F#, Immutability, and the MVU Pattern

Women Who Code

July 2021

Wallace Kelly

- Ph.D., 1997
- Corporate R&D / Contractor / Trainer / University / Startups
- Seven patents
- BASIC → FORTRAN → C → C++ → PHP → Java → C# → Python → F#

Outline

- ☐ About the presenter
- ☐ Start downloads of the Docker images
- ☐ Evolution of Software Paradigms
- ☐ F# overview (w/ code demos)
- ☐ Immutability (w/ code demos)
- ☐ Model-View-Update (w/ code demos)
- ☐ Practice Exercises

Objectives

At the end of this workshop,...

- ☐ Motivated to adopt functional techniques.
- ☐ Describe the MVU pattern.
- ☐ Have experience editing F# code.

Two types of slides

Big Idea

Techniques of functional programming are defining the next era in software languages.

12

Concepts

Practice

F# Immutability

https://try.fsharp.org

```
let x = 3
x = x + 1
printfn "%d" x
```

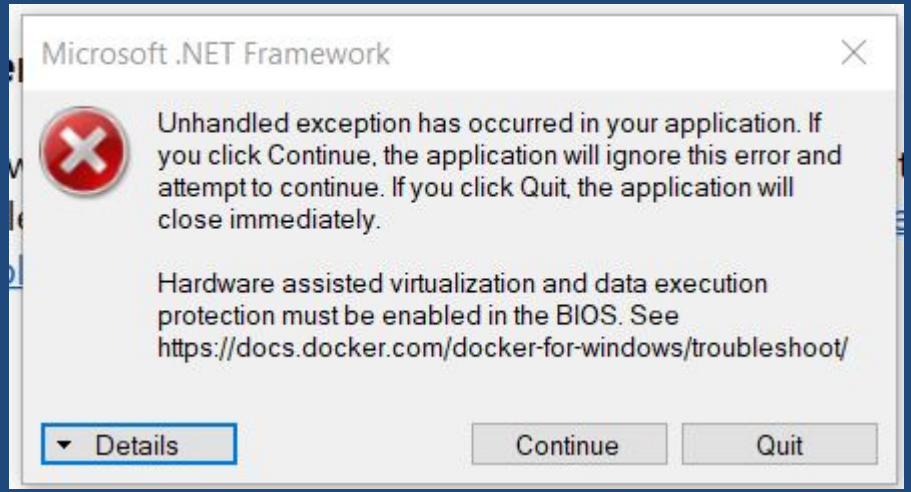
Prerequisite Software

Start both of these:

- **Visual Studio Code**
- **Docker Desktop**

Windows Installation Issues

- **Docker Desktop requires:**
  - Hardware virtualization enabled in BIOS
  - WSL2 is now a separate install



Install the required VS Code extension

In Visual Studio Code

1. Open the **Extensions** activity bar (Ctrl-Shift-X).
2. Search for “**Remote - Containers**”.
3. Press the **Install** button.

Or, on a command line

```
code --install-extension ms-vscode-remote.remote-containers
```

Start the download of Docker images

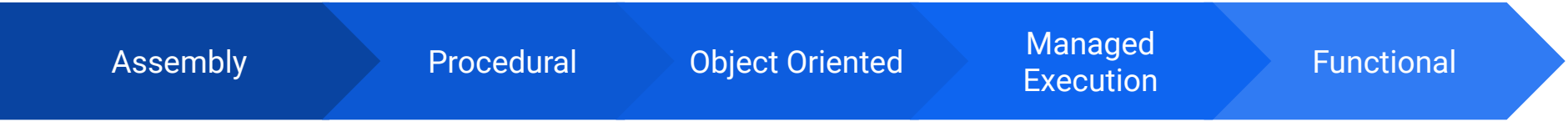
In Visual Studio Code

1. Open the Command Palette (**Ctrl-Shift-P**).
2. Type "**Container Volume**".
3. Select "**Clone Repository in Container Volume**".
4. Enter "**WWCode-SV/fsharp-workshop**"
5. Select the **main** branch.
6. Wait for it...
7. Click "Starting Dev Container (**show log**)"

Big Idea

Techniques of functional programming are defining the next era in software languages.

History of coding paradigms



<u>CPU instruction set</u>

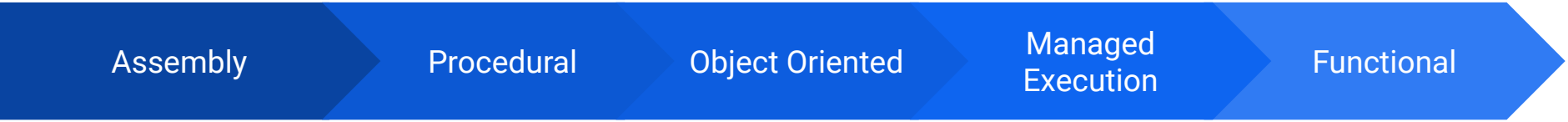
Copy bytes

Perform operation

Jump to new location

Conditionals

History of coding paradigms



<u>C / FORTRAN</u>

Compiler

Interpreter

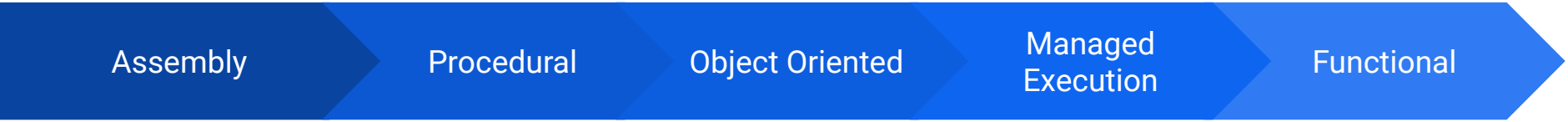
Variable

Data types

for loops

If / then / else

History of coding paradigms



<u>C++ / Objective C</u>

Classes

Objects

Encapsulation

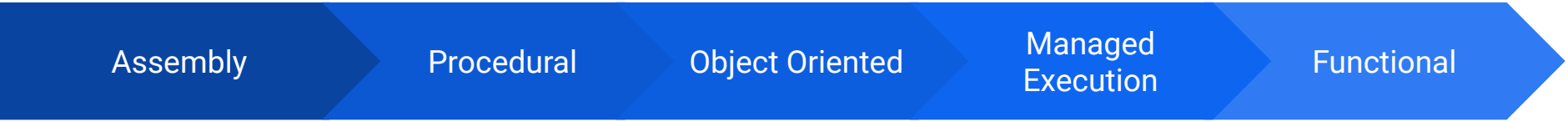
Inheritance

Polymorphism

Interfaces

Templates

History of coding paradigms



<u>Java / .NET</u>

Bytecode / IL

Garbage collection

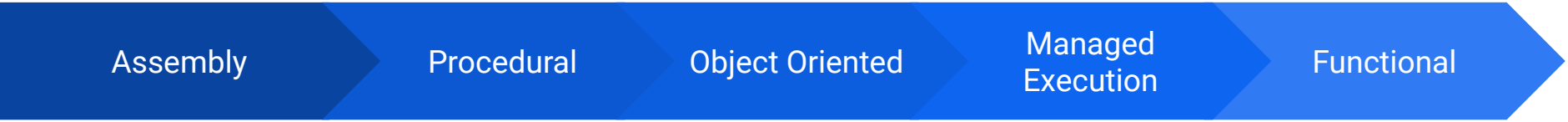
Bounds checking

Reflection

JIT compilation

Generics

History of coding paradigms



<u>Scala / F#</u>

Immutability

Expressions

Pattern matching

Composition

Currying

Emphasis of coding paradigms

Functional	Transform state with a series of functions.
Managed Execution	Run within a virtual machine.
Object Oriented	Model as objects with properties and methods.
Procedural	A series of instructions, but at a higher-level of abstraction.
Assembly	Interact with the CPU, RAM, and other devices.

Why the shift?

Functional	Improve our ability to reason about correctness.
Managed Execution	Provide helpful, cross-platform, runtime services.
Object Oriented	Organize lots of code into smaller, reusable components.
Procedural	Abstract away the details of the hardware.
Assembly	Interact with the CPU, RAM, and other devices.

Big Idea

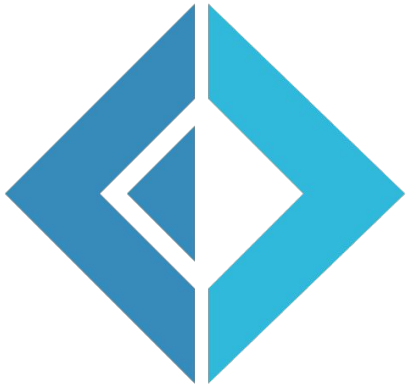
Techniques of functional programming are defining the next era in software languages.

Why? Easier to reason about correctness.

F#

One of the new languages designed around the functional style.

- For .NET / .NET Core
- Functional-first
- Cross-platform
- General-purpose
- Open source



Immutability

What?
Once data is initialized, it cannot be modified.

What?!
Make a copy of the data, with changes applied.

Familiar Example of Immutability

https://docs.oracle.com/javase/7/docs/api/java/lang/String.html

Overview Package Class Use Tree Deprecated Index Help

Java™ Platform Standard Ed. 7

java.lang

Class String

The `String` class represents character strings. All string literals in Java programs, such as `"abc"`, are implemented as instances of this class.

Strings are constant; their values cannot be changed after they are created. String buffers support mutable strings. Because String objects are immutable they can be shared.

`String`	`toUpperCase()` Converts all of the characters in this `String` to upper case

F# Immutability

https://try.fsharp.org

```
let x = 3
x = x + 1
printfn "%d" x
```

F# Records

F# records combine multiple values.

```
// F# record definition
type Workshop = {
    Name: string
    Attendees: int
}
```

```
// F# record instance
let workshop = {
    Name = "F# and Immutability"
    Attendees = 10
}
```

```
printfn "%A" workshop
```

F# Records

F# record instances can be defined:

- On multiple lines
- On the same line

```
// F# record instance
let workshop = {
    Name = "F# and Immutability"
    Attendees = 10
}
```

```
// F# record instance
let workshop = { Name = "F# and Immutability"; Attendees = 10 }
```

F# Records

F# records combine multiple values.

F# records are immutable.

```
workshop.Attendees = 13

workshop = {
    Name = "F# and Immutability"
    Attendees = 13
}
```

Practice summary

The let keyword instantiates a value

```
let x = 3
```

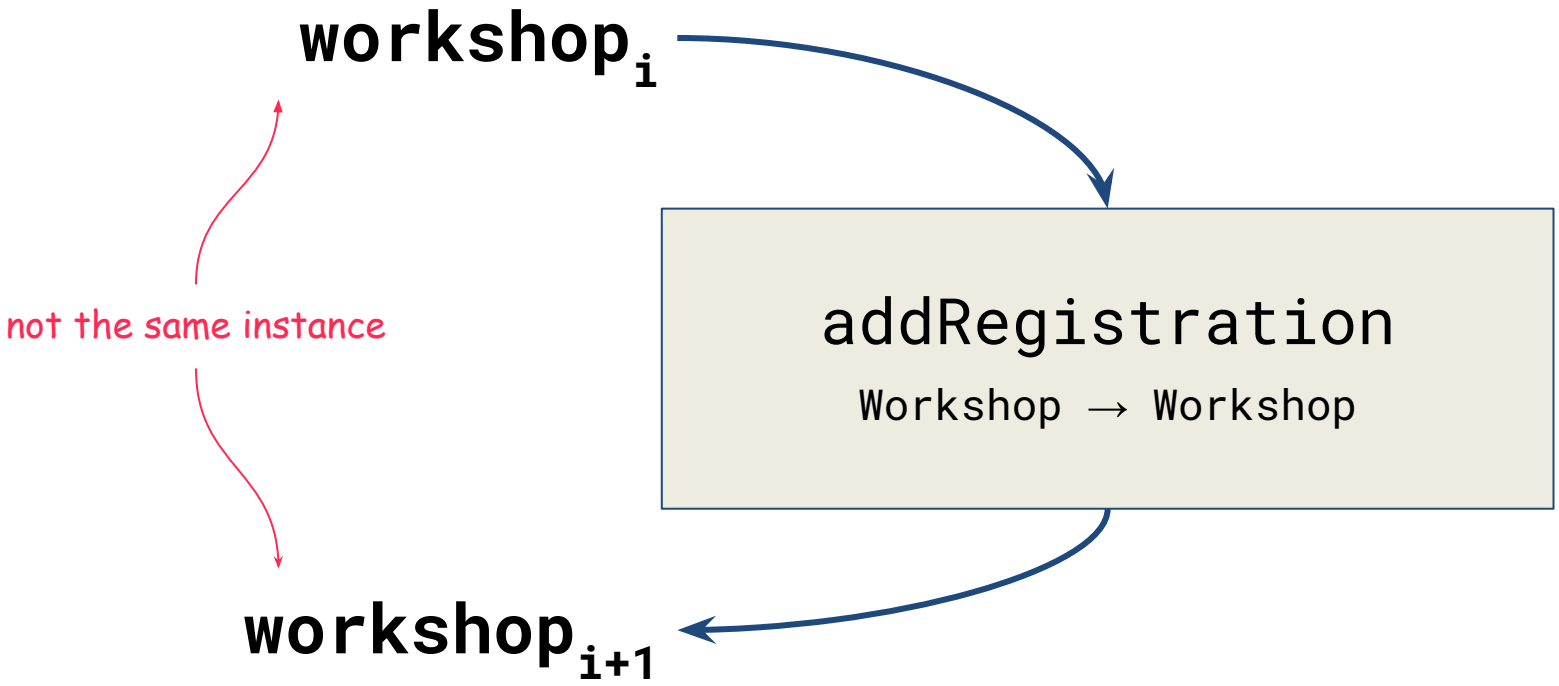
F# records combine multiple values.

```
// F# record definition
type Workshop = {
    Name: string
    Attendees: int
}
```

```
// F# record instance
let workshop = {
    Name = "F# and Immutability"
    Attendees = 10
}
```

F# values are immutable by default

Immutability and Functions



F# Records

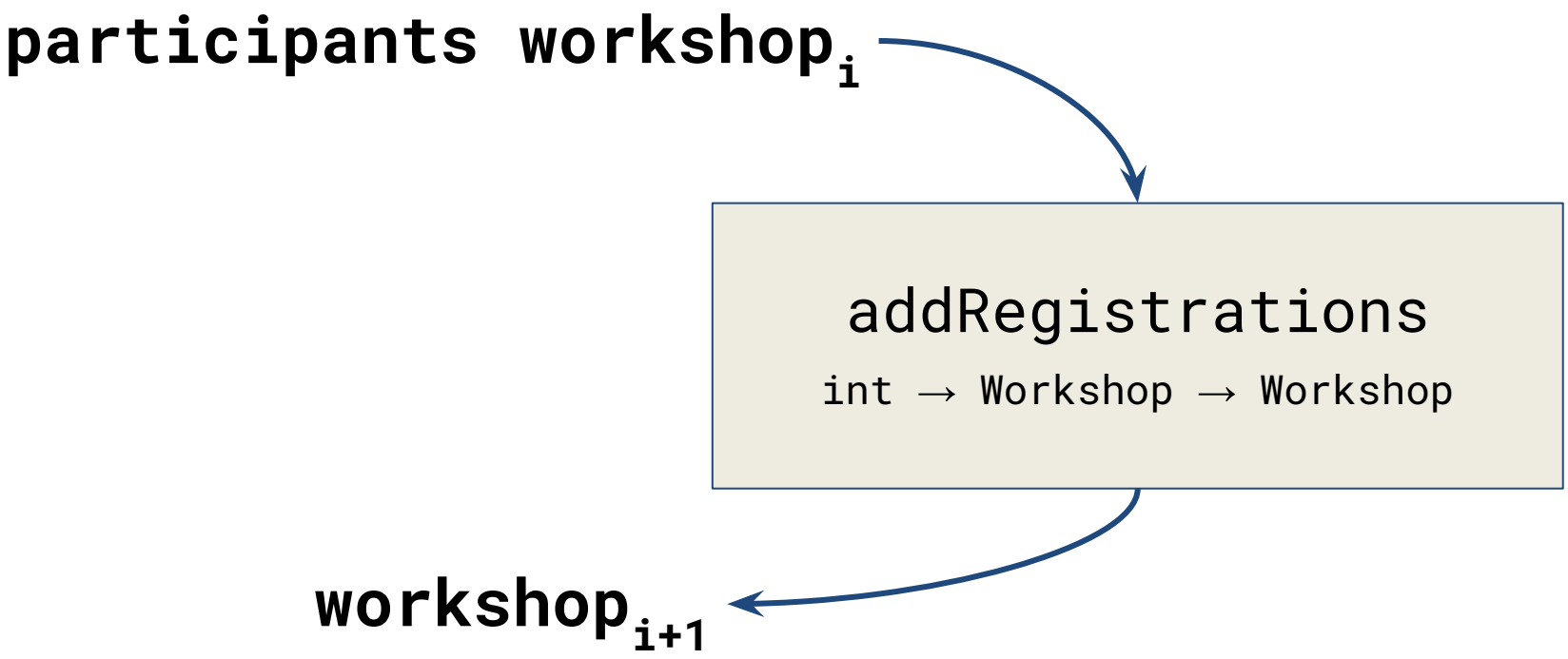
F# records combine multiple values.

F# records are immutable.

```
let addRegistration (w: Workshop) = // Workshop -> Workshop
    {
        Name = w.Name
        Attendees = w.Attendees + 1
    }


let workshop2 = addRegistration workshop
```

Immutability and Functions



F# Records

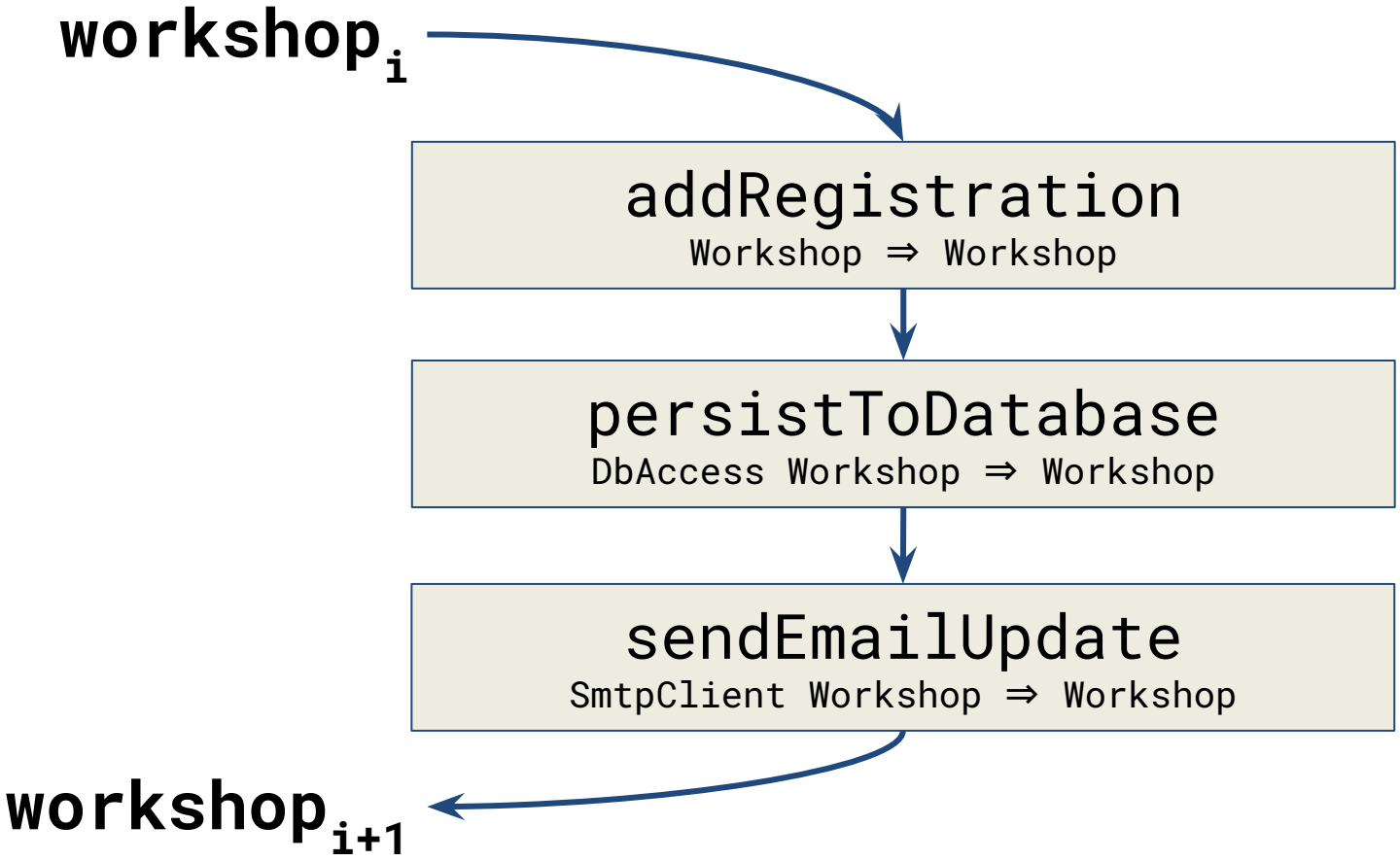
F# records combine multiple values.

F# records are immutable.

```
let addRegistrations (p: int) (w: Workshop) = // int -> Workshop -> Workshop
    {
        Name = w.Name
        Attendees = w.Attendees + p
    }


let workshop2 = addRegistrations 5 workshop
```

Function Pipelines



F# Pipe Operator

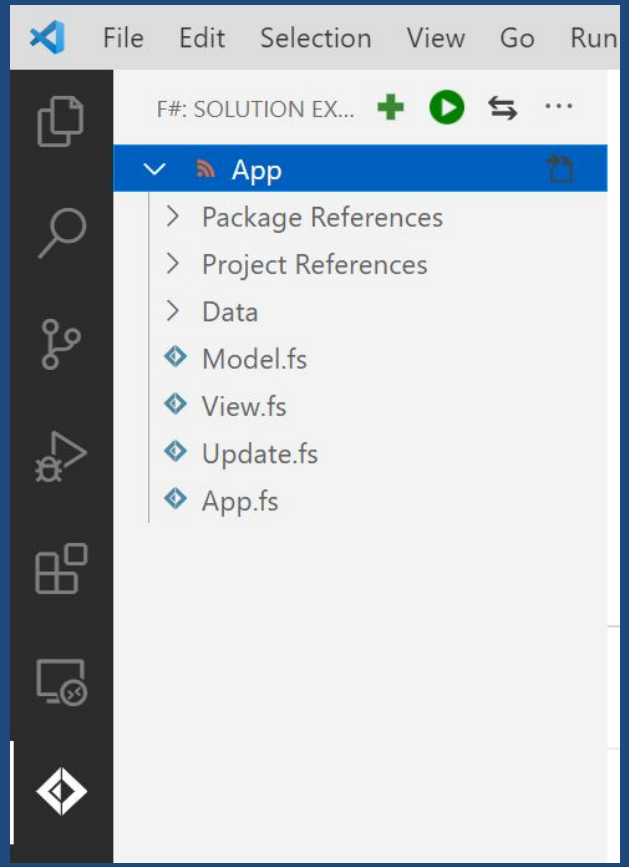
F# has an operator for piping the output of one function into another.

```
let nextWorkshop =
    prevWorkshop
    |> addRegistration
    |> persistToDatabase db
    |> sendEmailUpdate smtp
```

```
result
|> Seq.map toDateRanges
|> Seq.toList
|> function
    | [] -> State.Always
    | [ d ] -> State.Sometimes(d)
    | _ -> failwith "Duplicate keys!”
|> Concurrency.version nextVersion
```

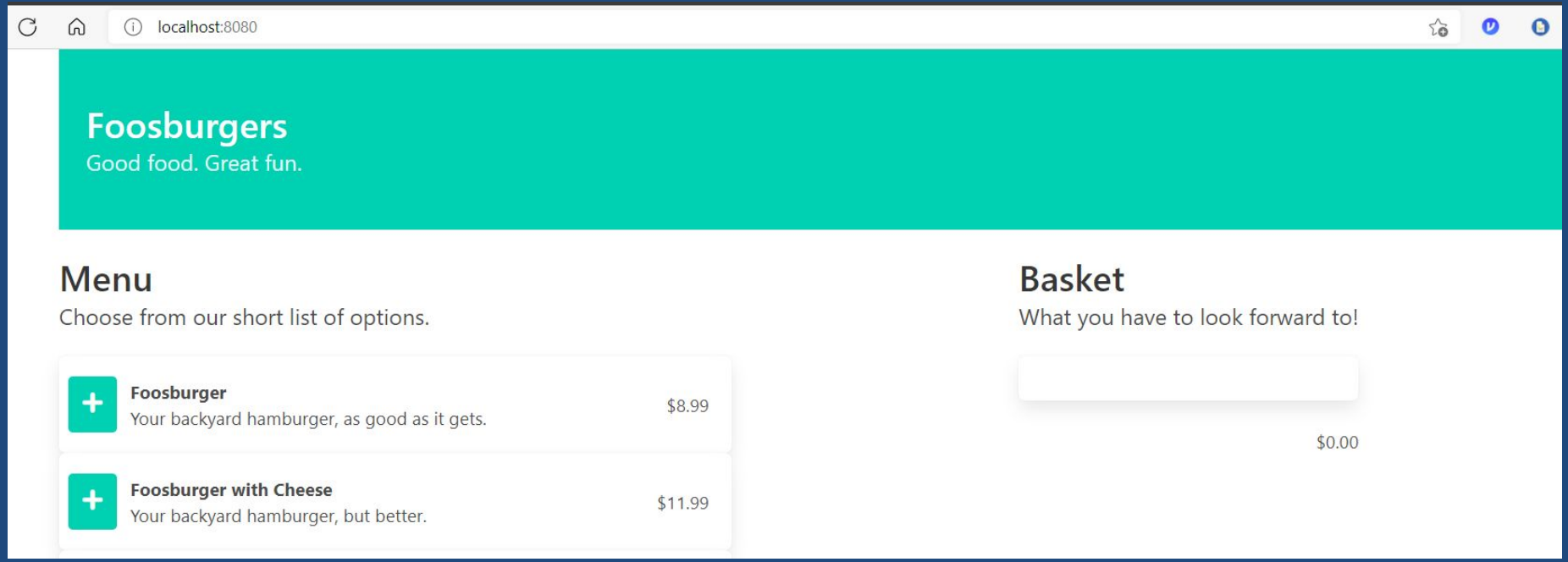
Confirm the setup is complete

- Look for the Ionide extension.



Confirm the website is running

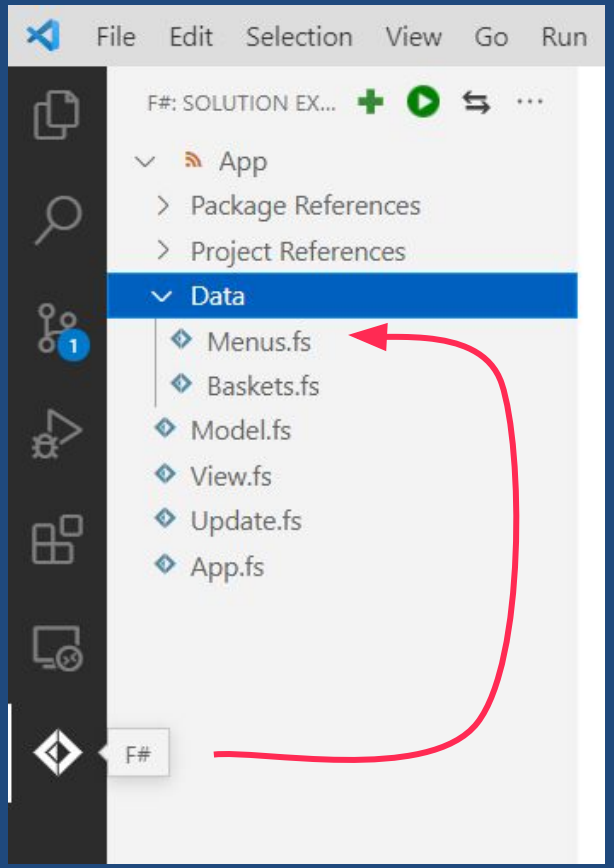
- **Open** http://localhost:8080



Add a record to the practice exercise

In Menu.fs:

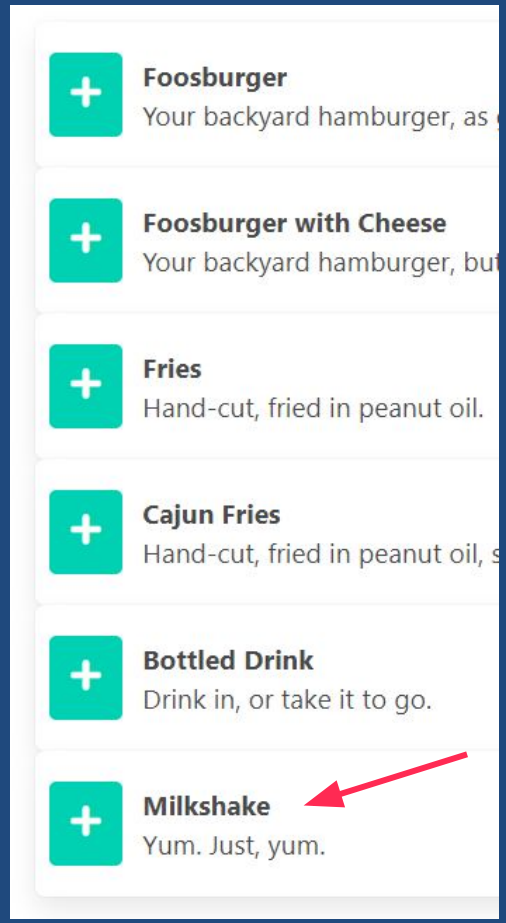
- Add an F# record instance to the list of sample menu items.



Add a record to the practice exercise

In Menu.fs:

- Add an F# record instance to the list of sample menu items.



Practice summary

Edit existing F# code.

Add an F# record to a list of values.

```
{ MenuItem.Id = 6
  Name = "Milkshake"
  Description = "Yum. Just, yum."
  Price = 5.99M }
```

F# Lists

Values (including records) can be grouped into lists.

```
let workshops = [
    { Name="Assembly for the dev"; Attendees=0 }
    { Name="Device driver coding"; Attendees=2 }
    { Name="Intro to OO Programming"; Attendees=5 }
    { Name="Java for JavaScript devs"; Attendees=10 }
    { Name="F# and Immutability"; Attendees=93 }
]
```

F# Lists

F# lists are immutable too!

```
let newWorkshop = {
    Name="F# and List Processing"
    Attendees=93
}

// prepend an item
let workshops2 = newWorkshop :: workshops

// append an item
let workshops3 = workshops @ [ newWorkshop ]
```

F# Lists

F# lists are immutable.

F# lists can be used in a pipeline.

```
workshops
|> List.sortByDescending (fun w -> w.Attendees)
|> List.take 3
|> List.map (fun w -> w.Name)
|> List.iter (printfn "%s")
```

F# Lambda Expressions

```
workshops
|> List.sortByDescending (fun w -> w.Attendees)
|> List.take 3
|> List.map (fun w -> w.Name)
|> List.iter (printfn "%s")
```

```
// Workshop -> int
fun w -> w.Attendees
```

"lambda expression"
"arrow function"
"anonymous function"

Notice the order of the basket items

Basket

What you have to look forward to!

Fries $4.99

Foosburger $8.99

Bottled Drink $2.99

Foosburger $8.99

$25.96

Sort the items in the basket

In Baskets.fs:

- Find the add(...) function.
- Sort the items in the nextBasket:

```
Items = prevBasket.Items
        |> List.append [ newBasketItem ]
        |> List.sortBy(fun i -> i.MenuItem.Id)
```

Practice summary

Add to an existing pipeline of functions.

Use a “lambda expression.”

```
Items = prevBasket.Items
        |> List.append [ newBasketItem ]
        |> List.sortBy(fun i -> i.MenuItem.Id)
```

Outline

- ☐ About the presenter
- ☐ Start downloads of the Docker images
- ☐ Evolution of Software Paradigms
- ☐ F# overview (w/ code demos)
- ☐ Immutability (w/ code demos)
- ☐ Model-View-Update (w/ code demos)
- ☐ Practice Exercises

Model View Update

Model: state of the component

View: render a given state

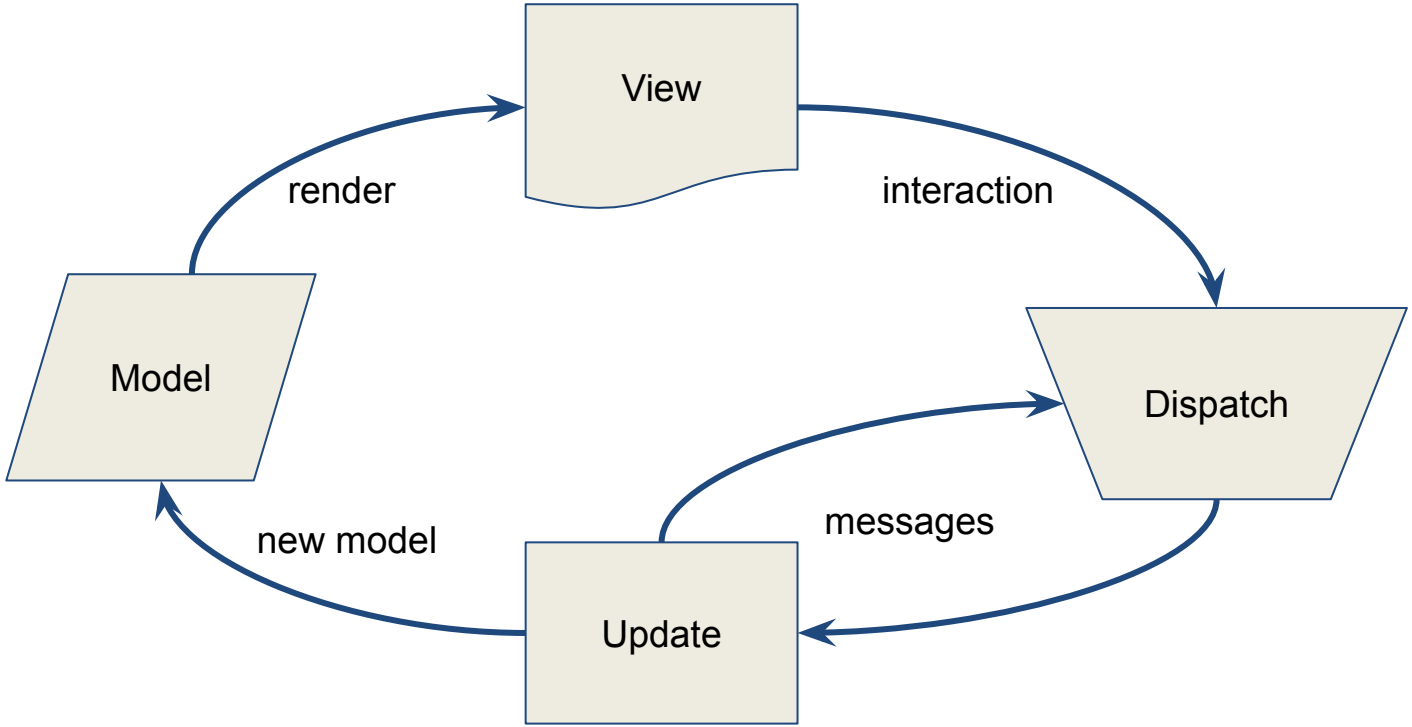
Update: new copy of the state after event

<u>Advantages</u>

Compatible with immutability.

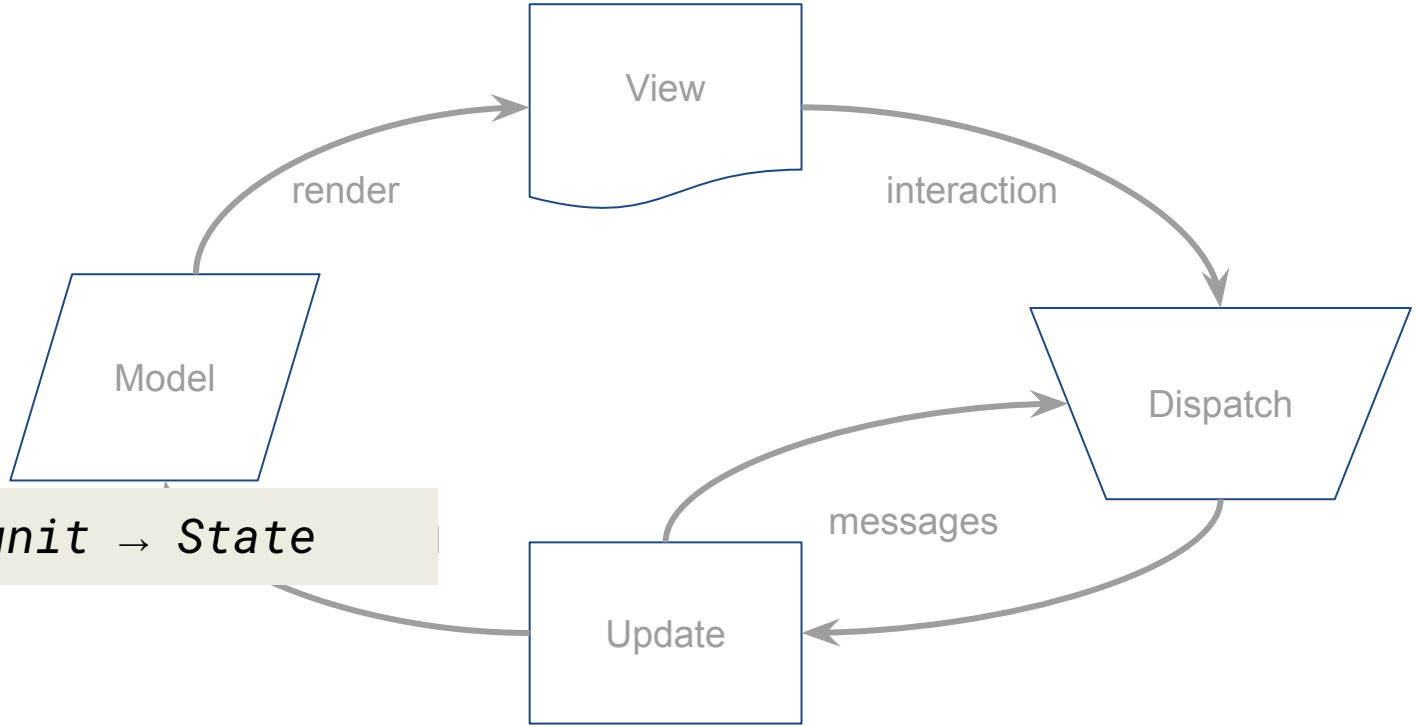
Works nicely with React.

Model View Update



Model View Update - Elmish

```
render: State → Dispatcher → View
```




```
init: unit → State
```

```
update: Message → State → State
```

Explore the Practice Exercise

Do you see MVU?

What are signatures of the Elmish functions?

F#: SOLUTION EX...

App
- Package References
- Project References
- Data
- Model.fs
- View.fs
- Update.fs
- App.fs

App.fs

src > App.fs > {} Olo.MvuMenu.App

```
module Olo.MvuMenu.App

open Elmish
open Elmish.React

Program.mkSimple Model.init Update.update View.render
|> Program.withReactSynchronous "mvu-menu-app"
|> Program.run
```

Complete Practice Exercise

https://github.com/WWCode-SV/fsharp-workshop/instructions.md

https://github.com/ololabs/fsharp-mvu-menu/blob/main/instructions.md

147 lines (110 sloc) | 5.43 KB

Raw Blame

3. Prefix the subtotal with the text "Subtotal: ".
   - In View.fs, find the `renderBasketItems` function definition.
   - Find the line that displays the subtotal.
   - Update that line to prefix the subtotal amount with the text "Subtotal: ".
4. Bold the subtotal.
   - Continuing in the `renderBasketItems` function, notice that the subtotal is styled to be right-aligned text.
   - Follow that example to set the subtotal text to also be bold.

Cleanup

In Visual Studio Code

1. Select File → **Exit**

In Docker Desktop

1. Select the **Containers / Apps** tab
2. **Delete** (trash can) the container.
3. Select the **Images** tab**.**
4. **Remove** the vsc-fsharp… image (and others).

Objectives

At the end of this workshop,...

- ☐ Motivated to adopt functional techniques.
- ☐ Describe the MVU pattern.
- ☐ Have experience editing F# code.

Wallace Kelly
wallace.kelly@olo.com