F# |> Wprowadzenie

Wykonanie:
Paweł Złakowski

Czym jest F#?

- Jest to język programowania, który był wzorowany na językach ML, F# najbardziej inspirował się OCaml.
- Jest językiem wieloparadygmatowym łączącym programowanie imperatywne, funkcje oraz obiektowe.
- Pierwszy raz pojawił się w 2005.
- Zaprojektowany przez Microsoft oraz Microsoft Research.

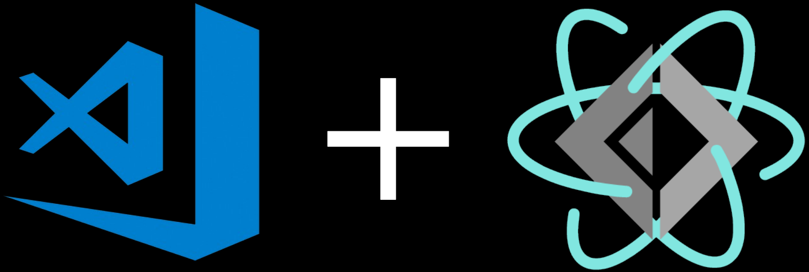
Rodzina języków ML

Założenia programowania funkcyjnego

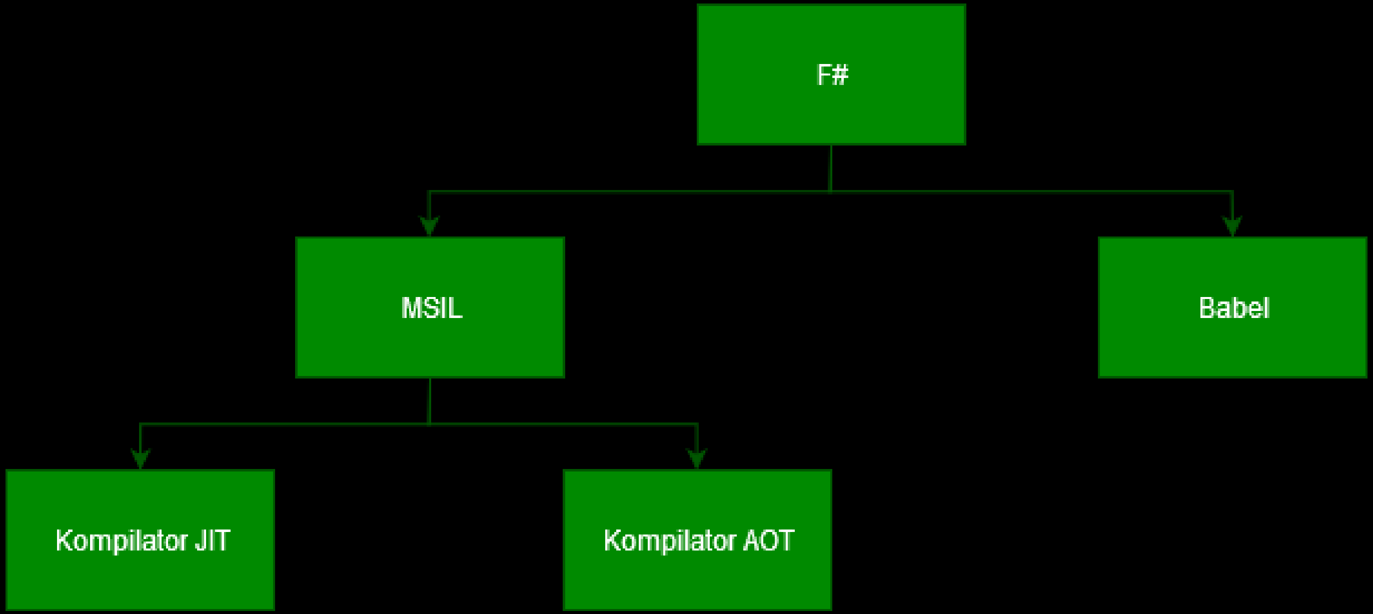
- Czyste funkcje definiujemy jako funkcja, która przyjmuje argumenty i operuje tylko i wyłącznie na nich by otrzymać rezultat. Jakiekolwiek funkcje, które odnoszą się do świata zewnętrznego np. operacje IO łamią ten kontrakt.
- Niezmienność oznacza iż nie możemy modyfikować zawartości danych. Każda transformacja danych zwraca nową wartość, która będzie przypisana do nowego identyfikatora lub nadpisze identyfikator.
- Funkcje są w ten sam uprzywilejowane jak dowolne inne typy danych.

Gdzie jest wykorzystywany?

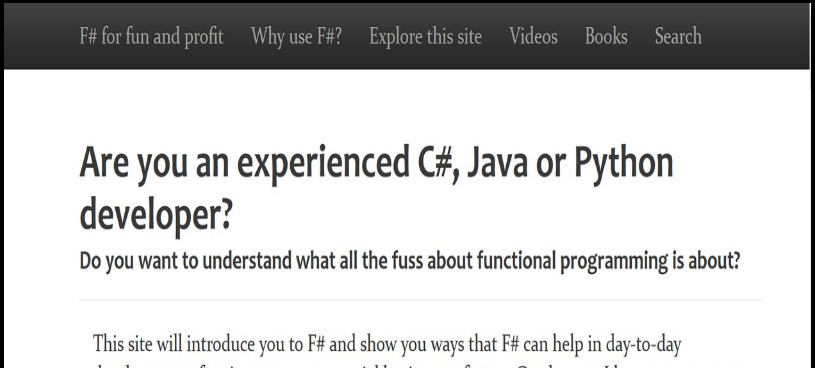
Środowiska programistyczne



Dostępne platformy kompilatora



Materiały do nauki

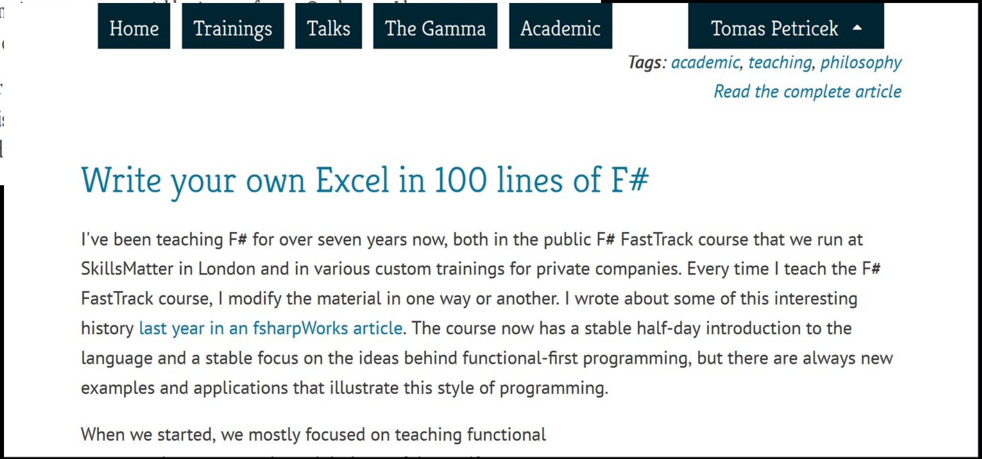


Are you an experienced C#, Java or Python developer?

Do you want to understand what all the fuss about functional programming is about?

This site will introduce you to F# and show you ways that F# can help in day-to-day development of r
mind to the joys

If you have never
language which i
open source, and
Foundation



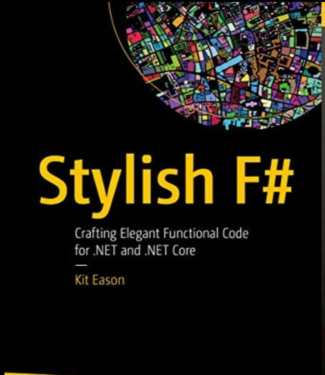
Tags: academic, teaching, philosophy

Read the complete article

Write your own Excel in 100 lines of F#

I've been teaching F# for over seven years now, both in the public F# FastTrack course that we run at SkillsMatter in London and in various custom trainings for private companies. Every time I teach the F# FastTrack course, I modify the material in one way or another. I wrote about some of this interesting history last year in an fsharpWorks article. The course now has a stable half-day introduction to the language and a stable focus on the ideas behind functional-first programming, but there are always new examples and applications that illustrate this style of programming.

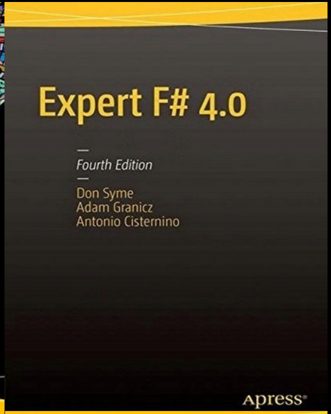
When we started, we mostly focused on teaching functional


Stylish F#

Crafting Elegant Functional Code for .NET and .NET Core

Kit Eason

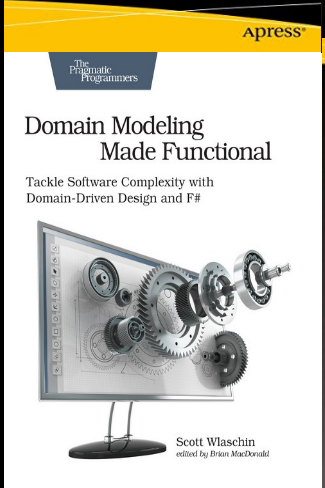
Apress


Expert F# 4.0

Fourth Edition

Don Syme
Adam Granicz
Antonio Cisternino

Apress


The Pragmatic Programmers

Domain Modeling Made Functional

Tackle Software Complexity with Domain-Driven Design and F#

Scott Wlaschin
edited by Brian MacDonald

THE EXPERT'S VOICE® IN .NET

Beginning F# 4.0

Second Edition

Robert Pickering
Kit Eason
Foreword by Don Syme, the inventor of F#

EXTRAS ONLINE

Apress

Deklaracja wartości

```
//val sampleInteger : int
let sampleInteger = 1
//val sampleString : string
let sampleString = "string"
//val sampleList : int list
let sampleList = [0..100]
//val sampleListOfSquares : int list
let sampleListOfSquares = [for i in 0..100 -> i * i]
//val mutable sampleMutableValue : int
let mutable sampleMutableValue = 5
sampleMutableValue <- 4
```

Deklaracja funkcji

```
//val add : x:('a -> 'b) -> y:'a -> 'b
let add x y = x y
//val add2 : x:(int -> 'a) -> 'a
let add2 x = add x 2
//val substract : x:int * y:int -> int
let substract (x, y) = x – y
//val multiply : x:int -> (int -> int)
let multiply x =
    let subMultiply y =
        x * y
    subMultiply
```

Pipe operator

```
let (|>) f x = x f
let add x y = x + y
let substract x y = x - y
let result = substract (add 5 2) 2
let result = 5 |> add 2 |> substract 2
```

Kompozycja funkcji

```
let add1 x = x + 1
let multiplyBy2 x = x * 2
//val add1ThenMultiplyBy2 : (int -> int)
let add1ThenMultiplyBy2 = add1 >> multiplyBy2
```

$$\begin{tikzcd}
X \arrow[r, "f"] \arrow[dr, "g \circ f"'] & Y \arrow[d, "g"] \\
 & Z
\end{tikzcd}$$

Pętle oraz rekurencja

```
for i = 0 to 10 do
    printf "%d " i

for i = 10 downto 0 do
    printf "%d " i

for i in 0..10 do
    printf "%d " i

let sampleList = [0..2..10]

for i in sampleList do
    printf "%d " i
```

```
let rec factorial n =
    if n <= 1 then
        1
    else
        n * factorial(n-1)

printfn "%d" (factorial 5)

let rec sumList = function
    | [] -> 0
    | head :: tail -> head + sumList(tail)

let sampleList = [1; 2; 3; 4; 5]

printfn "%d" (sumList sampleList)
```

Podstawowe kolekcje

```
let sampleList = [1; 2; 3; 4; 5]
let sampleSeq = {1..5}
let sampleTuple = (1, 1.0, "a", (1, 2))
let sampleSet = Set.ofList [1; 1; 2; 2; 3; 3]
let sampleMap = Map.ofList [(1, "a"); (2, "b")]
let sampleArray = [|1, 2, 3, 4, 5|]
```

Podstawowe funkcje wyższego rzędu

```
let data = [1; 2; 3; 4; 5]
let otherData = ["a", "b", "c", "d", "e"]

let square x = x * x

let dataMap = data |> List.map square
let dataFold = data |> List.fold (fun acc next -> acc + next) 0
let dataReduce = data |> List.reduce (fun acc next -> acc + next)
let dataFilter = data |> List.filter (fun value -> value % 2 = 0)
let dataZip = List.zip data otherData
```

Algebraiczne typy danych

```
type Soldier =
    | Private of Person
    | PrivateSecondClass of Person
    | PrivateFirstClass of Person
    | Specialist of Person
    | Corporal of Person
```

```
type Person =
    { FirstName: string
      LastName: string }
```

Dopasowanie wzorców

```
type Speed = Speed of float

let detectSpeed (speed: Speed) =
    match speed with
    | Speed 0.0 -> printfn "Hey, you're not moving at all."
    | Speed x when x > 90.0 -> printfn "Hey, you should slow down."
    | Speed x -> printfn "Your actual speed is: %f" x
```

Wartość opcjonalna

```
let validInteger = Some 5
let invalidInteger = None

match validInteger with
| Some value -> printfn "%d" value
| None -> printfn "Invalid value"
```

```
let divide x y =
    match (x, y) with
    | (_, 0) -> None
    | (x, y) -> Some (x / y)
```

Aktywne wzorce

```
let (|Even|Odd|) number =
    if number % 2 = 0 then
        Even
    else
        Odd

match 5 with
| Even -> printfn "Number is even."
| Odd -> printfn "Number is odd."
```

```
let (|Contains|_|) (element: string) (input: string) =
    if input.Contains(element) then
        Some ()
    else
        None

let ala = "Ala ma kota"

match ala with
| Contains "kota" -> printfn "Ala ma kota"
| _ -> printfn "Ala nie ma kota."
```

Wyrażenia obliczeń

```
type LoggingBuilder() =
    let log p = printfn "expression is %A" p

    member this.Bind(x, f) =
        log x
        f x

    member this.Return(x) =
        x
```

```
let logger = new LoggingBuilder()

let loggedWorkflow =
    logger {
        let! x = 42
        let! y = 43
        let! z = x + y
        return z
    }
```

Osobna prezentacja na temat wyrażeń obliczeń

https://bit.ly/2tK81Sx

Deklaracja jednostek miar

```
[<Measure>]
type m
[<Measure>]
type nm
[<Measure>]
type s
[<Measure>]
type kg
[<Measure>]
type N = (kg * m) / s^2

let metersToNanometers meters = meters / 1e-09<m/nm>
let velocity (meters: float<m>) (seconds: float<s>) = meters / seconds
```

Programowanie obiektowe w F#

```
type Square(side: float) =
    member this.Side = side

    member this.Area = this.Side * this.Side
    member this.Display = printfn "%fx%f" this.Side this.Side


 type Vehicle() =
    abstract member TopSpeed: unit -> int
    default this.TopSpeed() = 60

 type Rocket() =
    inherit Vehicle()
    override this.TopSpeed() = base.TopSpeed() * 10
```

```
type IAddable<'a> =
    abstract member Add: 'a -> 'a -> 'a

type AddService() =
    interface IAddable<int> with
        member this.Add x y =
            x + y

let adder = AddService() :> IAddable<int>
printfn "%d" (adder.Add 5 2)
```

Źródła

- https://en.wikibooks.org/wiki/F_Sharp_Programming
- https://fsharpforfunandprofit.com/
- Expert F# 4.0 Don Syme Adam Granicz Antonio Cisterino 2015 Apress
- Get Programming with F# Isaac Abraham 2018 Manning

Koniec