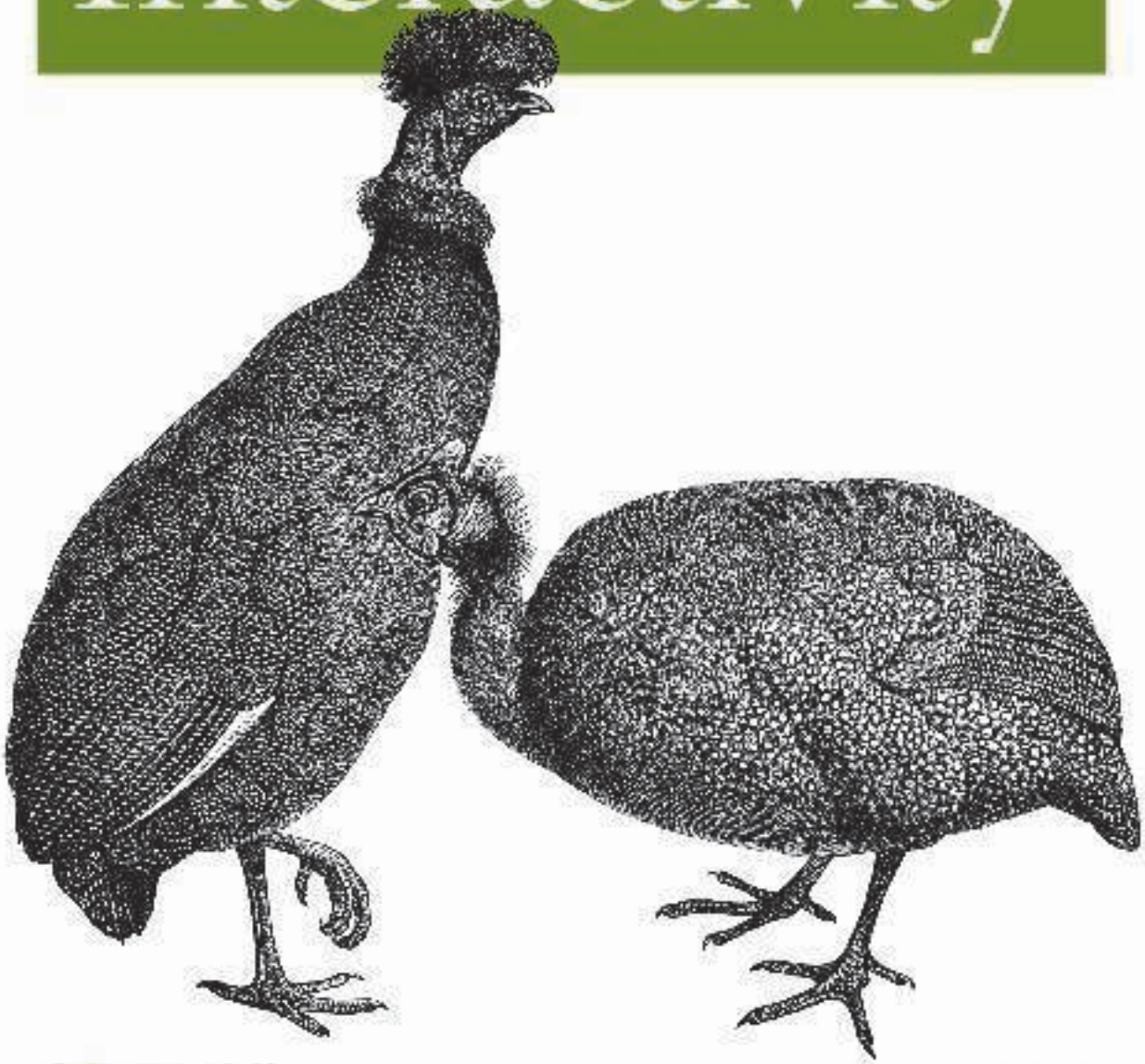
A Designer's Guide to Processing, Arduino, and openFrameworks

2nd Edition

Programming

Interactivity



O'REILLY®

Joshua Noble

Programming Interactivity

Ready to create rich interactive experiences with your artwork, designs, or prototypes? This is the ideal place to start. With this hands-on guide, you'll explore several themes in interactive art and design—including 3-D graphics, sound, physical interaction, computer vision, and geolocation—and learn the basic programming and electronics concepts you need to implement them. No previous experience necessary.

You'll get a complete introduction to three free tools created specifically for artists and designers: the Processing programming language, the Arduino microcontroller, and the openFrameworks toolkit. You'll also find working code samples you can use right away, along with the background and technical information you need to design, program, and build your own projects.

TOPICS INCLUDE:

- Learn cutting edge techniques for interaction design from leading artists and designers
- Let users provide input through buttons, dials, and other physical controls
- Produce graphics and animation, including 3-D images with OpenGL
- Use sounds to interact with users by providing feedback, input, or an element they can control
- Work with motors, servos, and appliances to provide physical feedback
- Turn a user's gestures and movements into meaningful input, using OpenCV

"This is a wonderful book that helped me fill in the gaps between what I knew a lot about (hardware and I/O) and what I knew very little of (Processing and graphics)."

—**Nathan Seidle**
CEO, Sparkfun Electronics

"This book is a great way for anyone to get started with the fundamentals of programming and electronics; beginners can start making cool projects right away."

—**Mark Frauenfelder**
editor-in-chief, MAKE

Joshua Noble, an interaction designer and developer who works extensively with tools discussed in this book, shares his knowledge in workshops throughout the U.S. He is the lead author of the *Flex 3 Cookbook* (O'Reilly) as well as the first edition of *Programming Interactivity*.

US $49.99 CAN $52.99
ISBN: 978-1-449-31144-5

Twitter: @oreillymedia
facebook.com/oreilly

O'REILLY®
oreilly.com

SECOND EDITION

Programming Interactivity

Joshua Noble

O'REILLY®
Beijing • Cambridge • Farnham • Köln • Sebastopol • Tokyo

Programming Interactivity, Second Edition
by Joshua Noble

Copyright © 2012 Joshua Noble. All rights reserved.
Printed in the United States of America.

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (*http://my.safaribooksonline.com*). For more information, contact our corporate/institutional sales department: (800) 998-9938 or *corporate@oreilly.com*.

Editors: Shawn Wallace and Brian Jepson
Production Editor: Melanie Yarbrough
Proofreader: Kiel Van Horn

Indexer: Ellen Troutman Zaig
Cover Designer: Karen Montgomery
Interior Designer: David Futato
Illustrator: Robert Romano

July 2009: First Edition.
January 2012: Second Edition.

Revision History for the Second Edition:
2012-01-10 First release
See *http://oreilly.com/catalog/errata.csp?isbn=9781449311445* for release details.

Nutshell Handbook, the Nutshell Handbook logo, and the O'Reilly logo are registered trademarks of O'Reilly Media, Inc. *Programming Interactivity, Second Edition*, the cover image of guinea fowl, and related trade dress are trademarks of O'Reilly Media, Inc.

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and O'Reilly Media, Inc., was aware of a trademark claim, the designations have been printed in caps or initial caps.

While every precaution has been taken in the preparation of this book, the publisher and authors assume no responsibility for errors or omissions, or for damages resulting from the use of the information contained herein.

ISBN: 978-1-449-31144-5

[LSI]

1326301522

Table of Contents

Preface

This book is broken into three parts. The first introduces the three projects that will be used throughout this book, the second introduces some of the most common themes in creating interaction in designs and applications, and the third introduces some of the more advanced topics that you may want to explore further. Also included with some of the chapters are interviews with programmers, artists, designers, and authors who work with the tools covered in this book. Covering such a massive range of topics means that this book doesn't go into great depth about most of them, but it is filled with references to other books, websites, designers, and artists that you may find helpful or inspiring.

What Is—and Isn't—in This Book

My excitement about the ideas and rapid growth of the field of interaction design is hard to contain. However, as exciting and far-reaching as interaction design is, the limitations of time and physical book size dictate that I be selective about what is and isn't covered in this book.

What's In

This book covers Processing, Arduino, and openFrameworks. To help novice programmers, it covers some of the core elements of programming in C and C++ for Arduino and openFrameworks and also covers the Processing language. We introduce dozens of libraries for openFrameworks and Processing—too many to list here. Some of these are official libraries or add-ons for the two frameworks, and some are simply extensions that have been created for this book or provided by altruistic coders.

We also introduce some of the basics of electronics and how computer hardware functions, as well as many tools and components that you can use with an Arduino. The Arduino and Processing IDEs are covered, as are two different IDEs for openFrameworks, namely, Code::Blocks, and Xcode. The Arduino Uno and Mini are covered in depth, and we discuss other boards only briefly. We cover many electronic

components that have designed expressly for the Arduino, called *shields*, in depth as well.

What's Not In

While this book shows how to create some circuits, it doesn't cover a great deal of the fundamentals of electronics or hardware, how to create circuits, or electronics theory. Chapter 17 lists some excellent tutorials and references. While the book does cover the Processing subset of the Java programming language, to conserve space and maintain focus, it doesn't cover Java. The book doesn't cover many aspects of C++, such as templates, inline functions, operator overloading, and abstract classes. Again, though, listed in Chapter 17 are several excellent resources that you can use to learn about these deeper topics in C++.

There are so many Arduino-compatible boards now that it's almost impossible to cover them all in depth; the book mentions the Mega, the Nano, Fio, and several other boards only in passing and leaves out many of the Arduino-compatible boards that are not created by the Arduino team. Quite a few components and other tools that we would have liked to discuss in depth could not be included to maintain scope and to save space.

Many topics that we would have liked to include have been left out because of space considerations: artificial intelligence, data visualization, and algorithmic music, among others. Though these are all potentially interesting areas for artists and designers, the focus of the book is on teaching some of the theory and techniques for interaction design as well as the basics of hardware and programming. The resources listed at the end of the book can provide the names of some materials that might help you explore these topics.

Conventions Used in This Book

The following typographical conventions are used in this book:

Italic
: Indicates new terms, URLs, email addresses, filenames, and file extensions.

`Constant width`
: Used for program listings, as well as within paragraphs to refer to program elements such as variable or function names, databases, data types, environment variables, statements, and keywords.

`Constant width bold`
: Shows commands or other text that should be typed literally by the user.

`Constant width italic`
: Shows text that should be replaced with user-supplied values or by values determined by context.

This icon signifies a tip, suggestion, or general note.

This icon indicates a warning or caution.

Companion Website

All the code included in this book is available for download from the book's companion website, *http://www.oreilly.com/catalog/9781449311445*.

Using Code Examples

This book is here to help you get your job done. In general, you may use the code in this book in your programs and documentation. You do not need to contact us for permission unless you're reproducing a significant portion of the code. For example, writing a program that uses several chunks of code from this book does not require permission. Selling or distributing a CD-ROM of examples from O'Reilly books does require permission. Answering a question by citing this book and quoting example code does not require permission. Incorporating a significant amount of example code from this book into your product's documentation does require permission.

We appreciate, but do not require, attribution. An attribution usually includes the title, author, publisher, and ISBN. For example: "*Programming Interactivity, Second Edition* by Joshua Noble (O'Reilly). Copyright 2012 Joshua Noble, 978-1-449-31144-5."

If you feel your use of code examples falls outside fair use or the permission given above, feel free to contact us at *permissions@oreilly.com*.

Safari® Books Online

Safari Books Online is an on-demand digital library that lets you easily search over 7,500 technology and creative reference books and videos to find the answers you need quickly.

With a subscription, you can read any page and watch any video from our library online. Read books on your cell phone and mobile devices. Access new titles before they are available for print, and get exclusive access to manuscripts in development and post feedback for the authors. Copy and paste code samples, organize your favorites, download chapters, bookmark key sections, create notes, print out pages, and benefit from tons of other time-saving features.

O'Reilly Media has uploaded this book to the Safari Books Online service. To have full digital access to this book and others on similar topics from O'Reilly and other publishers, sign up for free at *http://my.safaribooksonline.com*.

How to Contact Us

Please address comments and questions concerning this book to the publisher:

O'Reilly Media, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472
800-998-9938 (in the United States or Canada)
707-829-0515 (international or local)
707-829-0104 (fax)

We have a web page for this book, where we list errata, examples, and any additional information. You can access this page at:

http://shop.oreilly.com/product/0636920021735.do

To comment or ask technical questions about this book, send email to:

bookquestions@oreilly.com

For more information about our books, courses, conferences, and news, see our website at *http://www.oreilly.com*.

Find us on Facebook: *http://facebook.com/oreilly*

Follow us on Twitter: *http://twitter.com/oreillymedia*

Watch us on YouTube: *http://www.youtube.com/oreillymedia*

Acknowledgments

I need, first and foremost, to thank the wonderful engineers, artists, programmers, and dreamers who created the platforms that I've covered in this book. It is to all of them that I would like to dedicate this book. A woefully short list has to include Massimo Banzi, Tom Igoe, David Cuartielles, Gianluca Martino, David A. Mellis, Ben Fry, Casey Reas, Zach Lieberman, Theo Watson, Arturo Castro, and Chris O'Shea, the creators of the frameworks covered in this book. There are dozens, if not hundreds, of other names that should be on this list, but space is too limited to list them all. All I can say is thank to you to all the creators of these frameworks and to everyone who uses them to inspire, invent, amaze, and enrich the dialogue about design, technology, and art. This book is a humble attempt to thank you all for everything you've given to me—and to every other programmer, artist, or designer interested—for working with computing in novel and interesting ways and bringing more people into the conversation. I would also like to extend my deepest thanks to all my interviewees for taking the time

to respond to my questions and enrich this book and for so enriching the world of interaction design and art. To everyone who provided code for this book as well, created open source code, or answered questions on any of the forums for beginners: thank you for your efforts to create a community.

This book is as much my effort as it is the sum of the efforts of the editorial team that worked on it. My technical editors, Michael Margolis, Adam Parrish, Matt Obert, Jeff Crouse, and Jeremy Rotzstain, have been absolutely fantastic. Their expertise, suggestions, and fresh look at what I was working on shaped not only this book but enlightened me, showed me new ways of solving problems, introduced me to new tools and techniques, sharpened my thinking, and broadened my horizons for the better. This book is a collaboration among all four of us in every sense of the word. I cannot pay them enough thanks for their excellent work. I would also like to thank Justin Hunyh and Mike Gionfriddo from LiquidWare as well as Nathan Seidle from Sparkfun for all of their help. My editors—Shawn Wallace, Robyn Thomas, and Kim Wimpsett—have been incredible, helping me with my sometime torturous grammar and patiently working with my propensity for sending in extremely rough drafts to bounce ideas off of them. They have made this book better than it ever could have been without their watchful eyes and guidance. Finally, I need to thank Steve Weiss for listening to my idea when I first proposed it and helping guide it through to completion.

I need to thank all of my friends in New York, Portland, Amsterdam, Geneva, London, Zurich, Boston, Paris, Copenhagen, and Toulouse for their support, their ideas, their Internet, and their encouragement. I would like to thank my family as well, and particularly my mother, for their support and humor.

CHAPTER 1

Introducing Interaction Design

The scientist and philosopher Alfred Korzybski once remarked, "The map is not the territory," and it's in that spirit that this book was written. The map may not be the territory, but it is helpful for getting around the territory and for finding where you are and where you want to go. This book covers a vast range of topics from programming to electronics to interaction design to art, but it doesn't cover any one of them in great depth. It covers all of these topics because they are part of an emerging territory that is often called *interaction design*, and that territory encompasses art, design, psychology, engineering, and programming. It's also a territory that is becoming more and more accessible thanks to excellent projects like the ones that we'll be exploring in the book—tools that have been created to make code and coding easier to do.

You should use this book like a map to see what technologies exist and the areas in interaction design that you might want to explore. This isn't a cookbook or an in-depth technical manual, but it will point you in the direction of other books, researchers, designers, projects, and artists as you go along. This book will also give you the technical understanding to know how to find information on almost any kind of project that you want to explore and what to do with that information once you find it.

What This Book Is For

This book was created under the premise that technology and code are not tools solely for computer scientists or engineers to create applications and that no one be intimidated by or shy away from working with and exploring electronics, hardware, and code. Artists and designers can be interested in enabling interaction between users and between applications in ways that can be accentuated by the addition of custom computer applications or that can be realized only through the use of custom computer applications. You can focus on creating applications that emphasize their technological nature or on creating applications that feel very high-tech or use familiar metaphors like a keyboard and mouse or touchscreen. You can also choose to accentuate other aspects of the interaction or hide the technology behind a more organic interface. This book is specifically about the interactions that users or viewers can have with computers,

electronics, tools, and the platforms that artists and designers can use to create applications and electronics that users can interact with. You'll be learning about three tools: Processing, openFrameworks, and Arduino.

These frameworks are designed specifically for artists and designers and as such are perfect for discussing how we can begin to create interactive designs and artworks. Each of them has a different background and uses different kinds of technology, but all of them are created with the goal of helping you explore and create applications more painlessly and quickly. In addition to showing you specifics of those three tools, this book focuses on three slightly more abstract concepts: code, interaction design, and ideas. Creating code is a similar activity whether you're writing something in C++ for openFrameworks or you're creating some logic in a circuit with Arduino. In both cases, you're creating a process that will run many times, perhaps even thousands of times, and that will generate the outcome you want.

This book also makes a few assumptions about you, the reader. I assume that you don't have a deep, or even any, programming or technical background. I also assume that you're a designer, artist, or other creative thinker interested in learning about code to create interactive applications in some way or shape. You might be a designer wanting to begin playing with interactive elements in your designs, wanting to create physically reactive applications to explore some interaction design concept, or wanting to prototype an idea for a product. You might be an artist wanting to begin working with interactive installations or with interactive computer graphics. You might be an architect wanting to get a basic understanding of programming and hardware to explore reactive architecture. You might be none of these at all, which is fine, too, as long as you're interested in exploring these themes while you learn about the three frameworks this book describes.

You'll explore the nature of interaction through common tools and techniques as well as through some discussions with designers, engineers, and artists working with interaction. In all likelihood, this book will not radically alter your perception of what interaction is, nor will it introduce you to radically new modes of interaction. This book will introduce to you to methods of creating common interactive elements that you can then use to explore further techniques of facilitating interactions between users or creating interactive elements that a user or viewer can experience.

Programming for Interactivity

This book is called *Programming Interactivity* because it's focused primarily on programming for interaction design, that is, programming to create an application with which users interact directly. There are many styles of programming, and some techniques and ways of thinking about code are better suited to programming servers or databases than interaction. In this book, we're going to concentrate explicitly on things you can use to tell users something or to have users tell your application something.

One of the great challenges in interaction design is actually creating real interactions between what you're designing and the user who will be using it.

The Nature of Interaction

So then, what exactly is *interaction*? Interaction could be defined as the exchange of information between two or more active participants. The writer and video game designer Chris Crawford describes interaction as "an iterative process of listening, thinking, and speaking between two or more actors." Generally, when we're talking about interaction and programming it's because one element in the interaction is a computer system of some sort or some control element that a person is trying to get to do something. The person for whom the computer or mechanical system is being designed is called the *user*, and what the user is using is called the *system*. There are many different terms floating around today, such as *human computer interaction*, *computer human interaction*, or *experience design*. All mean more or less the same thing: designing a system of some sort that a person can interact with in a way that is meaningful to them. As an interaction designer, you're trying to understand what the user wants to do and how the system that you're creating should respond. That system can be almost anything: a game, a menu, a series of connected sensors and lights, a complicated physically interactive application, or even a group of other people.

There is another key concept in interaction design that you should understand: the *feedback loop*. The feedback loop is a process of an entity communicating with itself while checking with either an internal or external regulatory system. That sounds a little more complex than it actually is. You're actually already quite familiar with biological regulatory systems; sweating keeps your body cool, breathing keeps oxygen flowing through your body, and blinking keeps your eyes from drying out. When you need more oxygen, your body breathes harder. This isn't something you have to tell your body to do; it simply does it. To maintain a constant level of oxygen, it sends out signals to breathe more and more deeply or frequently until it reaches the correct level. It feeds back on itself, sending signals to itself to breathe more again and again until it doesn't need to send those signals anymore. You can also think of the feedback that you give yourself while staying upright on a bicycle. You're constantly adjusting your balance minutely, with your brain feeding data to your body and your body feeding data back in a constant loop that helps you stay balanced. These loops are important in the notion of a system that does something constantly. Without feedback, systems can't regulate themselves because they won't know what they're doing.

Let's start at *messaging* and work our way up to *interaction*. While one participant certainly may be more active than the other, the "interaction" doesn't really apply when we use it to describe a *transmission*, that is, a message sent to someone with no way of handling a response. Think of a television commercial or a radio broadcast: it's simply a signal that you can listen to if you're in the right place at the right time and you have the right equipment. These broadcasts flow on regardless of whether you or anyone else is listening, and they occur on their own time, in their own tempo.

When you give a user a way of *rewinding* or controlling the tempo of information, an extra layer of user control is added. You can't really *interact* with a book or a static web page, or even the vast majority of dynamic web pages, but you can control the speed at which you read them, and you can rewind information that you're not sure about. These are really guided transmissions in that they give you a chunk of information that is more or less established and ask you which part of it you want to view. Scrolling, linking, fast-forwarding, and rewinding are all the techniques of guided transmissions.

When you give a user a way to accomplish a task or input data into the system that changes it in a substantial way and you create a means for that system to respond to what the user is doing, then you're creating interaction. Reactive interaction is really the beginning of interaction because it gets you started thinking about what the user will do and how your system or object will react. For everything that user does, the system or object needs to have a response, even if that response is "I didn't understand" or another kind of error message. This can also be built into a single system. Many kinds of applications monitor their own performance, checking the state of a property in the system or the number of boxes available in a warehouse, for instance. If you imagine this as being an interaction between two people, then you might imagine a parent giving a child an order.

A somewhat more complex model of interaction is one where the system is constantly doing a task and the users' input regulates that task. Many industrial monitoring systems function this way, as do the underlying parts of game engines, and many interactive installations. The difficulty of creating this kind of interaction is ensuring that users always know what the system is doing at any given time, understand how they can modify it, and understand exactly how their modifications to one aspect of the system might affect another. If you imagine this between two people, then you might imagine a parent helping a child walk, ensuring that she doesn't fall over as she goes. You can also imagine how a regulatory system might function, where the system regulates the user as they're executing a task. This isn't really two entities fully communicating because the regulated system doesn't respond—it simply changes its behavior—but it does involve continuous systems. Systems can perform this task on their own as well, monitoring a process and providing regulation of an ongoing process.

This last mode of interaction blends into another. It is a very similar but slightly more complex model of creating interaction that might be described as the *didactic*, or learning, mode of interaction. Here, the system is still running continuously, and the user can see into the system, but instead of regulating the behavior, the user is learning from the output data. A lot of monitoring applications function this way, providing a view into relevant data and data points that the user can use to learn about a process. Again, the system isn't actively conversing with a user; it's just running and reporting information to the user. The user also has his process driven by the reporting from the system but not really modified by it, which is why it's a learning model. Both systems and people are more than capable of learning from themselves, albeit in quite different ways.

sample content of Programming Interactivity

- Collecting and Care of Fine Art: An Introduction to Purchasing, Investing, Evaluating, Restoring, and More pdf, azw (kindle)
- read Ice Cream: A Global History (Edible Series)
- read 7 Sessions (The 7 Novellas Series, Book 1) pdf, azw (kindle)
- Industrial Archaeology: Principles and Practice online
- Exploring Quantum Mechanics: A Collection of 700+ Solved Problems for Students, Lecturers, and Researchers book

- http://rodrigocaporal.com/library/Modern-Cosmology.pdf
- http://paulczajak.com/?library/Path-of-the-Assassin--Scot-Harvath--Book-2-.pdf
- http://www.1973vision.com/?library/7-Sessions--The-7-Novellas-Series--Book-1-.pdf
- http://chelseaprintandpublishing.com/?freebooks/Twelve-Months-of-Monastery-Salads--200-Divine-Recipes-for-All-Seasons.pdf
- http://www.experienceolvera.co.uk/library/Practical-Female-Psychology-for-the-Practical-Man.pdf