Functional v.s Imperative Programs

- Imperative language
  - Basic constructs are *imperative statements*
    - ◊ Change existing values, i.e., the "state" of a program
    - ◊ e.g., x := x + 1
  - Examples: C, Java, Fortran, Modula, Ada, Pascal, Algol, ...
- Functional language
  - Basic constructs are declarative
    - ◊ Declare new values
    - ◊ e.g., function f(int x) { return x+1 }
  - Computation proceeds primarily by evaluating expressions
  - "*Pure*" if all constructs are strictly declarative, i.e., *No side-effects*
  - Examples: Lisp/Scheme, ML, Miranda, Haskell, ...

No side-effects languages test

Within the scope of specific declarations of $x_1, x_2, \ldots, x_n$, all occurrences of a given expression containing only $x_1, x_2, \ldots, x_n$ necessarily produce the same value.

Example

```
int x=3;
int y=4;
int z = 5*(x+y)−3;
...    // no new declaration of x or y //
integer w = 4*(x+y)+z;
```

Turing lecture highlights functional programming

- Designer of Fortran, BNF, etc.
- Turing Award in 1977
  - Functional programming better than imperative programming
  - Easier to reason about functional programs
  - More efficient due to parallelism
  - Algebraic laws
    - ◊ Reason about programs
    - ◊ Optimizing compilers

John Backus