**Slide 6**:

In the real world, we'll sometimes only perform an action if a certain condition is met. This is the gist of conditionals in coding. For example, we may want to launch a rocket only if the temperature is above 32 degrees. Let's start by learning about the different conditions we can check to help us decide whether to perform an action or not.

## Conditions

Conditions are what need to be met before we change tasks. We can find out if a condition is met by comparing two values. To do this we use something called logical operators. Logical operators will return either True, if the statement is true, or False, if the statement is false. Remember: True and False values are called booleans.

For example, if our condition is to check whether the temperature is above freezing (to see if a rocket launch would be delayed) our condition would be temperature > 32. This would give us True if the temperature is above freezing and False if the temperature is at or below freezing.

Here are some common logical operators:

* Equals: x == y
* Not Equals: x != y
* Less than: x < y
* Less than or equal to: x <= y
* Greater than: x > y
* Greater than or equal: x >= y

Try to play around with some logical operators in Jupyter Notebooks. Notice how the single equal sign and double equal signs have different meanings. The single equal assigns a value to a variable, while the double equals compares values. This difference is a common gotcha beginner programmer can face.

Predict what will print and then run the following code cell to make sure you've understood this concept.

You can also use these operators on other variable types. Run this code cell to compare strings.

For reasons that are beyond the scope of this module, sometimes when comparing variables, you must use commands like "in". This just checks whether a word is inside the variable.

Now we know how to write the conditions to determine whether we want an action to execute or not, we can begin to specify what action we want to occur.

**Slide 8**:

We use conditional statements to tell the computer what to do if some condition is met, or is not met: do this; if not, do something else.

## If statement

The most basic conditional statement is the if statement. The if statement checks whether a condition is true or not true. If the condition is true, the if statement runs the code that's defined inside the statement. If the condition is not true, any code that's defined inside the if statement is skipped.

Some real world examples are:

* If the temperature is above freezing, the rocket will launch.
* If oxygen levels drop, grab a space suit and oxygen tank.

In Python, the format for an if statement is:

It's crucial to include the colon : at the end of the if statement, and also to indent each line of code that's defined inside the if statement. The indented lines of code will run when the if condition evaluates to True.

The following example shows how to use an if statement in Python. We use the if statement to test whether the count of basalt rocks is 0. If the count is 0, both print() statements run and display their output. If the count of rocks is not 0, the first print() statement is skipped, and only the second print() statement runs.

Notice how the first code cell above printed "We have found no basalt rocks." and "Done checking basalt rocks." because we assigned 0 to the varaible basalt and then the if-statement compared the value of the variable basalt to 0, resulting in a true statement. Since the if-statement is true, we run the code inside the if-statement (printing the first sentence), then the code exits the if-statement and prints the second sentence.

The second code cell above prints "Done checking basalt rocks." because we assigned 1 to the variable basalt and are still comparing the value of the variable basalt to 0, resulting in a false statement. Since the if-statement is false, we do not run the code inside the if-statement and move to the code that is after the if-statement and only prints the second sentence.

## Else statement

The else statement extends the if conditional statement and allows coders to have more control over what happens depending on the condition result. An else statement must be written after an if statement. The program always executes the code that's defined inside the else statement when the condition in the if statement is not met.

Notice in the above two code cell examples, they are similar to the first two code cells in this notebook, but this time if the statement is false, the code enters the else-statement and prints that sentence. So each code cell above will print two sentences, depending on if there were no basalt rocks or at least 1 basalt rock.

## Else-if statement

Python also supports else-if checks, for further control on actions based on data. The elif statement is written after an if statement and before an else statement. You can have as many elif statements as you want between the initial if statement and the final else statement.

The elif statement is just another if statement. When the conditions in the initial if statement are not met, the program checks whether the conditions in the next elif statement are met. When the conditions for the elif statement are met, the code that's defined inside the elif statement is run. Any code that's defined in a subsequent elif or else statement is skipped.

Here's an example of how an elif statement is used. Predict what will happen and then run the code cell to make sure you've understood it:

Notice that this is very similar to the previous examples in this notebook. The difference is we do a second check to see if we have found exactly 1 basalt rock.

Try changing basalt = 1 to other values; for example: basalt = 0 or basalt = 5.

**Slide 10**:

Iterations, in programming, allow coders to repeat a set of instructions until a condition is met. Think about this as being stuck in a loop that will continue until something tells you to break out.

## While loop

The while loop is one of two iteration types you'll learn about. In this loop you must specify a condition first and then include the code that you want the loop to iterate over. The loop will first check if the condition is True and if it is then it will look at the code inside the loop. When the condition becomes False, the code in the loop will be skipped over and the program will continue executing the rest of your code. If the condition in the loop is False to begin with, the code within the loop will never execute. During a single loop, the program will then go through the loop and run the code. Once the code is finished, it will look back at the condition and see if it is still True. It's essential to change the variables in your loop to eventually have a condition say that it is False, or else an infinite loop will occur.

As shown in the code below, to write a while loop you first must type "while" and then the condition you'll check before every loop. End the line with a colon and be sure to indent the next line, which will be the actual loop. The code below will print out a countdown for a rocket. As you can see, the countdown variable in the condition section decreases in every loop until it reaches -1, in which case the condition is False and the loop ends.

Predict what will happen when you run this code, then click the run button to verify you've understood.

In the example below, the condition is never met and the loop will continue forever (if we don't stop it). In this code, the developer forgot to decrease the timer variable, so the condition is always true.

This is an infinite loop and you must either wait for Python to terminate it or click the stop button at the top of the window. It's best to avoid infinite loops - if that wasn't already apparent.

## For loop

For loops essentially perform the same task as while loops, they tend to focus on iterating a set number of times. For loops are great when you want to go through a list and look at every single element. In the code below, we make a list and then go through all the elements and print them out.

**Slide 11**:

We can use functions to make our code easier to read and easier to reuse. For example, if you wrote code to count the number of basalt rocks, it would probably be almost identical to the code needed to count highland rocks. Instead of making an if statement to check the type of every single space rock, we can give a function the space rocks and it can run the same code to check which type of rock it is each time. Print() is an example of a function where the code to display output to the screen is the same, and the only difference is what you want to display. In this unit, you'll learn how to write your own function.

To make a function in Python, we must first tell Python that it is a function. We use the word "def" to define a function (tell Python that the following code is a function). You must then include a name for your function, parenthesis, and a colon. Finally, at the end of your function you must include the "return" key word. This tells Python the function is done and to return to the point where you called the function. Notice that all code within the function, including the return statement, should be indented.

Now let's play around with an example of making our own functions.

Run this code cell to see how functions work:

As you can see in the code, we start by defining a function that prints the countdown and rocket launch announcement. Then we add code that will call the function. If we had multiple rocket launches, we could just call the OutputRocketText function instead of typing out the whole print line. This would make our code much cleaner and less redundant.

**Slide 12**:

In many applications of Python, you'll need to use data from an external source. For example, an Excel file with data tracking the various parts of a rocket or a text file to read the rocks that a moon rover sees. In this unit, we'll learn how to read and write data from a text file using built-in Python functions.

## Setting up data

The first step is to create the file we want to read. You could also use an existing file that is already present - just place it in the same folder as the notebook you are running.

We'll first create a variable to hold the filename "text.txt".

Next, we'll use the open() function to create the file in Python. We'll make a new variable named fileObject to hold the data. The file will be created in the local folder - most often the same folder as the notebook.

Notice the second parameter is a string with the value "w". This indicates to *create* or *overwrite* any existing file. You can also pass "a" to append to a file, or omit the parameter to only allow read operations.

Next, let's use the write() function to write some text to the file. We want multiple lines, so we'll add a "\n" character to the end of the string.

Lastly, we need to *close* the file - this will flush any data that hasn't been written to the file on disk yet.

Now, let's do the same basic operations, but *read* back the contents of the file. We will use the same filename, but open the file for reading this time.

Finally, we'll have Python read this fileObject variable to give us a list of strings that we can explore with Python functions by using the readlines() function.

The readlines() function will take each line of the text file and make it an entry in a list. We'll store this list in another variable so we can print it out later. It's also good practice to close the fileObject when you're done with it. Remember we do this by calling close().

Now that we've read the file, we can print the contents of the file by iterating over the list:

The example above teaches us how to read the full data into Python. We can also read a text file and only go through certain parts of. This is called parsing in programming. To parse through the text we just imported, let's go line by line and print every line out. Since every line is an element in our list we can get the first line by looking at the first element in the list.

To read the second line from the text file, just get the second item in the list.

**Slide 13**:

Another interesting aspect of functions is that variables created inside them exist only inside the function. For example, this code would fail because we create the rocketNumber variable inside the function and then try to use it outside the function. Python doesn't know about that variable outside the function so it will produce an error.

Run this code cell to see the error produced by attempting to use a variable that only exists within the scope of the function:

Along side this, if you have variables created outside a function you can access them inside a function but you can't change them.

Run this code cell to see the error produced when you attempt to do this:

As you can see, this code produces an error as well, because we are trying to modify a variable that was created outside a function.

To combat this, we can do one of two things:

* Make a variable a global variable
* Give a variable to the function, so it knows what the variable is and then return it

The easiest way to change the value of a variable inside a function is to make it a global variable. Everything in the program can modify a global variable, even if it is modified inside a function. To make something a global variable, you must make a variable before you call the function and then include a line in your function that gives the global variable name:

Another way to modify variables inside a function is to use parameters. Parameters are when you give a function knowledge of a variable when you call it. To tell Python you want your functions to have parameters, use the following code. Try running this code cell to see that this works.

You can then use that variable inside the function, but it will have a new name, which is the name inside the parentheses. Also, the function will only change this new variable inside the function. To get the value from the function variable back into the code outside the function, you must return it. Do this by adding the variable name after the return keyword.

**Slide 15**:

In this module, you learned more advanced computer science topics such as conditions/conditionals, iterations, and common functions. You applied these topics to NASA-themed examples in the Python programming language.