

# F# Programming

---

## Contents

---

1. **Preface** - About this book and its authors.
2. **Introduction** - Introducing the F# Programming Language.

## F# Basics

*This section is suitable for complete beginners to F# and Functional Programming in general.*

1. **Getting Set Up** - Installing F# on Windows, Linux and Mac.
2. **Basic Concepts** - A lightweight crash course in functional programming concepts.

## Working With Functions

*F# is a functional programming language. Not surprisingly, functions are a big part of the language, and mastering them is the first step to becoming an effective F# developer.*

1. **Declaring Values and Functions** - This article will show you how to declare simple functions, how to use type inference, and how to read F#'s 'arrow' notation.
2. **Pattern Matching Basics** - Pattern matching is used for control flow. Its conceptually similar to a switch statement in other languages, but orders of magnitude more powerful.
3. **Recursion and Recursive Functions** - A recursive function is a special type of function which calls itself.
4. **Higher Order Functions** - Higher order functions take and return functions as values. Combined with currying, it allows programmers to write powerful and expressive programs.

## Immutable Data Structures

*"Data structure" is a fancy word which refers to anything that helps programmers group and represent related values in useful, logical units. F# has a number of built-in data structures which include tuples, records, lists, unions, and a number of others.*

1. **Option Types** - Option types are simple, lightweight data structures which are commonly used to represent successful or failed computation.
2. **Tuples and Records** - Tuples and records are simple data structures which allow programmers to group related values together into a single unit.
3. **Lists** - A list represents an ordered group of values. F#'s List library has extensive support for manipulating and working with lists.
4. **Sequences** - Sequence expressions represent sequences of data computed on-demand.
5. **Sets and Maps** - Sets are conceptually similar to lists, except they cannot hold duplicate items. Maps allows programmers to relate *keys* to *values* and find items in the collection very efficiently.

6. **Discriminated Unions** - Discriminated unions represent a finite, well-defined set of choices. Discriminated unions are often the tool of choice building up more complicated data structures including linked lists and a wide range of trees.

## Imperative Programming

*F# is an "impure" programming language, meaning it allows programmers to write functions with side-effects and mutable state, very similar to the programming style used by imperative programming languages such as C# and Java.*

1. **Mutable Data** - By default, variables in F# are immutable. However, F# supports mutable variables through mutable fields and ref cells.
2. **Control Flow** - Decision making and Loops.
3. **Arrays** - Arrays are ubiquitous mutable data structure used in imperative programming languages.
4. **Mutable Collections** - Lists and Dictionaries.
5. **Basic I/O** - Reading and writing to files and the console window.
6. **Exception Handling** - Exception handling allows programmers to catch and handle errors whenever an application enters an invalid state.

## Object Oriented Programming

*F# is a CLI/.NET programming language. CLI is an object-oriented platform. One of the most important features of F# is its ability to mix and match styles: since the .NET platform is Object Oriented, with F#, you often work with objects.*

1. **Operator Overloading** - C#-like operator overloading.
2. **Classes** - classes and objects are the foundation of object-oriented programming (OOP). They are used to model actions, processes, and any conceptual entities in applications.
3. **Inheritance** - inheritance makes OOP code reusable. It allows programmers to create classes which inherit features from another class and add its own modifications.
4. **Interfaces** - interfaces abstract away the implementation details of a class by defining a template of methods an object must implement and expose publicly.
5. **Events** - events allow classes to send and receive messages between one another.
6. **Modules and Namespaces** - modules and namespaces are used to organize classes into groups of related functionality.

## F# Advanced

*F# is easy enough for beginners to learn as their first language, yet it provides a powerful set of tools which can be appreciated by experienced developers. This section describes advanced syntactic constructs and techniques often used in F# programs.*

1. **Units of Measure** - Units of measure attach metadata to floats, which allows floats to represent kilograms, pounds, Newtons, hectares, and so on.
2. **Caching** - Techniques to store computed values for efficient future retrieval. Also called Memoization.
3. **Active Patterns** - Active patterns allow programmers to wrap ad hoc values and objects in union-like structures for use in pattern matching.

4. **Advanced Data Structures** - Overview of techniques used to implement immutable data structures.
5. **Reflection** - Reflection allows programmers to inspect types and metadata in objects.
6. **Quotations** - Quotations convert arbitrary F# code into an abstract syntax tree.
7. **Computation Expressions** - Similar to monads in Haskell, computation expressions are used to simplify code written continuation-passing style.

## Multi-threaded and Concurrent Applications

*Multi-threading is becoming increasingly important with the development of multi-core processors. Functional programmers can take advantage of immutable data structures to make massively scalable, concurrent applications that are simple and easy to write.*

1. **Async Workflows** - F#'s async primitive is fundamental for writing functional, simple multi-threaded code.
2. **MailboxProcessor Class** - Mailboxes are used to implement "message-passing concurrency," a style of concurrent programming used in massively parallel applications consisting of 10s or 1000s of independent nodes.

## F# Tools

1. **Lexing and Parsing** - FsLex and FsYacc, lexer/parser generators based on the GNU Bison family of generators, are used to implement custom grammars and domain-specific languages in F#.

## Authors

---

*If you have contributed to this book, please add your name to this list.*

1. [Awesome Princess](#)

## Notes

---

[Notes for Contributors](#)  
[Acknowledgments](#)  
[Exercise Solutions](#)  
[License](#)

## Resources

---

- [F# Language Reference on Microsoft Docs \(https://docs.microsoft.com/en-us/dotnet/fsharp/language-reference/\)](https://docs.microsoft.com/en-us/dotnet/fsharp/language-reference/)
- [F# Homepage on Microsoft Research \(http://www.research.microsoft.com/en-us/um/cambridge/projects/fsharp/\)](http://www.research.microsoft.com/en-us/um/cambridge/projects/fsharp/)
- [Microsoft F# Developer Center \(http://www.fsharp.net/\)](http://www.fsharp.net/)
- [Real-World F# Articles on MSDN \(http://tomasp.net/blog/real-world-book-msdn.aspx\)](http://tomasp.net/blog/real-world-book-msdn.aspx)
- [Language Specification \(http://research.microsoft.com/en-us/um/cambridge/projects/fsharp/manual/spec.html\)](http://research.microsoft.com/en-us/um/cambridge/projects/fsharp/manual/spec.html)

- [Language Specification PDF \(http://research.microsoft.com/en-us/um/cambridge/projects/fsharp/manual/spec.pdf\)](http://research.microsoft.com/en-us/um/cambridge/projects/fsharp/manual/spec.pdf)
- [F# Component Design Guidelines \(https://fsharp.github.com/fsharp-component-design-guidelines.htm\)](https://fsharp.github.com/fsharp-component-design-guidelines.htm)
- [F# Component Design Guidelines PDF \(http://research.microsoft.com/en-us/um/cambridge/projects/fsharp/manual/fsharp-component-design-guidelines.pdf\)](http://research.microsoft.com/en-us/um/cambridge/projects/fsharp/manual/fsharp-component-design-guidelines.pdf)
- [hubFS F# Community \(http://cs.hubfs.net/\)](http://cs.hubfs.net/)
- [fpish community-driven events and learning material \(http://fpish.net/\)](http://fpish.net/)
- [Community for F# monthly, virtual user group \(http://c4fsharp.net/\)](http://c4fsharp.net/)
- [F# Snippets \(http://fssnip.net/\)](http://fssnip.net/)
- [Try F# online \(http://www.tryfsharp.org/\)](http://www.tryfsharp.org/)
- [Using the F# Language for Teaching \(http://functional-teaching.net/\)](http://functional-teaching.net/)
- [Cross-platform and other F# extensions \(http://functional-variations.net/\)](http://functional-variations.net/)
- [F# source code and community projects on GitHub \(https://github.com/fsharp\)](https://github.com/fsharp)
- [F# cross-platform packages and samples \(http://fsxplat.codeplex.com/\)](http://fsxplat.codeplex.com/)
- [The F# Survival Guide \(http://web.archive.org/web/20110715231625/http://www.ctocorner.com/fsharp/book/default.aspx\)](http://web.archive.org/web/20110715231625/http://www.ctocorner.com/fsharp/book/default.aspx)
- [F# for fun and profit \(http://fsharpforfunandprofit.com/\)](http://fsharpforfunandprofit.com/)
- [Objective Caml](#)
- [F# for game development \(http://sharp-gamedev.blogspot.com/\)](http://sharp-gamedev.blogspot.com/)
- [Learning F# Through Game Development with XNA \(http://fsharpgamedev.codeplex.com/\)](http://fsharpgamedev.codeplex.com/)
- [F# Tutorials for Beginning through Advanced Learners \(http://fsharp4u.com/\)](http://fsharp4u.com/)

---

Retrieved from "[https://en.wikibooks.org/w/index.php?title=F\\_Sharp\\_Programming&oldid=3747641](https://en.wikibooks.org/w/index.php?title=F_Sharp_Programming&oldid=3747641)"

---

This page was last edited on 17 October 2020, at 00:07.

Text is available under the Creative Commons Attribution-ShareAlike License.; additional terms may apply. By using this site, you agree to the Terms of Use and Privacy Policy.