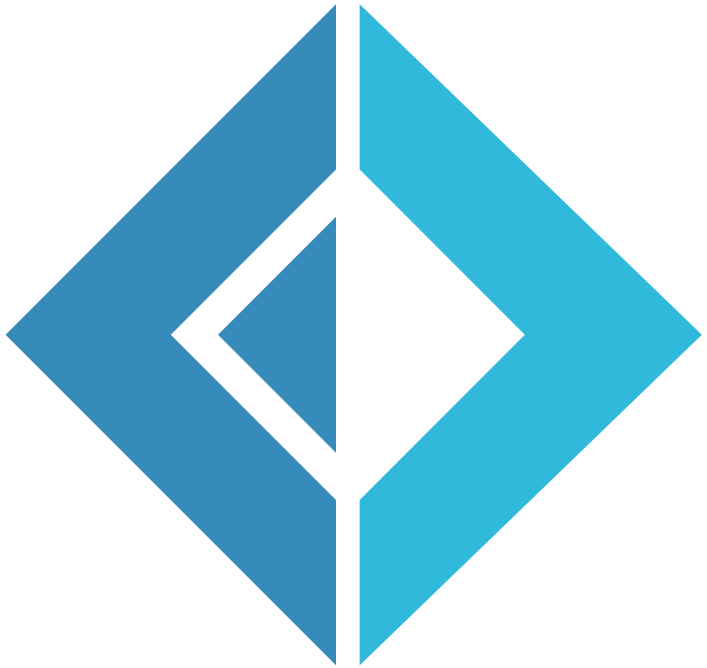
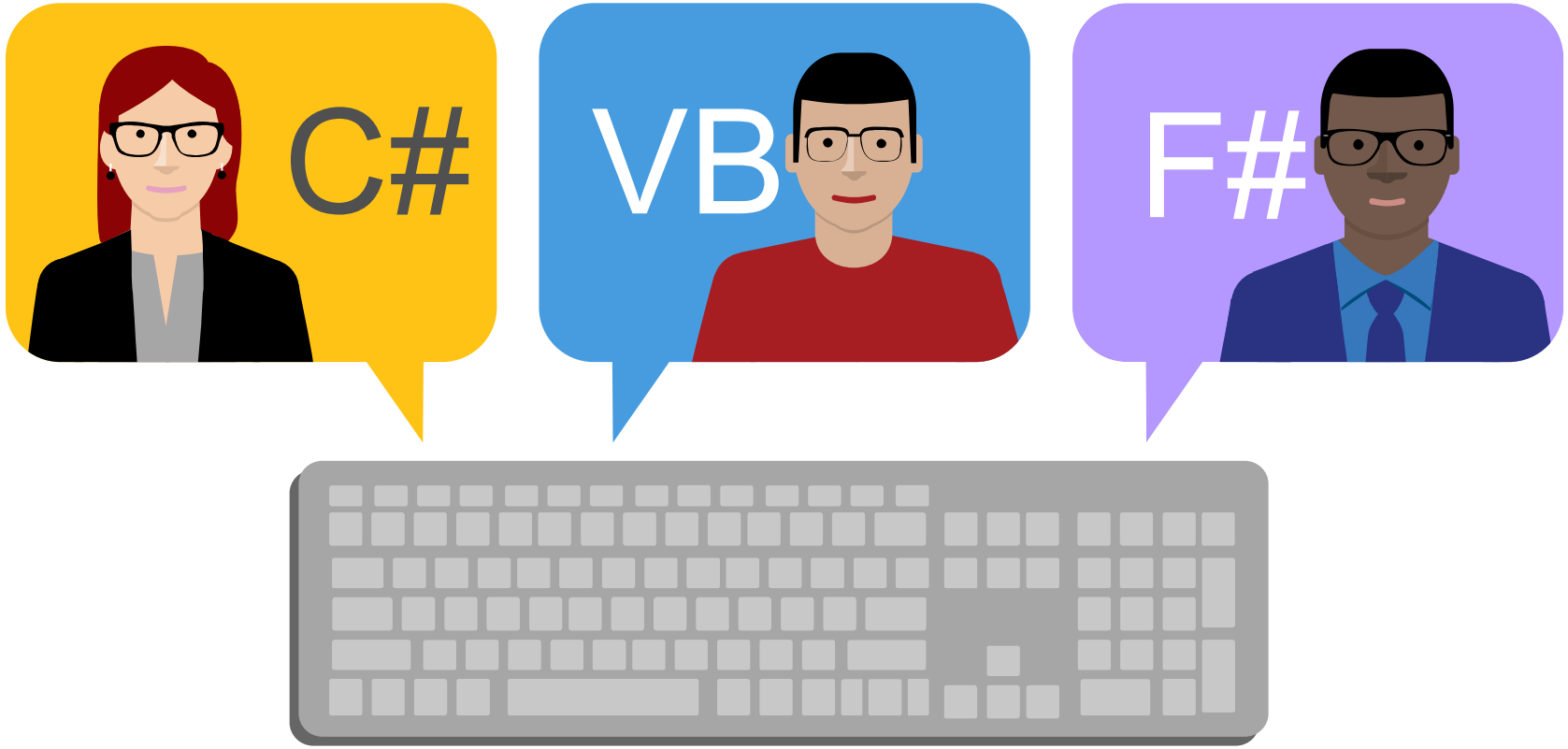


F#

Demo

Type Provider


C#
VB
F#

Что пишет Microsoft?

F# (pronounced "F sharp") is a cross-platform, open-source, functional programming language for .NET. It also includes object-oriented and imperative programming.

https://www.microsoft.com/net/learn/languages

Про синтаксис

1. Отступы вместо скобок
2. ~~`f(a, b, c)`~~ `f a b c`
3. `|>` - pipe operator

```
f3 (f2 (f1 x))
x |> f1 |> f2 |> f3
```

```
f a b c
c |> f a b
```

#1

Type Providers

Type Provider

```
type Weather = JsonProvider<"../weather.json">
```

Создаёт типы на основе информации, полученной компилятором из источника данных

Type Provider

Примеры:

- JSON
- XML
- CSV
- SQL

…

- R

#2

Discriminated Unions

Result.cs

```
public class None
{
    private None()
    {
    }
}

public struct Result<T>
{
    public Result(string error, T value = default(T))
    {
        Error = error;
        Value = value;
    }

    public bool IsSuccess => Error == null;
    public string Error { get; }
    internal T Value { get; }

    public T GetValueOrThrow() =>
        IsSuccess
            ? Value
            : throw new InvalidOperationException($"No value. Only Error {Error}");
}
```

Result.cs – что не так?

- Нужен тип None
- Дублирование ☹
- Нужны вспомогательные методы

```
public static Result<None> Ok()
public static Result<T> Ok<T>(T value)
public static Result<T> Fail<T>(string e)
```

- Можно написать

```
var fail = Result.Fail<int>("epic fail");
var value = fail.GetValueOrThrow();
```

Discriminated Union

```
type Result<'a> =
    | Ok of 'a
    | Fail of string

let ok = Ok 42
let fail = Fail "epic fail"
```

Demo

Discriminated Union

#3

Computation Expressions

Что общего?

- `yield return 1;`
- `await Task.Run(1000);`
-
```
var squares = from item in items
              select item * item;
```

Computation Expression: seq

```
let f() = seq {
    yield 1
    yield 2
    yield 3
}
```

Computation Expression: async

```
let f() = async {
    do! Async.Sleep 1000
}
```

Computation Expression: async

```
let fetchUrlAsync url = async {
    let req = WebRequest.Create(Uri(url))
    use! resp = req.AsyncGetResponse()
    use stream = resp.GetResponseStream()
    use reader = new IO.StreamReader(stream)
    let! html = reader.ReadToEndAsync() |> Async.AwaitTask
    printfn "finished downloading %s" url
    return html
}
```

Computation Expression: query

```
let items = [1;2;3]

let squares = query {
    for item in items do
    select (item * item)
}
```

Custom Computation Expression: result

```
let readFromDb id =
    if id < 8
    then Ok (id + 2)
    else Fail (sprintf "Value %d is not available" id)

let calculate() = result {
    let x = 3
    let! y = readFromDb x
    let! z = readFromDb (x + y)
    return x + y + z
}
```

#4

Records

Immutable class

```
class Song {
    public Song(string author, string name) {
        Author = author;
        Name = name;
    }
    public string Author { get; }
    public string Name { get; }
}
```

Immutable class

```
class Song : IEquatable<Song>
{
    public Song(string author, string name)
    {
        Author = author;
        Name = name;
    }

    public string Author { get; }
    public string Name { get; }

    public Song WithAuthor(string author) => new Song(author, Name);
    public Song WithName(string name) => new Song(Author, name);

    public override bool Equals(object obj) => Equals(obj as Song);

    public bool Equals(Song other) =>
        other != null &&
        Author == other.Author &&
        Name == other.Name;

    public override int GetHashCode() => HashCode.Combine(Author, Name);

    public override string ToString() =>
        $"Author: {Author}, Name: {Name}";
}
```

Record

```
type Song = { Author: string; Name: string }

let song1 = { Author="Queen"; Name="Bohemian Rhapsody" }
let song2 = { Author="Queen"; Name="Bohemian Rhapsody" }
printfn "%b" (song1=song2) // true

let song3 = { song2 with Name="We Are the Champions" }

let { Name=name } = song3 // name = song3.Name
printfn "%s" name // "We Are the Champions"
```

Инструменты

ionide.io

Где учить?

F# has plenty of strengths, many outlined on this outstanding website: F# for Fun and Profit

- из презентации Don Syme

- fsharpforfunandprofit.com

- fsharp.org/learn.html

Вопросы?

github.com/yevgeniyredko/shpora-2018-fsharp

email: r.e.s.1997@gmail.com

github/telegram/twitter/fb/vk: @yevgeniyredko