What is a functional language?

A functional language:

- defines computations as mathematical functions
- avoids mutable state

State: the information maintained by a computation

Mutable: can be changed (antonym: *immutable*)

Functional vs. imperative

Functional languages:

- Higher level of abstraction
- Easier to develop robust software
- Immutable state: easier to reason about software

Imperative languages:

- Lower level of abstraction
- Harder to develop robust software
- Mutable state: harder to reason about software

You don't have to believe me now.
If you master a functional language, you will. ☺

Imperative programming

Commands specify how to compute by destructively changing state:

```
x = x+1;
a[i] = 42;
p.next = p.next.next;
```

Functions/methods have **side effects**:

```
int wheels(Vehicle v) {
  v.size++; return v.numWheels;
}
```

Mutability

The fantasy of mutability:

- There is a single state
- The computer does one thing at a time

The reality of mutability:

- There is no single state
  - Programs have many threads, spread across many cores, spread across many processors, spread across many computers… each with its own view of memory
- There is no single program
  - Most applications do many things at one time

…mutable programming is not well-suited to modern computing!

Functional programming

Expressions specify what to compute

- Variables never change value
- Functions never have side effects

The reality of immutability:

- No need to think about state
- Powerful ways to build concurrent programs

Functional languages predict the future

- Garbage collection
  Java [1995], LISP [1958]
- Generics
  Java 5 [2004], ML [1990]
- Higher-order functions
  C#3.0 [2007], Java 8 [2014], LISP [1958]
- Type inference
  C++11 [2011], Java 7 [2011] and 8, ML [1990]
- **What's next?**

Functional languages in the real world

- F#, C# 3.0, LINQ (Microsoft)
- Scala (Twitter, LinkedIn, FourSquare)
- Java 8
- Haskell (dozens of small companies/teams)
- Erlang (distributed systems, Facebook chat)
- OCaml (Jane Street)

Example 1: Sum Squares

```
// returns: Σ1<=i<=n i2
int sum_squares(int n) {
  sum=0;
  for (int x = 1; x <= n; x++) {
    sum = sum + x*x
  }
  return sum;
}
```

How can you do that without mutability?

Example 1: Sum Squares

```
// returns: Σ1<=i<=n i²
int sum_squares(int n) {
  if (n==0) {
    return 0;
  } else {
    return n*n + sum_squares(n-1)
  }
}
```

Example 2: Reverse List

```
// return a copy of x,
// with the order of its elements reversed
List reverse(List x) {
   List y = null;
   while (x != null) {
      List t = x.next;
      x.next = y;
      y = x;
      x = t;
   }
   return y;
}
```

Example 2: Reverse List

```
(* return the reverse of lst *)
let rec reverse lst =
  match lst with
  | [] -> []
  | h::t -> (reverse t) @ [h]
```

This is not the most efficient algorithm

Example 3: Quicksort

- Describe quicksort in English.
- Describe quicksort in Java. (No.)
- Describe quicksort in OCaml:

```
(* returns lst sorted according to < *)
let rec qsort lst =
   match lst with
   | [] -> []
   | pivot::rest -> (* poor choice of pivot *)
      let (left,right) = partition ((<) pivot) rest
      in (qsort left) @ [pivot] @ (qsort right)
```

But definitely don't use this exact algorithm

OCaml

A pretty good language for writing beautiful programs

O = Objective, Caml=not important
ML is a family of languages; originally the "meta-language" for a tool