Sams Teach Yourself Arduino Programming in 24 Hours

Free Sample Chapter

Share with Others
## Contents at a Glance

Introduction .................................................................................... 1

### Part I: The Arduino Programming Environment

<table>
<thead>
<tr>
<th>HOUR</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Introduction to the Arduino</td>
<td>7</td>
</tr>
<tr>
<td>2</td>
<td>Creating an Arduino Programming Environment</td>
<td>23</td>
</tr>
<tr>
<td>3</td>
<td>Using the Arduino IDE</td>
<td>39</td>
</tr>
<tr>
<td>4</td>
<td>Creating an Arduino Program</td>
<td>57</td>
</tr>
</tbody>
</table>

### Part II: The C Programming Language

<table>
<thead>
<tr>
<th>HOUR</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>Learning the Basics of C</td>
<td>75</td>
</tr>
<tr>
<td>6</td>
<td>Structured Commands</td>
<td>89</td>
</tr>
<tr>
<td>7</td>
<td>Programming Loops</td>
<td>103</td>
</tr>
<tr>
<td>8</td>
<td>Working with Strings</td>
<td>119</td>
</tr>
<tr>
<td>9</td>
<td>Implementing Data Structures</td>
<td>133</td>
</tr>
<tr>
<td>10</td>
<td>Creating Functions</td>
<td>147</td>
</tr>
<tr>
<td>11</td>
<td>Pointing to Data</td>
<td>163</td>
</tr>
<tr>
<td>12</td>
<td>Storing Data</td>
<td>181</td>
</tr>
<tr>
<td>13</td>
<td>Using Libraries</td>
<td>201</td>
</tr>
</tbody>
</table>

### Part III: Arduino Applications

<table>
<thead>
<tr>
<th>HOUR</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>14</td>
<td>Working with Digital Interfaces</td>
<td>219</td>
</tr>
<tr>
<td>15</td>
<td>Interfacing with Analog Devices</td>
<td>235</td>
</tr>
<tr>
<td>16</td>
<td>Adding Interrupts</td>
<td>251</td>
</tr>
<tr>
<td>17</td>
<td>Communicating with Devices</td>
<td>267</td>
</tr>
<tr>
<td>18</td>
<td>Using Sensors</td>
<td>287</td>
</tr>
<tr>
<td>19</td>
<td>Working with Motors</td>
<td>303</td>
</tr>
<tr>
<td>20</td>
<td>Using an LCD</td>
<td>319</td>
</tr>
<tr>
<td>21</td>
<td>Working with the Ethernet Shield</td>
<td>337</td>
</tr>
<tr>
<td>22</td>
<td>Advanced Network Programming</td>
<td>355</td>
</tr>
<tr>
<td>23</td>
<td>Handling Files</td>
<td>373</td>
</tr>
<tr>
<td>24</td>
<td>Prototyping Projects</td>
<td>387</td>
</tr>
</tbody>
</table>

Index .......................................................................................... 403
# Table of Contents

Introduction ........................................................................... 1

**HOUR 1: Introduction to the Arduino** ................................................................. 7
   What Is an Arduino? ................................................................... 7
   Introducing the Arduino Family ................................................. 11
   Exploring Arduino Shields ....................................................... 18
   Summary .............................................................................. 20
   Workshop .................................................................................. 20

**HOUR 2: Creating an Arduino Programming Environment** .................. 23
   Exploring Microcontroller Internals ......................................... 23
   Moving Beyond Machine Code ............................................... 27
   Creating Arduino Programs ................................................... 29
   Installing the Arduino IDE ...................................................... 32
   Summary .............................................................................. 37
   Workshop .................................................................................. 38

**HOUR 3: Using the Arduino IDE** ................................................................. 39
   Overview of the IDE .................................................................. 39
   Walking Through the Menus .................................................... 40
   Exploring the Toolbar ............................................................ 49
   Exploring the Message Area and Console Window ................ 49
   Setting Up the Arduino IDE .................................................... 51
   Using the Serial Monitor ......................................................... 52
   Summary .............................................................................. 54
   Workshop .................................................................................. 54

**HOUR 4: Creating an Arduino Program** .................................................... 57
   Building an Arduino Sketch ..................................................... 57
   Creating Your First Sketch ...................................................... 59
   Interfacing with Electronic Circuits ......................................... 64
   Summary .............................................................................. 70
   Workshop .................................................................................. 71
<table>
<thead>
<tr>
<th>Contents</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>HOUR 5: Learning the Basics of C</strong></td>
<td>75</td>
</tr>
<tr>
<td>Working with Variables</td>
<td>75</td>
</tr>
<tr>
<td>Using Operators</td>
<td>80</td>
</tr>
<tr>
<td>Exploring Arduino Functions</td>
<td>83</td>
</tr>
<tr>
<td>Summary</td>
<td>87</td>
</tr>
<tr>
<td>Workshop</td>
<td>87</td>
</tr>
<tr>
<td><strong>HOUR 6: Structured Commands</strong></td>
<td>89</td>
</tr>
<tr>
<td>Working with the <em>if</em> Statement</td>
<td>89</td>
</tr>
<tr>
<td>Grouping Multiple Statements</td>
<td>90</td>
</tr>
<tr>
<td>Using <em>else</em> Statements</td>
<td>92</td>
</tr>
<tr>
<td>Using <em>else if</em> Statements</td>
<td>93</td>
</tr>
<tr>
<td>Understanding Comparison Conditions</td>
<td>95</td>
</tr>
<tr>
<td>Creating Compound Conditions</td>
<td>97</td>
</tr>
<tr>
<td>Negating a Condition Check</td>
<td>98</td>
</tr>
<tr>
<td>Expanding with the <em>switch</em> Statement</td>
<td>98</td>
</tr>
<tr>
<td>Summary</td>
<td>99</td>
</tr>
<tr>
<td>Workshop</td>
<td>100</td>
</tr>
<tr>
<td><strong>HOUR 7: Programming Loops</strong></td>
<td>103</td>
</tr>
<tr>
<td>Understanding Loops</td>
<td>103</td>
</tr>
<tr>
<td>Using <em>while</em> Loops</td>
<td>104</td>
</tr>
<tr>
<td>Using <em>do-while</em> Loops</td>
<td>106</td>
</tr>
<tr>
<td>Using <em>for</em> Loops</td>
<td>107</td>
</tr>
<tr>
<td>Using Arrays in Your Loops</td>
<td>109</td>
</tr>
<tr>
<td>Using Multiple Variables</td>
<td>112</td>
</tr>
<tr>
<td>Nesting Loops</td>
<td>112</td>
</tr>
<tr>
<td>Controlling Loops</td>
<td>113</td>
</tr>
<tr>
<td>Summary</td>
<td>116</td>
</tr>
<tr>
<td>Workshop</td>
<td>116</td>
</tr>
<tr>
<td><strong>HOUR 8: Working with Strings</strong></td>
<td>119</td>
</tr>
<tr>
<td>What's a String?</td>
<td>119</td>
</tr>
<tr>
<td>Understanding C-Style Strings</td>
<td>120</td>
</tr>
<tr>
<td>Introducing the Arduino <em>String</em> Object</td>
<td>126</td>
</tr>
<tr>
<td>Manipulating String Objects</td>
<td>130</td>
</tr>
<tr>
<td>Summary</td>
<td>131</td>
</tr>
<tr>
<td>Workshop</td>
<td>132</td>
</tr>
</tbody>
</table>
# HOUR 9: Implementing Data Structures

- **What's a Data Structure?**
- **Creating Data Structures**
- **Using Data Structures**
- **Manipulating Data Structures**
- **Arrays of Structures**
- **Working with Unions**
- **Summary**
- **Workshop**

---

# HOUR 10: Creating Functions

- **Basic Function Use**
- **Returning a Value**
- **Passing Values to Functions**
- **Handling Variables in Functions**
- **Calling Functions Recursively**
- **Summary**
- **Workshop**

---

# HOUR 11: Pointing to Data

- **What Is a Pointer?**
- **Working with Pointers**
- **Using Special Types of Pointers**
- **Pointer Arithmetic**
- **Strings and Pointers**
- **Combining Pointers and Structures**
- **Using Pointers with Functions**
- **Summary**
- **Workshop**

---

# HOUR 12: Storing Data

- **Arduino Memory Refresher**
- **Taking a Closer Look at SRAM**
- **Creating Dynamic Variables**
- **Using Flash to Store Data**
- **Using the EEPROM Memory**
- **Summary**
- **Workshop**
Contents

Writing a Network Program .......................................................... 349
Summary ......................................................................................... 351
Workshop ......................................................................................... 352

HOUR 22: Advanced Network Programming ................................... 355
The Web Protocol ............................................................................. 355
Reading Sensor Data from a Web Server ........................................ 361
Controlling an Arduino from the Web .............................................. 364
Summary ......................................................................................... 370
Workshop ......................................................................................... 370

HOUR 23: Handling Files ................................................................. 373
What Is an SD Card Reader? .......................................................... 373
SD Cards and the Arduino .............................................................. 375
The SD Library ................................................................................. 376
Interfacing with the SD Card .......................................................... 378
Storing Sensor Data ......................................................................... 382
Summary ......................................................................................... 385
Workshop ......................................................................................... 385

HOUR 24: Prototyping Projects ......................................................... 387
Determining Project Requirements .................................................. 387
Determining Interface Requirements .............................................. 389
Listing Components ......................................................................... 391
Creating a Schematic ........................................................................ 392
Creating the Breadboard Circuit ...................................................... 393
Designing the Sketch ........................................................................ 394
Writing the Sketch ........................................................................... 395
Testing the Sketch ............................................................................ 398
Creating a Prototype Board ............................................................ 399
Summary ......................................................................................... 401
Workshop ......................................................................................... 401

Index ............................................................................................... 403
About the Author

Richard Blum has worked in the IT industry for more than 25 years as a network and systems administrator, managing Microsoft, UNIX, Linux, and Novell servers for a network with more than 3,500 users. He has developed and teaches programming and Linux courses via the Internet to colleges and universities worldwide. Rich has a master’s degree in management information systems from Purdue University and is the author of several programming books, including Teach Yourself Python Programming for the Raspberry Pi in 24 Hours (coauthored with Christine Bresnahan, 2013, Sams Publishing), Linux Command Line and Shell Scripting Bible (coauthored with Christine Bresnahan, 2011, Wiley), Professional Linux Programming (coauthored with Jon Masters, 2007, Wiley), and Professional Assembly Language (2005, Wrox). When he’s not busy being a computer nerd, Rich enjoys spending time with his wife, Barbara, and two daughters, Katie Jane and Jessica.
Dedication

To my Uncle George.

Thanks for all your mentoring and troubleshooting help in my early electronics projects. I never would have gotten started in my career had those projects not worked!

“Iron sharpens iron, and one man sharpens another.” —Proverbs 27:17 (ESV)

Acknowledgments

First, all glory and praise go to God, who through His Son, Jesus Christ, makes all things possible and gives us the gift of eternal life.

Many thanks go to the fantastic team of people at Sams Publishing for their outstanding work on this project. Thanks to Rick Kughen, the executive editor, for offering us the opportunity to work on this book and keeping things on track, and to Andrew Beaster for all his production work. I would also like to thank Carole Jelen at Waterside Productions, Inc., for arranging this opportunity and for helping out in my writing career.

I am indebted to the technical editor, Jason Foster, who put in many long hours double-checking all the work and keeping the book technically accurate, all while getting a new job, having a new baby (congrats!), and moving to a new house in another state. His suggestions and eagle eyes have made this a much better book.

Finally I’d like to thank my wife, Barbara, and two daughters, Katie Jane and Jessica, for their patience and support while I was writing this.
We Want to Hear from You!

As the reader of this book, you are our most important critic and commentator. We value your opinion and want to know what we’re doing right, what we could do better, what areas you’d like to see us publish in, and any other words of wisdom you’re willing to pass our way.

We welcome your comments. You can email or write to let us know what you did or didn’t like about this book—as well as what we can do to make our books better.

Please note that we cannot help you with technical problems related to the topic of this book.

When you write, please be sure to include this book’s title and author as well as your name and email address. We will carefully review your comments and share them with the author and editors who worked on the book.

Email: consumer@samspublishing.com

Mail: Sams Publishing
      ATTN: Reader Feedback
      800 East 96th Street
      Indianapolis, IN 46240 USA

Reader Services

Visit our website and register this book at informit.com/register for convenient access to any updates, downloads, or errata that might be available for this book.
Introduction

Since being introduced in 2005 as a student project, the Arduino microcontroller has quickly become a favorite of both hobbyists and professionals. It’s a popular platform for creating many different types of automated systems—from monitoring water levels in house plants to controlling high-level robotic systems. These days you can find an Arduino behind lots of different electronic systems.

To control the Arduino, you need to know the Arduino programming language. The Arduino programming language derives from the C programming language, with some added features unique to the Arduino environment. However, beginners sometimes find the C programming somewhat tricky to navigate.

Programming the Arduino

The goal of this book is to help guide both hobbyists and students through using the Arduino programming language on an Arduino system. You don’t need any programming experience to benefit from this book; I walk through all the necessary steps to get your Arduino programs up and running.

▶ Part I, “The Arduino Programming Environment,” starts things out by walking through the core Arduino system and demonstrating the process of creating an Arduino program (called a sketch):

Hour 1, “Introduction to the Arduino,” shows the different Arduino models currently available and describes how each differs.

Hour 2, “Creating an Arduino Programming Environment,” shows how to load the Arduino IDE on a workstation and how to connect your Arduino to your workstation to get your sketches running on your Arduino.

Hour 3, “Using the Arduino IDE,” walks through all the features available to you in the IDE.
Introduction

Hour 4, “Creating an Arduino Program,” demonstrates the steps to build an Arduino circuit, design a sketch to control the circuit, and upload the sketch to the Arduino to run the circuit.

Part II, “The C Programming Language,” takes an in-depth look at the features of the C programming language that you need to know to write your Arduino sketches:

Hour 5, “Learning the Basics of C,” shows you how to use variables and math operators in C to manage data and implement formulas in your Arduino sketches.

Hour 6, “Structured Commands,” shows how to add logic to your sketches.

Hour 7, “Programming Loops,” demonstrates the different ways the Arduino language allows you to iterate through data, minimizing the amount of code you need to write.

Hour 8, “Working with Strings,” introduces the concept of storing and working with text values in your Arduino sketches.

Hour 9, “Implementing Data Structures,” walks through more complicated ways of handling data in sketches.

Hour 10, “Creating Functions,” provides useful tips to help minimize the amount of repeating code in your sketches.

Hour 11, “Pointing to Data,” introduces the complex topic of using pointers in the C language and shows how you can leverage their use in your sketches.

Hour 12, “Storing Data,” walks you through how to use the EEPROM storage available in the Arduino to store data between sketch runs.

Hour 13, “Using Libraries,” finishes the in-depth C language discussion by showing how to use prebuilt libraries in your sketches and how to create your own.

Part III, “Arduino Applications,” walks through the details for using your Arduino in different application environments:

Hour 14, “Working with Digital Interfaces,” shows how to read digital sensor values and use those values in your sketch and how to output digital values.

Hour 15, “Interfacing with Analog Devices,” shows how to read analog sensor values and how to use pulse width modulation to emulate an analog output voltage.

Hour 16, “Adding Interrupts,” demonstrates how to use asynchronous programming techniques in your Arduino sketches while monitoring sensors.

Hour 17, “Communicating with Devices,” covers the different communications protocols built in to the Arduino, including SPI and I2C.
Hour 18, “Using Sensors,” takes a closer look at the different types of analog and digital sensors the Arduino supports and how to handle them in your sketches.

Hour 19, “Working with Motors,” walks through how to control different types of motors from your Arduino sketch.

Hour 20, “Using an LCD,” provides instructions on how to utilize digital displays to output data from your sketch.

Hour 21, “Working with the Ethernet Shield,” discusses how to connect your Arduino to a network.

Hour 22, “Implementing Advanced Ethernet Programs,” demonstrates how to provide sensor data to remote network clients and how to control the Arduino from a remote client.

Hour 23, “Handling Files,” shows how to use SD card interfaces found on some Arduino shields to store data for long term.

Hour 24, “Prototyping Projects,” walks you through the process of creating a complete Arduino project, from design to implementation.

Who Should Read This Book?

This book is aimed at readers interested in getting the most out of their Arduino system by writing their own Arduino sketches, including these three groups:

▶ Students interested in an inexpensive way to learn electronics and programming
▶ Hobbyists interested in monitoring and controlling digital or analog circuits
▶ Professionals looking for an inexpensive platform to use for application deployment

If you are reading this book, you are not necessarily new to programming, but you may be new to the Arduino environment and need a quick reference guide.

Conventions Used in This Book

To make your life easier, this book includes various features and conventions that help you get the most out of this book and out of your Arduino:


Things you type—Whenever I suggest that you type something, what you type appears in a bold font.
Filenames, folder names, and code—These things appear in a monospace font.

Commands—Commands and their syntax use **bold**.

Menu commands—I use the following style for all application menu commands: **Menu, Command**, where **Menu** is the name of the menu you pull down and **Command** is the name of the command you select. Here’s an example: File, Open. This means you select the File menu and then select the Open command.

This book also uses the following boxes to draw your attention to important or interesting information:

---

**BY THE WAY**

By the Way boxes present asides that give you more information about the current topic. These tidbits provide extra insights that offer better understanding of the task.

---

**DID YOU KNOW?**

Did You Know? boxes call your attention to suggestions, solutions, or shortcuts that are often hidden, undocumented, or just extra useful.

---

**WATCH OUT!**

Watch Out! boxes provide cautions or warnings about actions or mistakes that bring about data loss or other serious consequences.
This page intentionally left blank
Now that you’ve seen what the Arduino is and how to program it using the Arduino IDE, it’s time to write your first program and watch it work. In this hour, you learn how to use the Arduino IDE software package to create, compile, and upload an Arduino program. You then learn how to interface your Arduino with external electronic circuits to complete your Arduino projects.

**Building an Arduino Sketch**

Once you have your Arduino development environment set up, you’re ready to start working on projects. This section covers the basics that you need to know to start writing your sketches and getting them to run on your Arduino.

**Examining the Arduino Program Components**

When you use the Arduino IDE package, your sketches must follow a specific coding format. This coding format differs a bit from what you see in a standard C language program.

In a standard C language program, there’s always a function named `main` that defines the code that starts the program. When the CPU starts to run the program, it begins with the code in the `main` function.

In contrast, Arduino sketches don’t have a `main` function in the code. The Arduino bootloader program that’s preloaded onto the Arduino functions as the sketch’s `main` function. The Arduino starts the bootloader, and the bootloader program starts to run the code in your sketch.
The bootloader program specifically looks for two separate functions in the sketch:

- `setup`
- `loop`

The Arduino bootloader calls the `setup` function as the first thing when the Arduino unit powers up. The code you place in the `setup` function in your sketch only runs one time; then the bootloader moves on to the `loop` function code.

The `setup` function definition uses the standard C language format for defining functions:
```
void setup() {
  code lines
}
```

Just place the code you need to run at startup time inside the `setup` function code block.

After the bootloader calls the `setup` function, it calls the `loop` function repeatedly, until you power down the Arduino unit. The `loop` function uses the same format as the `setup` function:
```
void loop() {
  code lines
}
```

The meat of your application code will be in the `loop` function section. This is where you place code to read sensors and send output signals to the outputs based on events detected by the sensors. The `setup` function is a great place to initialize input and output pins so that they’re ready when the `loop` runs, then the `loop` function is where you use them.

**Including Libraries**

Depending on how advanced your Arduino program is, you may or may not need to use other functions found in external library files. If you do need to use external libraries, you first need to define them at the start of your Arduino program, using the `#include` directive:
```
#include <library>
```

The `#include` directives will be the first lines in your sketch, before any other code.

If you’re using a standard Arduino shield, most likely the shield library code is already included in the Arduino IDE package. Just choose Sketch > Import Library from the menu bar, and then select the shield that you’re using. The Arduino IDE automatically adds the `#include` directives required to write code for the requested shield. For example, if you select the Ethernet shield, the following lines are imported into the sketch:
That saves a lot of time from having to go hunting around to find the libraries required for a specific shield.

Creating Your First Sketch

Now that you've seen the basics for creating an Arduino program, let's dive in and create a simple sketch to get a feel for how things work.

Working with the Editor

When you open the Arduino IDE, the editor window starts a new sketch. The name of the new sketch appears in the tab at the top of the editor window area, in the following format:

`sketch_mmddx`

where `mm` is a three-letter abbreviation of the month, `dd` is the two-digit numerical date, and `x` is a letter to make the sketch name unique for the day (for example, sketch_jan01a).

As you type your sketch code into the editor window, the editor will color-code different parts of the sketch code, such as making function names brown and text strings blue. This makes it easier to pick out syntax errors, and comes in handy when you're trying to debug your sketch.

Now you're ready to start coding. Listing 4.1 shows the code for the sketch0401 file that we'll use to test things out. Enter this code into the Arduino IDE editor window.

**LISTING 4.1 The sketch0401 Code**

```c
int counter = 0;
int pin = 13;

void setup() {
    Serial.begin(9600);
    pinMode(pin, OUTPUT);
    digitalWrite(pin, LOW);
}

void loop() {
    counter = counter + 1;
```
digitalWrite(pin, HIGH);
Serial.print("Blink "); Serial.println(counter);
delay(1000);
digitalWrite(pin, LOW);
delay(1000);
}

You’ll learn what all these different lines of code mean as you go through the rest of the hours, so don’t worry too much about the code for now. The main point now is to have a sketch to practice compiling and running.

The basic idea for this code is to make the Arduino blink the L LED connected to digital port 13 on the Arduino once per second, and also output a message to the Arduino serial port, counting each blink.

After you enter the code into the editor window, choose File > Save As from the menu bar to save the sketch as sketch0401. Now you’re ready to verify and compile the sketch.

**Compiling the Sketch**

The next step in the process is to compile the sketch code into the machine language code that the Arduino runs.

Click the verify icon on the toolbar (the checkmark icon), or choose Sketch > Verify/Compile from the menu bar. Figure 4.1 shows the results that you should get if things worked correctly.

As shown in Figure 4.1, you should see a message in the message area that the compile has completed, and the console window should show the final size of the compiled machine language code that will be uploaded to the Arduino.

If you have any typos in the sketch code that cause the compile process to fail, you’ll see an error message in the message area, as shown in Figure 4.2.

The Arduino IDE also highlights the line of code that generated the error, making it easier for you to pick out the problem. Also, a more detailed error message appears in the console window area to help even more.

After you get the sketch to compile without any errors, the next step is to upload it to your Arduino.
FIGURE 4.1
Compiling the sketch0401 code.

FIGURE 4.2
A compiler error displayed in the Arduino IDE.
Uploading Your Sketch

The key to successfully uploading sketches to your Arduino unit is in defining how the Arduino is connected to your workstation. Hour 3, “Using the Arduino IDE,” walked through how to use the Tools > Serial Port menu bar option to set which serial port your Arduino is connected to. After you set that, you should be able to easily upload your compiled sketches.

Just click either the upload icon on the toolbar (the right arrow icon), or select File > Upload from the menu bar. Before the upload starts, the Arduino IDE recompiles the sketch code. This comes in handy when you’re just making quick changes; you can compile and upload the new code with just one click.

When the upload starts, you should see the TX and RX LEDs on the Arduino blink, indicating that the data transfer is in progress. When the upload completes, you should see a message in both the Arduino IDE message area and console window indicating that the upload was completed. If anything does go wrong, you’ll see an error message appear in both the message area and the console window, as shown in Figure 4.3.

![Figure 4.3](image-url)

Upload problem message in the Arduino IDE.
Creating Your First Sketch

If all goes well, you’re ready to start running your sketch on the Arduino. The next section shows you how.

Running Your Program

Now that the sketch code is uploaded onto your Arduino, you’re ready to start running it. However, you might have noticed that once the upload process finished in the Arduino IDE, the L and the TX LEDs on your Arduino unit already started to blink. That’s your sketch running. When the upload process completes, the bootloader automatically reboots the Arduino and runs your program code.

The L LED is blinking because of the `digitalWrite()` function setting the digital pin 13 first to 0 (no voltage) and then after a second, setting it to 1 (producing a 5V signal). The TX LED is blinking because the `Serial.print()` function is sending data out the serial port.

You can view the output from the serial port on your Arduino using the serial monitor built in to the Arduino IDE. Just choose Tools > Serial Monitor from the menu bar, or click the serial monitor icon (the magnifying glass icon) on the toolbar. The serial monitor window appears and displays the output received from the Arduino, as shown in Figure 4.4.

![Serial Monitor](image)

**FIGURE 4.4**
Viewing the Arduino serial port output from the serial monitor.
You might have noticed that after you started the serial monitor, the blink count output restarted back at 1. When you start serial monitor, it sends a signal to the Arduino to reset it, which in turn runs the bootloader to reload the sketch and start over from the beginning.

You can also manually restart a running sketch using the Reset button on the Arduino. On the Arduino Uno R3, you’ll find the Reset button in the upper-left corner of the circuit board. Just push the button and release it to reset the Arduino.

You don’t have to connect the Arduino to the USB port on your workstation for it to run. You can run the Arduino from an external power source, as well, such as a battery pack or AC/DC converter. Just plug the power source into the power socket on the Arduino unit. The Arduino Uno R3 automatically detects power applied to either the USB port or the power port and starts the bootloader program to start your sketch.

### Interfacing with Electronic Circuits

While getting your sketch uploaded to the Arduino and running is a significant accomplishment, most likely you’ll want to do more in your Arduino projects than just watch the LED blink. That’s where you’ll need to incorporate some type of external electronic circuits into your projects. This section covers the basics of what you need to know to add external electronic circuits to your Arduino sketches.

#### Using the Header Sockets

The main use of the Arduino is to control external electronic circuits using the input and output signals. To do that, you need to interface your electronic circuits with the Arduino analog and digital signals. This is where the header sockets come into play.

If you remember from Hour 1, “Introduction to the Arduino,” the header sockets are the two rows of sockets at the top and bottom of the Arduino Uno circuit board. (Some more advanced Arduino units, such as the Arduino Mega, also include a third header socket on the right side of the board to support additional ports.) You’ll plug your electronic circuits into the sockets to gain access to the Arduino input and output signals, as well as the power from the Arduino.

The basic Arduino Uno unit that we’re using for our experiments uses the standard Arduino two-row header socket format. Figure 4.5 shows the layout of the upper and lower header sockets.
FIGURE 4.5
The Arduino Uno upper and lower header sockets.

The lower header socket has 13 ports on it, as described in Table 4.1.

<table>
<thead>
<tr>
<th>Label</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IOREF</td>
<td>Provides the reference voltage used by the microcontroller if not 5V.</td>
</tr>
<tr>
<td>RESET</td>
<td>Resets the Arduino when set to LOW.</td>
</tr>
<tr>
<td>3.3V</td>
<td>Provides a reduced 3.3V for powering low-voltage external circuits.</td>
</tr>
<tr>
<td>5V</td>
<td>Provides the standard 5V for powering external circuits.</td>
</tr>
<tr>
<td>GND</td>
<td>Provides the ground connection for external circuits.</td>
</tr>
<tr>
<td>GND</td>
<td>A second ground connection for external circuits.</td>
</tr>
<tr>
<td>Vin</td>
<td>An external circuit can supply 5V to this pin to power the Arduino, instead of using the USB or power jacks.</td>
</tr>
<tr>
<td>A0</td>
<td>The first analog input interface.</td>
</tr>
<tr>
<td>A1</td>
<td>The second analog input interface.</td>
</tr>
</tbody>
</table>
### HOUR 4: Creating an Arduino Program

<table>
<thead>
<tr>
<th>Label</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A2</td>
<td>The third analog input interface.</td>
</tr>
<tr>
<td>A3</td>
<td>The fourth analog input interface.</td>
</tr>
<tr>
<td>A4</td>
<td>The fifth analog input interface, also used as the SDA pin for TWI communications.</td>
</tr>
<tr>
<td>A5</td>
<td>The sixth analog input interface, also used as the SCL pin for TWI communications.</td>
</tr>
</tbody>
</table>

The upper header socket has 16 ports on it, as described in Table 4.2

**TABLE 4.2 The Arduino Uno Upper Header Socket Ports**

<table>
<thead>
<tr>
<th>Label</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AREF</td>
<td>Alternative reference voltage used by the analog inputs (by default, 5V).</td>
</tr>
<tr>
<td>GND</td>
<td>The Arduino ground signal.</td>
</tr>
<tr>
<td>13</td>
<td>Digital port 13, and the SCK pin for SPI communication.</td>
</tr>
<tr>
<td>12</td>
<td>Digital port 12, and the MISO pin for SPI communication.</td>
</tr>
<tr>
<td>-11</td>
<td>Digital port 11, a PWM output port, and the MOSI pin for SPI communications.</td>
</tr>
<tr>
<td>-10</td>
<td>Digital port 10, a PWM output port, and the SS pin for SPI communication.</td>
</tr>
<tr>
<td>-9</td>
<td>Digital port 9, and a PWM output port.</td>
</tr>
<tr>
<td>8</td>
<td>Digital port 8.</td>
</tr>
<tr>
<td>7</td>
<td>Digital port 7.</td>
</tr>
<tr>
<td>-6</td>
<td>Digital port 6, and a PWM output port.</td>
</tr>
<tr>
<td>-5</td>
<td>Digital port 5, and a PWM output port.</td>
</tr>
<tr>
<td>4</td>
<td>Digital port 4.</td>
</tr>
<tr>
<td>-3</td>
<td>Digital port 3, and a PWM output port.</td>
</tr>
<tr>
<td>2</td>
<td>Digital port 2.</td>
</tr>
<tr>
<td>TX -&gt; 1</td>
<td>Digital port 1, and the serial interface transmit port.</td>
</tr>
<tr>
<td>RX &lt;- 0</td>
<td>Digital port 0, and the serial interface receive port.</td>
</tr>
</tbody>
</table>

For our test sketch, we need to access the digital port 13 socket, in addition to a GND socket, to complete the electrical connection to power our electronic devices.

To access the sockets, you can plug wires directly into the socket ports. To make it easier, you can use jumper wires, which you can easily remove when you finish experimenting.
Building with Breadboards

When you build an electronic circuit, the layout is usually based on a schematic diagram that shows how the components should be connected. The schematic shows a visual representation of which components are connected to which, using standard symbols to represent the different components, such as resistors, capacitors, transistors, switches, relays, sensors, and motors.

Your job is to build the electronic circuit to mimic the layout and connections shown in the schematic diagram. In a permanent electronic circuit, you use a printed circuit board (called PCB) to connect the components according to the schematic.

In a PCB, connections between the electronic components are etched into the PCB using a metallic conductor. To place the electronic components onto the PCB, you must solder the leads of the components onto the PCB.

The downside to using a PCB for your electronic project is that because it’s intended to be permanent, you can’t easily make changes. Although that’s fine for final circuits, when you’re developing a new system and experimenting with different circuit layouts, it’s somewhat impractical to build a new PCB layout for each test.

This is where breadboards come in handy. A breadboard provides an electronic playground for you to connect and reconnect electronic components as you need. Figure 4.6 shows a basic breadboard layout.

![Figure 4.6](image)

A basic breadboard.
Breadboards come in many different sizes and layouts, but most breadboards have these features:

- A long series of sockets interconnected along the ends of the breadboard. These are called buses (or sometimes rails), and are often used for the power and ground voltages. The sockets in the bus are all interconnected to provide easy access to power in the circuit.

- A short series of sockets (often around five) interconnected, and positioned across a gap in the center of the breadboard. Each group of sockets is interconnected to provide an electrical connection to the components plugged into the same socket group. The gap allows you to plug integrated circuit chips into the breadboard and have access to the chip leads.

The breadboard allows you to connect and reconnect your circuits as many times as you need to experiment with your projects. Once you get your circuit working the way you want, you can transfer the breadboard layout onto a PCB for a more permanent solution.

**Adding a Circuit to Your Project**

Now that you’ve seen how to add external electronic circuits to your Arduino project, let’s create a simple circuit to add to our Arduino sketch. Instead of using the \( \star \) LED on your Arduino, let’s use an external LED.

For this project, you need the following parts:

- A standard breadboard (any size)
- A standard LED (any color)
- A 1000ohm resistor (color code brown, black, red)
- Jumper wires to connect the breadboard circuit to the Arduino

The circuit uses a 1000ohm resistor to limit the voltage that flows through the LED to help protect the LED. The LED doesn’t need the full 5V provided by the Arduino output, so by placing a resistor in series with the LED, the resistor helps absorb some of the voltage, leaving less for the LED. If you don’t have a 1000ohm resistor handy, you can use any other resistor value to help lessen the voltage applied to the LED.

Figure 4.7 shows connecting the resistor and LED to the GND and digital pin 13 ports on your Arduino Uno unit.

Just follow these steps to create your electronic circuit for the project.
TRY IT YOURSELF ▼

Creating the Electronic Circuit

1. Connect a jumper wire from one of the GND socket ports on the Arduino to a socket row on the breadboard.

2. Connect a jumper wire from the digital pin 13 socket port on the Arduino to another socket row on the breadboard (not the same as the one you used for the GND signal).

3. Plug the LED into the breadboard so that the longer lead of the LED is connected to the same socket row as the digital pin 13 wire and so that the other lead is plugged into a separate socket row on the breadboard.

CAUTION

Polarity in Electronic Circuits

While plugging the LED in the wrong way won’t harm the LED, there are other electronic components that can cause damage if plugged in the wrong way (such as transistors). Be careful when working with electronic components that have polarity requirements!
Now you should be ready to test things out. Power up the Arduino, either by connecting it to
the USB port of your workstation or by connecting it to an external power source. Because the
Arduino maintains the sketch in flash memory, you don’t need to reload your sketch; it should
start running automatically.

**CAUTION**

**Providing Power to the Arduino**

Be careful when plugging and unplugging your Arduino if you’re using a USB hub with other devices.
Stray voltages can result that may damage the other USB devices on the hub. It’s always a good
idea to power down your USB hub when plugging and unplugging the Arduino.

If things are working, you should see the LED on the breadboard blink once per second. If not,
double-check your wiring to ensure that you have everything plugged together correctly on the
breadboard and that the wires are plugged into the proper socket ports on the Arduino.

**TIP**

**Using the Serial Monitor**

If you connected the Arduino to the USB port on your workstation, you can still use the serial moni-
tor in the Arduino IDE to view the output from the sketch. However, if you use an external power
source to power the Arduino, you won’t be able to view that output unless you connect an external
serial device to the Arduino serial ports, which are digital ports 0 and 1 in the header sockets.

**Summary**

This hour walked you through your first Arduino project. First, we entered the sketch code into
the Arduino IDE editor window, then we compiled the sketch, and finally, we uploaded the com-
piled sketch to the Arduino. You also saw how to use the serial monitor feature in the Arduino
IDE to monitor output from your sketch. After that, you learned how to set up an external elec-
tronic circuit and interface it with your Arduino.

In the next hour, we take a closer look at the actual Arduino sketch code that we’ll be using in
our projects. You’ll learn how the Arduino programming language stores and manipulates data
within our sketches.
Workshop

Quiz

1. Which function must your Arduino sketch define to run the main part of your program code?
   A. setup
   B. loop
   C. main
   D. start

2. The Arduino IDE editor uses the same text color code to indicate functions as it does regular text in the code. True or false?

3. How do you interface external electronic circuits to your Arduino?

Answers

1. B. The loop function contains the sketch code that continually runs while the Arduino unit is powered on. This is where you need to place your main sketch code.

2. False. The Arduino IDE uses brown to indicate functions used in the sketch code, and uses blue to indicate text strings contained in the sketch code.

3. The Arduino header sockets are designed to easily interface external electronic circuits with the analog and digital input and output pins on the microcontroller.

Q&A

Q. Is there a limit to the size of the sketches I can upload to my Arduino?

A. Yes, the size of the sketch is limited by the amount of flash memory present on your Arduino. The Arduino Uno R3 has 32KB of flash memory. When you compile your sketch, the Arduino IDE console window will display the size of the compiled sketch code and how much space is remaining in the flash memory.

Q. Can I damage my Arduino by plugging in the wrong wires to the wrong header socket ports?

A. Yes, it’s possible, but the Arduino does contain some basic protections. The Arduino is designed with some basic voltage protection on each of the input and output ports. If you supply too large of voltages to the ports, however, you can risk burning out the microcontroller chip. Use caution when connecting wires to the Arduino header sockets, and always double-check your work before turning on the power.

Q. Is there an easy way to identify resistor values when working with electronic circuits?

A. Yes, all resistor manufacturers use a standard resistor color code. The resistor value and tolerance are indicated by color bands around the resistor. To find the value of a resistor, refer to a resistor color-code chart, as shown in the Wikipedia article on electronic color codes (http://en.wikipedia.org/wiki/Electronic_color_code).
This page intentionally left blank
## Symbols

- `&` (ampersand) reference operator, 164, 170
- `*` (asterisk) dereference operator, 164
  - retrieving data, 166-167
  - storing data, 167
- `->` operator, retrieving data structure values with pointers, 174-176
- `;` (semicolon), terminating statements, 77

## Numbers

- `-3` header socket port, 66
- `-5` header socket port, 66
- `-6` header socket port, 66
- `-9` header socket port, 66
- `-10` header socket port, 66
- `-11` header socket port, 66
- `2` header socket port, 66
- `3.3V` header socket port, 65
- `4` header socket port, 66
- `5V` header socket port, 65
- `7` header socket port, 66
- `8` header socket port, 66
- `12` header socket port, 66
- `13` header socket port, 66
- `A0` header socket port, 65
- `A1` header socket port, 65
- `A2` header socket port, 65
- `A3` header socket port, 65
- `A4` header socket port, 65
- `A5` header socket port, 65
- `abs()` function, 85
- `accessing header sockets`, 66
- `accessories`, 17-18
- `activating serial monitor`, 53
- `Adafruit Data Logging shield`, 376
- `ADC (analog-to-digital converter)`, 25, 236
- `Add File option (Sketch menu)`, 46
- `addition operator`, 80
- `advanced math functions`, 85
- `alphanumeric LCD devices`, 319-320
- `ALU (Arithmetic Logic Unit)`, 24
- `analogRead()` function, 238, 291
- `analogReference()` function, 245-246, 291
- `analog-to-digital converter (ADC)`, 25, 236
- `analogWrite()` function, 246, 307
- `architecture of Arduino`, 9-11
- `Archive Sketch option (Tools menu)`, 47
Arduino

accessories, 17-18
analog interfaces. See analog interfaces
architecture, 9-11
communication between units, 280-284
controlling from web browser, 364-370
defined, 7
digital interfaces. See digital interfaces
history of, 11-12
interrupts
  external interrupts, 252-253, 254-260
  ignoring, 264-265
  pin change interrupts, 253-254, 260-262
  polling versus, 251-252
  timer interrupts, 254, 262-264
memory
  comparison among types, 181-182
  creating dynamic variables, 185-189
  EEPROM memory, 194-197
  flash memory, 189-193
  SRAM memory, 183-185
  as microcontroller, 7-8
  models. See models of Arduino
  powering on/off with USB hub, 69
  shields. See shields
  trademark protection, 9
Arduino IDE, 31-32
  console window, 49-50
  downloading, 32-33
  Edit menu commands, 44-46
  editor window, 59-60
  File menu commands, 40-43
  Help menu commands, 48
  installing
    for Linux, 37
    for OS X, 36-37
    for Windows, 33-36
Arduino programming language. See also C programming language
  functions, 83-86
    advanced math functions, 85
    bit manipulation functions, 86
    calling, 148-150
    defining, 148
    global variables, 155-156
    local variables, 156-158
    passing values to, 152-154
    random number generators, 86
    recursive functions, 158-160
    returning values, 150-152
    scope of variables, 154
    Serial class, 83-84
    time functions, 84-85
  troubleshooting, 148
    user-defined, 147
  strings, 126-129
    creating and manipulating, 126-128
    manipulating, 130-131
    String object methods, 128-129
Arduino Starter Kit, 18
  AREF header socket port, 66
  arguments, 152
Arithmetic Logic Unit (ALU), 24
arrays
  creating, 110-111
  of data structures, 140-142
  defined, 120
  loops, 109-112
  pointer arithmetic, 168-171
  sizing, 111-112, 121-122
  strings, 120-126
  comparing, 125-126
  creating, 121-122
  functions for, 122-125
ASCII format, 119, 271
assembly language, 27-28
assigning values
  to data structures, 136-138
  to variables, 77
assignment statements
  equality comparison
    versus, 96
  equations versus, 82
asterisk (*) dereference operator, 164
retrieving data, 166-167
storing data, 167
ATmega AVR microcontrollers, 9-10
  components of, 23-26
  CPU, 24-25
  EEPROM memory, 194-197
  flash memory, 189-193
  I/O interface, 25-26
  memory, 25
  memory comparisons, 181-182
  SRAM memory, 183-185
  instruction set, downloading, 27
Atmel C library, 29-30
Atmel Studio package, 30
attach() function, 313
attached() function, 313
attachInterrupt() function, 254-255
Auto Format option (Tools menu), 47
autoscroll() function, 325
Autoscroll option (serial monitor), 53
available() function
  Serial library, 270
  Wire library, 278
available method
  EthernetClient class, 343
  EthernetServer class, 345
  EthernetUDP class, 347
  File class, 377
AVR Libc project, 29
avr-gcc package, 30

bit() function, 86
bit manipulation functions, 86
bitClear() function, 86
bitRead() function, 86
bitSet() function, 86
bitwise AND operator, 80
bitwise OR operator, 80
bitWrite() function, 86
blink() function, 325
blinking LED example sketch, 272-274, 280-284
Board option (Tools menu), 47
Boolean comparisons, 96-97
boolean data type, 77
Boolean logic, 81
bootloader, 42
  functions in, 57-58
  uploading, 48
breadboards, 17
  creating circuits, 393-394
  electronic circuit interfaces, 67-68
break statements, 99, 113-114
breakpoints, controlling Arduino from, 364-370
brushes in DC motors, 304
build method
  Ethernet class, 340
  EthernetServer class, 345
  EthernetUDP class, 347
  SD class, 376
build method
  Burn Bootloader option (Tools menu), 48
byte data type, 77

C programming language, 28-29.
  See also Arduino programming language
Arduino IDE, 31-32. See also Arduino IDE
Atmel C library, 29-30
data structures, 133-134
  arrays of, 140-142
  assigning values, 136-138
  copying, 138-140
  creating, 134-136
  initializing, 176
loops, 103-104
  arrays, 109-112
  break statements, 113-114
  continue statements, 114-116
  do-while statements, 106-107
  endless loops, 106
  multiple variables in, 112
  nesting, 112-113
  for statements, 107-109
  while statements, 104-106
operators
  compound operators, 82
  math operators, 80-82
  order of operations, 82
  pointers, 163-166
  arithmetic with arrays, 168-171
  data structures and, 173-176
  null pointers, 167-168
  passing to functions, 176-178
  printing, 166
  referencing strings, 172-173
  retrieving data, 166-167
  storing data, 167
  string manipulation, 171-172
  void pointers, 168
  shield libraries, 32
  statements, terminating, 77
  strings, 120-126
  comparing, 125-126
  creating, 121-122
  functions for, 122-125

How can we make this index more useful? Email us at indexes@samspublishing.com
structured commands
comparisons, 95-97
compound conditions, 97
else if statements, 93-95
else statements, 92-93
grouping multiple statements, 90-92
if statements, 89-90
negating condition checks, 98
switch statements, 98-99
unions, 142-145
variables
assigning values, 77
data types, 77-78
declaring, 76-77
dynamic variables, 185-189
qualifiers, 79
scope, 80
C++ programming language,
library creation
code files, creating, 208-210
header files, creating, 210-211
calculating factorials, 158-160
calling functions, 148-150,
158-160. See also referencing
calloc() function, 186-187
camel case, 77
CapacitiveSensor library, 298-300
capacitors, 297-298
capacity classes (SD cards), 374
case statements, 99
CHANGE external interrupt mode, 255
changing
dynamic variables, 187
reference voltages, 245-246, 290-291
char data type, 77, 79, 119-120
capability functions, 187
character arrays. See arrays, strings
charAt method, 128
chat server example sketch, 349-351
circuits. See electronic circuits
clear() function, 325
Client class. See EthernetClient class
clock speed, 25, 277
close method, 377
Close option (File menu), 42
code files in libraries, 202, 208-210
code libraries, 29
code listings. See listings
coding. See programming
microcontrollers
color types (LCDs), 320-321
comma-separated values (CSV) format, 380
Comment/Uncomment option (Edit menu), 45
communication
LCD (liquid crystal display) devices, 319
Arduino interface connections, 323-325
color types, 320-321
display types, 319-320
downloading and installing LCD shield library, 330-331
interface pins, 321-323
LC display, 329-330
LCD shield connections, 332-333
LCD shield library functions, 331-332
LiquidCrystal library, 325-327
temperature display
temperature measurement sketch, 327-329, 333-335
troubleshooting, 329
serial communication
protocols, 267-268
I²C (Inter-Integrated Circuit) protocol, 277-284
serial ports, 268-274
SPI (Serial Peripheral Interface) protocol, 274-277
compareTo method, 128
comparisons, 95-97
Boolean comparisons, 96-97
compound conditions, 97
negating condition checks, 98
numeric comparisons, 95-96
string comparisons, 96, 125-126
compilers, 28
compiling
functions in standard libraries, 205
sketches, 60-61
compound conditions, 97
compound operators, 82
concat method, 128
configuring Arduino IDE, 51-52
connect method, 343
CONNECT method token, 357
connected method, 343
connections. See also interfaces
with Ethernet shield, 18-19
with LCD devices, 323-325
with LCD shield, 332-333
console window (Arduino IDE), 49-50
const variable qualifier, 79
constants, 79
in flash memory, 190-191
memory locations, 184
constrain() function, 85, 241
continue statements, 114-116
contrast on LCD devices, 329
controlled libraries, 206-208.
See also building libraries
collectors, 24
copying
data structures, 138-140
strings, 125, 171-172
cos() function, 85
.cpp file extension, 202
Digital interface components of CPU, 24-25
programming assembly language, 27-28
C programming language, 28-29. See also C programming language machine code, 26
createChar() function, 325
CSV (comma-separated values) format, 380
current sinks, digital interfaces as, 222-223
current sources, digital interfaces as, 222-223
cursor() function, 325
Cut option (Edit menu), 44
declaration, 190-191
local variables, 156-158
variables, 76-77
declaration (Edit menu), 45
decrement operator, 80
default statements, 99
#define directive, 210
defining. See also declaring
  dynamic variables, 186-187
  functions, 148
  global variables, 155-156
delay() function, 84
delayMicroseconds() function, 84
  DELETE method token, 357
  dereference operators, 164
  retrieving data, 166-167
  storing data, 167
detach() function, 313
detachInterrupt() function, 255
DHCP (Dynamic Host Configuration Protocol), 342-343
digital interfaces
  input voltage levels, 226-229
  interface 13, 229
  interrupts
    external interrupts, 252-260
    ignoring, 264-265
    pin change interrupts, 253-254, 260-262
    polling versus, 251-252
    timer interrupts, 254, 262-264
  layouts, 220-221
  LCD (liquid crystal display) devices, 323-325
data display with LCD shield, 19
  number of, 219-220
  output voltage levels, 221-223
  planning, 390-391
  setting input/output modes, 221
  SPI signals, 276
  traffic signal example sketch, 223-226, 229-231, 364-370
  troubleshooting
    input voltage levels, 227
    with serial monitor, 226
digitalRead() function, 226
digital-to-analog converter (DAC), 236-237
digitalWrite() function, 63, 221
direction of DC motors, controlling, 307-308
display() function, 325
display types (LCDs), 319-320
displaying
  data with LCD shield, 19
  strings, 122
division operator, 80
documentation for standard libraries, 205
double data type, 77
do-while statements, 106-107
downloading
  Arduino IDE, 32-33
  ATmega AVR microcontroller instruction set, 27
  contributed libraries, 206-208
  LCD shield library, 330-331
  Timer One library, 263
  drivers, installing, 34-35
Due model, 13
  analog interfaces, 236
digital interfaces, 219
I2C interface pins, 278
serial port interfaces, 269
voltage levels, 288
durability of SD cards, 375
Dynamic Host Configuration Protocol (DHCP), 342-343
dynamic IP addresses, 342-343
dynamic variables, 184-189
  changing, 187
  defining, 186-187
  example usage, 187-189
  removing, 187
Eagle circuit board software, 400-401
Edit menu commands, 44-46
editor window (Arduino IDE), creating sketches, 59-60

How can we make this index more useful? Email us at indexes@samspublishing.com
floating-point values, integer
values versus, 78
flow control. See loops; structured
commands
flush() function, 270
flush method
 EthernetClient class, 343
 File class, 377
 flushing SD card data, 378
 folders on SD cards, 381-382
 for statements, 107-109, 112
 formatting sketches, 91
 free() function, 187
 functions
 in Arduino, 83-86
 advanced math
 functions, 85
 bit manipulation
 functions, 86
 calling, 148-150
 defining, 148
 global variables, 155-156
 local variables, 156-158
 passing values to, 152-154
 random number
 generators, 86
 recursive functions, 158-160
 returning values, 150-152
 scope of variables, 154
 Serial class, 83-84
 time functions, 84-85
 troubleshooting, 148
 user-defined, 147
 in bootloader, 57-58
 compiling in standard
 libraries, 205
 in EEPROM memory, 194
 for flash memory access, 191
 LCD shield library, 331-332
 LiquidCrystal library, 325-326
 passing pointers to, 176-178
 private functions, 211
 public functions, 211
 referencing in standard
 libraries, 204-205
 Serial library, 269-272
 Servo library, 313
 SPI library, 276-277
 for strings, 122-125
 testing results, 97
 Wire library, 278-280

G

gate leads in transistors, 305
GET method token, 357
getBytes method, 128
global variables, 80
 defining, 155-156
 memory locations, 184
 overriding, 158
GND header socket ports, 65=66
grounding analog sensors, 290
 group multiple statements
 in else statements, 92
 in if statements, 90-92
GSM library, 203

H

.h file extension, 202
hardware
 external interrupts, 252-253
 open source hardware, 9
 H-bridges, 307-308
 HD44780 controller chips, 321-322
 HEAD method token, 357
current headers in libraries, 202,
 210-211
 header sockets, 10-11
 accessing, 66
 electronic circuit interfaces, 64-66
 on Uno R3 unit, 15-16
 headers (HTTP)
 request headers, 358
 response header lines, 360-361
heap data area, 183-185
 dynamic variables, 185-189
 changing, 187
 defining, 186-187
 example usage, 187-189
 removing, 187
Help menu commands, 48
highByte() function, 86
high-current devices, digital
 interface connections, 221-223
higher-level programming
 languages, 28-29
history of Arduino, 11-12
home() function, 325
HTML in sketches, 44
HTTP (Hypertext Transfer
 Protocol), 355
 requests, 356-358
 request headers, 358
 request line, 357
 responses, 358-361
 response header lines, 360-361
 status line, 358-360
 sessions, 355-356

I

I^C (Inter-Integrated Circuit)
 protocol, 277-284
 blinking LED example sketch, 280-284
 interfaces, 278
 Wire library functions, 278-280
ICSP (in-circuit serial
 programming) header, 390
IDE (integrated development
 environment)
 Arduino IDE, 31-32
 console window, 49-50
 downloading, 32-33
 Edit menu commands,
 44-46
 editor window, 59-60

How can we make this index more useful? Email us at indexes@samspublishing.com
FILE  menu commands, 40-43
Help menu commands, 48
interface, 39-40
Linux installation, 37
message area, 49-50
OS X installation, 36-37
serial monitor, 52-54
setup, 51-52
shield libraries, 32
Sketch menu commands, 46
toolbar, 49
Tools menu commands, 46-48
Windows installation, 33-36
Atmel Studio package, 30
if statements, 89-90
compound conditions, 97
grouping multiple statements, 90-92
negating condition checks, 98
#ifndef directive, 210
ignoring interrupts, 264-265
Import Library option (Sketch menu), 46
importing. See also installing contributed libraries, 206-208
PinChangeInt library, 260-261
Timer One library, 263
in-circuit serial programming (ICSP) header, 390
#include directive, 58-59, 204
Ethernet Shield library, 340
header files, 208
including libraries, 58-59
EEPROM memory, 194-195
standard libraries, 204
Increase Indent option (Edit menu), 45
increment operator, 80
index values (arrays), 110
indexOf method, 128
initializing data structures, 176
input flapping, 227-228
INPUT interface mode setting, 221
input mode
for analog interfaces
detecting signals, 236
limiting values, 241
mapping values, 242-245, 292
potentiometer example sketch, 238-241
reference voltage changes, 245-246
for digital interfaces
setting, 221
traffic signal example sketch, 229-231
voltage levels, 226-229
INPUT_PULLUP interface mode setting, 221, 228-229
installing. See also importing
Arduino IDE
for Linux, 37
for OS X, 36-37
for Windows, 33-36
drivers, 34-35
LCD shield library, 330-331
libraries, 212
instruction set, 26-27
int data type, 77, 79
integer values, floating-point
values versus, 78
integrated development environment (IDE). See IDE
(integrated development environment)
interface 13 as input, 229
interfaces, 10-11. See also electronic circuits
analog interfaces. See also analog sensors
input signals, detecting, 236
layouts, 237-238
limiting input values, 241
mapping input values, 242-245, 292
output signals, generating, 236-237, 246-247
planning, 389-390
potentiometer input example sketch, 238-241
reference voltage changes, 245-246
Arduino IDE, 39-40
digital interfaces
input voltage levels, 226-229
interface 13, 229
layouts, 220-221
number of, 219-220
output voltage levels, 221-223
planning, 390-391
setting input/output modes, 221
traffic signal example sketch, 223-226, 229-231
troubleshooting, 226
I2C (Inter-Integrated Circuit) protocol, 278
interrupts
external interrupts, 252-260
ignoring, 264-265
pin change interrupts, 253-254, 260-262
polling versus, 251-252
timer interrupts, 254, 262-264
I/O interface, 25-26
LCD (liquid crystal display) devices
Arduino interface connections, 323-325
interface pins, 321-323
LCD shield connections, 332-333
SD cards, 375-376
serial port interfaces, 268-269
sketches with electronic circuits, 64-69
adding to projects, 68-69
header socket usage, 64-66
SPI (Serial Peripheral Interface) protocol, 274-276
on Uno R3 unit, 15
Inter-Integrated Circuit (I2C) protocol, 277-284
blinking LED example sketch, 280-284
interfaces, 278
Wire library functions, 278-280
internal reference voltages, 245-246
interrupt service routine (ISR), 252, 255
interrupts
external interrupts, 252-253
enabling, 254-255
traffic signal example sketch, 255-260
ignoring, 264-265
pin change interrupts, 253-254
importing PinChangeInt library, 260-261
traffic signal example sketch, 261-262
polling versus, 251-252
timer interrupts, 254
importing Timer One library, 263
testing, 263-264
interrupts() function, 264-265
I/O interface in ATmega AVR microcontrollers, 25-26
IOREF header socket port, 65
IP addresses
dynamic IP addresses, 342-343
static addresses, 341-342
IPAddress class, 340-342
isDirectory method, 377
ISR (interrupt service routine), 252, 255
K-L
kits, 18
lastIndexOf method, 128
LCD (liquid crystal display) devices, 319
Arduino interface connections, 323-325
color types, 320-321
display types, 319-320
interface pins, 321-323
LCD shield, 329-330
connections, 332-333
downloading and installing library, 330-331
library functions, 331-332
LiquidCrystal library example usage, 327
functions, 325-326
temperature display example sketch, 327-329, 333-335
troubleshooting, 329
LCD shield, 19, 329-330
connections, 332-333
downloading and installing library, 330-331
library functions, 331-332
temperature display example sketch, 333-335
LDR (light-dependent resistor), 296
LEDs
resistors and, 256
traffic signal example sketch. See traffic signal example sketch
on Uno R3 unit, 16-17
WiFi shield, 339
left shift operator, 80
leftToRight() function, 325
legal issues, trademark protection of Arduino name, 9
length method, 128
Leonardo model, 13
digital interfaces, 219
external interrupts, 253
I2C interface pins, 278
libraries, 201
building
code file creation, 208-210
temporary usage, 212-214
header file creation, 210-211
installing, 212
zip file creation, 211-212
components of, 202
contributed libraries, 206-208
including, 58-69, 194-195
location, 202-203
standard libraries
compiling functions, 205
documentation, 205
example usage, 205-206
including, 204
list of, 203-204
referencing functions in, 204-205
troubleshooting, 213
light meter example sketch, 296-297
light sources for LCDs, 320-321
light-dependent resistor (LDR), 296
LilyPad model, 15
limiting analog input values, 241
Linux, Arduino IDE installation, 37
LiquidCrystal library, 203
temporary usage, 327
functions, 325-326
temperature display example sketch, 327-329
listings
sketch0401 code, 59
sketch0602 code, 94

How can we make this index more useful? Email us at indexes@samspublishing.com
local variables, 80, 156-158
localIP method, 340
location of libraries, 202-203
logic flow control. See loops; structured commands
logical AND operator, 80, 97
logical NOT operator, 80, 98
logical OR operator, 80, 97
long data type, 77, 79
loop function, 58
loops, 103-104
arrays, 109-112
break statements, 113-114
continue statements, 114-116
do-while statements, 106-107
endless loops, 106
nesting, 112-113
for statements, 107-109, 112
while statements, 104-106
LOW external interrupt mode, 255
lowByte() function, 86
low-current devices, digital interface connections, 221-223
memory in ATmega AVR microcontrollers, 9-10, 25
comparison among types, 181-182
creating dynamic variables, 185-189
EEPROM memory, 194-197
flash memory, 189-193
SRAM memory, 183-185
pointers. See pointers variables. See variables
memory address wrap, 195
memory collisions, 185
memory leaks, 185
menu bar
Edit menu commands, 44-46
File menu commands, 40-43
Help menu commands, 48
Sketch menu commands, 46
Tools menu commands, 46-48
message area (Arduino IDE), 49-50
metal-oxide-semiconductor field-effect transistor (MOSFET), 306
methods
Ethernet class, 340
EthernetClient class, 343
EthernetServer class, 345
EthernetUDP class, 347
File class, 377-378
SD class, 376-377
String object methods comparison, 128-129
manipulation, 130-131
Micro model, 14
analog interfaces, 236
digital interfaces, 219
microcontrollers
ATmega AVR microcontrollers, 9-10
components of, 23-26
EEPROM memory, 194-197
flash memory, 189-193
memory comparisons, 181-182
SRAM memory, 183-185
block diagram, 8, 24
defined, 7-8
programming
Arduino programming language. See Arduino programming language assembly language, 27-28
C programming language, 28-29. See also C programming language machine code, 26
micros() function, 84
Micro-SD breakout board, 376
millis() function, 84
min() function, 85
Mini model
analog interfaces, 236
digital interfaces, 219
missing libraries, troubleshooting, 213
mkdir method, 376, 381
models of Arduino, 12-15
digital interfaces, number of, 219-220
Due, 13
Esplora, 14
Ethernet, 15, 339
Leonardo, 13
LilyPad, 15
Mega, 13
Micro, 14
Uno, 12-13, 15-17
Yun, 14-15
modifying example sketches, 41
modulus operator, 80
MOSFET (metal-oxide-semiconductor field-effect transistor), 306
motor shield, 19, 316
motors, 18
DC motors, 303-304
direction control, 307-308
powering on/off, 305-306, 308-311
speed control, 306-307, 311-313
How can we make this index more useful? Email us at indexes@samspublishing.com
PinChangeInt library, 253, 260-261
pinMode() function, 221
planning projects, 387-389
analog interfaces, 389-390
breadboard circuits, 393-394
components needed, 391-392
digital interfaces, 390-391
schematics, 392
sketches, 394-395
PoE (Power over Ethernet), 338
pointers, 163-166
arithmetic with arrays, 168-171
data structures and, 173-176
null pointers, 167-168
passing to functions, 176-178
printing, 166
referencing strings, 172-173
retrieving data, 166-167
storing data, 167
string manipulation, 171-172
void pointers, 168
polling, interrupts versus, 251-252
ports
lower header sockets, 65
upper header socket, 66
position method, 377
positioning servo motors example sketch, 314-316
positive LCD displays, 320-321
POST method token, 357
potentiometer example sketch
input mapping, 242-245
input mode, 238-241
servo motors, 314-316
pow() function, 85
power, external sources, 17
Power over Ethernet (PoE), 338
powering on/off
Arduino with USB hub, 69
DC motors, 305-306, 308-311
precedence in mathematical operations, 82
Preferences option (File menu), 43
print() function
LiquidCrystal library, 325
Serial library, 63, 83, 270-271
print method
EthernetClient class, 343
EthernetServer class, 345
File class, 377
Print option (File menu), 43
printed circuit board (PCB), 67
printing pointers, 166
println() function, 83, 270-271
println method
EthernetClient class, 343
EthernetServer class, 345
File class, 377
private functions, 211
Pro model, 288
processors on ATmega AVR microcontrollers, 9-10
prog_char flash data type, 190
prog_int16_t flash data type, 190
prog_int32_t flash data type, 190
prog_uchar flash data type, 190
prog_uint16_t flash data type, 190
prog_uint32_t flash data type, 190
PROGMEM keyword, 190
program counters, 24
Programmer option (Tools menu), 47
programming microcontrollers
Arduino programming language. See Arduino programming language assembly language, 27-28
C programming language, 28-29. See also C programming language machine code, 26
programs. See sketches
project development
breadboards, creating circuits, 393-394
prototype circuit boards, creating, 399-401
with Prototype shield, 20
requirements
for analog interfaces, 389-390
components needed, 391-392
determining, 387-389
for digital interfaces, 390-391
schematics, creating, 392
sketches
planning, 394-395
testing, 398-399
writing, 395-398
prototype circuit boards, creating, 399-401
Prototype shield, 20, 399-400
public functions, 211
pulldown circuits, 228
pullup circuits, 228-229
PUT method token, 357
PuTTY package, 351
PWM (pulse-width modulation), 26, 237, 246, 306-307, 311-313
Q
qualifiers for variables, 79
Quit option (File menu), 43
quotes for characters and strings, 121
R
random() function, 86
random number generators, 86
randomSeed() function, 86
RC circuits, 298
read() function
EEPROM library, 194
Serial library, 270, 272
Servo library, 313
Wire library, 278
serial communication protocols 415

read method
EthernetClient class, 343
EthernetUDP class, 347
File class, 377
readButtons() function, 331
readBytes() function, 270, 272
readBytesUntil() function, 270, 272
reading files on SD cards, 379-380
realloc() function, 187
recursive functions, 158-160
Redo option (Edit menu), 44
reference operators, 164, 170
reference voltages, changing, 245-246, 290-291
referencing. See also calling functions
Ethernet Shield library, 340
functions in standard libraries, 204-205
strings with pointers, 172-173
reformatting SD cards, 374
relays, 305
remoteIP method, 347
remotePort method, 347
remove method, 376
removing dynamic variables, 187
replace method, 128
request headers (HTTP), 358
request line (HTTP requests), 357
requestFrom() function, 278
requests (HTTP), 356-358
request headers, 358
request line, 357
requirements, determining, 387-389
for analog interfaces, 389-390
components needed, 391-392
for digital interfaces, 390-391
reserve method, 128
Reset button on Uno R3 unit, 17, 92
RESET header socket port, 65
resistance-based analog sensors, 295-297
resistors, 17, 223
LEDs and, 256
in motor circuits, 306
in RC circuits, 297-298
as voltage dividers, 289-290
response header lines (HTTP), 360-361
responses (HTTP), 358-361
response header lines, 360-361
status line, 358-360
restarting sketches, 64, 92
retrieving data
from EEPROM memory, 196-197
from flash memory, 191-192
with pointers, 166-167,
173-176
with serial ports, 272
return code (functions), testing, 97
return statement, 150
returning values
from functions, 150-152
in pointers, 178
rewindDirectory method, 377
right shift operator, 80
rightToLeft() function, 325
RISING external interrupt mode, 254
rmdir method, 376, 382
Robot_Control library, 203
RS-232 serial interfaces, 269
running sketches, 63-64
RX <= 0 header socket port, 66
S
Save As option (File menu), 42
Save icon (toolbar), 49
Save option (File menu), 42
saving text editor files, 210
schematics, creating, 392
scope of variables, 80
in functions, 154
global variables, 155-156
local variables, 156-158
scrollDisplayLeft() function, 325
scrollDisplayRight() function, 325
SD cards
files
reading, 379-380
writing to, 379
folder organization, 381-382
interfaces, 375-376
SD library, 376-378
File class methods, 377-378
SD class methods, 376-377
specifications, 373-375
temperature logging example sketch, 382-384
SD class, 376-377
SD library, 203, 376-378
File class methods, 377-378
SD class methods, 376-377
Secure Digital. See SD cards
seek method, 377
Select All option (Edit menu), 45
semicolon (;), terminating statements, 77
sensitivity
of touch sensors, 300
of voltage-based analog sensors, 291-292
sensors, 18. See also analog sensors
Serial class functions, 83-84
Serial class methods, 85-86
Serial class specifications, 83-84
Serial communication protocols, 267-268
I2C (Inter-Integrated Circuit) protocol, 277-284
blinking LED example sketch, 280-284
interfaces, 278
Wire library functions, 278-280
serial ports, 268-274
blinking LED example sketch, 272-274
How can we make this index more useful? Email us at indexes@samspublishing.com
serial communication protocols

interfaces, 268-269
Serial library, 269-272
SPI (Serial Peripheral Interface) protocol, 274-277
functions, 276-277
interfaces, 274-276
serial events, 274
Serial library
functions, 269-272
interrupts in, 255
serial monitor, 52-54
with external power source, 69
viewing serial port output, 63
Serial Monitor icon (toolbar), 49
Serial Monitor option (Tools menu), 47
Serial Peripheral Interface (SPI) protocol, 274-277, 390
Ethernet shield, 338
functions, 276-277
interfaces, 274-276
Serial Port option (Tools menu), 47
serial ports, 268-274
blinking LED example sketch, 272-274
finding in Windows, 52
interfaces, 268-269
output
Serial class functions, 83-84
viewing, 63
Serial library, 269-272
serialEvent() function, 274
Server class. See EthernetServer class
server communication protocols, 345
Servo library, 203, 313-314
servo motors, 304
positioning example sketch, 314-316
Servo library, 313-314
setBacklightColor() function, 331
setBitOrder() function, 276-277
setCharAt method, 130
setClockDivider() function, 276-277
setCursor() function, 325
setDataMode() function, 276-277
setTimeout() function, 270
setup for Arduino IDE, 51-52
setup function, 58
shield libraries, 32. See also shields
including, 58-59
list of, 203-204
shields
defined, 18
Ethernet shield, 18-19, 337-338
chat server example sketch, 349-351
dynamic IP addresses, 342-343
Ethernet class, 340-341
Ethernet Shield library, 340
EthernetClient class, 343-345
EthernetServer class, 345-347
EthernetUDP class, 347-349
IPAddress class, 341-342
LCD shield, 19, 329-330
connections, 332-333
downloading and installing library, 330-331
library functions, 331-332
temperature display example sketch, 333-335
motor shield, 19, 316
Prototype shield, 20, 399-400
SD card support, 375-376
WiFi shield, 339
Show Sketch Folder option (Sketch menu), 46
Sketchbook option (File menu), 41
sketches. See also Arduino programming language; listings
analog sensor interfaces
photoresistor example sketch, 296-297
temperature detection example sketch, 293-295
temperature LCD display example sketch, 327-329, 333-335
temperature logging example sketch, 382-384
temperature sensors for web servers, 361-364
touch sensor example sketch, 298-300
chat server example sketch, 349-351
coding format, 57-58
compiling, 60-61
DC motors
powering on/off, 308-311
speed control, 311-313
debugging, 83
downloading and installing library, 64-69
adding to projects, 68-69
analog output generation, 246-247
blinking LED example sketch, 272-274, 280-284
breadboards, 67-68
external interrupts, 255-260
header socket usage, 64-66
input mapping, 242-245
pin change interrupts, 261-262
potentiometer example sketch, 238-241
traffic signal example sketch, 223-226, 229-231, 364-370
example sketches, modifying, 41
file extensions, 41
formatting, 91
HTML in, 44
libraries, including, 58-59
planning, 394-395
restarting, 64, 92
running, 63-64
servo motors, positioning, 314-316
testing, 398-399
uploading, 62-63
writing, 395-398
slave mode (SPI), 274
sockets, 10-11
accessing, 66
electronic circuit interfaces, 64-66
on Uno R3 unit, 15-16
SoftwareSerial library, 203
speed of DC motors, controlling, 306-307, 311-313
SPI (Serial Peripheral Interface) protocol, 274-277, 390
Ethernet shield, 338
functions, 276-277
interfaces, 274-276
SPI library, 203, 276-277
sort() function, 85
SRAM memory, 25
comparison with EEPROM and flash memory, 181-182
dynamic variables, 185-189
changing, 187
defining, 186-187
example usage, 187-189
removing, 187
heap data area, 183-185
stack data area, 183-185
stack pointers, 24
standard libraries
compiling functions, 205
documentation, 205
example usage, 205-206
including, 204
list of, 203-204
referencing functions in, 204-205
startsWith method, 128
statements, terminating, 77
static IP addresses, 341-342
static random-access memory. See SRAM memory
status codes (HTTP), list of, 359
status line (HTTP responses), 358-360
status registers, 24
Stepper library, 203
stepper motors, 304
stop method
EthernetClient class, 343
EthernetUDP class, 347
storage. See also memory
SD cards
folder organization, 381-382
interfaces, 375-376
reading files, 379-380
SD library, 376-378
specifications, 373-375
temperature logging
example sketch, 382-384
writing to files, 379
of strings, 78
of values with pointers, 167, 173-176
strcmp() function, 123, 125-126
strcmp_P() function, 191
strcpy() function, 123, 125, 127
strict typing, 76
String objects, 126-129
creating and manipulating, 126-128
in data structures, 142
methods
comparison, 128-129
manipulation, 130-131
strings
in Arduino programming language, 126-129
creating and manipulating, 126-128
manipulating, 130-131
String object methods, 128-129
in C programming language, 120-126
comparing, 125-126
creating, 121-122
functions for, 122-125
comparisons, 96
copying, 125
displaying, 122
displayed, 119-120
manipulating with pointers, 171-172
referencing with pointers, 172-173
storing, 78
strlen() function, 123
strlen_P() function, 191
strstr() function, 123
struct statement, 134-136
structured commands. See also loops
comparisons, 95-97
Boolean comparisons, 96-97
compound conditions, 97
negating condition checks, 98
numeric comparisons, 95-96
string comparisons, 96
else if statements, 93-95
else statements, 92-93
if statements, 89-92
switch statements, 98-99
structures. See data structures
substring method, 128
subtraction operator, 80
switch bounce, 260, 332
switch statements, 98-99
switches, 17, 229-231

How can we make this index more useful? Email us at indexes@samspublishing.com
tan() function, 85
Telnet clients, 351

温度检测示例草图，293-295
对于LCD显示器，327-329, 333-335
对于SD卡，382-384
对于web服务器，361-364

温度监控示例项目
analog interfaces, 389-390
breadboard circuits, creating, 393-394
components needed, 391-392
digital interfaces, 390-391
planning, 388-389
schematics, creating, 392

terminating statements, 77
testing
function results, 97
I2C interface, 280-284
serial ports, 272-274
sketches, 398-399
timer interrupts, 263-264
text. See strings
text editor files, saving, 210
TFT library, 203
time functions, 84-85
timer interrupts, 254
importing Timer One library, 263
testing, 263-264
Timer One library, importing, 263
TMP36 sensor, 293-295, 361-364
toCharArray method, 128
tolnt method, 128
toLowerCase method, 130
toolBar (Arduino IDE), 49
Tools menu commands, 46-48
touch sensors, 297-300
toupperCase method, 130
TRACE method token, 357
trademark protection of Arduino name, 9
traffic signal example sketch controlling from web browser, 364-370
external interrupts, 255-260
input mapping, 242-245
input mode, 229-231
output mode, 223-226
pin change interrupts, 261-262
transfer() function, 276-277
transistors, 305-306
Transistor-transistor-logic (TTL)-level voltages, 269
trcpy_P() function, 191
trim method, 130
troubleshooting
compiler errors, 61
debugging sketches, 83
digital interfaces
input voltage levels, 227
with serial monitor, 226
flushing SD card data, 378
functions, 148
importing PinChangeInt library, 261
LCD (liquid crystal display) devices, 329
memory
EEPROM memory, 194-195
out of memory errors, 186
missing libraries, 213
modifying example sketches, 41
switch bounce, 260
TTL (Transistor-transistor-logic)-level voltages, 269
TX -> 1 header socket port, 66
.txt file extension, 210

Ubuntu, Arduino IDE installation, 37
UDP (User Datagram Protocol), 347-349
Undo option (Edit menu), 44
unions, 142-145
Universal Resource Indicator (URI), 357
unnamed data structures, 136
Uno model, 12-13
analog interfaces, 236
digital interfaces, 219
external interrupts, 253
I2C interface pins, 278
specifications, 15-17
unsigned variable qualifier, 79
Upload icon (toolbar), 49
Upload option (File menu), 42
Upload Using Programmer option (File menu), 42
uploading
bootloader, 48
sketches, 62-63
URI (Universal Resource Indicator), 357
USB A-B cables, 17
USB hub, powering on/off Arduino, 69
USB ports on Uno R3 unit, 16
USB serial interface, 268
Use Selection for Find option (Edit menu), 46
User Datagram Protocol (UDP), 347-349
user-created libraries. See building libraries
user-defined functions, 147
calling, 148-150
defining, 148
passing values to, 152-154
recursive functions, 158-160
returning values, 150-152
scope of variables, 154
global variables, 155-156
local variables, 156-158
troubleshooting, 148
V
values
- analog input values, 241
- limiting, 241
- mapping, 242-245, 292
- assigning
to data structures, 136-138
to variables, 77
- passing to functions, 152-154
- retrieving
from EEPROM memory, 196-197
from flash memory, 191-192
with pointers, 166-167, 173-176
- returning
from functions, 150-152
in pointers, 178
- storing with pointers, 167, 173-176
- voltage values, converting, 292-293
variable resistors, 17
variables
- arrays. See arrays
- assigning values, 77
- data structures. See data structures
data types, 77-78
declaring, 76-77
dynamic variables, 184-189
- changing, 187
defining, 186-187
- example usage, 187-189
- removing, 187
- in flash memory, 190-191
- pointers. See pointers
- qualifiers, 79
- scope, 80
- in functions, 154
global variables, 155-156
- local variables, 156-158
- unions, 142-145
- viewing, 83
Verify icon (toolbar), 49
Verify/Compile option (Sketch menu), 46
viewing
- serial port output, 63
- variables, 83
Vin header socket port, 65
void data type, 148
void pointers, 168
voltage dividers, 289-290, 295-296
voltage levels
- in analog sensors, 288-291
- in capacitors, detecting, 297-298
- for digital interfaces
  - in input mode, 226-229
  - in output mode, 221-223
- reference voltages, changing, 245-246, 290-291
voltage-based analog sensors, 288-293
- converting voltage values, 292-293
- sensitivity of, 291-292
temperature detection
- example sketch, 293-295
- voltage levels, 288-291
write() function
- EEPROM library, 194
- LiquidCrystal library, 325
- Serial library, 270-271
- Servo library, 313
- Wire library, 278
write method
- EthernetClient class, 343
- EthernetServer class, 345
- EthernetUDP class, 347
- File class, 377
write speeds (SD cards), 374
writeMicroseconds() function, 313
writing
- to files on SD cards, 379
- sketches, 395-398
Y
Yun model, 14-15
- analog interfaces, 236
digital interfaces, 219
Z
zip files, creating, 211-212

How can we make this index more useful? Email us at indexes@samspublishing.com