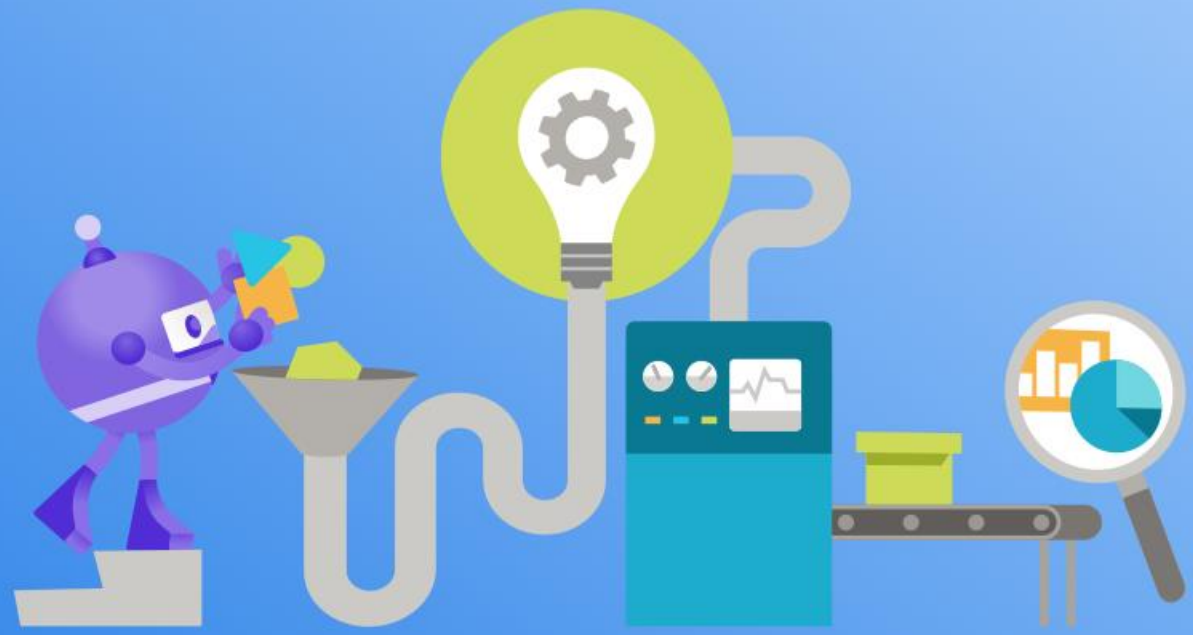


.NET Conf

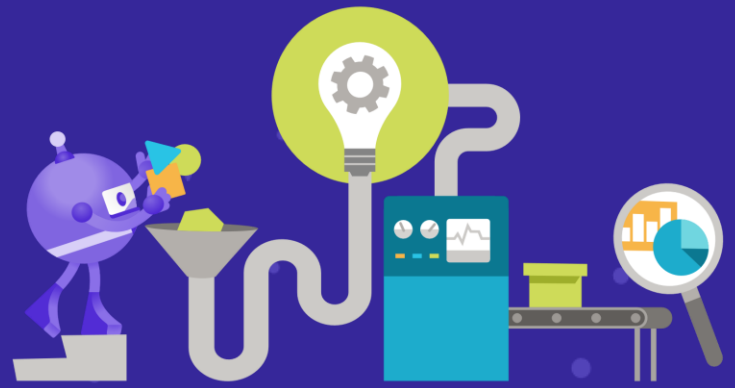
Focus on F#

July 29, 2021

focus.dotnetconf.net

Don teaches Guido F#

Don Syme & Guido van Rossum



https://github.com/dsyme/guido-learns-fsharp

The aim of F# is three things

Succinctness (like Python)
Performance (like C#/Java)
Robustness   (like strongly typed functional programming)

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

Setup

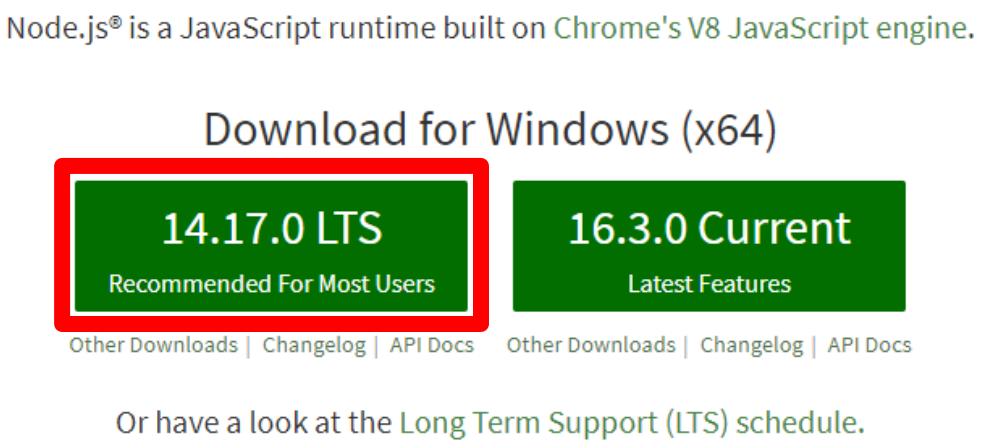
.NET SDK

https://dotnet.microsoft.com/download

Setup

Node.js and npm

https://nodejs.org/en/

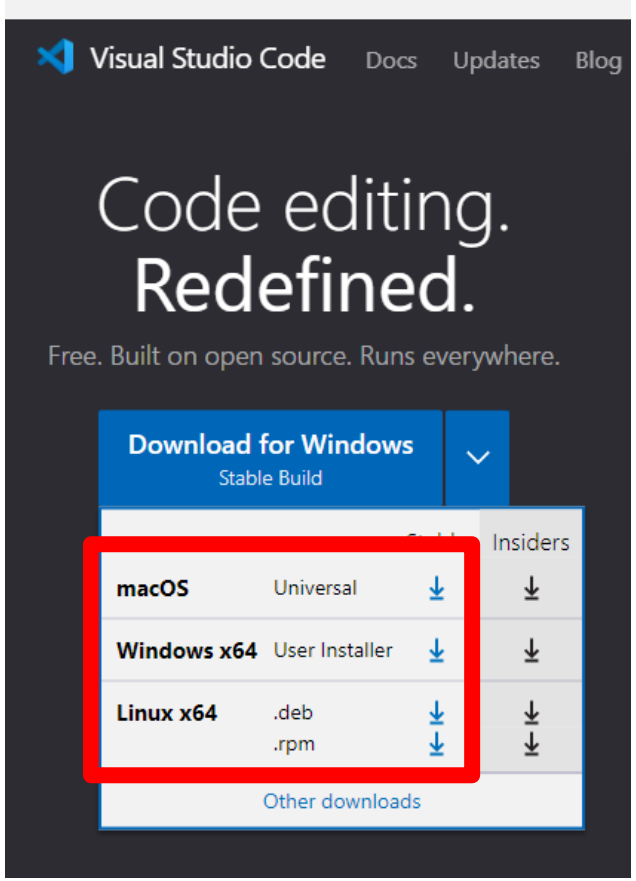


Setup

VS Code

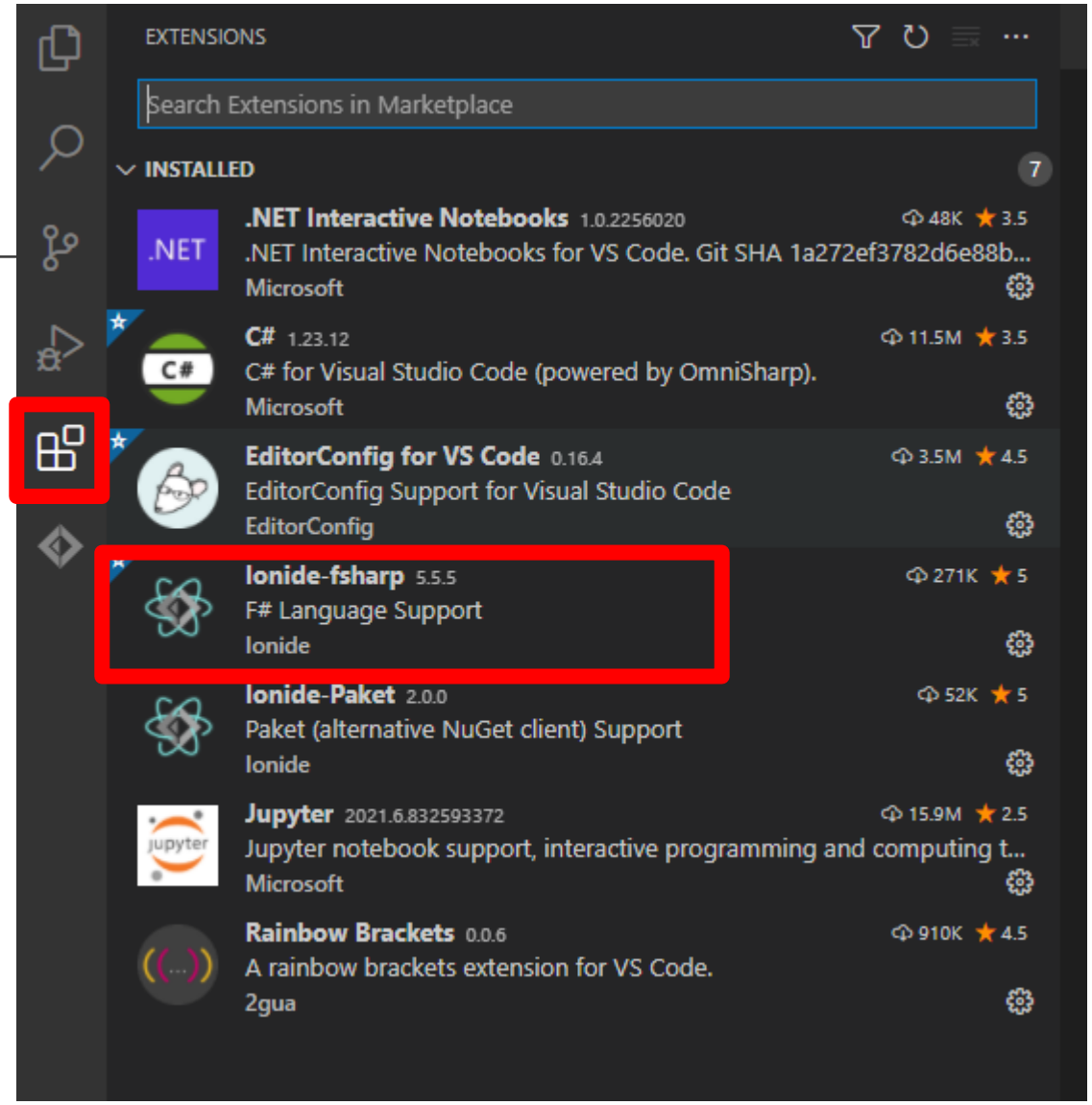
https://code.visualstudio.com/

ON LINUX, DON'T USE SNAP

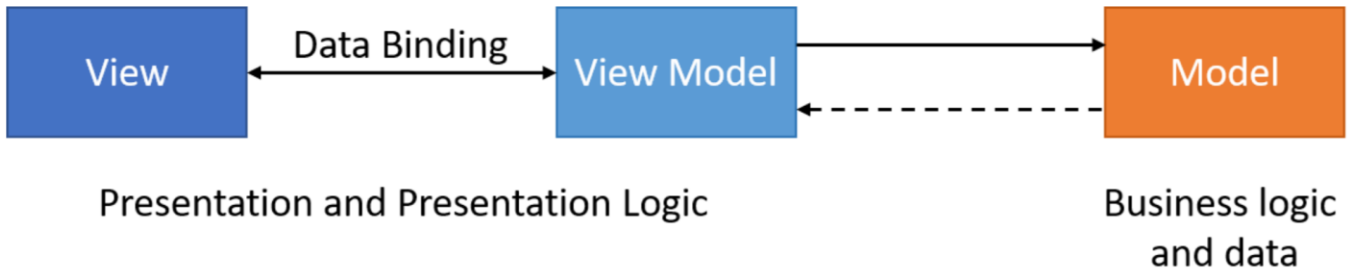


Setup - VS Code Extensions

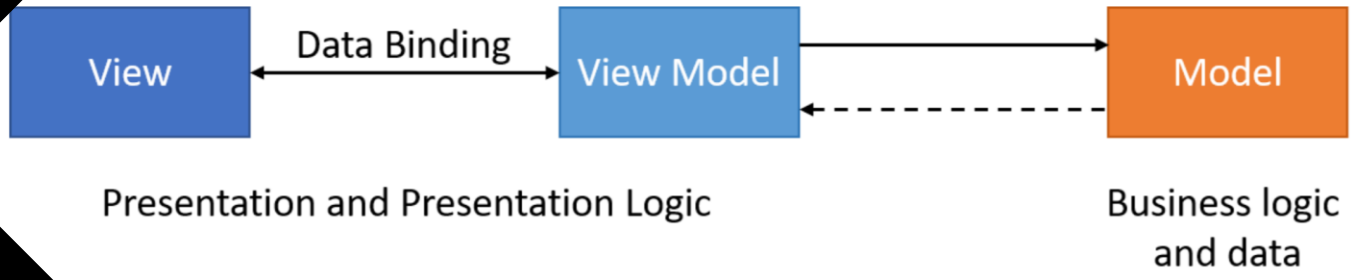
- Ionide



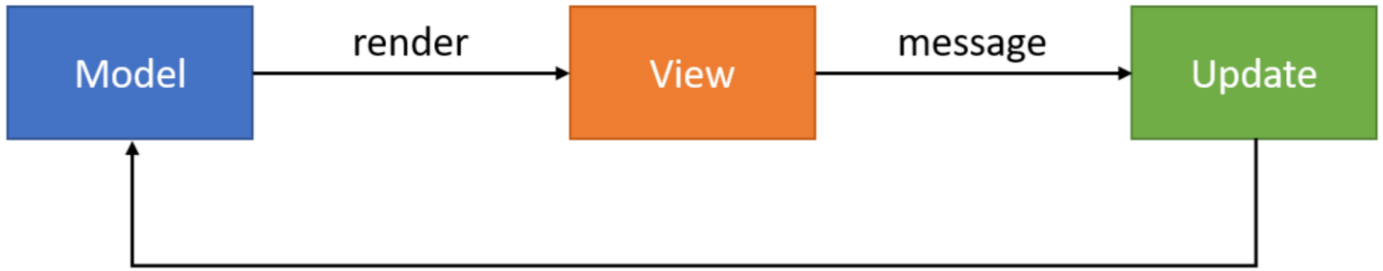
Model View View Model



Model Vie
View Mode



Model View Update



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.

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

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

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

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

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
9. If Python gives us metadata, we will interop smoothly with Python

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])
```

https://www.linkedin.com/pulse/graph-convolutional-network-model-strongly-typed-faisal-waris-phd/

Thanks for joining!

Ask your questions live on Twitter #dotNETConf

Other slides


ABLE
TRY ONLINE
GET STARTED
Fable is a compiler that brings F# into the JavaScript ecosystem

F# get started

```
dotnet new -lang F#

dotnet build
```

.NET

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

F# for the backend

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

dotnet giraffe
```

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

High perf, functional server-side programming

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: WebSharper also includes an excellent JavaScript compiler for F#