Programming Language and Development Concepts

by Sophia

WHAT'S COVERED

In this lesson, you will learn about the basics of programming languages. You will also explore the development lifecycle. Specifically, this lesson covers:

Table of Contents

1. Programming Basics

Imagine trying to make a recipe for the first time. You would need a list of the specific ingredients and their amounts, as well as the detailed instructions of how to prepare the recipe. Similarly, a computer program consists of lines of code that the computer runs. They must run in a specific order.

EXAMPLE If you were baking a cake, you wouldn't put the cake in the oven prior to mixing the ingredients.

You must think about all of the steps in logical order. You will also need to think about where the ingredients (input) will come from and what sort of final product (output) the program will produce.

1a. Input, Processing and Output

Programs typically have three main types of operations that are executed in any program.

- **Input**. The input to the program allows data to be entered in through a variety of ways, including by you through the keyboard and mouse. It could also be from files on the computer, other hardware devices, images, sounds, or many other potential options. The input from these sources is stored and placed into the memory of the computer and can include all types of data that include text and numeric values.
- **Processing**. When it comes to processing data, there can be many different steps that are performed on

the data. It could be storing them, using them for calculations, validating the data, or sorting the data.

- **Output**. After the data has been processed, output is sent to the monitor, printer, email, file, report, or other means where individuals can view the resulting data. The goal of most programs is to take the raw data or input, process it into information that can be useful, and then output it to the user.

1b. Syntax

In order to write these programs, we need to use a programming language that the computer understands. As discussed, in this course you will be using Java. Each programming language has its strengths and weaknesses for the tasks at hand. Most programming languages have similar logic that can be transferred between the different languages. Each programming language has a specific set of **syntax** rules that needs to be followed. Computers are not smart enough to understand a program unless the syntax is correct.

1c. Compiled vs. Interpreted Programming Languages

As you have learned, programs are sets of instructions to tell a computer how you want it to behave. Compilers and interpreters act like language translators between humans and machines. They take human-readable code and convert it into machine-readable code. In a compiled programming language, the computer is able to translate the program directly into machine language. In an interpreted programming language, another program, called the interpreter, is necessary to help the computer read and execute the program.

When writing a program in a compiled programming language like C or C++, users have to compile it. The **compiler** takes the human-readable code that we write and translates it to**machine code**, also known as machine language, that can be executed by the computer's processor. If a program is successfully compiled, the compiler creates a file that can be run. A compiled program has limitations since it can only be run on the specific platform for which it was compiled.

IN CONTEXT
Imagine you found a great new recipe, but it's written in another language. You could send the recipe off to a friend that is fluent in the language and ask them to translate it for you. When they send the translated recipe back, you can easily read it in your own language. This example is similar to a compiled programming language. You can read the recipe because your friend converted the

language for you, and you can make the recipe again anytime you'd like because you have a translated version.

There are other programming languages, such as Python, that are interpreted rather than compiled. The interpreter reads the text instructions and carries them out. The interpreter has to be installed on the computer. One advantage of using an interpreted language is that it can be run on any computer. Although there are those differences in how compilers and interpreters function, the core purpose is the same with interpreters having an added step.

IN CONTEXT

Think back to your recipe written in a different language. Another option would be to invite your fluent friend over to interpret the recipe for you, step-by-step, as you make it. This example is similar to an interpreted programming language. Your fluent friend is the interpreter that converts the instructions for you, allowing you to follow along. In this scenario, you'd need your friend to sit with you again the next time you want to make the recipe because you can't read it without their help.

Java, on the other hand, has characteristics of both a compiled and an interpreted language. It must be compiled first, then it can be interpreted. The Java compiler (javac) is part of the Java Development Kit (JDK). When compiling a program, the compiler takes the human-readable code that we write (in text files with the extension .java) and translates it to **bytecode** (in binary .class files) that can be executed by the Java Virtual Machine (JVM). The bytecode differs from machine code because the bytecode is a low-level set of instructions that can be run on the abstract Java Virtual Machine (JVM) instead of directly on specific hardware.

BIG IDEA

When a Java program is executed, the JVM carries out the bytecode instructions (not unlike an interpreter). The JVM has to be installed on the computer where the program will be run. Instead of running the program directly on the computer's hardware, the bytecode is run through the JVM. This allows Java to be platform independent. As long as the bytecode and the virtual machine have the same version, a Java program can be run on any platform regardless of whether it is a Windows, MacOS, or Linux system.

IN CONTEXT

Think back to your recipe written in a different language. Let's imagine you sent your recipe off to your friend to translate it for you. They translated it first, but then they also brought it over to your place and used it to interpret for you, step-by-step, as you prepare the dish. This example is similar to a combination of a compiled and interpreted programming language. Your fluent friend translated the recipe for you first, then they were also gracious enough to stop over and interpret the instructions for you, allowing you to follow along.

TERMS TO KNOW

Input

Ways a program gets its data; user input through the keyboard, mouse, or other device.

Processing

Takes the data inputs and prompts and calculates the results.

Output
The results at the end of a program based on user input and system processing.

Syntax
The syntax is the "grammar" rules of the programming language. Each programming language (like Python) has its own syntax.

Compiler
A compiler scans an entire program and attempts to convert the whole program at once to machine code.

Machine Code
The computer uses 0s and 1s to perform tasks.

Bytecode
An intermediary step for code conversion between the programming language that you write and the machine code that a computer uses.

Interpreter
An interpreter takes the bytecode one line at a time and converts it to machine code.

Virtual Machine
A software program that behaves like a completely separate computer within an application.

---

2. Understanding the Development Lifecycle

Developers do not typically just sit down and start writing code. They break their work down into individual, logical steps. The **development life cycle** is a high-level process developers use for planning, creating, testing, and deploying an application. There are different approaches to the development life cycle, but at a high level, they are largely the same. The development life cycle outlines seven steps that can be carried out when tackling a problem. Some of those steps can be combined and some may not be needed for every program. The steps include:

STEP BY STEP

1. Determine what the problem being asked is.
2. Plan out the logic for the problem by breaking it down into a sequence of steps or algorithm.
3. Write the code to perform the algorithm.
4. Compile/translate the code to a format the computer can read.
5. Test the program to ensure there are no syntax or logical errors.
6. Deploy the program to be used.
7. Support the maintenance of the program.

It makes sense to start out by defining the problem you intend to solve. If it is not clear what the problem is, the program will be off course from the start. This critical first step may require some planning to gather specific details and requirements from stakeholders. Gathering input is essential when developers do not have knowledge about the domain for which the program is being built. In such cases, the developer relies on subject matter experts to provide input on the business processes.

IN CONTEXT

If you are tasked with writing a program for a high school that sends an email to students notifying them about school fees, you may have some of the following questions:

- Are the fees the same for all students?
- Are they different by year or perhaps by the courses that they take?
- When should the students be notified?
- Where do we find the student rosters?
- Are there exemptions to the fees?

These are not questions that a developer can answer without more information. In order to understand the problem, you may have to go to someone that can provide that type of information and guidance.

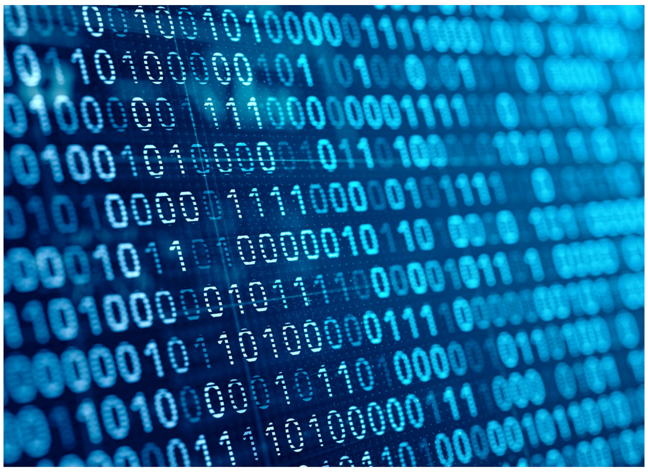
Once the problem is understood, the developer can begin to break it down further into logical steps. This will be discussed in upcoming tutorials, but the idea is that the problem needs to be broken down into individual and incremental steps so that the computer can carry out what needs to be done.

THINK ABOUT IT

Imagine that you had to direct someone wearing a blindfold through an obstacle course. You would need to be very specific about the steps to take and when. If you accidentally told the person to jump instead of ducking under a pole, he or she could get hurt. In the same way, programs need to be written in the correct logical order for them to work correctly. This is how you will approach formulation of an algorithm as the sequence of steps to solve the problem.

With the algorithms in place, the program can then be translated into a programming language. In our course, you will use Java. However, there are hundreds of languages that could be selected. Each has its own distinct advantages and disadvantages. Selection needs to consider several types of criteria. One criterion may be familiarity. There are some programming languages like C++, C#, and Java that are widely used in a variety of applications. If you are already familiar with a language, you may choose to use it. Another criterion to consider is to determine what languages are used in a customer's organization. Since a program may be developed by many individuals on different teams, having consistency in language choice among the developers is likely important. The availability of specific libraries and frameworks for a language is another important factor in the selection. For example, if we are developing code for a game engine called Unity, we would have to use C#, Boo (similar to Java), or JavaScript. We don't have any other language choices if we want to use Unity.

If there are errors in the syntax, they are normally caught by the code editor's syntax analyzer or when the code is compiled. If the program isn't written correctly, the compiler produces one or more error messages so the programmer can fix it. A **syntax error** would be similar to having an English program that said, "The ct and mouse were friends." This would result in an error since ct isn't a word in English.



Logic errors, or semantic errors, are issues that the computer can't catch based only on syntax. For example, if the program was meant to give everyone a 10% discount on their orders, but it was accidentally coded to give everyone a 100% discount, that wouldn't be caught by the compiler and would be a logical error. In order to find these errors, you will have to test the code and compare the results with the expected values.

Once the code has been tested, it can then be deployed for the program to be used in the purpose that it was originally developed for.

Once a program is deployed, there may be bugs or errors that come up that have to be addressed and patched. There may be improvements to performance that have to be addressed. All of these factors may require maintenance. In some cases, it may require some additional development which in turn would start right back at understanding the problem again.

TERMS TO KNOW

Development Life Cycle
A high-level process for planning, creating, testing, and deploying an application.

Syntax Error
An error that breaks one of Java's syntax rules and is detectable by the Java compiler.

Logic Error
An error in the logical design of the program that causes it to behave incorrectly, producing incorrect results. Also known as semantic errors. Logic errors are not detectable by the Java compiler.

SUMMARY

In this lesson, you learned about **programming basics**, including the three main types of operations of a program, **input, processing, and output**, as well as **syntax**, referring to the "grammar" rules of the programming language. Keep in mind that each programming language has its own syntax. You also learned the difference between **compiled vs. interpreted programming languages**. Finally, you explored how the **development life cycle** requires a high-level process for planning, creating, testing, and deploying an application. Developers generally do not just sit down and start writing code; rather, they break the process down into individual steps and build a program out of those individual steps.

Source: This content and supplemental material has been adapted from Java, Java, Java: Object-Oriented

Problem Solving. Source cs.trincoll.edu/~ram/jjj/jjj-os-20170625.pdf

It has also been adapted from “Python for Everybody” By Dr. Charles R. Severance. Source py4e.com/html3/

TERMS TO KNOW

Bytecode
An intermediary step for code conversion between the programming language that you write and the machine code that a computer uses.

Compiler
A compiler scans an entire program and attempts to convert the whole program at once to machine code.

Development Life Cycle
A high-level process for planning, creating, testing, and deploying an application.

Input
Ways a program gets its data; user input through the keyboard, mouse, or other device.

Interpreter
An interpreter takes the bytecode one line at a time and converts it to machine code.

Logic Error
An error in the logical design of the program that causes it to behave incorrectly, producing incorrect results. Also known as semantic errors. Logic errors are not detectable by the Java compiler.

Machine Code
The computer uses 0s and 1s to perform tasks.

Output
The results at the end of a program based on user input and system processing.

Processing
Takes the data inputs and prompts and calculates the results.

Syntax
The syntax is the “grammar” rules of the programming language. Each programming language (like Python) has its own syntax.

Syntax Error
An error that breaks one of Java’s syntax rules and is detectable by the Java compiler.

Virtual Machine
A software program that behaves like a completely separate computer within an application.