

Index

Special Characters

`*/`, 44
`/*`, 44
`+` (addition) operator, 515, 518, 519
`&&` (AND) operator, 520, 686–687
`\` (backslash), 123, 124, 502, 503
``` (backtick character), 502  
`!` (“bang” operator), 508  
`\b` (backspace), 503  
`{ }` curly braces, 40, 503, 510  
`--` (decrement) operator, 519  
`**` (exponentiation) operator, 519  
`/` (division) operator, 515, 516, 518  
`//` double-slash, 43, 482. *See also* `//` floor division operator  
`"` double quote, 123, 124, 503  
`<h1>` element, 685–686, 685–686  
`==` (equality) operator, 29, 32, 517–519  
`\f` (form feed), 503  
`//` floor division operator, 125, 126, 165  
`>` (greater than) operator, 517  
`>=` (greater than or equal to) operator, 517  
`#` hash symbol, 43  
`++` (increment) operator, 519, 529  
`!=` (inequality) operator, 517, 518  
`<` (less than) operator, 517  
`<=` (less than or equal to) operator, 517  
`*` (multiplication) operator, 30, 515, 518

`\n` (new line), 142, 503  
`!` (NOT) operator, 520  
`!!` operator, 507  
`||` (OR) operator, 520  
`\r` (carriage return), 503  
`%` (remainder) operator, 518  
`(=)` single equals sign, 29, 571  
`'` single quote, 503  
`[ ]` square brackets, 27, 172, 511  
`===` (strict equality) operator, 517  
`!==` (strict inequality) operator, 517  
`-` (subtraction) operator, 515, 518  
`===` (triple equals) operator, 564  
`\t` (tab), 503

## A

`abs()` function, 132  
abstraction, 598  
A/B testing, 358  
abusers, 325  
action attribute, 367, 370  
`addEventListener()` method, 615–621, 625, 631, 633  
advantages, 618  
basic syntax for, 616  
code example, 617  
event, 616  
example of, 618–619

- `addEventListener()` method (*continued*)
  - load event, 619
  - options, 617
  - properties, 617–618
- AJAX techniques. *See* asynchronous JavaScript and XML (AJAX) techniques
- `alert()`, 467, 631, 735
- ALICE, 294
- `align-items` property, 433, 440
- alignment formatting, 143–146
- `almost_midnight` variable, 160
- Amazon’s Alexa, 295
- American Standard Code for Information Interchange (ASCII), 12
- Andreesen, Marc, 19
- Android app development
  - Java, 63
  - Kotlin, 63
- ANNs. *See* artificial neural networks (ANNs)
- anonymous functions, 579, 581
- Anthropic’s Claude, 288, 306
- APIs. *See* application programming interfaces (APIs)
- `append()` method, 174, 175
- Apple Pages, 339–340
- Apple Safari, 2, 341, 375, 379
- Apple’s Siri, 295
- application programming interfaces (APIs), 61, 79, 206, 332–333
  - PHP (Hypertext Preprocessor), 65
- `app_title` variable, 198
- arguments, 40, 132, 160, 162, 173, 193–194, 568, 725–726
  - passing, 568, 571, 588–594
    - callback function, 575
    - parameter, 573
    - setting default parameters, 575–577
    - using arguments object, 574–575
    - using rest parameters, 574
  - return value, 578
- arguments*, 39
- arithmetic assignment operators, 32
- arithmetic operators, 31, 516, 518–519
  - Python, 125–126
- `Array()` constructor, 540–541
- array literal notation, 541
- `Array.map()` method, 688, 689
- `Array.reduce` method, 82
- arrays, 511
  - Array object, 539
  - callback functions, 549–551
  - creating and viewing, 538
    - using `Array()` constructor, 540–541
    - using array literal notation, 541
    - using `split()` function, 541–542
  - description of, 538
  - destructuring, 554
  - elements, 538, 539
    - accessing, 542–543
    - deletion of, 544
  - filtering, 553–554
  - immutable array, 540

- important aspects of, 539
- index, 538
- and lists, 26–27
- `map()` array function, 552–553
- mapping, 552–553
- methods, 539, 544–546
  - looping with, 549–554
- modifying, 543
- mutable array, 540
- `pop()` method, 546
- popping, 546–547
- properties, 539
- prototype, 539
- `push()` function, 546–547
- pushing, 546–547
- `reduce()` function, 551–552
- reducing, 551–552
- shifting, 547–548
- `shift()` method, 547
- `slice()` method, 548
- slicing, 548
- `splice()` method, 549
- splicing, 549
- spreading, 554–555
- unshifting, 547–548
- `unshift()` method, 548
- arrow functions, 581, 728–729
  - limits of, 582–583
  - simplifying, 581–582
- arrow module, 169
- arrow syntax, 581
- artificial intelligence (AI), 18, 221, 225
  - agents, workflows, 310
  - chatbots, 54, 74, 84, 86, 96
  - coding help from, 1
  - creating, reading, updating, and deleting (CRUD) with, 74–79
  - definition, 288
  - linting with, 84–85
    - detect bad code, 84–85
    - static code analysis, 85
  - pair programming, 86–87
    - backseat navigator, 87
    - driver-navigator, 87
    - ping-pong, 87
    - pros and cons of, 87–88
    - tour guide, 87
  - produce or transform, 9
  - recommendation system, 221
  - services to write Python code, 96
    - as tutor, 86
      - to avoid potential pitfalls, 86
      - in education, 86
    - write template, 70–73
- artificial neural networks (ANNs), 288, 289, 291
- assembler, 52
- assembly language, 50–53
  - architecture to write, 52
  - assembler, 52
  - sandbox, 52
  - simple examples, 52

- assignment operators, 29, 31, 128, 516, 517
- associativity
  - left associativity, 516
  - right associativity, 516
- async/await
  - doProcessing() function, 640–641
  - error handling with, 642–643
- asynchronous JavaScript
  - async function
    - await expression, 639
    - converting nested callbacks to, 639–641
    - converting promise chains to, 641–642
  - callback function, 633–634
  - definition of, 630
  - event handlers, 631–632
  - user interface, 632
- asynchronous JavaScript and XML (AJAX) techniques
  - features of, 643
  - Fetch API, 644–645
  - fetch init object, 648
  - handling fetch() errors, 646–647
  - Response object, 644–646
- attention, 296
- attributes, 343–344, 350, 351
  - action attribute, 367, 370
  - class attribute, 408–410
  - colspan attribute, 364, 365
  - deprecated attributes, 401
  - hidden attribute, 343
  - href attribute, 350

- id attribute, 408–409
- lang attribute, 344
- rowspan attribute, 364, 365
- rules, 344
- style attribute, 401
- title attribute, 343, 344

automatic semicolon insertion, 460

automating web tasks, 65

automation, 20, 61, 325

await expressions, 639, 643

aware datetime, 167

## B

Babel, 681, 682

back-end, 55

- PHP (Hypertext Preprocessor), 65

back end code, 449

back-end tasks, 21

background-image property, 387–391

- background-attachment property, 390–391
- background-position property, 389
- background-repeat property, 389–390
- background-size property, 388–389

backseat navigator, 87

banish boring tasks, 70

- artificial intelligence (AI) write template, 70–73
- creating, reading, updating, and deleting (CRUD) with artificial intelligence (AI), 74–79
- identification, 70

base classes, 599–600

- Beatty, Joy, 314
- Beck, Kent, 325
- Bell Labs, 254
- Berners-Lee, Tim, 346
- BertViz, 297
- big data, 63
- bigInt data type, 506–507
- binary digit/bit, 12
- bind() method, 586, 705, 727
- Bing.com, 696, 697
- birthdate, 164
- bitwise operators, 521
- bloaters, 325
- blob() method, 646
- blockchain, 66
- block comments, 482–483
- blocking, 613, 614
- block-scoped variables, 512, 530, 531
- block syntax, 34, 35
- boilerplate code, 70
- Boolean data type, 507–508
- Boolean literals, 29
- Boolean operators, 127
- Booleans, Python data types, 124–125
- Booth, Kathleen, 51
- bootloader, 8
- Bootstrap, 690
- borderRadius property, 692
- Box.com, 362
- box model, 397, 414–415
- box positioning, 415–419

- branching statements. *See* conditional statements
- Brave, 341
- break mode, 45, 46
- breakpoint, 45, 324
- break statements, 188, 528, 535
- browser-based JavaScript, 21, 41
- buffer, 58
- built-in datasets
  - files and URLs loading, 246–249
    - comma-separated values (CSV) file, 246–247
    - Excel spreadsheets, 247–248
    - tab-delimited text files, 248–249
    - from web, 249
  - sampling, 250–251
  - Scikit-learn datasets, 244–246, 251
    - iris datasets, 244–245
    - other datasets, 245–246
- built-in functions, 54, 79, 131–135
- built-in help system, Python, 101–103
- built-in JavaScript objects, 570–571
- byte, 12

## C

- C, 11, 58
  - compilers and interpreters, 58
  - embedded systems, 58
  - game development, 58
  - high-performance applications, 58
  - operating systems, 58
- C#, 60–61

- C++, 59–60
  - game development, 59
  - graphics and virtual reality (VR) applications, 59
  - high-performance software, 59
  - operating systems and embedded systems, 59
  - web browsers, 59
- `calculate_total()`, 194–195
- callback functions, 575, 588, 633–635
  - array methods, 549–551
  - `async/await`, 640–643
  - conversion to `async` functions, 639–641
  - `reject` arguments, 635
  - `resolve` arguments, 635
- camel case, 25, 128, 462, 498, 573, 682, 692, 724
- Cascading Style Sheets (CSS), 55, 690–692
  - accessibility, 375
  - basics, 374–375
  - `border-collapse` property, 403
  - code management, 375
  - grid layout, 439–442
    - example, 441–442
    - with practice, 442–443
    - properties, 440–441
  - history, 374–375
  - modules, 693–694
  - with practice, 394–395
  - structure, 375–380
    - element selection, 375–377
    - inspection, 378–380
    - properties and values, 377–378
  - styling properties, 391–393
    - embedded, 392
    - inline, 392
    - separate style sheets, 393
  - tasks and selectors, 380–391
    - background images, 387–391
    - font properties, 380–384
    - interactive link, 385–387
- `catch()` handler, 634
- Chacon, Scott, 493
- chaining function calls, 553
- chaining functions, 594–595
- change preventers, 326
- charset meta tag, 423
- charts, 261–266
  - bar charts, 262
  - boxplot, 264–265
  - histograms, 263
  - scatterplot, 265–266
- chatbots, 54, 74, 84, 86, 96, 306–307
- ChatGPT, 71, 73–75, 77, 78, 82, 83, 145, 166, 288, 295, 300, 306, 450. *See also* artificial intelligence (AI)
- creating, reading, updating, and deleting (CRUD) craft with, 74–79
- generate HTML template, 71–73
- cheat sheet, 3
- `ChildComponent`, 698–699
- `children` prop, 719–720
- child selector, 405–407
- `choice()`, 215
- `chr()` function, 151
- `class` attribute, 408–410, 682

- class components, 704–708, 711, 714, 716, 718, 726–729
- classes, 117
  - base classes, 599–600
  - class fields syntax, 604
  - constructor, 601–603
  - creating methods, 603
  - creation of, 607–609
  - derived classes, 600–601, 603–604
  - expressions, 600
  - fields, 604–605
  - overriding methods, 603–604
  - private members of, 605
  - public members of, 605
  - static members of, 606
- class fields syntax, 604
- `class` keyword, 103
- Claude Code, 310, 324
- Cleveland, William S., 220
- client, 64, 449, 648
- cloud computing
  - Go (Golang), 61
  - Java, 63
- cloud services, 21
  - Rust, 66
- CMS. *See* content management systems (CMS)
- code, definition of, 9
- Codecademy.com, 412, 413
- code cell, 111–113
- code completion, 80
- code efficiency with functions, 192–195
- code generation, 306, 317–319
  - acceptance criteria, 317
  - artificial intelligence (AI) limitations, 319
  - coding practice, 317–318
  - examples and context, 319
  - follow-up questions, 318
  - learning, 319
  - official documentation checking, 318
  - security prioritization, 319
  - specific, 318
  - stepwise thinking, 318
  - tests, 317
  - tools updates, 319
- Coder*, 100
- coders, 9, 12, 16, 21, 29, 36, 45, 49, 88, 171, 180, 192, 199, 209, 461
- code testing, 327–329
  - AI-generated test, 328
  - planning process, 327–328
  - test-driven development (TDD), 329
- code writing
  - in Jupyter Notebook, 109
    - closing file, 113
    - folder creation, 109
    - markdown cells, 112–113
    - typing and running, 111–112
  - in Python, 104–106, 108–109
    - closing file, 109
    - open folder in Visual Studio Code (VS Code), 104–105
    - running in Visual Studio Code (VS Code), 106–108
    - saving code, 106

- coding
  - automation, 20
  - break down project, 15
  - build anything, 17
  - creativity, 17
  - data science, 20
  - as future, 19
  - games, 21
  - high-paying jobs, 18
  - learning, 18–19
  - logical reasoning skills, 15
  - machine learning, 20
  - mobile apps, 21
  - real-world uses of, 20–21
  - scientific computing and engineering, 20
  - smart devices, 21
  - “three-and-a-half Rs” of, 13
    - repeat, 13
    - revise, 13
    - run, 13
    - write, 13
  - universal language, 17–18
  - web development, 20
  - web page development, 21
  - web server development, 21
- coding structures, Python
  - conditionals, 181
    - add else statement, 182–183
    - elif statement, 183–184
    - if statement, 181–182
  - lists, 172
    - adding and removing items, 174–176
    - changing item, 174
    - getting item, 173–174
    - making, 172
    - from range of numbers, 173
    - search, 176
  - loops, 185
    - break statement, 188
    - collection of things, 185–187
    - interruption, 188
    - while, 187–188
  - color property, 378, 379, 382–383
  - colspan attribute, 364, 365
  - combining operators, 521
  - command-line tools, 61
  - comma-separated value (CSV) files, 243, 246–247, 249, 251, 541
  - comments, 43–44, 47, 81, 87, 112, 189, 330, 332–333
    - Python, 119
  - comparison expressions, 29, 32–33
  - comparison operators, 32, 33, 517–518
    - Python, 126
  - compilers, 12, 58, 68
  - complex numbers, Python data types, 122
  - componentDidMount() method, 709–710
  - componentDidUpdate() method, 712, 714
  - component lifecycle
    - in function components, 713–717
    - lifecycle methods, 706
    - mounting methods
      - componentDidMount() method, 709–710
      - constructor() method, 707

- finishing the mount, 709–710
- `getDerivedStateFromProps()`
  - method, 707–708
- getting the derived state, 707–708
- rendering the output, 708–709
- `render()` method, 708–709
- setting state in the constructor, 707
- unmounting method, 712–713
- updating methods
  - `componentDidUpdate()` method, 712, 714
  - finishing the update, 712
  - `getSnapshotBeforeUpdate()`
    - method, 712
  - getting a snapshot, 712
  - `render()` method, 711
  - `shouldComponentUpdate()`
    - method, 711
- components, React
  - adding global styles, 689–690
  - Bing.com, 696, 697
  - built-in components, 682–683
  - class components, 704–706
  - composing components, 717–722
  - function components, 702–704, 713–717
  - HTML components, 681
  - lifecycle
    - mounting methods, 706–710
    - unmounting method, 706, 712–713
    - updating methods, 706, 710–712
  - props, 698–699
  - re-rendering, 702
  - returning valid values, 697–698
  - state, 699–701
  - styling strategies, 693–694
  - using local styles, 690
  - using `style` attribute, 691, 692
  - using style modules, 692–693
  - using style objects, 691–692
- `componentWillUnmount()` lifecycle
  - method, 712–713, 716
- composing components, 717
  - with custom hooks, 721–722
  - extending a class, 717–718
  - using `children` prop, 719–720
  - using explicit props, 718–719
- computer programming, 79, 129, 293, 471
- computers, 8, 9, 11–13, 16, 20, 42, 50–52
  - Coordinated Universal Time (UTC), 166
  - dates and times for scheduling or
    - calculating, 155–157, 161, 164, 167
  - decisions, 126
  - memory, 24, 25
  - numbers, 131
  - process, 12
  - program, 74, 79, 129
- concatenation operator, 32, 145, 147–149, 519
- conditional execution, 34
- conditional operator, 524, 687
- conditional rendering, 520
  - with `&&`, 686–687
  - with conditional operator, 687
  - with element variables, 684–686

- conditional statements, 34
  - decisions making, 34
    - `if . . . else` statement, 35–36, 523–524
    - true or false, 34–35
  - Python, 181
    - `add else` statement, 182–183
    - `elif` statement, 183–184
    - `if` statement, 181–182
  - `switch` statement, 525–529
  - ternary operator, 524–525
- console, JavaScript programmer, 463–465
- `console.log()`, 56
- constants
  - naming, 499
  - read-only reference, 499
  - uses, 499
  - using `const`, 498
  - vs variables, 498
- constructor, 601–603
- constructor function, 527, 540–541, 560–562, 600–602
- `constructor()` method, 707
- containers, 201, 361, 401, 412, 427, 429, 431–435
- content management systems (CMS), 64–65
  - PHP (Hypertext Preprocessor), 64–65
- `continue` statement, 188, 535
- Coordinated Universal Time (UTC), 157, 166–168
- Copilot, 82, 306
- Counter component, 705
- couplers, 326

- `Craigslist.org`, 356
- creating, reading, updating, and deleting (CRUD), 70
  - with artificial intelligence (AI), 74–79
- cross-axis, 433–434
- Cross-Origin Resource Sharing (CORS), 637
  - non-simple request, 652
  - simple request, 651
- cross-validation, 278–279
- CSS. *See* Cascading Style Sheets (CSS)
- cursive value, 377
- Cursor, 80, 472
- custom instructions, 307
- cybersecurity, 66

## D

- data aggregation, 240–241
  - and grouping, 240–241
  - simple column statistics, 240
- data dicing, 239
- data exploration, 222, 255
- DataFrames, 223, 228, 230–233, 235, 240, 242–244, 246, 249, 250
  - creation, pandas (`pandas.pydata.org`), 230–231
- data learning, 222
- data listing, 358–361
  - nesting lists, 359–361
  - ordered and unordered lists, 358–359
- data loss, 486
- data map creation, 233–235
- data preparation, 221–222

- data processing
  - aggregating data, 240–241
    - grouping and aggregating, 240–241
    - simple column statistics, 240
  - data dicing, 237–239
  - DataFrame creation, 230–231
    - pandas ([pandas.pydata.org](http://pandas.pydata.org)), 230–231
  - data slicing, 237–239
    - columns, 238–239
    - rows, 237–238
  - data validation, 231–235
    - data map creation, 233–235
    - exploratory analysis, 231–232
    - finding and removing duplicates, 233
  - integration, 241–242
  - missing data handling, 235–237
    - NaN values, 235
    - three choices, 236–237
  - NumPy ([numpy.org](http://numpy.org)), 228–230
    - array math functions, 229–230
    - arrays vs lists, 228–229
    - creating arrays, 229
  - data product development, 222
  - data reuse, 207–208
    - open file, 208–209
    - reading data from file, 210–211
    - writing data to file, 209
  - data science, 20
    - definition, 220–221
      - and artificial intelligence (AI), 221
      - and big data, 221
      - data scientist roles, 220–221
      - naming, 220
    - Kotlin, 64
    - library installation, 224–225
    - mathematical foundation, 220
    - pipeline, 221–222
      - data exploration, 222
      - data learning, 222
      - data preparation, 221–222
      - data product development, 222
      - definition, 221
      - visualization, 222
    - python ecosystem, 223–224
      - Matplotlib ([matplotlib.org](http://matplotlib.org)), 224
      - NumPy ([numpy.org](http://numpy.org)), 223
      - pandas ([pandas.pydata.org](http://pandas.pydata.org)), 223–224
      - Scikit-learn ([scikit-learn.org](http://scikit-learn.org)), 224
      - SciPy ([scipy.org](http://scipy.org)), 224
  - data scientists
    - analysis, 220
    - data capture, 220
    - presentation, 221
  - data slicing, 237–239
    - columns, 238–239
    - rows, 237–238
  - data types, 27
    - bigInt data type, 506–507
    - Boolean data type, 507–508
    - Boolean literals, 29
    - NaN data type, 509
    - number data type, 505–506

- data types (*continued*)
  - numeric literals, 27–28
  - object, 510–511
  - primitive data types, 501
  - Python, 121
    - Booleans, 124–125
    - complex numbers, 122
    - floats, 122
    - integers, 122
    - numbers, 121–122
    - words (strings), 122–124
  - string data type, 501–505
  - string literals, 28–29
  - symbol data type, 509
  - typeof operator, 500, 501
  - undefined data type, 509
- data validation, 231–235
  - data map creation, 233–235
  - exploratory analysis, 231–232
  - finding and removing duplicates, 233
- .date(), 162, 167
- date and time management, Python, 155
  - dates and times formatting strings, 158–159
  - datetime module, 156
  - timespans calculation, 162–166
  - working with dates, 156–161
  - working with times, 160–161
  - working with time zones, 166–169
- Date() function, 527
- dates and times formatting strings, 158–159
- datetime(), 167
- datetime.date, 156
- datetime.datetime.utcnow(), 168
- datetime module, 156, 160, 162, 199
  - datetime.date, 156
  - datetime.datetime, 156
  - datetime.time, 156
- datetime.now, 164
- datetime.now(), 169
- datetime.time, 156, 160
- datetime.timedelta class, 162
- date.today(), 156
- dateutil module, 169
- days\_between, 162
- days\_old, 164, 165
- debugging, 44–45, 108–109, 322–324
  - monitoring script values, 46
  - strategies
    - break down complex tasks, 47
    - break up long statements, 47
    - comment out problem statements, 47
    - comments to document your scripts, 47
    - indent your code, 46–47
  - techniques
    - monitoring script values, 46
    - pausing code execution, 45
    - stepping through your code, 46
- declaration, 377, 378
- declaration block, 377
- deep learning, 283, 288, 289. *See also* machine learning
- default exports, 659–661

- `delta_age`, 164
- `delta_age.days`, 164
- deprecated attributes, 401
- deprecation, 168
- derived classes, 600–601, 603–604
- descendant selector, 407–408
- description meta tag, 423
- desktop computer, on Chrome, 423
- development environment, 455–459
- DevOps, 61
- dictionaries, 178–180
- `dispatchEvent()` method, 622–624
- dispensables, 325
- distributed systems, 61
- DOCTYPE declaration, 344
- documentation, 332–333
  - application programming interfaces (APIs), 332–333
  - diagrams, 333
  - generating comments, 332–333
  - internal vs external, 332
  - visual documentation, 333
- document object model (DOM), 405
- `doLogin()` function, 633, 634
- domain requirements, 311–312
- `doMath()` function, 550
- `doProcessing()` function, 640, 641
- dot notation, 42, 201, 203, 207, 510, 558, 560, 562–563
- `do...while` statement, 534–535
- driver-navigator, 87
- `dt.date.today()`, 164
- DuckDuckGo, 341

## E

- Easter egg, 116
- `ebay.com`, 412
- ecommerce websites, 65
  - PHP (Hypertext Preprocessor), 65
- EDA. *See* exploratory data analysis (EDA)
- education, artificial intelligence (AI) in, 86
- elements
  - definition, 341
  - h1 element, 341, 342, 346, 348
  - prompt engineering, 304
  - structure, 341–343
- `elif` statement, 183–184
- ELIZA, 294
- `else` statement, 182–183
- embedded Cascading Style Sheets (CSS), 392
- embedded systems
  - C, 58
  - C++, 59
  - Rust, 66
- Encapsulated Postscript (EPS), 261
- encapsulation, 598
- error handling, 642–643, 646–647
- error indications, Python, 108
- errors, 44–45
  - logic errors, 45
  - runtime errors, 45
  - syntax errors, 45
  - types, 45
- escape characters, 123
- ESLint, 85, 323

- event bubbling, 621
- event handlers, 631–632
  - properties, 615
- event loop, 613–614
- event object, 620
- events
  - `addEventListener()` method, 615–621, 625
  - `dispatchEvent()` method, 622–624
  - dispatching events, 621–624
  - event bubbling, 621
  - event loop, 613–614
  - event object, 620
  - event target, 616
  - listening on multiple targets, 621
  - listening with event handler properties, 615
  - listening with HTML event attributes, 614–615
  - preventing default actions, 625
  - `removeEventListener()` method, 624–625
  - removing event listeners, 624–625
  - supported by HTML elements, 616–617
- event target, 616
- Excel spreadsheets, 243, 246–248, 251
- explicit props, 718–719
- exploratory data analysis (EDA), 222
  - charts, 261–266
    - bar charts, 262
    - boxplot, 264–265
    - histograms, 263
    - scatterplot, 265–266
  - complete workflow, 268–269
  - correlation measures, 266–268
  - Matplotlib, 258–261
    - Jupyter Notebook command, 258–259
    - lines and legends, 260
    - plot function, 259
    - Python package installer, 258
    - saving files, 261
  - philosophy, 254
  - purpose of, 254
  - statistical measures, 255–258
    - central tendency measures, 256–257
    - variance and range, 257–258
- exponential notation, 28, 505
- exponent operation, 126
- expression*, 35–37, 181, 183, 184, 187
- expression part, 137, 138
- expressions, 29, 30, 514
  - comparison expressions, 32–33
  - logical expressions, 33–34
  - numeric expressions, 31–32
  - string expressions, 32
- external libraries, 205
  - importing and using, 206–207
  - installation, 206
- extract, transform, and load (ETL), 221

## F

- fabrication, 86
- Facebook, 20, 21, 64, 374
- Featured Links, 405, 407, 408
- Fetch API, 644–645

- fetch() method, 635, 644, 645, 654
  - error handling, 646–647
  - init object, 648
- few-shot prompt, 305–306
- file\_exists() function, 215
- files and URLs loading, 246–249
  - comma-separated values (CSV) file, 246–247
  - Excel spreadsheets, 247–248
  - tab-delimited text files, 248–249
  - from web, 249
- file.write() methods, 209, 210
- filter() method, 553–554
- finding and eliminating bugs, 322–325
  - automating dependency hygiene, 324–325
  - debugging with artificial intelligence (AI), 324
  - knowing, 322–323
  - preventing with static analysis, 323
- Firefox, 375, 379
- firewalls, 454
- firmware, 8
- fixing and optimizing code
  - code testing, 327–329
    - AI-generated test, 328
    - planning process, 327–328
    - test-driven development (TDD), 329
- documentation, 332–333
  - application programming interfaces (APIs), 332–333
  - diagrams, 333
  - generating comments, 332–333
  - internal vs external, 332
  - visual documentation, 333
- finding and eliminating bugs, 322–325
  - automating dependency hygiene, 324–325
  - debugging with AI, 324
  - identification, 322–323
  - preventing with static analysis, 323
- optimizing performance, 331–332
  - AI-assisted optimization, 331–332
  - profiler measures, 331
- refactoring, 325–327
  - with artificial intelligence (AI), detecting smells, 326
  - with code formatters, 327
  - code smells, 325
  - safety nets, 326–327
- software maintenance, 334–335
  - adaptive maintenance, 334
  - corrective maintenance, 334
  - and lifecycle, 335
  - perfective maintenance, 334
  - preventative maintenance, 334
  - technical debt, 334–335
- translating code, 329–331
  - vs dialects, 329
  - preparation, 330
  - verification, 330–331
- Flexbox layout, 426–439
  - assumptions, 432–433
  - boxes creation, 427–428
  - containers, 431–432
  - cross-axis, 433–434

## Flexbox layout (*continued*)

- flex-wrap property, 431
- main axis, 434
- modification, 434–437
  - flex-basis property, 435
  - flex-grow property, 435–436
  - flex property, 436–437
  - flex-shrink property, 436
- normal flow, 426
- in one dimension, 428–431
- order property, 437–439

Flickr, 350

floating-point data type, 25

floating-point numbers, 27–28, 122

float keyword, 25

floor division, 126

focus() method, 624

font-family property, 377, 383–384

font properties, 380–384

- color property, 382–383
  - hex code, 382
  - name, 382
  - red, green, and blue (RGB) value, 382, 388
- font-family property, 383–384
  - generic font family, 384
  - name, 383
- font-size property, 381–382
- font-style property, 383
- font-weight property, 383
- text-decoration property, 384

font-weight property, 383

for . . . in statement, 531–532

for loop, 185, 186, 210

formatting numbers, 137

- dollar amounts, 138–139
- with f-strings, 137–138
- multiline format strings, 142–143
- percent numbers, 140–142
- width and alignment, 143–146

form filling, 366–371

- basic form creation, 367–371
- form element, 367, 368
- form processing, 366–367
- input element, 368

for . . . of statement, 532–533

for statement, 530–531

Fowler, Martin, 325

from dateutil.tz import gettz, 169

front-end, 55

- code, 449–453
- web application, 526, 527

f-strings, 157, 159

- formatting, 137–139, 142, 145

full correlation matrix, 266

functional bugs, 322

functional requirements, 311–313

function components, 702–704, 729–730

- lifecycle in, 713–717

*function\_name*, 39, 193

functions, 38–41

- anonymous, 579, 581

- arguments*, 39
- arrow functions, 581–583
- chaining, 594–595
- declaration, 572, 578–579
- definition of, 568
- expressions, 579–580
- function, 39
- naming, 573
- passing an object to, 585–586
- passing arguments, 568, 571, 588–594
  - callback function, 575
  - parameter, 573
  - setting default parameters, 575–577
  - using arguments object, 574–575
  - using rest parameters, 574
- passing by reference, 571–572
- passing by value, 571
- passing from one object to another, 587–588
- passing to child component, 588–594
- returning data, 577–578
- setting context of function
  - using `apply()`, 586
  - using `bind()`, 586–587
  - using `call()`, 586
- statements, 39, 40
- `String()` function, 568
- `this` keyword, 584, 585, 588
- top-level functions, 569–570
- without context, 584–585

- writing, 572–573
  - body of the function, 577
  - method, 583–584
- function-scoped variables, 512

## G

- game development
  - C, 58
  - C++, 59
  - Java, 63
  - Rust, 66
- `gamelet.js` file, 482–483
- games, 21, 54, 57, 202
- general-purpose chatbots, 306
- generative artificial intelligence (GenAI), 1, 69, 74, 300–301
  - chatbot, 74
  - chat interfaces, 82
  - code completion, 80
  - definition, 288
  - learn or remember syntax functions, 82
  - models, 80–82, 84, 86, 88
  - for natural-language processing (NLP), 301
- `getDerivedStateFromProps()` method, 707–708
- `getMonth()` function, 527
- `getSnapshotBeforeUpdate()` method, 712
- `getStateInfo()` function, 661
- `gettz`, 169
- `get_user_name()` function, 194

## Git

- basics of, 491–493
- configuration and testing, 488–491
- description of, 486–487
- file states
  - committed, 491
  - modified, 491
  - staged, 491
- installation of, 487–489
- workflow, 491–493

GitHub, 1, 50, 80, 85, 324, 493

GitHub Copilot, 1, 80, 81, 85, 91, 93–96  
in Visual Studio Code (VS Code), 94–95

global-scoped variables, 196–197, 512

Go (Golang), 61–62

- cloud computing and DevOps, 61
- command-line tools, 61
- high-performance applications, 61
- networking and distributed systems, 61
- web development, 61

good and bad Python numbers, 121–122

Google Chrome, 341, 375, 378, 379,  
415, 423

desktop computer on, 423

Google Gemini, 74, 84, 85, 306

as linter, 84, 85

Google Images, 350

Gorman, Kristen, 255

GPT-3.5, 82, 83

GPT-4, 82, 83

graphics and virtual reality (VR)  
applications, 59

Greenwich Mean Time (GMT), 166

`greet()` function, 197, 198

grouping operator, 516

## H

`h1` element, 341, 342, 346, 348

Hacker News, 319

hand-code rules, 295

heatmap, 258, 267, 268

`help()`, 101–102

high-level languages, 11, 51

high-performance applications

C, 58

C++, 59

Go (Golang), 61

hoisting, 579

hooks, 702, 721–722

Hopper, Admiral Grace, 322

hotlinking, 350, 388

human language, 10

rules, 10

words, 10

HyperText Markup Language (HTML), 55

content organization, 356–358

data listing, 358–361

nesting lists, 359–361

ordered and unordered lists, 358–359

`textarea` element, 733

element naming, 408–410

`class` attribute, 408–410

`id` attribute, 408–409

- elements, 616–617
- event attributes, 466, 614–615, 724
- form field, 730
- form filling, 366–371
  - basic form creation, 367–371
  - form element, 367, 368
  - form processing, 366–367
  - input element, 368
- history, 346
- select input, 733
- overview, 339–340
- with practice, 371–372
- structure, 340–346
  - attributes, 343–344
  - elements, 341–343
  - head, title, and body, 344–346
- styling, 352–354
  - basic elements, 352–353
  - superscript and subscript, 353–354
- table format, 361–366
  - basic structure, 361–364
  - columns and rows, 364–366
  - table element, 361–363
  - td element, 363, 364
  - th element, 363
  - tr element, 362–364
- tasks and elements, 346–351
  - headings level, 348
  - hyperlinks, 349–350
  - images, 350–351
  - paragraph text, 349
- web page design, 354

- Hypertext Transfer Protocol (HTTP), 448
  - DELETE, 649
  - description of, 648
  - GET, 649
  - Headers tab, 649
  - Network tab, 649
  - PATCH, 649
  - POST, 649
  - PUT, 649
  - status codes, 649–650
  - user agent, 648

- I**
  - id attribute, 408–409
  - IDEs, 80, 82, 319, 321, 323, 324
  - IEEE Spectrum, 50
  - if . . . else statements, 35–36, 523–524
  - if statement, 34, 35, 181–183, 524, 578
  - immutable array, 540
  - import datetime, 156, 169
  - import datetime as dt, 156
  - import locale, 140
  - import math, 134, 201, 202
  - import module, 204
  - import random, 120
  - import statement, 120, 203, 206, 660–662
  - incrementCount() method, 705
  - indentations, Python, 117–119
  - index, 173–174
  - indexing, 150
  - index number, 26, 27, 174, 542, 543, 548
  - inheritance, 598–600, 717–718

- init object, 648
- in keyword, 176
- inline Cascading Style Sheet (CSS), 392
- inline JavaScript, 466–467
- inline styling, 691
- innerHTML property, 529
- input() prompt, 182
- insert() method, 174, 175
- installation
  - external libraries, 206
  - Python, 92–94
  - Visual Studio Code (VS Code), 93, 94
- instance methods, 570
- integers, 27, 122
- integration bugs, 322
- IntelliSense text, 80, 106
- Intel x86 processor, 52
- interactive mode, Python, 98, 173
  - built-in help system, 101–103
  - exit help system, 103–104
  - interpreter, 14, 92, 101–103, 105, 107, 198, 207
  - Terminal pane, 99–100
  - version, 100
- interpretability, 292, 300
- interpreted vs compiled languages, 68
- interpreters, 11, 12, 24, 25, 68
  - C, 58
  - Python, 14, 92, 101–103, 105, 107, 198, 207
- interquartile range (IQR), 264
- iris datasets, 244–245

- isOn variable, 699, 700
- iterable, 186
- iterable object, 532
- iterate, 178
- iterative statements. *See* looping statements

## J

- Java, 62–63
  - Android app development, 63
  - big data and cloud computing, 63
  - game development, 63
  - web applications, 63
- JavaScript, 9, 41, 55–57, 67
  - add cool features, 57
  - asynchronous JavaScript (*see* asynchronous JavaScript)
  - basics, 447–454
  - build fun mini-projects, 57
  - case-sensitive code, 461
  - code documentation, 481
    - block comments, 482–483
    - line comments, 482
  - README file, 483–485
  - code written in Python, 118–119
  - coding process, 14–15
  - connect to online data, 57
  - create simple games, 57
  - development categories, 21
    - games, 21
    - mobile apps, 21
    - smart devices, 21

- web page development, 21
- web server development, 21
- development environment, 455–459
- dynamically typed language, 500
- embed code inside HTML file, 15
- expressions, 514
- external JavaScript file creation, 15
- functions (*see* functions)
- Git, 486–493
- ignoring white space, 461
- importing CSS, 690
- installation issues, 480–481
- Live Server, 478–480
- loosely typed language, 500
- make interactive web pages, 56–57
- name creator program, 459
- naming classes and components
  - camel case, 462
  - hyphens, 462–463
  - underscore (`_`), 462
- objects
  - basics, 558–559
  - built-in objects, 570–571
  - comparing and copying, 564–566
  - creation of, 559–562
  - modifying, 562–564
- operators, 515–521
- Prettier, 474–478
- promises (*see also* callback functions)
  - `catch()` handler, 634
  - definition of, 634

- `fetch()` method, 635
- fulfilled, 634
- new keyword, 635
- pending, 634
- promise chain, 634, 635
- Promise constructor, 635
- rejected, 634
- `then()` function, 635
- writing, 635–638
- `XMLHttpRequest()` method, 635–637
- React (*see* React)
- running code in browser window, 465
  - importing files in HTML, 467–470
  - inline JavaScript, 466–467
  - running from HTML event attributes, 466
- running code in console, 463–465
- runtime model, 612–613
- on server, 454
- source code, to view, 450–453
- special characters, 503
- statement
  - expressions, 460
  - keywords, 460
  - operators, 460
  - values, 460
- synchronous JavaScript (*see* synchronous JavaScript)
- JavaScript modules
  - description of, 658
  - exporting, 658

## JavaScript modules (*continued*)

- constants .js file, 658
- default exports, 659–660
- named exports, 659

export keyword, 658

### importing

- default exports, 661
- dynamic loading, 662
- into HTML, 662–663
- named exports, 661

import keyword, 658, 660

JavaScript Object Notation (JSON) data,  
246, 249, 251

example of, 653–654

getting data, 654–655

sending data, 655

Java Virtual Machine (JVM), 62

join() function, 553

*Journal of Data Science*, 220

json() method, 645, 646, 654

JSON.parse() method, 688

JSON string, 655, 687–688

JSON.stringify() method, 655

## JSX

Babel, 681, 682

built-in components, 682–683

code, 680–681

conditional rendering

- with &&, 686–687

- with conditional operator, 687

- with element variables, 684–686

event attributes, 724

JavaScript expressions in, 683–684

writing HTML output with, 681

Jupyter Notebook, 96–97, 152, 224

code writing in, 109

- closing file, 113

- folder creation, 109

- markdown cells, 112–113

- typing and running, 111–112

creation, 109–110

## K

Kayak.com, 356, 357

*kebab case*, 692

Keep-Alive instruction, 449

key, 10, 178–180, 208

key attribute, 689

keydown events, 468

key-value pairs, 178

keywords, 10, 102, 103, 181, 389, 460

Kotlin, 63–64

- Android app development, 63

- cross-platform apps, 63

- data science and machine learning, 64

- web development, 63

## L

Laravel, 65

large language models (LLMs), 296, 297,  
299, 301, 303–305

layout and alignment, 410–419

- box model, 414–415
- box positioning, 415–419
  - clear properties, 415–417, 419
  - float properties, 415–417, 419
- data organization, 410–412
  - horizontal/menu navigation, 411–412
  - vertical navigation, 411
- div elements, 412–414
- left associativity, 516
- len() function, 149, 172, 178
- length, of strings, 149
- lexical scoping, 728
- libraries, 20, 54, 199. *See also* external libraries; standard library
  - Kotlin, 63, 64
- lifecycle events, 706
- line comments, 482
- lingua franca, 17–18
- link description, 349
- link destination, 349
- linters, 84, 85, 323, 324, 330
- linting with artificial intelligence (AI), 84–85
- list(), 173
- lists, 113. *See also* data listing; styling lists
  - vs array, 228–229
- Python, 172
  - adding and removing items, 174–176
  - changing item, 174
  - getting item, 173–174
  - making, 172
  - from range of numbers, 173
  - search, 176
- literal part, 137
- literals, 27
  - Boolean literals, 29
  - numeric literals, 27–28
  - string literals, 28–29
- Live Server, 478–480
- LLMs. *See* large language models (LLMs)
- loadBook() function, 641–642
- local scope, 197–198
- logical bugs, 322
- logical expressions, 33–34, 36, 37, 181, 183, 187
- logical operators, 33, 519–520
- logic errors, 45, 46
- Long-Term Ecological Research (LTER) program, 255
- looping statements
  - do...while statement, 534–535
  - for...in statement, 531–532
  - for...of statement, 532–533
  - for statement, 530–531
  - while statement, 533–534
- loops, 36–38. *See also* looping statements
  - for, 185, 186, 210
  - event loop, 613–614
  - prototype loop, 315–316
  - Python, 185
    - break statement, 188
    - collection of things, 185–187
    - interruption, 188
  - while, 37, 38, 187–188, 195, 215
  - while True, 188

## M

- machine code, 50–52, 68
- machine language, 11, 50–51
- machine learning, 20
  - artificial intelligence (AI) limitations, 301
  - definition, 288–289
  - generative AI model, 300–301
  - iris datasets, 244
  - Kotlin, 64
  - natural-language processing (NLP), 293–296
    - challenges, 295–296
    - history, 293–295
    - purposes of, 293
  - neural networks, 289–291
    - hidden layer, 290
    - input layer, 290
    - output layer, 290–291
  - responsible artificial intelligence (AI), 299–300
    - fairness, 299–300
    - interpretability, 300
    - privacy, 300
    - safety and security, 300
  - training and testing models, 291–292
  - transformer models, 296–299
    - self-attention mechanism, 296–297
    - tokenization, 297–299
- machine-learning library, 244
- machine learning model
  - algorithms, 272
  - end-to-end workflow, 280–282
  - prediction problems, 272
- Scikit-learn ([scikit-learn.org](https://scikit-learn.org)), 272–275
  - feature matrix, 273–274
  - feature scaling, 274–275
  - target vector, 273–274
- training, 275–280
  - cross-validation, 278–279
  - model interpretation, 279–280
  - training and test sets, 276–277
- machine translation, 293
- main axis, 434
- `main()` function, 215
- `map()` array function, 552–553
- Markdown, 483–485
- markdown cells, 112–113
  - code formatting, 113
  - emphasis, 112
  - headings, 112
  - lists, 113
- math constants, 135, 137
- math module, 134–137, 199
- `math.sqrt()`, 135
- MATLAB, 223
- Matplotlib ([matplotlib.org](https://matplotlib.org)), 224, 230, 258–261
  - Jupyter Notebook command, 258–259
  - lines and legends, 260
  - plot function, 259
  - Python package installer, 258
  - saving files, 261
- mean absolute error (MAE), 277

- `mean()` function, 202
- metadata, 423. *See also* HyperText Markup Language (HTML)
- microservices, 61
- Microsoft Copilot, 306
- Microsoft Edge, 341, 375, 379
- Microsoft's Tay chatbot, 301
- Microsoft Word, 339
- minified code, 453
- missing data handling, 235–237
  - NaN values, 235
  - three choices, 236–237
- mnemonics, 51
- mobile apps, 21
- mobile-first design, 422
- mobile web design, 422
- modern data science, 254
- module*, 200, 201, 203, 204
- modulename*, 120
- modules, Python, 119–120, 199–200
  - creating and using, 203–204
  - import, 200–201, 203
  - rolling, 203–205
  - using, 201–203
- modulus operation, 126
- MongoDB database, 75, 77, 79
- Mongoose, 75, 78
- mounting methods, React components
  - `componentDidMount()` method, 709–710
  - `constructor()` method, 707
  - finishing the mount, 709–710
  - `getDerivedStateFromProps()` method, 707–708
  - getting the derived state, 707–708
  - rendering the output, 708–709
  - `render()` method, 708–709
  - setting state in the constructor, 707
- Mozilla Firefox, 341
- multiline format strings, 142–143
  - `\n`, 142
  - triple quotation marks (single or double), 143
- multiple-line comments, 43
- mutable array, 540
- mutable lists, 174

## N

- naïve datetime, 167
- named exports, 659
- naming functions, 573
- NaN data type, 509
- NaN values, 235
- natural-language generation (NLG), 293
- natural-language processing (NLP), 293–296
  - challenges, 295–296
    - supervised learning, 295
    - unsupervised learning, 295–296
  - hand-code rules, 295
  - history, 293–295
  - pattern-matching rules, 294
  - purposes of, 293
- natural-language understanding (NLU), 293
- nest functions, 134
- nesting block statements, 47

- nesting lists, 359–361
- nest quotation marks, 29
- .NET, 60
- Network Transfer Protocol (NTP), 157
- neurons, 288–291
- new keyword, 527, 560, 600–602, 622, 635
- The New York Times*, 346, 347
- Node.js, 21, 75, 77, 78, 454, 473
- non-functional requirements, 311, 313–314
  - business assumptions, 314
  - economic constraints, 314
  - environmental assumptions, 314
  - interface constraints, 314
  - life cycle constraints, 314
  - operating constraints, 314
  - operational assumptions, 314
  - performance constraints, 314
  - technical assumptions, 313
- now() method, 161
- null string or empty string, 28–29
- number data type, 505–507, 510
- numbers, 131
  - formatting, 137
    - dollar amounts, 138–139
    - with f-strings, 137–138
    - multiline format strings, 142–143
    - percent numbers, 140–142
    - width and alignment, 143–146
- Python data types, 121–122
  - complex numbers, 122
  - floats, 122
  - integers, 122

- numerical vector, 298
- numeric expressions, 31–32
- numeric literals, 27–28
  - floating-point numbers, 27–28
  - integers, 27
- NumPy ([numpy.org](http://numpy.org)), 223, 224, 255, 256
  - array math functions, 229–230
  - arrays vs lists, 228–229
  - creating arrays, 229

## O

- Object.create() method, 562
- object-oriented programming (OOP), 117
  - abstraction, 598
  - encapsulation, 598
  - inheritance, 598–599
  - polymorphism, 599
  - Python, 117
- objects, 41–43, 510–511
  - basics, 558–559
  - comparing and copying, 564–566
  - creation of
    - with class, 561–562
    - with Object.create(), 562
    - using constructor function, 560–561
    - using literal notation, 559–560
  - definition of, 558
  - events, 41, 42
  - methods, 41–43
  - modifying, 562–564
  - pencil object, 558, 559
  - properties, 41–43, 558

- Olson database or tz database, 166–167
- one-way data binding, 730
- onMouseMove() function, 620
- OpenAI, 298, 300, 306. *See also* ChatGPT
- open-ended vs closed-ended prompts, 304–305
- open() function, 208–210
- Opera, 341, 379
- operands, 30–32, 515
- operators, 30
  - arithmetic operators, 125–126, 516, 518–519
  - assignment operators, 516, 517
  - associativity, 516
  - bitwise operators, 521
  - Boolean operators, 127
  - combining operators, 521
  - comparison operators, 126, 517–518
  - concatenation operator, 519
  - grouping operator, 516
  - logical operators, 519–520
  - precedence, 515–516
  - Python, 125–127
- optimizing performance, 331–332
  - AI-assisted optimization, 331–332
  - profiler measures, 331
- ordered lists, 358–359
- ord() function, 151
- outer function, 575, 578
- out-of-bound bugs, 323
- overfitting, 276, 277, 282

## P

- Pacific Daylight Time (PDT), 168
- Pacific Standard Time (PST), 168
- pair programming with artificial intelligence (AI), 86–87
  - backseat navigator, 87
  - driver-navigator, 87
  - ping-pong, 87
  - pros and cons of, 87–88
  - tour guide, 87
- pandas (pandas.pydata.org), 223–224, 230–231
- parameters, 193, 291, 292, 300
  - parameters*, 193
- ParentComponent, 698–699
- Pencil Adventure, 558
- PEP. *See* Python enhancement proposals (PEP)
- percent number formatting, 140–142
- Phaser.js, 21
- PHP (Hypertext Preprocessor), 64–65
  - application programming interfaces (APIs) and back-end services, 65
  - automating web tasks, 65
  - content management systems, 64–65
  - ecommerce websites, 65
  - server-side web development, 64
- ping-pong style, 87
- pip, 206
- pip install requests, 207
- polymorphism, 599
- pop() method, 175, 546

- Portable Document Format (PDF), 261
- Portable Network Graphic (PNG)
  - format, 261
- Portable Operating System Interface (POSIX), 139
- postfix notation, 530
- Postscript (PS), 261
- prefab Python code, 199
- prefix notation, 529
- presentational/dumb components. *See* function components
- Prettier
  - code formatting, 476–478
  - extensions settings, 475, 476
  - installation and configuration, 474–475
  - User tab, 475
  - Workspace tab, 475
- primitive data types, 501, 510, 564, 571
- `print(dir(module))`, 201
- `print()` function, 54, 138, 140, 141, 149, 150, 157, 160, 161
- `print(message)` statement, 197
- `print()` statement, 161, 182, 185, 186, 197
- Pro Git* (Chacon and Straub), 493
- programming languages, 9–10, 49. *See also specific entries*
  - C, 58–59
  - C#, 60–61
  - C++, 59–60
  - Go (Golang), 61–62
  - Java, 62–63
  - JavaScript (*see* JavaScript)
  - Kotlin, 63–64
  - PHP (Hypertext Preprocessor), 64–65
  - Python (*see* Python)
  - ranking, 50
    - IEEE Spectrum, 50
    - TIOBE Index, 50
  - role of, 10–12
  - Rust, 65–66
  - Swift, 66–67
  - TypeScript, 67
- project requirements, 311–317
  - code generation, 316–317
  - domain requirements, 311–312
  - functional requirements, 312–313
  - loop, 315–316
  - non-functional requirements, 313–314
  - software requirements, 311
  - specification writing, 314–315
- Promise constructor, 635, 640
- promises. *See also* callback functions
  - `catch()` handler, 634
  - conversion to async functions, 641–642
  - definition of, 634
  - `fetch()` method, 635
  - fulfilled, 634
  - new keyword, 635
  - pending, 634
  - promise chain, 634, 635
  - Promise constructor, 635
  - rejected, 634
  - `then()` function, 635
  - writing, 635–638
  - `XMLHttpRequest()` method, 635–637

- `promisifiedSetTimeout()` function, 640
- `prompt()`, 631
- prompt engineering, 75, 304
  - elements of, 304
  - improving prompts, 306
  - open-ended vs closed-ended prompts, 304–305
  - types, 305–306
    - few-shot prompt, 305–306
    - zero-shot prompt, 305
- prompts, 9, 74, 84, 100–104, 107, 111, 116, 120, 132, 134, 152, 157, 173, 194, 206, 300, 304–306, 326
- prototype
  - with AI agents, 310
- code generation, 317–319
  - acceptance criteria, 317
  - AI's limitations, 319
  - coding practice, 317–318
  - examples and context, 319
  - follow-up questions, 318
  - learning, 319
  - official documentation checking, 318
  - security prioritization, 319
  - specific, 318
  - stepwise thinking, 318
  - tests, 317
  - tools updates, 319
- project requirements, 311–317
  - code generation, 316–317
  - domain requirements, 311–312
  - functional requirements, 312–313
  - loop, 315–316
  - non-functional requirements, 313–314
  - software requirements, 311
  - specification writing, 314–315
- pure function, 711
- `push()` function, 546–547
- `.py` extension, 105, 120, 157, 190, 204
- Pylance, 93
- Python, 2, 9, 20, 53–55, 91–92
  - automating repetitive tasks, 54
  - build games, 54
  - build websites, 55
  - built-in functions, 131–134
  - code writing, 104
    - closing file, 109
    - open folder in Visual Studio Code (VS Code), 104–105
    - running Python in Visual Studio Code (VS Code), 106–108
    - saving code, 106
    - writing, 105–106
- coding process, 14
- conditionals in, 181
  - `add else` statement, 182–183
  - `elif` statement, 183–184
  - `if` statement, 181–182
- control hardware and IoT projects, 55
- create apps, 54
- date and time management, 155
  - `datetime` module, 156
  - timespans calculation, 162–166
  - working with dates, 156–161
  - working with times, 160–161
  - working with time zones, 166–169

## Python (*continued*)

- debugging, 108–109
- dictionaries, 178–180
- experiment with AI and machine learning, 54
- function in, 131
- installation, 92–94
- interactive mode, 98
  - built-in help system, 101–103
- interpreter, 14, 92, 101–103, 105, 107, 198, 207
- Terminal pane, 99–100
- version, 92, 100
- internationalization, 139–140
- JavaScript code written in, 118–119
- lists, 172
  - adding and removing items, 174–176
  - changing item, 174
  - getting item, 173–174
  - making, 172
  - from range of numbers, 173
  - search, 176
- loops, 185
  - break statement, 188
  - collection of things, 185–187
  - interruption, 188
  - while, 187–188
- numbers, 131
  - formatting, 137–146
- online data sources, 54
- sets, 180–181
- setup verification, 97–98

- strings manipulation, 147
  - common string operators, 149–152
  - concatenation, 147–149
  - length, 149
  - methods, 152–154
- tools, 92–98
  - GitHub Copilot, 93–96
  - Jupyter Notebook, 96–97
  - setup verification, 97–98
  - Visual Studio Code (VS Code), 92–94
- tuples, 177–178
- work and analyze data, 54

Python, design principles, 115

- comments, 119
- data types, 121
  - Booleans, 124–125
  - complex numbers, 122
  - floats, 122
  - integers, 122
  - numbers, 121–122
  - words (strings), 122–124
- indentations, 117–119
- modules, 119–120
- object-oriented programming (OOP), 117
- operators, 125
  - arithmetic operators, 125–126
  - Boolean operators, 127
  - comparison operators, 126
- variables, 127
  - creation in code, 128
  - manipulation, 129

- valid variable names, 127–128
- Zen of Python, 116–117
- Python data science ecosystem, 223–224, 258
  - Matplotlib ([matplotlib.org](http://matplotlib.org)), 224
  - NumPy ([numpy.org](http://numpy.org)), 223
  - pandas ([pandas.pydata.org](http://pandas.pydata.org)), 223–224
  - Scikit-learn ([scikit-learn.org](http://scikit-learn.org)), 224
  - SciPy ([scipy.org](http://scipy.org)), 224
- Python enhancement proposals (PEP), 117
  - PEP 8, 127
  - PEP 20, 117
- Python internationalization, 140
- Python interpreter, 14, 92, 101–103, 105, 107, 198, 207
- Python modules, 199–200
  - import, 200–201, 203
  - prototype loop, 315–316
  - rolling, 203–205
  - use, 201–203
- Python Package Index (PyPI), 205
- Python sequence operators, 150
- Python Software Foundation (PSF), 199
- Python string methods, 154

## Q

- quotations archive, 211–215

## R

- random module, 199
- random-number generator, 250
- `range()` function, 173, 186

## React

- components
  - adding global styles, 689–690
  - Bing.com, 696, 697
  - built-in components, 682–683
  - class components, 704–706
  - composing components, 717–722
  - function components, 702–704
  - HTML components, 681
  - lifecycle, 706–717
  - props, 698–699
  - re-rendering, 702
  - returning valid values, 697–698
  - state, 699–701
  - styling strategies, 693–694
  - using local styles, 690
  - using `style` attribute, 691, 692
  - using style modules, 692–693
  - using style objects, 691–692
- description of, 666
- event handling
  - arrow functions, 728–729
  - binding class methods, 726–727
  - binding function in constructor, 727–728
  - dispatching synthetic events, 724
  - event attributes, 723–724
  - function component, 729–730
  - passing arguments, 725–726
  - passing functions as props, 726
  - specifying handler function, 725

- form input
  - controlled inputs, 731–733
  - one-way data binding, 730
  - two-way data binding, 730
  - uncontrolled inputs, 733–735
- hot reloading, 677
- JSX (*see* JSX)
- library, 672
- modifications, 678
- one-way data binding, 730
- React UI, 666–672
- two-way data binding, 730
- Vite, 673–677
- ReactDOM, 672
- `React.PureComponent` class, 711
- React UI, 666
  - building static version, 668–670
  - designing component hierarchy, 667–668
  - determining the state, 670, 671
  - identifying the state, 670
  - implementing inverse data flow, 671–672
- readability of code, 35, 46, 53, 118, 182, 355, 356, 378, 381
- `readlines()` method, 210
- README file, 483–485
- `read()` method, 210
- recurrent neural networks (RNNs), 296
- Reddit, 319
- `Reddit.com`, 357
- `reduce()` function, 551–552
- refactoring, 325–327
  - with AI, detecting smells, 326
  - with code formatters, 327
  - code smells, 325
  - safety nets, 326–327
- Refactoring* (Fowler), 325
- `removeEventListener()` method, 624–625
- `remove()` method, 175
- `render()` method, 708–709, 711
- requests, 206
- `requests.get`, 207
- reserved words, 10, 682
- Response object, 644, 645
  - `blob()` method, 646
  - `text()` method, 646
- responsible, artificial intelligence (AI)
  - fairness, 299–300
  - interpretability, 300
  - privacy, 300
  - safety and security, 300
- responsive design web pages
  - Cascading Style Sheets (CSS) grid, 439–442
  - Flexbox, 426–439
    - assumptions, 432–433
    - boxes creation, 427–428
    - containers, 431–432
    - cross-axis, 433–434
    - main axis, 434
    - modification, 434–437
    - in one dimension, 428–431
    - order property, 437–439
  - mobile compatibility, 422

- mobile-first design, 422
- mobile web design, 422
- overview, 421–426
- viewport meta tag, 422–426
- return value, 43, 193, 194, 509, 570, 573, 577, 578, 595, 660, 697–698, 708, 711, 712
- reusing data, 207–208
  - open file, 208–209
  - reading data from file, 210–211
  - writing data to file, 209
- reversed() function, 173
- right associativity, 516
- root.unmount() method, 712
- round() function, 132, 133
- rowspan attribute, 364, 365
- runtime errors, 45, 322
- runtime model, JavaScript
  - callback queue, 613
  - call stack, 612
  - memory heap, 612–613
- Rust, 65–66
  - cybersecurity and blockchain, 66
  - embedded systems, 66
  - game development, 66
  - networking and cloud services, 66
  - systems programming, 65
  - WebAssembly, 66

## S

- same-origin policy, 637–638, 650
- sandbox, 52, 454
- sanity-check question, 232
- Scalable Vector Graphics (SVG), 261
- scientific computing and engineering, 20
- Scikit-learn ([scikit-learn.org](https://scikit-learn.org)), 224, 230, 236, 256, 272–275
  - datasets, 244–246, 251
    - iris dataset, 244–245
    - other datasets, 245–246
  - feature matrix, 273–274
  - feature scaling, 274–275
  - fit, 273
  - instantiate, 272
  - predict or score, 273
  - target vector, 273–274
- SciPy ([scipy.org](https://scipy.org)), 224
- scope, 195–196
  - global scope, 196–197
  - local scope, 197–198
  - variable scope, 195–196
- scraping, 20, 371
- search engines, 80, 86, 344, 375, 410, 423, 670, 672, 696
- security bugs, 323
- selectors, 404–410
  - HyperText Markup Language (HTML)
    - element naming, 408–410
    - specific element styling, 404–408
- self-attention mechanism, 296–297
- sentiment analysis, 293
- separate style sheets, 393
- separation of concerns, 467, 615, 680
- server, 56, 366, 449, 454, 637, 648, 651
- server-side scripting language, 64
- server-side services, 454

- server-side web development, 64, 65
- `set(collection)`, 181
- `setState()` method, 705
- `setTimeout()` function, 639, 640
- shallow copy, 555, 559, 566
- `shift()` method, 547
- short-circuit operators, 520
- short-term memory, 24
- `shouldComponentUpdate()` method, 711
- `show_random_quotation()` function, 215
- side effect, 296, 306, 682, 714
- simulated neural networks (SNNs), 289
- single-line comments, 43
  - double-slash (`//`), 43
  - hash symbol (`#`), 43
- single-line syntax, 34
- single quotation marks, 123, 124
- `slice()` method, 542, 548
- slicing, 150, 237–239
- smart devices, 21
- Snyk, 324
- software bugs
  - functional bugs, 322
  - integration bugs, 322
  - logical bugs, 322
  - out-of-bound bugs, 323
  - runtime errors, 322
  - security bugs, 323
  - syntax errors, 322
  - unit-level bugs, 322
  - workflow bugs, 322
- software maintenance, 334–335
  - adaptive maintenance, 334
  - corrective maintenance, 334
  - and lifecycle, 335
  - perfective maintenance, 334
  - preventative maintenance, 334
  - technical debt, 334–335
- software requirements, 311, 314
- software requirements specification (SRS), 311, 313, 318
- source code, 3, 68, 254, 438, 447, 450–453, 455
- spanning, 364
- specific element styling, 404–408
  - child selector, 405–407
  - descendant selector, 407–408
- speech recognition, 293
- `splice()` method, 549
- `split()` function, 541–542
- spread syntax, 554–555
- `sqrt()` function, 134–135, 203
- square brackets notation, 563–564
- StackOverflow, 319
- standard deviation, 257, 274, 278, 279
- standardization, 274
- standard library, 199, 200, 203, 205, 211
- static code analysis, 84, 85
  - artificial intelligence (AI) integration, 85
  - detect bad code, 84–85
- static methods, 570, 571, 606, 608, 707
- statistical measures, 255–258
  - central tendency measures, 256–257

- variance and range, 257–258
- statistics module, 199, 200, 202
- stats, 200
- Straub, Ben, 493
- strftime, 159
- string data type, 501–505
  - creating strings with template literal notation, 502–503
  - escaping characters, 502
  - quotation marks, 502
  - string functions, 503–505
- string expressions, 32
- string functions, 503–505
- stringify() method, 655
- string literals, 28–29, 67
- string.methodname(params)*, 152
- string operators, 149–152
- strings manipulation, 147
  - common string operators, 149–152
  - concatenation, 147–149
  - length, 149
  - methods, 152–154
- string variables, 26
- strip() method, 210
- str methods, 152
- str object (string object), 152
- structure
  - of Cascading Style Sheets (CSS), 375–380
    - element selection, 375–377
    - inspection, 378–380
    - properties and values, 377–378
  - HyperText Markup Language (HTML), 340–346
    - attributes, 343–344
    - elements, 341–343
      - head, title, and body, 344–346
- style attribute, 392, 401, 690–692
- styling, HyperText Markup Language (HTML), 352–354
  - basic elements, 352–353
  - superscript and subscript, 353–354
- styling elements
  - family tree, 404, 406
  - layout and alignment, 410–419
    - box model, 414–415
    - box positioning, 415–419
    - data organization, 410–412
    - div elements, 412–414
  - with practice, 419
- selectors, 404–410
  - HTML element naming, 408–410
  - specific element styling, 404–408
- styling lists, 398–401
- table design, 401–404
- styling lists, 398–401
  - Cascading Style Sheets (CSS), properties and values, 398–399
  - modified lists and list items, 399–400
- styling properties, 391–393
  - embedded, 392
  - inline, 392
  - separate style sheets, 393
- sub-word tokenization, 298
- Survey Bot, 189–190
- Swift, 66–67
- switch statement, 525–529

- symbol data type, 509
- synchronous JavaScript
  - synchronous programming, 630–631
  - user interface, 631
- syntax, 10, 79–80
  - adaption, 82–84
  - block, 34
  - code mastery, 80–82
  - errors, 45, 322
  - functions, 82
  - single-line, 34
- syntax-heavy, 53, 62
- SyntheticBase Event object, 724

## T

- tab-delimited text files, 243, 246, 248–249, 251
- table design, 401–404
  - Cascading Style Sheets (CSS), properties and values, 401–402
- table format, 361–366
  - basic structure, 361–364
  - columns and rows, 364–366
- table element, 361–363
- td element, 363, 364
- th element, 363
- tr element, 362–364
- tasks and elements, HyperText Markup Language (HTML), 346–351
  - headings level, 348
  - hyperlinks, 349–350
  - images, 350–351

- paragraph text, 349
- tasks and selectors, 380–391
  - background-image property, 387–391
    - background-attachment property, 390–391
    - background-position property, 389
    - background-repeat property, 389–390
    - background-size property, 388–389
- font properties, 380–384
  - color property, 382–383
  - font-family property, 383–384
  - font-size property, 381–382
  - font-style property, 383
  - font-weight property, 383
  - text-decoration property, 384
- interactive link, 385–387
- pseudo-class selectors, 387
- template literal notation, 502–503
- ternary operator, 524–525
- test-driven development (TDD), 329
- text-decoration property, 384
- text() method, 646
- text summarization, 293
- then() function, 635, 639
- thin clients, 449, 450
- Thucydides, 220
- time(), 160, 167
- timedelta, 155, 162–165, 167
- times, Python handling, 160–161
- timespans calculation, 162–166

- time zones, Python handling, 166–169
  - aware datetime, 167
  - naïve datetime, 167
- TIOBE Index, 50
- `today=dt.date.today()`, 164
- `today()` method, 156
- `ToggleVisibility` class, 726–727
- tokens, 297–299
  - limits, 298
  - sub-word tokenization, 298
  - word tokenization, 298
- top-level functions, 569–570, 578–579
- tour guide, 87
- traditional code completion, 80
- training, 275–280
  - cross-validation, 278–279
  - model interpretation, 279–280
  - and test sets, 276–277
- transformer models, 296–299
  - attention, 296–297
  - self-attention mechanism, 296–297
- tokens, 297–299
  - limits, 298
  - sub-word tokenization, 298
  - word tokenization, 298
- transistors, 12
- translating code, 329–331
  - vs dialects, 329
  - preparation, 330
  - verification, 330–331
- Transmission Control Protocol/Internet Protocol (TCP/IP), 448

- triple quotation marks (single or double), 142, 143
- triple-quoted strings, 119
- troubleshooting problems, 16
- Tukey, John, 254
- tuples, 176–180
- two-way data binding, 730
- `type()`, 162, 163
- `typeof` operator, 500, 501
- TypeScript, 67

## U

- undefined data type, 509
- Unicode code point, 151, 152
- unit-level bugs, 322
- unmounting methods, React components, 712–713
- unordered lists, 358–359, 398, 400, 407, 408, 419
- unpacking, tuple, 178
- `unshift()` method, 548
- updating methods, React components
  - `componentDidUpdate()` method, 712
  - finishing the update, 712
  - `getSnapshotBeforeUpdate()` method, 712
  - getting a snapshot, 712
  - `render()` method, 711
  - `shouldComponentUpdate()` method, 711
- `useEffect()`, 713–715
- `useState()`, 702, 703, 713
- UTC. *See* Coordinated Universal Time (UTC)
- `utcnow()`, 168

## V

*variablename*, 128

variables, 24

- arrays and lists, 26–27

- vs constants, 498

- data type

  - BigInt data type, 506–507

  - Boolean data type, 507–508

  - NaN data type, 509

  - number data type, 505–506

  - object, 510–511

  - primitive data types, 501

  - string data type, 501–505

  - symbol data type, 509

  - typeof operator, 500, 501

  - undefined data type, 509

- declaring, 24–25, 496, 497

- initializing, 496, 497

- naming, 496–498

- in other statements, 25–26

- Python, 127

  - creation in code, 128

  - manipulation, 129

  - valid variable names, 127–128

- using, 496–498

- using `let`, 495–497

variable scope, 195–196

- block-scoped variables, 512

- function-scoped variables, 512

- global scope, 196–197, 512

- local scope, 197–198

`var` keyword, 25, 498, 512, 531

vectorization, 229

version control, 486. *See also* Git

viewport meta tag, 422–426

visualization, 221, 222, 253, 258, 266, 297

Visual Studio (VS) Code, 91–94, 104–108,  
119, 354, 371, 372, 394, 395, 455,  
457–459, 673–674

- colors and icons, 99

- command palette, 457

- configuring, 473–474

- creating a project, 455–456

- Extensions panel, 474

- Git, 488–491

- Python

  - extension, 93

  - open folder, 104–105

  - running, 106–108

Vite, 690

- VS Code terminal, 673–674

- `.gitignore` file, 676–677

- `node_modules` folder, 676

- `package.json` file, 677

- `package-lock.json` file, 677

- `public` folder, 676

- `src` folder, 676

- `vite.config.js` file, 677

Vivaldi, 341

void elements, 341, 349, 351

VS Code. *See* Visual Studio (VS) Code

## W

Wallace, Richard, 294

web applications, 63

- WebAssembly, 66
- web browsers, 448
  - built-in limitations, 454
  - C++, 59
  - firewalls, 454
  - sandbox, 454
  - security precautions, 454
  - server-side services, 454
- web development, 20
  - Go (Golang), 61
  - Kotlin, 63
- web page design, 354
- web page development, 21
- web server, 55, 64, 454, 648
  - development, 21
- while loop, 37, 38, 187–188, 195, 215
- while statement, 533–534
- while True, 188, 215
- white space, 461, 501
- width formatting, 143–146
- Wiegers, Karl, 314

- Wikipedia, 353, 385, 386
- with block, 209, 210
- word processor, 339, 340
- words (strings), 122–124
- word tokenization, 298
- workflow bugs, 322
- write() method, 209
- www.huffpost.com, 378

## X

- x86 (64-bit) assembly language, 52–53
- XMLHttpRequest() method, 635–637

## Y

- Yahoo.com, 358
- YouTube, 50, 318

## Z

- Zen of Python, 116–117
- zero-shot prompt, 305

