Some Puzzles First

A man has two cats. At least one of them is male. What is the probability that both are male?

1 out of 3

A man has two cats - one black and one white. The white cat is male. What is the probability that both are male?

1 out of 2

A Controversial Puzzle

- I show you three boxes labeled A, B, and C, only one of which contains a prize (the others are empty).
- I know which box contains a prize, but you do not.
- You pick one of the boxes at random.
- Before you open it, I open up one of the other boxes that I know to be empty and show you that it is empty.
- I give you the choice of keeping the box you have or trading it for the third, unopened box.
- Is there any advantage in your trading?

Programming Languages

- What’s your favorite language?
- How many different languages are there?
- Why so many?
- Exactly what is a programming language?

As CSE Majors

You will learn/use

- C, C++, Java
- Database Languages (Oracle)
- Hardware Definition Languages (VHDL)
- Functional Programming Languages (SML)
- Logic Programming Languages (Prolog)
Study of Programming Languages

• Specification of Languages
• Design of Languages
• Implementation Issues of Languages
• Advanced Programming Language Features
• Different Programming Paradigms

Studying a New Language
Imagine a new Language with the following description:

• has a sound syntax
• comes with mark-and-sweep-garbage collection
• is interpreted, typeless, and highly reflective
• is pure object-oriented
• implements modules and mix-ins
• implements block closures
• implements operator & method overloading
• implements a sound exception handling
• comes bundled with a few "go4"-patterns
• has a powerful regular expression implementation

Paradigms of Programming
Different programs compute in different ways.

• Imperative (C, C++, FORTRAN, Java)
• Logic Programming (Prolog)
• Functional Programming (Standard ML, Lisp)
• Object-Oriented (Smalltalk, C++, Java)
• Concurrent (Java)
• Event-Driven (Java)

Levels of Programming Languages

• Machine Language
• Assembly Language
• Algorithmic Languages
• Fourth Generation Languages
• Fifth Generation Languages
Machine Language

- Native language of a computer: directly executed by hardware
- Everything is in binary form
- Examples: Intel Pentium, AMD Athlon, SUN Sparc, IBM PowerPC
- Currently most based on 32-bit words
- Basic ideas: arithmetic/logical operations on one or two operands (data) moving data sequence of such operations
- Sample:

  00111010 1100100 10110110 10001011
  11110101 00111001 11100101 11001101
...

Assembly Language

- Human readable version of machine language
- Uses mnemonic names for instructions, some data
- Converted to machine language by an Assembler
- Sample:

  .L2:     movl   -4(%ebp), %eax
          cmpl   -8(%ebp), %eax
          jl      .L4
          jmp     .L3
  .L4:    leal   -4(%ebp), %eax
          incl   (%eax)
          movl   -8(%ebp), %eax
          imull   -4(%ebp), %eax
          movl    %eax, -8(%ebp)
          jmp     .L2
  .L3:

Algorithmic Languages

- Languages like Fortran, C, C++, Pascal, Basic
- Hardware independent
- Basic statements: assignment, conditional, while loop
- Much easier to use than assembly languages
- Programs converted to assembly language by a Compiler
- Sample:

  while (a<b) {
    a++; 
    b = a*b; 
}
Fourth Generation Languages

- Based on mathematics (functions, logic)
- Independent of idea of machine computation
- No Assignment Statement!
- No While/Loop Statement!
- Functional programs consist of a set of function definitions
- Logic programs consist of a set of logical formula

Another Kind of Machine

What is a Virtual Machine?

- Defined as a moderately low-level machine language
- Not intended to be implemented directly in hardware
- Designed to be easily implemented (via simulation) on a wide variety of hardware (portability)
- Should not contain hardware dependent features
- Any language that can be compiled to a virtual machine language can be run on any machine that simulates this virtual machine:
  
  Write Once, Run Anywhere.

Java Virtual Machine

- The Java programming language was developed by Sun Microsystems specifically to run on a virtual machine.
- Java was designed to be translated to a virtual machine language, specifically the JVM.
- Any hardware machine with a JVM simulator can run Java programs:
  1. Java programs translated to JVM programs
  2. JVM programs executed by the JVM simulator
- A Simulator is just a program designed to simulate the execution of some machine

Learning a New Programming Language

How to learn a new programming language?

Need two things:

- what programs look like (Syntax)
- what programs do (Semantics)

We normally learn these by looking at examples

Designers/Implementers need formal techniques
A Simple Language

- Three constants: S, K, I.
- Programs are combinations of these constants
  - S, K, I are all programs
  - if \( P_1 \) and \( P_2 \) are programs then \((P_1 P_2)\) is a program
- Examples:
  \[
  S \quad (SK) \quad ((SK)K) \quad ((SK)K)K)
  
  ( (((SK)I)(KS)) ((IK)S))
  
Notation

- We abbreviate \((P_1P_2P_3)\) as \((P_1P_2P_3)\)
- Note that this is different from \((P_1(P_2P_3))\)
- We say that combination is left associative
- We see this in programming languages: \((x-y)-z)\) written as \((x-y-z)\) we can’t do this to \((x-(y-z))\)
- We leave out parentheses whenever no confusion occurs

Computation Rules

- Computation proceeds by rewriting a program (fragment) by a new program.
- Computation Rules tell us how to do this.
- The language has three computation rules
  - \(Ix \rightarrow x\)
  - \(Kxy \rightarrow x\)
  - \(Sxyz \rightarrow (xz)(yz)\)
  (where \(x, y, z\) are any programs)
- Fully parenthesized these are
  - \((Ix) \rightarrow x\)
  - \(((Kx)y) \rightarrow x\)
  - \(((Sx)y)z \rightarrow ((xz)(yz))\)

Computation

- Start with a program
- Find any subprogram of the form \(IP, KP_1P_2\) or \(SP_1P_2P_3\)
- Replace that subprogram according to rule: \(P, P_1\), or \((P_1P_3)(P_2P_3)\), respectively.
- Keep doing this until no more reduction steps possible
Examples

- $III$
- $(II)(II)$
- $KI(SK)K$
- $SKKI$
- $SKKSKKI$

The SKI Language

- This language was studied by Haskell Curry at Penn State (1930s)
- Provides a simple notion of computation
- Provides a powerful notion of computability
- Possible to represent integers, booleans, conditional, recursion

Adding New Definitions

- To make the language more expressive we can add new constants and rules (based on existing constants and rules)
- These provide a shortcut, but no additional power
- We can’t compute anything new, just more simply
- $Bxyz \rightarrow x(yz)$
- $Cxyz \rightarrow xzy$
- These can be defined in terms of $S$, $K$, and $I$:
  \[
  B = S(KS)K \\
  C = S(BBS)(KK)
  \]

Example: Conditional

How can we add a conditional to our language?

- Add constants $T$ and $F$ and $If$
- We would like:
  \[
  \text{If } T \ x \ y \rightarrow x \quad \text{If } F \ x \ y \rightarrow y
  \]
- Define:
  \[
  Txy = x \\
  Fxy = y \\
  \text{If} = I
  \]
- In fact we can define $T$ and $F$ as $K$ and $KI$ respectively.
Object-Oriented Programming

- Objects
- Classes
- Message Passing
- Inheritance

Object-Oriented Programming

- Program = Collection of Objects & Methods
- Send Messages to Objects
- Objects determine meaning of message
- Example to compute the factorial of \( n \) send the message fact to the object \( n \): \( n \).fact
- The object determines (via code) what to do with the message
- Sending the message fact to another object might result in something else happening

OO Languages

- Simula 67 - Introduced the concept of classes (encapsulating data & methods)
- Smalltalk - First pure OO language
- C++ - C extended with ideas from Simula (OO features)
- Java - mostly OO
- Ruby - recently defined, pure OO

Examples in Ruby

- **Objects**: 52, 3.14, true, "hello world", [1,2,3]
- **Messages**: 52.next, 3.14.round, true.to_s, "hello world".length, [1,2,3].length
- **Dynamic Typing**:
  
  ```ruby
  if b then x = "hello" else x = [1,2,3]
  y = x.length
  t = x.type
  ```

A Simple Ruby Class

```ruby
class Student
  def hello
    puts "Hi, I’m a student"
  end
end
```

Instances (new objects) of this class are created using the new method:

```ruby
s = Student.new
```

What is s.type?
What is Student.type?
Student Class Again

class Student
  def initialize(sname, sid, sgrade)
    @name = sname
    @id = sid
    @grade = sgrade
  end

  def hello
    puts "Hi, I'm a student. My name is #{@name}."  
  end
end