INTRODUCTION TO
Programming
in Java

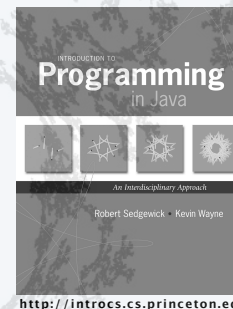*An Interdisciplinary Approach*

Robert Sedgewick · Kevin Wayne

http://introcs.cs.princeton.edu

16. 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.

INTRODUCTION TO
Programming
in Java

An Interdisciplinary Approach

Robert Sedgewick · Kevin Wayne

http://introcs.cs.princeton.edu

16. Programming Languages

- **Popular languages**
- Java in context
- Object-oriented programming
- Type checking
- Functional programming

Several ways to solve a transportation problem

Several ways to solve a programming problem

Java

You can write Java code.

3-sum
- Read int values from StdIn.
- Print triples that sum to 0.
- [See *Performance* lecture]

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
```

C

You can *also* write C code.

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

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
```

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

NO DATA ABSTRACTION
- No objects in C.
- A C program is a sequence of static methods.

C++ (Stroustrup 1989)
- Adds data abstraction to C.
- "C with classes".
- Embodies many OOP innovations.

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

– Bjarne Stroustrup

C++

You can *also* write C++ code.

Example 1. Use C++ like C.

Noticable differences
- library conventions
- standard input idiom
- standard *output* idiom
- pointer manipulation (stay tuned)

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
```

C++

You can *also* write C++ code.

Example 2. Use C++ like Java to implement the symbol table ADT.

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

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);
    }
...
}
```

Algorithms IN C++
Parts 1-4
ROBERT SEDGEWICK
1990
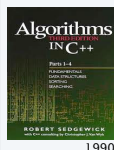
A big difference between C/C++ and Java (there are many!)

C/C++: YOU are responsible for memory allocation
- Programs manipulate pointers.
- System provides memory allocation library.
- Programs explicitly call methods that "allocate" and "free" memory for objects.
- Pitfall: "memory leaks".

Java: Automatic "garbage collection"
- System keeps track of references.
- System manages memory use.
- System reclaims memory that is no longer accessible from your program.

Fundamental challenge. C/C++ code that manipulates pointers is inherently *unsafe*.

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];
```

Python

You can *also* use Python.

Example 1. Use Python like 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 *also* write Python code.

Example 2. Use Python like Java.

Noticable differences
- No braces (indents instead).
- No type declarations.
- Array creation idiom.
- I/O idioms.
- for (iterable) idiom.

threesum.py
```
import sys
def main():
    N = int(sys.argv[1])
    a = [0]*N                          range(8) is [0,1,2,3,4,5,6,7]
    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()
```

"printable representation"

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

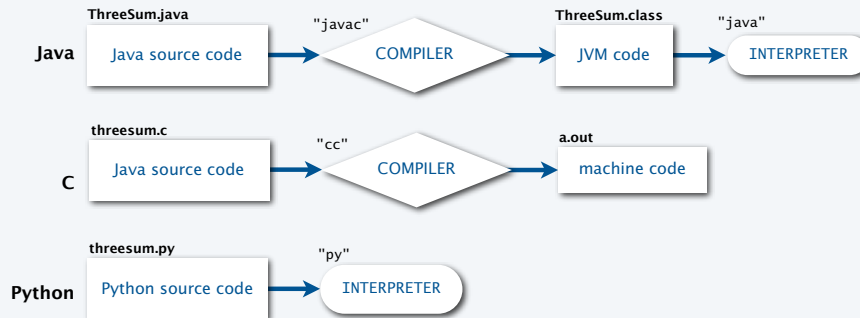## Compilation vs. Interpretation

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

Definition. An interpreter simulates the operation of a (virtual) machine running your code.

Java

ThreeSum.java
Java source code → "javac" COMPILER → **ThreeSum.class** JVM code → "java" INTERPRETER

C

threesum.c
Java source code → "cc" COMPILER → **a.out** machine code

Python

threesum.py
Python source code → "py" INTERPRETER

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.

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

Using Python for large problems is playing with fire.

Typical (nightmare) scenario
- Scientist/programmer makes a small mistake in a big program.
- Program runs for hours or days.
- Program crashes without writing results.

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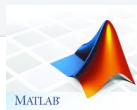
Matlab

You can write Matlab code.

Example 1. Use Matlab like 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
...
```

Example 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
```

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

17

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

1. MATLAB IS NOT FREE.
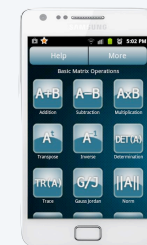2. Most Matlab programmers use only ONE data type (matrix).

Example. Matlab code "i = 0" means
   "*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.]
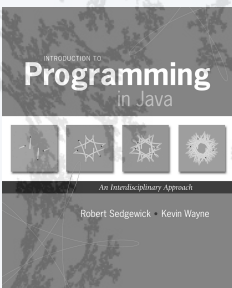• Matrix libraries (written in C) are accessible from C/C++/Java/Python.

Reasonable approaches
• Use Matlab as a "matrix calculator" (if you own it).
• Convert to or use Java or Python if you want to do anything else.
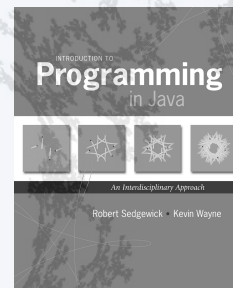
Matrix calculator
Android app

18

INTRODUCTION TO
Programming
in Java

An Interdisciplinary Approach

Robert Sedgewick · Kevin Wayne

http://introcs.cs.princeton.edu

16. Programming Languages

• **Popular languages**
• Java in context
• Object-oriented programming
• Type checking
• Functional programming

INTRODUCTION TO
Programming
in Java

An Interdisciplinary Approach

Robert Sedgewick · Kevin Wayne

http://introcs.cs.princeton.edu

16. Programming Languages

• Popular languages
• **Java in context**
• Object-oriented programming
• Type checking
• Functional programming

Why Java? [revisited from second lecture]

Java features
- Widely used.
- Widely available.
- Continuously under development since early 1990s.
- Embraces full set of modern abstractions.
- Variety of automatic checks for mistakes in programs.

James Gosling
http://java.net/jag

Java economy
← $100 billion, 5 million developers
- Mars rover.
- Cell phones.
- Blu-ray Disc.
- Web servers.
- Medical devices.
- Supercomputing.
- ...

Why do we use Java in this course?

language	widely used	widely available	full set of modern abstractions	modern libraries and systems	automatic checks for bugs
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.
- Need to interface with co-workers.
- Better than Java for the application at hand.
- Provides an intellectual challenge
- Opportunity to learn something about computation.
- Introduces a new programming style.

Something new: a few examples

1960s: Assembly language
- symbolic names
- relocatable code

1970s: C
- "high-level" language
- statements, conditionals, loops
- machine-independent code
- functions and libraries

1990s: C++/Java
- data abstraction (OOP)
- extensive libraries

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

Programming styles

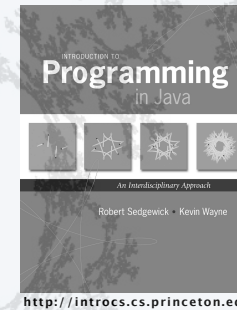| style	execution model	examples
procedural	step-by-step instruction execution usually compiled	
scripted	step-by-step command execution usually interpreted	
special-purpose	optimized around certain data types	
object-oriented	focus on objects that do things	
functional	focus on *what* to do as opposed to *how* to do it	

INTRODUCTION TO
Programming
in Java

An Interdisciplinary Approach

Robert Sedgewick · Kevin Wayne

http://introcs.cs.princeton.edu

16. Programming Languages

- Popular languages
- **Java in context**
- Object-oriented programming
- Type checking
- Functional programming

INTRODUCTION TO
Programming
in Java

An Interdisciplinary Approach

Robert Sedgewick · Kevin Wayne

http://introcs.cs.princeton.edu

16. Programming Languages

- Popular languages
- Java in context
- **Object-oriented programming**
- Type checking
- Functional programming

Object-oriented programming

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

Procedural programming
- Tell the computer to do this.   ⟵ VERB-oriented
- Tell the computer to do that.

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

Why OOP?

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

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

Warning: OOP involves deep, difficult, and controversial issues. Further study may be fruitful, but is likely to raise more questions than answers!

Does OOP make it easy to write and maintain correct and efficient programs?
- Difficult for you to know, because you haven't programmed in another style.
- Ongoing debate among experts intensifies as time goes on.
- Meanwhile, millions of people (including YOU) are reaping the benefits of OOP.

OOP pioneers

Kristen Nygaard and O.J. Dahl. (U. Oslo 1960s)
- Invented OOP for simulation.
- Developed Simula programming language.
- Studied formal 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.
- Computer science visionary.

Alan Kay
2003 Turing Award

Barbara Liskov. (MIT 1970s)
- Developed CLU programming language.
- Pioneered focus on data abstraction.
- Research provided basis for Java, C++, ...

Barbara Liskov
2008 Turing Award

Alan Kay: a computer science visionary

1970s

Typical "mainframe" computer: IBM 360/50

1970s

First PC: Xerox Alto

1970s
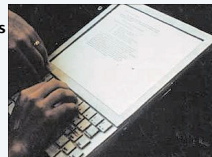
Alan Kay's vision for the future
Dynabook prototype
Key feature: **OOP software** (Smalltalk)

"The best way to predict the future is to invent it." (1971)
"The computer revolution hasn't happened yet." (1997)

– Alan Kay

Still relevant today!
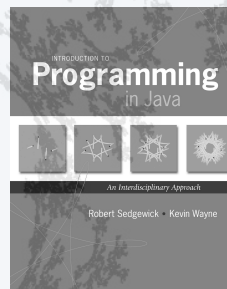
2010s

Modern personal computer
MacBook Air
Key feature: **OOP software** (Objective C)

COMPUTER SCIENCE
SEDGEWICK / WAYNE

INTRODUCTION TO
Programming
in Java

An Interdisciplinary Approach

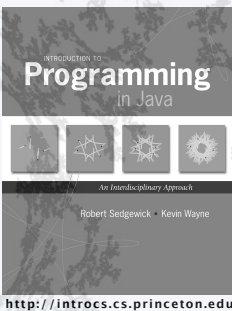Robert Sedgewick · Kevin Wayne

http://introcs.cs.princeton.edu

16. Programming Languages

- Popular languages
- Java in context
- **Object-oriented programming**
- Type checking
- Functional programming

16. Programming Languages

- Popular languages
- Java in context
- Object-oriented programming
- **Type checking**
- Functional programming

INTRODUCTION TO
Programming
in Java

An Interdisciplinary Approach

Robert Sedgewick · Kevin Wayne

http://introcs.cs.princeton.edu

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.

Q. Which is best?

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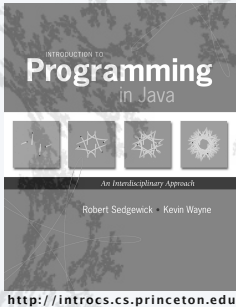- Compiled code more efficient?
- Type-checked code more reliable?
- Advanced features (e.g. generics) too difficult to use with static typing?

34

Example: Diametrically opposed points of view

Issue. Type checking or automated program testing?

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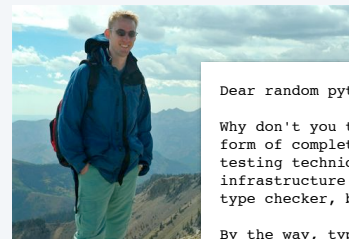
— *Edsgar Dijkstra (1969)*

" *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)*

35

A letter from Dave 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)
```

36

Programming folklore: Hungarian type system

Early programming languages had little support for types.

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

Example. `arru8Fbn`

array of 8-bit
integers (unsigned)

variable name
short for Fibonacci

Charles Simonyi
Introduced OOP to Microsoft

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
Xerox PARC (1970s)

Oversaw development of
MS Word/Excel (1983)

Simonyi Hall at the Institute for Advanced Study

Space tourist
(2007 and 2009)

Owns 230' luxury yacht Skat

Dated Martha Stewart (1993-2008)

Windows 2000 mansion in Seattle
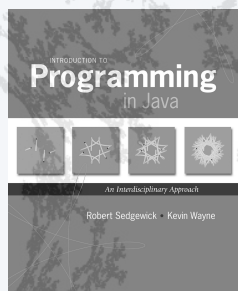
INTRODUCTION TO
Programming
in Java

An Interdisciplinary Approach

Robert Sedgewick · Kevin Wayne

http://introcs.cs.princeton.edu

16. Programming Languages

• Popular languages
• Java in context
• Object-oriented programming
• **Type checking**
• Functional programming

INTRODUCTION TO
Programming
in Java

An Interdisciplinary Approach

Robert Sedgewick · Kevin Wayne
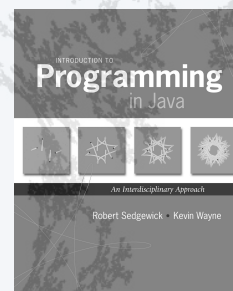
http://introcs.cs.princeton.edu

16. Programming Languages

• Popular languages
• Java in context
• Object-oriented programming
• Type checking
• **Functional programming**

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).
- On-demand execution model.
- "What" rather than "how".

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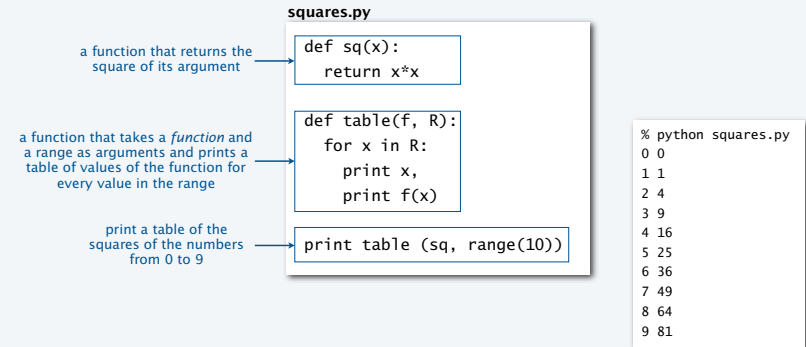- More easily supports concurrency (programming on multiple processors).

Disadvantage. Can be more difficult to focus on performance.

HASKELL
Scala
OCaml
python
ERLANG

Functional programming example

A Python program that prints a tables of squares.

squares.py

a function that returns the square of its argument
```
def sq(x):
    return x*x
```

a function that takes a *function* and a range as arguments and prints a table of values of the function for every value in the range
```
def table(f, R):
    for x in R:
        print x,
        print f(x)
```

print a table of the squares of the numbers from 0 to 9
```
print table (sq, range(10))
```

```
% python squares.py
0 0
1 1
2 4
3 9
4 16
5 25
6 36
7 49
8 64
9 81
```

Functions that operate on functions

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

Example 1. The MAP operation takes a function and a list as arguments.
MAP(f, L) is the result of replacing every x in L by f(x).

map.py

a function that returns the square of its argument
```
def sq(x):
    return x*x

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

print the squares of the numbers from 0 to 9
```
print map (odd, range(10))
print map (sq, range(10))
```

```
% python map.py
[1, 3, 5, 7, 9, 11, 13, 15, 17, 19]
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```
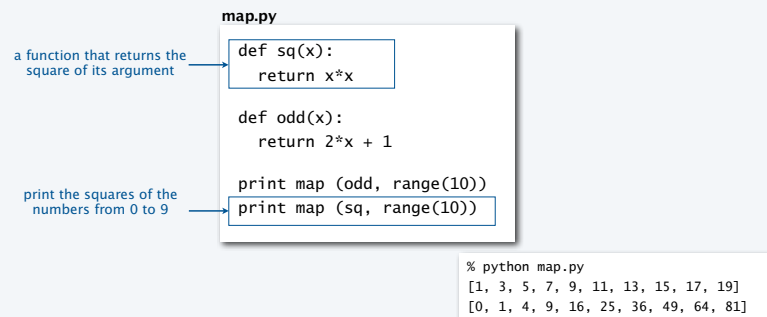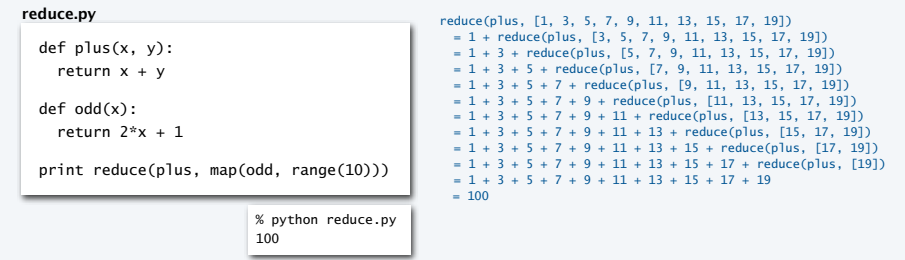

Functions that operate on functions

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

Example 2. The REDUCE operation takes a function and a list as arguments.
REDUCE(f, L) is f(car(L), REDUCE(f, cdr(L))).

first entry on L   all but first entry on L

reduce.py

```
def plus(x, y):
    return x + y

def odd(x):
    return 2*x + 1

print reduce(plus, map(odd, range(10)))
```

```
reduce(plus, [1, 3, 5, 7, 9, 11, 13, 15, 17, 19])
 = 1 + reduce(plus, [3, 5, 7, 9, 11, 13, 15, 17, 19])
 = 1 + 3 + reduce(plus, [5, 7, 9, 11, 13, 15, 17, 19])
 = 1 + 3 + 5 + reduce(plus, [7, 9, 11, 13, 15, 17, 19])
 = 1 + 3 + 5 + 7 + reduce(plus, [9, 11, 13, 15, 17, 19])
 = 1 + 3 + 5 + 7 + 9 + reduce(plus, [11, 13, 15, 17, 19])
 = 1 + 3 + 5 + 7 + 9 + 11 + reduce(plus, [13, 15, 17, 19])
 = 1 + 3 + 5 + 7 + 9 + 11 + 13 + reduce(plus, [15, 17, 19])
 = 1 + 3 + 5 + 7 + 9 + 11 + 13 + 15 + reduce(plus, [17, 19])
 = 1 + 3 + 5 + 7 + 9 + 11 + 13 + 15 + 17 + reduce(plus, [19])
 = 1 + 3 + 5 + 7 + 9 + 11 + 13 + 15 + 17 + 19
 = 100
```

```
% python reduce.py
100
```

Why learn functional programming?

Good reasons to learn a programming language
- Offers something new. ✓
- Need to interface with co-workers. ✓
- Better than Java for the application at hand. ✓
- Provides an intellectual challenge ✓
- Opportunity to learn something about computation. ✓
- Introduces a new programming style. ✓

Intro CS at MIT is taught in
Scheme (a functional language)
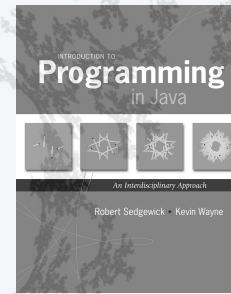
Modern applications
- Communications systems
- Financial systems
- Google MapReduce

Functional Programming Jobs ?!!

Deep and direct connections to theoretical CS (stay tuned).

⚠ Warning. Functional programming may be addictive.

45

Programming in Java

An Interdisciplinary Approach

Robert Sedgewick · Kevin Wayne

http://introcs.cs.princeton.edu

16. Programming Languages

- Popular languages
- Java in context
- Object-oriented programming
- Type checking
- **Functional programming**
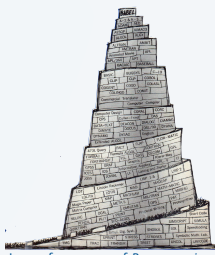

The Tower of Babel

Image from cover of *Programming Languages* by Jean Sammet (1969).
Already 120+ 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
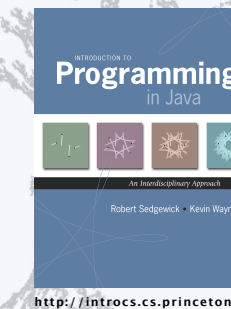 "*confounds the language of all the earth*"

- Why?

Proliferation of cultural differences (and multiple languages) is one basis of civilization.

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?

47

Programming in Java

An Interdisciplinary Approach

Robert Sedgewick · Kevin Wayne

http://introcs.cs.princeton.edu

16. Programming Languages