Programming Languages

Stephen Marz

COSC365

Topics

- Classification
- Very Abridged History
- Covered Languages

Classification

- Declarative
  - Functional: Lisp/Scheme, Ocaml, Haskell, Elixir, Erlang
  - Dataflow: Id, Val
  - Logic, Constraint: Prolog, Excel, SQL

- Imperative
  - von Neumann: C, Ada, Fortran
  - Object-oriented: Smalltalk, Eiffel, Java, C#, C++
  - Scripting: PERL, PHP, Python, Javascript

Functional Languages

- Functional languages use a computational model of **recursive definition of functions**.
- Inspired by *lambda calculus*.
- That is, functions act like mathematical functions.
  - They take in data and produce an output.
- Tends to lead to much simpler, but many more functions.

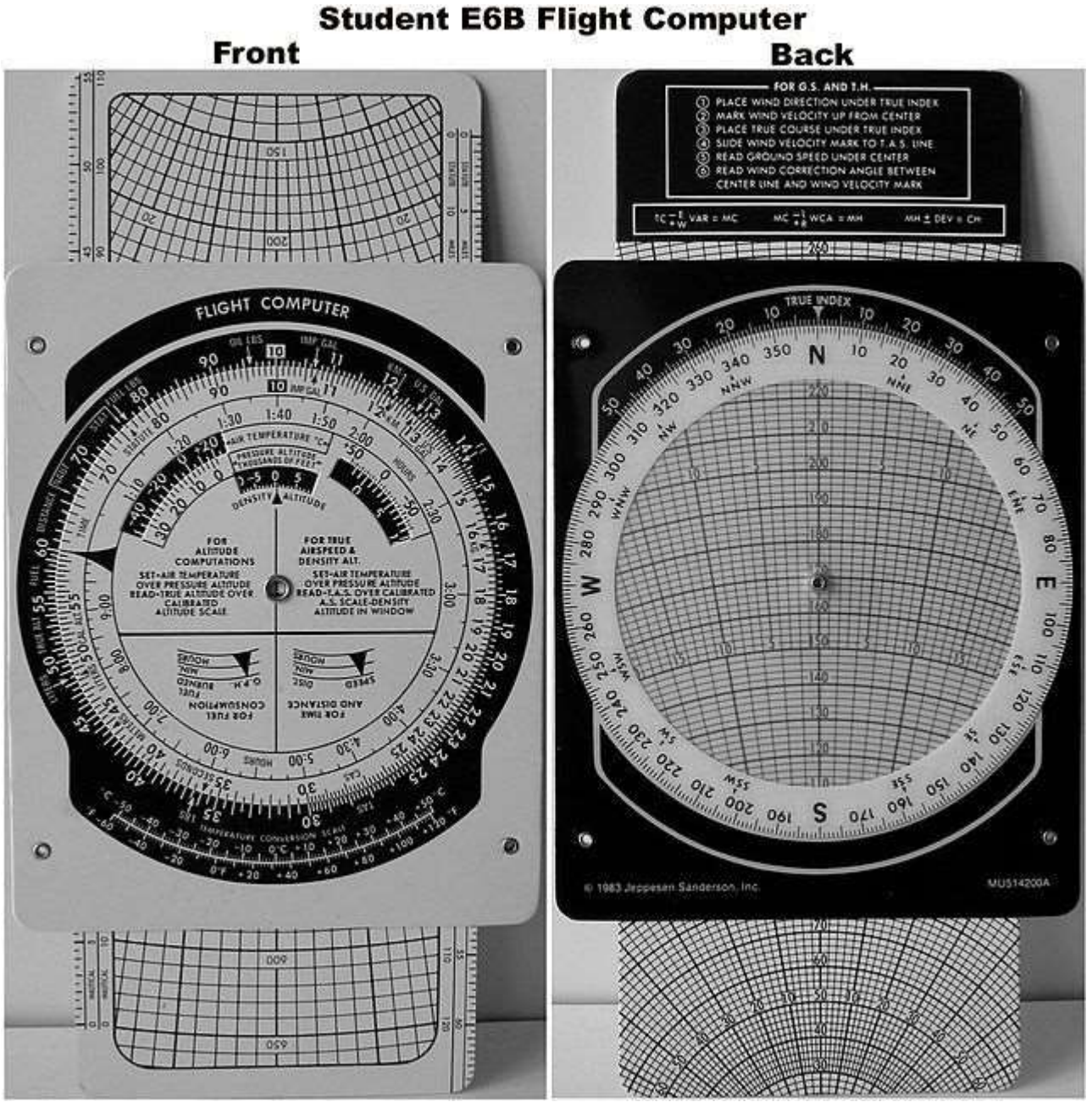
Imperative Languages

- Stored-program architecture.
- Changing a mutating state.
  - i = i + 1
- Imperative languages tell the machine how to run.
  - Declarative languages tell the machine what to run.

"Computing"

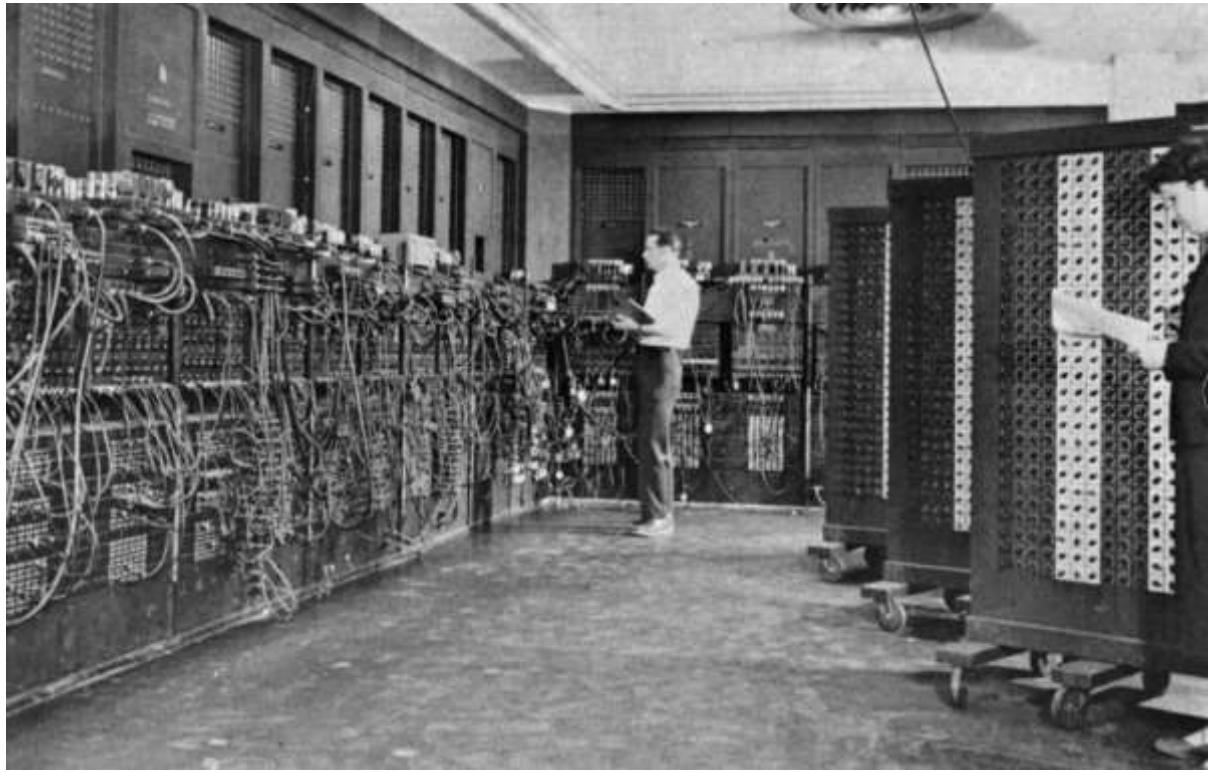
- A computer just solves a problem.

- Can be
  - Mechanical
  - Electrical
  - Physical
  - Abstract



Our Concept of Computing

- Started in the early 1940s
- Required "operators" to physically plug wires.
- Programmers had to know the architecture and machine.
- The "program" was all based on the machine.



Electronic Numerical Integrator and Computer (ENIAC), 1945

Stored Program Concept

- "von Neumann" Architectures
- Programs are "data" in memory that can change other data.
- Earliest programs were lookup tables.
  - If you want to add a and b: here's the binary.

Boolean Logic

- Vacuum tubes and transistors allowed "Boolean logic" to function.

- With these simple switches:
  - AND, OR, XOR, NOT

Assembly

- Assembly was the first abstraction of writing **machine code**.

- It was human-readable (arguably)
  - It was strictly imperative.
  - You told the computer how to calculate something.

"Coding"

- The concept of coding came from the "autocode" language in 1952 for the Mark I computer.
- This was the first abstraction from telling the machine exactly what to do.
- Required a compiler to translate the language into assembly/machine code.

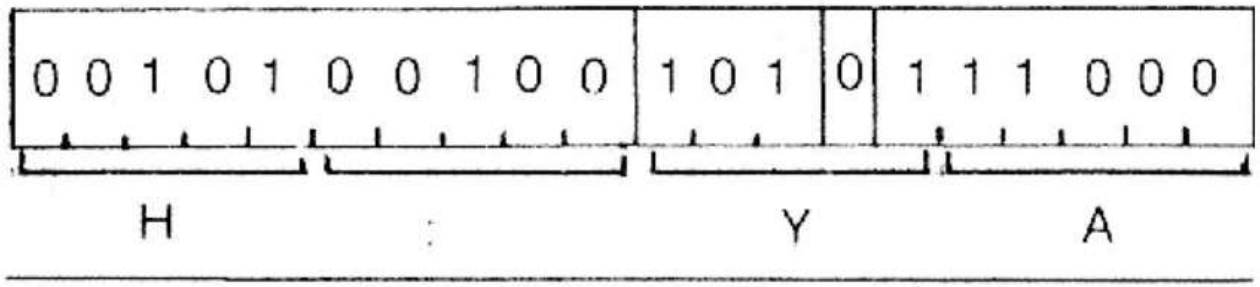


Figure 3. Specimen Mark I instruction.

Procedural Concept

- 1960s concept to program by call *subroutines* which were responsible for executing sections in memory.

- These subroutines could now be "called" and introduced "control transfer".

FORTRAN

- Made for IBM in 1957.
- Had high-level language concepts.
  - Functions (subroutines)
  - Pass-by-value/pass-by-reference
  - Complex number types
- High level language.
  - Spawned about 51 compilers in use by 1965

List Processing (LISP)

- Created in 1960 in MIT
- High-level, functional programming language
  - Still used today.
- Composed of "symbolic expressions" (S-expressions)

```
(define (factorial x)
    (if (= x 0)
        1
         (* x (factorial (- x 1)))))
```

BASIC

- Written to be simple.
- Home computers would boot to a BASIC program editor.

```
05 HOME : TEXT : REM Fibonacci
numbers
10 LET MAX = 5000
20 LET X = 1 : LET Y = 1
30 IF (X > MAX) GOTO 100
40 PRINT X
50 X = X + Y
60 IF (Y > MAX) GOTO 100
70 PRINT  Y
80 Y = X + Y
90 GOTO 30
100 END
```

Languages

- C#
- Haskell
- Rust
- Python – we will only use this for extending Python using C++ to look at foreign function interface (FFI).

C# (.cs) [created in 2000]

- Attributes
  - compiled
  - object-oriented
  - virtual machine (.NET)
  - procedural (imperative)

- Emphasizes rapid and large-scale development

- C# is fully portable now
  - MSFT supports Mac and Linux via "dotnet"
  - Third-party Mono

Haskell (.hs) [created in 1989]

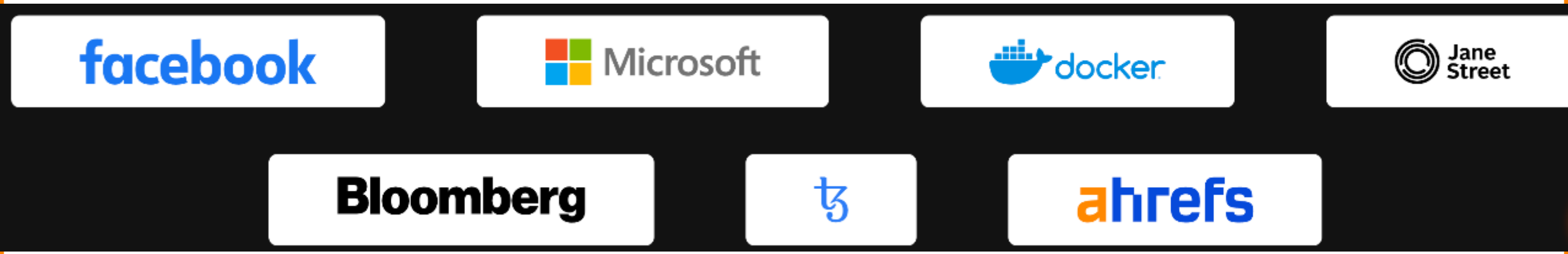
- Attributes
  - purely functional
  - compiled

- Emphasizes reliability and bug reduction at time of writing code

OCaml (.ml) [created in 1996]

- Attributes
  - interpreted (has a compiled-into-bytecode option)
  - declarative

- Emphasizes reliability



Elixir (.ex/.exs) [created in 2012]

- Attributes
  - compiled (.ex) or interpreted as a script (.exs)
  - declarative/functional

- Emphasizes distributed applications and fault-tolerance.

- Runs on the Erlang VM

Rust (.rs) [created in 2010]

- Attributes
  - compiled (has an interpreted option)
  - blended: imperative and/or functional
- Systems language
- Emphasizes safety
  - Ensures your memory is valid when it is accessed
- Has an **unsafe** mode, where YOU enforce safety/compliance.

Topics

- Classification
- Very Abridged History
- Covered Languages

Programming Languages

Stephen Marz

COSC365