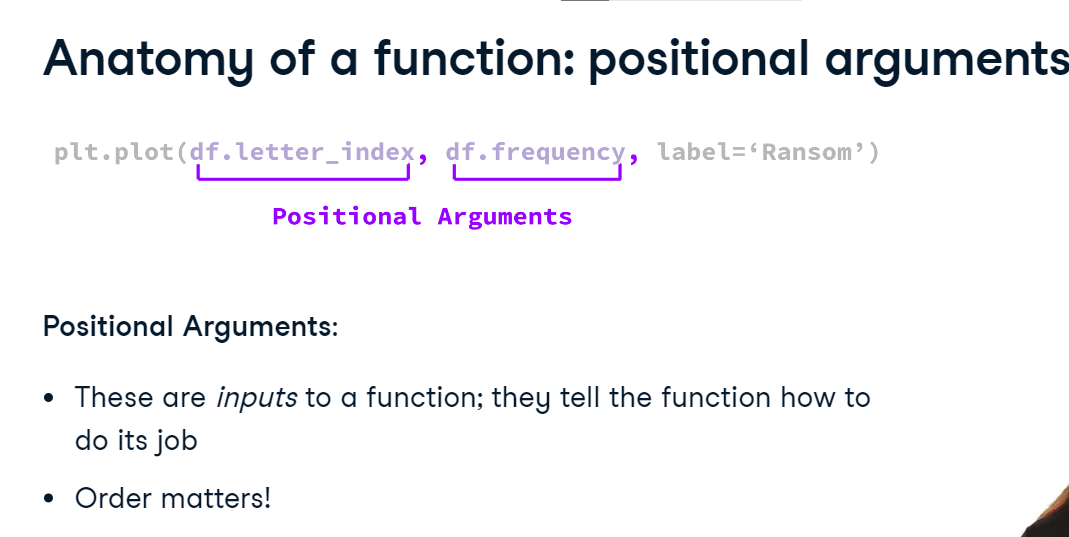
**Anatomy of a function: function name**

First, let’s look at the function name. The function name has two parts: the first tells us what module the function comes from. In this case, it's from plt, which was the alias we used when we imported matplotlib. The second part (which comes after the period) is the name of the function: plot. After the name of the function, comes a set of parentheses. The inputs to the function will all come inside of these parentheses.



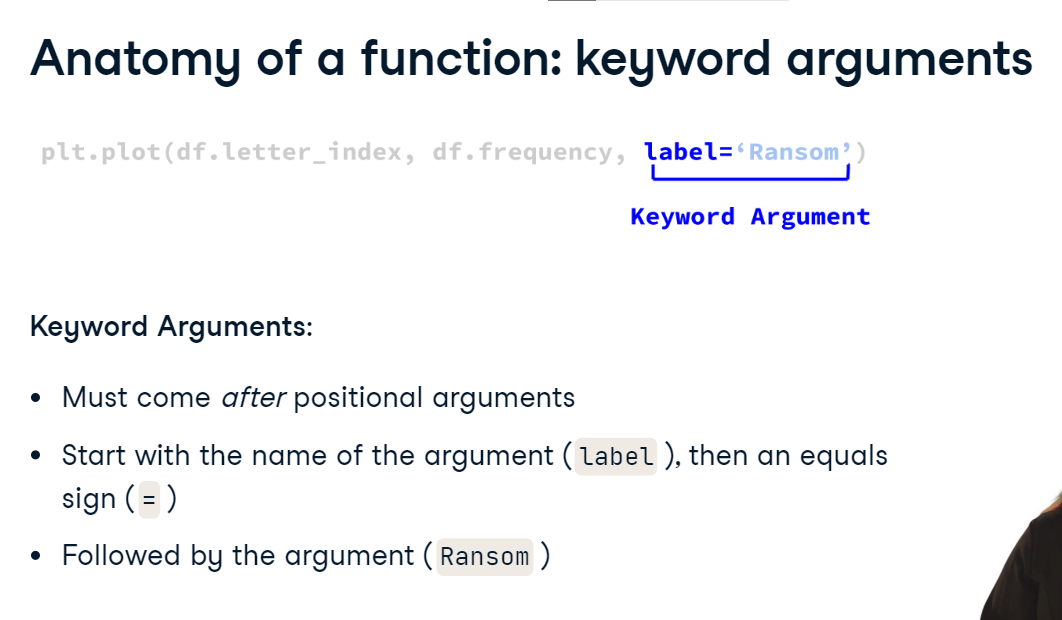
**Anatomy of a function: positional arguments**

Positional arguments are one type of input that a function can have. Positional arguments must come in a specific order. In this case, the first argument is the x-value of each point, and the second argument is the y-value of each point. Each argument is separated by a comma. If you forget the comma, you will get a syntax error. It's good practice to put a space after the comma, but your code will run even if you forget that space. Keyword arguments come after



**Anatomy of a function: keyword arguments**

positional arguments, but if there are multiple keyword arguments, they can come in any order. In this case, the keyword argument is called "label". After the equals sign, we've put the actual input to the function, which is "Ransom". Eventually, this argument will let us create a legend for our graph.



**Common function errors**

two common errors for functions: If you get a syntax error, you might be missing commas between each argument. Both positional and keyword arguments need to be separated by commas. Alternatively, you might be missing a parenthesis at the end of your function. While you write code, syntax highlighting in the script editor will help remind you to close your parentheses.