

# .NET Conf

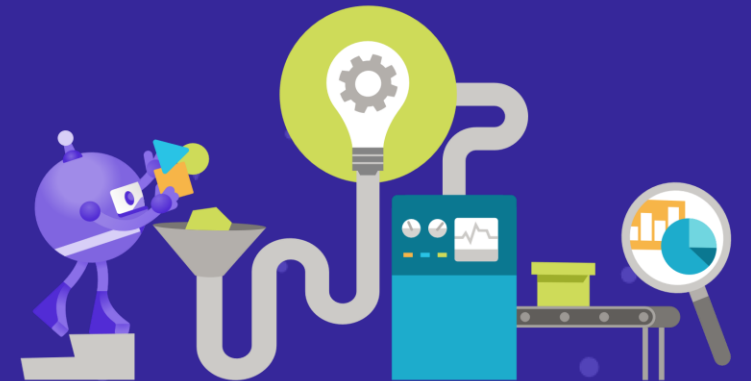
Focus on F#

July 29, 2021

[focus.dotnetconf.net](https://focus.dotnetconf.net)

# Don teaches Guido some F#

Don Syme & Guido van Rossum



<https://github.com/dsyne/guido-learns-fsharp>

# Setup

---

.NET SDK

<https://dotnet.microsoft.com/download>

.NET

.NET 5.0 (recommended)

Current ⓘ

.NET is a free, cross-platform, open-source developer platform for building many different types of applications.

Download .NET SDK x64 ⓘ

Download .NET Runtime ⓘ

[All .NET downloads](#)

# Setup

---

Node.js and npm  
<https://nodejs.org/en/>

Node.js® is a JavaScript runtime built on Chrome's V8 JavaScript engine.

Download for Windows (x64)

**14.17.0 LTS**

Recommended For Most Users

**16.3.0 Current**

Latest Features

[Other Downloads](#) | [Changelog](#) | [API Docs](#)

[Other Downloads](#) | [Changelog](#) | [API Docs](#)

Or have a look at the [Long Term Support \(LTS\) schedule](#).

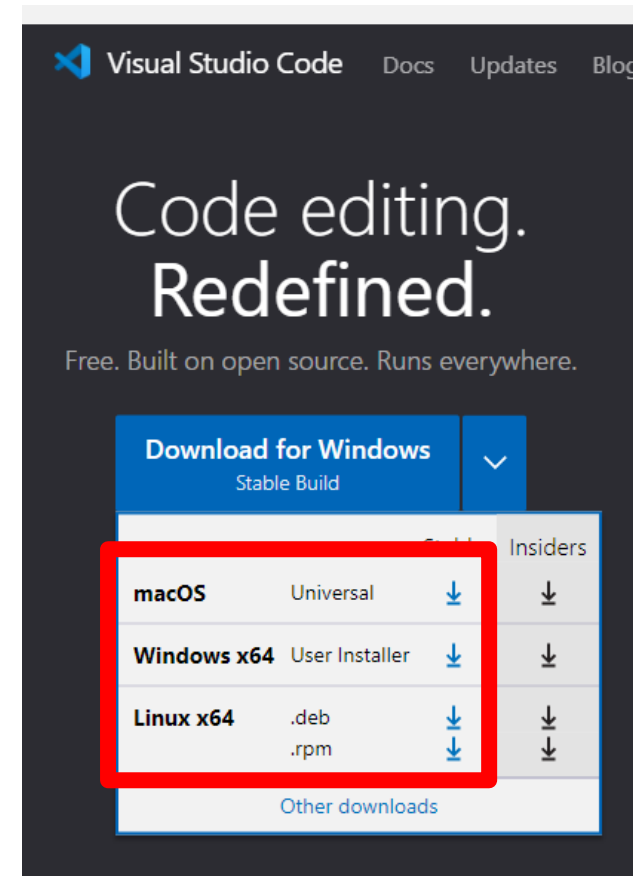
# Setup

---

VS Code

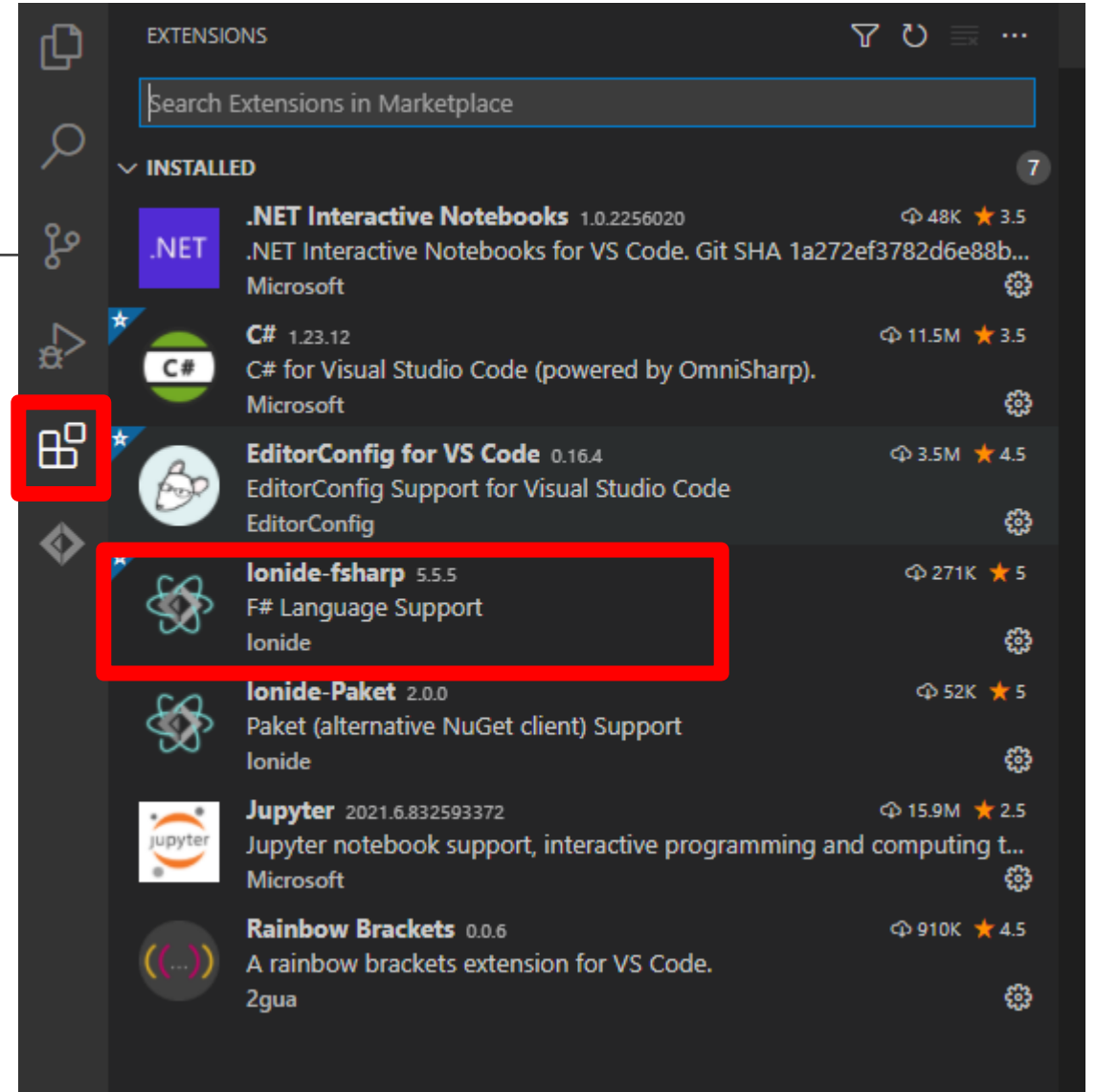
<https://code.visualstudio.com/>

**ON LINUX, DON'T USE SNAP**



# Setup - VS Code Extensions

- Ionide



Now do Tasks 1.\*



# Language Lessons from Tasks 1.\*

---

You've learned or seen all of the following:

1. `let let let let` (function definitions and values)
2. `match` for pattern matching and strings
3. dot notation. F# supports object programming
4. strings, including interpolated strings
5. F# is strongly typed. The IDE knew your types and checks on the fly
6. F# knows how symbols resolve: rename, goto-definition etc.

F# achieves three key things

Succinctness (like Python)

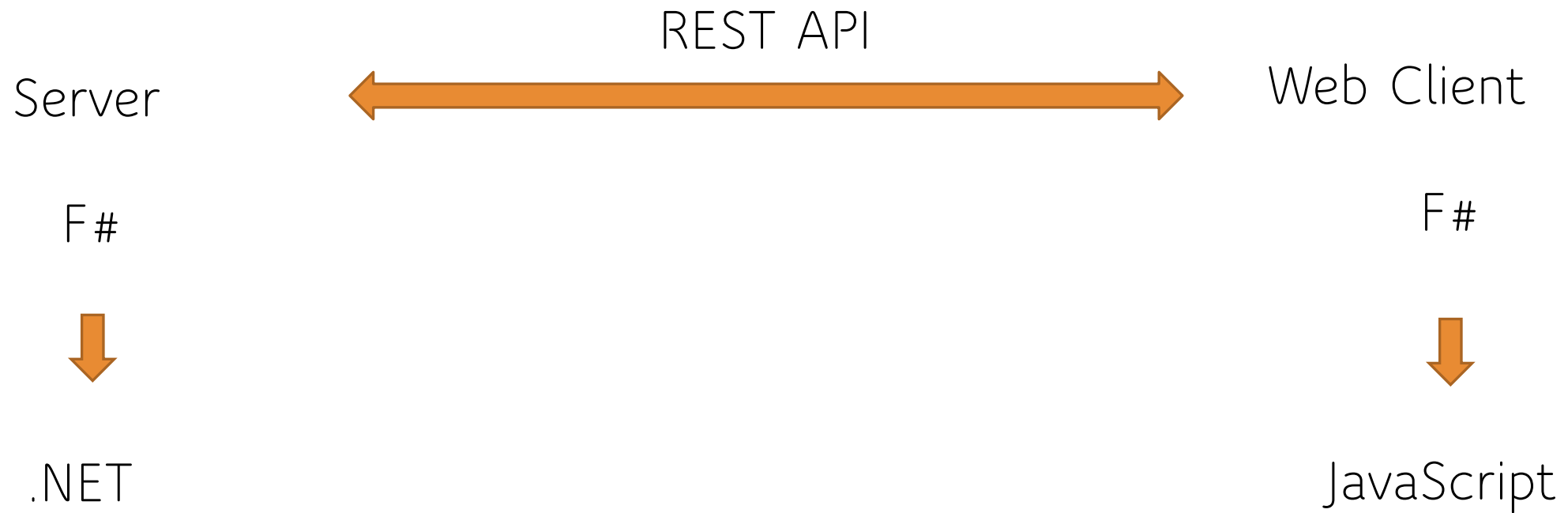
Performance (like C#/Java)

Robustness (like strongly typed functional programming)

(also interop, support, cross-platform,  
community, reach etc. )

# This is a typical Client/Server App

---

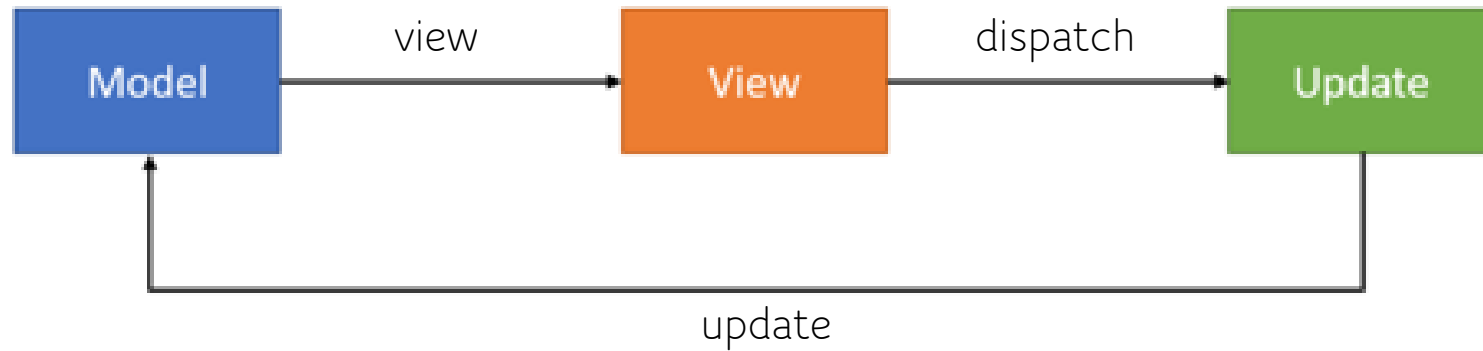


# The Web Client uses “Model View Update” c.f. React

---

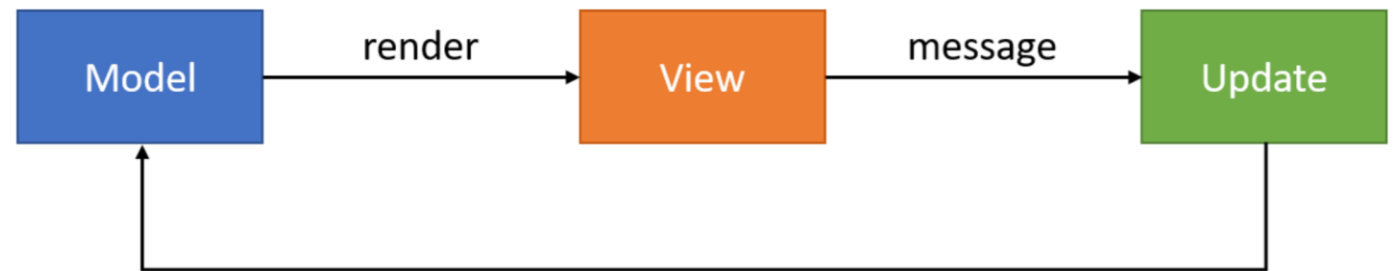
REST API

Web Client

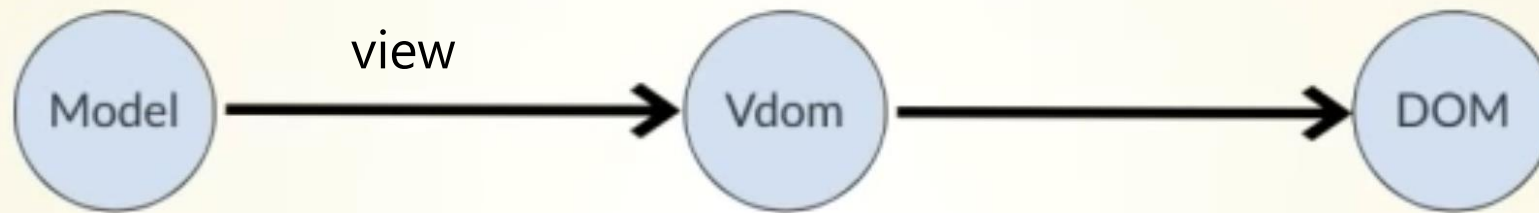


# Model View Update

---



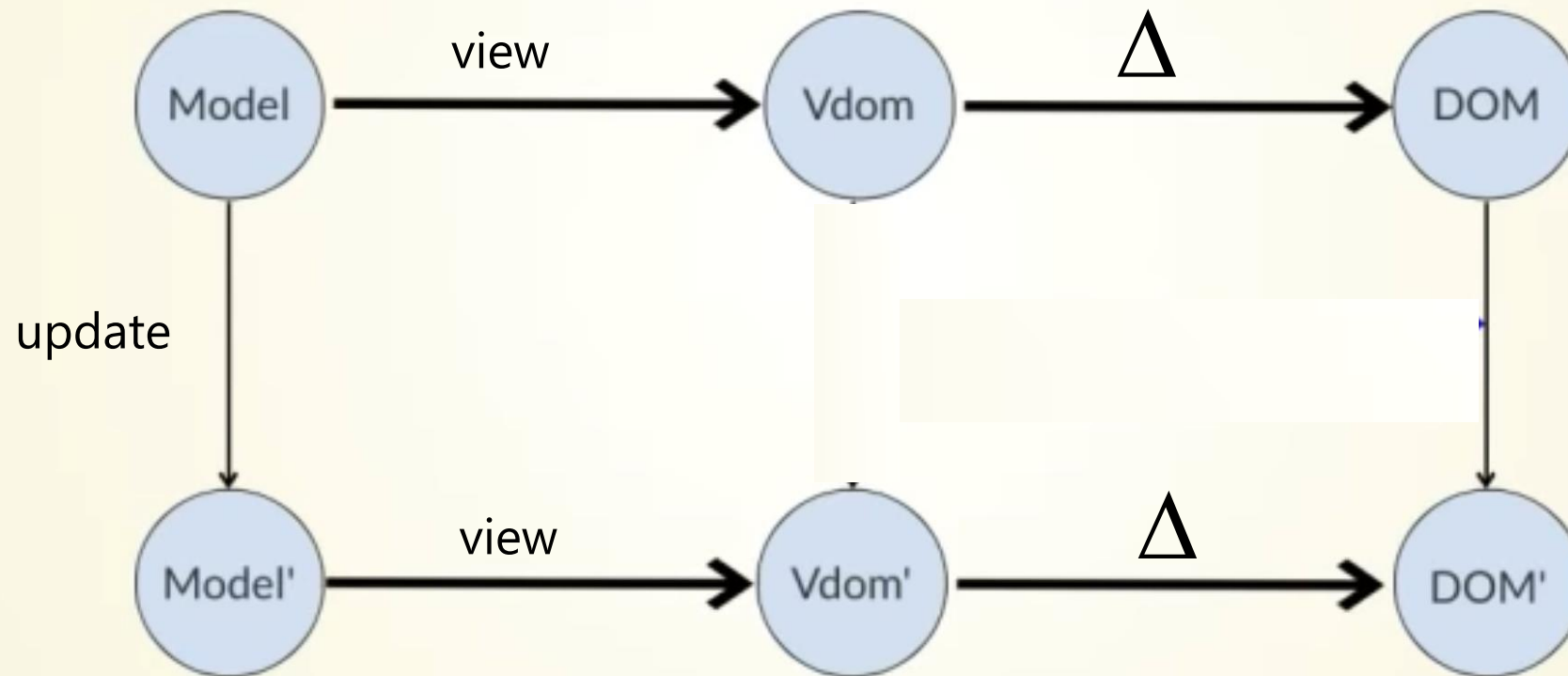
# Your UI as an Incremental Computation



Sept 27-28, 2018  
thestrangeloop.com



# Your UI as an Incremental Computation



Sept 27-28, 2018  
thestrangeloop.com



Now do Tasks 2.\*



# Lessons from Tasks 2.\*

---

You've learned or seen all of the following:

1. In this app, display views are functional data
2. The view is recalculated and applied to the actual DOM
3. The functional view uses computed list expressions,  
a super-powerful form of list comprehensions
4. tuples and helper functions

Now do Tasks 3.\*

# Lessons from Tasks 3.\*

---

You've learned or seen all of the following:

1. “let” and “type” all day long
2. pipelining with `|>`
3. record types - cheap and cheerful functional data
4. async programming for server requests
5. strongly typed string interpolation

Now do Tasks 4.\*

# Lessons from Tasks 4.\*

---

You've learned or seen all of the following:

1. discriminated union types for messages in a web UI
2. pattern matching and completeness checks
3. functional objects with method/property members
4. dispatching a new message in this UI architecture

Now do Tasks 5.\*

# Lessons from Tasks 5.\*

---

You've learned or seen all of the following:

1. evaluating code in the F# REPL
2. referencing a package with a strong version
3. using FSharp.Data with a sample from a REST API
4. exploring the data in preparation for adding a backend service

Further discussion points! 😊

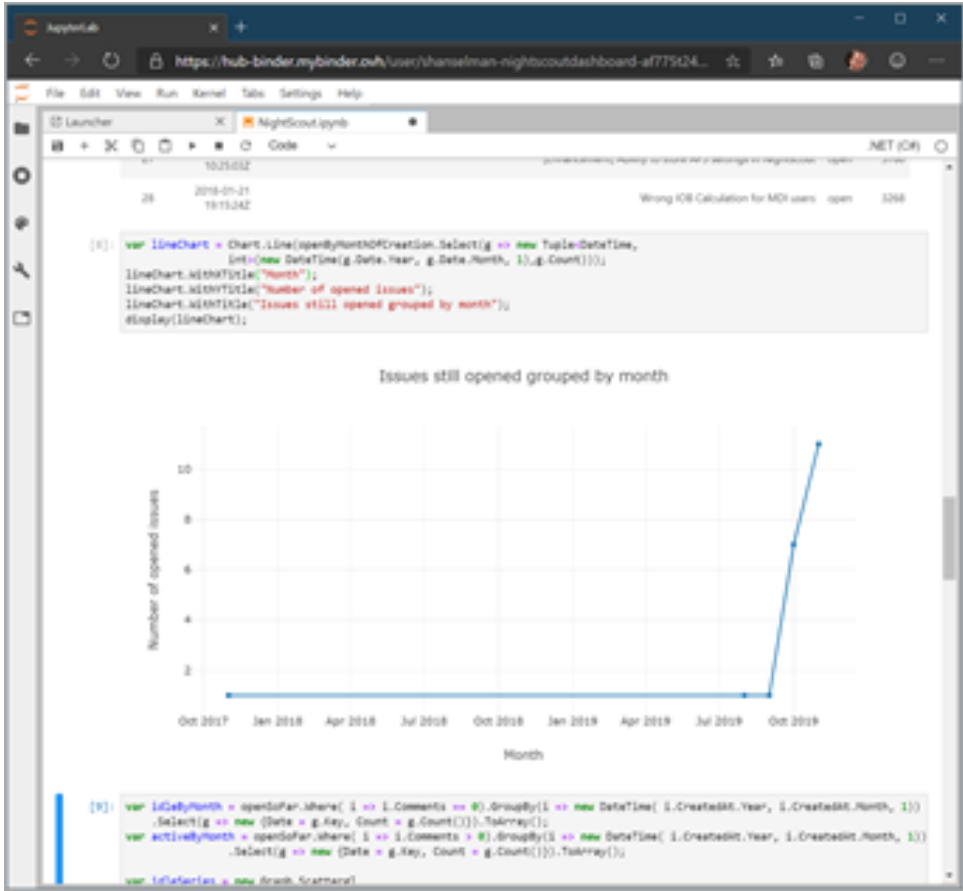


# Other F# Language Discussion Points

---

1. `let mutable`
2. `box/unbox`
3. `reflection`
4. `performance`
5. `namespace` and `modules` for code organization
6. F# is not lazy by default (see 'lazy' expressions)
7. Bi-directional .NET and Javascript interop
8. Fable now includes an F# to Python transpiler. If Python gives us metadata, we will interop smoothly with Python

# F# in .NET Interactive/Jupyter Notebooks



# Python GCN

---

```
def train(epoch):
    t = time.time()
    model.train()
    optimizer.zero_grad()
    output = model(features, adj)
    loss_train = F.nll_loss(output[idx_train], labels[idx_train])
    acc_train = accuracy(output[idx_train], labels[idx_train])
    loss_train.backward()
    optimizer.step()

    if not args.fastmode:
        # Evaluate validation set performance separately,
        # deactivates dropout during validation run.
        model.eval()
        output = model(features, adj)

    loss_val = F.nll_loss(output[idx_val], labels[idx_val])
    acc_val = accuracy(output[idx_val], labels[idx_val])
```

# F# GCN

```
let train epoch =  
    let t = DateTime.Now  
    model.Module.Train()  
    optimizer.zero_grad()  
    let output = model.forward(features)  
    let loss_train = nll_loss(output.[ idx_train], labels.[idx_train])  
    let acc_train = accuracy(output.[idx_train], labels.[idx_train])  
    loss_train.backward()  
    optimizer.step()  
  
    let output =  
        if fastmode then  
            output  
        else  
            model.Module.Eval()  
            model.forward(features)  
  
    let loss_val = nll_loss(output.[idx_val], labels.[idx_val])  
    let acc_val = accuracy(output.[idx_val], labels.[idx_val])
```

# Thank you Guido!

Ask your questions live on Twitter [#dotNETConf](#)



# Thanks for joining!

Ask your questions live on Twitter #dotNETConf



# Other slides



[TRY ONLINE](#)

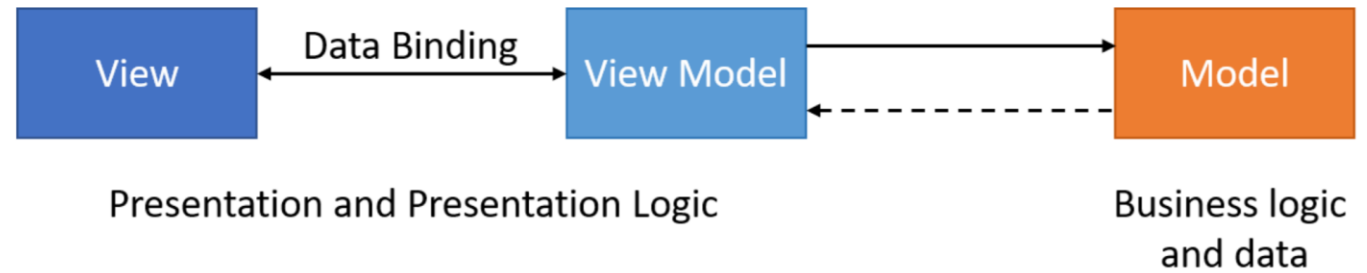
[GET STARTED](#)

Fable is a compiler that brings F# into the JavaScript ecosystem

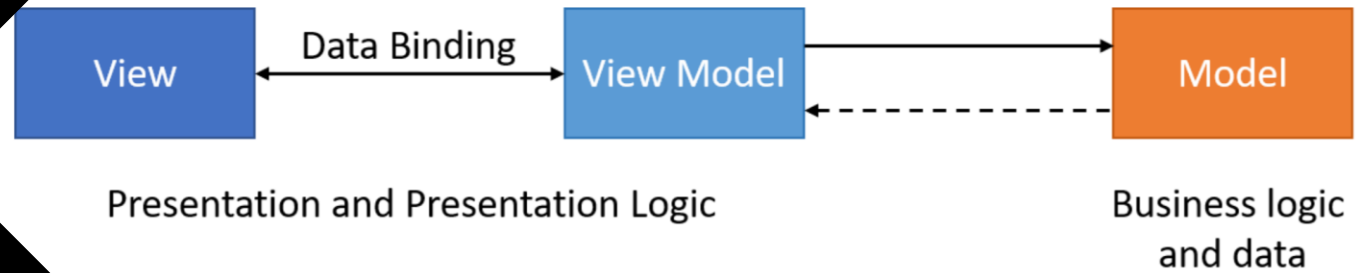


# Model View View Model

---



Model View  
View Model



# MVU

---

*Calculate your views with functional programming*

# A Simple MVU App

```
type Model =  
  { ... }
```

The model from which  
the view is generated

```
type Msg =  
  | ...  
  | ...
```

Messages which cause  
updates to the model

```
let init() =  
  { ... }
```

Initial state

```
let update msg model =  
  match msg with  
  | ... -> { model with ... }  
  | ... -> { model with ... }
```

Update the model

```
let view model dispatch =  
  if model.IsPressed then  
    ... dispatch msg ...  
  else  
    ... dispatch msg ...
```

Compute the view

# A Simple MVU App

```
type Model =  
  { IsPressed: bool }  
  
type Msg =  
  | Pressed  
  
let init() =  
  { IsPressed=false }  
  
let update (msg: Msg) (model: Model) =  
  match msg with  
  | Pressed -> { model with IsPressed = true }  
  
let view (model: Model) dispatch =  
  if model.IsPressed then  
    View.Label(text="I was pressed!")  
  else  
    View.Button(text="Press Me!", command=(fun () -> dispatch Pressed))
```

The model from which the view is generated

Messages which cause updates to the model

Initial state

Update the model

Compute the view

# F# get started

```
dotnet new -lang F#
```

```
dotnet build
```

F# tools are part of the  
.NET SDK, available  
everywhere



# F# for the backend

```
dotnet new -i "giraffe-template::*"
```

```
dotnet giraffe
```

High perf, functional  
server-side  
programming



**GIRAFFE**

A functional ASP.NET  
Core micro web  
framework for  
building rich web  
applications.

[github.com/giraffe-fsharp/Giraffe](https://github.com/giraffe-fsharp/Giraffe)

# F# for the frontend (JS)



```
dotnet new -i "Fable.Template::*"
```

```
dotnet new fable  
npm install  
npm start
```

You can use F# as a  
JavaScript language

NOTE: See also WebSharper