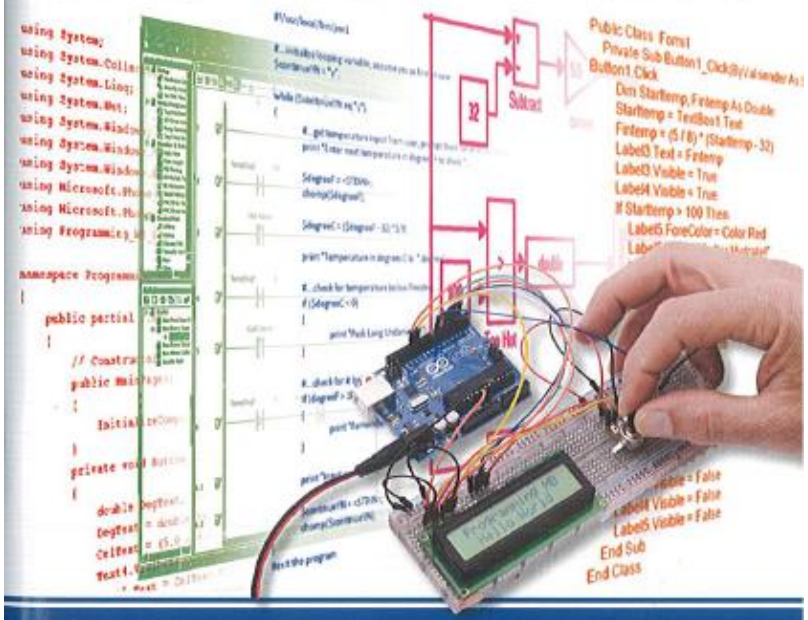
MERIT BADGE SERIES

PROGRAMMING



BOY SCOUTS OF 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 keep his project choices fun and relatively simple.

Supporting this merit badge is an online resource, www.boyslife.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 requirements 5a–5d).

Before the Scout spends a lot of time and effort on the requirements, he should be sure you will give him credit for that work.

35710
ISBN 978-0-8395-0028-2
©2013 Boy Scouts of America
2013 Printing

SUSTAINABLE FORESTRY INITIATIVE Certified Chain of Custody

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. The Scout should write the instructions in the languages and development environments of his 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 there is no requirement that the code be run 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

1. Safety. Do the following:
   a. Show your counselor your current, up-to-date Cyber Chip.
   b. Discuss first aid and prevention for the types of injuries or illnesses that could occur during programming activities, including repetitive stress injuries and eyestrain.

Earn the Cyber Chip

Earning the Cyber Chip can help you learn how to stay safe while you are online and using social networks or the latest electronic gadgets. Topics include cell phone use, texting, blogging, gaming, cyberbullying, and identity theft. Find out more about the Cyber Chip at www.scouting.org/cyberchip.

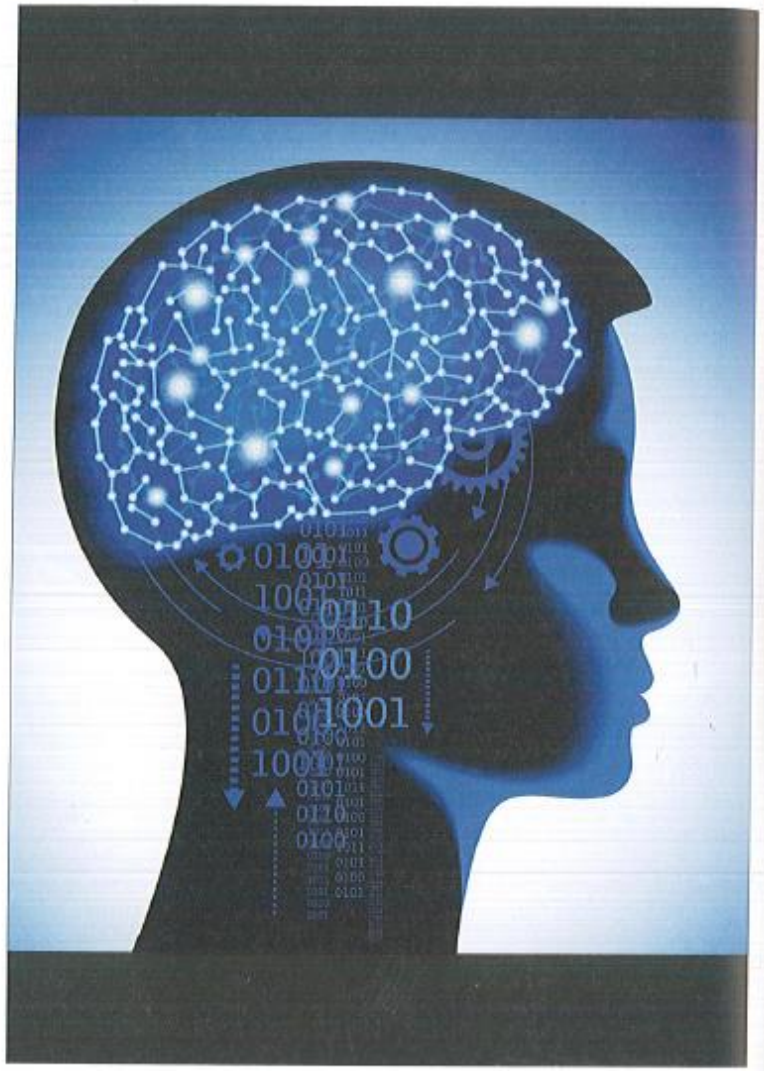
2. History. Do the following:
   a. Give a brief history of programming, including at least three milestones related to the advancement or development of programming.
   b. Describe the evolution of programming methods and how they have improved over time.
3. General knowledge. Do the following:
   a. Create a list of 10 popular programming languages in use today and describe which industry or industries they are primarily used in and why.
   b. Describe three different programmed devices you rely on every day.
4. Intellectual property. Do the following:
   a. Explain how software patents and copyrights protect a programmer.
   b. Describe the difference between licensing and owning software.
   c. Describe the differences between freeware, open source, and commercial software, and why it is important to respect the terms of use of each.

5. Projects. Do the following:
   a. With your counselor's approval, choose a sample program. Then, as a minimum, modify the code or add a function or subprogram to it. Debug and demonstrate the modified program to your counselor.

The Programming merit badge website, http://www.boyslife.org/programming, has a number of sample programs that you could use for requirement 5a. However, you have the option of finding a program on your own. It's a good idea to seek your merit badge counselor's guidance.

   b. With your counselor's approval, choose a second programming language and development environment, different from those used for requirement 5a and in a different industry from 5a. Then write, debug, and demonstrate a functioning program to your counselor, using that language and environment.
   c. With your counselor's approval, choose a third programming language and development environment, different from those used for requirements 5a and 5b and in a different industry from 5a or 5b. Then write, debug, and demonstrate a functioning program to your counselor, using that language and environment.
   d. Explain how the programs you wrote for requirements 5a, 5b, and 5c process inputs, how they make decisions based on those inputs, and how they provide outputs based on the decision making.
6. Careers. Find out about three career opportunities in programming. Pick one and find out the education, training, and experience required. Discuss this with your counselor and explain why this career might be of interest to you.

Contents



Introduction: The Code Behind the Screen

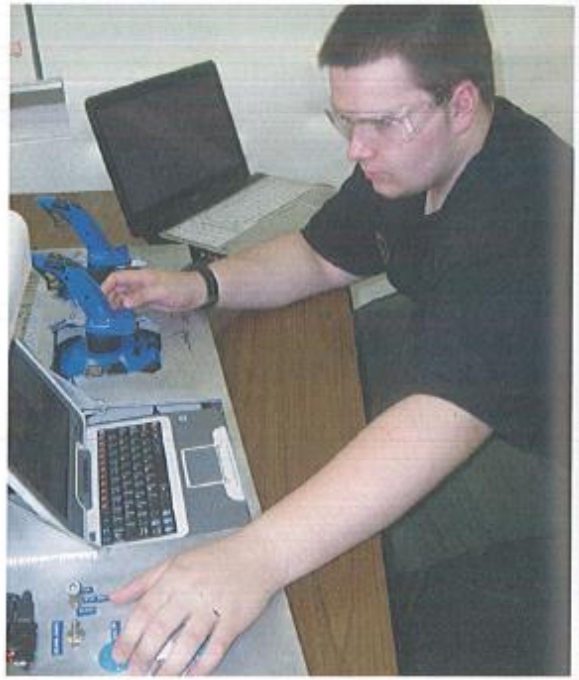
You have probably surfed the Internet, played video games, or posted updates on social media more times than you can count. But maybe you have never stopped to think about the underlying instructions that make these kinds of activities possible.

To do what they do, computers, game consoles, smartphones, and other digital devices follow instructions called *programs*. Each instruction or command in a program is put there by a person. People who write programs for digital applications are called *programmers*.

Earning the Programming merit badge will take you "behind the screen" for a look at the complex codes that make digital devices useful and fun. Without programs, today's high-tech gadgets would be little more than empty shells. But given clear instructions, digital devices can do amazing things and perform operations that would have seemed like magic to people in the past.

You benefit from the creative work of a programmer every time you download a new "app" to a smartphone or tablet computer, or play a video game, or update the maps in your GPS device, or upgrade your computer with the latest software.

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!



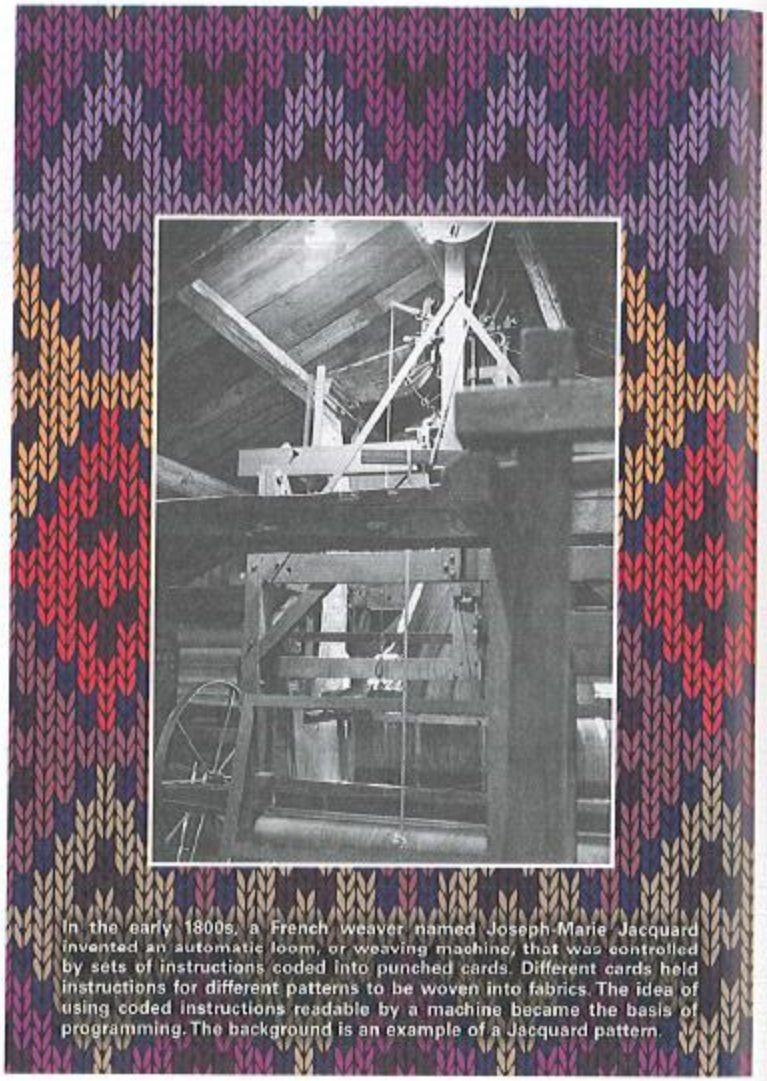
What Languages Do You Speak?

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 Startemp, Fintemp As Double
        Startemp = TextBox1.Text
        Fintemp = (5 / 9) * (Startemp - 32)
        Label3.Text = Fintemp
        Label3.Visible = True
        Label4.Visible = True
        If Startemp > 100 Then
            Label5.ForeColor = Color.Red
            Label5.Text = "Better Hydrate!"
            Label5.Visible = True
        ElseIf Startemp < 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



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.

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.

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*.

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.

```java
// 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: 38.888888888888886
Remember to hydrate
Enter a temperature in Fahrenheit: -2.
The temperature in Celsius is: -18.88888888888889
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, Ada, 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 (*For*mula *Tran*slation), 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 (*CO*mmon *B*usiness-*O*riented *L*anguage) 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.

"C" Temperature Example

```
// 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 Celsius (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
}
```

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: 38.888888888888886
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.

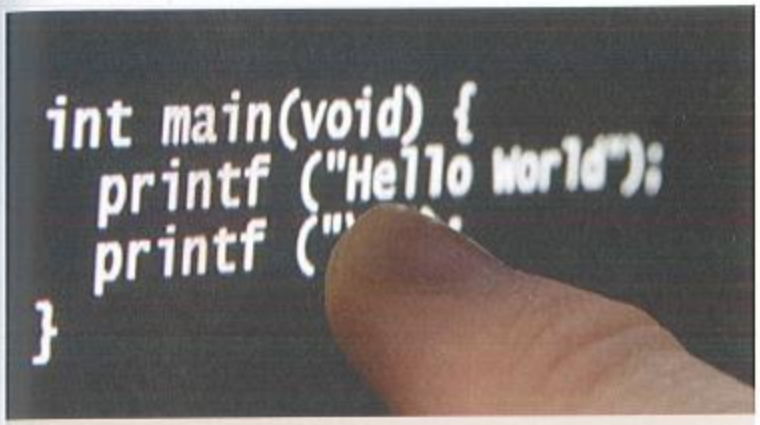
C# (C-Sharp)
Temperature
Example

```
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 operating 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 Web pages. 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 Web pages in a way that HTML could not. JavaScript allowed Web pages to be interactive.

PHP is a general-purpose programming language designed to produce dynamic Web pages. 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 *P*HP *H*ypertext *P*reprocessor.

PHP
Temperature Example

```
<?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);
    }
?>
```

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>
```

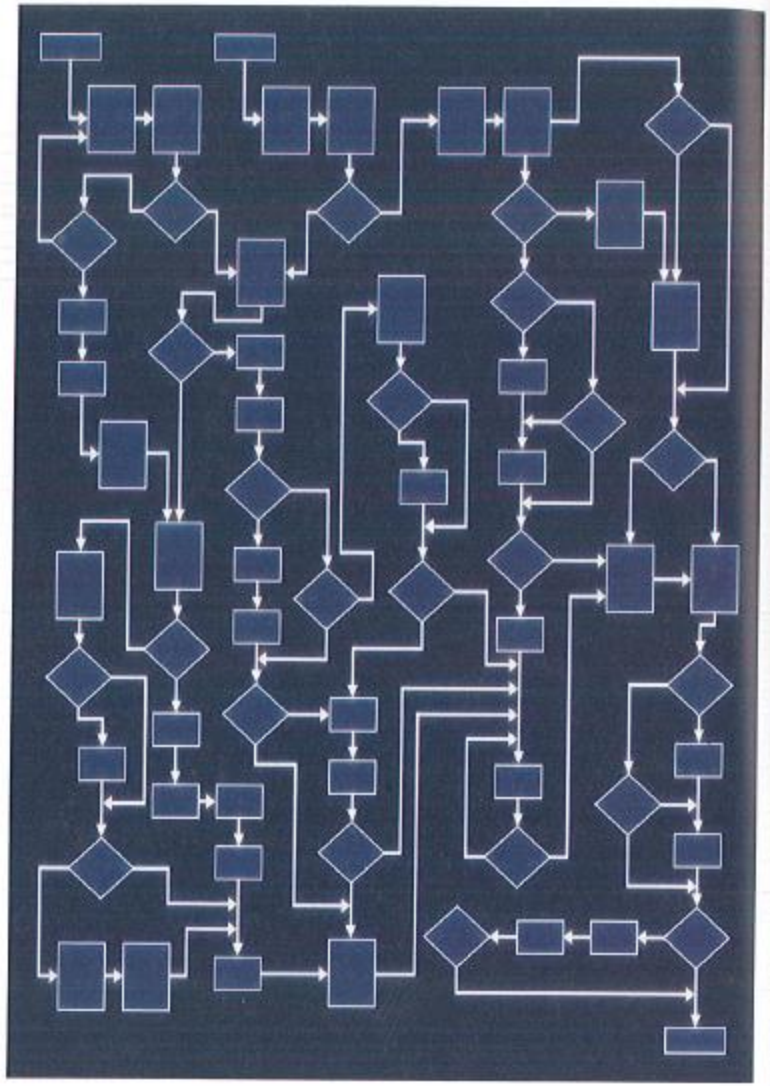
BSA PHP Temperature Conversion

You converted 101 F to 38 C: Remember to Hydrate

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

Degrees: [ ] Submit

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.

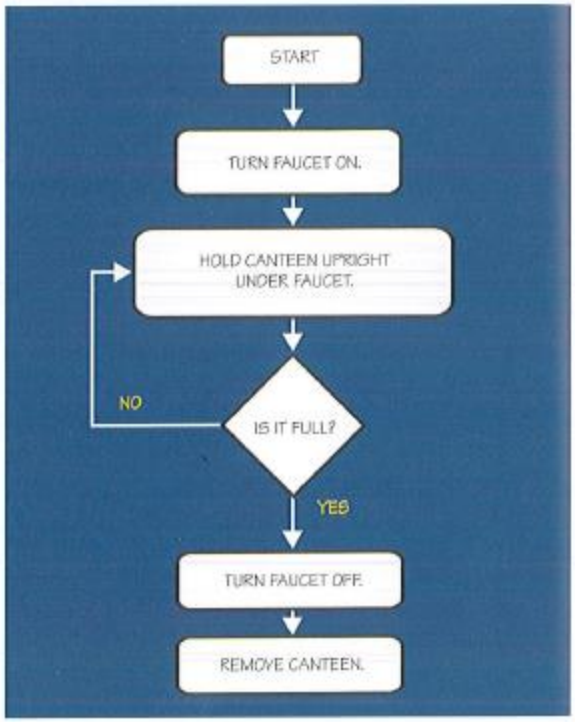
So how do these instructions get translated from an idea into something that processors will understand? This activity of converting ideas into instructions is called *programming*. When you create a program, the instructions can be simple or highly complex. It's a bit like working with uncut wood: the 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 to understand and to 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
```

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

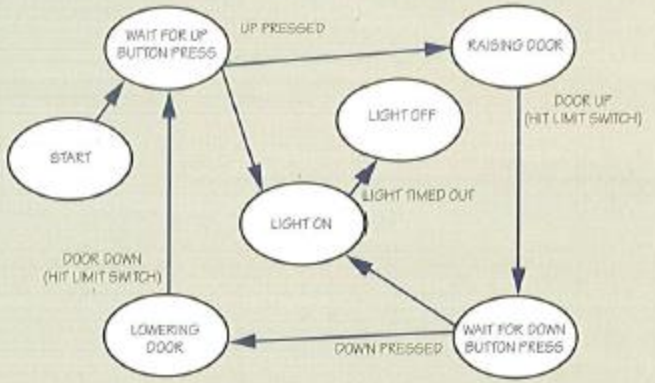
Step 3

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.

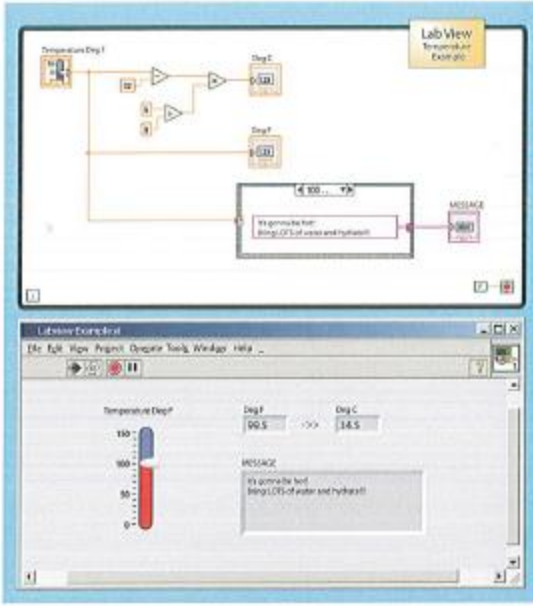
State Diagrams

State diagrams are great for describing programs that do tasks sequentially. 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*. It gives you a file that you can hand to the same processor, or any number of processors of the same type, to run. These other processors will not need the language translation program (that is, 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 Web page, 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, http://www.boyslife.org/programming. With your parent's permission, visit the website for hands-on tutorials, practical advice, and detailed support for fulfilling the Programming merit badge requirements.

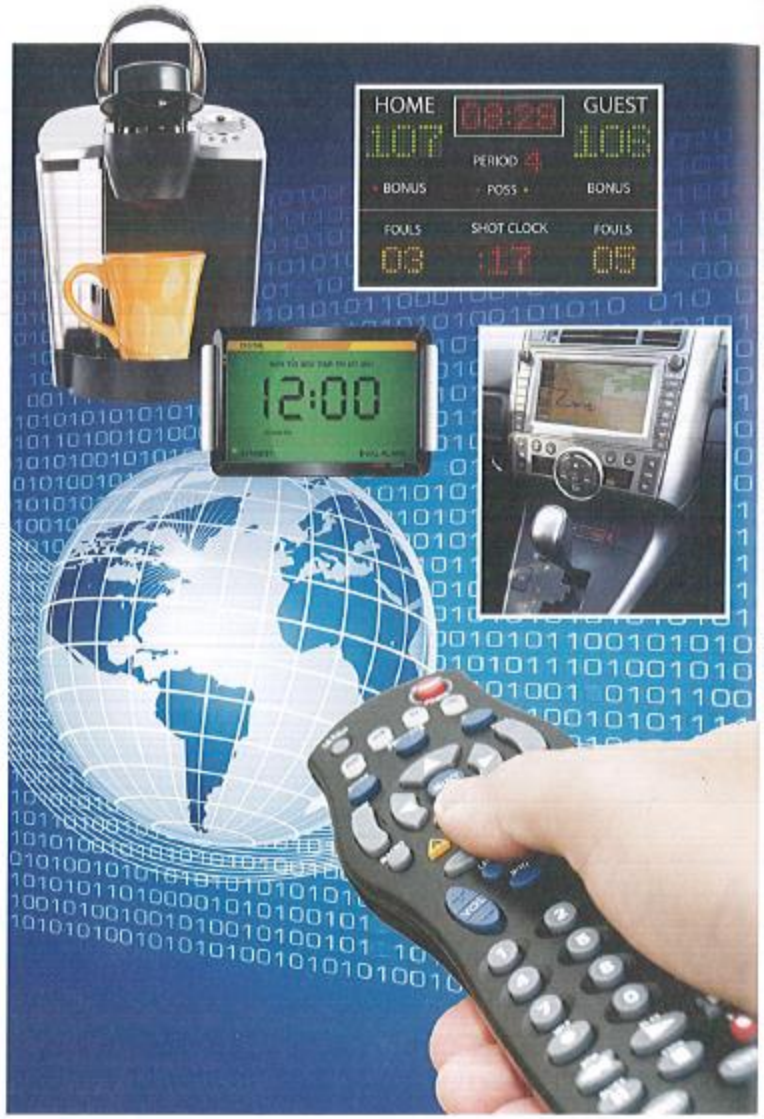
Also study the code "snippets" that appear throughout this pamphlet in the blue boxes. Each snippet performs a similar function but in a different language so you can compare them and get a feel for what each programming language looks like. Each example that appears in this pamphlet is also on the companion 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 5a–5c.

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 following the then 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 then. 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.



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 (www.boyslife.org/programming) to get up and running quickly. You will be programming in no time!

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

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.

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.)

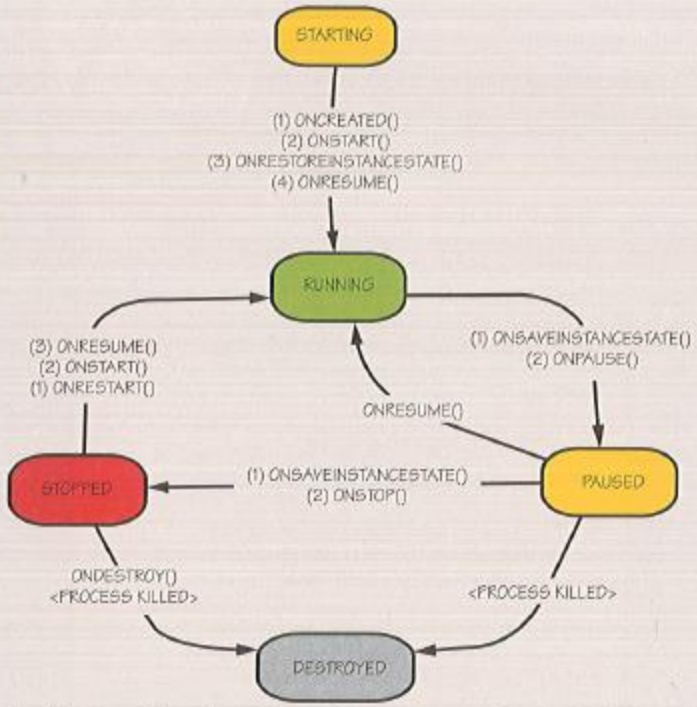
The learning curve for programming mobile devices can be steep. Don't let that stop you. Just start with simple examples and build from there. The next thing you know, you will be programming your own mobile-device applications. (Be sure to check out the Programming merit badge website for examples and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

PYTHON
Temperature Example

```
#!/usr/bin/env python3
#...initialize looping variable, assume yes as first answer
continueT= "Y"

while continueYN == "Y";
    #...get temperature input from the user, prompt them
for what we expect
    degreeF = input ("Enter next temperature in degrees F
to check:")

    degreeC = (degreeF -32)* 5/9

    print "Temperature in degrees C is: "% degreeC

    #...check for temperature below freezing....
    if degreeC <0;
        print "Pack long underwear!"

        #...check for it being a very hot day..
        if degreeF > 100:
            print "Remember to Hydrate!"

        continueYN = input("Input another?")

#exit the program
```

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.

Android life cycle



The diagram shows the various states a program using the Android operating system can be in: It can start the app, run the app, pause while the mobile device does something else, "destroy" the app (shut down), or stop. The items between each bubble are the program modules that can be used during each transition.

For example, suppose the mobile device needs to pause your program so it can do something else. This diagram tells you that the "onPause()" routine will be run when your program is paused, so you would modify the "onPause()" routine to save any work in progress. Then when the mobile device un-pauses your program so it can run again, the program will run the "onResume()" routine. You would modify the "onResume()" routine to restore the work in progress so your program can pick up where it left off. Your task as a programmer of a mobile device is to manage each of these transitions by modifying each of these routines appropriately.

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 an application that keeps track of your patrol dues, for example.

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—air conditioners, motors, sensors, lights, bottling machines, and so on—the machines themselves are not programmed. 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.

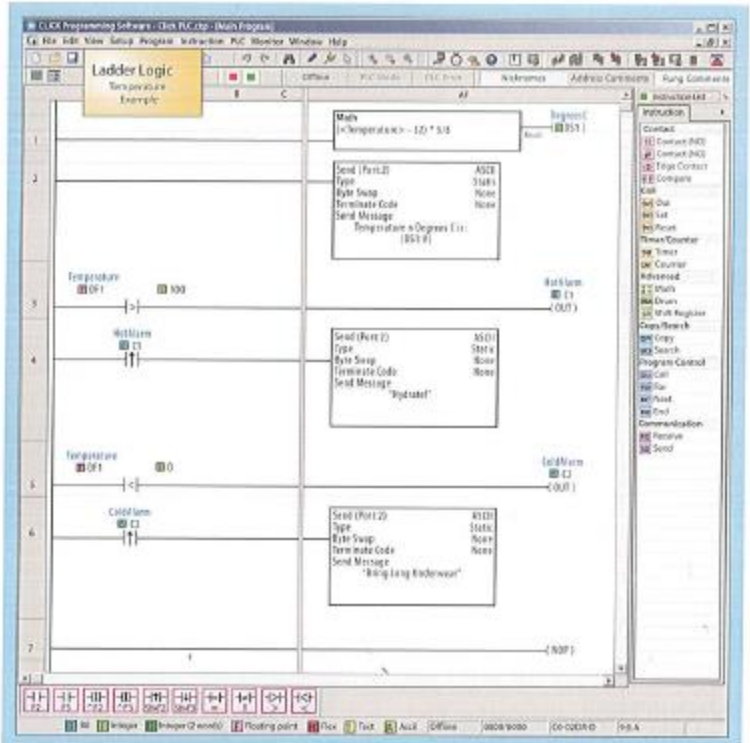
This PLC controls motors and sensors in an automated distribution facility, including this spiral conveyor system.

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."

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.

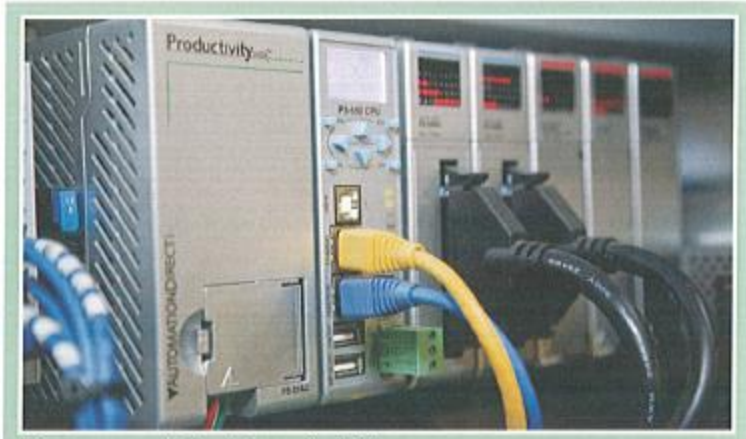


Ladder Logic gets its name from the way the code looks—like rungs on a ladder. Inputs and decisions are made on the left and outputs are on the right. Ladder Logic code is used in super-reliable and durable computers called PLCs, which control factory machines. Almost everything made in a factory today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

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

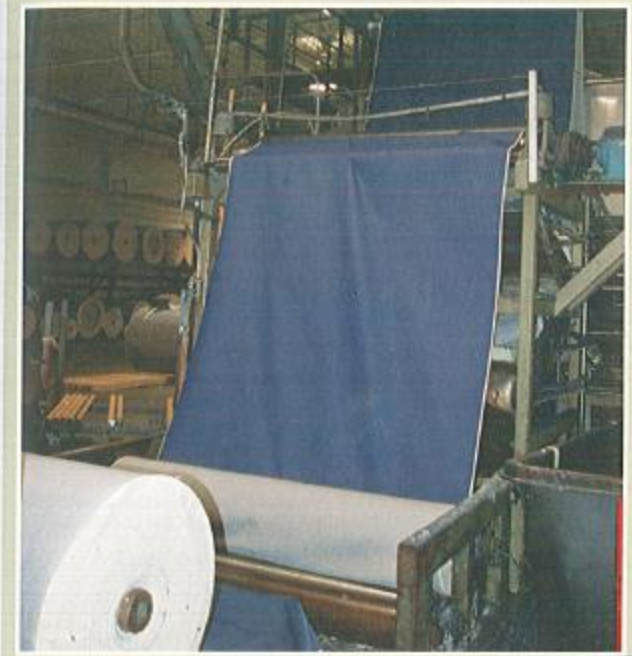
- They are designed to be rugged and reliable enough to operate continuously for many years in factory environments.
- They are programmed to do the exact same thing repeatedly, all day long, and they are typically programmed using Ladder Logic.
- They can read lots of inputs (sensors, switches, temperature,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

A PLC, or programmable logic controller



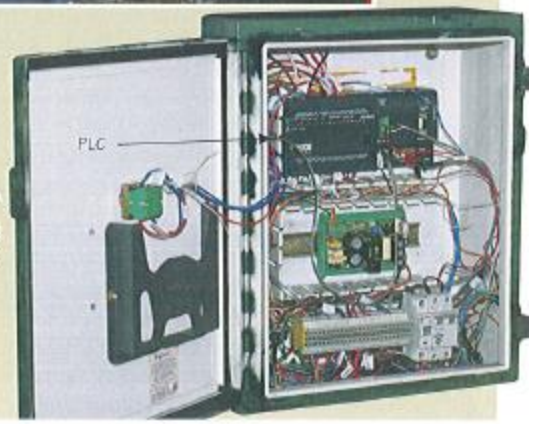
This Programmable Logic Controller (PLC) controls motors and sensors . . .

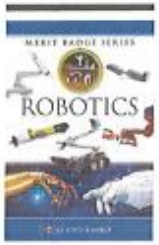
in this automated distribution plant.



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

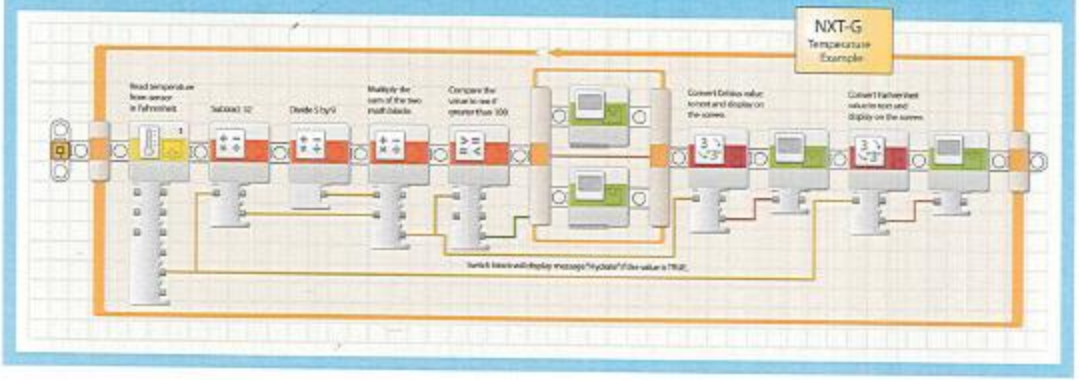
Programming a Robot

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 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.

Robot used in law enforcement

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. www.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. www.usfirst.org

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

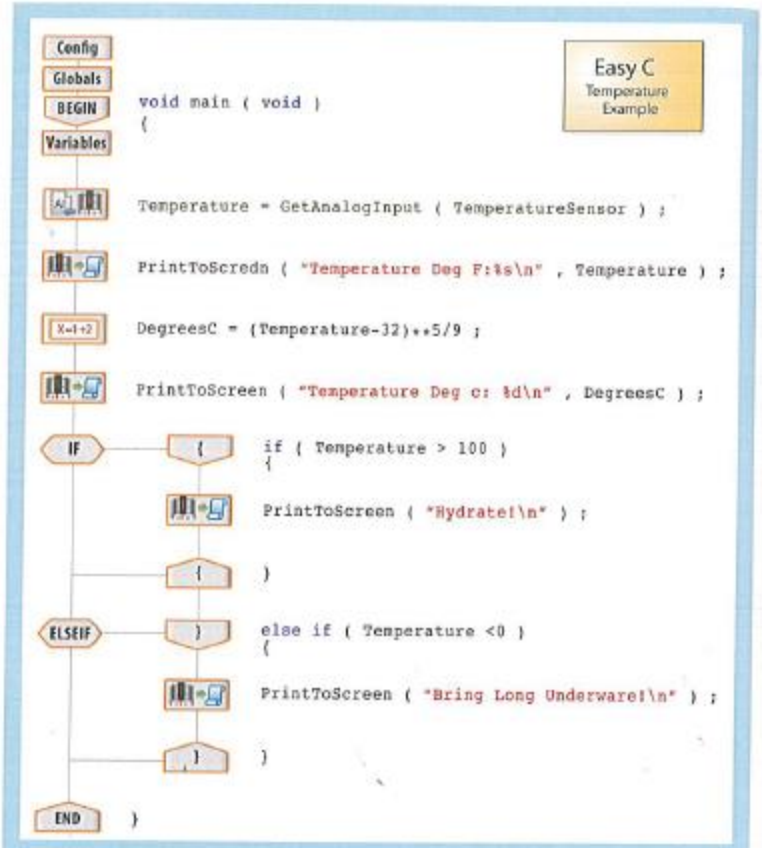
FIRST LEGO League (FLL)—A build-and-compete challenge using robots built with LEGO Mindstorms® technology. Ages: Middle school and elementary school. www.usfirst.org

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. www.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. www.vexrobotics.com and www.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



The popular Easy C robotics programming language is used around the world by VEX Robotics Competition teams. Easy C is a great language for beginner programmers to learn because it removes all the syntax normally encountered with text-based languages—which allows the robotics to get up and running quickly. Easy C allows you to view the actual C code generated, which will help you transition from graphical to text-based 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 were well over 600 million websites in the world as of December 2012. This is an increase of more than 70 million from the previous year. Billions of Web requests are logged each day. With so many websites in the world, learning how to program for the Internet could launch great career opportunities.