

<http://introcs.cs.princeton.edu>

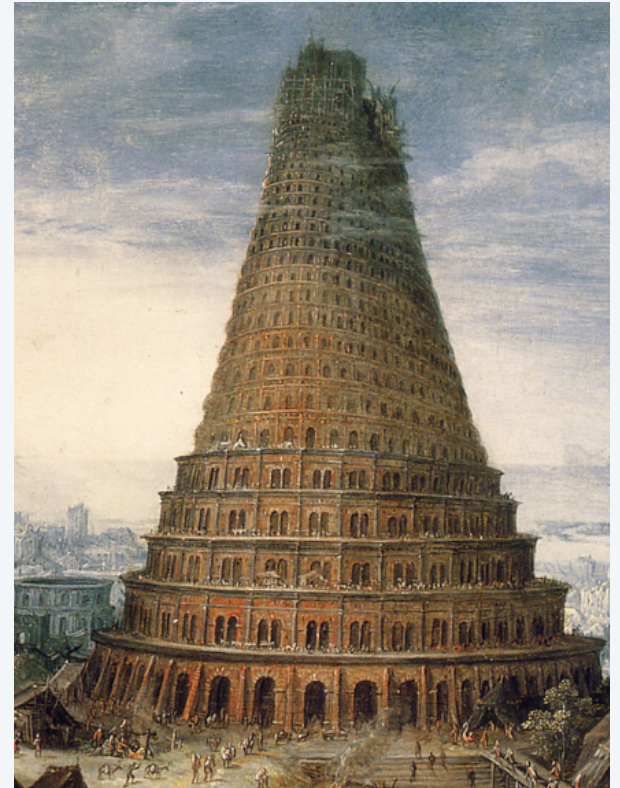
# 16. Programming Languages

# The Tower of Babel

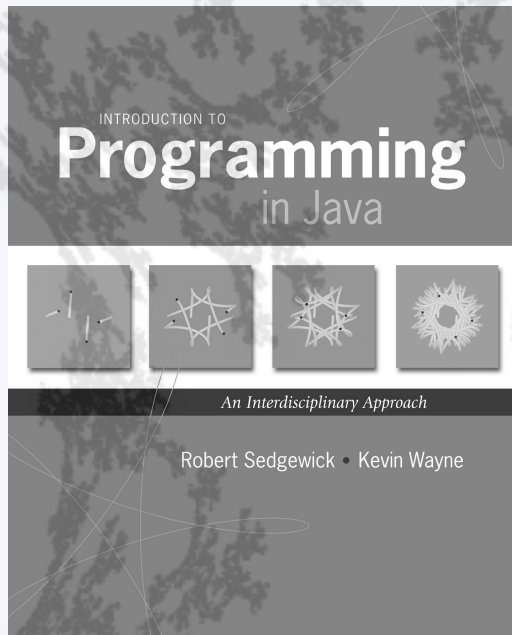
---

## 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.



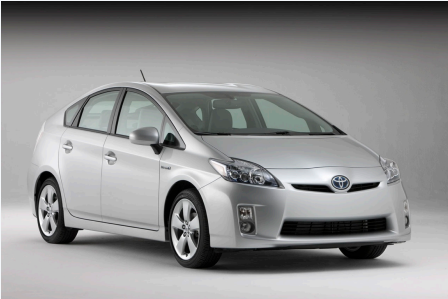
<http://introcs.cs.princeton.edu>

## 16. Programming Languages

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

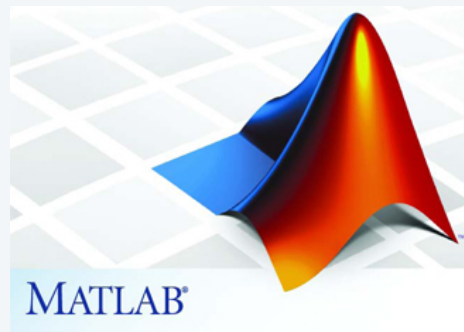
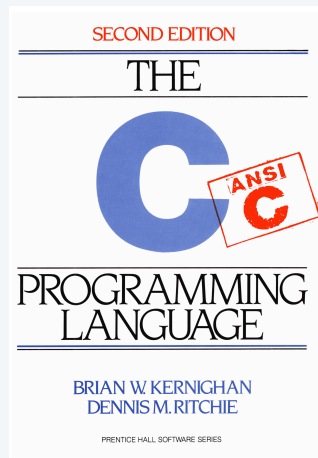
# Several ways to solve a transportation problem

---



## 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

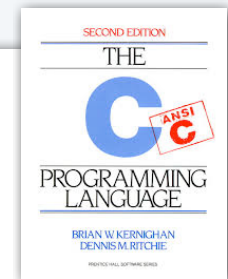
- library conventions
- array creation idiom
- standard input idiom
- pointer manipulation (stay tuned)

## 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("%d %d %d\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>
#include <cstdlib>
int 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++)
        std::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)
                    std::cout << a[i] << " " << a[j] << " " << a[k] << std::endl;
}
```



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

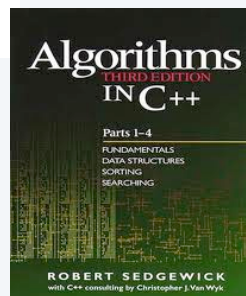
# C++

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)



1990

## BST.cxx

```
template <class Item, class Key>
class ST
{
    private:
        struct node
        { Item item; node *left, *right;
          node(Item x)
          { item = x; left = 0; right = 0;}
        };
        typedef node *link;
        link head;

        Item searchR(link x, Key key)
        { if (x == 0) return 0;
          Key t = x->item.key();
          if (key == t) return x->item;
          if (key < t) return searchR(x->left, key);
          else return searchR(x->right, key);
        }
        ...
    }
}
```



## 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



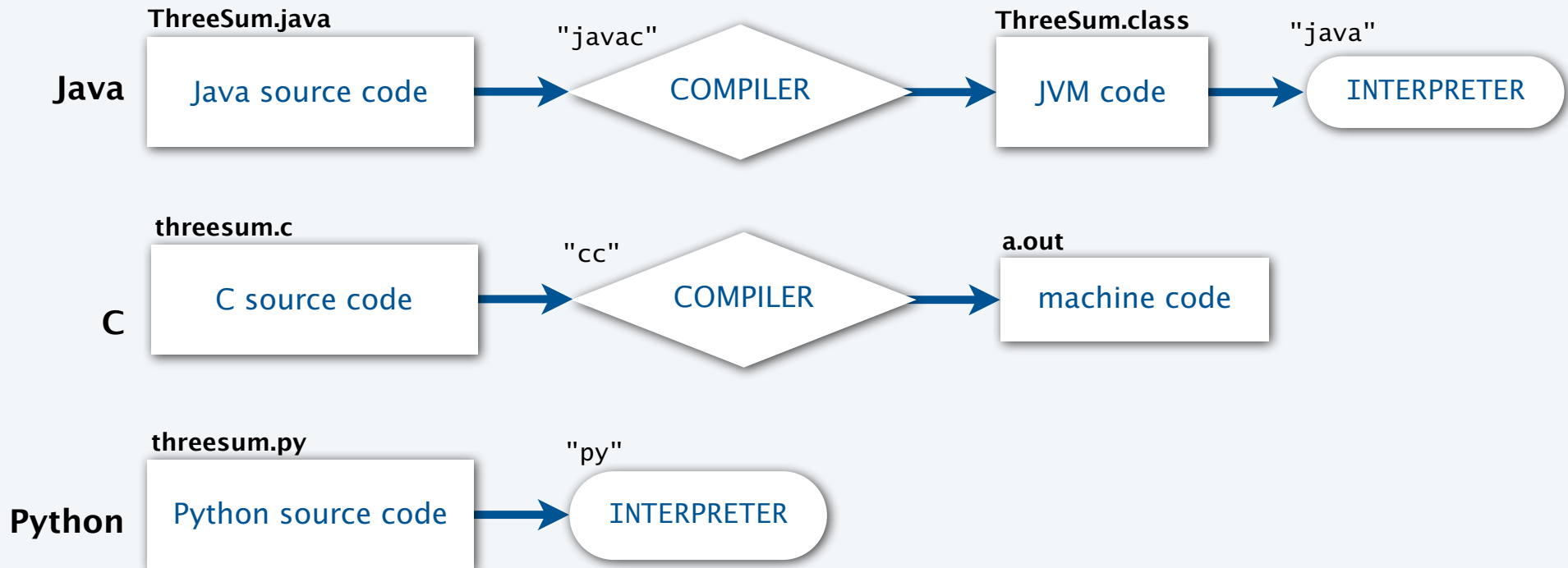
```
import sys
import stdio
N = int(sys.argv[1])
a = [0]*N
for i in range(N):
    a[i] = int(sys.stdin.readline())
for i in range(N):
    for j in range(i+1, N):
        for k in range(j+1, N):
            if (a[i] + a[j] + a[k]) == 0:
                stdio.writef('%d %d %d\n', a[i], a[j], a[k])
```

```
% 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.



## 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 type error in a big program.
- Program runs for hours or days (because Python is 50-100 times slower than Java).
- Program crashes without writing results.

### Reasonable approaches

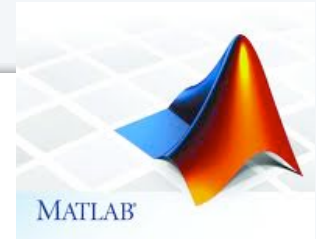
- Throw out your calculator; use Python.
- Prototype in Python, then convert to Java for "production" use.

# 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
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
```

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



## 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

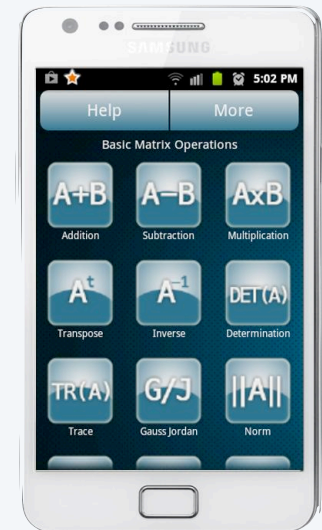
*"redefine the value of the complex number  $i$  to be a 1-by-1 matrix whose entry is 0"*

### Notes

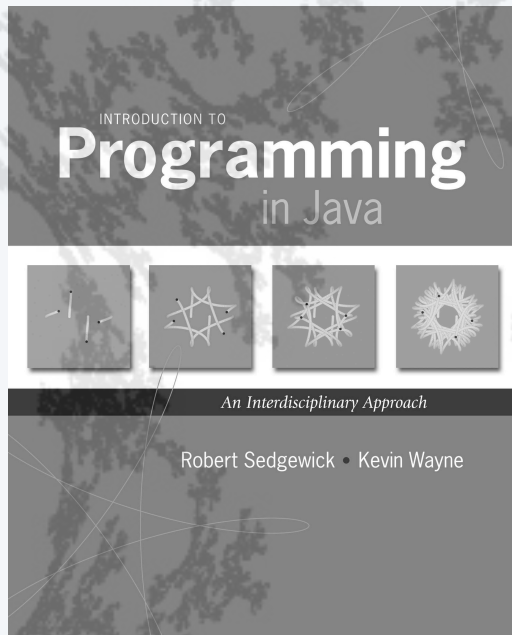
- Matlab is written in Java.
- The Java compiler and interpreters are written in C.  
[Modern C compilers are written in C.]
- Matrix libraries (written in C) are accessible from C/C++/Java/Python.

### Reasonable approaches

- Use Matlab as a "matrix calculator" and data analysis (if you own it).
- Convert to or use Java/C/C++ if you want to do anything else.



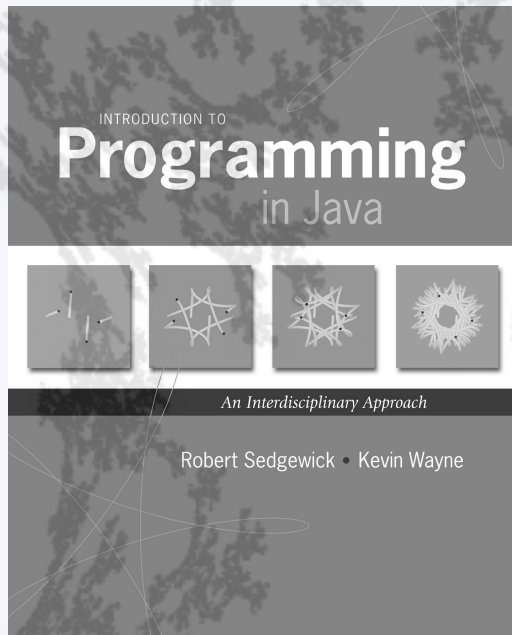
Matrix calculator  
Android app



<http://introcs.cs.princeton.edu>

## 16. Programming Languages

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



<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.

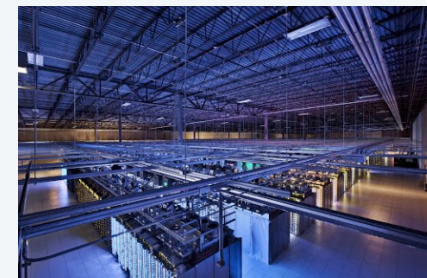


James Gosling  
<http://java.net/jag>






## Java economy

- Mars rover.
- Cell phones.
- Blu-ray Disc.
- Web servers.
- Medical devices.
- Supercomputing.
- ...

← \$100 billion,  
3 million developers  
9 billion devices



## Why do we use Java in this course?

<i>language</i>	<i>widely used</i>	<i>widely available</i>	<i>full set of modern abstractions</i>	<i>modern libraries and systems</i>	<i>automatic checks for bugs</i>
	✓	✓	✗	✗	✓
	✓	✓	✓	maybe	✓
	✓	✓	✓	✓	✓
	✓	\$	maybe*	✓	✗
	✓	✓	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

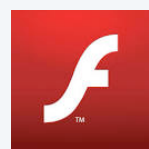
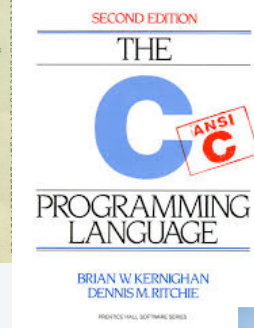
- data abstraction (OOP)
- extensive libraries

### 2000s: AJAX/PHP/Ruby/Flash





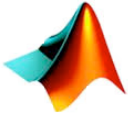




- scripting
- libraries for web development

name	address	code	remarks
ADD I(1)	AD	14	R1,R2
ADD I(2)	AD	5A	RX R1,D2(X2,B2)
ADD Decimal (c,d)	AD	FA	D(1,L1,B1),D2(L2,B2)
ADD Logical (c)	AD	AA	R1,R2
ADD Logical (d)	ALH	1C	RX R1,R2
AND I(1)	AN	14	R1,R2
AND I(2)	AN	54	RX R1,D2(X2,B2)
AND I(3)	AN	34	D(1,L1,B1)
AND I(4)	AN	D4	SS D(1,L1,D2(B2))
AND I(5)	AN	74	R1,R2
AND I(6)	AN	3C	R1,R2
AND I(7)	AN	5C	RX R1,D2(X2,B2)
AND I(8)	AN	7C	R1,R2
AND I(9)	AN	5E	RX R1,D2(X2,B2)
AND I(10)	AN	7E	R1,R2
AND I(11)	AN	5F	RX R1,D2(X2,B2)
AND I(12)	AN	7F	R1,R2
AND I(13)	AN	5D	RX R1,D2(X2,B2)
AND I(14)	AN	7D	R1,R2
AND I(15)	AN	5B	RX R1,D2(X2,B2)
AND I(16)	AN	7B	R1,R2
AND I(17)	AN	59	RX R1,D2(X2,B2)
AND I(18)	AN	79	R1,R2
AND I(19)	AN	57	RX R1,D2(X2,B2)
AND I(20)	AN	77	R1,R2
AND I(21)	AN	55	RX R1,D2(X2,B2)
AND I(22)	AN	75	R1,R2
AND I(23)	AN	53	RX R1,D2(X2,B2)
AND I(24)	AN	73	R1,R2
AND I(25)	AN	51	RX R1,D2(X2,B2)
AND I(26)	AN	71	R1,R2
AND I(27)	AN	59	RX R1,D2(X2,B2)
AND I(28)	AN	79	R1,R2
AND I(29)	AN	57	RX R1,D2(X2,B2)
AND I(30)	AN	77	R1,R2
AND I(31)	AN	55	RX R1,D2(X2,B2)
AND I(32)	AN	75	R1,R2
AND I(33)	AN	53	RX R1,D2(X2,B2)
AND I(34)	AN	73	R1,R2
AND I(35)	AN	51	RX R1,D2(X2,B2)
AND I(36)	AN	71	R1,R2
AND I(37)	AN	59	RX R1,D2(X2,B2)
AND I(38)	AN	79	R1,R2
AND I(39)	AN	57	RX R1,D2(X2,B2)
AND I(40)	AN	77	R1,R2
AND I(41)	AN	55	RX R1,D2(X2,B2)
AND I(42)	AN	75	R1,R2
AND I(43)	AN	53	RX R1,D2(X2,B2)
AND I(44)	AN	73	R1,R2
AND I(45)	AN	51	RX R1,D2(X2,B2)
AND I(46)	AN	71	R1,R2
AND I(47)	AN	59	RX R1,D2(X2,B2)
AND I(48)	AN	79	R1,R2
AND I(49)	AN	57	RX R1,D2(X2,B2)
AND I(50)	AN	77	R1,R2
AND I(51)	AN	55	RX R1,D2(X2,B2)
AND I(52)	AN	75	R1,R2
AND I(53)	AN	53	RX R1,D2(X2,B2)
AND I(54)	AN	73	R1,R2
AND I(55)	AN	51	RX R1,D2(X2,B2)
AND I(56)	AN	71	R1,R2
AND I(57)	AN	59	RX R1,D2(X2,B2)
AND I(58)	AN	79	R1,R2
AND I(59)	AN	57	RX R1,D2(X2,B2)
AND I(60)	AN	77	R1,R2
AND I(61)	AN	55	RX R1,D2(X2,B2)
AND I(62)	AN	75	R1,R2
AND I(63)	AN	53	RX R1,D2(X2,B2)
AND I(64)	AN	73	R1,R2
AND I(65)	AN	51	RX R1,D2(X2,B2)
AND I(66)	AN	71	R1,R2
AND I(67)	AN	59	RX R1,D2(X2,B2)
AND I(68)	AN	79	R1,R2
AND I(69)	AN	57	RX R1,D2(X2,B2)
AND I(70)	AN	77	R1,R2
AND I(71)	AN	55	RX R1,D2(X2,B2)
AND I(72)	AN	75	R1,R2
AND I(73)	AN	53	RX R1,D2(X2,B2)
AND I(74)	AN	73	R1,R2
AND I(75)	AN	51	RX R1,D2(X2,B2)
AND I(76)	AN	71	R1,R2
AND I(77)	AN	59	RX R1,D2(X2,B2)
AND I(78)	AN	79	R1,R2
AND I(79)	AN	57	RX R1,D2(X2,B2)
AND I(80)	AN	77	R1,R2
AND I(81)	AN	55	RX R1,D2(X2,B2)
AND I(82)	AN	75	R1,R2
AND I(83)	AN	53	RX R1,D2(X2,B2)
AND I(84)	AN	73	R1,R2
AND I(85)	AN	51	RX R1,D2(X2,B2)
AND I(86)	AN	71	R1,R2
AND I(87)	AN	59	RX R1,D2(X2,B2)
AND I(88)	AN	79	R1,R2
AND I(89)	AN	57	RX R1,D2(X2,B2)
AND I(90)	AN	77	R1,R2
AND I(91)	AN	55	RX R1,D2(X2,B2)
AND I(92)	AN	75	R1,R2
AND I(93)	AN	53	RX R1,D2(X2,B2)
AND I(94)	AN	73	R1,R2
AND I(95)	AN	51	RX R1,D2(X2,B2)
AND I(96)	AN	71	R1,R2
AND I(97)	AN	59	RX R1,D2(X2,B2)
AND I(98)	AN	79	R1,R2
AND I(99)	AN	57	RX R1,D2(X2,B2)
AND I(100)	AN	77	R1,R2

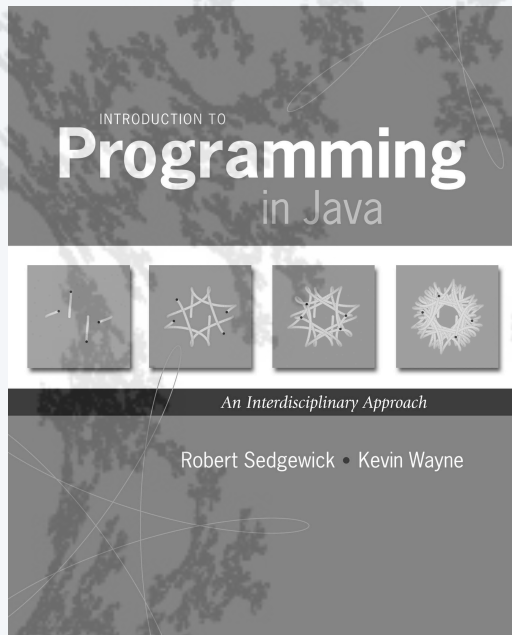
Branch and Store (e)	BAS	4D	RX	R1,D2(X2,B2)
Branch on Condition	BCR	07	RR	M1,R2
Branch on Condition	BC	47	RX	M1,D2(X2,B2)
Branch on Count	BCTR	06	RR	R1,R2
Branch on Count	BCT	46	RX	R1,D2(X2,B2)
Branch on Index High	BXH	86	RS	R1,R3,D2(B2)
Branch on Index Low or Equal	BXLE	87	RS	R1,R3,D2(B2)
Compare (c)	CR	19	RR	R1,R2
Compare (c)	C	59	RX	R1,D2(X2,B2)
Compare Decimal (c,d)	CP	F9	SS	D1(L1,B1),D2(L2,B2)
Compare Halfword (c)	CH	49	RX	R1,D2(X2,B2)
Compare Logical (c)	CLR	15	RR	R1,R2
Compare Logical (c)	CL	55	RX	R1,D2(X2,B2)
Compare Logical (c)	CLC	D5	SS	D1(L,B1),D2(B2)
Compare Logical (c)	CLI	95	SI	D1(B1),I2
Convert to Binary	CVB	4F	RX	R1,D2(X2,B2)
Convert to Decimal	CVD	4E	RX	R1,D2(X2,B2)
Convert to Decimal	DR	1D	RR	R1,R2



## Programming styles

<i>style</i>	<i>execution model</i>	<i>examples</i>
procedural	step-by-step instruction execution usually compiled	
scripted	step-by-step command execution usually interpreted	 python 
special-purpose	optimized around certain data types	  MATLAB
object-oriented	focus on objects that do things	 Java 
functional	executable functions can be arguments or return values, or stored as data	 OCaml  HASKELL

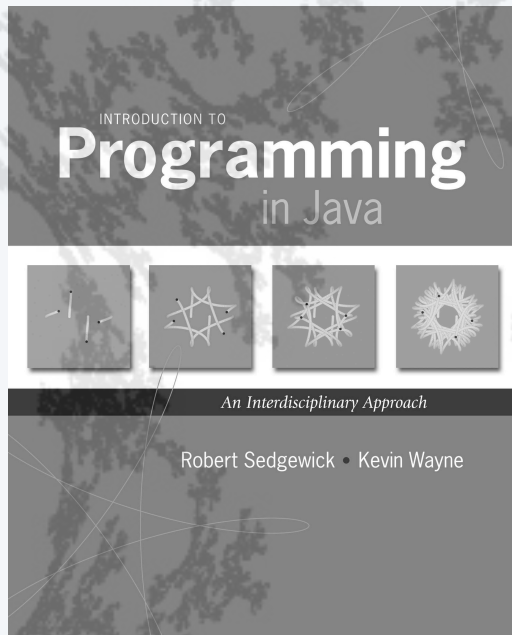




<http://introcs.cs.princeton.edu>

## 16. Programming Languages

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



<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.

## Object 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.

### 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.



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

## 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



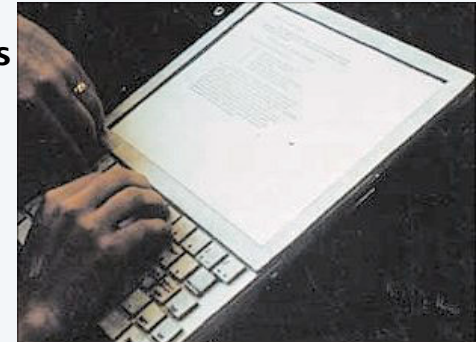
Typical "mainframe" computer: IBM 360/50

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)*

↑  
Still relevant today!

– Alan Kay

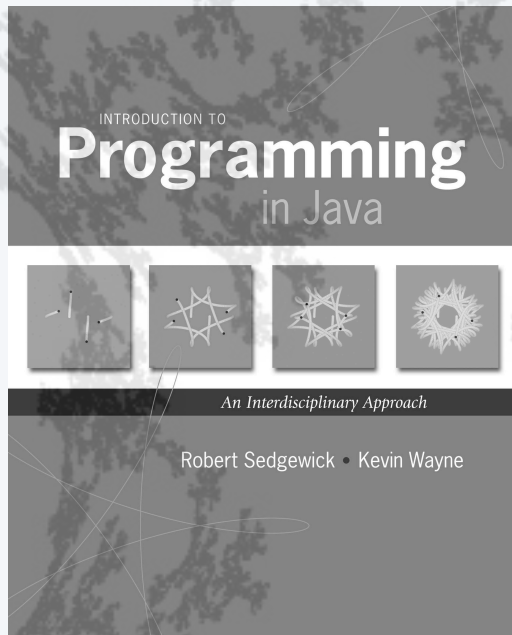


2010s



Modern personal computer  
MacBook Air

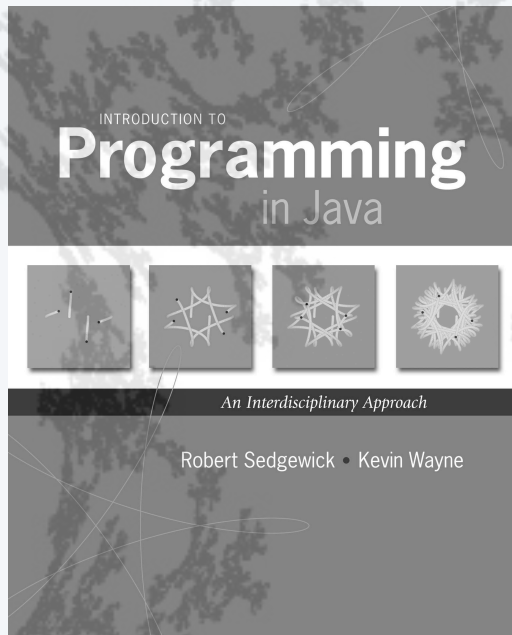
Key feature: **OOP software** (Objective C)



<http://introcs.cs.princeton.edu>

## 16. Programming Languages

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



<http://introcs.cs.princeton.edu>

## 16. Programming Languages

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



## 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?



## 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.”*

— Edsger Dijkstra (1969)

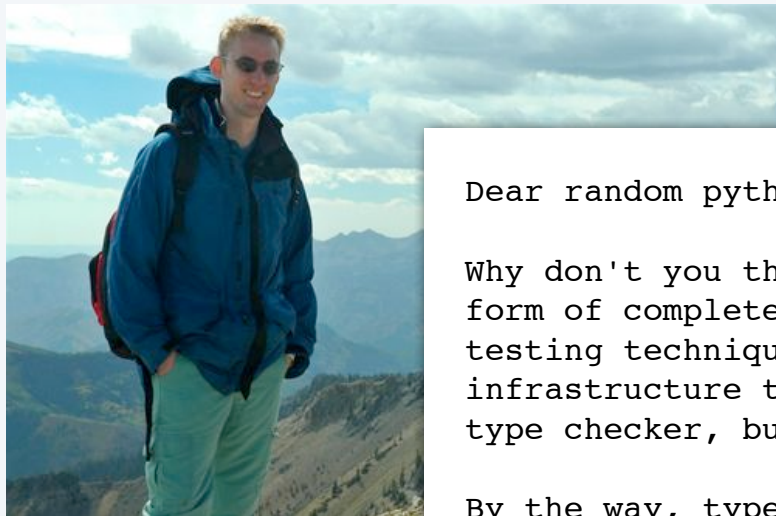


*“ Since static type checking can't cover all possibilities, you will need automated testing. Once you have automated testing, static type checking is redundant.”*

— Python blogger (2009)

## A letter from 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)

## 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)



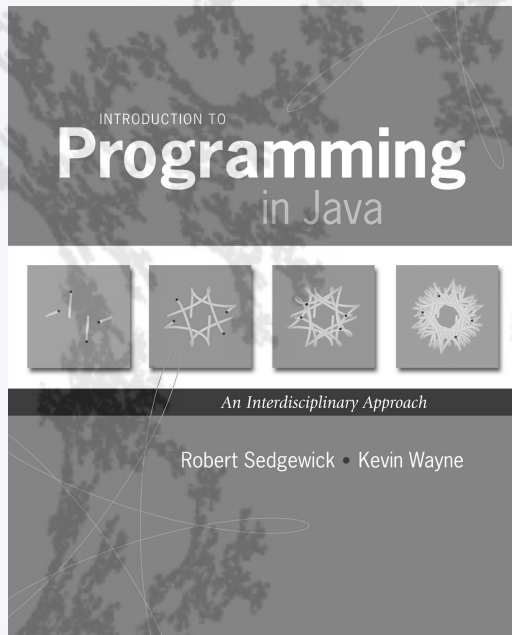
Owns 230' luxury yacht Skat



Dated Martha Stewart (1993-2008)



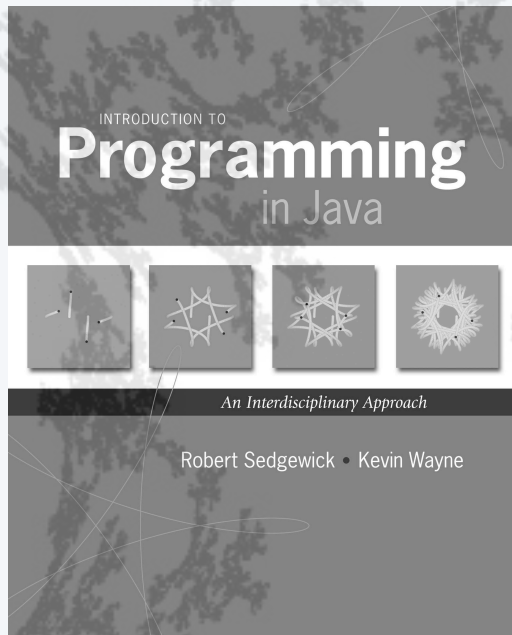
Windows 2000 mansion in Seattle



<http://introcs.cs.princeton.edu>

## 16. Programming Languages

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



<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



Java<sup>8</sup>

 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 square(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, sequence):  
    for x in sequence:  
        print x,  
        print f(x)
```

print a table of the  
squares of the numbers  
from 0 to 9

```
table (square, 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.

$\text{MAP}(f, \text{sequence})$  is the result of replacing every  $x$  in sequence by  $f(x)$ .

**map.py**

```
def square(x):  
    return x*x  
  
def odd(x):  
    return 2*x + 1  
  
print map (odd, range(10))  
print map (square, range(10))
```

a function that returns the square of its argument →

print the squares of the numbers from 0 to 9 →

```
% 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.

$\text{REDUCE}(f, L)$  is  $f(\text{car}(L), \text{REDUCE}(f, \text{cdr}(L)))$ .

↑  
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)))
```

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

```
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
```

## 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 was taught in Scheme (a functional language) for decades

### Modern applications

- Communications systems
- Financial systems
- Google MapReduce

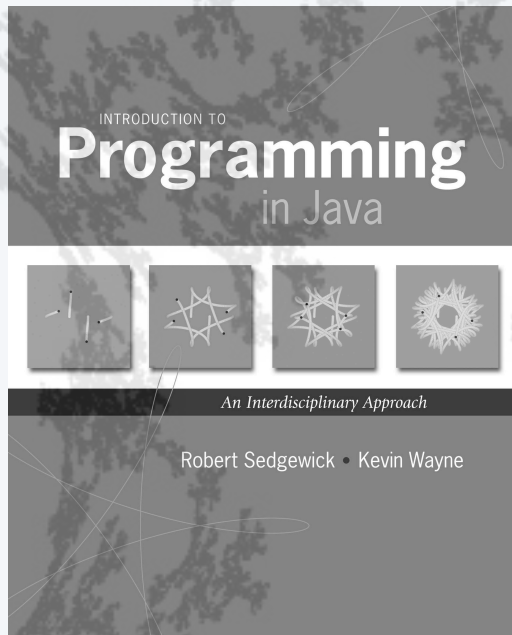
Deep and direct connections to theoretical CS (stay tuned).



**Warning.** Functional programming may be addictive.



*Functional Programming Jobs ?!?!?!?*

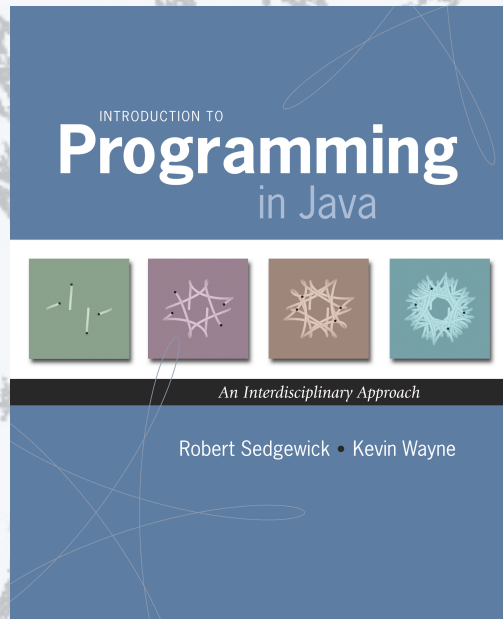


<http://introcs.cs.princeton.edu>

## 16. Programming Languages

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





<http://introcs.cs.princeton.edu>

# 16. Programming Languages