Procedural Languages

(Most languages are *procedural*.)

- **Imperative**: Emphasis is **doing**.

- *Actions*

  $\boxed{\text{change}}$ value of variable
  $\boxed{\text{goto}}$ location
  $\boxed{\text{perform}}$ operation (*e.g.*, multiplication)
  $\boxed{\text{call}}$ a procedure
  $\boxed{\text{read}}$, $\boxed{\text{write}}$
  ...

  (*InActive:* declarations don't **do** anything).

Examples of Concurrency

- Multiplying two $n \times n$ matrices
  Need to compute n^2 independent values:

  ($R_{ij} = \sum_k M(i,k) \times N(k,j)$)

  Use n^2 processes, one for each R_{ij}!

  Each computes 1 value

  [*Eg* 5×5 matrices: 25 processors
  Each performs 5 multiplications, 4 additions]

- Automated Teller Machines:
  Each processes a different transaction

- Airline Reservation System
  Each processes a different transaction

Concurrency – General

- Many different processes

  All running "at same time"

  Each executing a different instruction

- Issues:

  Allocation of Utilities

  Deadlock, Starvation, . . .

Object Oriented Programming

- Emphasis is $\boxed{\textbf{objects}}$ (not processes).

Eg: Compute area of figure, X

where X might be Square, Circle, Polygon, ...

Standard Approach

– Write $\underline{1}$ AREA procedure,

which dispatches on X's type

Object Oriented Approach

– Write $\underline{n}$ AREA (sub)processes,

1 for each type of X

≈ 1 AREA for squares, 1 AREA for circles,...

– Pass X to AREA (collection of) processes

CLOS finds X's type – eg, `Circle`

Asks `Circle` for its AREA procedure

—– eg `circleAREA`

Applies `circle:`AREA to X

Comments on Object Oriented Approach

- Uses single name — AREA
  but invokes different procedures
  for different types of objects

- Each "subprocess" is short
  Specific to its type

$\approx$ "sending AREA msg to X"

- Advantages:
  Easier to code-up, remember
  ?More efficient?

- Issue: What if X in more than one class?
  - both Square and Polygon?
  - Square and Approximate-Figures?

Declarative Languages
(Logic Programming)

- Emphasis is $\boxed{\textbf{facts}}$.

  1. User first enters *facts*

     (which describe a situation).

  2. User then asks questions

     (about that situation).

  3. Language determines if query is a *fact*

     (given the *facts* already known).

- Language tries to *prove* query from facts.

  – Language returns appropriate values of variables in the query.

  – Language supplies the proof strategy.

Types of Facts
(as used in Logic Programs)

- Statements:

  E.g.,

  “`Math is Boring.`”

  “`Clyde is an elephant.`”

- Rules:

  E.g.,

If	x is an animal,	and
	x has big ears,	and
	x has a long trunk	
Then	x is an elephant.	

Functional Languages

- Emphasis is **values**

  Values derived by

  applying **functions** to objects.

- Execution $\equiv$ evaluating expressions

  (performing function call).

- Given a piece of program, question is not

  "*What does it do?*" but rather

  "*What does it evaluate to?*".

- *Pure* Functional Programming

  No side-effects, no "variables", recursion, . . .