How To Design Programs An Introduction Programming And Computing Matthias Felleisen

This report has been developed to assist the National Aeronautics and Space Administration (NASA) in responding to a Congressional request to evaluate the feasibility of initiating a National Scholars Program. It includes detailed design for a program to increase the number of underparticipating minorities earning Ph. D.s in mathematics, the physical sciences, and engineering. The proposed program will provide (a) a continuum of academic and financial support for participants beginning in high school and continuing through Ph. D. study and (b) coordination with other science education initiatives funded by NASA and/or other agencies, organizations, and institutions.

Intelligent readers who want to build their own embedded computer systems-- installed in everything from cell phones to cars to handheld organizers to refrigerators-- will find this book to be the most in-depth, practical, and up-to-date guide on the market. Designing Embedded Hardware carefully steers between the practical and philosophical aspects, so developers can both create their own devices and gadgets and customize and extend off-the-shelf systems. There are hundreds of books to choose from if you need to learn programming, but only a few are available if you want to learn to create hardware. Designing Embedded Hardware provides software and hardware engineers with no prior experience in embedded systems with the necessary conceptual and design building blocks to understand the architectures of embedded systems. Written to provide the depth of coverage and real-world examples developers need, Designing Embedded Hardware also provides a road-map to the pitfalls and traps to avoid in designing embedded systems. Designing Embedded Hardware covers such essential topics as: The principles of developing computer hardware Core hardware designs Assembly language concepts Parallel I/O Analog-digital conversion Timers (internal and external) UART Serial Peripheral Interface Inter-Integrated Circuit Bus Controller Area Network (CAN) Data Converter Interface (DCI) Low-power operation This invaluable and eminently useful book gives you the practical tools and skills to develop, build, and program your own application-specific computers.

Get a grounding in polymorphism and other fundamental aspects of object-oriented program design and implementation, and learn a subset of design patterns that any practicing Java professional simply must know in today’s job climate. Java Program Design presents program design principles to help practicing programmers up their game and remain relevant in the face of changing trends and an evolving language. The book enhances the traditional design patterns with Java's new functional programming features, such as functional interfaces and lambda expressions. The result is a fresh treatment of design patterns that expands their power and applicability, and reflects current

best practice. The book examines some well-designed classes from the Java class library, using them to illustrate the various object-oriented principles and patterns under discussion. Not only does this approach provide good, practical examples, but you will learn useful library classes you might not otherwise know about. The design of a simplified banking program is introduced in chapter 1 in a non-object-oriented incarnation and the example is carried through all chapters. You can see the object orientation develop as various design principles are progressively applied throughout the book to produce a refined, fully object-oriented version of the program in the final chapter. What You'll Learn Create well-designed programs, and identify and improve poorly-designed ones Build a professional-level understanding of polymorphism and its use in Java interfaces and class hierarchies Apply classic design patterns to Java programming problems while respecting the modern features of the Java language Take advantage of classes from the Java library to facilitate the implementation of design patterns in your programs Who This Book Is For Java programmers who are comfortable writing non-object-oriented code and want a guided immersion into the world of object-oriented Java, and intermediate programmers interested in strengthening their foundational knowledge and taking their object-oriented skills to the next level. Even advanced programmers will discover interesting examples and insights in each chapter.
"This book explains how to design an optical system using the high-end optical design program CODE V. The design process, from lens definition to the description and evaluation of lens errors and onto the improvement of lens performance, will be developed and illustrated using the program. The text is organized so that readers can (1) reproduce each step of the process including the plots for evaluating lens performance and (2) understand the significance of each step in producing a final design"--
Presents a multifaceted model of understanding, which is based on the premise that people can demonstrate understanding in a variety of ways.
How to use design as a tool to create not only things but ideas, to speculate about possible futures. Today designers often focus on making technology easy to use, sexy, and consumable. In Speculative Everything, Anthony Dunne and Fiona Raby propose a kind of design that is used as a tool to create not only things but ideas. For them, design is a means of speculating about how things could be—to imagine possible futures. This is not the usual sort of predicting or forecasting, spotting trends and extrapolating; these kinds of predictions have been proven wrong, again and again. Instead, Dunne and Raby pose “what if” questions that are intended to open debate and discussion about the kind of future people want (and do not want). Speculative Everything offers a tour through an emerging cultural landscape of design ideas, ideals, and approaches. Dunne and Raby cite examples from their own design and teaching and from other projects from fine art, design, architecture, cinema, and photography. They also draw on futurology, political theory, the philosophy of technology, and literary

fiction. They show us, for example, ideas for a solar kitchen restaurant; a flypaper robotic clock; a menstruation machine; a cloud-seeding truck; a phantom-limb sensation recorder; and devices for food foraging that use the tools of synthetic biology. Dunne and Raby contend that if we speculate more—about everything—reality will become more malleable. The ideas freed by speculative design increase the odds of achieving desirable futures.

The second edition of the Impact Evaluation in Practice handbook is a comprehensive and accessible introduction to impact evaluation for policy makers and development practitioners. First published in 2011, it has been used widely across the development and academic communities. The book incorporates real-world examples to present practical guidelines for designing and implementing impact evaluations. Readers will gain an understanding of impact evaluations and the best ways to use them to design evidence-based policies and programs. The updated version covers the newest techniques for evaluating programs and includes state-of-the-art implementation advice, as well as an expanded set of examples and case studies that draw on recent development challenges. It also includes new material on research ethics and partnerships to conduct impact evaluation. The handbook is divided into four sections: Part One discusses what to evaluate and why; Part Two presents the main impact evaluation methods; Part Three addresses how to manage impact evaluations; Part Four reviews impact evaluation sampling and data collection. Case studies illustrate different applications of impact evaluations. The book links to complementary instructional material available online, including an applied case as well as questions and answers. The updated second edition will be a valuable resource for the international development community, universities, and policy makers looking to build better evidence around what works in development.

The definitive resource for understanding what coding is, designed for educators and parents Even though the vast majority of teachers, parents, and students understand the importance of computer science in the 21st century, many struggle to find appropriate educational resources. Don't Teach Coding: Until You Read This Book fills a gap in current knowledge by explaining exactly what coding is and addressing why and how to teach the subject. Providing a historically grounded, philosophically sensitive description of computer coding, this book helps readers understand the best practices for teaching computer science to their students and their children. The authors, experts in teaching computer sciences to students of all ages, offer practical insights on whether coding is a field for everyone, as opposed to a field reserved for specialists. This innovative book provides an overview of recent scientific research on how the brain learns coding, and features practical exercises that strengthen coding skills. Clear, straightforward chapters discuss a broad range of questions using principles of computer science, such as why we should teach students to code and is coding a science, engineering, technology, mathematics, or language?

Helping readers understand the principles and issues of coding education, this book: Helps those with no previous background in computer science education understand the questions and debates within the field Explores the history of computer science education and its influence on the present Views teaching practices through a computational lens Addresses why many schools fail to teach computer science adequately Explains contemporary issues in computer science such as the language wars and trends that equate coding with essential life skills like reading and writing Don't Teach Coding: Until You Read This Book is a valuable resource for K-12 educators in computer science education and parents wishing to understand the field to help chart their children’s education path.
Sams Teach Yourself Beginning Programming in 24 Hours, Second Edition explains the basics of programming in the successful 24-Hours format. The book begins with the absolute basics of programming: Why program? What tools to use? How does a program tell the computer what to do? It teaches readers how to program the computer and then moves on by exploring the some most popular programming languages in use. The author starts by introducing the reader to the Basic language and finishes with basic programming techniques for Java, C++, and others.
Describes the LISP programming language, and covers basic procedures, data, and modularity
The Art of UNIX Programming poses the belief that understanding the unwritten UNIX engineering tradition and mastering its design patterns will help programmers of all stripes to become better programmers. This book attempts to capture the engineering wisdom and design philosophy of the UNIX, Linux, and Open Source software development community as it has evolved over the past three decades, and as it is applied today by the most experienced programmers. Eric Raymond offers the next generation of "hackers" the unique opportunity to learn the connection between UNIX philosophy and practice through careful case studies of the very best UNIX/Linux programs.
Despite the promise of competency-based education (CBE), learner-centered issues related to support, retention, and program completion rates remain problematic. In addition, the infrastructure for higher education, including issues related to faculty (intellectual property, workload, and curriculum), pose barriers and challenges in the design, development, implementation, and delivery of CBE. In response, administrators, faculty, designers, and developers of competency-based experiences must incorporate innovative strategies that are foreign to the traditional institution. A strong emphasis on retention and graduation rates must surround the student with support, starting with the design and development of the CBE system. There are few resources that can help prepare instructional designers, advisors, academic administrators, and faculty to meet the many challenges of designing, developing, implementing, and managing CBE. Career Ready Education Through Experiential Learning is an essential reference book that includes strategies for design and development of competency-based

education (CBE) programs, as well as administrative and delivery strategies as examples of how CBE can be implemented. Through a strong theoretical framework, chapters present the best practices, strategies, and practical tips as examples and scenarios that can be used in higher education settings. While highlighting education courses, programs, and lessons across various institutions and educational domains, this book is ideal for higher education administrators and policy designers/implementors, instructional designers, curriculum developers, faculty, public policy leaders, students in curriculum and instruction and instructional technology programs, along with researchers and practitioners interested in CBE and experiential learning in higher education.

A completely revised edition, offering new design recipes for interactive programs and support for images as plain values, testing, event-driven programming, and even distributed programming. This introduction to programming places computer science at the core of a liberal arts education. Unlike other introductory books, it focuses on the program design process, presenting program design guidelines that show the reader how to analyze a problem statement, how to formulate concise goals, how to make up examples, how to develop an outline of the solution, how to finish the program, and how to test it. Because learning to design programs is about the study of principles and the acquisition of transferable skills, the text does not use an off-the-shelf industrial language but presents a tailor-made teaching language. For the same reason, it offers DrRacket, a programming environment for novices that supports playful, feedback-oriented learning. The environment grows with readers as they master the material in the book until it supports a full-fledged language for the whole spectrum of programming tasks. This second edition has been completely revised. While the book continues to teach a systematic approach to program design, the second edition introduces different design recipes for interactive programs with graphical interfaces and batch programs. It also enriches its design recipes for functions with numerous new hints. Finally, the teaching languages and their IDE now come with support for images as plain values, testing, event-driven programming, and even distributed programming.

This book is intended to support educators in the design and implementation of comprehensive gifted education plans. From planning to actual implementation, this book takes the reader from goals and purpose to assessing student needs and program design. The authors begin with a broad overview of best practices in programming and services, highlighting connections to student needs, programming standards, and state laws. Their recommendations include philosophical, cultural, and practical considerations and data-based decision making. In this book, Peters and Brulles guide the reader through the process of determining the most optimal programming methods for schools to take based on their individual needs and circumstances. With this book, schools will be able to design and develop programs and/or services that lay the foundation necessary to ensure all students are appropriately challenged.

An entertaining activity book packed with fun design projects - from lettering and book covers to costumes and gadgets. Full of helpful tips and space to imagine, draw and create. This write-in activity book explores all sorts of design skills, from how to create stunning new typefaces and furniture, to designing costumes, games and websites. Aspiring designers will have hours of fun coming up with their own designs, guided by lots of handy tips and tricks to help them along the way. Combines real design skills with imaginative activities and creative projects. Wide-ranging activities cover everything from graphic design, fashion and interiors, to designing websites, typefaces and branding. Includes links to templates to download for activities in the book.

Processing simple forms of data - Processing arbitrarily large data - More on processing arbitrarily large data - Abstracting designs - Generative recursion - Changing the state of variables - Changing compound values.

Build real-world Artificial Intelligence applications with Python to intelligently interact with the world around you About This Book Step into the amazing world of intelligent apps using this comprehensive guide Enter the world of Artificial Intelligence, explore it, and create your own applications Work through simple yet insightful examples that will get you up and running with Artificial Intelligence in no time Who This Book Is For This book is for Python developers who want to build real-world Artificial Intelligence applications. This book is friendly to Python beginners, but being familiar with Python would be useful to play around with the code. It will also be useful for experienced Python programmers who are looking to use Artificial Intelligence techniques in their existing technology stacks. What You Will Learn Realize different classification and regression techniques Understand the concept of clustering and how to use it to automatically segment data See how to build an intelligent recommender system Understand logic programming and how to use it Build automatic speech recognition systems Understand the basics of heuristic search and genetic programming Develop games using Artificial Intelligence Learn how reinforcement learning works Discover how to build intelligent applications centered on images, text, and time series data See how to use deep learning algorithms and build applications based on it In Detail Artificial Intelligence is becoming increasingly relevant in the modern world where everything is driven by technology and data. It is used extensively across many fields such as search engines, image recognition, robotics, finance, and so on. We will explore various real-world scenarios in this book and you'll learn about various algorithms that can be used to build Artificial Intelligence applications. During the course of this book, you will find out how to make informed decisions about what algorithms to use in a given context. Starting from the basics of Artificial Intelligence, you will learn how to develop various building blocks using different data mining techniques. You will see how to implement different algorithms to get the best possible results, and will understand how to apply them to real-world scenarios. If you want to add an intelligence layer to any application that's based on images, text, stock market, or some other form of data, this exciting book on Artificial Intelligence will definitely be your guide! Style and approach This highly practical book will show you how to implement Artificial Intelligence. The book provides multiple examples enabling you to create smart applications to meet the needs of your organization. In every chapter, we explain an algorithm, implement it, and then build a smart application.

A quick overview of Object-oriented program design, with special regard for operating-system development, this book will be of the greatest interest to those developers who are working with Taligent and its operating partners, as well as many other C++ programmers who are interested in a provocative summary of good OOP techniques.

Using research in neurobiology, cognitive science and learning theory, this text loads patterns into your brain in a way that lets you put them to work immediately, makes you better at solving software design problems, and improves your ability to speak the language of patterns with others on your team.

Strategies for building large systems that can be easily adapted for new situations with only minor programming modifications. Time pressures encourage programmers to write code that works well for a narrow purpose, with no room to grow. But the best systems are evolvable; they can be adapted for new situations by adding code, rather than changing the existing code. The authors describe techniques they have found effective--over their combined 100-plus years of programming experience--that will help programmers avoid programming themselves into corners. The authors explore ways to enhance flexibility by: • Organizing systems using combinators to compose mix-and-match parts, ranging from small functions to whole arithmetics, with standardized interfaces • Augmenting data with independent annotation layers, such as units of measurement or provenance • Combining independent pieces of partial information using unification or propagation • Separating control structure from problem domain with domain models, rule systems and pattern matching, propagation, and dependency-directed backtracking • Extending the programming language, using dynamically extensible evaluators

The Mata Book: A Book for Serious Programmers and Those Who Want to Be is the book that Stata programmers have been waiting for. Mata is a serious programming language for developing small- and large-scale projects and for adding features to Stata. What makes Mata serious is that it provides structures, classes, and pointers along with matrix capabilities. The book is serious in that it covers those advanced features, and teaches them. The reader is assumed to have programming experience, but only some programming experience. That experience could be with Stata's ado language, or with Python, Java, C++, Fortran, or other languages like them. As the book says, "being serious is a matter of attitude, not current skill level or knowledge". The author of the book is William Gould, who is also the designer and original programmer of Mata, of Stata, and who also happens to be the president of StataCorp.

Given the increasing diversity of the United States and students entering schools, the value of teacher learning in clinical contexts, and the need to elevate the profession, national organizations have been calling for a re-envisioning of teacher preparation that turns teacher education upside down. This change will require PK-12 schools and universities to partner in robust ways to create strong professional learning experiences for aspiring teachers. University faculty, in particular, will not only need to work?in?schools, but they will need to work?with?schools in the preparation of future teachers. This collaboration should promote greater equity and justice for our nation’s students. The purpose of this book is to support individuals in designing clinically based teacher preparation programs that place equity at the core. Drawing from the literature as well as our experiences in designing and coordinating award-winning teacher?education programs, we offer a vision for equity-centered, clinically based preparation that promotes powerful teacher professional learning and develops high-quality, equity-centered teachers for schools. The chapter topics include policy guidelines, partnerships, intentional clinical experiences, coherence, curriculum and coursework, university-based teacher educators, school-based teacher educators, teacher candidate supervision and evaluation, the role of research, and instructional leadership in teacher preparation. While the concepts we share are research-based and grounded in the empirical literature, our primary intention is for this book to be of practical use. We hope that by the time you finish reading, you will feel inspired and equipped to make change within your own program, your institution, and your local context. We begin each chapter with a “Before You Read” section that includes introductory activities or self-assessment questions to prompt reflection about the current state of your teacher preparation program. We also weave examples, a “Spotlight from Practice,” in the form of vignettes designed to spark your thinking for program improvement. Finally, we conclude each chapter with a section called “Exercises for Action,” which are questions or activities to help you (re)imagine and move toward action in the (re)design of your teacher preparation program. We hope that you will use the exercises by yourself, but perhaps more importantly, with others to stimulate conversations about how you can build upon what you are already doing well to make your program even better. Praise for (Re)Designing Programs: A Vision for Equity-Centered, Clinically Based Teacher Preparation: "Jennifer Jacobs and Rebecca West Burns’ book, “(Re)Designing Programs: A Vision for Equity-Centered, Clinically Based Teacher Preparation,” is a must-read for all teacher educators, especially those involved in the creation and/or direction of clinically based teacher education programs. Their text provides a roadmap for higher education and school-based teacher educators to collaboratively design a program that prepares teachers to meet the needs of future students. They not only redefine the terms and language we use within clinical practice programs but also encourage us to reflect upon how teachers should be prepared in an equity-centered, clinically based teacher education program. Their text deserves to be on the book shelves of all teacher educators." - D. John McIntyre

Learning to program isn't just learning the details of a programming language: to become a good programmer you have to become expert at debugging, testing, writing clear code and generally unsticking yourself when you get stuck, while to do well in a programming course you have to learn to score highly in coursework and exams. Featuring tips, stories and explanations of key terms, this book teaches these skills explicitly. Examples in Python, Java and Haskell are included, helping you to gain transferable programming skills whichever language you are learning. Intended for students in Higher or Further Education studying early programming courses, it will help you succeed in, and get the most out of, your course, and support you in developing the software engineering habits that lead to good programs.

#1 NEW YORK TIMES BEST SELLER • At last, a book that shows you how to build—design—a life you can thrive in, at any age or stage Designers create worlds and solve problems using design thinking. Look around your office or home—at the tablet or smartphone you may be holding or the chair you are sitting in. Everything in our lives was designed by someone. And every design starts with a problem that a designer or team of designers seeks to solve. In this book, Bill Burnett and Dave Evans show us how design thinking can help us create a life that is both meaningful and fulfilling, regardless of who or where we are, what we do or have done for a living, or how young or old we are. The same design thinking responsible for amazing technology, products, and spaces can be used to design and build your career and your life, a life of fulfillment and joy, constantly creative and productive, one that always holds the possibility of surprise.

Racket is a descendant of Lisp, a programming language renowned for its elegance, power, and challenging learning curve. But while Racket retains the functional goodness of Lisp, it was designed with beginning programmers in mind. Realm of Racket is your introduction to the Racket language. In Realm of Racket, you'll learn to program by creating increasingly complex games. Your journey begins with the Guess My Number game and coverage of some basic Racket etiquette. Next you'll dig into syntax and semantics, lists, structures, and conditionals, and learn to work with recursion and the GUI as you build the Robot Snake game. After that it's on to lambda and mutant structs (and an Orc Battle), and fancy loops and the Dice of Doom. Finally, you'll explore laziness, AI, distributed games, and the Hungry Henry game. As you progress through the games, chapter checkpoints and challenges help reinforce what you've learned. Offbeat comics keep things fun along the way. As you travel through the Racket realm, you'll: –Master the quirks of Racket's syntax and semantics –Learn to write concise and elegant functional programs –Create a graphical user interface using the 2htdp/image library –Create a server to handle true multiplayer games Realm of Racket is a lighthearted guide to some serious programming. Read it to see why Racketeers have so much fun!

The Fifth Edition of the classic Designing and Managing Programs for human services helps readers grasp the meaning and significance of measuring performance and evaluating outcomes. The authors, all leaders in the field, incorporate the principles of effectiveness-based planning as they address the steps of designing, implementing, and evaluating a human services program at the local agency level. Meaningful examples at every stage of the process—from problem analysis and needs assessment to evaluating effectiveness and calculating costs—enhance reader understanding of how concepts are implemented in the real world.

This is a book for students at every level who are learning to program for the first time - and for the considerable number who learned how to program but were never taught to structure their programs. The author presents a simple set of guidelines that show the programmer how to design in a manageable structure from the outset. The method is suitable for most languages, and is based on the widely used 'JSP' method, to which the student may easily progress if it is needed at a later stage. Most language specific texts contain very little if any information on design, whilst books on design approach the topic at too high a level for someone learning their first language. This inexpensive introduction to design can be used alongside whatever programming book suits the student's particular needs.

A first programming course should not be directed towards learning a particular programming language, but rather at learning to program well; the programming language should get out of the way and serve this goal. The simple, powerful Racket language (related to Scheme) allows us to concentrate on the fundamental concepts and techniques of computer programming, without being distracted by complex syntax. As a result, this book can be used at the high school (and perhaps middle school) level, while providing enough advanced concepts not usually found in a first course to challenge a college student. Those who have already done some programming (e.g. in Java, Python, or C++) will enhance their understanding of the fundamentals, un-learn some bad habits, and change the way they think about programming. We take a graphics-early approach: you'll start manipulating and combining graphic images from Chapter 1 and writing event-driven GUI programs from Chapter 6, even before seeing arithmetic. We continue using graphics, GUI and game programming throughout to motivate fundamental concepts. At the same time, we emphasize data types, testing, and a concrete, step-by-step process of problem-solving. After working through this book, you'll be prepared to learn other programming languages and program well in them. Or, if this is the last programming course you ever take, you'll understand many of the issues that affect the programs you use every day. I have been using Picturing Programs with my daughter, and there's no doubt that it's gentler than Htdp. It does exactly what Stephen claims, which is to move gradually from copy-and-change exercises to think-on-your-own exercises within each section. I also think it's nice that the "worked exercises" are clearly labeled as such. There's something psychologically appealing about the fact that you first see an example in the text of the book, and then a similar example is presented as if it were an exercise but they just happen to be

giving away the answer. It is practically shouting out "Here's a model of how you go about solving this class of problems, pay close attention ."" Mark Engelberg "1. Matthias & team have done exceptional, highly impressive work with HtDP. The concepts are close to genius. (perhaps yes, genius quality work) They are a MUST for any high school offering serious introductory CS curriculum. 2. Without Dr. Blochs book "Picturing Programs," I would not have successfully implemented these concepts (Dr. Scheme, Racket, Design Recipe etc) into an ordinary High School Classroom. Any high school instructor who struggles to find ways to bring these great HtDP ideas to the typical high schooler, should immediately investigate the Bloch book. Think of it as coating the castor oil with chocolate." Brett Penza
??????????????????
??????C++???????????????,????????????????????????
An introduction to dependent types, demonstrating the most beautiful aspects, one step at a time. A program's type describes its behavior. Dependent types are a first-class part of a language, and are much more powerful than other kinds of types; using just one language for types and programs allows program descriptions to be as powerful as the programs they describe. The Little Typer explains dependent types, beginning with a very small language that looks very much like Scheme and extending it to cover both programming with dependent types and using dependent types for mathematical reasoning. Readers should be familiar with the basics of a Lisp-like programming language, as presented in the first four chapters of The Little Schemer. The first five chapters of The Little Typer provide the needed tools to understand dependent types; the remaining chapters use these tools to build a bridge between mathematics and programming. Readers will learn that tools they know from programming—pairs, lists, functions, and recursion—can also capture patterns of reasoning. The Little Typer does not attempt to teach either practical programming skills or a fully rigorous approach to types. Instead, it demonstrates the most beautiful aspects as simply as possible, one step at a time.
An introduction to the Racket functional programming language and DrRacket development environment to explore topics in mathematics (mostly recreational) and computer science. At last, a lively guided tour through all the features, functions, and applications of the Racket programming language. You'll learn a variety of coding paradigms, including iterative, object oriented, and logic programming; create interactive graphics, draw diagrams, and solve puzzles as you explore Racket through fun computer science topics--from statistical analysis to search algorithms, the Turing machine, and more. Early chapters cover basic Racket concepts like data types, syntax, variables, strings, and formatted output. You'll learn how to perform math in Racket's rich numerical environment, and use programming constructs in different problem domains (like coding solutions to the Tower of Hanoi puzzle). Later, you'll play with plotting, grapple with graphics, and visualize data. Then, you'll escape the confines of the command line to produce animations, interactive games, and a card trick program that'll dazzle your friends. You'll learn how to: • Use DrRacket, an interactive development environment (IDE) for writing programs • Compute classical math problems, like the Fibonacci sequence • Generate two-dimensional function plots and create drawings using graphics primitives • Import and export data to and from Racket using ports, then visually analyze it • Build simple computing devices (pushdown

automaton, Turing machine, and so on) that perform tasks • Leverage Racket's built-in libraries to develop a command line algebraic calculator Racket Programming the Fun Way is just like the language itself--an embodiment of everything that makes programming interesting and worthwhile, and that makes you a better programmer. You know how to code in Elixir; now learn to think in it. Learn to design libraries with intelligent layers that shape the right data structures, flow from one function into the next, and present the right APIs. Embrace the same OTP that's kept our telephone systems reliable and fast for over 30 years. Move beyond understanding the OTP functions to knowing what's happening under the hood, and why that matters. Using that knowledge, instinctively know how to design systems that deliver fast and resilient services to your users, all with an Elixir focus. Elixir is gaining mindshare as the programming language you can use to keep you software running forever, even in the face of unexpected errors and an ever growing need to use more processors. This power comes from an effective programming language, an excellent foundation for concurrency and its inheritance of a battle-tested framework called the OTP. If you're using frameworks like Phoenix or Nerves, you're already experiencing the features that make Elixir an excellent language for today's demands. This book shows you how to go beyond simple programming to designing, and that means building the right layers. Embrace those data structures that work best in functional programs and use them to build functions that perform and compose well, layer by layer, across processes. Test your code at the right place using the right techniques. Layer your code into pieces that are easy to understand and heal themselves when errors strike. Of all Elixir's boons, the most important one is that it guides us to design our programs in a way to most benefit from the architecture that they run on. The experts do it and now you can learn to design programs that do the same. What You Need: Elixir Version 1.7 or greater. How the theoretical tools of literacy help us understand programming in its historical, social and conceptual contexts. The message from educators, the tech community, and even politicians is clear: everyone should learn to code. To emphasize the universality and importance of computer programming, promoters of coding for everyone often invoke the concept of “literacy,” drawing parallels between reading and writing code and reading and writing text. In this book, Annette Vee examines the coding-as-literacy analogy and argues that it can be an apt rhetorical frame. The theoretical tools of literacy help us understand programming beyond a technical level, and in its historical, social, and conceptual contexts. Viewing programming from the perspective of literacy and literacy from the perspective of programming, she argues, shifts our understandings of both. Computer programming becomes part of an array of communication skills important in everyday life, and literacy, augmented by programming, becomes more capacious. Vee examines the ways that programming is linked with literacy in coding literacy campaigns, considering the ideologies that accompany this coupling, and she looks at how both writing and programming encode and distribute information. She explores historical parallels between writing and programming, using the evolution of mass textual literacy to shed light on the trajectory of code from military and government infrastructure to large-scale businesses to personal use. Writing and coding were institutionalized, domesticated, and then established as a basis for literacy. Just as societies demonstrated a “literate mentality” regardless of the literate status of individuals, Vee argues, a “computational mentality”

is now emerging even though coding is still a specialized skill.
Products, technologies, and workplaces change so quickly today that everyone is continually learning. Many of us are also teaching, even when it's not in our job descriptions. Whether it's giving a presentation, writing documentation, or creating a website or blog, we need and want to share our knowledge with other people. But if you've ever fallen asleep over a boring textbook, or fast-forwarded through a tedious e-learning exercise, you know that creating a great learning experience is harder than it seems. In Design For How People Learn, you'll discover how to use the key principles behind learning, memory, and attention to create materials that enable your audience to both gain and retain the knowledge and skills you're sharing. Using accessible visual metaphors and concrete methods and examples, Design For How People Learn will teach you how to leverage the fundamental concepts of instructional design both to improve your own learning and to engage your audience.
There is a lot of material on Scratch Programming on the Internet, including videos, online courses, Scratch projects, and so on, but, most of it is introductory. There is very little that can take students to the next level, where they can apply their Scratch and CS concepts to exciting and challenging problems. There is also very little material that shows students how to design complex projects, and introduces them to the process of programming.This book is meant to fill these gaps.In short, this book is for students who are already familiar with Scratch: its various commands, its user interface, and how it represents a variety of CS concepts such as, variables, conditional statements, looping, and so on. The book does not attempt to teach these concepts, but, it does provide a quick introduction to each concept in the free Supplement to the book.I call this an "interactive book" because it is something between a traditional book - which is static and passive - and a fully interactive online course. It does look like a book: it has a series of chapters, diagrams, a lot of text, etc. But it also contains links to online Scratch programs, code snippets, references, which the reader is expected to click and explore to fully benefit from the ideas presented.I have organized the book as a series of independent Scratch projects - each of which describes how to design and build an interesting and challenging Scratch program. Each project progresses in stages - from a simple implementation to increasingly complex versions. You can read these chapters in any order you like, although I have tried to arrange the chapters in an increasing order of challenge.Programming is a powerful tool that can be applied to virtually any field of human endeavor. I have tried to maintain a good diversity of applications in this book. You will find the following types of projects:-Simple ball games-Puzzle games-Memory games-Science simulations-Math games-Geometric designsLearn the concepts:As the experts will tell you, concepts are really understood and internalized when you apply them to solve problems. The purpose of this book is to help you apply Scratch and CS concepts to solve interesting and challenging programming problems. Every chapter lists, at the very start, the Scratch and CS concepts that you will apply while building that project.Learn the design process:Besides these technical concepts, you will also learn the "divide and conquer" approach of problem-solving. This is a fancy term for the technique of breaking down a bigger problem into many smaller problems and solving them separately one by one.You will also learn the "iterative design process" for designing programs. This is another fancy name that describes the idea that something complex can be designed in a repeated idea -> implement -> test

cycle, such that in each cycle we add a little more complexity.You will also learn a bit of "project management". Project management helps you undertake a project, such as creating a complex program, and complete it in a reasonable time, with reasonable effort, and with reasonable quality. It involves things such as planning tasks, tracking their progress, etc.Audience for the book:The book is intended for students who are already familiar with Scratch. The level of challenge is tuned for middle- and high-school students, but elementary-school students who have picked up all the concepts in an introductory course might also be able to enjoy the projects presented in this book.The book would be a great resource for teachers who teach Scratch programming. They could use the projects to teach advanced tricks of programming and to show how complex programs are designed.Finally, the book is for anyone who wants to get the wonderful taste of the entertaining and creative aspect of Computer Programming.