Programming Basics

A Primer

By Allen Moore

Introduction

There are easily hundreds if not thousands of computer programming languages. Some of the names you may have heard in the past include: Assembly Language, BASIC, C++, C#, COBOL, ColdFusion, dBase, Fortran, FoxPro, Haskell, Java, JScript, Machine Code, Pascal, Python, Ruby, VBScript, Visual Basic, VBA, Visual Basic .NET, Visual C++, Visual C#, and many others. Some of these languages are built into other software packages for use in programming macros. Some are designed specifically for web use. Some are designed for text-based programs. Others are designed for Graphical User Interfaces (GUIs) such as Windows. Some are very powerful and are used to create some of the most successful programs available. Others are very simplistic and are used more for hobby programming and less intense programs. However, for all of the different names (and, for some languages, types under those names), all programming languages do share many things. In this primer, we're going to look at some of the commonalities among programming languages. These are things that nearly every language makes available and nearly every program will use. One important thing to remember for all of these programming constructs is that the "Syntax" of every language is different. In other words, the way that you will actually type the code will vary from language to language. This is intended simply to give you an idea of how a program is built in general. There is not room in any single book to provide specifics on how to apply each of these items for every language. If you are interested in programming, you would be well served to consult with your local library, bookstore or online search engine for the answers to your language-specific questions.

Programming Syntax

Each programming language has its own way of allowing you to create different lines of code. In most cases, a line of code is a single command together with the attributes or parameters needed to run the command. For a few languages, you don't need to do anything except press enter to begin the next line of code. The drawback of that is if you have an extra-long line of code for some reason, you can't separate it into multiple lines for easier reading. For this reason, nearly all computer languages have some other way to determine when the next line of code begins. In many languages, you will end each line of code with a common character such as a semi-colon (;). In the early days of BASIC, each line of code had to begin with a line number. In CSS (the design language for web sites), each line of "code" or attribute specification ends with a semi-colon. It is important to know how each line of code is to end. A great many problems can be encountered by improperly terminating your lines of code (or not terminating them at all). In this case, the best case is that the program doesn't run at all. If the program does run, you may get some very undesirable results from the code as a result of not properly terminating your lines of code.

Chapter #1—Variables

Every programming language allows you to create, store information to, and pull information from variables. A variable is simply a unit of memory set aside to store some piece of information. In general, there are a few different data types available for variables: strings, numbers and Boolean.

- Strings (as they are called in most languages) contain some group of characters that is saved as text. This could include words, sentences, abbreviations, usernames, passwords, full paragraphs, or a single character. Most programming languages, when you are assigning text to a string, you must enclose the text with either single-quotes (') or double quotes ("). By using those characters, you are telling the language that the text inside is not to be translated as code but is rather to be used together as a group of characters. The group of characters inside the quotation marks is called a literal string as it is displayed by the language exactly as it is entered without any changes.
- Numbers, in most programming languages, can be specified to be either integer or decimal. If a variable is explicitly declared (in programming languages where it is necessary) as an integer, attempting to save a number with a decimal point and fractional parts to the variable may cause an error and the program to stop. In other languages, rather than causing an error, it will simply only save the integer part of the number. In still other languages, rather than doing either of these, it will change the data type of the variable to the appropriate type. Even within the integer and decimal data types, many languages offer more options. For instance, a regular integer can usually be a number between approximately 65,000 and -65,000 (2^{16}); a long-integer can be between approximately 2 million and negative 2 million (2^{21}); a byte is between 256 and -256 (2^{8}). All of these limits are based on powers of two and relate to how much memory space is allocated for the variable…it takes more memory to store the number 79,265 than it does 2,000 because of the number of powers of two that it incorporates. For the same reasons, some programs will offer various levels of decimal, also sometimes called floating point or double, types. For instance, a float data type might allow up to 38 decimal places while a double might allow up to 308 decimal places. As with the integers, the size requirements are based on the amount of memory required to hold the data.
- Boolean is simply a true or false variable. However, instead of storing the words "True" and "False" for the variable value, it will store a 0 for False and a 1 for True. In some programming languages, if the variable contains anything except for 0, it will return true for the variable.

Some programming languages require you to declare variables and their types before you use them in the program. Often, if you declare a variable, it is done either at the beginning of the program or at the beginning of a "function" or "procedure". By requiring you to declare the variables before you use them, the computer can set aside the necessary memory for the entire program. For languages that require you to declare a variable before using it, if you attempt to use a variable that has not been declared previously, it will cause an error and will usually cause the program to stop.

Each language has its own way of declaring, naming storing to, getting from, and manipulating its variables. Some languages require that the variable begin with a certain character such as $ or #. Nearly all languages require that a variable name not contain any spaces. In order to effectively name your variables, you will adopt some method of identifying them. For example, you may connect multiple words with an underscore (_) character (i.e. net_profit) or you may capitalize the first letter of each new word (i.e. NetProfit).

Variables are a very useful part of any program as they allow you to store information that will be needed later in the program in a location where you can easily manipulate the data or retrieve the data as needed.

Chapter #1 Review:

1) What is the difference between a Number, String and Boolean variable type?
2) What types of information might you store in each of the three data types related to the type of program you are interested in building?

Chapter #2—Arrays

Arrays, available in nearly every language, are a special kind of variable that can hold more than one piece of information. Most languages will allow you to create "multi-dimensional" arrays for use in your programs. Arrays can be thought of like a table of information where, in most cases, each cell contains data of the same data type. A one-dimensional array would be like a single-column table. Maybe it stores all of the names of the people on your team. A two-dimensional array (sometimes referred to as an array of arrays) is like a multi-column table. It may contain not the just names of your team members but also the age, grade level, and score on the last test. A three-dimensional array, is like a cubic table. Three- and more dimensions are much less common in programming. This is mostly due to the difficulty in visualizing how data is stored in more than three dimensions. While a computer can easily keep track of a 10-dimensional array, the humans programming and debugging them have a considerably more difficult time keeping the information stored correctly. As the number of dimensions increase, the task of maintaining the integrity of the data increases exponentially.

Accessing specific cells of an array varies from language to language. Most will allow you to refer to the cells by number. For instance, x[1][1] might refer to the first row and first column of the array called x. Other languages will allow you to refer to parts of an array by a key. For instance x[name] would refer to the data in a cell called name in the array called x.

One word on numbering in arrays. In most languages, the numbering of the cells in array begins with 0 instead of 1. For this reason, the last cell in an array will actually be one less than the total number of cells used. This is important to know especially if you use a looping construct (described below) to go through all of the elements in an array.

Chapter #2 Review:

1) What type of data might it store in an array variable for your program?
2) If an array contains 10 cells, in most programming languages, what number would refer to the last of the cells?

Chapter #3—Conditional Programming

All programming languages include commands that allow you to test the contents of a variable and make "decisions" as to the next steps in the program based on the results of that test. For instance, perhaps you have a variable that contains the age of your user. If the age is under 18, perhaps you include a certain text in your document; if the person is over the age of 18, you would include a different text. These types of decisions are made by conditional programming commands. Most programming languages use the same type of construct for this situation: the If-Then-Else construct. Basically, what an If-Then-Else construct is for is determining which steps are followed based on the results of the test. In the above example, if our variable is called "A", then the construct would look like this:

```
If A<18, Then Write "Welcome youth", Else write "Welcome adult."
```

This conditional programming block can be extended by the use of an ElseIf command which, in essence says, "so the first text was false, now I will do this additional test." If the statement after the ElseIf is true then the related command(s) is/are run, otherwise the command(s) is/are skipped. It is possible, if there is never an "Else" statement, for nothing to be done as the result of an If statement block.

Another conditional programming block common to many languages is the Case block. If, in the above example, there had been more than two or three options, a case block might be the better route. For instance, maybe there is a different text to print for those under 12, those between 12 and 21, those between 21 and 45, those between 45 and 65 and those older than 65. In that case, you might create a construct like this:

```
Case A
        <12: Print text for under 12
        12<21: Print text for 12 to 21
        21<45: Print text for 21 to 45
        45<65: Print text for 45 to 65
        Else: Print text for over 65
```

By using the Case statement block, rather than having to go through an entire string of if/then. Elseif tests for the variable, the code tests the variable once and runs the appropriate set of code for whatever the value of the variable equals. Notice that the Case statement can, in most programs, also include an Else to be run if all of the other possible cases are not met.

As with variables and arrays, the syntax for the If-Then-Else and Case constructs will vary greatly from language to language. Some languages also include what is known as a ternary operator function which is a shortcut version of the If-Then-Else code block.

Chapter #3 Review:

1) What is the difference between an If-Then Statement, an If-Then-Else statement and a Case statement?
2) What is a situation in which you might use each of the three types of statements?

Chapter #4—Looping in Programs

Many times, it will be necessary to run a block of code multiple times. To do this, there are a few methods that most programming languages will provide. Each of the methods has their own features and advantages to a particular situation. However, it's important to know all of the various ways to accomplish this task so that you can choose the best way for your particular situation.

The For-Loop

The first and most common method of looping is called the For Loop. A For Loop is used when the code needs to repeat a fixed number of times. For instance, if you wanted to print the numbers 1 through 10, you could use a For Loop. The basic syntax of a For-Loop is this:

```
For counter=1 to 10
        Code to run
Next counter
```

The counter is a variable that keeps track of the number of times to run the loop. The 1 to 10 indicates the initial and final number to use for the counter. It establishes the number of times to loop. The code, indicates the command(s) that need to be run during each "iteration" of the loop (i.e. "**print counter**"). The **Next counter** command tells the computer to increment the counter variable and go back to the beginning of the loop block. However, the For-Loop can be set to not just increment the counter variable by one, it can be set to increment or decrement the counter variable by whatever step is desired. For instance, to print only the even numbers, you could set up the For Loop with **For counter=2 to 10 step 2** to tell the computer to start the variable at 2 and increment the variable by 2 with each iteration. To add flexibility to the For loop, you can also set the initial value and/or the final value to a variable…just be sure 1) the variable has been set to something before you run the loop and 2) that the variable contains an integer. If either of these are not true, you may get some interesting results from your loop.

The While Loop

The While Loop allows you to create a loop that, like the if-then statement, will start by testing a variable. Then, as long as the content of the variable meets the tested condition, it will continue to run the code in the block. For instance:

```
While X<10
        Code to run
Loop
```

This loop will check the variable X each time it loops back to the top and, if the value of X is less than 10, it will run the code block. The While-Loop block does not necessarily have to run at all. If the value of X is 11, it will simply skip the code block and go to the code after the word "Loop". It is very important to note that, in and of itself, the While-Loop statement does not change the variable at all. For that reason, it is important to be sure that the value of X is changed within the code block at some point. If not, you will create an "infinite loop" (i.e. a loop that will never stop) because the condition that it tests will never fail.

The Loop-Until Loop

The Loop-Until construct is very similar to the While Loop in that it tests a variable and runs the code if the variable passes the test. However, there are a couple very significant differences between the two. First, an example:

```
Loop
        Code to run
Until X>10
```

You should notice a few differences right off. First of all, the variable isn't tested until the end of the code. Secondly, I've used greater than instead of less than. The first difference is notable because it ensures that the code in the block will run at least one time. Even if X is 11 at the beginning of the programming block, because it doesn't test the variable until after the block has been run, the code will still process. The second difference is because the Loop-Until construct runs as long as the test is FALSE while the While Loop runs the code as long as the variable tests TRUE. As with the While Loop, it is important that you change the value of X at some point in the code. Otherwise, you will again cause an infinite loop to occur.

Chapter #4 Review:

1) What are the differences between a For-Loop, While Loop and a Loop-Until Loop?
2) Describe an instance where you might use each of the types of loops.

Chapter #5—Functions and Procedures

Nearly all programming languages allow you to create re-usable code in your programs using either Procedures, Functions or both. Every programming language has its own built-in functions and procedures. For instance, all of the languages have a built-in procedure to display something on the screen, for most it is either **Print** or **Write**, which accepts a string or variable as the parameter and outputs the contents of the parameter to the screen. Another example is that most languages have a square root function…perhaps **sqrt** which accepts a number or variable containing a number as its attribute and returns the square root of the given number.

The primary difference between a function and a procedure, in most languages, is that a procedure DOES something while a function RETURNS something. So the Print or Write procedure as given above, writes the contents of the attribute to the screen. The Sqrt function returns the square root of the number which can then be stored in a variable and later sent to a procedure to display it.

Every language has its own way of defining the code for a procedure or function. However, all of them provide the same benefits: They allow you to write code once and use it many times throughout the application. For instance, if you're writing a shopping cart application, rather than writing the code to add an item to a cart once for every item in the store, you can write the code once and "call" it every time it is needed (while passing it the appropriate parameters) and it will run the same way each time. Another benefit of creating re-usable code is that you can store the various sections of code in different files and only load them when they are needed.

Creating re-usable code provides another very big benefit: if changes must be made to how a procedure or function runs, the change only needs to be made in one place. If you copied the code into each place that it is required, to make a change to the functionality would require a change in each place the code exists. This becomes even more problematic if the code is used in multiple files (i.e. in a web page) or in hundreds or thousands of places within the program.

Chapter #5 Review:

1) What are the primary differences between the procedure and the function in most programming languages?
2) What is an advantage of re-using code within an application?

Unit Review:

1) In three complete paragraphs, briefly describe loops, conditional branching, procedures and functions.
2) In a complete paragraph, briefly describe the difference between a variable and an array.

Chapter #6—Where to Go From Here

Now that you have a basic understanding (very basic, in fact) of programming constructs available in most programming languages. So, where do you go from here? That depends on what you intend to do with the programming. If you are interested in learning to design and build interactive or gaming websites, your next step would be HTML, CSS, and PHP for the website and Java for the gaming or interactivity. If you're looking to make the next great computer application, then you should look into some form of C (C++ is one of the most popular) or Java (which allows for cross-platform programs usable in multiple operating systems).

If you're looking for a simple language to learn how to implement these constructs, Microsoft has created a languages called Small BASIC which, as the name would imply, is a very simplistic language. You're not going to create the next great computer application but you can at least create a fun text-based program and learn the basics of programming in an operational context.