COMP 302 Winter 2016 Lecture 2
Introduction to F#

Prakash Panangaden¹

¹School of Computer Science
McGill University

McGill University, Montréal, January 2016
What is F#?

1. A *functional* language but it also has imperative, object-oriented and other features.
What is F#?

1. A *functional* language but it also has imperative, object-oriented and other features.

2. Based on SML, but with some syntactic changes and integrated into the .NET framework.
What is F#?

1. A functional language but it also has imperative, object-oriented and other features.
2. Based on SML, but with some syntactic changes and integrated into the .NET framework.
What is F#?

1. A *functional* language but it also has imperative, object-oriented and other features.
2. Based on SML, but with some syntactic changes and integrated into the .NET framework.
3. Standard *meta* language *not* standard *markup* language.
4. Invented by Robin Milner in the early 1970s as a *meta*language for his proof development system LCF.
What is F#?

1. A *functional* language but it also has imperative, object-oriented and other features.
2. Based on SML, but with some syntactic changes and integrated into the .NET framework.
4. Invented by Robin Milner in the early 1970s as a *meta* language for his proof development system LCF.
5. It is a *polymorphically-typed* higher-order programming language.
Basic Types

1. Integers
2. Booleans
3. Characters and strings
4. Floats
Compound Types

1. Unit
Compound Types

1. Unit
2. Tuples
Compound Types

1. Unit
2. Tuples
3. Records
Compound Types

1. Unit
2. Tuples
3. Records
4. Lists: an inductively defined type
Compound Types

1. Unit
2. Tuples
3. Records
4. Lists: an inductively defined type
5. User-defined types
Compound Types

1. Unit
2. Tuples
3. Records
4. Lists: an inductively defined type
5. User-defined types
6. User-defined inductive types
Compound Types

1. Unit
2. Tuples
3. Records
4. Lists: an inductively defined type
5. User-defined types
6. User-defined inductive types
7. Function types
Compound Types

1. Unit
2. Tuples
3. Records
4. Lists: an inductively defined type
5. User-defined types
6. User-defined inductive types
7. Function types
8. Classes and other OO features.
Starting up

F# Interactive for F# 3.1 (Open Source Edition)
Freely distributed under the Apache 2.0 Open Source License
For help type #help;;
> 2 + 3;;
> val it : int = 5
> -3;;
val it : int = -3
1.23;;
val it : float = 1.23
> 1.23e-7;;
val it : float = 1.23e-07
> System.Math.PI;;
val it : float = 3.141592654
An interactive session

Some basic arithmetic

> 17 / 5;;
val it : int = 3
> 17 % 5;;
val it : int = 2
> 1/2.0;;
error FS0001: The type 'float' does not match the type 'int'
> 1.0/2.0;;
val it : float = 0.5
Booleans and Unit

> true ;;
val it : bool = true

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

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

> not true ;;
val it : bool = false

> ();;
val it : unit = ()
Char and Strings

>`'a';; val it : char = 'a`
>`"a"';; val it : string = "a"
>`System.Char.IsLower 'a'`;; val it : bool = true
>`String.length "MacBeth"`;; val it : int = 7
>`"Is this a dagger I see before me?"`;; val it : string = "Is this a dagger I see before me?"
>`"Come let me" + " clutch thee."`;; val it : string = "Come let me clutch thee."
>`it.[3]`;; val it : char = 'e'
Bindings

> let result = 2 + 3;;
val result : int = 5

> let result =
  let x = 1
  let y = 2
  let z = x + y
  x + y + z
val result : int = 6

Beware of F# indentation rules!
Learn about verbose vs lightweight syntax.
Functions

> let inc n = n + 1
val inc : n:int -> int
> let sum_squares n m = n * n + m * m
val sum_squares : n:int -> m:int -> int
> sum_squares 3 4;;
val it : int = 25
> let sum_squares_pair(n,m) = n * n + m * m
val sum_squares_pair : n:int * m:int -> int
> sum_squares_pair 3 4;;
error FS0003: This value is not a function
> sum_squares_pair(3,4);;
val it : int = 25
> sum_squares 3;;
val it : (int -> int) = <fun:it@19-1>
> it 4;;
val it : int = 25
Recursive Functions

```fsharp
let rec fact n =
    if n = 0 then 1 else n * fact(n-1)

let rec fastfact(n,m) =
    if n = 0 then m
    else fastfact(n-1,n * m)

let iterfact n =
    let rec helper(x,y) =
        if x = 0 then y
        else
            helper(x-1,x*y)
    in
    helper(n,1)
```
let delta = 0.0001

let square u:float = u * u

let close guess x = (abs((square guess) - x)) < delta

let update guess x = (guess + x/guess)/2.0

let rec sqrt guess x =
  if (close guess x) then guess
  else (sqrt (update guess x) x)
let rec mysqrt(x : float, guess: float, delta: float) =
  let close(guess, x) =
    (abs((square guess) - x)) < delta in
  let update(guess, x) = (guess + x/guess)/2.0 in
  if close(guess, x) then guess
  else
    mysqrt(x, update(guess, x), delta)

Note the use of in; verbose syntax.
An interactive session

Square roots in action

val delta : float = 0.0001
val square : u:float -> float
val close : guess:float -> x:float -> bool
val update : guess:float -> x:float -> float
val sqrt : guess:float -> x:float -> float
> sqrt 1.0 17.0;;
val it : float = 4.123106717
val mysqrt : x:float * guess:float * delta:float -> fl
Pitfalls

let rec rpe(b,power) =
    if (b = 0) then 0
    elif (power = 0) then 1
    elif (power = 1) then b
    elif (power % 2 = 1) then b * rpe(b,power-1)
    else
        let temp = rpe(b, power/ 2) in
        temp * temp

Why not the following?

    else rpe(b, power/ 2) * rpe(b, power/ 2)