Picturing Programs
An introduction to computer programming

Stephen Bloch\textsuperscript{1}

\textsuperscript{1}Math/CS Department, Adelphi University. Supported in part by NSF grant 0618543. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect the views of the National Science Foundation (NSF).
Dedicated to my wife Deborah, with whom I would have done more fun things in the past year if I hadn’t been busy writing a book.
Contents

0 Introduction ................................................. 1
  0.1 Languages and dialects ................................. 1
  0.2 Problems, programs, and program testing .............. 2
  0.3 Using DrRacket ........................................ 3
    0.3.1 Getting DrRacket ................................. 3
    0.3.2 Starting DrRacket ................................. 3
    0.3.3 Choosing languages ............................... 4
    0.3.4 Installing libraries .............................. 4
    0.3.5 Getting help .................................... 5
  0.4 Textbook web site ..................................... 5

PART I Running and writing programs ......................... 7

1 Drawing pictures ........................................ 9
  1.1 Working with pictures ................................ 9
    1.1.1 Importing pictures into DrRacket ............... 9
    1.1.2 The Interactions and Definitions panes .......... 9
    1.1.3 Choosing libraries .............................. 10
  1.2 Manipulating pictures ................................ 10
    1.2.1 Terminology ................................... 11
    1.2.2 Combining pictures .............................. 11
    1.2.3 A Syntax Rule, Sorta ............................ 12
  1.3 Making mistakes ..................................... 13
    1.3.1 Leaving out the beginning left-parenthesis ..... 13
    1.3.2 Leaving out the ending right-parenthesis ..... 14
    1.3.3 Misspelling the operation name ................. 15
    1.3.4 Too few or too many arguments ................ 15
    1.3.5 Putting the operation and arguments in the wrong order 15
    1.3.6 Doing something different from what you meant ... 16
  1.4 Getting Help ......................................... 16
  1.5 More complex manipulations ........................... 16
  1.6 Saving Your Work: the Definitions pane ............. 18
  1.7 The Stepper .......................................... 19
  1.8 Syntax and box diagrams ............................. 20
  1.9 Review .............................................. 23
  1.10 Reference ........................................... 24
2 Variables
2.1 Defining a variable ............................................. 25
2.2 The Definitions pane ........................................... 27
2.3 What’s in a name? ............................................ 27
2.4 More syntax rules ............................................ 28
2.5 Variables and the Stepper ................................... 29
2.6 Review ..................................................... 29
2.7 Reference .................................................... 29

3 Building more interesting pictures
3.1 Other kinds of arguments ..................................... 31
  3.1.1 Strings as arguments ..................................... 31
  3.1.2 Numbers as arguments ................................... 33
3.2 More mistakes .................................................. 34
3.3 Creating simple shapes ....................................... 34
3.4 Data types and contracts ...................................... 36
  3.4.1 String literals and identifiers ............................ 36
  3.4.2 Function contracts ........................................ 38
  3.4.3 Comments ................................................ 39
  3.4.4 Comments in Practice .................................... 40
3.5 More functions on pictures .................................... 41
  3.5.1 Cutting up pictures ....................................... 41
  3.5.2 Measuring pictures ....................................... 42
  3.5.3 Placing images precisely ................................. 43
  3.5.4 Text .................................................... 44
  3.5.5 For further reading........................................ 45
  3.5.6 Playing with colors ...................................... 45
3.6 Specifying results and checking your work ................. 46
3.7 Reading and writing images .................................. 47
3.8 Expanding the syntax rules ................................... 48
3.9 Review ..................................................... 48
3.10 Reference .................................................... 49

4 Writing functions
4.1 Defining your own functions .................................. 51
4.2 What’s in a definition? ....................................... 53
  4.2.1 Terminology .............................................. 53
  4.2.2 Lines and white space .................................... 53
4.3 Parameters and arguments .................................... 55
4.4 Parameters, arguments, and the Stepper ..................... 56
4.5 Testing a Function Definition ................................. 58
  4.5.1 Testing with string descriptions .......................... 58
  4.5.2 Common beginner mistakes ............................... 60
  4.5.3 The check-expect function ............................... 61
4.6 A new syntax rule ............................................. 62
4.7 Scope and visibility .......................................... 64
4.8 An analogy from English ..................................... 65
  4.8.1 Proper nouns and literals ................................. 65
  4.8.2 Pronouns and variables .................................. 65
  4.8.3 Improper nouns and data types ......................... 66
## CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.8.4</td>
<td>Verbs and functions</td>
<td>66</td>
</tr>
<tr>
<td>4.8.5</td>
<td>Noun phrases and expressions</td>
<td>66</td>
</tr>
<tr>
<td>4.9</td>
<td>Review</td>
<td>67</td>
</tr>
<tr>
<td>4.10</td>
<td>Reference</td>
<td>68</td>
</tr>
<tr>
<td>5</td>
<td><strong>A recipe for defining functions</strong></td>
<td>69</td>
</tr>
<tr>
<td>5.1</td>
<td>Step-by-step recipes</td>
<td>69</td>
</tr>
<tr>
<td>5.2</td>
<td>A more detailed recipe</td>
<td>69</td>
</tr>
<tr>
<td>5.3</td>
<td>Function contracts and purpose statements</td>
<td>70</td>
</tr>
<tr>
<td>5.4</td>
<td>Examples (also known as Test Cases)</td>
<td>73</td>
</tr>
<tr>
<td>5.5</td>
<td>The function skeleton</td>
<td>75</td>
</tr>
<tr>
<td>5.6</td>
<td>Common beginner mistakes</td>
<td>75</td>
</tr>
<tr>
<td>5.7</td>
<td>Checking syntax</td>
<td>78</td>
</tr>
<tr>
<td>5.8</td>
<td>Exercises on writing skeletons</td>
<td>79</td>
</tr>
<tr>
<td>5.9</td>
<td>The inventory</td>
<td>79</td>
</tr>
<tr>
<td>5.10</td>
<td>Inventories with values</td>
<td>82</td>
</tr>
<tr>
<td>5.11</td>
<td>The function body</td>
<td>83</td>
</tr>
<tr>
<td>5.12</td>
<td>Testing</td>
<td>86</td>
</tr>
<tr>
<td>5.13</td>
<td>Using the function</td>
<td>86</td>
</tr>
<tr>
<td>5.14</td>
<td>Putting it all together</td>
<td>87</td>
</tr>
<tr>
<td>5.15</td>
<td>Review</td>
<td>88</td>
</tr>
<tr>
<td>5.16</td>
<td>Reference</td>
<td>89</td>
</tr>
<tr>
<td>6</td>
<td><strong>Animations in DrRacket</strong></td>
<td>91</td>
</tr>
<tr>
<td>6.1</td>
<td>Preliminaries</td>
<td>91</td>
</tr>
<tr>
<td>6.2</td>
<td>Tick handlers</td>
<td>92</td>
</tr>
<tr>
<td>6.3</td>
<td>Common beginner mistakes</td>
<td>93</td>
</tr>
<tr>
<td>6.4</td>
<td>Writing tick handlers</td>
<td>95</td>
</tr>
<tr>
<td>6.5</td>
<td>Writing draw handlers</td>
<td>97</td>
</tr>
<tr>
<td>6.6</td>
<td>Other kinds of event handlers</td>
<td>99</td>
</tr>
<tr>
<td>6.7</td>
<td>Design recipe</td>
<td>105</td>
</tr>
<tr>
<td>6.8</td>
<td>A note on syntax</td>
<td>106</td>
</tr>
<tr>
<td>6.9</td>
<td>Recording</td>
<td>107</td>
</tr>
<tr>
<td>6.10</td>
<td>Review</td>
<td>107</td>
</tr>
<tr>
<td>6.11</td>
<td>Reference</td>
<td>108</td>
</tr>
<tr>
<td>7</td>
<td><strong>Working with numbers</strong></td>
<td>109</td>
</tr>
<tr>
<td>7.1</td>
<td>Arithmetic syntax</td>
<td>109</td>
</tr>
<tr>
<td>7.2</td>
<td>Variables and numbers</td>
<td>110</td>
</tr>
<tr>
<td>7.3</td>
<td>Prefix notation</td>
<td>112</td>
</tr>
<tr>
<td>7.4</td>
<td>A recipe for converting from infix to prefix</td>
<td>113</td>
</tr>
<tr>
<td>7.5</td>
<td>Kinds of numbers</td>
<td>115</td>
</tr>
<tr>
<td>7.5.1</td>
<td>Integers</td>
<td>115</td>
</tr>
<tr>
<td>7.5.2</td>
<td>Fractions</td>
<td>115</td>
</tr>
<tr>
<td>7.5.3</td>
<td>Inexact numbers</td>
<td>116</td>
</tr>
<tr>
<td>7.6</td>
<td>Contracts for built-in arithmetic functions</td>
<td>116</td>
</tr>
<tr>
<td>7.7</td>
<td>Writing numeric functions</td>
<td>117</td>
</tr>
<tr>
<td>7.8</td>
<td>Manipulating colors in images</td>
<td>125</td>
</tr>
<tr>
<td>7.8.1</td>
<td>Images, pixels, and colors</td>
<td>125</td>
</tr>
<tr>
<td>7.8.2</td>
<td>Building images pixel by pixel</td>
<td>125</td>
</tr>
</tbody>
</table>
CONTENTS

7.8.3 Error-proofing ............................................. 127
7.8.4 Building images from other images ......................... 129
7.8.5 A sneak preview ........................................... 131
7.8.6 A problem with bit-maps .................................. 131
7.9 Randomness .................................................. 131
7.9.1 Testing random functions .................................. 132
7.9.2 Exercises on randomness .................................. 133
7.10 Review ....................................................... 134
7.11 Reference .................................................. 134

8 Animations involving numbers .................................. 137
8.1 Model and view ............................................... 137
8.2 Design recipe ................................................ 139
8.3 Animations using add1 ....................................... 141
8.4 Animations with other numeric functions ..................... 143
8.5 Randomness in animations .................................... 146
8.6 Review ....................................................... 147
8.7 Reference ................................................... 147

9 Working with strings ............................................ 149
9.1 Operations .................................................... 149
9.2 String variables and functions ................................ 150
9.3 Review ....................................................... 151
9.4 Reference ................................................... 151

10 Animations with arbitrary models ............................. 153
10.1 Model and view ............................................... 153
10.2 Design recipe ................................................ 153
10.3 Review ....................................................... 156
10.4 Reference ................................................... 156

11 Reduce, re-use, recycle ......................................... 157
11.1 Planning for modification and extension ..................... 157
11.2 Re-using variables .......................................... 157
11.3 Composing functions ........................................ 161
11.4 Designing for re-use ........................................ 163
11.5 Designing multi-function programs: a case study .......... 164
11.6 Practicalities of multi-function programs .................... 175
11.7 Re-using definitions from other files ....................... 176
11.7.1 require and provide ..................................... 177
11.7.2 provide-ing everything .................................. 178
11.8 Review ....................................................... 178
11.9 Reference ................................................... 179

PART II Definition by Choices .................................. 181

12 Defining types ................................................ 183
CONTENTS

18.5 Reference .......................................................... 254

19 Handling errors ..................................................... 255
  19.1 Error messages .................................................. 255
  19.2 Testing for errors .............................................. 256
  19.3 Writing user-proof functions ................................. 257
  19.4 Review .......................................................... 257
  19.5 Reference ........................................................ 258

PART III Definition by Parts ........................................ 259

20 Using Structures ................................................... 261
  20.1 The posn data type ............................................ 261
  20.2 Definition by parts ............................................ 263
  20.3 Design recipe .................................................. 263
  20.4 Writing functions on posns ................................... 264
  20.5 Functions that return posns ................................... 268
  20.6 Writing animations involving posns ......................... 270
  20.7 Colors .......................................................... 275
    20.7.1 The color data type ...................................... 275
    20.7.2 Building images pixel by pixel .......................... 276
    20.7.3 Building images pixel by pixel from other images .... 276
  20.8 Review .......................................................... 277
  20.9 Reference ........................................................ 277

21 Inventing new structures .......................................... 279
  21.1 Why and how .................................................... 279
  21.2 Design recipe .................................................. 282
  21.3 Exercises on Defining Structs ............................... 284
  21.4 Writing functions on user-defined structs .................. 285
  21.5 Functions returning user-defined structs ................... 287
  21.6 Animations using user-defined structs ..................... 289
  21.7 Structs containing other structs ............................ 295
  21.8 Decisions on types, revisited ............................... 297
  21.9 Review .......................................................... 302
  21.10 Reference ........................................................ 303

PART IV Definition by Self-reference ................................ 305

22 Lists ....................................................................... 307
  22.1 Limitations of structs ......................................... 307
  22.2 What is a list? .................................................... 307
  22.3 Defining lists in Racket ....................................... 308
    22.3.1 Data definitions ............................................ 309
    22.3.2 Examples of the los data type ............................ 311
    22.3.3 Writing a function on los ................................. 314
    22.3.4 Collapsing two functions into one ....................... 316
  22.4 The way we really do lists .................................... 317
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>27 Local definitions</td>
<td>395</td>
</tr>
<tr>
<td>27.1 Using locals for efficiency</td>
<td>395</td>
</tr>
<tr>
<td>27.2 Using locals for clarity</td>
<td>398</td>
</tr>
<tr>
<td>27.3 Using locals for information-hiding</td>
<td>399</td>
</tr>
<tr>
<td>27.4 Using locals to insert parameters into functions</td>
<td>402</td>
</tr>
<tr>
<td>27.5 Review</td>
<td>405</td>
</tr>
<tr>
<td>27.6 Reference</td>
<td>405</td>
</tr>
<tr>
<td>28 Functions as objects</td>
<td>407</td>
</tr>
<tr>
<td>28.1 Adding parameters</td>
<td>407</td>
</tr>
<tr>
<td>28.2 Functions as parameters</td>
<td>408</td>
</tr>
<tr>
<td>28.3 Functions returning lists</td>
<td>413</td>
</tr>
<tr>
<td>28.4 Choosing a winner</td>
<td>415</td>
</tr>
<tr>
<td>28.5 Accumulating over a list</td>
<td>416</td>
</tr>
<tr>
<td>28.6 Anonymous functions</td>
<td>417</td>
</tr>
<tr>
<td>28.7 Functions in variables</td>
<td>418</td>
</tr>
<tr>
<td>28.8 Functions returning functions</td>
<td>419</td>
</tr>
<tr>
<td>28.9 Sequences and series</td>
<td>422</td>
</tr>
<tr>
<td>28.10 Review</td>
<td>425</td>
</tr>
<tr>
<td>28.11 Reference</td>
<td>425</td>
</tr>
<tr>
<td>29 Input, output, and sequence</td>
<td>427</td>
</tr>
<tr>
<td>29.1 The symbol data type</td>
<td>428</td>
</tr>
<tr>
<td>29.2 Console output</td>
<td>429</td>
</tr>
<tr>
<td>29.3 Sequential programming</td>
<td>432</td>
</tr>
<tr>
<td>29.4 Console input</td>
<td>436</td>
</tr>
<tr>
<td>29.4.1 The read function</td>
<td>436</td>
</tr>
<tr>
<td>29.4.2 Testing with input</td>
<td>436</td>
</tr>
<tr>
<td>29.4.3 Exercises</td>
<td>438</td>
</tr>
<tr>
<td>29.5 Input streams</td>
<td>438</td>
</tr>
<tr>
<td>29.6 Files</td>
<td>442</td>
</tr>
<tr>
<td>29.7 The World Wide Web</td>
<td>443</td>
</tr>
<tr>
<td>29.8 Review</td>
<td>443</td>
</tr>
<tr>
<td>29.9 Reference</td>
<td>444</td>
</tr>
<tr>
<td>30 Mutation</td>
<td>445</td>
</tr>
<tr>
<td>30.1 Remembering changes</td>
<td>445</td>
</tr>
<tr>
<td>30.2 Mutating variable values</td>
<td>446</td>
</tr>
<tr>
<td>30.3 Memoization</td>
<td>449</td>
</tr>
<tr>
<td>30.4 Static and dynamic scope</td>
<td>452</td>
</tr>
<tr>
<td>30.5 Encapsulating state</td>
<td>453</td>
</tr>
<tr>
<td>30.6 Mutating structures</td>
<td>456</td>
</tr>
<tr>
<td>30.7 Review</td>
<td>459</td>
</tr>
<tr>
<td>30.8 Reference</td>
<td>459</td>
</tr>
<tr>
<td>31 Next Steps</td>
<td>461</td>
</tr>
</tbody>
</table>
Chapter 0

Introduction

0.1 Languages and dialects

Computers don’t naturally understand human languages such as English. Instead, we invent artificial languages to communicate with them. These artificial languages are typically much simpler than any human language, so it’s easier to learn them than for, say, an English speaker to learn Chinese. But it’s still hard work. As with any language, you’ll need to learn the spelling, punctuation, grammar, vocabulary, and idioms\(^1\) of the new language.

Among the artificial languages people use to communicate with computers (and computers use to communicate with one another) are HTML, XML, SQL, Javascript, Java, C++, Python, Scheme, PHP, Ruby, sh, awk, Racket, and hundreds more. Some of these languages are called *programming* languages because they are used mostly to write programs — that is, to teach a computer new tricks by combining the tricks it already knows.

This is a book about how to write computer programs. Pretty much every such book chooses one particular programming language. I’ve chosen to use a new language called Racket (which is based on a 30-year-old language named Scheme, which is based on a 50-year-old language named Lisp, which is based on an 80-year-old mathematical theory named lambda-calculus. . . ). But it’s not a Racket book; the Racket language is not the *goal*, but only a *means* towards the goal of knowing how to program.

Here’s why: throughout the history of computers, the dominant languages have changed every five to ten years. (Fortran, Cobol, BASIC, PL/I, Pascal, C++, Java, Python, . . . ) No matter which of these languages you learn, it will probably become obsolete in a few years. If you plan to get a job as a computer programmer next month, then by all means study the language(s) used in industry right now. But if you plan to get a job programming several years from now, you’ll have to learn a new language then anyway. The current school term will be better spent learning more long-lasting skills, habits, and principles: how to structure a program, what steps to take in developing a program, how to manage your time so you finish the program on time, *etc*. And if you

\(^1\)“Idiom” means the way a particular language is *typically* used by those who use it heavily. For example, if I said “This book is more good than any other programming book.” you would know what I meant, but you would also know I wasn’t a native English-speaker; a native English speaker would say “This book is *better* than any other programming book.” Every language, including computer programming languages, has its own idioms.
don’t plan to be a professional programmer at all, then you don’t need to learn this year’s “hot” language at all; you need to learn the important principles of programming, in whatever language will “get out of the way” and let you learn them.

In fact, we won’t even be using very much of the Racket language. The software we use, a program named DrRacket, provides several dialects of Racket, intended for different kinds of users. (By way of analogy, the United States and England use different dialects of English: most of the words are the same, but sometimes the same words mean completely different things in different countries. Furthermore, an elementary school student, an economist, and a sculptor may all use English, but they use it differently, and they may use the same word to mean different things.) The “Beginning Student” dialect, in which we’ll start, doesn’t allow you to do some things that are technically legal Racket, but which tend to confuse beginning programmers. If you really need to do these things, you can switch to a larger dialect with a few mouse-clicks.

In this book, there will be no “black magic”: nothing that you need to memorize on faith that you’ll eventually understand it. On the first day, you will see just enough language to do what you need on the first day. By the end of the term, you will see just enough language to do what you need in one term. Any language feature that doesn’t help to teach an important programming principle doesn’t belong in this book. Most programming languages, frankly, don’t allow me to do that: in C++ or Java, for example, the very first program you write requires knowing dozens of language features that won’t be fully explained for months. Racket allows me to postpone irrelevant language features, and concentrate on the important stuff.

Racket is also a much simpler, more consistent language than C++, Java, or Python, so it takes much less time to learn. This, too, allows you to concentrate on the important stuff, which is how to write a program.

Again, Racket is only a means to an end. If six months after taking this course you don’t remember any Racket at all but can follow the steps of solving a problem, as explained in this book, the course has been a success.

0.2 Problems, programs, and program testing

A computer program that answered only one specific question, like

\[
\text{add 3 and 4}
\]

wouldn’t be very useful. Most computer programs are written to be general, in that a single program can answer any one of many similar questions:

- add 3 and 4
- add 19 and -5
- add 102379 and -897250987

etc. Somebody writes the program to add two numbers once and for all; later on, when you run the program, you provide specific values like 3 and 4, and the program produces the right answer for those values. Run it again with different values, and it should produce the right answer for the new values instead.

To take a more realistic example, a word processor program is written to handle whatever words you choose to write. When you run the program, you provide specific words — a grocery list, a letter to your grandmother, the next best-selling novel — and
0.3 Using DrRacket

This section doesn’t cover any “big ideas”, only the details of how to get DrRacket to work the way you need it to in this book. If you’ve already got DrRacket and the picturing-programs library installed, you can skip this section.

0.3.1 Getting DrRacket

If you haven’t got the DrRacket program installed on your computer already (it usually has a red-white-and-blue icon, a circle with the Greek letter λ on it), you’ll need to get it. You can download it for free, for Windows, Macintosh, and Linux, from http://www.racket-lang.org. This textbook assumes you have a version of DrRacket numbered 5.0.1 or higher.

0.3.2 Starting DrRacket

Once you’ve got DrRacket downloaded and installed, you should be able to run it by double-clicking the icon. It should open a window with a few buttons across the top, and two large panes. In the lower pane (the “Interactions Pane”, where we’ll be working at first) should be a welcome message like

Welcome to DrRacket, version 5.1.
Language: Beginning Student.
>

(Your version number and language may be different.)
The “> ” prompt is where you’ll type things.
0.3.3 Choosing languages

DrRacket provides a number of different computer languages, most of which are dialects of Racket. For now, we want to be working in the “Beginning Student” language. If the welcome message says something other than “Beginning Student” (or perhaps “Beginning Student custom”) after the word “Language:”, do the following:

1. Pull down the “Language” menu and select “Choose Language...”
2. Find the group of languages named “How to Design Programs”
3. If necessary, click the triangle to the left of “How to Design Programs” to show its sub-headings
4. Select “Beginning Student”
5. Click “OK”
6. Quit DrRacket and start it again, and it should now say “Language: Beginning Student”.

(You don’t really have to quit and re-start DrRacket; you can get the same effect by clicking the “Run” button. However, quitting and restarting demonstrates that DrRacket remembers your choice of language from one time you use it to the next.)

0.3.4 Installing libraries

A “library”, or “teachpack”, is a collection of optional tools that can be added into DrRacket. For most of this book, we’ll need one named picturing-programs.

Skip this section if you have DrRacket version 5.1 or later: picturing-programs is already installed on your computer.

If you don’t already have the picturing-programs library, here’s how to get it. You’ll only have to do this once on any given computer.

1. Make sure your computer is connected to the Internet.
2. Start DrRacket.
3. From the “Language” menu, “Choose Language”, then select “Use the language declared in the source”.
4. Click “Run”.
5. At the “>” prompt in the bottom half of the screen, type
   \(\text{(require (planet sbloch/picturing-programs:2))}\)
   exactly like that, with the parentheses and the slash and all. It may take a few seconds to a few minutes (most of which is updating the help system to include information on this library), but eventually you should see the message “Wrote file “picturing-programs.ss” to installed-teachpacks directory.”
6. From the “Language” menu, “Choose Language”, then click on to “How to Design Programs”, then select “Beginning Student”. Hit “Run” again.
0.3.5 Getting help

If you want to look up reference information about this library (or anything else in the language),

1. from the “Help” menu, choose “Help Desk”.

2. find the search box at the top of the screen and type the name of a library or function you want to learn about. Then hit ENTER.

3. If the name is found, you’ll get a list of places it appeared in the documentation. Click one of them (probably one that says it’s from the “picturing-programs” library).

4. Documentation for that library or function should appear on the screen.

0.4 Textbook web site

In order to keep the cost of this book down, we’ve put all the illustrations in black and white. You can find colored versions of many of them, as well as corrections, updates, additions, image files, and downloadable versions of worked exercises (so you don’t have to type them in by hand), etc. at http://www.picturingprograms.com.
Index

<, 197
<=, 197
>, 197
>=, 197
*, 134
+, 134
-, 134
/, 134
=, 197
Lukasiewicz, Jan, 112
above, 11, 12, 24
above/align, 32, 49
abs, 122, 129, 134
accumulative recursion, 461
ActionListener, 107
add-curve, 45, 50
add-line, 50
add1, 110, 134
add1?, 359
aliasing, 457
all-defined-out, 178, 179
all-from-out, 179
ambiguity, 113
ambiguous expressions, 112, 114
and, 194, 197
append, 345, 353
arguments, 11, 23, 55
auxiliary function, 166, 179
awk, 1

base case, 325
BASIC, 1
begin, 432, 444
Beginning Student language, 2, 4
Beginning Student with List Abbreviations, 347
beside, 12, 24
beside/align, 31, 49
big-bang, 91, 100, 105, 108, 138, 140, 154, 155, 204
binary trees, 461
bit-maps, 131
bitmap, 49, 442
black magic, 2
blank lines, 53
boolean, 185
Boolean operators, 192
boolean=?, 192, 197
boolean?, 191, 197
borderline, 187–190, 211, 214
box diagrams, 20, 28, 48, 63
build-image, 131, 135, 276, 278
build-image/extra, 277, 278
build-list, 415, 426
build3-image, 125, 135
bullseye, 72, 74, 79, 82, 86
C++, 1, 2
callback, 107
callbacks, 96
caption-below, 87
char, 338
char-alphabetic?, 339, 341
char-downcase, 350, 353
char-lower-case?, 350, 353
char-upper-case?, 350, 353
char=?, 338, 341
char?, 338, 341
characters, 338
Classes, 462
Check Syntax, 78
check-error, 256, 258
check-expect, 61, 68
check-member-of, 132, 135
check-range, 132, 135
check-with, 137, 138, 154, 204
checkerboard2, 72, 74, 79, 80, 84
circle, 35, 49
circle-in-square, 87
classes, 462
client-server programming, 461
Cobol, 1

color, 275
color-blue, 275, 278
color-green, 275, 278
color-red, 275, 278
color=?, 278
color?, 275, 278
comments, 39, 49
components, 125
compose, 421
cond, 208, 229
conjugation, 42
cons, 318, 341
cons?, 318, 341
constructor, 279
constructors, 302
continuations, 461
contract, 88
contracts, 38, 48, 70, 462
Conway, John, 404
copies-beside, 71, 74, 79, 80, 84
Copying images, 9
counterchange, 55, 71, 73, 75, 80, 82, 83
CPU time, 391
crop, 42, 45, 50
crop-bottom, 41, 49
crop-left, 42, 49
crop-right, 42, 49
crop-top, 42, 49
data types, 36, 48, 66
define, 29, 68
define-struct, 280, 303
define-struct/contract, 462
define/contract, 462
Definitions Pane, 9, 18, 27
dialects, 2
diamond, 87
discriminator, 279
discriminators, 192, 229, 302
display, 429, 444
dot-grid, 72, 75, 79, 82, 84
draw handlers, 91, 97, 100, 138–140, 154, 155, 204
DrRacket, 2, 3
dynamic programming, 452
dynamic scope, 453
elipse, 35, 49
else, 210, 229
empty, 318, 341
empty-scene, 45, 50
empty?, 318, 341
equal?, 229, 329
error, 256, 258
error messages, 13, 86
event handler, 107
event handlers, 91, 92
event-driven programming, 96
examples, 72, 88
executables, 462
expressions, 11, 66
field, 279
fields, 302
filter, 426
finite-state automaton, 332
finite-state machine, 332
first, 318, 341
flip-horizontal, 10, 24
flip-vertical, 10, 24
foldl, 417, 426
foldr, 417, 426
format, 257, 258
Fortran, 1
four-square, 54
frame, 45, 50
function body, 53, 79, 83, 88
function contract, 88
function definitions, 51, 62
function header, 53, 79
function inventories, 79
function skeleton, 88
function skeletons, 75
function template, 232, 235, 238
functions, 11, 23, 66
Game of Life, 404
garbage collection, 391, 392
GC time, 391
generative recursion, 461
get-pixel-color, 277, 278, 403
getter, 279
getters, 302
graphics, 462
GUI, 462
Help Desk, 5
helper function, 124, 133, 166, 179
higher-order functions, 404, 405, 408
hope, 13
HTML, 1

identifiers, 15, 27–29
identity, 414, 426
idioms, 1
if, 218, 229
image-height, 42, 49
image-width, 42, 49
image=?, 197
image?, 137, 191, 197
Importing images, 9
improper nouns, 66
indentation, 14, 54
infix operators, 112
information-hiding, 399, 401, 405
Inserting images, 9
instance, 279
instance variable, 279
instance variables, 302
instances, 302
Interactions Pane, 3, 9, 27
inventories, 79, 88
inventories with values, 82, 88, 118, 120, 122, 158, 173, 268–270, 272, 277, 288, 290, 291, 293, 294, 303, 315, 322, 327, 334, 357, 365, 369
inventory, 235
isosceles-triangle, 50
jaggies, 131
Java, 1, 2, 461
Javascript, 1

key handler, 250
key handlers, 100, 105, 138, 140, 154, 155, 204
key=?, 254
KeyListener, 107

lambda, 425
lambda-calculus, 1
lexical scope, 453
libraries, 4, 10
life, 404
line, 50
Lisp, 1
List Abbreviations, 347
list function, 347
list->string, 353

list-ref, 332, 341, 378
literals, 11, 23, 26, 28, 36, 48, 65
local, 405
lollipop, 72, 75, 79, 82, 86
macros, 462
make-color, 45, 49, 275, 278
make-posn, 261, 277
map, 426
map-image, 131, 276, 278
map-image/extra, 277, 278
map3-image, 135
max, 122, 134
memoization, 450
method dispatch, 455
min, 122, 127, 134
mirror-image, 52
misspellings, 15
model, 91, 97, 105, 107
model checking, 138, 154
model type checkers, 204
model-view framework, 139, 153, 156
models, 140
modules, 462
mouse handlers, 100, 101, 105, 138, 140, 154, 155, 204
MouseListener, 107

n-ary trees, 461
name->color, 46
name-¿color, 49
natural?, 355
network programming, 461
newline, 444
not, 194, 197
noun phrases, 66
number->string, 152
number?, 137, 191, 197
numbers, 33, 48

object-oriented programming, 461, 462
on-draw, 92, 100, 108, 138, 154, 204
on-key, 100, 105, 106, 108, 138, 140, 154, 155, 204
on-mouse, 100, 105, 106, 108, 138, 140, 154, 155, 204
on-release, 253, 254
on-tick, 92, 100, 105, 106, 108, 138, 140, 154, 155, 204
operations, 11, 23
or, 194, 197
order of operations, 112, 113
outventories, 80
outventory, 235
overlay, 12, 24, 92
overlay/align, 33, 49
overlay/xy, 50

paint method, 107
parameter names, 77, 79
parameterizing, 407
parameters, 55, 63
parenthesis matching, 13
PEMDAS, 112, 113
PHP, 1
pi, 135
pinwheel, 71, 74, 79, 80, 84
pixel-maps, 131
pixels, 125
PL/I, 1
place-image, 43, 49, 92
place-image/align, 49
Polish notation, 112
polymorphic functions, 229
positive?, 359, 373
posn-x, 261, 277
posn-y, 261, 278
posn?, 261, 278
prayer, 13
precedence, 112
predicates, 192, 196
prefix notation, 112
prefix operators, 112
prefix-out, 179
printf, 430, 444
procedures, 11
programming languages, 1
pronouns, 65
proper nouns, 65
provide, 177, 179
purpose statement, 88
purpose statements, 70
Python, 1, 2

quadratic formula, 111
quote, 436
quotient, 134
random, 134
read, 436, 444
read-line, 436, 438, 444
Real time, 391
real->int, 127
record?, 107, 108
rectangle, 34, 49, 92
recursion, 324
recursive, 324
regular-polygon, 50
remainder, 134
rename-out, 179
require, 4, 10, 24, 177, 179
reserved word, 229
rest, 318, 341
reverse, 345, 353
RGB, 125
RGBA, 125
rhombus, 50
right-triangle, 50
rotate, 33, 49
rotate-180, 11, 24
rotate-ccw, 11, 24
rotate-cw, 11, 24
Ruby, 1

save-image, 47, 49, 442
scale, 33, 49
scale/xy, 33, 45, 50
scene+curve, 45, 50
scene+line, 45, 50
Scheme, 1
scope, 63, 65
selector, 279
selectors, 302
set, 459
setter functions, 459
setters, 456
sh, 1
short-circuit evaluation, 196
show-it, 91, 92, 97, 100, 108, 138, 154, 155, 204
sin, 134
skeleton, 88
skeletons, 75
special form, 196, 209, 258
special forms, 93
SQL, 1
sqrt, 134
square, 45, 50

Racket, 1, 2
INDEX

star, 49
star-polygon, 50
static scope, 453
Stepper, 19, 29, 56
stop handlers, 204
stop-when, 204, 205
stop-with, 205
string<?, 197
string<=?, 197
string>??, 197
string>=?, 197
string->list, 338, 341
string->number, 152
string-append, 151
string-ci<?, 197
string-ci<=?, 197
string-ci>??, 197
string-ci>=?, 197
string-ci=?, 197
string-length, 151
string=?, 197
string?, 191, 197
strings, 32, 36, 48
struct-out, 179
structural recursion, 461
Structure and Interpretation of Computer
Programs, 419
sub-expressions, 17
sub-range, 190, 211, 214
sub1, 110, 134, 373
sub1?, 359
substring, 152
surround, 55
Syntax rules, 28
syntax rules, 20
teachpacks, 4
template, 238
test cases, 3, 72, 88
test-driven development, 73
testing, 58, 86, 88
text, 44, 49
text-box, 87
text/font, 45, 50
thunks, 432
tick handlers, 92, 99, 100, 105, 138, 140,
154, 155, 204
time, 393
triangle, 35, 49
two-eyes, 87
type predicates, 192, 197, 226, 229
underlay, 45, 50
underlay/align, 45, 50
underlay/xy, 45, 50
universe, 91, 461
value of an expression, 9, 11
variable definitions, 25, 28
variables, 25, 26, 29, 36, 65
verbs, 66
vert-mirror-image, 54
visibility, 63
void, 458
web server, 461
white space, 53
whole numbers, 355
with-input-from-file, 444
with-input-from-string, 436, 444
with-input-from-url, 444
with-io-strings, 438, 444
with-output-to-file, 444
with-output-to-string, 431, 444
world, 91
write, 429, 444
XML, 1
zero?, 359, 373
Bibliography


