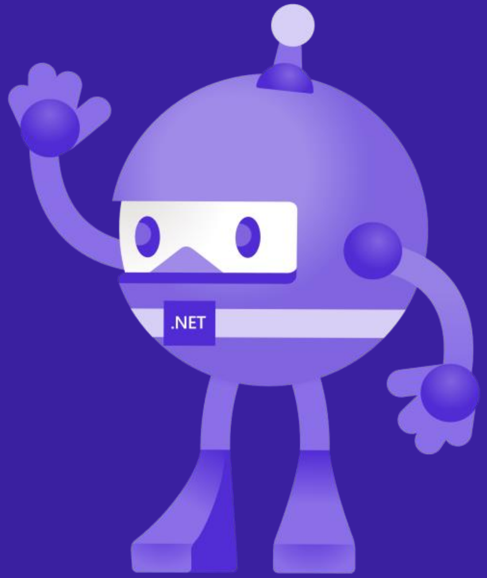
What’s new in F# 8

[Tomáš Grošup; Adam Boniecki; Petr Semkin;]

Microsoft -> DevDiv -> F# Compiler & Tools

Code available at T-Gro/FSharp8_news: Examples of new F#8 features (github.com)



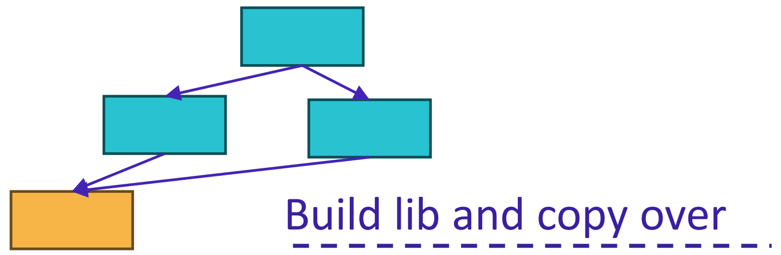
Use F#

- F# Software Foundation (fsharp.org)
- dotnet/fsharp ← code is here
- fsharp/fslang-**suggestions** ← + discussions
- fsharp/fslang-**design** ← RFCs here

Areas of F#8 improvements

- Fsharp.Core library additions
- Fsharp.Core performance optimizations
- Language features
- Diagnostics – new & reworked
- Improved support for .NET features – ref assemblies, trimming
- Bugfixes, compiler performance and much more

Compiler performance

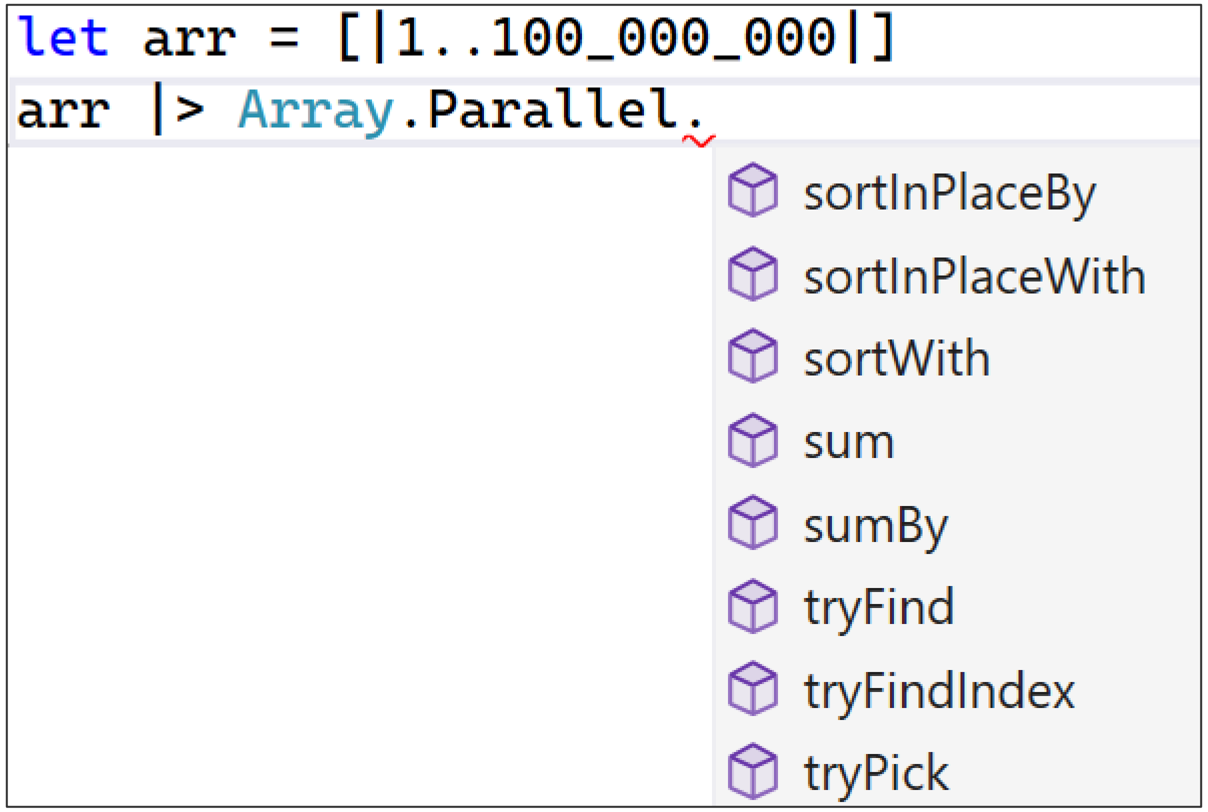


- Robustness of reference assemblies (reducing the effect of F# embedded resources – signature data, reducing optimization data in DEBUG scenarios)
  - => faster rebuilds in case of changing implementation details only! (also: `<AccelerateBuildsInVisualStudio>true</..>`)
- Optional (exp.) feature flags: `<OtherFlags>--test:..</..>`
  - Parallel Graph-based typechecking `--test:GraphBasedChecking`
  - Parallel Optimization `--test:ParallelOptimization`
  - Parallel IL code generation `--test:ParallelIlxGen`

OR globally like this: `$env:FSHARP_EXPERIMENTAL_FEATURES = '1'`

Array.Parallel.* functions

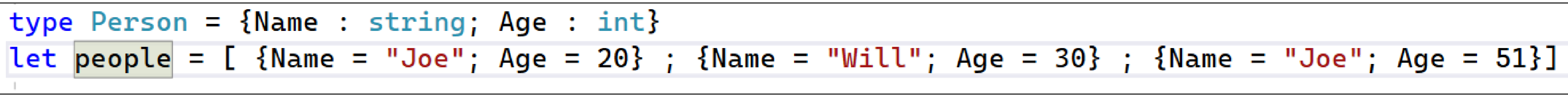
- `filter,zip,min,max,sum,average,reduce + ..by`
- `groupBy,sorting,tryFindIndex,tryFind,tryPick`



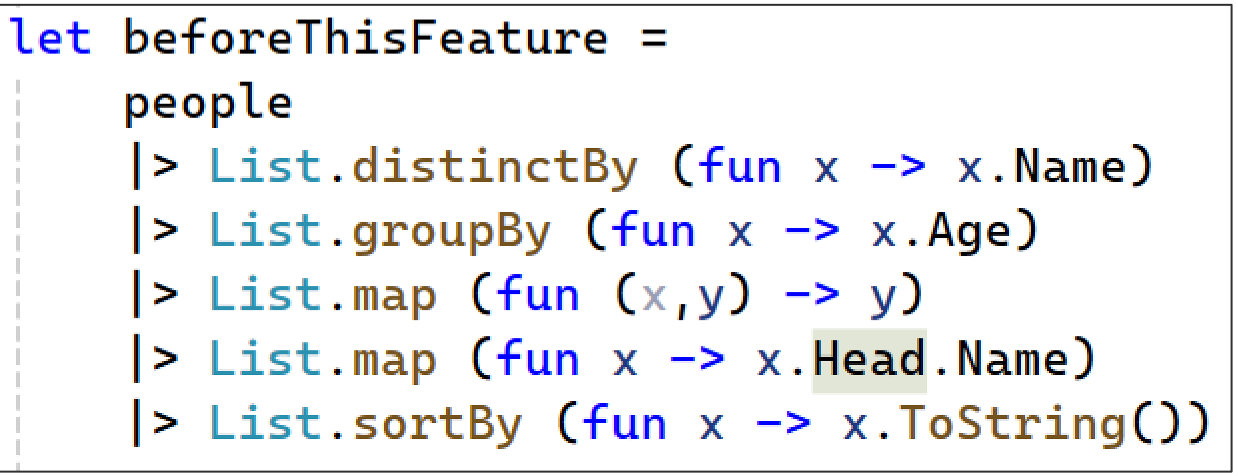
```
let arr = [|1..100_000_000|]
arr |> Array.Parallel.
```

```
| Method                | Categories                    | Mean         | Ratio    | Allocated | Alloc Ratio |
|---------------------- |------------------------------ |-------------:|---------:|----------:|------------:|
| ArrayGroupBy2         | GroupBy - calculation         | 169,024.8 us | baseline |  70.17 MB |             |
| PlinqGroupBy2         | GroupBy - calculation         |  74,683.8 us |     -56% | 103.93 MB |        +48% |
| ArrayParallelGroupBy2 | GroupBy - calculation         |  62,574.3 us |     -63% |  70.61 MB |         +1% |
|                       |                               |              |          |           |             |
| ArrayGroupBy          | GroupBy - field only          |  14,274.3 us | baseline |  57.28 MB |             |
| PlinqGroupBy          | GroupBy - field only          |  30,933.6 us |    +117% |  88.77 MB |        +55% |
| ArrayParallelGroupBy  | GroupBy - field only          |  18,318.6 us |     +29% |  47.72 MB |        -17% |
|                       |                               |              |          |           |             |
| ArrayMinBy            | MinBy(calculationFunction)    | 157,463.5 us | baseline |  11.44 MB |             |
| PlinqMinBy            | MinBy(calculationFunction)    | 160,243.5 us |      +2% |  11.44 MB |         +0% |
| ArrayParallelMinBy    | MinBy(calculationFunction)    |  48,768.7 us |     -68% |  11.45 MB |         +0% |
|                       |                               |              |          |           |             |
| ArraySort             | Sort - by int field           |  27,352.1 us | baseline |  17.17 MB |             |
| PlinqSort             | Sort - by int field           |  38,723.7 us |     +42% | 172.89 MB |       +907% |
| ArrayParallelSort     | Sort - by int field           |  76,744.8 us |    +179% | 112.76 MB |       +557% |
|                       |                               |              |          |           |             |
| ArraySortBy           | SortBy - calculation          | 214,042.4 us | baseline |  30.52 MB |             |
| PlinqSortBy           | SortBy - calculation          |  97,214.3 us |     -55% | 193.99 MB |       +536% |
| ArrayParallelSortBy   | SortBy - calculation          | 125,951.7 us |     -41% | 130.49 MB |       +328% |
|                       |                               |              |          |           |             |
| ArraySumBy            | SumBy(plain field access)     |     466.7 us | baseline |         - |          NA |
| PlinqSumBy            | SumBy(plain field access)     |     984.1 us |    +112% |   0.01 MB |          NA |
| ArrayParallelSumBy    | SumBy(plain field access)     |     687.6 us |     +47% |   0.01 MB |          NA |
|                       |                               |              |          |           |             |
| ArrayTryFind          | TryFind - calculationFunction |  76,509.7 us | baseline |   5.72 MB |             |
| PlinqTryFind          | TryFind - calculationFunction |  41,256.7 us |     -47% |  10.74 MB |        +88% |
| ArrayParallelTryFind  | TryFind - calculationFunction |  23,094.4 us |     -69% |   5.73 MB |         +0% |
LanguageFeature.AccessorFunctionShorthand, languageVersion80
LanguageFeature.MatchNotAllowedForUnionCaseWithNoData, languageVersion80
LanguageFeature.CSharpExtensionAttributeNotRequired, languageVersion80
LanguageFeature.ErrorForNonVirtualMembersOverrides, languageVersion80
LanguageFeature.WarningWhenInliningMethodImplNoInlineMarkedFunction, languageVersion80
LanguageFeature.EscapeDotnetFormattableStrings, languageVersion80
LanguageFeature.ArithmeticInLiterals, languageVersion80
LanguageFeature.ErrorReportingOnStaticClasses, languageVersion80
LanguageFeature.TryWithInSeqExpression, languageVersion80
LanguageFeature.WarningWhenCopyAndUpdateRecordChangesAllFields, languageVersion80
LanguageFeature.StaticMembersInInterfaces, languageVersion80
LanguageFeature.NonInlineLiteralsAsPrintfFormat, languageVersion80
LanguageFeature.NestedCopyAndUpdate, languageVersion80
LanguageFeature.ExtendedStringInterpolation, languageVersion80
LanguageFeature.WarningWhenMultipleRecdTypeChoice, languageVersion80
LanguageFeature.ImprovedImpliedArgumentNames, languageVersion80
LanguageFeature.DiagnosticForObjInference, languageVersion80
LanguageFeature.WarningWhenTailRecAttributeButNonTailRecUsage, languageVersion80
LanguageFeature.StaticLetInRecordsDusEmptyTypes, languageVersion80
LanguageFeature.StrictIndentation, languageVersion80
LanguageFeature.ConstraintIntersectionOnFlexibleTypes, languageVersion80
LanguageFeature.WhileBang, languageVersion80
LanguageFeature.ExtendedFixedBindings, languageVersion80
LanguageFeature.PreferStringGetPinnableReference, languageVersion80
```

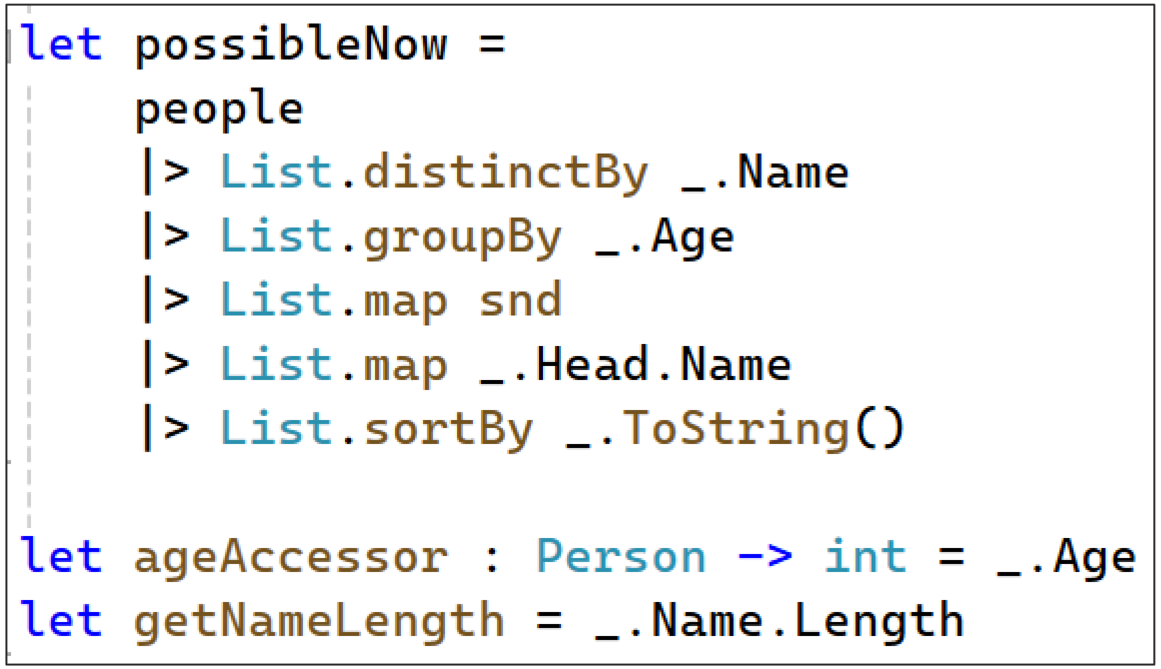
Lambda shorthand: _.Prop / _.MethodCall() / _.Indexer[]



```fsharp
type Person = {Name : string; Age : int}
let people = [ {Name = "Joe"; Age = 20} ; {Name = "Will"; Age = 30} ; {Name = "Joe"; Age = 51}]
```

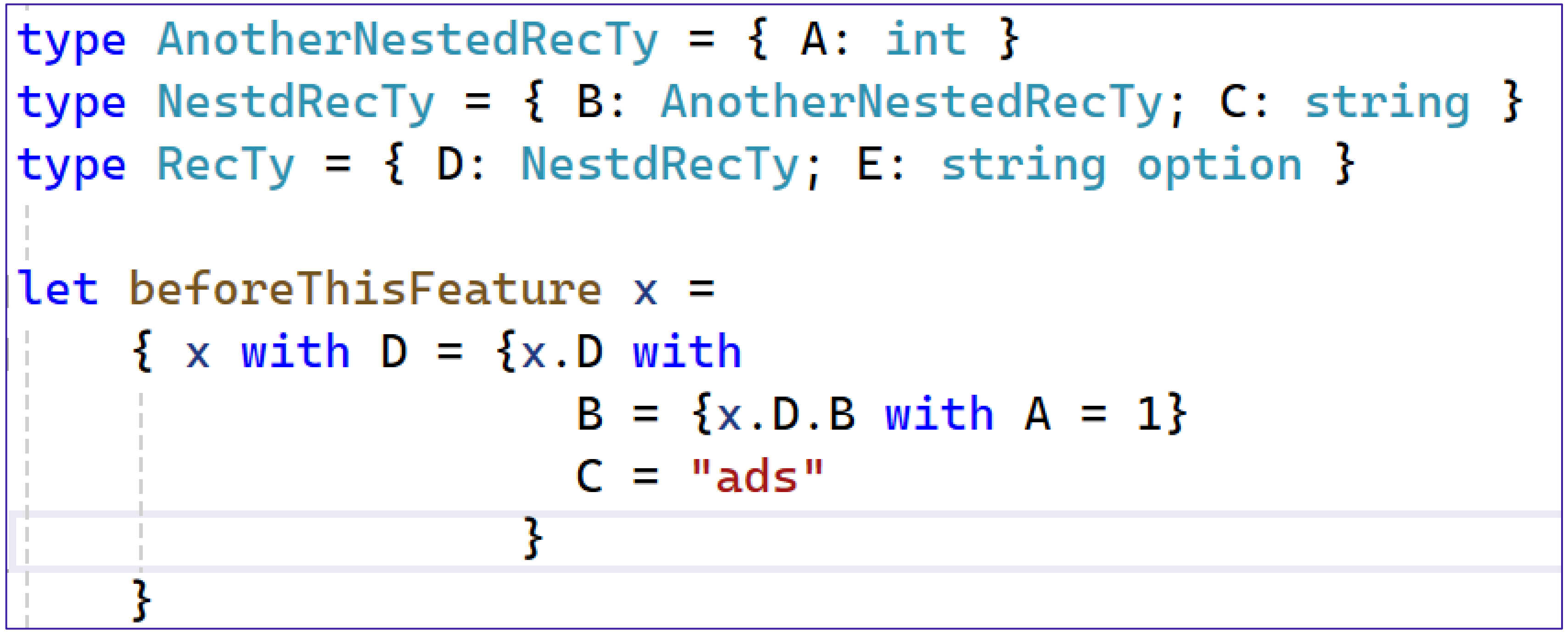


```fsharp
let beforeThisFeature =
    people
    |> List.distinctBy (fun x -> x.Name)
    |> List.groupBy (fun x -> x.Age)
    |> List.map (fun (x,y) -> y)
    |> List.map (fun x -> x.Head.Name)
    |> List.sortBy (fun x -> x.ToString())
```

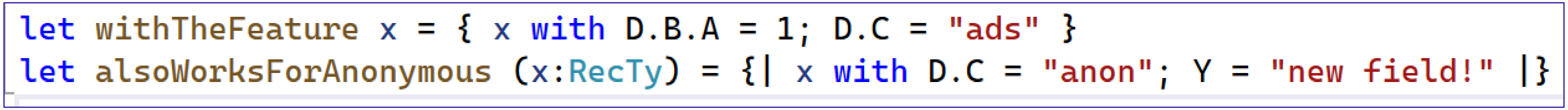


```fsharp
let possibleNow =
    people
    |> List.distinctBy _.Name
    |> List.groupBy _.Age
    |> List.map snd
    |> List.map _.Head.Name
    |> List.sortBy _.ToString()

let ageAccessor : Person -> int = _.Age
let getNameLength = _.Name.Length
```

Nested Record Field Copy and Update



```
type AnotherNestedRecTy = { A: int }
type NestdRecTy = { B: AnotherNestedRecTy; C: string }
type RecTy = { D: NestdRecTy; E: string option }

let beforeThisFeature x =
    { x with D = {x.D with
                    B = {x.D.B with A = 1}
                    C = "ads"
                  }
    }
```

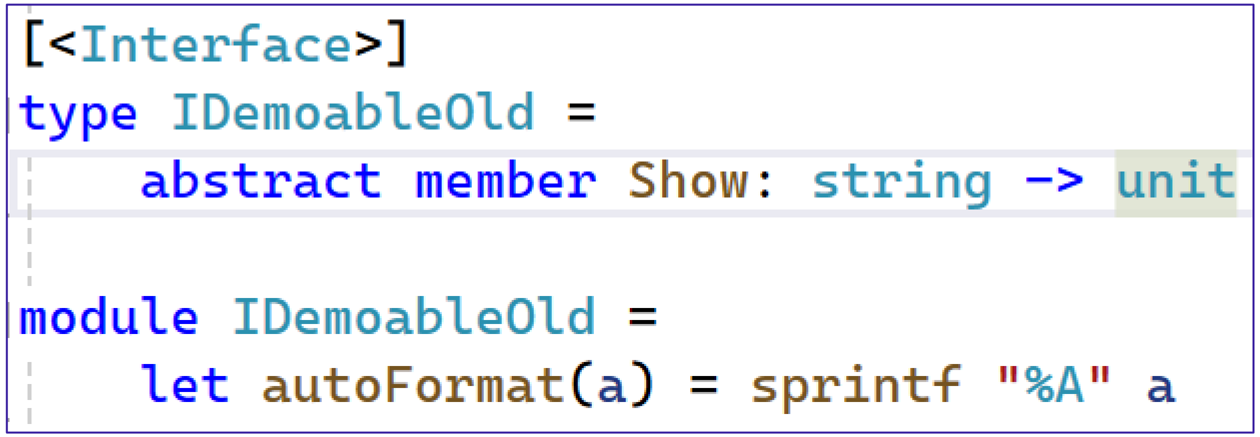


```
let withTheFeature x = { x with D.B.A = 1; D.C = "ads" }
let alsoWorksForAnonymous (x:RecTy) = {| x with D.C = "anon"; Y = "new field!" |}
```

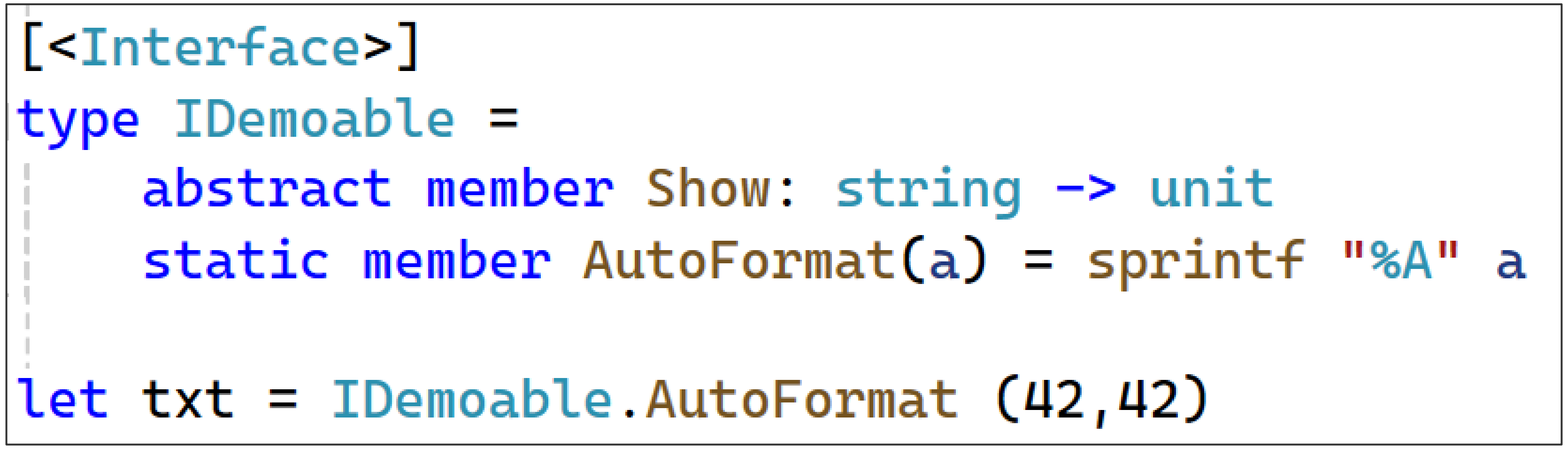
Uniformity: These are possible now!

- Static members in interfaces
- ‘static let’ (+ mutable), ‘static do’ allowed in:
  - Discriminated unions
  - Records
  - Structs
  - Types without constructor arguments
- Try-with can be used inside seq{} expressions
  - Also applies to more complex [] and [||] builders

Static members in interfaces

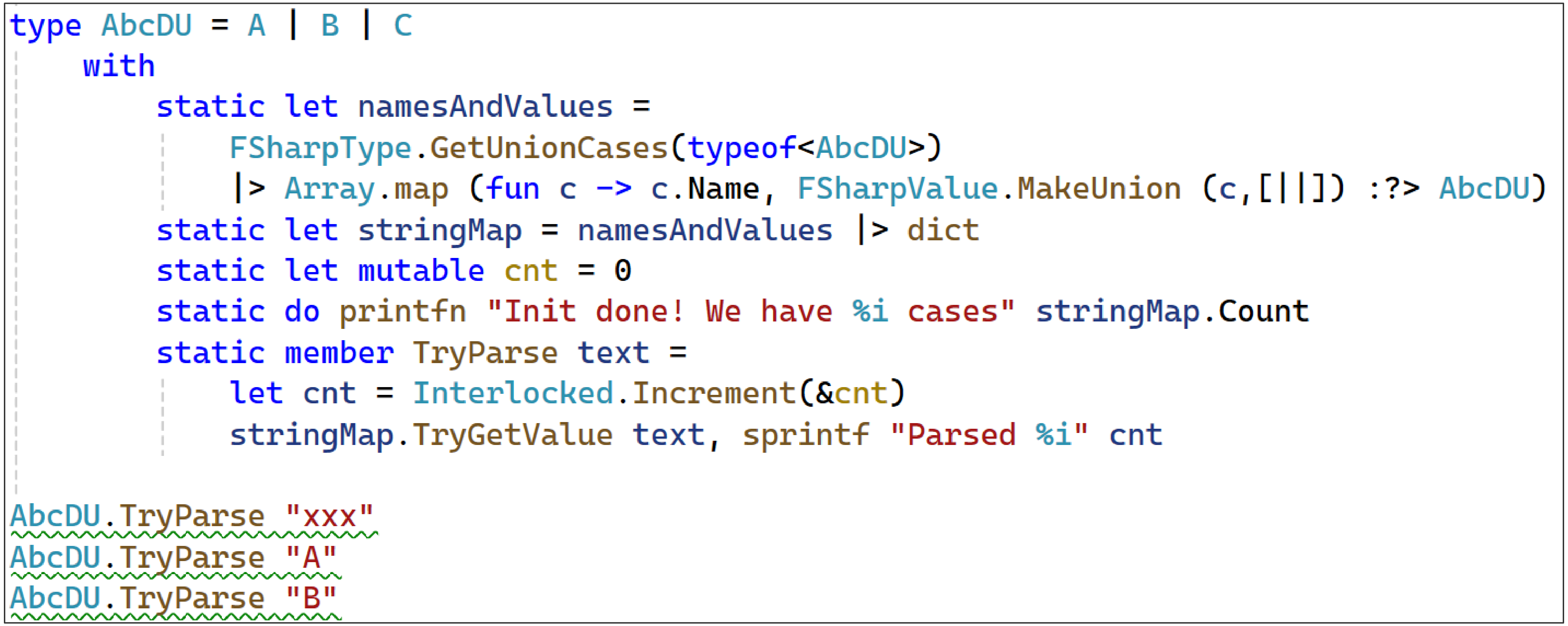


```fsharp
[<Interface>]
type IDemoableOld =
    abstract member Show: string -> unit

module IDemoableOld =
    let autoFormat(a) = sprintf "%A" a
```



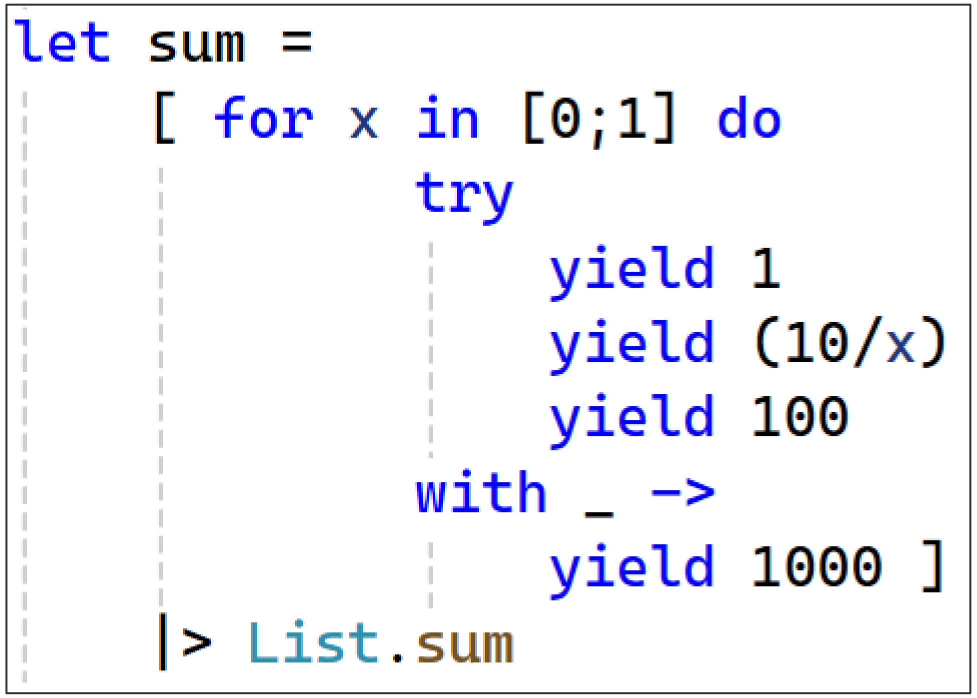
```fsharp
[<Interface>]
type IDemoable =
    abstract member Show: string -> unit
    static member AutoFormat(a) = sprintf "%A" a

let txt = IDemoable.AutoFormat (42,42)
```

Static let

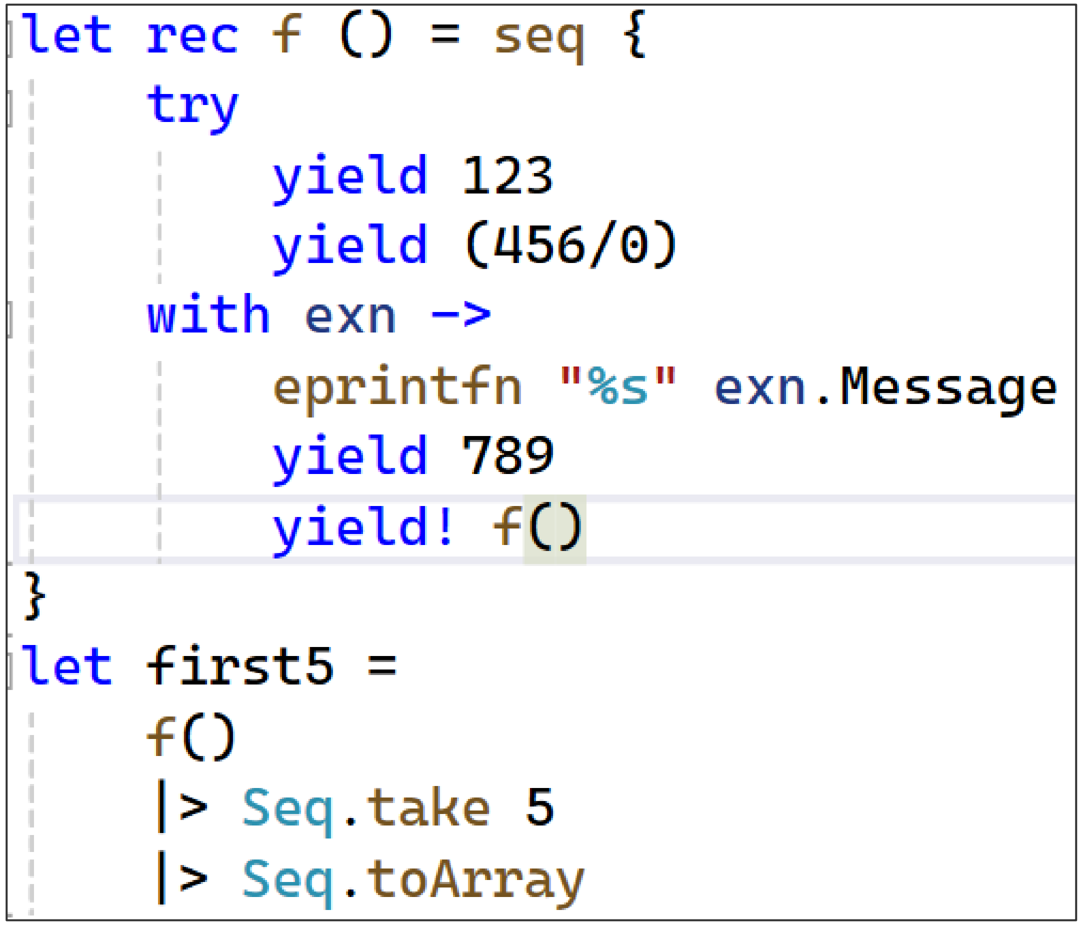


```fsharp
type AbcDU = A | B | C
    with
        static let namesAndValues =
            FSharpType.GetUnionCases(typeof<AbcDU>)
            |> Array.map (fun c -> c.Name, FSharpValue.MakeUnion (c,[||]) :?> AbcDU)
        static let stringMap = namesAndValues |> dict
        static let mutable cnt = 0
        static do printfn "Init done! We have %i cases" stringMap.Count
        static member TryParse text =
            let cnt = Interlocked.Increment(&cnt)
            stringMap.TryGetValue text, sprintf "Parsed %i" cnt

AbcDU.TryParse "xxx"
AbcDU.TryParse "A"
AbcDU.TryParse "B"
```

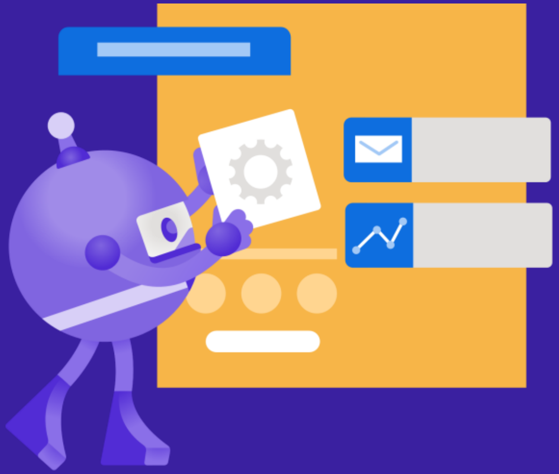
Try-with in seq{}

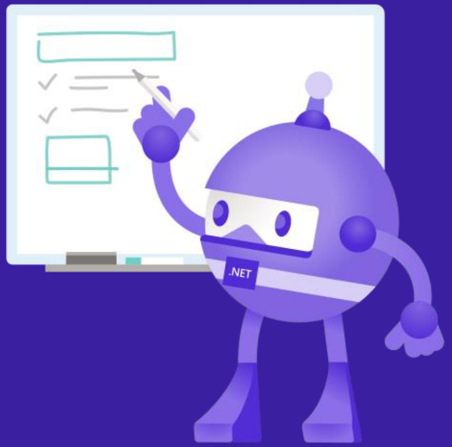


```
let sum =
    [ for x in [0;1] do
            try
                yield 1
                yield (10/x)
                yield 100
            with _ ->
                yield 1000 ]
    |> List.sum
```



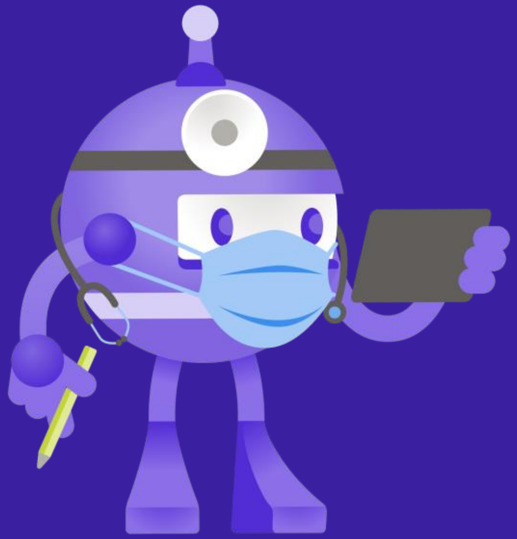
```
let rec f () = seq {
    try
        yield 123
        yield (456/0)
    with exn ->
        eprintfn "%s" exn.Message
        yield 789
        yield! f()
}
let first5 =
    f()
    |> Seq.take 5
    |> Seq.toArray
```





Another chance to answer your questions.

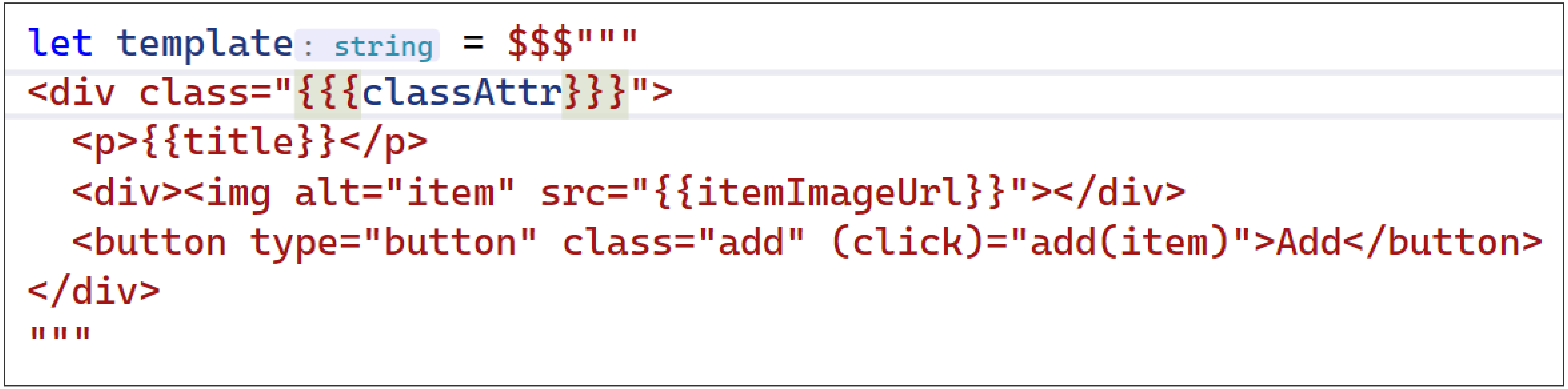
(Tomas has to leave)



Printing – extended interpolation syntax

[New syntax for string interpolation in F# - .NET Blog (microsoft.com)]

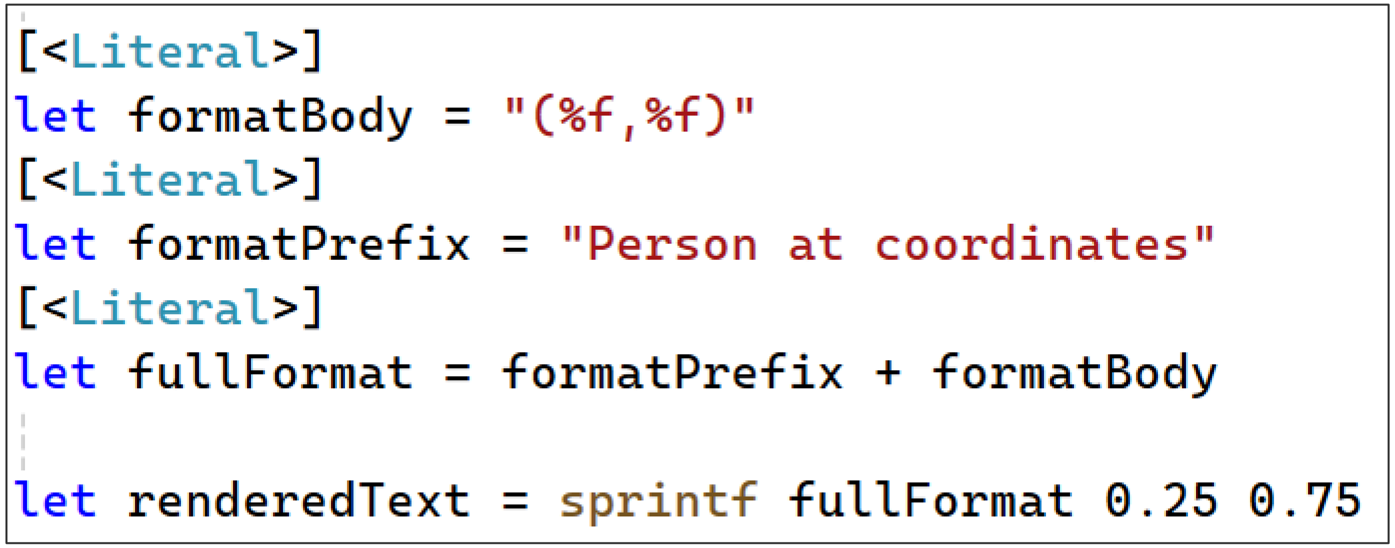
Number of $ at the beginning dictates the number of { for including values, less { do not need escaping.



```
let template : string = $$$"""
<div class="{{{classAttr}}}">
  <p>{{title}}</p>
  <div><img alt="item" src="{{itemImageUrl}}"></div>
  <button type="button" class="add" (click)="add(item)">Add</button>
</div>
"""
```

Printing – use literals for printfn family

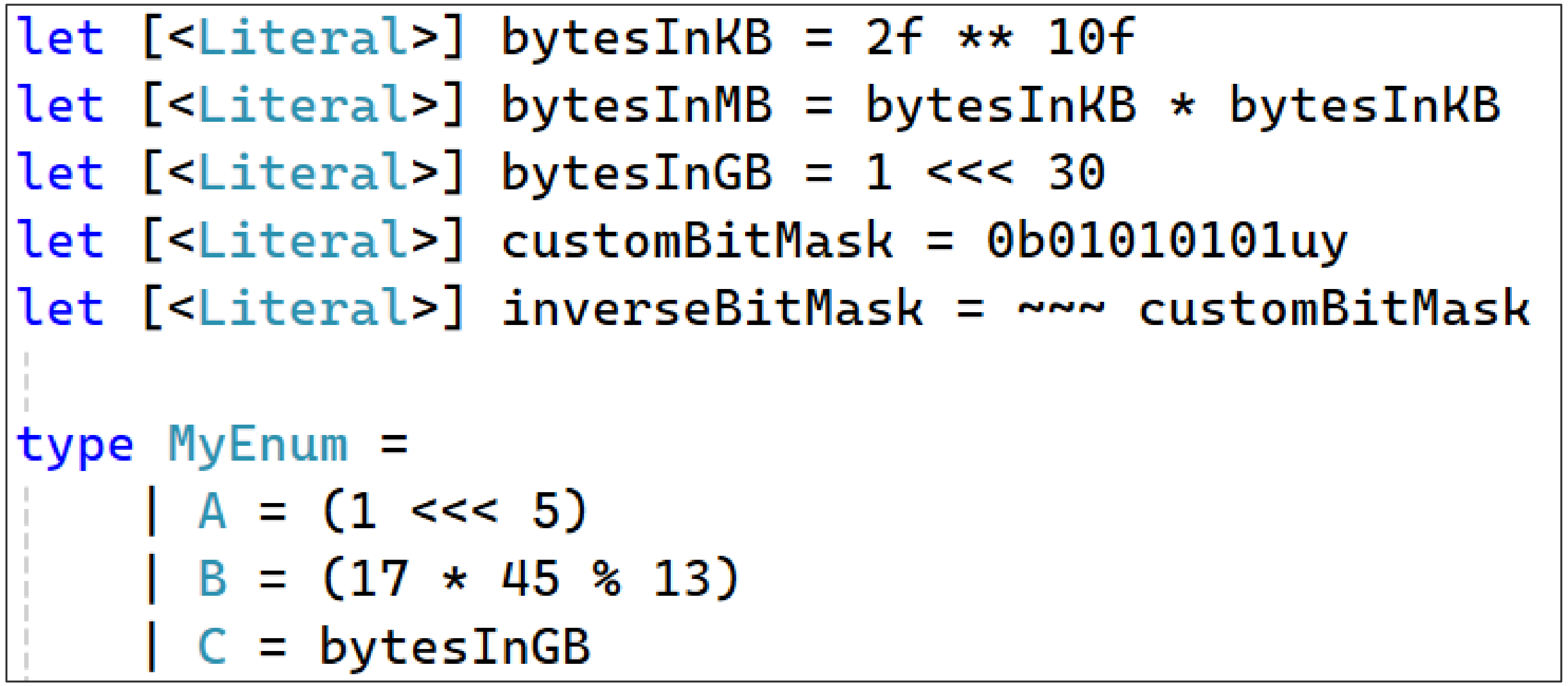
Compose print formats from reusable snippets, DRY



```
[<Literal>]
let formatBody = "(%f,%f)"
[<Literal>]
let formatPrefix = "Person at coordinates"
[<Literal>]
let fullFormat = formatPrefix + formatBody

let renderedText = sprintf fullFormat 0.25 0.75
```

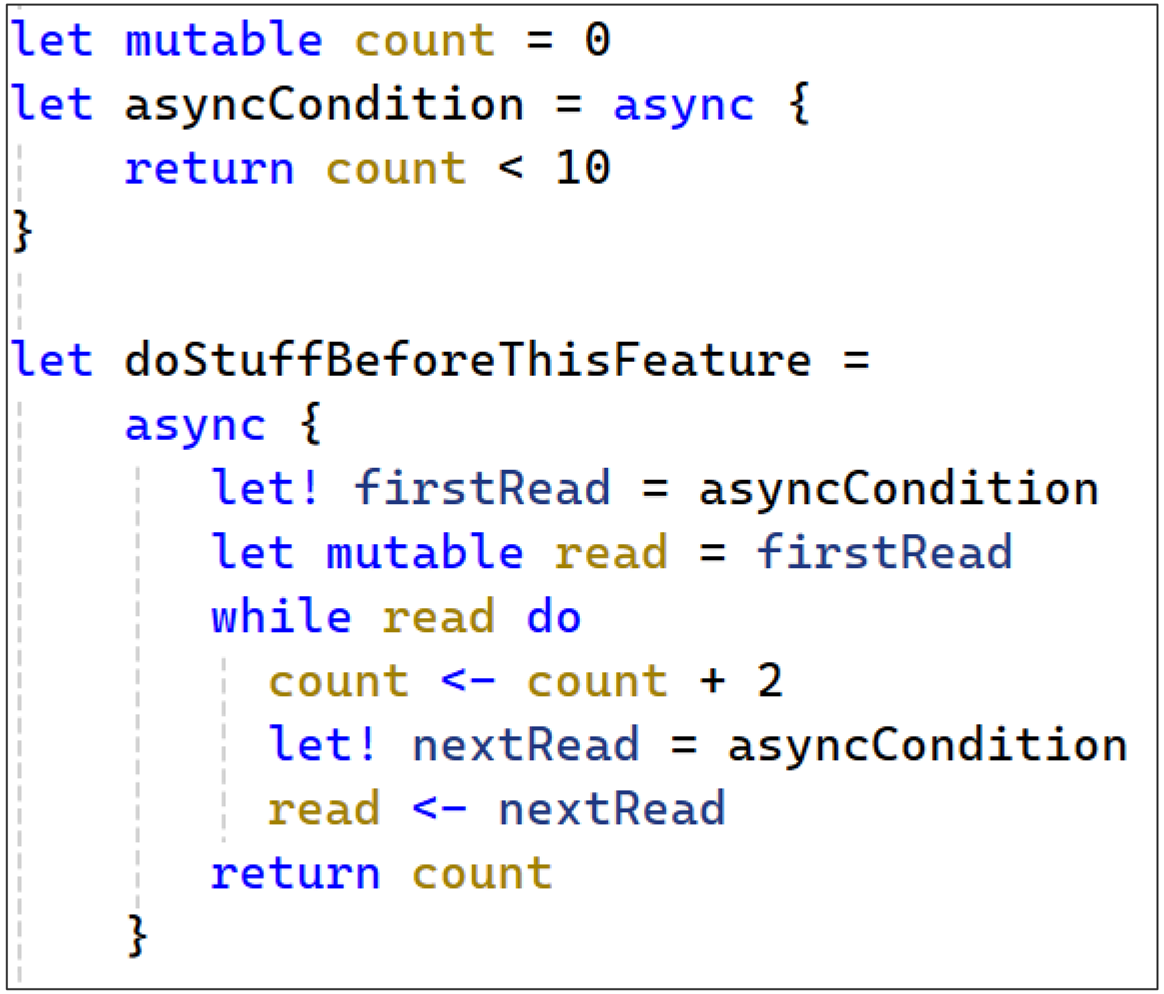
Arithmetic operators in literals

- +,-,*, /, %, &&&, |||, <<<, >>>, ^^^, ~~~, **
- not, &&, || are allowed for bools.

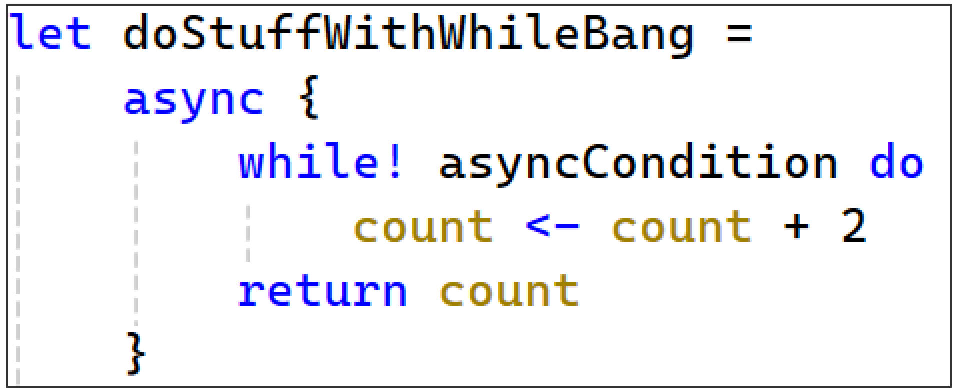


```
let [<Literal>] bytesInKB = 2f ** 10f
let [<Literal>] bytesInMB = bytesInKB * bytesInKB
let [<Literal>] bytesInGB = 1 <<< 30
let [<Literal>] customBitMask = 0b01010101uy
let [<Literal>] inverseBitMask = ~~~ customBitMask

type MyEnum =
    | A = (1 <<< 5)
    | B = (17 * 45 % 13)
    | C = bytesInGB
```

While! (while bang) in computation expressions



```fsharp
let mutable count = 0
let asyncCondition = async {
    return count < 10
}

let doStuffBeforeThisFeature =
    async {
        let! firstRead = asyncCondition
        let mutable read = firstRead
        while read do
          count <- count + 2
          let! nextRead = asyncCondition
          read <- nextRead
        return count
    }
```

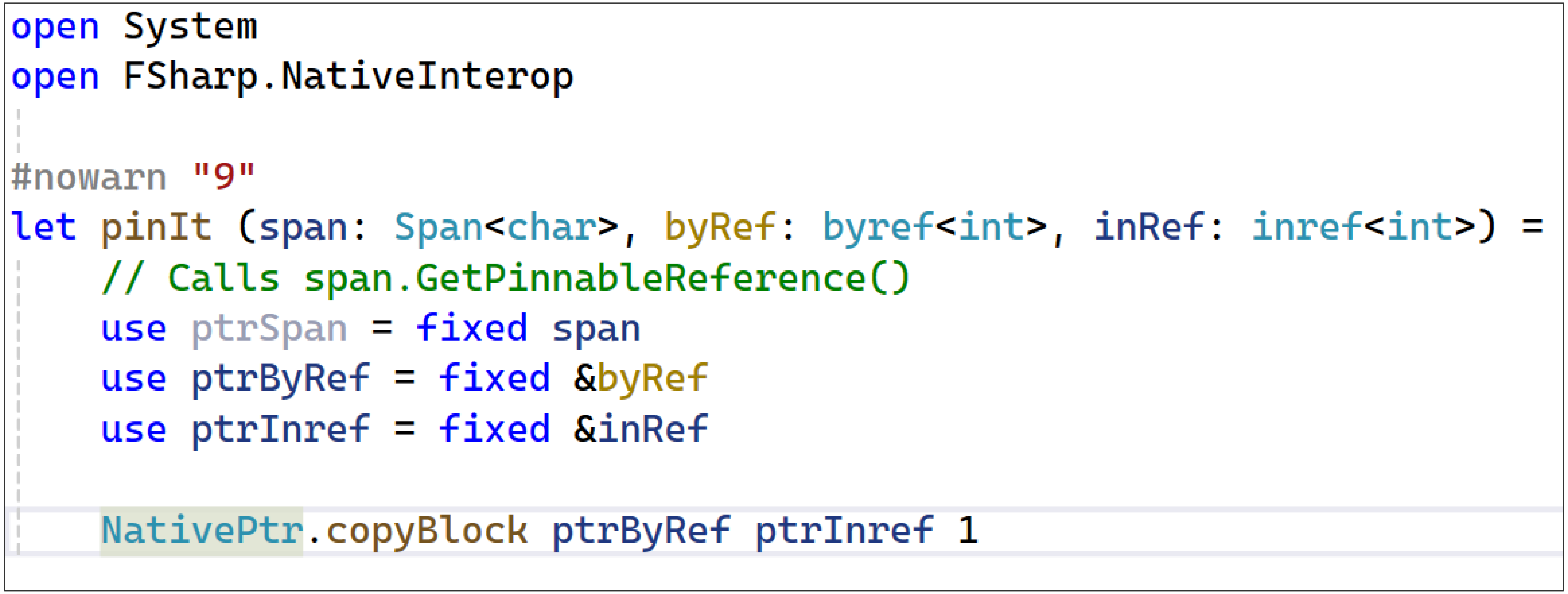


```fsharp
let doStuffWithWhileBang =
    async {
        while! asyncCondition do
            count <- count + 2
        return count
    }
```

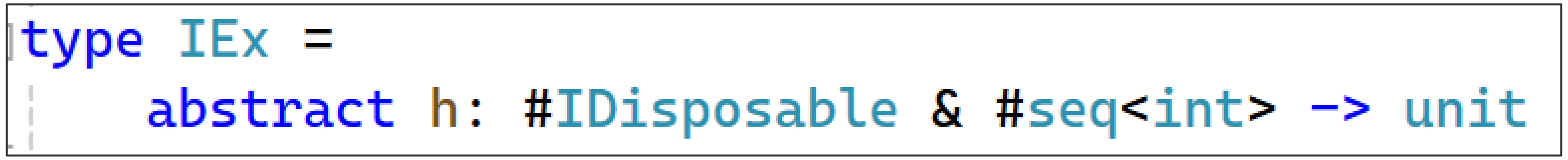
Extended fixed bindings

Before F#8, statements of the following form: **use ptr = fixed expr** were allowed : Array String Address of an array element Address of a field	<u>**Newly added support:**</u> byref<'t> inref<'t> outref<'t> any 'a when 'a has an instance method GetPinnableReference : unit -> byref<‘t> **OR** inref<‘t> (or extension method)

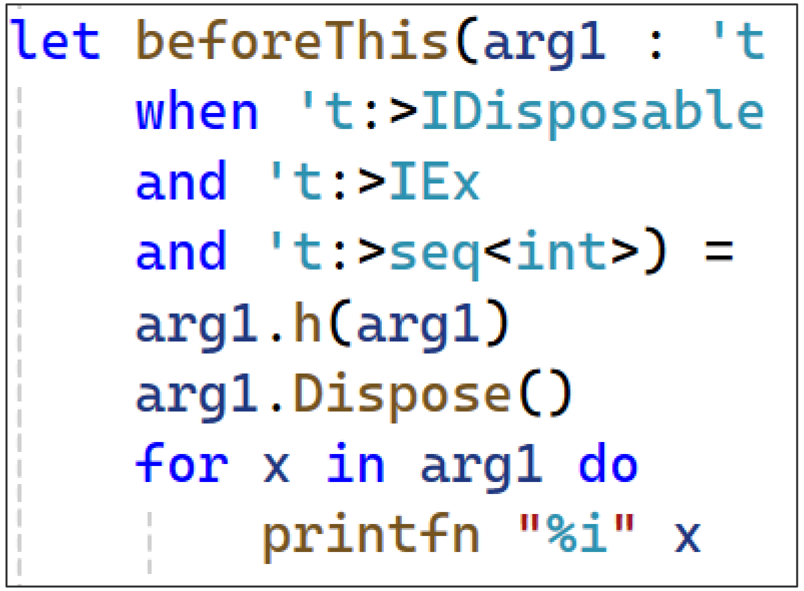
Extended fixed bindings


```
open System
open FSharp.NativeInterop

#nowarn "9"
let pinIt (span: Span<char>, byRef: byref<int>, inRef: inref<int>) =
    // Calls span.GetPinnableReference()
    use ptrSpan = fixed span
    use ptrByRef = fixed &byRef
    use ptrInref = fixed &inRef

    NativePtr.copyBlock ptrByRef ptrInref 1
```

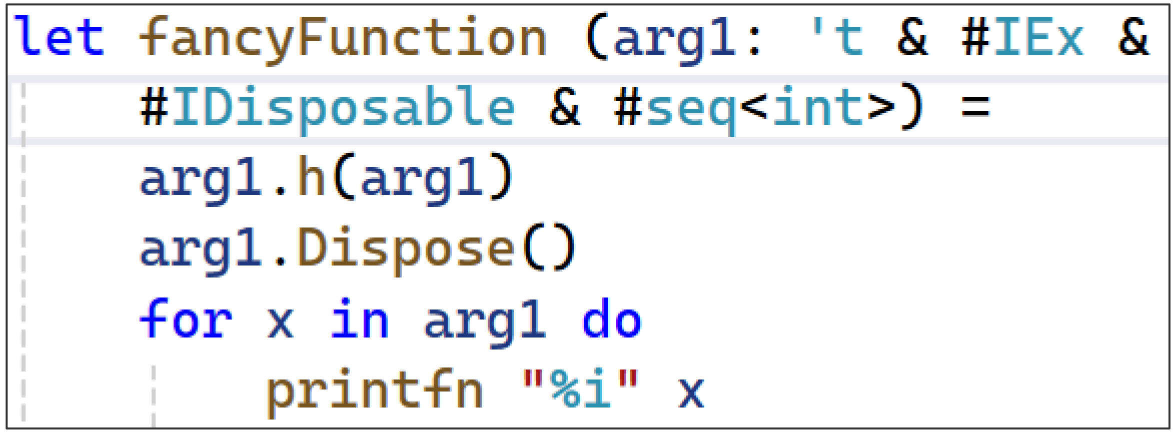
Type constraint intersection syntax “&”



```
type IEx =
    abstract h: #IDisposable & #seq<int> -> unit
```



```
let beforeThis(arg1 : 't
    when 't:>IDisposable
    and 't:>IEx
    and 't:>seq<int>) =
    arg1.h(arg1)
    arg1.Dispose()
    for x in arg1 do
        printfn "%i" x
```



```
let fancyFunction (arg1: 't & #IEx &
    #IDisposable & #seq<int>) =
    arg1.h(arg1)
    arg1.Dispose()
    for x in arg1 do
        printfn "%i" x
```

Quality of life

- **Trimmability** – discriminated unions, records, anonymous records now trimmable – for Native AOT
- [<Struct>] Discriminated Unions can now have > 49 cases

Fsharp.Core - performance improvements of library functions

- ValueOption – functions + lambdas inlined<-map 1.5x faster
- Option – functions + lambdas inlined <- 3x faster for map
- List.contains inlines type equality <- 16x faster for int
- List<_>.GetHashCode() no longer stack overflows at > 50.000 elements
- Seq.toArray reduced allocations for small sizes
- Reflection -> FsharpType.MakeStructTupleType has a new faster overload without Assembly argument
- Binding (let!) of async within a task{} expression starts on the same thread now

Visual Studio updates for F#

Hints

- Type hints, returns type hints, parameter name hints
- Compiler inferred information
- More useful for less clear code
- Options → Text Editor → F# → Advanced → Hints
- Related tickets on GitHub

Code fixes

- Quick actions (light bulb menu)
- Triggered by diagnostics
- 30+ F# code fixes
- Options → Text Editor → F# → Code fixes
- Related tickets on GitHub

IntelliSense

- Code completion, quick info tooltips
- Better autocomplete in pattern matching, attributes, types
- Options → Text Editor → F# → IntelliSense
- Options → Text Editor → F# → QuickInfo
- Related tickets on GitHub (1, 2)

Diagnostics

- Improved parser recovery
- Background diagnostics analysis
- Options → Text Editor → F# → Advanced → Background Analysis
- Related tickets on GitHub

C# → F# navigation

- F# code instead of decompiled metadata
- Convenient development in mixed F#+C# solutions

Performance

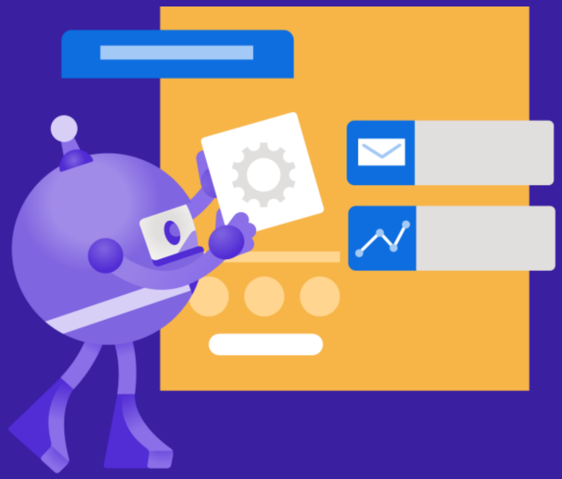
- Universal search (Ctrl + T)
- Semantic highlighting
- Allocation improvements
- Fast find references
- Related tickets on GitHub

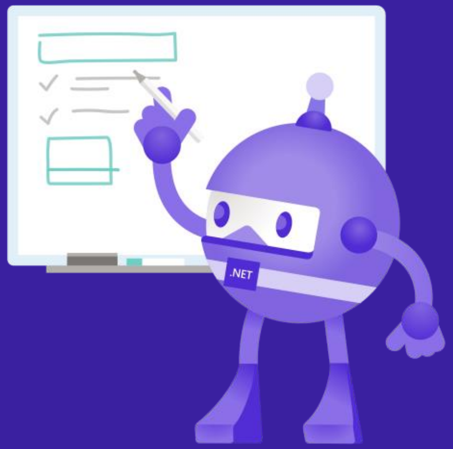
Contribute to F# in Visual Studio!

- Tracking ticket for Visual Studio F# improvements
- Good first issues

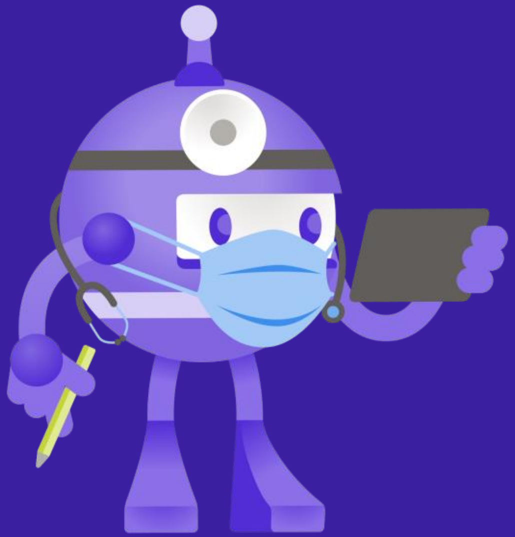
F# everywhere

- F# on .NET blog
- F# on ~~Twitter~~ X
- F# communities in Slack and Discord





Time to answer your questions.



Thanks for joining!