

What's Inside

new releases2-3

Teaching Fluency with Information Technology:
Q & A with Eugene Stafford, *Iona College*

Combining Form and Function:
The Programming Logic and Design Course
Elizabeth Drake, *Santa Fe Community College*

teaching tips4-5

Highly Effective Teaching Practices
Tony Gaddis, *Haywood Community College*

the latest from Addison-Wesley6-7

college spotlight8

Alice in UNCWland: Using Alice to Motivate and
Engage Beginning Programmers
Laurie Patterson, *University of North Carolina,
Wilmington*

The Loop is Moving Online!

For Spring 2008, Addison-Wesley is planning to move *The Loop* into a new searchable, online format, which will allow readers access to stories that are of particular use to them. We hope these issues are a valuable resource for your teaching and help you stay up-to-date with computer science higher education news.

■ Fresh Introductory Approaches

Computing instructors often tell us about students who enroll in computing because they like to play video games. The story always ends with the student dropping the course after just a few classes. These students say that computing isn't what they thought it was going to be. For many, the realization that a high score in "Dance, Dance Revolution" isn't strongly correlated to algorithmic thinking is discouraging. It is also true that at some point in the past, instructors might have happily let these students move on.

Today, however, we're hearing about a growing number of courses designed to appeal to these types of students—those who are curious about what's going on behind the scenes in the computer systems they use, and may just need a little more time to gain comfort with the basic logic of computing.

In this issue of *The Loop*, Eugene Stafford of Iona College discusses the development of their fluency course (based on the AW book written by Larry Snyder). The course introduces a general student population to computing by teaching not only skills, but concepts and capabilities, too. Graduates of fluency are confident, thoughtful users of technology who feel comfortable when a system crashes or when directions are unclear. Fluency posits that problem solving and reasoning are central to an introductory IT course—and not exclusive to math and physical science courses.

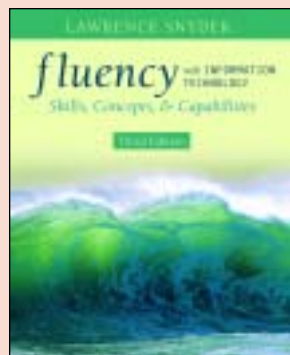
Also in this issue, Elizabeth Drake of Santa Fe Community College in Gainesville, FL, argues for introducing students to basic programming in a pre-CS1 computing logic course that dispenses with the complicated syntax of an industrial-strength language. These are students who might typically drop a standard CS1 course early on.

Starting Out with Alice: A Visual Introduction to Programming by Tony Gaddis meets students halfway by embracing their comfort with video games, virtual worlds, and graphical interfaces. Students manipulate images through a graphically based set of tools that replicate commands in some of today's higher-level languages. Students begin to grasp important programming concepts and are more prepared to deal with the logic in CS1 courses, while also having fun!

As more students return to computing classrooms, these approaches offer a few ways to keep students engaged through the end of your course. Best wishes for a great Fall 2007!

Matt Goldstein
Acquisitions Editor
Addison-Wesley Computing
matt.goldstein@aw.com





Fluency with Information Technology: Skills, Concepts, and Capabilities, Third Edition

Lawrence Snyder
978-0-321-51239-0 • 0-321-51239-1
© 2008 • 792 pages

Contents

Part 1. BECOMING SKILLED AT INFORMATION TECHNOLOGY

1. Terms of Endearment: Defining Information Technology
2. What the Digerati Know: Exploring the Human-Computer Interface
3. Making the Connection: The Basics of Networking
4. Marking Up with HTML: A Hypertext Markup Language Primer
5. Searching for Truth: Locating Information on the WWW
6. Searching for Guinea Pig B: Case Study in Online Research

Part 2. ALGORITHMS AND DIGITIZING INFORMATION

7. To Err Is Human: An Introduction to Debugging
8. Bits and the "Why" of Bytes: Representing Information Digitally
9. Following Instructions: Principles of Computer Operation
10. What's the Plan?: Algorithmic Thinking
11. Light, Sound, Magic: Representing Multimedia Digitally

Part 3. DATA AND INFORMATION

12. Computers in Polite Society: Social Implications of IT
13. Shhh, It's a Secret: Privacy and Digital Security
14. Fill-in-the-Blank Computing: Basics of Spreadsheets
15. "What If" Thinking Helps: Advanced Spreadsheets for Planning
16. A Table with a View: Database Queries
17. iDiary: A Case Study in Database Design



Teaching Fluency with Information Technology

Q & A with Eugene Stafford, Associate Professor of Computer Science, *Iona College*

Q: What are the reasons behind your switch to teaching fluency?

A: We have been teaching a "computer literacy" or a "fluency" course since the 70s. Originally it was programming in the 70s and early 80s, then slowly it became an applications course with word processing, spreadsheets, and databases, and then came Microsoft® Office. The course evolved into a Microsoft Office course. The problem was also that the quality of the textbooks depreciated and they turned into "picture books"—the course simply became a "how to" course based on Office. The question emerged: Was this really college-level content and were these "picture books" really college-level books?

Q: What are the differences between your current fluency-based course and traditional literacy courses?

A: The intent of this course must be concepts, and whatever hands-on material is needed should be based on content and concepts, not simply teaching an application. Concepts and theory remain constant while application software is constantly changing. We do not teach applications—there is no word processing in this college course. While we use spreadsheets and database software, it is to implement theory. We do not teach applications per se, we use applications. It is not college-level material if you are simply teaching Microsoft Office. Also, for programming concepts we use HTML and JavaScript™.

Q: How have your students responded to the fluency course?

A: They particularly enjoy the HTML and JavaScript. They do not find it a repetition of their high school course and they appreciate the fact that it is more of a challenge.

Q: What are the benefits of teaching fluency?

A: Above all, we can cover topics such as problem solving and debugging, topics that will be used and needed, not only in college, but in the students' lives.

Part 4. PROBLEM SOLVING

18. Get with the Program: Fundamental Concepts Expressed in JavaScript
19. The Bean Counter: A JavaScript Program
20. Thinking Big: Programming Functions
21. Once Is Not Enough: Iteration Principles
22. The Smooth Motion: Case Study in Algorithmic Problem Solving
23. Computers Can Do Almost (Everything, Nothing): Limits to Computation
24. A Fluency Summary: Click to Close

APPENDICES

- Appendix A: HTML Reference
- Appendix B: iDiary Database
- Appendix C: JavaScript Programming Rules
- Appendix D: Bean Counter Program
- Appendix E: Memory Bank Program
- Appendix F: Smooth Motion Program



Starting Out with Programming Logic and Design

Tony Gaddis
978-0-321-47127-7 • 0-321-47127-X
© 2008 • 650 pages



Extended Prelude to Programming, Third Edition

Stewart Venit & Elizabeth Drake
978-0-321-41851-7 • 0-321-41851-4
© 2007 • 560 pages

Also available in a concise version:

Concise Prelude to Programming, Third Edition

Stewart Venit & Elizabeth Drake
978-0-321-48266-2 • 0-321-48266-2
© 2007 • 400 pages

Combining Form and Function: The Programming Logic and Design Course

Elizabeth Drake, *Santa Fe Community College*

Think back to your algebra class in high school. We all learned that to solve an equation for x , the first step is to collect like terms. Using algebra, the equation $2x + 3 = 7 + x$ becomes $2x - x = 7 - 3$, ending with $x = 4$. Now recall the first time you saw a similar expression in a programming language, $x = x + 1$. It made no sense. How can $x = x + 1$? Doesn't that mean $0 = 1$? Of course not. It's just an assignment statement.

Assignment statements, like `for` loops and swap routines, are such integral parts of writing a computer program that they are second nature to those of us who have done any programming. To students with years of basic math training and not one day of programming instruction, however, this simple equation is far from intuitive.

At SFCC, we found that our students needed a course that focused specifically on the basic logic of programming before they tackled a modern, high-level language replete with GUI interface. Our Programming Logic and Design course takes beginning students through both procedural and OOP design. Students learn why, in a computer program, the integer "6" is different from the real number "6.0." They see that $x = x + 1$ actually makes sense and recognize loops and arrays as familiar instead of foreign concepts. We prepare students with a foundation for the logic they will use in programming for software development, database programming, and Web-based applications.

The challenge of this course is to engage students and ensure that they see the real-world value of learning these concepts. A language-specific introductory course does not focus students on the critical concepts because so much time is spent learning an interface and debugging syntax errors. On the other hand, a course that uses only pseudocode does not afford students the opportunity to see the results of their hard work applied to a final product.

To address this dichotomy, we decided to combine form and function in one course. We use a non-language-specific approach and integrate a hands-on component to teach programming logic. After experimenting with other approaches, we chose to use Alice in the course. Alice allows students to use the concepts and basic programming constructs to design, debug, run, and see pseudocode in action.

The results have been extremely positive. Students enjoy the Programming Logic and Design course and go on to begin programming in a modern language with comprehension and confidence. Alice assignments parallel the concepts taught throughout the semester and the focus remains where it belongs—on the logic of programming.

Tony Gaddis is the author of the best-selling *Starting Out with* series, which is written specifically for instructors who want an accessible and step-by-step presentation of introductory programming concepts. Included in this series are books covering Java™, C++, Alice, and Visual Basic®, all of which have been designed to fit a variety of teaching approaches. Every book in the *Starting Out with* series conforms to the teaching tips mentioned here, making them ideal for promoting successful student learning.

Starting Out with Series Features:

- Written in down-to-earth, easy-to-understand language
- Demonstrates abstract concepts with an abundant number of practical examples
- Thoroughly explains the syntactic details that most beginning students have trouble with
- Includes a wealth of short exercises, Checkpoint questions, review questions, and programming exercises

Highly Effective Teaching Practices

Tony Gaddis, *Haywood Community College*

Do you teach beginners or non-major students in your introductory programming classes? Here are some tips from Tony Gaddis, a veteran teacher with twenty years of experience at Haywood Community College.

Concrete First, Then Abstract

Beginners often have trouble understanding the abstract concepts they encounter in a programming class. Here are some tips for starting your presentations with concrete examples and explanations:

- When explaining abstract topics, try to use examples students can relate to. For example, most students are acquainted with the process of doing a Google search. You can use something like this to explain the concepts of input, processing, and output.
- Try to explain abstract topics in a way that provides students with a mental picture of something. For example, the process of emptying a gumball machine, one piece of gum at a time, might be used to describe loops. Everyday devices like alarm clocks and telephones can be used as metaphors for objects.

The Gaddis Series *Understanding from the Start*



**Starting Out with C++:
From Control Structures
through Objects, Fifth Edition**
Gaddis © 2007
978-0-321-40939-3 • 0-321-40939-6



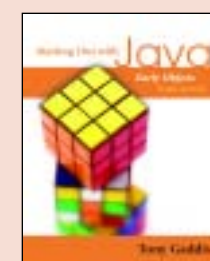
**Starting Out with C++:
Early Objects,
Sixth Edition**
Gaddis/Walters/Muganda © 2008
978-0-321-51238-3 • 0-321-51238-3



**Starting Out with C++:
Brief, Fifth Edition**
Gaddis/Krupnow © 2007
978-0-321-41291-1 • 0-321-41291-5



**Starting Out with Java:
From Control Structures
through Objects, Third Edition**
Gaddis © 2008
978-0-321-47927-3 • 0-321-47927-0



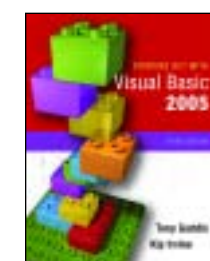
**Starting Out with Java:
Early Objects, Third Edition**
Gaddis © 2008
978-0-321-49768-0 • 0-321-49768-6



**Starting Out with Java:
From Control Structures
through Data Structures**
Gaddis/Muganda © 2007
978-0-321-42102-9 • 0-321-42102-7



**Advanced Visual Basic
2005, Fourth Edition**
Irvine/Gaddis © 2007
978-0-321-47712-5 • 0-321-47712-X



**Starting Out with
Visual Basic 2005,
Third Edition**
Gaddis/Irvine © 2007
978-0-321-39399-9 • 0-321-39399-6



**Starting Out with Alice:
A Visual Introduction
to Programming**
Gaddis © 2008
978-0-321-47515-2 • 0-321-47515-1



**Starting Out with
Programming
Logic and Design**
Gaddis © 2008
978-0-321-47127-7 • 0-321-47127-X

- Be careful not to use programming jargon until your students understand it. Rather than asking what an unexplained term means, some students will make an incorrect assumption about its meaning. Can you imagine what a novice student might think the term “mutator” means if you forget to explain it?

Sweat the Small Stuff!

When it comes to programming, the devil is in the details. In addition to learning the logic of programming, beginners must take time to learn syntactic details like where to place semicolons, what to put inside braces, etc. However, many want to focus on the big picture of what a program is supposed to do, especially “right-brainers.” When they don’t take time to learn the details of their programming language, they get frustrated because they don’t understand the errors in their programs.

As an instructor, keep in mind:

- The things that are obvious to you are not obvious to your students.
- No detail of a programming construct is too small to explain in class.
- When important details are left unexplained, students often assume they are unimportant.

Use Repetition

Doing a particular task over and over again strengthens our ability to do that task. This is why athletes continually practice their sport and musicians rehearse their music. Repetition can also play an important role in an introductory computer science class. When a new programming construct is introduced, you might want to give your students two or three “code snippet” assignments related to that construct. For example, after you first introduce the `if` statement, you might have your students pull out a sheet of paper and do the following exercises:

1. Write an `if` statement that assigns 100 to the variable `x` if the variable `y` is equal to 0.
2. Write an `if-else` statement that assigns 0 to the variable `x` if the variable `y` is equal to 10. Otherwise assign 1 to `x`.
3. Write an `if` statement that sets the variable `hours` to 10 if the flag variable `minimum` is true.

These assignments can be done quickly in class, and students can get immediate feedback from you and other students.

Please visit us at aw.com/computing to see a full list of our computing titles in these areas:

Introduction to Computer Science • Literacy/Fluency • Programming • Data Structures Algorithms • Computer Ethics • Web Programming • Database • Operating Systems Computer Graphics • Networking • Security • Software Engineering Theory of Computation • User Interface Design • Artificial Intelligence

To order any Addison-Wesley title, e-mail us at computing@aw.com

FLUENCY



Fluency with Information Technology: Skills, Concepts, and Capabilities, 3e
Snyder © 2008
978-0-321-51239-0 • 0-321-51239-1

PROGRAMMING LOGIC



Starting Out with Programming Logic and Design
Gaddis © 2008
978-0-321-47127-7 • 0-321-47127-X

INTRODUCTORY ALICE PROGRAMMING



Starting Out with Alice: A Visual Introduction to Programming
Gaddis © 2008
978-0-321-47515-2 • 0-321-47515-1

INTRODUCTORY MATLAB PROGRAMMING



Engineering Computation with MATLAB
Smith © 2008
978-0-321-48108-5 • 0-321-48108-9

COMPUTER NETWORKING



Computer Networking: A Top-Down Approach, 4e
Kurose/Ross © 2008
978-0-321-49770-3 • 0-321-49770-8

INTRODUCTORY C++ PROGRAMMING

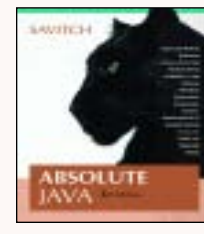


Absolute C++, 3e
Savitch © 2008
978-0-321-46893-2 • 0-321-46893-7



Starting Out with C++: Early Objects, 6e
Gaddis/Walters/Muganda © 2008
978-0-321-51238-3 • 0-321-51238-3

INTRODUCTORY JAVA PROGRAMMING



Absolute Java, 3e
Savitch © 2008
978-0-321-48792-6 • 0-321-48792-3

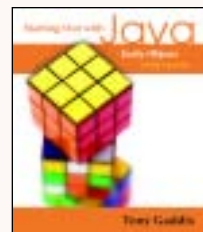


The Art and Science of Java
Roberts © 2008
978-0-321-48612-7 • 0-321-48612-9



Introduction to Programming in Java: An Interdisciplinary Approach
Sedgewick/Wayne © 2008
978-0-321-49805-2 • 0-321-49805-4

INTRODUCTORY JAVA PROGRAMMING, CONTINUED



Starting Out with Java: Early Objects, 3e
Gaddis © 2008
978-0-321-49768-0 • 0-321-49768-6



Starting Out with Java: From Control Structures through Objects, 3e
Gaddis © 2008
978-0-321-47927-3 • 0-321-47927-0



Building Java Programs: A Back to Basics Approach
Reges/Stepp © 2008
978-0-321-38283-2 • 0-321-38283-8



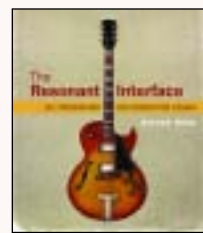
Java Foundations: Introduction to Program Design and Data Structures
Lewis/DePasquale/Chase © 2008
978-0-321-42972-8 • 0-321-42972-9

INTERNET/WORLD WIDE WEB



Web 101, 3e
Lehnert/Kopec © 2008
978-0-321-42467-9 • 0-321-42467-0

HCI/USER INTERFACE DESIGN



The Resonant Interface: HCI Foundations for Interaction Design
Heim © 2008
978-0-321-37596-4 • 0-321-37596-3

A+ CERTIFICATION/PC REPAIR



The Complete A+ Guide to PC Repair, 4e
Schmidt © 2008
978-0-321-51358-8 • 0-321-51358-4

PROGRAMMING LANGUAGES



Concepts of Programming Languages, 8e
Sebesta © 2008
978-0-321-49362-0 • 0-321-49362-1

WEB PROGRAMMING AND DESIGN



Programming the World Wide Web, 4e
Sebesta © 2008
978-0-321-48969-2 • 0-321-48969-1

ACCESS



Access 2007 Guidebook, 6e
Trigg/Dobson © 2008
978-0-321-51701-2 • 0-321-51701-6

Also Available

The Must-Have Reference for Fall



Addison-Wesley's Backpack Reference Guides are small, clear, and easy-to-use. With syntax examples, keyword descriptions, and programming tips in one handy place, they are the must-have quick guide for your students.

Quickly reference the Java guide for frequently used keywords and APIs. The C++ guide puts frequently used keywords and libraries at students' fingertips. The Java and C++ Backpack Reference Guides are offered at a discount when packaged with a new Addison-Wesley programming textbook.

Textbook-Specific Tutorial Tools



MyCodeMate, a book-specific Web resource, provides a variety of learning tools to help students learn programming concepts, prepare for tests, and earn better grades.

In MyCodeMate, students work on programming problems from the text or instructor-created homework problems, and receive guided hints with page references and English explanations of compiler errors. MyCodeMate also includes learning activities, interactive tutorials, quizzes, and labs.

For more information, visit www.mycodemate.com

