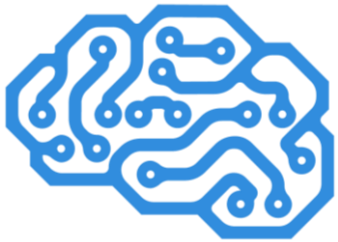
Mastering English in Software development

Loy Vanich (laploy@gmail.com 084 007 5544)

INTRODUCTION TO COURSE



INSTRUCTOR INFORMATION

Loy Vanich
084 007 5544
Line ID: laploy v
laploy@gmaill.com
www.laploy.com

TIME FRAME

- Course duration
- Start time
- Coffee break
- Lunch break
- Course end

REPOSITORY AND NOTE SHARE

Repository

github.com/laploy/MES

COURSE OUTLINE

- วิธีอ่านเท็กซ์บุ๊ค
- แนวบริโภคทิวโทเรียล
- เก่งคำศัพท์แบบด่วน
- เคล็ดลับการอ่านดาต้าชีต หนังสือคู่มือ

อธิบายเกี่ยวกับหลักสูตร

ผมได้รับการบอกเล่าถึงปัญหาหนักอกของนักพัฒนาซอฟต์แวร์ไทยที่ไม่สามารถพัฒนาตนเองให้ทันตามเทคโนโลยีได้อย่างรวดเร็ว สาเหตุเกิดจากการขาดแคลนตำราภาษาไทยที่ดีและตรงตามหัวข้อที่ต้องการ น่าเสียดายที่ทั้ง ๆ ที่ในอินเตอร์เน็ตมีแหล่งความรู้เหล่านี้อย่างอุดม ไม่ว่าจะเป็นตำราที่เป็น pdf ทิวโทเรียล (ทั้งแบบบทความและวิดีโอ) ดาต้าชีต สถาบันสอนออนไลน์ ฯลฯ แต่นักพัฒนาซอฟต์แวร์ไทยไม่สามาถตักตวงความรู้เหล่านี้ได้ เพราะอ่านและฟังภาษาอังกฤษไม่เข้าใจ

นักพัฒนาซอฟต์แวร์ไทยส่วนมากเรียนภาษาอังกฤษมาตั้งแต่ชั้นอนุบาลจนถึงระดับอุดมศึกษา แต่ไม่สามารถเข้าใจสื่อการสอนในเทคโนโลยีภาษาอังกฤษได้ ผมเองก็เคยเป็นหนึ่งในคนเหล่านั้น จนกระทั่งผมได้ค้นพบเคล็ดลับ การนำเคล็ดลับนี้มาประยุกต์ใช้งาน ทำให้ผมสามาถอ่านเท็กซ์บุ๊คภาษาอังกฤษได้เข้าใจดีกว่าภาษาไทย ดูทิวโทเรียลได้ และวิธีเข้าใจภาษาอังกฤษในทุกแง่มุมที่จำเป็นต่อการติดตามเทคโนโลยีที่เปลี่ยนทุกวันได้อย่างทันท่วงที ผมอยากเห็นนักพัฒนาซอฟต์แวร์ไทยคนอื่น ๆ ทำได้เช่นนี้บ้าง ผมจึงนำเคล็ดลับนี้มาเปิดเผยให้ท่านทราบในคอร์สนี้

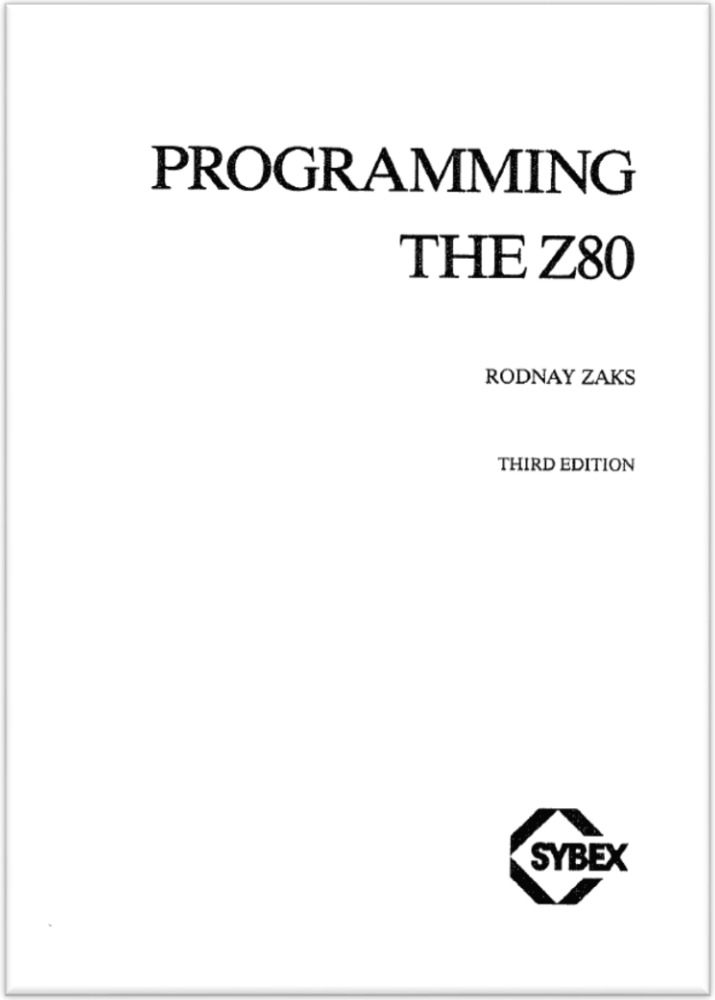
เหมาะสำหรับ

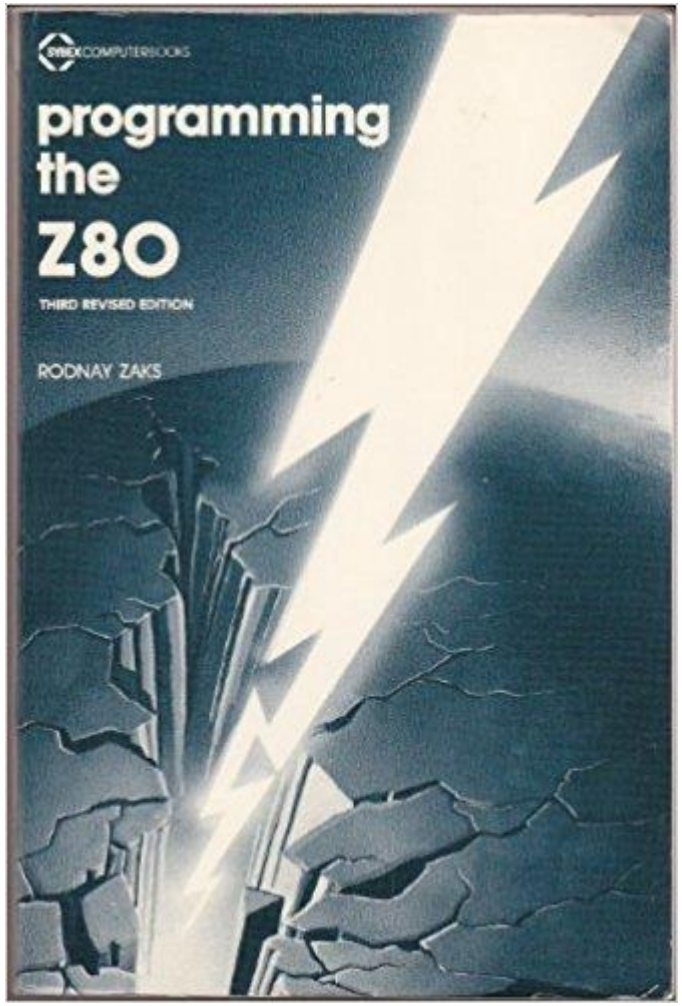
Software Developer

ความรู้เบื้องต้นที่ต้องมี

ความรู้พื้นฐานในภาษาอังกฤษ

วิธีอ่านเท็กซ์บุ๊ค


PROGRAMMING
THE Z80
RODNAY ZAKS
THIRD EDITION
SYBEX


SYBEX COMPUTER BOOKS
programming
the
Z80
THIRD REVISED EDITION
RODNAY ZAKS

1

BASIC CONCEPTS

INTRODUCTION

This chapter will introduce the basic concepts and definitions relating to computer programming. The reader already familiar with these concepts may want to glance quickly at the contents of this chapter and then move on to Chapter 2. It is suggested, however, that even the experienced reader look at the contents of this introductory chapter. Many significant concepts are presented here including, for example, two's complement, BCD, and other representations. Some of these concepts may be new to the reader; others may improve the knowledge and skills of experienced programmers.

WHAT IS PROGRAMMING?

Given a problem, one must first devise a solution. This solution, expressed as a step-by-step procedure, is called an *algorithm*. An algorithm is a step-by-step specification of the solution to a given problem. It must terminate in a finite number of steps. This algorithm may be expressed in any language or symbolism. A simple example of an algorithm is:

1—insert key in the keyhole
2—turn key one full turn to the left
3—seize doorknob
4—turn doorknob left and push the door

ADDRESSING TECHNIQUES

INTRODUCTION

This chapter will present the general theory of addressing and the various techniques which have been developed to facilitate the retrieval of data. In a second section, the specific addressing modes available in the Z80 will be reviewed, along with their advantages and limitations. Finally, in order to familiarize the reader with the various trade-offs possible, an applications section will demonstrate possible trade-offs between the various addressing techniques by studying specific application programs.

Because the Z80 has several 16-bit registers, in addition to the program counter, which can be used to specify an address, it is important that the Z80 user understand the various addressing modes, and in particular, the use of the index registers. Complex retrieval modes may be omitted at the beginning stage. However, all the addressing modes are useful in developing programs for this microprocessor. Let us now study the various alternatives available.

POSSIBLE ADDRESSING MODES

Addressing refers to the specification, within an instruction, of the location of the operand on which the instruction will operate. The main addressing methods will now be examined. They are all illustrated in Figure 5.1.

Implicit Addressing (or "Implied," or "Register")

Instructions which operate exclusively on registers normally use *implicit addressing*. This is illustrated in Figure 5.1. An implicit instruc-

tion derives its name from the fact that it does not specifically contain the address of the operand on which it operates. Instead, its opcode specifies one or more registers, usually the accumulator, or else any other register(s). Since internal registers are usually few in number (commonly eight), this will require a small number of bits. As an example, three bits within the instruction will point to one out of eight internal registers. Such instructions can, therefore, normally be encoded within eight bits. This is an important advantage, since an eight-bit instruction normally executes faster than any two- or three-byte instruction.

An example of an implicit instruction is:

LD A, B

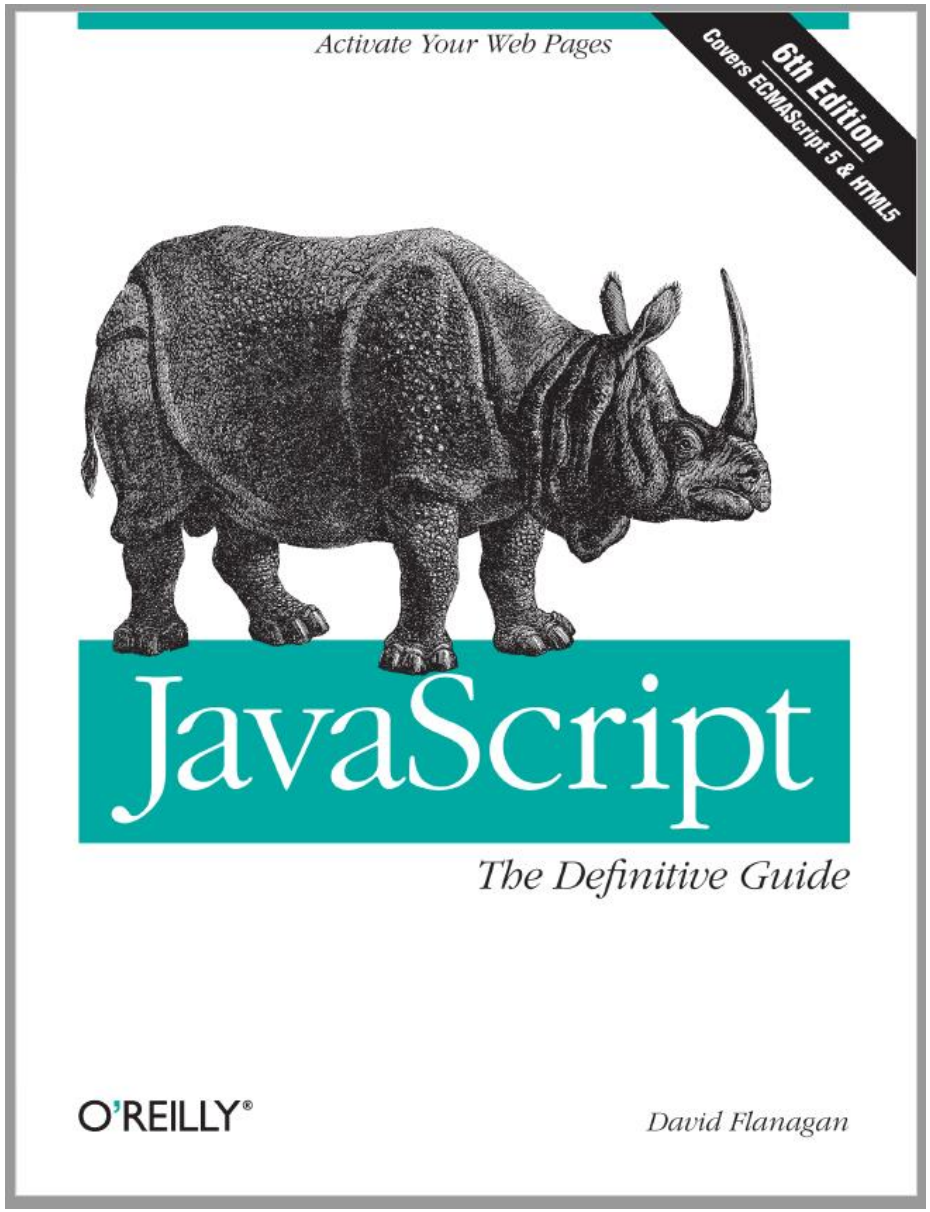
which specifies "transfer the contents of B into A" (Load A from B.)

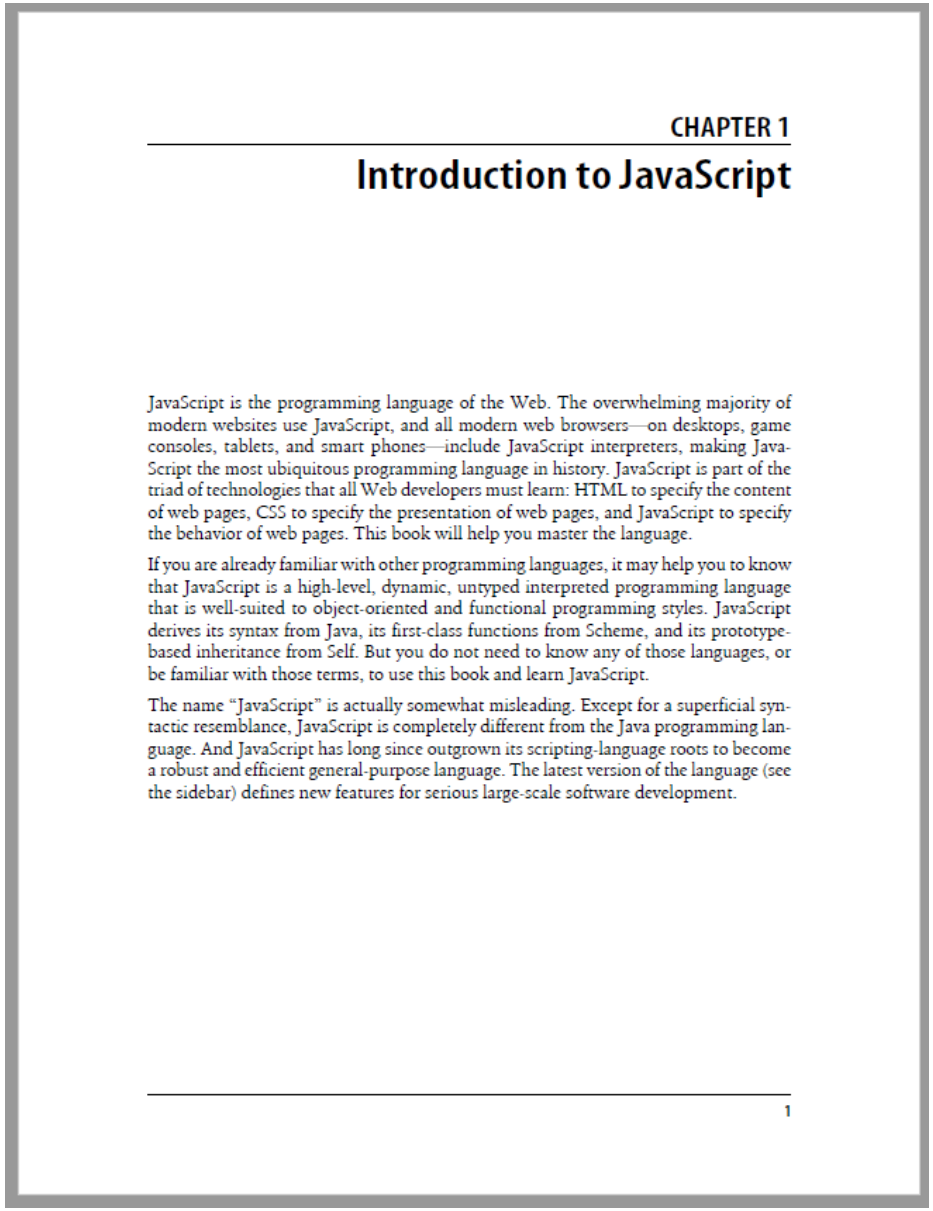
Immediate Addressing

Immediate addressing is illustrated in Figure 5.1. The eight-bit op-code is followed by an 8- or 16-bit literal (a constant). This type of instruction is needed, for example, to load an eight-bit value in an eight-bit register. Since the microprocessor is equipped with 16-bit registers, it may also be necessary to load 16-bit literals. An example of an immediate instruction is:

```
ADD A, 0H
```

The second word of this instruction contains the literal "0", which is added to the accumulator.


Activate Your Web Pages
6th Edition
Covers ECMAScript 5 & HTML5
JavaScript
The Definitive Guide
O'REILLY®
David Flanagan



CHAPTER 1

Introduction to JavaScript

JavaScript is the programming language of the Web. The overwhelming majority of modern websites use JavaScript, and all modern web browsers—on desktops, game consoles, tablets, and smart phones—include JavaScript interpreters, making JavaScript the most ubiquitous programming language in history. JavaScript is part of the triad of technologies that all Web developers must learn: HTML to specify the content of web pages, CSS to specify the presentation of web pages, and JavaScript to specify the behavior of web pages. This book will help you master the language.

If you are already familiar with other programming languages, it may help you to know that JavaScript is a high-level, dynamic, untyped interpreted programming language that is well-suited to object-oriented and functional programming styles. JavaScript derives its syntax from Java, its first-class functions from Scheme, and its prototype-based inheritance from Self. But you do not need to know any of those languages, or be familiar with those terms, to use this book and learn JavaScript.

The name "JavaScript" is actually somewhat misleading. Except for a superficial syntactic resemblance, JavaScript is completely different from the Java programming language. And JavaScript has long since outgrown its scripting-language roots to become a robust and efficient general-purpose language. The latest version of the language (see the sidebar) defines new features for serious large-scale software development.

1

JavaScript is the programming language of the Web. The overwhelming majority of modern websites use JavaScript, and all modern web browsers—on desktops, game consoles, tablets, and smart phones—include JavaScript interpreters, making JavaScript the most ubiquitous programming language in history. JavaScript is part of the triad of technologies that all Web developers must learn: HTML to specify the content of web pages, CSS to specify the presentation of web pages, and JavaScript to specify the behavior of web pages. This book will help you master the language.

If you are already familiar with other programming languages, it may help you to know that JavaScript is a high-level, dynamic, untyped interpreted programming language that is well-suited to object-oriented and functional programming styles. JavaScript derives its syntax from Java, its first-class functions from Scheme, and its prototype-based inheritance from Self. But you do not need to know any of those languages, or be familiar with those terms, to use this book and learn JavaScript.

If you are already familiar with other programming languages, it may help you to know that JavaScript is a high-level, dynamic, untyped interpreted programming language that is well-suited to object-oriented and functional programming styles. JavaScript derives its syntax from Java, its first-class functions from Scheme, and its prototype-based inheritance from Self. But you do not need to know any of those languages, or be familiar with those terms, to use this book and learn JavaScript.

The name “JavaScript” is actually somewhat misleading. Except for a superficial syntactic resemblance, JavaScript is completely different from the Java programming language. And JavaScript has long since outgrown its scripting-language roots to become a robust and efficient general-purpose language. The latest version of the language (see the sidebar) defines new features for serious large-scale software development.

CHAPTER 8

Functions

A *function* is a block of JavaScript code that is defined once but may be executed, or *invoked*, any number of times. You may already be familiar with the concept of a function under a name such as *subroutine* or *procedure*. JavaScript functions are *parameterized*: a function definition may include a list of identifiers, known as *parameters*, that work as local variables for the body of the function. Function invocations provide values, or *arguments*, for the function's parameters. Functions often use their argument values to compute a *return value* that becomes the value of the function-invocation expression. In addition to the arguments, each invocation has another value—the *invocation context*—that is the value of the this keyword.

If a function is assigned to the property of an object, it is known as a *method* of that object. When a function is invoked *on* or *through* an object, that object is the invocation context or this value for the function. Functions designed to initialize a newly created object are called *constructors*. Constructors were described in §6.1 and will be covered again in Chapter 9.

In JavaScript, functions are objects, and they can be manipulated by programs. JavaScript can assign functions to variables and pass them to other functions, for example. Since functions are objects, you can set properties on them, and even invoke methods on them.

JavaScript function definitions can be nested within other functions, and they have access to any variables that are in scope where they are defined. This means that JavaScript functions are *closures*, and it enables important and powerful programming techniques.

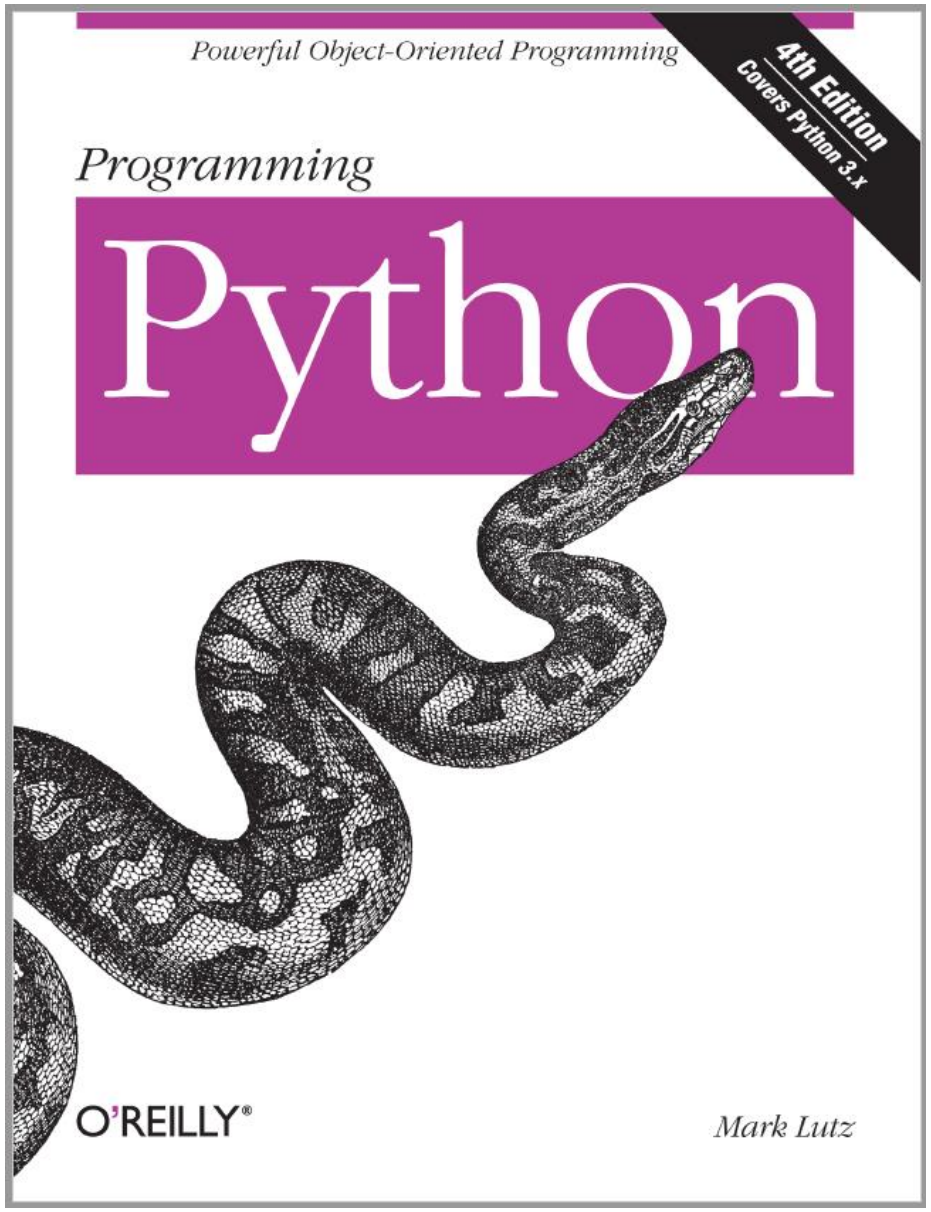
163

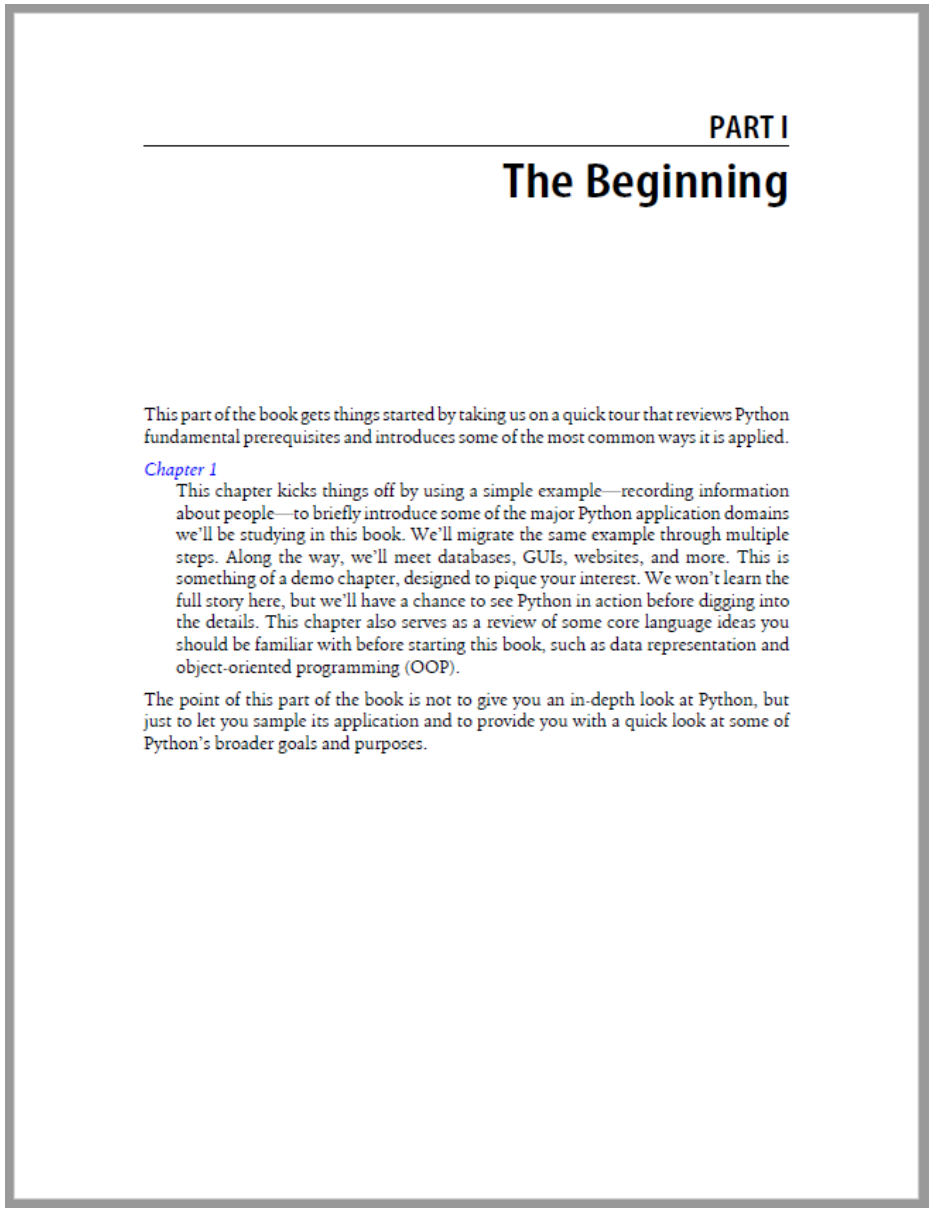
A *function* is a block of JavaScript code that is defined once but may be executed, or *invoked*, any number of times. You may already be familiar with the concept of a function under a name such as *subroutine* or *procedure*. JavaScript functions are *parameterized*: a function definition may include a list of identifiers, known as *parameters*, that work as local variables for the body of the function. Function invocations provide values, or *arguments*, for the function's parameters. Functions often use their argument values to compute a *return value* that becomes the value of the function-invocation expression. In addition to the arguments, each invocation has another value—the *invocation context*—that is the value of the `this` keyword.

If a function is assigned to the property of an object, it is known as a *method* of that object. When a function is invoked *on* or *through* an object, that object is the invocation context or `this` value for the function. Functions designed to initialize a newly created object are called *constructors*. Constructors were described in §6.1 and will be covered again in Chapter 9.

In JavaScript, functions are objects, and they can be manipulated by programs. JavaScript can assign functions to variables and pass them to other functions, for example. Since functions are objects, you can set properties on them, and even invoke methods on them.

JavaScript function definitions can be nested within other functions, and they have access to any variables that are in scope where they are defined. This means that JavaScript functions are *closures*, and it enables important and powerful programming techniques.


Powerful Object-Oriented Programming
4th Edition
Covers Python 3.x
Programming
Python
O'REILLY®
Mark Lutz



PART I

The Beginning

This part of the book gets things started by taking us on a quick tour that reviews Python fundamental prerequisites and introduces some of the most common ways it is applied.

Chapter 1

This chapter kicks things off by using a simple example—recording information about people—to briefly introduce some of the major Python application domains we'll be studying in this book. We'll migrate the same example through multiple steps. Along the way, we'll meet databases, GUIs, websites, and more. This is something of a demo chapter, designed to pique your interest. We won't learn the full story here, but we'll have a chance to see Python in action before digging into the details. This chapter also serves as a review of some core language ideas you should be familiar with before starting this book, such as data representation and object-oriented programming (OOP).

The point of this part of the book is not to give you an in-depth look at Python, but just to let you sample its application and to provide you with a quick look at some of Python's broader goals and purposes.

This part of the book gets things started by taking us on a quick tour that reviews Python fundamental prerequisites and introduces some of the most common ways it is applied.

Chapter 1

This chapter kicks things off by using a simple example—recording information about people—to briefly introduce some of the major Python application domains we'll be studying in this book. We'll migrate the same example through multiple steps. Along the way, we'll meet databases, GUIs, websites, and more. This is something of a demo chapter, designed to pique your interest. We won't learn the full story here, but we'll have a chance to see Python in action before digging into the details. This chapter also serves as a review of some core language ideas you should be familiar with before starting this book, such as data representation and object-oriented programming (OOP).

The point of this part of the book is not to give you an in-depth look at Python, but just to let you sample its application and to provide you with a quick look at some of Python's broader goals and purposes.

CHAPTER 12

Network Scripting

"Tune In, Log On, and Drop Out"

Over the 15 years since this book was first published, the Internet has virtually exploded onto the mainstream stage. It has rapidly grown from a simple communication device used primarily by academics and researchers into a medium that is now nearly as pervasive as the television and telephone. Social observers have likened the Internet's cultural impact to that of the printing press, and technical observers have suggested that all new software development of interest occurs only on the Internet. Naturally, time will be the final arbiter for such claims, but there is little doubt that the Internet is a major force in society and one of the main application contexts for modern software systems.

The Internet also happens to be one of the primary application domains for the Python programming language. In the decade and a half since the first edition of this book was written, the Internet's growth has strongly influenced Python's tool set and roles. Given Python and a computer with a socket-based Internet connection today, we can write Python scripts to read and send email around the world, fetch web pages from remote sites, transfer files by FTP, program interactive websites, parse HTML and XML files, and much more, simply by using the Internet modules that ship with Python as standard tools.

In fact, companies all over the world do: Google, YouTube, Walt Disney, Hewlett-Packard, JPL, and many others rely on Python's standard tools to power their websites. For example, the Google search engine—widely credited with making the Web usable—makes extensive use of Python code. The YouTube video server site is largely implemented in Python. And the BitTorrent peer-to-peer file transfer system—written in Python and downloaded by tens of millions of users—leverages Python's networking skills to share files among clients and remove some server bottlenecks.

Many also build and manage their sites with larger Python-based toolkits. For instance, the Zope web application server was an early entrant to the domain and is itself written and customizable in Python. Others build sites with the Plone content management

771

"Tune In, Log On, and Drop Out"

Over the 15 years since this book was first published, the Internet has virtually exploded onto the mainstream stage. It has rapidly grown from a simple communication device used primarily by academics and researchers into a medium that is now nearly as pervasive as the television and telephone. Social observers have likened the Internet's cultural impact to that of the printing press, and technical observers have suggested that all new software development of interest occurs only on the Internet. Naturally, time will be the final arbiter for such claims, but there is little doubt that the Internet is a major force in society and one of the main application contexts for modern software systems.

The Internet also happens to be one of the primary application domains for the Python programming language. In the decade and a half since the first edition of this book was written, the Internet's growth has strongly influenced Python's tool set and roles. Given Python and a computer with a socket-based Internet connection today, we can write Python scripts to read and send email around the world, fetch web pages from remote sites, transfer files by FTP, program interactive websites, parse HTML and XML files, and much more, simply by using the Internet modules that ship with Python as standard tools.

In fact, companies all over the world do: Google, YouTube, Walt Disney, Hewlett-Packard, JPL, and many others rely on Python's standard tools to power their websites. For example, the Google search engine—widely credited with making the Web usable—makes extensive use of Python code. The YouTube video server site is largely implemented in Python. And the BitTorrent peer-to-peer file transfer system—written in Python and downloaded by tens of millions of users—leverages Python's networking skills to share files among clients and remove some server bottlenecks.

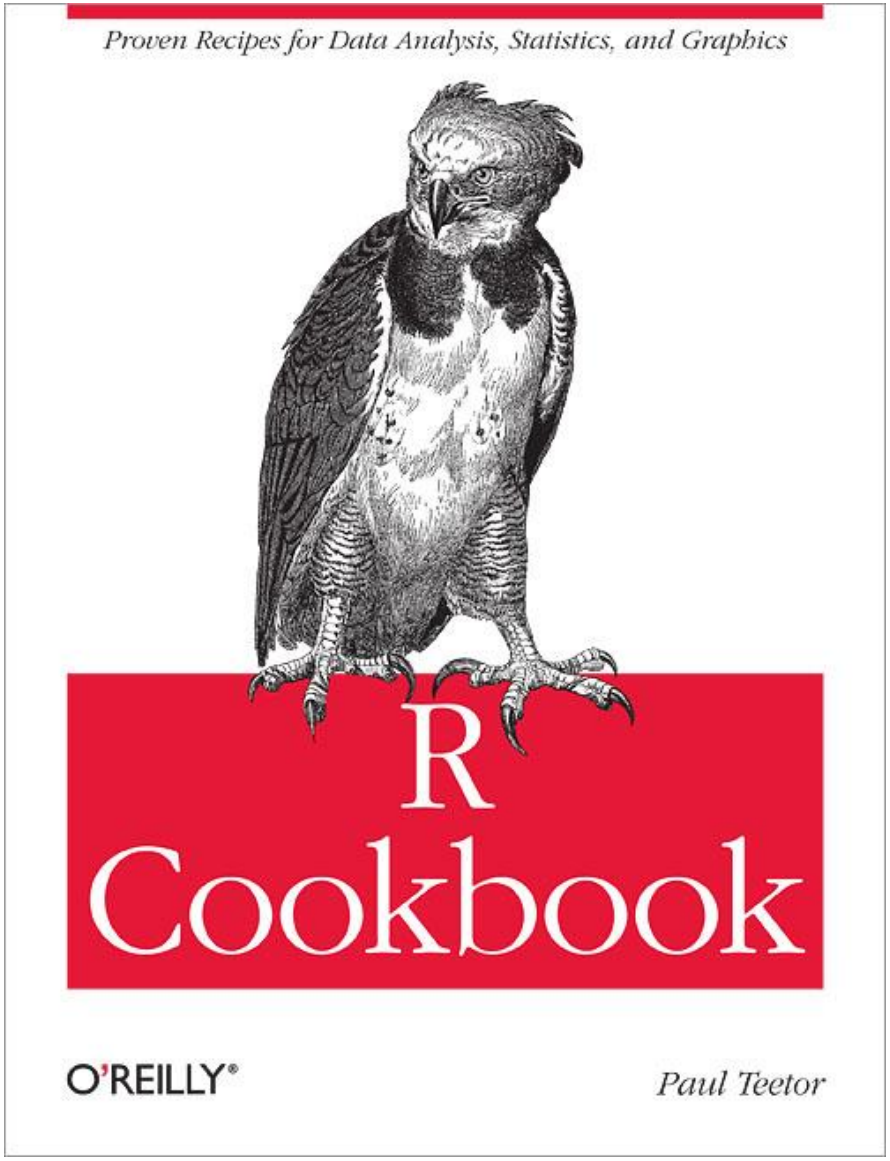
Socket Basics

Although we won't get into advanced socket use in this chapter, basic socket transfers are remarkably easy to code in Python. To create a connection between machines, Python programs import the `socket` module, create a socket object, and call the object's methods to establish connections and send and receive data.

Sockets are inherently bidirectional in nature, and socket object methods map directly to socket calls in the C library. For example, the script in Example 12-1 implements a program that simply listens for a connection on a socket and echoes back over a socket whatever it receives through that socket, adding `Echo=>` string prefixes.

Example 12-1. PP4E\Internet\Sockets\echo-server.py

```
"""
Server side: open a TCP/IP socket on a port, listen for a message from
a client, and send an echo reply; this is a simple one-shot listen/reply
conversation per client, but it goes into an infinite loop to listen for
more clients as long as this server script runs; the client may run on
a remote machine, or on same computer if it uses 'localhost' for server
"""

from socket import *                    # get socket constructor and constants
myHost = ''                             # '' = all available interfaces on host
myPort = 50007                          # listen on a non-reserved port number

sockobj = socket(AF_INET, SOCK_STREAM)      # make a TCP socket object
sockobj.bind((myHost, myPort))              # bind it to server port number
sockobj.listen(5)                           # listen, allow 5 pending connects

while True:                                 # listen until process killed
    connection, address = sockobj.accept()  # wait for next client connect
    print('Server connected by', address)   # connection is a new socket
    while True:
        data = connection.recv(1024)        # read next line on client socket
        if not data: break                  # send a reply line to the client
        connection.send(b'Echo=>' + data)   # until eof when socket closed
    connection.close()
```


Proven Recipes for Data Analysis, Statistics, and Graphics
R
Cookbook
O'REILLY
Paul Teetor

2.5 Creating a Vector

Problem

You want to create a vector.

Solution

Use the `c(...)` operator to construct a vector from given values.

Discussion

Vectors are a central component of R, not just another data structure. A vector can contain either numbers, strings, or logical values but not a mixture.

The `c(...)` operator can construct a vector from simple elements:

```
> c(1,1,2,3,5,8,13,21)
[1]  1  1  2  3  5  8 13 21
> c(1*pi, 2*pi, 3*pi, 4*pi)
[1]  3.141593  6.283185  9.424778 12.566371
> c("Everyone", "loves", "stats.")
[1] "Everyone" "loves"    "stats."
> c(TRUE,TRUE,FALSE,TRUE)
[1]  TRUE  TRUE FALSE  TRUE
```

If the arguments to `c(...)` are themselves vectors, it flattens them and combines them into one single vector:

```
> v1 <- c(1,2,3)
> v2 <- c(4,5,6)
> c(v1,v2)
[1] 1 2 3 4 5 6
```

Vectors cannot contain a mix of data types, such as numbers and strings. If you create a vector from mixed elements, R will try to accommodate you by converting one of them:

```
> v1 <- c(1,2,3)
> v3 <- c("A","B","C")
> c(v1,v3)
[1] "1" "2" "3" "A" "B" "C"
```

Here, the user tried to create a vector from both numbers and strings. R converted all the numbers to strings before creating the vector, thereby making the data elements compatible.

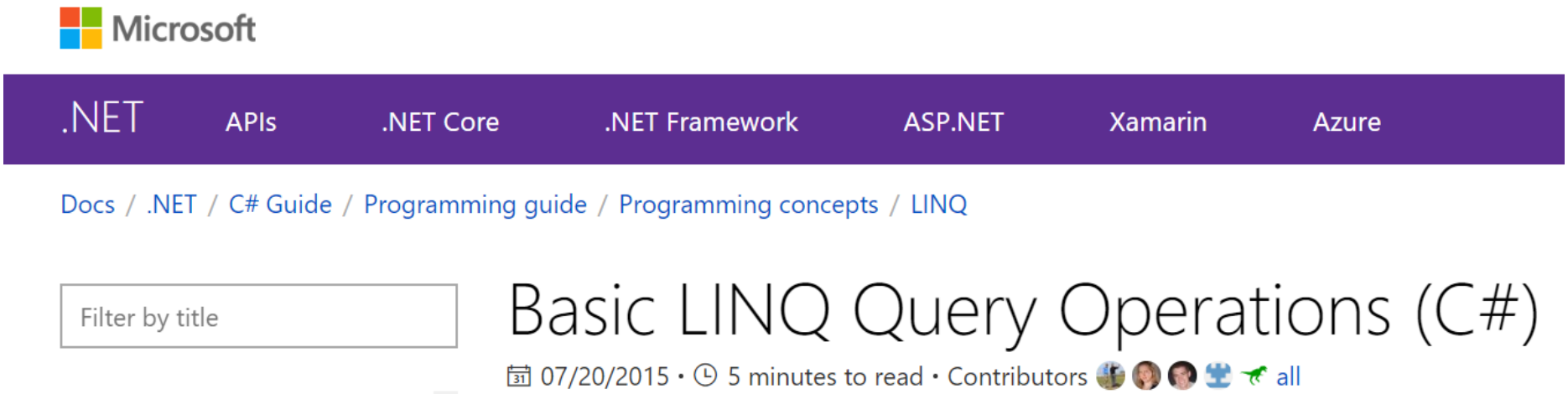
Technically speaking, two data elements can coexist in a vector only if they have the same *mode*. The modes of `3.1415` and `"foo"` are numeric and character, respectively:

```
> mode(3.1415)
[1] "numeric"
> mode("foo")
[1] "character"
```

Those modes are incompatible. To make a vector from them, R converts 3.1415 to character mode so it will be compatible with `"foo"`:

```
> c(3.1415, "foo")
[1] "3.1415" "foo"
> mode(c(3.1415, "foo"))
[1] "character"
```

แนวบริโภคทิวโทเรียล



https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/linq/basic-linq-query-operations

Basic LINQ Query Operations (C#)

07/20/2015 • 5 minutes to read • Contributors all

This topic gives a brief introduction to LINQ query expressions and some of the typical kinds of operations that you perform in a query. More detailed information is in the following topics:

LINQ Query Expressions

Standard Query Operators Overview (C#)

Walkthrough: Writing Queries in C#

> **Note**
>
> If you already are familiar with a query language such as SQL or XQuery, you can skip most of this topic. Read about the "`from` clause" in the next section to learn about the order of clauses in LINQ query expressions.

Obtaining a Data Source

In a LINQ query, the first step is to specify the data source. In C# as in most programming languages a variable must be declared before it can be used. In a LINQ query, the `from` clause comes first in order to introduce the data source (`customers`) and the *range variable* (`cust`).

C# Copy

```
//queryAllCustomers is an IEnumerable<Customer>
var queryAllCustomers = from cust in customers
                        select cust;
```

The range variable is like the iteration variable in a `foreach` loop except that no actual iteration occurs in a query expression. When the query is executed, the range variable will serve as a reference to each successive element in `customers`. Because the compiler can infer the type of `cust`, you do not have to specify it explicitly. Additional range variables can be introduced by a `let` clause. For more information, see let clause.

> ⓘ **Note**
>
> For non-generic data sources such as ArrayList, the range variable must be explicitly typed. For more information, see How to: Query an ArrayList with LINQ (C#) and from clause.

Filtering

Probably the most common query operation is to apply a filter in the form of a Boolean expression. The filter causes the query to return only those elements for which the expression is true. The result is produced by using the `where` clause. The filter in effect specifies which elements to exclude from the source sequence. In the following example, only those `customers` who have an address in London are returned.

C# Copy

```
var queryLondonCustomers = from cust in customers
                           where cust.City == "London"
                           select cust;
```

You can use the familiar C# logical `AND` and `OR` operators to apply as many filter expressions as necessary in the `where` clause. For example, to return only customers from "London" `AND` whose name is "Devon" you would write the following code:

C# Copy

```
where cust.City=="London" && cust.Name == "Devon"
```

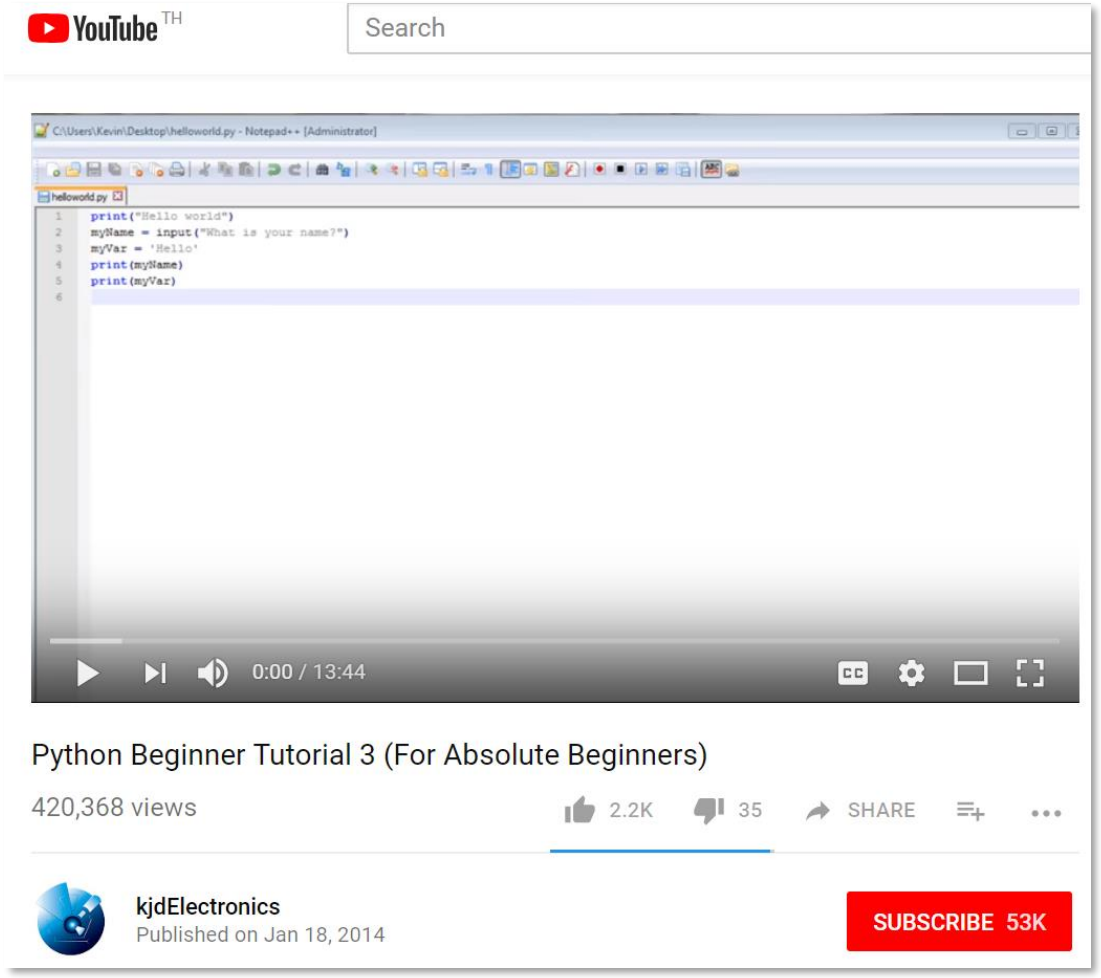
To return customers from London or Paris, you would write the following code:

C# Copy

```
where cust.City == "London" || cust.City == "Paris"
```

For more information, see where clause.

Python Beginner Tutorial 3 (For Absolute Beginners) kjdElectronics



dotConnect for MySQL Documentation

Entity Framework Tutorial

support@devart.com

This tutorial guides you through the process of creating a simple application powered by ADO.NET Entity Framework. In less than 5 minutes you will have a ready-to-use data access layer for your business objects.

Please note that this tutorial is not applicable for Entity Framework Core. It is intended for previous Entity Framework versions. You can find links to Entity Framework Core tutorials in the See Also section, in the end of this topic.

In this walkthrough:

https://www.devart.com/dotconnect/mysql/docs/Tutorial_EF.html

Introducing the ADO.NET Entity Framework

ADO.NET Entity Framework is an object-relational mapping (ORM) framework for the .NET Framework. It is designed to enable developers to create data access applications by programming against a conceptual application model instead of programming directly against a relational storage schema. The goal is to decrease the amount of code and maintenance required for data-oriented applications.

Requirements

In order to connect to MySQL server you need the server itself running, dotConnect for MySQL installed and IDE running. ADO.NET Entity Framework requires .NET Framework 3.5 Service Pack 1 or higher, Visual Studio 2008 Service Pack 1 or higher, and MySQL server 5.0 or higher. Note that Entity Framework support is available only in Professional and Developer Editions of dotConnect for MySQL.

For Entity Framework 6, you will also need a NuGet Visual Studio extension installed since it is used for adding EntityFramework NuGet package. Alternatively you may create model for Entity Framework v1 or v4, which don't require NuGet, in this tutorial.

In this tutorial it is assumed that you already have the database objects created. You have to execute a script from the following file if you have not done so yet:
\Program Files\Devart\dotConnect\MySQL\Samples\crm_demo.sql

In this sample we will create a simple console application. It could be any other project type as well, but for simplicity's sake we'll use console project throughout the tutorial. Start Visual Studio and create a new console application.

Entity Framework v6

The following actions are required if you want to create an Entity Framework v6 model.

Open the Package Manager Console window and execute the following command in it.

```
install-package EntityFramework
```

After this add the following line:

```
<provider invariantName="Devart.Data.MySql" type="Devart.Data.MySql.Entity.MySqlEntityProviderServices,
Devart.Data.MySql.Entity.EF6, Version=8.3.215.0, Culture=neutral, PublicKeyToken=09af7300eec23701" />
```

to the **entityFramework -> providers** section.

```
<entityFramework>
    <providers>
      <provider invariantName="Devart.Data.MySql" type="Devart.Data.MySql.Entity.MySqlEntityProviderServices,
      Devart.Data.MySql.Entity.EF6, Version=8.3.215.0, Culture=neutral, PublicKeyToken=09af7300eec23701" />
    </providers>
</entityFramework>
```

Note: replace 8.3.215.0 with the actual assembly version.

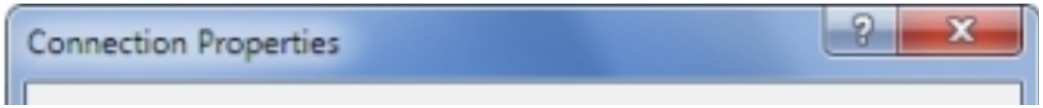
You also need to add the following assemblies to the project references:

- Devart.Data.dll
- Devart.Data.MySql.dll
- Devart.Data.MySql.Entity.EF6.dll

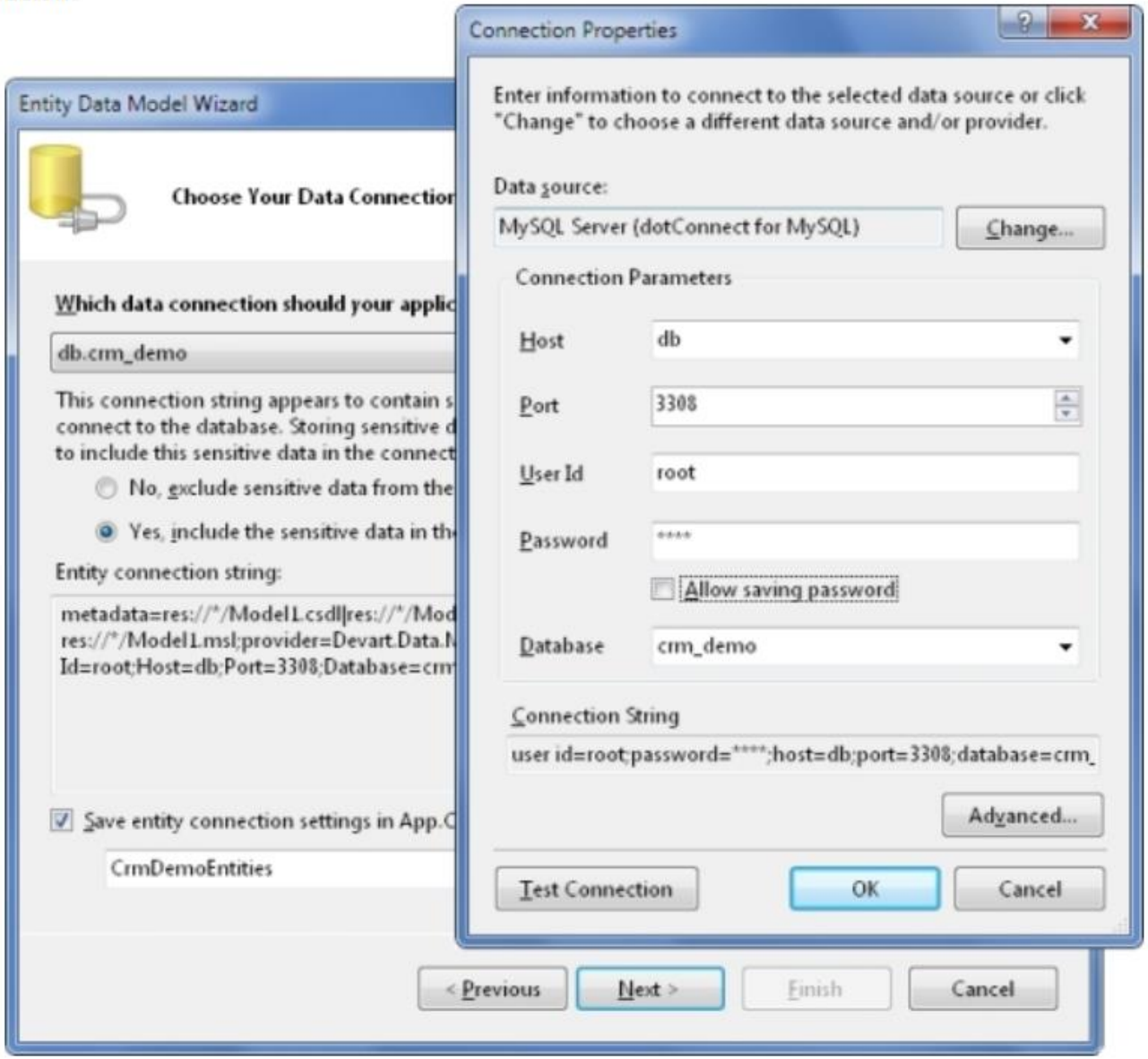
After this you need to rebuild the project before running the EDM wizard.

Generating Model from Database

1. In the Solution Explorer right-click on the project and choose **Add** | **New Item**.
2. In the dialog select **ADO.NET Entity Data Model**, click **Add**. This launches Entity Data Model Wizard.
3. In the wizard select **Generate from database**, click **Next**.
4. Pick an existing dotConnect for MySQL connection string or create a new one. When creating a new connection select **MySQL Server** in the *Data Source* list, then **dotConnect for MySQL** in the *Data provider* list. See the screenshot below.



below:


Entity Data Model Wizard
Choose Your Data Connection
Which data connection should your applic
db.crm_demo
This connection string appears to contain s
connect to the database. Storing sensitive d
to include this sensitive data in the connect
No, exclude sensitive data from the
Yes, include the sensitive data in th
Entity connection string:
metadata=res://*/Model1.csdl|res://*/Mod
res://*/Model1.msl;provider=Devart.Data.M
Id=root;Host=db;Port=3308;Database=crm
Save entity connection settings in App.C
CrmDemoEntities
< Previous
Next >
Finish
Cancel
Connection Properties
Enter information to connect to the selected data source or click "Change" to choose a different data source and/or provider.
Data source:
MySQL Server (dotConnect for MySQL)
Change...
Connection Parameters
Host
db
Port
3308
User Id
root
Password
****
Allow saving password
Database
crm_demo
Connection String
user id=root;password=****;host=db;port=3308;database=crm_
Advanced...
Test Connection
OK
Cancel

5. Agree to include the sensitive data in the connection string.
6. In the **Save entity connection settings...** box type *CrmDemoEntities*. This will be the name of the main data access class. Click **Next**.
7. Select database objects that will be used in the model. These are all objects from the crm_demo script, including auxiliary tables.

Querying Data

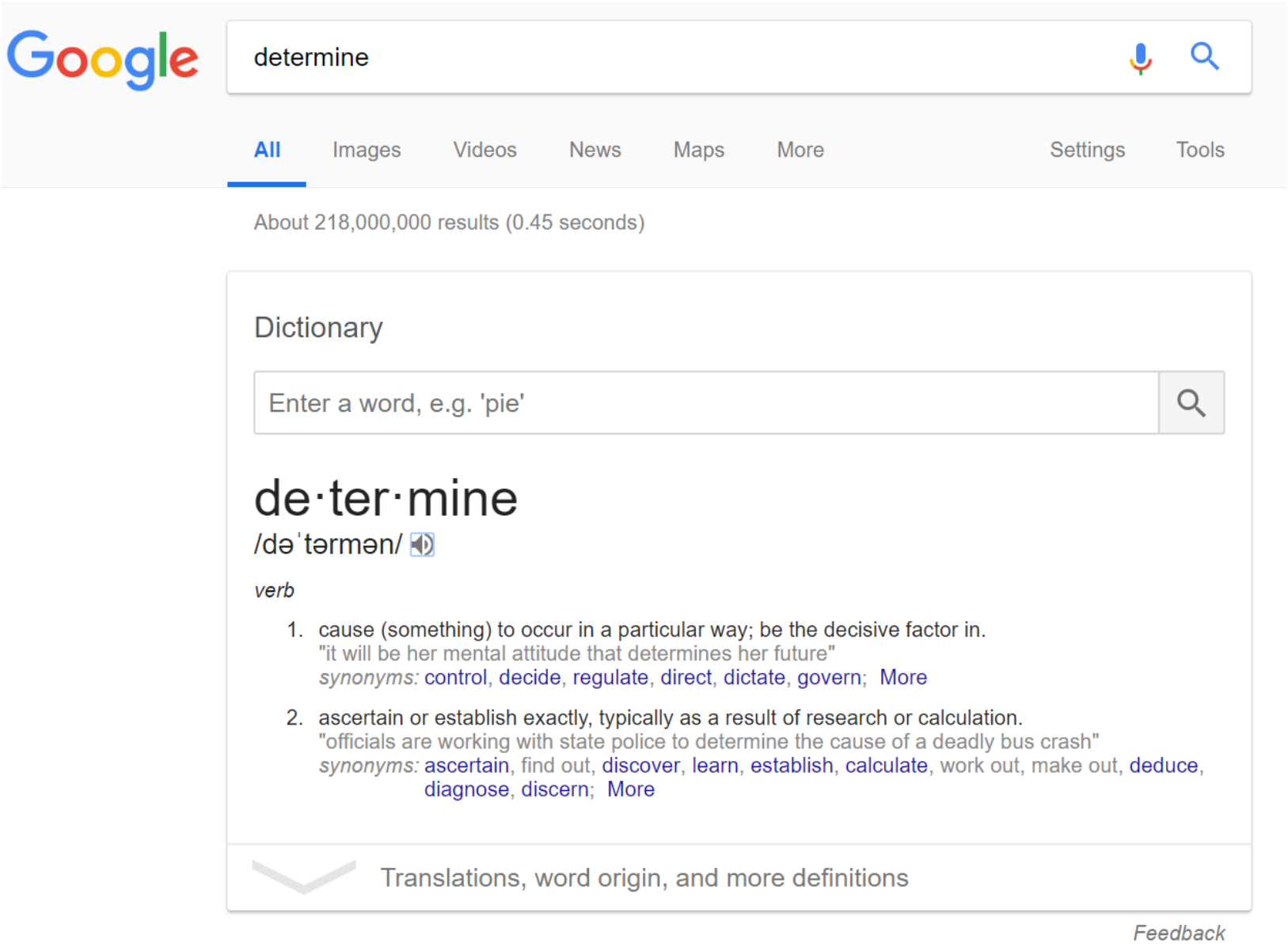
All Entity Framework operations are executed through the ObjectContext descendant, which is named *CrmDemoEntities* in this tutorial. To retrieve data you have to first create an instance of the context, then prepare a query with **LINQ to Entities** or **EntitySQL** or their mix, and then access the object returned by the query, which may be a collection of objects or a single object.

Let's read all the data from the table Company, sort it by CompanyID, and output some columns. Add the following block of code to the method Main:

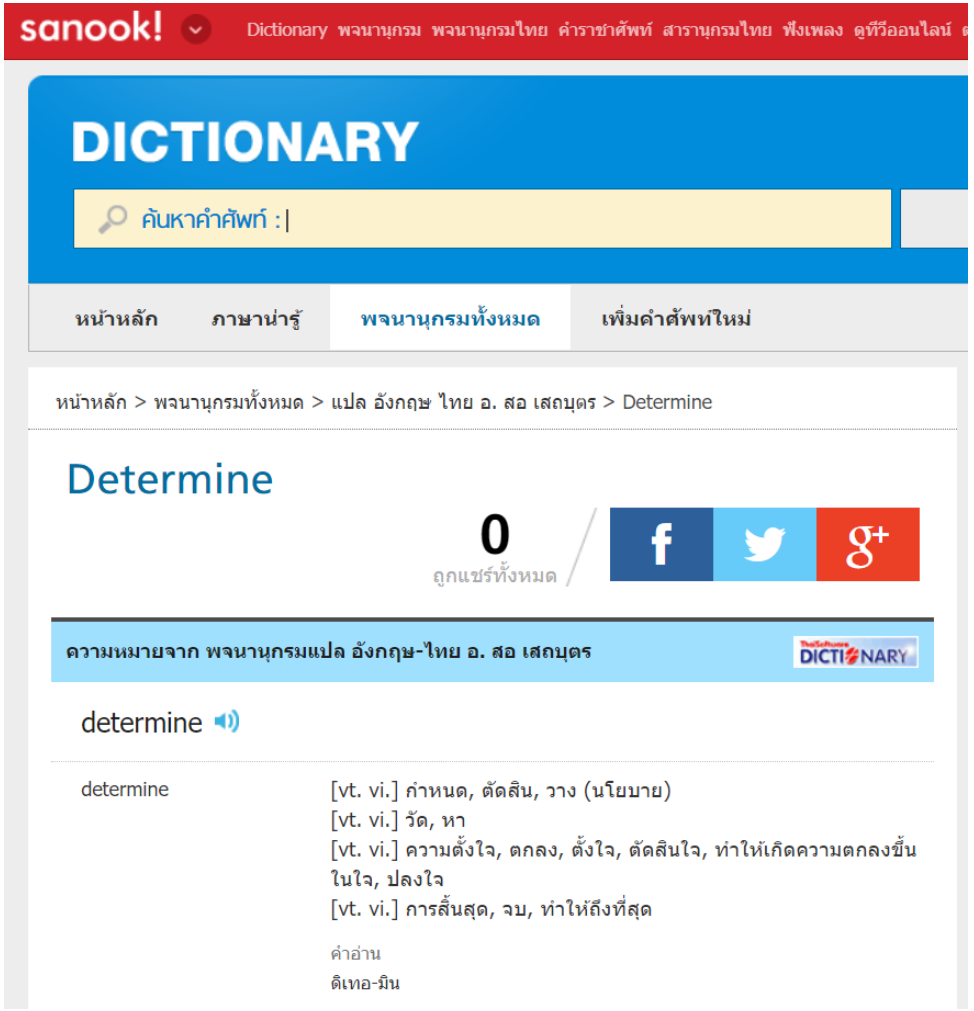
C# Copy Code

```
CrmDemoEntities context = new CrmDemoEntities();
var query = from it in context.company
            orderby it.CompanyID
            select it;

foreach (company comp in query)
  Console.WriteLine("{0} | {1} | {2}", comp.CompanyID, comp.CompanyName, comp.Country);

Console.ReadLine();
```

เก่งคำศัพท์แบบด่วน


Google
determine
All
Images
Videos
News
Maps
More
Settings
Tools
About 218,000,000 results (0.45 seconds)
Dictionary
Enter a word, e.g. 'pie'
de·ter·mine
/dəˈtərmən/
verb
1. cause (something) to occur in a particular way; be the decisive factor in.
"it will be her mental attitude that determines her future"
synonyms: control, decide, regulate, direct, dictate, govern; More
2. ascertain or establish exactly, typically as a result of research or calculation.
"officials are working with state police to determine the cause of a deadly bus crash"
synonyms: ascertain, find out, discover, learn, establish, calculate, work out, make out, deduce, diagnose, discern; More
Translations, word origin, and more definitions
Feedback

https://dictionary.sanook.com

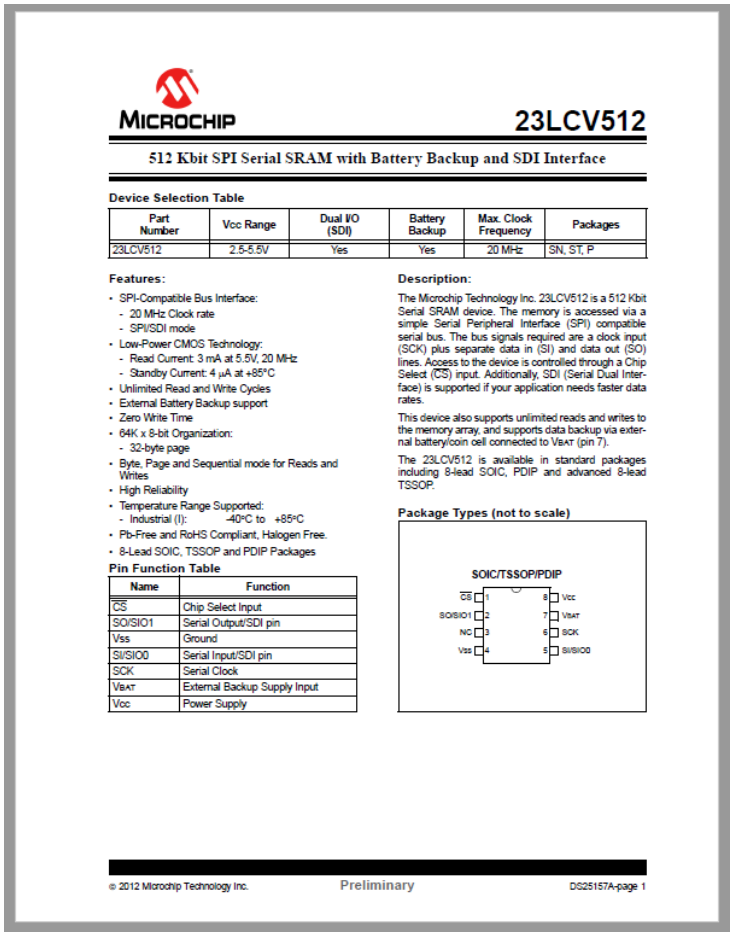


http://www.online-english-thai-dictionary.com

เคล็ดลับการอ่านดาต้าชีต

หนังสือคู่มือ

512 Kbit SPI Serial SRAM with Battery Backup and SDI Interface 25157A.pdf



MICROCHIP

23LCV512

512 Kbit SPI Serial SRAM with Battery Backup and SDI Interface

Device Selection Table

Part Number	Vcc Range	Dual I/O (SDI)	Battery Backup	Max. Clock Frequency	Packages
23LCV512	2.5-5.5V	Yes	Yes	20 MHz	SN, ST, P

Features:

- SPI-Compatible Bus Interface:
  - 20 MHz Clock rate
  - SPI/SDI mode
- Low-Power CMOS Technology:
  - Read Current: 3 mA at 5.5V, 20 MHz
  - Standby Current: 4 µA at +85°C
- Unlimited Read and Write Cycles
- External Battery Backup support
- Zero Write Time
- 64K x 8-bit Organization:
  - 32-byte page
- Byte, Page and Sequential mode for Reads and Writes
- High Reliability
- Temperature Range Supported:
  - Industrial (I): -40°C to +85°C
- Pb-Free and RoHS Compliant, Halogen Free.
- 8-Lead SOIC, TSSOP and PDIP Packages

Description:

The Microchip Technology Inc. 23LCV512 is a 512 Kbit Serial SRAM device. The memory is accessed via a simple Serial Peripheral Interface (SPI) compatible serial bus. The bus signals required are a clock input (SCK) plus separate data in (SI) and data out (SO) lines. Access to the device is controlled through a Chip Select ($\overline{CS}$) input. Additionally, SDI (Serial Dual Interface) is supported if your application needs faster data rates.

This device also supports unlimited reads and writes to the memory array, and supports data backup via external battery/coin cell connected to V_{BAT} (pin 7).

The 23LCV512 is available in standard packages including 8-lead SOIC, PDIP and advanced 8-lead TSSOP.

Pin Function Table

Name	Function
$\overline{CS}$	Chip Select Input
SO/SIO1	Serial Output/SDI pin
Vss	Ground
SI/SIO0	Serial Input/SDI pin
SCK	Serial Clock
V_{BAT}	External Backup Supply Input
Vcc	Power Supply

Package Types (not to scale)

SOIC/TSSOP/PDIP

 Preliminary

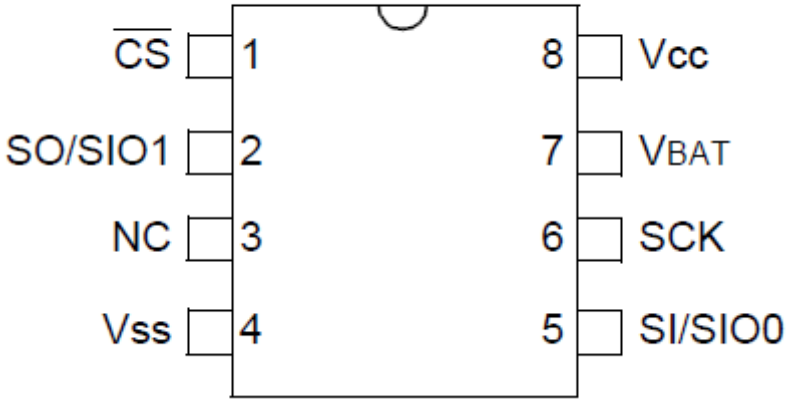
Description:

The Microchip Technology Inc. 23LCV512 is a 512 Kbit Serial SRAM device. The memory is accessed via a simple Serial Peripheral Interface (SPI) compatible serial bus. The bus signals required are a clock input (SCK) plus separate data in (SI) and data out (SO) lines. Access to the device is controlled through a Chip Select ($\overline{CS}$) input. Additionally, SDI (Serial Dual Interface) is supported if your application needs faster data rates.

This device also supports unlimited reads and writes to the memory array, and supports data backup via external battery/coin cell connected to V_{BAT} (pin 7).

The 23LCV512 is available in standard packages including 8-lead SOIC, PDIP and advanced 8-lead TSSOP.

SOIC/TSSOP/PDIP



2.1 Principles of Operation

The 23LCV512 is an 512 Kbit Serial SRAM designed to interface directly with the Serial Peripheral Interface (SPI) port of many of today's popular microcontroller families, including Microchip's PIC® microcontrollers. It may also interface with microcontrollers that do not have a built-in SPI port by using discrete I/O lines programmed properly in firmware to match the SPI protocol. In addition, the 23LCV512 is also capable of operating in SDI (or dual SPI) mode.

The 23LCV512 contains an 8-bit instruction register. The device is accessed via the SI pin, with data being clocked in on the rising edge of SCK. The $\overline{CS}$ pin must be low for the entire operation.

Table 2-1 contains a list of the possible instruction bytes and format for device operation. All instructions, addresses and data are transferred MSB first, LSB last.

TABLE 2-1: INSTRUCTION SET

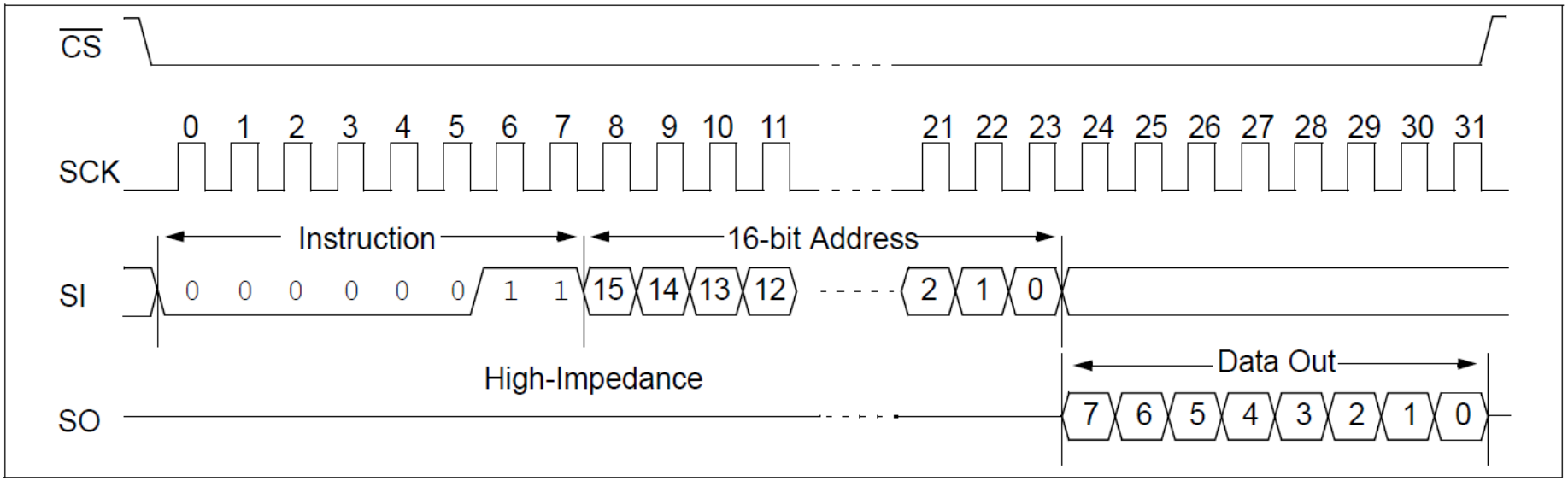
Instruction Name	Instruction Format	Hex Code	Description
READ	0000 0011	0x03	Read data from memory array beginning at selected address
WRITE	0000 0010	0x02	Write data to memory array beginning at selected address
EDIO	0011 1011	0x3B	Enter Dual I/O access
RSTIO	1111 1111	0xFF	Reset Dual I/O access
RDMR	0000 0101	0x05	Read Mode Register
WRMR	0000 0001	0x01	Write Mode Register

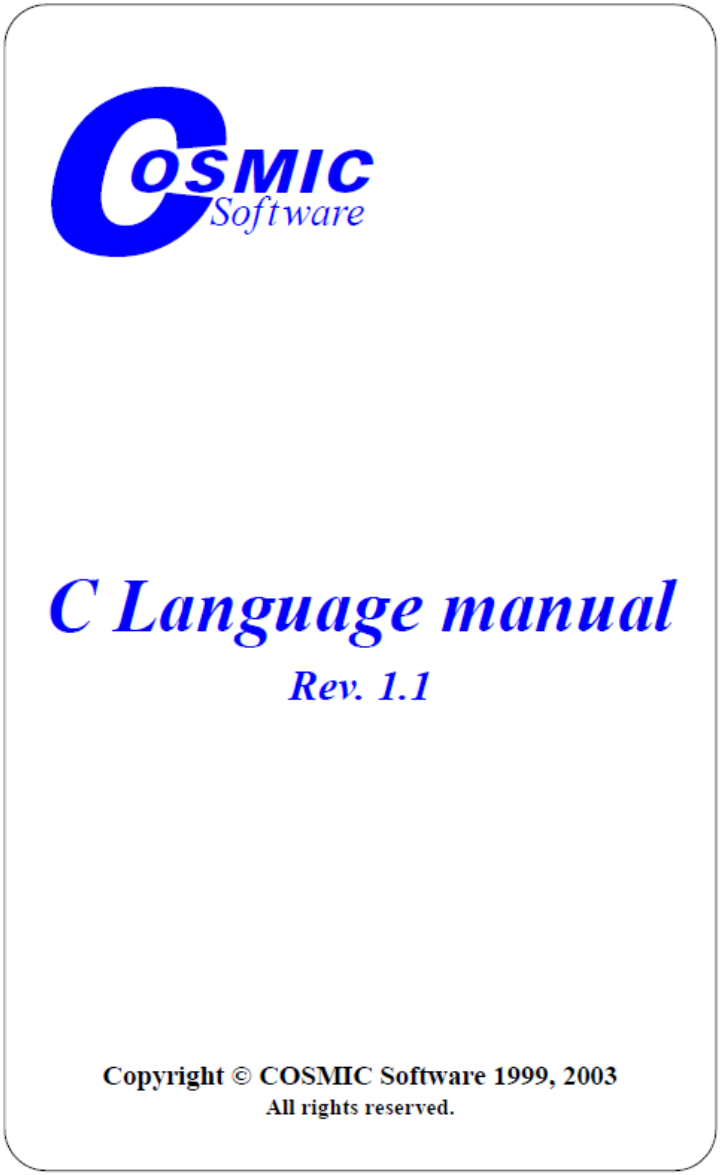
2.2 Modes of Operation

The 23LCV512 has three modes of operation that are selected by setting bits 7 and 6 in the MODE register. The modes of operation are Byte, Page and Burst.

Byte Operation – is selected when bits 7 and 6 in the MODE register are set to 00. In this mode, the read/write operations are limited to only one byte. The command followed by the 16-bit address is clocked into the device and the data to/from the device is transferred on the next eight clocks (Figure 2-1, Figure 2-2).

FIGURE 2-1: BYTE READ SEQUENCE (SPI MODE)




COSMIC
Software
C Language manual
Rev. 1.1
Copyright © COSMIC Software 1999, 2003
All rights reserved.

Declarations

An object **declaration** in C follows the global format:

<*class*> <*type*> <*name*> <*initialization*> ;

This global format is shared by all the C objects including functions. Each field differs depending on the object type.

<*class*> is the **storage class**, and gives information about how the object is allocated and where it is known and then accessible.

<*type*> gives the basic **type** of the object and is generally completed by the name information.

<*name*> gives a **name** to the object and is generally an identifier. It may be followed or preceded by additional information to declare complex objects.

<initialization> gives an **initial value** to the object. Depending on how the object is allocated, it may be static information to be built by the linker, or it may be some executable code to set up the variable when it is created. This field is optional and may be omitted.

Each declaration is terminated by the semicolon character `;`. To be more convenient, a declaration may contain several occurrences of the pair *<name>* *<initialization>*, separated by commas. Each variable shares the same *storage class* and the same basic *type*.

Integers

An **integer** variable is declared using the following basic types:

```
char
short or short int
int
long or long int
```

These basic types may be prefixed by the keyword **signed** or **unsigned**. The type **unsigned int** may be shortened by writing only **unsigned**. Types *short*, *int* and *long* are *signed* by default. Type *char* is defaulted to either signed *char* or *unsigned char* depending on the target processor. For most of the small microprocessors, a plain *char* is defaulted to **unsigned char**.

A **real** variable is declared using the following basic types:

```
float
double
long double
```

In most of the cases, type **long double** is equivalent to type **double**. For small microprocessors, all real types are mapped to type **float**.

For these numerical variables, an initialization is written by an equal sign = followed by a constant. Here are some examples:

```
char c;
short val = 1;
int a, b;
unsigned long l1 = 0, l2 = 3;
```

Bits

A **bit** variable is declared with the following basic type:

```
_Bool
```

These variables can be initialised like a numerical type, but the assignment rules do not match the integer rules as described in the next chapter. Here are some examples:

```
_Bool ready;
_Bool led = 0;
```

It is not possible to declare a pointer to a bit variable, nor an array of bit variables.

Pointers

A **pointer** is declared with two parameters. The first one indicates that the variable is a pointer, and the second is the type of the pointed object. In order to match the global syntax, the <*type*> field is used to declare the type of the pointed object. The fact that the variable is a pointer will be indicated by prefixing the variable name with the character *. Beware that declaring a pointer does not allocate memory for the pointed object, but only for the pointer itself. The initialization field is written as for a numerical variable, but the constant is an address constant and not a numerical constant, except for the numerical value zero which is conventionally representing the **NULL** pointer. Here are some examples:

```
char *pc;       /* pointer to a char */
int *pv = &a;   /* initialized to address of a */
short *ps = 0; /* initialized to NULL pointer */
```

A pointer can be declared with the special type **void**. Such a pointer is handled by the compiler as a plain pointer regarding the assignement operations, but the object pointed at by the pointer cannot be accessed directly. Such a syntax is interesting when a pointer has to share different types.

Arrays

An **array** is declared with three parameters. The first one indicates that the variable is an array, the second indicates how many elements are in the array and the third is the type of one element. In order to match the global syntax, the <*type*> field is used to declare the type of one element. The fact that the variable is an array and its dimension will be indicated by adding the dimension written between square brackets `[10]` after the name. The dimension is an integer constant which may be omitted in some cases. An array initialization will be written by an equal sign = followed by a list of values placed between curly braces, and separated by commas. Here are some examples:

```
char tab[10];   /* array of 10 chars */
int values[3] = {1, 2, 3};
```

An initialization list may be smaller than the dimension, but never larger. If the list is smaller than the specified dimension, the missing elements are filled with zeroes. If an initialization list is specified, the dimension may be omitted. In that case, the compiler gives the array the same length than the specified list:

```
int values[]={1, 2, 3};/* array of 3 elements */
```

Variables

Variables are simply expressed by their name, which is a C identifier which should have been previously defined, otherwise the compiler does not know the type of that variable and does not know how to access it. The only exception is that you can call a function which has not been declared. The compiler will define that unknown function as an external function returning an **int**. If the function is declared later in the same file and the actual return type does not match, or if strict checking options are used (**-pp**, **+strict**), the compiler will complain.