

Здравствуйте, меня зовут Роман и я  
люблю функциональное  
программирование

Павел 13:53

А вот эти ваши функциональные эфшарпы вообще реально в продакшене использовать?

**Павел** 13:53

А вот эти ваши функциональные эфшарпы вообще реально в продакшене использовать?

**Владимир** 18:20

это все интересно выглядит, конечно, но менеджер меня не поймет

**Павел** 13:53

А вот эти ваши функциональные эфшарпы вообще реально в продакшене использовать?

**Владимир** 18:20

это все интересно выглядит, конечно, но менеджер меня не поймет

**Игорь** 19:11

да ну фигня какаято!



# REMEMBER

# ONLY YOU

can prevent  
object-oriented  
programming

```
var adminNames = users
    .Where(x => x.Group == Group.Admin)
    .Select(x => x.Name);
```

```
var adminNames = users  
    .Where(x => x.Group == Group.Admin)  
    .Select(x => x.Name);
```

- **Иммутабельность** - в результате операции мы не изменяем существующий объект, а создаем новый.

```
var adminNames = users
    .Where(x => x.Group == Group.Admin)
    .Select(x => x.Name);
```

- **Иммутабельность** - в результате операции мы не изменяем существующий объект, а создаем новый.
- **Функции первого порядка** - LINQ-методы принимают в качестве параметров функции.



```
var adminNames = users
    .Where(x => x.Group == Group.Admin)
    .Select(x => x.Name);
```

- **Иммутабельность** - в результате операции мы не изменяем существующий объект, а создаем новый.
- **Функции первого порядка** - LINQ-методы принимают в качестве параметров функции.
- **Прозрачность** - при вызове методов с одинаковыми данными мы получаем одинаковый результат.

Получаем пользу от  
параметризации действий

# Функциональная «Стратегия»

```
public interface ICalculator
```

```
{
```

```
    1 reference
```

```
    int Calculate(int value);
```

```
}
```

```
public class StuffMaker
```

```
{
```

```
    public ICalculator Calculator;
```

```
    0 references
```

```
    public StuffMaker(ICalculator calculator)
```

```
    {
```

```
        Calculator = calculator;
```

```
    }
```

```
    0 references
```

```
    public int MakeStuff(int value)
```

```
    {
```

```
        return Calculator.Calculate(value);
```

```
    }
```

```
}
```

А как бы вы реализовали это в функциональном стиле?

```
public static int MakeStuff(int value, Func<int, int> calculator)
{
    return calculator(value);
}
```

А существуют какие-нибудь паттерны функционального программирования?

## **OO pattern/principle**

- Single Responsibility Principle
- Open/Closed principle
- Dependency Inversion Principle
- Interface Segregation Principle
- Factory pattern
- Strategy pattern
- Decorator pattern
- Visitor pattern

## **FP pattern/principle**

## **OO pattern/principle**

- Single Responsibility Principle
- Open/Closed principle
- Dependency Inversion Principle
- Interface Segregation Principle
- Factory pattern
- Strategy pattern
- Decorator pattern
- Visitor pattern

## **FP pattern/principle**

- Functions



## **OO pattern/principle**

- Single Responsibility Principle
- Open/Closed principle
- Dependency Inversion Principle
- Interface Segregation Principle
- Factory pattern
- Strategy pattern
- Decorator pattern
- Visitor pattern

## **FP pattern/principle**

- Functions
- Functions

## **OO pattern/principle**

- Single Responsibility Principle
- Open/Closed principle
- Dependency Inversion Principle
- Interface Segregation Principle
- Factory pattern
- Strategy pattern
- Decorator pattern
- Visitor pattern

## **FP pattern/principle**

- Functions
- Functions
- Functions, also

## **OO pattern/principle**

- Single Responsibility Principle
- Open/Closed principle
- Dependency Inversion Principle
- Interface Segregation Principle
- Factory pattern
- Strategy pattern
- Decorator pattern
- Visitor pattern

## **FP pattern/principle**

- Functions
- Functions
- Functions, also
- Functions

## **OO pattern/principle**

- Single Responsibility Principle
- Open/Closed principle
- Dependency Inversion Principle
- Interface Segregation Principle
- Factory pattern
- Strategy pattern
- Decorator pattern
- Visitor pattern

## **FP pattern/principle**

- Functions
- Functions
- Functions, also
- Functions
- Yes, functions

## **OO pattern/principle**

- Single Responsibility Principle
- Open/Closed principle
- Dependency Inversion Principle
- Interface Segregation Principle
- Factory pattern
- Strategy pattern
- Decorator pattern
- Visitor pattern

## **FP pattern/principle**

- Functions
- Functions
- Functions, also
- Functions
- Yes, functions
- Oh my, functions again!

## **OO pattern/principle**

- Single Responsibility Principle
- Open/Closed principle
- Dependency Inversion Principle
- Interface Segregation Principle
- Factory pattern
- Strategy pattern
- Decorator pattern
- Visitor pattern

## **FP pattern/principle**

- Functions
- Functions
- Functions, also
- Functions
- Yes, functions
- Oh my, functions again!
- Functions

## **OO pattern/principle**

- Single Responsibility Principle
- Open/Closed principle
- Dependency Inversion Principle
- Interface Segregation Principle
- Factory pattern
- Strategy pattern
- Decorator pattern
- Visitor pattern

## **FP pattern/principle**

- Functions
- Functions
- Functions, also
- Functions
- Yes, functions
- Oh my, functions again!
- Functions
- Functions 😊

## **OO pattern/principle**

- Single Responsibility Principle
- Open/Closed principle
- Dependency Inversion Principle
- Interface Segregation Principle
- Factory pattern
- Strategy pattern
- Decorator pattern
- Visitor pattern

## **FP pattern/principle**

- Functions
- Functions
- Functions, also
- Functions
- Yes, functions
- Oh my, functions again!
- Functions
- Functions 😊

*Seriously, FP patterns are different*



Учим сигнатуры говорить правду,  
только правду и ничего кроме правды!

```
public static List<User> ListUsers()
```

Вопрос : что возвращает эта функция?

```
public static List<User> ListUsers()
```

Вопрос : что возвращает эта функция?

Ответ : список пользователей, конечно!

```
public static List<User> ListUsers()
```

Вопрос : что возвращает эта функция?

Ответ : список пользователей, конечно!  
или null...

```
public static List<User> ListUsers()
```

Вопрос : что возвращает эта функция?

Ответ : список пользователей, конечно!  
или null...  
а может упасть исключение...

## Полнота функций : Maybe (оно же Option)

```
public class Maybe<T> where T : class
{
    3 references
    public T Value { get; private set; }
    0 references
    public bool HasValue => Value != null;
    1 reference
    public Maybe(T someValue) {
        if (someValue == null)
            throw new ArgumentNullException(nameof(someValue));
        this.Value = someValue;
    }
    1 reference
    private Maybe() { }
    0 references
    public static Maybe<T> None() => new Maybe<T>();
}
```

## Полнота функций : Maybe (оно же Option)

```
var result = ListUsers();

if (result.HasValue)
{
    var users = result.Value;
    // process users
}
else
{
    // log error
}
```

## Полнота функций : класс Result

```
public class Result<T> {  
    0 references  
    public T Value { get; set; }  
    1 reference  
    public ErrorType Error { get; set; }  
    0 references  
    public bool IsSuccess => Error == ErrorType.None;  
}
```



## Полнота функций : класс Result

```
var result = ListUsers();

if (result.IsSuccess)
{
    var users = result.Value;
    // process users
}
else if (result.Error == ErrorType.DatabaseError)
{
    // log error
}
```

Боремся за простоту,  
последовательность ~~и мир во всем мире~~


## Цепочка продолжений

```
public User Sample(string input)
{
    var a = MakeStuff(input);
    if (a != null)
    {
        var b = MakeOtherStuff(a);
        if (b != null)
        {
            return MakeMoreStuff(b);
        }
        else return null;
    }
    else return null;
}
```

# Цепочка продолжений

```
public User Sample(string input)
{
    var a = MakeStuff(input);
    if (a != null)
    {
        var b = MakeOtherStuff(a);
        if (b != null)
        {
            return MakeMoreStuff(b);
        }
        else return null;
    }
    else return null;
}
```

```
function register()
{
    if (!empty($_POST)) {
        $msg = '';
        if ($_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;
    }
    return register_form();
}
```



pikaburu

```
if (input != null) {  
    // process data  
}  
else {  
    // return none  
}
```

## Цепочка продолжений

```
if (input != null) {  
    // process data  
}  
else {  
    // return none  
}
```

```
public R Bind<T,R>(Func<T,R> nextFunction, T input)  
{  
    if (input != null)  
    {  
        return nextFunction(input);  
    }  
    else return null;  
}
```

```
public User Sample(string input)
{
    return input
        .Bind(MakeStuff)
        .Bind(MakeOtherStuff)
        .Bind(MakeMoreStuff);
}
```

## Цепочка продолжений : снова Result

```
public static Result<R> Bind<T,R>(Func<Result<T>,Result<R>> nextFunction,  
    Result<T> input) where R : class  
{  
    if (input.IsSuccess)  
    {  
        return nextFunction(input);  
    }  
    else return Result<R>.Fail(input.Error);  
}
```



Делаем Result немного полезней, а обработку ошибок - проще и понятнее.

## Продолжения во благо валидации

---

```
public string UpdateUser(Request request)
{
    ValidateRequest(request);
    FormatPhoneNumber(request);
    db.updateDbFromRequest(request);
    smsService.sendMessage(request.Message);

    return "OK";
}
```

## Продолжения во благо валидации

---

```
public string UpdateUser(Request request)
{
    var validationResult = ValidateRequest(request);
    if (!validationResult)
    {
        return "BAD";
    }
    FormatPhoneNumber(request);
    db.updateDbFromRequest(request);
    smsService.sendMessage(request.Message);

    return "OK";
}
```

## Продолжения во благо валидации

---

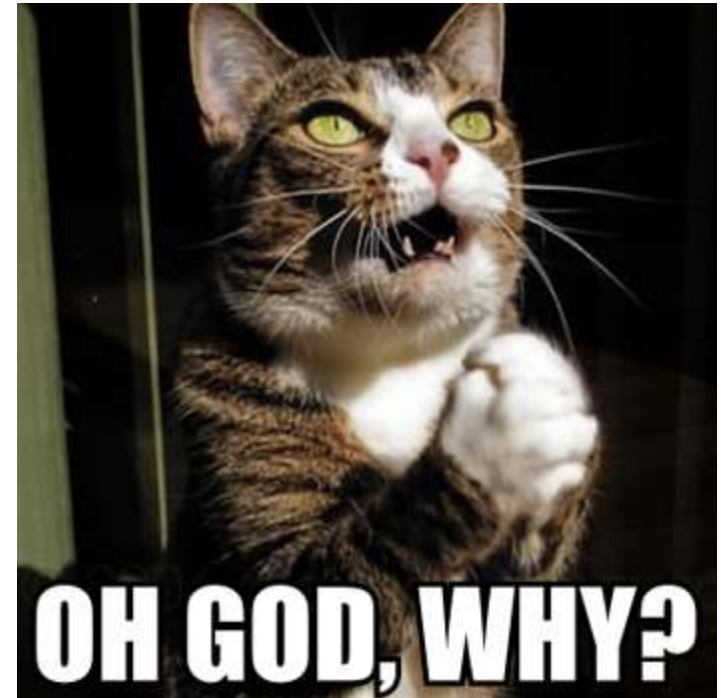
```
public string UpdateUser(Request request)
{
    var validationResult = ValidateRequest(request);
    if (!validationResult)
    {
        return "Validation isn't passed";
    }
    FormatPhoneNumber(request);
    var updateResult = db.updateDbFromRequest(request);
    if (!updateResult.Success)
    {
        return "Record can't be updated";
    }
    smsService.sendMessage(request.Message);

    return "OK";
}
```

## Продолжения во благо валидации

```
public string UpdateUser(Request request)
{
    var validationResult = ValidateRequest(request);
    if (!validationResult)
    {
        return "Validation isn't passed";
    }
    FormatPhoneNumber(request);
    try
    {
        var updateResult = db.updateDbFromRequest(request);
        if (!updateResult.Success)
        {
            return "Record can't be updated";
        }
    }
    catch (DatabaseUpdateException e)
    {
        return e.Message;
    }
    if (!smsService.sendMessage(request.Message))
    {
        logger.LogError("Message was not send");
    }

    return "OK";
}
```



```
public Result<User> UpdateUser(Request request)
{
    return ValidateRequest(request)
        .Bind(FormatPhoneNumber)
        .Bind(UpdateDbFromRequest)
        .Bind(SendMessage)
        .Bind(ReturnMessage);
}
```

А если жуть как хочется добавить,  
например, логгирование?

```
public Result<User> UpdateUser(Request request)
{
    return ValidateRequest(request)
        .Bind(FormatPhoneNumber)
        .Bind(Log)
        .Bind(UpdateDbFromRequest)
        .Bind(SendMessage)
        .Bind(ReturnMessage);
}
```





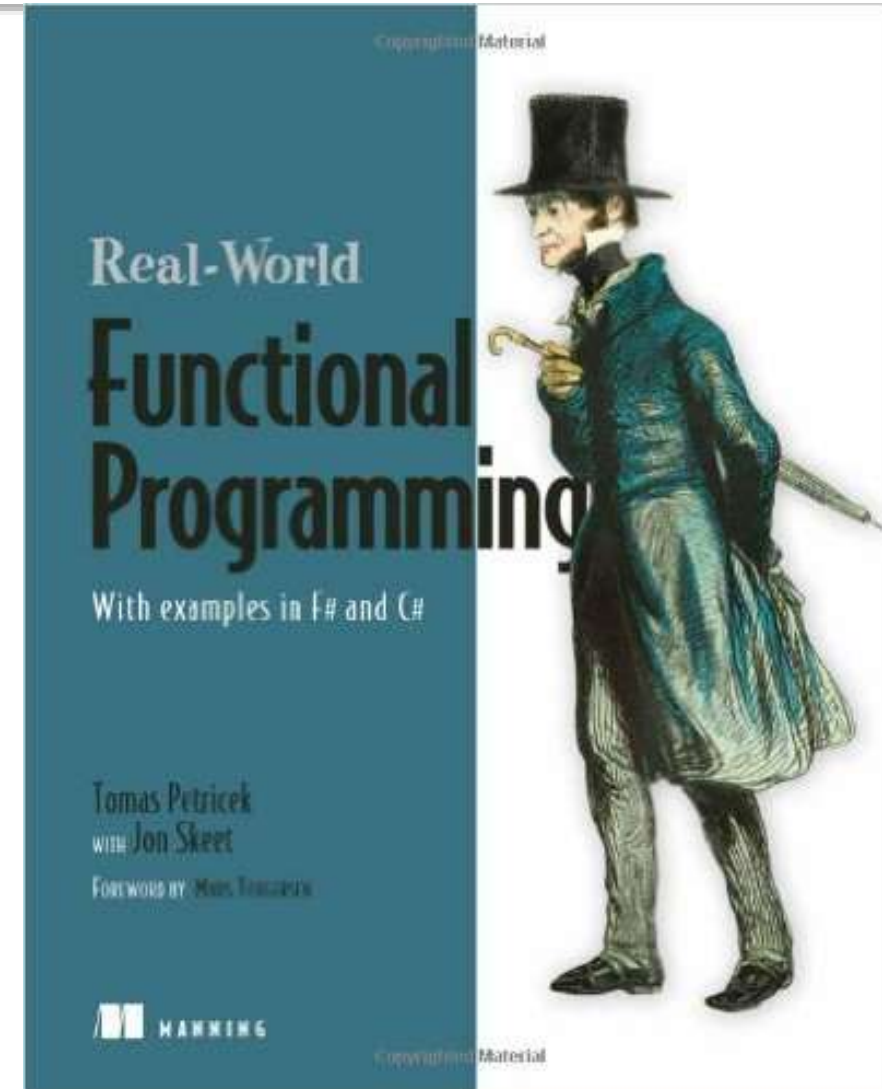
<https://github.com/louthy/language-ext>

# Что бы такого почитать про эту вашу функциональщину?

**Tomas Petricek & Jon Skeet**

Real-World Functional Programming: With  
Examples in F# and C#

[a.co/3wDvJVva](https://a.co/3wDvJVva)



Но вообще, это все - только начало.  
Когда-нибудь мы с вами посмотрим на  
еще более веселые штуки.

Спасибо за внимание!

✉ nevoroman@gmail.com

📧 nevoroman

vk nevoroman