

Introducción a la programación funcional en C#

Alberto Colom Monfort

@acolommonfort

Agenda



¿Que es? y que no es



Paradigma de programación

Funciones, funciones y más funciones

Se centra en describir en lugar de como obtener

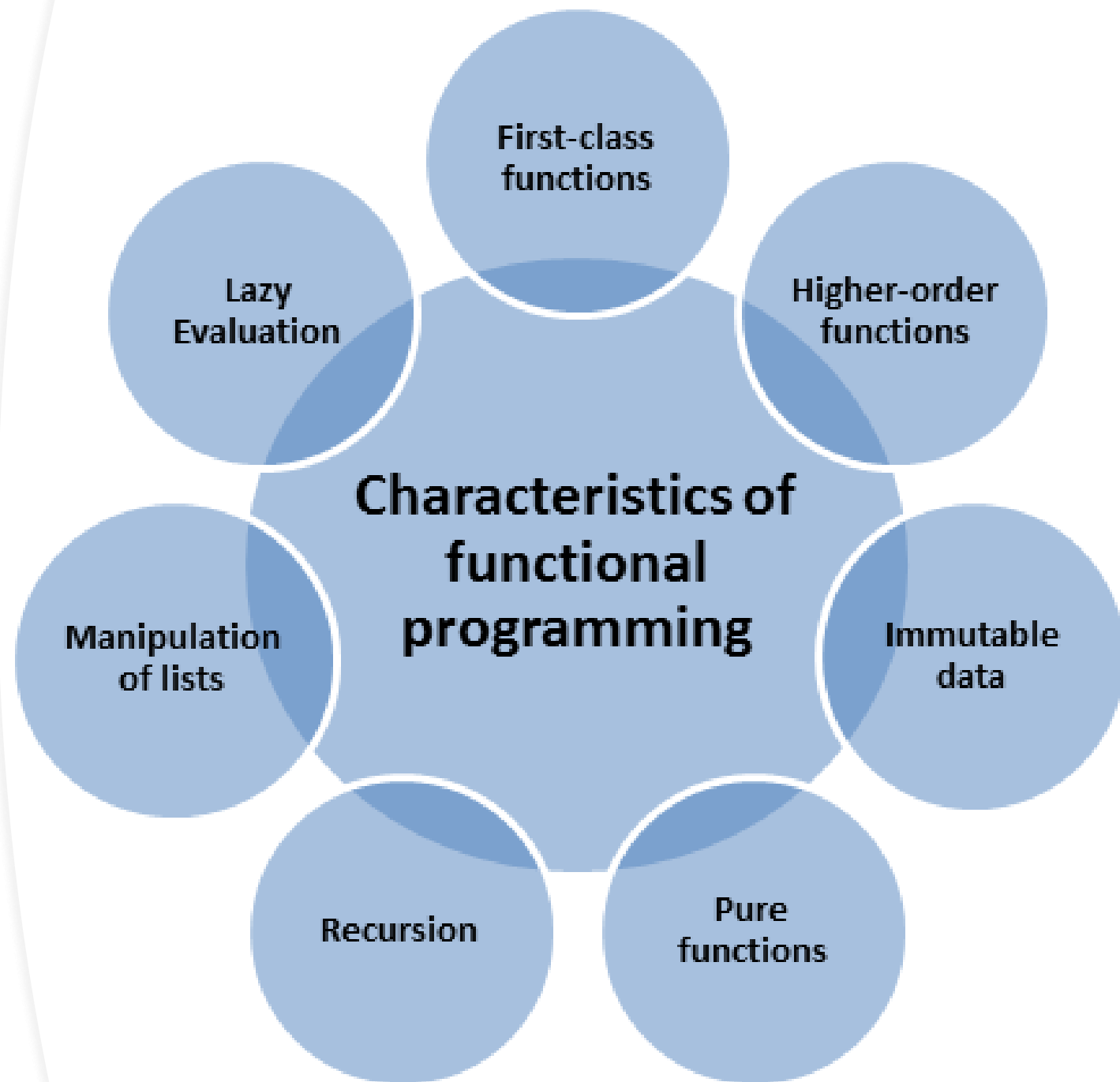
Composición

Expresiones vs Sentencias

No es una bala de plata

En que se basa la programación funcional

- Eliminar complejidad aplicando composición
- Inmutability + Pure function



OO vs FP (Clean Coder)

Orientado a objetos

- Single Responsibility Principle
- Open Closed Principle
- Dependency Inversión Principle
- Interface Segregation Principle
- Factory Pattern
- Strategy Pattern
- Decorator Pattern
- Visitor Pattern

Funcional

- Functions
- Functions
- Functions, also
- Functions
- Yes, functions
- Ohhh functions!
- Functions
- Functions

Características de lenguajes funcionales

Estructuras
monadicas

Funciones

Currying /
Partial
application

Ausencia de
efectos
colaterales

Tuplas

Recursion por
cola (Tail
Recursion)

Beneficios



Historia de C#

Y su aproximación al paradigma funcional

Un poquito de historia

- Década de 1930 Lambda calculo ([Alonzo Church](#) y [Stephen Kleene](#))
- En 1948 Turing define la máquina de Turing
- En la década de los 50, lenguajes de alto nivel
 - FORTRAN,LISP,COBOL
- C y ML en los 70C
- C++ en los 80
- En los 90 Haskell, Python, Java, JavaScript, Ruby, PHP
- C# (2001) F# (2002) Scala Go Clojure

Camino C# funcional

- C# 2001
- C# 2.0 2005 Generics
- C# 3 2007 Lambda Expressions, Extension Methods
 - Marca el inicio de lenguaje hibrido
 - Permite dar soporte a la implementación de LINQ en 2007 por el .NET Framework 3.5
- C# 5 2012 Async
- C# 6 2015 Expression Body Members, Import static methods from a namespace
- C# 7 2017 Tuples and deconstruction, Pattern Matching, Local Functions, Discards, throw Expression
- C# 8 Pattern matching enhancements, static local functions
- C# 9 Records, Pattern matching enhancements, Attributes on local functions, Static Anonymous functions
- C# 10 Lambda expresión improvements, Record structs

LINQ en C#

LINQ - Where

```
IEnumerable<TSource> Where<TSource>(this IEnumerable<TSource>  
source, Func<TSource, bool> predicate);
```

```
(IEnumerable<TSource>, (TSource => bool)) => IEnumerable<TSource>
```

LINQ - Select / Map



```
IEnumerable<TResult> Select<TSource, TResult>(this  
IEnumerable<TSource> source, Func<TSource, TResult> selector);
```

```
(IEnumerable<TSource>, (TSource => TResult)) => IEnumerable<TResult>  
(Algo<TSource>, (TSource => TResult)) => Algo<TResult>
```

LINQ - SelectMany / Bind

- `IEnumerable<TResult> SelectMany<TSource, TResult>(this
IEnumerable<TSource> source, Func<TSource, IEnumerable<TResult>>
selector)`

```
IEnum<TSource>, (TSource => IEnum<TResult>)) => IEnum<TResult>  
(Algo<TSource>, (TSource => Algo<TResult>)) => Algo<TResult>
```

LINQ - Aggregate / Reduce

```
TAccumulate Aggregate<TSource, TAccumulate>(this  
IEnumerable<TSource> source, TAccumulate seed, Func<TAccumulate,  
TSource, TAccumulate> func)
```

```
(Algo<TSource>, TAccumulate, ((TAccumulate, TSource) =>  
TAccumulate)) => TAccumulate
```

LINQ - Select vs SelectMany o Map vs Bind

Select:

```
(Algo<TSource>, (TSource => TResult)) => Algo<TResult>
```

```
(Algo<TSource>, (TSource => Algo<TResult>)) => Algo<Algo<TResult>>
```



SelectMany:

```
(Algo<TSource>, (TSource => Algo<TResult>)) => Algo<TResult>
```

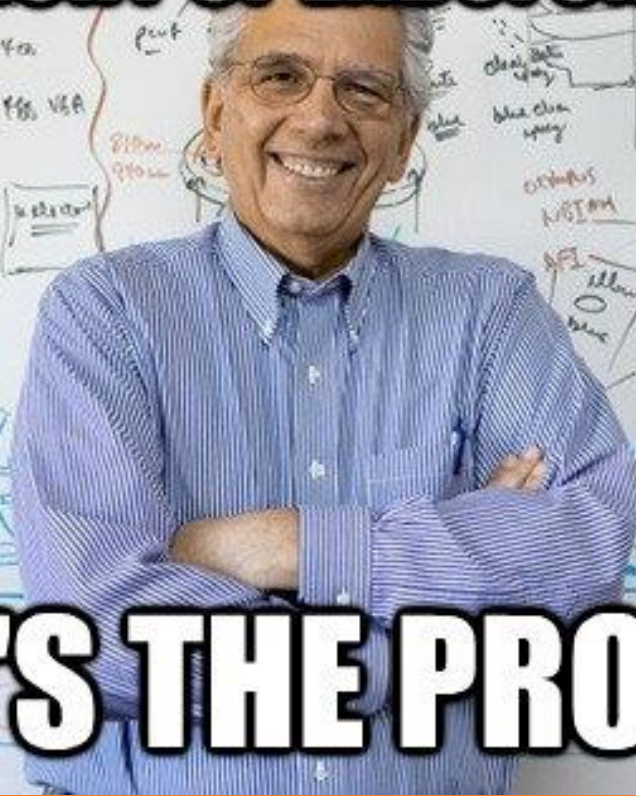

Monads en C#

Que son y algunas construcciones

```
empty = 1;
if ($_POST['user_name']) {
    if ($_POST['user_password_new']) {
        if ($_POST['user_password_new'] === $_POST['user_password_repeat']) {
            if (strlen($_POST['user_password_new']) > 5) {
                if (strlen($_POST['user_name']) < 65 && strlen($_POST['user_name']) > 1) {
                    if (preg_match('/^(a-z\d){2,64}$/i', $_POST['user_name'])) {
                        $user = read_user($_POST['user_name']);
                        if (!isset($user['user_name'])) {
                            if ($_POST['user_email']) {
                                if (strlen($_POST['user_email']) < 65) {
                                    if (filter_var($_POST['user_email'], FILTER_VALIDATE_EMAIL)) {
                                        create_user();
                                        $_SESSION['msg'] = 'You are now registered so please login';
                                        header('Location: ' . $_SERVER['PHP_SELF']);
                                        exit();
                                    } else $msg = 'You must provide a valid email address';
                                } else $msg = 'Email must be less than 64 characters';
                            } else $msg = 'Email cannot be empty';
                        } else $msg = 'Username already exists';
                    } else $msg = 'Username must be only a-z, A-Z, 0-9';
                } else $msg = 'Username must be between 2 and 64 characters';
            } else $msg = 'Password must be at least 6 characters';
        } else $msg = 'Passwords do not match';
    } else $msg = 'Empty Password';
} else $msg = 'Empty Username';
$_SESSION['msg'] = $msg;
```



A MONAD IS JUST A MONOID IN THE CATEGORY OF ENDOFUNCTORS



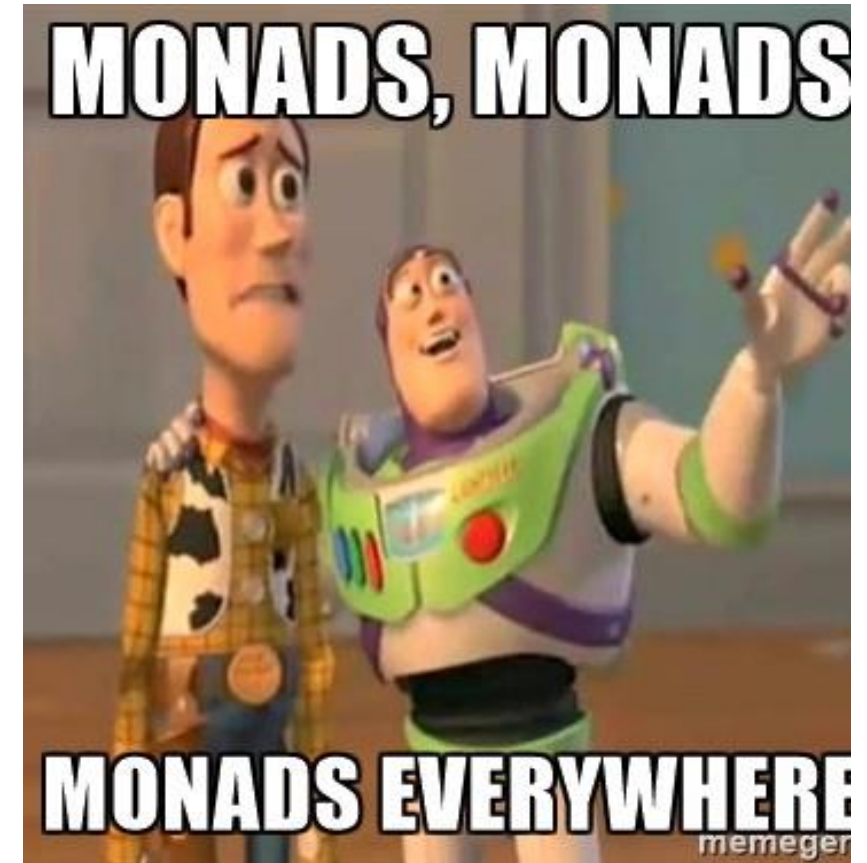
WHAT'S THE PROBLEM?

Que es un “monad”

- Para desarrolladores (Eric Lippert)
 - Una mónada es un amplificador de tipos, que sigue ciertas reglas y ofrece ciertas operaciones
 - Amplificador de tipos
 - Coge un tipo y le agrega funcionalidad (Nullable, IEnumerable-...)
 - Reglas
 - Siguen las reglas de la composición funcional*
 - Operaciones
 - Identidad: $(T \Rightarrow \text{Algo}\langle T \rangle)$
 - Bind: $(\text{Algo}\langle T \rangle, (T, \text{Algo}\langle Y \rangle)) \Rightarrow \text{Algo}\langle Y \rangle$
 - Dado un tipo amplificado podemos recuperar el valor

Construcciones "monads"

- Built In:
 - `IEnumerable<T>` + LINQ
 - `Task<T>`
 - `Nullable<T>`
 - `IOption<T>`
- `Maybe<T>` or `Result<T>`
- `Either<TError, TSucces>` o `Result<TSuccess, TError>`



Maybe y Either

- `Maybe<T>` encapsula un valor que puede o no puede existir, similar a `Nullable<T>`
- `Either<L,R>` encapsula dos possible valores de los que unicamente exisitirá uno de ellos

Talk is cheap.
Show me the
code

- Leer de Excel y extraer la celda A1 de la hoja Hoja2
 - Existe el fichero
 - Es un Excel
 - Existe la hoja 2
 - Existe la celda A1
- Ejemplo llamada a APIs:
 - Saber si la llamada a terminado bien o mal
 - Controlar los posibles errores que devuelve la api sin utilizar excepciones

Conclusiones

Como empezar

Piensa en los métodos de Linq sustituyendo el IEnumerable

Familiarízate con los generics

Identifica lo que ya existe (IEnumerable, Task, Promesas en JS)

Poco a poco, piensa en implementar $f(x)$ puras

Utiliza funciones como variables o como parámetros

Introduce algunos elementos sencillos tipo Tuplas, Maybe o Either

Identifica estructuras de control repetitivas

Empieza a hacer composición de funciones

Conclusiones



Referencias

- [functional programming - Monad in plain English? \(For the OOP programmer with no FP background\) - Stack Overflow](#)
- Videos:
 - [Por qué deberías aprender programación funcional ya mismo – YouTube](#)
 - [Programación funcional: Próximamente en un lenguaje de programación cerca de usted - YouTube](#)
 - [Functional Programming in C# - YouTube \(Simon Painter\)](#)
 - [Functional Techniques for C# - Kathleen Dollard – YouTube](#)
 - [Get value out of your monad - Mark Seemann - YouTube](#)
- Blogs:
 - [Enterprise Craftsmanship](#)
- Librerías:
 - [vkhorikov/CSharpFunctionalExtensions: Functional extensions for C# \(github.com\)](#)
 - [GitHub - louthy/language-ext: C# functional language extensions - a base class library for functional programming](#)