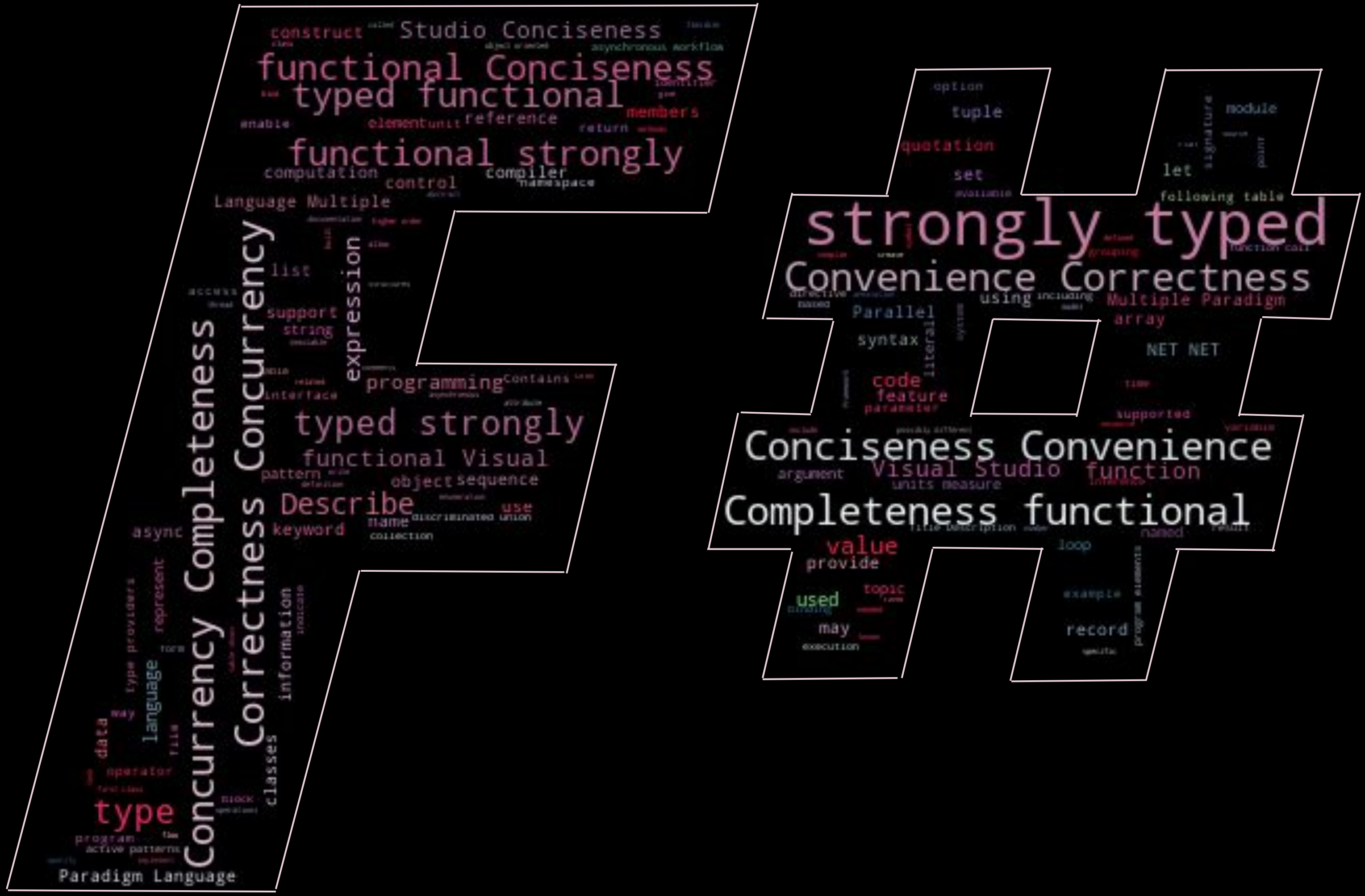
Programación Funcional: F#

Brayan Esteban Garzón
Federico Gómez
Juan Camilo Lozano

Contenido de la presentación

- Conceptos clave y principales características
- ¿Por qué F#?
- Referencia del Lenguaje
- Ejemplos prácticos


Studio Conciseness
construct
functional Conciseness
typed functional
members
reference
enable
return
functional strongly
computation
compiler
control
namespace
Language Multiple
list
access
support
string
expression
programming
Contains
interface
typed strongly
functional Visual
pattern
object
sequence
Describe
use
name
keyword
async
collection
Concurrency Completeness
Correctness Concurrency
information
represent
language
data
operator
classes
type
program
active patterns
Paradigm Language
option
tuple
quotation
set
module
let
following table
strongly typed
Convenience Correctness
using
Multiple Paradigm
Parallel
array
syntax
NET NET
code
feature
parameter
supported
Conciseness Convenience
argument
Visual Studio
function
units measure
Completeness functional
value
provide
loop
used
topic
example
may
record
execution

Conceptos clave

F#

es un lenguaje de programación:

- Fuertemente tipado
- Multiparadigma
- “Functional-First”
- De código abierto
- Multiplataforma

Principales características

Inferencia de tipos:

Pese a que F# es un lenguaje fuertemente tipado, los tipos de objetos no necesitan ser declarados explícitamente ya que son inferidos por el compilador en base al valor asignado al objeto.

Principales características

Funciones como valores:

F# permite hacer con funciones todo lo que se puede hacer con los valores de forma sencilla tales como asignación como parámetro, almacenamiento en estructuras o funciones cruzadas

Principales características

Uso de expresiones lambda:

Las expresiones lambda son especialmente útiles para operar en colecciones a la vez que se ahorra la declaración de una función, y tienen esta forma:

```
fun parameter-list -> expression
```

Principales características

Generalización automática:

El compilador analiza cada parámetro en una función dada y determina si la función tiene dependencia de un tipo específico de dato.

```
let max a b = if a > b then a else b
```

La función max permite la generalización de sus parámetros ya que los operadores de la función no hacen uso de un tipo específico.

Principales características

Aplicación parcial de argumentos:

Mediante el método de control de argumentos *"currificación"* si se proporciona a una función un número menor de argumentos que el esperado se creará una nueva función que espera los argumentos restantes.

```
let cylinderVolume radius length =
    // Define a local value pi.
    let pi = 3.14159
    length * pi * radius * radius
```

```
let smallPipeRadius = 2.0
let bigPipeRadius = 3.0
```

Principales características

Se definen las funciones que recogen la longitud como argumento restante:

```
let smallPipeVolume = cylinderVolume smallPipeRadius
let bigPipeVolume = cylinderVolume bigPipeRadius
```

Se proporciona el argumento adicional para diversas longitudes:

```
let length1 = 30.0
let length2 = 40.0
let smallPipeVol1 = smallPipeVolume length1
let smallPipeVol2 = smallPipeVolume length2
let bigPipeVol1 = bigPipeVolume length1
let bigPipeVol2 = bigPipeVolume length2
```

¿Por qué usar F#?

Concisión:

F# no está lleno de "ruido" de codificación , como llaves, punto y coma, etc.

Casi nunca se tiene que especificar el tipo de un objeto, gracias a un potente sistema de inferencia de tipo. Y en comparación con C#, generalmente requiere menos líneas de código para resolver el mismo problema.

¿Por qué usar F#?

Implementation	C#	F#
Braces	56,929	643
Blanks	29,080	3,630
Null Checks	3,011	15
Comments	53,270	487
Useful Code	163,276	16,667
App Code	305,566	21,442
Test Code	42,864	9,359
Total Code	348,430	30,801

¿Por qué usar F#?

Conveniencia:

Muchas tareas de programación comunes son mucho más simples en F#. Esto incluye cosas como crear y usar definiciones de tipos complejos, hacer el procesamiento de listas, comparación e igualdad, máquinas de estado y mucho más.

¿Por qué usar F#?

Exactitud:

F# tiene un poderoso sistema de tipado que evita una gran cantidad de errores de excepción de referencia nula.

Los valores son inmutables por defecto, lo que evita una gran clase de errores de asignación y compilación.

¿Por qué usar F#?

Concurrencia:

F# tiene una serie de bibliotecas incorporadas para ayudar cuando ocurre más de una cosa a la vez. La programación asíncrona es muy fácil de implementar, al igual que el paralelismo.

Además, como las estructuras de datos son inmutables por defecto, evitar bloqueos es mucho más fácil.

¿Por qué usar F#?

Completitud:

Aunque es un lenguaje funcional en el fondo, F# admite otros estilos que no son 100% puros, lo que hace que sea mucho más fácil interactuar con el mundo “no puro” de sitios web, bases de datos, otras aplicaciones, etc.

En particular, F# está diseñado como un lenguaje híbrido funcional/OO, por lo que puede hacer prácticamente todo lo que C# puede hacer.

¿Por qué usar F#?

Prototipos veloces:

Usando F# interactivo, es posible ejecutar código de inmediato sin compilarlo previamente, lo que facilita la exploración fluida de problemas.

¿Por qué usar F#?

Ejecución eficiente:

F# presenta compilación en **tiempo de ejecución**, también llamada *traducción dinámica*. El código de F# corre sin cambios en sistemas de 32 y 64 bits utilizando las instrucciones disponibles para cada arquitectura.

Lo anterior resulta en código que corre a velocidades mucho mayores que lenguajes como Python, JavaScript y en algunos casos significativamente más rápido que C#.

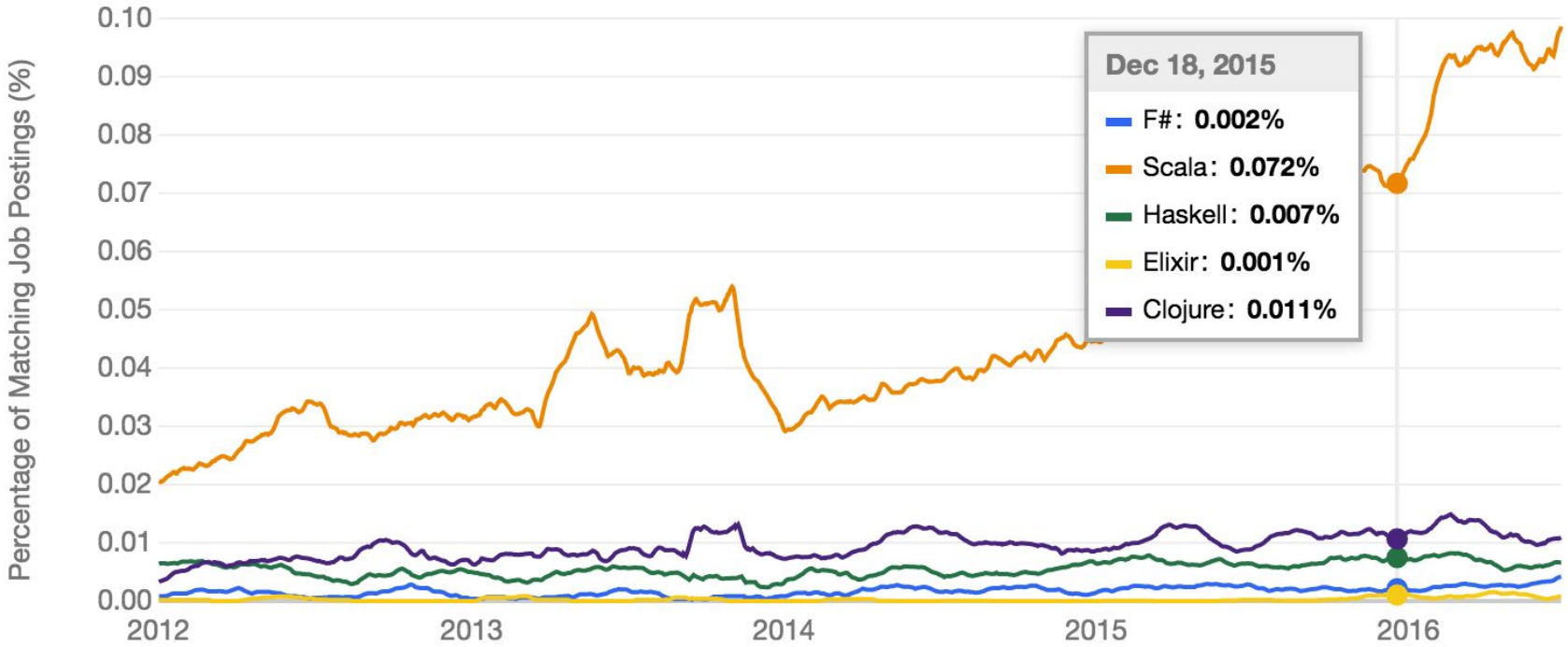
Desventajas de F#

- Soporte pobre o inexistente para desarrollo de aplicaciones Web, Android o iOS.
- Menos desarrolladores - Existen aproximadamente 100 veces más personas trabajando en C#
- Mínimo soporte para hacer refactorización, no hay soporte para herramientas de diseño de GUI.
- Vive a la sombra de C#

F# × Scala × Haskell × Elixir × Clojure × + Add Term

Find Trends

Scale: **Absolute** | Relative



Referencia del Lenguaje

Sintaxis de F# en 60 segundos (o un poco más)

Diferencias clave entre F# y una sintaxis estándar de C:

- No se usan corchetes para delimitar bloques de código. En cambio, se usa identación. (Como en Python)
- Para separar parámetros se usa un espacio a diferencia de una coma.

A continuación vamos a revisar un script que contiene las estructuras más comunes para programar en F#.

Input, output y mutables

```
open System
let a = Console.ReadLine()
printfn "%s" a

let mutable a = 8
a <- 16
```

Funciones

Las funciones se definen mediante la palabra clave **let** o, si la función es recursiva, mediante la combinación de palabras clave **let rec**.

```
let cylinderVolume radius length =
    // Define a local value pi.
    let pi = 3.14159
    length * pi * radius * radius
```

```
let rec fib n = if n < 2 then 1 else fib (n - 1) + fib (n - 2)
```

If...then...else expression

La expresión **if...then...else** ejecuta diferentes bifurcaciones de código y se evalúa como un valor distinto según la expresión booleana especificada.

```
if x = y then "equals"
elif x < y then "is less than"
else "is greater than"
```

Loops

---

for...in

```
for pattern in enumerable-expression do
    body-expression
```

for...to

```
for identifier = start [ to | downto ] finish do
    body-expression
```

while...do

```
while test-expression do
    body-expression
```

match expression

La expresión **match** proporciona control de bifurcación basado en la comparación de una expresión con un conjunto de patrones.

```
// Match expression.
match test-expression with
    | pattern1 [ when condition ] -> result-expression1
    | pattern2 [ when condition ] -> result-expression2
    | ...
```

Expresiones con procesamiento diferido

Las expresiones con procesamiento diferido no se evalúan inmediatamente, sino cuando realmente se necesita el resultado. Esto puede ayudar a mejorar el rendimiento del código.

```
let x = 10
let result = lazy (x + 10)
printfn "%d" (result.Force())
```

Referencias

---

https://es.wikipedia.org/wiki/F_Sharp
http://www.tryfsharp.org/Explore
https://fsharpforfunandprofit.com/why-use-fsharp/
https://msdn.microsoft.com/es-es/library/dd233181(v=vs.120).aspx
https://msdn.microsoft.com/visualfsharpdocs/conceptual/visual-fsharp
http://blog.deiser.com/descubriendo-fsharp/