CS 222: Programming Languages

Heather M. Guarnera

```
print(“Welcome!”)
printf(“Welcome!\n”);
System.out.println(“Welcome!”);
main = putStrLn “Welcome!”
```

4 (or 5) generations of programming languages

- **1GL**: machine code
- **2GL**: symbolic assemblers
- **3GL**: (machine-independent) imperative languages
  - Ex: FORTRAN, Pascal, C ...
- **4GL**: domain specific application generators
  - Scinapse a generator for mathematical modeling software
  - Mousetrap generates efficient real-time code for Motorola
  - R, SAS, SPSS, XSLT, Xquery ...
- **5GL**: AI languages ...

Each generation is at a higher level of abstraction

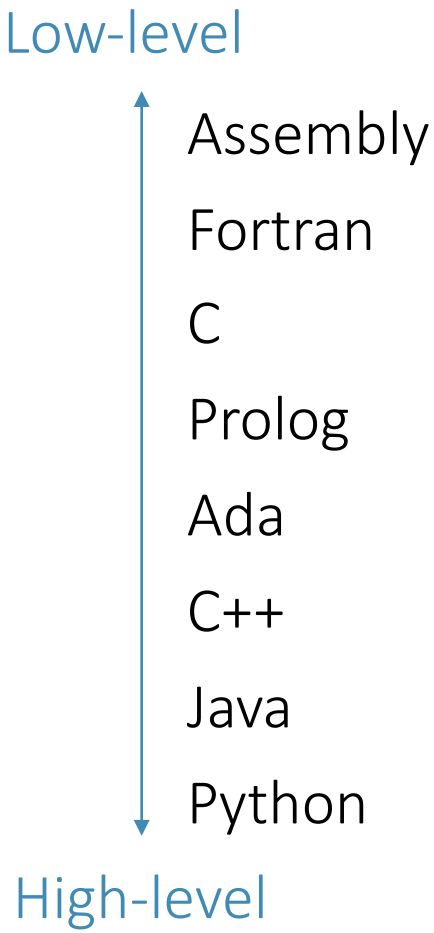
Performance vs. ease of writing

Low-level language:

- Native (or close to) to physical machine
- Efficient

High-level language:

- Higher abstraction
- Easier to read / write
- Tradeoff with efficiency



Low-level

Assembly

Fortran

C

Prolog

Ada

C++

Java

Python

High-level

Common ideas in modern imperative languages

- Extensive features
- Rich type system
- Mechanisms to support (in varying degree)
  - Procedural programming
  - Object-oriented programming
  - Concurrent programming
  - Generic programming
  - Abstractions
  - Information hiding

How do programming languages differ?

Common constructs

basic data types (numbers, etc.); variables; expressions; statements; keywords; control constructs; procedures; comments; errors ...

Uncommon constructs

type declarations; special types (strings, arrays, matrices, ...); concurrency constructs; packages/modules; objects; general functions; generics; ...

Programming paradigms

A programming language is a problem solving tool.

Imperative style	program = algorithms + data Good for decomposition
Functional style	program = functions ∘ functions Good for reasoning
Logic programming style	program = facts + rules Good for searching
Object-oriented style	program = objects + messages Good for modeling

Imperative Paradigm

- A program is: a sequence of *state*-changing actions

- Manipulate an abstract machine with
  - variables that name memory locations
  - arithmetic and logical operations
  - reference, evaluate, assign operations
  - explicit control flow statements
- Fits the Von Neumann architecture closely

- Key operations: assignment, if, while

Imperative Paradigm

Task: Sum up twice each number from 1 to N.

Fortran

```
     SUM = 0
     DO 11 K=1,N
         SUM = SUM + 2*K
11   CONTINUE
```

C

```
sum = 0;
for (k = 1; k <= n; ++k)
    sum += 2*k;
```

Pascal

```
sum := 0;
for k := 1 to n do
    sum := sum + 2*k;
```

Functional Paradigm

- A program is: a composition of functions on data

- Characteristics (in pure form):
  - Name values, not memory locations
    - Bind rather than assign
    - A variable is a table entry not a memory location
  - Value binding through parameter passing
  - Recursion rather than iteration

- Key operations: function application and function abstraction
  - Based on lambda calculus

Functional Paradigm

Scheme

```
(define (sum n)
   (if (= n 0)
      0
      (+ (* n 2) (sum (- n 1)))
   )
)

(sum 4)  evaluates to 20
```

Logic Paradigm

- A program is: a formal logical specification of a problem

- Characteristics (in pure form):
  - Programs say what properties the solution must have, not how to find it
  - Solutions are obtained through a specialized form of theorem-proving

- Key operations: unification and non-deterministic search
  - Based on first order predicate logic

Logic Paradigm

Prolog

```
sum(0,0).
sum(N,S) :- N>0,
            NN is N - 1,
            sum(NN, SS),
            S is N * 2 + SS.
```

```
?- sum(1,2).
yes
?- sum(2,4).
no
?- sum(4,S).
S = 20
?- sum(X,Y).
X = 0 = Y
```

Object-oriented Paradigm

- A program is: communication between abstract objects

- Characteristics:
  - Objects collect both the data and the operations
  - Objects provide data abstraction
  - Can be either imperative or functional (or logical)

- Key operations: message passing or method invocation

Object-oriented Paradigm

Java

```
public class IntSet {
  ...

  public Integer sum() {
    Integer s = 0;
    ListIterator<Integer> it = intVals.listIterator()
    while (it.hasNext()) {
      s = s + 2 * it.next();
    }
    return s;
  }
}
```

```
IntSet mySet = new IntSet(3);
mySet.sum();
```

Goals of this course

- Realize differences between programming languages
    - Paradigm
    - Purpose / support for problem solving
    - Features
- Understand how a translator reads a program, both theoretically and practically
- Develop marketable skills
    - Become a polyglot
    - Writing, research, collaboration, algorithm design & software development skills
    - Technical skills: Git, GitHub, Java, Haskell, (Prolog?)