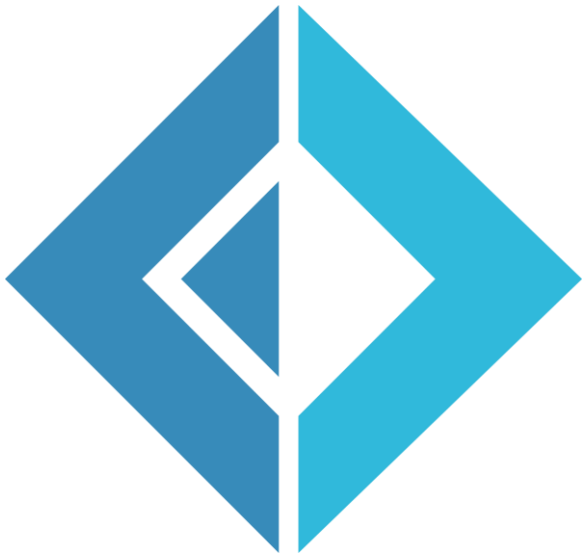
F# Functional Language

Nathan Blankenship, Tatum Gray, Tucker Kane

Why Did We Choose F#?

- To better understand functional programming languages
- F# seemed simpler than Scheme
- Shared a bit of C#'s identity
- Exposed us to the .NET framework
- Opportunity to learn a niche language



History

- Created by Don Syme
- Approached by Microsoft Research
- Started as Haskell.NET
- Inspired by OCaml Language
- Strong Typed Languages
  - C
  - Basic
  - Prolog
  - Scheme
- Development 2001 - Current

Syntax

- Comments
  - // for single line comment
  - (* ... *) for multiple line comment
- Assignment
  - let a = 1
- Arithmetic Operators
  - +, -, *, /, %, **

Syntax

- Output
  - printf / printfn "%b" bool "%i" integer "%s" string
  - printfn "Name: %s Value: %i Weight: %i" Expects 3 arguments: of types string, int, int
- Bitwise Operations
  - &&& = AND
  - >>> = bit shift right, <<< = bit shift left
  - ^^^ = OR , ||| = exclusive OR
  - ~~~ = NOT
- Boolean Operations
  - && = AND
  - || = OR

Data Types

- Boolean
- Integer, Float, byte
- String
    - "..."
- Char
    - '...'

Data Type Name	F# type	.Net Type	Type Symbol	Example	Size	Represents
Signed byte	sbyte	System.SByte	y	4y	1 byte with a sign bit	8-bit signed integer
Unsigned byte	byte	System.Byte	uy	6uy	1 byte with no sign bit	8-bit unsigned integer
Short	int16	System.Int16	s	10s	2 byte with a sign bit	16-bit signed integer
Unsigned Short	uint16	System.UInt16	us	80us	2 byte with no sign bit	16-bit unsigned integer
Int	int	System.Int32	none	456	4 byte with a sign bit	32-bit signed integer
Unsigned int	uint32	System.Uint32	u	456u	4 byte with no sign bit	32-bit unsigned integer
Long	int64	System.Int64	l	9999999l	8 byte with a sign bit	64-bit signed integer
Unsigned long	uint64	System.UInt64	ul	9999999ul	8 byte with no sign bit	64-bit unsigned integer

Float or double	float or double	System.Double	optional e	3., 2.01, 2.20e10	8 byte with a sign bit	64-bit signed floating point number
Single	float32 or single	System.Single	e (optional), f	2.0f, 2.02f, 2.02e20	4 byte with a sign bit	32-bit signed floating point number
Native int	nativeint	System.IntPtr	n	456n	Size is platform specific	store native int values
Unsigned Native int	unativeint	System.UIntPtr	un	456un	Size is Platform specific	store unsigned native values
Decimal	decimal	System.Decimal	m	456m	Maps to System.Decimal	128-bit signed floating point number
Big number	bignum	Microsoft.FSahrp.Math.BigInt	I	456I	Maps to Microsoft.FSahrp.Math.BigInt	arbitrary precision Integer

Example Code (Recursion)

Recursion in C++:

```
int fib(int x) {
   if((x==1)||(x==0)) {
      return(x);
   }
   else {
      return(fib(x-1)+fib(x-2));
   }
}
```

Recursion in F#: (EXPLICIT)

```
let rec fib n =
    match n with
    | 0 | 1 -> n
    | n -> fib (n-1) + fib (n-2)
```

F# is capable of recursion through the explicit identifier *rec*. Rec is *ahem* RECommended for the example problem at the end of this presentation

Example Code (Recursion)

Recursion in F# (IMPLICIT)

```
type MyClass() =
    member this.Fib(n) =
        match n with
        | 0 | 1 -> n
        | n -> this.Fib(n-1) + this.Fib(n-2)
```

F# also supports implicit recursion, though notably the let keyword commits the programmer to using rec explicitly

Pros

- Less lines of code
- Is not particularly picky about syntax
  - Doesn’t use bracket
  - Doesn’t care about indentation
  - Only picky about spacing/indentation
- Do not have to declare types
- Strongly typed language

Drawbacks

- Unreadability
  - Strong undeclared typing
  - Sometimes hard to find bugs
- Bugs within Language itself
- Small Community
- Personal Preference

Citations

- Cartermp. (n.d.). *Symbol and operator reference - F#*. Symbol and Operator Reference - F# | Microsoft Docs. Retrieved September 29, 2021, from https://docs.microsoft.com/en-us/dotnet/fsharp/language-reference/symbol-and-operator-reference/.
- DanManPanther (2020, May 31). *Reasons NOT to Use F# for a Web App?* Online forum post. https://www.reddit.com/r/fsharp/comments/gu5ln8/reasons_not_to_use_f_for_a_web_app/
- *Data Types in F#*. (n.d.). Www.c-Sharpcorner.com. Retrieved September 30, 2021, from https://www.c-sharpcorner.com/uploadfile/f5b919/data-types-in-fsharp/
- Depositphotos, I. (n.d.). *Questionmark stock photos & Royalty-Free Images*. Depositphotos. Retrieved September 30, 2021, from https://depositphotos.com/stock-photos/questionmark.html.
- Don Syme. (June 2020). *The Early History of F#*. Proc. ACM Program. Lang. 4, HOPL, Article 75, 58 pages. https://doi.org/10.1145/3386325
- F# - basic syntax. (n.d.). Retrieved September 29, 2021, from https://www.tutorialspoint.com/fsharp/fsharp_basic_syntax.htm.
- Heller, M. (2018, April 23). *14 excellent reasons to use F#*. InfoWorld. Retrieved September 29, 2021, from https://www.infoworld.com/article/3269057/14-excellent-reasons-to-use-f-sharp.html#:~:text=F%23%20is%20for%20scripting.%20F%23%20supports%20functional%20programming,way%20of%20the%20partial%20application%20of%20function%20arguments.
- McCaffery, J. D. (2015, March 1). *Why I Don't Like the F# Language*. WordPress. Retrieved September 29, 2021, from https://jamesmccaffrey.wordpress.com/2015/03/01/why-i-dont-like-the-f-language/.
- Wikimedia Foundation. (2021, August 10). *F sharp (programming language)*. Wikipedia. Retrieved September 28, 2021, from https://en.wikipedia.org/wiki/F_Sharp_(programming_language)
- Neeraj Mishra (2021) *C/C++ Program for Fibonacci Series Using Recursion* (https://www.thecrazyprogrammer.com/2014/12/c-cpp-program-for-fibonacci-series-using-recursion.html
- Philippe Vlérick (2018) *Recursive Fibonacci with F#* https://pvlerick.github.io/2009/07/recursive-fibonacci-with-f
- Carter et al (2020, August 12) *Recursive Functions: The rec Keyword* https://docs.microsoft.com/en-us/dotnet/fsharp/language-reference/functions/recursive-functions-the-rec-keyword

Questions?