CS 3210
Principles of PL

Lesson 01

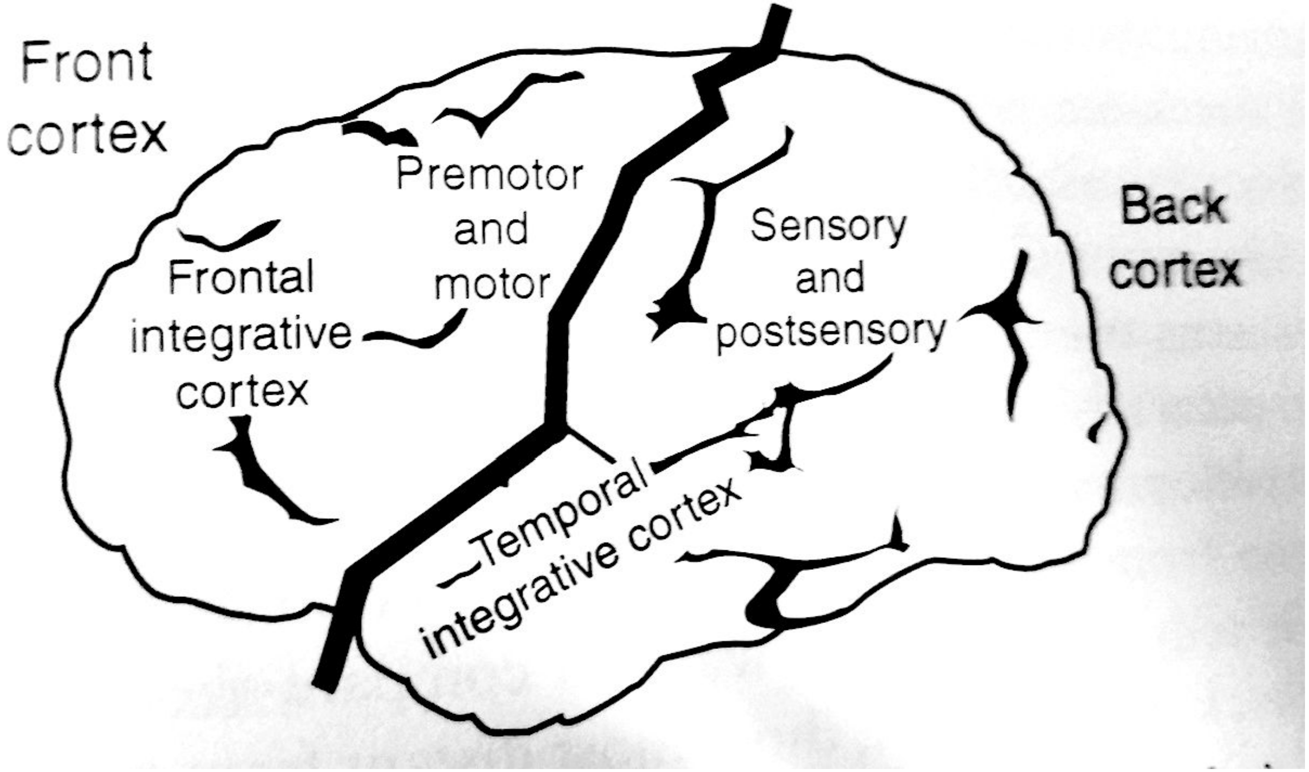
Agenda

- Introductions
- Teaching Philosophy
- Plan for CS 3210
- Let’s Get Started!

Teaching Philosophy

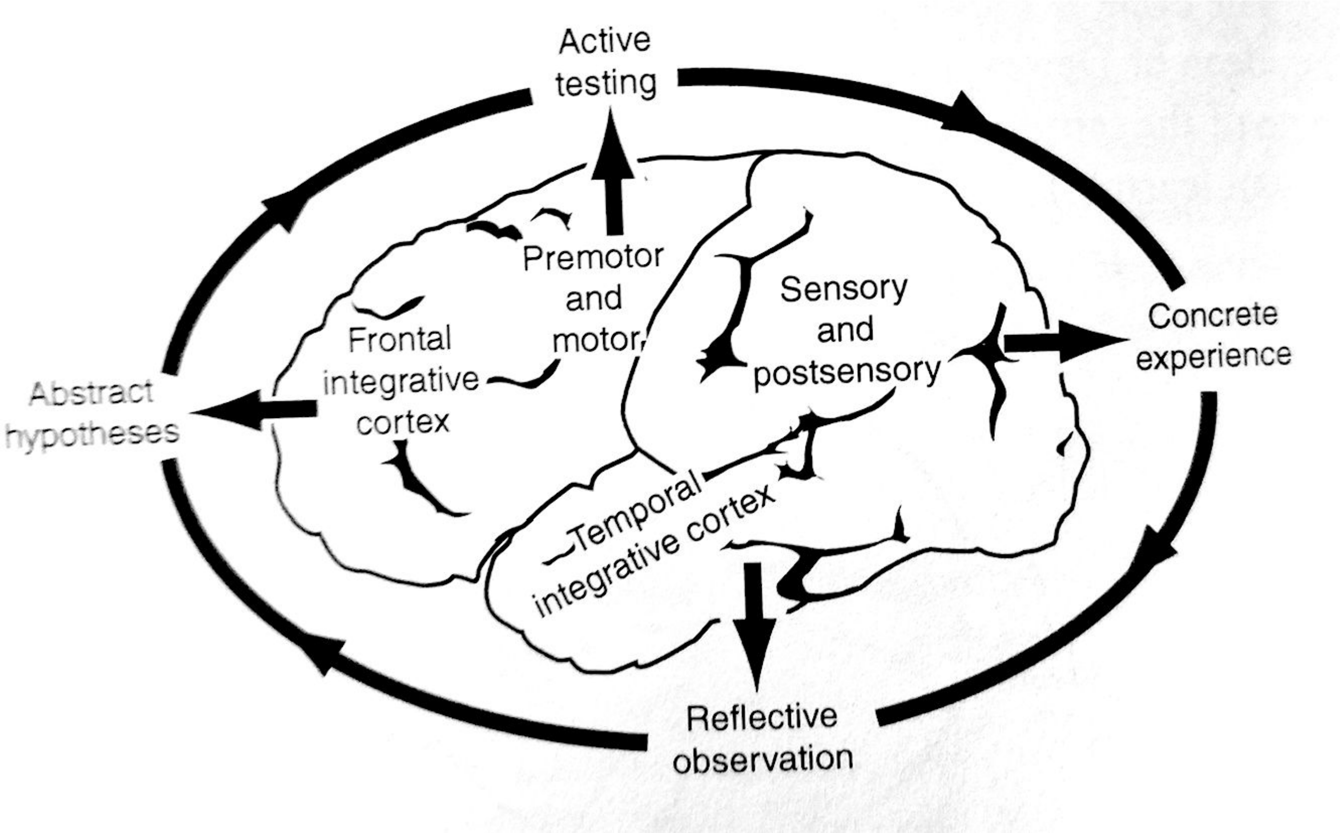
- Active Learning
- Build on Prior Knowledge
- Motivation is Key
- I Can’t Do It Alone

Teaching Philosophy



METROPOLITAN STATE UNIVERSITY OF DENVER
LIVES TRANSFORMED

Teaching Philosophy



Teaching Philosophy

- Kolb's Learning Cycle (example):

Expression: **"descascar o abacaxi"**

"des" is a negative prefix

"casca" means "rind"

"abacaxi" is the word for "pineapple"

Teaching Philosophy

- Kolb’s Learning Cycle (example):

Expression: **“descascar o abacaxi”**

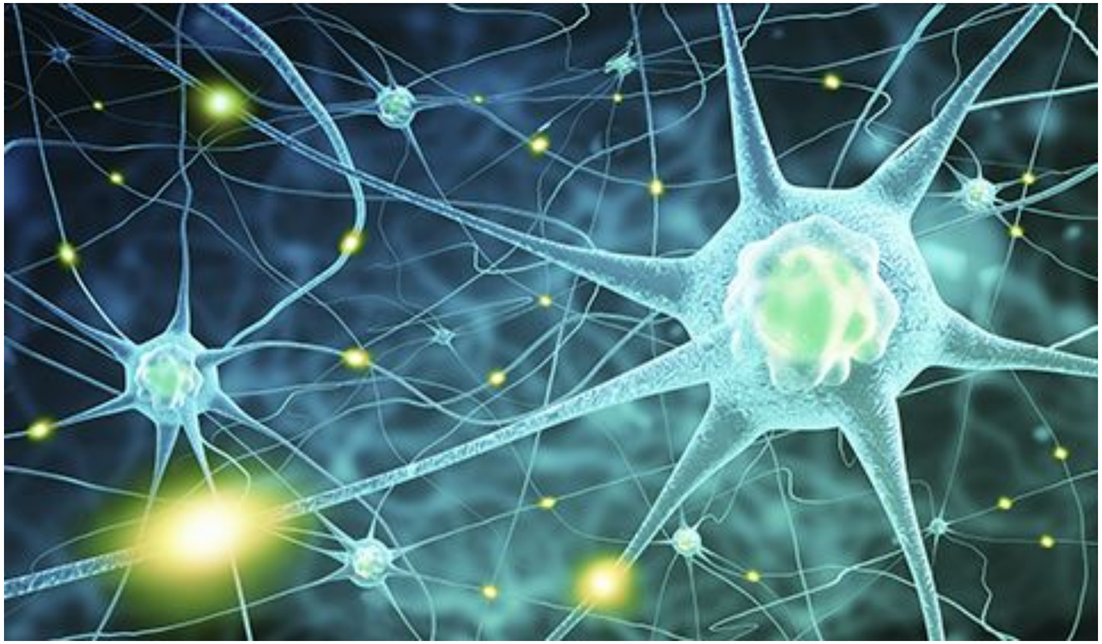
Translation: **“peel the pineapple”**

Example: “Count on me. I won’t let you **peel this pineapple** alone”

Metropolitan State University of Denver
LIVES TRANSFORMED

Teaching Philosophy

- Build on Prior Knowledge:



Teaching Philosophy

- Motivation is Key:
  - Sensory input signals compete for attention

Teaching Philosophy

- I Can’t Do It Alone:

Learning
is a two-way street
You get back *exactly*
What you put in.

Plan for CS 3210

- Course Material on Google sites (http://sites.google.com/view/thyagomota)
- Click on "Courses"
- Assignments submission and grades through Blackboard (https://metrostate-bb.blackboard.com)

Let’s Get Started

- Plickers Setup
- Why Study PL?
- Principles of PL Design
- PL Classification: Paradigms

Why Study PL?


Microsoft
Language
Pascal
ABAP
SYMPL
C/AL
Cg
Cesil
Scala
Rust
Bistro
GDL
JOSS
Genie
Inform
SenseTalk
Blue
HAL/S
SIMSCRIPT
Revolution
Sawzall
Chapel
Rapira
CSKA
SIMPLE
ALGOL
DATATRIEVE
PL/I
ELAN
C
Ajax
Agora
Clojure
Object
AutoIt
Microcode
code
Lynx
Not
Trac
Adenine
SAM76
Redcode
Logo
SALSA
ATS
Fjölnir
Fuxi
Assembly
TACPOL
FP
shell
E
language
A+
Lisp
VB
MicroScript
Squirrel
Windows/Dos
Kaleidoscope
IMP
C++
Fancy
SuperTalk
P
LC-3
JavaScript
K
Visual
FL
JavaFX
R
Forth
b
T
Newspeak
ABC
D
GameMonkey
Prolog
XOTcl
L
dc
Cayenne
Go
Delphi
DASL
Caml
Z
ECMAScript
GJ
bc
S
JASS
GM
Constraint
EusLisp
ObjectScript
Java
Oxygene
PEARL
G
Programming
BLISS
notation
Script
PHP
Ruby
EXEC
RPG
M
CorVision
Cyclone
Action
REXX
F
RPL
Little
NET
Q
Simula
Boomerang
COBOL
GraphTalk
A
ChucK
Common
Agena
Strand
Turing
ISWIM
PL/SQL
Arc
Moby
ORCA/Modula-2
Uniface
Python
J
Robot
rex
HaXe
S/SL
Eiffel
SP/k
Coral
WATFOR

Principles of PL Design

- Syntax
- Names
- Values
- Types
- Semantics

Principles of PL Design

- Syntax:
  - It describes what constitutes a structurally correct program

Principles of PL Design

- Names:
  - The label given to various kinds of entities commonly present in a PL, like variables, types, functions, parameters, classes, objects, etc.
  - PLs define a set of rules for naming entities in a program
  - Each entity will have different semantics associated with it

Principles of PL Design

- Types:
  - a collection of values and operations on those values
  - the *type system* of a language can help to determine legal operations and detect type errors

Principles of PL Design

- Semantics:
  - the meaning of a program
  - the exact effect of each statement when executed

```java
import java.util.Scanner;
public class Test {
    public static void main(String[] args) {
        int x, y;
        Scanner sc = new Scanner(System.in);
        System.out.print("x? ");
        x = sc.nextInt();
        System.out.print("y? ");
        y = sc.nextInt();
        if (x < y)
            System.out.println(x + " is less than " + y);
        else if (x > y)
            System.out.println(x + " is greater than " + y);
        else
            System.out.println(x + " is equal to " + y);
    }
}
```

Principles of PL Design

- PL Grammars precisely define what is a syntactically correct code
- Most PLs use a type of grammar called Context Free Grammar (CFG) also called type-2 grammars

Principles of PL Design

<program> → **begin** <stmt_list> **end**

<stmt_list> → <stmt>
| <stmt> ; <stmt_list>

<stmt> → <var> = <expression>

<var> → A | B | C

<expression> → <var> + <var>
| <var> – <var>
| <var>

Principles of PL Design

```
Statement:
    Block
    ;
    Identifier : Statement
    StatementExpression ;
    if ParExpression Statement [else Statement]
    ...
```

```
ParExpression:
    ( Expression )
```

PL Classification

- Paradigm Classification
- Abstraction Level Classification

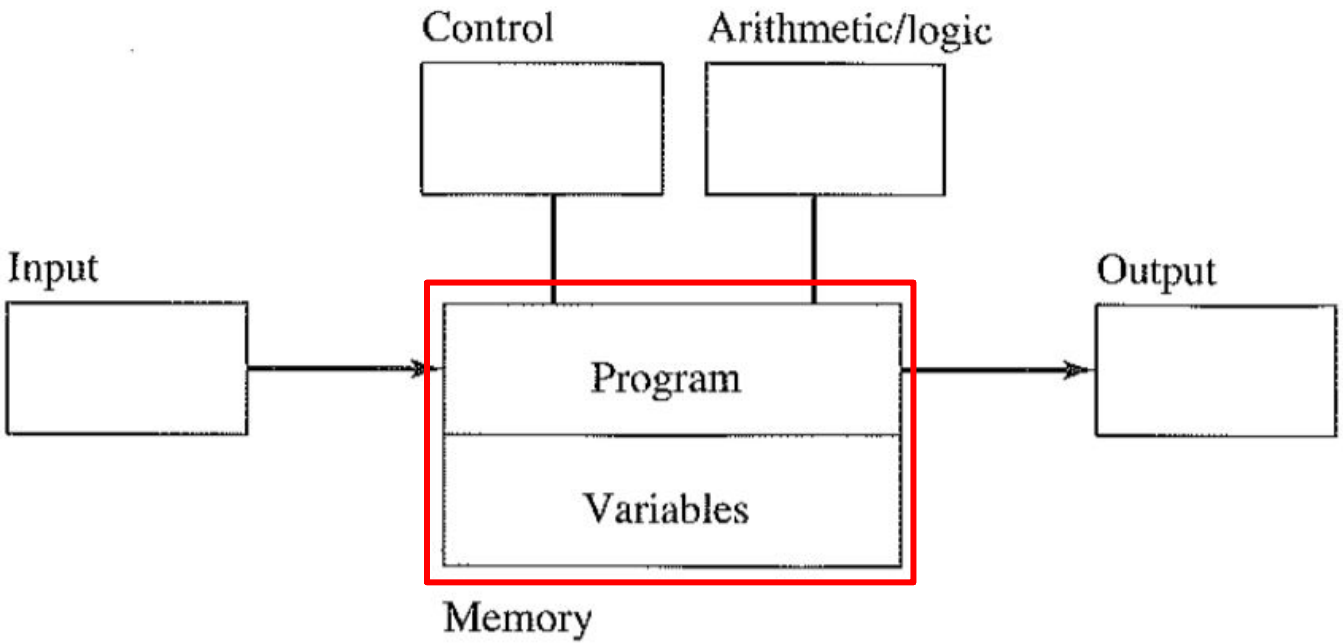
PL Paradigms

- A pattern of problem-solving thought that underlies a particular genre of programs and languages
- Didactically helpful as it puts PLs into categories
- Just be aware that many PLs may not fit into only one paradigm

PL Paradigms

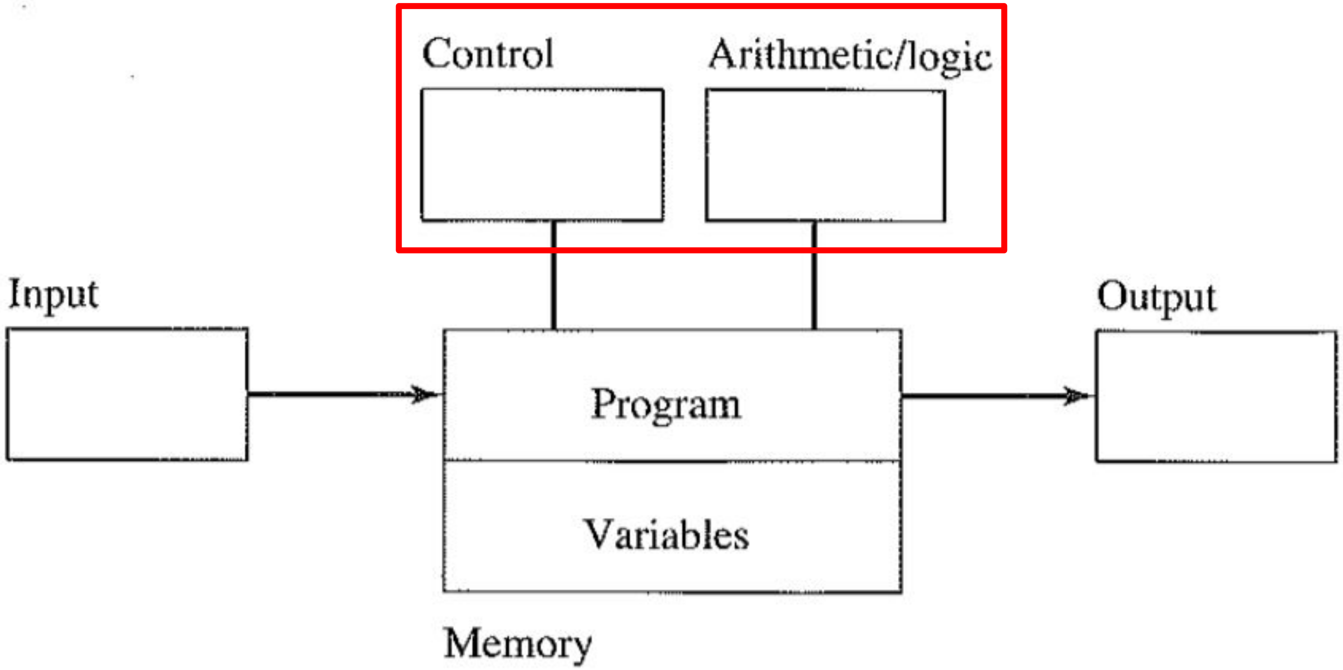
- Imperative
- Object-oriented
- Functional
- Logic

Imperative Paradigm



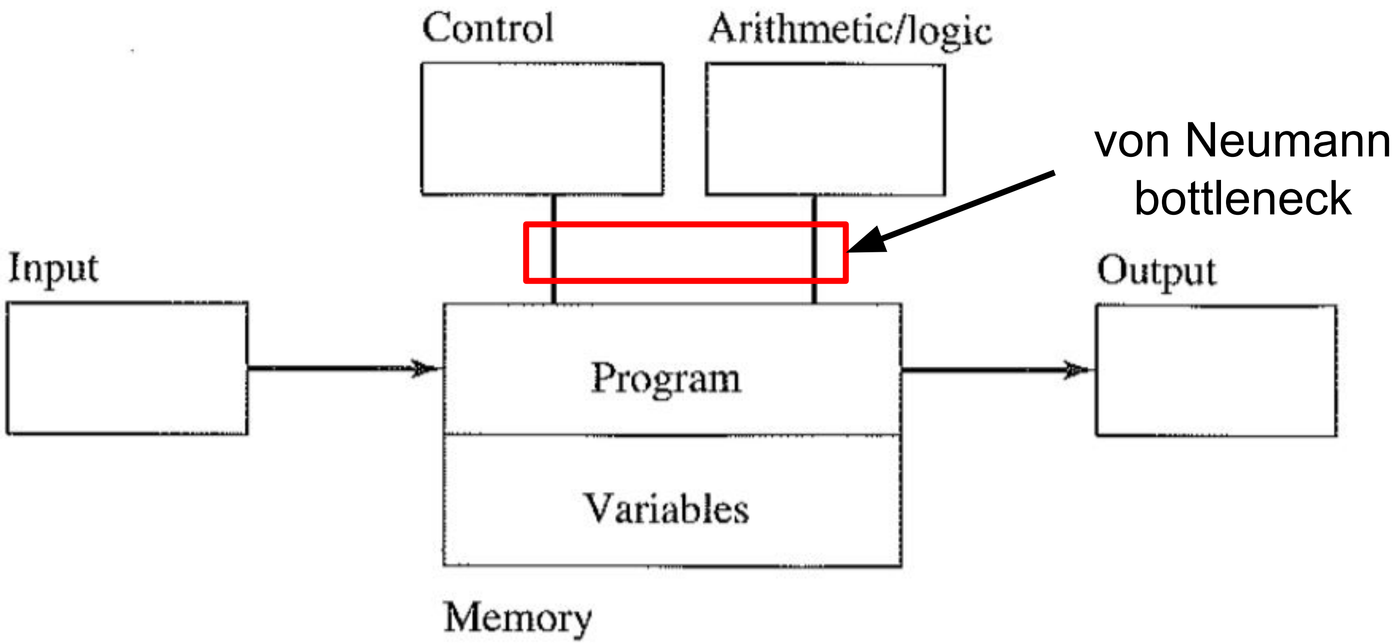
von Neumann Architecture

Imperative Paradigm



von Neumann Architecture

Imperative Paradigm



von Neumann Architecture

Imperative Paradigm

```
initialize the program counter
repeat forever
    fetch the instruction pointed to by the program counter
    increment the program counter to point at the next instruction
    decode the instruction
    execute the instruction
end repeat
```

Fetch-execute Cycle

Imperative Paradigm

- A program is a sequence of commands that are executed one after the other
- Variables maintain the state of the program's execution
- Program and data are indistinguishable in memory

Imperative Paradigm

- Typical Constructs: assignments, conditionals statements, loops and exception handling
- Large programs use procedural abstraction
- Examples: Fortran, Cobol, C, Basic, Pascal, Algol, Ada, etc.
- Still present in most PL and expect it to be around for decades to come

Object-oriented Paradigm

- A program is described as a collection of objects that interact by passing messages that transform their states
- It brought new concepts to PLs, such as message passing, inheritance, polymorphism, etc.
- Examples: Smalltalk, C++, C#, Java, Kotlin, Python, Ruby, etc.

Functional Paradigm

- Models computation as a collection of mathematical functions
- For example, to assign the result of the expression “a + b” to variable “c” you would do (in Lisp):

```
(setq c (+ a b))
```

Functional Paradigm

- Fundamentals features are:
  - functional composition
  - conditionals
  - recursion
  - stateless programming
- Examples: Lisp, Scheme, Haskell, Scala, etc.

Logic Paradigm

- Following the logic paradigm, a programmer **declares** what outcome the program should accomplish, rather than how it should be accomplished
- Programs are written as a series of constraints on a problem
- Example: Prolog

For Next Class

- Review the Course Syllabus
- Get the Textbook
- Read sections 1.1 and 1.3