|  |  |  |  |
| --- | --- | --- | --- |
| v1.1 |  |  | Nov-2014 |

SSeasdfad

F# Introduction Workshop

Exercises

Table of Contents

1. [Introduction 2](#_Toc402896132)
2. [Exercise 1 4](#_Toc402896138)
3. [Exercise 2 7](#_Toc402896139)
4. [Exercise 3 10](#_Toc402896140)
5. [Exercise 4 12](#_Toc402896141)

# 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**

## Minimum Requirements

* Visual Studio 2013
* Visual F# tools 3.1.2 or greater
* XUnit Runner
* Sql Server
* Visual F# Power Tools (optional)

## Nuget Packages

* XUnit
* Unquote
* FSharp.Data
* FSharp.Data.SqlClient

## 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

# Exercise 1

* Bindings
* Functions
* Tuples
* Records

1. Open the solution Module1\Pre\FSharpIntro.sln and go to the Applications project.

2. Open the file Types.fs and create a record type called “Customer” as follows:



3. Send the customer type in the F# interactive by highlighting it and pressing “Alt+Enter” or right-click “Execute in Interactive” (do not highlight the module line), you should see the following output:

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

4. Create one customer in the F# interactive (press enter after each “;;”)

|  |  |
| --- | --- |
|  | > let customer1 = { Id = 1; IsVip = false; Credit = 10.0 };; |

This should be the result:

|  |  |
| --- | --- |
|  | val customer1 : Customer = {Id = 1;  IsVip = false;  Credit = 10.0;} |

5. Create another customer:

|  |  |
| --- | --- |
|  | > let customer2 = { Id = 2; IsVip = false; Credit = 0.0 };; |

This should be the result:

|  |  |
| --- | --- |
|  | val customer2 : Customer = {Id = 2;  IsVip = false;  Credit = 0.0;} |

6. Go to the Tests.fs file located in the Tests project, uncomment and run the test 1-1

7. Open the file Functions.cs and add a function called “tryPromoteToVip”:



7. Highlight the function (without including “module Functions” and “open Types” lines), send it to the F# Interactive and test it by typing the following:

|  |  |
| --- | --- |
|  | > tryPromoteToVip (customer1, 101.0);; |

You should see this output:

|  |  |
| --- | --- |
|  | val it : Customer = {Id = 1;  IsVip = true;  SpendingLimit = 10.0;} |

8. Now test it with customer2

9. Uncomment and run tests 1-2 and 1-3

10. Add a function called “getSpendings” to the Functions.fs file:



11. Send it to the F# Interactive and test it with customer1 and customer2

12. Uncomment and run tests 1-4 and 1-5

# Exercise 2

* High order functions
* Pipelining
* Partial application
* Composition

1. Without closing Visual Studio, go to File -> Close Solution and then to File -> Open Project/Solution… and select Module2\Pre\FSharpIntro.sln and go to the Applications project. You need to keep Visual Studio open so you don’t lose the F# Interactive session.

2. Create a function called “increaseCredit” in the file Functions.fs:



3. Send it to the F# Interactive and test it with customer1 and customer2.

4. Refactor “increaseCredit” to be able receive the condition as a parameter:



5. Send the function to the F# Interactive and test it using a lambda expression in this way:

|  |  |
| --- | --- |
|  | > increaseCredit (fun c -> c.IsVip) customer1;; |

6. Uncomment and run tests 2-1, 2-2 and 2-3

7. Create a function called “vipCondition” in the file Functions.fs:



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

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

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

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

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

|  |  |
| --- | --- |
|  | > increaseCredit vipCondition;;  val it : (Customer -> Customer) = <fun:it@5-4> |

11. Uncomment and run tests 2-4 and 2-5

12. Create a function called “increaseCreditUsingVip” in the file Functions.fs:



13. Uncomment and run test 2-6

14. Create a function called “upgradeCustomer” in the file Functions.fs:



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

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



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

18. Refactor “upgradeCustomer” again to use composition:



19. Uncomment and run tests 2-7 and 2-8

# Exercise 3

* Options
* Pattern matching
* Discriminated unions
* Units of measure

1. Without closing Visual Studio, go to File -> Close Solution and then to File -> Open Project/Solution… and select Module3\Pre\FSharpIntro.sln and go to the Applications project. You need to keep Visual Studio open so you don’t lose the F# Interactive session.

2. Create a new record called “PersonalDetails”, a discriminated union called “Notifications” and two units of measure “AUD” and “USD”. Add them to the “Customer” in the file Types.fs (note that you need to declare them before “Customer”):



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

4. Open the file Data.fs, uncomment both customers and send them to the F# Interactive (do not select the “module …” and “open …: lines).

5. Update the “increaseCredit” function to use USD:



6. Uncomment and run tests 3-1 and 3-2 (you will also need to uncomment the “customer” value defined at the top of the file Test.fs)

7. Create a function called “isAdult” in the file Functions.fs:



8. Send “isAdult” to the F# Interactive and test it with customer1 and customer2.

9. Uncomment and run tests 3-3, 3-4 and 3-5

10. Create a function called “getAlert” in the file Functions.fs:



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

12. Uncomment and run tests 3-6 and 3-7

# Exercise 4

* Functional lists
* Recursion
* Object Oriented Programming
* Type Providers

1. Without closing Visual Studio, go to File -> Close Solution and then to File -> Open Project/Solution… and select Module4\Pre\FSharpIntro.sln and go to the Applications project. You need to keep Visual Studio open so you don’t lose the F# Interactive session.

2. Create a new function called “getSpendingsByMonth” in the file Functions.fs right before the “getSpendings” function:



4. Send it to the F# Interactive and test it with customer1 and customer2

5. Uncomment and run tests 4-1 and 4-2

6. Create another function called “weightedMean” right after the “getSpendingsByMonth”:



7. Uncomment and run test 4-3

8. Change the implementation of “getSpenginds” to use “getSpendingsByMonth” and “weightedMean”:



9. Send both “weightedMean” and “getSpendings” to the F# Interactive and test “getSpendings” with customer1 and customer2

10. Uncomment and run test 4-4

11. Add the following Nuget packages (use the option include pre-releases when searching)

- F# Data (Id: FSharp.Data)

- FSharp.Data.SqlClient

After installing each package and trying to compile the solution, you will see one or two security dialog asking you if you want to enable the type provider, click “Enable”.

12. Open the Data.fs file located in the Application project and add the following code:



13. Change the “getSpendingsByMonth” function located in the file Functions.fs:



14. Create a database called “FSharpIntro” in your local SqlServer and run the Data.sql file located in the Application project.

15. Open the Data.fs file and add the following code (adjust the connection string to point to your local SQLServer):



16. Right click on the “References” folder of the Application project and select “Send References to F# Interactive”.

17. Highlight everything in the Data.fs file but the first two lines (“module Data” and “open Types” should not be included) and executed in the F# Interactive.

18. Execute the following code in the F# Interactive:

|  |  |
| --- | --- |
|  | > getSpendings 1;; |

You should get the following result:

|  |  |
| --- | --- |
|  | val it : float list =  [60.1; 60.1; 60.1; 60.1; 60.1; 60.1; 60.1; 60.1; 60.1; 60.1; 60.1; 60.1] |

19. Now execute the following code:

|  |  |
| --- | --- |
|  | > getCustomers ();; |

You should get the following result:

|  |  |
| --- | --- |
|  | val it : seq<Customer> =  seq  [{Id = 1;  IsVip = false;  Credit = 0.0;  PersonalDetails = null;  Notifications = NoNotifications;}; {Id = 2;  IsVip = false;  Credit = 10.0;  PersonalDetails = null;  Notifications = NoNotifications;};  {Id = 3;  IsVip = false;  Credit = 30.0;  PersonalDetails = null;  Notifications = NoNotifications;}; {Id = 4;  IsVip = true;  Credit = 50.0;  PersonalDetails = null;  Notifications = NoNotifications;}] |

20. In the Data.fs file, write the following “updateCustomer” function right after the “getCustomers” one:



21. Open the file Services.fs and add the following class:



22. Open Program.fs, uncomment all the code and run the application