# Python Practice Workbook

Programming Techniques | Lectures 2–20

472 exercises

## 2.1 Variables and arithmetic expressions

### Exercise 2.1.01 — A grocery total

Store the prices 2.50, 3.20 and 1.80 in three variables. Calculate and print their total.

### Exercise 2.1.02 — Price after discount

A product costs 80 euros. Store its price and a discount of 15%. Compute and print the discounted price.

### Exercise 2.1.03 — Rectangle measurements

Store a rectangle width of 7 and height of 4. Print its area and perimeter.

### Exercise 2.1.04 — Converting minutes

A trip lasts 145 minutes. Use // and % to print the number of complete hours and the remaining minutes.

### Exercise 2.1.05 — Average score

Store three scores, 12, 15 and 17. Print their arithmetic average.

### Exercise 2.1.06 — Compound expression

Given a=5, b=3 and c=2, calculate (a+b)**2/(c+2) and print the result.

### Exercise 2.1.07 — Temperature conversion

Store a temperature of 68 degrees Fahrenheit. Convert it to Celsius using (F - 32) * 5 / 9.

### Exercise 2.1.08 — Sharing an amount

A group has 137 euros to distribute equally to 6 people in whole euros. Print each share and the undistributed remainder.

### Exercise 2.1.09 — Successive changes

Start with a balance of 250 euros. Add 75, subtract 48, and print the final amount using augmented assignment operators.

### Exercise 2.1.10 — Simple interest

Given principal 1200 euros, annual rate 0.04 and 3 years, compute simple interest and the final capital.

## 3.1 Strings and basic text operations

### Exercise 3.1.01 — Full name

Store first name "Alice" and last name "Green". Print them together with one space.

### Exercise 3.1.02 — Text length

Store the text "Mathematical Finance". Print the number of characters, including spaces.

### Exercise 3.1.03 — First and last characters

Given text "Python", print its first and last characters using indexing.

### Exercise 3.1.04 — Extract a prefix

Given "ISEG-2026-PT", print the first four characters using slicing.

### Exercise 3.1.05 — Reverse a short code

Given "AB123", print its characters in reverse order using slicing.

### Exercise 3.1.06 — Normalize input text

Store " LiSbOn ". Remove leading and trailing spaces and convert the result to lowercase.

### Exercise 3.1.07 — Count a letter

Store a string in a variable and print how many times the letter "a" appears. Test with strings such as "banana bread" and "Python".

### Exercise 3.1.08 — Replace a separator

Store a date-like string in a variable and replace every "/" with "-". Try values such as "2026/11/13" and "2026-11-13".

### Exercise 3.1.09 — Check a prefix and suffix

Given a string variable, print `True` only if it starts with "report" and ends with ".csv"; otherwise print `False`. Test "report_final.csv", "data.csv" and "report.txt".

### Exercise 3.1.10 — Search for a pattern

Given a string variable, print whether it contains the substring "gram" and how many times the letter "m" occurs. Test with "Programming Techniques" and "Python".

## 4.1 Interactive programs and conversions

### Exercise 4.1.01 — Greet a user

Read a first name using input() and print "Welcome, NAME" with the provided name.

### Exercise 4.1.02 — Age next year

Read an integer age. Print the age the person will have next year. Assume a valid integer is entered.

### Exercise 4.1.03 — Trip cost

Read distance in kilometers and cost per kilometer as numbers. Print the total.

### Exercise 4.1.04 — Minutes to hours

Read an integer number of minutes. Print complete hours and leftover minutes.

### Exercise 4.1.05 — Area from input

Read the width and height of a rectangle as floats. Print its area.

### Exercise 4.1.06 — Case-insensitive city

Read a city name, remove external spaces and lowercase it. Print the normalized result.

### Exercise 4.1.07 — Two-number calculator

Read two floats. Print their sum, difference and product.

### Exercise 4.1.08 — Gross price

Read a net price and a VAT percentage. Print the gross price.

### Exercise 4.1.09 — Three-part identifier

Read a first name, a last name and a two-digit year as text. Print a code made of their first letters (uppercase) and the year.

### Exercise 4.1.10 — Change from a purchase

Read the amount paid and the purchase price as floats. Print the difference. Assume the payment is sufficient.

## 5.1 Boolean expressions and membership

### Exercise 5.1.01 — Adult check

Read an integer age and print True if the age is at least 18, otherwise False.

### Exercise 5.1.02 — Within a range

Read an integer and print whether it lies between 10 and 20 inclusive.

### Exercise 5.1.03 — Even number

Read an integer and print whether it is even.

### Exercise 5.1.04 — Access condition

Read an age and a membership answer ("yes" or "no"). Print whether a person aged at least 18 AND holding membership is admitted.

### Exercise 5.1.05 — Weekend condition

Read a weekday name in lowercase. Print whether it is Saturday OR Sunday.

### Exercise 5.1.06 — Not empty

Read a text and print whether it contains at least one non-space character.

### Exercise 5.1.07 — Divisible by two numbers

Read an integer. Print whether it is divisible by both 3 and 5.

### Exercise 5.1.08 — Outside an interval

Read a float and print whether it is outside the closed interval [0, 100].

### Exercise 5.1.09 — Text membership

Read a sentence and a substring. Print whether the substring occurs in the sentence, ignoring letter case.

### Exercise 5.1.10 — Eligibility expression

Read age, score and a membership flag ("yes" or "no"). Print whether a person is at least 18 and has either score >= 70 or membership.

### Exercise 5.1.11 — Half-open interval

Read a number and print whether it belongs to the interval from 5 included to 12 excluded. Test 5, 12 and values on either side.

### Exercise 5.1.12 — Two separate intervals

Read a number and print True if it belongs to either of these intervals: from 0 included to 10 excluded, or from 20 excluded to 30 included. Print False otherwise.

### Exercise 5.1.13 — Complement of an interval

Read a number and print True if it is NOT in the interval from -5 excluded to 5 included. Check the boundary values -5 and 5.

### Exercise 5.1.14 — Range and divisibility

Read an integer and print True only when it belongs to the inclusive interval from 10 to 30 AND is divisible by 3. Otherwise print False.

### Exercise 5.1.15 — Intervals with one excluded value

Read a number and print True if it lies between 0 and 10 inclusive OR between 20 and 30 inclusive, but is NOT equal to 25. Otherwise print False.

### Exercise 5.1.16 — Combined eligibility rule

Read an integer age, a numerical score and a flag ("yes" or "no"). Print True only when age is at least 18 but below 65 AND either the score is at least 70 OR the flag is "yes".

### Exercise 5.1.17 — Rectangular region

Read two numerical coordinates x and y. Print True when x lies between -2 and 2 inclusive AND y is strictly greater than 0 but at most 5. Otherwise print False.

### Exercise 5.1.18 — Three numerical regions

Read a number and print True if it is at most -10, strictly between 3 and 7, OR at least 15. Print False for all other values.

### Exercise 5.1.19 — Time window with valid limits

Read integer hour and minute values. Print True only when they form a valid time (hour 0 to 23, minute 0 to 59) AND the hour lies in either 9 to 11 inclusive or 14 to 17 inclusive.

### Exercise 5.1.20 — Text conditions with exclusion

Read a filename and a status ("active" or "inactive"). Print True only when (the filename starts with "data" or "report") AND the filename ends with ".csv" AND the status is not "inactive". Otherwise print False.

## 5.2 Conditional programs

### Exercise 5.2.01 — Sign of a number

Read a float and print "positive", "negative" or "zero".

### Exercise 5.2.02 — Even or odd

Read an integer and print "even" or "odd".

### Exercise 5.2.03 — Pass or fail

Read a score out of 20. Print "pass" for scores at least 9.5, otherwise "fail".

### Exercise 5.2.04 — Larger number

Read two distinct integers and print the larger one.

### Exercise 5.2.05 — Temperature advice

Read a temperature. Print "cold" below 15, "mild" from 15 up to but not including 25, and "hot" otherwise.

### Exercise 5.2.06 — Ticket prices

Read an age. Set a ticket price of 5 for under-12s, 8 for ages 12–64 and 6 for ages 65 and above. Print the price.

### Exercise 5.2.07 — Safe division

Read two floats. If the second is zero, print "Cannot divide"; otherwise print their quotient.

### Exercise 5.2.08 — Shipping category

Read a parcel weight. Print "light" for at most 2 kg, "medium" for at most 10 kg, and "heavy" otherwise. Assume a positive weight.

### Exercise 5.2.09 — Password rule

Read a password. Print "accepted" only when its length is at least eight and it contains the character "!".

### Exercise 5.2.10 — Two conditions for a discount

Read a purchase amount and membership status ("yes"/"no"). Members get 10% off purchases >= 50; all other purchases retain their original price. Print the amount to pay.

### Exercise 5.2.11 — Five numerical categories

Read a number and print "very low" when it is below -10, "low" from -10 included to 0 excluded, "zero" at 0, "medium" above 0 through 10 included, and "high" above 10. Use if, elif and else.

### Exercise 5.2.12 — Check validity before classification

Read a numerical percentage. Print "invalid" outside the inclusive interval 0 to 100. For valid values, print "below" when less than 50, "equal" when exactly 50, and "above" otherwise. Use a nested decision.

### Exercise 5.2.13 — Independent adjustments

Read a nonnegative amount and two flags ("yes" or "no"): discounted and priority. First reduce the amount by 10 percent if discounted is "yes". Then add 5 if priority is "yes". Print the final amount; both adjustments may apply.

### Exercise 5.2.14 — Position on the coordinate axes

Read two numbers x and y. Print "origin" if both are zero; otherwise print "axis" if at least one is zero. For all other positions, print "same sign" if x and y have the same sign, or "opposite signs" otherwise. Use nested conditions.

### Exercise 5.2.15 — Validate then assign a grade

Read a numerical score. If outside 0 to 20 inclusive, print "invalid". Otherwise print "fail" below 9.5; for passing scores print "excellent" when at least 18, or "pass" otherwise. Use nested if statements.

### Exercise 5.2.16 — Priority among overlapping rules

Read an integer n. Print "multiple of 6" if divisible by 6; otherwise print "multiple of 2" if divisible by 2; otherwise "multiple of 3" if divisible by 3; otherwise "other". Ensure that exactly one label is printed.

### Exercise 5.2.17 — Check and classify three lengths

Read three positive lengths. First print "invalid" if they cannot form a triangle (the sum of any two lengths must exceed the third). Otherwise print "equilateral" if all equal, "isosceles" if exactly two equal, or "scalene" otherwise.

### Exercise 5.2.18 — Choose and check an operation

Read a nonnegative balance, an operation ("add" or "subtract") and a nonnegative amount. For "add", print the new balance. For "subtract", print "insufficient" if the amount exceeds the balance; otherwise print the new balance. Print "unknown operation" for other operation names.

### Exercise 5.2.19 — Two-stage time decision

Read integer hour and minute. Print "invalid" for a time outside 00:00 to 23:59. Otherwise print "morning" before 12:00, "afternoon" from 12:00 to 17:59, and "evening" from 18:00 onward. Use a validity decision followed by a classification.

### Exercise 5.2.20 — Largest value with ties

Read three numerical values a, b and c. Print "tie" if the largest value occurs more than once; otherwise print the name ("a", "b" or "c") of the unique largest value. Use nested decisions to distinguish ties and strict comparisons.

## 6.1 Basic for loops and range()

### Exercise 6.1.01 — Count upwards

Print the integers from 1 to 10 inclusive with a for loop.

### Exercise 6.1.02 — Count backwards

Print the integers from 10 down to 1 using range().

### Exercise 6.1.03 — Even numbers

Print all even integers from 2 to 20 using range() with a step.

### Exercise 6.1.04 — Odd numbers

Print all odd integers from 1 to 19 using range().

### Exercise 6.1.05 — Multiplication table

Read an integer n and print its products by 1 through 10, one per line.

### Exercise 6.1.06 — Square numbers

Print each number from 1 to 8 alongside its square.

### Exercise 6.1.07 — Characters in a word

Read a word and print one character per line using for.

### Exercise 6.1.08 — Fixed repetitions

Read a word and print it exactly five times, numbering the lines from 1 to 5.

### Exercise 6.1.09 — Multiples of a parameter

Read a positive integer k and print its first eight positive multiples.

### Exercise 6.1.10 — Fizz-style marks

Print numbers from 1 to 20, but print "multiple of 3" instead of numbers divisible by 3.

## 6.2 Accumulators and counters

### Exercise 6.2.01 — Sum 1 through n

Read a positive integer n. Use a for loop and an accumulator to sum 1 through n. Do not use sum().

### Exercise 6.2.02 — Sum of even numbers

Read n and sum all even positive integers up to n using a for loop. Do not use sum().

### Exercise 6.2.03 — Count multiples

Read n. Count integers in 1..n divisible by 4.

### Exercise 6.2.04 — Sum entered values

Read exactly five floats using a for loop. Print their total.

### Exercise 6.2.05 — Average of readings

Read exactly six temperature readings. Print their average using an accumulator.

### Exercise 6.2.06 — Count positive values

Read seven integers. Count and print how many are strictly positive.

### Exercise 6.2.07 — Product of integers

Read n. Compute the product of all integers 1..n using a for loop and an accumulator initialized to 1.

### Exercise 6.2.08 — Count vowels

Read a sentence and count how many characters are vowels (a, e, i, o, u), ignoring case.

### Exercise 6.2.09 — Maximum of five inputs

Read five floats. Use the first value as the initial maximum, then update it with a loop over the next four.

### Exercise 6.2.10 — Conditional average

Read eight integer measurements. Print the average of the strictly positive measurements; if none is positive, print "No positive values".

## 6.3 While loops and stopping conditions

### Exercise 6.3.01 — Countdown with while

Read a positive integer n and count down from n to 1 with a while loop.

### Exercise 6.3.02 — Count to a limit

Read positive integer n. Print all integers from 1 to n with while.

### Exercise 6.3.03 — Doubling

Start at 1. Print powers of two up to 128 inclusive, doubling a variable inside while.

### Exercise 6.3.04 — Threshold counter

Read a positive target. Starting from total 0, repeatedly add successive positive integers (1, 2, 3, ...). Stop as soon as the total is at least the target. Print the final total and the last integer added.

### Exercise 6.3.05 — Sentinel sum

Read integer values until the user enters 0. Print the sum of all values before the sentinel.

### Exercise 6.3.06 — Sentinel count

Read integer values until -1 is entered. Count the number of values read before -1.

### Exercise 6.3.07 — Repeated subtraction

Read a nonnegative integer n. Repeatedly subtract 3 while n is at least 3. Print the number of subtractions and the remainder.

### Exercise 6.3.08 — Password retry

Keep requesting a password until the user enters "python". Print "Access granted" once, after the loop.

### Exercise 6.3.09 — Savings target

Read a positive target balance. Begin with 100 euros and multiply by 1.1 each year. Print the number of complete years needed to reach or exceed the target.

### Exercise 6.3.10 — Digit count

Read a nonnegative integer. Count its decimal digits using integer division by 10. Treat 0 as a one-digit number.

## 6.4 Nested loops

### Exercise 6.4.01 — Fixed star grid

Print four rows of five asterisks using nested loops. Print one row per line.

### Exercise 6.4.02 — Input-sized rectangle

Read positive height and width and print a rectangle of # characters using nested loops.

### Exercise 6.4.03 — Increasing triangle

Read positive n. Print a left-aligned triangle with 1 star in the first row and n stars in the last.

### Exercise 6.4.04 — Decreasing triangle

Read positive n. Print a triangle with n stars in its first row and one in its last.

### Exercise 6.4.05 — Multiplication grid

Print a 5-by-5 multiplication grid for row and column factors from 1 to 5.

### Exercise 6.4.06 — Coordinate pairs

Print all pairs (i, j) with i in 1..3 and j in 1..4.

### Exercise 6.4.07 — Pairs meeting a condition

Count pairs (i, j), with i and j in 1..5, whose sum exceeds 7. Print the count.

### Exercise 6.4.08 — Sum of pairwise products

Compute the sum of i*j over i in 1..3 and j in 1..4 using nested loops.

### Exercise 6.4.09 — Upper triangular numbers

Read a positive n. Print an n-by-n grid containing 1 when the column index is at least the row index and 0 otherwise. Use row and column indices starting at 0.

### Exercise 6.4.10 — Matching parity pairs

For i and j in 1..6, count pairs where both numbers have the same parity. Print the count and the sum of i*j over those pairs.

## 7.1 Lists: creation, indexing and slicing

### Exercise 7.1.01 — First and last

Given scores = [12, 15, 9, 18, 14], print the first and last values using indexes.

### Exercise 7.1.02 — Third element

Create a list containing five city names and print the third city.

### Exercise 7.1.03 — A middle slice

Given values = [4, 8, 12, 16, 20, 24], extract and print the three elements at indexes 1, 2 and 3.

### Exercise 7.1.04 — Every second observation

Given readings = [2, 5, 8, 11, 14, 17, 20], print the values at even indexes using slicing.

### Exercise 7.1.05 — Last four values

Given a list of ten integers from 1 to 10, print its last four elements with negative-index slicing.

### Exercise 7.1.06 — Reverse a list

Given letters = ["a", "b", "c", "d"], print a reversed copy using slicing without changing the original list.

### Exercise 7.1.07 — Two halves

Given [3, 6, 9, 12, 15, 18], print the first three and last three elements as separate slices.

### Exercise 7.1.08 — Check membership

Given a list of strings and a value to search for, print whether the value belongs to the list. Test both a present and an absent value, using for example ["red", "green", "blue"].

### Exercise 7.1.09 — Replace one element

Given prices = [10, 15, 20], replace the second price with 12 and print the updated list.

### Exercise 7.1.10 — Swap two positions

Given values = [7, 2, 9, 4], exchange the first and last elements using a temporary variable, then print the list.

## 7.2 Lists: loops, updates and simple computations

### Exercise 7.2.01 — Append a result

Start with [3, 6, 9]. Append 12 and print the updated list.

### Exercise 7.2.02 — Remove an item

Start with ["pen", "book", "ruler"]. Remove "book" and print the list.

### Exercise 7.2.03 — Print one value per line

Given temperatures = [18, 21, 19, 23], use a for loop to print each temperature separately.

### Exercise 7.2.04 — Sum without sum

Given costs = [5, 12, 8, 15], compute the total using a loop and an accumulator, without sum().

### Exercise 7.2.05 — Count high scores

Given scores = [7, 15, 12, 19, 8, 16], count the entries greater than or equal to 14.

### Exercise 7.2.06 — Build squared numbers

Given numbers = [1, 2, 3, 4, 5], build a new list of their squares using a loop and append().

### Exercise 7.2.07 — Filter even values

Given [3, 8, 11, 14, 17, 20], build and print a new list containing only even numbers.

### Exercise 7.2.08 — Update all elements

Given quantities = [2, 5, 3, 8], increase every value by 1 directly in the original list using indexes.

### Exercise 7.2.09 — Find the largest

Given [13, 9, 22, 17, 5], find the maximum with a loop and comparisons, without max().

### Exercise 7.2.10 — Parallel lists

Given names = ["Alice", "Bob", "Charlie"] and scores = [14, 11, 17], print each name with its corresponding score using indexes.

## 8.1 Functions: parameters, arguments and return values

### Exercise 8.1.01 — A greeting function

Write greet(name) that returns "Hello, " followed by the name. Call it twice using different names.

### Exercise 8.1.02 — Double a value

Write double(number) that returns twice its argument, and test it with 7 and -3.

### Exercise 8.1.03 — Rectangle area

Write rectangle_area(width, height) that returns the area and test it with width 6 and height 4.

### Exercise 8.1.04 — Positive or not

Write is_positive(value) that returns True for strictly positive numbers and False otherwise. Test three values.

### Exercise 8.1.05 — Discounted price

Write final_price(price, discount_percent) that returns a price after the given percentage discount. Test with 80 and 25.

### Exercise 8.1.06 — Larger of two values

Write larger(a, b) that returns the greater argument using if-else (do not use max()). Test two cases.

### Exercise 8.1.07 — Seconds converter

Write to_seconds(hours, minutes, seconds) to convert a duration to seconds. Test with 1 hour, 2 minutes and 3 seconds.

### Exercise 8.1.08 — Two return values

Write quotient_remainder(a, b) that returns the integer quotient and remainder of a divided by b. Assume b is not zero.

### Exercise 8.1.09 — Grade label

Write grade_label(score) that returns "Pass" for score at least 10 and "Fail" otherwise. Test at 9, 10 and 15.

### Exercise 8.1.10 — Compose functions

Write tax(amount, rate) returning the tax and total_with_tax(amount, rate) that calls tax() and returns the final total. Test with 100 and 0.20.

## 8.2 Functions with loops and lists

### Exercise 8.2.01 — Count entries

Write count_items(values) returning the number of elements using a loop, without len(). Test on two lists.

### Exercise 8.2.02 — Sum elements

Write list_total(values) that returns the sum of a list of numbers using a for loop, without sum().

### Exercise 8.2.03 — Count positives

Write count_positives(values) returning how many values are strictly greater than zero.

### Exercise 8.2.04 — Apply a threshold

Write above_threshold(values, threshold) returning a new list containing values strictly above threshold.

### Exercise 8.2.05 — Reverse without slicing

Write reverse_list(values) returning a reversed copy using a loop and append(), without slicing or reverse().

### Exercise 8.2.06 — Count a target

Write occurrences(values, target) returning the number of times target appears, without count().

### Exercise 8.2.07 — Arithmetic mean

Write mean(values) using a loop. Assume the list is not empty. Do not use sum().

### Exercise 8.2.08 — Find a matching name

Write contains_name(names, target) that returns True if a matching string is present and False otherwise, using a loop.

### Exercise 8.2.09 — Replace negatives

Write replace_negatives(values) returning a new list where negative entries are replaced by zero. Do not modify the input list.

### Exercise 8.2.10 — Index of first maximum

Write max_index(values) returning the index of the first largest number in a nonempty list without max().

## 8.3 Integrated functions, strings and iterative selection

### Exercise 8.3.01 — Filter and summarize numbers

Write `select_values(values, minimum)` to return numbers at least `minimum`. Write `summarize(values)` to return their sum and count as a tuple; return `(0, 0)` for an empty list. Apply the second function to the selected values. Example: `[4, 11, 7, 15, 3]`, minimum 8.

### Exercise 8.3.02 — Find and modify a matching code

Write `first_matching_code(codes)` using `while` to return the first string starting with `"ID-"` and ending with `"-OK"`, or `None` if absent. When a match exists, print a copy with hyphens replaced by underscores. Test `["ID-1-NO", "ID-3-OK"]`.

### Exercise 8.3.03 — Select strings by two conditions

Write `select_names(names)` that removes outer spaces from each string and returns only those containing at least five characters and starting with `"A"` or `"B"`. Preserve order and duplicates. Test `[" Alice ", "Ben", "Brenda", "Aaron"]`.

### Exercise 8.3.04 — Classify signs and count results

Write `classify_sign(value)` returning `"positive"`, `"negative"` or `"zero"`. Use it in `sign_report(values)` to return a dictionary with counts under `"positive"` and `"negative"`, and the sum under `"total"`. Test `[30, -10, 0, 20, -5]`.

### Exercise 8.3.05 — Stop at the first out-of-range number

Write `values_before_invalid(values)` returning all numbers before the first entry outside 0–100 (inclusive), using a `while` loop. Do not include the invalid entry. For `[10, 15, 101, 20]`, return `[10, 15]`. Empty input gives an empty list.

## 9.1 Tuples and sets

### Exercise 9.1.01 — Unpack a coordinate

Store the point (4, 9) as a tuple, unpack it into x and y, and print both values.

### Exercise 9.1.02 — Tuple indexing

Given temperatures = (18, 21, 20, 19), print its first, last and second-to-last elements.

### Exercise 9.1.03 — Return a tuple

Write min_max_pair(a, b) returning a tuple (smaller, larger) without min() or max().

### Exercise 9.1.04 — Unique values and membership

Given a list of strings, create a set of its distinct values and print the set size. Then check whether a target string belongs to that set. Test with both a present and an absent target.

### Exercise 9.1.05 — Insert into a set

Begin with {2, 4, 6}. Add 8 and then add 4 again. Print the number of elements.

### Exercise 9.1.06 — Common participants

Given group_a = {"Alice", "Bob", "Dan"} and group_b = {"Bob", "Cara", "Dan"}, print the intersection.

### Exercise 9.1.07 — All participants

For the same two groups, print their union in alphabetical order.

### Exercise 9.1.08 — Only in the first group

Using group_a = {1, 2, 3, 4} and group_b = {3, 4, 5}, print items present only in group_a.

### Exercise 9.1.09 — Remove duplicates

Write unique_count(values) returning the number of distinct integers in a list using a set.

### Exercise 9.1.10 — Shared keywords

Write shared_keywords(first, second) taking two lists of words and returning a set of common words.

## 9.2 Dictionaries: creation, lookup and updates

### Exercise 9.2.01 — A price dictionary

Create a dictionary mapping "apple" to 2, "bread" to 3 and "milk" to 4, then print the price of milk.

### Exercise 9.2.02 — Change a value

Given stock = {"pens": 10, "books": 5}, update pens to 8 and print the dictionary.

### Exercise 9.2.03 — Insert a new key

Start with {"Alice": 14}. Add Bob with score 16 and print the dictionary.

### Exercise 9.2.04 — Safe lookup

Given prices = {"tea": 2, "coffee": 3}, use .get() to read "juice" with a default price of 0.

### Exercise 9.2.05 — Print all names

Given scores = {"Alice": 14, "Bob": 11, "Cara": 17}, use a loop to print each key.

### Exercise 9.2.06 — Print keys and values

For the same scores dictionary, print each name and its score using .items().

### Exercise 9.2.07 — Total stock

Given stock = {"pens": 6, "books": 4, "folders": 3}, compute the sum of quantities with a loop.

### Exercise 9.2.08 — Count above threshold

Given scores = {"Alice": 14, "Bob": 9, "Cara": 17}, count the scores at least 10.

### Exercise 9.2.09 — Word frequencies

Given words = ["red", "blue", "red", "green", "blue", "red"], construct a dictionary of occurrence counts.

### Exercise 9.2.10 — Invert a unique mapping

Given id_to_name = {1: "Alice", 2: "Bob", 3: "Cara"}, build name_to_id. Assume all names are distinct.

## 9.3 Nested lists as tables

### Exercise 9.3.01 — Read one cell

Given table = [[2, 4, 6], [8, 10, 12]], print the value in row 2, column 3 (counting from 1).

### Exercise 9.3.02 — Count rows

Given table = [[1, 2], [3, 4], [5, 6]], print the number of rows and number of columns.

### Exercise 9.3.03 — Print each row

Given [[1, 3], [5, 7], [9, 11]], use a for loop to print each inner list.

### Exercise 9.3.04 — Print every cell

Use nested loops to print each number in [[1, 2], [3, 4]] on its own line.

### Exercise 9.3.05 — Sum one row

For [[3, 6, 9], [2, 4, 8]], compute the total of the second row using a loop.

### Exercise 9.3.06 — Sum a column

For [[1, 5], [2, 6], [3, 7]], compute the sum of the second column.

### Exercise 9.3.07 — Total of a table

Compute the sum of all entries of [[1, 2, 3], [4, 5, 6]] with nested loops.

### Exercise 9.3.08 — Build a multiplication grid

Using nested loops and append(), build a 3-by-4 table where the entry in row i and column j equals i*j for i = 1..3 and j = 1..4.

### Exercise 9.3.09 — Count positive cells

Given [[-1, 2, 3], [0, -4, 5]], count strictly positive elements with nested loops.

### Exercise 9.3.10 — Row totals

Write row_totals(table) that returns a list of the sums of its rows. Test with [[1, 2], [3, 4], [5, 6]].

## 9.4 Combining dictionaries and nested data

### Exercise 9.4.01 — Dictionary of lists

Given grades = {"Alice": [12, 14], "Bob": [10, 16]}, print Alice's second grade.

### Exercise 9.4.02 — Append a new grade

Given grades = {"Alice": [12, 14], "Bob": [10, 16]}, append 18 to Bob's list.

### Exercise 9.4.03 — Dictionary of dictionaries

Given people = {"Alice": {"age": 20, "city": "Lisbon"}, "Bob": {"age": 22, "city": "Porto"}}, print Bob's city.

### Exercise 9.4.04 — Update a nested field

Using a dictionary of people with nested age fields, increase Alice's age from 20 to 21 and print the updated value.

### Exercise 9.4.05 — Average per student

Given grades = {"Alice": [12, 14, 16], "Bob": [10, 12, 14]}, print each student's average using loops.

### Exercise 9.4.06 — List of dictionaries

Given records = [{"name": "Alice", "score": 14}, {"name": "Bob", "score": 9}], print the names of records whose score is at least 10.

### Exercise 9.4.07 — Count records by category

Given records = [{"type": "A"}, {"type": "B"}, {"type": "A"}], build a dictionary with the count for each type.

### Exercise 9.4.08 — Stock in two shops

Given shops = {"North": {"pens": 3, "books": 2}, "South": {"pens": 4, "books": 5}}, compute the total number of pens.

### Exercise 9.4.09 — Find the highest score

Given results = [{"name": "Alice", "score": 14}, {"name": "Bob", "score": 18}, {"name": "Cara", "score": 16}], find and print the name with the highest score.

### Exercise 9.4.10 — Group names by department

Given records = [{"name": "Alice", "dept": "A"}, {"name": "Bob", "dept": "B"}, {"name": "Cara", "dept": "A"}], build a dictionary mapping each department to its list of names.

## 9.5 Nested collections and loop combinations

### Exercise 9.5.01 — Totals in a dictionary of lists

Given a dictionary mapping keys to lists of numbers, create a dictionary containing `(total, qualifying_count)` for each key. The total includes all values; the count includes only values at least `minimum`. Use nested `for` loops. For `{"A": [3, 5, 2], "B": [4, 8]}` and minimum 4, obtain `{"A": (10, 1), "B": (12, 2)}`.

### Exercise 9.5.02 — Count matching labels by key

Given a list of dictionaries with `"group"` and a `"labels"` list, count labels starting with `"pre-"` for each group. Use nested loops and combine records sharing a group. If group A has `["pre-1", "other"]` and later `["pre-2"]`, its count is 2.

### Exercise 9.5.03 — Find matching column indexes

Write `qualifying_cells(table, minimum)` returning, for each row index, the column indexes whose values are at least `minimum`. Use nested loops, zero-based indexes and include rows with no matches. Example: `[[2, 6, 1], [8, 3, 7]]`, minimum 6 gives `{0: [1], 1: [0, 2]}`.

### Exercise 9.5.04 — Count values in nested dictionaries

Given a dictionary of dictionaries containing numbers, use nested loops to count values at least 10 in each inner dictionary. Return one count per outer key, including zero for empty dictionaries. Example: `{"A": {"x": 12, "y": 8}, "B": {"z": 15}}` gives `{"A": 1, "B": 1}`.

### Exercise 9.5.05 — Find an index in each list

Given a dictionary of numeric lists, return the zero-based index of the first value at least `minimum` for every key, or `None` if absent. Use an outer `for` and an inner `while`. Example: `{"A": [2, 9], "B": [3], "C": []}`, minimum 8 gives `{"A": 1, "B": None, "C": None}`.

### Exercise 9.5.06 — Sum qualifying numbers in rows

Write `row_totals(table, minimum)` to return, for each row, the sum of numbers at least `minimum`. Use a `while` for the rows and a `for` for their elements. Example: `[[2, 7, 9], [3, 8], []]`, minimum 7 gives `[16, 8, 0]`.

### Exercise 9.5.07 — Count table values with two while loops

Write `count_large_cells(table, minimum)` to count entries at least `minimum` in a rectangular list of lists. Use a `while` over rows and a nested `while` over columns. Example: `[[1, 5, 7], [8, 3, 6]]`, minimum 6 gives 3; empty input gives zero.

### Exercise 9.5.08 — Count substring matches in texts

Given a list of strings `texts` and a list of substrings `patterns`, use nested loops to count how many strings contain each pattern. Return a dictionary including zero counts. Example: `["abc", "abd"]` and `["ab", "xy"]` gives `{"ab": 2, "xy": 0}`. Match case-sensitively.

### Exercise 9.5.09 — Combine counts for repeated groups

Given a list of dictionaries with keys `"group"` and `"values"`, count numbers at least 10 for each group. Use nested loops and combine repeated keys. Example: `[{"group":"A","values":[12,7]}, {"group":"A","values":[14]}]` gives `{"A": 2}`.

## 10.1 Recursion: base cases and simple reductions

### Exercise 10.1.01 — Countdown

Write countdown(n) that prints n, n-1, ..., 1 using recursion, and stops when n reaches 0. Test with 4.

### Exercise 10.1.02 — Count up

Write count_up(n) that recursively prints integers 1 through n in ascending order. Test with 5.

### Exercise 10.1.03 — Sum from 1 to n

Write recursive_sum(n) returning 1 + 2 + ... + n, with recursive_sum(0) = 0. Test with 5.

### Exercise 10.1.04 — Factorial

Write factorial(n) recursively with factorial(0) = 1. Assume n is nonnegative and test with 5.

### Exercise 10.1.05 — Power of two

Write power_of_two(n) recursively, using power_of_two(0) = 1 and doubling the result at each step. Test n=6.

### Exercise 10.1.06 — Product of the first n numbers

Write product_to_n(n) recursively, with product_to_n(1) = 1. Assume n is at least 1.

### Exercise 10.1.07 — Count digits

Write digit_count(n) recursively for a nonnegative integer. For n below 10, return 1; otherwise remove its final digit using // 10.

### Exercise 10.1.08 — Digit sum

Write digit_sum(n) recursively for a nonnegative integer, using % 10 and // 10. Test with 572.

### Exercise 10.1.09 — Greatest common divisor

Write gcd(a, b) recursively using the Euclidean rule gcd(a, b) = gcd(b, a % b) and base case gcd(a, 0) = a. Test with 48 and 18.

### Exercise 10.1.10 — Trace a recursion

Write trace(n) that prints "Enter n" before its recursive call and "Leave n" after it, stopping at n=0. Test with 3 and observe the order.

## 10.2 Recursion with sequences and strings

### Exercise 10.2.01 — Sum a list recursively

Write recursive_list_sum(values) using the first element and the remaining slice; return 0 for an empty list. Test with [2, 4, 6].

### Exercise 10.2.02 — Count list entries recursively

Write recursive_length(values) that counts list elements without len() for the final count. Use len() only to test the base case.

### Exercise 10.2.03 — Count even numbers recursively

Write recursive_even_count(values) that returns the number of even integers in a list.

### Exercise 10.2.04 — Find a target recursively

Write recursive_contains(values, target) that returns True if target appears in a list and False otherwise.

### Exercise 10.2.05 — Reverse a word recursively

Write reverse_text(text) that recursively returns a reversed string using slicing. Test with "python".

### Exercise 10.2.06 — Count a character recursively

Write recursive_char_count(text, target) counting occurrences of a one-character target in a string.

### Exercise 10.2.07 — Recursive maximum

Write recursive_max(values) returning the maximum of a nonempty list without max(). Compare the first element with the result on the rest.

### Exercise 10.2.08 — All values positive

Write all_positive(values) recursively, returning True for the empty list and False if any item is not strictly positive.

### Exercise 10.2.09 — Flatten a two-level list

Write flatten_rows(rows) recursively for a list of lists of integers. Return one flat list without nested elements. Assume there are only two levels.

### Exercise 10.2.10 — Recursive palindrome test

Write is_palindrome(text) recursively, checking the first and last character and then the middle substring. Assume lowercase text without spaces.

## 10.3 Recursion in nested and hierarchical data

### Exercise 10.3.01 — Count strings in nested lists

Write recursive function `count_strings(data)` to count all strings in a list containing strings or other lists, with arbitrary nesting depth. Count repeated strings separately; empty lists contribute zero. Example: `["a", ["b", ["a"]], []]` gives 3.

### Exercise 10.3.02 — Test membership in nested lists

Write recursive function `contains_string(data, target)` returning `True` if a nested list contains the complete string `target` at any depth, otherwise `False`. Example: `["A1", ["B2", ["C3"]]]` contains `"C3"`, but not `"C"`.

### Exercise 10.3.03 — Find the first string with a suffix

Write recursive function `find_first_csv(data)` to return the first string ending with `".csv"` in nested lists, or `None`. Visit elements from left to right and match case-sensitively. Example: `[["a.txt", "b.csv"], "c.csv"]` gives `"b.csv"`.

### Exercise 10.3.04 — Collect all strings with a suffix

Write recursive function `collect_files(data, extension)` to return a flat list of strings ending with `extension` in nested lists. Preserve order and duplicates; return `[]` when none match. Example: `["a.csv", ["b.txt", "c.csv"], "a.csv"]` gives `["a.csv", "c.csv", "a.csv"]` for `".csv"`.

### Exercise 10.3.05 — Count strings with a prefix

Write recursive function `count_prefix(data, prefix)` for nested dictionaries whose values are strings or dictionaries. Count string values starting with `prefix` at any depth, ignoring keys. Example: `{"one":{"a":"X-1", "b":"Y"}, "two":"X-2"}` with `"X-"` gives 2.

### Exercise 10.3.06 — Find the first dictionary value with a prefix

Write recursive function `find_first_value(data, prefix)` returning the first string value starting with `prefix` in nested dictionaries, or `None`. Follow dictionary insertion order and ignore keys. Example: `{"a":{"x":"B1", "y":"A1"}, "b":"A2"}` with `"A"` gives `"A1"`.

### Exercise 10.3.07 — Sum numbers in nested lists

Write recursive function `sum_nested(data)` returning the sum of all numbers in a nested list, regardless of depth. Negative and repeated values are included; empty lists contribute zero. Example: `[3, [2, [-1, 4]], []]` gives 8. Assume every non-list value is numeric.

### Exercise 10.3.08 — Find the maximum in nested dictionaries

Write recursive function `max_nested(data)` returning the maximum numeric value in nested dictionaries, ignoring keys. Return `None` if there are no numbers; negative values are allowed. Example: `{"one":{"a":-8,"b":4},"two":{"c":2}}` gives 4.

### Exercise 10.3.09 — Collect matching strings from mixed structures

Write recursive function `collect_matching(data, pattern)` for values that may be strings, lists or dictionaries, nested arbitrarily. Return string values containing `pattern` in traversal order (list order; dictionary value insertion order), ignoring keys. Example: `{"A":["rain",{"x":"train"}], "B":"sun"}` with `"ain"` gives `["rain", "train"]`.

### Exercise 10.3.10 — Find and transform a string recursively

Write two recursive functions. `find_first_matching(data)` returns the first string starting with `"data-"` and ending with `".csv"` in nested lists, or `None`. `replace_hyphens(text)` replaces every hyphen with an underscore recursively, one character at a time. Example: `[["a.txt", "data-q1.csv"], "data-q2.csv"]` gives `"data-q1.csv"`, then `"data_q1.csv"`.

### Exercise 10.3.11 — Measure the depth of nested lists

Write a recursive function `max_depth(data)` that returns the maximum nesting depth of a list containing values or other lists. The outer list is level 1; every nested list adds one level, even when empty. Thus `[]` and `["a", "b"]` both have depth 1, while `["a", ["b", []]]` has depth 3. The function must work with arbitrary nesting depth and must not modify the input.

### Exercise 10.3.12 — Find the widest list in a hierarchy

Write a recursive function `widest_sublist(data)` that returns the list containing the greatest number of *direct non-list elements* anywhere in a nested-list structure. Nested lists themselves do not count toward their parent's width, but each nested list must also be considered as a candidate. For example, `["a", ["x", "y", ["p"], "z"], "b", ["q", ["r"]]]` has width 2 at the outer level. The list `["x", "y", ["p"], "z"]` has width 3 because its nested list `["p"]` is not counted; it must be returned unchanged. Count repeated values separately. Include the outer list among the candidates, even if it is empty. In a tie, return the first candidate encountered when visiting each list before its nested lists, from left to right. Return the actual list, not only its width, and do not modify the input.

## 11.1 Creating and importing Python modules

### Exercise 11.1.01 — A reusable greeting

Create greetings.py with a function greet(name) returning "Hello, " followed by the name. In main.py import the function and print a greeting for Alice.

### Exercise 11.1.02 — Circle calculation

Create geometry.py containing circle_area(radius), using 3.14 as pi. Import the module using import geometry and print the area for radius 3.

### Exercise 11.1.03 — A temperature converter

Define to_celsius(fahrenheit) in temperatures.py. Import only that function and print the Celsius equivalent of 68 °F.

### Exercise 11.1.04 — Two helper functions

In arithmetic.py define double(value) and triple(value). In main.py import both and print the results for 7.

### Exercise 11.1.05 — A module constant

In rates.py define TAX_RATE = 0.20 and a function total_with_tax(price) that uses it. In main.py import the module and calculate the total for 50.

### Exercise 11.1.06 — Average from a module

In statistics_tools.py define mean(values) using a loop and len(). Import it into main.py to calculate the mean of [5, 7, 9, 11].

### Exercise 11.1.07 — Main guard

Write a utilities.py file containing square(n). Add a main guard that prints square(4) when utilities.py is run directly. Import square in main.py, and check that importing does not trigger that print.

### Exercise 11.1.08 — Separate calculation and display

In discounts.py define final_price(price, discount_rate). Use main.py to calculate and display prices after a 10% discount for [20, 40, 60].

### Exercise 11.1.09 — A small modular report

Define count_positive(values) in counters.py. Import it in main.py, count positive numbers in [-3, 4, 0, 7, -1] and display the result.

### Exercise 11.1.10 — Two cooperating modules

Create math_tools.py with percentage(value, rate), report_tools.py with format_result(value) returning a string, and main.py importing both to report 15% of 200.

## 11.2 Command-line arguments and entry points

### Exercise 11.2.01 — Print one argument

Write main.py that prints the first command-line argument. Run it as python main.py Alice. Assume the argument is provided.

### Exercise 11.2.02 — Convert a numeric argument

Read one integer from sys.argv and print its square. Try python main.py 8.

### Exercise 11.2.03 — Two arguments

Read two integers from sys.argv and print their sum. Try python main.py 12 5.

### Exercise 11.2.04 — Check argument count

Accept exactly one user-supplied argument. If the number of arguments is wrong, print Usage: python main.py NAME; otherwise print the supplied name.

### Exercise 11.2.05 — Command-line rectangle

Accept width and height as two numeric command-line arguments. If exactly two are given, print their product; otherwise print a usage message.

### Exercise 11.2.06 — A reusable CLI calculation

Place cube(value) in calculations.py and read a number from sys.argv in main.py. Import cube and print the result.

### Exercise 11.2.07 — argparse: one integer

Use argparse to declare a required integer positional argument n. Print the numbers from 1 through n. Try python main.py 4.

### Exercise 11.2.08 — argparse: two values

Use argparse to declare two float positional arguments price and quantity. Print their product.

### Exercise 11.2.09 — argparse: an optional rate

Accept a required float price and an optional float `--rate` with default 0.10. Print price after adding the rate.

### Exercise 11.2.10 — Command-line report with a module

Define total_cost(units, unit_price) in prices.py. Use argparse in main.py to read both values, call the imported function and print the total.

## 12.1 Reading and writing text files

### Exercise 12.1.01 — Read a text file

Create message.txt containing Hello Python. Write a program that reads its full content using with open(...) and prints it without adding a second newline.

### Exercise 12.1.02 — Count lines

Given notes.txt with four lines, count its lines using a for loop over the file and print the count.

### Exercise 12.1.03 — Strip line endings

Read names.txt and print each name without its trailing newline. Use .strip().

### Exercise 12.1.04 — Write three lines

Create results.txt containing exactly the lines Pass, Review and Pass, using mode w.

### Exercise 12.1.05 — Append a record

Given log.txt containing Start, append the line Completed without overwriting the previous line.

### Exercise 12.1.06 — Sum numbers from a file

Read values.txt, where each line contains one integer, and print their sum.

### Exercise 12.1.07 — Count matching words

Read words.txt and count the lines that contain exactly the word apple after stripping whitespace.

### Exercise 12.1.08 — Copy a file

Copy the complete contents of source.txt into copy.txt using with for both files.

### Exercise 12.1.09 — Keep positive values

Read integers from numbers.txt and write only positive numbers to positive.txt, one per line.

### Exercise 12.1.10 — Produce a summary file

Read measurements.txt, where every line contains one valid number. Calculate the count and average of all readings, then write two lines to summary.txt: Count: ... and Average: .... Assume the file contains at least one reading.

## 12.2 CSV tables with the csv module

### Exercise 12.2.01 — Read rows with DictReader

Read products.csv and print each product name using csv.DictReader.

### Exercise 12.2.02 — Sum one CSV column

Read sales.csv with columns item and units. Print the sum of units as an integer.

### Exercise 12.2.03 — Count rows by category

Read entries.csv and count records whose category is Food.

### Exercise 12.2.04 — Compute CSV revenue

Given orders.csv with quantity and price, calculate and print the sum of quantity $\times$ price.

### Exercise 12.2.05 — Write a CSV table

Write scores.csv with header name,score and two rows: Alice,15 and Bob,18. Use csv.DictWriter.

### Exercise 12.2.06 — Filter CSV records

Read students.csv and write passed.csv containing only records with score >= 10, retaining the same header.

### Exercise 12.2.07 — Add a calculated column

Read prices.csv with item and price, and write taxed.csv with item,price,total where total = price * 1.2.

### Exercise 12.2.08 — Grouped totals in a dictionary

Read transactions.csv with category and amount. Use a dictionary to calculate the total amount for each category and print it.

### Exercise 12.2.09 — CSV with a quoted comma

Read contacts.csv, where a quoted company name contains a comma, and print each company. Use csv.DictReader rather than manually splitting lines.

### Exercise 12.2.10 — CSV summary report

Read shipments.csv, calculate the number of shipments and total weight per destination, and write summary.csv with columns destination,count,total_weight.

## 12.3 JSON and nested data

### Exercise 12.3.01 — Load a JSON dictionary

Read profile.json containing name and age, and print the name.

### Exercise 12.3.02 — Write a JSON dictionary

Create settings.json with keys theme set to dark and notifications set to true. Use json.dump with indent=4.

### Exercise 12.3.03 — Read a JSON list

Read scores.json containing a list of numbers and print the highest value using max().

### Exercise 12.3.04 — Update and save a field

Read account.json, add 50 to its balance and save the updated dictionary to account_updated.json.

### Exercise 12.3.05 — Count nested records

Read library.json with a books list. Print how many books it contains.

### Exercise 12.3.06 — Average nested values

Read students.json containing a list of dictionaries with a name and grade. Calculate and print the average grade.

### Exercise 12.3.07 — Filter a JSON collection

Read items.json, keep only dictionaries with available true, and write them as a JSON list to available.json.

### Exercise 12.3.08 — A nested settings update

Read config.json with a database dictionary containing host and port. Change the port to 5433 and write config_updated.json.

### Exercise 12.3.09 — CSV to JSON

Read products.csv with name and price, convert price to float, and write a JSON list of product dictionaries to products.json.

### Exercise 12.3.10 — JSON to CSV

Read employees.json (a list of name and hours dictionaries), calculate the weekly pay at 12 per hour, and write payroll.csv with name and pay.

## 13.1 Exceptions, validation and debugging

### Exercise 13.1.01 — Handle a failed conversion

Given a string variable, attempt to convert it to `float`. Print the converted number if valid or "Invalid number" if conversion fails. Test "12.5" and "unknown".

### Exercise 13.1.02 — Handle a missing dictionary key

Given a dictionary and a key stored in a variable, print the corresponding value or "Missing key" if the key does not exist. Test with `{"A": 10}` and keys "A" and "B".

### Exercise 13.1.03 — Handle an invalid list index

Given a list and an integer index stored in a variable, print the selected element or "Invalid index" if that position does not exist. Test with `[3, 5]` and indexes 1 and 5.

### Exercise 13.1.04 — Raise an invalid quantity

Write require_positive(amount) that raises ValueError with message Amount must be positive when amount <= 0, and otherwise returns amount. Demonstrate the call with 3.

### Exercise 13.1.05 — Handle an invalid quantity

Use require_positive(amount) from the previous exercise. Call it with -4 inside try-except and print the exception message without terminating the program.

### Exercise 13.1.06 — Catch a missing file

Try to read a file named absent.txt and catch FileNotFoundError. Print File not found if the file does not exist.

### Exercise 13.1.07 — Use else after conversion

Given a string variable, convert it to `float` inside `try-except-else`. Print the result only in `else`, or "Invalid number" in `except ValueError`. Test "12.5" and "bad".

### Exercise 13.1.08 — Process valid records

Given records = ["10", "bad", "20", ""], use try-except to sum valid numeric records and skip invalid ones. Print the sum.

### Exercise 13.1.09 — Debug an incorrect average

A program computes the average of [4, 6, 8] by dividing the sum by len(values) - 1. Correct the logic and print the right average.

### Exercise 13.1.10 — A robust file report

Read records.txt and sum its numeric lines. Skip lines that cannot be converted to floats. If the file does not exist, print File not found instead. For the supplied file, print the sum.

## 13.2 File processing, modules and validation

### Exercise 13.2.01 — Apply an imported function to a file

In `helpers.py`, define `double(value)` to return twice an integer. In `main.py`, import it, read integers from `values.txt` (one per line), write the results to `doubled.txt` and print their total as `Total: N`. Assume valid input. For lines 3, 8 and 4, the total is 30.

### Exercise 13.2.02 — Sum CSV values by category

Read `input.csv` with columns `category,quantity` using `csv.DictReader`. Sum integer quantities for each category and print category totals in alphabetical order. Catch `FileNotFoundError` and print `Input file not found` when the file is absent. Otherwise assume valid records. Do not use pandas.

### Exercise 13.2.03 — Filter JSON records through an imported function

In `filters.py`, define `keep_valid(records, minimum)` returning dictionaries whose `"value"` is at least `minimum`. In `main.py`, read a list of dictionaries from `records.json`, call the function with 10, write the selected records to `selected.json` and print their count. Preserve order.

### Exercise 13.2.04 — Read arguments and sum a text file

Use `sys.argv` to read a filename and a positive integer multiplier. Sum the integers in that text file and print `Total: N` after multiplication. Print `Usage: python main.py FILE MULTIPLIER` for an incorrect argument count and `Invalid multiplier` for an invalid multiplier. Use `try-except` for conversion; assume valid file contents.

### Exercise 13.2.05 — Count values with argparse

In `main.py`, use `argparse` for a required filename and an optional integer `--minimum` (default 10). Read integers from the text file and call `count_qualifying(values, minimum)` imported from `filters.py`. Print `Qualifying: N`. Assume the file exists and contains valid numbers.

### Exercise 13.2.06 — Skip invalid lines in a text file

Read `numbers.txt` with one text entry per line. Use `try-except ValueError` to skip entries that cannot be converted to `float`. Write `Count: N` and `Total: X` on separate lines in `summary.txt`, using only valid numbers. Assume the input file exists.

## 14.1 NumPy arrays: creation, shape and indexing

### Exercise 14.1.01 — Create a one-dimensional array

Create a NumPy array from [3, 6, 9, 12]. Print the array and its number of dimensions.

### Exercise 14.1.02 — Check the shape

Create a two-row NumPy array containing [1, 2, 3] and [4, 5, 6]. Print its shape and size.

### Exercise 14.1.03 — A sequence of measurements

Create the NumPy array [2, 4, 6, 8, 10] using np.arange(). Print it.

### Exercise 14.1.04 — Evenly spaced values

Use np.linspace() to create five equally spaced numbers from 0 to 1, including both endpoints.

### Exercise 14.1.05 — Find array elements

Given np.array([10, 20, 30, 40, 50]), print its first, third and last elements.

### Exercise 14.1.06 — Slice an array

Given np.array([5, 10, 15, 20, 25, 30]), print elements at indexes 1 through 4 using slicing.

### Exercise 14.1.07 — Read a matrix cell

Create a 3-by-3 array containing numbers 1 through 9 in row order. Print the entry in its second row and third column.

### Exercise 14.1.08 — Extract one column

Given a 3-by-2 NumPy array [[2, 5], [4, 7], [6, 9]], print the second column.

### Exercise 14.1.09 — Reshape a sequence

Use np.arange() to build the numbers 1 through 12 and reshape them into a 3-by-4 array. Print its shape and contents.

### Exercise 14.1.10 — Change matrix values

Create [[1, 2], [3, 4]] as a NumPy array. Replace the top-right value with 20 and print the updated array.

## 14.2 NumPy: vector operations, masks and reductions

### Exercise 14.2.01 — Scale every observation

Given np.array([4, 6, 8, 10]), calculate and print all values multiplied by 3 without a Python loop.

### Exercise 14.2.02 — Add two arrays

Given arrays [1, 2, 3] and [10, 20, 30], print their element-wise sum.

### Exercise 14.2.03 — Compute daily changes

Given successive prices [10, 12, 11, 15, 14], use array slicing to calculate the change from each day to the next.

### Exercise 14.2.04 — Total and average

Given np.array([6, 8, 4, 10]), print its sum and mean using NumPy.

### Exercise 14.2.05 — Minimum and maximum

Given temperatures [19, 23, 17, 25, 21], print the minimum, maximum and range (maximum minus minimum).

### Exercise 14.2.06 — Select large values

Given [12, 7, 19, 4, 15], print a Boolean mask identifying values greater than 10, then print those values.

### Exercise 14.2.07 — Count valid scores

For scores [8, 13, 5, 17, 10, 14], count how many are at least 10 using a Boolean comparison and NumPy.

### Exercise 14.2.08 — Combine two conditions

Given [3, 8, 12, 17, 21, 25], select numbers that are at least 10 and strictly less than 22.

### Exercise 14.2.09 — Row totals

Given [[3, 5, 2], [4, 1, 6]], calculate the sum of each row using axis=1.

### Exercise 14.2.10 — Column averages

Given [[2, 4, 6], [8, 10, 12]], calculate the mean of each column using axis=0.

## 15.1 pandas: DataFrame construction and selection

### Exercise 15.1.01 — Make a DataFrame

Create a DataFrame with name Alice, Bob, Charlie and score 14, 17, 12. Print it.

### Exercise 15.1.02 — Inspect dimensions

Create a DataFrame with columns city and population and three rows. Print its shape and column names.

### Exercise 15.1.03 — Select one column

Given a DataFrame with product and price columns, select and print the price column as a Series.

### Exercise 15.1.04 — Select two columns

Create a DataFrame with name, age and city for three people. Print only the name and city columns.

### Exercise 15.1.05 — Inspect head and tail

Create a DataFrame with value 10 through 60 by tens. Print its first two and last two rows using .head() and .tail().

### Exercise 15.1.06 — Access one entry with loc

Create a DataFrame indexed by Alice, Bob and Charlie with scores 12, 15 and 18. Print Bob's score using .loc[].

### Exercise 15.1.07 — Access a position with iloc

Create a three-row DataFrame with product and quantity. Print its second row using .iloc[].

### Exercise 15.1.08 — Filter high prices

Create a DataFrame with product prices 4, 12, 7 and 20. Print rows where price is greater than 10.

### Exercise 15.1.09 — Add a calculated column

Create a DataFrame with quantity [2, 3, 5] and unit_price [4, 6, 2]. Add a total column and print the result.

### Exercise 15.1.10 — Update selected rows

Create a DataFrame with scores [9, 14, 7, 18]. Add 2 points only to scores below 10 using .loc[]. Print the updated DataFrame.

## 15.2 pandas: CSV files and missing values

### Exercise 15.2.01 — Read a CSV table

Read prices.csv with pandas and print the complete DataFrame. The file contains product and price columns.

### Exercise 15.2.02 — Inspect CSV records

Read sales.csv and print its shape and first three rows.

### Exercise 15.2.03 — Detect missing values

Create a DataFrame with values [2, None, 5, None] and print its Boolean missing-value mask using .isna().

### Exercise 15.2.04 — Count missing values

Create a DataFrame with temperature [20, None, 23, None] and city ["A", "B", None, "D"]. Print missing-value counts by column.

### Exercise 15.2.05 — Remove incomplete rows

Create a DataFrame with item ["A", "B", "C"] and price [5, None, 8]. Remove incomplete rows with .dropna() and print the result.

### Exercise 15.2.06 — Replace missing values

Given scores [15, None, 12, None, 18], replace missing values by 0 using .fillna(), then print the result.

### Exercise 15.2.07 — Forward fill readings

Given measurements [10, None, None, 13, None, 15], use .ffill() to complete missing positions and print the result.

### Exercise 15.2.08 — Backward fill readings

Given readings [None, 4, None, 8, 10], use .bfill() to complete missing positions and print the result.

### Exercise 15.2.09 — Interpolate missing readings

Given a Series [10, None, 16, None, 24], use .interpolate() to estimate missing values at evenly spaced positions.

### Exercise 15.2.10 — Clean and export a CSV

Read readings.csv with a missing numeric value. Replace missing readings by the column mean, then export only the cleaned table to cleaned_readings.csv without an index.

## 15.3 pandas: grouping and aggregation

### Exercise 15.3.01 — Group and sum

Create a DataFrame with team [A,A,B,B] and points [5,7,3,6]. Use .groupby() to compute each team's total points.

### Exercise 15.3.02 — Group and average

Using departments [IT,HR,IT,HR] and salaries [2000,1800,2200,1900], calculate average salary by department.

### Exercise 15.3.03 — Count rows per group

Given cities [Rome,Rome,Paris,Porto,Paris,Paris], use .groupby().size() to count records by city.

### Exercise 15.3.04 — Find maximum per category

Create a DataFrame with category [A,A,B,B] and value [8,13,5,10]. Print each category's maximum.

### Exercise 15.3.05 — Aggregate two statistics

Using group [A,A,B,B] and amount [3,7,6,8], compute sum and mean per group using .agg().

### Exercise 15.3.06 — Keep the group key as a column

Create a DataFrame with region [North,South,North] and sales [4,7,6]. Calculate sales totals using .groupby(..., as_index=False).

### Exercise 15.3.07 — Named aggregations

Given team [A,A,B,B] and score [4,7,6,9], use .agg() with named outputs total and average.

### Exercise 15.3.08 — Summarize two columns

Given team [A,A,B,B], points [8,12,6,10] and hours [2,4,2,3], group by team and calculate total points and mean hours.

### Exercise 15.3.09 — Group by two columns

For area [N,N,N,S,S], kind [A,B,A,A,B] and sales [4,6,8,3,7], group by area and kind and sum sales.

### Exercise 15.3.10 — Aggregate transactions from a CSV

Read transactions.csv and compute total amount and number of rows for each category. Print the resulting DataFrame.

## 16.1 pandas: merging, pivoting and reshaping

### Exercise 16.1.01 — Merge on a shared key

Create DataFrames for customers (id, name) and orders (id, total). Merge them on id and print the result.

### Exercise 16.1.02 — Left join with a missing match

Create two DataFrames: people with ids [1,2,3] and cities for ids [1,3]. Use a left merge and show the unmatched row.

### Exercise 16.1.03 — Keep only matching rows

Create DataFrames containing ids [1,2,3] and ids [2,3,4]. Use an inner merge and print matching ids.

### Exercise 16.1.04 — Preserve all records

Create tables of ids [1,2] and [2,3]. Merge using how="outer" and print the result, including missing matches.

### Exercise 16.1.05 — Merge with different key names

Create employees with emp_id [1,2] and departments with id [1,2]. Merge using left_on and right_on.

### Exercise 16.1.06 — Pivot an unaggregated table

Create data with day [Mon,Mon,Tue,Tue], product [A,B,A,B] and sales [3,4,5,6]. Use .pivot() to show days as rows and products as columns.

### Exercise 16.1.07 — Aggregate repeated observations

Create records where (day, product) repeats. Use .pivot_table() with aggfunc="sum" to combine the sales.

### Exercise 16.1.08 — Pivot with averages

Create repeated test results for Alice and Bob in subjects Math and Python. Use .pivot_table() to calculate average score for each person-subject pair.

### Exercise 16.1.09 — Fill empty pivot cells

Create sales data where one product is absent on a day. Use .pivot_table(..., fill_value=0) to show 0 instead of NaN.

### Exercise 16.1.10 — Merge customer information from files

Read customers.csv and purchases.csv, merge on customer_id, then group by name and print each customer's total spending.

## 16.2 pandas: sorting, statistics and exporting

### Exercise 16.2.01 — Sort numerical values

Create a DataFrame with product [A,B,C] and price [12,4,9]. Sort by price ascending.

### Exercise 16.2.02 — Sort from largest to smallest

Given names [Alice,Bob,Charlie] and scores [12,19,15], show the rows ordered by descending score.

### Exercise 16.2.03 — Sort by two columns

Given groups [B,A,A,B] and scores [4,8,5,7], sort first by group ascending and then by score descending.

### Exercise 16.2.04 — Select top rows

Given scores [13,9,18,15,11], sort descending and print the first three rows.

### Exercise 16.2.05 — A simple correlation

Create columns x [1,2,3,4] and y [2,4,6,8]. Calculate their correlation with .corr().

### Exercise 16.2.06 — Correlation between two Series

Create two numeric Series [1,2,3,4] and [8,6,4,2]. Calculate their correlation using .corr().

### Exercise 16.2.07 — A covariance matrix

Create DataFrame with columns x [1,2,3] and y [2,4,6]. Print the covariance matrix using .cov().

### Exercise 16.2.08 — Convert to NumPy

Create a DataFrame containing numeric columns a and b. Convert only those columns to a NumPy array with .to_numpy() and print its shape.

### Exercise 16.2.09 — Export sorted results

Create a DataFrame with participant and score for three participants. Sort descending by score, then export ranked.csv without the index.

### Exercise 16.2.10 — Create a clean summary file

Read scores.csv, drop rows with a missing score, compute average scores by team, sort descending and export summary.csv without an index.

## 17.1 Matplotlib: line charts and scatter plots

### Exercise 17.1.01 — A first line plot

Plot days [1,2,3,4] against temperature [18,20,19,22] as a line chart. Save the chart as plot.png.

### Exercise 17.1.02 — Label the axes

Plot months [1,2,3,4] against sales [12,17,15,20], adding x-label Month and y-label Sales. Save the chart as plot.png.

### Exercise 17.1.03 — Add a title

Make a line plot for x [0,1,2,3] and y [0,1,4,9], titled Square values. Save the chart as plot.png.

### Exercise 17.1.04 — Show points on a line

Plot weeks [1,2,3,4] with values [3,6,5,9] as a line with circular markers. Save the chart as plot.png.

### Exercise 17.1.05 — Two lines in one chart

Plot sales of Shop A [5,7,8] and Shop B [4,6,9] over days [1,2,3], adding a legend. Save the chart as plot.png.

### Exercise 17.1.06 — A line plot from a DataFrame

Make a DataFrame with day [1,2,3] and demand [8,11,10], then draw a line plot from its columns. Save the chart as plot.png.

### Exercise 17.1.07 — A first scatter plot

Draw a scatter plot of study hours [1,2,3,4] against scores [8,10,13,16]. Save the chart as plot.png.

### Exercise 17.1.08 — Scatter plot with labels

Draw a scatter plot of height [150,160,170,180] against weight [48,55,65,72], with appropriate axis labels. Save the chart as plot.png.

### Exercise 17.1.09 — Line and scatter together

For x [1,2,3,4] and y [2,3,5,4], draw a line and overlay scatter points using the same coordinates. Save the chart as plot.png.

### Exercise 17.1.10 — Plot a CSV series

Read trend.csv and draw a line plot of its time and value columns with markers and a title. Save the chart as plot.png.

## 17.2 Matplotlib: bars, histograms and box plots

### Exercise 17.2.01 — Vertical bars

Draw a vertical bar chart for products A, B, C with quantities 4, 8, 6. Save the chart as plot.png.

### Exercise 17.2.02 — Horizontal bars

Draw horizontal bars for categories North, South, West with values 9, 5, 7. Save the chart as plot.png.

### Exercise 17.2.03 — Bar chart with a title

Create a bar chart of quarterly sales [10,14,12,18], label the quarters Q1–Q4 and title it Quarterly sales. Save the chart as plot.png.

### Exercise 17.2.04 — Compare grouped totals

Create a DataFrame with team [A,A,B,B] and points [3,4,5,2]. Group and sum points, then plot totals as bars. Save the chart as plot.png.

### Exercise 17.2.05 — Plot numeric frequencies

Draw a histogram for [2,3,4,4,5,6,6,6,7,8] with four bins. Save the chart as plot.png.

### Exercise 17.2.06 — A histogram with labelled axes

Create a histogram of test scores [8,9,10,12,12,15,16,18] with five bins and labelled axes. Save the chart as plot.png.

### Exercise 17.2.07 — Compare two histograms

Draw two overlapping histograms for Group A [2,3,3,4,5] and Group B [4,5,5,6,7], with a legend. Save the chart as plot.png.

### Exercise 17.2.08 — One box plot

Draw a box plot of the measurements [7,9,10,10,11,13,20]. Save the chart as plot.png.

### Exercise 17.2.09 — Two box plots

Draw two side-by-side box plots from [5,7,8,9,10] and [6,8,9,12,14], with group labels A and B. Save the chart as plot.png.

### Exercise 17.2.10 — A box plot after CSV cleaning

Read measurements.csv, remove missing numeric readings and create one box plot of the remaining values. Save the chart as plot.png.

## 17.3 Matplotlib and Seaborn: subplots and heatmaps

### Exercise 17.3.01 — Create two panels

Create a figure with two panels in one row using plt.subplots(1, 2). Draw a line plot in the first and scatter plot in the second. Save the chart as plot.png.

### Exercise 17.3.02 — Vertical panels

Create two vertically stacked panels using plt.subplots(2, 1). Put a bar chart above a line chart. Save the chart as plot.png.

### Exercise 17.3.03 — Four plots in a grid

Create a 2-by-2 grid of subplots and draw four short line plots with different y-values. Save the chart as plot.png.

### Exercise 17.3.04 — Label each subplot

Create two horizontal panels containing line plots, each with its own title and axis labels. Save the chart as plot.png.

### Exercise 17.3.05 — Shared horizontal axis

Create two vertically stacked line plots sharing the x-axis with sharex=True and label the bottom x-axis Day. Save the chart as plot.png.

### Exercise 17.3.06 — Add grid lines

Plot x [1,2,3,4] and y [2,6,4,7], add a grid using .grid() and a title. Save the chart as plot.png.

### Exercise 17.3.07 — Show a legend on an axis

Use fig, ax = plt.subplots() to draw two line series with labels First and Second and add a legend. Save the chart as plot.png.

### Exercise 17.3.08 — Heatmap of a small matrix

Create the 3-by-3 NumPy array [[1,2,3],[4,5,6],[7,8,9]] and display it with seaborn.heatmap(), showing cell annotations. Save the chart as plot.png.

### Exercise 17.3.09 — Heatmap of correlations

Create a DataFrame with three numerical columns a, b and c (three or more rows). Display its .corr() matrix with a labelled seaborn heatmap. Save the chart as plot.png.

### Exercise 17.3.10 — Summary chart and correlation

Create a DataFrame with category [A,A,B,B], x [2,4,3,5] and y [4,5,6,8]. Use two subplots: a bar chart of average x by category, and a heatmap of the numeric-column correlations. Save the chart as plot.png.

## 18.1 Classes, instances and attributes

### Exercise 18.1.01 — A first class

Define a class Book with no methods and create an instance named book_1. Print the type of the new object.

### Exercise 18.1.02 — Two independent objects

Define an empty class Lamp. Create lamp_1 and lamp_2; assign a different color attribute to each and print both colors.

### Exercise 18.1.03 — An initializer for a book

Define a Book class whose __init__() stores title and pages. Create book_1 for a 120-page book titled Atlas and print its attributes.

### Exercise 18.1.04 — A student record

Define Student with name and score instance attributes. Create one Student and print the score in a readable sentence.

### Exercise 18.1.05 — Independent counters

Define Counter with an initial value stored in __init__(). Create two counters with values 2 and 9; increase only the first by 1 and print both values.

### Exercise 18.1.06 — A simple description method

Define Rectangle with width and height, and an .area() method that returns their product. Create rectangle_1 and print its area.

### Exercise 18.1.07 — Read an attribute through a method

Define Car with brand and mileage. Add a .describe() method that returns a string containing both values, then call it on car_1.

### Exercise 18.1.08 — A list of objects

Define Item storing a name and price. Create three Item objects in a list and use a loop to print each name and price.

### Exercise 18.1.09 — Total price from objects

Define Product with name and price. Create a list of three products and use a loop to calculate their total price.

### Exercise 18.1.10 — Constructor parameters and local variables

Define Circle storing radius. In __init__(), use a local variable doubled = 2 * radius, then store self.diameter = doubled. Create a circle of radius 3 and print diameter.

## 18.2 Class attributes and modules

### Exercise 18.2.01 — A class-level unit

Define Temperature with class attribute unit = "C" and instance attribute value. Read unit through Temperature.unit and through an instance.

### Exercise 18.2.02 — A class attribute shared by instances

Define Ticket with class attribute currency = "EUR" and instance attribute price. Create two tickets and print currency from each object.

### Exercise 18.2.03 — Changing a shared class value

Define Device with a class attribute status = "offline". Create two devices, assign Device.status = "online" and print status from both instances.

### Exercise 18.2.04 — An instance attribute hides a class attribute

Define Invoice with class attribute currency = "EUR". Create two invoices, set invoice_a.currency to "USD" and print the currencies of the two instances and the class.

### Exercise 18.2.05 — Avoid a shared list

Define Notebook so every object starts with its own empty notes list in __init__(). Append one note to the first notebook and print both lists.

### Exercise 18.2.06 — Import a class from a module

Create person.py containing Person, whose initializer stores name. In main.py import Person using from person import Person, create person_alice and print its name.

### Exercise 18.2.07 — Use the module-name prefix

Create sensor.py defining Sensor with an identifier attribute. In main.py use import sensor and construct sensor_1 through sensor.Sensor(...). Print its identifier.

### Exercise 18.2.08 — Use an imported method

Define Converter in converter.py with a .double(value) method returning twice the argument. Import the class into main.py, construct converter_1 and print .double(8).

### Exercise 18.2.09 — Two independent imported objects

Define Tank in tank.py storing liters in its initializer. Import Tank into main.py, create tank_a and tank_b with different quantities, change only tank_a and print both.

### Exercise 18.2.10 — Multiple calls on a class from another file

Define Counter in counter.py with .increase() adding 1 to the stored value. In main.py import the class, create counter_1 starting at 0, call .increase() three times and print the result.

## 19.1 Methods and controlled state changes

### Exercise 19.1.01 — Increase stored stock

Define Stock with quantity. Implement .add(amount) to increase its quantity and return the new quantity. Test with an initial stock of 4 and an addition of 3.

### Exercise 19.1.02 — Decrease stored fuel

Define FuelTank with liters. Implement .consume(amount) to decrease liters, without validation for this exercise. Call it twice and print the final liters.

### Exercise 19.1.03 — Cap an amount at zero

Define Battery with charge. Its .use(amount) method must never reduce charge below 0. Start at 12, use 5 and then 20, and print the final charge.

### Exercise 19.1.04 — Capacity-limited addition

Define WaterTank with liters and capacity. Implement .add(amount) that stops at capacity. Starting at 20 with capacity 30, add 6 and then 9; print the result.

### Exercise 19.1.05 — Reject an invalid price

Define Product with price and a .set_price(value) method that returns False if value is negative; otherwise update the price and return True. Test both cases.

### Exercise 19.1.06 — Return the amount actually removed

Define Warehouse with quantity. Its .dispatch(requested) method removes at most the available quantity and returns the number removed. Test a request for 8 from a stock of 5.

### Exercise 19.1.07 — Count successful uses

Define AccessCard with remaining uses and successful_uses initially zero. Its .use() returns True and updates both attributes if uses remain; otherwise it returns False. Test three attempts starting with 2 uses.

### Exercise 19.1.08 — Transfer between two objects

Define Wallet with balance and a .transfer(other, amount) method. If amount is positive and not greater than this wallet balance, subtract it here, add it to other and return True; otherwise return False. Test one successful transfer.

### Exercise 19.1.09 — Apply rules to a list of objects

Define SeatGroup with free_seats and .reserve(count) returning True only if count is positive and enough seats remain. Create two groups and loop through one reservation request for each.

### Exercise 19.1.10 — Find the object with greatest state

Define Team with name and points, plus .add_points(value) for positive values. Create three teams, update their points and print the name of the team with the largest total.

## 19.2 Raising and handling exceptions

### Exercise 19.2.01 — Raise on nonpositive input

Define Account with balance and .deposit(amount) that raises ValueError if amount is not positive. Start with 10 and deposit 5; print the new balance.

### Exercise 19.2.02 — Catch a rejected deposit

Extend Account so .deposit() raises ValueError for nonpositive amounts. Use try-except around account_1.deposit(-3) and print a short error message. Verify the balance is unchanged.

### Exercise 19.2.03 — Reject a withdrawal exceeding the balance

Define Account with .withdraw(amount). Raise ValueError if amount is nonpositive or larger than balance; otherwise subtract it. Handle a withdrawal of 20 from balance 15 using try-except.

### Exercise 19.2.04 — Protect a grade range

Define GradeRecord with score and a .set_score(value) method. Raise ValueError unless the value is between 0 and 20 inclusive. Catch an invalid attempt and print the unchanged score.

### Exercise 19.2.05 — Validate an input used by a method

Define Membership with .add_months(amount) that raises ValueError for nonpositive amounts. Read an integer number of months using int(input(...)) and catch the method error. Assume the user enters a valid integer.

### Exercise 19.2.06 — Recover from invalid numeric text

Define Converter with .read_number(text) that uses try-except inside the method. Return float(text) when conversion succeeds and None when it fails. Test strings "3.5" and "wrong".

### Exercise 19.2.07 — Handle a missing dictionary key

Define PriceList storing a dictionary of prices. Its .get_price(name) method should return the price or None if the key is missing, using try-except KeyError internally. Test one present and one missing product.

### Exercise 19.2.08 — Different reasons for failure

Define Wallet with .pay(amount). Raise ValueError for a nonpositive amount and for an amount exceeding balance, using different messages. Test both failures with separate try-except calls.

### Exercise 19.2.09 — Keep processing after one failure

Define Stock with quantity and .remove(amount) raising ValueError for nonpositive or excessive requests. Process requests [3, 20, 2] in a loop, catching each failure so later requests still run. Start with stock 10.

### Exercise 19.2.10 — Internal handling versus an external error

Define Sensor with .set_reading(text). Inside the method, catch a failed float conversion and return False; for a valid number, update the stored reading and return True. Test "12.5", "oops" and "9".

## 20.1 Inheritance and derived classes

### Exercise 20.1.01 — Create a derived class

Define Animal storing name. Define Dog(Animal) with pass, create dog_buddy and print its inherited name.

### Exercise 20.1.02 — Use an inherited method

Define Person storing name and .introduce() returning the name. Define Student(Person) without new methods and call .introduce() on student_alice.

### Exercise 20.1.03 — Add a new method in a child

Define Vehicle storing brand. Define Bicycle(Vehicle) with a new .ring_bell() method returning "Ring!". Create bicycle_1 and print its brand and .ring_bell().

### Exercise 20.1.04 — Specialize a basic animal

Define Animal storing name and age. Define Cat(Animal) adding .purr() returning "Purr". Instantiate cat_luna and print age and the result of .purr().

### Exercise 20.1.05 — Reuse initialization with super()

Define Person with initializer storing name. Define Employee(Person) with a new initializer accepting name and department; call super().__init__(name) then store department. Print both attributes.

### Exercise 20.1.06 — An extra numeric attribute

Define Account storing balance. Define SavingsAccount(Account) with an initializer accepting balance and rate, using super().__init__(). Add .interest() returning balance * rate and print it for balance 200 and rate 0.05.

### Exercise 20.1.07 — Inherited changes to shared state

Define Counter with value and .increase(). Define LabeledCounter(Counter) with initializer adding a label via super().__init__(). Call .increase() twice and print label and value.

### Exercise 20.1.08 — List with parent and child objects

Define Animal with .describe() returning its name. Define Dog(Animal) with pass. Place an Animal and a Dog in one list and call .describe() on each.

### Exercise 20.1.09 — Reuse a method from a base class

Define Product with price and .discount(amount) subtracting amount. Define Food(Product) adding an expiry_day attribute using super().__init__(). Discount a Food instance and print its remaining price.

### Exercise 20.1.10 — One base class and two children

Define Person storing name. Define Student(Person) with .study() returning "studying" and Teacher(Person) with .teach() returning "teaching". Create one of each and print their names and actions.

## 20.2 Overriding and super()

### Exercise 20.2.01 — Different animal sounds

Define Animal with .make_sound() returning "Some sound". Override .make_sound() in Dog to return "Woof". Print the results from Animal and Dog objects.

### Exercise 20.2.02 — Two specialized sounds

Define Animal with .make_sound(). Create Dog and Cat subclasses that override the method with "Woof" and "Meow". Call it on one object of each child class.

### Exercise 20.2.03 — Override a description

Define Person storing name and returning "Person: <name>" from .describe(). Define Student(Person) overriding .describe() to return "Student: <name>". Print both descriptions.

### Exercise 20.2.04 — Call an overridden method through a loop

Define Shape with .label() returning "shape" and Circle(Shape) overriding it to return "circle". Store one instance of each in a list and print .label() from a loop.

### Exercise 20.2.05 — Extend a description with super()

Define Person storing a name, with .describe() returning that name. Define Student(Person) overriding .describe() to append " - student" to the description obtained through super().describe(). Create a Student named Alice and print the result.

### Exercise 20.2.06 — Parent initialization plus overriding

Define Vehicle storing brand and returning it from .describe(). Define ElectricCar(Vehicle) storing range_km via super().__init__() and overriding .describe() to append the range. Print one description.

### Exercise 20.2.07 — Overriding a calculation

Define Ticket storing price with .final_price() returning price. Define DiscountTicket(Ticket) with rate, using super().__init__(), overriding .final_price() to return the discounted price. Test price 50 and rate 0.2.

### Exercise 20.2.08 — Extend an action instead of replacing it

Define Counter with .increase() adding 1 to value. Define DoubleCounter(Counter) overriding .increase(): call super().increase() twice. Start at 0, call once and print value.

### Exercise 20.2.09 — Different attacks

Define Character storing name and power with .attack() returning power. Define Mage(Character) with bonus and overriding .attack() to return super().attack() + bonus. Print attack values of a normal character and a mage.

### Exercise 20.2.10 — Polymorphic cost computation

Define Delivery with .cost() returning its base price. Define ExpressDelivery adding a surcharge and overriding .cost() using super().cost(). Place standard and express deliveries in a list and calculate the sum of .cost() results.

## 20.3 Custom exceptions and inheritance

### Exercise 20.3.01 — Create an exception type

Define a custom exception class InvalidScoreError inheriting from ValueError. Raise it with a message for an invalid score and catch it using except InvalidScoreError.

### Exercise 20.3.02 — Raise a custom error from a method

Define InsufficientWaterError(ValueError) and Tank with .take(amount). Raise the custom exception when amount exceeds liters; otherwise subtract it. Catch an excessive request and print the remaining liters.

### Exercise 20.3.03 — Catch a child using the parent type

Define MissingTicketError(ValueError) and raise it inside try. Catch the error using except ValueError rather than its specific name and print a message.

### Exercise 20.3.04 — Specific except before a general one

Define OverLimitError(ValueError). Raise it from a try block and provide except OverLimitError first and except ValueError second. Print which branch runs.

### Exercise 20.3.05 — Two error types in one class

Define TooYoungError(ValueError) and TooOldError(ValueError). Define Registration with .set_age(age) raising these errors for ages below 18 and above 90; otherwise store the age. Try ages 12, 40 and 100, handling both error types separately.

### Exercise 20.3.06 — Class and custom error in separate modules

In errors.py define LimitError(ValueError). In counter.py import it and define LimitedCounter with .increase() that raises LimitError after reaching a fixed limit. In main.py catch LimitError during repeated calls.

### Exercise 20.3.07 — One specific error, one standard error

Define OutOfStockError(ValueError) and Shop with .buy(amount). Raise ValueError for nonpositive amount and OutOfStockError for insufficient stock. Catch the specific error before ValueError for requests 5, -1 and 20.

### Exercise 20.3.08 — An inherited method raising a custom error

Define CapacityError(ValueError), Container storing quantity and capacity, and .add(amount) that raises CapacityError when the limit would be exceeded. Define Box(Container) with pass. Create a Box and catch the inherited method error.

### Exercise 20.3.09 — Override a method while preserving its validation

Define InvalidWithdrawal(ValueError) and Account with .withdraw(amount) that rejects nonpositive or excessive amounts. Define FeeAccount(Account) overriding .withdraw(): use super().withdraw(amount + 1) to include a fee. Catch the error for an attempted withdrawal of 10 from balance 10.

### Exercise 20.3.10 — Different exception responses for specialized accounts

Define InsufficientFundsError(ValueError), BankAccount with .withdraw(amount) raising it when the balance is insufficient, and SavingsAccount(BankAccount) with .add_interest(rate) updating the inherited balance. Start with a SavingsAccount balance of 100. Attempt to withdraw 150 and catch InsufficientFundsError without changing the balance. Then apply an interest rate of 0.05 and print the final balance (105).