As recognized, adventure as well as experience roughly lesson, amusement, as capably as bargain can be gotten by just checking out a book **history of programming languages** as a consequence it is not directly done, you could take even more in the region of this life, almost the world.

We have the funds for you this proper as with ease as easy exaggeration to get those all. We meet the expense of the history of programming languages and numerous books collections from fictions to scientific research in any way. in the course of them is this history of programming languages that can be your partner.

**History of Programming Languages**

- **Author:** Richard L. Wexelblat
- **Edition:** 14 - 2014-05-27
- **Description:** The book presents information pertinent to the technical aspects of the language design and creation. This book provides an understanding of the processes of language design as related to the environment in which languages are developed and the knowledge base available to the originators. Organized into 14 sections encompassing 77 chapters, this book begins with an overview of the programming techniques to use to help the system produce efficient programs. This text then discusses how to use parentheses to help the system identify identical subexpressions within an expression and thereby eliminate their duplicate calculation. Other chapters consider FORTRAN programming techniques needed to produce optimum object programs. This book discusses as well the developments leading to ALGOL 60. The final chapter presents the biography of Adin D. Falkoff. This book is a valuable resource for graduate students, practitioners, historians, statisticians, mathematicians, programmers, as well as computer scientists and specialists.

**Programming Languages: History and Fundamentals**

- **Author:** Jean E. Sammet
- **Edition:** 2 - 1969
- **Description:** The primary purpose of this book is to serve as a reference for an overall view of higher level languages. The book brings together in one place, and in a consistent fashion, fundamental information on programming languages, including history, general characteristics, similarities, and differences. A second purpose of the book is to provide specific basic information on all the significant, and most of the minor, higher level languages developed in the United States. The third purpose of the book is to provide history and perspective for this particular aspect of the programming field. - Preface.

**Programming Languages: History and Fundamentals**

- **Author:** Jean E. Sammet
- **Edition:** 2 - 1969
- **Description:** The primary purpose of this book is to serve as a reference for an overall view of higher level languages. The book brings together in one place, and in a consistent fashion, fundamental information on programming languages, including history, general characteristics, similarities, and differences. A second purpose of the book is to provide specific basic information on all the significant, and most of the minor, higher level languages developed in the United States. The third purpose of the book is to provide history and perspective for this particular aspect of the programming field. - Preface.

**Masterminds of Programming**

- **Author:** Federico Biancuzzi
- **Edition:** 3 - 2009-03-21
- **Description:** The book provides a valuable comparison of the syntax of a number of the influential programming languages, using the famous "Hello, World!" code.

**Introduction to Programming Languages**

- **Author:** Arvind Kumar Bansal
- **Edition:** 1 - 2013-12-14
- **Description:** In programming courses, using the different syntax of multiple languages, such as C++, Java, PHP, and Python, for the same abstraction often confuses students new to computer science. Introduction to Programming Languages separates programming language concepts from the restraints of multiple language syntax by discussing the concepts at an abstract level. Designed for a one-semester undergraduate course, this classroom-tested book teaches the principles of programming language design and implementation. It presents: Common features of programming languages at an abstract level rather than a comparative level The implementation model and behavior of programming paradigms at abstract levels so that students understand the power and limitations of programming paradigms Language constructs at a paradigm level A holistic view of programming language design and behavior To make the book self-contained, the author introduces the necessary concepts of data structures and discrete structures from the perspective of programming language theory. The text covers classical and modern programming paradigms, program structures, information exchange between subprograms, object-oriented programming, logic programming, and functional programming. It also explores newer topics, including dependency analysis, communicating sequential processes, concurrent programming constructs, web and multimedia programming, event-based programming, agent-based programming, synchronous languages, high-productivity programming on massive parallel computers, models for mobile
Introduction to Programming Languages - Arvind Kumar Bansal - 2013-12-14

In programming courses, using the different syntax of multiple languages such as C++, Java, PHP, and Python, for the same abstraction often confuses students new to computer science. Introduction to Programming Languages separates programming language concepts from the restraints of multiple language syntax by discussing the concepts at an abstract level. Designed for one-semester undergraduate course, this classroom-tested book teaches the principles of programming language design and implementation. It presents: Common features of programming languages at an abstract level rather than a comparative level The implementation model and behavior of programming paradigms at abstract levels so that students understand the power and limitations of programming paradigms Language constructs at a paradigm level A holistic view of programming language design and behavior To make the book self-contained, the author introduces the necessary concepts of data structures and discrete structures from the perspective of programming language theory. The text covers classical topics, such as syntax and semantics, imperative programming, program structures, information exchange between subprograms, object-oriented programming, logic programming, and functional programming. It also explores newer topics, including dependency analysis, communicating sequential processes, concurrent programming constructs, web and multimedia programming, event-based programming, agent-based programming, synchronous languages, high-productivity programming on massive parallel computers, models for mobile computing, and much more. Along with problems and further reading in each chapter, the book includes in-depth examples and case studies using various languages that help students understand syntax in practical contexts.

The C Programming Language - Brian W. Kernighan - 1988

Introduces the features of the C programming language, discusses data types, variables, operators, control flow, functions, pointers, arrays, and structures, and looks at the UNIX system interface.

The C Programming Language - Brian W. Kernighan - 1988

Introduces the features of the C programming language, discusses data types, variables, operators, control flow, functions, pointers, arrays, and structures, and looks at the UNIX system interface.

The Cambridge Companion to Electronic Music - Nick Collins - 2017-10-31

Musicians are always quick to adopt and explore new technologies. The fast-paced changes wrought by electrification, from the microphone via the analogue synthesiser to the computer, have led to a wide range of new musical styles and techniques. Electronic music has grown to a broad field of investigation, taking in historical movements such as musique concrète and elektronische Musik, and contemporary trends such as electronic dance music and electronica. The first edition of this book won the 2009 Nicolas Bessaraboff Prize as it brought together researchers at the forefront of the sonic explorations empowered by electronic technology to produce accessible and insightful overviews of core topics and uncover some hitherto less publicised corners of worldwide movements. This updated and expanded second edition includes four entirely new chapters, as well as new original statements from globally renowned artists of the electronic music scene, and celebrates a diverse array of technologies, practices and music.

The Cambridge Companion to Electronic Music - Nick Collins - 2017-10-31

Musicians are always quick to adopt and explore new technologies. The fast-paced changes wrought by electrification, from the microphone via the analogue synthesiser to the computer, have led to a wide range of new musical styles and techniques. Electronic music has grown to a broad field of investigation, taking in historical movements such as musique concrète and elektronische Musik, and contemporary trends such as electronic dance music and electronica. The first edition of this book won the 2009 Nicolas Bessaraboff Prize as it brought together researchers at the forefront of the sonic explorations empowered by electronic technology to produce accessible and insightful overviews of core topics and uncover some hitherto less publicised corners of worldwide movements. This updated and expanded second edition includes four entirely new chapters, as well as new original statements from globally renowned artists of the electronic music scene, and celebrates a diverse array of technologies, practices and music.


A comprehensive undergraduate textbook covering both theory and practical design issues, with an emphasis on object-oriented languages. Each chapter, the book includes in-depth examples and case studies using various languages that help students understand syntax in practical contexts.


A comprehensive undergraduate textbook covering both theory and practical design issues, with an emphasis on object-oriented languages.

Abstracting Away the Machine - Mark Jones Lorenzo - 2019-08-22

At the dawn of the computer age, an elite development team at IBM built the most influential computer programming language in history: FORTRAN. Abstracting Away the Machine tells the epic story of how they did it—and what happened next. Over the past six decades, programming languages like ALGOL, BASIC, C/C++, COBOL, Java, LISP, LOGO, Pascal, PL/I, Python, Visual Basic, and many others opened up the field of computer science, and of computer programming in general, to the masses. But all of these high-level languages (HLLs)—computer languages that automate, hide, or otherwise abstract away the underlying operations of the machine—owe a huge debt of gratitude to FORTRAN (FORMula TRANslation), the first HLL to achieve widespread adoption. Many programming practices that we take for granted now came about as a result of FORTRAN. Created over a three-year period at IBM by a development team led by a brilliant but wayward mathematician named John W. Backus, FORTRAN was implemented initially on the IBM 704 mainframe computer in the mid-1950s, with dialects of the language quickly spreading thereafter to other platforms. FORTRAN’s powerful compiler, which translated human-readable code into code a computer could understand, produced incredibly clean and optimized standalone executable programs, all of which could be run independently of the compiler, setting the standard for decades to come—and overcoming the doubts of many skeptics along the way, who thought the FORTRAN project would never succeed. In the 1960s the language was standardized, with machine-dependent commands excised, and many platform-independent implementations followed. With the language now portable, able to run on any computer (at least in theory), FORTRAN, almost by accident, secured a stranglehold in the fields of science and engineering. The language also came to dominate in the supercomputing industry. But FORTRAN, a blue-collar workhorse more concerned with results than with style, was a victim of its own success—the language overrode the sense of its own limitations. High-level languages sprouted up, stealing the good bits from FORTRAN while simultaneously defining themselves in opposition to it. FORTRAN had become the foil. As these new languages pierced the cutting edge of the programming landscape, they redefined computing paradigms (e.g., with structured programming and object-oriented programming), and the language, FORTRAN—though eventually (and repeatedly) modernized and formally renamed Fortran—struggled to keep up through multiple standardization efforts, finally ceding significant ground to its successors as it slowly withdrew from the spotlight. To add insult to injury, even John Backus eventually turned against his creation. This is not a book on how to program in FORTRAN, nor is it a technical manual. Rather, the book is an account of how abstracting away the machine, which chronicles the complete history and development of the FORTRAN programming language, is set squarely on telling three interlocking stories: (1) How an elite group of computer trailblazers built FORTRAN, (2) Why the conditions at the time were ripe for them to succeed, and (3) What happened after they did. Tracing the long arc of FORTRAN’s development and maturation is integral to understanding not only the history of programming but also the state of computer science today. The birth of FORTRAN planted a seed that led to the full flowering of high-level languages, since FORTRAN overcame initial skepticism by demonstrating to the world that a well-made HLL really could abstract away the machine.

Abstracting Away the Machine - Mark Jones Lorenzo - 2019-08-22

At the dawn of the computer age, an elite development team at IBM built the most influential computer programming language in history: FORTRAN. Abstracting Away the Machine tells the epic story of how they did it—and what happened next. Over the past six decades, programming languages like ALGOL, BASIC, C/C++, COBOL, Java, LISP, LOGO, Pascal, PL/I, Python, Visual Basic, and many others opened up the field of computer science, and of computer programming in general, to the masses. But all of these high-level languages (HLLs)—computer languages that automate, hide, or otherwise abstract away the underlying operations of the machine—owe a huge debt of gratitude to FORTRAN (FORMula TRANslation), the first HLL to achieve widespread adoption. Many programming practices that we take for granted now came about as a result of FORTRAN. Created over a three-year period at IBM by a development team led by a brilliant but wayward mathematician named John W. Backus, FORTRAN was implemented initially on the IBM 704 mainframe computer in the mid-1950s, with dialects of the language quickly spreading thereafter to other platforms. FORTRAN’s powerful compiler, which translated human-readable code into code a computer could understand, produced incredibly clean and optimized standalone executable programs, all of which could be run independently of the compiler, setting the standard for decades to come—and overcoming the doubts of many skeptics along the way, who thought the FORTRAN project
programs, and to the formal verification language properties such as type language implementation, to the development of logics for reasoning about fundamental data types such as sums and products, polymorphic and abstract types, dynamic typing, dynamic dispatch, subtyping and refinement types, symbols and dynamic classification, parallelism and cost semantics, and concurrency and communication. The methods discussed deal with the development of a language implementation, to the development of logics for reasoning about programs, and to the formal verification language properties such as type safety. This thoroughly revised second edition includes exercises at the end of nearly every chapter and a new chapter on type refinements.

Concepts Of Programming Languages - Sebesta - 2008

Concepts Of Programming Languages - Sebesta - 2008


This book is a collection of refereed invited papers on the history of computing from the 1940s to the 1990s with one paper going back to look at Italian calculating/computing machines from the first century to the 20th century. The 22 papers cover a wide range of computing related topics such as specific early computer machines, their usage by users; software programming and operating systems; people involved in the theory, design and use of these computers; computer education; and conservation of computing technology. Most of the authors were actually involved in the events they describe and share their specific reflections on the history of computing.


This book is a collection of refereed invited papers on the history of computing from the 1940s to the 1990s with one paper going back to look at Italian calculating/computing machines from the first century to the 20th century. The 22 papers cover a wide range of computing related topics such as specific early computer machines, their construction, their use and their users; software programming and operating systems; people involved in the theory, design and use of these computers; computer education; and conservation of computing technology. Most of the authors were actually involved in the events they describe and share their specific reflections on the history of computing.

Practical Foundations for Programming Languages - Robert Harper - 2016-04-04

This text develops a comprehensive theory of programming languages based on type systems and structural operational semantics. Language concepts are precisely defined by their static and dynamic semantics, presenting the essential tools both intuitively and rigorously while relying on only elementary mathematics. These tools are used to analyze and prove properties of languages and provide the framework for combining and comparing language features. The broad range of concepts includes fundamental data types such as sums and products, polymorphic and abstract types, dynamic typing, dynamic dispatch, subtyping and refinement types, symbols and dynamic classification, parallelism and cost semantics, and concurrency and communication. The methods discussed deal with the development of logics for reasoning about programs, and to the formal verification language properties such as type safety. This thoroughly revised second edition includes exercises at the end of nearly every chapter and a new chapter on type refinements.

Practical Foundations for Programming Languages - Robert Harper - 2016-04-04

This text develops a comprehensive theory of programming languages based on type systems and structural operational semantics. Language concepts are precisely defined by their static and dynamic semantics, presenting the essential tools both intuitively and rigorously while relying on only elementary mathematics. These tools are used to analyze and prove properties of languages and provide the framework for combining and comparing language features. The broad range of concepts includes fundamental data types such as sums and products, polymorphic and abstract types, dynamic typing, dynamic dispatch, subtyping and refinement types, symbols and dynamic classification, parallelism and cost semantics, and concurrency and distribution. The methods are directly applicable to language implementation, to the development of logics for reasoning about programs, and to the formal verification language properties such as type safety. This thoroughly revised second edition includes exercises at the end of nearly every chapter and a new chapter on type refinements.

Endless Loop - Mark Jones Lorenzo - 2017-08-22

"Endless Loop" chronicles the complete history of the BASIC programming language--from its humble beginnings at Dartmouth College, to its widespread adoption and dominance in education, to its decline and subsequent modern rebirth. In the early morning hours of May 1, 1964, Dartmouth College birthed fraternal twins: BASIC, the Beginner's All-purpose Symbolic Instruction Code programming language, and, simultaneously, the Dartmouth Time-Sharing System (DTSS). It hadn't been an easy birth, and the gestation period was likewise difficult. BASIC was primarily the idea of one man, mathematicians professor John Kemeny, a brilliant Hungarian mathematician who had once been an assistant to Albert Einstein, while the DTSS satisfied the vision of another, mathematics and statistics professor Thomas Kurtz, who had brought a democratizing spirit to Dartmouth's campus in the form of free computing for all BASIC and DTSS caught on at Dartmouth quickly, with a vast majority of undergraduates (and faculty) making use of the computer system via teleletypewriters only several years after its inception. But by the early 1970s, with the personal computer revolution fast approaching, Kemeny and Kurtz began to lose control over BASIC as it achieved widespread popularity outside of Dartmouth. The language was being adapted to run on a wide variety of computers, some much too small to contain the full set of Dartmouth BASIC features. Most notably, Microsoft built its business on the back of ROM-based BASIC interpreters for a variety of microcomputers. Although the language was ubiquitous in schools by the early 1980s, it came under attack by such notable computers as scientific computer Edsger W. Dijkstra for its lack of structure as well as by Kemeny and Kurtz themselves, who viewed non-Dartmouth "Street BASIC" as blasphemous and saw it as their mission to right the ship of BASIC standardization and the release of TRUE BASIC. But by then it was too late: the era of BASIC's global dominance was over. In "Endless Loop," author Mark Jones Lorenzo documents the history and development of Dartmouth BASIC, True BASIC, Tiny BASIC, Microsoft BASIC--including Altair BASIC, AppleSoft BASIC, Color BASIC, Commodore BASIC, TRS-80 Level II BASIC, TI BASIC, IBM BASIC, QuickBASIC, Visual Basic, and Small Basic--as well as 9845 BASIC, Atari BASIC, BBC BASIC, CBASIC, Locomotive BASIC, MacBASIC, QB64, Simons' BASIC, Sinclair BASIC, SuperBASIC, and Turbo Basic/PowerBASIC, among a number of other implementations. The ascendancy of BASIC paralleled the emergence of the personal computer, so the story of BASIC is first and foremost a story--actually, many interlocking stories about computers--also a tale of talent people who built a language out of a set of primal ingredients: sweat, creativity, rivalry, jealousy, cooperation, and plain hard work, and then set the language loose in the world filled with unintended consequences. How those unintended consequences played out, leading to the demise of the most popular computer language the world has ever known, is the focus of "Endless Loop."

Endless Loop - Mark Jones Lorenzo - 2017-08-22

"Endless Loop" chronicles the complete history of the BASIC programming language--from its humble beginnings at Dartmouth College, to its widespread adoption and dominance in education, to its decline and subsequent modern rebirth. In the early morning hours of May 1, 1964, Dartmouth College birthed fraternal twins: BASIC, the Beginner's All-purpose Symbolic Instruction Code programming language, and, simultaneously, the Dartmouth Time-Sharing System (DTSS). It hadn't been an easy birth, and the gestation period was likewise difficult. BASIC was primarily the idea of one man, mathematicians professor John Kemeny, a brilliant Hungarian mathematician who had once been an assistant to Albert Einstein, while the DTSS satisfied the vision of another, mathematics and statistics professor Thomas Kurtz, who had brought a democratizing spirit to Dartmouth's campus in the form of free computing for all BASIC and DTSS caught on at Dartmouth quickly, with a vast majority of undergraduates (and faculty) making use of the computer system via teleletypewriters only several years after its inception. But by the early 1970s, with the personal computer revolution fast approaching, Kemeny and Kurtz began to lose control over BASIC as it achieved widespread popularity outside of Dartmouth. The language was being adapted to run on a wide
of Dartmouth BASIC features. Most notably, Microsoft built its business on the back of ROM-based BASIC interpreters for a variety of microcomputers. Although the language was ubiquitous in schools by the early 1980s, it came under attack by such notables as computer scientist Edsger W. Dijkstra for its lack of structure as well as by Kemeny and Kurtz themselves, who viewed non-Dartmouth “Street BASIC” as blasphemy and saw it as their mission to rid the world of such a ship through language standardization and the ascendance of True BASIC. By then it was too late; the era of BASIC’s global dominance was over. In “Endless Loop,” author Mark Jones Lorenzo documents the history and development of Dartmouth BASIC, True BASIC, Tiny BASIC, Microsoft BASIC—including Altair BASIC, Applesoft BASIC, Color BASIC, Commodore BASIC, TRS-80 Level II BASIC, TI BASIC, IBM BASIC/GW-BASIC, QuickBASIC/QBASIC, Visual Basic, and Small Basic—as well as 9845 BASIC, Atari BASIC, BBC BASIC, CBASIC, Locomotive BASIC, MacBASIC, Q864, Simons’ BASIC, Sinclair BASIC, SuperBASIC, and Turbo Basic/PowerBASIC, among a number of other implementations. The ascendancy of BASIC paralleled the emergence of the personal computer, so the story of BASIC is first and foremost a story—actually, many interlocking stories—about computers. But it is also a tale of talented people who built a new concept is motivated by programming examples and the more theoretical sections are driven by the needs of implementations. Each chapter is accompanied by numerous exercises and solutions, as well as a running implementation, available via the Web. Dependencies between chapters are explicitly identified, allowing readers to choose a variety of paths through the material. The core topics include the untyped lambda-calculus, simple type systems, type reconstruction, universal and existential polymorphism, subtyping, bounded quantification, recursive types, kinds, and type operators. Extended case studies develop a variety of approaches to modeling the features of object-oriented languages.

Types and Programming Languages - Benjamin C. Pierce - 2002-01-04
A comprehensive introduction to type systems and programming languages. A type system is a syntactic method for automatically checking the absence of certain erroneous behaviors by classifying program phrases according to the kinds of values they compute. The study of type systems—and of programming languages from a type-theoretic perspective—has important applications in software engineering, language design, high-performance compilers, and security. This text provides a comprehensive introduction both to type systems in computer science and to the basic theory of programming languages. The approach is pragmatic and operational; each new concept is motivated by programming examples and the more theoretical sections are driven by the needs of implementations. Each chapter is accompanied by numerous exercises and solutions, as well as a running implementation, available via the Web. Dependencies between chapters are explicitly identified, allowing readers to choose a variety of paths through the material. The core topics include the untyped lambda-calculus, simple type systems, type reconstruction, universal and existential polymorphism, subtyping, bounded quantification, recursive types, kinds, and type operators. Extended case studies develop a variety of approaches to modeling the features of object-oriented languages.

Semantics of Programming Languages and Model Theory - Manfred Droste - 1993-09-10
Fourteen papers presented at the conference on [title], held at the International Conference and Research Center for Computer Science, Schloss Dagstuhl, June 1991, as well as a few others submitted by colleagues unable to attend, reflect the interplay between algebra, logic, and semantics of programming languages. Among the topics are a formal specification of PARLOG, synthesis of nondeterministic asynchronous automata, observable modules and power domain constructions, the Smyth-completion of a quasi-uniform space, current trends in the semantics of data flow, and a theory of unary pairfunctions. Annotation copyright by Book News, Inc., Portland, OR.

Semantics of Programming Languages and Model Theory - Manfred Droste - 1993-09-10
Fourteen papers presented at the conference on [title], held at the International Conference and Research Center for Computer Science, Schloss Dagstuhl, June 1991, as well as a few others submitted by colleagues unable to attend, reflect the interplay between algebra, logic, and semantics of programming languages. Among the topics are a formal specification of PARLOG, synthesis of nondeterministic asynchronous automata, observable modules and power domain constructions, the Smyth-completion of a quasi-uniform space, current trends in the semantics of data flow, and a theory of unary pairfunctions. Annotation copyright by Book News, Inc., Portland, OR.

Semantics of Programming Languages and Model Theory - Manfred Droste - 1993-09-10
Fourteen papers presented at the conference on [title], held at the International Conference and Research Center for Computer Science, Schloss Dagstuhl, June 1991, as well as a few others submitted by colleagues unable to attend, reflect the interplay between algebra, logic, and semantics of programming languages. Among the topics are a formal specification of PARLOG, synthesis of nondeterministic asynchronous automata, observable modules and power domain constructions, the Smyth-completion of a quasi-uniform space, current trends in the semantics of data flow, and a theory of unary pairfunctions. Annotation copyright by Book News, Inc., Portland, OR.

Programming Languages for MIS - Hai Wang - 2014-01-23
Programming Languages for MIS: Concepts and Practice supplies a synopsis of the major computer programming languages, including C++, HTML, JavaScript, CSS, VB.NET, C# .NET, ASP.NET, PHP (with MySQL), XML (with XSLT, DTD, and XML Schema), and SQL. Ideal for undergraduate students in IS and IT programs, this textbook and its previous versions have been used in the authors’ classes for the past 15 years. Focused on web application development, the book considers client-side computing, server-side computing, and database applications. It emphasizes programming techniques, including structured programming, object-oriented programming, client-side programming, server-side computing, and graphical user interface. Introduces the basics of computer languages along with the key characteristics of all procedural computer languages Covers C++ and the fundamental concepts of the two programming paradigms: function-oriented and object-oriented Considers HTML, JavaScript, and CSS for web page development Presents VB.NET for graphical user interface development Introduces PHP, a popular open source programming language, and explains the use of the MySQL database in PHP Discusses XML and its companion languages, including XSTL, DTD, and XML Schema With this book, students learn the concepts shared by all computer languages as well as the unique features of each language. This self-contained text includes exercise questions, project requirements, report formats, and operational manuals of programming environments. A test bank and answers to exercise questions are also available upon qualified course adoption. This book supplies professors with the opportunity to structure a course consisting of two distinct modules: the teaching module and the project module. The teaching module covers an overview of representative computer languages. The project module provides students with the opportunity to gain hands-on experience with the various computer languages through projects.

Programming Languages for MIS - Hai Wang - 2014-01-23
Programming Languages for MIS: Concepts and Practice supplies a synopsis of the major computer programming languages, including C++, HTML, JavaScript, CSS, VB.NET, C# .NET, ASP.NET, PHP (with MySQL), XML (with XSLT, DTD, and XML Schema), and SQL. Ideal for undergraduate students in IS and IT programs, this textbook and its previous versions have been used in the authors’ classes for the past 15 years. Focused on web application development, the book considers client-side computing, server-side computing, and database applications. It emphasizes programming techniques, including structured programming, object-oriented programming, client-side programming, server-side computing, and graphical user interface. Introduces the basics of computer languages along with the key characteristics of all procedural computer languages Covers C++ and the fundamental concepts of the two programming paradigms: function-oriented and object-oriented Considers HTML, JavaScript, and CSS for web page development Presents VB.NET for graphical user interface development Introduces PHP, a popular open source programming language, and explains the use of the MySQL database in PHP Discusses XML and its companion languages, including XSTL, DTD, and XML Schema With this book, students learn the concepts shared by all computer languages as well as the unique features of each language. This self-contained text includes exercise questions, project requirements, report formats, and operational manuals of programming environments. A test bank and answers to exercise questions are also available upon qualified course adoption. This book supplies professors with the opportunity to structure a course consisting of two distinct modules: the teaching module and the project module. The teaching module supplies an overview of representative computer languages. The project module provides students with the opportunity to gain hands-on experience with the various computer languages through projects.

The Anatomy of Programming Languages - Alice E. Fischer - 1993
Covers the nature of language, syntax, modeling objects, names, expressions, functions, control structures, global control, logic programming, representation and semantics of types, modules, generics, and domains
types, including a treatment of the eager and lazy lambda-calculi.

The Anatomy of Programming Languages - Alco E. Fischer - 1993
Covers the nature of language, syntax, modeling objects, names, expressions, functions, control structures, global control, logic programming, representation and semantics of types, modules, generics, and domains

Handbook of Programming Languages - Peter H. Salus - 1998-12-01
A complete handbook covering the most widely used object-oriented programming languages with comprehensive coverage of each language, including history, syntax, variables, tips and traps. Unique leaders in the field of object oriented programming provide insightful information about the language that they helped to create. The books in the bundle are "Handbook of Programming Languages, Vol. I" and "Handbook of Programming Languages, Vol. II".

Handbook of Programming Languages - Peter H. Salus - 1998-12-01
A complete handbook covering the most widely used object-oriented programming languages with comprehensive coverage of each language, including history, syntax, variables, tips and traps. Unique leaders in the field of object oriented programming provide insightful information about the language that they helped to create. The books in the bundle are "Handbook of Programming Languages, Vol. I" and "Handbook of Programming Languages, Vol. II".

The Formal Semantics of Programming Languages - Glynn Winskel - 1993-02-05
The Formal Semantics of Programming Languages provides the basic mathematical techniques necessary for those who are beginning a study of the semantics and logics of programming languages. These techniques will allow students to invent, formalize, and justify rules with which to reason about a variety of programming languages. Although the treatment is elementary, several of the topics covered are drawn from recent research, including the vital area of concurrency. The book contains many exercises ranging from simple to miniprojects. Starting with basic set theory, structural operational semantics is introduced as a way to define the meaning of programming languages along with associated proof techniques. Denotational and axiomatic semantics are illustrated on a simple language of while-programs, and fall proofs are given of the equivalence of the operational and denotational semantics and relative completeness of the axiomatic semantics. A proof of Gödel's incompleteness theorem, which emphasizes the impossibility of achieving a fully complete axiomatic semantics, is included. It is supported by an appendix providing an introduction to the theory of computability based on while-programs. Following a presentation of domain theory, the semantics and methods of proof for several functional languages are treated. The simplest language is that of recursion equations with both call-by-value and call-by-name evaluation. This work is extended to lan guages with higher and recursive evaluation. This work is extended to languages with higher and recursive

Throughout, the relationship between denotational and operational semantics is stressed, and the proofs of the correspondence between the operation and denotational semantics are provided. The treatment of recursive types - one of the more advanced parts of the book - relies on the use of information systems to represent domains. The book concludes with a chapter on parallel programming languages, accompanied by a discussion of methods for specifying and verifying nondeterministic and parallel programs.

The Formal Semantics of Programming Languages - Glynn Winskel - 1993-02-05
The Formal Semantics of Programming Languages provides the basic mathematical techniques necessary for those who are beginning a study of the semantics and logics of programming languages. These techniques will allow students to invent, formalize, and justify rules with which to reason about a variety of programming languages. Although the treatment is elementary, several of the topics covered are drawn from recent research, including the vital area of concurrency. The book contains many exercises ranging from simple to miniprojects. Starting with basic set theory, structural operational semantics is introduced as a way to define the meaning of programming languages along with associated proof techniques. Denotational and axiomatic semantics are illustrated on a simple language of while-programs, and fall proofs are given of the equivalence of the operational and denotational semantics and relative completeness of the axiomatic semantics. A proof of Gödel's incompleteness theorem, which emphasizes the impossibility of achieving a fully complete axiomatic semantics, is included. It is supported by an appendix providing an introduction to the theory of computability based on while-programs. Following a presentation of domain theory, the semantics and methods of proof for several functional languages are treated. The simplest language is that of recursion equations with both call-by-value and call-by-name evaluation. This work is extended to lan guages with higher and recursive evaluation. This work is extended to languages with higher and recursive

Throughout, the relationship between denotational and operational semantics is stressed, and the proofs of the correspondence between the operation and denotational semantics are provided. The treatment of recursive types - one of the more advanced parts of the book - relies on the use of information systems to represent domains. The book concludes with a chapter on parallel programming languages, accompanied by a discussion of methods for specifying and verifying nondeterministic and parallel programs.

History of Programming Languages II - Thomas J. Bergin - 1996
This comprehensive overview of programming languages, their history, current application, and future direction, is based on the proceedings of the second conference on the History of Programming Languages. Its contents include a summary of the HOPL conferences, plus sections addressing successful programming languages by some of the most prominent names in computing.

History of Programming Languages II - Thomas J. Bergin - 1996
This comprehensive overview of programming languages, their history, current application, and future direction, is based on the proceedings of the second conference on the History of Programming Languages. Its contents include a summary of the HOPL conferences, plus sections addressing successful programming languages by some of the most prominent names in computing.

Programming Language Explorations - Ray Toal - 2017-08-09
Programming Language Explorations is a tour of several modern programming languages as they exist today. The book teaches the fundamental language concepts using a language-by-language approach. As each language is presented, the authors introduce new concepts as they appear, and revisit familiar ones, comparing their implementation with those from languages seen in prior chapters. The goal is to present and explain common theoretical concepts of language design and usage, illustrated in the context of practical language overviews. Twelve languages have been carefully chosen to illustrate a wide range of programming styles and paradigms. The book introduces each language with a common trio of example programs, and continues with a brief tour of its basic elements, type system, functional forms, scoping rules, concurrency patterns, and sometimes, metaprogramming facilities. Each language chapter ends with a summary, pointers to open source projects, research materials for further study, and a collection of exercises, designed as further explorations. Following the twelve featured language chapters, the authors provide a brief tour of over two dozen additional languages, and a summary chapter bringing together many of the questions explored throughout the text. Targeted to both professionals and advanced college undergraduates looking to expand the range of languages and programming patterns they can apply in their work and studies, the book pays attention to modern programming practice, covers cutting-edge languages and patterns, and provides many runnable examples, all of which can be found in an online GitHub repository. The exploration style places this book between a tutorial and a reference, with a focus on understanding modern programming language design and usage. Instructors looking for material to supplement a programming languages or software engineering course may find the approach unconventional, but hopefully, a lot more fun.

Programming Language Explorations - Ray Toal - 2017-08-09
Programming Language Explorations is a tour of several modern programming languages in use today. The book teaches fundamental language concepts using a language-by-language approach. As each language is presented, the authors introduce new concepts as they appear, and revisit familiar ones, comparing their implementation with those from languages seen in prior chapters. The goal is to present and explain common theoretical concepts of language design and usage, illustrated in the context of practical language overviews. Twelve languages have been carefully chosen to illustrate a wide range of programming styles and paradigms. The book introduces each language with a common trio of example programs, and continues with a brief tour of its basic elements, type system, functional forms, scoping rules, concurrency patterns, and sometimes, metaprogramming facilities. Each language chapter ends with a summary, pointers to open source projects, research materials for further study, and a collection of exercises, designed as further explorations. Following the twelve featured language chapters, the authors provide a brief tour of over two dozen additional languages, and a summary chapter bringing together many of the questions explored throughout the text. Targeted to both professionals and advanced college undergraduates looking to expand the range of languages and programming patterns they can apply in their work and studies, the book pays attention to modern programming practice, covers cutting-edge languages and patterns, and provides many runnable examples, all of which can be found in an online GitHub repository. The exploration style places this book between a tutorial and a reference, with a focus on understanding modern programming language design and usage. Instructors looking for material to supplement a programming languages or software engineering course may find the approach unconventional, but hopefully, a lot more fun.

The Formal Semantics of Programming Languages - Glynn Winskel - 1993-02-05
The Formal Semantics of Programming Languages provides the basic mathematical techniques necessary for those who are beginning a study of the semantics and logics of programming languages. These techniques will allow students to invent, formalize, and justify rules with which to reason about a variety of programming languages. Although the treatment is elementary, several of the topics covered are drawn from recent research, including the vital area of concurrency. The book contains many exercises ranging from simple to miniprojects. Starting with basic set theory, structural operational semantics is introduced as a way to define the meaning of programming languages along with associated proof techniques. Denotational and axiomatic semantics are illustrated on a simple language of while-programs, and fall proofs are given of the equivalence of the operational and denotational semantics and relative completeness of the axiomatic semantics. A proof of Gödel's incompleteness theorem, which emphasizes the impossibility of achieving a fully complete axiomatic semantics, is included. It is supported by an appendix providing an introduction to the theory of computability based on while-programs. Following a presentation of domain theory, the semantics and methods of proof for several functional languages are treated. The simplest language is that of recursion equations with both call-by-value and call-by-name evaluation. This work is extended to lan guages with higher and recursive evaluation. This work is extended to languages with higher and recursive evaluation. This work is extended to languages with higher and recursive evaluation. This work is extended to languages with higher and recursive evaluation. This work is extended to languages with higher and recursive
Foundations of Programming Languages - John C. Mitchell - 1996

Programming languages embody the pragmatics of designing software systems, and also the mathematical concepts which underlie them. Anyone who wants to know how, for example, object-oriented programming rests upon a firm foundation in logic should read this book. It guides one sorely needed through the rich variety of basic programming concepts developed over the past forty years. – Robin Milner, Professor of Computer Science, The Computer Laboratory, Cambridge University

Beginning C# 3.0 - Jack Purdum - 2008-08-11

Learn all the basics of C# 3.0 from Beginning C# 3.0: An Introduction to Object Oriented Programming, a book that presents introductory information in an intuitive format. If you have no prior programming experience but want a thorough, easy-to-understand introduction to C# and Object Oriented Programming, this book is an ideal guide. Using the tutorials and hands-on coding examples, you can discover tried and true tricks of the trade, understand design concepts, employ debugging aids, and design and write C# programs that are functional and that embody safe programming practices.

Real World OCaml - Yaron Minsky - 2013-11-04

This fast-moving tutorial introduces you to OCaml, an industrial-strength programming language designed for expressiveness, safety, and speed. Through the book's many examples, you'll quickly learn how OCaml stands out as a tool for writing fast, succinct, and readable systems code. Real World OCaml takes you through the concepts of the language at a brisk pace, and then helps you explore the tools and techniques that make OCaml an effective and practical tool. In the book’s third section, you’ll delve deep into the details of the compiler toolchain and OCaml’s simple and efficient runtime system. Learn the foundations of the language, such as higher-order functions, algebraic data types, and modules. Explore advanced features such as functors, first-class modules, and objects. Then, leverage OCaml's expressive and powerful approach to abstraction and modularity. Tackle practical programming problems from command-line parsing to asynchronous network programming. Examine profiling and interactive debugging techniques with tools such as GNU gdb.

Foundations of Programming Languages - John C. Mitchell - 1996

“Programming languages embody the pragmatics of designing software systems, and also the mathematical concepts which underlie them. Anyone who wants to know how, for example, object-oriented programming rests upon a firm foundation in logic should read this book. It guides one sorely needed through the rich variety of basic programming concepts developed over the past forty years.” – Robin Milner, Professor of Computer Science, The Computer Laboratory, Cambridge University

History of Computing in the Twentieth Century - Nicholas Metropolis - 2014-06-28

This clearly written textbook provides an accessible introduction to the three programming paradigms of object-oriented/impertative, functional, and logic programming. Highly interactive in style, the text encourages learning through practice, offering test exercises for each topic covered. Review questions and programming projects are also presented, to help reinforce the concepts outside of the classroom. This updated and revised new edition features new material on the Java implementation of the JCoCo virtual machine. Topics and features: includes review questions and solved practice exercises, with supplementary code and support files available from an associated website; presents an historical perspective on the models of computation used in implementing the programming languages used today; provides the foundations for understanding how the syntax of a language is formally defined by a grammar; illustrates how programs execute at the level of assembly language, through the implementation of a stack-based Python virtual machine called JCoCo and a Python disassembler; introduces object-oriented languages through examples in Java, functional programming with Standard ML, and programming using the logic language Prolog; describes a case study involving the development of a compiler for the high level functional language Small, a robust subset of Standard ML. Undergraduate students of computer science will find this engaging textbook to be an invaluable guide to the skills and tools needed to become a better programmer. While the text assumes some background in an imperative language, and prior coverage of the basics of data structures, the hands-on approach to following the story will enable the reader to quickly grasp the essentials of programming languages, frameworks, and architectures.

Foundations of Programming Languages - Kent D. Lee - 2017-12-10

This clearly written textbook provides an accessible introduction to the three programming paradigms of object-oriented/impertative, functional, and logic programming. Highly interactive in style, the text encourages learning through practice, offering test exercises for each topic covered. Review questions and programming projects are also presented, to help reinforce the concepts outside of the classroom. This updated and revised new edition features new material on the Java implementation of the JCoCo virtual machine. Topics and features: includes review questions and solved practice exercises, with supplementary code and support files available from an associated website; presents an historical perspective on the models of computation used in implementing the programming languages used today; provides the foundations for understanding how the syntax of a language is formally defined by a grammar; illustrates how programs execute at the level of assembly language, through the implementation of a stack-based Python virtual machine called JCoCo and a Python disassembler; introduces object-oriented languages through examples in Java, functional programming with Standard ML, and programming using the logic language Prolog; describes a case study involving the development of a compiler for the high level functional language Small, a robust subset of Standard ML. Undergraduate students of computer science will find this engaging textbook to be an invaluable guide to the skills and tools needed to become a better programmer. While the text assumes some background in an imperative language, and prior coverage of the basics of data structures, the hands-on approach to following the story will enable the reader to quickly grasp the essentials of programming languages, frameworks, and architectures.
The Structure of Typed Programming Languages - David A. Schmidt - 1994

The Structure of Typed Programming Languages describes the fundamental syntactic and semantic features of modern programming languages, carefully spelling out their impacts on language design. Using classical and recent research from lambda calculus and type theory, it presents a rational reconstruction of the Algol-like imperative languages such as Pascal, Ada, and Modula-3, and the higher-order functional languages such as Scheme and ML. David Schmidt’s text is based on the premise that although few programmers ever actually design a programming language, it is important for them to understand how restructuring techniques in a reconstruction of existing programming languages and in the design of new ones allows programmers and would-be programmers to see why existing languages are structured the way they are and how new languages can be built using variations on standard themes. The text is unique in its tutorial presentation of higher-order lambda calculus and intuitionistic type theory. The latter in particular reveals that a programming language is a logic in which its typing system defines the propositions of the logic and its well-typed programs constitute the proofs of the propositions. The Structure of Typed Programming Languages is designed for use in a first or second course on principles of programming languages. It assumes a basic knowledge of programming languages and mathematics equivalent to a course based on books such as Friedman, Wand, and Haynes: Essentials of Programming Languages. As Schmidt covers both the syntax and the semantics of programming languages, his text provides a perfect precursor to a more formal presentation of programming language semantics such as Gunter’s Semantics of Programming Languages.

Principles of Programming Languages - Bruce J. MacLennan - 1999

In-depth case studies of representative languages from five generations of programming language design (Fortran, Algol-60, Pascal, Ada, LISP, Smalltalk, and Prolog) are used to illustrate larger themes.—BOOK JACKET.

Principles of Programming Languages - Bruce J. MacLennan - 1999

In-depth case studies of representative languages from five generations of programming language design (Fortran, Algol-60, Pascal, Ada, LISP, Smalltalk, and Prolog) are used to illustrate larger themes.—BOOK JACKET.

The Study of Programming Languages - Ryan D. Stansifer - 1995

For one-semester, senior/graduate-level courses in Programming Languages. Rigorous, thorough, and foundational, this text reveals the character of programming languages as a field of study and explores some of the interesting, important, and conceptually more challenging topics that are often ignored by other texts on the subject.

The Study of Programming Languages - Ryan D. Stansifer - 1995

For one-semester, senior/graduate-level courses in Programming Languages. Rigorous, thorough, and foundational, this text reveals the character of programming languages as a field of study and explores some of the interesting, important, and conceptually more challenging topics that are often ignored by other texts on the subject.

Nim in Action - Dominik Picheta - 2017-08-04

Summary Nim is a multi-paradigm language that offers powerful customization options with the ability to compile to everything from C to JavaScript. In Nim in Action you’ll learn how Nim compares to other languages in style and performance, master its structure and syntax, and discover unique features. Purchase of the print book includes a free eBook in PDF, Kindle, and ePub formats from Manning Publications. About the Technology Nim is a multi-paradigm programming language that offers powerful customization options with the ability to compile to everything from C to JavaScript. It can be used in any project and illustrates that you don’t have to sacrifice performance for expressiveness! About the Book Nim in Action is your guide to application development in Nim. You’ll learn how Nim compares to other languages in style and performance, master its structure and syntax, and discover unique features. By carefully walking through a Twitter clone and other real-world examples, you’ll see just how Nim can be used every day while also learning how to tackle concurrency, package finished applications, and interface with other languages. With the best practices and rich examples in this book, you’ll be able to start using Nim today. What’s Inside Language features and implementation Nimbble package manager Asynchronous I/O Interfacing with C and JavaScript Metaprogramming About the Reader For developers comfortable with mainstream languages like Java, Python, C++ or C#. About the Author Dominik Picheta is one of the principal developers of Nim and author of the Nimbble package manager. Summary PART 1 - THE BASICS OF NIM Why Nim? Getting started PART 2 - NIM IN PRACTICE 3 Writing a chat application 4 A tour through the standard library 5 Package management 6 Parallelism 7 Building a Twitter clone PART 3 - ADVANCED CONCEPTS 8 Interfacing with other languages 9 Metaprogramming

Design Concepts in Programming Languages - Franklyn Turbak - 2007-18

Key ideas in programming language design and implementation explained using a simple and concise framework; a comprehensive introduction suitable for use as a textbook or a reference for researchers. Hundreds of programming languages are in use today—scripting languages for Internet commerce, user interface programming tools, spreadsheet macros, page format specification languages, and many others. A programming language is a metaprogramming activity that bears certain similarities to programming in a regular language, with clarity and simplicity even more important than in ordinary programming. This comprehensive text uses a simple and concise framework to teach key ideas in programming language design and implementation. The book’s unique approach is based on a family of syntactically similar, yet typical languages that allow other students to explore programming language concepts systematically. It takes as premise and starting point the idea that when language behaviors become incredibly complex, the description of the behaviors must be incredibly simple. The book presents a set of tools (a mathematical metalamguage, abstract syntax, operational and denotational semantics) and uses it to explore a comprehensive set of programming language design dimensions, including
readable and informal but rigorous coverage of the gamut of programming type reconstruction, polymorphism, effects), and pragmatics (compilation, garbage collection, examples and exercises offer students opportunities to apply the foundational ideas explained in the text. Specialized topics and code that implements many of the algorithms and compilation methods in the book can be found on the book's Web site, along with additional material as a section on concurrency and proofs of the theorems in the text. The book is suitable as a text for an introductory graduate or advanced undergraduate programming languages course; it can also serve as a reference for researchers and practitioners.

**Design Concepts in Programming Languages** - Franklin Turbak - 2008-07-18

Key ideas in programming language design and implementation explained using a simple and concise framework; a comprehensive introduction suitable for use as a textbook or a reference for researchers. Hundreds of programming languages are in use today—scripting languages for Internet commerce, user interface programming tools, spreadsheet macros, page format specification languages, and many others. Designing a programming language is a metaprogramming activity that bears certain similarities to programming in a regular language, with clarity and simplicity even more important than in ordinary programming. This comprehensive text uses a simple and concise framework to teach key ideas in programming language design and implementation. The book's unique approach is based on a family of syntactically simple pedagogical languages that allow students to explore programming language concepts systematically. It takes as premise and starting point the idea that when language behaviors become incredibly complex, the description of the behaviors must be incredibly simple. The book presents a set of tools (a mathematical metalanguage, abstract syntax, operational and denotational semantics) and uses it to explore a comprehensive set of program language design dimensions, including dynamic semantics (naming, state, control, data), static semantics (types, type reconstruction, polymorphism, effects), and pragmatics (compilation, garbage collection). The many examples and exercises offer students opportunities to apply the foundational ideas explained in the text. Specialized topics and code that implements many of the algorithms and compilation methods in the book can be found on the book's Web site, along with such additional material as a section on concurrency and proofs of the theorems in the text. The book is suitable as a text for an introductory graduate or advanced undergraduate programming languages course; it can also serve as a reference for researchers and practitioners.

**An Experiential Introduction to Principles of Programming Languages** - Hridesh Rajan - 2022-04-26

Programming Linguistics - David Hillel Gelernter - 1990

Programming Linguistics examines a wide range of programming language designs from Fortran to the newest research languages, to discover their common patterns, relationships, and antecedents. In studying the evolution of programming languages, the authors are also studying a series of answers to the central (and still unanswered) questions of what programs are and how they should be built. Programming Linguistics approaches language design as an attempt to define the nature of programming and the shape and structure of programs, rather than as the attempt to solve a series of narrow, disjoint technical problems. It emphasizes the structural engineering rather than mathematical approach to programming, the importance of aesthetics and elegance in the success of language design, and provides an integrated treatment of concurrency and parallelism. Its language designs are based on a simple and general programming model called the Ideal Software Machine. There are helpful exercises throughout. David Gelernter is an Associate Professor in the Department of Computer Science at Yale University. Suresh Jagannathan is an Associate Research Scientist at Yale.

Programming Linguistics - David Hillel Gelernter - 1990

Programming Linguistics examines a wide range of programming language designs from Fortran to the newest research languages, to discover their common patterns, relationships, and antecedents. In studying the evolution of programming languages, the authors are also studying a series of answers to the central (and still unanswered) questions of what programs are and how they should be built. Programming Linguistics approaches language design as an attempt to define the nature of programming and the shape and structure of programs, rather than as the attempt to solve a series of narrow, disjoint technical problems. It emphasizes the structural engineering rather than mathematical approach to programming, the importance of aesthetics and elegance in the success of language design, and provides an integrated treatment of concurrency and parallelism. Its

language designs are based on a simple and general programming model called the Ideal Software Machine. There are helpful exercises throughout. David Gelernter is an Associate Professor in the Department of Computer Science at Yale University. Suresh Jagannathan is an Associate Research Scientist at Yale.

**Selected Papers on Computer Languages** - Donald Ervin Knuth - 2003

This volume, sixth in a series of collected works by world-renowned computer scientist Donald E. Knuth, assembles approximately two dozen of his pioneering contributions to the field of computer languages, including papers on ALGOL, SOL, RUNCIBLE, and FORTRAN. Papers on the early development of programming languages, the history of writing compilers, the characterization of parenthesis languages, and the semantics of context-free languages are also included.

**Selected Papers on Computer Languages** - Donald Ervin Knuth - 2003

This volume, sixth in a series of collected works by world-renowned computer scientist Donald E. Knuth, assembles approximately two dozen of his pioneering contributions to the field of computer languages, including papers on ALGOL, SOL, RUNCIBLE, and FORTRAN. Papers on the early development of programming languages, the history of writing compilers, the characterization of parenthesis languages, and the semantics of context-free languages are also included.

**A Review of Programming Paradigms Throughout the History** - Elad Shalom - 2015-11-30

This book aims to review most of the different programming paradigms that were common since the 1960's. This book discusses object oriented, functionality programming, real time and many more. A programming paradigm is a fundamental style of computer programming, serving as a way of building the structure and elements of computer programs. Various programming languages have different capabilities and styles and they are defined by their supported programming paradigms. Some programming languages follow only one paradigm, while others support multiple paradigms. This book is meant to get the reader familiarized with different programming paradigms and their use. 'Those who do not learn history are doomed to repeat it.' - George Santayana

**A Review of Programming Paradigms Throughout the History** - Elad Shalom - 2015-11-30

This book aims to review most of the different programming paradigms that were common since the 1960's. This book discusses object oriented, functionality programming, real time and many more. A programming paradigm is a fundamental style of computer programming, serving as a way of building the structure and elements of computer programs. Various programming languages have different capabilities and styles and they are defined by their supported programming paradigms. Some programming languages follow only one paradigm, while others support multiple paradigms. This book is meant to get the reader familiarized with different programming paradigms and their use. 'Those who do not learn history are doomed to repeat it.' - George Santayana

**The Art of UNIX Programming** - Eric S. Raymond - 2003-09-23

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.

**The Art of UNIX Programming** - Eric S. Raymond - 2003-09-23

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.

**Programming Language Pragmatics** - Michael Lee Scott - 2006

Accompanying CD-ROM contains "advanced/optional content, hundreds of working examples, an active search facility, and live links to manuals, tutorials, compilers, and interpreters on the World Wide Web."--Page 4 of cover.