Computer Science

Nguyễn Quản Bá Hồng[1]

November 19, 2022

[1]Independent Researcher, Ben Tre City, Vietnam
e-mail: nguyenquanbahong@gmail.com; website: https://nqbh.github.io.

Contents

Chapter 1

Wikipedia's

1.1 Wikipedia/Computer Science

Fundamental areas of computer science. Programming language theory, Computational complexity theory, Artificial intelligence, Computer architecture. History, Outline, Glossary, Category.

Computer science is the study of computation, automation, & information. Computer science spans theoretical disciplines (such as algorithms, theory of computation, & information theory) to practical disciplines (including the design & implementation of hardware & software). Computer science is generally considered an area of academic research & distinct from computer programming.

Algorithms & data structures are central to computer science. The theory of computation concerns abstract models of computation & general classes of problems that can be solved using them. The fields of cryptography & computer security involve studying the means for secure communication & for preventing security vulnerabilities. Computer graphics & computational geometry address the generation of images. Programming language theory considers approaches to the description of computational processes, & database theory concerns the management of repositories of data. Human–computer interaction investigates the interfaces through which humans & computers interact, & software engineering focuses on the design & principles behind developing software. Areas such as operating systems, networks & embedded systems investigate the principles & design behind complex systems. Computer architecture describes the construction of computer components & computer-operated equipment. Artificial intelligence & machine learning aim to synthesize goal-orientated processes such as problem-solving, decision-making, environmental adaptation, planning & learning found in humans & animals. Within artificial intelligence, computer vision aims to understand & process image & video data, while natural-language processing aims to understand & process textual & linguistic data.

The fundamental concern of computer science is determining what can & cannot be automated. The Turning Award is generally recognized as the highest distinction in computer science." – Wikipedia/computer science

1.1.1 History

1.1.2 Etymology

1.1.3 Philosophy

1.1.3.1 Epistemology of computer science

1.1.3.2 Paradigms of computer science

1.1.4 Fields

1.1.4.1 Theoretical computer science

1.1.4.1.1 Theory of computation.

1.1.4.1.2 Information & coding theory.

1.1.4.1.3 Data structures & algorithms.

1.1.4.1.4 Programming language theory & formal methods.

1.1.4.2 Computer systems & computational processes

1.1.4.2.1 Artificial intelligence.

1.1.4.2.2 Computer architecture & organization.

1.1.4.2.3 Concurrent, parallel & distributed computing.

1.1.4.2.4 Computer networks.

1.1.4.2.5 Computer security & cryptography.

1.1.4.2.6 Databases & data mining.

1.1.4.2.7 Computer graphics & visualization.

1.1.4.2.8 Image & sound processing.

1.1.4.3 Applied computer science

1.1.4.3.1 Computational science, finance & engineering.

1.1.4.3.2 Social computing & human–computer interaction.

1.1.4.3.3 Software engineering.

1.1.5 Discoveries

1.1.6 Programming paradigms

1.1.7 Academia

1.1.8 Education

1.2 Wikipedia/How to Solve it by Computer

"*How to Solve it by Computer* is a computer science book by R. G. Dromey, 1st published by Prentice-Hall in 182. It is occasionally used as a textbook, especially in India.

It is an introduction to the *whys* of algorithms & data structures. Features of the book:

1. The design factors associated with problems
2. The creative process behind coming up with innovative solutions for algorithms & data structures
3. The line of reasoning behind the constraints, factors & the design choices made.

The very fundamental algorithms portrayed by this book are mostly presented in pseudocode &/or Pascal notation." – Wikipedia/How to Solve it by Computer

Part I

The Art of Computer Programming

The Art of Computer Programming (TAOCP)

"At the end of 1999, these books were named among the best 12 physical-science monographs of the century by American Scientists, along with: Dirac on quantum mechanics, Einstein on relativity, Mandelbrot on fractals, Pauling on the chemical bond, Russell & Whitehead on foundations of mathematics, von Neumann & Morgensstern on game theory, Wiener on cybernetics, Woodward & Hoffmann on orbital symmetry, Feynmann on quantum electrodynamics, Smith on search for structure, & Einstein's collected papers. Wow" "historic" publisher's brochure from the 1st edition of Vol. 1 (1968). A complimentary *downloadable PDF containing the collected indexes* is available from the publisher to registered owners of the 4-volume boxed set. This PDF also includes the complete indexes of Vols. 1, 2, 3, & 4A, as well as to Vol. 1 Fascicle 1 & to Vol. 4 Fascicles 5 & 6."

eBook versions

"These volumes are now available also in portable electronic form, using PDF format prepared by the experts at Mathematical Sciences Publishers. Special care has been taken to make the search feature work well. Thousands of useful "clickable" cross-references are also provided – from exercises to their answers & back, from the index to the text, from the text to important tables & figures, etc.

Warning. Unfortunately, however, non-PDF versions have also appeared, against my recommendations, & those versions are frankly quite awful. A great deal of expertise & care is necessary to do the job right. If you have been misled into purchasing 1 of these inferior versions (e.g.,a Kindle version), the publishers have told me that they will replace your copy with the PDF edition that I have personally approved. **Do not purchase eTAOCP in Kindle format if you expect the mathematics to make sense**. (The ePUB format may be just as bad; I really don't want to know, & I am really sorry that it was released). Please do not tell me about errors that you find in a non-PDF eBook; such mistakes should be reported directly to the publisher. Some non-PDF versions also masquerade as PDF. You can tell an authorized version because its copyright page (with the exception of Vol. 4 Fascicle 5) will say 'Electronic version by Mathematical Sciences Publishers (MSP)'.

The authorized PDF versions can be purchased at `www.informit.com/taocp`. If you have purchased a different version of the eBook, & can provide proof of purchase of that eBook, you can obtain a gratis PDF verson by sending email & proof of purchase to `taocp@pearson.com`."

Volume 1

- *Fundamental Algorithms*, 3rd Edition (Reading, Massachusetts: Addison-Wesley, 1997), xx+650pp.
- *Volume 1 Fascicle 1*, MMIX: A RISC Computer for the New Millennium (2005), v+134pp.

1.2.0.1 Brochure

> "I am overwhelmed by the wealth of exciting & fresh material you have managed to pack into the book, especially in view of the fact that it is only the 1st of 7 volumes! "Monumental" is the only word for it ... Moreover, it is written with a grace & humor that is, as you know, exceedingly rare in books on mathematics. I greatly enjoyed your dedication, your flow-chart for reading the series, your notes on the exercises; above all, your choice of illustrative material throughout & the clarity & brevity with which you explain everything." – Martin Gardner, *Mathematical Games, Scientific American*

"This combined reference & text, *Fundamental Algorithms*, is the 1st volume of a planned 7-volume series. The series will provide a unified, readable, & theoretically sound summary of the present knowledge of computer programming techniques, plus a study of their historical development.

The point of view adopted by the author differs from many contemporary books about compute programming. The author does not try to teach the reader how to use somebody else's subroutines, but is concerned rather with teaching the reader how to write better subroutines himself.

A reader who is interested primarily in programming rather than in the associated mathematics may stop reading each section as soon as the mathematics become recognizably difficult. On the other hand, a mathematically oriented reader will find a wealth of interesting material.

As a reference the series provides valuable information for system programmers, analyst programmers, & others in the computer & related software industries. All 7 volumes of this series may also be used in senior or graduate courses such as: Information Structures, Computer Science, Combinatorial Mathematics, Computer-Oriented Finite Mathematics, or Fundamentals of Symbolic Machine Language Programming.

Among the areas covered in Vol. 1 are the representation of information inside a computer; the structural interrelations between data elements & how to deal with them efficiently; plus applications to simulation, numerical methods, software design, & other factors. Also included is an introduction to fundamental topics in discrete mathematics, of special importance in the study of computer programming techniques.

There are over 850 exercises, graded according to the level of difficulty from extremely simple questions to unsolved research problems. Answers are supplied for over 90% of the exercises. This enhances the value of the book for self-study, classroom use, & for reference. & it helps make it possible to organize the book so that it can be read by both mathematicians & non-mathematicians." 634 pages, 71 figures, (1968), $19.50

Fig. 28. Representation of polynomials using 4-directional links. Shaded areas of nodes indicate information irrelevant in the context considered.

Chapter 2

The Art of Computer Programming. Vol. 1: Fundamental Algorithms

This series of books is affectionately[1] *dedicated to the Type 650 computer once installed at Case Institute of Technology, in remembrance*[2] *of many pleasant*[3] *evenings.*

Preface

"Here is your book, the one your thousands of letters have asked us to publish. It has taken us years to do, checking & rechecking countless[4] *recipes*[5] *to bring you only the best, only the interesting, only the perfect. Now we can say, without a shadow of a doubt, that every single 1 of them, if you follow the directions to the letter, will work for you exactly as well as it did for us, even if you have never cooked before."* – McCall's Cookbook (1963)

"THE PROCESS of preparing programs for a digital[6] computer is especially attractive[7], not only because it can be economically[8] & scientifically[9] rewarding[10], but also because it can be an aesthetic[11] experience much like composing[12] poetry[13] or music. This book is the 1st volume of a multi-volume set of books that has been designed to train the reader in various skills that go into a programmer's craft[14].

The following chapters are *not* meant to serve as an introduction to computer programming; the reader is supposed to have had some previous experience. The prerequisites[15] are actually very simple, but a beginner requires time & practice in

[1]**affectionately** [adv] in a way that shows caring feelings & love for somebody.

[2]**remembrance** [n] [uncountable] the act or process of remembering an event in the past or a person who is dead.

[3]**pleasant** [a] (**pleasanter, pleasantest**) (**more pleasant** & **most pleasant** are more common) enjoyable, pleasing or attractive, OPPOSITE: **unpleasant**.

[4]**countless** [a] [usually before noun] very many; too many to be counted or mentioned.

[5]**recipe** [n] **1. recipe (for something)** a set of instructions that tells you how to cook something & the ingredients you need for it; **2. recipe for something** a method or an idea that seems likely to have a particular result, SYNONYM: **formula**.

[6]**digital** [a] **1.** using a system of receiving & sending information as a series of the numbers 1 & 0, showing that an electronic signal is there or is not there; connected with computer technology; **2.** (of clocks, watches, etc.) displaying only the appropriate numbers, rather than pointing to numbers from a larger set of numbers; other information displayed in this way; **3.** connected with a finger or the fingers of the hand.

[7]**attractive** [a] **1.** (of a person or an animal) pleasant to look at, especially in a sexual way; making an animal interested in a sexual way, OPPOSITE: **unattractive**; **2.** (of a thing or a place) pleasant to look at or be in, OPPOSITE: **unattractive**; **3.** having features or qualities that make something seem interesting & worth having, SYNONYM: **appealing**, OPPOSITE: **unattractive**; **4.** (*physics*) involving the force thta pulls things towards each other, OPPOSITE: **repulsive**.

[8]**economically** [adv] **1.** in a way that is connected with the trade, industry & development of wealth of a country, an area or a society; **2.** in a way that provides good service or value in relation to the amount of time or money spent.

[9]**scientifically** [adv] **1.** in a way that is connected with science; **2.** in a careful & organized way.

[10]**rewarding** [a] **1.** (of an activity) worth doing; that makes you happy because you think it is useful or important; **2.** producing a lot of money, SYNONYM: **profitable**.

[11]**aesthetic** [a] (*North American English also* **esthetic**) **1.** concerned with beauty & art & the understanding of beautiful things; **2.** beautiful to look at; [n] (*North American English also* **esthetic**) **1.** [countable] **aesthetic (of something)** a set of principles that express the aesthetic qualities & ideas of a particular artist or a particular group of artists, writers, etc.; **2.** (**aesthetics**) [uncountable] the branch of philosophy that studies the principles of beauty, especially in art.

[12]**compose** [v] **1.** (**be composed of something**) to be made or formed from several substances, parts of people; **2.** (not used in the progressive tenses) **compose something** to combine together to form a whole, SYNONYM: **make something up**; **3.** to write a piece of music; **4.** to write something, especially a poem.

[13]**poetry** [n] [uncountable] a collection of poems; poems in general, SYNONYM: **verse**.

[14]**craft** [n] **1.** [countable, uncountable] an activity involving a special skill at making things with your hands; **2.** [singular] all the skills needed for a particular activity; **3.** (plural **craft**) [countable] a boat or ship; [v] [usually passive] **craft something** to make something using a special skill, SYNONYM: **fashion**.

[15]**prerequisite** [n] [usually singular] something that must exist or happen before something else can happen or be done, SYNONYM: **precondition**.

order to understand the concept of a digital computer. The reader should possess:

a) Some idea of how a stored-program digital computer works; not necessarily the electronics[16], rather the manner[17] in which instructions[18] can be kept in the machine[19]'s memory[20] & successively[21] executed[22].

b) An ability to put the solutions to problems into such explicit[23] terms that a computer can "understand" them. (These machines have no common sense[24]; they do exactly[25] as they are told, no more & no less. This fact is the hardest concept to grasp[26] when one 1st tries to use a computer.)

c) Some knowledge of the most elementary[27] computer techniques, such as looping[28] (performing[29] a set of instructions[30] repeatedly[31]), the use of subroutines[32], & the use of indexed[33] variables[34].

[16]**electronics** [n] **1.** [uncountable] the branch of science & technology that studies electric currents in electronic equipment; **2.** [uncountable] the use of electronic technology; the making of electronic products; **3.** [plural] the electronic circuits & components used in electronic equipment.

[17]**manner** [n] **1.** [singular] the way that something is done or happens; **2.** [singular] the way that somebody behaves towards other people; **3.** (**manners**) [plural] behavior that is considered to be polite in a particular society or culture; **4.** (**manners (of somebody/something)**) [plural] the habits & customs of a particular group of people; **all manner of somebody/something** [idiom] many different types of people or things; **in the manner of somebody/something** [idiom] in a style that is typical of somebody/something.

[18]**instruction** [n] **1.** (**instructions**) [plural] detailed information on how to do or use something, SYNONYM: **direction**; **2.** [countable, usually plural] something that somebody tells you to do, SYNONYM: **order**; **3.** [countable] (*computing*) a code in a program that tells a computer to perform a particular operation; **4.** [uncountable] the act of teaching something to somebody.

[19]**machine** [n] **1.** (often in compounds) a piece of equipment with moving parts that is designed to do a particular job. The power used to work a machine may be electricity, steam, gas, etc. or human power; **2.** a group of people who operate in an efficient way within an organization.

[20]**memory** [n] (plural **memories**) **1.** [countable, uncountable] your ability to remember things; the part of your mind in which you store things that you remember; **2.** [uncountable] **in/within ... memory** the period of time that somebody is able to remember events; **3.** [countable] a thought of something that you remember from the past; **4.** [uncountable, countable] the part of a computer where data are stored; the amount of space in a computer for storing data; **5.** [uncountable] **memory (of somebody)** what is remembered about somebody after they have died; **from memory** [idiom] without reading or looking at notes; **in memory of somebody, to the memory of somebody** [idiom] intended to show respect & remind people of somebody who has died; **within/in living memory** [idiom] at a time, or during the time, that is remembered by people still alive.

[21]**successive** [a] [only before noun] following immediately one after the other, SYNONYM: **consecutive**.

[22]**execute** [v] **1.** [usually passive] to kill somebody, especially as a legal punishment; **2. execute something** to do a piece of work, perform a duty, put a plan into action, etc.; **3. execute something** (*computing*) carry out an instruction or program; **4. execute something** (*law*) to follow the instructions in a legal document; to make a document legally valid.

[23]**explicit** [a] **1.** saying something clearly & exactly; **2.** showing or referring to sex in a very obvious or detailed way.

[24]**common sense** [n] [uncountable] the ability to think about things in a practical way & make sensible decisions.

[25]**exactly** [adv] used to emphasize that something is correct in every way or in every detail, SYNONYM: **precisely**.

[26]**grasp** [v] **1.** to understand something completely; **2. grasp an opportunity** to take an opportunity without hesitating & use it; **3. grasp somebody/something** to take a firm hold of somebody/something, SYNONYM: **grip**; [n] [usually singular] **1.** a person's understanding of a subject; **2.** a firm hold of somebody/something or control over somebody/something; **3.** the ability to get or achieve something.

[27]**elementary** [a] **1.** connected with the 1st stages of a course of study, or the 1st years at school; **2.** of the most basic kind; **3.** very simple & easy.

[28]**loop** [n] **1.** a shape like a curve or circle made by a line curving right around; **2.** a piece of rope, wire, etc. in the shape of a curve or circle; **3.** a long, narrow piece of film or tape on which the pictures & sound are repeated continuously; **4.** (*computing*) a set of instructions that is repeated again & again until a particular condition is satisfied; **5.** a complete circuit for electrical current; **6.** (*British English*) a railway line or road that leaves the main track or road & then joins it again; [v] **1.** [transitive] **loop something + adv./prep.** to form or bend something into a loop; **2.** [intransitive] **+ adv./prep.** to move in a way that makes the shape of a loop; **loop the loop** [idiom] to fly or make a plane fly in a circle going up & down.

[29]**perform** [v] **1.** [transitive] **perform something** to do something, such as a piece of work, task or duty, SYNONYM: **carry something out**; **2.** [intransitive] **+ adv./prep.** to work or function well or badly. In this meaning, where there is no adverb or preposition, **perform** means 'perform well'.; **3.** [transitive, intransitive] **perform (something)** to entertain an audience by playing a piece of music, acting in a play, etc.

[30]**instruction** [n] **1.** (**instructions**) [plural] detailed information on how to do or use something, SYNONYM: **direction**; **2.** [countable, usually plural] something that somebody tells you to do, SYNONYM: **order**; **3.** [countable] (*computing*) a code in a program that tells a computer to perform a particular operation; **4.** [uncountable] the act of teaching something to somebody.

[31]**repeatedly** [adv] many times; again & again.

[32]**subroutine** [n] (also **subprogram**) (*computing*) a set of instructions which perform a task within a program.

[33]**index** [n] **1.** (plural **indexes**) **index (to something)** (in a book or set of books) an alphabetical list of names, subjects, etc. with the numbers of the pages on which they are mentioned; **2.** (**indexes, indices**) a number in a system or scale that represents the average value of particular prices, shares, etc. compared with a previous or standard value; **3.** (**indices**) **index of something** a sign or measure that something else can be judged by; **4.** (in compounds) a number that gives the value of a physical quality in terms of a standard formula; **5.** (usually **indices** [plural] *mathematics*) a small number written above another number to show how many times the other number must be multiplied by itself. In the equation $4^2 = 16$, the number 2 is an index.; **6.** (**indexes**) (*computing*) a list of items, each of which identifies a particular record in a computer file or database & contains information about its address; [v] **1. index something** to record names, subjects, etc. in an index; **2. index something** to provide an index to something; **3.** [usually passive] **index something (to something)** to link salaries, prices, etc. to the level of prices of food & other goods so that they both increase at the same rate.

[34]**variable** [n] **1.** an element or a feature that is likely to vary or change, OPPOSITE: **constant**; **2.** a property that is measured or observed in an experiment or a study; a property that is adjusted in an experiment, OPPOSITE: **constant**; **3.** (*mathematics*) a quantity in a calculation that can take any of a set of different numerical values, represented by a symbol such as x, OPPOSITE: **constant**; [a] **1.** often changing; likely to change, SYNONYM: **fluctuating**, OPPOSITE: **constant**; **2.** not the same in all parts or cases; not having a fixed pattern, SYNONYM: **diverse**. When **variable** is used to describe the quality of something, the tone is slightly disapproving, meaning that some parts of it are good & some are bad, SYNONYM: **inconsistent, mixed**, OPPOSITE: **consistent, uniform**; **3.** that can be changed to meet different needs or suit different conditions, OPPOSITE: **fixed**; **4.** (*mathematics*) (of a quantity) that can take any of a set of different numerical values, represented by a symbol such as x, OPPOSITE: **constant**.

d) A little knowledge of common computer jargon[35] – "memory," "registers[36]," "bits[37]," "floating[38] point," "overflow[39]," "software[40]." Most words not defined in the text are given brief definitions[41] in the index at the close of each volume.

These 4 prerequisites can perhaps be summed up into the single requirement that the reader should have already written & tested at least, say, 4 programs for at least 1 computer.

I have tried to write this set of books in such a way that it will fill several needs. In the 1st place, these books are reference works that summarize[42] the knowledge that has been acquired[43] in several important fields. In the 2nd place, they can be used as textbooks[44] for self-study[45] or for college[46] courses[47] in the computer & information[48] sciences[49]. To meet both of these objectives, I have incorporated[50] a large number of exercises into the text & have furnished[51] answers for most of them. I have also made an effort to fill the page with facts rather than with vague[52], general commentary[53].

This set of books is intended[54] [55] for people who will be more than just casually[56] interested[57] in computers, yet it is by

[35]**jargon** [n] [uncountable] (*often disapproving*) words or expressions that are used by a particular profession or group of people, & are difficult for others to understand.

[36]**register** [v] **1.** [transitive, intransitive] to record the name of somebody/something on an official list; **2.** [transitive] **register something** to make your opinion known officially or publicly; **3.** [intransitive] + **noun** (of a measuring instrument) to show or record an amount; **4.** [transitive] **register something** to achieve a particular score or result; **5.** [transitive] **register something** t notice something & remember it; [n] **1.** [countable] **register (of something)** an official list or record of names or items; a book that contains such a list; **2.** [countable, uncountable] (*linguistics*) the level & style of a piece of writing or speech, that is usually appropriate to the situation that it is used in; **3.** [countable] (*computing*) (in electronic devices) a location in a store of data, used for a particular purpose & with quick access time.

[37]**bit** [n] **1.** [countable] the smallest unit of information used by a computer; **2.** (**a bit**) [singular] (used as an adverb) (*especially British English, rather informal*) rather; **3.** (**a bit**) [singular] (used as an adverb) a small amount; a little; **4.** [countable] **bit of something** (*especially British English, rather informal*) a small piece or part of something; **5.** [countable] **bit of something** (*especially British English, rather informal*) a small amount of something; **bit by bit** [idiom] (*rather informal*) a piece or part at a time; gradually; **every bit as good, bad, etc. (as somebody/something)** [idiom] (*rather informal*) just as good, bad, etc.; equally good, bad, etc.

[38]**float** [v] **1.** [intransitive] + **adv./prep.** to move slowly on or in water or in the air; **2.** [intransitive] to stay on or near the surface of a liquid & not sink; **3.** [transitive] **float something (+ adv./prep.)** to make something move on or near the surface of a liquid; **4.** [transitive] **float something** to suggest an idea or a plan for other people to consider; **5.** [transitive] **float something** to sell shares in a company or business to the public for the 1st time; **6.** [transitive, intransitive] (*economics*) if a government floats its country's money or allows it to float, it allows its value to change freely according to the value of the money of other countries.

[39]**overflow** [v] **1.** [intransitive, transitive] to be so full that the contents go over the sides; **2.** [intransitive] **overflow (with something)** (of a place) to have too many people in it; **3.** [intransitive, transitive] **overflow (into something)** | **overflow (something)** to spread beyond the limits of a place or container that is too full; [n] **1.** [uncountable, singular] a number of people or things that do not fit into the space available; **2.** [uncountable, singular] the action of liquid flowing out of a container, etc. that is already full; the liquid that flows out; **3.** (also **overflow pipe**) [countable] a pipe that allows extra liquid to flow away safely when a container is full; **4.** [countable, usually singular] (*computing*) a fault that happens because a number or data item is too large for the computer to represent it exactly.

[40]**software** [n] [uncountable] the programs used by a computer for doing particular jobs.

[41]**definition** [n] **1.** [countable] an exact statement or description of the nature, extent or meaning of something; **2.** [countable] a statement of the exact meaning of a word or phrase, especially in a dictionary; **3.** [uncountable] the action or process of stating the exact meaning of a word or phrase; **by definition** [idiom] as a result of what something is.

[42]**summarize** [v] (*British English also* **summarise**) [transitive, intransitive] **summarize (something)** to give a summary of something.

[43]**acquire** [v] **1. acquire something** to learn or develop a skill, habit or quality; **2. acquire something** to obtain something by buying or being given it; **3. acquire something** to come to have a particular reputation.

[44]**textbook** [n] (*North American English also* **text**) a book that teaches a particular subject & that is used especially in schools & colleges.

[45]**self-study** [n] [uncountable] the activity of learning about something without a teacher to help you; [a] designed to help students to learn about something without a teacher to help them.

[46]**college** [n] **1.** [countable, uncountable] (often in names) (in the US) a university where students can study for a degree after they have left school; **2.** [countable, uncountable] (often in names) (in Britain) a place where students go to study or to receive training after they have left school; **3.** [countable, uncountable] (often in names) 1 of the separate institutions that come British universities, such as Oxford & Cambridge, are divided into; **4.** [countable, uncountable] (often in names) (in the US) 1 of the main divisions of some large universities; **5.** [countable + singular or plural verb] (usually in names) an organized group of professional people with special interests, duties or powers.

[47]**course** [n] **1.** [countable] a series of classes or lectures on a particular subject; **2.** [countable] (*especially British English*) a period of study at a college or university that leads to an exam or a qualification; **3.** [singular] the way that something develops or should develop; **4.** (also **course of action**) [countable] a way of acting in or dealing with a particular situation; **5.** [countable, usually singular] the general direction in which somebody's ideas or actions are moving; **6.** [uncountable, countable, usually singular] a direction or route follows by a ship or an aircraft, or by another moving object; **7.** [countable] **course (of something)** a series of medical treatments.

[48]**information** [n] [uncountable] **1.** facts or details about somebody/something that are provided or learned; **2.** data that are stored, analyzed or passed on by a computer; **3.** what is shown by a particular arrangement of things.

[49]**information scienec** [n] (also **informatics**) [uncountable] (*computing*) the study of processes for storing & obtaining information.

[50]**incorporate** [v] **1.** to include something so that it forms a part of something; **2.** [usually passive] (*business*) to create a legally recognized company.

[51]**furnish** [v] **1. furnish something (with something)** to put furniture in a house, room, etc.; **2.** (*formal*) to supply something; to supply or provide somebody/something with something.

[52]**vague** [a] (**vaguer, vaguest**) not having or giving enough information or details about something.

[53]**commentary** [n] (plural **commentaries**) **1.** [countable] **commentary (on something)** a written explanation or discussion of something such as a theory or book; **2.** [countable, uncountable] **commentary (on something)** a criticism or discussion of something.

[54]**intend** [v] **1.** to have a plan, result or purpose in your mind when you do something; **2. intend something as something** | **intend something to be something** to plan that something should have a particular meaning or use.

[55]**intended** [a] **1.** meant or designed to be something or to be used by somebody; **2.** [only before noun] that you are trying to achieve or reach.

[56]**casual** [a] **1.** [usually before noun] without paying attention to detail; **2.** [usually before noun] not showing much care or thought; **3.** [usually before noun] (of a relationship) lasting only a short time & without deep affection; **4.** [usually before noun] (*British English*) (of work) not permanent; not regular; **5.** not formal; **6.** [only before noun] happening by chance; doing something by chance.

[57]**interested** [a] **1.** [not usually before noun] giving your attention to something because you enjoy finding out about it or doing it; showing

no means only for the computer specialist[58]. Indeed, 1 of my main goals has been to make these programming[59] techniques more accessible[60] to the many people working in other fields who can make fruitful use of computers, yet who cannot afford[61] the time to locate[62] all of the necessary information that is buried[63] in technical[64] journals[65].

We might call the subject of these books "nonnumerical analysis." Computers have traditionally[66] been associated[67 68] with the solution[69] of numerical[70] problems such as the calculation[71] of the roots[72] of an equation[73], numerical interpolation[74] & integration[75], etc., but such topics are not treated here except in passing[76]. Numerical computer programming

interest in something & finding it exciting; **2.** [usually before noun] in a position to gain from a situation or be affected by it.

[58]**specialist** [n] **1.** a doctor who has specialized in a particular area of medicine; **2. specialist (in something)** a person who is an expert in a particular area of work or study; [a] [only before noun] **1.** connected with a doctor who has specialized in a particular area of medicine; **2.** having or involving detailed knowledge of a particular topic or area of study.

[59]**programming** [n] [uncountable] **1.** the process of writing & testing programs for computers; **2. programming (of something)** the activity of planning which television or radio programmes to broadcast; the programmes that are broadcast; **3.** factors, ranging from genetic to social, that instruct a person or animal to behave in a certain way.

[60]**accessible** [a] **1.** that can be reached, entered, used or obtained; **2.** easy to understand.

[61]**afford** [v] **1.** [no passive] (usually used with *can, could* or *be able to*, especially in negative sentences or questions) to have enough money or time to be able to buy or to do something; **2.** [no passive] **afford to do something** (usually used with *can* or *could*, especially in negative sentences & questions) if you say that you cannot afford to do something, you mean that you should not do it because it will cause problems for you if you do; **3.** (*formal*) to provide somebody with something.

[62]**locate** [v] **1.** [transitive] **locate somebody/something** to find the exact position of somebody/something; **2.** [transitive] **locate something + adv./prep.** to put or build something in a particular place; **3.** [intransitive] + **adv./prep.** (*especially North American English*) to start a business in a particular place.

[63]**bury** [v] **1. bury somebody/something** to place something in the ground, especially a dead body in a grave; **2.** [often passive] **bury somebody/something** to cover something with soil, rocks, leaves, etc.; **3. buy something** to ignore or hide a feeling, a mistake, etc.

[64]**technical** [a] **1.** [usually before noun] connected with the use of science or technology; involving the use of machines; **2.** [usually before noun] connected with a particular type of activity, or the skills & processes needed for it; **3.** [usually before noun] (of language, writing or ideas) requiring knowledge & understanding of a particular subject; **4.** connected with the details of a law or set of rules.

[65]**journal** [n] **1.** a newspaper or magazine that deals with a particular subject or profession; **2.** a written record of the things you do or see every day.

[66]**traditionally** [adv] **1.** according to what has always or usually happened in the past; **2.** according to the beliefs, customs or way of life that have existed for a long time among a particular group of people; according to what is believed.

[67]**associate** [v] **1.** [transitive] to make a connection between people or things in your mind, SYNONYM: **connect, relate, link**, OPPOSITE: **dissociate**; **2.** [intransitive] **associate with somebody** to spend time with somebody, especially somebody that others do not approve of; **3.** [transitive] **associate yourself with something** to show that your support or agree with something, OPPOSITE: **dissociate**; [n] **1.** a person that you work with, do business with or spend a lot of time with; **2.** (**Association**) **associate (of something)** an associate member of an organization; [a] [only before noun] **1.** (often in titles) of a lower rank; having fewer rights in a particular profession or organization; **2.** joined or connected with a profession or an organization.

[68]**associated** [a] **1.** if 1 thing is associated with another, the 2 things are connected because they happen together or 1 thing causes the other, SYNONYM: **connected**; **2.** if a person is associated with a person, organization or idea, they support it; **3.** (of a company) connected or joined with another company or companies.

[69]**solution** [n] **1.** [countable] a way of solving a problem or dealing with a difficult situation, SYNONYM: **answer**; **2.** [countable] **solution (to/for/of something)** an answer to a problem in mathematics; **3.** [countable, uncountable] a liquid in which a substance has been dissolved, so that the substance has become part of the liquid; the state of being dissolved in a liquid.

[70]**numerical** [a] (also *less frequent* **numeric**) [usually before noun] connected with numbers; expressed in numbers.

[71]**calculation** [n] [countable, uncountable] **1.** the act or process of using numbers to find out an amount; **2.** the process of using your judgment to decide what the results would be of doing something.

[72]**root** [n] **1.** [countable] the part of a plant that grows under the ground & absorbs water & minerals that it sends to the rest of the plant; **2.** [countable, usually singular] **root of something** the main cause of something, such as a problem or difficult situation; **3.** [countable, usually plural] the basis of something; **4.** (**roots**) [plural] the feelings or connections that you have with a place because you have lived there or your family came from there; **5.** [countable] (*linguistics*) the part of a word that has the main meaning & that its other forms are based on; a word that other words are formed from; **6.** [countable] **root (of something)** (*mathematics*) a quantity which, when multiplied by itself a particular number of times, produces another quantity; **root & branch** [idiom] thorough & complete; **take root** [idiom] **1.** (of a plant) to develop roots; **2.** (of an idea) to become widely accepted; [v] [intransitive] (of a plant) to grow roots; **root something/somebody out** [phrasal verb] to find a person or thing that is causing a problem & remove or get rid of them.

[73]**equation** [n] **1.** [countable] (*mathematics*) a statement showing that 2 amounts or values are equal; **2.** [countable] (*chemistry*) a statement using symbols to show the changes that happen in a chemical reaction; **3.** [uncountable, singular] the act of making something equal or considering something as equal; **4.** [countable, usually singular] a situation in which several factors must be considered & dealt with.

[74]**interpolation** [n] [uncountable, countable] **1.** (*formal*) a remark that interrupts a conversation; the act of making a remark that interrupts a conversation; **2.** (*formal*) a thing that is added to a piece of writing; the act of adding something to a piece of writing, SYNONYM: **insertion**; **3.** (*mathematics*) the act of adding a value into a series by calculating it from surrounding known values.

[75]**integration** [n] **1.** [uncountable, countable] the act or process of combining 2 or more things so that they work together; **2.** [uncountable] the act or process of mixing people who have previously been separated, usually because of color, race or religion; **3.** [uncountable, countable] **integration (of something)** (*mathematics*) the process of finding an integral or integrals.

[76]**passing** [n] [uncountable] **1. the passing of time/the years** the process of time going by; **2. passing (of somebody/something)** the fact of something ending or of somebody dying; **3. the passing of something** the act of making something become a law; **in passing** [idiom] done or said while you are giving your attention to something else.

is an extremely[77] interesting & rapidly[78] expanding[79] field, & many books have been written about it. Since the early 1960s, however, computers have been used even more often for problems in which numbers occur only by coincidence[80]; the computer's decision-making[81] capabilities[82] are being used, rather than its ability[83] to do arithmetic[84] [85]. We have some use for addition[86] & subtraction[87] in nonnumerical problems, but we rarely feel any need for multiplication[88] & division[89]. Of course, even a person who is primarily[90] concerned[91] with numerical computer programming will benefit[92] from a study of the nonnumerical techniques, for they are present in the background of numerical programs as well.

The results of research in nonnumerical analysis are scattered[93] [94] throughout numerous[95] technical journals. My approach has been to try to distill[96] this vast literature by studying the techniques that are most basic, in the sense that they can be applied to many types of programming situations[97]. I have attempted[98] [99] to coordinate[100] the ideas into more or less of a "theory," as well as to show how the theory applies to a wide variety[101] of practical problems.

Of course, "nonnumerical analysis" is a terribly[102] negative name for this field of study; it is much better to have a positive,

[77]**extremely** [adv] (usually with adjectives & adverbs) to a very high degree.

[78]**rapidly** [adv] in a short period of time or at a fast rate.

[79]**expand** [v] **1.** [intransitive, transitive] to become greater in size, number or importance; to make something greater in size, number or importance; **2.** [transitive] **expand something** to write something such as a scientific formula in a longer form; **expand on/upon something** [phrasal verb] to add more details & give more information about something.

[80]**coincidence** [n] **1.** [countable, uncountable] the fact of 2 things happening at the same time by chance, often in a surprising way; **2.** [singular, uncountable] **coincidence of A with/& B** the fact of things being present in the same place at the same time; **3.** [singular, uncountable] **coincidence of something** the fact of 2 or more opinions, etc. being the same.

[81]**decision-making** [n] (also **decision making**) [uncountable] the process of deciding about something important, especially in a group of people or in an organization.

[82]**capability** [n] (plural **capabilities**) **1.** [countable, uncountable] the ability or qualities necessary to do something; **2.** [countable] the power or weapons that a country has for war or for military action.

[83]**ability** [n] (plural **abilities**) **1.** [singular] the fact that somebody/something is able to do something, OPPOSITE: **inability**; **2.** [uncountable, countable] a level of skill or intelligence.

[84]**arithmetic** [n] [uncountable] **1.** the type of mathematics that deals with the use of numbers in counting & calculation; **2.** the use of numbers in counting & calculation; [a] (*mathematics*) **1. arithmetic progression/series** a series in which the interval between each term & the next remains constant; **2. arithmetic mean = mean**; **3. = arithmetical**.

[85]**arithmetical** [a] (also **arithmetic**) connected with arithmetic.

[86]**addition** [n] **1.** [uncountable, countable] the act of adding something to something else, OPPOSITE: **removal**; **2.** [countable] **addition (to something)** a thing that is added to something else; **3.** [uncountable, countable] the process of adding 2 or more numbers together to find their total; **in addition (to somebody/something)** [idiom] used to introduce a new fact or argument.

[87]**subtraction** [n] [uncountable, countable] the process of taking a number or amount away from another number or amount.

[88]**multiplication** [n] [uncountable] **1.** the act or process of multiplying 1 number by another, OPPOSITE: **division**; **2.** the process of increasing very much in number or amount.

[89]**division** [n] **1.** [uncountable, countable, usually singular] the process or result of dividing into separate parts; the process or result of dividing something or sharing it out; **2.** [uncountable] the process of providing 1 number by another; **3.** [countable, usually plural, uncountable] a disagreement or difference in people's opinions of ways of life, especially between members of a society or an organization; **4.** [countable] **division (of something)** a part of something into which it is divided; **5.** [countable + singular or plural verb] (abbr., **Div.**) a large & important unit or section of an organization.

[90]**primarily** [adv] mainly, SYNONYM: **chiefly**.

[91]**concerned** [a] **1.** worried & feeling concern about something; **2.** interested in something; **as/so far as somebody/something is concerned, as/so far as somebody/something goes** [idiom] usd to give facts or an opinion about a particular aspect of something.

[92]**benefit** [n] **1.** [countable, uncountable] a helpful & useful effect that something has; an advantage that something provides; **2.** [uncountable, countable] (*British English*) money provided by the government to people who need financial help because they are unemployed, sick, etc.; **give somebody the benefit of the doubt** [idiom] to accept that somebody has told the truth or has not done something wrong because you cannot prove that they have not told the truth/have done something wrong; [v] **1.** [intransitive] to be in a better position because of something; **2.** [transitive] **benefit somebody/something** to be useful or provide an advantage to somebody/something.

[93]**scatter** [v] **1.** [intransitive, transitive] (*physics*) to change direction or spread in many directions; to make something change direction or spread in many directions; **2.** [transitive, often passive] **scatter something (on/over/around something)** to throw or drop things in different directions so that they cover an area; **3.** [transitive, often passive] to be found spread over an area rather than all together; **4.** [intransitive, transitive] (of people or animals) to move very quickly in different directions; to make people or animals do this, SYNONYM: **disperse**; [n] (also **scattering**) [singular] **scatter (of something)** a small amount of something or a small number of people or things spread over an area.

[94]**scattered** [a] **scattered (throughout/across/around something)** spread far apart over a wide area, SYNONYM: **dispersed**.

[95]**numerous** [a] [usually before noun] existing in large numbers, SYNONYM: **many**.

[96]**distil** [v] (*North American English also* **distill**) **1. distil something (from something)** to make a liquid pure by heating it until it becomes a gas, then cooling it & collecting the drops of liquid that form; **2. distil something** to make something such as a strong alcoholic drink in this way; **3. distil something (from/into something)** to get the essential meaning or ideas from thoughts, information or experiences.

[97]**situation** [n] **1.** all the circumstances & things that are happening at a particular time & in a particular place; **2.** the area or place where something is located.

[98]**attempt** [n] **1.** [countable, uncountable] an act of trying to do something difficult, often with no success; **2.** [countable] an act of trying to kill somebody; [v] to try to do or provide something, especially something difficult.

[99]**attempted** [a] [only before noun] (of a crime, etc.) that somebody has tried to do but without success.

[100]**coordinate** [v] (*British English also* **co-ordinate**) to organize the different parts of an activity & the people involved in it so that it works well; **coordinate with somebody** [phrasal verb] to reach an agreement with other people about how to work together effectively; [n] (*British English also* **co-ordinate**) 1 of the numbers or letters used to fix the position of a point on a map or graph.

[101]**variety** [n] (plural **varieties**) **1.** [singular] **variety (of something)** a number or range of different things of the same general type; **2.** [uncountable] the quality of not being the same in all parts or not doing the same thing all the time, SYNONYM: **diversity**; **3.** [countable] a type of a thing, e.g. a plant or language, that is different from others in the same general group. In biology, a **variety** is a category below a **species** & **subspecies**, used especially to describe plants.

[102]**terribly** [adv] **1.** (*especially British English*) very; **2.** very much; very badly.

descriptive[103] term that characterizes[104] the subject. "Information processing[105]" is too broad[106] a designation[107] for the material[108] I am considering, & "programming techniques" is too narrow[109]. Therefore I wish to propose[110] *analysis*[111] *of algorithms*[112] as an appropriate[113] name for the subject matter covered in these books. This name is meant to imply[114] "the theory of the properties of particular computer algorithms."

The complete set of books, entitled *The Art of Computer Programming*, has the following general outline:

- *Vol. 1: Fundamental*[115] *Algorithms*
  - Chap. 1. Basic Concepts
  - Chap. 2. Information Structures[116]
- *Vol. 2. Seminumerical Algorithms*
  - Chap. 3. Random[117] Numbers
  - Chap. 4. Arithmetic

[103]**descriptive** [a] **1.** describing what something is like, rather than saying what it should be like or what category it belongs to; **2.** saying or showing clearly what something is like; giving a clear account of something.

[104]**characterize** [v] (*British English also* **characterise**) **1.** [usually passive] **characterize something** to be the most typical or most obvious quality or feature of something, SYNONYM: **typify**; **2.** [usually passive] **characterize something** to be the feature or quality that makes something different from similar things, SYNONYM: **distinguish**; **3.** [often passive] **characterize somebody/something (as something)** to describe something/somebody in a particular way.

[105]**process** [n] **1.** a series of actions that are taken in order to achieve a particular result; **2.** a series of things that happen, especially ones that result in natural changes; **3.** a method of doing or making something, especially one that is used in industry; **in process** [idiom] being done; continuing; **in the process** [idiom] while doing something else, especially as a result that is not intended; **in the process of (doing) something** [idiom] in the middle of doing something that takes some time to do; [v] **1.** to treat raw material, food, etc. in order to change it, preserve it, etc.; **2. process something** to deal officially with something such as a document, application or request; **3. process something** to deal with information by performing a series of operations on it, especially using a computer; **processing** [n] [uncountable].

[106]**broad** [a] (**broader, broadest**) **1.** wide, OPPOSITE: **narrow**; **2.** including a great variety of things, OPPOSITE: **narrow**; **3.** [only before noun] general; not detailed; **4.** with most people agreeing about something in a general way; **5.** covering a wide area.

[107]**designation** [n] **1.** [uncountable, countable] the fact of describing somebody/something as having a particular character, status or purpose; **2.** [countable] a way of naming or describing somebody/something; a name or label.

[108]**material** [n] **1.** [countable, uncountable] a substance from which a thing is or can be made; a substance with a particular quality; **2.** [uncountable] information or ideas used in books or other work; **3.** [countable, usually plural, uncountable] things that are needed in order to do a particular activity, SYNONYM: **resource**; **4.** [uncountable, countable] cloth used for making clothes, etc., SYNONYM: **cloth, fabric**; [a] **1.** [only before noun] connected with money & possessions rather than with the needs of the mind or spirit, OPPOSITE: **spiritual**; **2.** [only before noun] connected with the physical world rather than with the mind or spirit, OPPOSITE: **spiritual**; **3.** important & needing to be considered. In law, **material** is used to describe evidence or facts that are important, especially when these facts might have an effect on the result of a case.

[109]**narrow** [a] (**narrower, narrowest**) **1.** measuring a short distance from 1 side to the other, especially in relation to length, OPPOSITE: **broad, wide**; **2.** limited in extent, amount or variety, SYNONYM: **restricted**, OPPOSITE: **wide**; **3.** limited in range & unwilling or unable to accept opinions that are different from yours, OPPOSITE: **broad**; **4.** strict in meaning, OPPOSITE: **broad**; **5.** [usually before noun] only just achieved; [v] [intransitive, transitive] **1.** to become or make something more limited in extent, amount or variety; **2.** to become or make something less wide; **narrow something down (to something)** [phrasal verb] to reduce the number of possibilities or choices.

[110]**propose** [v] **1.** to suggest a plan or an idea for people to consider & decide on; **2.** to suggest an explanation of something for people to consider.

[111]**analysis** [n] (plural **analyses**) **1.** [uncountable, countable] the detailed study or examination of something in order to understand more about it; the result of the study; **2.** [uncountable, countable] a careful examination of a substance in order to find out what it consists of; **3.** [uncountable] = **psychoanalysis**; **in the final/last analysis** [idiom] used to say what is most important after everything has been discussed or considered.

[112]**algorithm** [n] a process or set of rules to be followed when solving a particular problem, especially by a computer.

[113]**appropriate** [a] suitable, acceptable or correct for the particular circumstances; [v] **1. appropriate something** to start to use something that belongs to a different time, place or culture; **2. appropriate something** to take somebody's idea, property or money for your own use, especially without permission; **3. appropriate something** to take something from somebody, legally & often by force, SYNONYM: **seize**; **4. appropriate something** to take or give something, especially money, for a particular purpose.

[114]**imply** [v] **1.** to make it seem likely that something is true or exists, SYNONYM: **suggest**; **2.** to suggest that something is true or that you feel or think something, without saying so directly; **3.** (of a fact or event) to suggest something as a likely or necessary result.

[115]**fundamental** [a] **1.** serious & very important; affecting the most central & important parts of something, SYNONYM: **basic**; **2.** forming the necessary basis of something, SYNONYM: **essential**.

[116]**structure** [n] **1.** [uncountable, countable] the way in which the parts of something are connected together, arranged or organized; a particular arrangement of parts; **2.** [countable] a thing that is made of several parts arranged in a particular way, e.g. a building; **3.** [uncountable, countable] the state of being well organized or planned with all the parts linked together; a careful plan; [v] [often passive] to arrange or organize something into a system or pattern.

[117]**random** [a] **1.** [usually before noun] done, chosen, etc. so that all possible choices have an equal chance of being considered; **2.** without any regular pattern; [n] **at random** [idiom] without deciding in advance what is going to happen; without any regular pattern.

- *Vol. 3. Sorting*[118] *& Searching*[119]
    - Chap. 5. Sorting
    - Chap. 6. Searching
- *Vol. 4. Combinatorial Algorithms*
    - Chap. 7. Combinatorial Searching
    - Chap. 8. Recursion[120]
- *Vol. 5. Syntactical*[121] [122] *Algorithms*
    - Chap. 9. Lexical[123] Scanning[124]
    - Chap. 10. Parsing[125]

Vol. 4 deals with such a large topic, it actually represents[126] several[127] separate[128] books (Vols. 4A, 4B, & so on). 2 additional[129] volumes on more specialized[130] topics[131] are also planned: Vol. 6, *The Theory of Languages* (Chap. 11); Vol. 7, *Compilers*[132] (Chap. 12).

I started out in 1962 to write a single book with this sequence[133] of chapters, but I soon found that it was more important

[118]**sort** [n] a group or type of people or things that have similar characteristics; a particular variety or type, SYNONYM: **kind**; **of sorts, of a sort, a sort of something** [idiom] used when you are saying that something is the thing mentioned, but only to a certain degree, or that something is not a good example of something; [v] [often passive] to arrange things in groups or in a particular order according to their type or to what they contain; **sort something out** [phrasal verb] (*rather informal*) to deal with something successfully; to stop something from being a problem. **Sort something out** is much more common in general English; in academic English, a more formal word or expression such as **resolve** or **deal with** is more common.; **sort something out (from something)** [phrasal verb] to separate something from something else; to recognize something as different from something else; **sort through something** [phrasal verb] to look at or consider different things in order to find the most useful or important ones.

[119]**search** [n] **1.** an attempt to find somebody/something, especially by looking carefully for them/it; **2.** an act of looking for information in a computer database, on the Internet, etc.; an act of looking for information in official documents; [v] **1.** [intransitive, transitive] to look carefully for something/somebody; to examine a particular place when looking for something/somebody; **2.** [transitive] **search somebody (for something)** (especially of the police) to examine somebody's clothes, etc. in order to find something that they may be hiding; **3.** [intransitive, transitive] to look in a computer database, on the Internet, etc. in order to find information; to look at official documents in order to find information; **4.** [intransitive] **search for something** to think carefully, especially in order to find the answer to a problem; **search something/somebody out** [phrasal verb] to look for something/somebody until you find them.

[120]**recursion** [n] [uncountable] (*mathematics*) the process of repeating a function, each time applying it to the result of the previous stage.

[121]**syntactic** [a] (*linguistics*) connected with syntax

[122]**syntactically** [adv] (*linguistics*) in a way that is connected with syntax.

[123]**lexical** [a] [usually before noun] (*linguistics*) connected with the words of a language.

[124]**scan** [v] **1.** [transitive, intransitive] to look at every part of something carefully, especially because you are looking for a particular thing or person; **2.** [transitive] **scan something (for something)** to look quickly at something in order to find relevant information; **3.** [transitive, usually passive] to use a machine to get an image of the inside of somebody's body & show it on a computer screen; **4.** [transitive, usually passive] to use a machine that uses light or another means to read data or to produce an image of an object or document in digital form; [n] **1.** a medical test in which a machine produces an image of the inside of a person's body & shows it on a computer screen; **2. scan (of something)** the use of a machine to read data or produce an image of something; **3. scan (of something)** the act of looking quickly through something written or printed, usually in order to find something or to get a general idea about what is in it.

[125]**parse** [v] **parse something** to divide a sentence into parts & describe the grammar of each word or part.

[126]**represent** [v] **1. + noun** *linking verb* (not used in the progressive tenses) to be something; to be equal to something, SYNONYM: **constitute**; **2.** (not used in the progressive tenses) **represent something** to be a symbol or sign of something, SYNONYM: **symbolize**; **3.** [no passive] **represent something** to be typical of something; **4.** to show or describe somebody/something in a particular way, SYNONYM: **present**; **5.** (not used in the progressive tenses) **represent somebody/something** to include a particular type or number of people or things; **6. be represented + adv./prep.** to be present to a particular degree; **7.** to act or speak for somebody/something; to attend or take part in an event on behalf of somebody; **8. represent somebody/something** to show somebody/something, especially in a picture or diagram, SYNONYM: **depict**; **9. represent something** to say or suggest something that you want somebody to believe or pay attention to.

[127]**several** [determiner, pronoun] more than 2 but not very many.

[128]**separate** [a] **1.** forming a unity by itself; not joined to, touching or close to something/somebody else; **2.** [usually before noun] different; not connected; **go your (own) separate ways** [idiom] to end a relationship with somebody; [v] **1.** [intransitive, transitive] to divide into different parts or groups; to divide things into different parts or groups; **2.** [intransitive, transitive] to move apart; to make people or things move apart; **3.** [transitive] to be between 2 people, areas, countries, etc. so that they are not touching or connected; **4.** [intransitive] to stop living together as a couple with your husband, wife or partner; **5.** [transitive] to make somebody/something different in some way from somebody/something else, SYNONYM: **divide**; **separate out | separate something out** [phrasal verb] to divide into different parts; to divide something into different parts.

[129]**additional** [a] more than was 1st mentioned or is already present or is usual, SYNONYM: **extra**.

[130]**specialized** [a] (*British English also* **specialised**) **1.** connected with a particular area of work or study; **2.** requiring or involving detailed & particular knowledge or training; **3.** designed or developed for a particular purpose or area of knowledge.

[131]**topic** [n] a particular subject that is studied, written about or discussed.

[132]**compiler** [n] **1.** a person who complies something; **2.** (*computing*) a program that translates instructions from 1 computer language into another for a computer to understand.

[133]**sequence** [n] **1.** [countable] **sequence (of something)** a set of connected events, actions, numbers, etc. that have a particular order; **2.** [countable, uncountable] the order that connected events, actions, etc. happen in or should happen it; **3.** [countable] (*biology*) the order in which a set of genes or parts of molecules are arranged; **4.** [countable] a part of a film that deals with 1 subject or topic or consists of 1 scene; **5.** [countable] (*mathematics*) an ordered list of numbers; [v] **1. sequence something** (*biology*) to identify the order in which a set of genes or parts of molecules are arranged; **2. sequence something** to arrange things in a sequence.

to treat[134] the subjects in depth[135] rather than to skim[136] over them lightly[137]. The resulting length of the text has meant that each chapter by itself contains more than enough material for a 1-semester[138] college course; so it has become sensible[139] to publish[140] the series[141] in separate volumes. I know that it is strange[142] to have only 1 or 2 chapters in an entire[143] book, but I have decided to retain[144] the original[145] chapter numbering in order to facilitate[146] cross references[147]. A shorter version of Vols. 1–5 is planned, intended specifically[148] to serve[149] as a more general reference &/or text for undergraduate computer courses; its contents[150] will be a subset[151] of the material in these books, with the more specialized information omitted[152]. The same chapter numbering will be used in the abridged[153] edition[154] as in the complete work.

The present volume may be considered as the "intersection" of the entire set, in the sense that it contains basic material that is used in all the other books. Vols. 2–5, on the other hand, may be read independently[155] of each other. Vol. 1 is not only a reference book to be used in connection with the remaining[156] volumes; it may also be used in college courses or for self-study as a text on the subject of *data structures* (emphasizing the material of Chap. 2), or as a text on the subject of *discrete mathematics* (emphasizing the material of Sects. 1.1, 1.2, 1.3.3, & 2.3.4), or as a text on the subject of *machine-language programming* (emphasizing the material of Sects. 1.3 & 1.4).

[134]**treat** [v] **1.** to behave in a particular way when dealing with somebody/something; **2.** to consider something in a particular way; **3.** [often passive] to deal with or discuss a subject in a piece of writing or speech; **4.** [often passive] to give medical care or attention to a person, an illness or an injury; **5.** to use a chemical substance or process to clean, protect, preserve, etc. something.

[135]**depth** [n] **1.** [countable, uncountable] the distance from the top or surface to the bottom of something; how deep something is; **2.** [uncountable] **depth (of something)** the fact of having or providing a lot of information or knowledge; **3.** [uncountable] **depth (of something)** the fact of being very important or serious; **4.** [uncountable] the quality in an image that makes it appear not to be flat; **the depths of something** [idiom] **1.** the deepest part of something; **2.** the most serious or extreme part of something; **in depth** [idiom] in a detailed & thorough way.

[136]**skim** [v] **1.** [intransitive, transitive] to read something quickly in order to find a particular point or the main points; **2.** [transitive] **skim something** to remove a substance such as fat from the surface of a liquid; **3.** [intransitive, transitive, no passive] to move quickly & lightly over a surface, not touching it or only touching it occasionally.

[137]**lightly** [adv] **1.** gently; with very little force or effort; **2.** to a small degree; not much; **3.** without being seriously considered.

[138]**semester** [n] 1 of the 2 periods that the school or college year is divided into in some countries, e.g. the US.

[139]**sensible** [a] **1.** (of actions, plans, decisions, etc.) done or chosen with good judgment based on reason & experience rather than emotion; practical; **2.** (of people) able to make good judgments based on reason & experience rather than emotion.

[140]**publish** [v] **1.** [transitive] **publish something** to produce a book, magazine, etc. & sell it to the public; **2.** [transitive] to print a letter, an article, etc. in a newspaper or magazine; **3.** [transitive, intransitive] (of an author) to have your work printed in a newspaper, magazine, etc., or printed & sold to the public; **4.** [transitive] to make information available to the public, SYNONYM: **release**.

[141]**series** [n] (plural **series**) **1.** [countable, usually singular] **series of something** several events or things of a similar kind that come or happen one after the other; **2.** [countable] a set of radio or television programmes that deal with the same subject or that have the same characters; **3.** [uncountable, countable] (*specialist*) an electrical circuit in which the current passes through all the parts in the correct order; **4.** [countable] (*mathematics*) the sum of the terms in a sequence.

[142]**strange** [a] (**stranger, strangest**) **1.** unusual or surprising, especially in a way that is difficult to understand or explain; **2.** not familiar because you have not visited, seen or experienced it before.

[143]**entire** [a] [only before noun] (used when you are emphasizing that the whole of something is involved) including everything, everyone or every part, SYNONYM: **whole**.

[144]**retain** [v] **1. retain somebody/something** to keep somebody/something; to continue to have something & not lose it or get rid of it; **2. retain something** to take in a substance & keep holding it; **3. retain something** to remember or continue to hold something; **4. retain somebody/something** (*law*) to employ a professional person such as a lawyer or doctor; to make regular payments to such a person in order to keep their services.

[145]**original** [a] **1.** [only before noun] present or existing from the beginning; 1st or earliest; **2.** new & interesting in a way that is different from anything that has existed before; able to produce new & interesting ideas; **3.** [usually before noun] painted, written, etc. by the artist rather than copied; [n] **1.** the earliest form of something, from which copies are later made; **2.** a book, text or play in the language in which it was 1st written; **in the original** [idiom] in the language in which a book, etc. was 1st written, before being translated.

[146]**facilitate** [v] **facilitate something** to make an action or a process possible or easier.

[147]**cross references** [n] **cross reference (to something)** a note that tells a reader to look in another part of a book or file for further information; [v] **cross-reference something** to give cross references to another text or part of a text.

[148]**specifically** [adv] **1.** in a way that is connected with or intended for 1 particular person, thing or group only; **2.** in a detailed, exact & clear way; **3.** used when you want to add more detailed & exact information.

[149]**serve** [v] **1.** [intransitive, transitive] to have a particular effect, use or result; **2.** [transitive] to be useful to somebody in achieving something; **3.** [transitive] to provide an area or a group of people with a product or service; **4.** [intransitive, transitive] to work or perform duties for a person, an organization, a country, etc.; **5.** [transitive] **serve something** to spend a period of time in prison; **6.** [transitive] to give somebody food or drink, e.g. at a restaurant or during a meal; **7.** [transitive] (*law*) to give or send somebody an official document, especially one that orders them to appear in court; **serve something up** [phrasal verb] to give, offer or provide something.

[150]**content** [n] **1.** (**contents**) [plural] **content (of something)** the things that are contained in something; **2.** (**content**) [plural] the different sections that are contained in a book, magazine, journal or website; a list of these sections; **3.** [singular] the subject matter of a book, speech, programme, etc.; **4.** [singular] (following a noun or an adjective) the amount of a substance that is contained in something else; **5.** [uncountable] the information or other material contained on a website, CD-ROM, etc.; [a] [not before noun] satisfied & happy with what you have; willing to do or accept something; [v] **content yourself with something** to accept & be satisfied with something & not try to have or do something better.

[151]**subset** [n] (*specialist*) a smaller group of things or people formed from the members of a larger group.

[152]**omit** [v] **1.** to not include something/somebody, either deliberately or because you have forgotten it/them; **2. omit to do something** to not do or fail to do something.

[153]**abridged** [a] (of a book, play, etc.) made shorter by leaving parts out, OPPOSITE: **unabridged**.

[154]**edition** [n] **1.** (abbr., **ed.**) **edition (of something)** the total number of copies of a book, newspaper or magazine published at 1 time; **2.** the form in which a book is published.

[155]**independently** [adv] **1.** without being controlled or influenced by somebody/something else; **2.** without help from other people; **3.** in a way that is not connected with somebody/something else, SYNONYM: **separately**.

[156]**remaining** [a] [only before noun] **1.** not yet used, dealt with or resolved; **2.** still existing, present or in use.

The point of view I have adopted[157] [158] while writing these chapters differs from that taken in most contemporary[159] books about computer programming in that I am not trying to teach the reader how to use somebody else's software. I am concerned rather with teaching people how to write better software themselves.

My original goal was to bring readers to the frontiers[160] of knowledge in every subject that was treated. But it is extremely[161] difficult to keep up with a field that is economically profitable[162], & the rapid[163] rise of computer science has made such a dream impossible. The subject has become a vast tapestry[164] with tens of thousands of subtle[165] results contributed by tens of thousands of talented[166] people all over the world. Therefore my new goal has been to concentrate[167] on "classic" techniques that are likely to remain[168] important for many more decades, & to describe[169] them as well as I can. In particular, I have tried to trace[170] the history[171] of each subject, & to provide a solid foundation[172] for future progress. I

[157]**adopt** [v] **1.** [transitive] **adopt something** to start to use a particular method or to show a particular attitude towards somebody/something; **2.** [transitive] **adopt something** to formally accept a suggestion or policy by voting; **3.** [transitive, intransitive] **adopt (somebody)** to take somebody else's child into your family & become its legal parent(s); **4. adopt something** to choose a new name or custom & begin to use it as your own; to choose & move to a country as your permanent home; **5.** [transitive] **adopt something** to use a particular manner or way of speaking.

[158]**adopted** [a] **1.** an adopted child has legally become part of a family that is not the one in which they were born; **2.** an adopted country is one in which somebody chooses to live although it is not the one they were born in.

[159]**contemporary** [a] **1.** belonging to the present time, SYNONYM: **modern**; **2.** (especially of people & society) belonging to the same time as somebody/something else; [n] (plural **contemporaries**) a person or thing living or existing at the same time as somebody/something else, especially somebody who is about the same age as somebody else.

[160]**frontier** [n] **1.** [countable] a line that separates 2 countries, etc.; the land near this line, SYNONYM: **border**; **2.** (**the frontier**) [singular] the edge of land where people live & have built towns, beyond which the country is wild & unknown, especially in the western US in the 19th century; **3.** [countable, usually plural] **(at the) frontier of something** the limit of something, especially the limit of what is known about a particular subject or activity.

[161]**extremely** [adv] (usually with adjectives & adverbs) to a very high degree.

[162]**profitable** [a] **1.** that makes or is likely to make money; **2.** that gives somebody an advantage or a useful result.

[163]**rapid** [a] [usually before noun] happening in a short period of time or at a fast rate.

[164]**tapestry** [n] [countable, uncountable] (plural **tapestries**) a picture or pattern that is made by weaving colored wool onto heavy cloth; the art of doing this.

[165]**subtle** [a] (**subtler, subtlest**) (**more subtle** is also common) **1.** (*often approving*) (especially of a change or difference) not very obvious; not easy to notice; **2.** (of a person or their behavior) behaving in a clever way & using indirect methods in order to achieve something; **3.** showing a good understanding of things that are not obvious to other people.

[166]**talented** [a] having a natural ability to do something well.

[167]**concentrate** [v] **1.** [transitive, often passive] **concentrate something + adv./prep.** to bring something together in 1 place; **2.** [intransitive, transitive] to give all your attention to something & not think about anything else; **3.** [transitive] **concentrate something** to increase the strength of a substance by reducing its volume, e.g. by boiling it; **concentrate on something** [phrasal verb] to spend more time doing 1 particular thing than others; [n] [countable, uncountable] **concentrate (of something)** a substance that is made stronger because water or other substances have been removed.

[168]**remain** [v] (not usually used in the progressive tenses) **1.** *linking verb* to continue to be something; to be still in the same state or condition; **2.** [intransitive] **remain (of something)** to still be present after the other parts have been removed or used; to continue to exist; **3.** [intransitive] to still need to be done, said or dealt with; **4.** [intransitive] **+ adv./prep.** to stay in the same place; to not leave; **it remains to be seen (whether/what, etc.), something remains to be seen** [idiom] used to say that you cannot yet know something.

[169]**describe** [v] **1.** [often passive] to give an account of something in words. **Described** is often used after a noun phrase, without *that is/was*, etc.; **2.** [often passive] to say what somebody/something is like; to say what somebody/something is; **3. describe something** to make a movement which has a particular shape; to form a particular shape; **4. describe something** (*specialist*) (of a diagram or calculation) to represent something.

[170]**trace** [v] **1.** [often passive] **trace something (back) (to something)** to discover or describe when or how something began; **2. trace something (from something) (to something)** to describe a process or the development of something; **3.** [often passive] to find or discover somebody/something by looking carefully for them/it; **4. trace something (out)** to draw a line or lines on a surface; **5. trace something** to take a particular path or route; **6. trace something** to follow the shape or outline of something; [n] **1.** [countable, uncountable] a mark, object or sign that shows that somebody/something existed or was present. A **trace fossil** is the evidence of animal activity preserved in a rock.; **2.** [countable] a very small amount of something; **3.** [countable] a line or pattern displayed by a machine to show information about something that is being recorded or measured; **4.** [countable] **trace (of something)** a line following the path of something.

[171]**history** [n] (plural **histories**) **1.** [uncountable] all the events that happened in the past; **2.** [uncountable, singular] **history (of something)** the past events concerned in the development of a particular place, subject, etc.; **3.** [singular] **history (of something)** a record of something happening frequently in the past life of a person, family or place; the set of faces that are known about somebody's past life; **4.** [countable] **history (of something)** a written or spoken account of past events; **5.** [uncountable] the study of past events as a subject at school or university; **make history** [idiom] to do something so important that it will be recorded in history.

[172]**foundation** [n] **1.** [countable] a principle, an idea or a face that something is based on & that it grows from; **2.** [uncountable] **foundation (of something)** the act of starting a new organization or institution, SYNONYM: **establishment**; **3.** [uncountable] **foundation (of something)** the act of being the 1st to start living in a town or country; **4.** [countable] an organization that is established to provide money for a particular purpose, e.g. for scientific research or charity; **5.** [countable, usually plural] **foundation (of something)** a layer of bricks, etc. that forms the solid underground based of a building.

have attempted to choose terminology[173] that is concise[174] & consistent[175] with current[176] usage[177]. I have tried to include all of the known ideas about sequential[178] computer programming that are both beautiful & easy to state.

A few words are in order about the mathematical content of this set of books. The material has been organized so that persons with no more than a knowledge of high-school algebra may read it, skimming briefly[179] over the more mathematical portions[180]; yet a reader who is mathematically inclined[181] [182] will learn about many interesting mathematical techniques related to discrete[183] mathematics. This dual[184] level of presentation has been achieved in part by assigning ratings to each of the exercises so that the primarily mathematical ones are marked specifically as such, & also by arranging mos sections so that the main mathematical results are stated *before* their proofs. The proofs are either left as exercises (with answers to be found in a separate section) or they are given at the end of a section.

viii

” – Knuth, 1997, Preface, pp. v–

[173]**terminology** [n] (plural **terminologies**) **1.** [uncountable, countable] the set of technical words or expressions used in a particular subject; **2.** [uncountable] words used with particular meanings.

[174]**concise** [a] giving only the information that is necessary & important, using few words.

[175]**consistent** [a] **1. consistent with something** in agreement with something; not contradicting something, OPPOSITE: **inconsistent**; **2.** happening in the same way & continuing for a period of time, OPPOSITE: **inconsistent**; **3.** always behaving in the same way, or having the same opinions or standards, OPPOSITE: **inconsistent**; **4.** (of an argument or a set of ideas) having different parts that all agree with each other, OPPOSITE: **inconsistent**.

[176]**current** [a] **1.** [only before noun] existing, happening or being used now; of the present time; **2.** being used by or accepted by many people at the present time; [n] **1.** an area of water or air moving in a definite direction, especially through a surrounding area of water or air in which there is less movement; **2.** the rate of flow of electric charge through a wire, etc., measured in units of amperes; **3.** the fact of particular ideas, opinions or feelings being present in a group of people.

[177]**usage** [n] **1.** [uncountable] the fact of something being used; how much something is used; **2.** [uncountable, countable] the way in which words are used in a language; **3.** [countable] a custom, practice or habit that people have.

[178]**sequential** [a] forming or following in a logical order.

[179]**briefly** [adv] **1.** in few words; **2.** for a short time.

[180]**portion** [n] **1. portion (of something)** 1 part of something larger; **2. portion (of something)** an amount of food that is large enough for 1 person; **3.** [usually singular] **portion (of something)** a part of something that is shared with other people, SYNONYM: **share**.

[181]**incline** [v] **1.** [intransitive, transitive] to tend to think or behave in a particular way; to make somebody do this; **2.** [intransitive, transitive] to lean or slope in a particular direction; to make something lean or slope.

[182]**inclined** [a] **1. inclined to do something** tending to do something; likely to do something; **2.** [not before noun] **inclined (to do something)** wanting to do something; **3. inclined to agree, believe, think, etc.** used when you are expressing an opinion but do not want to express it very strongly; **4.** (used with particular adverbs) having a natural ability for something; preferring to do something; **5.** sloping; at an angle.

[183]**discrete** [a] independent of other things of the same type, SYNONYM: **separate**.

[184]**dual** [a] [only before noun] having 2 parts or aspects.

Chapter 3

Chacon and Straub, 2014. Scott Chacon, Ben Straub. *Pro Git*

About the Authors

Preface by Scott Chacon

Preface by Ben Straub

3.1 Getting Started

"This chapter is about getting started with Git. We will begin at the beginning by explaining some background on version control tools, then move on to how to get Git running on your system & finally how to get it set up to start working with. At the end of this chapter you should understand why Git is around, why you should use it, & you should be all set up to do so." – Chacon and Straub, 2014, p. 1

3.1.1 About Version Control

"What is "version control," & why should you care? Version control is a system that records changes to a file or a set of files over time so that you can recall specific versions later. For the examples in this book you will use software source code as the files being version controlled, though in reality you can do this with nearly any type of file on a computer.

If you are a graphic or web designer & want to keep every version of an image or layout (which you would most certainly want to), a Version Control System (VCS) is a very wise thing to use. It allows you to revert[1] files to a previous state, revert the entire project to a previous state, compare changes over time, see who last modified something that might be causing a problem, who introduced an issue & when, & more. Using a VCS also generally means that if you screw things up or lose files, you can easily recover. In addition, you get all this for very little overhead." – Chacon and Straub, 2014, p. 1

3.1.1.1 Local Version Control Systems

"Many people's version-control method of choice is to copy files into another directory (perhaps a time-stamped[2] directory, if they're clever). This approach is very common because it is so simple, but it is also incredibly error prone[3]. It is easy to forget which directory you're in & accidentally write to the wrong file or copy over files you don't mean to.

To deal with this issue, programmers long ago developed local VCSs that had a simple database that kept all the changes to files under revision control. Fig. 1-1. Local version control.

1 of the more popular VCS tools was a system called RCS, which is still distributed with many computers today. Even the popular Mac OS X operating system includes the `rcs` command when you install the Developer Tools. RCS works by keeping patch sets (i.e., the differences between files) in a special format on disk; it can then re-create what any file looked like at any point in time by adding up all the patches." – Chacon and Straub, 2014, pp. 1–2

[1]**revert** [v] **revert to somebody/something** (*law*) (of property or rights) to return to the original owner; **revert to something 1.** to return to a former state or practice; **2.** to return to an earlier topic or subject.

[2]**timestamp** [n] a mark or record that shows when something happened, especially a digital record of when something was done in a computer or other electronic system; [v] **timestamp something** to mark something with a record that shows when something happened, especially a digital record of when something was done in a computer or other electronic system.

[3]**prone** [a] **1.** likely to suffer from something or to do something bad, SYNONYM: **liable**; **2.** (**-prone**) (in adjectives) likely to suffer or do the thing mentioned; **3.** lying flat with the front of your body touching the ground.

3.1.1.2 Centralized Version Control Systems

"The next major issue that people encounter is that they need to collaborate with developers on other systems. To deal with this problem, Centralized Version Control Systems (CVCSs) were developed. These systems, such as CVS, Subversion[4], & Perforce[5], have a single server that contains all the versioned files, & a number of clients that check out files from that central place. For many years, this has been the standard for version control. Fig. 1-2. Centralized version control.

This setup offers many advantages, especially over local VCSs. E.g., everyone knows to a certain degree what everyone else on the project is doing. Administrators have fine-grained control over who can do what; & it's far easier to administer a CVCS than it is to deal with local databases on every client.

However, this setup also has some serious downsides[6]. The most obvious is the single point of failure that the centralized server represents. If that server goes down for an hour, then during that hour nobody can collaborate at all or save versioned changes to anything they're working on. If the hard disk the central database is on becomes corrupted, & proper backups haven't been kept, you lose absolutely everything – the entire history of the project except whatever single snapshots people happen to have on their local machines. Local VCS systems suffer from this same problem – whenever you have the entire history of the project in a single place, you risk losing everything." – Chacon and Straub, 2014, p. 3

3.1.1.3 Distributed Version Control Systems

"This is where Distributed Version Control Systems (DVCSs) step in. In a DVCS (such as Git, Mercurial, Bazaar, or Darcs), clients don't just check out the latest snapshot of the files – they fully mirror the repository. Thus if any server dies, & these systems were collaborating via it, any of the client repositories can be copied back up to the server to restore it. Every checkout is really a full backup of all the data. Fig. 1-3. Distributed version control.

Furthermore, many of these systems deal pretty well with having several more repositories they can work with, so you can collaborate with different groups of people in different ways simultaneously within the same project. This allows you to set up several types of workflows that aren't possible in centralized systems, such as hierarchical models." – Chacon and Straub, 2014, p. 4

3.1.2 A Short History of Git

"As with many great things in life, Git began with a bit of creative destruction & fiery[7] controversy.

The Linux kernel is an open source software project of fairly large scope. For most of the lifetime[8] of the Linux kernel maintenance (1991–2002), changes to the software were passed around as patches & archived files. In 2002, the Linux kernel project began using a proprietary DVCS called BitKeeper.

In 2005, the relationship between the community that developed the Linux kernel & the commercial company that developed BitKeeper broke down, & the tool's free-of-charge status was revoked[9]. This prompted the Linux development community (& in particular Linus Torvalds, the creator of Linux) to develop their own tool based on some of the lessons they learned while using BitKeeper. Some of the goals of the new system were as follows: • Speed. • Simple design. • Strong support for nonlinear development (thousands of parallel branches). • Fully distributed. • Able to handle large projects like the Linux kernel efficiently (speed & data size). Since its birth in 2005, Git has evolved & matured to be easy to use & yet retain these initial qualities. It's incredibly fast, it's very efficient with large projects, & it has an incredible branching system for nonlinear development (see Chap. 3)." – Chacon and Straub, 2014, p. 5

3.1.3 Git Basics

"So, what is Git in a nutshell? This is an important section to absorb, because if you understand what Git is & the fundamentals of how it works, then using Git effectively will probably be much easier for you. As you learn Git, try to clear your mind of the things you may know about other VCSs, such as Subversion & Perforce; doing so will help you avoid subtle confusion when using the tool. Git stores & thinks about information much differently than these other systems, even though the user interface is fairly similar, & understanding those differences will help prevent you from becoming confused while using it." – Chacon and Straub, 2014, p. 5

[4]**subversion** [n] (*formal*) **1.** [uncountable] the process of trying to destroy the authority of a political, religious, etc. system by attacking it secretly or indirectly; **2.** [uncountable, countable] **subversion (of something)** an act of changing something to its opposite, especially when this challenges fixed ideas or expectations.

[5]**perforce** [adv] (*old use or formal*) because it is necessary or cannot be avoided, SYNONYM: **necessarily**.

[6]**downside** [n] [singular] **downside (of something/doing something)** the disadvantages or less positive aspects of something.

[7]**fiery** [a] [usually before noun] (**fierier, fieriest**) **1.** looking like fire; consisting of fire; **2.** quickly or easily becoming angry; **3.** showing strong emotions, especially anger, SYNONYM: **passionate**; **4.** (of food or drink) causing a part of your body to feel as if it is burning.

[8]**lifetime** [n] the length of time that somebody lives or that something lasts.

[9]**revoke** [v] **revoke something** to officially cancel something so that it is no longer valid.

3.1.3.1 Snapshots, Not Differences

"The major difference between Git & any other VCS (Subversion & friends included) is the way Git thinks about its data. Conceptually[10], most other systems store information as a list of file-based changes. These systems (CVS, Subversion, Perforce, Bazaar, & so on) think of the information they keep as a set of files & the changes made to each file over time. Fig. 1-4. Storing data as changes to a base version of each file.

Git doesn't think of or store its data this way. Instead, Git thinks of its data more like a set of snapshots of miniature[11] filesystem. Every time you commit, or save the state of your project in Git, it basically takes a picture of what all your files look like at that moment & stores a reference to that snapshot. To be efficient, if files have not changed, Git doesn't store the file again, just a link to the previous identical file it has already stored. Git thinks about its data more like a *stream of snapshots*. Fig. 1-5. Storing data as snapshots of the project over time.

This is an important distinction between Git & nearly all other VCSs. It makes Git reconsider almost every aspect of version control that most other systems copied from the previous generation. This makes Git more like a mini filesystem with some incredibly powerful tools built on top of it, rather than simply a VCS. We'll explore some of the benefits you gain by thinking of your data this way when we cover Git branching in Chap. 3." – Chacon and Straub, 2014, pp. 6–7

3.1.3.2 Nearly Every Operation Is Local

"Most operations in Git only need local files & resources to operate – generally no information is needed from another computer on your network. If you're used to a CVCS where most operations have that network latency[12] overhead, this aspect of Git will make you think that the gods of speed have blessed Git with unworldly[13] powers. Because you have the entire history of the project right there on your local disk, most operations seem almost instantaneous.

E.g., to browse the history of the project, Git doesn't need to go out to the server to get the history & display it for you – it simply reads it directly from your local database. This means you see the project history almost instantly. If you want to see the changes introduced between the current version of a file & the file a month ago, Git can look up the file a month ago & do a local difference calculation, instead of having to either ask a remote server to do it or pull an older version of the file from the remote server to do it locally.

This also means that there is very little you can't do if you're offline or off VPN. If you get on an airplane or a train & want to do a little work, you can commit happily until you get to a network connection to upload. If you go home & can't get your VPN client working properly, you can still work. In many other systems, doing so is either impossible or painful. In Perforce, e.g., you can't do much when you aren't connected to the server; & in Subversion & CVS, you can edit files, but you can't commit changes to your database (because your database is offline). This may not seem like a huge deal, but you may be surprised what a big difference it can make." – Chacon and Straub, 2014, p. 7

3.1.3.3 Git Has Integrity

"Everything in Git is check-summed before it is stored & is then referred to by that checksum[14]. This means it's impossible to change the contents of any file or directory without Git knowing about it. This functionality is built in to Git at the lowest levels & is integral to its philosophy. You can't lose information in transit[15] or get file corruption[16] without Git being able to detect it.

The mechanism that Git uses for this checksumming is called a SHA-1 hash. This is a 40-character string composed of hexadecimal characters (0–9 & a–f) & calculated based on the contents of a file or directory structure in Git. A SHA-1 hash looks something like this: `24b9da6552252987aa493b52f8696cd6d3b00373`. You will see these hash values all over the place in Git because it uses them so much. In fact, Git stores everything in its database not by filename but by the hash value of its contents." – Chacon and Straub, 2014, p. 7

[10]**conceptual** [a] connected with or based on ideas.

[11]**miniature** [a] [only before noun] very small; much smaller than usual; [n] **1.** a very small detailed painting, often of a person; **2.** a very small copy or model of something; a very small version of something; **in miniature** [idiom] on a very small scale.

[12]**latency** [n] [uncountable] (*formal*) **1.** the condition of existing, but not being clear, active or well developed; **2.** [uncountable, countable] (*computing*) the delay before data begins to move after it has been sent an instruction to do so.

[13]**unworldly** [a] **1.** not interested in money or the things that it buys; **2.** having little experience of life, SYNONYM: **naive**, OPPOSITE: **worldly**; **3.** having qualities that do not seem to belong to this world.

[14]**checksum** [n] (*computing*) the total of the numbers in a piece of digital data, used to check that the data is correct.

[15]**transit** [n] [uncountable] **1.** the process of being moved or carried from 1 place to another; **2.** the act of going through a place on the way to somewhere else; **3.** (NAE) the system of buses, trains, etc. which people use to travel from 1 place to another.

[16]**corruption** [n] **1.** [uncountable] dishonest or illegal behavior, especially of people in authority; **2.** [uncountable] **corruption (of something)** the act or effect of making somebody change from moral to immoral standards of behavior; **3.** [countable, usually singular] **corruption of something** the form of a word or phrase that has become changed from its original form in some way; **4.** [uncountable] (*computing*) the process by which mistakes are introduced into a computer file, etc. with the result that the data in it is no longer correct.

3.1.3.4 Git Generally Only Adds Data

"When you do actions in Git, nearly all of them only add data to the Git database. It is hard to get the system to do anything that is not undoable or to make it erase data in any way. As in any VCS, you can lose or mess up changes you haven't committed yet; but after you commit a snapshot into Git, it is very difficult to lose, especially if you regularly push your database to another repository.

This makes using Git a joy because we know we can experiment without the danger of severely screwing things up." – Chacon and Straub, 2014, p. 7

3.1.3.5 The 3 States

3.2 Git Basics

3.3 Git Branching

3.4 Git on the Server

3.5 Distributed Git

3.6 Github

3.7 Git Tools

3.8 Customizing Git

3.9 Git & Other Systems

3.10 Git Internals

3.11 Appendix A: Git in Other Environments

3.12 Appendix B: Embedded Git in Your Applications

3.13 Appendix C: Git Commands

Chapter 4

Matthes, 2019. Python Crash Course: A Hands-on, Project-based Introduction to Programming

Preface to the 2nd Edition

"*Python Crash Course* is being used in middle schools & high schools, & also in college classes. Students who are assigned more advanced textbooks are using *Python Crash Course* as a companion text for their classes & finding it a worthwhile supplement. People are using it to enhance their skills on the job & to start working on their own side projects. In short, people are using the book for the full range of purposes I had hoped they would.

The opportunity to write a 2nd edition of *Python Crash Course* has been thoroughly enjoyable. Although Python is a mature language, it continues to evolve as every language does. My goal in revising the book was to make it $\boxed{\text{leaner \& simpler}}$. There is no longer any reason to learn Python 2, so this edition focuses on Python 3 only. Many Python packages have become easier to install, so setup & installation instructions are easier. I've added a few topics that I've realized readers would benefit from, & I've updated some sections to reflect new, simpler ways of doing things in Python. I've also clarified some sections where certain details of the languages were not presented as accurately as they could have been. All the projects have been completely updated using popular, well-maintained libraries that you can confidently use to build your own projects.

The following is a summary of specific changes that have been made in the 2nd edition:

- In Chap. 1, the instructions for installing Python have been simplified for users of all major operating systems. I now recommend the text editor Sublime Text, which is popular among beginner & professional programmers & works well on all operating systems.
- In Chap. 2 includes a more accurate description of how variables are implemented in Python. Variables are described as *labels* for values, which leads to a better understanding of how variables behave in Python. The book now users f-strings, introduced in Python 3.6. This is a much simpler way to use variable values in strings. The use of underscores to represent large numbers, such as `1_000_000`, was also introduced in Python 3.6 & is included in this edition. Multiple assignment of variables was previously introduced in 1 of the projects, & that description has been generalized & moved to Chap. 2 for the benefit of all readers. Finally, a clear convention for representing constant values in Python is included in this chapter.
- In Chap. 6, I introduce the `get()` method for retrieving values from a dictionary, which can return a default value if a key does not exist.
- The Alien Invasion project (Chaps. 12–14) is now entirely class-based. The game itself is a class, rather than a series of functions. This greatly simplifies the overall structure of the game, vastly reducing the number of function calls & parameters required. Readers familiar with the 1st edition will appreciate the simplicity this new class-based approach provides. Pygame can now be installed in 1 line on all systems, & readers are given the option of running the game in fullscreen mode or in a windowed mode.
- In the data visualization projects, the installation instructions for Matplotlib are simpler for all operating systems. The visualizations featuring Matplotlib use the `subplots()` function, which will be easier to build upon as you learn to create more complex visualizations. The Rolling Dice project in Chap. 15 uses Plotly, a well-maintained visualization library that features a clean syntax & beautiful, fully customizable output.

- In Chap. 16, the weather project is based on data from NOAA, which should be more stable over the next few years than the site used in the 1st edition. The mapping project focuses on global earthquake activity; by the end of this project you'll have a stunning visualization showing Earth's tectonic plate boundaries through a focus on the locations of all earthquakes over a given time period. You'll learn to plot any data set involving geographic points.
- Chap. 17 uses Plotly to visualize Python-related activity in open source projects on GitHub.
- The Learning Log project (Chaps. 18–20) is built using the latest version of Django & styled using the latest version of Bootstrap. The process of deploying the project to Heroku has been simplified using the `django-heroku` package, & uses environment variables rather than modifying the `settings.py` files. This is a simpler approach & is more consistent with how professional programmers deploy modern Django projects.
- Appendix A has been fully updated to recommend current best practices in installing Python. Appendix B includes detailed instructions for setting up Sublime Text & brief descriptions of most of the major text editors & IDEs in current use. Appendix C directs readers to newer, more popular online resources for getting help, & Appendix D continues to offer a mini crash course in using Git for version control.
- The index has been thoroughly updated to allow you to use *Python Crash Course* as a reference for all of your future Python projects." – Matthes, 2019, Preface to the 2nd Edition, pp. xxvii–xxix

Introduction

Who is This Book For?

"The goal of this book is to bring you up to speed with Python as quickly as possible so you can build programs that work – games, data visualizations, & web applications – while developing a foundation in programming that will serve you well for the rest of your life. *Python Crash Course* is writing for people of any age who have never before programmed in Python or have never programmed at all. This book is for those who want to learn the basics of programming quickly so they can focus on interesting projects, & those who like to test their understanding of new concepts by solving meaningful problems. *Python Crash Course* is also perfect for middle school & high school teachers who want to offer their students a project-based introduction to programming. If you're taking a college class & want a friendlier introduction to Python than the text you've been assigned, this book could make your class easier as well." – Matthes, 2019, Introduction, p. xxxiv

What Can You Expect to Learn?

"The purpose of this book is to make you a good programmer in general & a good Python programmer in particular. You'll learn efficiently & adopt good habits as I provide you with a solid foundation in general programming concepts. After working your way through *Python Crash Course*, you should be ready to move on to more advanced Python techniques, & your next programming language will be even easier to grasp.

In the 1st part of this book, you'll learn basic programming concepts you need to know to write Python programs. These concepts are the same as those you'd learn when starting out in almost any programming language. You'll learn about different kinds of data & the ways you can store data in lists & dictionaries within your programs. You'll learn to build collections of data & work through those collections in efficient ways. You'll learn to use `while` loops & `if` statements to test for certain conditions so you can run specific sections of code while those conditions are true & run other sections when they're not – a technique that greatly helps you automate processes.

You'll learn to accept input from users to make your programs interactive & to keep your programs running as long as the user is active. You'll explore how to write functions to make parts of your program reusable, so you only have to write blocks of code that perform certain actions once & then use that code as many times as you like. You'll then extend this concept to more complicated behavior with classes, making fairly simple programs respond to a variety of situations. You'll learn to write programs that handle common errors gracefully. After working through each of these basic concepts, you'll write a few short programs that solve some well-defined problems. Finally, you'll take your 1st step toward intermediate programming by learning how to write tests for your code so you can develop your programs further without worrying about introducing bugs. All the information in Part I will prepare you for taking on larger, more complex projects.

In Part II, you'll apply what you learned in Part I to 3 projects. You can do any or all of these projects in whichever order works best for you. In the 1st project (Chaps. 12–14), you'll create a *Space Invaders*–style shooting game called *Alien Invasion*, which consists of levels of increasing difficulty. After you've completed this project, you should be well on your way to being able to develop your own 2D games.

The 2nd project (Chaps. 15–17) introduces you to data visualization. Data scientists aim to make sense of the vast amount of information available to them through a variety of visualization techniques. You'll work with data sets that you generate through code, data sets that you download from online sources, & data sets your programs download automatically. After you've completed this project, you'll be able to write programs that sift through large data sets & make visual representations of that stored information.

In the 3rd project (Chaps. 18–20), you'll build a small web application called Learning Log. This project allows you to keep a journal of ideas & concepts you've learned about a specific topic. You'll be able to keep separate logs for different topics & allow others to create an account & start their own journals. You'll also learn how to deploy your project so anyone can access it online from anywhere." – Matthes, 2019, Introduction, pp. xxxiv–xxxv

Online Resources

"You can find all the supplementary resources for the book online at `https://nostarch.com/pythoncrashcourse2e/` or `http://ehmatthes.github.io/pcc_2e/`. These resources include:

- **Setup instructions.** These instructions are identical to what's in the book but include active links you can click for all the difference pieces. If you're having any setup issues, refer to this resource.
- **Updates.** Python, like all languages, is constantly evolving. I maintain a thorough set of updates, so if anything isn't working, check here to see whether instructions have changed.
- **Solutions to exercises.** You should spend significant time on your own attempting the exercises in the "Try It Yourself" sections. But if you're stuck & can't make any progress, solutions to most of the exercises are online.
- **Cheat sheets.** A full set of downloadable cheat sheets for a quick reference to major concepts is also online." – Matthes, 2019, Introduction, p. xxxv

Why Python?

"Every year I consider whether to continue using Python or whether to move on to a different language – perhaps one that's newer to the programming world. But I continue to focus on Python for many reasons. $\boxed{\text{Python is an incredibly efficient language}}$: your programs will do more in fewer lines of code than many other languages would require. Python's syntax will also help you write "clean" code. Your code will be easy to read, easy to debug, & easy to extend & build upon compared to other languages.

People use Python for many purposes: to make games, build web applications, solve business problems, & develop internal tools at all kinds of interesting companies. Python is also used heavily in scientific fields for academic research & applied work.

1 of the most important reasons I continue to use Python is because of the Python community, which includes an incredibly diverse & welcoming group of people. $\boxed{\text{Community is essential to programmers because programming isn't a solitary pursuit.}}$ Most of us, even the most experienced programmers, need to ask advice from others who have already solved similar problems. Having a well-connected & supportive community is fully supportive of people like you who are learning Python as your 1st programming language. Python is a great language to learn ..." – Matthes, 2019, Introduction, p. xxxvi

Part I: Basics

"Part I of this book teaches you the basic concepts you'll need to write Python programs. Many of these concepts are common to all programming languages, so they'll be useful throughout your life as a programmer.

- In Chap. 1 you'll install Python on your computer & run your 1st program, which prints the message *Hello world!* to the screen.
- In Chap. 2 you'll learn to store information in variables & work with text & numerical values.
- Chaps. 3–4 introduce lists. Lists can store as much information as you want in 1 variable, allowing you to work with that data efficiently. You'll be able to work with hundreds, thousands, & even millions of values in just a few lines of code.
- In Chap. 5 you'll use `if` statements to write code that responds 1 way if certain conditions are true, & responds in a different way if those conditions are not true.
- Chap. 6 shows you how to use Python's dictionaries, which let you make connections between different pieces of information. Like lists, dictionaries can contain as much information as you need to store.
- In Chap. 7 you'll learn how to accept input from users to make your programs interactive. You'll also learn about `while` loops, which run blocks of code repeatedly as long as certain conditions remain true.

- In Chap. 8 you'll write functions, which are named blocks of code that perform a specific task & can be run whether you need them.
- Chap. 9 introduces *classes*, which allow you to model real-world objects, such as dogs, cats, people, cars, rockets, & much more, so your code can represent anything real or abstract.
- Chap. 10 shows you how to work with files & handle errors so your programs won't crash unexpectedly. You'll store data before your program closes, & read the data back in when the program runs again. You'll learn about Python's exceptions, which allow you to anticipate errors, & make your programs handle those errors gracefully.
- In Chap. 11 you'll learn to write tests fr your code to check that your programs work the way you intend them to. As a result, you'll be able to expand your programs without worrying about introducing new bugs. Testing your code is 1 of the 1st skills that will help you transition from beginner to intermediate programmer." – Matthes, 2019, pp. 1–2

4.1 Getting Started

4.1.1 Setting Up Your Programming Environment

"Python differs slightly on different operating systems, so you'll need to keep a few considerations in mind. In the following sections, we'll make sure Python is set up correctly on your system." – Matthes, 2019, p. 3

4.1.1.1 Python Versions

"Every programming language evolves as new ideas & technologies emerge, & the developers of Python have continually made the language more versatile & powerful." "In this section, we'll find out if Python is already installed on your system & whether you need to install a newer version. Appendix A contains a comprehensive guide to installing the latest version of Python on each major operating system as well. Some old Python projects still use Python 2, but you should use Python 3. If Python 2 is installed on your system, it's probably there to support some older programs that your system needs." – Matthes, 2019, p. 4

4.1.1.2 Running Snippets of Python Code

"You can run Python's interpreter in a terminal window, allowing you to try bits of Python code without having to save & run an entire program." "The `>>>` prompt indicates that you should be using the terminal window, & the bold text is the code you should type in & then execute by pressing `Enter`. Most of the examples in the book are small, self-contained programs that you'll run from your text editor rather than the terminal, because you'll write most of your code in the text editor. But sometimes basic concepts will be shown in a series of snippets run through a Python terminal session to demonstrate particular concepts more efficiently. When you see 3 angle brackets in a code listing, you're looking at code & output from a terminal session." – Matthes, 2019, p. 4

4.1.2 About the Sublime Text Editor

"Sublime Text is a simple text editor that can be installed on all modern operating systems. Sublime Text lets you run almost all of your programs directly from the editor instead of through a terminal. Your code runs in a terminal session embedded in the Sublime Text window, which makes it easy to see the output.

Sublime Text is a beginner-friendly editor, but many professional programmers use it as well. If you become comfortable using it while learning Python, you can continue using it as you progress to larger & more complicated projects. Sublime Text has a very liberal licensing policy: you can use the editor free of charge as long as you want, but the developers request that you purchase a license if you like it & want to keep using it.

Appendix B provides information on other text editors. If you're curious about the other options, you might want to skim that appendix at this point. If you want to begin programming quickly, you can use Sublime Text to start & consider other editors once you've gained some experience as a programmer." – Matthes, 2019, pp. 4–5

4.1.3 Python on Different Operating Systems

"Python is a cross-platform programming language, which means it runs on all the major operating systems. Any Python program you write should run on any modern computer that has Python installed. However, the methods for setting up Python on different operating systems vary slightly." – Matthes, 2019, p. 5

4.1.3.1 Python on Windows

"Window doesn't always come with Python, so you'll probably need to install it, & then install Sublime Text." – Matthes, 2019, p. 5

4.1.3.1.1 Installing Python. "1st, check whether Python is installed on your system. Open a command window by entering `command` into the Start menu or by holding down the `Shift` key while right-clicking on your desktop & selecting `Open command window here` from the menu. In the terminal window, enter `python` in lowercase. If you get a Python prompt (`>>>`) in response, Python is installed on your system. If you see an error message telling you that `python` is not a recognized command, Python isn't installed.

In that case, or if you see a version of Python earlier than Python 3.6, you need to download a Python installer for Windows. Go to `https://python.org/` & hover over the **Downloads** link. You should see a button for downloading the latest version of Python. Click the button, which should automatically start downloading the correct installer for your system. After you've downloaded the file, run the installer. Make sure you select the option `Add Python to PATH`, which will make it easier to configure your system correctly." – Matthes, 2019, pp. 5–6

4.1.3.1.2 Running Python in a Terminal Session. "Open a command window & enter `python` in lowercase. You should see a Python prompt (`>>>`), which means Windows has found the version of Python you just installed."

```
C:\> python
Python 3.7.2 (v3.7.2:9a3ffc0492, Dec 23 2018, 23:09:28) [MSC v.1916 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

"Any time you want to run a snippet of Python code, open a command window & start a Python terminal session. To close the terminal session, press `Ctrl-Z` & then press `Enter`, or enter the command `exit()`." – Matthes, 2019, p. 6

4.1.3.1.3 Installing Sublime Text. "You can download an installer for Sublime text at `https://sublimetext.com/`. Click the download link & look for a Windows installer. After downloading the installer, run the installer & accept all of its defaults." – Matthes, 2019, p. 7

4.1.3.2 Python on macOS

"Python is already installed on most macOS systems, but it's most likely an outdated version that you won't want to learn on." – Matthes, 2019, p. 7

4.1.3.2.1 Checking Whether Python 3 Is Installed. "Open a terminal window by going to `Applications` ▷ `Utilities` ▷ `Terminal`. You can also press `cmd`-spacebar, type `terminal`, & then press `Enter`. To see which version of Python is installed, enter `python` with a lowercase `p` – this also starts the Python interpreter within the terminal, allowing you to enter Python commands. You should see output telling you which Python version is installed on your system & a `>>>` prompt where you can start entering Python commands, like this:

```
$ python
Python 2.7.15 (default, Aug 17 2018, 22:39:05)
[GCC 4.2.1 Compatible Apple LLVM 9.1.0 (clang-902.0.39.2)] on darwin
Type "help", "copyright", "credits", or "license" for more information.
>>>
```

This output indicates that Python 2.7.15 is currently the default version installed on this computer. Once you've seen this output, press `Ctrl-D` or enter `exit()` to leave the Python prompt & return to a terminal prompt.

To check whether you have Python 3 installed, enter the command `python3`. You'll probably get an error message, meaning you don't have any versions of Python 3 installed. If the output shows you have Python 3.6 or a later version installed, you can skip ahead to "Running Python in a Terminal Session" on p. 8. If Python 3 isn't installed by default, you'll need to install it manually. Note that whenever you see the `python` command in this book, you need to use the `python3` command instead to make sure you're using Python 3, not Python 2; they different significantly enough that you'll run into trouble trying to run the code in this book using Python 2.

If you see any version earlier than Python 3.6, follow the instructions in the next section to install the latest version." – Matthes, 2019, p. 7

4.1.3.2.2 Installing the Latest Version of Python. "You can find a Python installer for your system at https://python.org/. Hover over the **Download** link, & you should see a button for downloading the latest Python version. Click the button, which should automatically start downloading the correct installer for your system. After the file downloads, run the installer. When you're finished, enter the following at a terminal prompt:

```
$ python3 --version
Python 3.7.2
```

You should see output similar to this, in which case, you're ready to try out Python. Whenever you see the command `python`, make sure you see `python3`." – Matthes, 2019, pp. 7–8

4.1.3.2.3 Running Python in a Terminal Session. "You can now try running snippets of Python code by opening a terminal & typing `python3`." "Remember that you can close the Python interpreter by pressing `Ctrl-D` or by entering the command `exit()`." – Matthes, 2019, p. 8

4.1.3.2.4 Installing Sublime Text. "To install the Sublime Text editor, you need to download the installer at https://sublimetext.com/. Click the **Download** link & look for an installer for macOS. After the installer downloads, open it & then drag the Sublime Text icon into your `Applications` folder." – Matthes, 2019, p. 8

4.1.3.3 Python on Linux

"$\boxed{\text{Linux systems are designed for programming, so Python is already installed on most Linux computers}}$. The people who write & maintain Linux expect you to do your own programming at some point & encourage you to do so. For this reason, there's very little to install & only a few settings to change to start programming." – Matthes, 2019, p. 8

4.1.3.3.1 Checking Your Version of Python. "Open a terminal window by running the Terminal application on your system (in Ubuntu, you can press `Ctrl-Alt-T`). To find out which version of Python is installed, this command starts the Python interpreter. You should see output indicating which version of Python is installed & a `>>>` prompt where you can start entering Python commands, like this:

```
$ python3
Python 3.7.2 (default, Dec 27 2018, 04:01:51)
[GCC 7.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

This output indicates that Python 3.7.2 is currently the default version of Python installed on this computer. When you've seen this output, press `Ctrl-D` or enter `exit()` to leave the Python prompt & return to a terminal prompt. Whenever you see the `python` command in this book, enter `python3` instead." – Matthes, 2019, pp. 8–9

4.1.3.3.2 Running Python in a Terminal Session. "You can try running snippets of Python code by opening a terminal & entering `python3`, as you did when checking your version. Do this again, & when you have Python running, enter the following line in the terminal session:

```
>>> print("Hello Python interpreter!")
Hello Python interpreter!
>>>
```

The message should print directly in the current terminal window. Remember that you can close the Python interpreter by pressing `Ctrl-D` or by entering the command `exit()`." – Matthes, 2019, p. 9

4.1.3.3.3 Installing Sublime Text. "On Linux, you can install Sublime Text from the Ubuntu Software Center. Click the Ubuntu Software icon in your menu, & search for **Sublime Text**. Click to install it, & then launch it." – Matthes, 2019, p. 9

4.1.4 Running a Hello World Program

"With a recent version of Python & Sublime Text installed, you're almost ready to run your 1st Python program written in a text editor. But before doing so, you need to make sure Sublime Text is set up to use the correct version of Python on your system." – Matthes, 2019, p. 9

4.1.4.1 Configuring Sublime Text to Use the Correct Python Version

"If the `python` command on your system runs Python 3, you won't need to configure anything & can skip to the next section. If you use the `python3` command, you'll need to configure Sublime Text to use the correct Python version when it runs your programs.

Click the Sublime Text icon to launch it, or search for Sublime Text in your system's search bar & then launch it. Go to `Tools ▷ Build System ▷ New Build System`, which will open a new configuration file for you. Delete what you see & enter the following: `Python3.sublime-build`

```
{
    "cmd": ["python3", "-u", "$file"],
}
```

This code tells Sublime Text to use your system's `python3` command when running your Python program files. Save the file as `Python3.sublime-build` in the default directory that Sublime Text opens when you choose Save." – Matthes, 2019, pp. 9–10

4.1.4.2 Running `hello_world.py`

"Before you write your 1st program, make a folder called `python_work` somewhere on your system for your projects. It's best to use lowercase letters & underscores for spaces in file & folder names, because Python uses these naming conventions.

Open Sublime Text, & save an empty Python (`File ▷ Save As`) called `hello_world.py` in your `python_work` folder. The extension `.py` tells Sublime Text that the code in your file is written in Python, which tells it how to run the program & highlight the text in a helpful way. After you've saved your file, enter the following line in the text editor: `hello_world.py`

```
print("Hello Python world!")
```

If the `python` command works on your system, you can run your program by selecting `Tools ▷ Build` in the menu or by pressing `Ctrl-B` (`cmd-B` on macOS). If you had to configure Sublime Text in the previous section, select `Tools ▷ Build System` & then select `Python 3`. From now on you'll be able to select `Tools ▷ Build` or just press `Ctrl-B` (or `cmd-B`) to run your programs. A terminal screen should appear at the bottom of the Sublime Text window, showing the following output:

```
Hello Python world!
[Finished in 0.1s]
```

If you don't see this output, something might have gone wrong in the program. Check every character on the line you entered. Did you accidentally capitalize `print`? Did you forget 1 or both of the quotation marks or parentheses? Programming languages expect very specific syntax, & if you don't provide that, you'll get errors." – Matthes, 2019, p. 10

4.1.5 Troubleshooting

"If you can't get `hello_world.py` to run, here are a few remedies you can try that are also good general solutions for any programming problem:

- When a program contains a significant error, Python displays a *traceback*, which is an error report. Python looks through the file & tries to identify the problem. Check the traceback; it might give you a clue as to what issue is preventing the program from running.
- Step away from your computer, take a short break, & then try again. Remember that syntax is very important in programming so even a missing colon, a mismatched quotation mark, or mismatched parentheses can prevent a program from running properly. Reread the relevant parts of this chapter, look over your code, & try to find the mistake.
- Start over again. You probably don't need to uninstall any software, but it might make sense to delete your `hello_world.py` file & re-create it from scratch.
- Ask someone else to follow the steps in this chapter, on your computer or a different one, & what they do carefully. You might have missed 1 small step that someone else happens to catch.
- Find someone who knows Python & ask them to help you get set up. If you ask around, you might find that you unexpectedly know someone who uses Python.
- The setup instructions in this chapter are also available through the book's companion website at `https://nostarch.com/pythoncrashcourse2e/`. The online version of these instructions might work better for you because you can simply cut & paste code.

- Ask for help online. Appendix C provides a number of resources, such as forums & live chat sites, where you can ask for solutions from people who've already worked through the issue you're currently facing.

Never worry that you're bothering experienced programmers. Every programmer has been struck at some point, & most programmers are happy to help you set up your system correctly. As long as you can state clearly what you're trying to do, what you've already tried, & the results you're getting, there's a good chance someone will be able to help you. As mentioned in the Introduction, the Python community is very friendly & welcoming to beginners.

Python should run well on any modern computer. Early setup issues can be frustrating, but they're well worth sorting out. Once you get `hello_world.py` running, you can start to learn Python, & your programming work will become more interesting & satisfying." – Matthes, 2019, p. 11

4.1.6 Running Python Programs from a Terminal

"Most of the programs you write in your text editor you'll run directly from the editor. But sometimes it's useful to run programs from a terminal instead. E.g., you might want to run an existing program without opening it for editing. You can do this on any system with Python installed if you know how to access the directory where the program file is stored. To try this, make sure you've saved the `hello_world.py` file in the `python_work` folder on your desktop." – Matthes, 2019, p. 12

4.1.6.1 On Windows

"You can use the terminal command `cd`, for *change directory*, to navigate through your filesystem in a command window. The command `dir`, for *directory*, shows you all the files that exist in the current directory. Open a new terminal window & enter the following commands to run `hello_world.py`:

```
C:\> cd Desktop\python_work
C:\Desktop\python_work> dir
hello_world.py
C:\Desktop\python_work> python hello_world.py
Hello Python world!
```

At **1** you use the cd command to navigate to `python_work` folder, which is in the `Desktop` folder. Next, you use the `dir` command to make sure `hello_world.py` is in this folder **2**. Then you run the file using the comand `python hello_world.py`. Most of your programs will run fine directly from your editor. But as your work becomes more complex, you'll want to run some of your programs from a terminal." – Matthes, 2019, p. 12

4.1.6.2 On macOS & Linux

"Running a Python program from a terminal session is the same on Linux & macOS. You can use the terminal command `cd`, for *change directory*, to navigate through your filesystem in a terminal session. The command `ls`, for *list*, shows you all the nonhidden files that exist in the current directory. Open a new terminal window & enter the following commands to run `hello_world.py`:

```
~$ cd Desktop/python_work/
~/Desktop/python_work$ ls
hello_world.py
~/Desktop/python_work$ python hello_world.py
Hello Python world!
```

At **1** you use the `cd` command to navigate to the `python_work` folder, which is in the `Desktop` folder. Next, you use the `ls` command to make sure `hello_world.py` is in this folder **2**. Then you run the file using the command `python hello_world.py` **3**. It's that simple. You just use the `python` (or `python3`) command to run Python programs."

"**python.org.** Explore the Python home page `https://python.org/` to find topics that interests you. As you become familiar with Python, different parts of the site will be more useful to you." "**Infinite Skills.** If you had infinite programming skills, what would you build? You're about to learn how to program. If you have an end goal in mind, you'll have an immediate use for your new skills; now is a great time to draft descriptions of what you want to create. It's a good habit to keep an "ideas" notebook that you can refer to whenever you want to start a new project. Take a few minutes now to describe 3 programs you want to create." – Matthes, 2019, pp. 12–13

4.2 Variables & Simple Data Types

Content. *Different kinds of data you can work with in your Python programs, how to use variables to represent data in your programs.*

4.2.1 What Really Happens When You Run `hello_world.py`

"When you run the file `hello_world.py`, the ending `.py` indicates that the file is a Python program. Your editor then runs the file through the *Python interpreter*, which reads through the program & determines what each word in the program means. E.g., when the interpreter sees the word `print` followed by parentheses, it prints to the screen whatever is inside the parentheses. As you write your programs, your editor highlights different parts of your program in different way. E.g., it recognizes that `print()` is the name of a function & displays that word in 1 color. It recognizes that `"Hello Python world!"` is not Python code & displays that phrase in a different color. This feature is called *syntax highlighting* & is quite useful as you start to write your own programs." – Matthes, 2019, p. 16

4.2.2 Variables

"Every variable is connected to a *value*, which is the information associated with that variable." "Adding a variable makes a little more work for the Python interpreter." – Matthes, 2019, p. 16

"You can change the value of a variable in your program at any time, & Python will always keep track of its current value." – Matthes, 2019, p. 17

4.2.2.1 Naming & Using Variables

"When you're using variables in Python, you need to adhere to a few rules & guidelines. Breaking some of these rules will cause errors; other guidelines just help you write code that's easier to read & understand. Be sure to keep the following variable rules in mind: • Variable names can contain only letters, numbers, & underscores. They can start with a letter or an underscore, but not with a number. E.g., you can call a variable `message_1` but not `1_message`. • Spaces are not allowed in variable names, but underscores can be used to separate words in variable names. E.g., `greeting_message` works, but `greeting message` will cause errors. • Avoid using Python keywords & function names as variable names; i.e., do not use words that Python has reserved for a particular programmatic purpose, such as the word `print`. • Variable names should be short but descriptive. E.g., `name` is better than `n`, `student_name` is better than `s_n`, & `name_length` is better than `length_of_persons_name`. • Be careful when using the lowercase letter *l* & the uppercase letter *O* because they could be confused with the numbers 1 & 0.

It can take some practice to learn how to create good variable names, especially as your programs become more interesting & complicated. As you write more programs & start to read through other people's code, you'll get better at coming up with meaningful names." – Matthes, 2019, p. 17

4.2.2.2 Avoiding Name Errors When Using Variables

"Every programmer makes mistakes, & most make mistakes every day. Although good programmers might create errors, they also know how to respond to those errors efficiently. Let's look at an error you're likely to make early on & learn how to fix it." – Matthes, 2019, p. 17

"When an error occurs in your program, the Python interpreter does its best to help you figure out where the problem is. The interpreter provides a traceback when a program cannot run successfully. A *traceback* is a record of where the interpreter ran into trouble when trying to execute your code." "The Python interpreter doesn't spellcheck your code, but it does ensure that variable names are spelled consistently." "Programming languages are strict, but they disregard good & bad spelling. As a result, you don't need to consider English spelling & grammar rules when you're trying to create variable names & writing code. Many programming errors are simple, single-character typos in 1 line of a program. If you're spending a long time searching for 1 of these errors, know that you're in good company. Many experienced & talented programmers spend hours hunting down these kinds of tiny errors. Try to laugh about it & move on, knowing it will happen frequently throughout your programming life." – Matthes, 2019, p. 18

4.2.2.3 Variables Are Labels

"Variables are often described as boxes you can store values in. This idea can be helpful the 1st few times you use a variable, but it isn't an accurate way to describe how variables are represented internally in Python. It's much better to think of variables as labels that you can assign to values. You can also say that a variable references a certain value.

This distinction probably won't matter much in your initial programs, but it's worth learning earlier rather than later. At some point, you'll see unexpected behavior from a variable, & an accurate understanding of how variables work will help you identify what's happening in your code." – Matthes, 2019, pp. 18–19

4.2.3 Strings

"Because most programs define & gather some sort of data, & then do something useful with it, it helps to classify different types of data. The 1st data type we'll look at is the string. Strings are quite simple at 1st glance, but you can use them in many different ways. A *string* is a series of characters. Anything inside quotes is considered a string in Python, & you can use single or double quotes around your strings like this: `"This is a string." 'This is also a string.'`. This flexibility allows you to use quotes & apostrophes within your strings:

```
'I told my friend, "Python is my favorite language!"'
"The language 'Python' is named after Monty Python, not the snake."
"One of Python's strengths is its diverse and supportive community."
```

" – Matthes, 2019, p. 19

4.2.3.1 Changing Case in a String with Methods

"1 of the simplest tasks you can do with strings is change the case of the words in a string." – Matthes, 2019, p. 20

```
name = "ada lovelace"
print(name.title())
```

Output: `Ada Lovelace`. "In this example, the variable `name` refers to the lowercase string `"ada lovelace"`. The method `title()` appears after the variable in the `print()` call. A *method* is an action that Python can perform on a piece of data. The dot (`.`) after `name` in `name.title()` tells Python to make the `title()` method act on the variable `name`. Every method is followed by a set of parentheses, because methods often need additional information to do their work. That information is provided inside the parentheses. The `title()` function doesn't need any additional information, so its parentheses are empty.

The `title()` method changes each word to title case, where each word begins with a capital letter. This is useful because you'll often want to think of a name as a piece of information. E.g., you might want your program to recognize the input values `Ada`, `ADA`, & `ada` as the same name, & display all of them as `Ada`.

Several other useful methods are available for dealing with case as well. E.g., you can change a string to all uppercase or all lowercase letters like this:

```
name = "Ada Lovelace"
print(name.upper())
print(name.lower())
```

This will display the following:

```
ADA LOVELACE
ada lovelace
```

The `lower()` method is particularly useful for storing data. Many times you won't want to trust the capitalization that your users provide, so you'll convert strings to lowercase before storing them. Then when you want to display the information, you'll use the case that makes the most sense for each string." – Matthes, 2019, p. 20

4.2.3.2 Using Variables in Strings

"In some situations, you'll want to use a variable's value inside a string. E.g., you might want 2 variables to represent a 1st name & a last name respectively, & then want to combine those values to display someone's full name:

```
first_name = "ada"
last_name = "lovelace"
full_name = f"{first_name} {last_name}"
print(full_name)
```

To insert a variable's value into a string, place the letter `f` immediately before the opening quotation mark. Put braces around the name or names of any variable you want to use inside the string. Python will replace each variable with its value when the string is displayed.

These strings are called *f-strings*. The *f* is for *format*, because Python formats the string by replacing the name of any variable in braces with its value. The output from the previous code is: `ada lovelace`. You can do a lot with f-strings. E.g., you can use f-strings to compose complete messages using the information associated with a variable, as shown here:

```
first_name = "ada"
last_name = "lovelace"
full_name = f"{first_name} {last_name}"
print(f"Hello, {full_name.title()}!")
```

The full name is used in a sentence that greets the user, & the `title()` method changes the name to title case. This code returns a simple but nicely formatted greeting: `Hello, Ada Lovelace!` You can also use f-strings to compose a message, & then assign the entire message to a variable:

```
first_name = "ada"
last_name = "lovelace"
full_name = f"{first_name} {last_name}"
message = f"Hello, {full_name.title()}!"
print(message)
```

This code displays the message `Hello, Ada Lovelace!` as well, but by assigning the message to a variable, we make the final `print()` call much simpler." – Matthes, 2019, p. 21

Remark 4.1. *"F-strings were 1st introduced in Python 3.6. If you're using Python 3.5 or earlier, you'll need to use the* `format()` *method rather than this f syntax. To use* `format()`*, list the variables you want to use in the string inside the parentheses following* `format`*. Each variable is referred to by a set of braces; the braces will be filled by the values listed in parentheses in the order provided:* `full_name = "{} {}".format(first_name, last_name)`*." – Matthes, 2019, p. 22*

4.2.3.3 Adding Whitespace to Strings with Tabs or Newlines

"In programming, *whitespace* refers to any nonprinting character, such as spaces, tabs, & end-of-line symbols. You can use whitespace to organize your output so it's easier for users to read. To add a tab to your text, use the character combination `\t` as shown:

```
>>> print("Python")
Python
>>> print("\tPython")
    Python
```

To add a newline in a string, use the character combination `\n`:

```
>>> print("Languages:\nPython\nC\nJavaScript")
Languages:
Python
C
JavaScript
```

You can also combine tabs & newlines in a single string. The string `"\n\t"` tells Python to move to a new line, & start the next line with a tab. The following example shows how you can use a 1-line string to generate 4 lines of output:

```
>>> print("Languages:\n\tPython\n\tC\n\tJavaScript")
Languages:
    Python
    C
    JavaScript
```

Newlines & tabs will be very useful in the next 2 chapters when you start to produce many lines of output from just a few lines of code." – Matthes, 2019, p. 22

4.2.3.4 Stripping Whitespace

"Extra whitespace can be confusing in your programs. To programmers `'python'` & `'python '` look pretty much the same. But to a program, they are 2 different strings. Python detects the extra space in `'python '` & considers it significant unless you tell it otherwise.

It's important to think about whitespace, because often you'll want to compare 2 strings to determine whether they are the same. E.g., 1 important instance might involve checking people's usernames when they log in to a website. Extra whitespace can be confusing in much simpler situation as well. Fortunately, Python makes it easy to eliminate extraneous whitespace from data that people enter.

Python can look for extra whitespace on the right & left sides of a string. To ensure that no whitespace exists at the right end of a string, use the `rstrip()` method.

```
>>> favorite_language = 'python '
>>> favorite_language
'python '
>>> favorite_language.rstrip()
'python'
>>> favorite_language
'python '
```

The value associated with `favorite_language` at 1st line contains extra whitespace at the end of the string. When you ask Python for this value in a terminal session, you can see the space at the end of the value. When the `rstrip()` method acts on the variable `favorite_language`, this extra space is removed. However, it is only removed temporarily. If you ask for the value of `favorite_language` again, you can see that the string looks the same as when it was entered, including the extra whitespace.

To remove the whitespace from the string permanently, you have to associate the stripped value with the variable name:

```
>>> favorite_language = 'python '
>>> favorite_language = favorite_language.rstrip()
>>> favorite_language
'python'
```

To remove the whitespace from the string, you strip the whitespace from the right side of the string & then associate this new value with the original variable. Changing a variable's value is done often in programming. This is how a variable's value can be updated as a program is executed or in response to user input.

You can also strip whitespace from the left side of a string using the `lstrip()` method, or from both sides at once using `strip()`:

```
>>> favorite_language = ' python '
>>> favorite_language.rstrip()
' python'
>>> favorite_language.lstrip()
'python '
>>> favorite_language.strip()
'python'
```

In this example, we start with a value that has whitespace at the beginning & the end. We then remove the extra space from the right side, from the left side, & from both sides. Experimenting with these stripping functions can help you become familiar with manipulating strings. In the real world, these stripping functions are used most often to clean up user input before it's stored in a program." – Matthes, 2019, pp. 22–23

4.2.3.5 Avoiding Syntax Errors with Strings

"1 kind of error that you might see with some regularity is a syntax error. A *syntax error* occurs when Python doesn't recognize a section of your program as valid Python code. E.g., if you use an apostrophe within single quotes, you'll produce an error. This happens because Python interprets everything between the 1st single quote & the apostrophe as a string. It then tries to interpret the rest of the text as Python code, which causes errors. Here's how to use single & double quotes correctly. Save this program as `apostrophe.py` & then run it:

```
message = "One of Python's strengths is its diverse community."
print(message)
```

The apostrophe appears inside a set of double quotes, so the Python interpreter has no trouble reading the string correctly:

```
One of Python's strengths is its diverse community.
```

However, if you use single quotes, Python can't identify where the string should end:

```
message = 'One of Python's strengths is its diverse community.'
print(message)
```

You'll see the following output:

```
  File "apostrophe.py", line 1
    message = 'One of Python's strengths is its diverse community.'
                            ^
SyntaxError: invalid syntax
```

In the output you can see that the error occurs at ^ right after the 2nd single quote. This syntax error indicates that the interpreter doesn't recognize something in the code as valid Python code. Errors can come from a variety of sources, & I'll point out some common ones as they arise. You might see syntax errors often as you learn to write proper Python code. Syntax errors are also the least specific kind of error, so they can be difficult & frustrating to identify & correct. If you get stuck on a particularly stubborn error, see the suggestions in Appendix C." – Matthes, 2019, p. 24

Remark 4.2. *"Your editor's syntax highlighting feature should help you spot some syntax errors quickly as you write your programs. If you see Python code highlighted as if it's English or English highlighted as if it's Python code, you probably have a mismatched quotation mark somewhere in your file." – Matthes, 2019, p. 25*

4.2.4 Numbers

"Numbers are used quite often in programming to keep score in games, represent data in visualizations, store information in web applications, & so on. Python treats numbers in several different ways, depending on how they're being used." – Matthes, 2019, p. 25

4.2.4.1 Integers

"You can add (+), subtract (-), multiply (*), & divide (/) integers in Python.

```
>>> 2 + 3
5
>>> 3 - 2
1
>>> 2 * 3
6
>>> 3 / 2
1.5
```

In a terminal session, Python simply returns the result of the operation." "Python uses 2 multiplication symbols to represent exponents:

```
>>> 3 ** 2
9
>>> 3 ** 3
27
>>> 10 ** 6
1000000
```

Python supports the order of operations too, so you can use multiple operations in 1 expression. You can also use parentheses to modify the order of operations so Python can evaluate your expression in the order you specify. E.g.:

```
>>> 2 + 3*4
14
>>> (2 + 3) * 4
20
```

The spacing in these examples has no effect on how Python evaluates the expressions; it simply helps you more quickly spot the operations that have priority when you're reading through the code." – Matthes, 2019, p. 26

4.2.4.2 Floats

"Python calls any number with a decimal point a *float*. This term is used in most programming languages, & it refers to the fact that a decimal point can appear at any position in a number. Every programming language must be carefully designed to properly manage decimal numbers so numbers behave appropriately no matter where the decimal point appears.

For the most part, you can use decimals without worrying about how they behave. Simply enter the numbers you want to use, & Python will most likely do what you expect:

```
>>> 0.1 + 0.1
0.2
>>> 0.2 + 0.2
0.4
>>> 2 * 0.1
```

```
0.2
>>> 2 * 0.2
0.4
```

But be aware that you can sometimes get an arbitrary number of decimal places in your answer:

```
>>> 0.2 + 0.1
0.30000000000000004
>>> 3 * 0.1
0.30000000000000004
```

This happens in all languages & is of little concern. Python tries to find a way to represent the result as precisely as possible, which is sometimes difficult given how computers have to represent numbers internally. Just ignore the extra decimal places for now; you'll learn ways to deal with the extra places when you need to in the projects in Part II." – Matthes, 2019, pp. 26–27

4.2.4.3 Integers & Floats

"When you divide any 2 numbers, even if they are integers that result in a whole number, you'll always get a float:

```
>>> 4/2
2.0
```

If you mix an integer & a float in any other operation, you'll get a float as well:

```
>>> 1 + 2.0
3.0
>>> 2 * 3.0
6.0
>>> 3.0 ** 2
9.0
```

Python defaults to a float in any operation that uses a float, even if the output is a whole number." – Matthes, 2019, p. 27

4.2.4.4 Underscores in Numbers

"When you're writing long numbers, you can group digits using underscores to make large numbers more readable:

```
>>> universe_age = 14_000_000_000
```

When you print a number that was defined using underscores, Python prints only the digits:

```
>>> print(universe_age)
14000000000
```

Python ignores the underscores when storing these kinds of values. Even if you don't group the digits in threes, the value will still be unaffected. To Python, `1000` is the same as `1_000`, which is the same as `10_00`. This feature works for integers & floats, but it's only available in Python 3.6 & later." – Matthes, 2019, p. 28

4.2.4.5 Multiple Assignment

"You can assign values to > 1 variable using just a single line. This can help shorten your programs & make them easier to read; you'll use this technique most often when initializing a set of numbers. E.g., here's how you can initialize the variables x, y, z to 0:

```
>>> x, y, z = 0, 0, 0
```

You need to separate the variable names with commas, & do the same with the values, & Python will assign each value to its respectively positioned variable. As long as the number of values matches the number of variables, Python will match them up correctly." – Matthes, 2019, p. 28

4.2.4.6 Constants

"A *constant* is like a variable whose value stays the same throughout the life of a program. Python doesn't have built-in constant types, but Python programmers use all capital letters to indicate a variable should be treated as a constant & never be changed: `MAX_CONNECTIONS = 5000`. When you want to treat a variable as a constant in your code, make the name of the variable all capital letters." – Matthes, 2019, p. 28

4.2.5 Comments

"Comments are an extremely useful feature in most programming languages. Everything you've written in your programs so far is Python code. As your programs become longer & more complicated, you should add notes within your programs that describe your overall approach to the problem you're solving. A *comment* allows you to write notes in English within your programs." – Matthes, 2019, p. 29

4.2.5.1 How Do You Write Comments?

"In Python, the hash mark (`#`) indicates a comment. Anything following a hash mark in your code is ignored by the Python interpreter. E.g.,

```
# Say hello to everyone.
print("Hello Python people!")
```

Python ignores line 1 & executes line 2. `Hello Python people!`" – Matthes, 2019, p. 29

4.2.5.2 What Kind of Comments Should You Write?

"The main reason to write comments is to explain what your code is supposed to do & how you are making it work. When you're in the middle of working on a project, you understand how all of the pieces fit together. But when you return to a project after some time away, you'll likely have forgotten some of the details. You can always study your code for a while & figure out how segments were supposed to work, but writing good comments can save you time by summarizing your overall approach in clear English.

If you want to become a professional programmer or collaborate with other programmers, you should write meaningful comments. Today, most software is written collaboratively, whether by a group of employees at 1 company or a group of people working together on an open source project. Skilled programmers expect to see comments in code, so it's best to start adding descriptive comments to your programs now. Writing clear, concise comments in your code is 1 of the most beneficial habits you can form as a new programmer.

When you're determining whether to write a comment, ask yourself if you had to consider several approaches before coming up with a reasonable way to make something work; if so, write a comment about your solution. It's much easier to delete extra comments later on than it is to go back & write comments for a sparsely commented program." – Matthes, 2019, pp. 29–30

4.2.6 The Zen of Python

"Experienced Python programmers will encourage you to avoid complexity & aim for simplicity whenever possible. The Python's community's philosophy is contained in "The Zen of Python" by Tim Peters. You can access this brief set of principles for writing good Python code by entering `import this` into your interpreter. I won't reproduce the entire "Zen of Python" here, but I'll share a few lines to help you understand why they should be important to you as a beginning Python programmer.

```
>>> import this
The Zen of Python, by Tim Peters
Beautiful is better than ugly.
```

Python programmers embrace the notion that code can be beautiful & elegant. In programming, people solve problems. Programmers have always respected well-designed, efficient, & even beautiful solutions to problems. As you learn more about Python & use it to write more code, someone might look over your shoulder 1 day & say, "Wow, that's some beautiful code!"

```
Simple is better than complex.
```

If you have a choice between a simple & a complex solution, & both work, use the simple solution. Your code will be easier to maintain, & it will be easier for you & others to build on that code later on.

```
Complex is better than complicated.
```

Real life is messy, & sometimes a simple solution to a problem is unattainable. In that case, use the simplest solution that works.

```
Readability counts.
```

Even when your code is complex, aim to make it readable. When you're working on a project that involves complex coding, focus on writing informative comments for that code.

```
There should be one-- and preferably only one --obvious way to do it.
```

If 2 Python programmers are asked to solve the same problem, they should come up with fairly compatible solutions. This is not to say there's no room for creativity in programming. On the contrary! But much of programming consists of using small, common approaches to simple situation within a larger, more creative project. The nuts & bolts of your programs should make sense to other Python programmers.

```
Now is better than never.
```

You should spend the rest of your life learning all the intricacies of Python & of programming in general, but then you'd never complete any projects. Don't try to write perfect code; write code that works, & then decide whether to improve your code for that project or move on to something new.

As you continue to the next chapter & start digging into more involved topics, try to keep this philosophy of simplicity & clarity in mind. Experienced programmers will respect your code more & will be happy to give you feedback & collaborate with you on interesting projects." – Matthes, 2019, pp. 30–31

```
>>> import this
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```

4.2.7 Summary

"In this chapter you learned to work with variables. You learned to use descriptive variable names & how to resolve name errors & syntax errors when they arise. You learned what strings are & how to display strings using lowercase, uppercase, & title case. You started using whitespace to organize output neatly, & you learned to strip unneeded whitespace from different parts of a string. You started working with integers & floats, & learned some of the ways you can work with numerical data. You also learned to write explanatory comments to make your code easier for you & others to read. Finally, you read about the philosophy of keeping your code as simple as possible, whenever possible." – Matthes, 2019, p. 32

4.3 Introducing Lists

"... learn what lists are & how to start working with the elements in a list. Lists allow you to store sets of information in 1 place, whether you have just a few items or millions of items. Lists are 1 of Python's most powerful features readily accessible to new programmers, & they tie together many important concepts in programming." – Matthes, 2019, p. 33

4.3.1 What Is a List?

"A *list* is a collection of items in a particular order. You can make a list that includes the letters of the alphabet, the digits from 0–9, or the names of all the people in your family. You can put anything you want into a list, & the items in your list don't have to be related in any particular way. Because a list usually contains >1 element, it's a good idea to make the name of your list plural, such as `letters, digits`, or `names`.

In Python, square brackets (`[]`) indicate a list, & individual elements in the list are separated by commas. Here's a simple example of a list that contains a few kinds of bicycles:

```
bicycles = ['trek', 'cannondale', 'redline', 'specialized']
print(bicycles)
```

If you ask Python to print a list, Python returns its representation of the list, including the square brackets:

```
['trek', 'cannondale', 'redline', 'specialized']
```

Because this isn't the output you want your users to see, let's learn how to access the individual items in a list." – Matthes, 2019, pp. 33–34

4.3.1.1 Accessing Elements in a List

"Lists are ordered collections, so you can access any element in a list by telling Python the position, or *index*, of the item desired. To access an element in a list, write the name of the list followed by the index of the item enclosed in square brackets. E.g., let's pull out the 1st bicycle in the list `bicycles`:

```
bicycles = ['trek', 'cannondale', 'redline', 'specialized']
print(bicycles[0])
```

The syntax for this is shown at line 2. When we ask for a single item from a list, Python returns just that element without square brackets: `trek`. This is the result you want your users to see – clean, neatly formatted output. You can also use the string methods from Chap. 2 on any element in this list. E.g., you can format the element `'trek'` more neatly by using the `title()` method:

```
bicycles = ['trek', 'cannondale', 'redline', 'specialized']
print(bicycles[0].title())
```

This example produces the same output as the preceding example except `'Trek'` is capitalized." – Matthes, 2019, p. 34

4.3.1.2 Index Positions Start at 0, Not 1

"Python considers the 1st item in a list to be at position 0, not position 1. This is true of most programming languages, & the reason has to do with how the list operations are implemented at a lower level. If you're receiving unexpected results, determine whether you are making a simple off-by-1 error.

The 2nd item in a list has an index of 1. Using this counting system, you can get any element you want from a list by subtracting 1 from its position in the list. E.g., to access the ith item in a list, you request the item at index $i - 1$. The following asks for the bicycles at index `1` & index `3`:

```
bicycles = ['trek', 'cannondale', 'redline', 'specialized']
print(bicycles[1])
print(bicycles[3])
```

This code returns the 2nd & 4th bicycles in the list:

```
cannondale
specialized
```

Python has a special syntax for accessing the last element in a list. By asking for the item at index `-1`, Python always returns the last item in the list:

```
bicycles = ['trek', 'cannondale', 'redline', 'specialized']
print(bicycles[-1])
```

This code returns the value `'specialized'`. This syntax is quite useful, because you'll often want to access the last items in a list without knowing exactly how long the list is. This convention extends to other negative index values as well. The index `-2` returns the 2nd item from the end of the list, the index `-3` returns the 3rd item from the end, & so forth." – Matthes, 2019, p. 35

4.3.1.3 Using Individual Values from a List

"You can use individual values from a list just as you would any other variable. E.g., you can use `f`-strings to create a message based on a value from a list. Let's try pulling the 1st bicycle from the list & composing a message using that value.

```
bicycles = ['trek', 'cannondale', 'redline', 'specialized']
message = f"My first bicycle was a {bicycles[0].title()}."
print(message)
```

At line 2, we build a sentence using the value at `bicycles[0]` & assign it to the variable `message`. The output is a simple sentence about the 1st bicycle in the list: `My first bicycle was a Trek.`" – Matthes, 2019, pp. 35–36

4.3.2 Changing, Adding, & Removing Elements

"Most lists you create will be dynamic, meaning you'll build a list & then add & remove elements from it as your program runs its course. E.g., you might create a game in which a player has to shoot aliens out of the sky. You could store the initial set of aliens in a list & then remove an alien from the list each time one is shot down. Each time a new alien appears on the screen, you add it to the list. Your list of aliens will increase & decrease in length throughout the course of the game." – Matthes, 2019, p. 36

4.3.2.1 Modifying Elements in a List

"The syntax for modifying an element is similar to the syntax for accessing an element in a list. To change an element, use the name of the list followed by the index of the element you want to change, & then provide the new value you want that item to have. E.g., let's say we have a list of motorcycles, & the 1st item in the list is `'honda'`. How would we change the value of this 1st item?

```
motorcycles = ['honda', 'yamaha', 'suzuki']
print(motorcycles)
motorcycles[0] = 'ducati'
print(motorcycles)
```

The code at line 1 defines the original list, with `'honda'` as the 1st element. The code at line 3 changes the value of the 1st item to `'ducati'`. The output shows that the 1st item has indeed been changed, & the rest of the list stays the same:

```
['honda', 'yamaha', 'suzuki']
['ducati', 'yamaha', 'suzuki']
```

You can change the value of any item in a list, not just the 1st item." – Matthes, 2019, pp. 36–37

4.3.2.2 Adding Elements to a List

"You might want to add a new element to a list for many reasons. E.g., you might want to make new aliens appear in a game, add new data to a visualization, or add new registered users to a website you've built. Python provides several ways to add new data to existing lists." – Matthes, 2019, p. 37

4.3.2.2.1 Appending Elements to the End of a List. "The simplest way to add a new element to a list is to *append* the item to the list. When you append an item to a list, the new element is added to the end of the list. Using the same list we had in the previous example, we'll add the new element `'ducati'` to the end of the list:

```
motorcycles = ['honda', 'yamaha', 'suzuki']
print(motorcycles)
motorcycles.append('ducati')
print(motorcycles)
```

The `append()` method at line 3 adds `'ducati'` to the end of the list without affecting any of the other elements in the list:

```
['honda', 'yamaha', 'suzuki']
['honda', 'yamaha', 'suzuki', 'ducati']
```

The `append()` method makes it easy to build lists dynamically. E.g., you can start with an empty list & then add items to the list using a series of `append()` calls. Using an empty list, let's add the elements `'honda'`, `'yamaha'`, & `'suzuki'` to the list:

```
motorcycles = []
motorcycles.append('honda')
motorcycles.append('yamaha')
motorcycles.append('suzuki')
print(motorcycles)
```

The resulting list looks exactly the same as the lists in the previous examples: `['honda', 'yamaha', 'suzuki']`. Building lists this way is very common, because you often won't know the data your users want to store in a program until after the program is running. To put your users in control, start by defining an empty list that will hold the users' values. Then append each new value provided to the list you just created." – Matthes, 2019, pp. 37–38

4.3.2.2.2 Inserting Elements into a List. "You can add a new element at any position in your list by using the `insert()` method. You do this by specifying the index of the new element & the value of the new item.

```
motorcycles = ['honda', 'yamaha', 'suzuki']
motorcycles.insert(0, 'ducati')
print(motorcycles)
```

In this example, the code at line 2 inserts the value `'ducati'` at the beginning of the list. The `insert()` method opens a space at position `0` & stores the value `'ducati'` at that location. This operation shifts every other value in the list 1 position to the right: `['ducati', 'honda', 'yamaha', 'suzuki']`." – Matthes, 2019, p. 38

4.3.2.3 Removing Elements from a List

"Often, you'll want to remove an item or a set of items from a list. E.g., when a player shoots down an alien from the sky, you'll most likely want to remove it from the list of active aliens. Or when a user decides to cancel their account on a web application you created, you'll want to remove that user from the list of active users. You can remove an item according to its position in the list or according to its value." – Matthes, 2019, pp. 38–39

4.3.2.3.1 Removing an Item Using the del Statement. "If you know the position of the item you want to remove from a list, you can use the `del` statement.

```
motorcycles = ['honda', 'yamaha', 'suzuki']
print(motorcycles)
del motorcycles[0]
print(motorcycles)
```

The code at line 3 uses `del` to remove the 1st item, `'honda'`, from the list of motorcycles:

```
['honda', 'yamaha', 'suzuki']
['yamaha', 'suzuki']
```

You can remove an item from any position in a list using the `del` statement if you know its index. E.g., here's how to remove the 2nd item, `'yamaha'`, in the list:

```
motorcycles = ['honda', 'yamaha', 'suzuki']
print(motorcycles)
del motorcycles[1]
print(motorcycles)
```

The 2nd motorcycle is deleted from the list:

```
['honda', 'yamaha', 'suzuki']
['honda', 'suzuki']
```

In both examples, you can no longer access the value that was removed from the list after the `del` statement is used." – Matthes, 2019, p. 39

4.3.2.3.2 Removing an Item Using the `pop()` Method.

"Sometimes you'll want to use the value of an item after you remove it from a list. E.g., you might want t get the *x* & *y* position of an alien that was just shot down, so you can draw an explosion at that position. In a web application, you might want to remove a user from a list of active members & then add that user to a list of inactive members.

The `pop()` method removes the last item in a list, but it lets you work with that item after removing it. The term *pop* comes from thinking of a list as a stack of items & popping 1 item off the top of the stack. In this analogy, the top of a stack corresponds to the end of a list. Let's pop a motorcycle from the list of motorcycles:

```
motorcycles = ['honda', 'yamaha', 'suzuki']
print(motorcycles)
popped_motorcycle = motorcycles.pop()
print(motorcycles)
print(popped_motorcycle)
```

We start by defining & printing the list `motorcycles` at line 1. At line 2 we pop a value from the list & store that value in the variable `popped_motorcycle`. We print the list at line 3 to show that a value has been removed from the list. Then we print the popped value at line 4 to prove that we will have access to the value that was removed. The output shows that the value `'suzuki'` was removed from the end of the list & is now assigned to the variable `popped_motorcycle`:

```
['honda', 'yamaha', 'suzuki']
['honda', 'yamaha']
suzuki
```

How might this `pop()` method be useful? Imagine that the motorcycles in the list are stored in chronological order according to when we owned them. If this is the case, we can use the `pop()` method to print a statement about the last motorcycle we bought:

```
motorcycles = ['honda', 'yamaha', 'suzuki']
last_owned = motorcycles.pop()
print(f"The last motorcycle I owned was a {last_owned.title()}.")
```

The output is a simple sentence about the most recent motorcycle we owned: `The last motorcycle I owned was a Suzuki.`" – Matthes, 2019, pp. 39–40

4.3.2.3.3 Popping Items from any Position in a List.

"You can use `pop()` to remove an item from any position in a list by including the index of the item you want to remove in parentheses.

```
motorcycles = ['honda', 'yamaha', 'suzuki']
first_owned = motorcycles.pop(0)
print(f"The first motorcycle I owned was a {first_owned.title()}.")
```

We start by popping the 1st motorcycle in the list at line 2, & then we print a message about that motorcycle at line 3. The output is a simple sentence describing the 1st motorcycle I ever owned: `The first motorcycle I owned was a Honda.`. Remember that each time you use `pop()`, the item you work with is no longer stored in the list.

If you're unsure whether to use the `del` statement or the `pop()` method, here's a simple way to decide: when you want to delete an item from a list & not use that item in any way, use the `del` statement; if you want to use an item as you remove it, use the `pop()` method." – Matthes, 2019, pp. 40–41

4.3.2.3.4 Removing an Item by Value.

"Sometimes you don't know the position of the value you want to remove from a list. If you only know the value of the item you want to remove, you can use the `remove()` method. E.g., let's say we want to remove the value `'ducati'` from the list of motorcycles.

```
motorcycles = ['honda', 'yamaha', 'suzuki', 'ducati']
print(motorcycles)
motorcycles.remove('ducati')
print(motorcycles)
```

The code at line 3 tells Python to figure out where `'ducati'` appears in the list & remove that element:

```
['honda', 'yamaha', 'suzuki', 'ducati']
['honda', 'yamaha', 'suzuki']
```

You can also use the `remove()` method to work with a value that's being removed from a list. Let's remove the value `'ducati'` & print a reason for removing it from the list:

```
motorcycles = ['honda', 'yamaha', 'suzuki', 'ducati']
print(motorcycles)
too_expensive = 'ducati'
motorcycles.remove(too_expensive)
print(motorcycles)
print(f"\nA {too_expensive.title()} is too expensive for me.")
```

After defining the list at line 1, we assign the value `'ducati'` to a variable called `too_expensive` line 3. We then use this variable to tell Python which value to remove from the list at line 4. At line 6 the value `'ducati'` has been removed from the list but is still accessible through the variable `too_expensive`, allowing us to print a statement about why we removed `'ducati'` from the list of motorcycles:

```
['honda', 'yamaha', 'suzuki', 'ducati']
['honda', 'yamaha', 'suzuki']
A Ducati is too expensive for me.
```

Remark 4.3. *The `remove()` method deletes only the 1st occurrence of the value you specify. If there's a possibility the value appears more than once in the list, you'll need to use a loop to make sure all occurrences of the value are removed." – Matthes, 2019, pp. 41–42*

4.3.3 Organizing a List

"Often, your lists will be created in an unpredictable order, because you can't always control the order in which your users provide their data. Although this is unavoidable in most circumstances, you'll frequently want to present your information in a particular order. Sometimes you'll want to preserve the original order of your list, & other times you'll want to change the original order. Python provides a number of different ways to organize your lists, depending on the situation." – Matthes, 2019, p. 43

4.3.3.1 Sorting a List Permanently with the `sort()` Method

Python's `sort()` method makes it relatively easy to sort a list. Imagine we have a list of cars & want to change the order of the list to store them alphabetically. To keep the task simple, let's assume that all the values in the list are lowercase.

```
cars = ['bmw', 'audi', 'toyota', 'subaru']
cars.sort()
print(cars)
```

The `sort()` method, shown at line 2, changes the order of the list permanently. The cars are now in alphabetical order, & we can never revert to the original order: `['audi', 'bmw', 'subaru', 'toyota']`. You can also sort this list in reverse alphabetical order by passing the argument `reverse=True` to the `sort()` method. The following example sorts the list of cars in reverse alphabetical order:

```
cars = ['bmw', 'audi', 'toyota', 'subaru']
cars.sort(reverse=True)
print(cars)
```

Again, the order of the list is permanently changed: `['toyota', 'subaru', 'bmw', 'audi']`." – Matthes, 2019, pp. 43–44

4.3.3.2 Sorting a List Temporarily with the `sorted()` Function

"To maintain the original order of a list but present it in a sorted order, you can use the `sorted()` function. The `sorted()` function lets you display your list in a particular order but doesn't affect the actual order of the list. Let's try this function on the list of cars.

```
cars = ['bmw', 'audi', 'toyota', 'subaru']
print("Here is the original list:")
print(cars)
print("\nHere is the sorted list:")
print(sorted(cars))
print("\nHere is the original list again:")
print(cars)
```

We 1st print the list in its original order at line 2 & then in alphabetical order at line 4. After the list is displayed in the new order, we show that the list is still stored in its original order at line 6.

```
Here is the original list:
['bmw', 'audi', 'toyota', 'subaru']
Here is the sorted list:
['audi', 'bmw', 'subaru', 'toyota']
Here is the original list again:
['bmw', 'audi', 'toyota', 'subaru']
```

Notice that the list still exists in its original order at line 5 after the `sorted()` function has been used. The `sorted()` function can also accept a `reverse=True` argument if you want to display a list in reverse alphabetical order.

Remark 4.4. *Sorting a list alphabetically is a bit more complicated when all the values are not in lowercase. There are several ways to interpret capital letters when determining a sort order, & specifying the exact order can be more complex than we want to deal with at this time. However, most approaches to sorting will build directly on what you learned in this section." – Matthes, 2019, pp. 44–45*

4.3.3.3 Printing a List in Reverse Order

"To reverse the original order of a list, you can use the `reverse()` method. If we originally stored the list of cars in chronological order according to when we owned them, we could easily rearrange the list into reverse chronological order:

```
cars = ['bmw', 'audi', 'toyota', 'subaru']
print(cars)
cars.reverse()
print(cars)
```

Notice that `reverse()` doesn't sort backward alphabetically; it simply reverses the order of the list:

```
['bmw', 'audi', 'toyota', 'subaru']
['subaru', 'toyota', 'audi', 'bmw']
```

The `reverse()` method changes the order of a list permanently, but you can revert to the original order anytime by applying `reverse()` to the same list a 2nd time." – Matthes, 2019, p. 45

4.3.3.4 Finding the Length of a List

"You can quickly find the length of a list by using the `len()` function. The list in this example has 4 items, so its length is 4:

```
>>> cars = ['bmw', 'audi', 'toyota', 'subaru']
>>> len(cars)
4
```

You'll find `len()` useful when you need to identify the number of aliens that still need to be shot down in a game, determine the amount of data you have to manage in a visualization, or figure out the number of registered users on a website, among other tasks.

Remark 4.5. *Python counts the items in a list starting with 1, so you shouldn't run into any off-by-1 errors when determining the length of a list." – Matthes, 2019, p. 45*

4.3.4 Avoiding Index Errors When Working with Lists

"1 type of error is common to see when you're working with lists for the 1st time. Let's say you have a list with 3 items, & yo ask for the 4th item:

```
motorcycles = ['honda', 'yamaha', 'suzuki']
print(motorcycles[3])
```

This example results in an *index error*:

```
Traceback (most recent call last):
  File "motorcycles.py", line 2, in <module>
    print(motorcycles[3])
IndexError: list index out of range
```

Python attempts to give you the item at index 3. But when it searches the list, no item in `motorcycles` has an index of 3. Because of the off-by-1 nature of indexing in lists, this error is typical. People think the 3rd item is item number 3, because they start counting at 1. But in Python the 3rd item is number 2, because it starts indexing at 0.

An index error means Python can't find an item at the index you requested. If an index error occurs in your program, try adjusting the index you're asking for by 1. Then run the program again to see if the results are correct.

Keep in mind that whenever you want to access the last item in a list you use the index `-1`. This will always work, even if your list has changed size since the last time you accessed it:

```
motorcycles = ['honda', 'yamaha', 'suzuki']
print(motorcycles[-1])
```

The index `-1` always returns the last item in a list, in this case the value `'suzuki'`. The only time this approach will cause an error is when you request the last item from an empty list:

```
motorcycles = []
print(motorcycles[-1])
```

No items are in `motorcycles`, so Python returns another index error:

```
Traceback (most recent call last):
  File "motorcyles.py", line 3, in <module>
    print(motorcycles[-1])
IndexError: list index out of range
```

Remark 4.6. *If an index error occurs & you can't figure out how to resolve it, try printing your list or just printing the length of your list. Your list might look much different than you thought it did, especially if it has been managed dynamically by your program. Seeing the actual list, or the exact number of items in your list, can help you sort out such logical errors." – Matthes, 2019, pp. 46–47*

4.3.5 Summary

"In this chapter you learned what lists are & how to work with the individual items in a list. You learned how to define a list & how to add & remove elements. You learned to sort lists permanently & temporarily for display purposes. You also learned how to find the length of a list & how to avoid index errors when you're working with lists." – Matthes, 2019, p. 48

4.4 Working with Lists

4.5 `if` Statements

4.6 Dictionaries

4.7 User Input & `while` Loops

4.8 Functions

4.9 Classes

4.10 Files & Exceptions

4.11 Testing Your Code

Part II: Projects

Project 1: Alien Invasion

4.12 A Ship that Fires Bullets

4.13 Aliens!

4.14 Scoring

Project 2: Data Visualization

4.15 Generating Data

4.16 Downloading Data

4.17 Working with APIs

Project 3: Web Applications

4.18 Getting Started with Django

4.19 User Accounts

4.20 Styling & Deploying an App

Afterword

4.21 Appendix A: Installation & Troubleshooting

4.22 Appendix B: Text Editors & IDEs

4.23 Appendix C: Getting Help

4.24 Appendix D: Using Git for Version Control

Bibliography

Chacon, Scott and Ben Straub (2014). *Pro Git*. 2nd. Apress, p. 458.

Knuth, Donald Ervin (1997). *The Art of Computer Programming. Volume 1: Fundamental Algorithms*. 3rd edition. Addison-Wesley Professional, pp. xx+652.

Matthes, Eric (2019). *Python Crash Course: A Hands-on, Project-based Introduction to Programming*. 2nd edition. No Starch Press, pp. xxxvi+506.