- **History**
  - Fortran
    - 1954–1957, John Backus at IBM
    - Fortran I compiler took 18 man-years to write
    - Arrays, three-way arithmetic if, goto, do loops, I/O, complex numbers
    - Alternative to assembly language
      - Freed the programmer from:
        - Coding mathematical expressions
        - Where in memory to place variables
      - Compute by modifying the state of variables
      - Branches similar to assembly language
      - Static storage => no recursion
  - Many later languages have followed this imperative style
    - Algol, Pascal, C, COBOL, ...
  - BASIC
    - Similar to Fortran in many ways, also imperative
    - Main contribution: Interactive, interpretive style
      - In constrast, even the first Fortran compile was an optimizing compiler
        - Efficiency of the computer was very important
        - Fortran would not have been accepted if it had generated slow code
      - BASIC had other priorities
    - John Kemeny and Thomas Kurtz at Dartmouth, 1963
    - Design principles
      - Be easy for beginners to use.
      - Be a general-purpose programming language.
      - Allow advanced features to be added for experts while keeping the language simple for beginners.
      - Be interactive.
      - Provide clear and friendly error messages.
      - Respond quickly for small programs.
      - Not require an understanding of computer hardware.
      - Shield the user from the operating system.
  - Lisp
    - John McCarthy, 1957
    - Not intended as a programming language
      - Not intended as a way to tell the computer what to do
      - Based on Lambda calculus (Church, 1936)
      - Theoretical exercise: "Recursive Functions of Symbolic Expressions"

        > “Another way to show that Lisp was neater than Turing machines was to write a universal Lisp function and show that it is briefer and more comprehensible than the description of a universal Turing machine. This was the Lisp function eval..., which computes the value of a Lisp expression.... Writing eval required inventing a notation representing Lisp functions as Lisp data, and such a

notation was devised for the purposes of the paper with no thought that it would be used to express Lisp programs in practice." —John McCarthy

- Steve Russell went ahead and implemented it

"Steve Russell said, look, why don't I program this eval..., and I said to him, ho, ho, you're confusing theory with practice, this eval is intended for reading, not for computing. But he went ahead and did it. That is, he compiled the eval in my paper into [IBM] 704 machine code, fixing bugs, and then advertised this as a Lisp interpreter, which it certainly was. So at that point Lisp had essentially the form that it has today...." —John McCarthy

- Far removed from how the computer actually works
  - Functions, values, name binding
- Lisp led to a whole field of functional languages
  - Pure functional languages, such as Haskell, don't have variables.
  - Lisp is not a pure functional language; it also has assignment, so you can write in an imperative style when needed.
- Logic Programming and Prolog
  - PROgrammation en LOGique
  - Alain Colmerauer, Philippe Roussel and Robert Kowalski (1972)
  - Express programs in terms of first-order predicate logic, not according to how the computer should solve it.
  - More about Prolog later.
- **This course is not supposed to be about the history of programming languages, so we'll stop on this note.**
  - BASIC and Lisp opened up the design space for programming languages.
  - Just because today's computers work by modifying the state of registers and memory locations doesn't mean that that's how a programming language has to be.
- **Learning about programming languages more important now than ever because you now have greater opportunity to choose your language**
  - Compiler technology -> better performance from high-level languages
  - Computer technology -> low-lever languages not so good anymore
    - Linux networking code randomized
    - OCaml benchmarks
  - Web deployment -> now possible to use "strange" languages and to use more than one language
  - But in another sense it has always been important
    - Alan Perlis: "A language that doesn't affect the way you think about programming is not worth knowing."
    - Even if you were stuck writing Java, you might learn something from Smalltalk, Lisp, Scheme, ML, Haskell, Forth, or Prolog
- **Goals of this course**
  - Explore the design space of programming languages
    - Learn about recurring concepts
      - Assuming that you know programming in Java
      - Light on object-orientation, etc. (known from Java)

    - Immersion in a pure functional language, Haskell (project)
- **What this course is not**
  - Learning any particular language in detail
    - But you will learn quite a bit of Haskell as a side effect of learning about functional programming.
  - Advocacy
    - Usability of a language depends on many details
    - We don't cover any specific languages in great detail
    - Xenophobia—anything that's different is scary
  - Compiler design and optimization
    - Will write a compiler
    - Working, not optimized code
- **Logistics**
  - Active style
    - Very few slides
    - Notes after class
  - See syllabus, information about office hours, etc. on the web page, http://www.cs.ucsb.edu/~cs162/
  - Textbooks
  - Homeworks
    - Optional, but highly recommended because they
      - Check that you keep up with the material
      - Detect holes in your understanding
      - Help you prepare for the midterm and final
  - Projects
    - Large Haskell project (7 weeks, starting next week)
    - Small Prolog project
  - Exams
    - Midterm and final
  - Quizzes
    - Unannounced
    - Will only be used if you don't prove your understanding in class
  - Grading
    - Midterm 20%
    - Projects 45%
    - Final and quizzes/participation 35%
  - Cheating
  - Announcement
    - Web page
    - Mailing list
    - Short office hours today
    - No discussion this week

- No TA office hours this week
- Baby -> cancelled class, will be made up