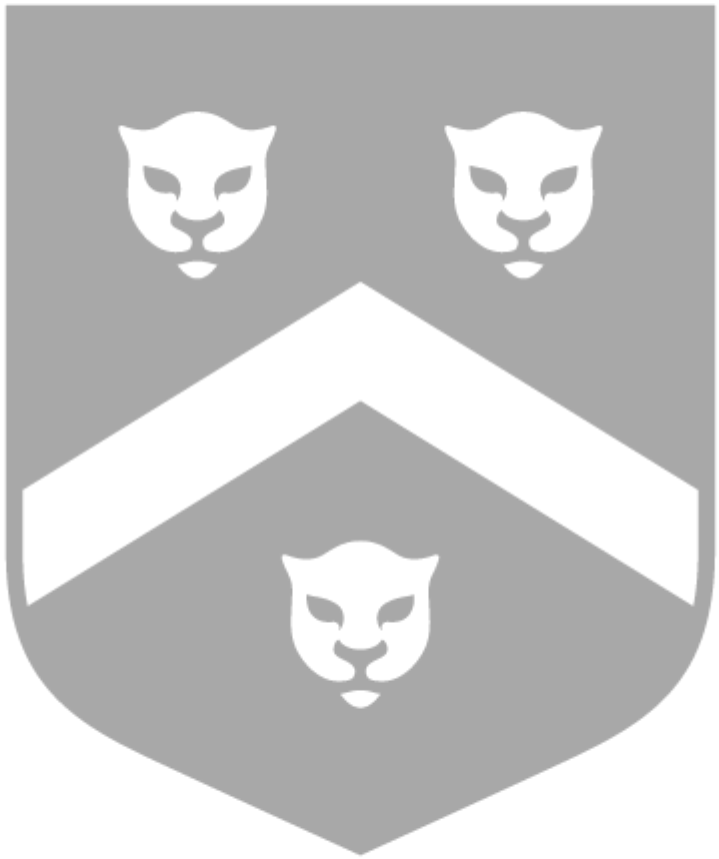
Intro to Programming Languages



Professor Frank Kreimendahl

School of Computing and Data Science

Wentworth Institute of Technology

May 15, 2023

A First Look at F#
Motivation
Functional Languages
F#
Running F# in IDE
1+2*3
The F# langauge
F# Types

A First Look at F#

Why F#?

1. Very high level above machine architecture — powerful
2. Functional language — everything returns a result
3. Interactive — code and test immediately
4. Minimal side effects – easier to reason about program behavior
5. Object-oriented
6. Pattern matching programming style
7. Useful for studying fundamentals of languages
8. F# runs on the .NET CLR.
9. C# and F# classes can be freely mixed.

Functional Languages

Example (a factorial function in F#)

```
let rec factorial x =
  if x <= 0 then 1 else x * factorial (x-1);;
```

- Hallmarks of functional languages:
  - Single-valued variables
  - Heavy use of recursion
  - Functions are first-class citizens, can be used as parameters, function results, etc.
  - Minimal use of assignments and side-effects

A First Look at F#
Motivation
Functional Languages
F#
Running F# in IDE
1+2*3
The F# langauge
F# Types

F#

- Open standard adopted/supported by Microsoft
- Created as an implementation of OCaml
- Full support for functional programming
- Very strong static type system
- Useful in Artificial Intelligence and programming languages
- Compatible with .NET
- Object oriented

A First Look at F#
Motivation
Functional Languages
F#
Running F# in IDE
1+2*3
The F# langauge
F# Types

- In lab this week: Download IDE installers from course website
- Enter F# instructions into the Read Eval Print Loop (REPL)
  - `dotnet fsi` on Linux/OS X
  - `fsi` on Windows
- F# has two running modes with slightly different syntax
  - **interactive mode:** `;;` at the end of expressions
  - **source code:** no `;;` at the end of expressions
- Lecture examples will always use interactive formatting
- Lecture examples also include '>' prompt character, which should not be typed as part of an expression

A First Look at F#
Motivation
Functional Languages
F#
Running F# in IDE
1+2*3
The F# langauge
F# Types

$1+2\times3$

Example

```
> 1+2*3;;
val it : int = 7
```

- Type an expression after > prompt; F# replies with value and type
- Variable **it** is a variable with the returned result.
- Notice F# inferred the type as int.

A First Look at F#

The F# langauge

Constants

Operators

Variables

Tuples and Lists

Func Definitions

F# Types

The F# langauge

A First Look at F#

The F# langauge

Constants

Number constants

bool constants

char and string

Operators

Variables

Tuples and Lists

Func Definitions

F# Types

Number constants

Example

```
> 1234;;
val it : int = 1234
> 123.4;;
val it : float = 123.4
```

- Integer constants: standard decimal
- Float constants: standard decimal notation
- Note the type names: **int**, **float**

A First Look at F#
The F# langauge
Constants
Number constants
bool constants
char and string
Operators
Variables
Tuples and Lists
Func Definitions
F# Types

bool constants

Example

```
> true;;
val it : bool = true
> false;;
val it : bool = false
```

- bool constants **true** and **false**
- F# is case-sensitive: use **true**, not **True** or **TRUE**
- Note type name: **bool**

char and string constants

Example

```
> "fred";;
val it : string = "fred"
> "H";;
val it : string = "H"
> 'H';;
val it : char = 'H'
```

- String constants: text inside double quotes
- Can use C-style escapes: **\n**, **\t**, \\, \", etc.
- Character constants: 1 character inside single quotes
- Note type names: **string** and **char**

A First Look at F#
The F# langauge
Constants
Operators
Arithmetic
Concat
More Relations
bool operators
Short-circuiting
Conditionals
Exercises
Conversion
Exercises
Variables
Tuples and Lists
Func Definitions
F# Types

Arithmetic

Example

```
> -1 + 2 - 3 * 4 / 5 % 6;;
val it : int = -1

> -1.0 + 2.0 - 3.0 * 4.0 / 5.0;;
val it : float = -1.4
```

- Standard operators for integers, using $-$ for unary negation and for binary subtraction
- Same operators for floats
- Left associative, precedence is $\{+, -\} < \{*, /, \%\} < \{-\}$.

A First Look at F#
The F# langauge
Constants
Operators
Arithmetic
Concat
More Relations
bool operators
Short-circuiting
Conditionals
Exercises
Conversion
Exercises
Variables
Tuples and Lists
Func Definitions
F# Types

Concatenation and Relations

Example

```
> "bibity" + "bobity" + "boo";;
val it : string = "bibitybobityboo"
> 2 < 3;;
val it : bool = true
> 1.0 <= 1.0;;
val it : bool = true
> 'd' > 'c';;
val it : bool = true
> "abce" >= "abd";;
val it : bool = false
```

- String concatenation: **+** operator
- Ordering comparisons: $<, >, <=, >=$, apply to all types so far: these are *comparable types*

A First Look at F#
The F# langauge
Constants
Operators
Arithmetic
Concat
More Relations
bool operators
Short-circuiting
Conditionals
Exercises
Conversion
Exercises
Variables
Tuples and Lists
Func Definitions
F# Types

More Relations

Example

```
> 1=2;;
val it : bool = false
> 1 = 2;;
val it : bool = false
> true <> false;;
val it : bool = true
> 1.3 = 1.3;;
val it : bool = true
> [1; 4; 6] = [1; 4; 6];;
val it : bool = true
```

- Equality comparisons: = and <>
- Most types are equality testable: these are *equality types*

A First Look at F#
The F# langauge
Constants
Operators
Arithmetic
Concat
More Relations
bool operators
Short-circuiting
Conditionals
Exercises
Conversion
Exercises
Variables
Tuples and Lists
Func Definitions
F# Types

bool operators

Example

```
> 1 < 1 + 1 || 3 > 4;;
val it : bool = true
> 1 < 2 && not (3 < 4);;
val it : bool = false
```

- bool operators: &&, ||, **not**. (And we can also use = for equivalence and <> for exclusive or.)
- Precedence so far: $\{||\} < \{\&\&\} < \{=, <>, <, >, <=, >=\} < \{+, -\} < \{*, /, \%\} < \{-, \textbf{not}\}$

A First Look at F#

The F# langauge

Constants

Operators

Arithmetic

Concat

More Relations

bool operators

Short-circuiting

Conditionals

Exercises

Conversion

Exercises

Variables

Tuples and Lists

Func Definitions

F# Types

Short-circuiting bool operations

Example

```
> true || 1 / 0 = 0;;
val it : bool = true
```

Note: && and || are short-circuiting operators: if the first operand of || is true, the second is not evaluated; likewise if the first operand of && is false.

A First Look at F#
The F# langauge
Constants
Operators
Arithmetic
Concat
More Relations
bool operators
Short-circuiting
Conditionals
Exercises
Conversion
Exercises
Variables
Tuples and Lists
Func Definitions
F# Types

Conditionals

Example

```
> if (1 < 2) then "1 < 2" else "2 < 1";;
val it : string = "1 < 2"
> if (1 > 2) then 34 else 56;;
val it : int = 56
> 1 + (if (1 < 2) then 34 else 56);;
val it : int = 35
```

- Value of the expression is the value of the **true** part if the test part is true or the value of the **else** part otherwise
- **if...** construct throws an error, result type cannot be determined

A First Look at F#

The F# langauge

Constants

Operators

Arithmetic

Concat

More Relations

bool operators

Short-circuiting

Conditionals

Exercises

Conversion

Exercises

Variables

Tuples and Lists

Func Definitions

F# Types

Conditional errors

Example

```
> if (true) then 1.0 else 'a';;
error FS0001: All branches of an 'if' expression
  must return values implicitly convertible to
  the type of the first branch, which here is
  'float'. This branch returns a value of
  type 'char'.

> if (false) then "OK" else 1.0;;
error FS0001: <clipped error>
```

A First Look at F#
The F# langauge
Constants
Operators
Arithmetic
Concat
More Relations
bool operators
Short-circuiting
Conditionals
Exercises
Conversion
Exercises
Variables
Tuples and Lists
Func Definitions
F# Types

Exercises

What is the value and F# type for each expression?

```
1 "abc" + "def" + "gh";;
2 if (1 < 2) then 3.0 else 4.0;;
3 5 * (if (1 < 2) then 3 else 4);;
4 1 < 2 || (1 / 0) == 0;;
5 if (3 < 4) then 5;;
6 if (3 > 4) then 5 else 0;;
7 if (3 > 4) then 2 else '2';;
```

A First Look at F#
The F# langauge
Constants
Operators
Arithmetic
Concat
More Relations
bool operators
Short-circuiting
Conditionals
Exercises
Conversion
Exercises
Variables
Tuples and Lists
Func Definitions
F# Types

Implicit type conversion

Example

```
> 1 * 2;;
val it : int = 2
> 1.0 * 2;;
error FS0001: The type 'int' does not match the
type 'float'
> 1.0 < 2;;
val it: bool = true
```

- The $*, +$ and other arithmetic operators are overloaded to have one meaning on pairs of ints, and another on pairs of floats.
- F# **does not** perform implicit type conversion with one important exception:
  - `int32` $\rightarrow$ `double`

A First Look at F#

The F# langauge

Constants
Operators
Arithmetic
Concat
More Relations
bool operators
Short-circuiting
Conditionals
Exercises
Conversion
Exercises
Variables
Tuples and Lists
Func Definitions

F# Types

Explicit type conversion

Example

```
> float 123;;
val it : float = 123.0
> (float 123) * 2.0;;
val it: float = 246.0
> floor 3.6;;
val it : float = 3.0
> (floor 3.6) < 4.0;;
val it : bool = true
> float "123";;
val it : float = 123.0
```

Exercises

A First Look at F#

The F# langauge

Constants

Operators

Arithmetic

Concat

More Relations

bool operators

Short-circuiting

Conditionals

Exercises

Conversion

Exercises

Variables

Tuples and Lists

Func Definitions

F# Types

What is the result for each expression?

```
1 floor 5;;
2 ceil 5.5;;
3 5+4.0;;
4 if (0) then 1 else 2;;
5 if (true) then 1 else 2.0;;
6 string 97.34;;
7 97.34 + "2";;
8 97.34 + '2';;
```

Variable definition

A First Look at F#
The F# langauge
Constants
Operators
Variables
Variable definition
Variable definition
Exercises
Garbage Collection
Tuples and Lists
Func Definitions
F# Types

Example

```
> let x = 1+2*3;;
val x : int = 7
> x;;
val it : int = 7
> let y = if (x = 7) then 1.0 else 2.0;;
val y : float = 1.0
```

- Define a new variable and bind to a value using **let**.
- Variable names should consist of a letter, followed by zero or more letters, digits, and/or underscores (or most things surrounded with ``).

A First Look at F#
The F# langauge
Constants
Operators
Variables
Variable definition
Variable definition
Exercises
Garbage Collection
Tuples and Lists
Func Definitions
F# Types

Multiple variable definitions

Example

```
> let x = 23;;
val x : int = 23
> let x = true;;
val x : bool = true
> x = 23;;
error FS0001: This expression was expected to
  have type 'bool' but here has type 'int'
```

- Can define a new variable with the same name as an old one, even using a different type. (This is not particularly useful.)
- This is not the same as assignment. It defines a new variable but **does not** change the old one. Any part of the program that was using the first definition of **x**, still uses the first definition after the second definition is made.

A First Look at F#
The F# langauge
Constants
Operators
Variables
Variable definition
Variable definition
Exercises
Garbage Collection
Tuples and Lists
Func Definitions
F# Types

Variable definition

Example

```
> let fred = 23;;
val fred : int = 23
> fred <- fred + 1;;
error FS0027: This value is not mutable...
> let mutable fred = 23;;
val mutable fred : int = 23
> fred <- fred + 1;;
val it : unit = ()
> fred;;
val it : int = 24
```

- Assignment: Variables can change value using *side effects*.
- In functional programming, side effects, (e.g. assignments) are avoided.

A First Look at F#
The F# langauge
Constants
Operators
Variables
Variable definition
Variable definition
Exercises
Garbage Collection
Tuples and Lists
Func Definitions
F# Types

Exercises

Suppose we make these F# declarations

- `let a = "123";;`
- `let b = "456";;`
- `let c = a + b + "789";;`
- `let a = 3 + 4;;`

What is the value and type of each of these expressions?

- `a;;`
- `b;;`
- `c;;`
- `a = 6;;`

A First Look at F#
The F# langauge
Constants
Operators
Variables
Variable definition
Variable definition
Exercises
Garbage Collection
Tuples and Lists
Func Definitions
F# Types

Garbage Collection

- F# runs under Common Language Runtime – the VM for .NET programs
- Garbage collection responsibility of CLR
- Reclaiming pieces of memory that are no longer being used
- We'll see much more about this when we look at C#.

A First Look at F#
The F# langauge
Constants
Operators
Variables
Tuples and Lists
Tuples
No Tuple of 1
Tuple Type
Exercises
Lists
Empty Lists
List Type
IsEmpty
@ Operator
:: Operator
Head and Tail
Exercises
Func Definitions
F# Types

Tuples

Example

```
> let barney = (1+2, 3.0*4.0, "hi");;
val barney : int * float * string = (3, 12.0, "hi")
> let pt1 = ("red", (30, 20));;
val pt1 : string * (int * int) = ("red", (30, 20))
> fst pt1;;
val it : string = "red"
> fst (snd pt1);;
val it : int = 30
```

A First Look at F#
The F# langauge
Constants
Operators
Variables
Tuples and Lists
Tuples
No Tuple of 1
Tuple Type
Exercises
Lists
Empty Lists
List Type
IsEmpty
@ Operator
:: Operator
Head and Tail
Exercises
Func Definitions
F# Types

Tuples

- *Heterogeneous*: tuples can contain mixed types
- Parentheses define tuples
- A tuple is similar to a struct in C++ but with no field names
- **fst x** is the first element of 2-tuple x, **snd x** is the second.

Example

```
> snd ("red", 50);;
val it : int = 50
```

A First Look at F#
The F# langauge
Constants
Operators
Variables
Tuples and Lists
Tuples
No Tuple of 1
Tuple Type
Exercises
Lists
Empty Lists
List Type
IsEmpty
@ Operator
:: Operator
Head and Tail
Exercises
Func Definitions
F# Types

No Tuple of 1

Example

```
> (5, 6);;
val it : int * int = (5, 6)
> (5);;
val it : int = 5
> fst (5, 6);;
val it : int = 5
> fst (5);;
error FS0001: This expression was expected
  to have type ''a * 'b' but here has type
  'int'
```

A First Look at F#
The F# langauge
Constants
Operators
Variables
Tuples and Lists
Tuples
No Tuple of 1
Tuple Type
Exercises
Lists
Empty Lists
List Type
IsEmpty
@ Operator
:: Operator
Head and Tail
Exercises
Func Definitions
F# Types

Tuple Type Constructor

- The type of a tuple gives (,) as a type constructor
- For tuple `(5,true);;`, `int * bool` is the type of pairs (x, y) where x is an **int** and y is a **bool**
- Parentheses have structural significance, each below are different:
  - `(int, (int, bool)): (5,(6, true))`
  - `((int, int), string): ((5,6), "Hi")`
  - `(int, int, bool): (5, 6, true)`
  - `((bool, int), (int, float)): ((true,4),(5,3.1))`

A First Look at F#
The F# langauge
Constants
Operators
Variables
Tuples and Lists
Tuples
No Tuple of 1
Tuple Type
Exercises
Lists
Empty Lists
List Type
IsEmpty
@ Operator
:: Operator
Head and Tail
Exercises
Func Definitions
F# Types

Exercises

What is the result for each expression?

```
1 snd (3, 4);;
2 let x = (1+2, 3.0*0.5, "zig" + "zag");;
3 x;;
4 (4, 5) = (4, 5);;
5 snd (3, 4, 5);;
6 (4, "zig") = (4, 5);;
7 (4, 5.0) = (4, 5);;
8 (3, "zig", 5.3);;
9 (3, (4, "zig"), 5.3);;
```

A First Look at F#
The F# langauge
Constants
Operators
Variables
Tuples and Lists
Tuples
No Tuple of 1
Tuple Type
Exercises
Lists
Empty Lists
List Type
IsEmpty
@ Operator
:: Operator
Head and Tail
Exercises
Func Definitions
F# Types

Lists

Example

```
> [1; 2; 3];;
val it : int list = [1; 2; 3]
> [1.0; 2.0];;
val it : float list = [1.0; 2.0]
> [1.0; "Hello"];;
error FS0001: <error>
> [[1; 2; 3]; [1; 2]];;
val it : int list list = [[1; 2; 3]; [1; 2]]
```

- *Homogeneous*: all list elements must be the same type.
- Mixing types results in an exception

A First Look at F#

The F# langauge

Constants
Operators
Variables
Tuples and Lists
Tuples
No Tuple of 1
Tuple Type
Exercises
Lists
Empty Lists
List Type
IsEmpty
@ Operator
:: Operator
Head and Tail
Exercises
Func Definitions

F# Types

Empty Lists

Example

```
> [];;
val it : 'a list = []
> let empty = [];;
val empty : 'a list
> empty = [];;
val it : bool = true
```

- Empty list is []
- **List.empty** is an *alias* for []
- `'a list` means a list of elements, type `'a`

A First Look at F#
The F# langauge
Constants
Operators
Variables
Tuples and Lists
Tuples
No Tuple of 1
Tuple Type
Exercises
Lists
Empty Lists
List Type
IsEmpty
@ Operator
:: Operator
Head and Tail
Exercises
Func Definitions
F# Types

List Type Constructor

- The type **list** is type constructor
- For example, in [5;6] the type **int list** means each element is of type int
- A list is not a tuple: [5;6;7] is not (5,6,7)

A First Look at F#
The F# langauge
Constants
Operators
Variables
Tuples and Lists
Tuples
No Tuple of 1
Tuple Type
Exercises
Lists
Empty Lists
List Type
IsEmpty
@ Operator
:: Operator
Head and Tail
Exercises
Func Definitions
F# Types

IsEmpty

Example

```
> [].IsEmpty;;
val it : bool = true
> [1;2;3].IsEmpty;;
val it : bool = false
```

- **IsEmpty** tests for the empty list
- Can also use an equality test, as in `x = []`

A First Look at F#
The F# langauge
Constants
Operators
Variables
Tuples and Lists
Tuples
No Tuple of 1
Tuple Type
Exercises
Lists
Empty Lists
List Type
IsEmpty
@ Operator
:: Operator
Head and Tail
Exercises
Func Definitions
F# Types

@ Operator

Example

```
> [1;2;3] @ [4;5;6];;
val it : int list = [1; 2; 3; 4; 5; 6]
```

- `@` operator concatenates two lists
- Both operands must be lists
- Both lists must have the same type

A First Look at F#
The F# langauge
Constants
Operators
Variables
Tuples and Lists
Tuples
No Tuple of 1
Tuple Type
Exercises
Lists
Empty Lists
List Type
IsEmpty
@ Operator
:: Operator
Head and Tail
Exercises
Func Definitions
F# Types

:: Operator

Example

```
> let n = 5::[6;7];;
val n : int list = [5; 6; 7]
> let x = 5::[];;
val x : int list = [5]
> let y = 6::x;;
val y : int list = [6; 5]
> let y = 5::6::7::[];;
val y : int list = [5; 6; 7]
```

- :: operator is a list-builder (pronounced **cons**) for *cons*tructor
- Constructs a new list by prepending an element to a list
- :: operator is right-associative

A First Look at F#
The F# langauge
Constants
Operators
Variables
Tuples and Lists
Tuples
No Tuple of 1
Tuple Type
Exercises
Lists
Empty Lists
List Type
IsEmpty
@ Operator
:: Operator
Head and Tail
Exercises
Func Definitions
F# Types

Head and Tail functions

Example

```
> let z = 1::2::[];;
val z : int list = [1; 2]
> z.Head;;
val it : int = 1
> z.Tail;;
val it : int list = [2]
> z.Tail.Tail;;
val it : int list = []
```

- The **Head** function returns the head of a list: the first element
- The **Tail** function returns the list tail: the *list* without the Head element

A First Look at F#
The F# langauge
Constants
Operators
Variables
Tuples and Lists
Tuples
No Tuple of 1
Tuple Type
Exercises
Lists
Empty Lists
List Type
IsEmpty
@ Operator
:: Operator
Head and Tail
Exercises
Func Definitions
F# Types

Exercises

What is the result for each expression?

```
1  [1;2] @ [3;4];;
2  1::2::[];;
3  [1::2::[]];;
4  (1::2::[]).Head;;
5  [[1;2];[3;4]];;
6  [[1;2];[3;4]].Tail;;
7  [5].tail;;
8  (snd ([1;2], [3;4])).Tail.Head;;
9  [(3,4),(5,6)].Head;;
10 [1;3;4].Head::2::[1;3;4].Tail;;
```

A First Look at F#

The F# langauge

Constants
Operators
Variables
Tuples and Lists
Tuples
No Tuple of 1
Tuple Type
Exercises
Lists
Empty Lists
List Type
IsEmpty
@ Operator
:: Operator
Head and Tail
Exercises
Func Definitions

F# Types

Exercises

What is the result for each expression?

1. `[].Head;;`
2. `[].tail;;`
3. `[1;2].Tail.Tail.Head;;`
4. `1 @ [2];;`
5. `[1]::[2;3];;`
6. `1::2::[].Head;;`

A First Look at F#

The F# langauge

Constants
Operators
Variables
Tuples and Lists
Tuples
No Tuple of 1
Tuple Type
Exercises
Lists
Empty Lists
List Type
IsEmpty
@ Operator
:: Operator
Head and Tail
Exercises
Func Definitions

F# Types

Exercises

Implement each expression in F#

1. Concatenate [1;2] with [3;4].
2. 1 cons'ed to [2;3].
3. Second element of [1;2;3;4].
4. Last element of [1;2;3;4].
5. 2 cons'ed to [1;3] to yield [1;2;3].

A First Look at F#
The F# langauge
Constants
Operators
Variables
Tuples and Lists
Func Definitions
Defining Functions
let keyword
Func Syntax
Func Types
Parameter Passing
Func Signatures
Exercises
Recursion
Factorial
Examples
F# Types

Defining Functions

Example

```
> let add x y = x + y;;
val add : x:int -> y:int -> int
```

- **add** – the function name
- **x y** – parameter list
- **x + y** – function result
- **add : x:int -> y:int -> int** – inferred function result type

Example (C/C++/Java equivalent)

```
int add(int x, int y)
{ return x + y; }
```

A First Look at F#
The F# langauge
Constants
Operators
Variables
Tuples and Lists
Func Definitions
Defining Functions
let keyword
Func Syntax
Func Types
Parameter Passing
Func Signatures
Exercises
Recursion
Factorial
Examples
F# Types

let keyword

Example

```
> let first (x: (int * string)):int = fst x;;
val first : int * string -> int
> first (2, "abc");;
val it : int = 2
```

- **let** defines a new function and binds to variable **first**
- **first** is an **int * string -> int** function whose argument **x** type is `int * string` and the return type is `int`
- It is not necessary to declare any return types, since F# infers them.

A First Look at F#
The F# langauge
Constants
Operators
Variables
Tuples and Lists
Func Definitions
Defining Functions
let keyword
Func Syntax
Func Types
Parameter Passing
Func Signatures
Exercises
Recursion
Factorial
Examples
F# Types

Function Definition Syntax

Definition

```
<fun-def> ::=
      let <function-name> (<parameter>:type):type =
            <expression> ;;
```

- `<function-name>` can be any legal F# name
- The simplest `<parameter>` is just a single variable name: the formal parameter of the function
- The `<expression>` is any F# expression; its value is the value the function returns
- This is not the full F# function declaration. . . more later

A First Look at F#
The F# langauge
Constants
Operators
Variables
Tuples and Lists
Func Definitions
Defining Functions
let keyword
Func Syntax
Func Types
Parameter Passing
Func Signatures
Exercises
Recursion
Factorial
Examples
F# Types

Function Type Constructor

- F# gives the type of functions using **->** as a type constructor
- For example, **int -> float** is the type of a function that takes an **int** parameter (the *domain* type) and produces a **float** result (the *range* type)
- From math: a function *maps* domain values (inputs) to range values (outputs).
- `let f (x:int):float = float (x % 4);;` maps the domain of all integers to the float range of `[0.0..3.0]`.
- The type is: `f:int -> float`

A First Look at F#
The F# langauge
Constants
Operators
Variables
Tuples and Lists
Func Definitions
Defining Functions
let keyword
Func Syntax
Func Types
Parameter Passing
Func Signatures
Exercises
Recursion
Factorial
Examples
F# Types

Parameter Passing

Example

```
> let quot (a:float) (b:float) = a / b;;
val quot : a:float -> b:float -> float
> quot 6.0 2.0;;
val it : float = 3.0
> quot 6 2;;
val it: float = 3.0
```

- Remember: Type promotion from `int` to `float` only
- Java/C#/etc. promotes `char -> int -> float`

A First Look at F#
The F# langauge
Constants
Operators
Variables
Tuples and Lists
Func Definitions
Defining Functions
let keyword
Func Syntax
Func Types
Parameter Passing
Func Signatures
Exercises
Recursion
Factorial
Examples
F# Types

Func Signatures

Example

```
> let cons a L = a::L;;
val cons : a:'a -> L:'a list -> 'a list
```

- Function name: cons
- Parameter tuple: `'a, list of 'a`
- Element one, type unknown: `'a`
- Element two, list of type unknown: `list of 'a`
- Result list type unknown: `list of 'a`

A First Look at F#
The F# langauge
Constants
Operators
Variables
Tuples and Lists
Func Definitions
Defining Functions
let keyword
Func Syntax
Func Types
Parameter Passing
Func Signatures
Exercises
Recursion
Factorial
Examples
F# Types

Exercises

What is the result for each function call?

```
let fa (x:int) = x + 1;;
fa 3;;
let fb (x:'a) (y:'b) = x;;
fb [1;2;3] 4;;
fb 3 4;;
let fc (x:'a list) = x.Tail;;
fc [1; 2; 3];;
let fd (x:'a list) (y:'a list) =
x.Head::y.Tail;;
fd [1;2;3] [4;5;6];;
let fe (x:int list) = x.Head + 1;;
fe [3;2;1];;
```

A First Look at F#

The F# langauge

Constants
Operators
Variables
Tuples and Lists
Func Definitions
Defining Functions
let keyword
Func Syntax
Func Types
Parameter Passing
Func Signatures
Exercises
Recursion
Factorial
Examples

F# Types

Exercises

What is the return type for each function?

- `let fa (x:int) = x + 1;;`
- `let fb (x:'a) (y:'b) = x;;`
- `let fc (x:'a list) = x.Tail;;`
- `let fd (x:'a list, y:'a list) = x.Head :: y.Tail;;`
- `let fe (x:int) = x.Head + 1;;`

A First Look at F#
The F# langauge
Constants
Operators
Variables
Tuples and Lists
Func Definitions
Defining Functions
let keyword
Func Syntax
Func Types
Parameter Passing
Func Signatures
Exercises
Recursion
Factorial
Examples
F# Types

Recursion

- Functional languages are characterized by use of recursion for repetition
- Functional languages such as F# minimize assignment hazard of side effects
- Repetition (e.g. for, while, do while, etc.) requires assignment

Example (C++ loop)

```
for (i=0; i<10; i=i+1)
  cout << i;
```

A First Look at F#
The F# langauge
Constants
Operators
Variables
Tuples and Lists
Func Definitions
Defining Functions
let keyword
Func Syntax
Func Types
Parameter Passing
Func Signatures
Exercises
Recursion
Factorial
Examples
F# Types

Recursive factorial function

Example

```
> let rec fact n =
    if (n = 0) then 1 else n * fact (n-1);;
    // call a f'n with space-separated parameters
val fact : n:int -> int
> fact 5;;
val it : int = 120
```

Many recursive functions consist of a pattern of two steps:

- Test for base case, terminating condition:
  - `if (n = 0) then 1`
- Recurse, moving closer to terminating condition:
  - `else n * fact (n-1)`

A First Look at F#
The F# langauge
Constants
Operators
Variables
Tuples and Lists
Func Definitions
Defining Functions
let keyword
Func Syntax
Func Types
Parameter Passing
Func Signatures
Exercises
Recursion
Factorial
Examples
F# Types

Exercises

Write the recursive functions in F#

```
int Fib( int n ) {
  if ( n <= 1 ) return n;
  else return Fib(n-1) + Fib(n-2);
}

double interest(rate, principle, year) {
  if (year == 0)
    return principle;
  else return
    interest(rate, (1.0+rate) * principle, year-1);
}
```

A First Look at F#
The F# langauge
Constants
Operators
Variables
Tuples and Lists
Func Definitions
Defining Functions
let keyword
Func Syntax
Func Types
Parameter Passing
Func Signatures
Exercises
Recursion
Factorial
Examples
F# Types

Recursive int list summation

Example

```
> let rec summation (x:int list) =
    if (x.IsEmpty) then 0
    else x.Head + summation (x.Tail);;
val summation : x:int list -> int

> summation [1;2;3;4;5];;
val it : int = 15
```

Common recursive function pattern for list arguments:

- base case test for empty list:
  - `if (x.isEmpty) then 0`
- recursive call moving closer to base case of `x.isEmpty`:
  - `summation(x.Tail)`

A First Look at F#
The F# langauge
Constants
Operators
Variables
Tuples and Lists
Func Definitions
Defining Functions
let keyword
Func Syntax
Func Types
Parameter Passing
Func Signatures
Exercises
Recursion
Factorial
Examples
F# Types

Recursive list length

Example

```
> let rec length (x:'a list) =
    if (x.IsEmpty) then 0
    else 1 + length x.Tail;;
val length : x:'a list -> int
> length [true;false;true];;
val it : int = 3
> length [1;2;3];;
val it : int = 3
```

- Function to compute the list length is predefined in F#.
- Note type length : `x:'a list -> int` works on any type of list.
- *Polymorphic*, while summation operates on int lists only.

Recursive last element in list

Example

```
> let rec last (L:'a list) =
    if (L.Tail = []) then L.Head
    else last L.Tail;;
val last : L:'a list -> 'a
> last [2;4;6;8;10];;
val it : int = 10
```

Notice recursion pattern of:

- Test base case `L.Tail=[]`, return base case value `L.Head`
  1. One element remaining in `L`
  2. Recurse `last(L.Tail)`, moving closer to base case

A First Look at F#
The F# langauge
Constants
Operators
Variables
Tuples and Lists
Func Definitions
Defining Functions
let keyword
Func Syntax
Func Types
Parameter Passing
Func Signatures
Exercises
Recursion
Factorial
Examples
F# Types

Recursive n^{th} element in list

Example

```
> let rec Nth n (L:'a list) =
    if (n = 1) then L.Head
    else Nth (n-1) L.Tail;;
val Nth : n:int -> L:'a list -> 'a
> Nth 2 [true;false;true];;
val it : bool = false
> Nth 2 [("a", 4); ("b", -2)];;
val it : string * int = ("b", -2)
```

- Type `n:int -> L:'a list -> 'a` works on any type of list, *polymorphic* on the list parameter.
- Fails when list has less than n elements.

A First Look at F#

The F# langauge

Constants
Operators
Variables
Tuples and Lists
Func Definitions
Defining Functions
let keyword
Func Syntax
Func Types
Parameter Passing
Func Signatures
Exercises
Recursion
Factorial
Examples

F# Types

Recursive list identity

Example

```
> let rec identity (L:'a list) =
    if (L.IsEmpty) then []
    else L.Head::identity(L.Tail);;
val identity : L:'a list -> 'a list
> identity [1;2;3];;
val it : int list = [1; 2; 3]
```

Notice recursion pattern of:

- Test for base case `L.IsEmpty`, return base case value `[]`
- Recurse `identity(L.Tail)`, moving closer to base case

A First Look at F#
The F# langauge
Constants
Operators
Variables
Tuples and Lists
Func Definitions
Defining Functions
let keyword
Func Syntax
Func Types
Parameter Passing
Func Signatures
Exercises
Recursion
Factorial
Examples
F# Types

Recursive all-but-last element

Example

```
> let rec allbutthelast (L:'a list) =
    if (L.Tail=[]) then []
    else L.Head::allbutthelast L.Tail;;
val allbutthelast : L:'a list -> 'a list
allbutthelast [1;2;3];;
  val it : int list = [1; 2]
```

Notice recursion pattern of:

- Test for base case `L.Tail=[]`, return base case value `[]`
- Recurse `allbutthelast L.Tail`, moving closer to base case

A First Look at F#
The F# langauge
Constants
Operators
Variables
Tuples and Lists
Func Definitions
Defining Functions
let keyword
Func Syntax
Func Types
Parameter Passing
Func Signatures
Exercises
Recursion
Factorial
Examples
F# Types

Recursive list reverse

Example

```
> let rec reverse (L:'a list) =
    if (L=[]) then []
    else reverse(L.Tail)@[L.Head];;
val reverse : L:'a list -> 'a list
> reverse [1;2;3];;
val it : int list = [3; 2; 1]
```

Notice recursion pattern of:

- Test for base case `L=[]`, return base case value `[]`
- Recurse `reverse(L.Tail)`, moving closer to base case

F# Types

F# Types So Far

- So far we have the primitive F# types **int**, **float**, **bool**, **char**, and **string**
- Also we have three type constructors:
  - Tuple types using **(,)**
  - List types using **[;]**
  - Function types