

```
let talk = {  
    theme = "Функциональный .NET";  
    speaker = "Роман Неволин";  
    company = "ЕРАМ";  
    event = "SPB .NET Meetup #14";  
}
```

Сегодня будет

Много функционального кода.

Сегодня будет

Много функционального кода.

Много простого функционального кода.

Сегодня будет

Много функционального кода.

Много простого функционального кода.

Много простого функционального кода на слайдах.

Сегодня будет

Много функционального кода.

Много простого функционального кода.

Много простого функционального кода на слайдах.

Естественно, монады (иначе меня просто не поймут).

# Столпы функционального подхода

**Функция** — это объект (первого класса)

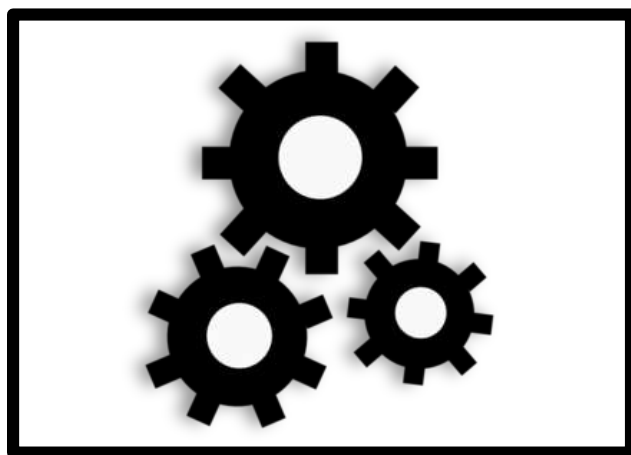
**Композиция функций.** Много композиций.

**Типы** вместо классов

**Неизменяемое состояние** (Immutable state)

Функция — это объект

Функция — это объект



Tree -> Firewood



ФУНКЦИЯ — ЭТО ОБЪЕКТ

String



Integer -> String



Integer

Employee



Employee -> SalaryReport



SalaryReport

HttpRequest  
t



HttpRequest ->  
HttpResponse



HttpResponse

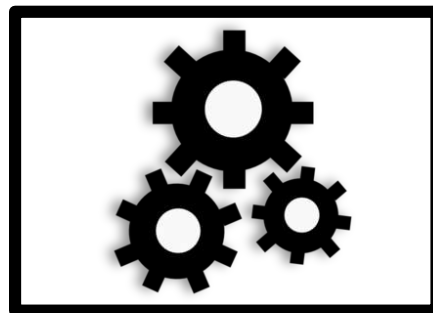
ФУНКЦИЯ — ЭТО ОБЪЕКТ

let num = 7

```
val num : int = 7
```

let add x y = x + y

```
val add : x:int -> y:int -> int
```



int -> int -> int

Функция как результат или параметр

```
let multiplyTo x = (fun y -> x * y)
```

```
val multiplyTo : x:int -> y:int -> int
```

```
let transform f x = (f x) * 2
```

```
val transform : f:('a -> int) -> x:'a -> int
```

Ода оператору pipe-forward

|>

Ода оператору pipe-forward

Input |> function  
function input

Ода оператору pipe-forward

Input |> function  
function input

```
let (|>) x f = f x
```

# Ода оператору pipe-forward

Input |> function  
function input

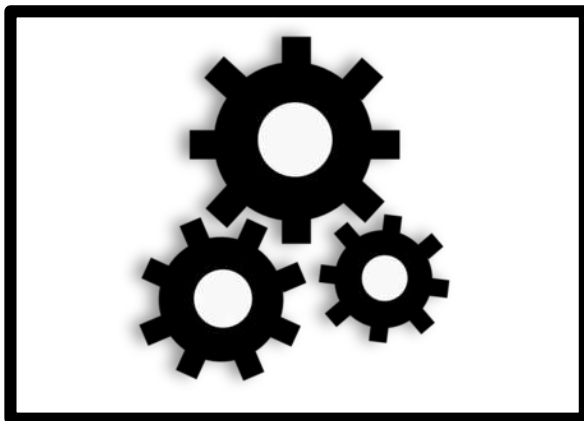
let (|>) x f = f x

```
let topWords =  
  tweets.Rows  
  |> Seq.map(fun x -> ClearMessage (x.Text.ToLower()))  
  |> Seq.map(fun x -> x.Split(' '))  
  |> Seq.concat  
  |> Seq.filter(fun x -> x.Length > 2 && not (stopwords |> Array.contains(x)))  
  |> Seq.groupBy(fun x -> x)  
  |> Seq.sortByDescending(fun (_,x) -> (x |> Seq.length))  
  |> Seq.take(100)  
  |> Seq.map(fun (x,y) -> x, Seq.length y)  
  |> Seq.toArray
```

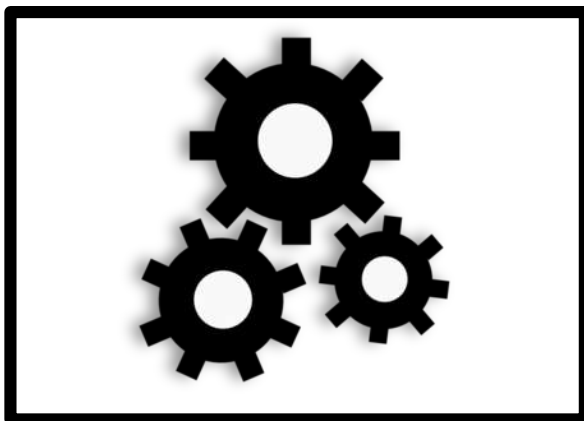
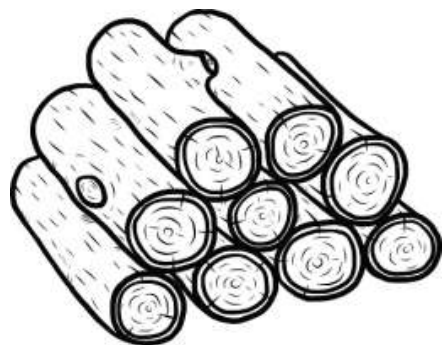
Композиция. Повсюду.



# Композиция. Повсюду.



Tree -> Firewood



Firewood -> Home



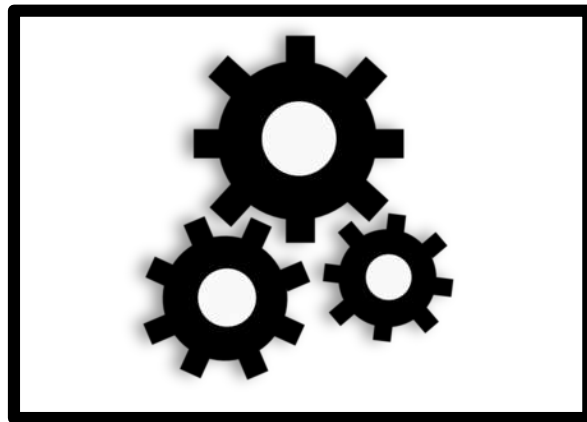
# Композиция. Повсюду.

Tree -> Firewood

>

Firewood -> Home

>



Tree -> Home

Плохая новость — это работает только для тех функций, которые принимают один параметр ☹

Плохая новость — это работает только для тех функций, которые принимают один параметр ☹️

Хорошая новость — все функции в F# принимают один параметр! 😊

# Каррирование

```
let add x y = x + y
```

```
val add : x:int -> y:int -> int
```

# Каррирование

```
let add x y = x + y
```

```
val add : x:int -> y:int -> int
```

```
let add2 = add 2
```

```
val add2 : (int -> int)
```

# Каррирование

```
let add x y = x + y
```

```
val add : x:int -> y:int -> int
```

```
let add2 = add 2
```

```
val add2 : (int -> int)
```

```
let additionResult = add2 5
```

```
val additionResult : int = 7
```

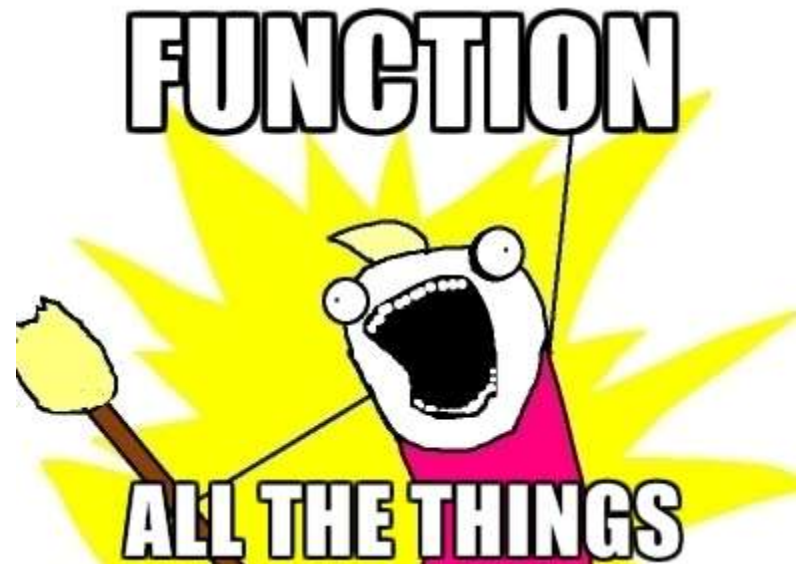
Функции в малом,  
объекты в большом



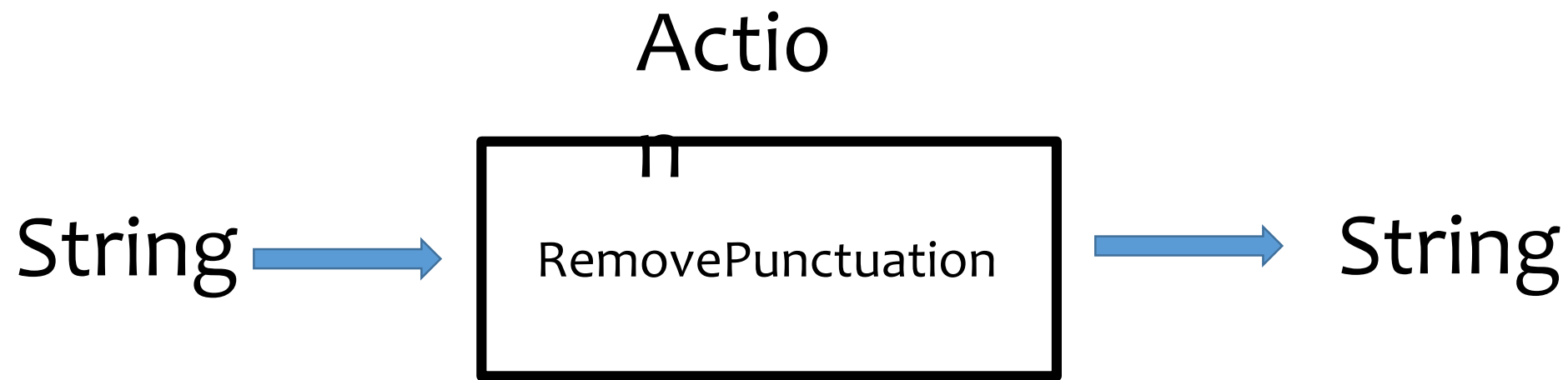
Функции в малом,  
~~объекты~~ в большом

Функции в малом,  
~~объекты~~ в большом  
функции

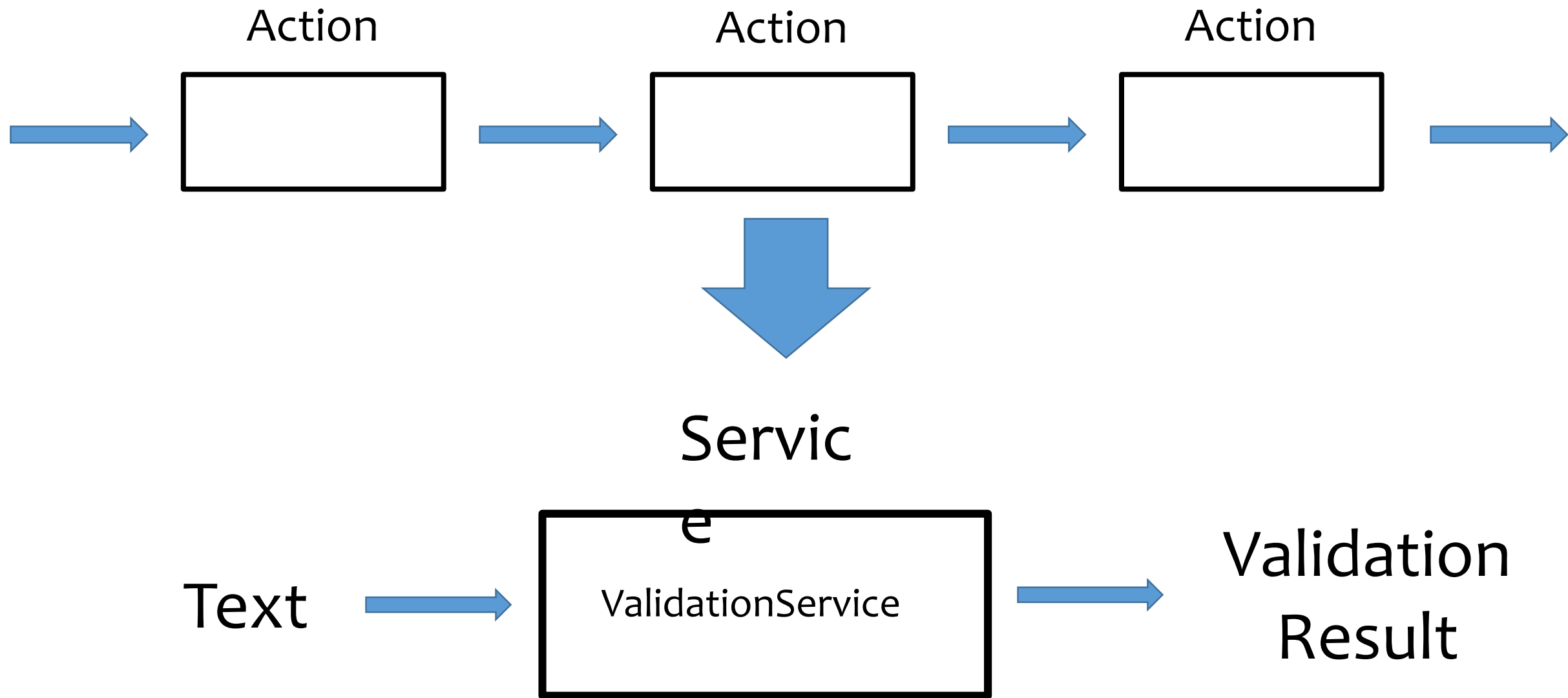
Функции в малом,  
~~объекты~~ в большом  
функции



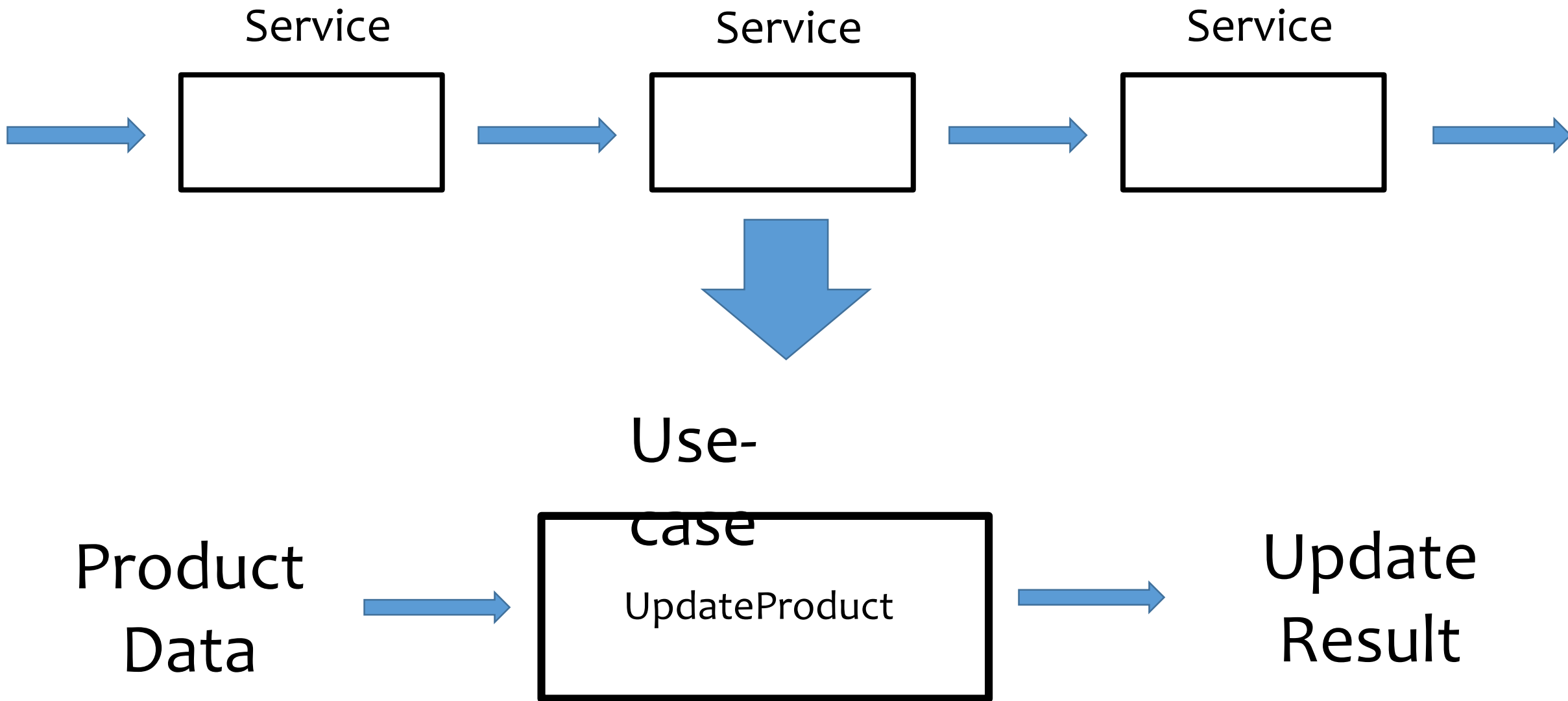
Композиция. Повсюду.



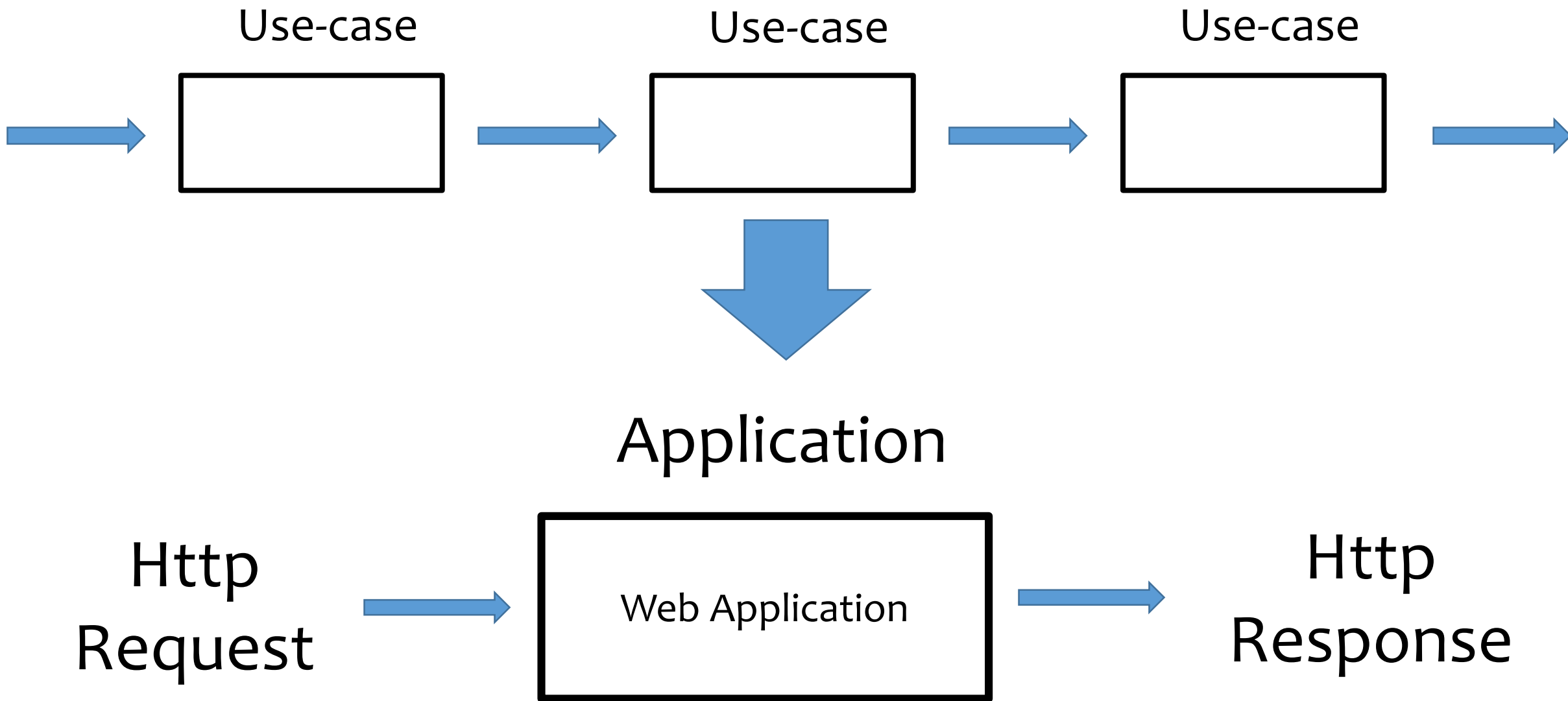
# Композиция. Повсюду.



# Композиция. Повсюду.



# Композиция. Повсюду.



Типы вместо классов



# Типы вместо классов

Integer  Is a type

# ТИПЫ ВМЕСТО КЛАССОВ

Id : Integer

Name : String

User  Is a type

# ТИПЫ ВМЕСТО КЛАССОВ

Id : Integer

Name : String

User  Is a type

## Производные типы

User X DateTime = Birthdate  
(User\*DateTime)

# ТИПЫ ВМЕСТО КЛАССОВ

Id : Integer

Name : String

User  Is a type

## ПРОИЗВОДНЫЕ ТИПЫ

User X DateTime = Birthdate

(User\*DateTime)

User + Employee =

| User of Id\*Name

| Employee of Id \* Position

Здесь должен быть долгий  
рассказ о том, как прекрасна  
функциональщина...

Но лучше один раз увидеть!

Параметризируем поведение

# Параметризируем поведение

```
let processList() =  
  for i in [1..10] do  
    printfn "The number is %i" i
```



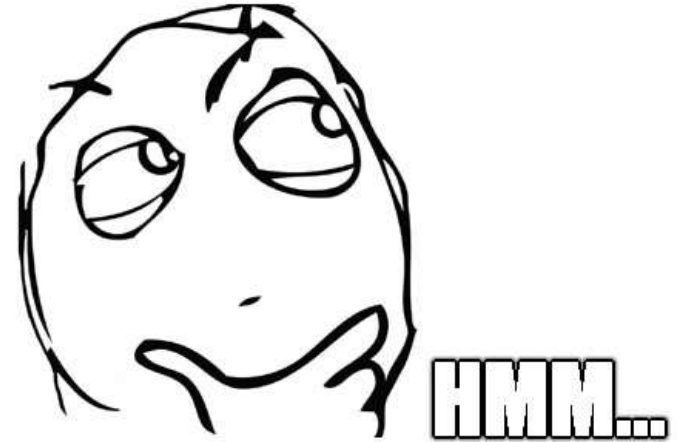
# Параметризируем поведение

```
let processList() =  
  for i in [1..10] do  
    printfn "The number is %i" i
```

→ Захардкоженные данные — это плохо!

# Параметризируем поведение

```
let processList list =  
  for i in list do  
    printfn "The number is %i" i
```



# Параметризируем поведение

```
let processList list action =  
    for i in list do  
        action i
```

На самом деле, мы только что реализовали одну из базовых функций F# - Seq.iter

# Параметризируем поведение. Опять.

ООП (на самом деле,  
императивная) версия

```
public static int Product(int n)
{
    int product = 1;
    for (int i = 1; i <= n; i++)
    {
        product *= i;
    }
    return product;
}
```

```
public static int Sum(int n)
{
    int sum = 0;
    for (int i = 1; i <= n; i++)
    {
        sum += i;
    }
    return sum;
}
```

# Параметризируем поведение. Опять.

ООП (на самом деле,  
императивная) версия

```
public static int Product(int n)
{
    int product = 1;
    for (int i = 1; i <= n; i++)
    {
        product *= i;
    }
    return product;
}
```

```
public static int Sum(int n)
{
    int sum = 0;
    for (int i = 1; i <= n; i++)
    {
        sum += i;
    }
    return sum;
}
```

Функциональная  
версия

```
let product n =
    let initialValue = 1
    let action productSoFar x = productSoFar * x
    [1..n] |> List.fold action initialValue

let sum n =
    let initialValue = 0
    let action sumSoFar x = sumSoFar+x
    [1..n] |> List.fold action initialValue
```

# Параметризируем поведение. Опять.

ООП (на самом деле,  
императивная) версия

```
public static int Product(int n)
{
    int product = 1;
    for (int i = 1; i <= n; i++)
    {
        product *= i;
    }
    return product;
}
```

```
public static int Sum(int n)
{
    int sum = 0;
    for (int i = 1; i <= n; i++)
    {
        sum += i;
    }
    return sum;
}
```

Функциональная  
версия

```
let product n =
    let initialValue = 1
    [1..n] |> List.fold (*) initialValue

let sum n =
    let initialValue = 0
    [1..n] |> List.fold (+) initialValue
```

Функции вместо интерфейсов

# Функции вместо интерфейсов

```
interface ICalculator
{
    int Calculate(int value);
}

public class StuffMaker
{
    private readonly ICalculator _calculator;
    public StuffMaker(ICalculator calculator)
    {
        _calculator = calculator;
    }

    public int MakeStuff(int value)
    {
        return _calculator.Calculate(value);
    }
}
```



# Функции вместо интерфейсов

```
interface ICalculator
{
    int Calculate(int value);
}

public class StuffMaker
{
    private readonly ICalculator _calculator;
    public StuffMaker(ICalculator calculator)
    {
        _calculator = calculator;
    }

    public int MakeStuff(int value)
    {
        return _calculator.Calculate(value);
    }
}
```

А что если я скажу тебе

Что классы не нужны?

```
let makeStuff value calculator =
    calculator value
```

# Функции вместо интерфейсов

```
interface ICalculator
{
    int Calculate(int value);
}

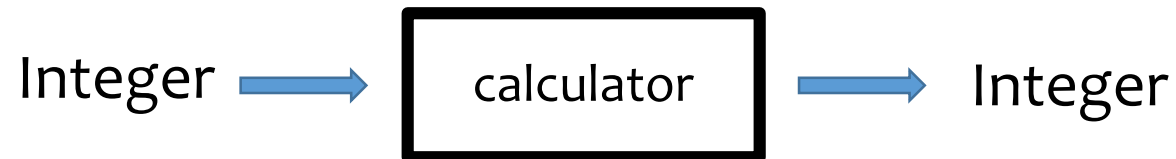
public class StuffMaker
{
    private readonly ICalculator _calculator;
    public StuffMaker(ICalculator calculator)
    {
        _calculator = calculator;
    }

    public int MakeStuff(int value)
    {
        return _calculator.Calculate(value);
    }
}
```

А что если я скажу тебе

Что классы не нужны?

```
let makeStuff value calculator =
    calculator value
```



Логгируем функционально

```
let add x y = x + y
```

Логгируем функционально

```
let add x y = x + y
```

```
let logger f input =
```

```
    let output = f input
```

```
    // Здесь мы как-то это логгируем  
    output
```

Логгируем функционально

```
let add x y = x + y
```

```
let logger f input =
```

```
    let output = f input
```

```
    // Здесь мы как-то это логгируем  
    output
```

```
let addWithLogging = logger add
```

# Внедряем зависимости

```
let logger logFun action input =  
    let output = action input  
    logFun input output  
    output
```

# Внедряем зависимости

```
let logger logFun action input =  
    let output = action input  
    logFun input output  
    output
```

```
let basicLog input output =  
    printf "Input: %A\nOutput: %A" input output
```

# Внедряем зависимости

```
let logger logFun action input =  
    let output = action input  
    logFun input output  
    output
```

```
let basicLog input output =  
    printf "Input: %A\nOutput: %A" input output
```

```
let basicLogger = logger basicLog
```



# Внедряем зависимости

```
let logger logFun action input =  
    let output = action input  
    logFun input output  
    output
```

```
let basicLogger = logger (printf "Input: %A\nOutput: %A")
```

# Внедряем зависимости

```
let logger logFun action input =  
    let output = action input  
    logFun input output  
    output
```

```
let basicLogger = logger (printf "Input: %A\nOutput: %A")
```



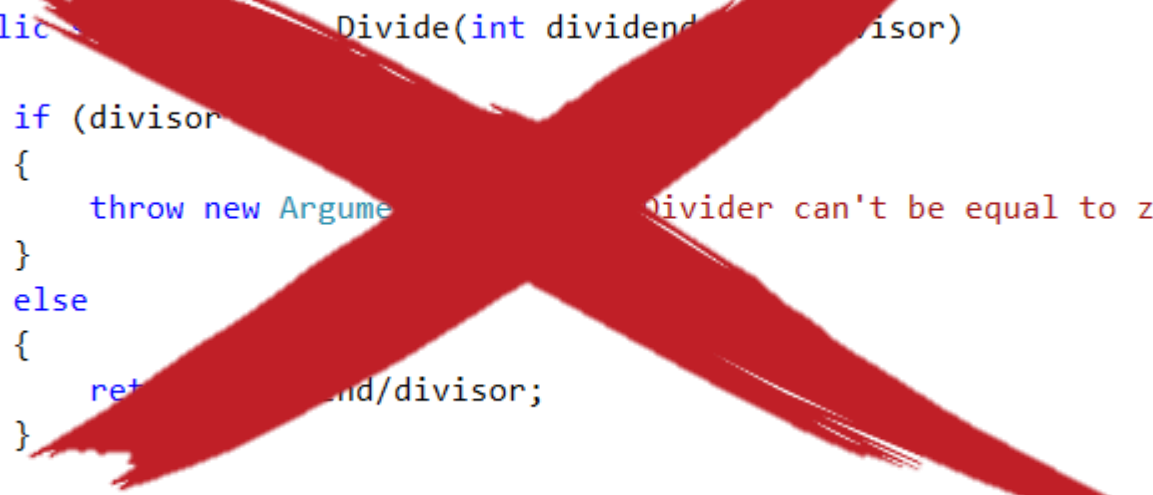
Продолжения  
Или «Голливудский принцип»

# Продолжения

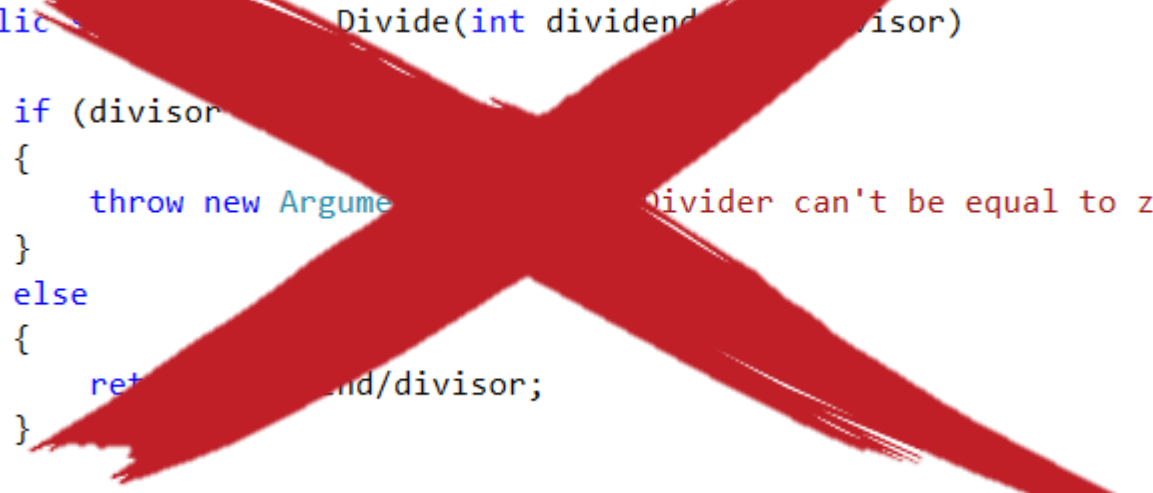
```
public static double Divide(int dividend, int divisor)
{
    if (divisor == 0)
    {
        throw new ArgumentException("Divider can't be equal to zero!");
    }
    else
    {
        return dividend/divisor;
    }
}
```

# Продолжения

```
public static Divide(int dividend, int divisor)
{
    if (divisor == 0)
    {
        throw new ArgumentException("Divider can't be equal to zero!");
    }
    else
    {
        return dividend/divisor;
    }
}
```



# Продолжения



```
public static Divide(int dividend, int divisor)
{
    if (divisor == 0)
    {
        throw new ArgumentException("Divisor can't be equal to zero!");
    }
    else
    {
        return dividend/divisor;
    }
}
```

Мы не хотим отправлять исключения,  
мы хотим обработать ошибку!

# Продолжения

```
let divide ifZero ifSuccess dividend divisor =  
  if (divisor = 0) then ifZero()  
  else ifSuccess (dividend / divisor)
```

# Продолжения

Серьезно, 4 параметра?!

```
let divide ifZero ifSuccess dividend divisor =  
  if (divisor = 0) then ifZero()  
  else ifSuccess (dividend / divisor)
```



# Продолжения

Серьезно, 4 параметра?!

```
let divide ifZero ifSuccess dividend divisor =  
  if (divisor = 0) then ifZero()  
  else ifSuccess (dividend / divisor)
```

```
let ifZero() = printfn "Div by 0"  
let ifSuccess = printfn "Div result : %d"
```

```
let printDivide = divide ifZero ifSuccess
```

```
val printDivide : (int -> int -> unit)
```

Цепочка продолжений  
Во славу Сатане

# Цепочка продолжений Во славу Сатане



Цепочка продолжений  
Во славу Са~~не~~  
добра



# Цепочка продолжений

```
let sample input =  
  let a = makeStuff input  
  if a <> null then  
    let b = makeOtherStuff a  
    if b <> null then  
      let c = makeMoreStuff b  
      if c <> null then  
        c  
      else null  
    else null  
  else null
```

# Цепочка продолжений

```
let sample input =  
  let a = makeStuff input  
  if a <> null then  
    let b = makeOtherStuff a  
    if b <> null then  
      let c = makeMoreStuff b  
      if c <> null then  
        c  
      else null  
    else null  
  else 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

# Цепочка продолжений

```
let sample input =  
  let a = makeStuff input  
  if a <> null then  
    let b = makeOtherStuff a  
    if b <> null then  
      let c = makeMoreStuff b  
      if c <> null then  
        c  
      else null  
    else null  
  else null
```

null — это плохой стиль!

Возможно, стоит...

# Цепочка продолжений

```
let sample input =  
  let a = makeStuff input  
  if a.IsSome then  
    let b = makeOtherStuff (a.Value)  
    if b.IsSome then  
      let c = makeMoreStuff (b.Value)  
      if c.IsSome then  
        c  
      else None  
    else None  
  else None
```



# Цепочка продолжений

```
let sample input =  
  let a = makeStuff input  
  if a.IsSome then  
    let b = makeOtherStuff (a.Value)  
    if b.IsSome then  
      let c = makeMoreStuff (b.Value)  
      if c.IsSome then  
        c  
      else None  
    else None  
  else None
```



# Цепочка продолжений

```
let sample input =  
  let a = makeStuff input  
  if a.IsSome then  
    let b = makeOtherStuff (a.Value)  
    if b.IsSome then  
      let c = makeMoreStuff (b.Value)  
      if c.IsSome then  
        c  
      else None  
    else None  
  else None
```



(на самом деле нет)

# Цепочка продолжений

```
if input.IsSome then
```

```
    // Обрабатываем значение
```

```
else
```

```
    None
```

```
let bind nextFunction input =
```

```
    match input with
```

```
    | Some s -> nextFunction s
```

```
    | None -> None
```

# Цепочка продолжений

```
let sample input =  
  input  
  |> bind makeStuff  
  |> bind makeOtherStuff  
  |> bind makeMoreStuff
```

# Цепочка продолжений

```
let sample input =  
  input  
  |> bind makeStuff  
  |> bind makeOtherStuff  
  |> bind makeMoreStuff
```



# Цепочка продолжений

```
let sample input =  
  input  
  |> bind makeStuff  
  |> bind makeOtherStuff  
  |> bind makeMoreStuff
```



```
let (>>=) x f = bind f x
```

```
let sample input =  
  input  
  >>= makeStuff  
  >>= makeOtherStuff  
  >>= makeMoreStuff
```

# Цепочка продолжений

```
let sample input =  
  input  
  |> bind makeStuff  
  |> bind makeOtherStuff  
  |> bind makeMoreStuff
```



```
let (>>=) x f = bind f x
```

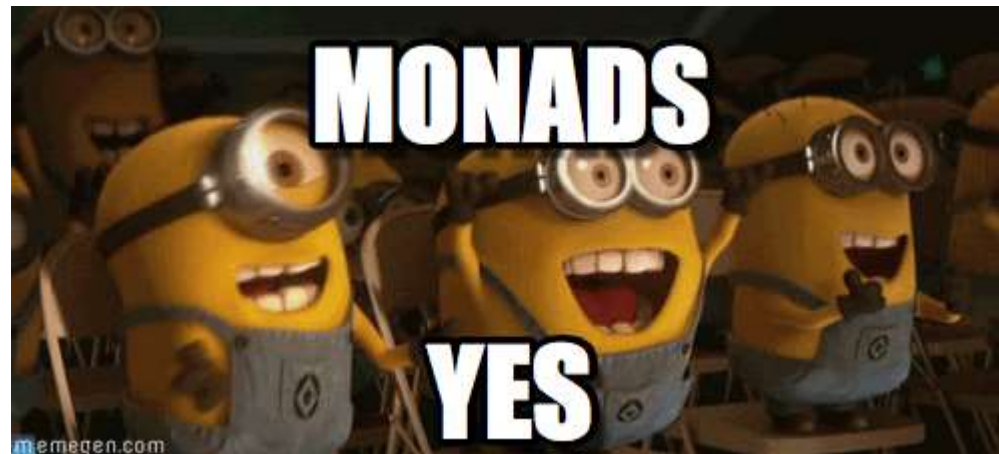
```
let sample input =  
  input  
  >>= makeStuff  
  >>= makeOtherStuff  
  >>= makeMoreStuff
```

```
let bindTask f (task:Task) =  
  task.ContinueWith (fun result -> f result)
```

Кстати, здесь были монады



Кстати, здесь были монады



# Что? Монады?

```
let bind nextFunction input =  
  match input with  
  | Some s -> nextFunction s  
  | None -> None
```

# Что? Монады?

```
let bind nextFunction input =  
  match input with  
  | Some s -> nextFunction s  
  | None -> None
```

Monadic Bind

# Монады на службе валидации

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

    return "OK";
}
```

# Монады на службе 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";
}
```

# Монады на службе валидации

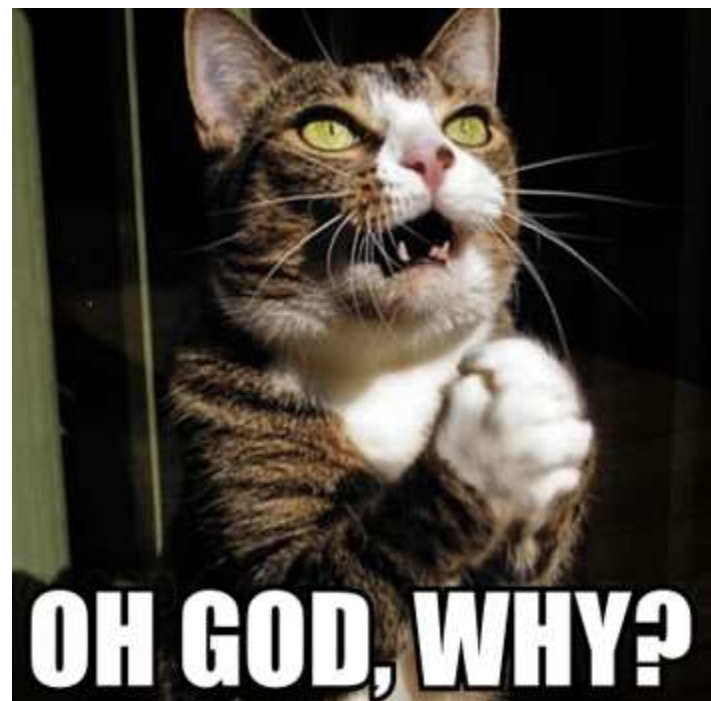
```
public string UpdateUser(Request request)
{
    var validationResult = ValidateRequest(request);
    if (!validationResult)
    {
        return "Validation isn't passed";
    }
    FormatPhoneNumber(request);
    var updateResult = db.updateDbFromRequest(request);
    if (!updateResult.Success)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

    return "OK";
}
```

# Монады на службе 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";
}
```



# Функциональный вариант : без валидации

```
let updateUser request =  
    request  
    |> validateRequest  
    |> formatPhoneNumber  
    |> updateDbFromRequest  
    |> sendMessage  
    |> returnMessage
```

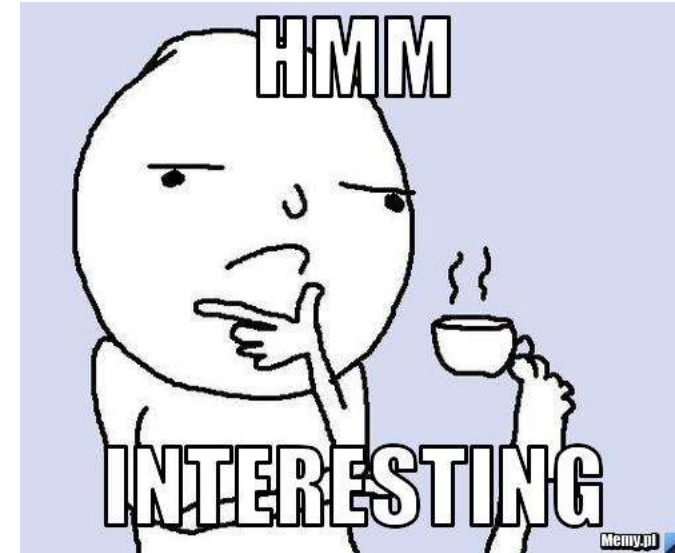


# Функциональный вариант : с валидацией

```
let updateUserValidated request =  
    request  
    |> validateRequest  
    >>= formatPhoneNumber  
    >>= updateDbFromRequest  
    >>= sendMessage  
    >>= returnMessage
```

# Функциональный вариант : с валидацией

```
let updateUserValidated request =  
  request  
  |> validateRequest  
  >>= formatPhoneNumber  
  >>= updateDbFromRequest  
  >>= sendMessage  
  >>= returnMessage
```

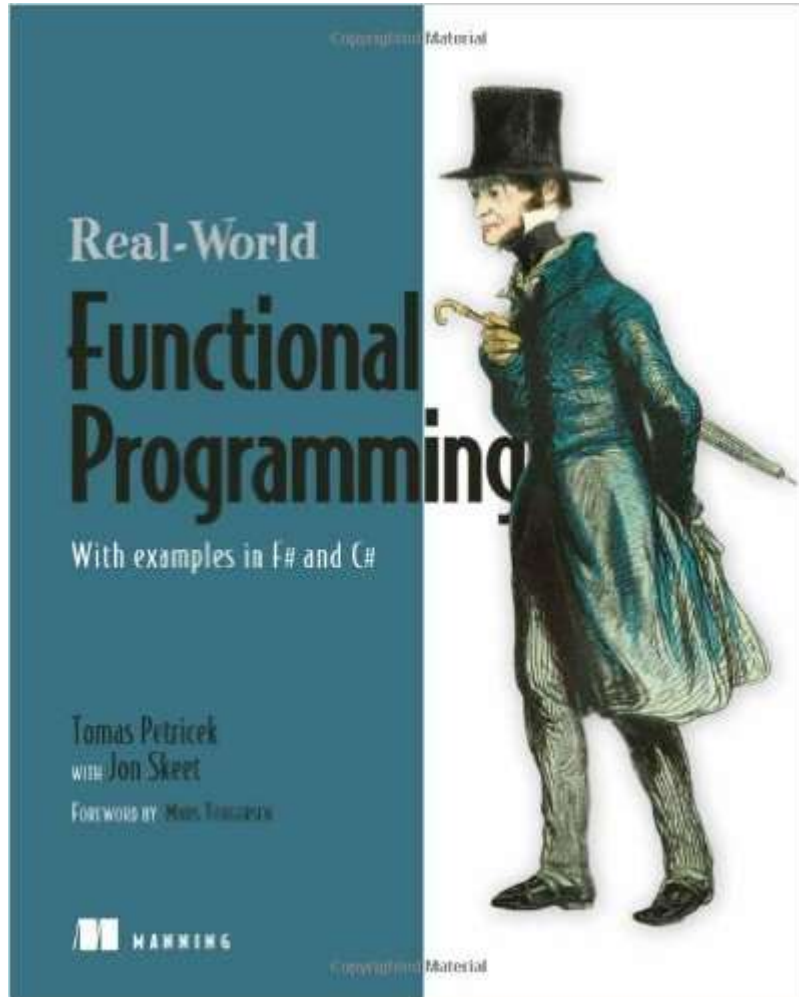


Функциональщина — это мощно

Функциональщина — это мощно\*

\* При разумном и своевременном использовании

# Хочу учить ЭТОТ ваш F#!



**Tomas Petricek**

Real-World Functional Programming

[amzn.com/1933988924](https://amzn.com/1933988924)

Хочу учить этот ваш F#!

**F# | >**

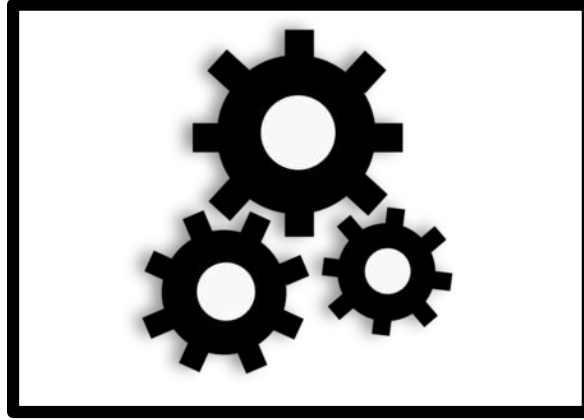
**Scott Wlaschin**

F# for Fun and Profit

**I** 

[fsharpforfunandprofit.com](http://fsharpforfunandprofit.com)

# Q



# A

Question -> Answer

```
let contacts = {  
  email = "nevoroman@gmail.com";  
  skype = "nevoroman";  
  github = "nevoroman"  
}
```