# Functional Programming

## With focus on History and F# .NET

**By: Aryan Ebrahimpour**

همکاران سیستم
SYSTEM GROUP

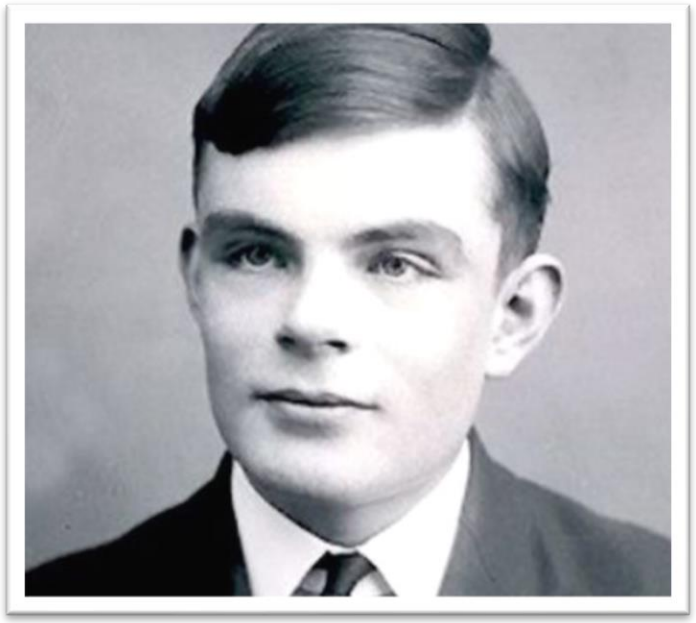
# Table of Contents

# Computation and Languages

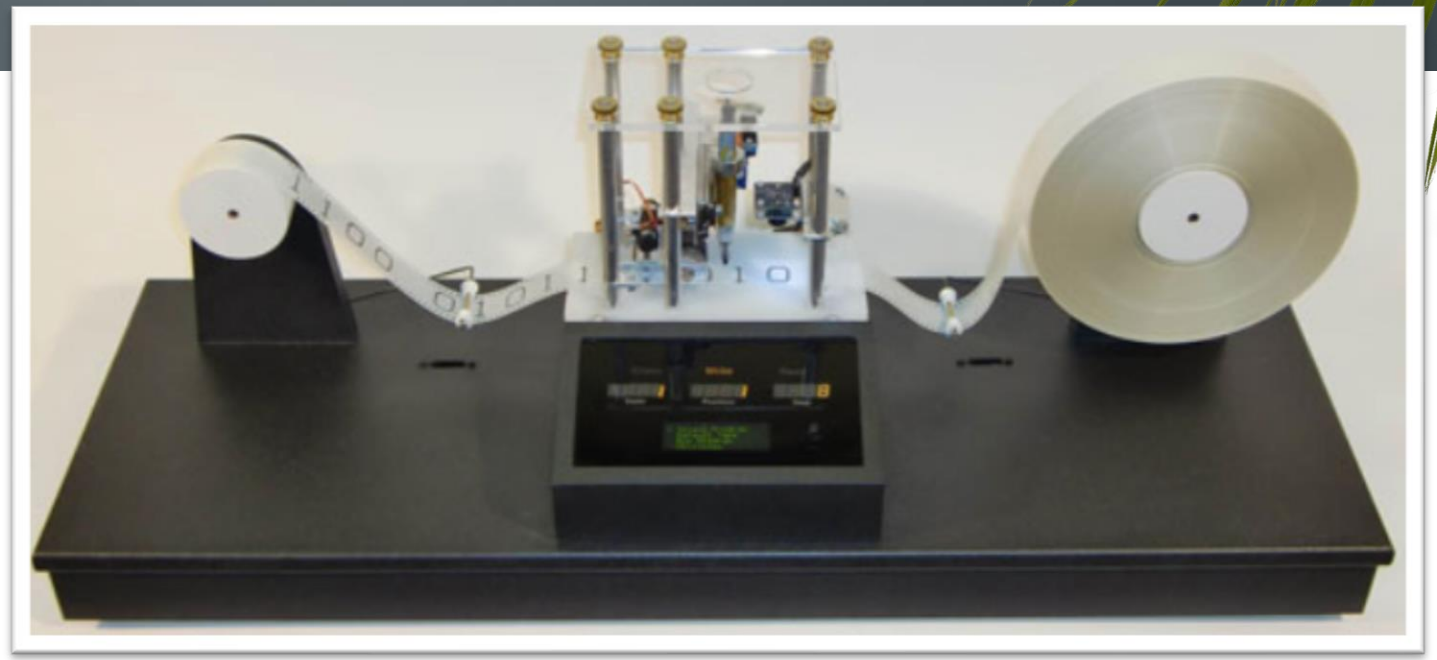
Lets see some about programming languages and history

# What is computation? (1934)

And what does it mean to compute something?



Alan Turing



- An Imperative machine
- Princeton
- Powerful Machine for Computation

# What is computation? (1934)

And what does it mean to compute something?

Alonzo Church

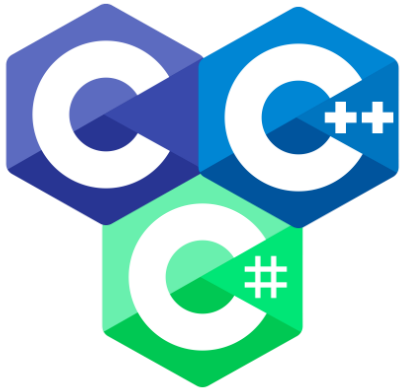
```
T = λx . λy . x
F = λx . λy . y

var T = x => y => x
var F = x => y => y

and = λa . λb . a b F

var and = a => b => a(b)(F)

and T T // result: T
and F T // result: F
and T F // result: F
```

- Close to Math and Logic, declarative expressions
- Princeton
- Powerful Machine for Computation (But powerful as Turing machines?)
  - Hint: Yes, anything you can do with one, you can do with the other

# What is computation? (1934)

And what does it mean to compute something?

## Same Power, but they gave rise to different type of PLs






### Imperative

They might added functional features overtime







### Declarative → FP

They might added some imperative workflows overtime

# Object Oriented vs Functional

## OOP

- You say "*do this, do this, then this*"
- Mutable by default / Impure
- Side-effects all over the place
- Loops to iterate
- Values and functions are different to the eyes of the programmer

## FP

- You *declare* what you want
- Immutable by default / Pure
- No/Controlled side-effects
- Recursions to iterate
- Values and Functions are defined in the same way (called first-class)

# Declarative Example

Example of finding right triangle sides in a declarative way

```
let rightTriangles = [ (a,b,c) | c <- [1..10],
                                 b <- [1..c],
                                 a <- [1..b],
                                 a^2 + b^2 == c^2]
```

# Pure Example

Example of a Pure function in C#

```
[Pure]
int Multiply(int a, int b) => a * b;
```

# FP in practice with F#

# Case 1

Hello

```cs
using System;

namespace cs
{
    0 references
    class Program
    {
        0 references
        static void Main(string[] args)
        {
            Console.WriteLine("Hello World!");
        }
    }
}
```

```fs
open System

[<EntryPoint>]
let main argv = // string [] -> int
    printfn "Hello World from F#!"
    0 // return an integer exit code
```

Case 2

What do you mean by

# 1st Class Functions

```
using System;

0 references
class MyClass {

    0 references
    public string MyName => "Aryan";

    0 references
    private int myAge = 12;

    0 references
    public int MyFunction(string s, int extra){
        return s.Length + extra;
    }

}
```

```
open System

let myName = "string" // string

let myAge = 21 // int

let myFunc (s:string, extra) = s.Length + extra // string * int -> int
```

Case 3

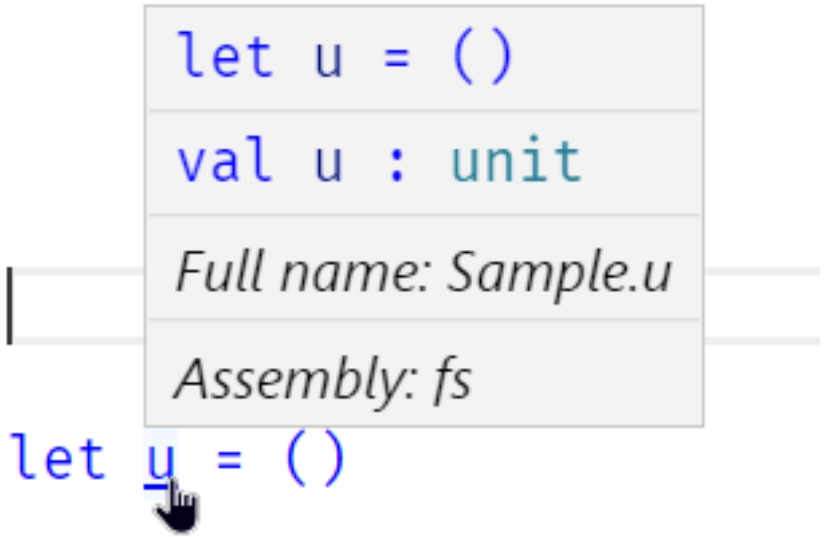
# Partial Apply

```
1   using System;
2
    0 references
3   class MyClass {
4
        0 references
5       public static int Add1(int x, int y) => x + y;
6
        0 references
7       public static Func<int, Func<int, int>> Add2 = x => y => x + y;
8
9   }
10
11  // Add1(10, 12); Add2(10)(12)
```

```
1   open System
2
3   let add1 (x, y) = x + y // int * int -> int
4
5   let add2 x y = x + y // int -> int -> int
6
7   add1 (10, 12)
8   add2 10 12
```

```
using System;

0 references
class MyClass {

    1 reference
    public static Func<string, Action<bool>> WriteInfo = name => isHappy => {
        if(isHappy) Console.WriteLine($"{name} is happy!");
        else Console.WriteLine($"{name} is not happy!");
    };

    0 references
    public Action<bool> WriteAboutAryan(string name) => WriteInfo(name);

    // WriteAboutAryan(true);

}
```

```
let writeInfo name isHappy = // string -> bool -> unit
    if isHappy then printfn "%s is happy!" name
    else printfn "%s is not happy!" name

let writeAboutAryan = writeInfo "aryan" // bool -> unit

writeAboutAryan true
```




```
let f () = 2 // equvalient of f() = 2
```

Case 4

# Types

# Models

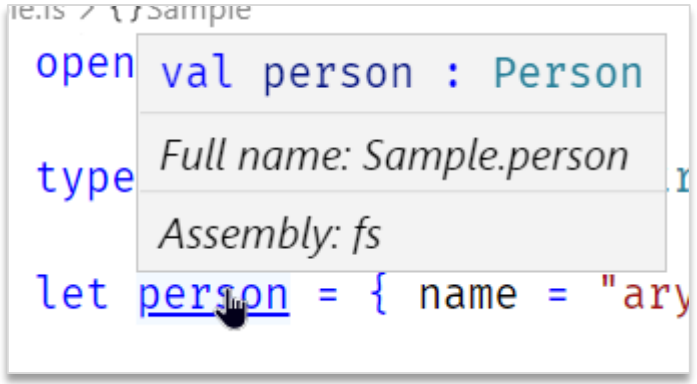
```
    2 references
3   public class Person {
        0 references
4       public string Name { get; set; }
5
        0 references
6       public string LastName { get; set;}
7
        0 references
8       public DateTime Birthday { get; set; }
9
        0 references
10      public Parents Parents { get; set; }
11
12  }
13
    1 reference
14  public class Parents {
        0 references
15      public Person Father { get; set; }
16
        0 references
17      public Person Mother { get; set; }
18  }
```

```
3   type Parents = {
4       father : Person
5       Mother : Person
6   }
7   and Person = {
8       name : string
9       lastName : string
10      birthDay : DateTime
11      parents : Parents
12  }
```

# Models Instance

```
    0 references
4   class Person {
        0 references
5       public string Name { get; set; }
6
        0 references
7       public int Age { get; set; }
8   }
9
10  // var person = new Person {
11  //     Name = "aryan",
12  //     Age = 21
13  // };
```

```
3   type Person = { name : string; age : int }
4
5   let person = { name = "aryan"; age = 21} // Person
```



# Interfaces

```
    0 references
4   public interface IVehicle
5   {
        0 references
6       string Name { get; set; }
7
        0 references
8       int GetFuel();
9
        0 references
10      Point GetLocation(bool acurate);
11  }
```

```
4   type IVehicle =
5       abstract member Name : string with get, set
6       abstract member GetFuel : unit -> int
7       abstract member GetLocation : bool -> Point
8
```

# Sum Types

```
 3   type Shape =
 4       | Diamond
 5       | Clubs
 6       | Hearts
 7       | Spades
 8
 9   type Rank =
10       | King
11       | Queen
12       | Jack
13       | Number of int
14
15   type Card = (Rank * Shape)
```

```
17   let card1 = (King, Spades) // Rank * Shape
18   let card2 = (Number 5, Hearts) // Rank * Shape
19
20   let card : Card = card2 // Card
```

**C# does not have this feature**

# Sum Types

```
 3  type Shape =
 4      | Diamond
 5      | Clubs
 6      | Hearts
 7      | Spades
 8
 9  type Rank =
10      | King
11      | Queen
12      | Jack
13      | Number of int
14
15  type Card = (Rank * Shape)
```

```
17  let calculateScore lastCard = // Rank * Shape -> int
18      match lastCard with
19      | (King | Queen), Hearts -> 10
20      | Jack, (Clubs | Spades) -> 20
21      | Queen, _ -> 24
22      | Number x, _ -> x + 10
23      | _ -> 0
```

**C# does not have this feature**

# Exceptions

```
    0 references
4   class MyException : Exception {
        0 references
5       public string ExceptionTag {get;set;}
6
7   }
```

```
4   exception MyException of customTag:string
5
```

# Type aliases

```
4   type NameDictionary<'b> = Dictionary<string, 'b>
```

**In C#, there is “using x = …” syntax, but it’s limited**

# Units of measure

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

let distance = 12.0<m>
let time = 6.0<s>

let thisWillFail = distance + time
// ERROR: The unit of measure 'm' does
// not match the unit of measure 's'

let thisWorks = distance / time
// 2.0<m/s>
```

**C# does not have this feature**

Case 5

# Type Signatures

# Type Sig of Map(Select) in C# and F#

```
public static IEnumerable<TResult>
Select<TSource,TResult>(this IEnumerable<TSource>
source, Func<TSource, TResult> selector)
```

```
let map : ('T -> 'U) -> Seq<'T> -> Seq<'U>
```

# Case 6

Null? No.

```
type Option<'a> =
    | Some 'a
    | None


type Person = {
    father : Person option // also option<Person>
}


let p1 = { father = None   } // Person
let p2 = { father = Some p1} // Person
```

Case 7

# Custom Operators

# Custom Operator Definitions

```
4   let (=>) a b = (a, b) // 'a -> 'b -> 'a * 'b
5
6   let myDictionary = [ // (string * int) list
7       "Hi" => 1
8       "Hello" => 2
9       "Sup?" => 1000
10  ]
```

```
(string, int) list = list<(string, int)>
```

**C# does not have this feature**

Case 8

# Pipe and Composition

# Pipes

```
myCar
|> assemble
|> paintBody Color.Red
|> removePart (fun part -> part.IsBroken)
|> restoreRemovedParts
|> releaseToCustomer
```

# Pipes

```
4   type Person = { name : string; birthYear : int }
5
6   let manipulatePeople onlyEvens people = // bool -> Person list -> Person list
7     let isEven x = x % 2 = 0
8     people
9     |> List.filter (fun p -> if onlyEvens then isEven p.birthYear else true)
10    |> List.map (fun p ->  { p with birthYear = p.birthYear - 10 })
11  let people = [ // Person list
12      { name = "Mohammad Hassan Dehghan", 1362 }
13      { name = "Saman Hashemi", 1366 }
14  ]
15
16  people |> manipulatePeople false
17
```

# Can I define pipe myself?

```
// 'a -> ('a -> 'b) -> 'b
let (|>) x f = f x
```

```
public static G Pipe<T, G>(T x, Func<T, G> f) => f(x);
```

# Composition

```
3
4   let getFirstChar (str:string) = str.[0] // string -> char
5
6   let isLetterorDigit = System.Char.IsLetterOrDigit // char -> bool
7
8   let firstCharIsLetterOrDigit = getFirstChar >> isLetterorDigit // string -> bool
```

# Can I define composition myself?

```
// ('a -> 'b) -> ('b -> 'c) -> 'a -> 'c
let (>>) f g = fun x -> g (f x)
```

```
public static Func<A, C> Compose<A, B, C>(Func<A, B> f, Func<B, C> g) => x => g(f(x));
```

Case 9

# Collections

```fsharp
let myNumbers = [| 1; 2; 3 |] // int []

let operations = [ // (string * (int -> int -> int)) list
    "sum",  (+)
    "sub",  (-)
    "mult", (*)
    "div",  (/)
]
```

Case 10

# Type Providers

```
[<Literal>]
let ResultSample = """
{
    "message":" ",
    "result":" ",
    "nextQuestion":" ",
    "numbers" : [1,2,3],
    "rules": [
        { "number":2, "response":"Beep" },
        { "number":5, "response":"Boop" }
    ]
}
"""
type FizzResponse = JsonProvider<ResultSample>

let x = FizzResponse.GetSample()

x.
```

- Message
- NextQuestion
- Numbers
- Result
- Rules
- JsonValue

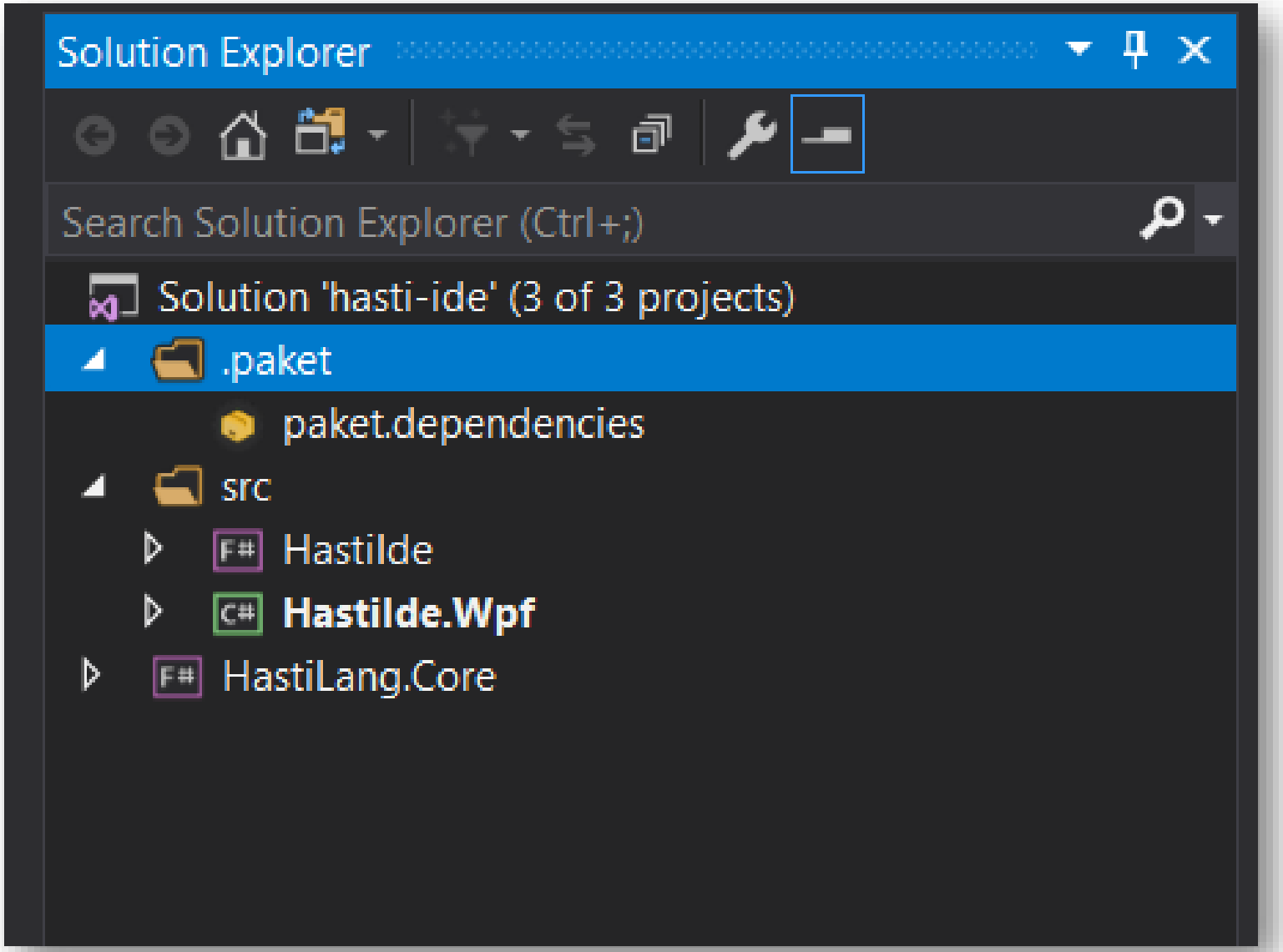
property JsonProvider<...>.Root.NextQuestion: string with get

Case 11

Can I still access to my

# C# Projects and Libs

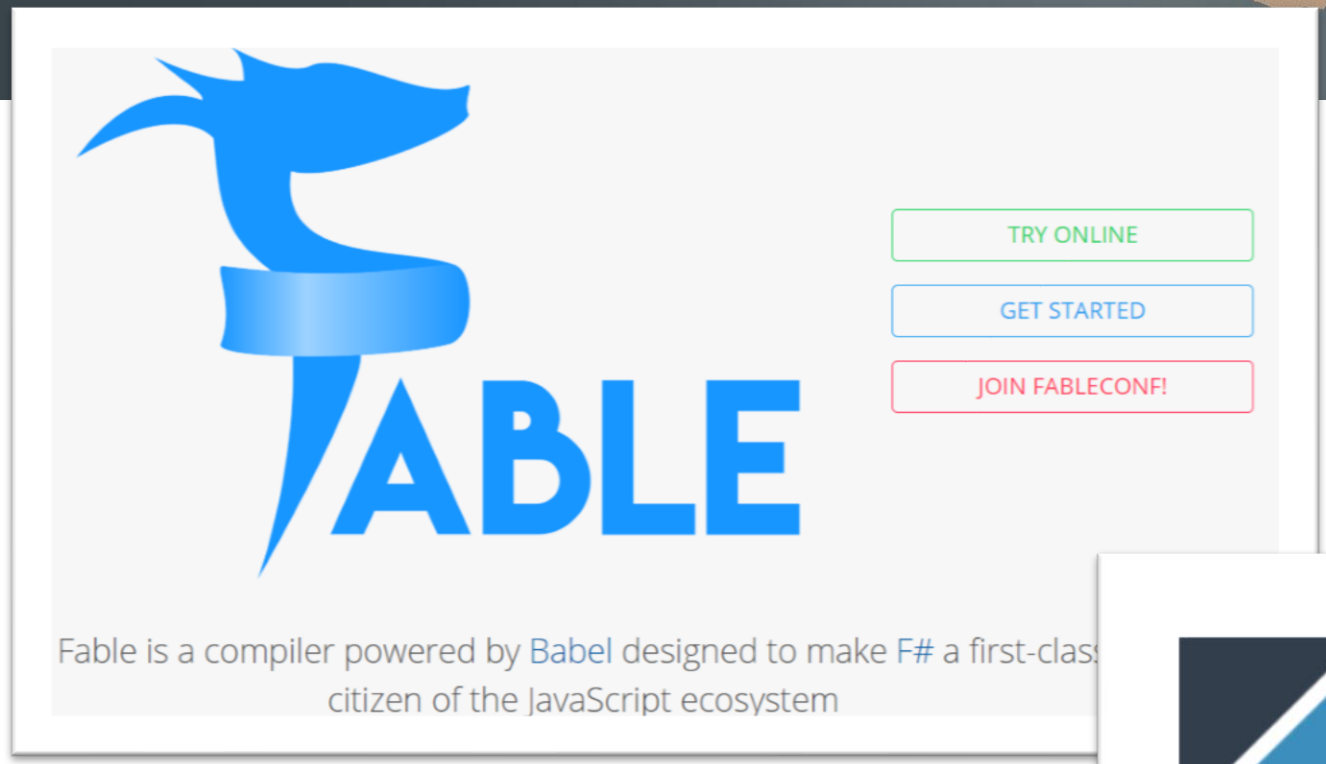
From F#?


Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'hasti-ide' (3 of 3 projects)
.paket
paket.dependencies
src
Hastilde
Hastilde.Wpf
HastiLang.Core

# Case 12

# Web

# F# and GUI – Elmish MVU





# F# and GUI – Elmish MVU

Elmish.fsx

T: > Elmish.fsx

```fsharp
type Model = int

type Msg =
| Increment
| Decrement

let init() : Model = 0 // unit → Model

let update (msg:Msg) (model:Model) = // Msg → Model → Model
    match msg with
    | Increment → model + 1
    | Decrement → model - 1

```

1- State
2- Event
3- update
4- view

```fsharp
let view model dispatch =
  div []
      [ button [ OnClick (fun _ → dispatch Decrement) ] [ str "-" ]
        div [] [ str (sprintf "%A" model) ]
        button [ OnClick (fun _ → dispatch Increment) ] [ str "+" ] ]
```

Case 13

# Parsing

# F# and Parser Combinators - FParsec

stephan-tolksdorf / **fparsec**

Code | Issues 13 | Pull requests 2 | Projects 0 | Wiki | S

A parser combinator library for F#

**228** commits | **3** branches | **11** releases

Branch: **master** | **New pull request**

**enricosada** Update projects to .NET sdk, support `netstandard2.0`, drop tfms, etc (... ...


## Daan Leijen

Researcher

### Contact Info

Email

# F# and Parser Combinators - FParsec

```
let floatBetweenBrackets = str "[" >>. pfloat .>> str "]"

test (many floatBetweenBrackets) "[2][3][4]" // result: Success [2.0; 3.0; 4.0]
```

# زبان برنامه‌نویسی هستی

زبان برنامه‌نویسی تابعی فارسی، رایگان و متن‌باز برای اهداف آموزشی

پروژه
سورس کد پروژه در گیت‌هاب

مستندات
مشاهده مستندات و قابلیت‌ها

دانلود (به زودی)
دانلود کامپایلر + محیط توسعه فارسی

وبلاگ
وبلاگ زبان برنامه‌نویسی هستی

زبان برنامه نویسی هستی

```
نام : رشته = "آرین"

اگه نام == "آر" + "ین" اونوقت
    بنویس "درسته"
از الف = 1 تا 10 اونوقت
    اگه الف % 2 == 0 اونوقت بنویس "زوج" وگرنه بنویس "فرد"
```

زبان برنامه نویسی هستی

اجرا

```
[Binding {identifier = HstIdentifier "نام";
          args = None;
          bindingType = Atomic (TypeName "رشته");
          exps = [Value (String "آرین")];}; IfStmt
   {cond = ExpOpCall {op = Operator "==";
                      exp1 = BindingExpr (HstIdentifier "نام");
                      exp2 = ExpOpCall {op = Operator "+";
                                        exp1 = ExpValue (String "آر");
                                        exp2 = ExpValue (String "ین");};};};
    thendo =
     [PrintStatement (ExpValue (String "درسته"));
```

```
درسته
فرد
زوج
فرد
زوج
فرد
زوج
```

زبان برنامه نویسی هستی

```
1
2  نام : رشته = "آرین"
3
4  اگه نام == "آر" + "ین" اونوقت
5      بنویس "درسته"
6  از الف = 1 تا 10 اونوقت
7      اگه 0 == الف % 2 اونوقت بنویس "زوج/ح/" وگرنه بنویس "فرد"
```

زبان برنامه نویسی هستی

اجرا

خطا در خط: ۷ ستون: ۴۴
اگه ۰ == الف % ۲ اونوقت بنویس "زوج/ح/" وگرنه بنویس "فرد"

انتظار: یکی از کاراکتر های ویژه «ن» یا «ر» یا «ت» انتظار میرفت

Case 14

# Package Manager (Paket)

# Nuget

- ✔ Built-in package manager
- ✘ No transitive-dependencies handling
- ✘ You have no control over conflicting packages
- ✘ No Lock file (NuGet main source is packages.config xml file)
- ✔ Command-line support
- ✘ Limited command line commands
- ✘ No other source of Code to restore in project

# Paket

- ✔ Native NuGet Support
- ✔ Handles Transitives
- ✔ Full control over conflicting packages
- ✔ “paket.lock” file for maintaining info of packages
- ✔ Command Line support
- ✔ Rich command line commands
- ✔ Git, GitHub and HTTP dependencies

# Root folder: paket.lock, paket.dependencies

paket.dependencies - Notepad

File Edit Format View Help

```
source https://www.nuget.org/api/v2

nuget Castle.Core
nuget Castle.Windsor
nuget EntityFramework
nuget LinqKit framework: >= net45
nuget log4net 2.0.0
nuget Microsoft.Bcl
```

paket.lock - Notepad

File Edit Format View Help

```
NUGET
  remote: https://nuget.org/api/v2
  specs:
    Castle.Core (3.3.3)
    Castle.Windsor (3.3.0)
      Castle.Core (>= 3.3.0)
    EntityFramework (6.1.3)
    LinqKit (1.1.2) - framework: >= net45
      EntityFramework (>= 6.0.2)
    log4net (2.0.0)
    Microsoft.Bcl (1.1.10)
```

# Every project folder has own: paket.references

paket.references - Notepad

File Edit Format View Help

```
EntityFramework
```

paket.references - Notepad

File Edit Format View Help

```
Castle.Core
Castle.Windsor
EntityFramework
LinqKit
Newtonsoft.Json
```

paket.references - Notepad

File Edit Format View Help

```
Castle.Core
Castle.Windsor
Microsoft.Bcl.Async
Microsoft.Bcl
```

# NodaTime 3.0.0-alpha01

Noda Time is a date and time API acting as an alternative to the built-in DateTime types built into the .NET framework.

ⓘ This is a prerelease version of NodaTime.

Package Manager | .NET CLI | PackageReference | **Paket CLI**

```
> paket add NodaTime --version 3.0.0-alpha01
```

⚠ The NuGet Team does not provide support for this client. Please contact its maintainers for suppo

Case 14

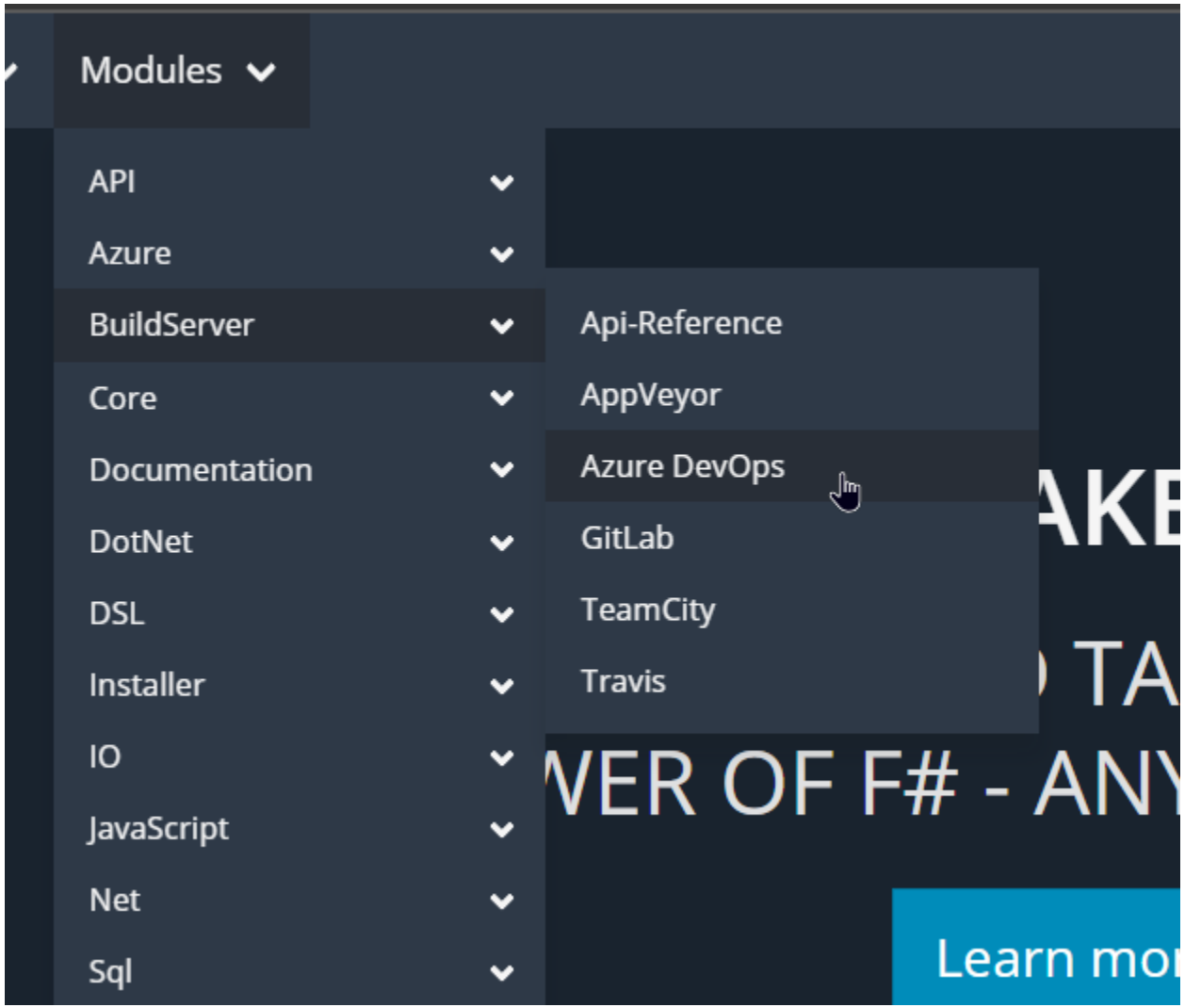
# Build System (FAKE)

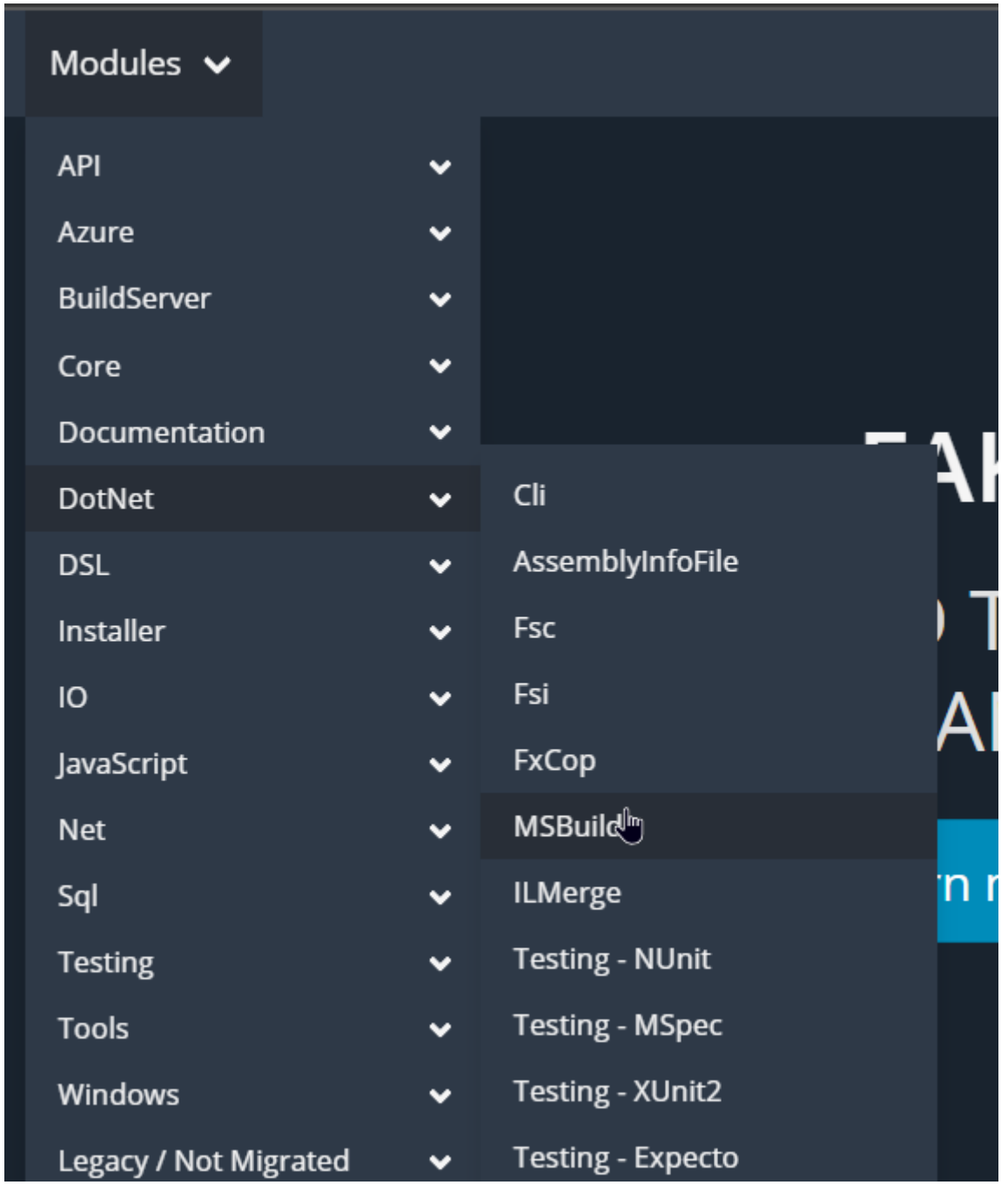
# FAKE

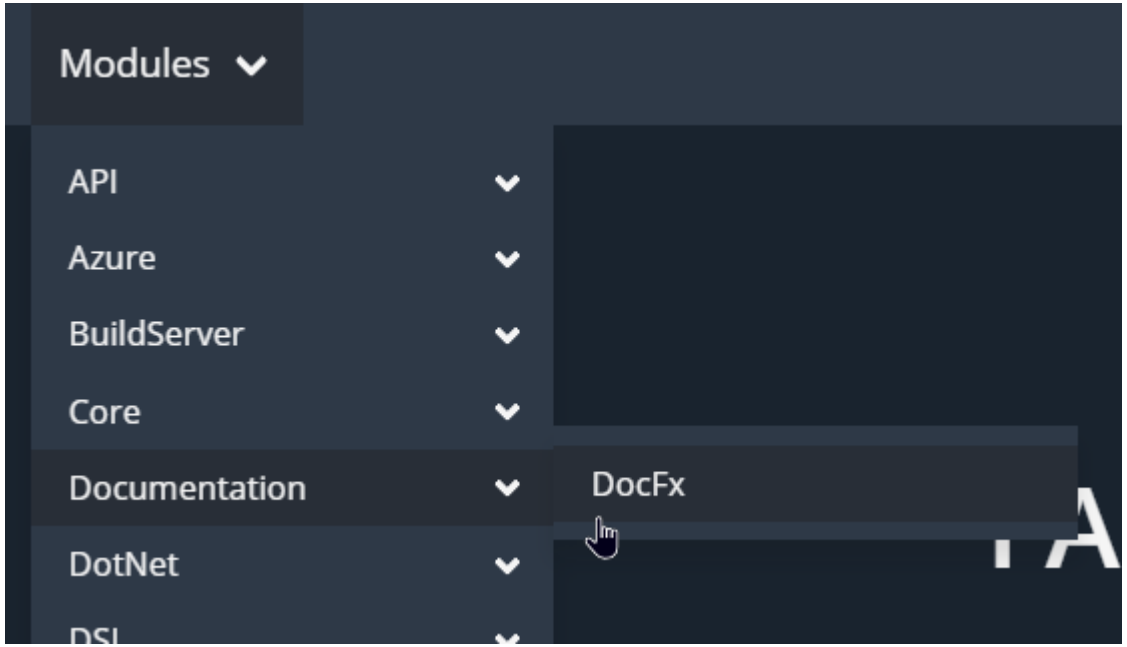
A DSL FOR BUILD TASKS AND MORE
THE POWER OF F# - ANYWHERE - ANYTIME

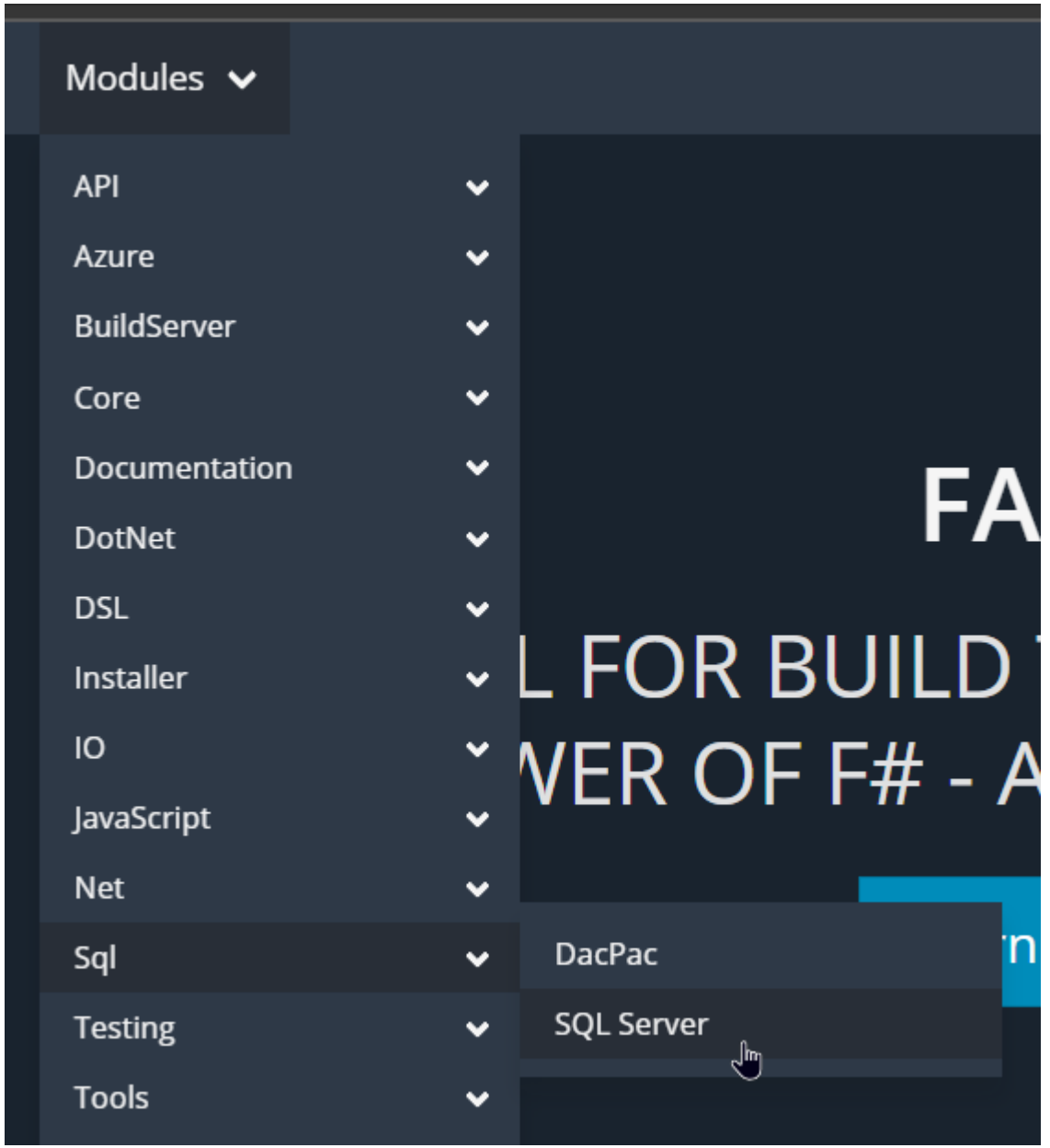
Learn more

```
open Fake.Core
open Fake.IO

// Properties
let buildDir = "./build/“

// Targets
Target.create "Clean" (fun _ ->
    Shell.cleanDir buildDir
)

Target.create "Default" (fun _ ->
    Trace.trace "Hello World from FAKE“
)

// Dependencies
open Fake.Core.TargetOperators

"Clean“
     ==> "Default“

// start build

Target.runOrDefault "Default"
```


Modules
API
Azure
BuildServer
Core
Documentation
DotNet
DSL
Installer
IO
JavaScript
Net
Sql
Api-Reference
AppVeyor
Azure DevOps
GitLab
TeamCity
Travis
VER OF F# - ANY
Learn mo


Modules
API
Azure
BuildServer
Core
Documentation
DotNet
DSL
Installer
IO
JavaScript
Net
Sql
Testing
Tools
Windows
Legacy / Not Migrated
Cli
AssemblyInfoFile
Fsc
Fsi
FxCop
MSBuild
ILMerge
Testing - NUnit
Testing - MSpec
Testing - XUnit2
Testing - Expecto


Modules
API
Azure
BuildServer
Core
Documentation
DotNet
DocFx


Modules
API
Azure
BuildServer
Core
Documentation
DotNet
DSL
Installer
IO
JavaScript
Net
Sql
Testing
Tools
DacPac
SQL Server

# Case 15

# Test

```
let ``satisfies work item #13 in tfs LOL XD`` = // bool
    assert 1 = 1
```

Case 16

# Famous Algorithms

Implemented in C# and F#

# Sum of a list

C#: Implementation of Enumerable.Sum extension method

```
public static int Sum(this IEnumerable<int> source) {
    if (source == null) throw Error.ArgumentNull("source");
    int sum = 0;
    checked {
        foreach (int v in source) sum += v;
    }
    return sum;
}
```

# Sum of a list

F#: Implementation of List.Sum

```
let rec sum list = // int list -> int
  match list with
  | [] -> 0
  | x :: rem -> x + sum rem
```

# Quick sort

```
public class QuickSortHelper
{
    2 references
    public static List<T> QuickSort<T>(List<T> values)
        where T : IComparable
    {
        if (values.Count == 0)
        {
            return new List<T>();
        }
        T firstElement = values[0];
        var smallerElements = new List<T>();
        var largerElements = new List<T>();
        for (int i = 1; i < values.Count; i++) {
            var elem = values[i];
            if (elem.CompareTo(firstElement) < 0)
            {
                smallerElements.Add(elem);
            }
            else
            {
                largerElements.Add(elem);
            }
        }
        var result = new List<T>();

        result.AddRange(QuickSort(smallerElements.ToList()));
        result.Add(firstElement);
        result.AddRange(QuickSort(largerElements.ToList()));
        return result;
    }
}
```

# C# by not using Linq (Imperative)

- ⚠ LOC: 30
- ❌ Hardly readable
- ❌ Code Noises

```
0 references
public static class QuickSortExtension
{
    2 references
    public static IEnumerable<T> QuickSort<T>(
    this IEnumerable<T> values) where T : IComparable
    {
        if (values == null || !values.Any())
        {
            return new List<T>();
        }

        var firstElement = values.First();
        var rest = values.Skip(1);

        var smallerElements = rest
                .Where(i => i.CompareTo(firstElement) < 0)
                .QuickSort();

        var largerElements = rest
                .Where(i => i.CompareTo(firstElement) >= 0)
                .QuickSort();

        return smallerElements
            .Concat(new List<T> { firstElement })
            .Concat(largerElements);
    }
}
```

# C# using Linq (FP)

- ⚠ LOC: 26
- ✔ Readable
- ⚠ Code Noises

```fsharp
let rec quicksort list = // 'a list -> 'a list
    match list with
    | [] -> []
    | firstElem::otherElements ->
        let smallerElements =
            otherElements
            |> List.filter (fun e -> e < firstElem)
            |> quicksort
        let largerElements =
            otherElements
            |> List.filter (fun e -> e >= firstElem)
            |> quicksort

        List.concat [smallerElements; [firstElem]; largerElements]
```

F#

- ✔ LOC: 14
- ✔ Readable
- ✔ No Code Noises

```fsharp
let rec qsort seq = // 'a list -> 'a list
    match seq with
    | [] -> []
    | pivot :: rest ->
      let left, right = rest |> List.partition (fun x -> x < pivot)
      qsort left @ [pivot] @ qsort right
```

F#

- ✔ LOC: 6
- ✔ Readable
- ✔ No Code Noises

Case 15

# Other Features

Code Quotations, Computational Expressions, …

# History

Early related history since 1970s

— FP and Microsoft Research
— Java Revolution
— Microsoft's Response to Java and .NET Runtime Principles
— FP and .NET
— Formal Verification
— .NET Generics

# History at a glance :: FP and MSR

1. Origins of FP in 1970, 1980 and 1990
2. Microsoft founded in 1975
3. In 1997, as a response to Java, Microsoft started a project that became .NET and C#.
4. At the same time, World of Academic and Industrial FP combined in MSR
5. Researchers engaged with the company through Project 7
6. Result: .NET Generics in 1998 and F# in 2002

# History at a glance :: Java Revolution

1. Java developed in 1991-1995 and released in 1996
2. It was a deep challenge for Microsoft:
    1. Java was OO and modern
    2. Write Once Run Anywhere
    3. Positioned as a Web technology
    4. Used a set of technical devices such as VM and GC
    5. Recognized as a contribution to Academic Computer Science

# History at a glance :: Microsoft's Response to Java

1. Microsoft first was slow to respond
2. For RAD development Microsoft embraced Java, developed Microsoft J++
3. Then changed tactic, started to develop a programming platform
4. Initially called COM+2.0, then Lightning, and eventually .NET

# History at a glance :: Runtime Principles

Founding principles of Runtime were:

1. It would support multiple languages including VB, C++ and Java.
2. Additionally a new language was started under design of Andres Hejlsberg, called COOL
3. Later called C#

1. The project was a Multi-language runtime, rather than a fixed set of languages
2. Project 7 initiated
3. MSR hired:
    1. Andy Gordon (young high-profile researcher in PL theory)
    2. Luca Cardelli (author of one of first ML implementations)
    3. Dr Simon Peyton Jones (a leading Haskell contributor)
    4. Don Syme (that later designed F#)
    5. And many others…

# History at a glance

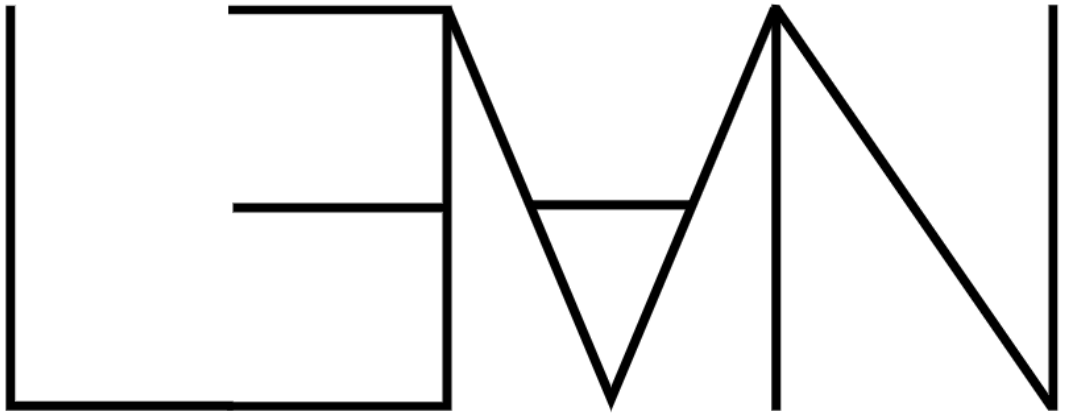
1. Microsoft was entering a phase where it was deeply committed to a multi-language runtime
2. Lightning already had many of core elements of a typical FP language
    1. GC
    2. JIT
    3. ByteCode
3. Interesting!

# History at a glance :: Formal Verification

1. Government and commercial offerings were used to model, formalize and verify aspects of software and hardware designs
2. Functional languages were often used to implement and script these systems
3. Significant increase in formal verification investment (FDIV 1994)
4. Proof assistants
5. Strongly-typed FP languages and compilers saw an ongoing trickle of interest, adoption and use
6. A response to mainstream OO was to integrate FP features in them
    1. Parametric Polymorphism
    2. Discriminated Unions
    3. First-class function values
7. GJ heavily influenced C#, Scala and F# later.

# Microsoft Research LEAN (https://leanprover.github.io)

ABOUT DOCUMENTATION DOWNLOAD PUBLICATIONS PEOPLE




Microsoft Research

# dafny

Microsoft Research

Is this program correct?

```
1 function Fibonacci(n: int): int
2   decreases n
3 {
4   if n < 2 then n else Fibonacci(n+2) + Fibonacci(n+1)
5 }
6
```

tutorial

home video permalink

'►' shortcut: Alt+B

samples

Hello
Fibonacci
Ackermann
Add
Mul
CountToN
CountToAndReturnN
Cube
zb
z

## about Dafny - A language and program verifier for functional correctness

Dafny is a programming language with imperative and functional features, as well as specification constructs for describing intended behavior. The Dafny verifier checks that programs live up to their specifications.



# Introduction

F* (pronounced F star) is a general-purpose functional programming language with effects aimed at program verification. It puts together the automation of an SMT-backed deductive verification tool with the expressive power of a proof assistant based on dependent types. After verification, F* programs can be extracted to efficient OCaml, F#, C, WASM, or ASM code. This enables verifying the functional correctness and security of realistic applications. The main ongoing use case of F* is building a verified, drop-in replacement for the whole HTTPS stack in Project Everest. This includes verified implementations of TLS 1.2 and 1.3 and of the underlying cryptographic primitives.

F*'s type system includes dependent types, monadic effects, refinement types, and a weakest precondition calculus. Together, these features allow expressing precise and compact specifications for programs. The F* type-checker aims to prove that programs meet their specifications using a combination of SMT solving and interactive proofs.

F* is written entirely in F*, and bootstraps in OCaml and F#. It is open source and under active development on GitHub. A detailed description of the current F* variant is available in a series of POPL and ICFP papers (2016, 2017, 2018, and 2019). You can learn more about F* by following the online tutorial and reading our papers. Materials from recent talks are also available below. And to keep up to date with the latest news on F* you can read our blog and join our mailing list and public chat.

# Project 7 and .NET Generics

1. Project 7 kicked off, recommended following languages: Eiffel, Mercury, Standard ML, OCaml, Scheme, Alice and Haskell
2. Bias was clear: 6 of 7 Strongly-typed, 3 of 7 was firmly “Strongly-typed FP”
3. Problems: Build and Maintain was high.
4. Project 7 had technical impacts: MSIL Tail
5. Project 7 also had impact on “language interop”
6. Syme et al. proposed internal “Proposed Extensions to COM+ VOS”

# Tail Call Optimization

Sample.fs

```
let apply f x = f x // ('a → 'b) → 'a → 'b

```

IL.il

```
Main.apply:
IL_0000:  ldarg.0
IL_0001:  ldarg.1
IL_0002:  tail.
IL_0004:  callvirt   Microsoft.FSharp.Core.FSharpFunc<,>.Invoke
IL_0009:  ret
```

# Linq

## Haskell influenced Linq

```
from x in xs
where x > 0
select x+1
```

```
[ x+1 | x ← xs, x > 0]
```

# Project 7 and .NET Generics

1. 5 technical features proposed but two of them was more serious:
    1. Functions as first-class values (generalized delegates)
    2. Enhanced Parametric Polymorphism.
2. Enormous dedication to deliver .NET Generics in C# and .NET by Don Syme, Kennedy and Russo
3. Released in 2005 (.NET 2/0 codenamed “Whidbey”)

# Project 7 and .NET Generics

MSR Cambridge
Programming Principles & Tools

## Generics for C# and .NET

TechFest 2002

soft

### What is it?

Code that works for values of many types

- In C#: generic classes, interfaces, structs, methods, delegates
- Think C++ templates without the complexity or code bloat
- e.g. Type-safe collection classes

```
// Generic interface
interface ISet<T> : ICollection<T> {
   void Add(T item);
   void Remove(T item);
   bool Contains(T item);
}

// Generic class
public class ArraySet<T> : ISet<T> {
   private T[] items;
   private int nitems;
   public ArraySet() {
     nitems = 0; items = new T[10];
   } ...
}
...
// Generic methods
static void Reverse<T>(T[] arr)
static void Sort<T>(T[] arr, IComparer<T> cmp)

// Instantiation at a primitive type
static int Sum(ISet<int> s) {
  int sum = 0;
```

### When can I use it?

Current plan is to ship in V2 of C# and the CLR. MSR implementation is already integrated into the Whidbey (V2) codebase.

### Why should I use it?

It's better than the alternative – using Object in C#:

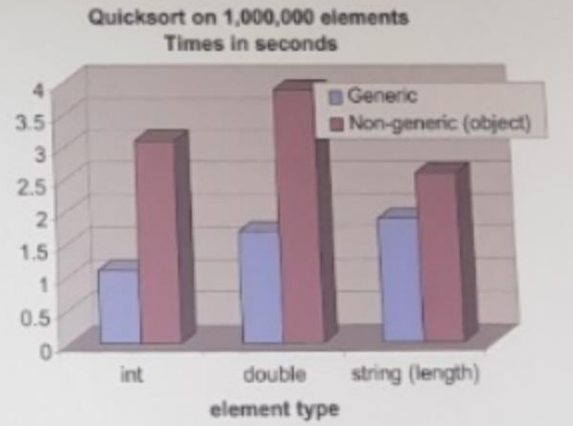
- *Expressivity*
  Invariants clearly expressed in source.
  Without generics:
  `int Sum(ISet s)`
- *Clarity*
  No ugly casts.
  Without generics:
  `foreach (object obj in s) { sum += (int) obj; }`
- *Safety*
  Bugs caught at compile-time.
  Without generics:
  InvalidCastException!
- *Efficiency*

### What about performance?

Generic types are managed automatically by the CLR, and are optimized for performance:

- Just-in-time code specialization reduces start-up times
- Code sharing avoids bloat (e.g. code for List<string> shares with List<object>)
- Code specialization avoids costly boxing (e.g. List<int> and List<float>)

*Example:*



# F# Language

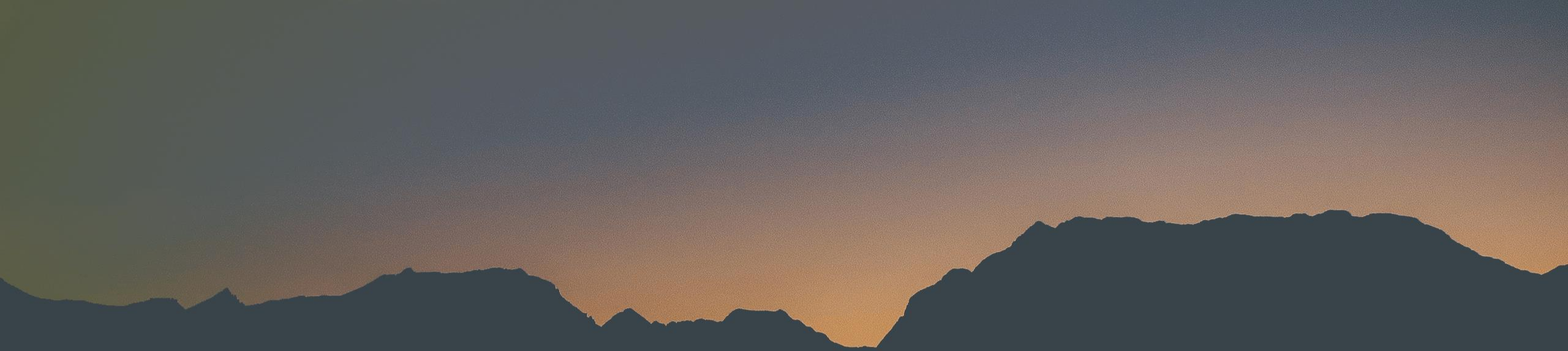
The decision to create F#

# The decision to create F#

1. Initially in 2000, DSyme attempted an implementation of Haskell for .NET
2. Experience was partly successful, but advices from Simon PJ led Syme to believe the project will not be successful.
    1. Haskell couldn't fully interop with .NET
    2. Type Systems were a lot different
    3. HKT, Type Classes, etc…
3. Work on Haskell.NET stopped
4. Discussion about bringing OCaml to .NET Started
5. Don Syme decided to move ahead with a Caml.NET in December 2001
6. Later rebranded to F#

# F# and .NET Interop

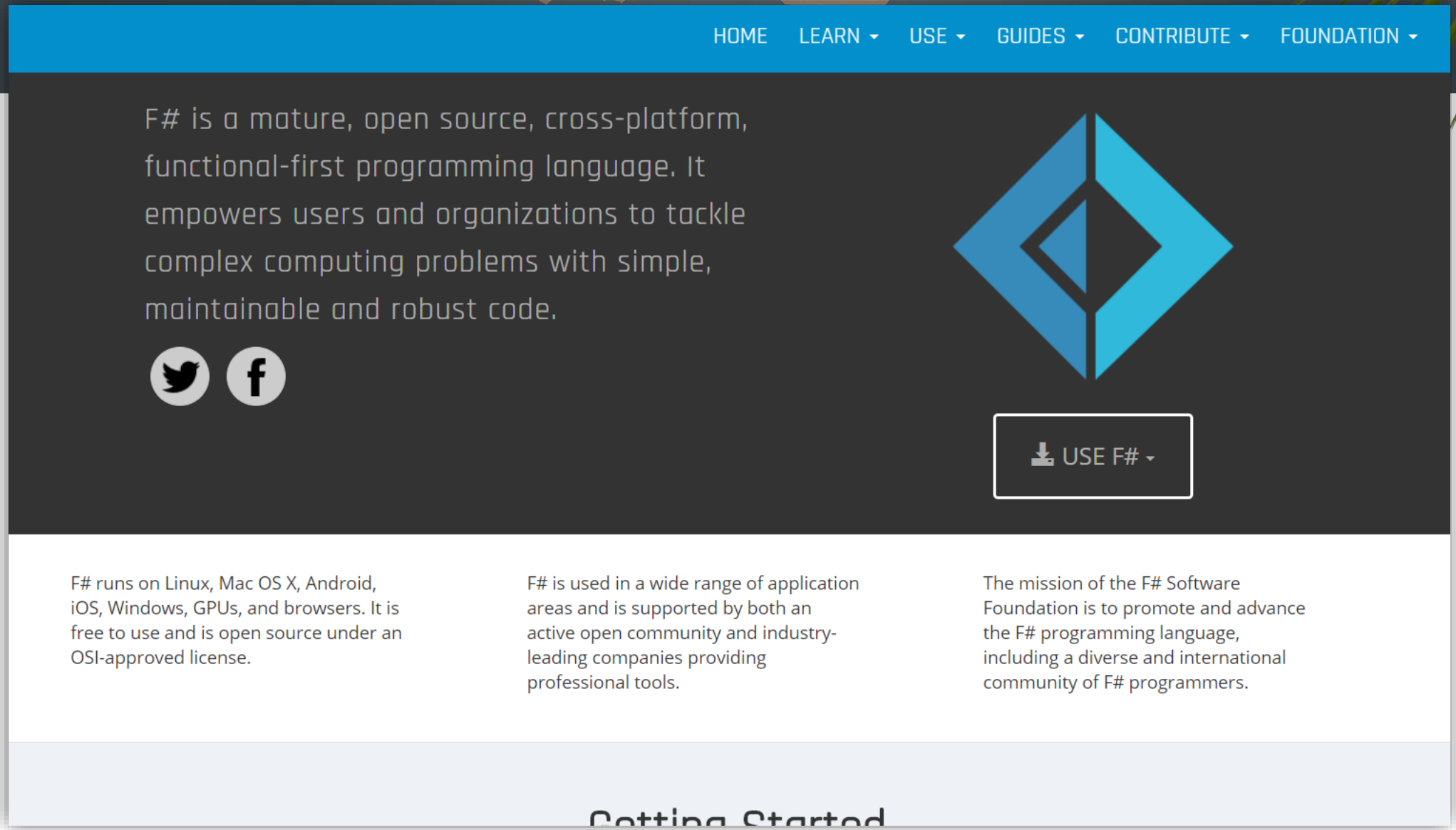
1. At the end:
    1. .NET types are F# Types
    2. .NET Values are F# Values
    3. .NET Exceptions are F# Exceptions
    4. .NET Threads are F# Threads
    5. .NET Strings are F# Strings
2. Cool, No Marshalling, No type conversion, 2-way interop
3. F# Stuck with whatever .NET does



# Wanna know more?

همکاران سیستم
SYSTEM GROUP

# Fsharp.org


HOME
LEARN
USE
GUIDES
CONTRIBUTE
FOUNDATION
F# is a mature, open source, cross-platform, functional-first programming language. It empowers users and organizations to tackle complex computing problems with simple, maintainable and robust code.
USE F#
F# runs on Linux, Mac OS X, Android, iOS, Windows, GPUs, and browsers. It is free to use and is open source under an OSI-approved license.
F# is used in a wide range of application areas and is supported by both an active open community and industry-leading companies providing professional tools.
The mission of the F# Software Foundation is to promote and advance the F# programming language, including a diverse and international community of F# programmers.
Getting Started

# F# for fun and profit



## 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 mainstream commercial business software. On the way, I hope to open your mind to the joys of functional programming -- it really is fun!

If you have never heard of F#, it is a general purpose functional/hybrid programming language which is great for tackling almost any kind of software challenge. F# is free and open source, and runs on Linux, Mac, Windows and more. Find out more at the F# Foundation.

### 💡 Learn to think functionally

"Thinking functionally" is critical to getting the most out of F#, so I will spend a lot of time on getting the basics down, and I will generally avoid too much discussion of the hybrid and OO features.

### $ Useful examples

The site will mostly focus on mainstream business problems, such as domain driven design, website development, data processing, business rules, and so on. In the examples I will try to use business concepts such as Customer, Product, and Order, rather than overly academic ones.

### 💀 Don't be scared

F# can look very intimidating if you look at complex code without any background. In the beginning I will keep it very simple, and I have tried to anticipate the questions that a newcomer to functional

### ❤ Have fun!

Many people claim that learning to think functionally will "blow your mind". Well, it's true! Learning a completely new paradigm is exciting and stimulating. You may fall in love with programming

GET THE BOOKS

Reading offline? Download the ebook of this site.

GREATEST HITS

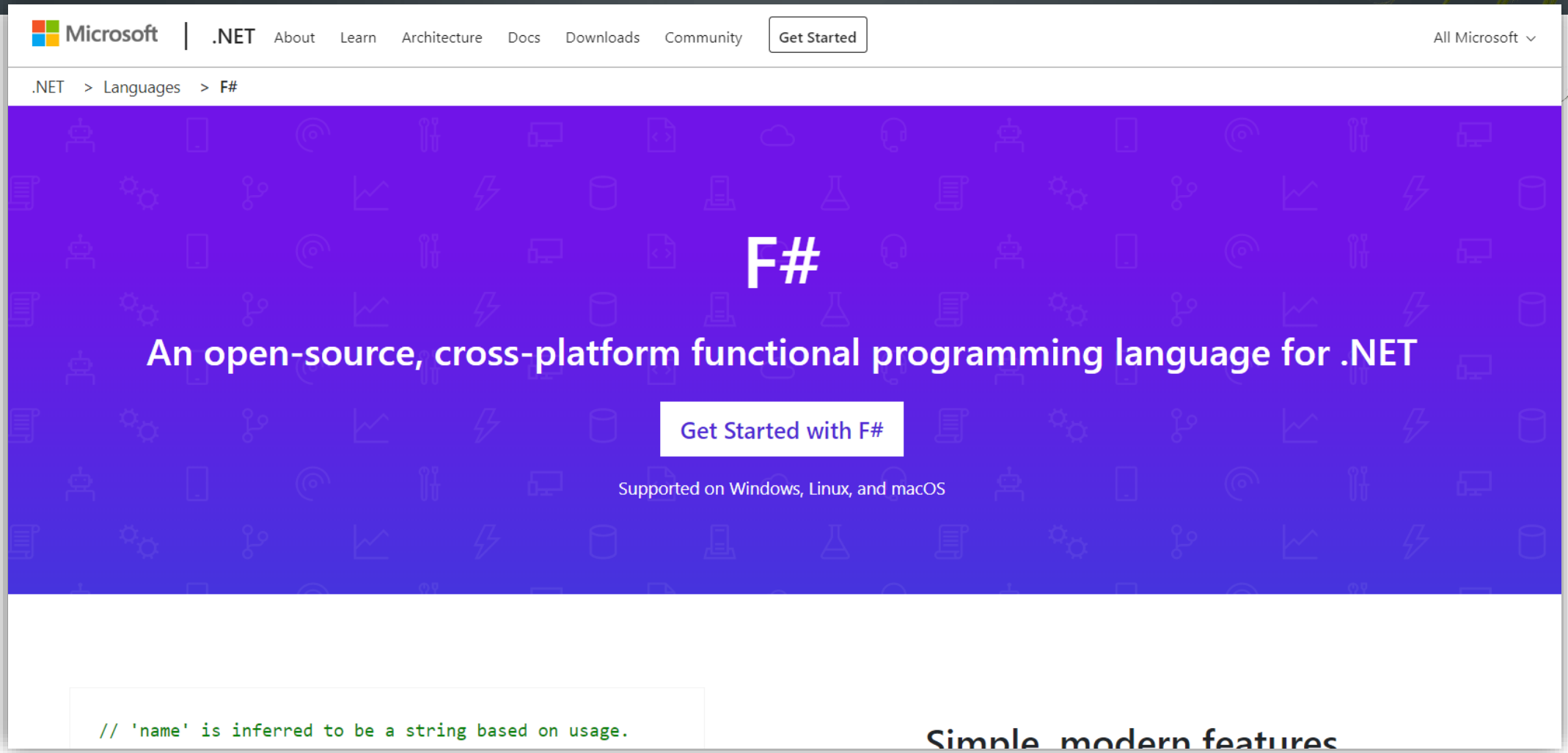
New here? Try one of these:
Explore this site
Why use F#?
Tips for learning F#
Troubleshooting F#
Functional design patterns (talk)
Thinking Functionally
Domain modeling with F# (talk)
Designing with Types
Railway Oriented Programming (talk) and the original post
Thirteen ways of looking at a turtle
Understanding Parser Combinators
Roman Numerals Kata
Choosing properties for property-based testing

RECENT POSTS

Why F# is the best enterprise language
Serializing your domain model

# Microsoft .NET Website for F#

https://dotnet.microsoft.com/languages/fsharp



# References

همکاران سیستم
SYSTEM GROUP

# References


Microsoft Research Podcast, Functional Programming Languages and the Pursuit of Laziness with Dr. Simon Peyton Jones

The Early History of F# (HOPL IV - first draft), fsharp.org/history

Why learn F#? And what is the advantage? By Aryan Ebrahimpour https://virgool.io/@0xaryan/learn-fsharp-for-fun-and-profit-ox5rxavm9tq3

# Thank you!

If you liked my presentation, please consider starting this repo:

☆ github.com/0xaryan/Presentation