WIKIPEDIA

F Sharp (programming language)

F# (pronounced **F sharp**) is a functional-first, general purpose, strongly typed, multi-paradigm programming language that encompasses functional, imperative, and object-oriented programming methods. F# is most often used as a cross-platform Common Language Infrastructure (CLI) language on .NET, but it can also generate JavaScript[9] and graphics processing unit (GPU) code.[10]

F# is developed by the F# Software Foundation,[11] Microsoft and open contributors. An open source, cross-platform compiler for F# is available from the F# Software Foundation.[12] F# is a fully supported language in Visual Studio[13] and JetBrains Rider.[14] Plug-ins supporting F# exist for many widely used editors, most notably the Ionide (https://ionide.io) extension for Visual Studio Code, and integrations for other editors such as Vim, and Emacs.

F# is a member of the ML language family and originated as a .NET Framework implementation of a core of the programming language OCaml.[6][7] It has also been influenced by C#, Python, Haskell,[5] Scala, and Erlang.

F#

F# logomark

Paradigm	Multi-paradigm: functional, imperative, object-oriented, metaprogramming, reflective, concurrent
Family	ML
Designed by	Don Syme, Microsoft Research
Developer	Microsoft, The F# Software Foundation
First appeared	2005, version 1.0
Stable release	6.0[1] / 19 October 2021
Preview release	5.0 preview / April 2, 2019[2]
Typing discipline	Static, strong, inferred
OS	Cross-platform: .NET, .NET Framework, Mono
License	MIT License[3][4]
Filename extensions	.fs, .fsi, .fsx, .fsscript
Website	fsharp.org (http://fsharp.org)

Influenced by

C#, Erlang, Haskell,[5] ML, OCaml,[6][7] Python, Scala

Contents

Influenced
C#,[8] Elm, F*, LiveScript
F Sharp Programming at Wikibooks

History

Versions

In the course of its development, the language has gone through several versions:

Version	Language specification	Date	Platforms	Runtime
F# 1.x		May 2005[15]	Windows	.NET 1.0 - 3.5
F# 2.0	August 2010 (http://fsharp.org/specs/language-spec/index.html#f-20)	April 2010[16]	Linux, macOS, Windows	.NET 2.0 - 4.0, Mono
F# 3.0	November 2012 (http://fsharp.org/specs/language-spec/index.html#f-30)	August 2012[17]	Linux, macOS, Windows; JavaScript,[9] GPU[10]	.NET 2.0 - 4.5, Mono
F# 3.1	November 2013 (http://fsharp.org/specs/language-spec/index.html#f-31)	October 2013[18]	Linux, macOS, Windows; JavaScript,[9] GPU[10]	.NET 2.0 - 4.5, Mono
F# 4.0	January 2016 (http://fsharp.org/specs/language-spec/index.html#f-40)	July 2015[19]		
F# 4.1	May 2018 (http://fsharp.org/specs/language-spec/index.html#f-41)	March 2017[20]	Linux, macOS, Windows, JavaScript,[9] GPU[10]	.NET 3.5 - 4.6.2, .NET Core, Mono
F# 4.5		August 2018[21]	Linux, macOS, Windows, JavaScript,[9] GPU[10]	.NET 4.5 - 4.7.2,[22] .NET Core SDK 2.1.400[23]
F# 4.6		March 2019[24]	Linux, macOS, Windows, JavaScript,[9] GPU[10]	.NET 4.5 - 4.7.2,[25] .NET Core SDK 2.2.300[26]
F# 4.7		September 2019[27]	Linux, macOS, Windows, JavaScript,[9] GPU[10]	.NET 4.5 - 4.8,[28] .NET Core SDK 3.0.100[29]
F# 5.0		November 2020[30]	Linux, macOS, Windows, JavaScript,[9] GPU[10]	.NET SDK 5.0.100[31]
F# 6.0		November 2021[32]	Linux, macOS, Windows, JavaScript,[9] GPU[10]	.NET SDK 6.0.100[33]

Language evolution

F# uses an open development and engineering process. The language evolution process is managed by Don Syme from Microsoft Research as the benevolent dictator for life (BDFL) for the language design, together with the F# Software Foundation. Earlier versions of the F# language were designed by Microsoft and Microsoft Research using a closed development process.

F# originates from Microsoft Research, Cambridge, UK. The language was originally designed and implemented by Don Syme,[6] according to whom in the fsharp team, they say the F is for "Fun".[34] Andrew Kennedy contributed to the design of units of measure.[6] The Visual F# Tools for Visual Studio are developed by Microsoft.[6] The F# Software Foundation developed the F# open-source compiler and tools, incorporating the open-source compiler implementation provided by the Microsoft Visual F# Tools team.[11]

Summary of versions

	Features added
F# 1.0	▪ Functional programming ▪ Discriminated unions ▪ Records ▪ Tuples ▪ Pattern matching ▪ Type abbreviations ▪ Object-oriented programming ▪ Structs ▪ Signature files ▪ Scripting files ▪ Imperative programming ▪ Modules (no functors) ▪ Nested modules ▪ .NET Interoperability
F# 2.0	▪ Active patterns ▪ Units of measure ▪ Sequence expressions ▪ Asynchronous programming ▪ Agent programming ▪ Extension members ▪ Named arguments ▪ Optional arguments ▪ Array slicing ▪ Quotations ▪ Native interoperability ▪ Computation expressions
F# 3.0[35]	▪ Type providers ▪ LINQ query expressions ▪ CLIMutable attribute ▪ Triple-quoted strings ▪ Auto-properties ▪ Provided units-of-measure
F# 3.1[36]	▪ Named union type fields ▪ Extensions to array slicing ▪ Type inference enhancements
F# 4.0[37]	▪ Printf on unitized values ▪ Extension property initializers ▪ Non-null provided types ▪ Primary constructors as functions ▪ Static parameters for provided methods ▪ Printf interpolation ▪ Extended #if grammar ▪ Tailcall attribute ▪ Multiple interface instantiations ▪ Optional type args ▪ Params dictionaries

F# 4.1[38]	• Struct tuples which inter-operate with C# tuples • Struct annotations for Records • Struct annotations for Single-case Discriminated Unions • Underscores in numeric literals • Caller info argument attributes • Result type and some basic Result functions • Mutually referential types and modules within the same file • Implicit "Module" syntax on modules with shared name as type • Byref returns, supporting consuming C# ref-returning methods • Error message improvements • Support for 'fixed'
F# 4.5[30]	• Versioning alignment of binary, package, and language • Support for 'Span<T>' and related types • Ability to produce 'byref' returns • The 'voidptr' type • The 'inref<'T>' and 'outref<'T>' types to represent readonly and write-only 'byref's • 'IsByRefLike' structs • 'IsReadOnly' structs • Extension method support for 'byref<'T>'/'inref<'T>'/'outref<'T>' • 'match!' keyword in computation expressions • Relaxed upcast with 'yield' in F# seq/list/array expressions • Relaxed indentation with list and array expressions • Enumeration cases emitted as public
F# 4.6	• Anonymous types
F# 4.7[39]	• Implicit yields • No more required double underscore • Indentation relaxations for parameters passed to constructors and static methods • 'nameof' function • Open static classes
F# 5.0[40]	• FSharp.Core now targets netstandard2.0 only • Package references in F# scripts • Support for Jupyter, nteract, and VSCode Notebooks • String Interpolation • Support for nameof • Open Type declarations • Enhanced Slicing • F# quotations improvements • Applicative Computation Expressions • Improved stack traces in F# async and other computation expressions • Improved .NET interop • Improved Map and Set performance in FSharp.Core • Improved compiler performance • Improved compiler analysis for library authors
F# 6.0[41]	• Tasks • Simpler indexing • Augments to "active patterns" • Overloaded custom operations in computation expressions • “as” patterns • Indentation syntax revisions

	- Additional implicit conversions - Additional implicit upcast conversions - Implicit integer conversions - First-class support for .NET-style implicit conversions - Optional warnings for implicit conversions - Formatting for binary numbers - Discards on use bindings - InlineIfLambda optimizer directive - Resumable code - Additional collection functions - Map has Keys and Values - Additional intrinsics for NativePtr - Additional numeric types with unit annotations - Informational warnings for rarely used symbolic operators

Language overview

Functional programming

While supporting object-oriented features available in C#, F# is strongly typed functional-first language with a large number of capabilities that are normally found only in functional programming languages. Together, these features allow F# programs to be written in a completely functional style and also allow functional and object-oriented styles to be mixed.

Examples of functional features are:

- Everything is an expression
- Type inference
- Anonymous functions with capturing semantics (i.e., closures)
- Immutable variables and objects
- Lazy evaluation support
- Higher-order functions
- Nested functions
- Currying
- Pattern matching
- Algebraic data types
- Tuples
- Monad pattern support

F# is an expression-based language using eager evaluation and also in some instances lazy evaluation. Every statement in F#, including `if` expressions, `try` expressions and loops, is a composable expression with a static type.[42] Functions and expressions that do not return any value have a return type of `unit`. F# uses the `let` keyword for binding values to a name.[42] For example:

```
let x = 3 + 4
```

binds the value 7 to the name `x`.

New types are defined using the `type` keyword. For functional programming, F# provides *tuple*, *record*, *discriminated union*, *list*, *option*, and *result* types.[42] A *tuple* represents a set of n values, where $n \geq 0$. The value n is called the arity of the tuple. A 3-tuple would be represented as `(A, B, C)`, where A, B, and C are values of possibly different types. A tuple can be used to store values only when the number of values is known at design-time and stays constant during execution.

A *record* is a type where the data members are named. Here is an example of record definition:

```
type R =
       { Name : string
        Age : int }
```

Records can be created as `let r = { Name="AB"; Age=42 }`. The `with` keyword is used to create a copy of a record, as in `{ r with Name="CD" }`, which creates a new record by copying `r` and changing the value of the `Name` field (assuming the record created in the last example was named `r`).

A discriminated union type is a type-safe version of C unions. For example,

```
type A =
    | UnionCaseX of string
    | UnionCaseY of int
```

Values of the union type can correspond to either union case. The types of the values carried by each union case is included in the definition of each case.

The *list* type is an immutable linked list represented either using a `head::tail` notation (`::` is the cons operator) or a shorthand as `[item1; item2; item3]`. An empty list is written `[]`. The *option* type is a discriminated union type with choices `Some(x)` or `None`. F# types may be generic, implemented as generic .NET types.

F# supports lambda functions and closures.[42] All functions in F# are first class values and are immutable.[42] Functions can be curried. Being first-class values, functions can be passed as arguments to other functions. Like other functional programming languages, F# allows function composition (computer science) using the >> and << operators.

F# provides *sequence expressions*[43] that define a sequence `seq { ... }`, list `[ ... ]` or array `[| ... |]` through code that generates values. For example,

```
seq { for b in 0 .. 25 do
          if b < 15 then
              yield b*b }
```

forms a sequence of squares of numbers from 0 to 14 by filtering out numbers from the range of numbers from 0 to 25. Sequences are generators – values are generated on-demand (i.e., are lazily evaluated) – while lists and arrays are evaluated eagerly.

F# uses pattern matching to bind values to names. Pattern matching is also used when accessing discriminated unions – the union is value matched against pattern rules and a rule is selected when a match succeeds. F# also supports *Active Patterns* as a form of extensible pattern matching.[44] It is used, for example, when multiple ways of matching on a type exist.[42]

F# supports a general syntax for defining compositional computations called *computation expressions*. Sequence expressions, asynchronous computations and queries are particular kinds of computation expressions. Computation expressions are an implementation of the monad pattern.[43]

Imperative programming

F# support for imperative programming includes

- `for` loops
- `while` loops
- arrays, created with the `[| ... |]` syntax
- hash table, created with the `dict [ ... ]` syntax or `System.Collections.Generic.Dictionary<_,_>` type.

Values and record fields can also be labelled as `mutable`. For example:

```
// Define 'x' with initial value '1'
let mutable x = 1
// Change the value of 'x' to '3'
x <- 3
```

Also, F# supports access to all CLI types and objects such as those defined in the `System.Collections.Generic` namespace defining imperative data structures.

Object-oriented programming

Like other Common Language Infrastructure (CLI) languages, F# can use CLI types through object-oriented programming.[42] F# support for object-oriented programming in expressions includes:

- Dot-notation, e.g., `x.Name`
- Object expressions, e.g., `{ new obj() with member x.ToString() = "hello" }`
- Object construction, e.g., `new Form()`
- Type tests, e.g., `x :? string`
- Type coercions, e.g., `x :?> string`
- Named arguments, e.g., `x.Method(someArgument=1)`
- Named setters, e.g., `new Form(Text="Hello")`
- Optional arguments, e.g., `x.Method(OptionalArgument=1)`

Support for object-oriented programming in patterns includes

- Type tests, e.g., `:? string as s`
- Active patterns, which can be defined over object types[44]

F# object type definitions can be class, struct, interface, enum, or delegate type definitions, corresponding to the definition forms found in C#. For example, here is a class with a constructor taking a name and age, and declaring two properties.

```
/// A simple object type definition
type Person(name : string, age : int) =
    member x.Name = name
    member x.Age = age
```

Asynchronous programming

F# supports asynchronous programming through *asynchronous workflows*.[45] An asynchronous workflow is defined as a sequence of commands inside an `async{ ... }`, as in

```
let asynctask =
    async { let req = WebRequest.Create(url)
            let! response = req.GetResponseAsync()
            use stream = response.GetResponseStream()
            use streamreader = new System.IO.StreamReader(stream)
            return streamreader.ReadToEnd() }
```

The `let!` indicates that the expression on the right (getting the response) should be done asynchronously but the flow should only continue when the result is available. In other words, from the point of view of the code block, it's as if getting the response is a blocking call, whereas from the point of view of the system, the thread won't be blocked and may be used to process other flows while the result needed for this one doesn't become available.

The async block may be invoked using the `Async.RunSynchronously` function. Multiple async blocks can be executed in parallel using the `Async.Parallel` function that takes a list of `async` objects (in the example, `asynctask` is an async object) and creates another async object to run the tasks in the lists in parallel. The resultant object is invoked using `Async.RunSynchronously`.[45] Inversion of control in F# follows this pattern.[45]

Parallel programming

Parallel programming is supported partly through the `Async.Parallel`, `Async.Start` and other operations that run asynchronous blocks in parallel.

Parallel programming is also supported through the `Array.Parallel` functional programming operators in the F# standard library, direct use of the `System.Threading.Tasks` task programming model, the direct use of .NET thread pool and .NET threads and through dynamic translation of F# code to alternative parallel execution engines such as GPU[10] code.

Units of measure

The F# type system supports units of measure checking for numbers.[46] The units of measure feature integrates with F# type inference to require minimal type annotations in user code.[47]

Metaprogramming

F# allows some forms of syntax customizing via metaprogramming to support embedding custom domain-specific languages within the F# language, particularly through computation expressions.[42]

F# includes a feature for run-time meta-programming called quotations.[48] A quotation expression evaluates to an abstract syntax tree representation of the F# expressions. Similarly, definitions labelled with the `[<ReflectedDefinition>]` attribute can also be accessed in their quotation form. F# quotations are used for various purposes including to compile F# code into JavaScript[9] and GPU[10] code. (Quotations represent their F# code expressions as data for use by other parts of the program while requiring it to be syntactically correct F# code).

Information-rich programming

F# 3.0 introduced a form of compile-time meta-programming through statically extensible type generation called F# type providers.[49] F# type providers allow the F# compiler and tools to be extended with components that provide type information to the compiler on-demand at compile time. F# type providers have been used to give strongly typed access to connected information sources in a scalable way, including to the Freebase knowledge graph.[50]

In F# 3.0 the F# quotation and computation expression features are combined to implement LINQ queries.[51] For example:

```
// Use the OData type provider to create types that can be used to access the Northwind database.
open Microsoft.FSharp.Data.TypeProviders

type Northwind = ODataService<"http://services.odata.org/Northwind/Northwind.svc">
let db = Northwind.GetDataContext()

// A query expression.
let query1 = query { for customer in db.Customers do
                     select customer }
```

The combination of type providers, queries and strongly typed functional programming is known as *information rich programming*.[52]

Agent programming

F# supports a variation of the Actor programming model through the in-memory implementation of lightweight asynchronous agents. For example, the following code defines an agent and posts 2 messages:

```
let counter =
    MailboxProcessor.Start(fun inbox ->
        let rec loop n =
            async { do printfn "n = %d, waiting..." n
                    let! msg = inbox.Receive()
                    return! loop(n+msg) }
        loop 0)
```

Development tools

- Visual Studio, with the Visual F# tools from Microsoft installed, can be used to create, run and debug F# projects. The Visual F# tools include a Visual Studio-hosted read–eval–print loop (REPL) interactive console that can execute F# code as it is written. Visual Studio for Mac also fully supports F# projects.
- Visual Studio Code contains full support for F# via the Ionide extension (http://ionide.io/).
- F# can be developed with any text editor. Specific support exists in editors such as Emacs.
- JetBrains Rider is optimized for the development of F# Code starting with release 2019.1.[53]
- LINQPad has supported F# since version 2.x.

Application areas

F# is a general-purpose programming language.

Web programming

The SAFE Stack (https://safe-stack.github.io/) is an end-to-end F# stack to develop web applications. It uses ASP.NET Core on the server side and Fable (https://fable.io) on the client side.[54]

An alternative end-to-end F# option is the WebSharper framework.[55]

Cross-platform app development

F# can be used together with the Visual Studio Tools for Xamarin (https://visualstudio.microsoft.com/xamarin/) to develop apps for iOS and Android. The Fabulous (https://fsprojects.github.io/Fabulous/) library provides a more comfortable functional interface.

Analytical programming

Among others, F# is used for quantitative finance programming,[56] energy trading and portfolio optimization,[57] machine learning,[58] business intelligence[59] and social gaming on Facebook.[60]

In the 2010s, F# has been positioned as an optimized alternative to C#. F#'s scripting ability and inter-language compatibility with all Microsoft products have made it popular among developers.[61]

Scripting

F# can be used as a scripting language, mainly for desktop read–eval–print loop (REPL) scripting.[62]

Open-source community

The F# open-source community includes the F# Software Foundation[11] and the F# Open Source Group at GitHub.[12] Popular open-source F# projects include:

- Fable (https://fable.io/), an F# to Javascript transpiler based on Babel (https://babeljs.io).
- Paket (https://fsprojects.github.io/Paket/), an alternative package manager for .NET that can still use NuGet repositories, but has centralised version-management.
- FAKE (https://fake.build/), an F# friendly build-system.
- Giraffe (https://github.com/giraffe-fsharp/Giraffe), a functionally oriented middleware for ASP.NET Core.
- Suave (https://suave.io/), a lightweight web-server and web-development library.

Compatibility

F# features a legacy "ML compatibility mode" that can directly compile programs written in a large subset of OCaml roughly, with no functors, objects, polymorphic variants, or other additions.

Examples

A few small samples follow:

```fsharp
// This is a comment for a sample hello world program.
printfn "Hello World!"
```

A Person class with a constructor taking a name and age and two immutable properties.

```fsharp
/// This is a documentation comment for a type definition.
type Person(name : string, age : int) =
    member x.Name = name
    member x.Age = age

/// class instantiation
let mrSmith = Person("Smith", 42)
```

A simple example that is often used to demonstrate the syntax of functional languages is the factorial function for non-negative 32-bit integers, here shown in F#:

```fsharp
/// Using pattern matching expression
let rec factorial n =
    match n with
    | 0 -> 1
    | _ -> n * factorial (n - 1)

/// For a single-argument functions there is syntactic sugar (pattern matching function):
let rec factorial = function
    | 0 -> 1
    | n -> n * factorial (n - 1)

/// Using fold and range operator
let factorial n = [1..n] |> Seq.fold (*) 1
```

Iteration examples:

```fsharp
/// Iteration using a 'for' loop
let printList lst =
    for x in lst do
        printfn "%d" x

/// Iteration using a higher-order function
let printList2 lst =
    List.iter (printfn "%d") lst

/// Iteration using a recursive function and pattern matching
let rec printList3 lst =
    match lst with
    | [] -> ()
    | h :: t ->
        printfn "%d" h
        printList3 t
```

Fibonacci examples:

```fsharp
/// Fibonacci Number formula
let fib n =
    let rec g n f0 f1 =
        match n with
        | 0 -> f0
        | 1 -> f1
        | _ -> g (n - 1) f1 (f0 + f1)
    g n 0 1

/// Another approach - a lazy infinite sequence of Fibonacci numbers
let fibSeq = Seq.unfold (fun (a,b) -> Some(a+b, (b, a+b))) (0,1)
```

```
// Print even fibs
[1 .. 10]
|> List.map     fib
|> List.filter  (fun n -> (n % 2) = 0)
|> printList

// Same thing, using a list expression
[ for i in 1..10 do
    let r = fib i
    if r % 2 = 0 then yield r ]
|> printList
```

A sample Windows Forms program:

```
// Open the Windows Forms library
open System.Windows.Forms

// Create a window and set a few properties
let form = new Form(Visible=true, TopMost=true, Text="Welcome to F#")

// Create a label to show some text in the form
let label =
    let x = 3 + (4 * 5)
    new Label(Text = $"{x}")

// Add the label to the form
form.Controls.Add(label)

// Finally, run the form
[<System.STAThread>]
Application.Run(form)
```

Asynchronous parallel programming sample (parallel CPU and I/O tasks):

```
/// A simple prime number detector
let isPrime (n:int) =
   let bound = int (sqrt (float n))
   seq {2 .. bound} |> Seq.forall (fun x -> n % x <> 0)

// We are using async workflows
let primeAsync n =
    async { return (n, isPrime n) }

/// Return primes between m and n using multiple threads
let primes m n =
    seq {m .. n}
        |> Seq.map primeAsync
        |> Async.Parallel
        |> Async.RunSynchronously
        |> Array.filter snd
        |> Array.map fst

// Run a test
primes 1000000 1002000
    |> Array.iter (printfn "%d")
```

See also

- OCaml
- C#
- .NET Framework

Notes

1. https://devblogs.microsoft.com/dotnet/announcing-net-6/.

2. "Nullable Reference Types in F# 5" (https://www.infoq.com/news/2019/04/FSharp-Nulls).
3. "F# Software Foundation's License" (https://github.com/fsharp/fsharp/blob/master/License.txt). *GitHub*. 14 October 2021.
4. "Microsoft's F# License" (https://github.com/Microsoft/visualfsharp/blob/master/License.txt). *GitHub*. 16 October 2021.
5. Syme, Granicz & Cisternino (2007:2)
6. "F# Historical Acknowledgements" (http://research.microsoft.com/en-us/um/cambridge/projects/fsharp/ack.aspx). Retrieved 2012-11-24.
7. Syme, Don (2006). "Leveraging .NET Meta-programming Components from F#" (http://research.microsoft.com/apps/pubs/default.aspx?id=147193). "[F#] is rooted in the Core ML design, and in particular has a core language largely compatible with that of OCaml"
8. for async
9. The F# Software Foundation. "Using F# for Web Applications" (http://fsharp.org/use/web-apps/). Retrieved 2020-07-30.
10. The F# Software Foundation. "Using F# for GPU Programming" (https://web.archive.org/web/20191225110926/http://fsharp.org/use/gpu/). Archived from the original (http://fsharp.org/use/gpu/) on 2019-12-25. Retrieved 2019-12-25.
11. The F# Software Foundation. "The F# Software Foundation" (http://fsharp.org). Retrieved 2012-11-24.
12. The F# Software Foundation. "F# Compiler (open source edition) @ github" (https://fsharp.github.io). Retrieved 2012-11-24.
13. "Develop with Visual F# in Visual Studio" (https://docs.microsoft.com/en-us/visualstudio/ide/fsharp-visual-studio). Retrieved 2020-07-30.
14. "F#" (https://www.jetbrains.com/help/rider/F_Sharp.html). Retrieved 2020-07-30.
15. Syme, Don. "F# 1.0.8 released" (http://blogs.msdn.com/b/dsyme/archive/2005/05/21/420795.aspx). Microsoft. Retrieved September 7, 2014.
16. Syme, Don. "F# 2.0 released as part of Visual Studio 2010" (http://blogs.msdn.com/b/dsyme/archive/2010/04/12/f-2-0-released-as-part-of-visual-studio-2010.aspx). Microsoft. Retrieved September 7, 2014.
17. Zander, Jason. "Visual Studio 2012 and .NET Framework 4.5 released to the web" (http://blogs.msdn.com/b/jasonz/archive/2012/08/15/visual-studio-2012-and-net-framework-4-5-released-to-the-web.aspx). Microsoft. Retrieved September 7, 2014.
18. "Visual Studio 2013 released to web" (http://blogs.msdn.com/b/visualstudio/archive/2013/10/17/visual-studio-2013-released-to-web.aspx). Microsoft. Retrieved September 7, 2014.
19. "Announcing the RTM of Visual F# 4.0" (http://blogs.msdn.com/b/dotnet/archive/2015/07/20/announcing-the-rtm-of-visual-f-4-0.aspx). Microsoft. Retrieved September 15, 2015.
20. "Announcing F# 4.1 and the Visual F# Tools for Visual Studio 2017" (https://blogs.msdn.microsoft.com/dotnet/2017/03/07/announcing-f-4-1-and-the-visual-f-tools-for-visual-studio-2017-2/). Retrieved 2017-03-08.
21. "Announcing F# 4.5" (https://blogs.msdn.microsoft.com/dotnet/2018/08/14/announcing-f-4-5/). Microsoft. 14 August 2018. Retrieved August 14, 2018.
22. "FSharp.Core 4.5.0" (https://www.nuget.org/packages/FSharp.Core/4.5.0).
23. "Download .NET Core 2.1 (Linux, macOS, and Windows)" (https://dotnet.microsoft.com/download/dotnet/2.1). Microsoft. Retrieved May 13, 2021.
24. "Announcing F# 4.6" (https://devblogs.microsoft.com/dotnet/announcing-f-4-6/). Microsoft. 29 March 2019. Retrieved March 29, 2019.
25. "FSharp.Core 4.6.0" (https://www.nuget.org/packages/FSharp.Core/4.6.0).
26. "Download .NET Core 2.2 (Linux, macOS, and Windows)" (https://dotnet.microsoft.com/download/dotnet/2.2). Microsoft. Retrieved May 13, 2021.

27. "Announcing F# 4.7" (https://devblogs.microsoft.com/dotnet/announcing-f-4-7/). Microsoft. 23 September 2019. Retrieved September 23, 2019.
28. "FSharp.Core 4.7.0" (https://www.nuget.org/packages/FSharp.Core/4.7.0).
29. "Download .NET Core 3.0 (Linux, macOS, and Windows)" (https://dotnet.microsoft.com/download/dotnet/3.0). Microsoft. Retrieved May 13, 2021.
30. "Announcing F# 5" (https://devblogs.microsoft.com/dotnet/announcing-f-5/). November 10, 2020.
31. "Download .NET 5.0 (Linux, macOS, and Windows)" (https://dotnet.microsoft.com/download/dotnet/5.0). Microsoft. Retrieved May 13, 2021.
32. "F# 6 is officially here!" (https://devblogs.microsoft.com/dotnet/fsharp-6-is-officially-here/). November 9, 2021.
33. "Download .NET 6.0 (Linux, macOS, and Windows)" (https://dotnet.microsoft.com/download/dotnet/6.0). Microsoft. Retrieved November 14, 2021.
34. Edwards, Kathryn (23 December 2008). "The A-Z of programming languages: F#" (http://www.networkworld.com/article/2271225/software/the-a-z-of-programming-languages--f-.html). *networkworld.com*. IDG. Retrieved 8 August 2016.
35. McNamara, Brian. "More About F# 3.0 Language Features" (http://blogs.msdn.com/b/fsharpteam/archive/2012/07/19/more-about-fsharp-3.0-language-features.aspx). Microsoft. Retrieved September 7, 2014.
36. McNamara, Brian. "Announcing a pre-release of F# 3.1" (http://blogs.msdn.com/b/fsharpteam/archive/2013/06/27/announcing-a-pre-release-of-f-3-1-and-the-visual-f-tools-in-visual-studio-2013.aspx). Microsoft. Retrieved September 7, 2014.
37. "Announcing the RTM of Visual F# 4.0" (https://blogs.msdn.microsoft.com/dotnet/2015/07/20/announcing-the-rtm-of-visual-f-4-0/). Retrieved 2017-03-08.
38. "Announcing F# 4.1 and the Visual F# Tools for Visual Studio 2017" (https://blogs.msdn.microsoft.com/dotnet/2017/03/07/announcing-f-4-1-and-the-visual-f-tools-for-visual-studio-2017-2/). Retrieved 2017-03-08.
39. "Announcing F# 4.7" (https://devblogs.microsoft.com/dotnet/announcing-f-4-7/). 23 September 2019.
40. "Announcing F# 5" (https://devblogs.microsoft.com/dotnet/announcing-f-5/). 10 November 2020.
41. https://docs.microsoft.com/en-us/dotnet/fsharp/whats-new/fsharp-6
42. "F# Language Overview" (http://tomasp.net/articles/fsharp-i-introduction/article.pdf) (PDF). Retrieved 2007-12-14.
43. "Some Details on F# Computation Expressions" (http://blogs.msdn.com/dsyme/archive/2007/09/22/some-details-on-f-computation-expressions-aka-monadic-or-workflow-syntax.aspx). Retrieved 2007-12-14.
44. "Pattern Matching in F# Part 2 : Active Patterns" (http://www.developerfusion.com/article/133772/pattern-matching-in-f-part-2-active-patterns/). Retrieved 2012-11-24.
45. "Introducing F# Asynchronous Workflows" (http://blogs.msdn.com/dsyme/archive/2007/10/11/introducing-f-asynchronous-workflows.aspx). Retrieved 2007-12-14.
46. "Units of Measure (F#)" (http://msdn.microsoft.com/en-us/library/dd233243.aspx). Retrieved 2012-11-24.
47. "Units of Measure in F#: Part One, Introducing Units" (http://blogs.msdn.com/b/andrewkennedy/archive/2008/08/29/units-of-measure-in-f-part-one-introducing-units.aspx). Retrieved 2012-11-24.
48. "Code Quotations (F#)" (http://msdn.microsoft.com/en-us/library/dd233212.aspx). Retrieved 2012-11-24.
49. "Type Providers" (http://msdn.microsoft.com/en-us/library/hh156509.aspx). Retrieved 2012-11-24.

50. "New Tech Report from Microsoft Research: Strongly-Typed Language Support for Internet-Scale Information Sources" (http://blogs.msdn.com/b/dsyme/archive/2012/09/21/new-tech-report-from-microsoft-research-strongly-typed-language-support-for-internet-scale-information-sources.aspx). Retrieved 2012-11-24.
51. "Query Expressions (F#)" (http://msdn.microsoft.com/en-us/library/vstudio/hh225374.aspx). Retrieved 2012-11-24.
52. "F# 3.0 – LINQ + Type Providers= Information Rich Programming" (http://www.infoq.com/news/2011/09/Fsharp-3.0). Retrieved 2012-11-24.
53. Alexander Kurakin. "Rider 2019.1 Kicks off its Early Access Program!" (https://blog.jetbrains.com/dotnet/2019/03/15/rider-2019-1-kicks-off-early-access-program/).
54. "Fable: JavaScript you can be proud of!" (http://fable.io/). *fable.io*. Retrieved 2017-12-09.
55. Intellifactory. "WebSharper home" (http://websharper.com). Retrieved 2012-11-24.
56. "Microsoft Case Studies:Microsoft Visual Studio 2012 - Financial Services Firm" (http://www.microsoft.com/casestudies/Case_Study_Detail.aspx?casestudyid=4000006794). *Microsoft*. Retrieved 2012-11-25.
57. "F# for Energy Trading and Portfolio Optimization" (http://blogs.msdn.com/b/dsyme/archive/2011/01/12/f-for-energy-trading-and-energy-portfolio-optimization.aspx). Retrieved 2012-11-25.
58. "Microsoft Case Study: Grange Insurance" (http://www.microsoft.com/casestudies/Case_Study_Detail.aspx?CaseStudyID=4000005226). *Microsoft*. Retrieved 2012-11-25.
59. Trelford, Phil (2007). "Learning with F#" (http://dl.acm.org/citation.cfm?id=1362702.1362709&coll=ACM&dl=ACM&type=series&idx=SERIES824%25E2%2588%2582=series&WantType=Proceedings&title=ICFP). *Proceedings of the 4th ACM SIGPLAN workshop on Commercial users of functional programming - CUFP '07*. Cufp '07. pp. 1–2. doi:10.1145/1362702.1362709 (https://doi.org/10.1145%2F1362702.1362709). ISBN 9781450378444. S2CID 24018363 (https://api.semanticscholar.org/CorpusID:24018363). Retrieved 2012-11-25.
60. "F# Job in Facebook Social Gaming" (http://blogs.msdn.com/b/dsyme/archive/2012/10/23/f-job-in-social-gaming-inn-london.aspx). Retrieved 2012-11-25.
61. "F# Developer Testimonials" (https://fsharp.org/testimonials/). Retrieved May 25, 2021.
62. "Scripting in F#" (https://docs.microsoft.com/en-gb/archive/blogs/chrsmith/scripting-in-f). Retrieved 2020-01-17.

References

- Syme, Don; Granicz, Adam; Cisternino, Antonio (2007), *Expert F#*, Apress
- Harrop, Jon (2010), *Visual F# 2010 for Technical Computing*, Flying Frog Consultancy
- Pickering, Robert (2007), *Foundations of F#*, Apress
- Smith, Chris (2009), *Programming F#*, O'Reilly
- Petricek, Tomas (2009), *Real World Functional Programming With Examples in F# and C#*, Manning Publications
- Hansen, Michael; Rischel, Hans (2013), *Functional Programming Using F#*, Cambridge University Press
- Astborg, Johan (2013), *F# for Quantitative Finance* (https://www.packtpub.com/big-data-and-business-intelligence/f-quantitative-finance), Packt Publishing
- Lundin, Mikael (2015), *Testing with F#* (https://www.packtpub.com/application-development/testing-f), Packt Publishing

External links

- Official website (http://fsharp.org) The F# Software Foundation
- The F# Open Source Group at GitHub (https://fsharp.github.io)

- The Visual F# Developer Center (http://www.fsharp.net)
- Tsunami, an embeddable desktop F# IDE (https://web.archive.org/web/20140703090417/http://tsunami.io/)
- Cloud Tsunami, an embeddable Silverlight F# IDE (https://web.archive.org/web/20130921130415/http://tsunami.io/cloud_tsunami.html)
- Try F#, for learning F# in a web browser (https://web.archive.org/web/20121130163443/http://www.tryfsharp.org/)
- F# Snippets Site (http://fssnip.net)
- The Visual F# team blog (http://blogs.msdn.com/fsharpteam)
- The original Microsoft Research website for F# (http://research.microsoft.com/fsharp)
- Planet F# (http://feedproxy.google.com/planet_fsharp)
- The F# Survival Guide, Dec 2009 (Web-based book) (https://web.archive.org/web/20110715231625/http://www.ctocorner.com/fsharp/book/default.aspx)
- The F# Language Specification (http://fsharp.org/specs/language-spec/)
- An introduction to F# programming (http://www.developerfusion.com/article/122079/intro-to-f/)
- A tutorial showing the *process* of reaching a functional design; includes test and parallel coding (http://opcoast.com/demos/fsharp/index.html)

Retrieved from "https://en.wikipedia.org/w/index.php?title=F_Sharp_(programming_language)&oldid=1058879226"