**Housekeeping**

The classroom has embedded spaces where you can write and execute your C++ code directly in your browser; however, you may find it helpful to write C++ code on your own computer for this lesson (and not in the browser).

Here are some suggested programs for writing and executing your code locally:

* [Sublime Text](https://www.sublimetext.com/) as a general text editor
* Download an IDE (Integrated Development Environment) such as [Xcode](https://developer.apple.com/xcode/downloads/) (Mac only), [Visual Studio](https://www.visualstudio.com/downloads/) or any other IDE.

**Detailed Setup Instructions**

Big thanks to Michael Ikemann! Michael is a student in this Nanodegree and put together some **exceptionally** detailed documentation to help you get started with C++.

* [Windows C++ Setup Guide](https://docs.google.com/document/d/1PA3YcIKpANBZuHLX6tz7bVw60ug_DM1_V_yFKyrMP-s/edit?usp=sharing)
* [Mac C++ Setup Guide](https://docs.google.com/document/d/1FoIc9X0NcM-0y0dgXCzJcpu5lVXGFqS62mb1ABLDT1M/edit)

**Python C++ Cheatsheet**

We are providing you with a cheatsheet showing you Python syntax and the C++ equivalent. You might find this cheatsheet helpful as you learn to program in C++. You'll find that some of the syntax between Python and C++ is exactly the same or at least similar.

Supporting Materials

[**C++ Python CheatSheet**](https://d17h27t6h515a5.cloudfront.net/topher/2018/January/5a4d862b_c-python-cheatsheet/c-python-cheatsheet.pdf)

A close up of a sign

Description automatically generated

Elecia White was kind enough to offer her time and expertise as we developed this course (and the next in the Nanodegree).

Elecia White is a principal embedded software engineer at [Logical Elegance, Inc.](http://logicalelegance.com/), a small consulting firm in California. She enjoys sharing her enthusiasm for engineering and devices through writing and speaking. Elecia is the author of O’Reilly’s [Making Embedded Systems](https://www.amazon.com/Making-Embedded-Systems-Patterns-Software/dp/1449302149), host of [Embedded.fm](http://embedded.fm/), and editor/blogger at [Embedded.fm](http://embedded.fm/blog/). Her [past projects](https://www.linkedin.com/in/elecia/) include children’s toys, a DNA scanner, inertial measurement units, Fitbit, and a gunshot location system.

A close up of a bird

Description automatically generated

# Python and C++ Comparison

### Goal of this Module

The main goal of this module is to prepare you for writing C++ code. Because you are already familiar with coding practices in Python, the module will emphasize the similarities and differences between the two languages.

These lessons assume you are already familiar with general programming ideas like writing for loops, while loops, assigning values to variables, and writing functions. The fundamentals of how to code remain the same.

### Learning a New Programming Language

The best way to learn a new programming language is to practice writing code; therefore, most of this lesson involves learning C++ syntax and then practicing the syntax in an exercise.

By the end of the lesson, you should feel confident translating Python code into C++ code.

Throughout this lesson, you will be presented with Python code and the C++ equivalent. Below is an example of a simple program in Python alongside a C++ version. Both versions do exactly the same thing; they assign an integer 5 to the variable x. Then they output the value of x to the terminal.

Study each example line by line. Notice the similarities as well as the differences:

A screenshot of a cell phone

Description automatically generated

One similarity is variable assignment: x = 5. And the overall structure of the programs are the same.

But there are also a few major differences:

* the C++ program is wrapped with a function called main()
* x has to be explicitly defined as an integer
* instead of using print() to output the results to terminal, the code uses std::cout

You are going to learn about all of these facets (and more) in this C++ lesson.

Yes that's right. In C++, the #include statement is equivalent to an import statement in Python. In this case, the program includes the iostream file where std::cout is defined.

A screenshot of a social media post

Description automatically generated

# Python is Dynamic, C++ is Static

### Dynamically Typed versus Statically Typed

On the surface, Python and C++ have a lot in common; for instance, you'll see that if statements, for loops, and basic mathematical expressions are quite similar.

But under the hood, Python and C++ have fundamental differences. One major difference is that C++ is **statically typed** whereas Python is **dynamically typed**.

Take a look at this Python code:

vehicle\_doors = 4

vehicle\_speed = 3.0

vehicle\_acceleration = 1.5

vehicle\_on = **True**

vehicle\_gear = 'D'

vehicle\_doors = vehicle\_doors + 1

Python automatically figures out that vehicle\_doors is an integer, vehicle\_speed is a float, and vehicle\_on is a boolean variable. Variable assignment is **dynamic**. In Python, you do not need to specify the type of value that will go into a variable.

Did you notice the typo "vehicle\_dors" instead of "vehicle\_doors"? That is legitimate python code, and it won't produce an error.

In C++, none of the above code would work. You need to declare the variable type before you define a value; therefore, C++ is a **statically typed** language. Below is a C++ version of the code:

**int** vehicle\_doors;

**float** vehicle\_speed;

**float** vehicle\_acceleration;

**char** vehicle\_gear;

**bool** vehicle\_on;

vehicle\_doors = 4;

vehicle\_speed = 3.0;

vehicle\_acceleration = 1.5;

vehicle\_gear = 'D';

vehicle\_on = True;

vehicle\_doors = vehicle\_doors + 1;

If you had typed: vehicle\_dors = vehicle\_doors + 1;, you would get an error. That is because the vehicle\_dors variable was never defined.

## Declaring Variables in a Statically Typed Language

In this quiz, you will write integer variable declarations in C++. Read through the code below and fill in the TODO sections:

### Variable Assignment Python vs C++

What if the quiz had been in Python instead of C++? Remember, Python is a dynamically typed language whereas C++ is statically typed. In Python, you can assign values and Python automatically figures out what type of variable you wanted to use; however, when programming in C++, you need to declare the variable type prior to assignment.

### C++ Tip

In the C++ quiz, you might have written a statement like:

**int** integer\_two;

integer\_two = 9;

You can also define and assign a variable in one line of code like this:

**int** integer\_two = 9;

### Variable Assignment Python vs C++

What if the quiz had been in Python instead of C++? Remember, Python is a dynamically typed language whereas C++ is statically typed. In Python, you can assign values and Python automatically figures out what type of variable you wanted to use; however, when programming in C++, you need to declare the variable type prior to assignment.

### C++ Tip

In the C++ quiz, you might have written a statement like:

**int** integer\_two;

integer\_two = 9;

You can also define and assign a variable in one line of code like this:

**int** integer\_two = 9;

# Basic C++ Data Types

Now you know how to declare a variable in a statically typed language like C++. The C++ language has a handful of basic data types that you can use directly in your programs. These include integers, floats, and characters. Here is a table showing the most important basic data types that you will be using in the lesson:

| **data type** | **declaration** |
| --- | --- |
| integer | int |
| floating point | float |
| double floating point | double |
| character | char |
| boolean | bool |
| valueless | void |

Some of these data types might look a bit unfamiliar. Here are some examples of each type:

**integer**

integers are whole numbers like -20 5 700 -19

**floating point**

floating points are real numbers containing decimals like 5.109 199.25 -1.278

**double floating point**

A double floating point can hold more decimals than a floating point; the tradeoff is that a double floating point requires more memory. The next part of the lesson goes into more detail about floating points versus double floating points.

**character**

The char type definition is for ASCII characters. ASCII represent the English language Roman alphabet and common mathematical symbols. A char variable can only hold one letter at a time; you cannot use a char type definition to represent a string.

examples of characters: a U l & @

**boolean**

Booleans are variables containing either true or false.

**valueless**

The void type definition is used for special cases. You cannot declare a void variable in C++. You'll find that void is used when a function does not return anything; a function might print something out to the terminal but not return a value.

# Floating and Double Floating Point Numbers

In the last section, you saw a table with fundamental C++ data types and their declarations:

| **data type** | **declaration** |
| --- | --- |
| integer | int |
| floating point | float |
| double floating point | double |
| character | char |
| boolean | bool |
| valueless | void |

What exactly is the difference between a float and a double?

**A screenshot of a social media post

Description automatically generated**

Both data types can represent numbers containing decimals:

**float** x = 5.79;

and

**double** x = 5.79;

However, a double can store about twice as many digits as a float. But there is also a cost; a double requires more memory.

As you'll see in the demonstration below, storing more digits can be important especially when calculations require precision.

### Demo: Floating versus Double Type Definitions

The following code will show you the difference between a float data type and a double data type. The code assigns the number 11.0 to a float variable as well as a double variable. Next, each variable is divided by one-hundred thousand.

A for loop then sums each variable one-hundred thousand times to see how close each variable can get to 11.0. You will see that neither variable gets back to exactly 11.0 because memory can only hold a finite number of decimal places. But, the double variable gets closer to exactly 11.0.

Read through the code and then hit the "Test Run" button to see the output of the demonstration.

# Common Errors and Error Messages

As you to start to write your own C++ code, you might end up with some errors when you try to run your code. C++ errors can be very long and difficult to read even with something simple like a missing semi-colon. In this section, you are going to see some common errors you might run into when writing your C++ code. The goal is to become comfortable debugging your own code.

Here is a simple program that you saw in the beginning of the lesson. The program defines an integer x, assigns the value 5, and finally prints the results to terminal.

**#include <iostream>**

**int** **main** ()

{

**int** x;

x = 5;

std::cout << x << std::endl;

**return** 0;

}

### Semi-colon Errors

What happens if you forget to end a line with a semi-colon?

**#include <iostream>**

**int** **main** ()

{

**int** x

x = 5;

std::cout << x;

**return** 0;

}

main.cpp:6:7: **error**: expected ';' **at** **end** **of** declaration

int x

^

;

1 **error** generated.

This error message says that code line 6 at the 7th character in main.cpp should have ended in a semi-colon. Every command in C++ needs to end with a semi-colon.

### Declaring and Defining Variable Errors

What about forgetting to declare a variable?

**#include <iostream>**

**int** **main** ()

{

x = 5;

std::cout << x;

**return** 0;

}

main.cpp:6:2: error: use of undeclared identifier 'x'

x = 5;

^

main.cpp:8:15: error: use of undeclared identifier 'x'

std::cout << x << std::endl;

^

2 errors generated.

This produced two errors: one for each time the x variable appeared. The error says that on line 6 at the 2nd character, the variable x needs to be declared. The same error occurs at line 8, character 15.

The undeclared identifier errors means that the variable needs a data type definition like int x;.

### Namespace Errors

What happens if you forget to include std in the line std::cout?

**#include <iostream>**

**int** **main** ()

{

**int** x;

x = 5;

cout << x;

**return** 0;

}

main.cpp:9:2: error: use of undeclared identifier 'cout'; did you mean

'std::cout'?

cout << x;

^~~~

std::cout

/Applications/Xcode.app/Contents/Developer/Toolchains/XcodeDefault.xctoolchain/usr/bin/../**include**/c++/v1/iostream:54:33: note:

'std::cout' declared here

extern \_LIBCPP\_FUNC\_VIS ostream cout;

^

1 error generated.

All eleven lines refer to just one error! If you read the error line by line, however, you can get a lot of useful information. The most important part comes at the beginning where you're told that line 9, character 2 has an undeclared identifier cout. The error message tries to help by mentioning you probably meant to use std::cout.

The rest of the error tells points you to the file where std::cout was originally defined.

### Library Include Errors

What happens if you forget to include the standard library file that defines std::cout?

**int** **main** ()

{

**int** x;

x = 5;

std::cout << x;

**return** 0;

}

main.cpp:7:2: error: use of undeclared identifier 'std'

std::cout << x;

^

1 error generated.

The error says that C++ does not recognize what std means on line 7, character 2.

The definition of std is in the iostream file of the standard library, which needs to be included at the top of the program with the line #include <iostream>. Otherwise, your program won't recognize what std means.

### Putting it All Together

So far, you've seen what happens when your program outputs one error at a time. What about when there are multiple errors?

**int** **main** ()

{

x = 5;

cout << x;

**return** 0

}

main.cpp:4:2: error: **use** **of** undeclared identifier 'x'

x = 5;

^

main.cpp:6:2: error: **use** **of** undeclared identifier 'cout'

cout << x;

^

main.cpp:6:10: error: **use** **of** undeclared identifier 'x'

cout << x;

^

main.cpp:8:10: error: expected ';' after return statement

return 0

^

;

4 errors generated.

You get a list of errors starting from the top of your program and working down.

Undeclared identifier implies a variable or function needs a definition. And the semi-colon errors reminds you that all C++ commands need to end in a semi-colon.

When you run your code, you might end up with a very long list of errors that can be difficult to decipher. Usually, the first line of the error has the most important information, so start by looking at the top of the output. If you cannot figure out what an error means, try copying the error text and pasting it into a search engine. Oftentimes, there are resources online explaining what the error is and how to fix it.

# Functions: Python vs C++

In both Python and C++, functions have the same role; functions group statements together to perform some task. Functions help you avoid copying and pasting the same code over and over again.

The syntax for writing functions is slightly different for mainly three reasons:

1. Python detects the end of a code line based on seeing a carriage return and new line feed. C++ uses a semi-colon for the same purpose.
2. Python uses indentation to group code statements together, but C++ uses curly braces.
3. Python is dynamically typed while C++ is statically typed. Much like how you declared variables, you need to declare your functions.

Let's start off with a simple function and compare the Python and C++ code side-by-side.

This function takes in a velocity and time. These are multiplied together to calculate a distance. Besides the differences in syntax, pay special attention to:

* the function declaration
* variable declarations
* what code goes inside main() and what code goes outside of main

A screenshot of a cell phone

Description automatically generated

### Dissecting the Code

So the C++ code looks much longer than the Python code because the C++ has some extra parts. You are going to dissect this code piece by piece.

The code starts off with

**#include <iostream>**

That is importing the iostream part of the C++ Standard Library. You need that line of code in order to use cout.

After importing the necessary libraries, you see a function declaration.

**float** **distance**(**float** velocity, **float** time\_elapsed);

That line of code informs your C++ program that there is a function called **distance**. The function accepts two float variables and returns a float. The first float variable is called velocity and the second float variable is called time\_elapsed.

Then comes the **main** function. All C++ programs require a main() function that returns a zero. The main() function calls the distance function and outputs the results to the terminal.

**int** **main**() {

std::cout << distance(5, 4) << std::endl;

std::cout << distance(12.1, 7.9) << std::endl;

**return** 0;

}

and finally, you have the function definition

**float** **distance**(**float** velocity, **float** time\_elapsed) {

**return** velocity \* time\_elapsed;

}

You have seen the **main()** function before, so this isn't the first time you have seen how functions work in C++. Notice how the main function and the distance function have very similar syntax. The only difference is that the main function does not accept any arguments and returns an integer of value zero; on the other hand, the distance function accepts two floats and returns a float.

You also don't make a separate declaration for the main function. On the next page, you'll get more practice with understanding functions and writing functions in C++.

# Anatomy of a Function

You have seen how to write a function in C++. More generically, a C++ functions consists of a function declaration and a function definition.

Because C++ is statically typed, you need to specify the data types for the function input variables and the data type of whatever the function returns.

*// function declaration*

returndatatype **functionname**(datatype variable\_a, datatype variable\_b, etc.);

*// function definition*

returndatatype **functionname**(datatype variable\_a, datatype variable\_b, etc.) {

statement\_1;

statement\_2;

etc...

**return** returndatatype;

}

# Functions with More than One Output

In Python, you can write a function that has multiple outputs. For example,

*## Python Code*

**def** **distance**(velocity, time\_elapsed):

**return** velocity \* time\_elapsed, velocity / 2

would output both velocity \* time\_elapsed and velocity / 2.

In C++, functions can only have one output. There are work-arounds, but these work-arounds go beyond the scope of this module.

# C++ Tip: Function Declarations

You do not have to put the function declaration at the top of your code to get a working solution. Much like how you can declare and define a variable simultaneously, int x = 5;, you can also declare and define a function simultaneously.

The following code would work as well:

*// C++ code*

**float** **distance**(**float** velocity, **float** time\_elapsed) {

**return** velocity \* time\_elapsed;

}

**int** **main**() {

std::cout << distance(5, 4) << std::endl;

std::cout << distance(12.1, 7.9) << std::endl;

**return** 0;

}

But note that you have to define your function before the main() function not after; otherwise your code would try to call the distance() function but not have a definition for the function.

However, we encourage you to always declare your functions before main() and define them after main. In the next lesson in the nanodegreee called practical C++, you will learn why; declaring and defining your functions separately helps keep your code organized as your programs become more complex.

Andy learns about typedef and is reminded to always be suspicious of repeated code.

Note: Elecia and Andy use the word "vector". For now you should think of a vector as something similar to a Python list. So when Elecia says "vector vector float", she is referring to a two-dimensional list (a list of lists) whose elements are floats.

The following line of code can be used to define an entirely new type called t\_grid which is a vector of vectors of floats (for now you can think of vectors as being similar to Python lists).

**typedef** vector < vector <**float**> > t\_grid;

Anywhere you would have written vector < vector <float> >, you can now just write t\_grid!

In the video below Andy discovered something surprising while translating his histogram filter code from Python to C++. He could have two **different** functions which each had the **same** name and this didn't cause any problems.

You can find the code Elecia and Andy discuss below the video.

**bool** **close\_enough**(**float** v1, **float** v2) {

**if** (abs(v2-v1) > 0.0001 ) {

**return** false;

}

**return** true;

}

**bool** **close\_enough**(vector < vector <**float**> > g1, vector < vector <**float**> > g2) {

**int** i, j;

**float** v1, v2;

**for** (i=0; i<g1.size(); i++) {

**for** (j=0; j<g1[0].size(); j++) {

v1 = g1[i][j];

v2 = g2[i][j];

**if** (abs(v2-v1) > 0.0001 ) {

**return** false;

}

}

}

**return** true;

}

The **signature** for the normalize function is:

vector< vector<**float**> > normalize(vector< vector <**float**> > grid);

# Control Statements

Now that you know how to declare variables and write functions, you are well on your way to C++ proficiency.

So far, the programs you've worked with have been relatively simple. You will need control statements to make more complex programs. Control statements like **if** and **for** are fundamental to many programming languages. They allow you to make conditions about when and how often code statements should be run.

In this section, you will learn to use C++ **if** statements and the associated boolean logic.

The next section will cover looping with **while** and **for**. And then finally, you will learn about the **switch** statement. Python has equivalents for **if**, **while** and **for**; however, the **switch** statement does not exist in Python.

After you learn about control statements, you will be ready to write more sophisticated C++ programs.

### Python vs. C++ If

Below is an example of a Python set of if statements versus the C++ equivalent.

You will see that the logical structure is exactly the same but the syntax is slightly different. You could imagine that the code below would be part of a traffic light classification program that tells a vehicle the current color of a traffic signal.

A screenshot of a cell phone

Description automatically generated

A generic if else statement in C++ looks like this:

**if** (<some criteria>) {

statement\_1;

statement\_2;

.... etc.

}

**else** **if** (<some other criteria>) {

statement\_1;

statement\_2;

.... etc.

}

**else** {

statement\_1;

statement\_2;

.... etc.

}

### Boolean Logic

You need boolean logic to make if statements useful. Boolean logic works the same way in Python and in C++; some of the syntax is the same and some is slightly different.

Here is a table showing comparison operators in the two languages:

| **Operator** | **Python** | **C++** |
| --- | --- | --- |
| equal | == | == |
| not equal | != | != |
| greater than | > | > |
| less than | < | < |
| greater than or equal | >= | >= |
| less than or equal | <= | <= |

Yes, indeed, comparison operators are exactly the same in the two languages!

What about logical operators such as **and**, **or**, as well as **not**?

These are not the same in the two languages:

| **Operator** | **Python** | **C++** |
| --- | --- | --- |
| and | and | && |
| or | or | || |
| not | not | ! |

The or operator in C++ is represented by two vertical bar characters. On English keyboards, you can find the vertical bar key above the enter key.

### Python vs. C++ While

Below you'll see an example of a Python while loop compared with a C++ while loop. They look quite similar!

The example starts with an integer 15 in the elapsed\_time variable. With each iteration, the integer is reduced by 1. Once the elapsed\_time reaches zero, the program leaves the while loop.

A screenshot of a cell phone

Description automatically generated

A generic while statement looks like this:

**while** (<some criteria>) {

statement\_1;

statement\_2;

statement\_3;

....etc

}

### Python vs. C++ For loops

For loop syntax is very similar in Python and C++ as well.

This following example is like the while loop except the count variable increases instead of decreases (this does not necessarily need to be the case, we just did it that way here).

A screenshot of a cell phone

Description automatically generated

One thing to note is how Python iterates through the **i** variable versus how C++ does the iteration.

For python the iterator was defined here:

i **in** range(0, elapsed\_time)

Python's range() function generates a list of numbers, which in this case would be

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14]

Then Python assign each of these values in turn to the **i** variable until reaching the end of the list.

For C++, the iteration happens in this line of code:

(**int** i = 0; i < elapsed\_time; i++)

First you declare the variable **i** and assigned a value (in this case zero). The for loop then checks if

i < elapsed\_time

If true, then the code block is run and then **i** increases by one. The code i++ is equivalent to saying i=i+1.

When

i = 14

that will be the last time that the code block runs. The code checks that 14 is less than 15, runs the code block and increases **i** to 15. Then the code checks if 15 is less than 15. Since that is false, the for loop does not run again.

One item to note in the above playground is that C++ considers something enclosed in single quotes ('a') to be a char, while double quotes ("fast") is a string.

# Switch

A switch statement is very similar to an if clause. In fact, you can write a program that does the exact same thing with either a switch statement or a series of if-else clauses.

Then why bother using a switch statement? For reasons we won't detail here, switch statements can oftentimes be faster to execute. Many programming languages have a switch statement including Java, Javascript, PHP, C++ among others; Python is an exception.

Since there is no Python switch statement, we will compare a set of if-else C++ clauses with a C++ switch statement.

Click on the image to zoom in.

A screenshot of a cell phone

Description automatically generated

The output of the code would be

Not Moving - Neutral

Your car is currently in gear: N

Let's break down what is happening in the switch statement:

**char** gear\_status = 'N';

**switch**(gear\_status) {

**case** 'D' :

std::cout << "Driving Forward" << std::endl;

**break**;

**case** 'N' :

std::cout << "Not Moving - Neutral" << std::endl;

**break**;

**case** 'P' :

std::cout << "Not Moving - Parked" << std::endl;

**break**;

**case** 'R':

std::cout << "Driving in Reverse" << std::endl;

**break**;

}

Each time **case** appears, the code checks to see if the gear\_status variables matches the case. Once gear\_status finds a matching case, the code inside the case runs.

In C++, the switch statement was designed to run the code in the matching case and then all of the cases below. You need the **break** lines of code if you want your code to leave the switch after executing the matching case.

In other words, if the code were written without using break,

**#include <iostream>**

**int** **main**() {

**char** gear\_status = 'N';

**switch**(gear\_status) {

**case** 'D' :

std::cout << "Driving Forward" << std::endl;

**case** 'N' :

std::cout << "Not Moving - Neutral" << std::endl;

**case** 'P' :

std::cout << "Not Moving - Parked" << std::endl;

**case** 'R':

std::cout << "Driving in Reverse" << std::endl;

}

std::cout << "Your car is currently in gear: " << gear\_status << std::endl;

**return** 0;

}

the code would still skip the 'D' case. But once the code found a match with the 'N' case, the code in the 'N', 'P', and 'R' cases would execute.

### Switch Limitations

If-else statements are much more flexible than switch statements. In fact, the case clauses in switch statements can only make comparisons between integer values. Switch cases can also compare characters like in the example code because C++ is actually converting the characters to integers.

On the other hand, if statements can make comparisons between floating point numbers as well as between integers.

The general form of a switch statement looks like this:

**int** variable = integer;

**switch**(variable) {

**case** 1:

code statements;

**break**;

**case** 2 :

code statements;

**break**;

**case** 3:

code statements;

**break**;

**case** 4:

code statements;

**break**;

**case** etc ...

}

### Switch Statement - Playground

Practice writing a switch statement in the playground below. The code comments will help you get started. You can run your code with the "Test Run" button and then compare your solution with "solution.cpp".

# C++ Libraries

You have learned how to declare variables, write functions, and use control statements. Those are the basic building blocks of any programming language, and you are ready to write programs in C++.

But, what if you want to store a string in a variable or do more advanced math like taking the square root of a number? Just like Python, C++ also uses pre-built libraries to help make programming easier. In python, you use these libraries with an

**import**

statement.

In C++, you use

**#include**

You have already been using a file called "iostream" from the [C++ Standard Library](https://en.wikipedia.org/wiki/C%2B%2B_Standard_Library). The "iostream" file contains functions and classes for outputting to a terminal and also reading in from a terminal.

You were able to take advantage of these pre-built functions by including the appropriate file like so:

**#include <iostream>**

That include statement essentially pastes the [iostream](http://en.cppreference.com/w/cpp/header/iostream) file to the top of your code and gives you access to all of its functions and classes.

As your programs become more complex, you will rely more and more on C++ libraries.

### C++ Standard Library

The [C++ Standard Library](http://en.cppreference.com/w/cpp/header) has a lot of functions and classes like a definition for a string, arrays, tuples, functions for reading in and outputting files, random number generators, definitions for complex number variables, mathematical functions and many other functions as well.

And the C++ Standard Library comes with a C++ installation.

Besides this [list](http://en.cppreference.com/w/cpp/header) of files in the Standard Library, search engines are your best friend for finding functionality and libraries in C++. For example, if you are not sure how to do something in C++ like using strings, open a search engine and type "C++ string syntax". You will find many examples online of how to use strings in C++ and quickly find an example like the following:

##### Example: Strings

To use the part of the library that defines strings, you would include a line at the top of your main.cpp file like:

**#include <string>**

**#include <string>**

**int** **main**() {

std::string stringvariable = "stringvalue";

**return** 0;

}

##### Example: Math

Here is another example from the standard library.

**#include <iostream>**

**#include <cmath>**

**int** **main** ()

{

*// calculate*

std::cout << pow(3.4, 4);

}

The pow() function is raising 3.4 to the fourth power.

Cmath includes logarithmic and exponential functions, power functions, and trigonometric functions. You can see a full list [here](http://www.cplusplus.com/reference/cmath/).

### Include Syntax

When you learned about structuring functions, you saw two different include statements:

**#include <iostream>**

**#include "distance.h"**

In fact, you could also write,

**#include "iostream"**

**#include "distance.h"**

but using quotes instead of brackets is less efficient. When using quotes, your program will first look for the iostream file in the main.cpp directory. When the program cannot find the file, the program will search where the standard library files are kept.

### Other Useful Libraries

As previously mentioned, the C++ Standard Library generally comes with a C++ installation; however, there are many other useful C++ libraries that you install separately. Each library will have its own installation procedure and usually comes with instructions. Again, search engines are your best friends when trying to find and install libraries.

This [link](http://en.cppreference.com/w/cpp/links/libs) contains a list of many open source C++ libraries. In the list, you will see all kinds of libraries for math, gaming, computer vision, machine learning, as well as many other topics.

In the next lesson, you will learn to use the C++ vector library.