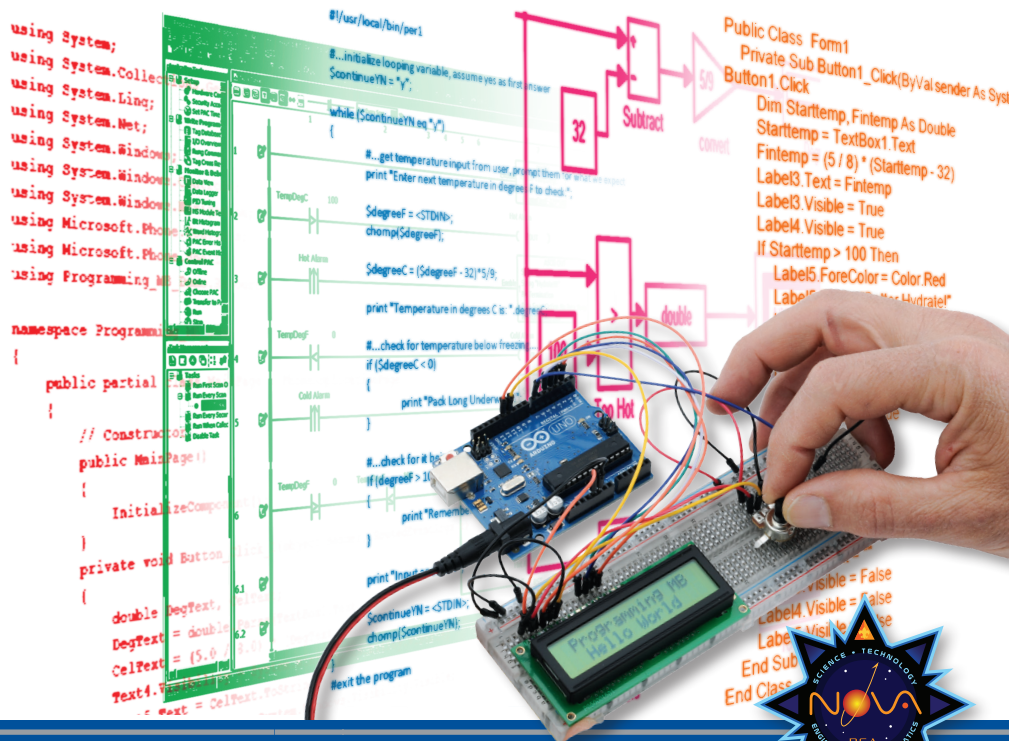


MERIT BADGE SERIES



PROGRAMMING



SCOUTING AMERICA
MERIT BADGE SERIES

PROGRAMMING



"Enhancing our youths' competitive edge through merit badges"

Scouting  America

Note to the Counselor

Thank you for offering your talents as a merit badge counselor. Scouting's merit badge program succeeds because of the dedication and generosity of people like you.

This merit badge is intended to introduce Scouts to programming, to help them understand how programming affects them in their everyday lives, and to help them realize that programming is something any Scout can do and even possibly pursue as a career.

To that end, the requirements are designed simply to expose the Scout to several kinds of programming in different industries—not to turn the Scout into a “programmer.” Please help guide the Scout to make project choices that are fun and relatively simple.

Supporting this merit badge is an online resource, scoutlife.org/programming, where the Scout will find many examples, video tutorials, and development tools provided at minimal or no cost. This resource is presented with the young Scout in mind and serves as the starting point for accomplishing the “meat” of the merit badge requirements, specifically requirement 5.

It is also possible that a Scout might want to satisfy requirement 5 by creating one or more programs using one of the AI chat bots available on the internet. This is acceptable as long as the Scout understands each AI-generated program sufficiently well to modify it, debug it, demonstrate its operation, and explain the operation of the program as called for in requirement 5.

Before spending a lot of time and effort on the requirements, the Scout should meet with you to be sure the requirements are being met to your satisfaction.

For the purposes of this badge, we expect a program to take an input, make decisions based on that input, and then provide an output based on those decisions. Scouts should write the instructions in the languages and development environments of their choice. While “programming” a home thermostat or a DVR is *not* an acceptable project, creating a macro in a spreadsheet program *could* qualify as a project if you think it is appropriate. Please guide the Scout accordingly.

Also note that the requirements do not restrict the code from being run natively on an actual ‘machine.’ Running the program on a simulated machine or in a virtual world is fine. For example, most cell phone development environments provide cell phone simulators with which the code can be tested.

Thank you again for your service. Now, let’s get some Scouts excited about programming!



Requirements

Scouts should go to www.scouting.org/merit-badges/Programing or check Scoutbook for the latest requirements.



Contents

Introduction: The Code Behind the Screen 9

The History of Programming Languages 13

What Is Programming? 25

Where Is Programming Used? 33

Intellectual Property 79

Safety 87

Programming Terms 90

Programming Resources 93



To do what they do, computers, game consoles, smartphones,

Earning the Programming merit badge will take you “behind

By the time you fulfill the requirements for the Programming merit badge, you will be able to work a little of that “magic” yourself. And you might find yourself joining the legions of young programmers who create so much innovative software. Whatever the need, somebody somewhere has written a program to answer it. You could become that somebody. Happy programming!



What Languages Do You Speak?

This pamphlet contains examples of codes that all “perform” the same task; look for the blue boxes throughout this pamphlet. You will be able to compare the codes visually, challenge yourself to figure out the differences, and decipher the language. This is an example of Visual Basic code.

VisualBasic.Net is a programming language used in the Windows and Windows phone environments. It is an object-oriented programming language that can be used to create a wide variety of useful applications. The large box shows the Visual Basic code, and the two smaller boxes show samples of what the user sees.

```
Public Class Form1
    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click
        Dim Starttemp, Fintemp As Double
        Starttemp = TextBox1.Text
        Fintemp = (5 / 8) * (Starttemp - 32)
        Label3.Text = Fintemp
        Label3.Visible = True
        Label4.Visible = True
        If Starttemp > 100 Then
            Label5.ForeColor = Color.Red
            Label5.Text = "Better Hydrate!"
            Label5.Visible = True
        ElseIf Starttemp < 32 Then
            Label5.ForeColor = Color.BlueViolet
            Label5.Text = "Brr! Pack the Long Johns!!!"
            Label5.Visible = True
        End If
    End Sub

    Private Sub Button2_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button2.Click
        TextBox1.Text = ""
        Label3.Visible = False
        Label4.Visible = False
        Label5.Visible = False
    End Sub
End Class
```

Visual Basic
Temperature Example

Programming MB Demo

Enter Temperature

31

Converted Temperature

Converted temperature is:

-0.625 Celsius

Brr! Pack the Long Johns!!!

Restart

Programming MB Demo

Enter Temperature

101

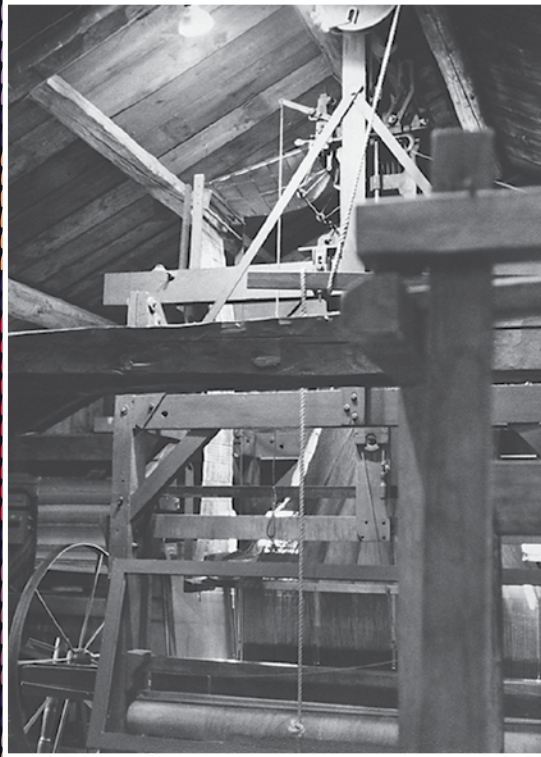
Converted Temperature

Converted temperature is:

43.125 Celsius

Better Hydrate!

Restart



In the early 1800s, a French weaver named Joseph-Marie Jacquard invented an automatic loom, or weaving machine, that was controlled by sets of instructions coded into punched cards. Different cards held instructions for different patterns to be woven into fabrics. The idea of using coded instructions readable by a machine became the basis of programming. The background is an example of a Jacquard pattern.

The History of Programming Languages

A *program* is a set of instructions that tells a *processor* how to do a particular task. A programmer writes the instructions. A *programming language* is the tool that converts the programmer's instructions into a form the processor can understand.

The roots of modern programming stretch back to before the first computers. Most programming languages have their origins in early efforts to automate machines like calculating machines.

Binary Code

Before the development of modern programming languages, developers used *binary code*, which is the “native language” of a machine/computer. A computer “speaks” in only 1's and 0's. These two numerals used in binary language were numeric codes for the operations a particular computer could execute (carry out) directly. Binary code allowed programmers to turn pieces of computer hardware (circuits) “on” (represented with a 1) or “off” (represented by the numeral 0). All modern programming languages are built off the concept of trying to turn human commands into binary or *machine code*.

A *processor* is an electronic circuit on a small chip inside computers and other electronic devices. The processor's job is to interpret and carry out instructions.

According to a popular saying, “There are only 10 kinds of people in the world: those who understand binary and those who don't.” If you understand that, then you are on your way to becoming a programmer.



About the Badge

Programs eventually get executed (carried out) on hardware. At the lowest level, the electronic circuits understand only “on” and “off”; that is, is the voltage (or current) turned on or off? Programmers refer to these on and off states as “1” and “0” simply because those digits are easier to write. Programmers call those 1’s and 0’s “bits.” Groups of eight bits are

usually referred to as “bytes.” Different patterns of these eight-bit bytes can represent different things. For example, “00000001” could represent the number 1; “00000010” could be 2. If we add those, we get “00000011,” which would be 3. “00000100” would be 4, “00000101” would be 5, etc. Do you see the pattern?

We can also assign patterns of bits to represent text characters. The most common standard that most programmers use is ASCII: the American Standard Code for Information Interchange. You can do an internet search to see a table that defines what each pattern of characters means when using that standard. You will see that the first light-green pattern of 1’s and 0’s on the Programming merit badge (01000010) represents B, the second line (01010011) represents S, and the third line (01000001) represents A. Taken together, those spell BSA in binary ASCII.

Try this: Turn the merit badge upside down and look up what those patterns mean. The “B” is the same because it is the same pattern forward or backward, but what about the others? They don’t spell “BSA” anymore, do they?

Bottom line: We will know who the real programmers are by who wears the badge the right way up!

```
01010001110000101010100010101010101
11011011100100100011111101011010101
11110100010110110111100010101110011
```

Machine code is the only language a processor’s circuits can read. This low-level language is hard for people to understand and very tedious to write.



The binary number system is different from the more familiar decimal number system. For example, written as a binary number, the decimal numeral 9 is 00001001. Binary numbers make up *machine code*, the low-level language that computers translate all data into before performing operations on it.

Assembly Language

Assembly language was the next step in the evolution of modern programming. Whereas machine language consists entirely of numbers and is difficult for humans to read and write, assembly language lets a programmer use “names” or characters instead of numbers. This slightly more human-friendly code allowed programmers to more easily program single commands for a processor. Each command in assembly language is specific to the processor (that is, each type of processor uses its own kind of code), so assembly language lacks the portability of most modern programming languages. However, assembly language is ultimately how a processor executes all *higher-level languages*.

A **high-level language** is an advanced programming language that is easier for humans to understand than binary code or assembly language and is not limited to a single processor.

```
// Example F to C in Java
// This file must be named FahrenheitToCelsius.java
import java.util.Scanner;
public class FahrenheitToCelsius {
    public static final double LOW_TEMP_F_WARNING=0;
    public static final double HIGH_TEMP_F_WARNING=100;
    public static final int MAX_LOOP=5;
    public static void main(String[] args) {
        Scanner scanFaren = new Scanner(System.in);
        double Farenheit = 0;
        double Celsius = 0;
        for(int i=0; i<MAX_LOOP; i++){
            System.out.print("\nEnter a temperature in Fahrenheit: ");
            if(scanFaren.hasNextDouble()) {
                Farenheit=scanFaren.nextDouble();
                Celsius = ( Farenheit- 32.)*5/9.;
            }else{
                System.out.println("Data entry error - try again\n");
                System.exit(-1); }
            System.out.println("The temperature in Celsius is: "+Celsius);

            // Check for high temperature and issue a warning if necessary
            if(Farenheit > HIGH_TEMP_F_WARNING){
                System.out.print("Remember to hydrate\n");}
            // Check for low temperature and issue a warning if necessary
            if(Farenheit < LOW_TEMP_F_WARNING){
                System.out.print("Remember to pack Long underwear\n");}
        }
        System.exit(-1);
    }
}
```

JAVA
Temperature
Example

Example Output:

```
Enter a temperature in Fahrenheit: 76.0
The temperature in Celsius is: 24.444444444444443
Enter a temperature in Fahrenheit: 102.
The temperature in Celsius is: 38.888888888888888888
Remember to hydrate
Enter a temperature in Fahrenheit: -2.
The temperature in Celsius is: -18.888888888888889
Remember to pack Long underwear
Enter a temperature in Fahrenheit:
```

Java is one of the most popular programming languages in the world because it is free and runs on many different platforms, a quality referred to as “Write Once, Run Anywhere.” Java is easy to learn if you are already familiar with another text-based language.

How many programming languages are there? As of the printing of this merit badge pamphlet, more than 2,500 programming languages have been developed.

Next-Generation Programming Languages

By the 1950s and 1960s, many new languages were being introduced, including BASIC, Fortran, and LISP. These languages took programming beyond binary and assembly to allow *portability* of languages beyond just the processor hardware of the time. That is, programs written in these languages could run on two or more kinds of processors or with two or more kinds of *operating systems*. Many languages, such as Pascal, were created to capture new concepts of computer science. Others, such as Fortran and COBOL, were developed to meet specific needs of science or business.

Important Early Languages

FORTRAN (*Formula Translation*), originally developed in the 1950s, was among the earliest programming languages. Well-suited for mathematical calculations, FORTRAN was used mainly for scientific and engineering programming. Today, Fortran (as it is now spelled, without all-capital letters) is the primary language for some of the most demanding supercomputing tasks, such as weather and climate modeling.

COBOL (*COmmon Business-Oriented Language*) was popular for business data-processing on larger computers. COBOL was designed for use by banks, utility companies, manufacturers, government agencies, and other big operations.

Pascal—named after the mathematician Blaise Pascal—put into practice the concept of *structured programming*. Structured programming was designed to help programmers write programs that are cleaner and easier to test, *debug* (remove errors), and modify. For years, Pascal was a popular language to teach beginning programming classes.

```
// Example F to C program in "c"
#include <stdio.h> // This includes the file stdio.h into your code so it knows what the functions
such as printf() mean.
```

```
const float LOW_TEMP_F_WARNING=0.; // Program constants
const float HIGH_TEMP_F_WARNING=100.;
const int MAX_LOOP=5;
```

```
int main() // Declaration of program
```

```
{
    float temp_f; // Declaration of variables that the program will use
    float temp_c;
    int i;
```

```
for(i=0; i<MAX_LOOP; i++){ // loop
    printf("\nEnter the temperature in degrees F : "); // Input the temperature to convert
    scanf("%f",&temp_f); // Reads the user input
```

```
temp_c = (temp_f- 32)/1.8; // Math formula to convert Farenheit to Celcius
printf("The temperature in Celcius (C) is %f\n",temp_c); // Output the Celcius result
```

```
if(temp_f > HIGH_TEMP_F_WARNING){ // Check for high temperature
    printf("Remember to hydrate\n");
}
```

```
if(temp_f < LOW_TEMP_F_WARNING ){ // Check for low temperature
    printf("Remember to pack Long underwear\n");
}
```

```
}
return(0); // exits the program
}
```

"C"
Temperature
Example

Example Output:

```
Enter a temperature in Fahrenheit: 76.0
The temperature in Celsius is: 24.44444444444443
Enter a temperature in Fahrenheit: 102.
The temperature in Celsius is: 38.888888888888886
Remember to hydrate
Enter a temperature in Fahrenheit: -2.
The temperature in Celsius is: -18.888888888888889
Remember to pack Long underwear
Enter a temperature in Fahrenheit:
```

The **"C"** programming language was designed to develop code for UNIX, a multiuser operating system, originally developed in 1969. Today, most code for general-purpose operating systems is written in C or its advanced version, C++ (pronounced "see plus plus").

The popular C programming language is widely used to create other computer programming languages, to program robots, for scientific engineering, and for many other applications. It is a great fundamental language to have in your programming bag of tricks.

PIONEER OF PROGRAMMING: JOHN VON NEUMANN

John von Neumann (1903–57) developed many important concepts that directly affected the path of programming languages, one of which is called “conditional control transfer.” This idea gave rise to the notion of *subroutines*, or small blocks of code that could be jumped to in any order, instead of a single set of consecutive steps for the processor to take. The second part of the idea stated that code should be able to loop (repeat) or to branch based on logical *statements* such as “if/then” statements. (In programming, a *statement* is a single line of code for performing a specific task.) “Conditional control transfer” gave rise to the idea of “libraries,” which are blocks of code that can be reused over and over. For example, a complex mathematical formula could be written once in a single block of code or library so any programmer could use it without having to write the code. By doing this, programmers can take advantage of previous work without having to “reinvent the wheel.”



Programming in Objects

Object-oriented programming (OOP) is a style of programming that treats concepts as “objects” that can interact with one another. OOP languages surfaced in the mid-1960s and allowed a software developer to write code independently of any specific software application. The programming is done by putting together groups, or *modules*, of commonly used commands—instructions on how to print, how to save information to a disk, and so on—into complete programs.

Object-oriented programming saves time because the programmer can reuse parts of programs already developed by others. OOP concepts began appearing in the 1960s with a programming language called Simula and further evolved in the 1970s with the arrival of Smalltalk. In the 1980s, OOP languages such as C++ and Eiffel appeared. OOP continued to grow in popularity in the 1990s, most notably with the advent of Java.

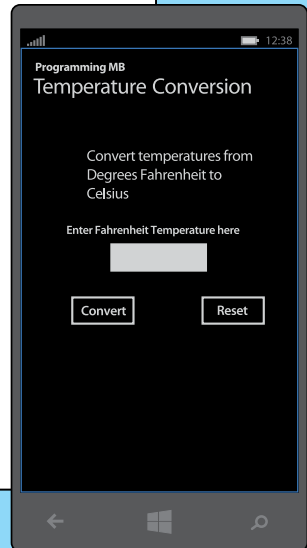
C# (C-Sharp) Temperature Example

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Net;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Navigation;
using Microsoft.Phone.Controls;
using Microsoft.Phone.Shell;
using Programming_MB_Demo.Resources;

namespace Programming_MB_Demo
{
    public partial class MainPage : PhoneApplicationPage
    {
        // Constructor
        public MainPage()
        {
            InitializeComponent();
        }

        private void Button_Click_1(object sender, RoutedEventArgs e)
        {
            double DegText, CelText;
            DegText = double.Parse(TextBox1.Text);
            CelText = (5.0 / 9.0) * (DegText - 32.0);
            Text4.Visibility = System.Windows.Visibility.Visible;
            Text5.Text = CelText.ToString();
            Text5B.Visibility = System.Windows.Visibility.Visible;
            if (DegText > 100)
            {
                Text6.Text = "It's Hot! Better Hydrate";
                Text6.Visibility = System.Windows.Visibility.Visible;
            }
            else if (DegText <= 32)
            {
                Text6.Text = "It's Cold! Better pack long underwear";
                Text6.Visibility = System.Windows.Visibility.Visible;
            }
        }

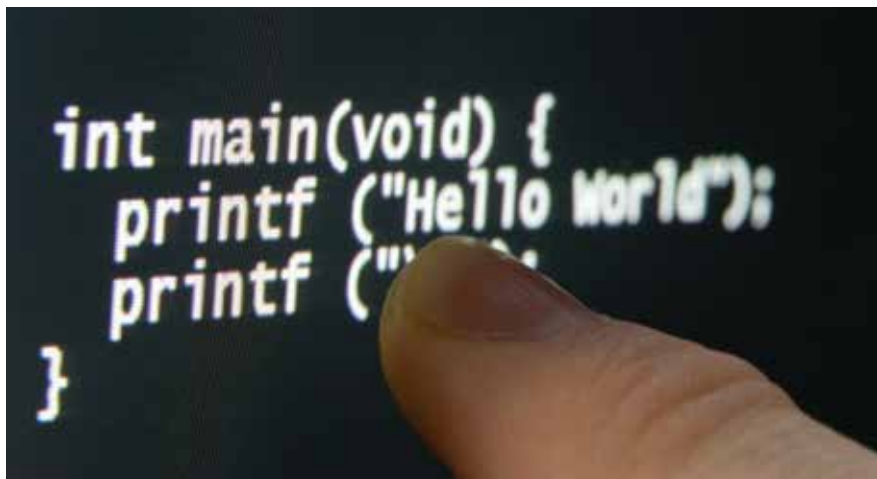
        private void Button_Click_2(object sender, RoutedEventArgs e)
        {
            TextBox1.Text = "";
            Text4.Visibility = System.Windows.Visibility.Collapsed;
            Text5B.Visibility = System.Windows.Visibility.Collapsed;
            Text5.Text = "";
            Text6.Visibility = System.Windows.Visibility.Collapsed;
        }
    }
}
```



C-Sharp, or C-#, is a programming language used in the Windows and Windows phone environments. It is an object-oriented programming language that can be used to create a wide variety of applications.

Originally known as “C with Classes,” C++ was developed starting in 1979 and is now among the most popular programming languages. Most graphical operating systems—like Microsoft Windows—and major related programs were written with C++.

Java appeared in 1995 as a general-purpose programming language, designed to be machine independent. It was intended to let software developers “write once, run anywhere,” meaning that code that runs on one computer system does not need to be rewritten to run on another.



One of the first programs most beginning software developers learn is the Hello World program. Its coded instructions produce a simple statement—“Hello World!”—on a computer screen. Whatever programming language the developer is learning, Hello World has become the traditional first program. Though the history of this tradition is not well documented, records point to Brian Kernighan’s 1974 tutorial for the computer language “B.” The best-known example is found in Kernighan and Dennis Ritchie’s 1978 book, *The C Programming Language*.





Programming for the World Wide Web

The creation of the World Wide Web brought rapid growth in the field of programming. The web itself is a set of files located on internet *servers* (computers) and accessed using “rules” known as the *hypertext transfer protocol* (HTTP). Internet languages allow everyday people to easily access the information. Programming internet languages (like JavaScript and PHP) were developed over time to better access this information and provide a better way to experience it.

Hypertext Markup Language (HTML) is a basic language for building webpages. Considered the first language specifically created for the World Wide Web, its purpose was to create a structure for information being presented on the web.

JavaScript, which was originally released in 1995 as LiveScript, was created to enhance webpages in a way that HTML could not. JavaScript allowed webpages to be interactive.

PHP is a general-purpose programming language designed to produce dynamic webpages. A *dynamic* page changes with each user, displaying different content each time the page is viewed. “PHP” originally stood for “Personal Home Page” but is now an abbreviation for *PHP Hypertext Preprocessor*.

```

<?php
function temp2celsius ($fahrenheit) {

    if (!isset($fahrenheit)) return FALSE;

    $celsius = (float)(( $fahrenheit - 32) / 1.8 );
    $returnText = "You converted ".round($fahrenheit)." F to ".round($celsius)." C ";

    If($fahrenheit < 32)
        $returnText .= ": Pack Long Underwear";
    Elseif ($fahrenheit > 100)
        $returnText .= ": Remember to Hydrate";
    return $returnText;
}

    $setVar=$_GET["degree"];
    if (!is_numeric($setVar)) {
        echo "Please enter only numbers for the temperature";
    } else {
        echo temp2celsius($setVar);
    }
?>

```

PHP
Temperature
Example

HTML Code:

```

<head>
<title>BSA PHP Temperature Conversion</title>
</head>
<body>
<h2>BSA PHP Temperature Conversion</h2>
<form action = "<?php echo $_SERVER['PHP_SELF']; ?>" method = "GET">
<P>Enter a number in the box below to convert the temperature to Celcius.</P>
Degrees: <input type = "text" name = "degree" size=5>
        <input name = "SubmitConvert" type = "submit" />
</form>
</body>

```

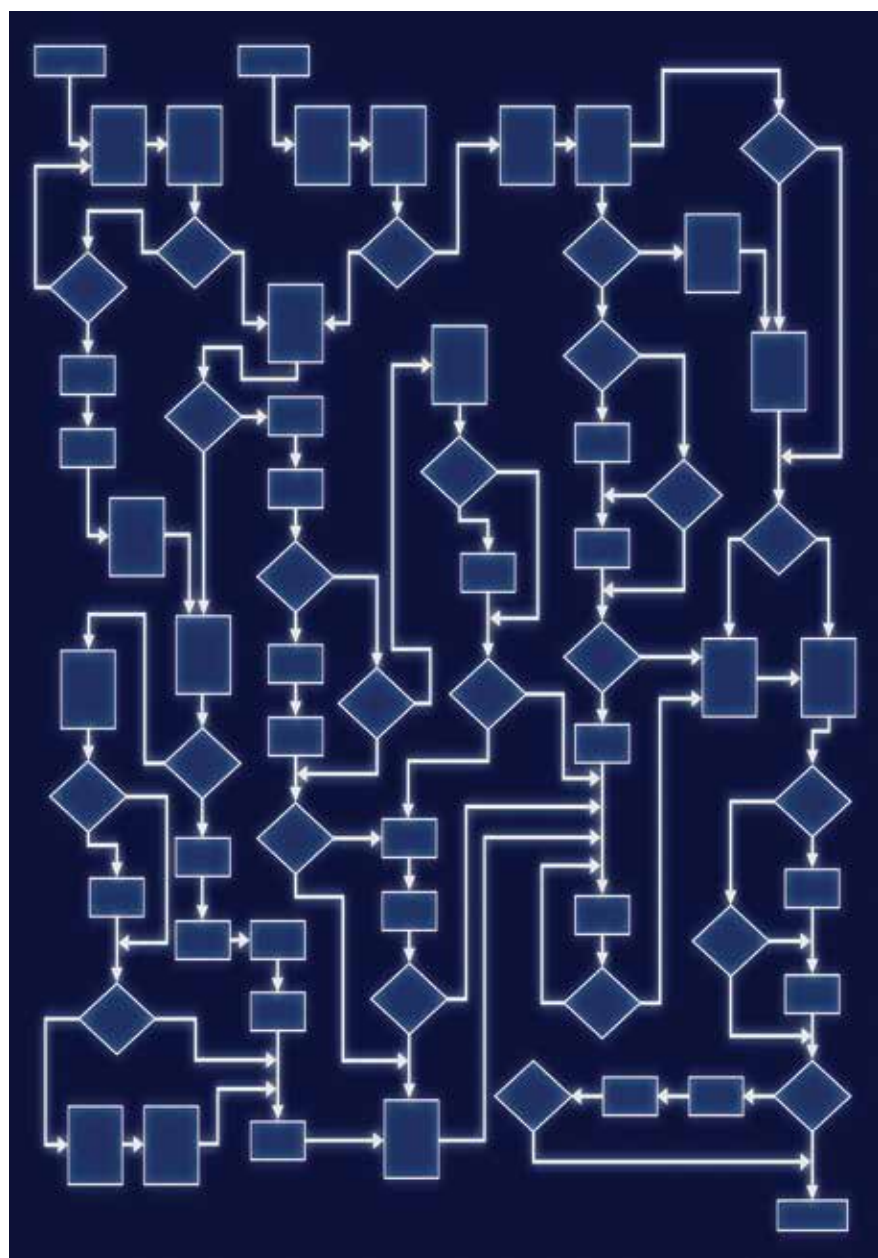
BSA PHP Temperature Conversion

You converted 101 F to 38 C: Remember to Hydrate

Enter a number in the box below to convert the temperature to Celcius.

Degrees:

PHP is a widely used general-purpose scripting language that is especially suited for web development. PHP is used to execute database commands, perform conditional operations, and perform high-level formatting functions. It integrates many other web technologies, including HTML, XML, JavaScript, and Java.



What Is Programming?

When you talk to a friend, you use a language you both understand quite well. Many times, your friend will get what you mean even if you don't say it in so many words.

Talking to a computer isn't as simple because a computer will not know what you mean unless you say it exactly. A computer will execute *exactly* the instructions it is given—nothing more, and nothing less.

When these instructions are precise and clear, a computer will run them over and over and never get tired. This flawless repetition of correct instructions, at the very high speeds that computers can run them, has led to the technological revolution of incredible devices that we use every day and are all around us: smartphones, DVD players, digital cameras, thermostats, garage door openers, remote controls, wristwatches, alarm clocks, traffic signal lights, industrial robots, and even musical greeting cards.

The processors that run digital devices operate at a simple level, turning electronic circuits on and off on tiny circuit boards. They are following instructions that someone placed there, and work together with other digital components (a memory chip, an electronic display, or a motor, for instance) to perform some higher function that was designed into their operation.

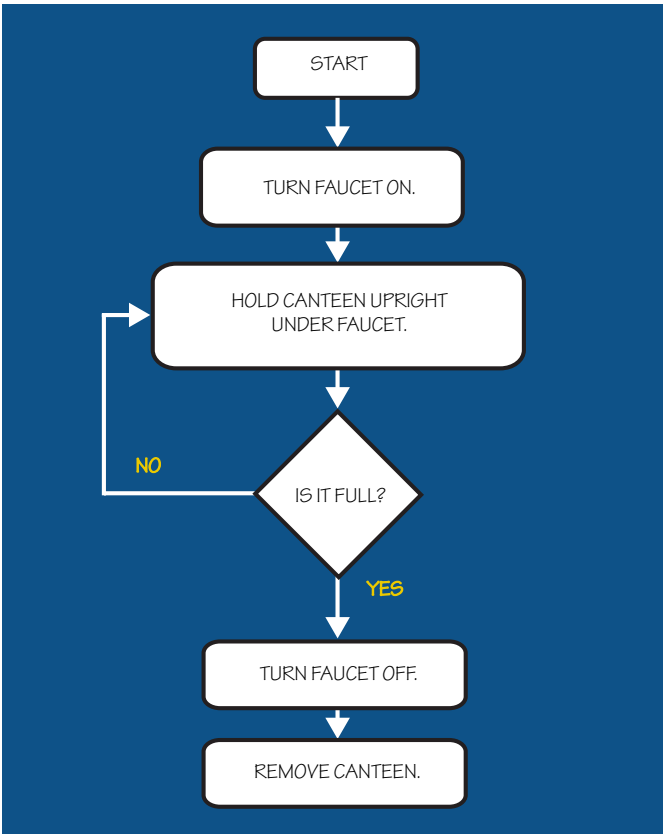
So how do these instructions get translated from an idea into something that processors will understand? This activity of converting ideas into instructions is called *programming*. When you create a program, the instructions can be simple or highly complex. It's a bit like working with uncut wood: the activity can be quick and useful, like splitting logs for a fire. Or it can be intricate and complex, like whittling a key whistle from a hickory branch.

Step-by-Step Communications

The challenge with programming is in communicating precisely what you want to happen. You might tell your friend: “Fill your canteen with water.” That is easy for your friend to both understand and do.

For a computer, however, the steps to do this will need to be spelled out exactly as you want them done. The sequence of steps is important, and the decisions you want the program to make can be based only on what it can detect on its own or learn along the way.

In the example, “Fill your canteen with water,” here are the steps broken down so that a computer (or robot) could understand them:



In this diagram, each process or action the program will do is represented by a rectangle listing what should happen. The *decision point* is represented by a diamond, which shows possible alternate paths depending on the answer to the question. The question is usually phrased to be answered either “yes” or “no.” This yes/no system, sometimes called true/false, is why binary numbers are used in programming. Binary numbers count with only the digits 0 and 1, instead of 0 through 9. This means a “yes” or “no” answer can easily be translated into a 1 (for “on”) or a 0 (for “off”) that the circuit boards can understand.

Step 1

Programming begins by defining the purpose of the program. What do you want it to do? Print a picture on the computer monitor or display? Play a sound when someone pushes a button? Have a robot make you a sandwich? All these programs need to be described, step-by-step, so a processor can run them. This first step, defining what the processor should do, is so critical it has its own name. It’s called the *requirements*.

Plan the Work, Work the Plan

You will find your programming goes much more smoothly if you plan out the work before starting to program. Goals are always easiest to achieve when they are written down. The more concise and clearly laid out, the better. That applies to Eagle Scout service projects, and programming is no different. You will find that even the most experienced programmers write down what they want to do before beginning to program.



Step 2

The second step is to describe the logical steps and necessary decisions that the processor must make. The requirements are translated (by you) into a logical *flowchart* or other kind of diagram. The diagram or description spells out the detailed steps the program will go through to accomplish the goal.

Among the many diagramming methods are data flow diagrams, system diagrams, storyboards, object-oriented designs, database designs, network architecture designs, and even plain English or *pseudocode*. Choose the method that works best for you. But first, always write down what you want to do in logical steps, and you will find your programming will be much easier.

Pseudocode is a great way to start planning your work. To write pseudocode, you simply write down a logical sequence of steps. Suppose you want a robot to move forward until it reaches a wall. Then you want it to stop, turn around, and start moving again. The pseudocode might look like this:

```
Move forward
Read wall sensor
Found wall?
Yes—Turn around
No—Keep moving
Repeat
```

Step 3

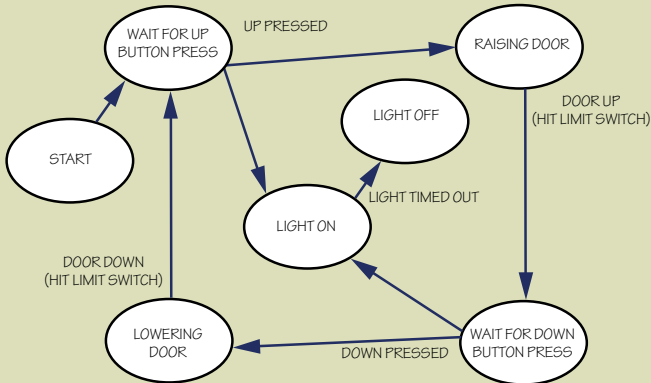
The third step is to translate this diagram into a processor language, a task also known as *coding*. Do you have to use binary numbers (1's and 0's) for this? Fortunately not! Many *higher-level languages* have been invented that define instructions and data storage, which make it easier for people to express the program's steps. When a programmer writes the program in one of these languages, the processor itself converts it into the binary numbers that the electronic circuits will understand.

This process of writing the program varies depending on which programming language you are using. Some languages can be entered into the processor and they will be executed right away. These languages are called *interpreted*. They are fast to write and experiment with, but they may not take full advantage of the processor's abilities. Examples of interpreted programming languages are JavaScript, Visual Basic Script (VBScript), PHP, and Perl.

The Six P's of
Programming:
Proper
Prior
Planning
Prevents
Poor
Programming

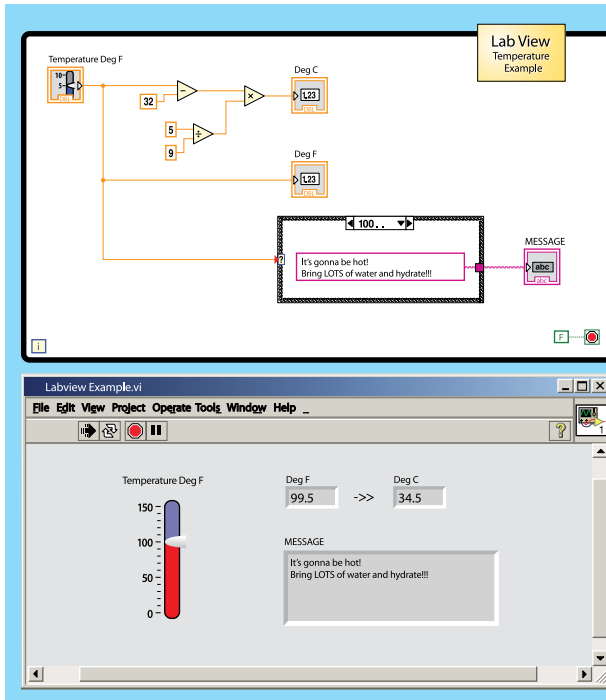
State Diagrams

State diagrams are great for describing programs that work in distinct conditions or situations. For example, a state diagram for a garage door opener might look like this:



The program begins in the “start” bubble and moves directly to the “WAIT FOR UP BUTTON PRESS” state. (Here, the word *state* means “condition” or “situation.”) When the “UP” button is pressed, we move to the “RAISING DOOR” bubble. The door goes up until it hits a switch at the end. Then we transition to the “WAIT FOR DOWN BUTTON PRESS” state. When the “DOWN” button is pressed, we move to both the “LOWERING DOOR” state and the “LIGHT ON” state. The light moves to the “LIGHT OFF” state when it times out. The door goes down until it hits the lower limit switch, where we transition back to the “WAIT FOR UP BUTTON PRESS” state.

Other languages need to be processed before the processor can run them. The instructions you write must be converted into 1’s and 0’s and stored in a binary file. This step of converting the instructions from source code to machine code is called *compiling*. A program called a *compiler* reads the instructions you have written and generates a file of machine code that you can hand to the same processor, or any number of processors of the same type, to run. By doing this once, the other processors will not need the compiler to figure out what to do. Examples of compiled programming languages include C++, Java, and Visual Basic.



Labview programming language uses a data flow model — things progress from left to right in the block diagram, and all inputs and outputs are shown on the front panel. It is used in such disciplines as science, engineering, robotics, technology, and factory automation. In this example, a user adjusting the temperature slider on the left will change the results in the text boxes.

The term *source code* simply means the original code (written in a high-level language) that was used to create the program.

Getting Started

Which programming language should you start with? First think about what you want the program to do, and what kind of hardware it is or what functions it performs (a laptop, a robot, a smartphone, a webpage, etc.). Sometimes only one language will work for a particular kind of problem or on certain hardware. In other cases, there might be several languages that could equally handle the task, and you can pick the one you are more familiar with. If you want a challenge, then it's fun to pick a new language and learn it as you go!

How to Get Started Quickly

In the fast-changing world of programming, books can be outdated by the time they are published. For this reason, you will find specific guidance and programming examples online at the companion website, scoutlife.org/programming. With your parent or guardian's permission, visit the website for hands-on tutorials, practical advice, and detailed support for fulfilling the Programming merit badge requirements.

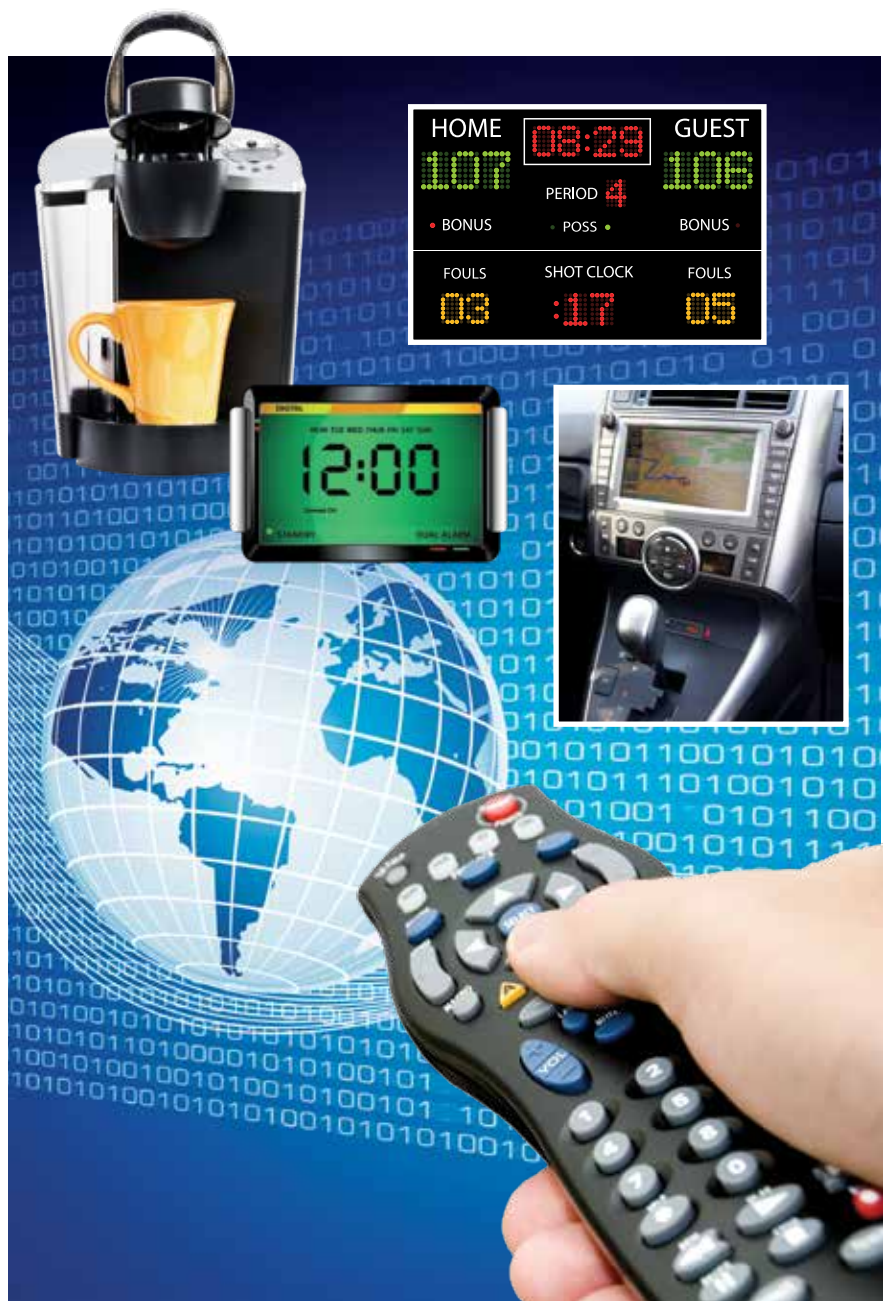
Also study the code “snippets” that appear throughout this pamphlet in the blue boxes. Each snippet performs a similar function but in a different language so you can compare them and get a feel for what each programming language looks like. Each example that appears in this pamphlet is also on the companion website with all the supporting tools you will need to get started and to learn more. Note that you can use these samples to fulfill requirements 5(a)–5(c).

Sequences of Commands

Programs are either linear or structured. Linear programs proceed in sequence from beginning to end. That is, the instructions are executed in the order they appear. “Robot move forward, stop, and return to the starting spot” is an example of linear programming. One command follows the next.

In structured programming, however, structures such as loops and “go to,” “if/then,” and “while” statements can cause the commands to move around in the program.

- *Loops* are used to make a segment of code repeat a specific number of times.
- *Go to* statements cause the program to jump to another area in the program.
- *If/then* statements are conditional statements. That is, if the condition is true (a specified condition exists), the instructions immediately following the if/then statement are executed. For instance, if a robot's sensor has been activated (the specified condition exists), the program will do the instructions that come after the if/then statement. But if the sensor has not been activated, those instructions will not be followed.
- *While* statements are also conditional. They will run a part (or subroutine) of the program while something is happening. After the “something” ends, this subroutine returns to the rest (or main part) of the program.



HOME	08:29	GUEST
107	PERIOD 4	106
• BONUS	• POSS •	BONUS •
FOULS	SHOT CLOCK	FOULS
03	:17	05



Where Is Programming Used?

Programming is in almost every aspect of modern life. Almost everything you touch either uses a computing device or was made by something that was programmed. For example, cars have dozens (sometimes hundreds) of processors onboard. Many home appliances have processors in them. Mobile phones and tablets are completely software driven. Programs are everywhere.

Many industries that use programming share common uses for programs, while other industries have unique needs. Several industries are discussed in the following pages. The companion website for this merit badge has many example programs and free or low-cost software development tools.

Once you find industries that interest you, head over to the website (scoutlife.org/programming) to get up and running quickly. You will be programming in no time!

Mobile Devices

Mobile devices are common and are among the best self-contained examples of the power of programming. Think about it and you will realize that almost everything on a mobile device is programmed: on-screen buttons, graphics, sound, controls, etc. There are very few physical buttons on a mobile device—almost everything is generated and controlled by software.

Don't forget to get your counselor's approval *before* you start.



The billions of mobile devices in use around the world all need to be programmed. Learning how to program a mobile device could be a great career choice, and an awesome and fun skill to have. While it might seem difficult, it is not out of your reach. Many apps (applications) and programs are written by young people like you every day.

Most mobile applications are different from traditional programming. The operating system is usually designed as a framework. That is, the basic parts of the program are already done for you (button presses, sounds, video, picture handling, GPS, communications, databases, etc.). All you have to do is write the code that takes advantage of these ready-to-go resources and insert the code into the correct portion of the framework. Your task as a mobile device programmer is to learn how that framework is set up and how to take advantage of it.

Another option for programming mobile devices is to write a website-style application using languages like JavaScript. Many applications used today are just mini website apps and can be written with the same programming tools. The advantage of this style of programming is you can write one app that works on several different kinds of phones. The limitation is that you can't always access the device-specific features (like a GPS) from these generic applications.

This brings up a third type of programming for mobile devices: Some companies have created a hybrid approach. They allow you to write one program that works on many devices but also interacts with device-specific functions. Some of these hybrid approaches use a simple drag-and-drop style of programming interface. (The *interface* is the means of communication between the computer and the user.)

The learning curve for programming mobile devices can be steep. Don't let that stop you. Just start with simple examples and build from there. The next thing you know, you will be programming your own mobile-device applications. (Be sure to check out the Programming merit badge website for examples and tutorials.)

Objective C
 Temperature
 Example

```
#import "campout.h"
@implementation campout

//...general function to get string input from keyboard...
NSString *getStringInput(NSString *prompt)
{
    printf("%s", [prompt cStringUsingEncoding:NSUTF8StringEncoding]);
    NSFileHandle *input = [NSFileHandle fileHandleWithStandardInput];
    NSData *inputData = [NSData dataWithData:[input readDataToEOF]];
    return [[NSString alloc] initWithData:inputData encoding:NSUTF8StringEncoding];
}

//...general function to get number vlaue from keyboard. Gets string first, then
// converts it to a number ...
int getNumberInput(NSString *prompt)
{
    NSString *inputString = getStringInput(prompt);
    return [inputString integerValue];
}

void howToPack()
{
    NSString *continueYN = @"y";
    int nDegreeF = 0;
    int nDegreeC = 0;

    while ([continueYN isEqualToString:@"y"])
    {
        nDgreeF = getNumberInput(@"What temperature in degrees F:");
        nDegreeC = (nDegreeF - 32) * 5/9;
        printf("The temperature is %i degrees C\n", nDegreeC);

        if (nDegreeC <= 0)
        {
            printf("Pack an extra sweater! \n");
        }
        if (nDegreeF > 100)
        {
            printf("Remeber to hydrate! \n");
        }
        continueYN = getStringInput(@"Continue with another temperature? (y/n): ");
    }
}
```

Objective-C is a compiled language that is derived from C and is purely object-oriented. This is a useful language to learn because the iPod, iPad, and other Apple computers use this as their native language for programs and apps.

Loops

Loops are essential program elements that allow a computer to perform a repetitive task. The computer will execute the code inside the loop over and over until a condition is satisfied. Different kinds of loops check the condition at different times.

Be careful when writing loops because if you write a loop where the condition is never satisfied, you will have what is called an “infinite loop” and your program will be stuck forever. When this happens, your program will crash!

For Loop

A For Loop executes the instructions in the loop a fixed number of times with one or more variables changing value each time through the loop.

```
For X = 1 to 10 begin
Execute the contents of the loop 10 times with X=1, 2, ..., 10
End
```

While Loop

A While Loop continuously executes the instructions in the loop and stops only when the condition is no longer true. The condition is first checked to see if it is true. If so, the instructions in the While Loop are executed. Then the condition is checked again and if the condition is still true, the instructions in the While loop are executed again. This continues until the condition becomes false.

```
While (X <> 10) do begin
Execute the contents of the loop while X does not equal 10.
Exit the loop when X becomes 10
End
```

Repeat Loop

A Repeat Loop is like a While Loop but the condition to check occurs at the end of the loop. A repeat loop executes the instructions in the loop once and then checks the condition. If the condition is false, the instructions in the loop execute again. This continues until the condition becomes true.

```
Repeat begin
Execute the contents of this loop one time and then check to
see if X equals 15. If so, stop. Otherwise, execute the contents
of the loop again and keep doing it until X equals 15
Until X = 15
```

Business Applications

Programs are used in all areas of a business. Programs help employees do their jobs and help businesses reach their goals.

Retail. Computers are used as point-of-sale systems where people pay for their purchases. These systems have printers and *peripherals* (external devices) like receipt printers, card readers, and cash drawers. Computers used as point-of-sale systems have control software and the basic operating system and little else.

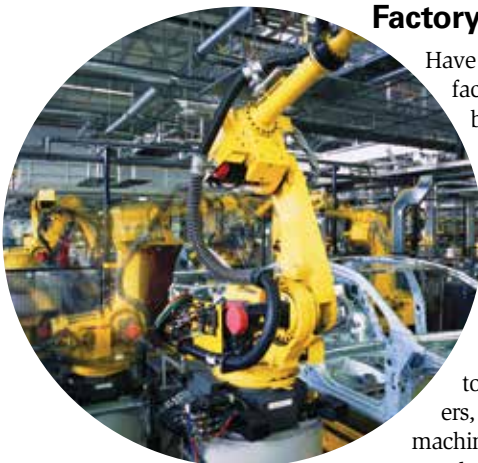
Warehouses. Inventory control is a major use of programming in warehouses. This software keeps track of the material that has come into the warehouse and what has been shipped out. The software communicates with the company accounting system to make sure the material sold is paid for and checks are written to buy new material.



Accounting. A business's accounting system keeps track of the money. Accounting systems communicate with both the retail point-of-sale systems to track the money coming into the company and the inventory control system to follow the funds going out.

Administrative. Most businesses have administrative functions such as letters to be sent and other communications that need to happen. People working in the administrative areas typically use programs like word-processing software to create letters, marketing materials, reports, and memos.

Many Scout troops use programs to organize the troop's activities, dues, and registrations. You could write a program that keeps track of your patrol dues by reading in how much each Scout has paid in dues and then printing out how much each Scout is ahead or behind in dues payments.



Factory Automation

Have you ever seen a video of an automotive factory where robotic arms automatically build a car? Have you seen robots in a bottling plant automatically filling and labeling bottles? Chances are that these robots were controlled by a special kind of computer called a PLC, or programmable logic controller.

What Is a PLC?

PLCs are computers designed specifically to control machines such as air conditioners, motors, sensors, lights, bottling machines, and so on. The PLC can be programmed to have a machine do the same mundane or dirty task over and over again so humans don't have to. They are designed to be durable and to work over wide temperature ranges and in harsh environments where your home computer wouldn't last long.



A Brief History of PLCs

Before computers, factories were controlled by relays. A relay is a switch that is controlled electronically and is either on or off. A factory might have a relay to turn on the lights, another to turn on a conveyor belt, another to turn on fans, and another to control an oven. Factories used hundreds and thousands of relay switches. They had cabinets full of relays and wires connecting the relays to various parts of the factory. The equipment was bulky and hard to fix when there was a problem. If the factory needed to change something about its process, people had to go in and rewire the relays. Rewiring was time-consuming and difficult. And the relays were not reliable because they were mechanical devices. They would wear out over time, and the factory would have to be shut down while the relays were being replaced.

In the late 1960s, a device called a Modular Digital Controller (the MODICON) was invented to replace all those relays with a simple computer. Because the computer didn't need all the wires and relays, it was much smaller than the old-fashioned relay box and far more reliable. And if a factory needed to change something, a programmer just changed the program. Workers didn't have to go in and rewire all the relays. The factory could get back up and running much faster, saving time and money. The MODICON was the first general-purpose PLC.

Today most factories in the world rely on PLCs to get the work done, so let's see how to program a PLC.

Programming a PLC

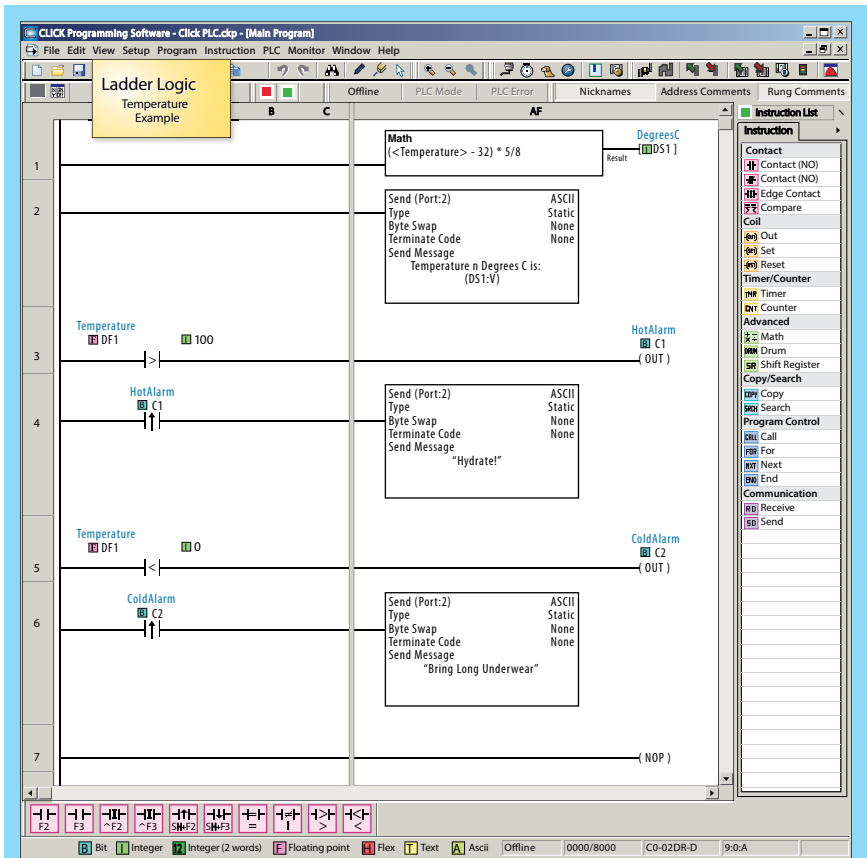
Programming a PLC can be very different from most other kinds of programming. PLCs are programmed in several ways, but the most popular method is called "ladder logic." Because most of the people who ran the factories already knew how to control the machinery using mechanical relays, it was important that the PLC could be programmed using the same concepts.

A ladder logic program closely resembles a relay wiring diagram. You have a power rail, typically shown as a vertical line. A wire connects the power rail to a switch or "contact," which is then wired to a "coil." A coil is a generic term for the thing that needs to be turned on (a light fan, motor, etc.). To control another motor, you just add another wire, switch, and output. After adding several of these, you get something that looks like a ladder, which is why people refer to this as "ladder logic programming."



This PLC controls motors and sensors in an automated distribution facility, including this spiral conveyor system.

As you can see in the example shown here, a ladder logic program looks like a wiring diagram. That made it easy for the factory engineers who were used to relays to adopt the new technology.



Ladder Logic gets its name from the way the code looks—like rungs on a ladder. Inputs and decisions are made on the left and outputs are on the right. Ladder Logic code is used in super-reliable and durable computers called PLCs, which control factory machines. Almost everything made in a factory today is controlled by a PLC programmed in Ladder Logic.



Although PLCs come in all shapes and sizes, they all have some common features.

- They are designed to be rugged and reliable enough to operate continuously for many years in factory environments.
- They are programmed to do the exact same thing repeatedly, all day long, and they are typically programmed using Ladder Logic.
- They can read lots of inputs (sensors, switches, temperature, humidity, pressure, etc.).
- They can control lots of outputs (machines, motors, lights, relays, servos, control panels).

WHERE IS PROGRAMMING USED? _____

A PLC, or programmable logic controller



This programmable logic controller (PLC) controls motors and sensors . . . in this automated distribution plant.

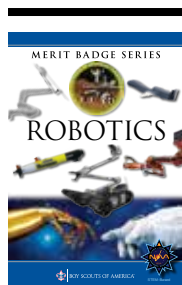




This denim factory ...

**is controlled
by this PLC.**





If you are crazy about robots, you might not know there is a Robotics merit badge waiting to be earned by you.

Robotics

Robots are mechanical systems that are programmed to sense and react to their environments. They are designed to perform specific tasks quickly and accurately. Programs in robots can be highly sophisticated and even appear to be intelligent. A robot can be designed to look like a human arm so it has the flexibility needed for the tasks it will perform. Depending on the task, a human-arm robot may have a simple two-finger gripper, a welding attachment, or a paint sprayer.

To plan for its tasks, a robot can accept input in multiple ways:

- Typed instructions via a keyboard
- Handheld control device, sometimes called a pendant
- Sensors like temperature, GPS, or ultrasonic, or mechanical switches to detect its location

In an industrial robot, the programming may be divided into several sections such as the following.

SAFETY

- Emergency stop of the robot if a safety sensor is activated
- Safety limits for the robot itself: if it contacts an obstacle and cannot move, then the program turns off the robot

USER INTERFACE

- A graphical user interface (GUI) helps the user type in the robot positions and speeds, and how to operate tools or grippers at the end of the robot's arms.
- A handheld pendant allows the user to move the robot to locations and store those locations for the task.

Miniature models of the robot are also used to program movements and locations into the computer. These models are a follower system and are used in special situations requiring human decisions and robot accuracy, such as robotic surgery.

ROBOTICS ACTIONS

- Turn motors on and off.
- Activate grippers and paint sprayers.
- Control welders.



ROBOT POSITION

Sensors are used to report where the robot and its appendages are in its environment. The sensor values are used to calculate where the robot is in the workspace. The program can then determine if the next position is possible for the robot.

ROBOT MOVEMENT

The robot position sensors also make sure the robot is not moving too fast or too slow. This is especially important for painting and welding operations.

Robots that perform dangerous tasks or move heavy objects need all of these features to be safe and reliable. Robotics projects by hobbyists that use low-power motors may need only a few of these sections in their programming.



Robot constructed from a LEGO Mindstorms kit

Programming a Robot

Robots can be programmed by many different kinds of languages depending on the environment they are used in. They can use general languages like C or C++ or they can use graphical languages.

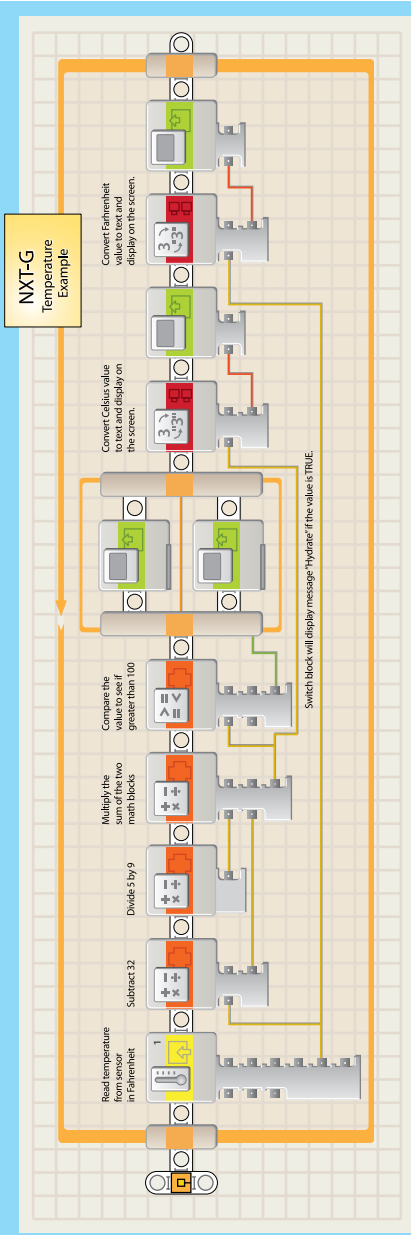
Robots operate in many areas including industry, law enforcement, military UAVs (unmanned aerial vehicles), medicine/surgery, space exploration, underwater oil-well repair, and deepwater scientific study. What do all these areas have in common? Robots in these fields perform tasks that are Dirty, Dangerous, Difficult, or Dull—the four D's



Robot used in law enforcement

of robotics. Because robots are used in so many different areas, career opportunities are good for those who know how to program them. For more about robots, see the *Robotics* merit badge pamphlet.

The NXT-G programming language is used to program Lego Mindstorms Robotics. The user simply drags the appropriate blocks onto the screen and configures them. In this example, the robot reads a temperature sensor, converts it to degrees Celsius, displays messages, and then displays the temperatures on the screen. The orange box around the program is a loop that takes the program back to the start, where it repeats everything again.



Robotics Competitions

Robotics has become a popular activity for Scouts, with a Robotics merit badge to earn and many robotics competitions in which to participate. Consider joining a local team or two and use that as the basis for one of your programming projects. Here is a list of popular robotics competitions.

Boosting Engineering Science and Technology (BEST)—A free competition because all the materials needed to build the robot are provided for you. This competition is especially popular in the Southeast United States but is growing fast all over the country. Ages: Middle school and high school. bestinc.org

FIRST Robotics Competition (FRC)—Referred to as “the varsity sport for the mind” because it combines the excitement of sport with the rigors of science and technology. Competitors build and program 120-pound robots. Ages: High school. firstinspires.org

FIRST Tech Challenge (FTC)—Similar to FRC but on a smaller scale. Ages: High school. firstinspires.org

FIRST LEGO League (FLL)—A build-and-compete challenge using robots built with LEGO Mindstorms® technology. Ages: Middle school and elementary school. firstinspires.org

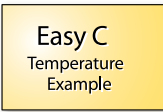
Marine Advanced Technology Education (MATE)—Competitions to build underwater ROVs (remotely operated vehicles). Programming is not required but can greatly enhance your robot. Ages: Elementary school through college. marinetech.org/rov-competition-2

VEX Robotics Competition (VEX)—Among the largest and fastest-growing robotics competitions in the world and also the most popular choice for school curriculums. Ages: Elementary school through college. vexrobotics.com and roboticseducation.org

You can use the websites listed to find a team and competition near you. If you can’t find a team nearby, start your own!



Robot created from VEX Robotics kit



48 PROGRAMMING

Programming for the Internet

What is the difference between your favorite online shopping website and a social media website or an online gaming site? The answer: nothing. They are all programmed websites using a similar type of programming. Internet programs are important because they allow people to access information, send email, or play games within a web browser.

If you could listen to a conversation between your computer and a website, you would hear many programming and internet languages. The various languages available to web developers are grouped into four main areas: query languages, scripting languages, markup languages, and programming languages.

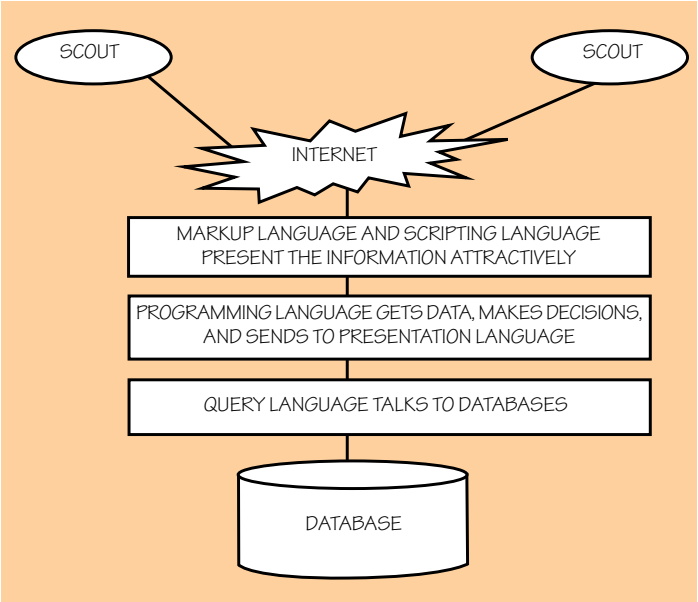
Programming languages offer the most flexibility. Can you make a website without a programming language? Yes, you can use a markup language such as HTML, but the website will simply sit there like a picture and won't be interactive. Every change will need to be made by a person.



While it is hard to get an exact number, the general consensus is that there are almost 1 billion websites in the world as of 2015. This is an increase of almost 400 million from 2012.

Web Languages

To understand the differences in web languages, you must first understand how web applications function. Once a request is received by a website, several layers must be navigated to present the information that was requested.



Databases

Databases are a critical part of a website. Adding a database to your website allows you to store, search, and organize large amounts of information. Think of it like a digital file cabinet containing many files. Leaving the files lying around creates a mess that is hard to organize. Putting those files into a well-organized system allows you to easily search and find what you're looking for. Databases help websites keep track of all the information and allow users to search and view that information.



Query Language

The query language is how you retrieve information from a database (files from the file cabinet) for presentation on-screen. To retrieve or alter information from the database, you use a query language like SQL (Structured Query Language).

Common commands for a query language include these:

SELECT allows you to search for the data you need.

INSERT allows you to add information to the database.

UPDATE allows you to make changes in the database.

DELETE allows you to remove information from your database.

Web Programming: Under the Hood

Once the query language gets the data, it hands the information off to the programming language for processing. This is where most of the programming happens. The programming language can decide what to do with the data: make decisions, search for specific things, or perform calculations. The programming language can also take a user's input, like a mouse click or keyboard entry, and make decisions. Think of this as asking and answering questions. The program asks: Do I have information to work with? What does the user want to do? Do I have the correct information? Do I need to change the information?

Internet programming languages offer many options. Choosing the language for a particular project often depends on the size of the project and the particular website features that will be needed.

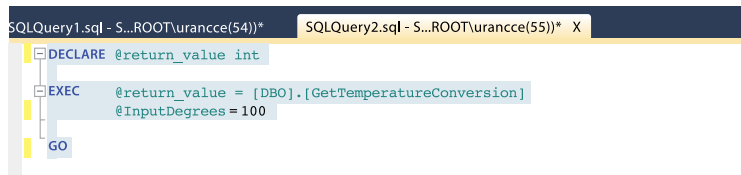
The programming language is the powerhouse of web development. It allows you to make changes to the data, process input from the computer screen, and do something with the data or input, if necessary.

SQL
Temperature
Example

```
Create proc [dbo].[GetTemperatureConversion]
@InputDegrees decimal(9,4)
as

select (@InputDegrees - 32.0) * (5.0/9.0) as DegreesCelcius
      , case when @InputDegrees <=32 then 'Pack Long Underwear'
              when @InputDegrees>100 then 'Remember to Hydrate'
              else " End as instructions

GO
```



Execution of the program

```
DECLARE          @return_value int

EXEC  @return_value = [dbo].[GetTemperatureConversion]
      @InputDegrees = 100

GO
```

Output of the program

Results		Messages
DegreesCelcius	Instructions	
1	37.7777400000	

Structured Query Language (SQL) is a special-purpose language designed for managing data in relational database management systems. SQL uses statements to execute database commands and to perform conditional operations and low-level functions. Unlike many of the other examples, SQL isn't designed to be a programming language and executes only when a user interacts with it.

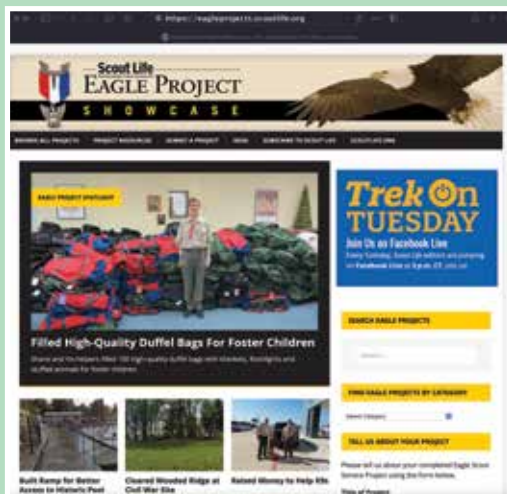
Common internet programming languages include these:

PHP—free, open-source. Easy to learn and used for small projects.

.NET—free and used for large-scale applications. Offers good performance, security, and stability. Limited to certain web servers; Windows®-based.

JAVA—free and used for small and large-scale applications. Offers good performance, security, and stability. Able to run on different web servers, such as Windows, Mac, and Linux.

Try this: Right-click on any webpage and select “View Source” or something similar, depending on the browser you are using, to view the source code of the webpage. You will see HTML and probably some JavaScript, too.



Markup and Presentation: What You See in the Browser

When your programming logic is completed, next is the presentation layer or interface. This is what you see on-screen when you visit a website. Think of this step as presenting a nice picture.

Typically, this final presentation is done with a combination of the languages, like adding style sheets, Hypertext Markup Language (HTML/HTML 5) for markup, and JavaScript for handling user input. These languages create what the user sees on the website.

Language Differences in a Nutshell

Programming language processes and organizes data and performs all the necessary decision making of your data to be handed off to the presentation interface. Examples: PHP, .NET, and Java

Markup languages describe how the data should be displayed on the screen, such as in bold, highlighted, or flashing green. Markup languages can be used simply to describe how the screen should be laid out, or they can describe all the coloring and other style elements of a webpage. Examples: HTML, CSS (Cascading Style Sheets), or XML (Extensible Markup Language)

Scripting languages, which can be embedded within HTML, are used to add functionality to a webpage, such as different menu styles or graphic displays, or to present dynamic (changing) information. Examples: JavaScript or jQuery

“HELLO WORLD!” EXAMPLES

JavaScript

```
<script type="text/javascript">
<!--function hello-World()
{
    alert('Hello World!');
}
// -->
</script>
```

SQL

```
Select * from myTable where text like "Hello World!"
```

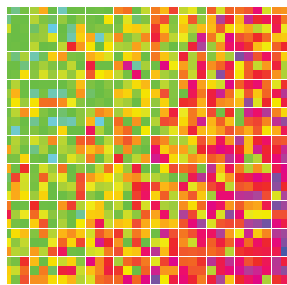
HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Hello World</title>
</head>
<body>
<h1>Hello World!</h1>
</body>
</html>
```

PHP

```
<? print("Hello World!"); ?>
```

Here is the same “Hello World” program in four different programming languages.



The word *pixel* is short for “picture element.”

Animation and Computer Graphics

A picture on a computer screen or a cell phone is made up of individual tiny dots, or *pixels*. Pixels can be stored and moved together in one big block, as in an image that you might record with a digital camera. Pixels can also be updated individually by a program.

A program can read or write the value of any pixel in the image. This value is a number that has a range depending on how many colors a computer monitor or smartphone can display. The larger the number, the more colors are possible.



On a computer, colors are represented by numbers, and each “color number” is split into three color components: red, green, and blue. These three colors, abbreviated RGB, are combined the way a painter might mix several paints to get something in between. Using these three primary colors, displayed in



various shades and strengths, any of 16,777,216 colors can be produced. For each color, 0 is the darkest, and 255 is the brightest when using eight bits to represent each color.

For example, when red, green, and blue are all dark—written as (0, 0, 0)—the color is equivalent to black; while RGB combined at each color’s brightest—(255, 255, 255)—is equivalent to white. Red at its brightest with the other colors dark—(255, 0, 0)—produces red; but red displayed only “half bright”—(128, 0, 0)—gives dark red or maroon. Green at its brightest with the other colors dark—(0, 255, 0)—is a bright shade of lime, while green displayed “less than half bright”—(0, 100, 0)—produces dark green.

Following this pattern, what color do you think (0, 0, 128) represents? (Hint: It’s a shade of dark blue with a nautical-sounding name.) What combination of red, green, and blue would produce violet? To check your answers, with your parent or guardian’s permission, search online for an RGB color-code chart.

How are pixels used in animation? If you have several small images that are the same size (height and width), you can write a program to display the images one at a time at the same place on the screen, with a brief wait between each image in the series. This has the effect of flipping through the images. If the “flipping” is done fast enough, the sequence of individual images can appear to move on the screen. That is, there’s “animation.”

A *frame* can be an individual drawing in a comic strip, a single picture on a strip of film, or a single complete image for digital display. When a movie is “streamed” on a computer or tablet device, each frame is drawn (displayed) many times a second to create the illusion of movement. (“Streaming” lets you view data while it is being received rather than having to wait for it to download completely.)

Computer programs can also draw images from scratch when no picture has been taken with a camera. This is the basis for computer animation, which draws each frame of a movie or sequence using mathematics to define objects in the scene, lighting sources, direction of view, and other factors. When three-dimensional (3-D) images are drawn or rotated, the mathematics involved can be advanced, using a form of math called trigonometry. Special effects can also be added to an image frame that might be impossible to capture with a camera, such as explosions, extreme detail, or the merging of human actors with added special effects.

All of these 3-D image manipulations and special effects are processed by the computer program, pixel-by-pixel. The program must keep track of every pixel at all times during the animation sequence, which is a huge task since most images have millions of pixels.

Because animation is used in movies, video games, websites, smartphones, tablets, and other devices and applications, learning how to program animations is a good career opportunity that is also fun.



Scratch
Temperature
Example

Enter degrees
Fahrenheit to
convert?



31

degreeF 212
degreeC 100.0

when clicked

go to x: 0 y: 0

ask Enter degrees Fahrenheit to convert? and wait

set degreeF to answer

set degreeC to degreeF

play note degreeF for 0.5 beats

if degreeF < 32

say Pack Long Underwear for 2 secs

if degreeF > 100

say Remember to Hydrate for 2 secs

say degreeC

stop script

move 10 steps

turn 15 degrees

turn 15 degrees

point in direction 90

paint towards

go to x1 y1 0

go to x1 y1 0

glide 1 secs to x1 31 y1 0

change x by 10

set x to 0

change y by 10

set y to 0

if on edge, bounce

x position

y position

direction

File Edit Share Help

Sprite 1

x: 0 y: 0

directions 90

Scripts

Costumes

Sounds

Control

Sensing

Operators

Looks

Pen

Variables

New Sprite:



Stage

x: -400 y: 250

Scratch, which was developed by the Massachusetts Institute of Technology, is a great programming language for beginners. It allows you to quickly create simple animations and games by dragging blocks from the left screen, or the blocks palette, onto the center pane, or the scripts area. Scratch is easy to learn and fun to experiment with.

Entertainment

Programming is essential to many forms of modern entertainment, both at home and away. When you consider various high-tech amusements, you may be amazed by how much programming goes into the devices we use for fun.

At Home

At home, programmed devices have become the heart of the entertainment center. You might not recognize all of them.

Let's start with the television. Many modern TVs can connect to the internet and stream videos or music using hardware or software designed for that purpose, without the need of a general-purpose computer. Some TVs can also display images and videos and play music stored on jump drives (small data-storage devices also known as flash drives or thumb drives). Such TVs have programming to handle these different forms of media.



Entertainment Programming Careers

In the entertainment industry, programming is vital, and it is big business.

Gaming. Video game programmers develop games for personal computers, tablets, smartphones, and game consoles. Gaming is a huge industry, with consumers spending billions of dollars each year on video games, hardware, and accessories.

Movies and television. Media production may require complex programming for animations, computer-generated images, and motion capture (that is, recording the actions of human actors and using that information to animate digital characters). Media editing and producing optical discs (such as DVDs and Blu-ray™ discs) also may involve programming.



Besides connecting to the internet, televisions may connect to many other programmed devices, including game consoles, disc players, and cable boxes.

Game consoles are special-purpose computers that run games and various kinds of media. The consoles include operating software and also execute specially developed game programs on removable media. Some consoles are programmed to support (work with) network-based multiplayer games.

Disc players such as DVD or Blu-ray™ players have programming to play DVD or Blu-ray videos. These devices are also often programmed to play audio or to connect with the internet to stream video or audio.

Set-top boxes supplied by cable or satellite television companies are programmed to receive, decrypt, and stream audio and video transmitted by the cable company. These devices also include programmed user interfaces that allow users to configure and select various features. Some set-top boxes have digital video recording (DVR) functions that save video content for later playback. A large market for these devices has led to the development of operating systems specifically for devices that manage television content, such as the OpenTV operating system developed in the 1990s and now found in well over 100 million set-top boxes.

And finally, digital televisions can also connect to computers and double as large computer monitors.

Outdoors

Programmed devices for entertainment are not limited to the home.

Many devices that people use outside also require serious programming. Programmed mobile computing devices allow people to communicate and play while on the go. Smartphones and tablets have all kinds of entertainment programming, from games to players for various types of media.



Programmed devices for entertainment include gadgets such as GPS units, music players, and e-books. Can you think of other programmed devices that people use for fun and adventure?



Science

Scientists study the world around us to discover how it works. Modern science requires lots of computing power to collect data, analyze the data, and ultimately make sense of it all. Here are a few examples of programming in science. Many more exist.

Chemical Instruments

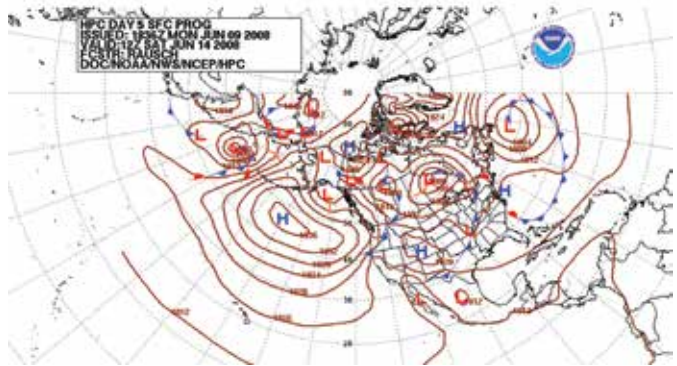
Chemists use various computer-controlled devices to test chemicals. These instruments can help determine what is in the samples being tested and how the chemicals behave. Programmers create the programs that control the devices so they can run for many hours without the operator needing to load new samples. One kind of instrument that is often used in chemical labs is the gas chromatograph. It helps scientists determine what is in liquid samples and how much of each chemical is present, using heat and gas pressure to separate the parts of the sample.



Materials Testing

Laboratories that test plastic, metal, and ceramic parts use physical testing instruments to measure the strength and durability of the parts. These instruments use control programs to operate the clamps and motors that hold and manipulate the parts that are being tested. Programs are also used to analyze the data that is collected during the tests. These tests help materials scientists determine whether critical parts in products will be durable and safe for the people who buy the products.

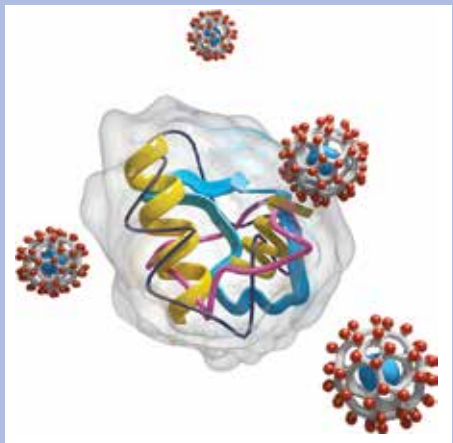




Weather prediction

Meteorologists and programmers work together to create weather forecasts. Scientists use computer programs to represent the atmosphere as a grid of points with each point containing math equations that describe weather. Millions of points form simulations that compute how weather patterns will evolve. Meteorologists collect weather data from weather stations all over the world. The data is input into simulations of the atmosphere. Math equations and large volumes of data require powerful supercomputers to answer the question of whether or not it will rain today.

Researchers use computers to create new drugs that fight cancer. With too many possible combinations of chemicals to test each one, scientists use simulations to predict how molecules will interact with cells. By testing many different possibilities with computers, scientists move a step closer to a cure for cancer.



The image shows a Notepad window titled "TempFtoC - Notepad" with the following Ruby code:

```

def getFTemp
  print "Enter Temperature Degrees F : "
  tempF = gets()
  tempC = (tempF.to_f - 32) * 5 / 9
  puts "Temperature Degrees C: #{tempC}"
  return tempF
end

def useTemp (temp)
  if (temp.to_f > 100)
    print "Hydrate!!"
  end
  if (temp.to_f < 32)
    print "Pack Long Underwear!!"
  end
end

begin
  Ftemp = getFTemp
  useTemp (Ftemp)
end

```

Below the Notepad window are two Command Prompt windows, both titled "Start Command Prompt with Ruby".

The first Command Prompt window shows the following output:

```

C:\Ruby193>
C:\Ruby193>TempFtoC
Enter Temperature Degrees F :102
Temperature Degrees C: 38.888888888888886
Hydrate!!
C:\Ruby193>

```

The second Command Prompt window shows the following output:

```

C:\Ruby193>
C:\Ruby193>TempFtoC
Enter Temperature Degrees F :24
Temperature Degrees C: 38.444444444444445
Pack Long Underwear!!
C:\Ruby193>

```

The Ruby programming language is used primarily for web application development. It is a simple, open source (free) object-oriented programming (OOP) language that is gaining popularity due to its flexibility and ease of use.

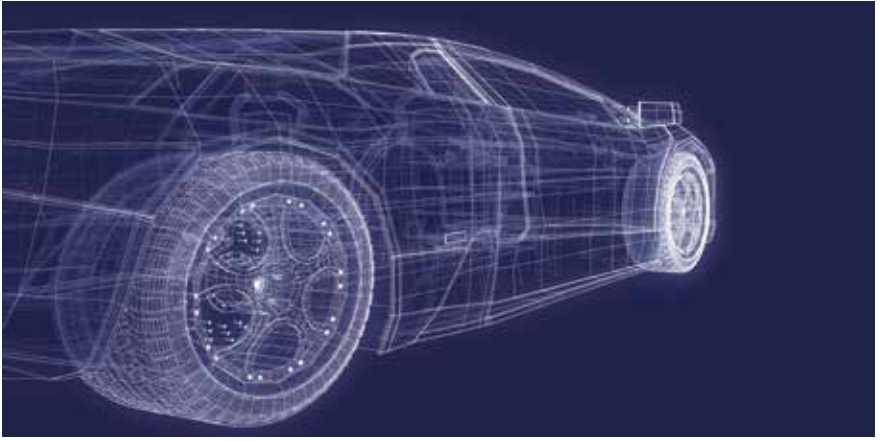
Engineering

Have you ever wondered how the space shuttle, rockets, or jet fighters were designed? Or how a bridge handles the stress of all those cars and trucks? Or how a building is designed to withstand an earthquake or hurricane?

Engineers apply science and mathematics to design and build dependable structures, systems, machines, devices, materials, and processes. The different fields of engineering include electrical, mechanical, civil, chemical, computer, and aerospace. All of these engineering disciplines require knowing how to program.

In fact, programs become more vital as engineering problems become more complicated. Engineers use programming to solve problems creatively and as inexpensively as possible. In the past, engineers had to rely on wind tunnels and scale models to test whether designs would work. Now, engineers use programs to create computer-generated models in place of physical mockups, which saves time and money and creates better designs. Whether for the tallest building in the world or the longest bridge, the plans can be designed and tested, all by computer.





To save time and money, CAD engineers can create and test new products in a virtual environment before production actually begins.

Computer-aided design (CAD) programs allow engineers to create models and schematics of their designs. The models can be placed into artificial environments where the engineer can virtually “walk” and move objects around. This solution avoids the need to build expensive physical models.

Programs have been created to check for flaws; measure the fit; and analyze the properties of things, such as stresses, temperatures, and movements. Customized programs have also been created for special purposes such as landing an astronaut on the moon, sending satellites into orbit, and producing heads-up displays for fighter pilots.

One thing common to all areas of engineering is problem-solving. How big does a thing have to be? How strong? Engineers design things based on the knowledge they have. Information about something may not always be available, and this is where computers are especially helpful. Programs can quickly solve complex math problems that humans cannot figure out. Engineers use computers in problem-solving so that they do not have to build something to find the answer. This saves money and time.



PYTHON
Temperature
Example

```
1 continueYN = "Y"
2
3 while continueYN == "Y":
4
5     degreeF = input("Enter next temperature in degrees F to check:")
6     degreeF = int(degreeF)
7
8     degreeC = (degreeF - 32) * 5 / 9
9
10    print("Temperature in degrees C is: %f" % degreeC)
11
12    if degreeC < 0:
13        print("Pack long underwear!")
14
15    if degreeF > 100:
16        print("Remember to Hydrate!")
17
18    continueYN = input("Input another?")
```

Python is a unique scripting language that uses indentation to form the if-then-else blocks in the code, making the code seem less cluttered. Like Perl, Python is supported on all computing platforms (Windows, Unix, Linux, Mac, and mobile devices) and executes immediately without the need for a compile step. The simplicity and self-documenting nature of the Python language make it a favorite language for script writers.

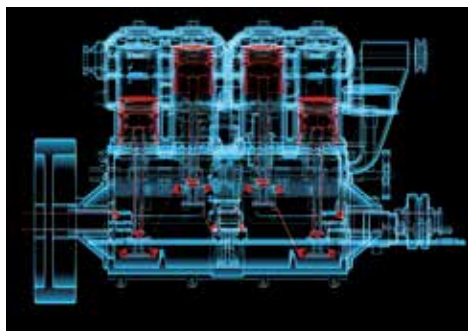
Engineering Programming

As an example, consider the design and building of an aircraft. Many different engineering talents are needed. Aerospace engineers work on the design and construction of the aircraft; they must understand the science of flying. Electrical engineers design the electrical systems that operate everything from the wheels to the lights. Computer engineers create the programs that control many of the devices on a plane, including navigation systems, flight controllers (autopilot), and temperature controls. Computer engineers also help visualize the designs of new aircraft using CAD tools.



To solve engineering problems such as aircraft design, general-purpose programming languages may be used, including Fortran, C, C++, BASIC, and Pascal. Specialized software helps with the complex mathematics involved. Engineering and scientific programming also uses object-oriented programming, parallel programming, and various modern languages such as Java and Ada.

Whatever engineering discipline you might choose to pursue as a career, all fields of engineering require knowledge and ability in programming. Programming can open the doors to many career opportunities in engineering, from building race cars and fighter jets to ensuring that buildings won't fall in an earthquake or a tornado.

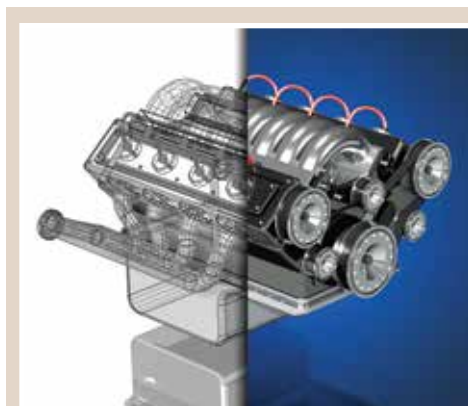


Automobiles and Traffic Control

Most cars today have dozens of computer controls for safety, performance, fuel economy, and driver convenience. These computers exchange information and work together—a built-in communications network right in the car. Multiple networks may be dedicated to different parts of the car: one network for the engine and transmission;

another for the cabin controls and dashboard. Then a separate computer is programmed as a dedicated communications controller between the networks. Programs even fine-tune the engine for better performance and fuel economy. All these computers work together to operate different features of the car such as the engine, transmission, dashboard display—even the windows. Automobiles today require *lots* of programming!

An example of programming for safety is the antilock braking system (ABS). Drivers used to be taught to pump the brakes to maintain control of a skidding car, especially in snow or ice. With ABS, the programming controls the on/off action of a brake much faster than a person could. This allows for better control in a skid. Even on a dry road, a driver making a very quick stop might hear or feel the rapid operation of the ABS.



The processors in a typical modern car require an estimated 50 million lines of programming. Luxury cars with up to 100 processors could have 100 million lines of programming to handle features such as automatic parking, hybrid drivetrains, tire air pressure, collision avoidance, etc. Some cars have as much programming as a modern fighter jet or commercial aircraft.

For security, a car may have a program that communicates with the remote key control. A driver can remotely lock and unlock the car, open windows and a sliding door, and even set off the car alarm.

Additionally, there is much programming for the dashboard, or instrument panel. The displays may be digital, and a driver information screen may present messages. Also, the radio cluster may have GPS, maps, and satellite radio, all operated by a touch screen. The doors may have a separate program to control window motion. Some cars have automatic one-touch operation of windows.

None of these features for comfort, convenience, or security would be possible without a computer program that someone has written. The program senses the environment, makes decisions on that information as well as user commands, and acts accordingly. All these actions must be done safely, quickly, reliably, and repeatedly, as expected by car owners.

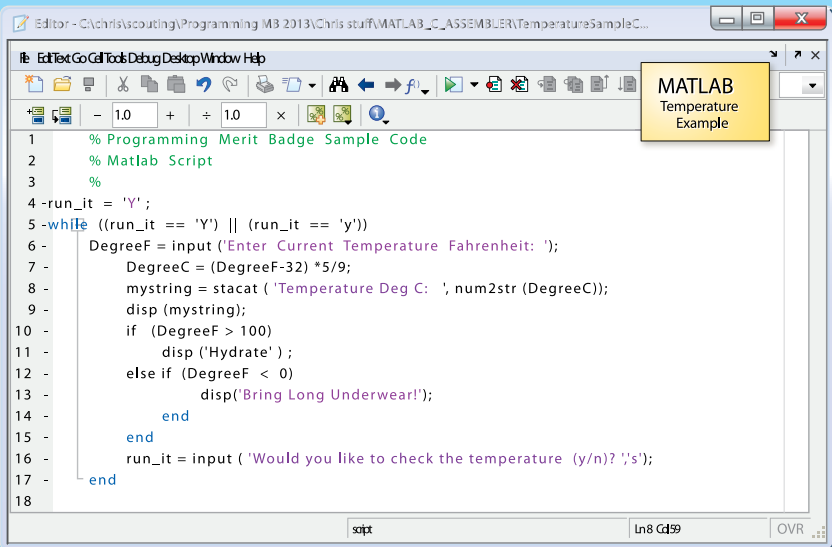
Traffic Control

Traffic control on roadways is necessary in order to have many cars operate safely together. Red-yellow-green signal lights are a national standard in the United States and in many other countries. These traffic-control devices are programmed to safely control the sharing of intersections by cars traveling on crossing roadways.

A traffic-light program has multiple inputs. Its own timer program controls the progress of the program. Also, there may be a communications connection to an external processor. The program can be developed and tested on that external processor, then downloaded into the traffic-controller processor and run. The output would be the signals to control circuits that activate and deactivate the traffic lights. These control programs can also be used to coordinate multiple signals at multiple intersections to improve the flow of traffic.

If a program is designed well, it's easy to add more features such as left-turn control from one or both streets. Another possible feature is for blinking lights overnight, when traffic is sparse. The timing of all the traffic lights could be adjusted based on other inputs such as sensors in the road. A roadbed sensor would detect a car's approach and adjust the traffic lights accordingly.





```

1 % Programming Merit Badge Sample Code
2 % Matlab Script
3 %
4 run_it = 'Y';
5 while ((run_it == 'Y') || (run_it == 'y'))
6     DegreeF = input('Enter Current Temperature Fahrenheit: ');
7     DegreeC = (DegreeF-32) *5/9;
8     mystring = strcat('Temperature Deg C: ', num2str(DegreeC));
9     disp(mystring);
10    if (DegreeF > 100)
11        disp('Hydrate');
12    else if (DegreeF < 0)
13        disp('Bring Long Underwear!');
14    end
15    end
16    run_it = input('Would you like to check the temperature (y/n)? ');
17 end
18

```

Command Window

```

>> TemperatureSampleCode
Enter Current Temperature Fahrenheit: 72
Temperature Deg C:22.2222
Would you like to check the temperature (y/n)? y
Enter Current Temperature Fahrenheit: 101
Temperature Deg C:39.3333
Hydrate
Would you like to check the temperature (y/n)? Y
Enter Current Temperatue Fahrenheit: -20
Temperature Deg C:-28.8889
Bring Long Underwear!
Would you like to check the temperature (y/n)? n
fx >> |

```

Matlab is a scripting language with lots of powerful numeric analysis functions. Matlab is widely used in the science, engineering, and mathematics fields.

Transportation Safety

Programming can be simple or complex, depending on the purpose. A program can be designed to adjust its behavior based on environmental input from sensors. Programs for safety-critical machines need to be tested many times and in different situations to be sure they operate properly, reliably, and safely.

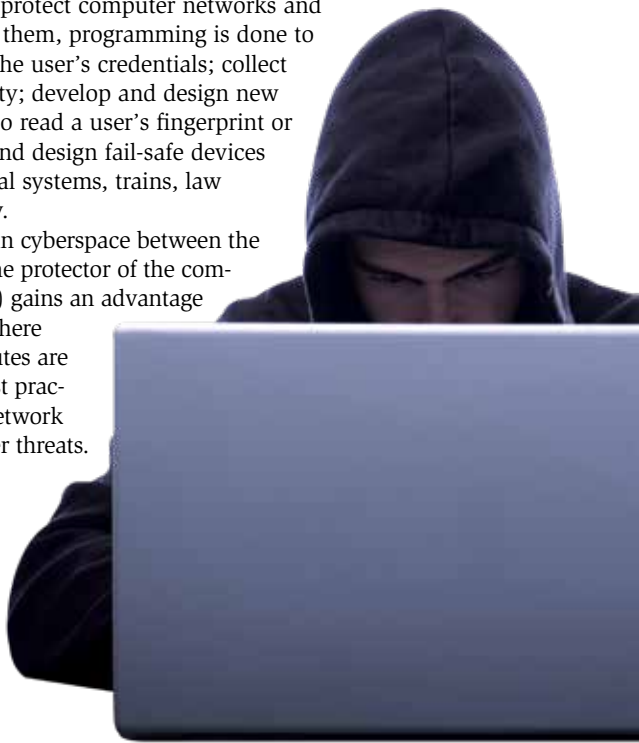
Programming is not limited to automotive transportation. Other means of transportation such as airplanes, trains, ships, and mass transit require just as much or even more programming. As you can see, programming is essential to all areas of transportation.

Computer and Information Security

Hackers are people who try to bypass the normal design of a program or stop a computer from working altogether. They often start out as programmers and experiment with the boundaries of what a computer can do, but wind up working for criminals. These hackers use the internet as their highway into many computer systems.

Computer security is a growing and rapidly changing area of computer programming. To protect computer networks and the businesses that depend on them, programming is done to encrypt (encode) data; check the user's credentials; collect and analyze logs of data activity; develop and design new and better biometric sensors (to read a user's fingerprint or voice patterns, for example); and design fail-safe devices for automotive and aeronautical systems, trains, law enforcement, and public safety.

A constant battle goes on in cyberspace between the hacker and the defender. But the protector of the computer system often (not always) gains an advantage because hackers can go only where the wires let them. If all the routes are known and protected using best practices, then the computer and network will be less vulnerable to hacker threats.

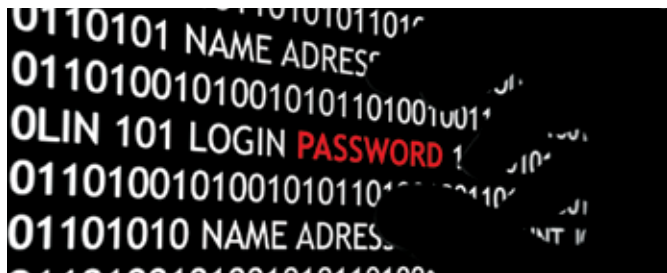


The “CIA” of Computer Security

One rule among programmers is that security should be designed into a program from the beginning. That is, try to think about how someone might intentionally cause the program to not work as expected, and write the program to handle all situations.

When designing for security, programmers keep three principles in mind: *confidentiality*, *integrity*, and *availability*. These principles are sometimes referred to as the “CIA” of computer security.

Confidentiality means that only the people who are authorized to see information are able to see it. Confidentiality can be protected by requiring passwords to computer accounts, using encryption to hide the data from plain viewing, or keeping a laptop or thumb drive in a locked desk drawer.



Integrity means the data being handled by the program is not altered from how it was originally stored or transmitted. An example is a bank program that processes a transfer of \$100 from one account to another. If the program did not check data integrity and a hacker got into the system, the hacker might change the amount and transfer \$10,000 instead. Fortunately, there are ways to check if data has changed along its path.

Availability means the computer is up and running and able to process instructions. A common internet attack that uses mobs of “zombie computers” (computers that have been secretly taken over by a hacker) is known as a distributed denial of service (DDOS). The “denial” happens when many computers are coordinated to request the same webpage from the same server at the same time. This overloads the web server and can cause it to crash. While the server is offline or is rebooting, no one can use it. Some businesses process millions of dollars a day using their internet connections, so being offline for a few hours can mean a huge loss of money.



Home Computer Security

These principles of computer security are used every day in business, but can also be used on your own laptop, your computer at home, and even a smartphone.

- If you have a home network, you can ensure its **confidentiality** by making sure it is running with strong Wi-Fi encryption on the wireless access point.
- You can help to ensure the **integrity** of a computer by running an update on the operating system to get the latest security patches. These are usually released by the software's seller for free to plug holes or weak points that might allow a hacker to get control of it.
- You can safeguard a computer's **availability** by running a backup of the disk and data files. A backup gives your data some protection from a hardware failure or a lost laptop or device. The data files can be loaded onto a replacement device. A spare battery or a power adapter are other ways to maintain the availability of a mobile computer.

The most advanced area of computer security programming is called reverse engineering. This involves examining a compiled module and trying to determine the code that was used to produce it. Most often, a programmer using reverse-engineering skills will analyze a virus that has infected a computer to figure out how to neutralize and remove the virus, and also determine what kind of problems it could cause.



Positive Identification

Another growing area of computer security involves user authentication. A concept called *two-factor identification* guarantees a person actually is the person he or she claims to be. This assurance is becoming increasingly important as people do more business online and do not actually meet each other face to face.

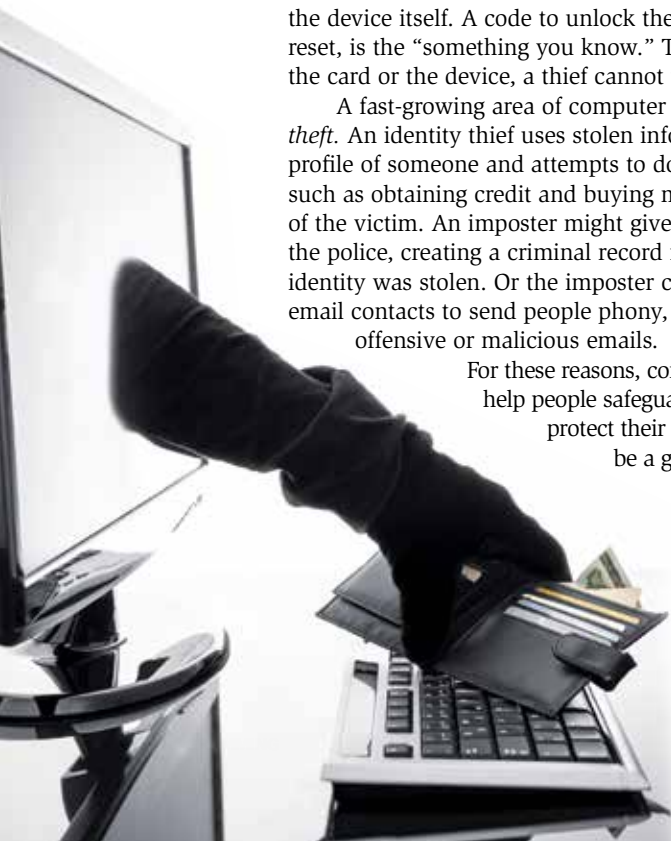
Two-factor identification means that any two things from this group can be used:

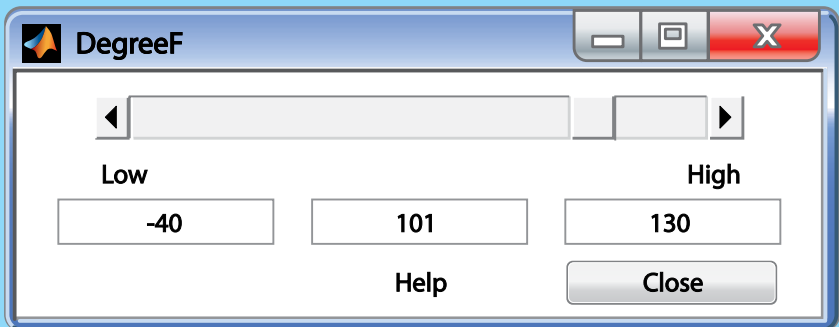
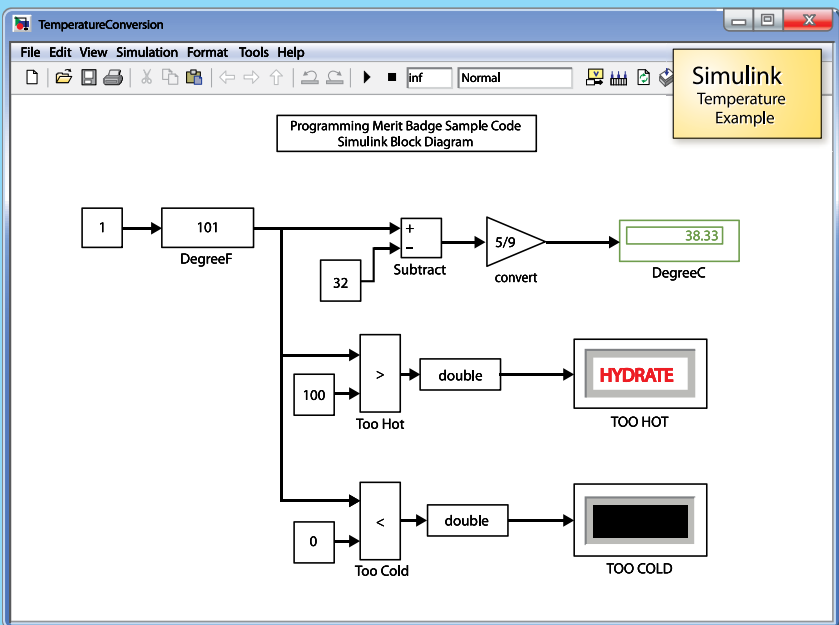
- Something you know (a secret, like a password)
- Something you have (a hardware token or a smart card)
- Something you are (a biometric characteristic like a fingerprint, an iris scan, or a facial scan)

If only one of the above is used, a hacker could copy or steal just that detail; the most common single-factor security system uses only passwords. But when a second factor is added, a hacker is far less likely to get both details. The most common form of two-factor identification is the smart card, which is in the device itself. A code to unlock the device, which you can reset, is the “something you know.” This means that if you lose the card or the device, a thief cannot use it.

A fast-growing area of computer hacking is called *identity theft*. An identity thief uses stolen information to create a fake profile of someone and attempts to do illegal things with it, such as obtaining credit and buying merchandise in the name of the victim. An imposter might give a false identification to the police, creating a criminal record for the person whose identity was stolen. Or the imposter could use the victim’s email contacts to send people phony, unwanted, and often offensive or malicious emails.

For these reasons, computer programming to help people safeguard their identity (and protect their money and reputations) will be a growing area in the future.





The Simulink programming language uses a block diagram. In this example, the program is launched from the main window by selecting the source file and is executed by using the “play” arrow. The user then selects a value by double-clicking on DegreeF to get the slider.



Training for a Programming Career

Along with the ability to think logically and solve problems, a good programmer also needs patience and creativity. Complicated programs demand creative approaches and patient attention to detail. A programmer also needs good communication skills and the ability to work in teams. Equally important, however, is the ability to work alone for long periods of time. Programming can be a lonely and isolated type of work.

Programmers are constantly learning. In the fast-changing world of technology, they must keep up with the latest advances and master complex new languages that are only now being developed.

Educational Opportunities

Some programmers are self-taught. Programming languages can be learned through online courses, guidebooks, and home study. Writing code and working out the bugs on your own is a straightforward way of getting practical experience.

When you have chosen the one career opportunity in programming that most interests you, your merit badge counselor or school guidance counselor can help you identify the specific education, training, and experience your chosen field requires.



Careers in Programming

As you can see from all the different languages and industries covered in this pamphlet, programming is used EVERYWHERE. And that means programmers are used everywhere, too. It would be hard to find an industry today that doesn't have some need for programmers.

Programmers can be self-taught or have college degrees. Regardless, with programming you are never finished with learning—there is *always* something new to discover. This may explain why the biggest factor in getting programming work is experience, and the more you can get, the better off you will be.

While you can make programming your career focus, more and more today, programming is becoming just another tool you use to get something else done. Very often people choose a field of study that interests them (not programming specifically) and then learn how to use programming to support their line of work. So whether you become a hard-core programmer or just use programming as a tool to complete a task, programming is an awesome skill to have!



Intellectual Property

When you create a new product, like a bicycle, you have a physical object you can touch and hold. When you create a computer program, however, you can hold the physical media that stores the programming but you cannot hold the programming itself. With the bicycle, the value to people is in the physical object. But with computer programs, the value is in the programming, not the physical disc or other media on which the program is stored.

Intellectual property, or IP, refers to creations that cannot be touched or held, such as ideas, plans, and designs.

Intellectual Property Rights

Over time, different kinds of rights have been developed to protect items you cannot touch and hold. IP rights include copyright, patent, trademark, and trade secret. Different aspects of computer programs may be legally safeguarded by each of these forms of protection.

Copyright Protections

A copyright protects a particular expression of an idea that an author has created. The author gets the copyright to the work when it is created and stored in some tangible media. For example, in a game involving a spaceship flying around a screen shooting asteroids, the programmer would get copyrights to several distinctive and original aspects of the game once the game is created.



First, the programmer would get a copyright on the particular background screen showing space and obstacles. The copyright would be limited to the unique background screen created by the programmer, so other people would be free to create their own background screens with space and obstacles. If someone copied the particular background screen from the copyrighted game, the programmer could rely on the protections of copyright to stop that other person from using the copied background screen.

Also, the programmer would get a copyright on the actual set of instructions that causes the game to operate. Someone else could create programs to perform similar game concepts, as long as the other person did not copy the exact or a very similar set of instructions developed by the programmer.

The Digital Millennium Copyright Act of 1998 extended the protection of copyright holders by imposing penalties for attempts to violate copy protection. Copy protection, also known as content protection, aims to stop the illegal reproduction of computer software as well as movies, music, and other media.


 US0006200138B1

(12) **United States Patent**
 Ando et al.

(10) Patent No.: **US 6,200,138 B1**
 (45) Date of Patent: **Mar. 13, 2001**

(54) **GAME DISPLAY METHOD, MOVING DIRECTION INDICATING METHOD, GAME APPARATUS AND DRIVE SIMULATING APPARATUS**

(75) Inventors: **Takeshi Ando, Kazumari Tsukamoto, Toshiya Yamaguchi, Tomoya Takasugi, Masaki Ito, Toshikazu Goh, all of Tokyo (JP)**

(73) Assignee: **Sega Enterprises, Ltd., Tokyo (JP)**

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

(21) Appl. No.: **09/182,909**
 (22) Filed: **Oct. 30, 1998**
 (30) Foreign Application Priority Data
 Oct. 30, 1997 (JP) 9-208852
 Oct. 28, 1998 (JP) 10-307321

(51) Int. Cl. **G09B 19/16**

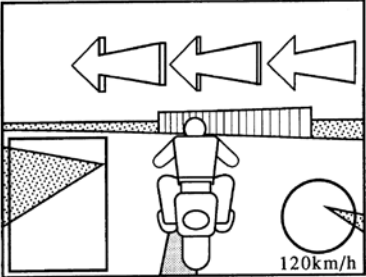
(52) U.S. Cl. **434/61; 434/307 R; 434/66; 434/29; 273/148 R; 273/442; 463/31; 463/23**

(58) Field of Search **434/61; 247; 307 R; 434/66; 29; 70; 308; 305; 273/148 R; 442; 348/121; 343/6; 463/23; 36; 34; 32**

(55) References Cited
 U.S. PATENT DOCUMENTS
 5,547,382 * 8/1996 Yamasaki et al. 434/61 X
 5,643,179 * 4/1997 Coppens et al. 434/69 X
 * cited by examiner
 Primary Examiner—Michael O'Neill
 Assistant Examiner—Chanda Harris
 (74) Attorney, Agent, or Firm—Dickstein Shapiro Morin & Oshinsky LLP

(57) **ABSTRACT**
 A game display method displays a driving game which permits characters to be present in a city and can prevent cruel images of collisions with characters. Characters in a dangerous area are intentionally moved away from a motorcycle B. Those H3, H4, H5 of the characters behind the motorcycle B as viewed in a moving direction of the motorcycle B are intentionally moved away from a current position H1 of the motorcycle B, a position of the center of the motorcycle B. Those H3, H4, H5 of the characters in front of the motorcycle B as viewed in the moving direction of the motorcycle B are intentionally moved toward the back of the motorcycle B, i.e., directions normal to a straight line interconnecting the position H1 of the center of the motorcycle B and the characters H3, H4, H5. The characters H3, H4, who are forward left of the motorcycle B, are moved left, and the character H5, who is forward right, is moved right.

26 Claims, 14 Drawing Sheets



Patents

A patent protects useful and innovative processes or methods, machines, manufactured items, or “compositions of matter” (things made of two or more substances or ingredients). Unlike copyrights that are automatic when the work is created, patents must be applied for. To seek patent protection, a programmer must file an application with the United States Patent and Trademark Office. A government patent examiner reviews the application to determine whether it is for something truly new and useful, and not just an obvious combination of instructions for performing the idea of the program. The application process is often lengthy and typically requires the services of a patent attorney or a patent agent.

As an example, think about the asteroid shooting game mentioned earlier. If aspects of the game are innovative, the programmer may be able to get patent protection. The programmer could then use the patent to stop others from making, using, selling, importing, or offering for sale the programmer's patented invention.



Trademarks

A trademark protects a word, phrase, symbol, sound, or color—the “mark”—that identifies and distinguishes the source of a particular product or service. You receive trademark rights by using the mark in connection with the sale of a product or service. Those rights develop over time as customers begin to connect the mark with a product.

People often use a superscript TM (™) next to their mark to put others on notice that they are claiming trademark rights. If the owner of a mark registers it with the United States Patent and Trademark Office, the owner of the mark may use a ® to show that the mark has been registered.

Programmers often use trademarks to keep others from using a similar name for a competing product. For example, if the programmer named the asteroid shooting game “Astroblitz,” he or she might want trademark protection to make sure someone else does not sell a competing asteroid shooting game with the same or a very similar name.





Trade Secrets

A trade secret protects commercially valuable information for which the owner has taken strong measures to maintain secrecy. This protection will exist as long as the information remains a secret.

In the example of the asteroid shooting game, the programmer could keep the source code a trade secret. To do so, the programmer would need to strictly control access to the source code. He or she would need to put everyone who has access to the code under an obligation to maintain the secrecy. This might be done by having everyone with access sign a contract promising not to reveal the secret.

Unlike a patent, trade secret protection will not prevent the use of the protected information by those who independently develop it or by those who acquire it by legitimate means. This means another programmer could independently develop a nearly identical program as long as he or she has not done so based on inappropriate access to the first programmer's source code.

How Software Is Sold

The value of software is in its electronically coded instructions, not in the plastic disc or other tangible media upon which the instructions are stored. Software developers do not sell DVDs or other storage media. What they actually sell is permission to use the programs they have created. This is called *licensing* the software.

Licensing is more like renting a bike than buying one. When you buy a bike, you own it outright and can do what you want with it. When you rent a bike, however, you have the right to use it, but you may have some restrictions on how long and where you can ride it. Similarly, with software, you purchase a license to use the software in a manner specified by the terms and conditions of the license.

Software may be licensed in several ways, including:

Freeware. Obtaining freeware costs nothing. “Free,” however, does not necessarily mean unlimited. For example, the license could restrict use to particular users (e.g., personal, noncommercial users), or the license could limit the ability of an individual user to redistribute the software.

Shareware. A software developer may release a program as shareware and expect people who use the program to make a donation or pay a fee. The developer is relying on your honor to help cover the costs of developing the software.

Demo. Demo or trial software will work for a limited time or with limited features. After the purchaser pays a license fee, the programmer provides a code that removes the limitations.

Open Source. An open-source license is a form of freeware. This particular type of license covers the executable program *and* the source code developed by the programmer. This license may come with limitations or restrictions.



Software Piracy

The unauthorized use of another person's intellectual property is called *piracy*. Software piracy is a costly, worldwide problem. According to the Business Software Alliance's 2011 Global Software Piracy Study: "[T]he global piracy rate hovered at 42 percent in 2011 while a steadily expanding marketplace in the developing world drove the commercial value of software theft to \$63.4 billion."

The Federal Bureau of Investigation is charged by the U.S. government to investigate piracy and intellectual property theft. This includes ideas, inventions, and creative expressions, such as trade secrets, music, movies, and software. The FBI's Anti-Piracy Warning Seal is used to help "detect and deter criminal violations of U.S. intellectual property laws by educating the public about the existence of these laws and the authority of the FBI to enforce them."

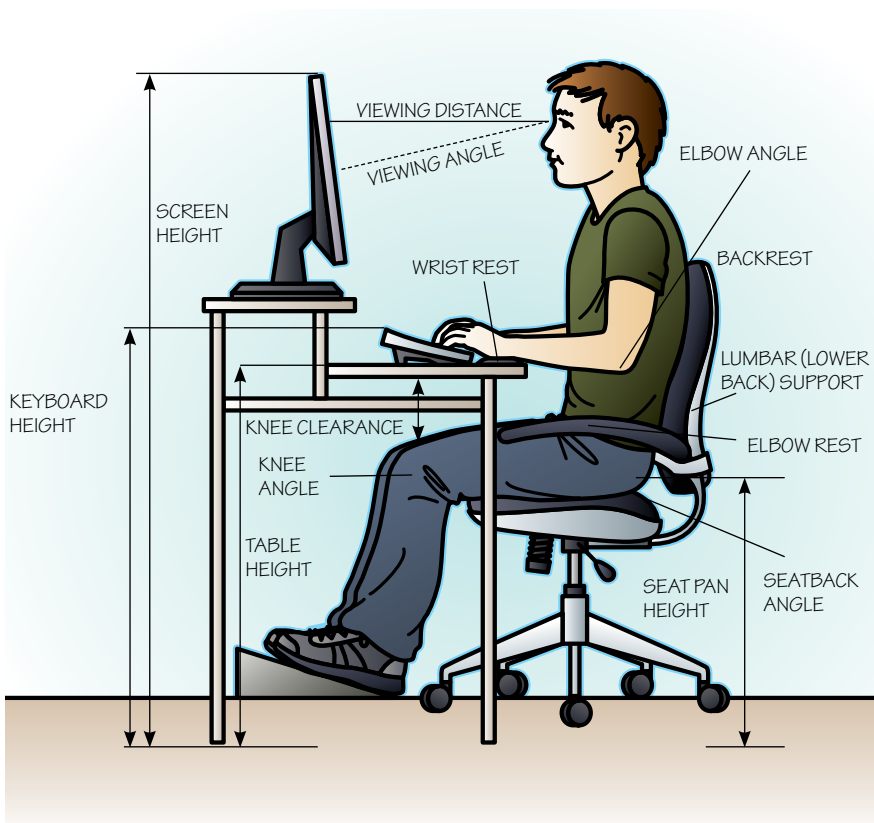


You have probably seen this seal alongside the following text:

The unauthorized reproduction or distribution of a copyrighted work is illegal. Criminal copyright infringement, including infringement without monetary gain, is investigated by the FBI and is punishable by fines and federal imprisonment.

A Scout Is Trustworthy

Stealing software is like stealing anything else. It's wrong. You would not sneak into a theater to see a movie without purchasing a ticket, or let a friend in through the back door without paying. Similarly, you should only use software you have paid for and not make copies for your friends.



Ergonomics is the science of equipment design, intended to increase efficiency by reducing operator fatigue and discomfort. For proper ergonomics:

- Monitor should be about an arm's length away from eyes (18–24 inches).
- Top of screen should be level with eyes so you look slightly down at the monitor.
- Knees should be at an open, 90- to 120-degree angle (legs not folded under you).
- Elbows at a 90-degree angle.
- Wrists resting on support and straight.
- Mouse directly in front of elbow (you shouldn't have to reach for it).
- Keyboard at elbow height.
- Seated with back against backrest, back at 90-degree angle to legs.
- Feet flat on floor or on footrest.
- Head balanced on neck, not tilted too far back or forward.

Safety

You might not think that writing programs could cause injuries. But in fact, injuries from programming have much in common with sports injuries and can be prevented using some of the same techniques that athletes use.

The main programming-related injuries are *repetitive stress injuries*, or RSIs. Just as in athletic activities, an RSI occurs when stress is placed on a joint, pulling on the tendons and muscles around the joint. When the stress happens repeatedly, the body does not have time to recover and becomes irritated. The body reacts to the irritation by increasing the amount of fluid in that area to reduce the stress placed on the tendon or muscle.

And just as athletes are trained to do, you can avoid issues like the following by developing good habits.

Carpal tunnel syndrome—swelling inside a narrow “tunnel” formed by bone and ligament in the wrist that can lead to pain, tingling, and numbness; the tunnel surrounds nerves that conduct sensory and motor impulses to and from the hand

Cervical radiculopathy—disk compression in the neck, often caused by repetitive cradling of a phone on the shoulder

Epicondylitis—elbow soreness often called “tennis elbow”

Reflex sympathetic dystrophy—a painful condition marked by dry, swollen hands and loss of muscle control

Tendonitis—tearing and inflammation of tendons connecting bones to muscles

Besides developing RSIs, programmers can become dehydrated, experience back problems, and develop eyestrain and headaches.



Injury Prevention

Proper equipment and preparation are the keys to prevention. Make sure your equipment is set up properly to prevent injuries (see the ergonomics sidebar).

Also be aware that hydration isn't solely for hiking or playing sports—programmers need to stay hydrated too. Four 8-ounce glasses of water per day is a good number. Soda, juices, and other sweet drinks are **not** a substitute for water.

To help prevent eyestrain, program in a well-lit room. Minimize the contrast between your monitor and the rest of the room but make sure there is no glare on the screen.

Proper posture is the key to avoiding back pain and injuries (see the information on ergonomics). And just as when you participate in athletic activities, taking breaks to give your body time to recover and prevent strain is essential.



Take a Break

Eye breaks. It's easy to get so focused on programming that you don't move for hours. Take eye breaks—look away from the monitor from time to time, preferably at something more than 20 feet away. This gives your eyes a chance to relax and helps prevent eyestrain.

Typing breaks. Most typing is done in bursts. Rest your hands in a relaxed, flat, straight manner to give them time to recover and prevent RSIs.

Rest breaks. Take a break every 30 minutes or so to give your body a chance to relax. You can use software programs that remind you to take a break so you don't get stuck in a trance staring at your monitor.

Exercise breaks. Get up and stretch, rotate your head and shoulders, move your arms and legs. You will find you can program better and longer if you do this regularly.



First Aid for RSIs

- Apply an ice pack to the injured area to help reduce pain and swelling.
- Use an elastic joint support or wrap the area firmly with an elastic bandage to limit the swelling and to protect the injury. Do not wrap it so tightly that blood circulation is restricted.
- Rest the injured area.
- Take an anti-inflammatory pain reliever as recommended by your physician.
- After 24 hours, heat (hot packs, heating pad, whirlpool) may be applied.
- As symptoms diminish, gently exercise the affected muscles or joints to help relieve remaining tenderness, stiffness, and tingling or numbness.
- If pain is severe or persistent, seek medical attention.

Features of a Human-Friendly Computer Workstation

- Sufficient indirect lighting to prevent eyestrain and glare
- Monitor at eye level to prevent hunching over
- Keyboard at elbow height for arm and shoulder alignment
- Padding in front of keyboard for wrist alignment and relaxation
- Adjustable seatback support for lower back
- Footrest for comfort and stability

Electrical Safety

If you are writing a program on a device that is plugged into the wall for power, you are dealing with potentially deadly electronic circuits. Keep liquids and food away from plugged-in machines, and make sure any cords are neatly stowed to prevent tripping. Be sure the equipment is properly grounded to prevent shock hazards. It's best to unplug the computer when it is not in use, especially during a thunderstorm. All of these suggestions protect you and the computer, too!

Programming Terms



binary code. A numbering system that uses only two digits. In programming, the two digits are a 1 and a 0, which represent turning an electronic circuit on and off.

bit. Short for *binary digit*. The smallest possible unit of information. Each bit represents one electronic switch (or “transistor”) in the processor that can be on or off.

byte. A group of eight bits of information.

client. Part of a client-server connection. The client is an application, such as an email program, that runs on a personal computer. That computer is networked to another computer, called a server, which helps the client perform its work, such as sending an email.

coding. Writing instructions using the protocol (rules) of a particular programming language.

commercial software. Software produced commercially and purchased for use.

compiled programming language. A programming language that is translated from a high-level language into another language, usually machine code.

compiler. A program that decodes instructions written in an English-like language and then translates, or compiles, them into machine language.

development environment. A collection of programming tools used for developing, testing, and debugging a program or software application.

embedded processor. A small special-purpose computer usually dedicated to a single task.

ergonomics. Designing and arranging the things people use so that the people and the things interact most efficiently, comfortably, and safely.

flowchart. A diagram that shows step-by-step progression through a procedure or system, usually using connecting lines and basic symbols. To show the flow of work, a process flowchart typically uses a rectangle for a step in the process; a diamond for a *decision point*; a flattened oval for the start or end of the flowchart; and a circle for a connection to another page of the chart.

macro. A sequence of commands in a software application that can be recorded or directly programmed to repeatedly execute the sequence.

malware. Damaging or “malicious” software intended to disrupt a network or a single computer. Types of malware include viruses, worms, and trojans.

operating system. Software that allows the computer to perform basic functions.

portability. A program’s ability to run on a variety of processors. Software that is portable (also known as *machine-independent*) does not depend on a particular type of hardware.

program. A set of processor instructions.

programming language. The language used to write instructions that a processor can understand or interpret.

protocol. The rules processors use to communicate with each other.

pseudocode. An outline of a program, written in a form resembling plain English that can be converted into real programming statements.

source code. The original code used to create the program. Depending on the language, this code could be compiled down to machine code or just interpreted as is.

state diagram. A type of diagram used in computer science to illustrate the possible states, or stages of behavior, and all the possible paths a program can use to transition from one state to another.

structured programming. A style of programming aimed at improving the clarity, quality, and development time of a program by using blocks (sections of code that are grouped together) and subroutines.

subroutine. A sequence of program instructions to perform a specific task, written as a unit that can then be used in programs wherever that particular task should be performed. Subprograms may be defined within programs, or separately in libraries that can be used by multiple programs. In different programming languages, a subroutine may be called a function, a routine, or a subprogram.

trojan. A virus or harmful program disguised to look like a useful program, such as a screensaver.

workstation. A terminal or personal computer usually connected to a computer network, or a powerful microcomputer used especially for scientific or engineering work.

worm. A software program that, once installed on a computer, copies itself and sends its copies over a network.



Programming Resources

Scouting Literature

Animation, Communication, Digital Technology, Electronics, Game Design, and Robotics merit badge pamphlets

With your parent or guardian's permission, visit Scouting America's official retail site, **scoutshop.org**, for a complete list of merit badge pamphlets and other helpful Scouting materials and supplies.

Books

Foxall, James. *Sams Teach Yourself Visual Basic 2015 in 24 Hours*. Sams Publishing, 2015.

Henney, Kevlin. *97 Things Every Programmer Should Know: Collective Wisdom From the Experts*. O'Reilly Media, 2010.

Horstmann, Cay S. *C++ for Everyone*, 2nd ed. Wiley, 2010.

Newsome, Bryan. *Beginning Visual Basic 2015*. Wrox, 2015.

Sharp, John. *Microsoft Visual C#, 9th ed.* Microsoft Press, 2018.

Watson, Karli, Jacob Vibe Hammer, Jon Reid, Morgan Skinner, et al. *Beginning Visual C# 2012 Programming*. Wrox, 2012.

Organizations and Websites

Android

Tutorials for Android app building developer.android.com/guide

Code.org

Free tutorials and introductions to programming
code.org

InterConnecting Automation Inc.

Free access to Scouts (send them a note); learn about PLCs (programmable logic controllers)
interconnectingautomation.com

The best place to start your programming journey is with the companion website for this merit badge, scoutlife.org/ programming. There you will find many examples and free resources appropriate for Scouts. You will be up and running quickly and be able to find what you need to fulfill the Programming merit badge requirements.

Learn C++

Free tutorials and other resources on how to program in C++
learncpp.com

Leampython.org

Interactive Python tutorial
leappython.org

Oracle Corporation

Java tutorials
docs.oracle.com/javase/tutorial

Robotics Academy of Summer Learning

From the Carnegie Mellon Robotics Academy, animation, robotics, web design, game design, and more
cs2n.org

Scratch

Good, free examples of programs
scratch.mit.edu

U.S. Copyright Office

copyright.gov

U.S. Patent and Trademark Office

uspto.gov

W3schools.com

Tutorials for all web design programming tools
W3schools.com

Acknowledgments

Scouting America thanks the following members of the Programming Merit Badge Development Team, who diligently worked to develop the merit badge requirements and content for this pamphlet.

Special thanks to AutomationDirect.com, an industry leader in factory automation, for providing the services of Rick Folea. He coordinated the development of the Programming merit badge, and AutomationDirect.com supported the launch of the merit badge at the 2013 National Scout Jamboree with equipment, supplies, and personnel.

Rick Folea—Eagle Scout; Programming merit badge lead; AutomationDirect.com; *Robotics* merit badge pamphlet co-author

Celeste Rance—Software development manager, Pearson

David Kerven, Ph.D., J.D.—Associate professor of Information Technology, Georgia Gwinnett College

Robert J. Caruso, CISSP—Information security architect, Battelle Memorial Institute; U.S. CyberPatriot mentor and lifelong Scouter

Laszlo Hideg, Ph.D., P.E.—Electrical engineering components, electric motor controls, Chrysler LLC; *Robotics* merit badge pamphlet co-author; *FIRST*[®] Robotics competition judge, volunteer, and mentor of Team 33, “The Killer Bees”

James Francisco, Ph.D.—Associate faculty, School of Advanced Studies, University of Phoenix, software quality engineering, software test automation

Steve Wincor—Chief systems architect, Lockheed Martin

Curtis Heisey—Eagle Scout; Master of Arts in physics; software engineer, MIT Lincoln Laboratory; coach, Team America Rocketry Challenge

We would also like to acknowledge James Trobaugh, Chip McDaniel, Eagle Scout Christopher Hideg, Talia Heisey, and Theresa Folea for their assistance in developing the code examples used throughout this book and on the companion website. Thanks to Eagle Scout Christopher Folea for graphics support and the cover design concept. We also appreciate VEX Robotics and RadioShack Corporation for their assistance with providing hardware in support of this project.

Photo and Illustration Credits

AutomationDirect.com, courtesy—pages 39 and 41–43 (*programmable logic controllers, distribution plant, denim factory*)

AutomationDirect.com/Christopher Folea, courtesy—page 62 (*molecules graphic*)

Frank da Cruz, courtesy—page 12 (*loom*)

Federal Bureau of Investigation, courtesy—page 85

Rick Folea, courtesy—page 47

Christopher Hideg and Laszlo Hideg, courtesy—page 10

LEGO, the LEGO logo, MINDSTORMS, and the MINDSTORMS logo are trademarks of the LEGO Group. ©2011 The LEGO Group. Colors and decorative designs may vary.—page 45 (*LEGO robot*)

Wikipedia.org/Evan-amos, courtesy—page 60 (*gaming system and controller*)

Wikipedia.org/Mcbort, courtesy—page 61 (*chemical instrument*)

Wikipedia.org/National Oceanic and Atmospheric Administration/Hydrometeorological Prediction Center, courtesy—page 62 (*weather prediction*)

Wikipedia.org/U.S. Dept. of Energy/LANL, courtesy—page 19

All other photos and illustrations not mentioned above are the property of or are protected by the Boy Scouts of America.

John McDearmon—page 86



Get ideas for your next merit badge adventure in every issue of *Scout Life* magazine.

Subscribe TODAY at
go.scoutlife.org/subscribe

Use promo code
SLMBP15

for a special Scout price!

