# F#

The F stands for Fun!

Nathan Mendel
Anthony Orlowski
Nathan Weir

# Background

- Started at Microsoft Research under Don Syme in 2002
- Commercially released in Visual Studio 2010
- Influenced by Objective Caml, C#, Haskell

# F# is…

- Whitespace sensitive
- Strong, inferred typing
- Multi-paradigm
  - Functional (impure), imperative, object-oriented
- First-class .NET language
  - Easy to reference in C# projects
  - Make use of .NET libraries

# Key Ideas

- [4; 9; 10]
- (4, 15, “foo”)
- Let vs let rec
- Let mutable, <-
- Pattern matching

```
let rec fib (x : int) =
    match x with
    | 0 -> 1
    | 1 -> 1
    | x -> (fib (x - 1)) + (fib (x - 2))
```

# OOP Support

```
type [<AbstractClassAttribute>]
    Building(city: City) =
    let mutable _city = city
    let mutable _fashionableUnits : List<Unit> = new List<Unit>()
    let mutable _actions = []

    interface IIcon with
        member x.Icon = "HUD/default“


    member this.FashionableUnits
        with get() = _fashionableUnits and set(value) = _fashionableUnits <- value
    member this.City with get() = _city and set(value) = _city <- value

    abstract member Pos : int
    abstract member Cost : int
    abstract member Act : unit -> unit
```

# OOP Support

```
type Factory(city: City) as this =
    inherit Building(city)
    do
        this.FashionableUnits.Add(new Truck(city))
    override this.Act() = ()
    override this.Pos = 2
    override this.Cost = 1000
    interface IIcon with
        member this.Icon = "HUD/factory"
    new() =  Factory(new City(0,0,""))
```

# Interaction with libraries and pipelining

```
let isPrime (n:int) =
   let bound = int (System.Math.Sqrt(float n))
   seq {2 .. bound} |> Seq.exists (fun x -> n % x = 0) |> not
let primeAsync n =
    async { return (n, isPrime n) }
let primes m n =
    seq {m .. n}
        |> Seq.map primeAsync
        |> Async.Parallel
        |> Async.RunSynchronously
        |> Array.filter snd
        |> Array.map fst
```