Quick Introduction to F#

Sharif University of Technology, Tehran, Iran

Presenters

Aryan Ahadinia

B.Sc. student, Computer Engineering, Sharif University of Technology

Soheil Mahdizadeh

B.Sc. student, Computer Engineering, Sharif University of Technology

Design of Programming Languages

Department of CE, Sharif University of Technology, Tehran, Iran

Spring 2021 (Feb 2021 - Jul 2021)

Instructor:

Dr. M. Izadi

Resources are available in GitHub.

https://github.com/AryanAhadinia/f-sharp-presentation.git

F# ID card

Paradigm	Functional, Imperative, Object-Oriented, Metaprogramming, Reflective, Concurrent
Designed by	Don Syme, Microsoft Research
Developer	Microsoft, The F# Software Foundation
Family	ML
First appeared	1.0 / 2005
Stable release	5.0 / November 10, 2020
Typing discipline	Static, Strong, Inferred

F# ancestors and descendants (1)

Influenced by

C#,

Erlang,

Haskell,

ML,

OCaml,

Python,

Scala

Influenced

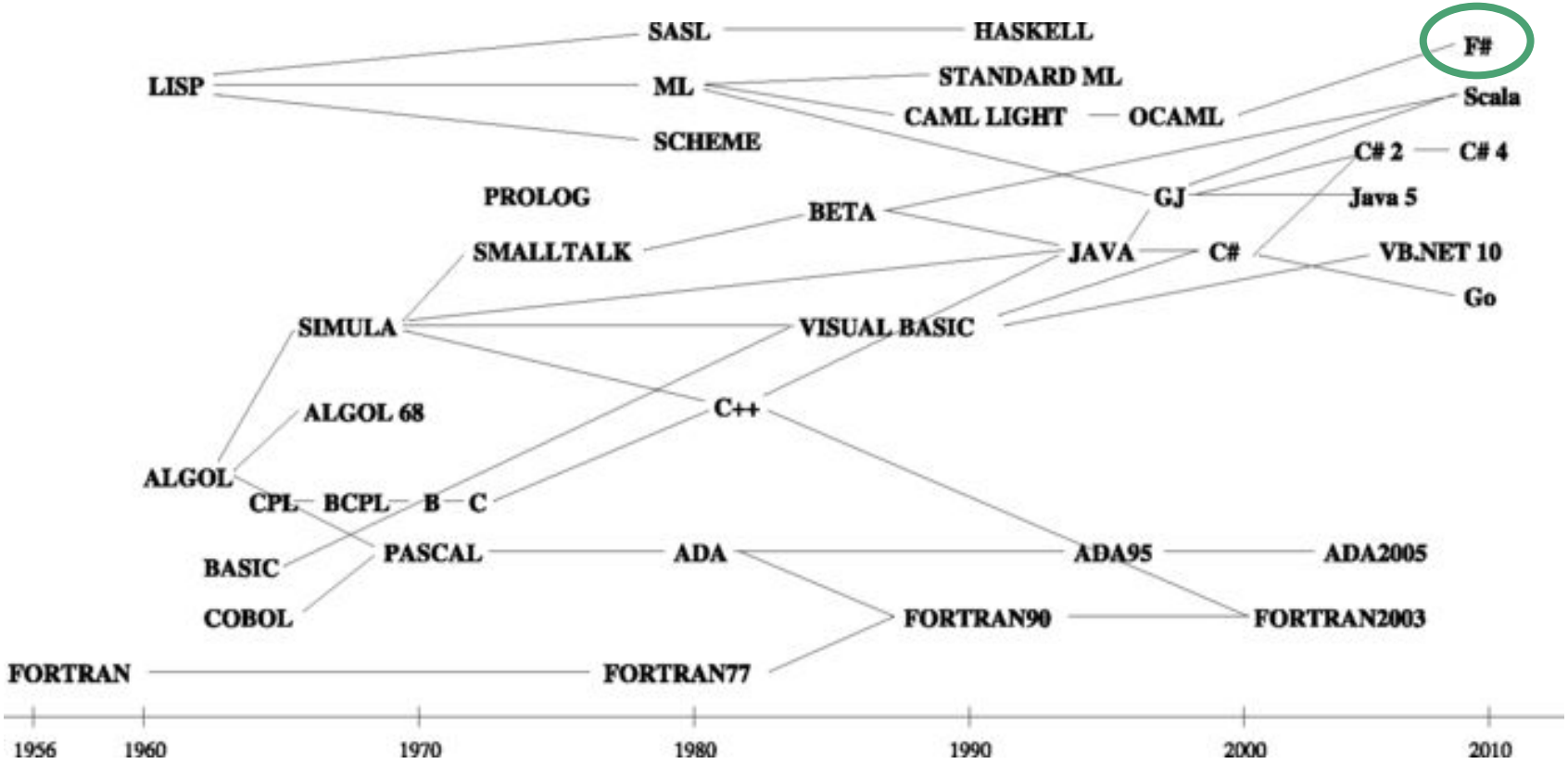
C#,

Elm,

F*,

LiveScript

F# ancestors and descendants (2)



F# platforms

Initial release of F#, Version 1.X

For Windows

2005

2010

Version 2.0

Extended to macOS and Linux

Version 3.0

For Windows, macOS, Linux, GPU, JavaScript

2012

Key features in updates

v1.x: 2005

- Functional programming
- Discriminated unions
- Tuples
- Pattern matching
- .NET interoperability
- Type Abbreviation
- Nested modules

v2.0: 2010

- Units of measure
- Sequence expressions
- Asynchronous programming
- Quotations
- Native interoperability
- Named arguments
- Optional arguments
- Array slicing

v3.x: 2012

- Type Providers
- CLImutable attribute
- Triple-quoted Strings
- Named Union type fields
- Extensions to array slicing

v4.x: 2016

- Printf on unitized values
- Primary constructors as functions
- Error message improvements
- Underscore in numeric literals
- “isReadOnly” structs
- Implicit yields

v5.0: 2020

- Improved stack traces in
- Improved .NET interop
- Improved compiler performance
- Improved compiler analysis for library authors
- Support for Jupyter, nteract, and VSCode Notebooks

Simple Program in F#

```fsharp
module Program =

    let x = 5

    let square x = x * x

    [<EntryPoint>]
    let main argv =
        printfn "Value of x is %d" x
        printfn "Square of x is %d" (square x)
        0 // return 0
```

Runtime Terminal

```
Value of x is 5
Square of x is 25
```

Recursive

```
module Recursive =
    let n = 5

    let rec fibb n =
        if (n = 0) then 1
        else if (n = 1) then 1
        else (fibb (n - 1)) + (fibb (n - 2))

    [<EntryPoint>]
    let main argv =
        printfn "Value of n is %d" n
        printfn "Fibb(%d) is %d" n (fibb n)
        0 // return 0
```

Runtime Terminal

```
Value of n is 5
Fibb(5) is 8
```

Mutually Recursive

```
module MutuallyRecursive =

    let rec even x =
        if x = 0 then true
        else odd (x-1)
    and odd x =
        if x = 0 then false
        else even (x-1)

    [<EntryPoint>]
    let main argv =
        printfn "Is 5 odd? %b" (odd 5)
        printfn "Is 5 even? %b" (even 5)
        printfn "Is 6 odd? %b" (odd 6)
        printfn "Is 6 even? %b" (even 6)
        0 // return 0
```

Runtime Terminal

```
Is 5 odd? true
Is 5 even? false
Is 6 odd? false
Is 6 even? true
```

Immutable and Mutable Variables (1)

```
module Immutability =

    let number = 5

    [<EntryPoint>]
    let main argv =
        number <- 3
        printfn "Number is %d" number
        0 // return 0
```

Compile Time Terminal

```
... : error FS0027: This value is
not mutable. Consider using the
mutable keyword, e.g. 'let mutable
number
= expression'. ...
```

Immutable and Mutable Variables (2)

```
module Immutability =

    let mutable number = 5

    [<EntryPoint>]
    let main argv =
        number <- 3
        printfn "Number is %d" number
        0 // return 0
```

Runtime Terminal

```
Number is 3
```

Type Declaration

```
module Program =
    let x: int = 5

    let odd (n: int): bool =
        if ((n % 2) <> 0) then true
        else false

    [<EntryPoint>]
    let main argv =
        printfn "%b" (odd x)
        0 // return 0
```

Runtime Terminal

```
true
```

Type Inference (1)

```
module Program =
    let x = 5

    let odd n =
        if ((n % 2) <> 0) then true
        else false

    [<EntryPoint>]
    let main argv =
        printfn "%b" (odd x)
        0 // return 0
```

Runtime Terminal

```
true
```

Strong Typing

```
module StronglyTyped =

    let mutable number = 5

    [<EntryPoint>]
    let main argv =
        number <- "salam"
        0 // return 0
```

Compile Time Terminal

```
... : error FS0001: This expression
was expected to have type↔
'int'    ↔but here has type↔
'string' ...
```

Type Inference (2)

```
module Program =
    let x = 5
    let b = false

    let odd n =
        if ((n % 2) <> 0) then true
        else false

    [<EntryPoint>]
    let main argv =
        printfn "%b" (odd x)
        printfn "%b" (odd b)
        0 // return 0
```

Compile Time Terminal

```
... : error FS0001: This expression
was expected to have type↔
'int'    ↔but here has type↔
'bool' ...
```

Type Inference (3)

```
let func n =
    if (n = 0) then 0
    else true
```

Compile Time Terminal

```
... : error FS0001: All branches of
an 'if' expression must return
values of the same type as the
first branch, which here is 'int'.
This branch returns a value of
type 'bool'. ...
```

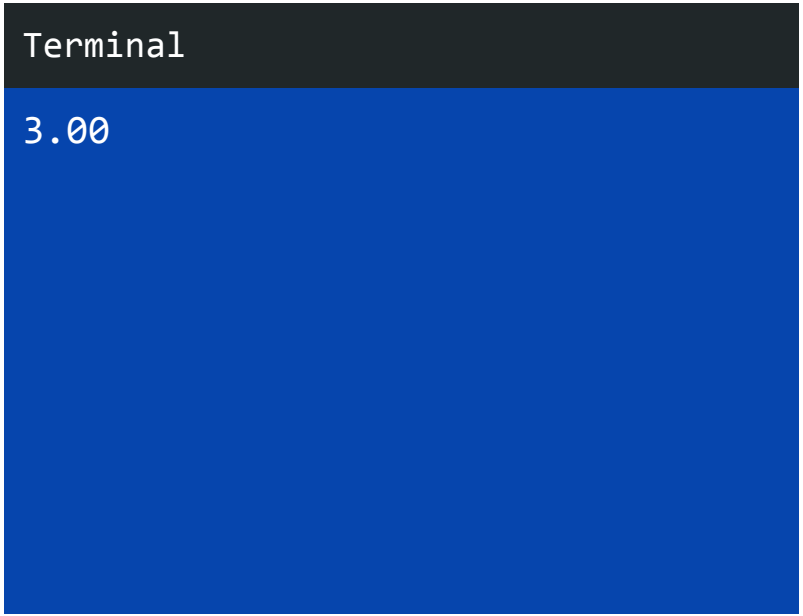
Unit of Measures (1)

```
[<Measure>] type m
[<Measure>] type s

let V = 1.0f<m/s>   // Type: float<m/s>

let x = 4.0f<m>     // Type: float<m>
let t = 2.0f<s>     // Type: float<s>
let v = x / t       // Type: float<m/s>

printfn "%.2f" (V + v)
```



Unit of Measures (2)

```
[<Measure>] type m
[<Measure>] type s

let V = 1.0f<m/s>   // Type: float<m/s>

let x = 4.0f<m>     // Type: float<m>
let t = 2.0f<s>     // Type: float<s>
let v = x / t       // Type: float<m/s>

printfn "%.2f" (V + x)
```

Compile Time Terminal

```
... : error FS0001: The unit of
measure 'm' does not match the
unit of measure 'm/s' ...
```

Pattern Matching

```
module PatternMatching =

    let numberList = [1 .. 100]

    let rec sumOfList L =
        match L with
        | [] -> 0
        | car::cdr -> car + (sumOfList cdr)

    [<EntryPoint>]
    let main argv =
        printfn "1 + 2 + 3 + .. + 100 = %d" (sumOfList numberList)
        0 // Return an integer exit code
```

Terminal

```
1 + 2 + 3 + .. + 100 = 5050
```

Discriminated Union Example

```
module UnionExample =
    type Shape =
    | Rectangle of width : float * length : float
    | Circle of radius : float
    | Prism of width : float * float * height : float
    let rect = Rectangle(length = 1.3, width = 10.0)
    let circ = Circle (1.0)
    let prism = Prism(5., 2.0, height = 3.0)
    let getShapeWidth shape =
        match shape with
        | Rectangle(width = w) -> printfn "Rectangle with width %f" w
        | Circle(radius = r) -> printfn "Circle with diameter %f" (2. * r)
        | Prism(width = w) ->  printf "Prism with width %f" w
    (getShapeWidth rect)
    (getShapeWidth circ)
    (getShapeWidth prism)
```

Terminal

```
Rectangle with width 10.000000
Circle with diameter 2.000000
Prism with width 5.000000
```

Lazy Evaluation, Fun Expression and Pipelining

```
module LazyEvaluationAndFunExp =
    let isPrime n =
        if n = 1 then false else
            let rec check i =
                i > n/2 || (n % i <> 0 && check (i + 1))
            check 2
    let numberList = [1..50]
    let primeInListLessThan20 L =
        L
        |> List.filter isPrime
        |> List.filter (fun x ->  x < 20)
    [<EntryPoint>]
    let main argv =
        let result = (primeInListLessThan20 numberList)
        for i in result do
            printfn "%d" i
        0 // Return an integer exit code
```

Terminal

```
2
3
5
7
11
13
17
19
```

Lazy Expression

```
module PatternMatching =

    let evaluateValue n =
        printfn "Value evaluated"
        n * n * n

    let result = lazy (evaluateValue 10)

    [<EntryPoint>]
    let main argv =
        for i in 1 .. 3 do
            printfn "Call #%d result: %d" i (result.Force())
        0 // return 0
```

Terminal

```
Value evaluated
Call #1 result: 1000
Call #2 result: 1000
Call #3 result: 1000
```

Asynchronous Example

```
open System
let userTimerWithAsync =
    // create a timer and associated async event
    let timer = new System.Timers.Timer(2000.0)
    let timerEvent =
            Async.AwaitEvent (timer.Elapsed)
            |> Async.Ignore
    // start
    printfn "Waiting for timer at %O" DateTime.Now.TimeOfDay
    timer.Start()
    // keep working
    printfn "Doing something useful while waiting for event"
    //waiting for the async to complete
    Async.RunSynchronously timerEvent
    // done
    printfn "Timer ticked at %O" DateTime.Now.TimeOfDay
```

Terminal

```
Waiting for timer at
17:44:12.4007182
Doing something useful while
waiting for event
Timer ticked at 17:44:14.4289524
```

Object Orientation Example

```
module ObjectOrientation =
    type Person(firstName,lastName,age) =
        //Private Fields
        let privateFirstName = firstName
        let privateLastName = lastName
        let mutable privateAge = age
        member this.getFirstName = privateFirstName
        member this.getLastname = privateLastName
        member this.getAge = privateAge
        member this.addAge y =
            privateAge <- privateAge + y
    let person1 = new Person("Soheil", "Mahdizadeh", 20)

    printfn "%s's Age Before Change is %d" person1.getFirstName person1.getAge
    (person1.addAge 5)
    printfn "%s's Age After Change is %d" person1.getFirstName person1.getAge
```

Terminal

```
Soheil's Age Before Change is 20
Soheil's Age After Change is 25
```

Struct Example

```
module Struct =
    type Line = struct
        val X1 : float
        val Y1 : float
        val X2 : float
        val Y2 : float

        new (x1, y1, x2, y2) =
        {X1 = x1; Y1 = y1; X2 = x2; Y2 = y2;}
    end
    let calcLength(a : Line) =
        let sqr a = a * a
        sqrt(sqr(a.X1 - a.X2) + sqr(a.Y1 - a.Y2) )

    let aLine = new Line(1.0, 1.0, 4.0, 5.0)
    let length = calcLength aLine
    printfn "Length of the Line: %g " length
```

Terminal

```
Length of the Line: 5
```

Explicit Reference Example

```
type Incrementor(z) =
    member this.Increment(i : int byref) =
       i <- i + z
let incrementor = new Incrementor(1)
let mutable x = 10
//Not recommended: Does not actually increment the variable.
incrementor.Increment(ref x)
printfn "%d" x
let mutable y = 10
incrementor.Increment(&y)
printfn "%d" y
let refInt = ref 10
incrementor.Increment(refInt)
printfn "%d" !refInt
```

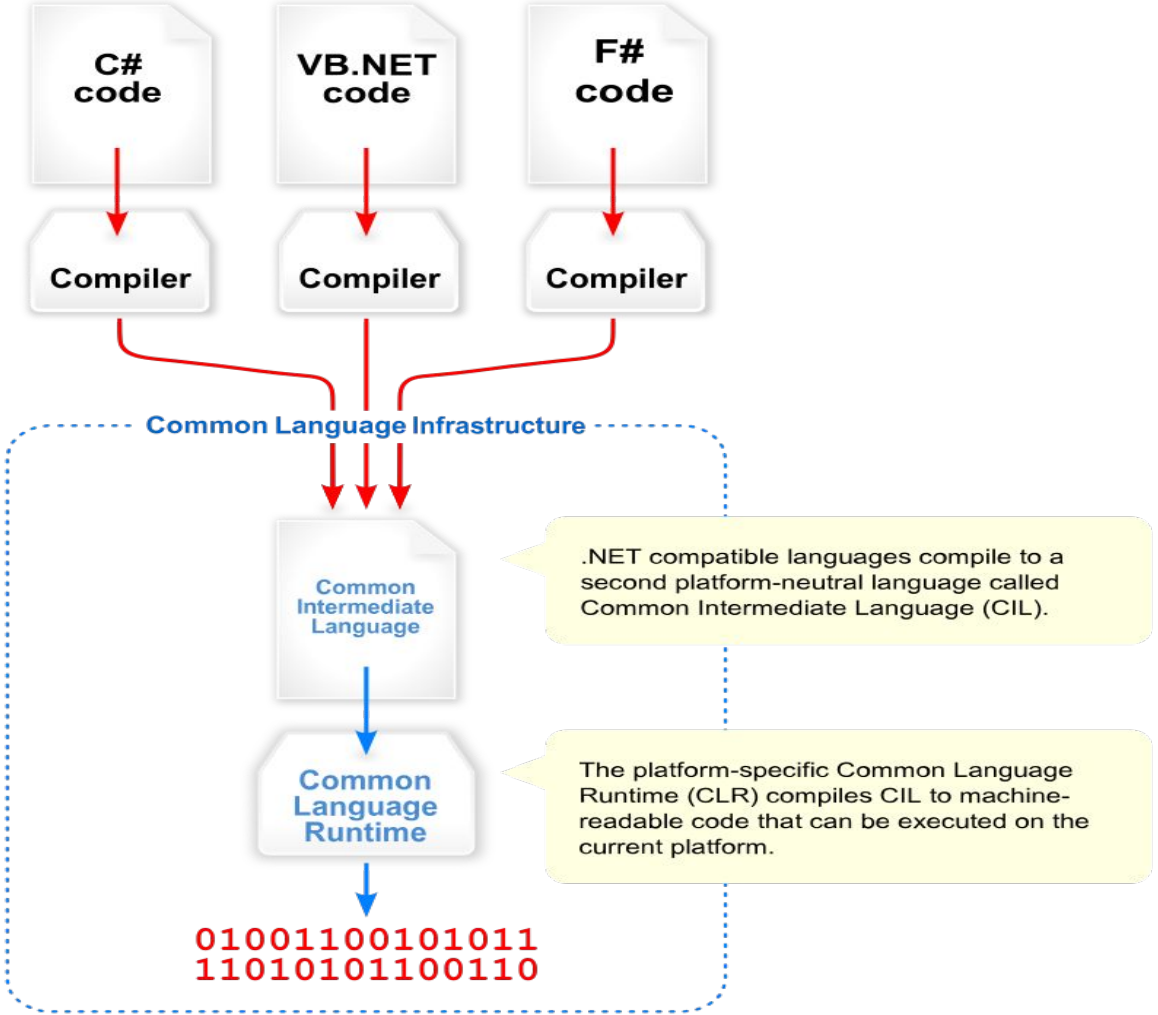
Terminal

```
10
11
11
```

Programs Written in F#:

1. Microsoft Security Risk Detection
2. F# tools in VS
3. IronJS
4. In parts of Azure and Office
5. The Xbox Live TrueSkill Algorithm

Common Language Infrastructure (CLI) :



How to get started?

1. Install .NET Core SDK
2. Install Visual Studio Code
3. Install Ionide-fsharp plugin in VSCode.
4. Install .NET Extension Pack
5. Enjoy!