Programming Languages

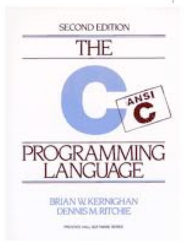
The Tower of Babel

A story about the origins of multiple languages.

- [After the flood] "the whole earth was of one language and one speech."
- They built a city and tower at Babel, believing that with a single language, people will be able to do anything they imagine.
- Yahweh disagrees and "confounds the language of all the earth"
- Why? Proliferation of cultural differences (and multiple languages) is one basis of civilization.

Several ways to solve a transportation problem

Several ways to solve a programming problem



Java

You can write a Java program.

ThreeSum.java

```
public class ThreeSum
{
   public static void main(String[] args)
   {
      int N = Integer.parseInt(args[0]);
      int[] a = new int[N];
      for (int i = 0; i < N; i++)
         a[i] = StdIn.readInt();
      for (int i = 0; i < N; i++)
         for (int j = i+1; j < N; j++)
            for (int k = j+1; k < N; k++)
               if (a[i] + a[j] + a[k] == 0)
                  StdOut.println(a[i] + " " + a[j] + " " + a[k]);
   }
}
```

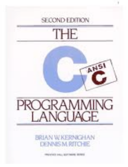
```
% more 8ints.txt
30 -30 -20 -10 40 0 10 5

% javac ThreeSum.java
% java ThreeSum 8 < 8ints.txt
 30 -30   0
 30 -20 -10
-30 -10  40
-10   0  10
```

Ex. Read N int values from standard input;
print triples that sum to 0.
[See lecture 8]

C

You can also write a C program.

ThreeSum.c



```
#include <stdio.h>
#include <stdlib.h>
main(int argc, char *argv[])
{
  int N = atoi(argv[1]);
  int *a = malloc(N*sizeof(int));
  int i, j, k;
  for (i = 0; i < N; i++)
    scanf("%d", &a[i]);
  for (i = 0; i < N; i++)
    for (j = i+1; j < N; j++)
      for (k = j+1; k < N; k++)
         if (a[i] + a[j] + a[k] == 0)
           printf("%4d %4d %4d\n", a[i], a[j], a[k]);
}
```

```
% more 8ints.txt
30 -30 -20 -10 40 0 10 5

% cc ThreeSum.c
% ./a.out 8 < 8ints.txt
 30 -30   0
 30 -20 -10
-30 -10  40
-10   0  10
```

Noticable differences:
library conventions
array creation idiom
standard input idiom
pointer manipulation (stay tuned)

A big difference between C and Java (there are many!)

No data abstraction
- no objects in C
- C program is sequence of static methods

C++ (Stroustrup 1989)
- "C with classes"
- adds data abstraction to C

C++

You can also write a C++ program.

Ex 1. Use C++ like C

ThreeSum.cxx

```
#include <iostream.h>
#include <stdlib.h>
main(int argc, char *argv[])
{
  int N = atoi(argv[1]);
  int *a = new int[N];
  int i, j, k;
  for (i = 0; i < N; i++)
    cin >> a[i];
  for (i = 0; i < N; i++)
    for (j = i+1; j < N; j++)
      for (k = j+1; k < N; k++)
        if (a[i] + a[j] + a[k] == 0)
          cout << a[i] << " " << a[j] << " " << a[k] << endl;
}
```

```
% cpp ThreeSum.cxx
% ./a.out 8 < 8ints.txt
 30 -30    0
 30 -20  -10
-30 -10   40
-10   0   10
```

Noticable differences:
- library conventions
- standard I/O idioms

C++

Ex 2. Use C++ like Java

BST.cxx

```
template <class Item, class Key>
class ST
{
   private:
     struct node
     { Item item; node *l, *r; int N;
       node(Item x)
         { item = x; l = 0; r = 0; N = 1; }
     };
   typedef node *link;
   link head;
   Item nullItem;

   Item searchR(link h, Key v)
   { if (h == 0) return nullItem;
     Key t = h->item.key();
     if (v == t) return h->item;
     if (v < t) return searchR(h->l, v);
           else return searchR(h->r, v);
   }
...
}
```

Challenges:
- libraries/idioms
- pointer manipulation
- templates (generics)

A big difference between C/C++ and Java (there are many!)

Programs directly manipulate pointers

C/C++: You are responsible for memory allocation
- system provides memory allocation library
- programs explicitly "allocate" and "free" memory for objects
- C/C++ programmers must learn to avoid "memory leaks"

Java: Automatic "garbage collection".

C code that reuses an array name

```
double arr[] =
calloc(5,sizeof(double));
...
free(arr);
arr = calloc(10, sizeof(double));
```

Java code that reuses an array name

```
double[] arr = new double[5];
...
arr = new double[10];
```

Fundamental challenge: Code that manipulates pointers is inherently "unsafe".

Python

You can write a Python program!

Ex 1. Use python as a calculator

```
% python
Python 2.7.1 (r271:86832, Jun 16 2011, 16:59:05)
Type "help" for more information.
>>> 2+2
4
>>> (1 + sqrt(5))/2
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'sqrt' is not defined
>>> import math
>>> (1 + math.sqrt(5))/2
1.618033988749895
```

Python

You can write a Python program!

Ex 2. Use Python like you use Java

threesum.py

```
import sys
def main():
  N = int(sys.argv[1])
  a = [0]*N
  for i in range(N):
    a[i] = int(sys.stdin.readline())
  for i in range(N):
    for j in range(i+1, N):
      for k in range(j+1, N):
        if (a[i] + a[j] + a[k]) == 0:
          print repr(a[i]).rjust(4),
          print repr(a[j]).rjust(4),
          print repr(a[k]).rjust(4)
main()
```

Noticable differences:
- no braces (indents instead)
- no type declarations
- array creation idiom
- I/O idioms
- for (iterable) idiom

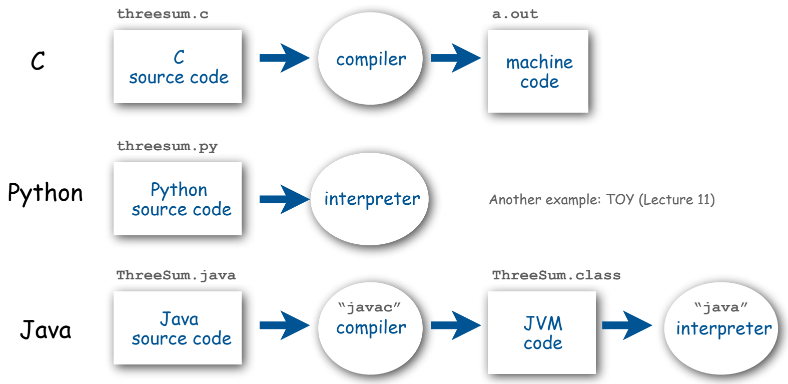
```
range(8) is [0, 1, 2, 3, 4, 5, 6, 7]
```

```
% python threesum.py 8 < 8ints.txt
 30 -30    0
 30 -20  -10
-30 -10   40
-10   0   10
```

Compile vs. Interpret

Def. A compiler translates your entire program to (virtual) machine code.

Def. An interpreter simulates a (virtual) machine running your code on a line-by-line basis.



Another example: TOY (Lecture 11)

A big difference between Python and C/C++/Java (there are many!)

No compile-time type checking
- no need to declare types of variables
- system checks for type errors at RUN time

Python programmers must remember their variable names and types

Implications:
- Easier to write small programs.
- More difficult to debug large programs.

Typical (nightmare) scenario:
- Scientist/programmer misspells variable name referring to output file in large program.
- Program runs for hours or days.
- Crashes without writing results.

Using Python for large problems is like playing with fire.

Reasonable approaches
- Throw out your calculator; use Python.
- Prototype in Python, then convert to Java for "production" use.

Matlab

You can write a Matlab program!

Ex 1. Use Matlab like you use Java.

```
...
for i = 0:N-1
  for j = i+1:N-1
    for k = j+1:N-1
      if (a(i) + a(j) + a(k)) == 0:
        sprintf("%4d %4d %4d\n", a(i), a(j), a(k));
      end
    end
  end
end
...
```

Ex 2 (more typical). Use Matlab for matrix processing.

```
A = [1 3 5; 2 4 7]
B = [-5 8; 3 9; 4 0]
C = A*B
C =
    24    35
    30    52
```

$$\begin{pmatrix} 1 & 3 & 5 \\ 2 & 4 & 7 \end{pmatrix} * \begin{pmatrix} -5 & 8 \\ 3 & 9 \\ 4 & 0 \end{pmatrix} = \begin{pmatrix} 24 & 35 \\ 30 & 52 \end{pmatrix}$$

Big differences between Matlab and C/C++/Java/Python (there are many!)

1. Matlab is not free.
2. Most Matlab programmers use only ONE data type (matrix).

FREE

Ex. Matlab code "`i = 0`" ← "redefine the value of the complex number i to be a 1-by-1 matrix whose entry is 0"

Notes:
- Matlab is written in Java.
- The Java compiler and interpreters are written in C.
  [Modern C compilers are written in C.]
- Matrix libraries (written in C) accessible from C/C++/Java/Python.

Reasonable approach:
- Use Matlab as a "matrix calculator" (if you own it).
- Convert to or use Java or Python if you want to do anything else.

Why Java? [revisited from second lecture]

Java features.
- Widely used.
- Widely available.
- Embraces full set of modern abstractions.
- Variety of automatic checks for mistakes in programs.

Facts of life.
- No language is perfect.
- We need to choose some language.

"There are only two kinds of programming languages: those people always [gripe] about and those nobody uses."
– Bjarne Stroustrup

Our approach.
- Minimal subset of Java.
- Develop general programming skills that are applicable to many languages

It's not about the language!

Why do we use Java in COS 126?

	widely used	widely available	full set of modern abstractions	modern libraries and systems	automatic checks for bugs
C ANSI C	✓	✓	✗	✗	✓
C++	✓	✓	✓	maybe	✓
Java	✓	✓	✓	✓	✓
MATLAB	✓	$	maybe*	✓	✗
python	✓	✓	maybe	✓	✗

* OOP recently added but not embraced by most users

Why learn another programming language?

Good reasons to learn a programming language:
- offers something new (see next slide)
- need to interface with legacy code or with coworkers
- better than Java for the application at hand
- intellectual challenge (opportunity to learn something about computation)

Something new: a few examples

1960s: Assembly language
- symbolic names
- relocatable code

1970s: C
- "high-level" language (statements, conditionals, loops)
- machine-independent code
- basic libraries

1990s: C++/Java
- data abstraction (object-oriented programming)
- extensive libraries

2000s: Javascript/PHP/Ruby/Flash
- scripting
- libraries for web development

Programming styles

Procedural
- step-by-step instruction execution model
- Ex: C

Scripted
- step-by-step command execution model, usually interpreted
- Ex: Python, Javascript

Special purpose
- optimized around certain data types
- Ex: Postscript, Matlab

Object-oriented (next)
- focus on objects that do things
- Ex: Java, C++

Functional (stay tuned)
- focus on defining functions
- Ex: Scheme, Haskell, Ocaml

Object-Oriented Programming

Object Oriented Programming

Procedural programming. [verb-oriented]
- Tell the computer to do this.
- Tell the computer to do that.

A different philosophy. Software is a simulation of the real world.
- We know (approximately) how the real world works.
- Design software to model the real world.

Objected oriented programming (OOP). [noun-oriented]
- Programming paradigm based on data types.
- Identify things that are part of the problem domain or solution.
- Things in the world know something: instance variables.
- Things in the world do something: methods.

Why Object-Oriented Programming?

Essential questions.
- Is my program easy to write?
- Is it easy to find errors and maintain my program?
- Is it correct and efficient?

OOP admits:
- Encapsulation: hide information to make programs robust.
- Type checking: avoid and find errors in programs.
- Libraries: reuse code.
- Immutability: guarantee stability of program data.

Warning.
- OOP involves deep, difficult, and controversial issues.
- Lots of hard questions; few easy answers.

Does OOP make it easy to write and maintain correct and efficient programs?

[Religious wars ongoing.]

OOP pioneers

Kristen Nygaard and O.J. Dahl. [U. Oslo 1960s]
- Invented OOP for simulation.
- Developed Simula programming language.
- Studied formal methods for reasoning about OO programs.

Kristen Nygaard and O.J. Dahl
2001 Turing Award

Alan Kay. [Xerox PARC 1970s]
- Developed Smalltalk programming language.
- Promoted OOP for widespread use.
- Conceived Dynabook portable computer.
- Computer science visionary.

Alan Kay
2003 Turing Award

Alan Kay's Vision (1970s)

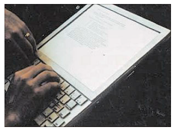
Typical "mainframe" computer

IBM 360/50

First personal computer

Xerox Alto

Alan Kay's vision

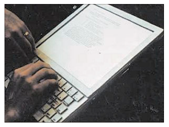


Dynabook

Dynabook features
integrated OOP software
written in Smalltalk
not real! (simulated on Alto)

Alan Kay's Vision

Alan Kay's vision in 1971



Typical computer in 2011

Visionary quotes (still relevant in 2011!)

"The best way to predict the future is to invent it. (1971)"

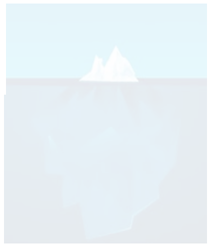
"The computer revolution hasn't happened yet. (1997)"

— Alan Kay

Why Object-Oriented Programming?

Essential questions.
- Is my program easy to write?
- Is it correct?
- Is it efficient?

OOP admits:
- Encapsulation: hide information to make programs robust.
- Type checking: avoid and find errors in programs.
- Libraries: reuse code.
- Immutability: guarantee stability of program data.



Warning.
- OOP involves deep, difficult, and controversial issues.
- Lots of hard questions; few easy answers.
- Interested? Take COS 441.

Ex. Does OOP make it easy to write correct and efficient programs?
[Religious wars ongoing.]

Encapsulation

Data type. Set of values and operations on those values.
Ex. `int`, `String`, `Complex`, `Vector`, `Document`, `GuitarString`, `Tour`, …

Encapsulated (abstract) data type.
- Hide internal representation of values.
- Expose operations to client (only via API).

Separates implementation from design specification.
- Class provides data representation and code for operations.
- Client uses data type as black box.
- API specifies contract between client and class.

Bond. What's your escape route?
Saunders. Sorry old man. Section 26 paragraph 5, that information is on a need-to-know basis only. I'm sure you'll understand.

Bottom line.
You don't need to know how a data type is implemented in order to use it

Intuition behind encapsulation

Client

API
- volume
- change channel
- adjust picture
- decode NTSC signal

Implementation
- cathode ray tube
- electron gun
- Sony Wega 36XBR250
- 241 pounds

client needs to know how to use API

implementation needs to know what API to implement

Implementation and client need to agree on API ahead of time.

Intuition behind encapsulation

Client

API
- volume
- change channel
- adjust picture
- decode NTSC signal

Implementation
- gas plasma monitor
- Samsung FPT-6374
- wall mountable
- 4 inches deep

client needs to know how to use API

implementation needs to know what API to implement

Can substitute better implementation without changing the client.

Changing Internal Representation

Encapsulation.
- Keep data representation hidden with `private` access modifier.
- Expose API to client code using `public` access modifier.

```
public class Complex
{
   private final double re, im;

   public Complex(double re, double im)  { ... }
   public double abs()                   { ... }
   public Complex plus(Complex b)        { ... }
   public Complex times(Complex b)       { ... }
   public String toString()              { ... }
}
```

e.g., to polar coordinates

Advantage. Can switch internal representation without changing client.

Note. All our data types are already encapsulated!

Example: Counter Data Type

Counter. Data type to count electronic votes.

```
public class Counter
{
   public int count;
   public final String name;

   public Counter(String id)  { name = id;     }
   public void increment()    { count++;       }
   public int value()         { return count;  }
}
```

Malevolent but legal Java client.

```
Counter c = new Counter("Volusia County");
c.count = -16022;
```

we don't write client code like this, but many programmers do.

Oops. Al Gore receives -16,022 votes in Volusia County, Florida.

OOPs ? (joke)

Example: Counter Data Type

Better Counter. Encapsulated data type to count electronic votes.

```
public class Counter
{
   private int count;
   private final String name;

   public Counter(String id)  { name = id;     }
   public void increment()    { count++;       }
   public int value()         { return count;  }
}
```

Malevolent code does not compile.

```
Counter c = new Counter("Volusia County");
c.count = -16022;
```

Benefit. Can guarantee that each data type value behaves as designed.

Time Bombs that might have been avoided with encapsulation

Internal representation "conventions" adopted by clients.
- [Y2K] Two digit years: January 1, 2000.
- [Y2038] 32-bit seconds since 1970: January 19, 2038.
- [VIN numbers] 2004 prediction: We'll run out by 2010.
- [IP addresses] 32-bit convention lasted only a few decades.

Fundamental problem. Need to examine all client code to change "convention".
Solution. Encapsulate!

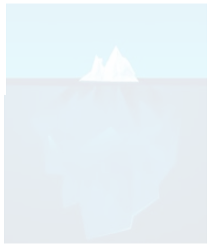
```
public class Date
{
   private ... // Internal representation

   public Date(int d, int m, int y)  { ... }
   public int day()                  { ... }
   public int month()                { ... }
   public int year()                 { ... }
}
```

Can change convention without changing any client code.

Why Object-Oriented Programming?

Essential questions.
- Is my program easy to write?
- Is it correct?
- Is it efficient?

OOP admits:
- Encapsulation: hide information to make programs robust.
- Type checking: avoid and find errors in programs.
- Libraries: reuse code.
- Immutability: guarantee stability of program data.



Warning.
- OOP involves deep, difficult, and controversial issues.
- Lots of hard questions; few easy answers.
- Interested? Take COS 441.

Ex. Does OOP make it easy to write correct and efficient programs?
[Religious wars ongoing.]

Type Checking

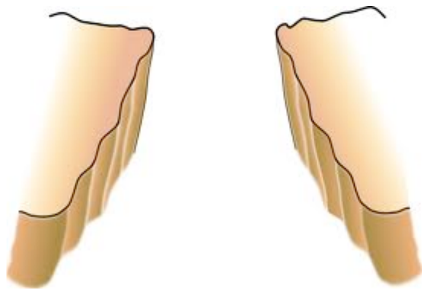
Static (compile-time) type checking (e.g. Java)
- All variables have declared types.
- System checks for type errors at compile time.

Dynamic (run-time) type checking (e.g. Python)
- Values, not variables, have defined types.
- System checks for type errors at run time.

Which is best? Religious wars ongoing!
- Static typing worth the trouble?
- Compiled code more efficient?
- Type-checked code more reliable?
- Advanced features (e.g. generics) too difficult to use with static typing?

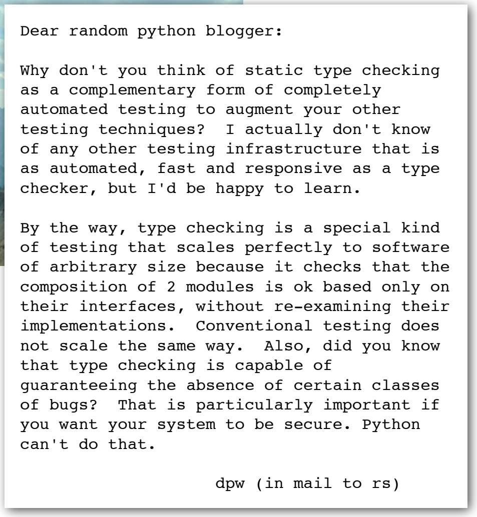
Vastly different points of view

" Program testing can be a very effective way to show the presence of bugs, but it is hopelessly inadequate for showing their absence.
— Edsgar Dijkstra (1969)



" Since static type checking can't cover all possibilities, you will need automated testing. Once you have automated testing, static type checking is redundant.
— Python blogger (2009)

A letter from Prof Walker



```
Dear random python blogger:

Why don't you think of static type checking
as a complementary form of completely
automated testing to augment your other
testing techniques?  I actually don't know
of any other testing infrastructure that is
as automated, fast and responsive as a type
checker, but I'd be happy to learn.

By the way, type checking is a special kind
of testing that scales perfectly to software
of arbitrary size because it checks that the
composition of 2 modules is ok based only on
their interfaces, without re-examining their
implementations.  Conventional testing does
not scale the same way.  Also, did you know
that type checking is capable of
guaranteeing the absence of certain classes
of bugs?  That is particularly important if
you want your system to be secure. Python
can't do that.

                    dpw (in mail to rs)
```

Programming Folklore: Hungarian type system

Early programming languages had no types

Hungarian type system (Charles Simonyi, 1970s)
- encode type in first few characters of variable name
- 8 character limit? Leave out the vowels, truncate.

introduced OOP to Microsoft

Ex. `arru8Fbn`

array of unsigned 8-bit integers

variable name short for `Fibonacci`

An advantage: Can "type check" while reading code.
A disadvantage: shrt vwl-lss vrbl nms.

Used in first version of Microsoft Word (and extensively before that time).

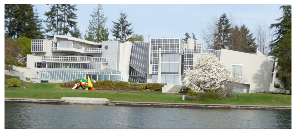
Lesson: Type-checking has always been important in large software systems.

Charles Simonyi: A Legendary Programmer

developed Bravo at PARC (1970s)

then MS Word (1983)

Simonyi Hall at IAS


Windows 2000 mansion in Seattle

owns 230' luxury yacht

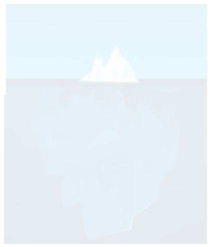
Martha Stewart's boyfriend (1993-2008)

space tourist (2007 and 2009)

Why Object-Oriented Programming?

Essential questions.
- Is my program easy to write?
- Is it correct?
- Is it efficient?

OOP admits:
- Encapsulation: hide information to make programs robust.
- Type checking: avoid and find errors in programs.
- Libraries: reuse code.
- Immutability: guarantee stability of program data.



Warning.
- OOP involves deep, difficult, and controversial issues.
- Lots of hard questions; few easy answers.
- Interested? Take COS 441.

Ex. Does OOP make it easy to write correct and efficient programs?
[Religious wars ongoing.]

Inheritance

Interface inheritance: Define an interface that formalizes the API
- enables modular programming encapsulation with libraries
- maybe not worth the trouble for small programs
- indispensable tool for avoiding errors in large programs
- required for some useful Java conventions
  [Ex: **`Comparable`** and **`Iterable`**]

```
public interface Comparable<Item>
{
  public int compareTo(Item that)
}
```

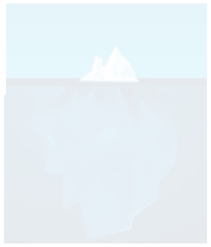
Subtyping: Define a new type that extends another one
- widely used
- controversial
- required for some useful Java conventions
  [all classes extend **`Object`** to implement **`toString()`** and **`equals()`**]
- some libraries are built to extend through subtyping
- otherwise, probably safe to ignore

For a few more details, see text (pp. 434-439).

Why Object-Oriented Programming?

Essential questions.
- Is my program easy to write?
- Is it correct?
- Is it efficient?

OOP admits:
- Encapsulation: hide information to make programs robust.
- Type checking: avoid and find errors in programs.
- Libraries: reuse code.
- Immutability: guarantee stability of program data.



Warning.
- OOP involves deep, difficult, and controversial issues.
- Lots of hard questions; few easy answers.
- Interested? Take COS 441.

Ex. Does OOP make it easy to write correct and efficient programs?
[Religious wars ongoing.]

Immutability

Immutable data type. Object's value does not change once constructed.

mutable	*immutable*
`Picture`	`Charge`
`Histogram`	`Color`
`Turtle`	`Stopwatch`
`StockAccount`	`Complex`
`Counter`	`String`
Java arrays	wrapper types

might also view primitive types as immutable [we don't write "`3 = 4`"]

Final Access Modifier

Final. Declaring an instance variable to be **`final`** means that you can assign it a value only once, in initializer or constructor.

```
public class Counter
{
   private final String name;
   private int count;

   public Counter(String id) { name = id;     }
   public void increment()   { count++;       }
   public int value()        { return count;  }
}
```

this value doesn't change once the object is constructed

this value changes when instance method `increment()` is invoked

Advantages.
- Helps enforce immutability.
- Prevents accidental changes.
- Makes program easier to debug.
- Documents that the value cannot change.

Immutability: Advantages and Disadvantages

Immutable data type. Object's value cannot change once constructed.

Advantages.
- Avoid aliasing bugs (see text p. 353)
- Makes program easier to debug.
- Limits scope of code that can change values.
- Pass objects around without worrying about modification.

aliasing bug, since **`Picture`** is not immutable

```
Picture a = new Picture("mandrill.jpg");
Picture b = a;
a.set(i, j, color1); // a is updated
b.set(i, j, color2); // a is updated again
```

Note: Most COS126 students encounter aliasing bugs in LFSR.

Disadvantage. New object must be created for every value.

Better design for electronic voting. Immutable **`VoteCount`**.
[to avoid malevolent code that takes advantage of aliasing]

TEQ on Data Type Design 1

[easy if you read pages 430-433]

Q. Is the following data type immutable?

```
public class Vector
{
   private final double[] coords;

   public Vector(double[] a)
   { // Make a defensive copy to ensure immutability.
     coords = a;
   }

   public Vector plus(Vector b)    { ... }
   public Vector times(Vector b)   { ... }
   public double dot(Vector b)     { ... }
}
```

TEQ on Data Type Design 2

[easy if you read pages 430-433]

Q. Is the following data type immutable?

```
public class Vector
{
   private final double[] coords;

   public Vector(double[] a)
   { // Make a defensive copy to ensure immutability.
     coords = new double[a.length];
     for (int i = 0; i < a.length; i++)
     coords[i] = a[i];
   }

   public Vector plus(Vector b)    { ... }
   public Vector times(Vector b)   { ... }
   public double dot(Vector b)     { ... }
}
```

Parting Food for Thought

Programming styles

Procedural
- step-by-step instruction execution model
- Ex: C

Scripted
- step-by-step command execution model, usually interpreted
- Ex: Python, Javascript

Special purpose
- optimized around certiain data types
- Ex: Postscript, Matlab

Object-oriented
- focus on objects that do things
- Ex: Java, C++

Functional
- focus on defining functions
- Ex: Scheme, Haskell, Ocaml

Functional programming

Q. Why can't we use functions as arguments in Java programs?
A. Good question. We can, but doing so requires interfaces and is cumbersome.

Functional programming is a function-oriented programming style.

- Functions are first-class entities
  [can be arguments and return values of other functions or stored as data].
- Immutable data structures by default.
- On-demand execution model.
- "What" rather than "how".
- Ex. Recursive code

Functional programming (Python)

```
def sq(x):
  return x*x
def table(f, R):
  for x   in R:
    print x,
    print f(x)
...
print table (sq, range(10))
```

```
0 0
1 1
2 4
3 9
4 16
5 25
6 36
7 49
```

Advantages of functional programming

- Often leads to much more compact code than alternatives.
- More easily admits type system that can result in "provably correct" code.
- More easily supports concurrency (programming on multiple processors).

Disadvantage: May need cumbersome mutable data structures for performance.

Functions that operate on functions

Functions as first-class objects admit compact code for powerful operations.

```
def odd(x):
  return 2*x + 1
def plus(x, y):
  return x + y
```

two Python functions

Ex 1: MAP(f, L): "replace each value x in L with f(x)"

```
range(8) is [0, 1, 2, 3, 4, 5, 6, 7]
map(odd, range(8)) is [1, 3, 5, 7, 9, 11, 13, 15]
map(sq, range(8)) is [0, 1, 4, 9, 16, 25, 36, 49]
```

Ex 2: REDUCE(f, L) = f($\overset{\text{first item on L}}{\text{car}}$(L), REDUCE(f, $\overset{\text{all but first item on L}}{\text{cdr}}$(L)))

```
reduce(plus, map(odd, range(8))) is 1 + 3 + 5 + 7 + 9 + 11 + 13 + 15 = 64
```

```
reduce(plus, [1, 3, 5, 7, 9, 11, 13, 15])
  = 1 + reduce(plus, [3, 5, 7, 9, 11, 13, 15])
  = 1 + 3 + reduce(plus, [5, 7, 9, 11, 13, 15])
  = 1 + 3 + 5 + reduce(plus, [7, 9, 11, 13, 15])
  = 1 + 3 + 5 + 7 + reduce(plus, [9, 11, 13, 15])
  = 1 + 3 + 5 + 7 + 9 + reduce(plus, [11, 13, 15])
  = 1 + 3 + 5 + 7 + 9 + 11 + reduce(plus, [13, 15])
  = 1 + 3 + 5 + 7 + 9 + 11 + 13 + reduce(plus, [15])
  = 1 + 3 + 5 + 7 + 9 + 11 + 13 + 15
```

Why learn functional programming?

- offers something new
- need to interface with legacy code
- offers specialized tools
- intellectual challenge

Intro CS at MIT is taught in Scheme (a functional language)

Modern applications

- communications systems
- financial systems
- Google map/reduce

!!

Deep and direct connections to theoretical CS (stay tuned)

Interested? Take COS441.

The Tower of Babel

A story about the origins of multiple languages.

- [After the flood] "the whole earth was of one language and one speech."
- They built a city and tower at Babel, believing that with a single language, people will be able to do anything they imagine.
- Yahweh disagrees and "confounds the language of all the earth"
- Why? Proliferation of cultural differences (and multiple languages) is the cradle of civilization.

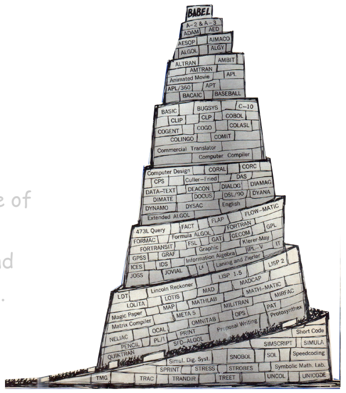


image from cover of Sammet "Programming Languages" (1969)
already 120+ languages!

An apt metaphor.

- Would a single programming language enable us to do anything that we imagine?
- Is the proliferation of languages a basis of civilization in programming?