Chapter 1

Ruby Basics

At a Glance

Instructor's Manual Table of Contents

Lecture Notes

Overview

Ruby is an object-oriented programming language originally developed in 1993 to run on UNIX. However, it has since been ported to many other popular operating systems, including Microsoft Windows, Mac OS X, and Linux. Ruby is distributed under an open-source license, allowing anyone to freely install and use it. In this chapter, the student will learn background information required to begin working with Ruby and to use it to create and execute Ruby scripts.

Chapter Objectives

- Get a brief history of computer programming
- Get an introduction to Ruby
- Get ready to work with Ruby
- Use Ruby interactively
- Develop Ruby programs
- Create the Ruby Joke game

Teaching Tips

Project Preview: The Ruby Joke Game

1. This section provides an overview of the chapter's Ruby development project, the Ruby Joke Game. Refer to the Instruction PowerPoint for Chapter 1 for overview slides.

A Brief History of Computers and Computer Programming

Make sure students understand the following critical facts:

- Computer systems have gone through a series of distinct changes over the last 70 years. These changes can be classified into different generations.
- Each generation marks a major jump in technology over the previous generation.
- Computer programming, often referred to simply as programming, is the process of developing, testing, debugging, and updating the code statements that make up computer programs.
- Computer programs tell the computer what to do.
- Like computer systems, computer programming has evolved significantly over the years.

The Mechanized Era

Highlight the following historical facts:

- The art and science of programming goes even further back than the creation of the first computer.
- Joseph Marie Jacquard developed the Jacquard loom in 1801. The loom was used to weave cloth. Its operation was managed by instructions provided on punch cards.
- Charles Babbage was an English mathematician and inventor. Many historians credit him with originating the first conceptual computer. In 1822, he began work on a machine that he called the Difference Engine, which was designed to process polynomial functions.
- He later began worked on a second machine, which he called the Difference Engine No. 2.
- Charles Babbage later used punch cards as the basis for programming a third machine, which he called the Analytical Engine. This device introduced a number of new programming features that would prove to be essential elements in modern programming, including sequential processing, loops, and branching logic.
- A mathematician named Ada Lovelace learned of Babbage's Analytical Engine. Her fascination with the device, which was never finished, led her to write extensively about it and to theorize on its capabilities and possible use. She went on to write programs for the Analytical Engine. Because of her work, Ada Lovelace is regarded as the world's first programmer.

The First Generation

Highlight the following historical facts:

- 1937 to 1953 is regarded as being the first generation of electronic computer systems.
- The British created Colossus in 1943 for breaking German military code in WWII.
- The U.S. created Electronic Numerical Integrator and Computer (ENIAC) for computing ballistics in WWII and later used it to perform calculations as part of the development of the hydrogen bomb.
- The first computer programs were all written in machine language (machine code).
- Machine language is regarded as a first generation computer programming languages.

The Second Generation

Highlight the following historical facts:

- 1954 to 1962 is regarded as being the second generation of electronic computer systems.
- In the early 1950s, assembly languages evolved, replacing 0s and 1s with a set of symbols or mnemonics.
- Programmers translated their assembly programs into executable programs using an assembler.

- In the mid-to-late 1950s, a new set of higher-level programming languages began to supplant assembly language application development.
  - Fortran
  - Cobol
  - ALGOB

The Third Generation

Highlight the following historical facts:

- 1967 to 1972 is regarded as being the third generation of electronic computer systems.
- Innovations included an increase in computer processing power and speed and the development of the computer operating system.
- Pascal was created as a teaching language. It combined many of the language features found in Fortran, ALGOL, and COBOL.
- C is a general-purpose programming language developed in 1972 at Bell Telephone Laboratories.
- C has been modified (ported) to work on every major computing platform.

The Fourth Generation

Highlight the following historical facts:

- 1972 to 1983 is regarded as being the fourth generation of electronic computer systems.
- Computers were no longer building or room size, and the age of personal computing began.
- A new way of programming known as object-oriented programming began to gain notoriety.
- Rather than concentrating on the development of procedures, object-oriented programming (OOP) focuses on the definition of objects within programs.
- Bell Labs developed a new programming language in 1979 named C with Classes, later changing its name to C++ in 1983.

The Fifth Generation

Highlight the following historical facts:

- 1984 to 1990 is regarded as being the fifth generation of electronic computer systems.
- Advances occurred in local area networking (LAN) and wide area networking (WAN). Processor and RAM costs dropped, allowing for the widespread deployment of desktop computers and network servers.
- New programming languages were developed during this period. One such language was Perl. Perl is a general-purpose, interpreted scripting language.
- Using Perl, programmers develop programs referred to as scripts.

1990 and Beyond

Highlight the following historical facts:

- Network bandwidth, most notably broadband technologies, has allowed the Internet to evolve into a major media for communication, commerce, education, and entertainment.
- A new generation of programming languages designed to enhance and leverage these new technologies has evolved.
- A major driving force in recent years has been a movement toward rapid application development (RAD), enabling programmers to develop new applications faster and with higher quality. Programming languages like Visual Basic, C#, and Java emerged to fill this need.
- Another major advance that has taken root during this period has been the rise of scripting languages, including Python, Java, Lua, JavaScript, PHP, and Ruby.

Introducing Ruby

1. Address the origins of Ruby and the programming languages that influenced its development.
2. Provide a high-level overview of Ruby's capabilities.
3. Address Ruby on Rails and explain its impact on Ruby's success and popularity.

Teaching Tip	Instruct students to visit www.ruby-lang.org and spend time getting comfortable with this Web site.

Ruby Is Interpreted

1. Explain how Ruby, as an interpreted language, is easy to work with and yet is also a powerful language capable of developing complete applications.
2. Explain that Ruby is interpreted, supports a natural English-like programming style, and has light syntax requirements.

Ruby Is Simple Yet Powerful

1. Ensure that students understand that Ruby is interpreted.
2. Stress that natural English-like programming style is supported.
3. Inform student that Ruby has light syntax requirements compared to other programming languages.

Ruby Is Object Oriented

1. Explain the term object-oriented programming.
2. Define the term class and contrast it to an object.
3. Define the terms property and method.

Ruby Is Extremely Flexible

1. Provide examples of Ruby's usage, including:
   - Processing text files
   - Network programming
   - Application prototyping
   - System administration
   - Web development

Ruby Exists in Many Different Environments

1. Explain that Ruby supports cross-platform development on many different operating systems, including:
   - Microsoft Windows
   - Mac OS X
   - Linux
   - UNIX

2. Explain that Ruby can also be run in various other virtual machine environments, including:
   - JRuby
   - IronRuby

Getting Ready to Work with Ruby

1. Explain that Ruby is often installed along with the operating system, and if Ruby is not installed, it is easy to download and install.

Determining Whether Ruby Is Already Installed

1. Explain that there are a number of different ways to determine whether or not Ruby is installed on a computer.
2. Explain that on Windows, the `ruby -v` command can be executed.
3. Note that the `ruby -v` command also works on Mac OS X, as does the irb shell.
4. Note that the `irb` command also works on Linux and UNIX.

Installing or Upgrading Ruby

1. Provide the student with the URLs where Ruby can be downloaded.
2. Provide an overview of the steps involved in installing Ruby on Windows, Mac OS X, Linux, and UNIX.

Teaching Tip	Students can download Ruby from the following sites: Windows: www.ruby-lang.org/en/downloads/ Mac OS X: http://locomotive.raaum.org/, www.macports.org/ www.finkproject.org/ Linux/UNIX: www.ruby-lang.org/en/downloads/

Working with Ruby

1. Reinforce that the `irb` is a Ruby shell and that it provides a command-line interface that facilitates interaction with Ruby.

Working at the Command Prompt

1. Demonstrate how to access the command prompt.
2. Explain that there are differences in command prompt usage on different operating systems.

IRB – Interactive Ruby

1. Demonstrate how to start an IRB session, and demonstrate the execution of different Ruby commands.
2. Explain that the IRB command prompt consists of multiple parts, and break those parts down.
3. Explain the concept of *nil*.

FXRI – Interactive Ruby Help and Console

1. Introduce the fxri – Interactive Ruby Help and Console, and explain that it is only supported on Microsoft Windows.
2. Explain the different parts of the fxri console and what they are used for.

Developing Ruby Scripts

1. Explain the difference between executing interacting Ruby commands and programming.
2. List advantages provided by Ruby programming.

Creating Ruby Scripts on Microsoft Windows

1. Explain how to create a Ruby script on Microsoft Windows.
2. Explain the need for a code editor and discuss available options.

Creating Ruby Scripts on Mac OS X

1. Identify TextEdit as a possible Ruby code editor and identify a number of other third-party editor options.

Creating Ruby Scripts on Linux and UNIX

1. Identify the vi editor as an option for creating and editing Ruby scripts and explain the challenge of working with vi.
2. Identify other code editor options.

Using a Cross-Platform Ruby Editor

1. Introduce FreeRide and explain that it can be used to perform cross-platform Ruby development, eliminating the need to learn how to work with different editors.

Creating Your First Ruby Script

1. Demonstrate the creation of a simple Ruby script that says “Hello World!”.
2. Explain that Ruby scripts end with the .rb file extension.

Running Your Ruby Script

1. Demonstrate the execution of the HelloWorld.rb script.
2. Explain different options for executing the script from the command line.
3. Explain the benefits of adding the shebang statement of Ruby scripts that runs on Mac OS X, Linux, and UNIX, and demonstrate its application.

Back to the Ruby Joke Game

1. Review the basic operation of the Ruby Joke game and instruct students to focus on the overall steps involved in the creation and execution of the script.

Designing the Game

1. Guide students through the steps that document the development of the Ruby Joke game, making sure that they understand the overall sequence of events.
2. Introduce comments as a means of creating self-documenting scripts.

Running Your New Ruby Script Game

1. Demonstrate the execution of the Ruby Joke game, stepping through the execution of all of its jokes.