|  |  |  |  |
| --- | --- | --- | --- |
| v2.0 |  |  | Jan-2016 |

SSeasdfad

F# Workshop

Exercises

Table of Contents

[Introduction 2](#_Toc439596078)

[Module 1 4](#_Toc439596084)

[Module 2 7](#_Toc439596085)

[Module 3 9](#_Toc439596086)

[Module 4 11](#_Toc439596087)

# Introduction

Do you want to learn F# and Functional Programming? Well, you better start coding!

Learning a new programming language is not easy, on top of reading a lot you need to practice even more.

This workshop is designed to teach you some of the basics of F# and Functional Programming by combining theory and practice.

The course is split into 4 modules, each of them contains a presentation (theory) and one exercise (practice). You can find exercises for each module in this document, for the presentation and source code, refer to the section “Source Code, Additional Material and Updates”.

**1**

**2**



**3**

**4**

## Pre-requisites

### Windows

* Visual Studio 2015 Community or
* Xamarin Studio 6 or
* Atom + F# Compiler + Ionide package or
* Visual Studio Code + F# Compiler + Ionide package

### Mac

* Xamarin Studio 6 + Mono or
* Atom + Mono + Ionide package or
* Visual Studio Code + Mono + Ionide package

### Linux

* Atom + Mono + Ionide package or
* Visual Studio Code + Mono + Ionide package

You also need internet connection to download the dependencies.

## Code Conventions

Every time you see a box with this icon: , it means you need to run that code in the F# Interactive.

|  |  |
| --- | --- |
|  | > increaseCredit vipCondition customer1;; |

When you see a white box, this is code you need to write in a source file.



## Source Code, Additional Material and Updates

<http://fsharpworkshop.com/>

<https://github.com/jorgef/fsharpworkshop>

## Author

Jorge Fioranelli (@jorgefioranelli)

Licensed under the Apache License, Version 2.0

# Module 1

* Bindings
* Functions

Do not copy and paste the code, you must type each exercise in, manually.

* Tuples
* Records

Duration: 15 minutes

## Before we start

#### Visual Studio Users (Windows)

* Open Visual Studio
* Open the solution FSharpWorkshop.sln located in the root folder.
* Build the solution (Build -> Build Solution). This process will download all the packages and will prompt a security dialog asking you to enable the type provider, click “Enable”.
* Double check that the build finishes successfully.
* Open the F# Interactive (View -> Other Windows -> F# Interactive)
* Go to the Module1/Application, open Try.fsx, write “let a = 1”, highlight the entire line, right click over it and click “Execute in Interactive”.
* Double check you see “val a : int = 1” in the F# Interactive window.

#### Xamarin Studio Users (Mac)

* Open Xamarin Studio
* Open the solution FSharpWorkshop.sln located in the root folder.
* Open the Terminal, go to the Module1 folder and run ./runtests.sh. If you get “Permission Denied” run chmod +x runtests.sh and try again (you will need to do the same for all the other .sh files).
* Double check that the build finishes successfully.
* Open the F# Interactive (View -> Other Windows -> F# Interactive)
* Go to the Module1/Application, open Try.fsx, write “let a = 1”, highlight the entire line, right click over it and click “Send selection to F# Interactive”.
* Double check you see “val a : int = 1” in the F# Interactive window.

#### Atom / Visual Studio Code Users (Windows, Mac or Linux)

* Open Visual Studio Code
* Open the root folder (File -> Open Folder)
* Open the F# Interactive (View -> Command Palette -> FSI: Start)
* Open the Command Prompt (Windows) / Terminal (Mac or Linux), go to the Module1 folder and execute runtests.bat (Windows) / runtests.sh (Mac or Linux). This process will compile and download all the packages (no tests are enabled yet). If you get “Permission Denied” run chmod +x runtests.sh and try again (you will need to do the same for all the other .sh files).
* Double check it finishes without errors.
* Open the F# Interactive (View -> Command Palette -> FSI: Start)
* Go to the Module1/Application, open Try.fsx, write “let a = 1”, highlight the entire line and go to View -> Command Palette -> FSI: Send Selection.
* Double check you see “val a : int = 1” in the F# Interactive window.

1. Go to the Module1/Application, open Types.fs and create a record type called “Customer” as follows:



2. Send the customer type in the F# interactive (do not highlight the “module Types” line). For more details about how to execute code in the F# Interactive see the “Before we start” section. You should see the following output:

|  |  |
| --- | --- |
|  | type Customer =  {Id: int;  IsVip: bool;  Credit: decimal;} |

3. Open Module1/Application/Try.fsx and create a new customer as follows and send it to the F# Interactive.

This should be the result:

|  |  |
| --- | --- |
|  | val customer1 : Customer = {Id = 1;  IsVip = false;  Credit = 10M;} |

4. Create another customer and send it to the F# Interactive.



This should be the result:

|  |  |
| --- | --- |
|  | val customer2 : Customer = {Id = 2;  IsVip = false;  Credit = 0M;} |

5. Open Module1/Tests/Tests.fs, uncomment the test 1-1, save all the files and run the tests by executing Module1/runtests.bat (Win) or Module1/runtests.sh (Mac or Linux) from the console. Check that the test passes.

6. Open the file Module1/Application/Functions.fs and add a function called “tryPromoteToVip”:



7. Highlight the function (without including “module Functions” and “open Types” lines) and send it to the F# Interactive. You should see this output:

|  |  |
| --- | --- |
|  | val tryPromoteToVip : customer:Customer \* spendings:decimal -> Customer |

8. Open Module1/Application/Try.fsx, invoke the new function and send it to the F# Interactive



You should see this output:

|  |  |
| --- | --- |
|  | val vipCustomer : Customer = {Id = 1;  IsVip = true;  Credit = 10M;} |

Now test it with customer2 using 99M as spendings in the Module1/Application/Try.fsx file.

9. Open Module1/Tests/Tests.fs, uncomment tests 1-2 and 1-3, save all the files and run the tests.

10. Add a function called “getSpendings” to Module1/Application/Functions.fs:



11. Send it to the F# Interactive and test it with customer1 and customer2 in Module1/Application/Try.fsx.

12. Open Module1/Tests/Tests.fs, uncomment tests 1-4 and 1-5, save all the files and run the tests.

# Module 2

* High order functions
* Pipelining

Do not copy and paste the code, you must type each exercise in, manually.

* Partial application
* Composition

Duration: 20 minutes

1. Go to the Module2/Application, open Functions.fs and create a function called “increaseCredit”:



2. Send it to the F# Interactive and test it with customer1 and customer2 in Module2/Application/Try.fsx.

3. Change “increaseCredit” to be able receive the condition as a parameter:



4. Send the function to the F# Interactive and test it in Module2/Application/Try.fsx using a lambda expression in this way: 

5. Open Module2/Tests/Tests.fs, uncomment the tests 2-1, 2-2 and 2-3, save all the files and run the tests by executing Module2/runtests.bat (Win) or Module1/runtests.sh (Mac or Linux) from the console. Check that the tests pass.

6. Create a function called “vipCondition” in the file Module2/Application/Functions.fs:



7. Send the function to the F# Interactive and test the “increaseCredit” function again but this time using the “vipCondition” function:

  
8. Now test it again but this time using the pipelining operator to:

9. Try calling “increaseCredit” with just “vipCondition” and check if the result is another function that expects the missing argument (customer):



You should see the following output:

|  |  |
| --- | --- |
|  | > val result : (Customer -> Customer) |

10. Uncomment tests 2-4 and 2-5, save all the files and run the tests.

11. Create a function called “increaseCreditUsingVip” in Module2/Application/Functions.fs:



12. Open Module2/Tests/Tests.fs, uncomment test 2-6, save all the files and run the tests.

13. Create a function called “upgradeCustomer” in Module2/Application/Functions.fs:



14. Send “increaseCreditUsingVip” and “upgradeCustomer” to the F# Interactive and test “upgradeCustomer” with customer1 and customer2.

15. Refactor “upgradeCustomer” to use the pipelining operator and test it in the F# interactive:



16. Send the new “upgradeCustomer” to the F# Interactive and test it again with customer1 and customer2.

17. Refactor “upgradeCustomer” again to use composition:



18. Open Module2/Tests/Tests.fs, uncomment tests 2-7 and 2-8, save all the files and run the tests.

# Module 3

* Options
* Pattern matching

Do not copy and paste the code, you must type each exercise in, manually.

* Discriminated unions
* Units of measure

Duration: 20 minutes

1. Go to the Module3/Application, open Types.fs and create a new record called “PersonalDetails”, a discriminated union called “Notifications” and two units of measure: “AUD” and “USD”. You also need to add them to the “Customer” (note that the types need to be declared before “Customer”):



2. Highlight all but the “module Types” line and send it to the F# Interactive (including “open System”).

4. Update the “increaseCredit” function to use USD in Module3/Application/Functions.fs:



5. Open Module3/Tests/Tests.fs, uncomment the tests 3-1, 3-2 and the customer defined at the top, save all the files and run the tests by executing Module3/runtests.bat (Win) or Module3/runtests.sh (Mac or Linux) from the console. Check that the tests pass.

6. Create a function called “isAdult” in Module3/Application/Functions.fs:



7. Send “isAdult” to the F# Interactive, open Module3/Application/Try.fsx and send customer1 and customer2 to the F# Interactive, and test isAdult with both.

8. Open Module3/Tests/Tests.fs, uncomment tests 3-3, 3-4 and 3-5, save all the files and run the tests.

9. Create a function called “getAlert” in Module3/Application/Functions.fs:



10. Send “getAlert” to the F# Interactive and test it with customer1 and customer2.

11. Open Module3/Tests/Tests.fs, uncomment tests 3-6 and 3-7, save all the files and run the tests.

# Module 4

* Functional lists
* Recursion

Do not copy and paste the code, you must type each exercise in, manually.

* Object-oriented Programming
* Type providers

Duration: 20 minutes

1. Go to the Module4/Application, open Data.fs and add the following code:



2. Create a new function called “getSpendingsByMonth” in Module4/Application/Functions.fs right after “tryPromoteToVip” and before “getSpendings”:



3 .Open Module4/Tests/Tests.fs, uncomment the test 4-1, save all the files and run the tests by executing Module4/runtests.bat (Win) or Module4/runtests.sh (Mac or Linux) from the console. Check that the test passes.

4. Open Module4/Application/Functions.fs and create another function called “weightedMean” right after the “getSpendingsByMonth”:



5. Open Module4/Tests/Tests.fs, uncomment test 4-2, save all the files and run the tests.

6. Open Module4/Application/Functions.fs and change the implementation of “getSpendings” to use “getSpendingsByMonth” and “weightedMean”:



7. Open Module4/Tests/Tests.fs, uncomment test 4-3, save all the files and run the tests.

8. Open Module4/Application/Data.fs and add the following code:



9. Open Module4/Application/Services.fs and add the following class:



10. Open Module4/Tests/Tests.fs, uncomment tests 4-4 and 4-5, save all the files and run the tests.

11. Open Module4/Application/Program.fs, uncomment all the code, save all the files and run the application by executing Module4/runtapp.bat (Win) or Module4/runapp.sh (Mac or Linux) from the console.

12. Try the application, upgrade different customers.