

F#



MICROSOFT RESOLVE

¿Qué es F#?

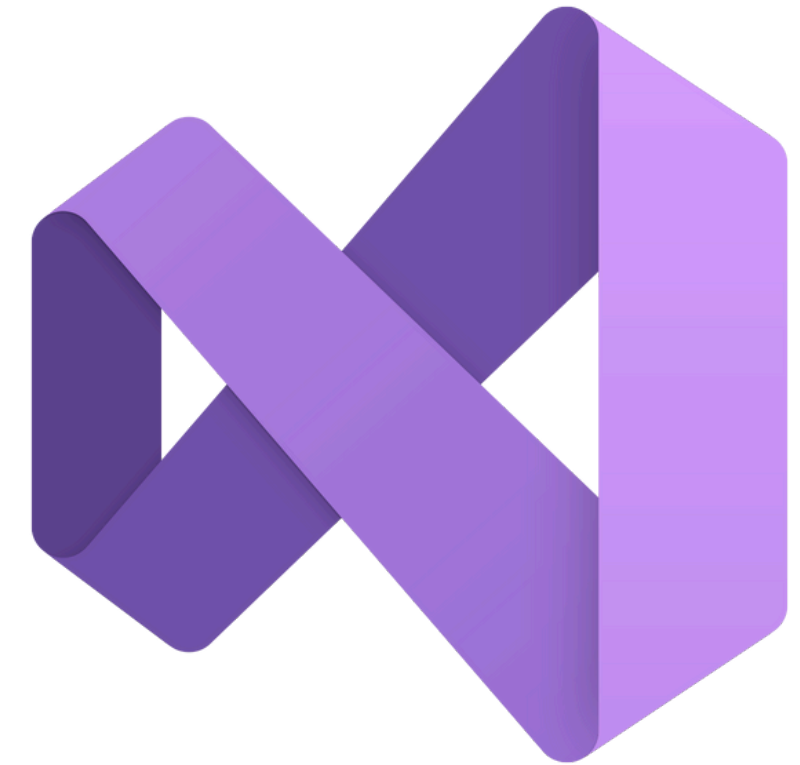
F# es un lenguaje de programación multiparadigma que funciona sobre la plataforma .NET. Aunque combina enfoques imperativos y orientados a objetos, su base principal es la programación funcional. Esto lo diferencia de otros lenguajes del mismo entorno, como C#, que han añadido elementos funcionales pero siguen priorizando la orientación a objetos.

Un poco de su historia

F# fue creado en 2005 por Don Syme en Microsoft Research, Cambridge, como una adaptación de OCaml para el .NET Framework. Pertenece a la familia ML y toma influencias de lenguajes como C#, Python, Haskell, Scala y Erlang. Desde el inicio incorporó rasgos de la programación funcional moderna, como la inferencia de tipos y el uso de funciones como valores.

Momento Importante !

Un momento crucial en la historia de F# fue su inclusión oficial en Visual Studio 2010. Este paso lo posicionó al mismo nivel que otros lenguajes de la plataforma, otorgándole soporte de primera clase en el IDE de Microsoft y haciéndolo ampliamente accesible. Posteriormente, el desarrollo y la evolución de F# se transformaron hacia un modelo de código abierto.



Microsoft

Paradigma y enfoque

El núcleo de F# se centra en la programación funcional, con funciones como valores, datos inmutables y sin efectos secundarios. Esto permite pasar o devolver funciones entre sí, facilitando la composición de la lógica y generando un código más conciso, potente y fácil de probar y paralelizar.

CARACTERISTICAS

01

Concision

Se escribe menos código para hacer lo mismo

02

Funcional por dft

Se centra en funciones y expresiones en lugar de objetos

03

Inmutabilidad

Fomenta que los valores no cambien, lo que evita errores comunes

04

Seguro

El compilador ayuda a detectar errores antes de ejecutar el programa

05

Interoperatibilidad

Puede usar librerías de C# y del ecosistema .NET

06

Multiparadigma

Combina programación funcional, orientada a objetos e imperativa

Características Principales



- Lenguaje fuertemente tipado
- Favorece la inmutabilidad y el uso de funciones puras.
- Usa inferencia de tipos, lo que permite una codificación más segura.
- Uso de expresiones Lambda.
- Es un lenguaje multiparadigma, aunque es funcional por defecto, también admite programación orientada a objetos e imperativa.

Características Principales



Lenguaje Fuertemente Tipado

// Ejemplo de tipado fuerte en F#

```
let x = 10           // x es un entero (int)
let y = 3.5          // y es un número de punto flotante (float)
```

// Intento de suma directa -> ERROR de compilación

```
let suma = x + y     // Esto no compila
```

// Solución correcta: conversión explícita

```
let sumaOk = float x + y // Ahora sí funciona
```

Características Principales



Funciones Puras

- Siempre devuelve el mismo resultado si recibe los mismos argumentos (no depende de factores externos).
- No tiene efectos secundarios (no modifica variables externas, no imprime en consola, no cambia el estado del programa).
- Las funciones puras devuelven resultados consistentes para entradas idénticas. No modifican estados externos ni dependen de datos mutables.

Características Principales



Funciones Puras

```
// Función pura: suma de dos números  
let sumar a b = a + b
```

```
// Función pura: calcula el cuadrado  
let cuadrado x = x * x
```

```
// Función pura: devuelve el mayor de dos números  
let maximo a b = if a > b then a else b
```

```
let r1 = sumar 3 5           // 8  
let r2 = cuadrado 4          // 16  
let r3 = maximo 10 7         // 10
```

Características Principales



Inmutabilidad por Defecto

En F#, los valores no pueden ser modificados una vez que han sido creados. La palabra clave `let` se utiliza para enlazar un valor a un nombre, y por defecto, ese valor es inmutable.

```
// Definimos un valor
```

```
let x = 10
```

```
// Intento de cambiarlo -> ERROR de compilación
```

```
x <- 20 //
```

Características Principales



Inmutabilidad por Defecto

Si realmente quieres mutabilidad, debes declararlo explícitamente con la palabra clave mutable

```
// Declaramos un valor mutable  
let mutable y = 10
```

```
// Ahora sí podemos cambiarlo  
y <- 20 // ☒ Esto funciona
```

```
printfn "Nuevo valor de y: %d" y
```

Características Principales



Usa inferencia de tipos

No necesitas declarar el tipo de cada variable o función, porque el compilador lo deduce automáticamente, pero aún así el código sigue siendo fuertemente tipado.

```
let x = 42          // El compilador infiere que x : int
let y = 3.14        // El compilador infiere que y : float

let suma a b = a + b
// Aquí F# infiere que 'a' y 'b' son int
// porque el operador (+) por defecto se aplica a enteros
```

Características Principales



Uso de expresiones Lambda

En F#, las expresiones lambda son funciones anónimas, es decir, funciones sin nombre, que puedes crear "al vuelo" para usarlas en una expresión.

```
fun argumentos -> expresión  
  
// Una función lambda que suma 1  
let incrementar = fun x -> x + 1  
printfn "%d" (incrementar 5)    // 6  
  
// Una lambda con dos parámetros  
let sumar = fun a b -> a + b  
printfn "%d" (sumar 3 7)      // 10
```

Características Principales



Multiparadigma: Funcional

```
≡ // --- Funcional ---  
[ // Lista inmutable y funciones puras  
  let numeros = [1; 2; 3; 4; 5]  
  
  let cuadrados = numeros |> List.map (fun x -> x * x)  
  printfn "Cuadrados de la lista original: %A" cuadrados  
  // [1; 4; 9; 16; 25]
```

Características Principales



Multiparadigma: Imperativo

```
- // --- Imperativo ---  
  // Variable mutable y bucle for  
  let mutable suma = 0  
- for n in cuadrados do  
  | suma <- suma + n  
  
printfn "La suma de los cuadrados es: %d" suma  
// 55
```


Características Principales

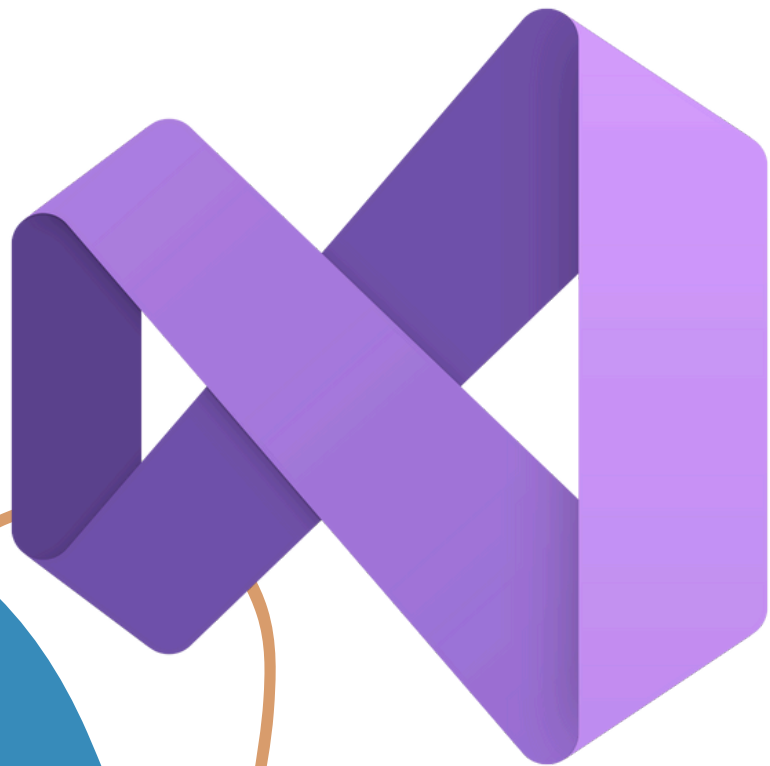


Multiparadigma: Orientado a Objetos

```
// --- Orientado a objetos ---  
// Definición de una clase con un método  
type Calculadora() =  
| member _.Multiplicar a b = a * b  
  
let calc = Calculadora()  
let producto = calc.Multiplicar 6 7  
  
printfn "El producto de 6 * 7 con la Calculadora es: %d" producto  
// 42
```

Entornos de Desarrollo que lo Soportan

F# se beneficia del soporte completo de la plataforma .NET, lo que le permite ser un lenguaje multiplataforma compatible con Windows, Linux y macOS.



FileEditSelectionViewGoRun

chess

EXTENSIONS: MARK...

ionide

ionide for F#
F# Language Support, pow...
1.6M
Install

Ionide-Paket
Paket (alternative NuGet cl...
73K
Install

Ionide-FAKE
FAKE (F# Make) Support
59K
Install

Ionide + Solution ...
Replace the default ionide ...
3K
Spencer Farley
Install


mechanic
F# File reordering tool
9K
Ionide
Install

Discover Panel
Extension creating alternat...
1K
Ionide
Install

Distributed Packa...
Distributed Package Mana...
231
Ionide
Install

universal-type-lens
Turns inlay hints of extensi...
7
nipah
Install

Extension: Ionide for F#



Ionide for F#

Ionide

1,647,343

★★★★★ (114)

F# Language Support, powered by FsAutoComplete

Install

☒ Auto Update

DETAILS

FEATURES

CHANGELOG

EXTENSION PACK

Ionide-VSCode: FSharp

Enhanced F# Language Features for Visual Studio Code

Part of the [ionide plugin suite](#). Read detailed documentation at [ionide docs page](#).

Visual Studio Marketplace

v7.27.1

downloads

10.95M

rating

★★★★¾

backers

246

sponsors


27

Contribute with

Gitpod

You can support Ionide development on [Open Collective](#).

DONATE TO OUR COLLECTIVE




Description

Ionide-VSCode is a VSCode plugin that turns VSCode into a fully-fledged IDE for F# development.

The LSP that powers language features is [FSAutoComplete](#).

The library that powers project and script loading is [proj-info](#)

You find a version of this plugin pre-packaged with the FOSS debugger from Samsung [here](#)



Marketplace

Identifier	ionide.ionide-fsharp
Version	7.27.1
Published	2015-11-11, 17:58:29
Last Released	2025-09-12, 11:33:17

Categories

Programming Languages

Snippets

Linters

Resources

Marketplace

Issues

Repository

License

Ionide





00

Live Share



Create a new project

Recent project templates

 Console App	F#
 Windows Forms App (.NET Framework)	C#
 Empty Project	C++
 Console App	C++



Search for templates (Alt+S)



[Clear all](#)

F#

All platforms

All project types



Console App

A project for creating a command-line application that can run on .NET on Windows, Linux and macOS

F#

Linux

macOS

Windows

Console



Class Library

A project for creating a class library that targets .NET or .NET Standard

F#

Android

Linux

macOS

Windows

Library



ASP.NET Core Empty

An empty project template for creating an ASP.NET Core application. This template does not have any content in it.

F#

Linux

macOS

Windows

Cloud

Service

Web



ASP.NET Core Web App (Model-View-Controller)

A project template for creating an ASP.NET Core application with example ASP.NET Core MVC Views and Controllers. This template can also be used for RESTful HTTP services.

F#

Linux

macOS

Windows

Cloud

Service

Web



ASP.NET Core Web API

A project template for creating an ASP.NET Core application with an example Controller for a RESTful HTTP service. This template can also be used for ASP.NET Core MVC Views and Controllers.

F#

Linux

macOS

Windows

Cloud

Service

Web

WebAPI



Worker Service

An empty project template for creating a worker service.

F#

Linux

macOS

Windows

Cloud

Service



MSTest Test Project

A project that contains MSTest unit tests that can run on .NET on Windows, Linux and MacOS.

F#

Linux

macOS

Windows

Test



xUnit Test Project

A project that contains xUnit.net tests that can run on .NET on Windows, Linux and MacOS.

F#

Linux

macOS

Windows

Test



NUnit Test Project

A project that contains NUnit tests that can run on .NET on Windows, Linux and MacOS.

F#

Linux

macOS

Windows

Desktop

Test

Web



Blank App (Android)

A project for creating a Xamarin.Android application.

F#

Android

Mobile

SINTAXIS Creación de Variables

```
//Declaración de variables  CONSTANTES
```

```
let edad = 25
```

```
let nombre = "Juan"
```

```
let pi = 3.14159
```

```
printfn "Edad: %d, Nombre: %s, Pi: %f" edad nombre pi
```

Microsoft Visual Studio

+

▼

```
Edad: 25, Nombre: Juan, Pi: 3.141590
```

SINTAXIS Creación de Variables

```
//Declaración de variables  MUTABLES  
let mutable numero = 20  
printfn "Numero: %d" numero  
numero <- numero * 2  
printfn "Numero: %d" numero
```

```
Numero: 20  
Numero: 40
```

SINTAXIS

Creación de Variables

```
//Tipado Explícito
```

```
let edad: int = 30
```

```
let nombre: string = "Ana"
```

```
let pi: float = 3.14159
```

```
printfn "Edad: %d, Nombre: %s, Pi: %f" edad nombre pi
```

```
Edad: 30, Nombre: Ana, Pi: 3.141590
```

SINTAXIS

Comentarios

```
//para comentar una linea se utilizan el doble slash  
(* y para comentar en bloque  
nueva linea de comentario  
nueva linea de comentario  
*)
```


SINTAXIS

Lectura

```
open System
```

```
printf "Introduce tu nombre: "
```

```
let nombre = Console.ReadLine()
```

```
printf "Hola %s! " nombre
```

```
Introduce tu nombre: sofia  
Hola sofia!
```

SINTAXIS

Lectura

```
open System
```

```
printf "Introduce un numero: "
```

```
let numero = Console.ReadLine() |> int
```

```
printf "El doble de tu numero es: %d" (numero*2)
```

```
Introduce un numero: 6
```

```
El doble de tu numero es: 12
```

SINTAXIS

Creacion de Clases

```
//Definicion de la clase persona
type Persona (nombre: string, edad: int)=
    //atributos
    member this.Nombre = nombre
    member this.Edad = edad

    //Metodos
    member this.Saludar() =
        | printf "Hola! Mi nombre es %s y tengo %d años" this.Nombre this.Edad

//Creacion de una instancia
let personal = Persona("Sofia", 19)
personal.Saludar()
```

Hola! Mi nombre es Sofia y tengo 19 años

SINTAXIS

Operadores

```
//OPERADORES DE COMPARACION
let esIgual = (5 = 5)           // True
let esDiferente = (5 <> 3)      // True
let menorQue = (3 < 5)          // True
let mayorQue = (5 > 3)          // True
let menorIgual = (3 <= 3)       // True
let mayorIgual = (5 >= 3)       // True
```

```
//OPERADORES ARITMETICOS
let suma = 5 + 3                // 8
let resta = 5 - 3               // 2
let multiplicacion = 5 * 3      // 15
let division = 10 / 2           // 5
let residuo = 10 % 3            // 1
let potencia = 2.0 ** 3.0       // 8.0
```

SINTAXIS

Operadores

```
// OPERADOR && (AND Logico)
```

```
let a = true
```

```
let b = false
```

```
let resultadoAnd = a && b // Será false porque ambos valores deben ser true
```

```
printfn "Resultado de AND lógico: %b" resultadoAnd
```

```
// OPERADOR || (OR Logico)
```

```
let x = true
```

```
let y = false
```

```
let resultadoOr = x || y // Será true porque al menos uno de los valores es true
```

```
printfn "Resultado de OR lógico: %b" resultadoOr
```

SINTAXIS

Operadores

```
// 1. Operador |> (Pipe Forward)
```

```
let duplicar x = x * 2
```

```
let sumarDiez x = x + 10
```

```
let resultado = 5 |> duplicar |> sumarDiez
```

```
printfn "%d" resultado // Output: 20
```

SINTAXIS

Operadores

```
// 2. Operador <| (Pipe Backward)
```

```
let suma x y = x + y
```

```
let resultado = suma 2 <| (3 * 4)
```

```
// Es igual a suma(2, (3 * 4))
```

```
printfn "%d" resultado    // Output: 14
```

SINTAXIS

Operadores

```
// 3. Operador >> (Composición de Funciones – Forward)
```

```
let duplicar x = x * 2
```

```
let sumarDiez x = x + 10
```

```
let nuevaFuncion = duplicar >> sumarDiez
```

```
printfn "%d" (nuevaFuncion 5) // Output: 20
```


SINTAXIS

Operadores

// 4. Operador << (Composición de Funciones – Backward)

```
let duplicar x = x * 2  
let sumarDiez x = x + 10
```

```
let nuevaFuncion = sumarDiez << duplicar  
printfn "%d" (nuevaFuncion 5)    // Output: 20
```

SINTAXIS

```
// Tipos de datos: numéricos

// int Entero de 32 bits
let x: int = 42
// int64 Entero de 64 bits (long)
let y: int64 = 9223372036854775807L
// uint8 Entero sin signo de 8 bits
let z: uint8 = 255uy
// float Número decimal de 64 bits
let a: float = 3.14
// decimal Número decimal de alta precisión
let xd: decimal = 3.141592653589793M
// bigint Entero de precisión arbitraria
let xdd: bigint = 9999999999999999999999999999999I
```

Tipo de Datos Soportados

```
// Tipos de datos : booleanos

// bool Valores true o false
let x: bool = true
let y: bool = false
```

```
// Tipos de datos : texto
// char Un solo carácter
let c: char = 'A'
// Cadena de texto
let s: string = "Hola, F#"
```

SINTAXIS

Tipo de Datos Soportados

```
// Tipos de datos : opción y resultado
// option<'T> Puede ser Some(valor) o None
let x: int option = Some 10
// Result<'T, 'E> Resultado de éxito o error
let r: Result<int, string> = Ok 42

// Tipos de datos : referencia (nullable)
// F# puede trabajar con tipos de referencia de .NET:
let texto: string option = Some "Hola"
let nulo: string option = None
```

SINTAXIS

```
// If simple
let edad = 18

if edad >= 18 then
    printfn "Eres mayor de edad"
else
    printfn "Eres menor de edad"

// Salida: Eres mayor de edad
```

Condicional IF

```
// IF ANIDADO (Condiciones Múltiples)
let x = 20

let result =
    if x < 10 then
        "Menor que 10"
    elif x = 20 then
        "Igual a 20"
    else
        "Mayor que 10 y diferente de 20"

printfn "%s" result // Salida: Igual a 20
```

SINTAXIS

Switch Case

```
// Match (Switch Case en otros lenguajes)
let clasificarNumero x =
    match x with
    | 0 -> "Es cero"
    | 1 -> "Es uno"
    | _ -> "Es otro número"

printfn "%s" (clasificarNumero 5) // Salida: Es otro número
```

SINTAXIS

Ciclo For

```
// For Simple  
for i in 1 .. 5 do  
    printfn "El valor de i es: %d" i
```

Microsoft Visual Studio

```
El valor de i es: 1  
El valor de i es: 2  
El valor de i es: 3  
El valor de i es: 4  
El valor de i es: 5
```

SINTAXIS

Ciclo For

```
// For Simple  
for i in 5 .. -1 .. 1 do  
    printfn "El valor de i es: %d" i
```

Microsoft Visual Studio

```
El valor de i es: 5  
El valor de i es: 4  
El valor de i es: 3  
El valor de i es: 2  
El valor de i es: 1
```

SINTAXIS

Ciclo Foreach

```
let lista = [1;3;6;9;12]

//foreach
for elemento in lista do
    printfn "Elemento: %d" elemento
```

Microsoft Visual Studio

```
Elemento: 1
Elemento: 3
Elemento: 6
Elemento: 9
Elemento: 12
```


SINTAXIS

Ciclo While

```
//ciclo While
let mutable contador = 0

while contador < 15 do
[
    printfn "Contador: %d" contador
    contador <- contador + 3
]
```

Microsoft Visual Studio

```
Contador: 0
Contador: 3
Contador: 6
Contador: 9
Contador: 12
```

SINTAXIS

Objetos

```
//Las listas en F# son inmutables por defecto y representan una colección ordenada de elementos del mismo tipo.
```

```
(*Características:
```

```
Inmutables: No puedes modificar una lista después de crearla.
```

```
Basadas en listas enlazadas.
```

```
Se utilizan comúnmente en programación funcional.
```

```
*)
```

```
let milista = [1; 2; 3; 4; 5] // Crear una lista
```

```
let otralista = 0 :: milista // Agregar un elemento al inicio
```

```
// Operaciones
```

```
let suma = List.sum milista // Suma de elementos
```

```
let nuevalista = List.map (fun x -> x * 2) milista // Multiplica cada elemento por 2
```

```
printfn "Lista original: %A" milista
```

```
printfn "Lista modificada: %A" nuevalista
```

Microsoft Visual Studio

```
Lista original: [1; 2; 3; 4; 5]  
Lista modificada: [2; 4; 6; 8; 10]
```

SINTAXIS

Objetos

```
(*  
Arrays (Array)  
Los arreglos en F# son mutables y permiten acceso rápido a elementos mediante índices.  
  
Características:  
Indexados (0-based).  
Mutables: Puedes cambiar el valor de un elemento en un índice específico.  
Adecuados para operaciones de alto rendimiento.  
*)
```

```
let miArray = [|1; 2; 3; 4; 5|] // Crear un array  
miArray.[0] <- 10 // Modificar el primer elemento  
  
// Operaciones  
let suma = Array.sum miArray // Suma de elementos  
let nuevoArray = Array.map (fun x -> x * 2) miArray // Multiplica cada elemento por 2  
  
printfn "Array original: %A" miArray  
printfn "Array modificado: %A" nuevoArray
```

Microsoft Visual Studio

```
Array original: [|10; 2; 3; 4; 5|]  
Array modificado: [|20; 4; 6; 8; 10|]
```

SINTAXIS

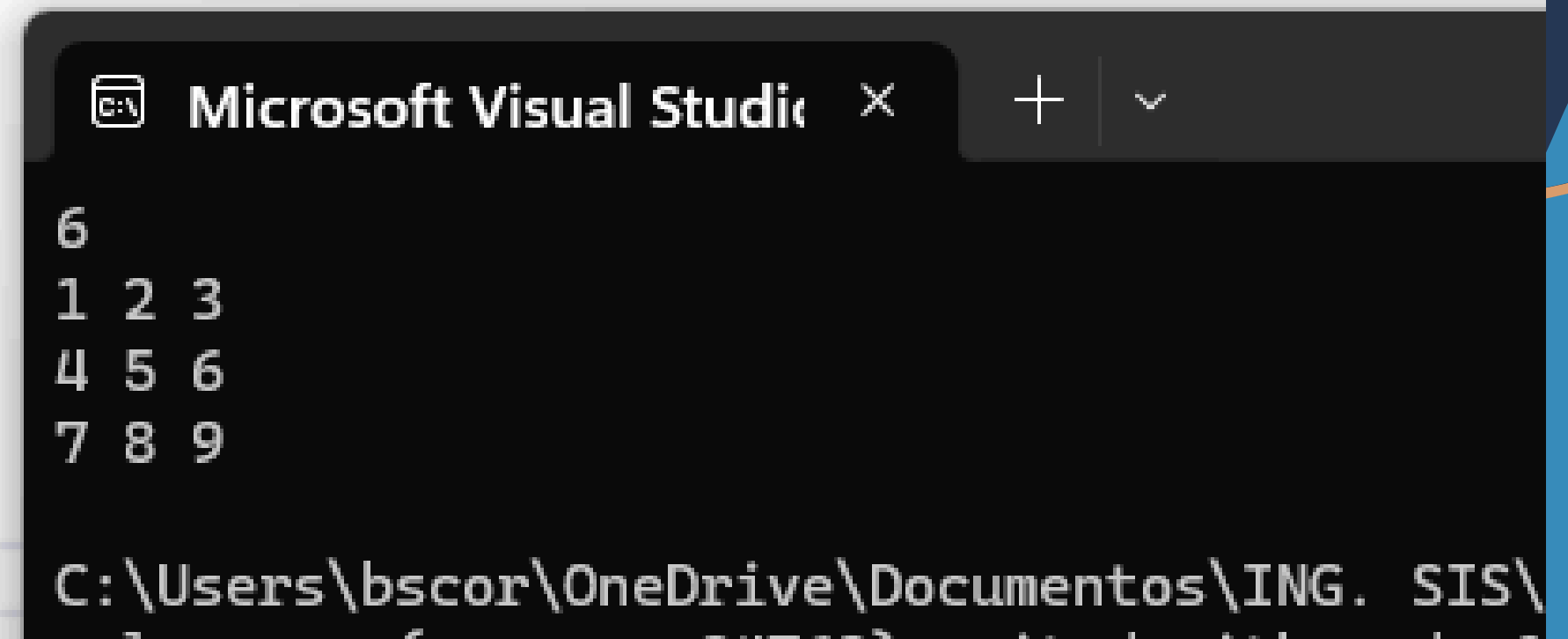
Objetos

```
// Definir una matriz Bidimensional de tipo int
let matriz2D = array2D [[ 1; 2; 3]; [4; 5; 6]; [7; 8; 9]]

// Acceder a un elemento de la matriz (por ejemplo, el elemento en la fila 1, columna 2)
let valor = matriz2D.[1, 2] // Esto devuelve el valor 6
printfn "%d" valor
```

```
let mostrarTablero2 (tablero: int[,]) =
    for i in 0..2 do
        for j in 0..2 do
            printf "%d " tablero.[i,j]
        printfn " "
```

```
mostrarTablero2 (matriz2D)
```



```
Microsoft Visual Studio x + v
6
1 2 3
4 5 6
7 8 9
C:\Users\b scor\OneDrive\Documentos\ING. SIS\
```

SINTAXIS

Objetos

```
(*  
Los mapas son colecciones inmutables de pares clave-valor, útiles para buscar valores por claves.
```

```
Características:
```

```
Inmutables.
```

```
Ordenados automáticamente por clave.
```

```
Similar a un diccionario.
```

```
*)
```

```
let miMapa = Map.ofList [("a", 1); ("b", 2); ("c", 3)] // Crear un mapa
```

```
// Operaciones
```

```
let valor = miMapa["b"] // Obtener valor por clave
```

```
let nuevoMapa = miMapa.Add("d", 4) // Agregar un nuevo par clave-valor
```

```
printfn "Mapa original: %A" miMapa
```

```
printfn "Mapa modificado: %A" nuevoMapa
```

Microsoft Visual Studio × + ▾

```
Mapa original: map [("a", 1); ("b", 2); ("c", 3)]
```

```
Mapa modificado: map [("a", 1); ("b", 2); ("c", 3); ("d", 4)]
```

SINTAXIS

Objetos

```
(*  
A diferencia de Map, los diccionarios son mutables  
y proporcionan un acceso más eficiente para buscar valores.  
*)
```

```
open System.Collections.Generic
```

```
let diccionario = Dictionary<string, int>()  
diccionario.Add("a", 1)  
diccionario.Add("b", 2)
```

```
// Operaciones
```

```
diccionario["a"] <- 10 // Modificar un valor existente  
printfn "Diccionario: %A" diccionario
```

Microsoft Visual Studio

```
Diccionario: seq [[a, 10]; [b, 2]]
```

```
C:\Users\bscor\OneDrive\Documentos\ING. SIS\7mo :  
mplos.exe (process 28236) exited with code 0.  
To automatically close the console when debugging  
le when debugging stops.  
Press any key to close this window . . .|
```

F#



DEMOS

Usos y Campos de Acción del Lenguaje

F# no es un lenguaje de uso general, sino una herramienta estratégica para casos de uso específicos.

- **Ciencia de Datos y Machine Learning:** Su tipado fuerte, la inmutabilidad y los proveedores de tipos lo hacen ideal para el análisis de datos. Librerías como FsLab, FSharp.Stats y Plotly.NET facilitan estas tareas.
- **Desarrollo Web y Full-Stack:** El enfoque funcional sin estado se adapta de forma natural al modelo de la web. Proyectos como SAFE Stack y Fable permiten construir aplicaciones web y de escritorio con F# tanto en el lado del servidor como en el del cliente.


Usos y Campos de Acción del Lenguaje

- **Sistemas Empresariales y Finanzas:** En estas industrias, la corrección y la fiabilidad del código son de suma importancia. F# permite modelar la lógica de negocio compleja de manera precisa, minimizando los errores y reduciendo el costo de mantenimiento.
- **Interoperabilidad con C# y el Ecosistema .NET:** Una de las mayores ventajas de F# es su perfecta interoperabilidad con C#. Un equipo puede utilizar F# para los componentes de lógica de negocio o de procesamiento de datos intensivo, y C# para las interfaces de usuario o la integración con bibliotecas específicas.

Estadísticas

F# es un lenguaje de programación que, aunque actualmente no es de los más utilizados, posee un gran potencial de crecimiento. A medida que evoluciona el desarrollo de software, es posible que F# gane mayor relevancia y supere las expectativas en la próxima década, convirtiéndose en una opción atractiva para distintos ámbitos de la programación.



[About us](#) [Knowledge](#) [News](#) [Coding Standards](#) [TIOBE Index](#) [Contact](#) 

[Products](#) ▾

[Quality Models](#) ▾

[Markets](#) ▾

[Schedule a demo](#)

The Next 50 Programming Languages

The following list of languages denotes #51 to #100. Since the differences are relatively small, the programming languages are only listed (in alphabetical order).

- ActionScript, Algol, Alice, Apex, Awk, B4X, CLIPS, Clojure, D, Eiffel, Elm, F#, Forth, Groovy, Hack, Icon, Inform, Io, J, JScript, Logo, Modula-2, Mojo, MQL5, NATURAL, Nim, Oberon, OCaml, Occam, OpenCL, OpenEdge ABL, PL/I, Q, Racket, REXX, Ring, RPG, Scheme, Simulink, Smalltalk, SPARK, Stata, Structured Text, SystemVerilog, Tcl, Vala/Genie, VHDL, Wolfram, X++, Xojo

CONCLUSIONES

- F# potencia la programación funcional, haciendo el código más seguro y conciso.
- Se integra con .NET, facilitando su uso en proyectos existentes.
- Ideal para análisis de datos, machine learning y sistemas complejos.
- Mejora la productividad y la detección temprana de errores.

