

Index of symbols and terms

Symbols are ordered by their most important occurrence; terms are ordered alphabetically. In both cases:

- *Bold page numbers indicate principal occurrences.*
- *A down arrow $\downarrow$ indicates “in footnote”.*
- *A small circle $\circ$ indicates “in figure”.*
- *The abbreviation “qv” is a cross-reference to the entry’s content: e.g. “anything ... set x to” cross-refers to “set x to”.*
- *A page number in brackets $[-]$ indicates a bibliographic reference.*

Index of symbols

- $[low:high]$, index-range in sequence is inclusive/exclusive, **5**
- `# ---` comment giving the program’s requirements, **5**
- $\sum$ means “the sum of”, **5**
- $\wedge$ is Boolean conjunction, **11**
- $\%$ is remainder, **17**
- $//$ is (integer) quotient, **17**
- ∞ is “infinity”, **309**
- $+$ is sequence concatenation, **20, 82**
- $!$ is factorial, **312**
- $d/-/y$ is Day d in Year y , **23**
- $**$ is “to the power of”, **41**
- $[x]*r$ is a sequence of length r containing only x ’s, **44**
- $=$, $==$ are both “is equal to”
 - $==$ is equality in **Python**, **3, 275** $\downarrow$
 - $=$ is mathematical equality, **275** $\downarrow$
- $\Rightarrow, \Rightarrow$ are, informally, both “implies”, **9, 274**
 - $\Rightarrow$ is more precisely a propositional connective, **274**

- $\Rightarrow$ is more precisely a relation between formulae, **274**
- $\rightarrow$ is written $\Rightarrow$ in this text, **274**
 - neither has an equivalent symbol in **Python**, **274**
- $\Leftrightarrow, \Leftarrow$ are $\Rightarrow, \Rightarrow$ reversed, **274**
- $\equiv, \Leftrightarrow$ are, informally, both “if and only if”, **274**
 - $\Leftrightarrow$ is more precisely a propositional connective, **274**
 - $\equiv$ is more precisely a relation between formulae, **274**
 - $\leftrightarrow$ is written $\Leftrightarrow$ in this text, **274**
- $\{\}$ is the empty set, **81**
- $\cup$ gives the union of two sets, **81**
- $\in$ indicates membership of a set, **81**
- $-$, minus, gives the difference of two sets, **82**
- $\notin$ indicates “not-membership” of a set, **84**
- $[x]*r$ is a sequence of length r containing only x ’s, **106** $\circ$
- $\sum$ means “the sum of”, **231**
- $+$ is the sum of two bags, **118**
- $-$, minus, gives the difference of two bags, **118**
- $\in$ indicates membership of a bag, **118**
- $\perp$ is “bottom”
 - meaning “no thread”, **159**
- $\perp$ is “bottom”, meaning “no thread”, **137**
- $x: [pre, post]$, a specification statement, **125, 251**
- $\$bb$ is a “lock variable” for bb , **137**
- $\prod$ means “the product of”, **232**
- α -conversion, *see* alpha conversion
- $post[x\backslash expr]$, substitution, **242**
- $???$ indicates nondeterministic (demonic) choice, **254**
- $\wedge$ is Boolean conjunction, corresponding to “and” between Booleans in **Python**, **268, 282**
- $\vee$ is Boolean disjunction, corresponding to

“or” between Booleans in Python, **272, 275**
 $\Leftrightarrow$ means if-and-only-if (iff), corresponding to “==” in Python if between Booleans, **272**
 $\neg$ is negation, corresponding to “not” applied to a Boolean in Python, **272**
“iff” is short for “if and only if”, **288**
 $\therefore$ “therefore” is archaic; we use $\Rightarrow$, **289**
 $\because$ “because” is archaic; we use $\Leftarrow$, **289**
 $\models$ is semantic entailment (but here we write it as $\Rightarrow$), **289**
 $\vdash$ is logical entailment (but here we write it $\Rightarrow$ as well), **289**
 $\forall$, for all, *see* quantification
 $\exists$, there exists, *see* quantification

Index of terms

$A[\text{low}:\text{high}]$ is a subsegment, 31
 $A[:\text{high}]$ is short for $A[0:\text{high}]$, 270, 38
 $A[\text{low}:]$ is short for $A[\text{low}:\text{len}(A)]$, 38
“... < x” means “every element of the (sub)segment ... is < x”, **38**
index-ranges are
 inclusive/exclusive, **5, 31, 39, 49**
 $A[0]$ is the head of A , 162
 $A[1:]$ is the tail of A , 162
 $A[n:n]$ is the empty sequence, 52
J.-R. Abrial, [219↓], [222], [222, 223], [225]
absorption, 282
abstract data-type, 83, 86–87
 See also concrete data-type.
abstract/concrete hybrid, *qv*
access modifier, *see* object orientation
accreditation
 of engineers, 73
 of programmers, 73
`acquire()` in Python, 135↓
aeroplane, *see* building an aeroplane
algorithms of reasoning about programs, 230
 See also Al-Khwārizmī.
algorithm
 can be incorrectly coded, xii
 ... or can be wrong in principle, xii
 is the “essence” of a program, xi
 See also Al-Khwārizmī.
Al-Khwārizmī, 229↓
(for) all $\forall$, *see* quantification
“all the way down”, *see* mathematics
alpha conversion, 242↓, **277, 286, 288, 291**

 example, 277
 sometimes necessary in substitutions, 242↓
analyst vs. customer, 118
“and” is Boolean conjunction, *qv*
animals, people behaving like, 288↓
antecedent
 contravariance
 of implication, 333
 of quantification, 287
 is false means the overall implication is true, 156
 left-hand side of implication, **9, 272**
 See also consequent, implication.
antisymmetric, **59**
anything, set x to, *qv*
arithmetic, of Booleans, *qv*
arity of operator, function or predicate, **270**
“arrays” in Python, 105
`assert` statement, 106, **125, 129**
 at runtime, 219
 encourages colleagues to respect your preconditions, 104, 106
 how to check, 104, **249**
 if it fails, allows
 unpredictable behaviour, 115
 in Python, 104↓
 used to validate data refinement, 253, 261, 331
 vertical stacking indicates *conjunction*, not implication, 198↓
 See also Dafny.
`assume` statement, *see* Dafny
assertions {...} or # ... , **9, 8–10, 16, 118**
 are often checked from the end (postcondition) towards the beginning (precondition) of a program fragment, 19, 67, 68, 70, 87, 201, 265
 braces vs. comment, 18, 129
 compile time vs. run time, 104↓, 198
 effective even if written informally as comments (e.g. in English), 21, 281
 horizontal vs. vertical format, 10, 12↓
 “housekeeping” assertions, 20, 303
 how to check, **250**
 intermediate, 19, 20, 22, 66, 201, 310
 pre-, postconditions and invariants are special cases of, 16
 several in a row vertically, 10, 12↓, 14, 34, 67, 68, 305
 trivial, 244
 used as “scaffolding”, 30

assignment statement	for variants, 56, 66, 70
checked by textual substitution, 8,	used to transform abstract data-type
12, 17, 19, 21, 22, 35, 65–67,	into concrete, 89
144, 241 , 242, 265, 285	average of sequence elements, 233
multiple, i.e. to several variables, 11,	await statement, 139, 153–156
67↓, 94, 299	comparison with if , 154↓
in <i>C</i> is different, 67↓	how to check, 154 , 248
associativity, 282	implemented with lock, 156
eliminates (some) parentheses, 95	<i>See also</i> concurrency.
of binary operators, 231	axiom, 202↓, 289
of conjunction $\wedge$, 282	
of disjunction $\vee$, 274, 282	<i>The B method</i> , 219↓
of multiplication, 95	<i>See also</i> Abrial .
assume statement, 129	B, node colour is black, 179
how to check, 250	B-counting, 182
used to validate data refinement, 253,	R.-J.R. Back , [222], [223], [225]
261, 331	Roland C. Backhouse , [223], [225], [306↓]
with or without frame, 126	Bad() , indicates <i>Bad</i> subsegment, 25 , 25–
<i>ATM</i> , <i>see</i> automated teller machine	26
atomic action, 135 , 134–136, 147, 165	bag, <i>see</i> multiset
atomicity is an <i>assumption</i> , 136	basic data-types, <i>qv</i>
expressions and assignments, 136	baton passing
that both reads <i>and</i> writes, 138, 156	of correctness, 8, 14
written on a single line, 135, 136, 148,	of true assertions, 10
150, 165, 180, 193, 327	<i>See also</i> program checking.
<i>See also</i> concurrency.	$B \mapsto B$, black node pointing only to black
atomicity	targets, 180
enforced within await -body, 154↓	“begging the question”, 149↓
implicit if two or more statements are	<i>See also</i> circular reasoning.
on the same line, 249	Mordechai Ben-Ari , [177↓], [182↓], [223↓],
<i>see also</i> atomic action	[225]
of expression evaluation, 249	J.L. Bentley , [199↓], [225]
removing atomicity, 187–190	binary of program, checking <i>it</i> for correct-
attribute, <i>see</i> object orientation	ness, 208
auto-marking, <i>see</i> testing	binary search, 38–40, 49, 50, 69, 80, 101,
automated teller machine (<i>ATM</i>), 134, 147	111, 115, 199
in Imperial currency, 57	90% of attempts incorrect, 199↓
invariant for, 135	interpreting its postcondition, 49
many active concurrently, 134	overflow bug, 199↓, 203, 209, 210
<i>See also</i> concurrency.	-trees, 102
auxiliary (ghost) variables, 88 , 88–92, 96–	using specification statement in, 251
97, 155◦, 246, 253, 264	works only for sorted sequences, 38,
adding, 89	49, 50, 80, 82, 124, 201◦, 201
do not affect running code, 156, 327	Dines Björner , [222]
just “go along for the ride”, 89	black nodes, <i>see</i> GARBAGE COLLECTION
label-like in concurrency, 166	blame
<i>see also</i> Peterson’s mutual-	finding out who is responsible for an
exclusion algorithm,	error, 106
garbage collection	<i>See also</i> debugging the programming
removing, 90, 96	process.
removing them, 327	blank slide, 268↓
to check concurrency, 154, 166, 189,	block structure, indicated by indentation
327	in Python , 299
use in postcondition, 66	body, of a quantification, 276
for variants, 251	Boogie intermediate language, 198↓
used in postcondition	Boolean

- arithmetic, 91, 263–275
- conjunction “and” or $\wedge$, *qv*
- disjunction “or” or $\vee$, *qv*
- equivalence = or $\equiv$, *qv*
- expression, 3, *see also* condition
- honorary, 163, 179↓
- implication $\Rightarrow$ or $\Rightarrow$, *qv*
- negation “not” or $\neg$, *qv*
- bound variable, 276, **277**, 277, 286
See also free variable.
- boxes Ex. ? refer to exercises relevant at
 that point, xiv
- break** to exit loop, 22, 72, 246, 301, 302
See also **continue**, loop, program components.
- K. **Broda**, [223]
- bubble sort, 71
 is inefficient, 71
See also PROGRAM EXAMPLES, EXERCISES.
- bugs
 found using **assert** statements, 249, 331
 in specification of program, 119
 introduced by carelessness, 188
 never-ending **while**-loop, 15
 their main source, 202
See also date calculation.
- building a garden shed, an office block, an
 aeroplane, a motorcycle, 73
- $B \rightarrow W$, black node points to white target,
179
- C*, programming language, 3, 4↓, 25↓, 117↓,
 172, 174, 209, 213↓, 244↓
 has different multiple assignments to
 Python, 67↓, 94, 97↓, 98, 99
 has no built-in sets, 81↓
 pointers, 172
 Python has built-in sets, 81↓
- calculator, mean, *qv*
- CAS*, *see* compare and swap
- cascading invariants, 30, 45–48, 94
See also invariants.
- case analysis in **if**-statements:
 reason about each **if**-case separately,
 39, 44
- caterpillar analogy, for *Good* subsegments,
 27–28
- ceil-ing function, 50, 238
 rounds up, 50
- checking
 all possible surrounding programs, 126
 components in isolation is easier, 68
 conditionals (**if**-statements), *qv*
See also PROGRAM EXERCISES.
- effort might double with each new
 variable added, 75
- while**-loop bodies, 14
See also invariants, loop, PROGRAM
 EXAMPLES.
- circular reasoning, 150
 not allowed in general, 149
 only apparent, 149–151, 324
- circular structure, made from pointers, 174,
 175o, 191
- class definition
 data-type invariants, 104
 global constants, 113
 global variables (references to), 112
 initialisation, 103
 local variables, 86
 of encapsulation, 102
 parameter, 103
 precondition, 103
 can refer to global variables, 103↓
 of procedure, 104
See also object orientation.
- coding
 reducing mistakes in, xiii
 what coding really is, xi
- coffee cups, disappearing, 172, 173
- Edward **Cohen**, [223], [225]
- Collector, 174, **175**
See also garbage collection.
- comments, xiii
 sometimes pointless, 5
 what comments are for, 16
 “What does this do?” vs. “What’s
 true here?”, 29, 75, 147
- common sense is built on sound engineer-
 ing principles, 123
- commutativity, **282**
 of conjunction $\wedge$, 282
 of disjunction $\vee$, 274, 282
 of quantifiers, 287
- compare and swap (*CAS*), **138**, 153, 156,
 161, 324
- complement, 283
- completeness of logic, **289**
- concatenation + of sequences, *qv*
- concrete data-type, 83, 87–91
See also abstract data-type.
- concurrency
 atomic action, for compare and swap,
 138
 atomicity, 135
 critical section, **139**, 138–143, 153–
 159, **161**
 body indicated by indentation, 165
 implemented with lock, 139
 implemented with queue, 162

- thread should not remain indefinitely within it, 163, 167
 - deadlock, 156–158, 161–167
 - fair scheduling, 158, 324
 - global variables, 161
 - “in the large” vs. “in the small”, 171
 - in the world generally, 134
 - interleaving, 134
 - livelock, 156–158, 161
 - liveness, 156–158
 - locks, 135, 137–138
 - message passing, 224
 - mutual exclusion, 138–139, **153**, 153–156, **161**
 - implemented with queue, 162
 - of *ATM*’s, 134
 - process algebra, 224
 - purpose and examples, 133–136
 - safety, 156–158, 166, 168
 - sharing variables, 133
 - simplest example of, 133
 - starvation, 156–158, 161–324
 - see also* Peterson’s mutual-exclusion algorithm
 - thread, **133**, 135, 153
 - transactions, 224
 - truly parallel, 151↓
 - within an operating system, 171
 - `yield()`, 324
- condition
- is Boolean-valued expression, 3
 - used for checking programs, 3
 - written using *Python* syntax, 3
 - See also* assertions, precondition, post-condition.
- conditional expression in *Python*, 128↓
- conditional strict majority *csm*, *see* majority voting
- conditionals (*if*-statements)
- default **else** is **skip**, 69
 - how to check an *if*-statement
 - examples, 70, 70
 - with **elif**, 70
 - with **else**, 70, **245**
 - without **else**, **244**
- conjunct (propositional calculus), 11↓, **34**
- splitting two of them, 33
 - expression next to “**and**” or **^**, **272**
 - splitting more than two conjuncts, 34
- conjunction, written “**and**” or “**^**”, 11↓, 34, 272, 282
- conjunctive normal form, **283**
- Cons*, 61↓
- consequent
- covariance of implication, 333
 - right-hand side of implication, **9**, 272
- See also* antecedent, implication.
- const** declaration, 113
- constant symbols in logic
- if introduced here are written in sans-serif, 270
 - otherwise are as usual, 270
- continue** to restart loop, 22, 301
- See also* **break**, loop, program components.
- continued fraction, *see* PROGRAM EXERCISES
- contrapositive, 191, 234, 284
- co- and contravariance
- of implication, 333
 - of quantification, 287
 - See also* antecedent, consequent.
- correctness by construction, 330
- counting black nodes, *see* GARBAGE COLLECTION
- coupling invariant, **83**, 81–92, 94, 95, 99–104, 110–113, 115–301
- examples, 162↓
 - explain how data-type operations cooperate, 82
 - for implementing mutual exclusion, 163
 - global variables (references to), 112
 - keeping only the last six digits, 99
 - linking matrices to scalars, 98
 - special cases of data-type invariants, 101
- COVID, xiv, 218, 219
- CPU*
- registers, 136
 - shared between threads, 144
- critical section, *see* concurrency
- CSIRO, Commonwealth Scientific and Research Organisation, xiv
- csm*, *see* majority voting
- Cuisenaire rods not good for calculation, 268↓
- the customer, 7
- the expectations that must be met, 205
 - vs. the analyst, 118
- Dafny programming language, 66↓, 99↓, 198–202
- assert** and **assume** statements, 278, 331
 - assume** statements do not generate executable code., 331↓
- data refinement, 81, 88, 139, 250, 253–261, 330
- data reification, 81↓
- data structures, 79–80

- triples, 79
- data-type encapsulation, 101–115
 - as a “locked box”, 98, 127
 - case studies
 - pocket calculator, 117–120
 - set* (data-type), 101–109
 - global variables (references to), 112
 - hides internal state, 98
 - local
 - precondition, 103
 - procedures, 105, 106◦, 107
 - variables, 86, 103, 107
 - more advanced techniques, 112–113
 - rules for
 - justification of, 110–112
 - summary of, 107–109
- data-type invariant, **104**, 101–106, 253
 - always induces a refinement, 112, 114
 - ensures consistency, 101
 - established by class initialisation, 105
 - implicit pre- and postcondition of every class procedure (method), 105
 - indicated by **#Dti**, **103**
 - maintained by every class procedure (method), 105
 - might be broken temporarily, 114, 301
- data-type representation
 - vs. encapsulation, 102
- data-types
 - basic, **79**
 - primitive, **79**
 - structured, **79**
 - their encapsulation, 80, 82, 84, *see also* data-type encapsulation
- Data61, *see* CSIRO
- date calculation
 - calculating “today” from day number, 4, 21–34, 49
 - is missing its variant, 15, 23, 212↓
 - leap-year bug, 4, 15
 - See also* PROGRAM EXERCISES, PROGRAM EXAMPLES.
- Jim **Davies**, [225]
- De Morgan’s laws, **283**
- dead code is never reached during execution, 37
- deadlock, *see* concurrency
- debugging
 - after the program has been written, xii, 229
 - is like “multiplying back”, 229
 - of the programming process, xiii, 4, 15, 26, 75, 184
 - See also* Kansas City walkway collapse.
- defensive programming, 106, 109, 120, 249, 258↓, 321–323, 331
- Theodorus **Dekker**, 138↓
- demonic nondeterminism, *qv*
- denarius, 57↓
- design patterns
 - for critical sections, 143
 - led to **await** statements, 139
 - led to **while**-loops, 139
 - led to data encapsulations, 139
 - with synchronised methods, 143
- determinant of empty matrix, *see* EXERCISES
- dictionary order, 58, *see also* termination, variants
- diff** file-comparison utility, 217
- E.W. **Dijkstra**, [42↓], [138↓], [139], [175↓], [177↓], [182↓], [188↓], [222], 222, [223], [223↓], [225], 268↓
 - his Golden Rule, 284↓, 284
 - introduces semaphores **P()** and **V()**, 139–143
- Rutger M. **Dijkstra**, 138↓
- DiM()**, days in month, 33, 49
- disasters and their causes, 118
- disjunct, expression next to “**or**” or **∨**, **272**
- disjunction, written “**or**” or “**∨**”, 272, 275, 282
- disjunctive normal form, **283**
- distributive rule, 282
 - from arithmetic, 276
- dividing a big programming problem into a *tree* of smaller ones, *see* program
- division, *see* quotient and remainder
- DiY()**, days in year, 21–23, 35, 49, 310
- documentation, based on assertions, 242
- dominoes falling, *see* program checking
- DRILLS
 - ceilings and floors, 238
 - Hoare-triple patterns, 236
 - implication $\Rightarrow$ and comparisons in arithmetic, 234
 - implication $\Rightarrow$ in everyday life, 234
 - intersection of disjoint sets, 234
 - invariants, 232, 233
 - invariants, variants for loops on sequences, 237
 - more complex substitution, 231
 - nondeterminism, 238
 - product of two negative numbers, 235
 - sequence expressions, 231
 - sequence special cases, 232
 - sequences and infinity, 232
 - simple substitution, 230

special cases of $\Rightarrow$, 234	delete <code>del()</code> from the Mean Calculator, 121
tricky Hoare-triples, 235, 236	determinant of empty matrix, 22
variants, 233	dictionary order vs. lexicographic, 61
(end DRILLS)	facts about sequences and $\sum$, 21
# <code>DTI</code> : , 104, 108, 258, 260	finite sets of integers, 62
might be “False”, 114	intermediate assertions, 22
See also data-type invariant.	lexicographic variant
dynamic memory allocation, see <code>malloc()</code> , <code>free()</code>	its underlying order, 60, 61
	<i>sometimes</i> can be reduced to simple variant, 61
$\sum$ Greek capital “sigma” (not an “E”) means “the sum of”, 5, 231	linear search, 91, 92
efficiency and neatness of programs, see also time complexity	loop invariants, 91
after correctness is easier, 37–38	loop unrolling, 49
“tweaking” for, xiii, 26, 37, 46, 92, 97	majority voting, 52
Susan Eisenbach, [223]	matrix, determinant if empty, 22
empty files not copied (a mistake), 207	minimum vs. minimal, 61
empty graph, 268↓	postcondition, 91
empty sequence or subsegment	precondition, 91
largest element of, 6, 7	program algebra, 52
maximum of, 22	<code>remove()</code> element from set, 114
product of it is 1, 22	requirements, 91
sum of it is 0, 22, 48, 52	rounding up or down, 50
encapsulated data-type, <i>qv</i>	sentinel and coupling invariants, 91
Herbert B. Enderton, [225], [288↓]	sequences and $\sum$, 21
entailment ($\Rightarrow$), 9, 274	sequential composition of programs, 21, 22
different from implication $\Rightarrow$, 272↓, 274	set membership, 91
distribution of, 275	specification used as program text, 128
equivalence ($\equiv$)	structural variant
chain of, 275	sometimes can be reduced to simple variant, 61
is not a propositional connective in logic (here), 274, 284↓	summing a sequence, 22
Euclid’s algorithm for <code>gcd</code> , the greatest common divisor, 56	if empty, <i>qv</i>
variant for termination, 56	termination
Euler and <i>e</i> , 270	social-distance puzzle, 63
“everyday” errors, 20	well-founded sets, 62
index out of range, 5, 20, 25, 46, 52, 168	<code>undo()</code> procedure for Mean Calculator, 120
is sometimes acceptable, 105	uninitialised variable, 51
uninitialised variable, 51	variants, 91
<i>EWD74: Concerning semaphores</i> , 140◦	well-founded sets, 61, 62
exchange sort, 72, see also PROGRAM EXERCISES	(end EXERCISES)
is inefficient, 72	existential quantification ($\exists$), <i>qv</i>
executing code “in your head”, 75, 87, 210↓, 225, 318	expectations of a customer, 205
EXERCISES	lead to requirements, 206
<code>assert</code> statement, 115	vs. specifications, 205
assertion {...} or # ...	expressions
braces vs. comment, 129	how to check for undefined, 243
assignment-statement checking, 19	“eyeballing” an implication, 20, 21
binary search (degenerate), 50	
breaking data-type invariants, 114	F, free-chain start, see GARBAGE COLLECTION
	factorial “!”, 312
	fair scheduling, see also concurrency

- false
 - unit of disjunction $\vee$, 282
 - zero of conjunction $\wedge$, 282
- “false”, *see* “true”
- Wim H.J. **Feijen**, [157↓], [223–225]
- Fibonacci numbers, **93**, 93–99
 - in logarithmic time, 96–97
- Fiji relocation project, *qv*
- finite sets of integers, 62
- Flatland, 62
- floor** function, 50, 238
 - rounds down, 50
- flowchart, 10
 - checking *between* the boxes vs. check-
ing *inside* the boxes, 76
 - examples, 14, 22, 310
 - making one, 22
 - very helpful intuitively, 75
- Robert W. **Floyd**, **10**, [76↓], [222], [225]
- for all $\forall$, *see* quantification
- for**-loop
 - assertions for, 50
 - cannot execute forever, 14
 - how to check, **247**
 - implicit increment, 319
 - vs. **while**-loop, 50, 114, 248, 300
 - check the **while** then convert to
for, 38, 48◦, 317
- forall**, 200, 201◦
- formal methods
 - a fascinating topic in its own right,
xii
 - the “Lost Art”, *qv*
 - use before the horse has bolted, 209↓
 - what formal methods is, xii
 - See also* (In-)Formal Methods.
- formula, in logic
 - atomic, **271**
 - general, **278**
 - propositional, 272
- frame, of a specification statement, 125
- free()**, **malloc()**, 172 ff
- free chain **F**, *see* GARBAGE COLLECTION
- free variable, 276, **277**, 277, 286
 - See also* bound variable.
- a fresh variable is one not already used for
something else, **139**, 254
- frozen music player, 4
 - See also* date calculation.
- function symbols in logic
 - if introduced here are written in sans-
serif, 270
 - otherwise are as usual, 270
- function, higher-order, **274**
- Galois connection, 306↓
- garbage
 - caused by pointers, 172
 - is in the heap, 172
 - sometimes called “space leak”, 173
 - what garbage is, 172–173
 - why garbage needs collection, 173
- GARBAGE COLLECTION, 171–194
 - Collector, **174**, 174
 - complete vs. safe, 191
 - free chain, 174
 - starts at **F**, **177**
 - label-like auxiliary variables, 189, *see*
also Peterson’s mutual-
exclusion algorithm
 - mark-sweep, 174, *see also* reference
counting
 - Mutator, **175**
 - and Collector interference easy to
avoid, 175
 - and Scanner interference, 175, 176◦,
178, 180, 181◦, 182, 185
 - N**, null pointer target, **177**
 - nodes, **177**
 - black, white, **178**
 - counting black nodes, 185–187
 - null pointer target **N**, **177**
 - “on the fly”, **174**
 - “poised” between atomic statements,
189
 - reachable node, 174, **177**
 - reference counting, 174, *see also* mark-
sweep
 - root node, **177**
 - special “**F**” and “**N**”, 177
 - safe vs. complete, 191
 - Scanner, **174**, 178–180, 182–184
 - global correctness (*GC*), 180–182,
184, 187, 189–190, 193
 - invariants for, 179, 180, 182, 183,
188, 189, 191, 327
 - local correctness, 179–180, 183–187,
189–193
 - variants for, 179, 182, 193, 327
 - Version 1 (wrong), 179, 182
 - Version 2 (ok, but), 183◦, 184
 - Version 3 (better, but), 185–187
 - Version 4 (correct), 192◦, 192
 - Scanner–Collector interference, 193
 - swinging a pointer, **177**
 - (end GARBAGE COLLECTION)
- garden shed, building one, *qv*
- Paul H.B. **Gardiner**, [222], [225]
- A.J.M. “Netty” **van Gasteren**, [157↓],
[225]
- GC*, *see* global correctness

- gcd**, greatest common divisor, *see* Euclid's algorithm
- genealogical order, 61
- ghost variable, *see* auxiliary variables
- global correctness (*GC*), 143, 147, **150**, 149–150, 159, 193, 248
- global invariance, 149
- global invariants, **148**, 163, 167
 compare globally correct, stable, 169
- global precondition, postcondition, *qv*
- global variable, 112, 161
 See also variable.
- globally correct (*GC*)
 compare stable, global invariants, 169
- K.F. **Gödel**, [202↓], [225↓]
 incompleteness theorem, 202↓
- Golden Rule, *see* E.W. Dijkstra
- Good* subsegment, longest, 31
 See also *Bad* subsegment.
- green route from Paris to Rome., 51
- David **Gries**, 144, 149↓, 177↓, [182↓], [223], [223], 223↓, [224, 225]
 See also **Owicki**.
- guard, *see* loop
- gummy bears, handing out, 238, 306
- handle, turning it to see the program flow, 10
- Ian J. **Hayes**, [222], [225]
- head-scratching
 “Use the assertions, Luke.”, 39
- heap, *see* garbage
- Eric C.R. **Hehner**, [223], [225]
- hint in proof, 334
- C.A.R. **Hoare**, 168, [222], 222, [225], 235–238
- horse, bolting, *see* formal methods
- “housekeeping” assertions, *qv*
- how to check programs, summary and definitions, **241**
- hybrid of abstract and concrete data-types, 89, 101, 119, 253
- hypothesis* package for **Python**, *qv*
- IBM**
 the **IBM 704** computer, 61↓
 Vienna Development Method, 222
- IDE*, interactive development environment, 230↓
- idempotent, 274, **282**
 idempotence of conjunction $\wedge$, 282
 idempotence of disjunction $\vee$, 274, 282
- if ... else: skip**, 69
- if-branches**
 reason about each **if**-case separately, 44
- if-branches**: reason about each **if**-case separately, 39
- “iff” is short for “if and only if”, 288
- IFM**, *see* (In-)Formal Methods
- Imperial currency: pounds, shillings and pence, 57
- implication, 272
 does not have an equivalent symbol in **Python**, 272
- entailment, *qv*
- five** different informal meanings, 289
- here written ($\Rightarrow$)
 different from entailment $\Rightarrow$, 274
- here written ($\Rightarrow$), **9**, 282
- used to be written
 antecedent $\supset$ *consequent*, 288↓
 See also antecedent, consequent.
- implicit increment, *see* **for**-loop
- index out of range, *see* “everyday” errors
- “indexitis”, 238↓
- infinite loop, avoiding, 14
 See also loop variants.
- infinitely descending chain, **59**
- infinity ∞
 eliminating from program, 53
 used as value in program, 52, 53, 309, 314, 315
 See also PROGRAM EXERCISES.
- (In-)Formal Methods (**IFM**)
 introduced using simple programs you already know how to write, xi
 some of its benefits, 74
 what it is, xii
- “inheriting” correctness, 247
- inlining, *see* procedure calls
- insertion sort, 72, *see also* PROGRAM EXAMPLES, exercises
- is inefficient, 72
- integer quotient $//$, *see* quotient
- interactive theorem provers, 208
- Isabelle, 225
- interference, **143**, 143–144, 149–324, 330,
 see also concurrency
 between **if** and **then/else**, 180
 between Mutator and Scanner, *see* GARBAGE COLLECTION
 not possible initially or finally, 324
- interleaving, 135, 147–148, 158
 between lines of code, 147↓
- intermediate assertions, 22, *see also* assertions
 figured out automatically, 199
- internal state, *see also* data-type encapsulation

- hidden by encapsulation, 98
- internet, program “fixes”, 4
- interrupts, 144, 324
 - await**-body inhibits, 154↓, 154
 - controlling them for mutual exclusion, 161
- introductory course, xiv
- # Inv**: indicates invariant, 18, 319
 - written *after* **while** in **Dafny**, 199↓
 - written before **while** but after **for**, 14↓, 245–247
- invalid, *see* valid
- invariants (and finding them), 21, 33–53, 232–233, 237
 - associated with locks, 137
 - cascading, 45–48, 94
 - choose them *before* coding, 33
 - coupling, *qv*
 - data-type, *qv*
 - even simple ones are helpful, 41, 46
 - for checking loops, 245
 - for garbage-collection scan, 179
 - See also* GARBAGE COLLECTION.
 - for recursion, 93, 98, **319**
 - global, *qv*
 - help to design loops, 28, 180
 - must usually be supplied by the programmer, 201
 - should be inductive, 48, 101, 105
 - tail invariants, 40–44, 44↓, 51, 56↓
 - try introducing a new variable, 35–40, 43
 - try splitting a conjunct, 28, 33–35
 - See also* loop, **# Inv**: .
- inversion (in sorting), **317**
- involution, **283**
- Isabelle/HOL, 225↓
- isolation, checking components in, *qv*
- iterating (in loops)
 - down, 49, 66, 70
 - see also* PROGRAM EXERCISES
 - towards the middle, 38–40
 - up, 29, 35, 41, 47, 66, 94, 179
- Jack
 - “be nimble”, 4
 - should be more careful, 4, 34, 55, 201, 212↓
- Daniel **Jackson**, [223], [225]
- Java* programming language, 3, 117↓, 209, 213↓
- Cliff B. **Jones**, [222], [224, 225]
- Anne **Kaldewaij**, [223], [225]
- Kansas City walkway collapse, xiii↓
- Kepler Conjecture, 225↓
- H. **Khoshnevisan**, [223]
- Jeffrey H. **Kingston**, [238↓]
- Gerwin **Klein**, [225]
- Morris **Kline**, [225], [235↓]
- Donald E. **Knuth**, [199↓], [225]
- König’s lemma, *see* Ex. 4.18
- label-like auxiliary variables, *qv*
 - See also* Peterson’s mutual-exclusion algorithm, garbage collection.
- LAD**, largest absolute difference, **36**
- lad**, *see* largest absolute difference
- Leslie **Lamport**, [157↓], [175↓], [177↓], [225]
- largest element
 - in a non-empty sequence, 7
 - in a (possibly empty) sequence, 6
- largest element+
 - in a non-empty sequence, 7
- LC*, *see* local correctness
- leap-year bug, *see* date calculation
- left hand does not know... , 88↓
- K. Rustan M. **Leino**, [198↓], 198, [199↓], [225]
- lex**, *see* longest end run
- lexicographic variant, **57**, 62
 - its underlying order, 60
 - sometimes* can be reduced to simple variant, 60, 61
 - but sometimes *can’t* be reduced to simple variant, 61
 - See also* termination.
- “libra” (£ for “pounds”), 57↓
- linear search, 50, 80–85, 91, 92, 124, *see also* PROGRAM EXAMPLES
 - sometimes too inefficient, 79
 - See also* binary search.
- linear-time exponential, 41
- Lisp** functional programming language, 61↓
- livelock, *see* concurrency
- liveness, *see* concurrency
- local
 - procedures in encapsulation, 105, 106◦, 107
 - variables
 - in concurrent thread, 136, 249
 - in encapsulation, 86, 103–105, 107, 118
- local** is not a declaration in **Python**, 86↓, 103↓
- local correctness (*LC*), 143, 148, 193
- local def**, 106◦, *see also* data-type encapsulation
 - find(s)** example, 124
- local reasoning, the importance of, 48

- lock variable **\$bb** associated with lock **b**,
 see locks
- lock-free implementation, 169
- “locked box”, *see* data-type encapsulation
- locks, 153
 - adding them, 134–136
 - implemented with *CAS*, *qv*
 - lock statement, 136
 - lock variable **\$bb** associated with lock
 bb, 137
 - originally used *P()* and *V()*, 139
 see also Dijkstra
 - their original motivation, 138
 - unlock statement, 136
 - what locks guarantee, 137, 138
 See also concurrency.
- log-time exponential, 42, 266
- logarithmic, *see* time complexity
- logic
 - and bicycles, 271↓
 - is the arithmetic of computer science,
 269
- lgs**, *see* longest *Good* subsegment
- longest run **lr**, **43**, 43, **53**, 129, 238, *see*
 also PROGRAM EXAMPLES
- of *x*’s, 238
- longest run of almost-consecutive *x*’s, 238,
 306
- longest up-sequence, 217
- longest *Good* subsegment **lgs**, **28**, 25–31
 See also PROGRAM EXAMPLES.
- looking outwards vs. looking inwards, *see*
 specification
- loop
 - (un)conventional thought processes for
 its design, 74
 - checking its body (invariant maintained,
 variant decreased), 14
 - checking its guard, for entry, 12
 - checking its initialisation, 12
 - early exit with **break**, 21, 22, 29◦, 38,
 49, 52, 57, 72, 82, 90, 92, 156,
 167, 169, 179, 237↓, 237, 246,
 247, 300, 301, 304, 311, 317
 - early restart with **continue**, 22, 38,
 301
 - exit, 14
 - (avoiding) infinite, 21, 144, 148↓, 153
 see also variants
 - invariants, 11–15, 18, 19, 25–53, 101,
 139
 - rolling-up, 12
 - unrolling, 8, 9, 11, 37
 - preserves correctness, 37
 - variants, 14–15, 55, 60, 66, 69, 70,
 72, 246
 - for spinlock, 144
- the “Lost Art”, xii↓, 76, 197, 215, 218
- lr** longest run, 43, 53, 129, 238, *see also*
 PROGRAM EXAMPLES, 334
- lunch, quick fix as you leave for, 37↓
- Macquarie University, xiv
- magic
 - ancient, 217↓
 - program, 220
- major unarticulated assumption *MUA*, *qv*
- majority voting **csm**, 42–44, *see also* PRO-
 GRAM EXAMPLES
- malloc()**, **free()**, 172 ff, 177
- margins, *see* rubrics
- mark-sweep, *see* GARBAGE COLLECTION
- A.J. **Martin**, [175↓], [177↓], [225]
- mathematics all the way down, xii, 196–
 203, 216
- matrix
 - multiplication used to calculate Fi-
 bonacci numbers, 94
 - puzzles, 22, 62, 63, 312
- matrix, determinant when empty
 see also exercises
- Matthew 7:26*, *see* (building on) sand
- max** takes the maximum of two elements,
 6, 22, 232
- MAX** takes the maximum of many elements,
 6, 22, 232
- maximum
 - of an empty sequence, 22
 - of no values, 232
 - of three named values, 18
 - of four named values, 70
 - of a sequence of values, 18, 22
- maximum end sum **mes**, 47
- maximum segment sum **mss**, 45–48
 - is $O(N^3)$, 46
 - is $O(N^2)$, 46
 - is $O(N)$, 48, 52
 - specification of, 45
 See also PROGRAM EXAMPLES.
- maximum segment-product **xsp**, 52
- maximum up to here **muh**, **46**
- Mean Calculator, 117–121
- B. **Meltzer**, [202↓], [225]
- mes**, *see* maximum end sum
- message passing, 224
- meta theorem, *see modus ponens*
- method, *see also* object orientation
 - get- and set-, 107
 - of a class, 89
- Bertrand **Meyer**, 7, [222, 223], [225]
- Robin **Milner**, [222], [225]

- min takes the minimum of two elements, 232
- MIN takes the minimum of many elements, 232
- minimal has no others less than it, 58, 61, 316
See also minimum.
- minimum
 is less than all others, 58, 61, 316
 of no values, 232
See also minimal.
- minimum spanning tree
 Prim's algorithm, [51]
 Jayadev **Misra**, [162↓], [225]
 and Peterson's algorithm, 162↓
- model checkers, 208
- modularity and re-use, xiii
- modus ponens*, **289**
 is a meta-theorem, 290
 is a rule of inference, 290
- mole, whack-a-, 202↓
- Moodle, 220↓
- Carroll **Morgan**, [117↓], 138↓, [216], [221], [223], [225], [281↓]
- motorcycle, tuning your own vs. building one, 73
- moving the goalposts in meta-mathematics, 202↓
- mss**, *see* maximum segment sum
- MUA*, major unarticulated assumption, 206
- muh**, *see* maximum up to here
- multi-core system, 151↓
- multiple assertions, how to check
 on a single line, **243**
 on successive lines, **243, 319**
- multiple assignment, *qv*
- multiplying back, *see* debugging
- multiset (also bag), 117, **118**, 322
See also termination.
- music player, frozen, *qv*
- Mutator, **175**, *see also* garbage collection
- mutual exclusion, 154, 155○, *see also* concurrency
- N, null-pointer target, *see* GARBAGE COLLECTION
- n.--**, either **n.left** or **n.right**, **183**
 “For want of a nail...”, 212↓
- negation, written “**not**” or “¬”, 272
- negative numbers, still confused?, 235↓
- nep**, *see* minimum end product
- Tobias **Nipkow**, [225]
- nodes, *see* GARBAGE COLLECTION
- non-empty-segment sum, 52
- nondeterminism, 256, 307
 caused by concurrency, 133↓
- demonic, 254
- not**, *see* negation
- NUL** at end of character string, *see* sentinel
- null pointer **N**, *see* GARBAGE COLLECTION
- numeral vs. number, 270
- OB* abbreviates “other business”, 153
- object orientation, **301**
 ... is not discussed generally in this text, 123↓
- access modifier, 301
 package, private, protected, public, 301
- class
 attribute, definition, instance, method, 112, 301
 encapsulation's role, 80
 in **Python**, 301
 inheritance, 123↓
 instantiation, 123↓
 its motivation, 123↓
 one public procedure calling another, 301
- office block, *see* building one
- one-point laws in logic, **286**
- operational intuition, 26
- operational reasoning, 26, *see* static reasoning, 318
 gives insight, 165
 often helpful, but not to be relied on exclusively, 39↓, 165↓
- opinions
Dafny does not have them, 201
See also program checking.
- “**or**”, *see* disjunction
- other business, abbreviated *OB*, 153
- outdenting
 assertions, **299**
 for **unlock**, 165↓
 of loop postcondition, 299
- outside-in program design, *see* program
- overflow bug, *see* binary search
- Susan **Owicki**, 144, [149↓], 177↓, 182↓, **223**, 223↓, [224, 225]
- Owicki–Gries Method, 144, 147–159, 223↓, *see also* concurrency
- owned variable, *qv*
- P()**, *see* locks
- parentheses sometimes added during substitution, 230, 303
- Paris, 51
- partial correctness, **55**, 237, 245
See also total correctness.
- partial order, **59**, *see also* total order
- pass**, *see* **skip**

- pence, *see* Imperial currency
 perendinate, 250↓
 G.L. **Peterson**, 75↓, 161–169
 his mutual-exclusion algorithm, 158–
 167, 188, 189
 “poised” between atomic statements,
 166
 cannot deadlock, 167
 complete code for, 166○
 is safe, 166
 label-like auxiliary variable, 166
 prevents starvation, 167
 pointers, 172, *see* *C*, programming lan-
 guage
 null, swinging, *see* GARBAGE COLLEC-
 TION
 “poised” between atomic statements, 189
 See also Peterson’s algorithm.
 polygons puzzle, 62, *see also* PROGRAM
 EXERCISES, termination
 #**POST**: indicates postcondition, 6, 8
 postcondition, 6, 16, 21
 assertion labelled **POST**: , 6
 at the end of the program, 6
 global, 148
 interpreting, 49
 that depends on the precondition, 56,
 66
 use of auxiliary variables for, 56, 66
 See also program.
 pounds, shillings and pence, *see* Imperial
 currency, PROGRAM EXAMPLES
 (ATM)
 (to the) power of **, 41
 #**PRE**: indicates precondition, 6, 8
 precedence
 of operators, **274**
 sometimes requires parentheses, 285
 precondition, 6, 16, 21, *see also* program
 assertion labelled **PRE**: , 6
 default is **True**, 8, 66
 global, 148
 of a class, 107
 weakest, 222, 310
 predicate
 in logic, **271**
 vs. predicate symbol, 271
 predicate calculus, 201, 275–278, 280, 284–
 288
 predicate symbols
 if introduced here are written in sans-
 serif, 271
 Robert C. **Prim**, *see* minimum spanning
 tree
 primitive data-types, *qv*
 Jan F. **Prins**, [225]
 procedure calls, inlining, 86
 process algebra, *see* concurrency, 224
 procrastination, *see* specification of pro-
 gram
 program
 basic, **241**
 checking
 if automated does not have “opin-
 ions”, 4, 201, 202
 like baton passing, 10
 like falling dominoes, 10
 counter, 10↓
 travelling with it, 13○
 dividing a big programming problem
 into a tree of smaller ones, xiii,
 8, 17, 21
 ... then putting the pieces back to-
 gether, 9
 flow, 10, *see* flowchart
 its purpose stated in requirements, 5
 magic, 217↓
 outside-in program design, 29, 34, 39,
 41, 44, 185, 188, 222, 238, 241,
 266
 postcondition, 6–7
 precondition, 6–7
 default is **True**, 8
 recursive, 93
 requirements, 6–7
 “What is a program for?”, 5
 “When does a program work?”, 5, 17
 program algebra, 52, 113
 program code-review, 4
 program components
 assignment statement **x= exp**, 65–67
 break (within loop), 21, 22, 29○, 49,
 52, 57, 72, 82, 90, 156, 167, 179↓,
 301, 317
 its interaction with invariants, 22,
 301
 continue (within loop), 22, 301
 its interaction with invariants, 301
 for-loop vs. **while**-loop, 12, 247, 300,
 309
 if-conditional, 68–70
 sequential composition, **244**
 skip (do nothing), 67
 while-loop, 12
 PROGRAM EXAMPLES
 automated teller machine (ATM)
 in Imperial currency, 57
 many active concurrently, 134
 binary search, 38–40, 50, 69, 80, 82,
 101, 111, 115, 199
 bubble sort, 71

- circular structure, made from pointers, 191
- critical section, 153, 159
- date calculation
 - calculating “today” from day number, 4, 21–23, 34
 - is missing its variant, 15
- Euclid’s algorithm for **gcd**, 56
- exchange sort, 72
- Fibonacci numbers, 93, 94
 - in log-time, 95–97
 - recursive version is inefficient, 93
- for**-loop vs. **while**-loop, 300
- garbage collection, 171–194
- Good* subsegment, 25
 - caterpillar analogy, 27
- insertion sort, 72
- LAD**, largest absolute difference, 36
- largest element in a
 - non-empty sequence, 7
 - (possibly empty) sequence, 6, 7
- leap-year bug, 4, 15
- least absolute difference, 36
- linear search, 80, 82, 83, 85
- linear-time exponential, 41
- log-time exponential, 42, 94, 95, 266
- log-time Fibonacci, 93–99
- longest run **lr**, 43, 53, 238
- longest *Good* subsegment **lgs**, 25–31
- loop
 - checking its body (invariant maintained, variant decreased), 14
 - checking its guard, for entry, 12
 - checking its initialisation, 12
- majority voting **csm**, 42–44, 51–52
- maximum
 - of three named values, 18
 - of four named values, 70
 - of a sequence of values, 18
- maximum segment sum **mss**
 - is $O(N^3)$, 46
 - is $O(N^2)$, 46
 - is $O(N)$, 48, 52
- mutual exclusion, 154, 155◦, *see also*
 - concurrency
- Peterson’s mutual-exclusion algorithm, 166◦, *qv*
- quotient and remainder (finding), 17, 60
- rectangular-matrix puzzle, 63, *see also*
 - PROGRAM EXERCISES, termination
- searching, binary, linear, *qv*
- set* (concrete)
 - represented by fixed-size array, 106◦
- shortest run **sr**, 53
- sorting
 - three named variables, 68
 - four named variables, 71
- summing a sequence, 5–6, 12–15, 19, 55
- swap two named variables, 70
- train-carriages puzzle, 63
- unrolling a loop, 8, 9, 11
- using infinity ∞ , 309
- vertical format for assertions, 10
- program examples
 - Good* subsegment, 31
- (end PROGRAM EXAMPLES)
- PROGRAM EXERCISES
 - assignment statement
 - checked by textual substitution, 35
 - binary search, 49–50
 - Boolean arithmetic, 91
 - break** to exit loop, 22, 72
 - bubble sort, 71
 - checking conditionals (**if**-statements), 70
 - circular reasoning, 150
 - continued fraction, 20
 - coupling invariants, 99, 120, 128, 158
 - critical section, 145, 159
 - date calculation
 - calculating “today” from day number, 23, 49
 - is missing its variant, 23
 - empty sequence or subsegment
 - sum of it is 0, 52
 - “everyday” errors, 20
 - exchange sort, 72
 - find maximum of 3 values, 18
 - flowchart
 - making one, 22
 - for**-loop
 - assertions for, 50
 - vs. **while**-loop, 49
 - global correctness (*GC*), 193
 - Good* subsegment, 31
 - gummy bears, handing out, 238, 306
 - index out-of-range, 20
 - infinity ∞ used as value in program, 53
 - insertion sort, 72
 - intermediate assertions, 22
 - iterating (in loops)
 - down, 19, 49
 - local correctness (*LC*), 193
 - lock-free implementation, 169
 - longest run **lr**, 53, 129
 - longest run of almost-consecutive **x**’s, 238
 - maximum

- of an empty sequence, 22
- of three named values, 18
- of four named values, 70
- of a sequence of values, 18, 22
- maximum segment-product **xsp**, 52
- non-empty-segment sum, 52
- postcondition, 6
- program algebra, 52, 113
- quadratic complexity, 52
- Rome (all roads lead to), 51
- rounding up or down, 50
- safety, 168
- sequential composition of programs
 - how to check, 21, 22, 70
- shortest run **sr**, 53
- sorting, 49, 71
 - four named variables, 71
 - a whole sequence, 71, 72
- spin lock, 144
- spin loop, 159
- stable condition, 169
- summing a sequence, 19
- swap two named variables, 70
- tail invariant, *qv*
- termination, 61, 62
 - dictionary order, 61
 - lexicographic variant, 61, 62
 - multiset ordering, **62**
 - polygons puzzle, 62
 - quotient and remainder (finding), 60
 - rectangular-matrix puzzle, 62
 - social-distance puzzle, 63
 - trains-carriages puzzle, 63
 - well-founded sets, 62, *qv*
- train-carriages puzzle, 63
- uninitialised variable, 51
- variants (and finding them), 193
 - bounded above and increasing, 21
 - bounded below and decreasing, 70
 - for infinite loop, 144
- while-loop** vs. **for-loop**, 49
- (end PROGRAM EXERCISES)
- proof, hint in, *qv*
- proof by induction, 48, 126↓
- propositional calculus, 272–275, 278–284
 - Do beginning programmers need it?, 215
 - examples of its utility, 179↓
- propositional connective, 272
- prototype
 - “quick and dirty” refined later to a more sophisticated implementation, 102
 - See also* data-type encapsulation.
- pseudocode
 - does not allow rigorous reasoning, 87
 - more precise than, 86–87, 102, 117, 124
- Python**, 3
 - caveats — take care, 299
 - font**, **3**
 - hypothesis* testing package, 31↓, 211↓, 219↓
 - mentioned mainly in footnotes, 3↓
 - used in this text, xiii, 299
 - uses reference counting for garbage collection, 174↓
- quadratic complexity $O(N^2)$, 52
- quantification
 - body of, **276**
 - existential, there exists ($\exists$), **276**
 - nested, **278**
 - typed, **277**
 - universal, for all ($\forall$), 200◦, 201, **276**
 - untyped, **277**
- queue, two-place for mutual exclusion, 162
- “quick and dirty”, *see* prototype
- quick fixes
 - to be avoided, 37
- quotient and remainder (finding), 60, *see also* termination
 - integer quotient **//**, 17
 - remainder **%**, 17
- range(low,high)** is inclusive/exclusive, 5
 - range(1,N)**, 6
 - default **low** is zero, 5
- reachable node, *see* GARBAGE COLLECTION
- rectangular-matrix puzzle, 63
- recursion, 93
 - See also* program.
- reference counting, *see* GARBAGE COLLECTION
- refinement, *see also* data refinement
 - can improve a specification, 118
 - data-, **112**
 - equality is a special case, 112
 - guaranteed by adding data-type invariants, 112
 - is preservation of properties, 261
 - its dependence on vocabulary, 261
 - of code, **110**
 - proper, **112**, 118
- refinement calculus, 222
- reflexive, **59**
- register, of *CPU*
 - atomic operations on, 136
- release()** in **Python**, 135↓
- relocation project in Fiji, 207
- remainder **%**, *see* quotient

- remove(s)** element from set, 114
- repeat-loop**, 179↓, 237↓
 - invariant need not be true on initialisation, 179
- requirements, 6, 7, **16**, 21
 - analysis, 7
 - examples, 26
 - indicated by “# ---”, 5
 - lead to specifications, 207
 - should not be too detailed or technical, 16, 83, 117, 118, 207
 - vs. specification, 26, 208
 - of program, 17
 - whether to buy a thing, vs. how to use it, 118
- “result” formal parameters, 256
- John C. **Reynolds**, [223], [225]
- Darren W. **Ringer**, 138↓
- Kenneth A. **Robinson**, 219↓
- Rodin* tool, 219↓
- rolling-up a loop, *qv*
- Rome (all roads lead to), *see* PROGRAM EXERCISES
- root node, **174**
 - See also* garbage collection.
- rote learning
 - “What’s the point?”, 229
- rounding up or down, 50
- Rubik’s Cube, 63
- rule of inference, *see* *modus ponens*
- safety, *see* concurrency
- salesman, 205
- sand, building on, 212↓
- J.W. **Sanders**, [225], [281↓]
- satisfaction modulo theories (*SMT*), 202↓, 230↓
 - See also* **Z3**.
- Scanner, **174**
 - See also* garbage collection.
- C.S. **Scholten**, [175↓], [177↓], [225]
- Stephen **Schumann**, [222], [225]
- scope rules, enforced by compiler, 102, 107
- scope, global vs. local, 86↓
- searching
 - binary (logarithmic), *qv*
 - linear, *see* PROGRAM EXAMPLES
- (sub-)segment, *see* sequence
- semantics, the meaning of a programming language, 225
- semaphore, 139–142
 - binary, 139
 - originally proposed by E.W. Dijkstra
 - in his note *EWD74*, 139
 - passering, vrijgeven, 139
 - prolaag, verhoog, 139↓
- sentinel, **83**, 83–85
 - NUL** ends character strings in *C*, 83↓
- sequence or subsegment
 - A[0:N]**, **5**, **231**
 - A[:high]** is short for **A[0:high]**, 231
 - A[low:]** is short for **A[low:len(A)]**, 231
 - A[low:high]** is inclusive/exclusive, **5**, 31, 39, 49, **231**
 - concatenation is **+**, **20**, 44, **82**, 82
 - empty, *qv*
 - is **A[n:n]**, 49, 52
 - “falling off the end”, 83
 - negative index, 231
- sequences are indexed from 0, 5
- sequential composition of programs
 - how to check, 21, 22, 70, **244**
 - in *Python*, in *C*, 244↓
 - of program components, 21, 22, 70
- sequential programs (not concurrent), 148
 - only one thread, 143
- ser**, *see* shortest end run
- serial access (to a class), 143
- serial use (in a concurrent system), 135
- set* data-type
 - abstract, 86
 - compared with concrete, 108
 - as primitive
 - difference **-**, **82**
 - membership **∈**, **82**
 - union **∪**, **81**
 - concrete
 - represented by fixed-size array, 103, 106◦
 - represented by sequence, 81–83, 87, 101
 - empty **{}**
 - written **set()** in *Python*, 81
 - See also* data-type encapsulation.
- (elementary) set theory, 267–268
 - is an element of **∈**, 267
 - set comprehension **{·, ·, ·}**, 267
- shillings, *see* Imperial currency
- shortest run **sr**, **53**, 302
 - See also* PROGRAM EXERCISES.
- $\sum$ means “the sum of”, **5**
- simple formula, *see* formula, atomic
- skill, programming as a, 73
- skip** does nothing, 67
 - called **pass** in *Python*, 37↓, 67↓, 242
 - checking by implication, 67, 69, **242**
 - default **else** of conditionals (**if**-statements), 69, 135
- SMT*, *see* satisfaction modulo theories
- Jan L.A. **van de Snepscheut**, [177↓], [182↓], [223↓], [225]

- social process of programming, xiii↓, xiii, 15, 73, 102, 314
- social-distance puzzle, 63
- Software-Engineering stream at UNSW, 219
- sorting, 49, 71
 - three named variables, 68
 - four named variables, 71
 - See also* PROGRAM EXAMPLES.
 - a whole sequence, 71, 72
 - bubble sort, 71
 - exchange sort, 72
 - insertion sort, 72
 - inefficiently, *see* bubble sort, exchange sort, insertion sort
- soundness, **289**
- space leak, *see* garbage
- spaghetti coding, 139↓
- specification of program, **7**, 6–10, 43↓, 185, 256
 - as contract, 7
 - examples of, 26
 - how to check, **250**
 - implementing, 193
 - looking outwards vs. looking inwards., 124
 - satisfying it, 16
 - seen as procrastination, 250
 - used as program text, 47◦, 47, 124–126, 128, 185, 186◦, 193
 - vs. **assert** then **assume**, 129
 - vs. requirements, 7, 16, 17
 - written **x**: [*pre*, *post*], 125
 - written in mathematical language, xii
- specification statement **x**: [*pre*, *post*] , 125, 128, 129, 185, 186◦, 193, **251**, 250–251
 - frame, 125
- spin lock, **138**, 144, 156, 157
 - vs. spin loop, 138↓, 157
- spin loop, **157**, 157, 159, 161, 167
- J. Michael **Spivey**, [222], [225]
- split a conjunct, 33–35, 43, 179
- spot (·) in quantification, **276**
- sr**, *see* shortest run
- stable condition, 167, 169
 - compare globally correct, global invariants, 169
- starvation, *see* concurrency
- state space of program
 - maps variables to values, 270
- static reasoning
 - much more effective than operational reasoning, 217
- E.F.M. **Steffens**, [175↓], [177↓], [225]
- N. **Stenning**, [188↓]
- structural variant
 - sometimes can be reduced to simple variant, 61
- structural variant, sometimes can be reduced to simple variant
 - See also* termination.
- structured data-types, 58
- subscript out of range, *see* index out of range
- subsegment vs. subsequence, 25↓
- substitution, 230–231
 - explains quantifiers, 286
 - for checking assignment statements, *qv*
 - into formula (notation for), 285
 - into quantified formulae, 277, *see also* alpha conversion, 286
 - replaces one string by another, **230**
 - variable capture, 277, 286
 - written **post** [**x***expr*], 8, **144**, **242**
- substitution for checking assignments, 230
 - adding parentheses is sometimes necessary, 65, 242, 309
 - kept separate from reasoning about assertions themselves, 65
- summing a sequence, 5–6, 12–15, 19
 - loop variant for, 55
- swap two named variables, 67, 70
 - See also* PROGRAM EXERCISES, PROGRAM EXAMPLES.
- swinging a pointer, *see* GARBAGE COLLECTION
- swoop, foul, 149↓
- synchronised methods, 143
- syntax errors, xi
- systems analyst, 205, 206
- tail invariants, 56↓, 40–56
 - See also* invariant.
- tail of a sequence, 231
- Tantalus, 202↓
- tax return, the importance of correct calculations there, 263
- term, in logic, **270**
- termination, 29, 55–62, 128, 179–187, 237, 245–247, 322, 324
 - dictionary order, 61
 - lexicographic variant, 61, 62
 - sometimes *can't* be reduced to simple variant, 61
 - sometimes can be reduced to simple variant, 60
 - lexicographic variant, sometimes can be reduced to simple variant, 61
 - multiset (also bag)
 - ordering, **62**

- polygons puzzle, 62
 - See also* PROGRAM EXERCISES.
- quotient and remainder (finding), 60
- rectangular-matrix puzzle, 62
 - See also* PROGRAM EXERCISES.
- social-distance puzzle
 - See also* PROGRAM EXERCISES, termination.
- structural variant
 - sometimes can be reduced to simple variant, 61
- train-carriages puzzle
 - See also* PROGRAM EXERCISES.
- well-founded sets, 61, 62
- testing
 - and “tweaking”, 26
 - assisted by assertions, 210
 - auto-marking, 217
 - encouraged at every stage, 220
 - is always necessary, xiii, 31↓, 53, 74↓, 75, 197, 205–212, 221, 323
 - reveals bugs, 199↓
 - using Python’s *hypothesis* package, 219↓
 - with **assert** statements, 323
 - catches errors early, 211
- there exists $\exists$, *see* quantification
- thread, **133**, 134–136, *see also* concurrency
 - guarantees execution order of its components, 136, 147
- time “complexity” (efficiency), **41**
 - logarithmic $O(\log N)$, 41, 42, 94, 101, 251
 - linear $O(N)$, 41–43, 47, 48, 52, 94, 314
 - $O(N \log N)$, 43
 - quadratic $O(N^2)$, 46, 52, 314
 - cubic $O(N^3)$, 46
 - exponential $O(c^N)$, 93, **98**
- times and addition tables, 229
- tomorrow, the day after, 250↓
- total correctness, **55**, 237, 245
 - See also* partial correctness.
- total order, **59**, *see also* partial order
- train-carriages puzzle, 63
 - See also* PROGRAM EXERCISES, termination.
- transactions, *see* concurrency
 - shared state, 224
- transitive, **59**, 275
 - example $\leq$, 68
- transliteration, 4↓, 97↓, 199, 212, 213, 222, 247
- trigonometry “on the fly”, 289↓
- “true”
 - True vs. **True**, 271, 272↓
 - used as English word, 264
 - written **True** as Boolean in Python, **8**, 8↓, 13
 - written True for validity of logical formula in a state, 279
- True** as default precondition, *qv*
- true** describes all states
 - unit of conjunction $\wedge$, 282
 - zero of disjunction $\vee$, 282
- Trustworthy Systems Group, xiv, xv
- truth table, 272
- “tweaking”, *see also* efficiency and neatness of programs
 - for correctness (really?), 26
- undefined expression, 243
- undefined variable, 255
- undo()** procedure for Mean Calculator, 120
- uninitialised variable, 51
 - See also* “everyday” errors.
- unit of an operator, 282, *see also* zero of, **false**, **true**
- universal quantification, for all ($\forall$), *qv*
- universally valid, 284
- University of New South Wales, xiv, 216, 219
- unlock, *see also* concurrency
 - twice in a row, 137
- unrolling a loop, *qv*
- UNSW, *see* University of New South Wales
- ur*-polygon, *qv*
- use cases, 87
- using infinity ∞ in your program, 309
- V()**, *see* locks
- “valid” means correct, in logic, 269↓
- validation, 7
- valuation, in formal logic, 270
- A.J.M. “Netty” **van Gasteren**, [223]
- VAR:**, **55**
- variable
 - bound, **277**, *qv*
 - capture, 277, 286
 - see also* substitution, alpha conversion
 - free, *qv*
 - fresh, **286**
 - global in concurrent thread, 144, 323
 - in logic written x , 270
 - in programs written x , 270
 - list of, in quantification, 276
 - owned by a thread, 144
- variance, statistical, 316↓
- variants (and finding them), **15**, **55**, 55–60, 193, 212↓, 233, 237
 - bounded above and increasing, 15, 21, 91

- bounded below and decreasing, 15, 35, 70
 - for infinite loop, 144, *see also* PROGRAM EXERCISES
 - for recursion, 93, 98, **319**
 - how to check loop termination, 246
 - lexicographic (dictionary order), **57**, 57–58, 61
 - loop termination can occur before the variant reaches its bound, 316
 - multiset ordering, 62
 - must strictly in- or decrease, 15, 35, 55, 56, 233, 317
 - “real valued”, 59, 62
 - simple, 55–63
 - structural, 58
 - use auxiliary variables to check, 66, 70, 72
 - variants do not necessarily count iterations, 316
 - well-founded, 59
 - when there can’t be one, 15
 - See also* loop.
- VDM, the Vienna Development Method, 222
- Venn diagram of sets, 268^o, 268
 - not very helpful here, 268, 269^o
- verification, 7, 8
- verifier, *see* checking programs automatically
- vertical format for assertions, 10, 12[↓]
- Steven **Vickers**, [223]
- Trevor N. **Vickers**, [225]
- vocabulary of refinement, *qv*
- w, node colour is white, **179**
- weakest precondition, *qv*
- weekends, getting them back, xii, 266
- well-formed formula, *see wff*
- well-founded sets, termination, **59**
- wff*, well-formed formula, **288**
- What does this do?* -style comment, *qv*
- What’s true here.* -style comment, *qv*
- “What’s true here.” assertions, **16**, 28, 29, 48, **51**, 75, 83, 88, 120, 147, 165[↓], 208, 210, 212, 242, 264–267, 281
- while**-loop
 - vs. **for**-loop, 300
 - how to check partial correctness, **245**
 - how to check termination, **246**
 - how to check total correctness, **247**
 - with **break** statements, **246**
- white nodes, *see* GARBAGE COLLECTION
- Who checks the programs that check the checks?*, 216[↓]
- Niklaus **Wirth**, [222], [225], [301[↓]]
- James C.P. **Woodcock**, [222], [225]
- M. **Woodger**, [188[↓]]
- “working” program, 17
- Joakim **von Wright**, [225]
- x**: [*pre*, *post*] , a specification statement, **125**
 - used as program text, 125
- xep**, *see* maximum end product
- xsp**, *see* maximum segment-product
- yield()**, 144, 324, *see also* concurrency
- Z3** satisfaction-modulo-theories automated theorem prover, 198[↓]
 - used by **Dafny**, *qv*
- zero of an operator, 282, *see also* unit, false, true
- zero-origin of sequence-indexing, *qv*
- Zune*, *see* PROGRAM EXAMPLES (leap-year bug)