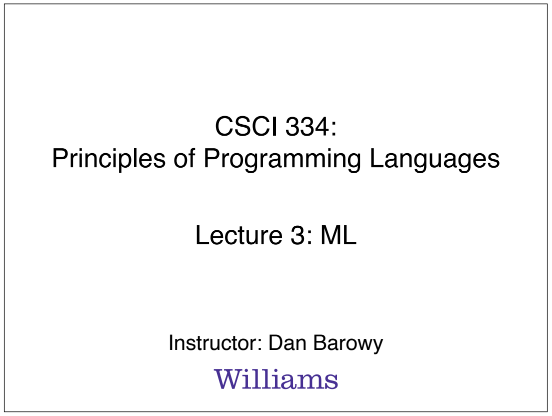


CSCI 334:
Principles of Programming Languages

Lecture 3: ML

Instructor: Dan Barowy

Williams

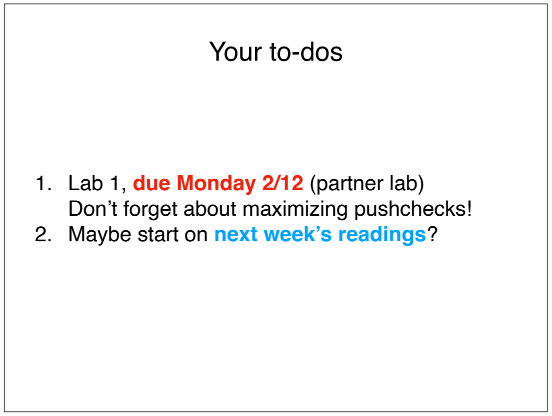
1

Topics

ML family of languages

F#

2



Your to-dos

1. Lab 1, **due Monday 2/12** (partner lab)
   Don't forget about maximizing pushchecks!
2. Maybe start on **next week's readings**?

3

Announcements

- CS Colloquium **today @ 11:45pm in Wege Auditorium (TCL 123)**

Laura South (Northeastern University)

Making Social Digital Platforms Accessible for People with Photosensitive Epilepsy

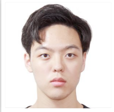
Participating in social digital spaces can be dangerous for people with photosensitive epilepsy (PSE), as they risk encountering flashing or strobing content in these social spaces that are capable of triggering seizures. Given the severe medical consequences of seizures, it is fundamental that digital platforms have protective systems in place to control the spread of content with hazardous strobing or flashing light sequences. However, identifying and handling seizure-inducing content on digital platforms is a challenging problem due to variability in the source and behavior of strobing sequences, as well as differences in sensitivity among people who have PSE. In this talk, I will describe my work on building theory for ensuring photosensitive accessibility in social online spaces, developing prototype systems for identifying and blocking seizure-inducing content, and conducting interviews with people with PSE to better understand accessibility needs.

4

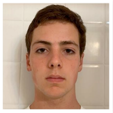
Announcements

- Another colloquium this **Friday, Feb 9** in Wege Auditorium at 2:35pm.
  Senior Thesis Proposals, Part 2

Ye Shu



Yufeng Wu



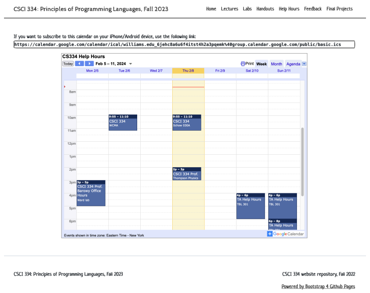
Alex Atherton

Friedrich Qiu

5

Announcements

- Reminder: where to find TA and office hours



CSCI 334: Principles of Programming Languages, Fall 2023

If you want to subscribe to this calendar on your iPhone/Android device, use the following link:

```
https://calendar.google.com/calendar/ical/williams.edu_6jehc8a6u6f4itst4h2a3pqemk%40group.calendar.google.com/public/basic.ics
```

CS334 Help Hours

Mon 2/5: 3p – 5p CSCI 334 Prof. Barowy Office Hours, Ward lab

Tue 2/6: 9:55 – 11:10 CSCI 334, WCMA

Thu 2/8: 9:55 – 11:10 CSCI 334, Schow 030A

Thu 2/8: 2p – 3p CSCI 334 Prof. Thompson Physics

Sat 2/10: 4p – 6p TA Help Hours, TBL 301

Sun 2/11: 4p – 6p TA Help Hours, TBL 301

Sun 2/11: 6p – 8p TA Help Hours

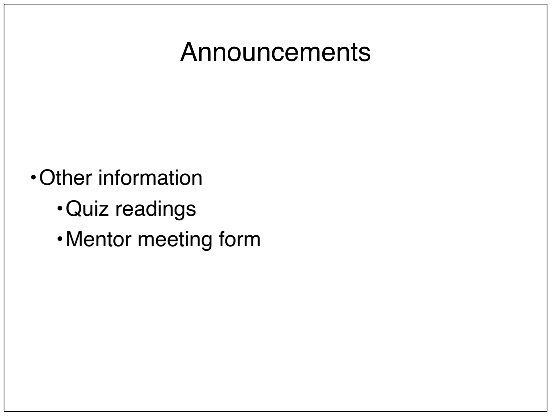
Events shown in time zone: Eastern Time - New York

CSCI 334: Principles of Programming Languages, Fall 2023

CSCI 334 website repository, Fall 2022

Powered by Bootstrap 4 Github Pages

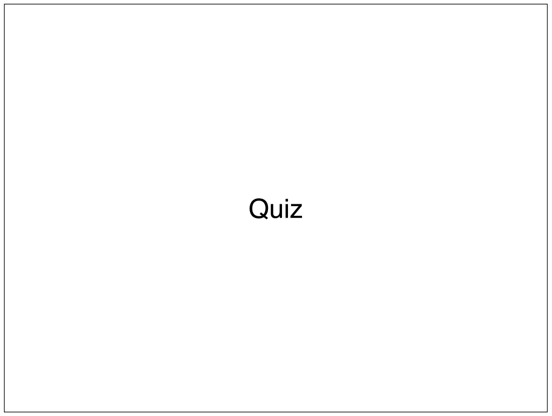
6



Announcements

- Other information
  - Quiz readings
  - Mentor meeting form

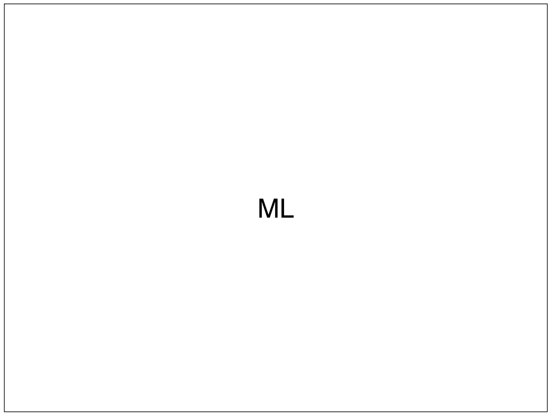
7



Quiz

8

Today we are going to talk about a family of programming languages, called “ML.” Note that this is a different “ML” than the term that refers to machine learning.



ML

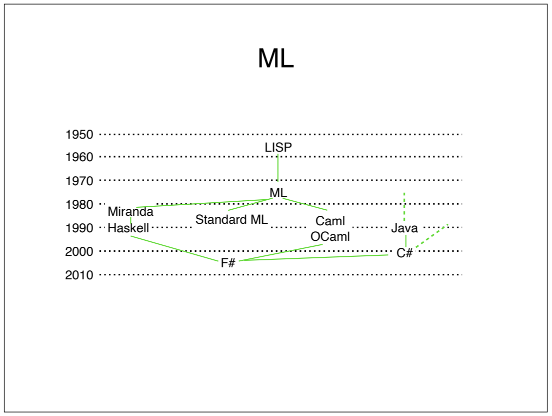
9

Today we are going to talk about a family of programming languages, called “ML.” Note that this is a different “ML” than the term that refers to machine learning.

“Free your mind”

10

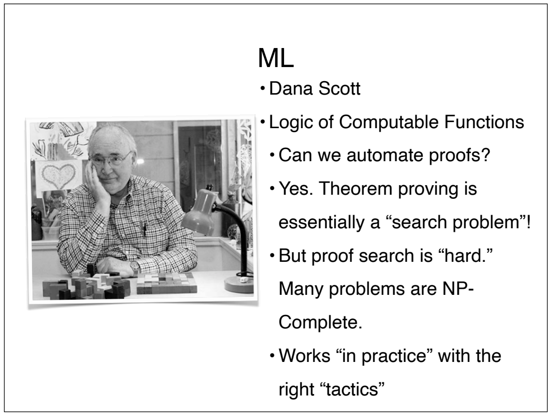
Before we start, I want you to free your mind. Learning ML requires you to do some mind bending things sometimes. Be prepared not to get it right the first time. Be like Neo.



ML

11

We now think of ML as a family of languages, but originally there was just one. It was strongly influenced by LISP, which we will also touch on this semester. But many others were inspired by ML, and created new languages that added many new features. We will primarily spend our time learning F#, which is most directly influenced by Haskell, OCaml, and C#. I really love F#, and I hope you enjoy it too.

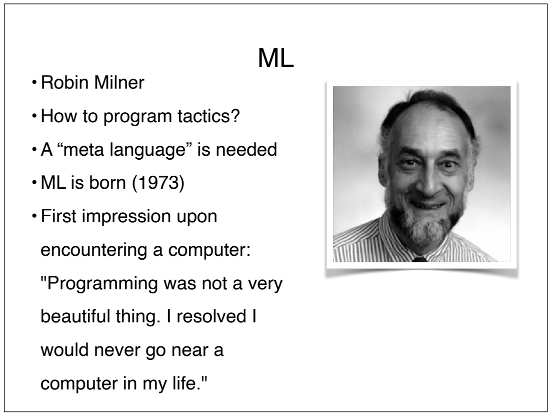


ML

- Dana Scott
- Logic of Computable Functions
  - Can we automate proofs?
  - Yes. Theorem proving is essentially a “search problem”!
  - But proof search is “hard.” Many problems are NP-Complete.
  - Works “in practice” with the right “tactics”

12

So where did ML come from? It was not born in a vacuum. Like many languages, it was created to solve a specific problem: can we write computer programs that automatically prove (mathy) things for us?

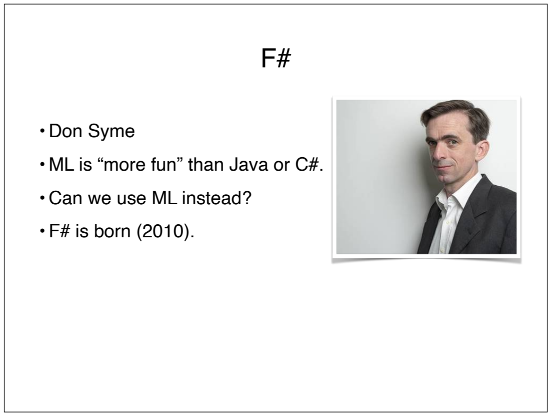


ML

- Robin Milner
- How to program tactics?
- A “meta language” is needed
- ML is born (1973)
- First impression upon encountering a computer: "Programming was not a very beautiful thing. I resolved I would never go near a computer in my life."

13

ML stands for “meta language.” Proofs, when automated, are themselves programs. So if you are generating proofs, you are generating programs, which is “meta.”

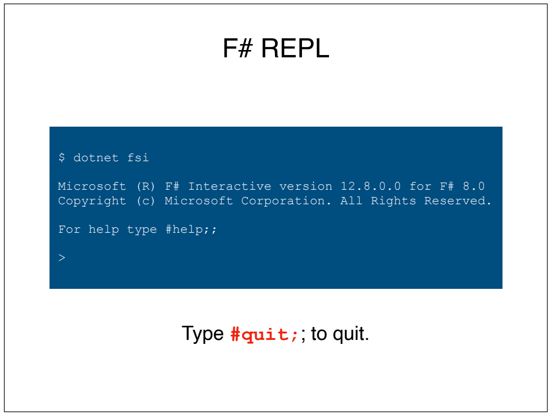


F#

- Don Syme
- ML is “more fun” than Java or C#.
- Can we use ML instead?
- F# is born (2010).

14

F# is a modern reinvention of ML for the .NET runtime produced by Microsoft. Unlike ML, it can be used with pre-existing codebases written in C# (e.g., many video games).



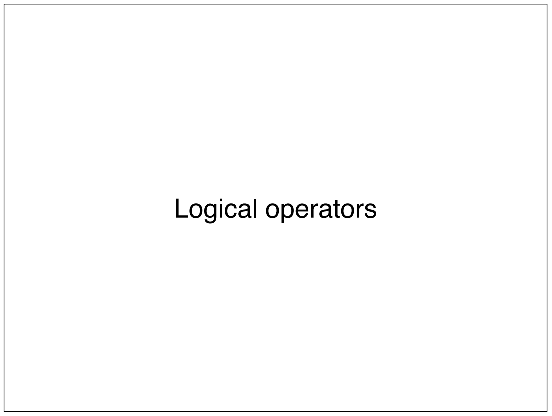
F# REPL

```
$ dotnet fsi

Microsoft (R) F# Interactive version 12.8.0.0 for F# 8.0
Copyright (c) Microsoft Corporation. All Rights Reserved.

For help type #help;;

>
```

Type `#quit;`; to quit.

15

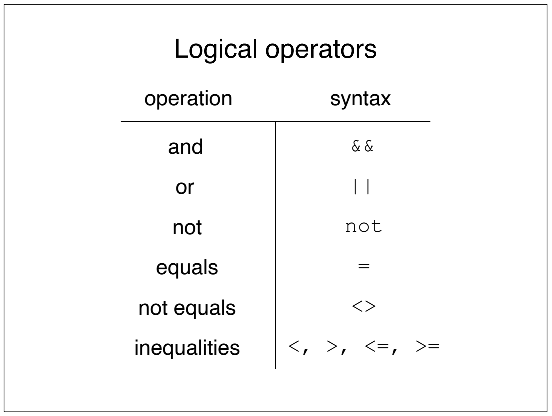
F# has an interactive command-line environment called a REPL (read-eval-print-loop). I strongly encourage you to get comfortable playing with with F# REPL. It will save you a lot of time and it will make experimentation be easy and fun.



Logical operators

16

Let's start with a few operations that trip people up when they first begin: logical operations.

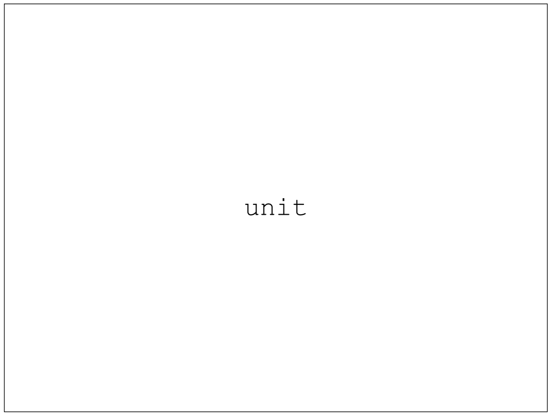


Logical operators

operation	syntax
and	`&&`
or	`||`
not	`not`
equals	`=`
not equals	`<>`
inequalities	`<, >, <=, >=`

17

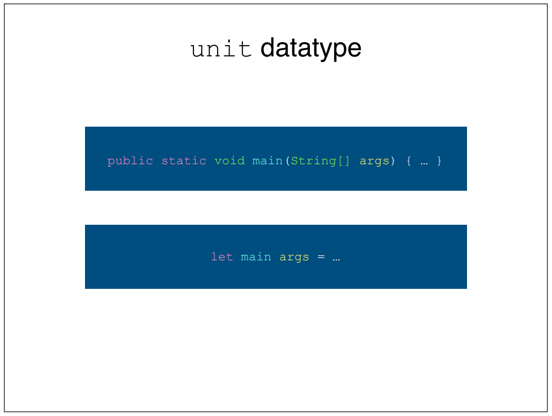
These work the same way as logical operators in other languages, except that they look different. Also, secretly, operators are just functions, as they should be.



`unit`

18

Because in F# everything is an expression, we need a way to express the idea that a function may return nothing. For that, we have a special value called "unit."



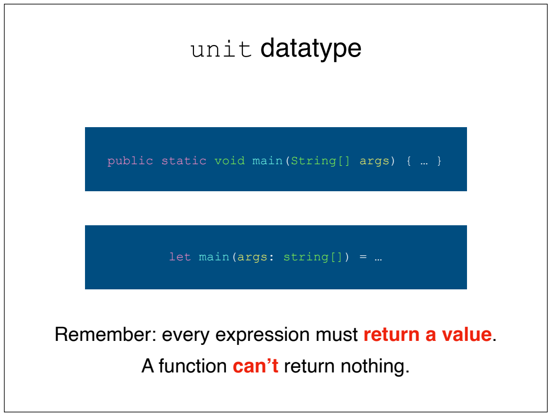
`unit` datatype

```
public static void main(String[] args) { … }
```

```
let main args = …
```

19

Recall that the main function in Java returns nothing. How do we express the same concept in F# if all expressions must return something?



`unit` datatype

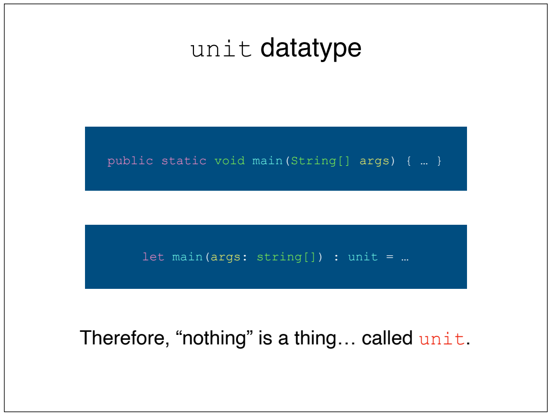
```
public static void main(String[] args) { … }
```

```
let main(args: string[]) = …
```

Remember: every expression must **return a value**.

A function **can't** return nothing.

20



`unit` datatype

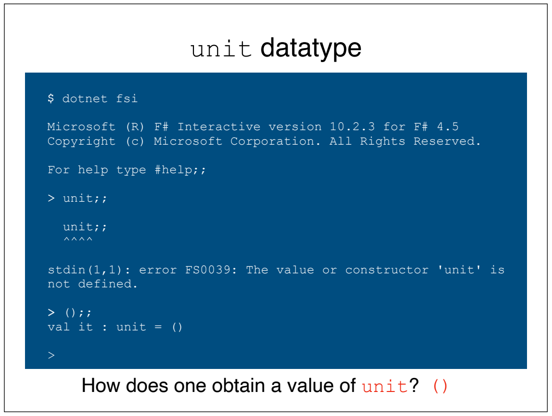
```
public static void main(String[] args) { … }
```

```
let main(args: string[]) : unit = …
```

Therefore, "nothing" is a thing… called `unit`.

21

Short answer: make a nothing a something.



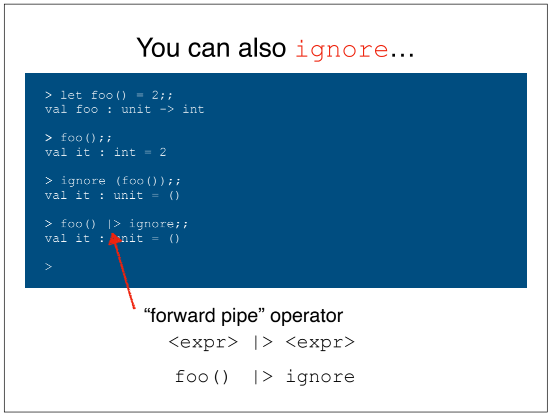
unit datatype

```
$ dotnet fsi

Microsoft (R) F# Interactive version 10.2.3 for F# 4.5
Copyright (c) Microsoft Corporation. All Rights Reserved.

For help type #help;;

> unit;;

  unit;;
  ^^^^

stdin(1,1): error FS0039: The value or constructor 'unit' is
not defined.

> ();;
val it : unit = ()

>
```

How does one obtain a value of unit? ()

22

Note that there is a difference between the type unit and the one value of unit, (). You already know the distinction between types and their values. E.g., 4 is an int, and “hello” is a string. () is a unit, but unlike int and string, there is only a single valid value for unit.



You can also ignore…

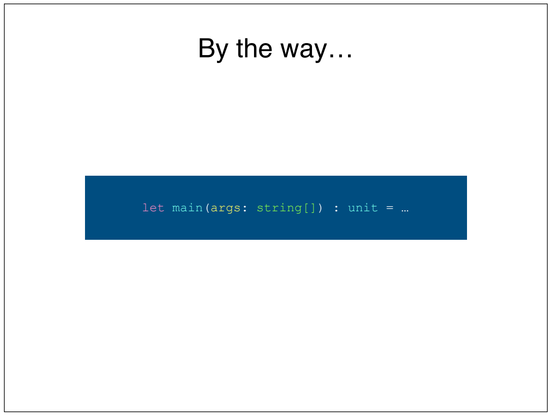
```
> let foo() = 2;;
val foo : unit -> int

> foo();;
val it : int = 2

> ignore (foo());;
val it : unit = ()

> foo() |> ignore;;
val it : unit = ()

>
```

“forward pipe” operator

```
<expr> |> <expr>

foo()  |> ignore
```

23

Another function called “ignore” allows you to “throw away” a value returned by a function. It replaces that value with unit. I am also showing my favorite F# operator here, which is called “forward pipe.” If you’ve ever used pipes in the unix shell, forward pipe should be familiar.

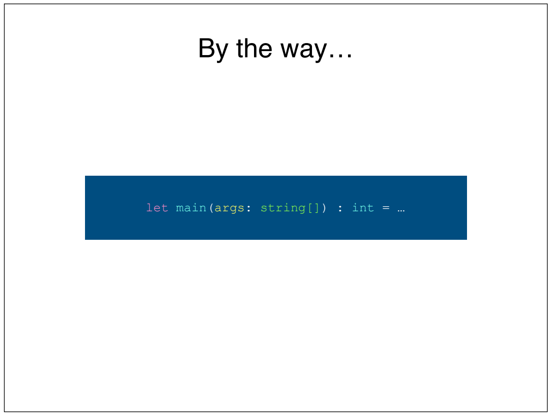


By the way…

```
let main(args: string[]) : unit = …
```

24

I used this example before, but…

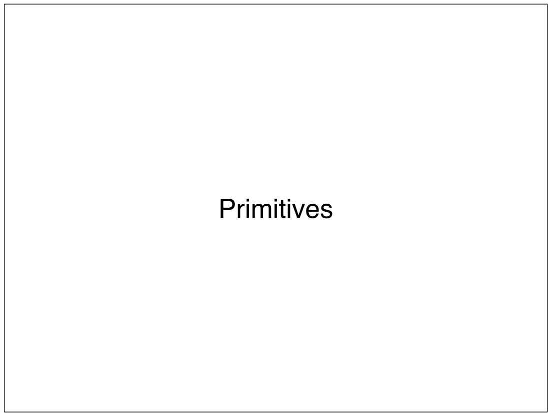


By the way…

```
let main(args: string[]) : int = …
```

25

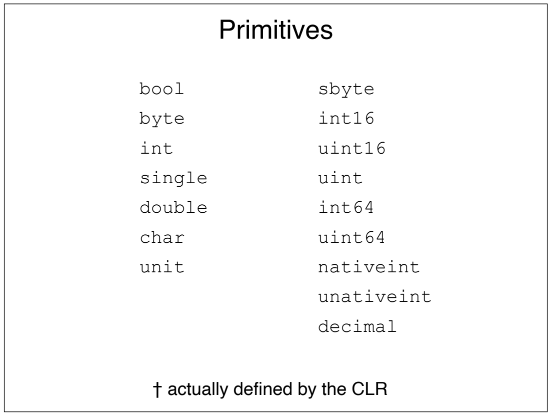
… to be more precise, F# requires that main methods return int. That integer signals to the OS whether the program succeeded or failed. 0 means success, nonzero means fail. Programs can define any number between 1-255 to have a specific error reason.



Primitives

26

Recall from your previous semesters of computer programming that we usually start by talking about the the kinds of data. Actually, we really do this because we want to start with the indivisible, most basic parts of the language, and for most languages, that happens to be kinds of data. More generally, though, any fundamental concept can be a kind of primitive in a language.



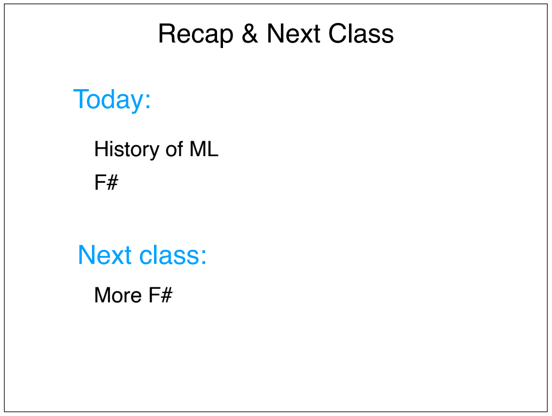
Primitives

```
bool
byte
int
single
double
char
unit
sbyte
int16
uint16
uint
int64
uint64
nativeint
unativeint
decimal
```

† actually defined by the CLR

27

F# inherits its primitive data types from C#, since they are designed to interoperate.



Recap & Next Class

Today:

History of ML
F#

Next class:

More F#