Python Programming for Arduino

The future belongs to applications and services that involve connected devices, requiring physical components to communicate with web-level applications. Arduino combined with the popular open source software platform Python can be used to develop the next level of advanced Internet of Things (IoT) projects with graphical user interfaces and Internet-connected applications.

Starting with designing hardware prototypes using Arduino, this book will then show you everything you need to know to be able to develop complex cloud applications. You will delve into domain-specific topics with incremental complexity, ending with real-world projects. You will quickly learn to develop user interfaces, plots, remote access, messaging protocols, and cloud connectivity. Each successive topic, accompanied by plenty of examples, will help you develop your cutting-edge hardware applications.

Who this book is written for

This is the book for you if you are a student, hobbyist, developer, or designer with little or no programming and hardware prototyping experience, and you want to develop IoT applications.

If you are a software developer or a hardware designer and want to create connected devices applications, then this book will help you get started.

What you will learn from this book

- Design and develop your own hardware prototypes using Arduino
- Interface Arduino to a computer using the Firmata protocol and Python
- Acquire hands-on experience in developing applications for a variety of popular sensors
- Develop graphical user interfaces to control your components and plots to visualize sensor data
- Access your Arduino prototype from the Internet
- Implement messaging protocols for Arduino communication
- Connect your hardware project to cloud-based services
- Create advanced Internet of Things applications through projects in the book

In this package, you will find:

- The author biography
- A preview chapter from the book, Chapter 1 "Getting Started with Python and Arduino"
- A synopsis of the book’s content
- More information on Python Programming for Arduino

About the Author

Pratik Desai, PhD, is the Principal Scientist and cofounder of a connected devices start-up, Imbue Labs, where he develops scalable and interoperable architecture for wearable devices and Internet of Things (IoT) platforms during the day. At night, he leads the development of an open source IoT initiative, the Semantic Repository of Things. Pratik has 8 years of research and design experience in various layers of the IoT and its predecessor technologies such as wireless sensor networks, RFID, and machine-to-machine (M2M) communication. His domains of expertise are the IoT, Semantic Web, machine learning, robotics, and artificial intelligence.

Pratik completed his MS and PhD from Wright State University, Ohio, and collaborated with the Ohio Center of Excellence in Knowledge-enabled Computing (Kno.e.sis) during his doctoral research. His doctoral research was focused on developing situation awareness frameworks for IoT devices, enabling semantic web-based reasoning and handling the uncertainty associated with sensor data.

In his personal life, Pratik is an avid DIY junkie and likes to get hands-on experience on upcoming technologies. He extensively expresses his views on technology and shares interesting developments on Twitter (@chheplo).

I would like to dedicate the book to my parents, who were responsible for building the foundation of what I am today. The book would not have been possible without the patience, support, and encouragement from my beloved wife, Sachi. I would also like to thank her for landing her photography skills that were used in development of some of the important images used in the book. I would also like to extend my sincere gratitude to the editors for their valuable feedbacks.
Python Programming for Arduino

In the era of the Internet of Things (IoT), it has become very important to rapidly develop and test prototypes of your hardware products while also augmenting them using software features. The Arduino movement has been the front-runner in this hardware revolution, and through its simple board designs it has made it convenient for anyone to develop DIY hardware projects. The great amount of support that is available through the open source community has made the difficulties that are associated with the development of a hardware prototype a thing of the past. On the software front, Python has been the crown jewel of the open source software community for a significant amount of time. Python is supported by a huge amount of libraries to develop various features, such as graphical user interfaces, plots, messaging, and cloud applications.

This book tries to bring you the best of both hardware and software worlds to help you develop exciting projects using Arduino and Python. The main goal of the book is to assist the reader to solve the difficult problem of interfacing Arduino hardware with Python libraries. Meanwhile, as a secondary goal, the book also provides you with exercises and projects that can be used as blueprints for your future IoT projects.

The book has been designed in such a way that every successive chapter has increasing complexity in terms of material that is covered and also more practical value. The book has three conceptual sections (getting started, implementing Python features, and network connectivity) and each section concludes with a practical project that integrates the concepts that you learned in that section.

The theoretical concepts and exercises covered in the book are meant to give you hands-on experience with Python-Arduino programming, while the projects are designed to teach you hardware prototyping methodologies for your future projects. However, you will still need extensive expertise in each domain to develop a commercial product. In the end, I hope to provide you with sufficient knowledge to jump-start your journey in this novel domain of the IoT.

What This Book Covers

Chapter 1, Getting Started with Python and Arduino, introduces the fundamentals of the Arduino and Python platforms. It also provides comprehensive installation and configuration steps to set up the necessary software tools.

Chapter 2, Working with the Firmata Protocol and the pySerial Library, discusses the interfacing of the Arduino hardware with the Python program by explaining the Firmata protocol and the serial interfacing library.
Chapter 3, The First Project – Motion-triggered LEDs, provides comprehensive guidelines to create your first Python-Arduino project, which controls different LEDs according to the detected motion.

Chapter 4, Diving into Python-Arduino Prototyping, takes you beyond the basic prototyping that we performed in the previous project and provides an in-depth description of prototyping methods, with appropriate examples.

Chapter 5, Working with the Python GUI, begins our two-chapter journey into developing graphical interfaces using Python. The chapter introduces the Tkinter library, which provides the graphical frontend for the Arduino hardware.

Chapter 6, Storing and Plotting Arduino Data, covers Python libraries, CSV and matplotlib that are used to store and plot the sensor data respectively.

Chapter 7, The Midterm Project – a Portable DIY Thermostat, contains a practical and deployable project that utilizes the material that we covered in previous chapters such as serial interfacing, a graphical frontend, and a plot of the sensor data.

Chapter 8, Introduction to Arduino Networking, introduces computer networking for Arduino while utilizing various protocols to establish Ethernet communication between the Python program and Arduino. This chapter also explores a messaging protocol called MQTT, with basic examples. This protocol is specifically designed for resource-constrained hardware devices such as Arduino.

Chapter 9, Arduino and the Internet of Things, discusses the domain of the IoT while providing step-by-step guidelines to develop cloud-based IoT applications.

Chapter 10, The Final Project – a Remote Home Monitoring System, teaches a design methodology for the hardware product, followed by a comprehensive project that interfaces the cloud platform with Arduino and Python.

Chapter 11, Tweet-a-PowerStrip, contains another IoT project that is based on everything that we learned in the book. The project explores a unique approach to integrate a social network, Twitter, with the Python-Arduino application.
Getting Started with Python and Arduino

This chapter introduces the Python programming language and the open source electronic prototyping platform Arduino. The first section of the chapter focuses on Python and briefly describes the benefits of Python along with installation and configuration steps. The remaining part of the chapter describes Arduino and Arduino’s development environment.

At the end of this chapter, you will have configured a programming environment for both Python and Arduino for your favorite operating system. If you are a beginner with either or both platforms (that is, Python and Arduino), it is advisable that you follow the given steps in this chapter, as the later chapters will assume that you have the exact configuration described here. If you have previous experience of working with these platforms, you can skip to the next chapter.

Introduction to Python

Since its introduction by Guido van Rossum in 1991, Python has grown into one of the most widely used general-purpose, high-level programming languages, and is supported by one of the largest open source developer communities. Python is an open source programming language that includes a lot of supporting libraries. These libraries are the best feature of Python, making it one of the most extensible platforms. Python is a dynamic programming language, and it uses an interpreter to execute code at runtime rather than using a compiler to compile and create executable byte codes.
The philosophy behind the development of Python was to create flexible, readable, and clear code to easily express concepts. The emphasis on using whitespace indentation in a unique way differentiates Python from other popular high-level languages. Python supports functional, imperative, and object-oriented programming with automatic memory management.

Why we use Python

Python is considered to be one of the easiest languages to learn for first-time programmers. Compared to other popular object-oriented languages such as C++ and Java, Python has the following major benefits for programmers:

- It is easy to read and understand
- It enables rapid prototyping and reduces development time
- It has a humongous amount of free library packages

Python has a huge open source community that drives forth the effort for continuous improvement of Python as a programming language. The Python community is also responsible for the development of a large amount of open library packages, which can be used to build applications that span from dynamic websites to complex data analysis applications, as well as the development of simple GUI-based applications to plot charts from complex math functions. The majority of Python library packages have systematically maintained the code that was obtained from the community with regular updates. The de facto repository that indexes the largest number of Python packages is PyPI (http://pypi.python.org). PyPI also provides simple ways to install various packages on your operating system, which will be covered in the upcoming section.

While working with the hardware platform, it is necessary to have some means of communication between the hardware and the computer that you are using for development. Among the common computer to hardware interfacing methods, serial-port-based communication is the most popular, and it is really simple to establish, especially for the Arduino platform. Python provides a library called pySerial that is really easy to use and quick to implement to interface a serial port. It is really simple to use similar libraries and Python's interactive programming abilities to rapidly test and implement your project ideas.
Nowadays, complex Internet of Things (IoT) applications not only require serial communication support, but they also need additional high-level features such as graphical user interfaces (GUIs) for operating systems, web interfaces for remote access, plots for data visualization, tools for data analysis, interfaces for data storage, and so on. Using any other programming language such as C++ or Java, the development of these features would require a large amount of programming effort due to the distributed and unorganized nature of the supporting tools. Thankfully, Python has been very successful at providing support for these types of applications for years. Python has a number of libraries to support the development of each of the features mentioned here, which are available through PyPI. These libraries are open source, easy to use, and widely supported by the community. This makes Python a language of choice for IoT applications. Additionally, Python also has support to create and ship your custom-built applications as libraries so that everyone else can also utilize them in their projects. This is a helpful feature if you are developing custom protocols, APIs, or algorithms for your own hardware products.

**When do we use other languages**

So, when should we not use Python for our projects? As mentioned earlier, Python is a dynamic language that reduces development time, but it also makes the execution of your code slower as compared to other static high-level languages such as C, C++, and Java. These static languages use a compiler to compile the code and create binaries that get executed during runtime, thereby increasing the runtime performance. When the performance of the code is more important than a longer development time and higher cost, you should consider these static languages. Some other drawbacks of Python include being memory heavy, not having the proper support for threading, and lacking data protection features. In short, we can say that even though Python provides quicker and easier ways for quick prototyping, we should consider other static high-level languages for development after we are done testing our prototype and we are ready to ship our product. Nowadays, this scenario is changing rapidly and companies have started utilizing Python for their industrial products.

You can obtain more Python-related information from the official website at http://www.python.org.
Installing Python and Setuptools

Python comes in two versions: Python v2.x and Python v3.x. (Here, x represents an appropriate version number.) While Python v2.x is a legacy branch and has better library support, Python v3.x is the future of Python. Most Linux distributions and Mac OS X operating systems are equipped with Python, and they have v2.x as their preferred and default version of Python. We will be using Python v2.7 as the default version of Python for the rest of the book due to the following reasons:

- It is the most current version of the Python v2.x branch
- It has large community support and solutions for its known issues are available through support forums
- It is supported by most of the major Python libraries

Even though the code samples, exercises, and projects provided in this book should work in any variant of Python 2.7.x, it's better to have the latest version.

Installing Python

Your fondness for an operating system is developed due to multiple factors, and you can never ignore someone's bias towards a particular OS. Thus, this book provides installation and configuration guidelines for three of the most popular operating systems: Linux, Mac OS X, and Windows. Let's begin by configuring Python for a Linux computer.

Linux

The majority of Linux distributions come with Python preinstalled. To check the latest version of the installed Python, use the following command at the terminal window:

```
$ python -V
```

Make sure that you are using an uppercase V as the option for the previous command. Once you execute it on the terminal, it will print the complete version number of your current Python installation. If the version is 2.7.x, you are good to go and your Linux is updated with the latest version of Python that is required for this book. However, if you have any version that is less than or equal to 2.6.x, you will need to first upgrade Python to the latest version. This process will require root privileges, as Python will be installed as a system component that will replace the previous versions.
Ubuntu

If you are using Ubuntu 11.10 or later versions, you should already have Python v2.7.x installed on your machine. You can still upgrade Python to the latest revision of v2.7.x using the following command:

$ sudo apt-get update && sudo apt-get --only-upgrade install python

If you are running an older version of Ubuntu (such as 10.04 or older), you should have 2.6 as the default version. In this case, you will need to run the following set of commands to install version 2.7:

$ sudo add-apt-repository ppa:fkrull/deadsnakes
$ sudo apt-get update
$ sudo apt-get install python2.7

The first command will add an external Ubuntu repository, which will allow you to install any version of Python. The next command will update and index the list of available packages. The last command will install the latest version of Python 2.7.

Fedora and Red Hat

Fedora and Red Hat Linux also ships with Python as an in-built package. If you want to upgrade the version of Python to the latest one, run the following command at the terminal:

$ sudo yum upgrade python

Windows

Installation and configuration of Python on Windows is not as straightforward as it is for Linux. First of all, you'll need to download a copy of Python from http://www.python.org/getit.

You need to be careful about the version of Python that you are downloading. From the system properties of your Windows OS, check whether the operating system is of 32 bit or 64 bit. At the time this book was being written, the latest version of Python was 2.7.6. So, download the latest available version of Python, but make sure that it is 2.7.x and not 3.x.
Getting Started with Python and Arduino

For many third-party Python libraries, the installation binary files for Windows are compiled for the 32-bit version. Due to this reason, we will recommend that you install the 32-bit version of Python for your Windows OS.

If you are really familiar with Python and know your way around installing libraries, you can install the 64-bit version of Python. Select and run the downloaded file to install Python. Although you can install it to any custom location, it is advisable to use the default installation location as the upcoming configuration steps use the default location. Once the installation is complete, you can find the Python command-line tool and IDLE (Python GUI) from the Start menu.

Although you can always open these tools from the Start menu for basic scripting, we will modify the Windows system parameters to make Python accessible through the Windows command prompt. To accomplish this, we will have to set up PATH in environment variables for the location of the Python installation directory. Let's open System Properties by right-clicking on My Computer and then selecting Properties. Otherwise, you can also navigate to Start | Control Panel | System and Security | System.

You will be able to see a window similar to the one that is displayed in the following screenshot. The System window shows you the basic information about your computer, including the type of Windows operating system that you are using (such as the 32-bit or the 64-bit version):

![System Properties Window](image-url)
In the **System** window, click on **Advanced system settings** in the left navigation bar to open a window called **System Properties**. Click on the **Environment Variables** button in the **System Properties** window, which is located at the bottom of the window. This will open an interface similar to the one shown in the following screenshot. In **Environment Variables**, you need to update the **PATH** system variable to add Python to the default operating system’s path.

Click on the **PATH** option as displayed in the following screenshot, which will pop up an **Edit System Variable** window. Add `C:\Python27` or the full path of your custom Python installation directory at the end of your existing **PATH** variable. It is required to put a semicolon (`;`) before the Python installation path. If you already see Python’s location in the **Path** variable, your system is set up for Python and you don’t need to perform any changes:

![Environment Variables](image)

The main benefit of adding Python to the environment variables is to enable access to the Python interpreter from the command prompt. In case you don’t know, the Windows command prompt can be accessed by navigating to **Start** | **Programs** | **Accessories** | **Command Prompt**.
Mac OS X
Mac OS X ships with a preinstalled copy of Python, but due to the long release cycle of the operating system, the frequency of updates for the default Python application is slow. The latest version of Mac OS X, which is 10.9 Maverick, comes equipped with Python 2.7.5, which is the latest version:

Tests-Mac:~ test$ python
Python 2.7.5 (default, Aug 25 2013, 00:04:04)
[GCC 4.2.1 Compatible Apple LLVM 5.0 (clang-500.0.68)] on darwin
Type "help", "copyright", "credits" or "license" for more information.

Previous versions such as Mac OS X 10.8 Mountain Lion and Mac OS X 10.7 Lion included Python 2.7.2 and Python 2.7.1 respectively, which are also compatible versions for this book. If you are an experienced Python user or someone who wants to work with the latest version of Python, you can download the latest version from http://www.python.org/getit.

Older versions of Mac OS X such as Snow Leopard and later, which came with an older version of Python, can be updated to the latest version by downloading and installing it from http://www.python.org/getit.

Installing Setuptools
Setuptools is a library containing a collection of utilities for building and distributing Python packages. The most important tool from this collection is called easy_install. It allows a user to look into PyPI, the Python package repository that we mentioned previously, and provides a simple interface to install any package by name. The easy_install utility automatically downloads, builds, installs, and manages packages for the user. This utility has been used in the later part of this book to install the necessary packages required for the upcoming projects of Python and Arduino. Although easy_install has been used as a simple way of installing Python packages, it misses out on a few useful features such as tracking actions, support for uninstallation, and support for other version control systems. In recent years, the Python community has started adopting another tool called pip over easy_install that supports these features. As both easy_install and pip utilize the same PyPI repository, going forward, you can use any of these utilities to install the required Python packages.

Just to narrow down the scope, we will be focusing on methods to install Setuptools and the default utilities that get installed with it, that is, easy_install. Later in this section, we will also install pip, just in case you want to use it too. Let's first begin by installing Setuptools for the various operating systems.
Linux

In Ubuntu, setuptools is available in the default repository and it can be installed using the following command:

$ sudo apt-get install python-setuptools

For Fedora, it can be installed using the default software manager yum:

$ sudo yum install python-setuptools

For other Linux distributions, it can be downloaded and built using the following single-line script:

$ wget https://bitbucket.org/pypa/setuptools/raw/bootstrap/ez_setup.py -O - | sudo python

Once it is installed on your Linux distribution, easy_install can be directly accessed from the terminal as a built-in command.

Windows

Installation of Setuptools is not that straightforward for Windows as compared to Linux. It requires the user to download the ez_setup.py file from the Windows section at https://pypi.python.org/pypi/setuptools.

Once this is downloaded, press Shift and right-click in the folder where you downloaded the ez_setup.py file. Select Open command window here and execute the following command:

> python ez_setup.py

This will install Setuptools in the Scripts folder of your default Python installation folder. Using the same method that we used when we added Python to Environment Variables, now include Setuptools by adding C:\Python27\Scripts to PATH, followed by the semicolon (;).

This will enable the installation of various Python packages using easy_install to your Python packages folder called Libs. Once you have added the package manager to the environment variables, you need to close and reopen the command prompt for these changes to take effect.
Mac OS X
Setuptools can be installed in Mac OS X using any of the following methods. It is advisable for beginners to use the first method, as the second method requires the external package manager Homebrew.

If you have never worked with Homebrew before, you will need to follow these steps to install Setuptools on your Mac:

1. Download `ez_setup.py` from the Unix/Mac section at https://pypi.python.org/pypi/setuptools.
2. Open the terminal and navigate to the directory where you downloaded this file. For most browsers, the file gets saved to the Download folder.
3. Run the following command in the terminal to build and set up Setuptools:
   ```
   $ sudo python ez_setup.py
   ```

If you are familiar with Homebrew-based software installation, just follow these quick steps to install Setuptools:

1. First, install `wget` from Homebrew if you don’t have it already:
   ```
   $ brew install wget
   ```
2. Once you have installed `wget`, run the following command in the terminal:
   ```
   $ wget https://bitbucket.org/pypa/setuptools/raw/bootstrap/ez_setup.py -O - | python
   ```

More information regarding the Homebrew utility can be obtained from http://brew.sh.

You can install Homebrew on your Mac by running the following simple script in the terminal:
```
ruby -e "$(curl -fSSL https://raw.githubusercontent.com/Homebrew/install/master/install)"
```

Installing pip
As you have successfully installed Setuptools, let’s use it to install pip. For Linux or Mac OS X, you can run the following command in the terminal to install `pip`:

```
$ sudo easy_install pip
```

For Windows, open the command prompt and execute the following command:
```
> easy_install.exe pip
```
If you have already installed pip on your computer, please make sure that you upgrade it to the latest version to overcome the few bugs that are associated with the upgrade. You can upgrade pip using the following command at the terminal:

$ sudo easy_install --upgrade pip

Since you have already used easy_install to install a Python package, let's get ourselves more familiar with Python package management.

**Installing Python packages**

With the installation of pip, you have two different options to install any third-party Python package listed on the PyPi repository (http://pypi.python.org). The following are the various procedures that you need to know to work with the installation of Python packages. In the following examples, the term PackageName is a pseudo name that is used for a Python package that you want to work with. For your package of choice, identify the appropriate package name from the PyPi website and put its name in place of PackageName. In some cases, you will need root (super user) privileges to install or uninstall a package. You can use sudo followed by an appropriate command for these cases.

To install a Python package, execute the following command at the terminal:

$ easy_install PackageName

Otherwise, you can also execute the following command:

$ pip install PackageName

If you want to install a specific version of a package, you can use the following command:

$ easy_install "PackageName==version"

If you are not aware of the exact version number, you can also use comparison operators such as >, <, >=, or <= to specify a range for the version number. Both easy_install and pip will select the best matching version of the package from the repository and install it:

$ easy_install "PackageName > version"

Meanwhile, for pip, you can use the following identical commands to perform similar operations:

$ pip install PackageName==version
$ pip install "PackageName>=version"
As an example, if you want to install a version between 1.0 and 3.0, you will need to use the following command:

```
$ pip install "PackageName>=0.1,<=0.3"
```

It is really easy to upgrade a package using either `easy_install` or `pip`. The command options used by both are also very similar:

```
$ easy_install --upgrade PackageName
$ pip install --upgrade PackageName
```

Although `easy_install` doesn't support clean uninstallation of a package, you can use the following command to make sure that Python stops searching for the specified package. Later, carefully remove the package files from the installation directory:

```
$ easy_install -mxN PackageName
```

A much better way to perform a clean uninstallation of the majority of packages is to use `pip` instead of `easy_install`:

```
$ pip uninstall PackageName
```

A detailed list of the Python packages supported by `Setuptools` can be found at the PyPI website at https://pypi.python.org/.

The fundamentals of Python programming

If you have previous experience of working with any other programming language, Python is very easy to get started with. If you have never done programming before, this section will walk you through some of the basics of Python. If you have already worked with Python, you should skip this section and move on to the next one.

Assuming that the setup instructions are followed correctly, let's open the Python interpreter by executing the Python command at the terminal or the command prompt. You should get results similar to those displayed in the following screenshot. If you have installed Python by downloading the setup files from the website, you should have the Python `integrated development environment` (IDLE) installed as well. You can also start the Python interpreter by opening its IDLE from the location where it was installed.
As you can see, after printing some system information, the interpreter opens a prompt with three greater-than signs (>>>, which is also known as the primary prompt. The interpreter is now in the interactive mode and it is ready to execute scripts from the prompt.

To close the interactive mode of the Python interpreter, run the either `exit()` or `quit()`, at the primary prompt. Another method to exit from the interactive mode is to use the keyboard shortcut Ctrl + D.

Note that Python's built-in functions are case sensitive. This means the following:

```python
exit() ≠ EXIT() ≠ Exit()
```

The official Python website provides comprehensive tutorials for beginners to get started with Python programming. It is highly recommended that you visit the official Python tutorials at [https://docs.python.org/2/tutorial/index.html](https://docs.python.org/2/tutorial/index.html) if you are looking for detailed programming tutorials as compared to the upcoming brief overviews.

### Python operators and built-in types

Now that you have a brief idea regarding the Python prompt, let's get you familiar with some of the basic Python commands. For these exercises, we will be using the Python IDLE, which also opens with the Python interactive prompt. You will require a method to describe the code segments, tasks, and comments when writing large and complex code. Non-executable content is called comments in any programming language, and in Python, they start with the hashtag character (#). Like comments, you will be frequently required to check the output by printing on the prompt using the print command:

```python
>>> # Fundamental of Python
>>> # My first comment
```
>>> name = "John"  # This is my name
>>> print name
John

Instead of IDLE, you can also access the Python interactive prompt from the terminal. When using Python from the terminal, make sure that you are taking care of the indentation properly.

Operators
Python supports the usage of basic mathematical operators such as +, -, *, and /, directly from the interpreter. Using these operators, you can perform basic calculations in the prompt, as shown in the following examples. Try these operations in your prompt in order to start using the Python interpreter as a calculator:

>>> 2 + 2
4
>>> (2*3) + 1
7
>>> (2*3) / 5
1

When working with the Python interpreter, it is recommended that you follow the Style Guide for Python Code, which is also popularly known as PEP-8 or pep8. For more information about PEP-8, visit https://www.python.org/dev/peps/pep-0008/.

Built-in types
Python is a dynamically typed language, which means that you don't have to explicitly declare the type of the variables when initializing them. When you assign a value to a variable, the Python interpreter automatically deduces the data type. For example, let's declare the following variables in the interactive mode of the interpreter:

>>> weight = height = 5
>>> weight * height
25
>>> type(weight)
<type 'int'>
While assigning the value to the `weight` variable, we didn't specify the data type, but the Python interpreter assigned it as an integer type, `int`. The interpreter assigned the `int` type due to the reason that the numerical value didn't contain any decimal points. Let's now declare a variable with a value containing a decimal point. The built-in function `type()` that can be used to find out the data type of a specified variable:

```python
>>> length = 6.0
>>> weight * height * length
150.0
>>> type(length)
<type 'float'>
```

As you can see, the interpreter assigns the data type as `float`. The interpreter can also deduce the type of complex numbers, as shown in following examples. You can access the real and imaginary value of a complex number using the dot (.) operator followed by `real` and `imag`:

```python
>>> val = 2.0 + 3.9j
>>> val.real
2.0
>>> val.imag
3.9
```

Just to play more with complex numbers, let's try the `abs()` and `round()` functions as displayed in the following examples. They are built-in Python functions to obtain the absolute value and the rounded number respectively:

```python
>>> abs(val)
4.382921400162225
>>> round(val.imag)
4.0
```

Like numbers, the Python interpreter can also automatically identify the declaration of string data types. In Python, string values are assigned using single or double quotes around the value. When the interpreter sees any value enclosed within quotes, it considers it to be a string. Python supports the usage of the `+` operator to concatenate strings:

```python
>>> s1 = "Hello"
>>> s2 = "World!"
>>> s1 + s2
'HelloWorld!'
>>> s1 + " " + s2
'Hello World!'
```
A character type is a string of size one and the individual characters of a string can be accessed by using index numbers. The first character of a string is indexed as 0. Play with the following scripts to understand indexing (subscripting) in Python:

```python
>>> s1[0]
'H'
>>> s1[:2]
'He'
>>> s1 + s2[5:]
'Hello!'```

Similar to the primary prompt with default notation `>>>`, the Python interactive interpreter also has a secondary prompt that uses three dots (…) when it is being used from the terminal. You won't be able to see the three dots in IDLE when you use the secondary prompt. The secondary prompt is used for a multiline construct, which requires continuous lines. Execute the following commands by manually typing them in the interpreter, and do not forget to indent the next line after the `if` statement with a tab:

```python
>>> age = 14
>>> if age > 10 or age < 20:
...    print "teen"

teen```

---

**Data structures**

Python supports four main data structures (`list`, `tuple`, `set`, and `dictionary`) and there are a number of important built-in methods around these data structures.

**Lists**

Lists are used to group together values of single or multiple data types. The `list` structure can be assigned by stating values in square brackets with a comma (,) as a separator:

```python
>>> myList = ['a', 2, 'b', 12.0, 5, 2]
>>> myList
['a', 2, 'b', 12.0, 5, 2]```
Like strings, values in a list can be accessed using index numbers, which starts from 0. A feature called **slicing** is used by Python to obtain a specific subset or element of the data structure using the colon operator. In a standard format, slicing can be specified using the `myList[start:end:increment]` notation. Here are a few examples to better understand the notion of slicing:

- You can access a single element in a list as follows:
  ```python
  >>> myList[0]
  'a'
  ```

- You can access all the elements in the list by having empty start and end values:
  ```python
  >>> myList[:]
  ['a', 2, 'b', 12.0, 5, 2]
  ```

- You can provide start and end index values to obtain a specific subset of the list:
  ```python
  >>> myList[1:5]
  [2, 'b', 12.0, 5]
  ```

- Use of the minus symbol with an index number tells the interpreter to use that index number backwards. In the following example, -1 backwards actually represents the index number 5:
  ```python
  >>> myList[1:-1]
  [2, 'b', 12.0, 5]
  ```

- You can obtain every other element of the list by providing the increment value with start and end values:
  ```python
  >>> myList[0:5:2]
  ['a', 'b', 5]
  ```

- You can check the length of a list variable using the `len()` method. The usage of this method will be handy in the upcoming projects:
  ```python
  >>> len(myList)
  6
  ```

- You can also perform various operations to add or delete elements in the existing list. For example, if you want to add an element at the end of the list, use the `append()` method on the list:
  ```python
  >>> myList.append(10)
  >>> myList
  ['a', 2, 'b', 12.0, 5, 2, 10]
  ```
• To add an element at a specific location, you can use the `insert(i, x)` method, where `i` denotes the index value, while `x` is the actual value that you want to add to the list:

```python
>>> myList.insert(5,'hello')
>>> myList
['a', 2, 'b', 12.0, 5, 'hello', 2, 10]
```

• Similarly, you can use `pop()` to remove an element from the list. A simple `pop()` function will remove the last element of the list, while an element at a specific location can be removed using `pop(i)`, where `i` is the index number:

```python
>>> myList.pop()
10
>>> myList
['a', 2, 'b', 12.0, 5, 'hello', 2]
>>> myList.pop(5)
'hello'
>>> myList
['a', 2, 'b', 12.0, 5, 2]
```

### Tuples

Tuples are immutable data structures supported by Python (different from the mutable structures of lists). An immutable data structure means that you cannot add or remove elements from the tuple data structure. Due to their immutable properties, tuples are faster to access compared to lists and are mostly used to store a constant set of values that never change.

The `tuple` data structure is declared like `list`, but by using parentheses or without any brackets:

```python
>>> tupleA = 1, 2, 3
>>> tupleA
(1, 2, 3)
>>> tupleB = (1, 'a', 3)
>>> tupleB
(1, 'a', 3)
```

Just like in a `list` data structure, values in `tuple` can be accessed using index numbers:

```python
>>> tupleB[1]
'a'
```
As tuples are immutable, list manipulation methods such as `append()`, `insert()`, and `pop()` don't apply for tuples.

### Sets
The `set` data structure in Python is implemented to support mathematical set operations. The `set` data structure includes an unordered collection of elements without duplicates. With its mathematical use cases, this data structure is mostly used to find duplicates in lists, as conversion of a list to a set using the `set()` function removes duplicates from the list:

```python
global listA = [1, 2, 3, 1, 5, 2]
global setA = set(listA)
global setA
set([1, 2, 3, 5])
```

### Dictionaries
The `dict` data structure is used to store key-value pairs indexed by keys, which are also known in other languages as associative arrays, hashes, or hashmaps. Unlike other data structures, `dict` values can be extracted using associated keys:

```python
>>> boards = {'uno':328, 'mega':2560, 'lily': '128'}
>>> boards['lily']
'lily'
>>> boards.keys()
['lily', 'mega', 'uno']
```

You can learn more about Python data structures and associated methods at [https://docs.python.org/2/tutorial/datastructures.html](https://docs.python.org/2/tutorial/datastructures.html).

### Controlling the flow of your program
Just like any other language, Python supports controlling the program flow using compound statements. In this section, we will briefly introduce these statements to you. You can get detailed information about them from the official Python documentation at [https://docs.python.org/2/reference/compound_stmts.html](https://docs.python.org/2/reference/compound_stmts.html).
The if statement

The if statement is the most basic and standard statement used to set up conditional flow. To better understand the if statement, execute the following code in the Python interpreter with different values of the age variable:

```python
>>> age = 14
>>> if age < 18 and age > 12:
    print "Teen"
elif age < 13:
    print "Child"
else:
    print "Adult"
```

This will result in Teen being printed on the interpreter.

The for statement

Python's for statement iterates over the elements of any sequence according to the order of the elements in that sequence:

```python
>>> celsius = [13, 21, 23, 8]
>>> for c in celsius:
    print " Fahrenheit: " + str((c * 1.8) + 32)
```

This will result in the Python interpreter generating the following output that will display the calculated Fahrenheit values from the given Celsius values:

Fahrenheit: 55.4  
Fahrenheit: 69.8  
Fahrenheit: 73.4  
Fahrenheit: 46.4

The while statement

The while statement is used to create a continuous loop in a Python program. A while loop keeps iterating over the code block until the condition is proved true:

```python
>>> count = 5
>>> while (count > 0):
    print count
    count = count - 1
```
The **while** statement will keep iterating and printing the value of the variable count and also reduce its value by 1 until the condition, that is $(\text{count} > 0)$, becomes true. As soon as the value of count is lower than or equal to 0, the while loop will exit the code block and stop iterating.

The other compound statements supported by Python are **try/catch** and **with**. These statements will be explained in detail in the upcoming chapters. Python also provides loop control statements such as **break**, **continue**, and **pass** that can be used while a loop is being executed using the compound statements mentioned earlier. You can learn more about these Python features from https://docs.python.org/2/tutorial/controlflow.html.

### Built-in functions

Python supports a number of useful built-in functions that do not require any external libraries to be imported. We have described a few of these functions as a collection of a respective category, according to their functionalities.

#### Conversions

Conversion methods such as **int()**, **float()**, and **str()** can convert other data types into integer, float, or string data types respectively:

```python
>>> a = 'a'
>>> int(a,base=16)
10
>>> i = 1
>>> str(i)
'1'
```

Similarly, **list()**, **set()**, and **tuple()** can be used to convert one data structure into another.

#### Math operations

Python also supports built-in mathematical functions that can find the minimum and/or maximum values from a list. Check out the following examples and play around with the different data structures to understand these methods:

```python
>>> list = [1.12, 2, 2.34, 4.78]
>>> min(list)
1.12
>>> max(list)
4.78
```
The `pow(x, y)` function returns the value of $x$ to the power of $y$:

```python
>>> pow(3.14159, 2)
9.869587728099999
```

## String operations

Python provides easy access to string manipulation through built-in functions that are optimized for performance. Let's take a look at the following examples:

- Code to replace occurrences of a string or substring with a different one:
  ```python
  >>> str = "Hello World!"
  >>> str.replace("World", "Universe")
  'Hello Universe!'
  ```

- Code to split a string with a separating character where the default character is space:
  ```python
  >>> str = "Hello World!"
  >>> str.split()
  ['Hello', 'World!']
  ```

- Code to split a string from a separating character for any other character:
  ```python
  >>> str2 = "John, Merry, Tom"
  >>> str2.split(",")
  ['John', 'Merry', 'Tom']
  ```

- Code to convert an entire string value into uppercase or lowercase:
  ```python
  >>> str = "Hello World!"
  >>> str.upper()
  'HELLO WORLD!'
  >>> str.lower()
  'hello world!'
  ```

The Python documentation on the official website covers every built-in function in detail with examples. For better understanding of Python programming, visit https://docs.python.org/2/library/functions.html.
Introduction to Arduino

Any electronic product that needs computation or interfacing with other computers first requires a quick prototyping of the concept using simple tools. Arduino is an open source hardware prototyping platform designed around a popular microcontroller family, and it includes a simple software development environment. Besides prototyping, you can also use Arduino for the development of your own do-it-yourself (DIY) projects. Arduino bridges the computational world with the physical world by letting you simply connect the sensors and actuators with a computer. Basically, you can write code to monitor and control various electronic components in your daily life by using Arduino's input/output pins and microcontroller. Examples of these components include motors, thermostats, lights, switches, and many more.

History

In 2005, Massimo Banzi, the Italian cofounder of Arduino, developed the technology for his students at Interaction Design Institute Ivrea (IDII). Since then, Arduino has developed into one of the largest open source hardware platforms. All software components and schematics of the Arduino design are open source, and you can buy the hardware at a very low cost—approximately 30 dollars—or you can even make it yourself.

Why Arduino?

The major goal of the Arduino community is to continuously improve the Arduino platform with the following objectives in mind:

- The Arduino platform should be an affordable platform
- It should be easy to use and easy to code
- It should be an open source and extensible software platform
- It should be an open source and extensible hardware platform
- It should have community-supported DIY projects
These simple but powerful objectives have made Arduino a popular and widely used prototyping platform. Arduino uses Atmel’s ATmega series of microcontrollers that are based on the popular hardware architecture of AVR. The huge support that is available for AVR architecture also makes Arduino a hardware platform of choice. The following image shows the basic version of the Arduino board, which is called Arduino Uno (Uno means one in Italian):

![Arduino Uno board](image)

**Arduino variants**

Like any other project, hardware requirements are driven by project specifications. If you are developing a project that requires you to interface with a large number of external components, you need a prototyping platform that has a sufficient number of input/output (I/O) pins for interfacing. If you are working on a project that needs to perform a huge amount of complex calculations, you require a platform with more computation capability.

Fortunately, the Arduino board exists in 16 different official versions, and each version of Arduino differs from the others in terms of form factor, computational power, I/O pins, and other on-board features. Arduino Uno is the basic and most popular version, which is sufficient enough for simple DIY projects. For the majority of exercises in this book, we will be using the Arduino Uno board. You can also use another popular variant called Arduino Mega, which is a larger board with extra pins and a powerful microcontroller. The following table shows the comparison of some of the more popular and active variants of the Arduino board:
Any of these variants can be programmed using a common integrated development environment called Arduino IDE, which is described in the upcoming section. You can select any one of these Arduino boards according to your project requirements, and the Arduino IDE should be able to compile and download the program to the board.

The Arduino Uno board

As Uno is going to be the de facto board for the majority of the projects in this book, let’s get ourselves familiar with the board. The latest revision of the Uno board is based on Atmel's ATmega328 microcontroller. The board extends the I/O pins of the microcontroller to the peripheral, which can then be utilized to interface components using wires. The board has a total of 20 pins to interface, out of which 14 are digital I/O pins and 6 are analog input pins. From the 14 digital I/O pins, 6 pins also support pulse-width modulation (PWM), which supports the controlled delivery of power to connected components.

The board operates on 5V. The maximum current rating of the digital I/O pins is 40 mA, which is sufficient to drive most of the DIY electronic components, excluding motors with high current requirements.
While the previous image provided an overview of the Uno board, the following diagram describes the pins on the Uno board. As you can see, the digital pins are located on one side of the board while the analog pins are on the opposite side. The board also has a couple of power pins that can be used to provide 5V and 3.3V of power to external components. The board contains ground pins on both sides of the board as well. We will be extensively using 5V of power and ground pins for our projects. Digital pins D0 and D1 support serial interfacing through the Tx (transmission) and Rx (receiver) interfaces respectively. The USB port on the board can be used to connect Arduino with a computer.

Now that we are familiar with the Arduino hardware, let's move on to programming the Arduino board.

**Installing the Arduino IDE**
The first step to start getting familiar with Arduino is to install the Arduino integrated development environment (IDE). According to the operating system that you selected at the beginning of the Python installation section, follow the appropriate subsection to install the correct IDE.
Linux

The installation of the Arduino IDE is really simple in Ubuntu. The Ubuntu repository already includes the Arduino IDE with the required dependencies.

For Ubuntu 12.04 or a newer version, execute the following command in the terminal to install Arduino:

\$ sudo apt-get update && sudo apt-get install arduino arduino-core

The latest version of the Arduino IDE in the Ubuntu repository is 1.0.3. You can obtain more information regarding other Ubuntu-related questions at http://playground.arduino.cc/Linux/Ubuntu.

For Fedora 17 or a newer version of Red Hat Linux, execute the following script in the terminal:

\$ sudo yum install arduino

Answers to additional installation questions for Fedora can be obtained at http://playground.arduino.cc/Linux/Fedora.

Mac OS X

To install the Arduino IDE on Mac OS X (10.7 or newer), perform the following steps:

1. From http://arduino.cc/en/Main/Software, download the latest version of the Arduino IDE for Mac OS X, which was 1.0.5 when this book was being written.

2. Unzip and drag Arduino to the application folder.

The Arduino IDE is built in Java and requires that your computer is equipped with the appropriate version of Java. Open the IDE from your applications. If you don't have Java installed on your Mac, the program will prompt you with a pop-up window and ask you to install Java SE 6 runtime. Go ahead and install Java (as per the request) as the OS X will automatically install it for you.

Windows

Installation of Arduino for Windows is very simple. Download the setup file from http://arduino.cc/en/Main/Software. Select the most recent version of the Arduino IDE, that is, 1.0.x or a newer version.
Make sure you download the appropriate version of the Arduino IDE according to your operating system, that is, 32 bit or 64 bit. Install the IDE to the default location as specified in the installation wizard. Once installed, you can open the IDE by navigating to Start | Programs.

**Getting started with the Arduino IDE**

The Arduino IDE is a cross-platform application developed in Java that can be used to develop, compile, and upload programs to the Arduino board. On launching the Arduino IDE, you will find an interface similar to the one displayed in the following screenshot. The IDE contains a text editor for coding, a menu bar to access the IDE components, a toolbar to easily access the most common functions, and a text console to check the compiler outputs. A status bar at the bottom shows the selected Arduino board and the port name that it is connected to, as shown here:

![Arduino IDE Interface](image)

**What is an Arduino sketch?**

An Arduino program that is developed using the IDE is called a **sketch**. Sketches are coded in Arduino language, which is based on a custom version of C/C++. Once you are done with writing the code in the built-in text editor, you can save it using the .ino extension. When you save these sketch files, the IDE automatically creates a folder to store them. If you are using any other supporting files for a sketch, such as header files or library files, they are all stored at this location (which is also called a **sketchbook**).
To open a new sketchbook, open the Arduino IDE and select **New** from the **File** menu, as shown in the following screenshot:

You will be prompted with an empty text editor. The text editor supports standard features (that is, copy/paste, select, find/replace, and so on). Before we go ahead with an Arduino program, let's explore the other tools provided by the IDE.

The Arduino IDE version prior to 1.0 used the **.pde** extension to save sketchbooks. Starting from 1.0, they are saved with the **.ino** extension. You can still open files with the **.pde** extension in the latest IDE. Later, the IDE will convert it to the **.ino** extension when you save them.
Working with libraries

The Arduino IDE uses libraries to extend the functionalities of existing sketches. Libraries are a set of functions combined to perform tasks around a specific component or concept. The majority of the built-in Arduino libraries provide methods to start working with external hardware components. You can import any library by navigating to Sketch | Import Library..., as shown in the following screenshot:

You can also use a library for your sketch by just specifying the library with the #include statement at the beginning of the sketch, that is, #include <Wire.h>.

The Arduino IDE also provides the capability to add an external library that supports a specific hardware or provides additional features. In the upcoming chapters, we will be dealing with some of these external libraries, and we will go through the process of importing them at that time.

Using Arduino examples

The Arduino IDE contains a large number of built-in example sketches. These examples are designed to get the user familiar with basic Arduino concepts and built-in Arduino libraries. The examples are well maintained by the Arduino community since they have comprehensive support for each example through the Arduino website (http://arduino.cc/en/Tutorial/HomePage). In the Arduino IDE, you can access these examples by navigating to File | Examples, as shown in the following screenshot:

Let's start with a simple in-built example. Open the Blink example by navigating to File | Examples | 01.Basics | Blink. The IDE will open a new window containing code that is similar to the code in the following program:

```c
/*
Blink
Turns on an LED on for one second, then off for one second, repeatedly.

This example code is in the public domain.
*/

// Pin 13 has an LED connected on most Arduino boards.
// give it a name:
```
```c
int led = 13;

// the setup routine runs once when you press reset:
void setup() {
    // initialize the digital pin as an output.
    pinMode(led, OUTPUT);
}

// the loop routine runs over and over again forever:
void loop() {
    digitalWrite(led, HIGH);   // turn the LED on (HIGH is the voltage level)
    delay(1000);               // wait for a second
    digitalWrite(led, LOW);    // turn the LED off by making the voltage LOW
    delay(1000);               // wait for a second
}
```

This Arduino sketch is designed to blink an LED on digital pin 13. You must be wondering why we didn't discuss or ask you to bring any hardware. That's because the Arduino Uno board is equipped with an on-board LED that is connected to digital pin 13. Now, instead of diving deeper into the Arduino code, we are going to focus on the process of dealing with the Arduino board through the IDE.

**Compiling and uploading sketches**

Once you have your code opened in the IDE, the first thing you need to do is to select the type of Arduino board on which you are going to upload your sketch. The Arduino IDE needs to know the type of board in order to compile the program for the appropriate microcontroller, as different Arduino boards can have different Atmel microcontrollers. Therefore, you need to perform this step before you go ahead with the compiling or uploading of the program to the board.
You can select the Arduino board by navigating to **Tools | Board**, as displayed in the following screenshot:

Select Arduino Uno from the list of boards, unless you are using a different Arduino board. Once you have selected the board, you can go ahead and compile the sketch. You can compile the sketch by navigating to **Sketch | Verify / Compile** from the menu bar or by using the keyboard shortcut Ctrl + R. If everything is set up well, you should be able to compile the code without any error.
After successfully compiling the sketch, it is time to upload the compiled code to the Arduino board. To do this, you need to make sure that your Arduino IDE is properly connected to your computer. If it is not already connected, connect your Arduino board to your computer using a USB port. Now, it is time to let your IDE know the serial port on which the board is connected. Navigate to Tools | Serial Ports and select the appropriate serial port.

In the case of some Linux distributions, you may not be able to see or upload the Arduino program to the board due to permission restriction(s) on the serial port. Running the following command on the terminal should solve that problem:

```
$ sudo usermod -a -G uucp, dialout, lock <username>
```

You can now upload the compiled sketch to your Arduino board by navigating to File | Upload. This process will use the serial connection to burn the compiled firmware in the microcontroller. Please wait for some time or until the LEDs (Tx and Rx LEDs) on the board stop flashing. Now, you have your Arduino board ready with your first sketch. You can observe the performance of the blinking LED near digital pin 13.

### Using the Serial Monitor window

In the previous process, we used a Universal Serial Bus (USB) cable to connect your Arduino board to a USB port of your computer. The USB port is an industrial standard to provide an interface for connecting various electronic components to a computer using the serial interface. When you connect an Arduino board using USB, the computer actually interfaces it as a serial peripheral device. Throughout the book, we are going to refer to the connections made using a USB as serial connections. The **Serial Monitor** window is a built-in utility of the Arduino IDE. The **Serial Monitor** window can be accessed by navigating to Tools | Serial Monitor or by using the Ctrl + Shift + M keyboard shortcut. It can be configured to observe data that is being sent or received on the serial port that is used to connect the Arduino board to the computer. You can also set the baud rate for the serial communication using the drop-down menu option. This utility is going to be very useful (further on in the book) when testing your prototypes and their performances.
Introduction to Arduino programming

The Arduino platform was introduced to simplify electronic hardware prototyping for everyone. For this reason, Arduino programming was intended to be easy to learn by nonprogrammers such as designers, artists, and students. The Arduino language is implemented in C/C++, while the fundamentals of the sketch and program structures are derived from an open source programming language called Processing and an open source electronic prototyping language called Wiring.

Comments

Arduino follows a commenting format that is adopted from C and it is similar to higher-level languages; however, it is different from the Python comment format that we learned earlier in this chapter. There are various methods of commenting, which are as follows:

- **Block comment**: This is done by covering the commented text between /* and */:

  ```
  /* This is a comment.
  * Arduino will ignore any text till it finds until the ending comment syntax, which is,
  */
  ``

- **Single-line or inline comment**: This is done by using // before the line:

  ```
  // This syntax only applies to one line.
  // You have to use it again for each next line of comment.
  int pin = 13;     //Selected pin 13
  ``

Usually, a block comment at the beginning of the sketch is mostly used to describe the program as a whole. Single-line comments are used to describe specific functions or to-do notes, such as the following one:

  ```
  //TODO: explain variables next.
  ```

Variables

Like any other high-level language, a variable is used to store data with three components: a name, a value, and a type. For example, consider the following statement:

```
int pin = 10;
```
Here, `pin` is the variable name that is defined with the type `int` and holds the value `10`. Later in the code, all occurrences of the `pin` variable will retrieve data from the declaration that we just made here. You can use any combination of alpha-numeric characters to select the variable name as long as the first character is not a number.

**Constants**

In the Arduino language, constants are predefined variables that are used to simplify the program:

- `HIGH`, `LOW`: While working with digital pins on the Arduino board, only two distinct voltage stages are possible at these pins. If a pin is being used to obtain an input, any measure above 3V is considered a `HIGH` state. If you are using a pin for output, then the `HIGH` state will set the pin voltage to 5V. The opposite voltage levels are considered as `LOW` states.

- `false`, `true`: These are used to represent logical true and false levels. `false` is defined as 0 and `true` is mostly defined as 1.

- `INPUT`, `OUTPUT`: These constants are used to define the roles of the Arduino pins. If you set the mode of an Arduino pin as `INPUT`, the Arduino program will prepare the pin to read sensors. Similarly, the `OUTPUT` setting prepares the pins to provide a sufficient amount of current to the connected sensors.

We will utilize these constants later in the book and we will also explain them with example code.

**Data types**

The declaration of each custom variable requires the user to specify the data type that is associated with the variable. The Arduino language uses a standard set of data types that are used in the C language. A list of these data types and their descriptions are as follows:

- `void`: This is used in the function declaration to indicate that the function is not going to return any value:

  ```
  void setup() {
  // actions
  }
  ```
• boolean: Variables defined with the data type boolean can only hold one of
two values, true or false:
boolean ledState = false;

• byte: This is used to store an 8-bit unsigned number, which is basically any
number from 0 to 255:
byte b = 0xFF;

• int: This is short for integers. It stores 16-bit (Arduino Uno) or 32-bit
(Arduino Due) numbers and it is one of the primary number storage data
types for the Arduino language. Although int will be used to declare
numbers throughout the book, the Arduino language also has long and
short number data types for special cases:
int varInt = 2147483647;
long varLong = varInt;
short varShort = -32768;

• float: This data type is used for numbers with decimal points. These are
also known as floating-point numbers. float is one of the more widely used
data types along with int to represent numbers in the Arduino language:
float varFloat = 1.111;

• char: This data type stores a character value and occupies 1 byte of memory.
When providing a value to char data types, character literals are declared
with single quotes:
char myCharacter = 'P';

• array: An array stores a collection of variables that is accessible by an
index number. If you are familiar with arrays in C/C++, it will be easier for
you to get started, as the Arduino language uses the same C/C++ arrays. The
following are some of the methods to initialize an array:
int myIntArray[] = {1, 2, 3, 4, 5};
int tempValues[5] = { 32, 55, 72, 75};
char msgArray[10] = "hello!";

An array can be accessed using an index number (where the index starts
from number 0):
myIntArray[0] == 1
msgArray[2] == 'e'
Conversions

Conversion functions are used to convert any data type value into the provided data types. The Arduino language implements the following conversion functions that can be utilized during programming:

- `char()`: This converts the value of any data type to the character data type
- `byte()`: This converts the value of any data type to the byte data type
- `int()`: This converts the value of any data type to the integer data type
- `float()`: This converts the value of any data type to the floating-point number data type

As a demonstration of using these functions, check out the following example:

```plaintext
int myInt = 10;
float myfloat = float(myInt);
```

Implementation of the preceding code will create a floating-point variable, `myFloat`, with value 10.0 using the integer value initialized by the `myInt` variable.

Functions and statements

Functions, also called subroutines or procedures, are a piece of code implemented to do specific tasks. The Arduino language has some predefined functions and the user can also write custom functions to implement certain program logic. These custom functions can then be called from any part of the sketch to perform a specific task. Functions help programmers to simplify debugging, to reduce chances for error, and to organize coding concepts:

```plaintext
void blinkLED()
{
    // action A;
    // action B;
}
```

The Arduino language has a set of library functions to simplify the programming experience. Although not all of these library functions are required by an Arduino sketch, `setup()` and `loop()` are mandatory functions and they are required to successfully compile the sketch.
The setup() function

When Arduino runs a sketch, it first looks for the setup() function. The setup() function is used to execute important programming subroutines before the rest of the program, such as declaring constants, setting up pins, initializing serial communication, or initializing external libraries. When Arduino runs the program, it executes the setup() functions only once. If you check out the Blink sketch that we used in the previous section, you can see the initialization of the setup() function, as displayed in the following code snippet:

```cpp
void setup() {
    // initialize the digital pin as an output.
    pinMode(led, OUTPUT);
}
```

As you can see in our example, we used the pinMode() function to assign the role of the LED pin in the setup() function.

The loop() function

Once Arduino has executed the setup() function, it starts iterating the loop() function continuously. While setup() contains the initialization parameters, loop() contains the logical parameters of your program:

```cpp
void loop() {
    digitalWrite(led, HIGH);
    delay(1000);
    digitalWrite(led, LOW);
    delay(1000);
}
```

As you can see in the preceding code snippet from the Blink sketch, the loop() function executes the main code that blinks the LED and repeats the process iteratively.

The pinMode() function

The pinMode() function is used to set the behavior of Arduino. As we saw in the setup() function of the Blink sketch, the pinMode() function configures the LED pin for OUTPUT:

```cpp
pinMode(led, OUTPUT)
```

Here, the led variable is assigned to digital pin 13, whose mode will be changed by the pinMode() function.
Working with pins

Once you are done configuring the pins that will be used by your program, you also need help in reading the input from these pins or for sending signals to them. Arduino provides a few specific functions to handle these scenarios:

- **digitalWrite()**: This was developed for digital I/O pins. This function sets the pin to HIGH (5V) or LOW (0V), which are already configured as OUTPUT using pinMode(). For example, the following line of code sets digital pin 13 to HIGH:

  ```c
  digitalWrite(13, HIGH);
  ```

- **digitalRead()**: Similar to digitalWrite(), this function helps you to read the state of a digital pin that is configured as INPUT:

  ```c
  value = digitalRead(13);
  ```

- **analogRead()**: This function reads the value from a specific analog pin. The value is linearly mapped between the integer value of 0 and 1023 to represent the voltage from 0V to 5V:

  ```c
  value = analogRead(0);
  ```

- **analogWrite()**: This function is used to provide analog output results at a digital pin. The technique is called PWM, and this will be explained in Chapter 4, Diving into Python-Arduino Prototyping. It is still important to note that this function is not designed for all digital pins, but it is only for pins that are designated as PWM pins.

Statements

If you are familiar with any other object-oriented programming language, you must have used statements extensively for your programs. The Arduino language uses traditional C/C++ statements such as if/else, while, switch/case, and for to control the flow of your program. Instead of diving deep into these statements right now, they are described later in the book with practical examples.
Summary

Alright! You have successfully completed the comparatively mundane tasks of installing and configuring Python and the Arduino IDE. Your system, whether it is a Mac OS X, Linux, or Windows system, is now ready for the upcoming chapters. In this chapter, we went through the history and building blocks of Arduino. We also learned the basics of Python programming and the Arduino language. Now, you are ready to get your hands on real hardware and start exploring computer to hardware interfacing. In the next chapter, we will go through the first step of interfacing, that is, connecting Arduino to the computer using a serial interface.
Where to buy this book

You can buy Python Programming for Arduino from the Packt Publishing website.
Alternatively, you can buy the book from Amazon, BN.com, Computer Manuals and most internet book retailers.
Click here for ordering and shipping details.