Programming Techniques
Project Catalogue

Student version

Python programming course

Projects calibrated for approximately 15–20 days of work

Contents

Project scope and PCP **2**

Common mandatory technical core **2**

Data policy and instance generators **3**

Project selection and assignment **3**

What to submit **3**

How to read a project card **4**

Project 1 – Personal Budget & Expense Analyzer **5**

Project 2 – Sales & Inventory Management System **6**

Project 3 – Loan & Mortgage Simulator **7**

Project 4 – Savings & Investment Simulator **8**

Project 5 – Pricing & Promotion Analyzer **9**

Project 6 – Customer Satisfaction & Survey Analyzer **10**

Project 7 – Employee Attendance & Working-Time Analyzer **11**

Project 8 – Workforce Shift Planning Assistant **12**

Project 9 – Hotel Booking & Occupancy Manager **13**

Project 10 – Restaurant Orders & Menu Performance Analyzer **14**

Project 11 – Event & Ticket Management System **15**

Project 12 – Warehouse & Order Fulfilment System **16**

Project 13 – Small Business KPI Dashboard **17**

Project 14 – Energy Consumption & Cost Analyzer **18**

Project 15 – Urban Mobility Demand Analyzer **19**

Project 16 – Service Desk Queue Simulator **20**

Project 17 – Supply & Demand Market Simulator **21**

Project 18 – Media Collection & Recommendation Tool **22**

Project 19 – Population / Epidemic Dynamics Simulator **23**

Project 20 – Tournament / Game Simulation Manager **24**

Alternative project proposals **25**

1. Project scope and Project Complexity Points (PCP)

The project is a small but complete Python application that combines the programming concepts covered in the course. It is not intended to become a large software product. The catalogue is calibrated so that a correctly selected configuration is suitable for approximately **15–20 days of work while students are also preparing for the final assessment**.

The required size depends on the number of students in the group:

Group size	Required project size
1 student	9 PCP
2 students	14 PCP
3 students	18 PCP

Each project-specific feature is labelled **M**, **S**, or **E**:

- **M – Mandatory core**: must always be implemented. Mandatory features cannot be removed, regardless of group size.
- **S – Secondary core**: part of the full core, but may be omitted when a smaller configuration is needed, provided the required PCP total is still reached.
- **E – Extension**: additional functionality selected to reach the PCP threshold for the group.

For a 2-student project, the selected extensions must include at least one feature worth **2 PCP or more**. For a 3-student project, the extensions must include either at least one **3-PCP** feature or at least two features worth **2 PCP or more**. This prevents the project from being enlarged only through many very small additions.

The line **Full core** shown in each project card already includes the 4 PCP common technical core described below. For example, a project with a 10-PCP full core normally needs 4 additional PCP of extensions for a 2-student group and 8 additional PCP for a 3-student group. A 1-student version may omit secondary-core features until the project reaches 9 PCP, but may never remove mandatory features.

Examples of valid sizing

For a project with a 10-PCP full core:

- a 1-student version can implement all mandatory features plus enough secondary core to reach exactly 9 PCP;
- a 2-student version could implement the 10-PCP full core plus two 2-PCP extensions = 14 PCP;
- a 3-student version could implement the 10-PCP full core plus, for example, two 3-PCP extensions and one 2-PCP extension = 18 PCP.

The objective is not to maximize the score. Choose a coherent configuration that reaches the required threshold without unnecessary scope.

2. Common mandatory technical core – 4 PCP

Every project automatically contains the following requirements:

- **C01 (1 PCP) File input/output and validation.** Read the specified project input files, validate relevant identifiers, data types, and value ranges, and handle invalid records clearly. Save processed data or computed results to at least one output file.
- **C02 (1 PCP) Program structure and command line.** The submission must contain `main.py` plus at least one separate module containing functions. Use meaningful command-line arguments, normally including an input/output path and/or a project-specific selection such as scenario, period, or threshold.
- **C03 (1 PCP) Summary report generation.** Generate a structured summary report bringing together the main results produced by the application. Identify the selected input or scenario, include relevant indicators, and provide enough context to interpret the results.
- **C04 (1 PCP) Representation of results.** Include at least one meaningful textual table, ASCII histogram, chart, or other readable representation of the results.

At least one substantial part of the application must be implemented using the fundamental Python constructs covered in the course: functions, conditions, loops, strings and native data structures. A project must not reduce its core logic to a few high-level library calls. For 2- and 3-student groups, the selected configuration should normally

include at least one meaningful use of `pandas` and/or `matplotlib`; if neither is appropriate, agree an equivalent extension with the instructor.

Classes are **optional unless explicitly agreed otherwise**. A suggested class is listed only where object-oriented modelling is natural. Using a class does not by itself add PCP.

3. Data policy and instance generators

Use the exact course files associated with the selected project. The data are intentionally sized so that the programming task, rather than data hunting or heavy cleaning, remains the main workload.

- **Synthetic course data** are designed specifically to contain relevant cases such as shortages, conflicts, alerts, imbalances, or unusual observations.
- **Adapted real data** are cleaned/reduced versions of public datasets. The original public source is cited for provenance and optional exploration; the course-adapted files are provided after project assignment.
- **Simulation scenarios** contain parameters for deterministic or stochastic models rather than observational data.

When an **instance generator** appears as an extension, a common reference instance is still provided and must remain usable by the application. The generator is an additional program component that creates new valid files in the same schema. Unless the project says otherwise, it should accept meaningful size/intensity parameters, use `random`, allow the random seed to be controlled for reproducibility, and generate test cases that are meaningful for the problem rather than arbitrary values.

4. Project selection and assignment

The catalogue is intentionally detailed enough to support an informed project choice. Students should use the project cards to compare the problem, input data, core tasks, possible extensions, and programming focus before proposing a project.

Once a project has been agreed with the instructor, the group receives the **project-specific working package**. The package contains a detailed specification and configuration checklist for the same project, the associated course data/supporting files, illustrative input/output examples, and any project-specific notes needed to make the implementation requirements unambiguous. The detailed specification clarifies what must be implemented, but it does not prescribe how the program must be designed or implemented unless a requirement explicitly states a rule that must be followed.

Unless otherwise agreed, each catalogue project can be assigned to **one group only**. After assignment, that project is marked as unavailable and is removed from the list of open choices. The final PCP configuration (group size, included secondary-core tasks, and selected extensions) must be agreed before implementation.

5. What to submit

Submit a runnable Python project containing:

- all `.py` source files, including `main.py` and the required module(s);
- a `README.md` with group members, the selected project, the agreed requirement IDs (C/M/S/E) and PCP total, required libraries, exact commands to run the program, and a concise implementation summary mapping each included requirement to the relevant source files/functions and explaining how it was implemented; this summary may instead be placed in a separate file linked from the README;
- the report/output files generated by one representative run;
- any additional input instances created by your program if an instance-generator extension is selected;
- any other small supporting files required by the selected extensions.

The implementation summary should briefly explain *how* each agreed requirement was fulfilled and *where* to find its implementation (for example: requirement ID, source file/function, and one- or two-sentence explanation). A compact table is acceptable. Include all common and mandatory requirements as well as the optional tasks selected in the agreed PCP configuration. This is documentation of the implemented scope, not an additional PCP requirement.

The main application must run as a Python program from the command line. A notebook may be submitted as supplementary analysis, but it must not replace the required structured application. Groups are free to explore

additional technologies, including a GUI, database, web application or machine learning, if they wish. Such additions do not automatically contribute extra PCP unless explicitly included in the project specification or agreed with the instructor in advance. Keep the agreed requirements and runnable command-line application available regardless of optional additions.

During the project discussion, every group member may be asked to explain the program structure, data flow, algorithms, selected extensions, and relevant Python constructs.

6. How to read a project card

Each project card contains a problem description, an overview of the data that will be provided after assignment, the expected result, project-specific core tasks, optional extensions, and the full-core PCP total. The **common 4-PCP technical core applies to every project and is not repeated in each card**. Mandatory project tasks are therefore the domain-specific work on top of those common requirements. The catalogue card defines the project scope used for selection. After assignment, the project-specific specification expands these same requirements into numbered implementation tasks, validation/edge-case expectations, and illustrative executions. No hidden functionality is introduced after the project has been chosen.

Project 1 – Personal Budget & Expense Analyzer

Description

A person wants to understand how money is spent over time and whether current spending habits are compatible with predefined monthly budgets. The available data contain both income and expenses, together with dates, descriptions, categories and amounts. The application must turn these individual transactions into a compact financial overview, identify budget problems and make comparisons across time. The final program should answer questions such as how much was earned and spent, which categories dominate expenses, which monthly limits were exceeded, and how spending changed from one month to another.

Input data

The group will receive two synthetic CSV datasets: a record of personal income and expense transactions and a set of monthly spending limits by category. Transactions cover six months and include dates, descriptions, amounts and spending categories. The reference instance deliberately contains recurring expenses, several budget violations and a few unusually large transactions.

Expected result

The application should produce a financial summary for the selected period, a category-level budget report, explicit alerts for exceeded limits, and at least one readable representation of the spending distribution.

Project-specific core tasks

- **M01 (2 PCP)** Compute total income, total expenses, balance, and expense totals by category for the selected period. Monthly totals must also be available because budget limits are monthly.
- **M02 (1 PCP)** Compare monthly category spending with `monthly_limit`. Report every violation and the amount by which the limit is exceeded.
- **M03 (1 PCP)** Produce a category distribution showing both amount and percentage of total expenses; present it as a structured table or equivalent representation.
- **S01 (1 PCP)** Compare at least two months using the same indicators and clearly report increases/decreases.
- **S02 (1 PCP)** Rank the highest-spending categories and report the top categories for the selected period.

Full core: 10 PCP (including the 4-PCP common technical core).

Extensions

- **E01 (1 PCP)** Allow category limits to be overridden or selected through command-line options.
- **E02 (1 PCP)** Add a Matplotlib chart of spending by category or month.
- **E03 (2 PCP)** Build an instance generator that writes valid transaction and budget files with configurable period, number of transactions, category profiles, and random seed.
- **E04 (2 PCP)** Add a Pandas-based monthly/category analysis that complements, rather than replaces, the core Python implementation.
- **E05 (2 PCP)** Detect recurring expenses using a documented matching rule based on description/category/amount and recurrence over time.
- **E06 (3 PCP)** Detect unusually large expenses using a transparent rule based on the history of each category and explain why each item is flagged.
- **E07 (3 PCP)** Compare two alternative budgeting strategies or sets of category limits and summarize which categories improve or deteriorate.

Suggested class (optional): `Transaction or Budget`.

Core programming focus: classification, aggregation, rules, alerts, and time-based comparison.

Project 2 – Sales & Inventory Management System

Description

A small company sells a catalogue of products while keeping a limited quantity of each item in stock. Sales arrive over time and the inventory must remain consistent after every transaction. If a request cannot be satisfied, the system must detect the shortage rather than allow negative stock. The company also wants reorder warnings and simple sales indicators. The application therefore combines sequential transaction processing with inventory control and reporting.

Input data

The group will receive two synthetic CSV datasets describing a company's product catalogue and a chronological history of sales requests. Product information includes prices, opening stock and reorder thresholds; sales indicate what was requested and when. The reference data contain fast- and slow-moving products, reorder cases and requests that exceed the stock available at the time.

Expected result

The application should write an updated inventory table, a log/summary of fulfilled and rejected sales, sales indicators, and explicit shortage/reorder alerts.

Project-specific core tasks

- **M01 (3 PCP)** Process sales in ascending `date` and then `sale_id` for ties, and update stock after each accepted sale. In the core policy a sale is all-or-nothing: if the full requested quantity is unavailable, reject the sale, record the shortage, and do not change stock.
- **M02 (1 PCP)** Detect products below their reorder level and all sales rejected for insufficient stock.
- **M03 (1 PCP)** Compute total fulfilled units, fulfilled revenue, final stock, and at least one stock-utilization indicator.
- **S01 (1 PCP)** Rank products by fulfilled sales quantity or revenue.
- **S02 (1 PCP)** Produce a category-level summary of sales and remaining stock.

Full core: 11 PCP (including the 4-PCP common technical core).

Extensions

- **E01 (1 PCP)** Suggest a reorder quantity using a clearly documented rule.
- **E02 (1 PCP)** Make one or more stock/reorder thresholds configurable from the command line.
- **E03 (2 PCP)** Build an instance generator for catalogues, stock levels and sales transactions, with configurable sizes, demand intensity and random seed.
- **E04 (2 PCP)** Add a Pandas inventory/sales analysis that complements the sequential processing logic.
- **E05 (2 PCP)** Add a graphical visualization of stock levels, sales, or reorder status.
- **E06 (3 PCP)** Compare the core all-or-nothing policy with a partial-fulfilment policy and quantify the difference.
- **E07 (3 PCP)** Simulate uncertain future demand and estimate which products are likely to require replenishment.
- **E08 (3 PCP)** Allocate limited stock fairly among a batch of competing orders using a documented fairness rule.

Suggested class (optional): `Product or Order`.

Core programming focus: state updates, transaction processing, shortage detection, and decision rules.

Project 3 – Loan & Mortgage Simulator

Description

A customer wants to compare fixed-rate loans or mortgages before selecting an offer. Each scenario defines the borrowed amount, annual interest rate, duration and payment frequency. The application must generate the repayment schedule period by period, separate interest from principal, track the remaining balance and summarize the total cost of borrowing. The aim is programming and numerical simulation, not advanced financial modelling.

Input data

The group will receive a synthetic CSV dataset of loan and mortgage offers with varying principal amounts, interest rates, repayment frequencies and durations. Some scenarios also specify an affordability limit for periodic payments. These are instructor-designed financial examples rather than observations of real lending contracts; they include cases suitable for comparing repayment costs and feasibility.

Expected result

For a selected scenario, produce a repayment schedule with period, opening balance, payment, interest, principal repayment and closing balance, plus a final summary. Comparisons should use the same indicators.

Project-specific core tasks

- **M01 (3 PCP)** Generate a fixed-rate amortization schedule. Use periodic rate `r = annual_rate / payments_per_year` and `n = years * payments_per_year`. Let `L` be the loan principal. For `r > 0`, use periodic payment `L*r*(1+r)^n / ((1+r)^n - 1)`; for zero interest use `L/n`. Adjust the final period if needed to avoid a small negative balance caused by floating-point rounding.
- **M02 (1 PCP)** Compute total interest, total amount repaid, and the periodic payment.
- **M03 (1 PCP)** Track and report the remaining debt over time.
- **S01 (1 PCP)** Compare at least two loan scenarios, including periodic payment and total interest/cost.

Full core: 10 PCP (including the 4-PCP common technical core).

Extensions

- **E01 (1 PCP)** Export a detailed amortization schedule to a separate CSV file.
- **E02 (2 PCP)** Support extra repayments and show their effect on duration and total interest.
- **E03 (2 PCP)** Support a documented variable-rate scenario in which the rate changes at predefined periods. Define and include a small additional rate-schedule CSV (or an equivalent clearly structured input) used by the program.
- **E04 (2 PCP)** Plot the remaining balance and cumulative interest with Matplotlib.
- **E05 (2 PCP)** Create a Pandas comparison table for all scenarios in the input file.
- **E06 (3 PCP)** Search the provided scenarios and identify the cheapest loan satisfying a maximum periodic-payment constraint.
- **E07 (3 PCP)** Perform an interest-rate sensitivity analysis around a selected scenario.

Suggested class (optional): `Loan`.

Core programming focus: iterative numerical computation, schedules, scenario comparison, and financial indicators.

Project 4 – Savings & Investment Simulator

Description

A person wants to plan a medium- or long-term savings strategy. They start with an initial capital and add a fixed contribution several times per year. Capital grows according to a deterministic annual return in the core model. The application should show how the account evolves, distinguish money contributed by the user from investment gains, and compare alternative saving strategies. Stochastic returns are optional extensions rather than part of the basic model.

Input data

The group will receive a synthetic CSV dataset of savings and investment scenarios with different starting capitals, contribution schedules, time horizons and expected annual returns. The figures describe stylized strategies for a deterministic teaching model, not historical market performance. They allow the same savings indicators to be compared under different assumptions.

Expected result

Produce the balance evolution for a selected scenario, cumulative contributions, accumulated investment gain, and a final summary. A comparison should make clear why two strategies differ.

Project-specific core tasks

- **M01 (2 PCP)** Simulate the capital period by period using the specified update order and store enough information to reconstruct the evolution.
- **M02 (2 PCP)** Track separately the initial capital, cumulative new contributions, and the gain generated by returns.
- **M03 (1 PCP)** Compute final capital, total contributions, total gain, and a growth indicator.
- **S01 (1 PCP)** Compare at least two deterministic strategies over their common horizon.

Full core: 10 PCP (including the 4-PCP common technical core).

Extensions

- **E01 (1 PCP)** Add inflation adjustment and report nominal and real final values.
- **E02 (2 PCP)** Replace the deterministic return with a documented random-return model and allow a random seed.
- **E03 (2 PCP)** Run repeated simulations and display a histogram of final capital.
- **E04 (2 PCP)** Support a contribution schedule that changes during the investment horizon.
- **E05 (2 PCP)** Plot the evolution of two or more strategies with Matplotlib.
- **E06 (3 PCP)** Perform a Monte Carlo comparison of at least two investment strategies.
- **E07 (3 PCP)** Estimate the probability of reaching a target capital by the end of the horizon.
- **E08 (3 PCP)** Compare strategies with simple risk/return indicators derived from repeated simulations.

Suggested class (optional): `InvestmentScenario`.

Core programming focus: time-step simulation, accumulation, scenario comparison, and optional stochastic analysis.

Project 5 – Pricing & Promotion Analyzer

Description

A company wants to evaluate the economic effect of discounts before applying them. Product data contain regular selling prices and unit costs, while sales records provide observed quantities. The core analysis treats observed quantities as fixed: a promotion changes the selling price and therefore revenue and margin, but does not automatically assume that demand increases. More realistic demand responses can be explored only in extensions.

Input data

The group will receive three related synthetic CSV datasets: a product catalogue with selling prices and unit costs, a history of observed sales quantities, and a selection of proposed discounts. Promotions can apply to a single product or a product category. The data support profitability comparisons while keeping the observed sales quantities fixed in the basic analysis.

Expected result

Produce a baseline profitability report, the effect of each proposed promotion under the fixed-quantity assumption, and warnings for products/promotions that create unacceptable margins.

Project-specific core tasks

- **M01 (2 PCP)** Compute baseline units sold, revenue, cost, gross margin, and margin percentage by product.
- **M02 (2 PCP)** Apply each promotion to the appropriate product(s), recompute price/revenue/margin for the same quantities, and compare the promotional result with the baseline.
- **M03 (1 PCP)** Flag negative or low-margin cases using a documented threshold and report the reason for each alert.
- **S01 (1 PCP)** Aggregate and compare profitability by product category.

Full core: 10 PCP (including the 4-PCP common technical core).

Extensions

- **E01 (1 PCP)** Make the low-margin threshold configurable.
- **E02 (2 PCP)** Build an instance generator for products and sales with configurable demand profiles and random seed.
- **E03 (2 PCP)** Add Pandas analysis by category and/or time period.
- **E04 (2 PCP)** Visualize prices, revenue or margins with Matplotlib.
- **E05 (2 PCP)** Compare several predefined promotion policies in one report.
- **E06 (3 PCP)** Search a range of discount levels and identify the maximum discount that still preserves a specified minimum gross-margin percentage; report the feasible range and chosen value.
- **E07 (3 PCP)** Introduce a transparent uncertain demand-response model in which quantity may change with the discount.
- **E08 (3 PCP)** Compare promotions using two objectives, such as revenue growth and margin preservation, without collapsing them into an unexplained single score.

Suggested class (optional): `Product or Promotion`.

Core programming focus: business-rule evaluation, profitability analysis, alerts, and simple decision support.

Project 6 – Customer Satisfaction & Survey Analyzer

Description

A company has collected satisfaction ratings for several service dimensions together with basic respondent attributes. Management wants an overall view of satisfaction, wants to identify weak service dimensions, and wants to understand whether some customer segments systematically report better or worse experiences. The data contain only a few invalid or missing values so that validation is present but does not dominate the project.

Input data

The group will receive a synthetic CSV dataset of customer satisfaction survey responses, including basic respondent characteristics, service channels and numerical ratings of several service dimensions. The data contain different customer segments and intentionally include a few missing or invalid ratings, allowing validation to be addressed without turning the project into a data-cleaning exercise.

Expected result

Produce a validation summary, overall and segment-level satisfaction indicators, rating distributions, rankings/alerts for weak dimensions, and a concise report.

Project-specific core tasks

- **M01 (2 PCP)** Validate rating fields, report invalid/missing values, and exclude or handle invalid records according to a documented rule before computing statistics.
- **M02 (2 PCP)** Compute satisfaction indicators for at least two meaningful segmentation dimensions, for example customer type, channel, or age group.
- **M03 (1 PCP)** Compute overall indicators for each survey dimension and a rating distribution for at least one dimension.
- **S01 (1 PCP)** Rank the best- and worst-performing satisfaction dimensions.

Full core: 10 PCP (including the 4-PCP common technical core).

Extensions

- **E01 (1 PCP)** Add textual histograms of rating distributions.
- **E02 (1 PCP)** Add Matplotlib rating-distribution charts.
- **E03 (2 PCP)** Build a survey instance generator with configurable segment sizes, satisfaction profiles, missing-value rate, and random seed.
- **E04 (2 PCP)** Add Pandas cross-segment analysis using two respondent attributes at the same time.
- **E05 (2 PCP)** Create a report that can be filtered from the command line by segment and/or survey dimension.
- **E06 (3 PCP)** Compare at least two documented weighting schemes for an overall satisfaction indicator.
- **E07 (3 PCP)** Identify priority segments by combining low satisfaction with segment size using a transparent priority rule.

Core programming focus: validation, classification, aggregation, segmentation, and ranking.

Project 7 – Employee Attendance & Working-Time Analyzer

Description

An organization records daily working time and attendance status for its employees. Managers need a tool to summarize weekly and overall hours, detect repeated overtime or under-time, and compare workload patterns across departments. The project is an operational analysis tool, not a payroll system: the emphasis is on record processing, transparent rules and reporting.

Input data

The group will receive two synthetic CSV datasets: employee profiles with departments and contractual weekly hours, and attendance records covering eight consecutive working weeks. The records indicate worked hours and attendance status. The reference instance includes absences, overtime, under-time and recurring workload imbalances.

Expected result

Produce weekly/overall employee summaries, explicit alerts, department-level information where selected, and a report that makes repeated patterns easy to identify.

Project-specific core tasks

- **M01 (2 PCP)** Compute total and average worked hours for each employee, including weekly totals based on the dates in the file.
- **M02 (1 PCP)** Compare weekly worked hours with `weekly_contract_hours`. In the core, classify a week as under-time when worked hours are more than 2 hours below the contract and as overtime when they are more than 2 hours above it; otherwise classify it as within tolerance.
- **M03 (1 PCP)** Generate employee-level alerts summarizing the relevant weeks and the reason for each alert.
- **S01 (1 PCP)** Compare workload indicators across departments.
- **S02 (1 PCP)** Detect repeated absence, overtime, or under-time patterns across multiple weeks.

Full core: 10 PCP (including the 4-PCP common technical core).

Extensions

- **E01 (1 PCP)** Make selected alert thresholds configurable from the command line.
- **E02 (2 PCP)** Add monthly/weekly trend analysis and a time-based representation.
- **E03 (2 PCP)** Build an attendance instance generator with configurable employees, weeks, absence/overtime rates, and random seed.
- **E04 (2 PCP)** Add a Pandas-based department and employee analysis.
- **E05 (3 PCP)** Create a simple workload-fairness analysis across employees or departments using a documented indicator.
- **E06 (3 PCP)** Detect recurrent workload imbalance and rank the employees/teams most affected.

Suggested class (optional): `Employee`.

Core programming focus: record processing, threshold rules, alerts, aggregation, and fairness indicators.

Project 8 – Workforce Shift Planning Assistant

Description

A manager must assign available employees to a weekly set of shifts. Each shift requires a minimum number of workers, but employee availability is limited. The objective is not to solve a mathematical optimization model: students must implement a transparent heuristic, verify the resulting coverage, and report understaffed shifts and workload distribution. The optional role file is reserved for an advanced extension.

Input data

The group will receive synthetic CSV datasets describing employees, their availability for a week of shifts, and staffing needs for each shift. The basic reference instance is feasible with respect to availability, making it suitable for comparing transparent assignment heuristics. An additional dataset of role requirements is provided for groups choosing the advanced role-aware extension.

Expected result

Produce an assignment file, coverage status for every shift, understaffing alerts if the chosen heuristic leaves gaps, and workload totals by employee.

Project-specific core tasks

- **M01 (3 PCP)** Implement and document a deterministic assignment heuristic. It must assign only available employees, never assign the same employee twice to the same shift, and attempt to meet each shift requirement. A reasonable example is to process shifts chronologically and prefer available employees with the fewest assignments so far, but another transparent heuristic is acceptable.
- **M02 (2 PCP)** Verify the final assignment and report required staff, assigned staff, and any understaffing for every shift.
- **S01 (1 PCP)** Compute the number of assigned shifts per employee and summarize workload distribution.
- **S02 (1 PCP)** Produce a compact summary of uncovered requirements; in the core, define numerical severity as the number of missing workers (`required_workers - assigned_workers`, with a minimum of 0).

Full core: 11 PCP (including the 4-PCP common technical core).

Extensions

- **E01 (1 PCP)** Add severity levels for understaffing alerts.
- **E02 (2 PCP)** Build an instance generator for employees, availability and shift requirements; generated instances must preserve the same schemas and allow a random seed.
- **E03 (2 PCP)** Add employee shift preferences and incorporate them into the assignment heuristic. Define and include a small preference input file (or a documented preference-score field added to a copy of the availability data).
- **E04 (2 PCP)** Visualize the weekly roster or workload distribution.
- **E05 (3 PCP)** Design a fairness-aware heuristic that explicitly reduces assignment imbalance.
- **E06 (3 PCP)** Implement two different heuristics and compare coverage and workload indicators.
- **E07 (3 PCP)** Use employee roles and `shift_role_requirements.csv`, ensuring both coverage and role requirements are checked.

Suggested class (optional): `Employee or Shift`.

Core programming focus: allocation heuristics, feasibility checks, coverage, and fairness.

Project 9 – Hotel Booking & Occupancy Manager

Description

A hotel has a fixed set of rooms and receives booking requests for date intervals and requested room types. Active booking requests must be assigned to actual rooms without overlapping stays, while cancelled requests do not occupy capacity. When no compatible room is available, the application must reject the request and report the conflict. The same assignments can then be used to calculate occupancy and simple revenue indicators.

Input data

The group will receive two synthetic CSV datasets: a hotel room inventory, including room types, capacity and nightly prices, and a history of booking requests with stay dates, guest counts and statuses. Some requests are cancelled, while high-occupancy periods create conflicts among active bookings. This makes it necessary to work with actual room assignments rather than occupancy totals alone.

Expected result

Produce a booking-assignment result, identify rejected/conflicting active requests, report occupancy over time and estimate revenue from accepted stays.

Project-specific core tasks

- **M01 (3 PCP)** Process active (`Requested`) bookings in ascending `arrival_date` and then `booking_id`. Assign each request to a room of the requested type. A room is compatible only if capacity is sufficient and its existing accepted stays do not overlap the interval `[arrival_date, departure_date)`. Mark a request as rejected/unassigned if no compatible room exists. Ignore `Cancelled` bookings for occupancy.
- **M02 (2 PCP)** Compute occupancy and revenue for accepted bookings. In the core, revenue is `base_price` per occupied night for the assigned room, and occupancy rate for a selected period is occupied room-nights divided by available room-nights in that period.
- **S01 (1 PCP)** Compare occupancy/revenue by room type or selected period.
- **S02 (1 PCP)** Produce an availability summary for a user-selected date or date interval.

Full core: 11 PCP (including the 4-PCP common technical core).

Extensions

- **E01 (1 PCP)** Add high-occupancy or low-availability alerts.
- **E02 (2 PCP)** Build an instance generator for rooms and booking requests, including configurable occupancy pressure and random seed.
- **E03 (2 PCP)** Create a calendar-style textual or graphical representation of room assignments.
- **E04 (2 PCP)** Add a Pandas occupancy/revenue analysis.
- **E05 (3 PCP)** Implement an alternative room-assignment policy that can use compatible upgrades/alternative room types and compare it with the core policy.
- **E06 (3 PCP)** Compare pricing or occupancy scenarios using a transparent rule.

Suggested class (optional): `Booking or Room`.

Core programming focus: interval conflicts, state consistency, occupancy, and rule-based assignment.

Project 10 – Restaurant Orders & Menu Performance Analyzer

Description

A restaurant stores its menu separately from the order history. Management wants to combine the two datasets to understand what sells, how much revenue and gross margin each item generates, and which categories perform best or worst. The project requires joining records by item ID, aggregating transactions, and producing operational rankings rather than simply printing the source files.

Input data

The group will receive synthetic CSV datasets containing a restaurant menu and a history of customer order lines. The menu provides product categories, prices and estimated costs; the order records identify items, quantities and transaction times. The data include lunch and dinner patterns as well as menu items with different levels of popularity and profitability.

Expected result

Produce a menu-performance report containing sales volume, revenue, cost/margin, category summaries and rankings, plus the required common report/representation.

Project-specific core tasks

- **M01 (2 PCP)** Match every valid order line to the menu using `item_id`, report invalid references if present, and accumulate quantities correctly.
- **M02 (2 PCP)** Compute units sold, revenue, estimated cost, and gross margin by menu item and by category.
- **M03 (1 PCP)** Rank the best- and worst-performing menu items according to at least two documented criteria, for example quantity and gross margin.
- **S01 (1 PCP)** Analyze how demand changes by time of day, date, or another meaningful period derived from the order timestamps.

Full core: 10 PCP (including the 4-PCP common technical core).

Extensions

- **E01 (1 PCP)** Add configurable alerts for low-margin or low-selling items.
- **E02 (2 PCP)** Build an order generator based on the provided menu, with configurable number of orders, time-of-day profiles and random seed.
- **E03 (2 PCP)** Add a more detailed day-of-week/time-of-day analysis.
- **E04 (2 PCP)** Create a Pandas/Matplotlib dashboard-style summary.
- **E05 (3 PCP)** Evaluate hypothetical price changes while keeping observed quantities fixed and compare profitability.
- **E06 (3 PCP)** Produce transparent rule-based recommendations for promotion, review, or removal of menu items using multiple indicators.

Suggested class (optional): `MenuItem or Order`.

Core programming focus: joining datasets, aggregation, ranking, time patterns, and profitability.

Project 11 – Event & Ticket Management System

Description

An event organizer sells tickets for several events with fixed capacities. Ticket transactions arrive sequentially and must never make attendance exceed capacity. The application must update remaining capacity, reject transactions that would oversell the event, calculate attendance/revenue indicators and highlight events that are nearly sold out or substantially under-used.

Input data

The group will receive two synthetic CSV datasets describing events with limited capacity and ticket transactions recorded over time. The sales include ticket categories, quantities and prices. The reference instance contains events approaching sell-out and requests that would exceed available capacity, so chronological processing and rejected sales are important.

Expected result

Produce an event status report with accepted/rejected ticket quantities, remaining capacity, attendance/utilization, revenue, and explicit capacity alerts.

Project-specific core tasks

- **M01 (2 PCP)** Process ticket transactions in ascending `sale_date` and then `transaction_id`, updating the remaining capacity for the corresponding event after every accepted transaction.
- **M02 (1 PCP)** Prevent overselling: reject a transaction when the full requested quantity would exceed remaining capacity, and record the reason.
- **M03 (1 PCP)** Compute accepted attendance and revenue by event.
- **S01 (1 PCP)** Summarize sales by ticket type.
- **S02 (1 PCP)** Compute event utilization as accepted ticket quantity divided by event capacity, and identify the most/least utilized events.

Full core: 10 PCP (including the 4-PCP common technical core).

Extensions

- **E01 (1 PCP)** Add configurable near-sellout alerts.
- **E02 (2 PCP)** Build a ticket-sales instance generator with configurable event demand profiles and random seed.
- **E03 (2 PCP)** Analyze ticket sales over time before each event.
- **E04 (2 PCP)** Support cancellations/refunds for previously accepted transactions and update capacity/revenue consistently. Define and include a small additional refund/cancellation input file used to demonstrate the feature.
- **E05 (3 PCP)** Implement a waiting-list allocation policy for rejected demand.
- **E06 (3 PCP)** Compare alternative ticket-pricing scenarios using a transparent rule.

Suggested class (optional): `Event or Ticket`.

Core programming focus: state updates, capacity constraints, transaction processing, and allocation.

Project 12 – Warehouse & Order Fulfilment System

Description

A warehouse receives customer order lines while holding limited stock for each product. Some products have insufficient inventory to satisfy all demand. The application must process orders, allocate available stock and classify each request as fully fulfilled, partially fulfilled or backordered. The basic policy is chronological; alternative priority/fairness rules belong to extensions.

Input data

The group will receive two synthetic CSV datasets describing a warehouse's initial product stock and a chronological stream of customer orders. The orders have quantities and priorities, but the basic allocation follows their arrival sequence. For several products, total requested quantities deliberately exceed stock, producing partial fulfilment and unfulfilled demand.

Expected result

Produce a fulfilment result for every order line, final inventory, shortage information and service-level indicators.

Project-specific core tasks

- **M01 (3 PCP)** Process orders by `order_time` (then `order_id` for ties). Fulfil `min(requested_quantity, available_quantity)`, update stock, and never allow negative inventory.
- **M02 (2 PCP)** Classify each line as fully fulfilled, partially fulfilled, or backordered/unfulfilled and record requested, fulfilled and unfulfilled quantities.
- **S01 (1 PCP)** Compute service-level indicators such as percentage of fully satisfied orders and percentage of requested units fulfilled.
- **S02 (1 PCP)** Summarize shortages by product and identify the products with the largest unmet demand.

Full core: 11 PCP (including the 4-PCP common technical core).

Extensions

- **E01 (1 PCP)** Create a detailed shortage report grouped by product/customer.
- **E02 (2 PCP)** Build an instance generator for inventory and orders with configurable demand pressure and random seed.
- **E03 (2 PCP)** Add a Pandas fulfilment analysis.
- **E04 (3 PCP)** Compare chronological allocation with a priority-based allocation policy.
- **E05 (3 PCP)** Design a fairness-aware policy for scarce stock and compare customer-level outcomes.
- **E06 (3 PCP)** Simulate multiple replenishment/order periods and carry stock forward between periods.

Suggested class (optional): `Product or Order`.

Core programming focus: allocation, state updates, shortages, and service-level analysis.

Project 13 – Small Business KPI Dashboard

Description

A small online retailer wants a concise management dashboard built from transaction-level sales data. The application should transform individual invoice lines into understandable business indicators, compare periods or market segments, identify unusually weak/strong areas and export a structured report. This project is intentionally more data-analysis oriented than the inventory/fulfilment projects, but fundamental Python logic must still be visible in at least one substantial component.

Input data

The group will receive a course-adapted CSV dataset of online retail transactions derived from the public UCI *Online Retail II* dataset. Records describe invoice lines, purchased products, quantities, prices, customers, dates and countries. The teaching sample is cleaned and balanced across calendar months to support meaningful period comparisons; invalid records and cancellations have been excluded. Source: D. Chen, *Online Retail II*, UCI Machine Learning Repository, DOI 10.24432/C5CG6D (CC BY 4.0): https://archive.ics.uci.edu/dataset/502/online+retail+ii.

Expected result

Produce a dashboard-style report with clearly defined KPIs, comparisons and threshold alerts, plus at least one concise visual or tabular representation suitable for management use.

Project-specific core tasks

- **M01 (2 PCP)** Compute a documented KPI set including at least total revenue, units sold, number of invoices/orders, number of customers, and average order value. Revenue per line is `quantity * unit_price`.
- **M02 (2 PCP)** Aggregate and compare the KPIs across at least two useful dimensions, for example month, country, customer, or product.
- **M03 (1 PCP)** Implement configurable or documented KPI thresholds/targets and flag periods or segments that violate them.
- **S01 (1 PCP)** Compare a selected period/segment with another period/segment or with a target and report absolute and percentage differences.

Full core: 10 PCP (including the 4-PCP common technical core).

Extensions

- **E01 (1 PCP)** Make KPI thresholds/targets configurable from a file or command line.
- **E02 (2 PCP)** Generate a richer report file containing ranked tables and multiple sections.
- **E03 (2 PCP)** Create a multi-chart Matplotlib dashboard.
- **E04 (2 PCP)** Add Pandas drill-down analysis across multiple dimensions.
- **E05 (3 PCP)** Construct a transparent composite performance score from several KPIs and explain the weights.
- **E06 (3 PCP)** Perform sensitivity analysis on the weights or thresholds used in the dashboard.

Core programming focus: aggregation, KPI design, threshold rules, reporting, and responsible use of data-analysis libraries.

Project 14 – Energy Consumption & Cost Analyzer

Description

A household or small organization wants to understand when energy is consumed and how a time-of-use tariff affects cost. The course data are derived from real household electricity measurements but are already aggregated to hourly resolution, so the student task is analysis rather than large-scale cleaning. The tariff file is an instructor-designed teaching tariff, not an actual commercial contract.

Input data

The group will receive an hourly electricity-consumption CSV dataset adapted from real household measurements, together with a separate synthetic time-of-use tariff table. The consumption data cover 2007 and include total energy and sub-metering information; severely incomplete source hours were removed during preparation. Tariff values are teaching assumptions, not a commercial offer. Source: G. Hébrail and A. Bérard, *Individual Household Electric Power Consumption*, UCI Machine Learning Repository, DOI 10.24432/C58K54 (CC BY 4.0): https://archive.ics.uci.edu/dataset/235/individual+household+electric+power+consumption.

Expected result

Produce consumption and cost summaries, identify peak/high-consumption periods, and provide an understandable temporal representation.

Project-specific core tasks

- **M01 (2 PCP)** Aggregate hourly energy into useful daily/monthly summaries and compute total, average, minimum and maximum consumption for selected periods.
- **M02 (2 PCP)** Apply the tariff schedule according to the hour of each observation and calculate energy cost by hour, period, and tariff band.
- **M03 (1 PCP)** Identify peak-consumption observations/periods or threshold violations using a documented rule.
- **S01 (1 PCP)** Compare two periods, such as months, weekdays/weekends, or selected date ranges.

Full core: 10 PCP (including the 4-PCP common technical core).

Extensions

- **E01 (1 PCP)** Add configurable peak-consumption alerts.
- **E02 (2 PCP)** Compare the provided tariff with an alternative tariff supplied in a second file using the same tariff schema; include the alternative file in the submission.
- **E03 (2 PCP)** Add Pandas-based time-series aggregation.
- **E04 (2 PCP)** Visualize consumption and/or cost over time with Matplotlib.
- **E05 (3 PCP)** Detect unusual consumption using a transparent historical rule.
- **E06 (3 PCP)** Simulate a simple load-shifting scenario and estimate the potential cost change while preserving total shifted energy.

Core programming focus: time-series aggregation, tariff rules, peak detection, and scenario analysis.

Project 15 – Urban Mobility Demand Analyzer

Description

A mobility operator wants to understand when a bike-sharing system experiences the highest demand and how demand differs across calendar and weather conditions. The project uses real hourly mobility data but a course-adapted schema with readable labels. The aim is descriptive analysis, comparison and reporting rather than prediction or machine learning.

Input data

The group will receive a CSV dataset adapted from real bike-sharing observations, with hourly rental demand and contextual information on working days, seasons and weather conditions. Casual and registered users can be analyzed separately. For readability, normalized temperature and humidity measurements have been converted into Celsius and percentages. Source: H. Fanaee-T, *Bike Sharing*, UCI Machine Learning Repository, DOI 10.24432/C5W894 (CC BY 4.0): `https://archive.ics.uci.edu/dataset/275/bike+sharing+dataset`.

Expected result

Produce demand indicators, identify peak periods, compare selected conditions and show at least one clear temporal/category representation of mobility demand.

Project-specific core tasks

- **M01 (2 PCP)** Compute demand indicators by hour and by at least one broader period such as day, month, or season.
- **M02 (2 PCP)** Compare demand across meaningful conditions and separately analyze casual and registered users.
- **M03 (1 PCP)** Identify peak-demand periods and/or periods exceeding a documented demand threshold.
- **S01 (1 PCP)** Produce one focused comparison, for example working vs non-working days, seasons, or weather categories, with absolute and relative differences.

Full core: 10 PCP (including the 4-PCP common technical core).

Extensions

- **E01 (1 PCP)** Make high-demand alerts configurable.
- **E02 (2 PCP)** Create time-of-day or seasonal Matplotlib visualizations.
- **E03 (2 PCP)** Add Pandas multi-factor demand analysis.
- **E04 (2 PCP)** Build a structured comparison of weekday/weekend and casual/registered user profiles.
- **E05 (3 PCP)** Simulate additional demand under a transparent scenario rule and compare it with a user-specified hourly capacity threshold; report the hours in which the simulated demand exceeds that threshold.
- **E06 (3 PCP)** Construct and compare a simple service-pressure indicator combining demand with selected conditions.

Core programming focus: time-based aggregation, classification, comparison, ranking, and mobility-demand indicators.

Project 16 – Service Desk Queue Simulator

Description

A service desk receives customers over time. Each customer has an arrival time and a required service time; if the server is busy, the customer waits in a first-come-first-served queue. The core project implements the queue explicitly rather than using a ready-made simulation package. Optional extensions introduce additional servers, priorities and stochastic generation.

Input data

The group will receive synthetic CSV datasets describing customers arriving at a service desk and several possible simulation settings. Customer records include arrival times and required service durations. The basic reference setting has a single server and first-come-first-served processing; the additional settings and customer categories support extensions such as multiple servers and priority rules.

Expected result

Produce one row or structured record per customer containing arrival, service start, departure and waiting time, together with queue performance indicators and congestion information.

Project-specific core tasks

- **M01 (3 PCP)** Implement a single-server FCFS simulation. Process customers by `arrival_time` and then `customer_id` for ties, compute service start as the later of arrival and server availability, compute departure, update server availability, and derive each customer waiting time. For the core, queue length means the number of arrived customers waiting for service, excluding the customer currently in service.
- **M02 (2 PCP)** Compute at least average waiting time, maximum waiting time, proportion of customers who waited, and maximum queue length.
- **S01 (1 PCP)** Analyze the waiting-time distribution with suitable bins or summary statistics.
- **S02 (1 PCP)** Identify when the maximum queue occurs or otherwise characterize the main congestion period.

Full core: 11 PCP (including the 4-PCP common technical core).

Extensions

- **E01 (1 PCP)** Add waiting-time threshold alerts.
- **E02 (2 PCP)** Generate random arrivals and service times with configurable parameters and random seed, saving the generated customers in the same CSV schema.
- **E03 (2 PCP)** Extend the simulator to multiple parallel servers and use the additional scenarios.
- **E04 (2 PCP)** Visualize waiting-time or queue-length distributions.
- **E05 (3 PCP)** Run repeated Monte Carlo simulations and report distributions of performance measures.
- **E06 (3 PCP)** Add priority customers with a documented non-preemptive priority rule.
- **E07 (3 PCP)** Compare two service configurations using the same customer data.
- **E08 (3 PCP)** Compare waiting-time outcomes across customer groups using a fairness indicator.

Suggested class (optional): `Customer or Server`.

Core programming focus: explicit simulation, queue state, event processing, and performance analysis.

Project 17 – Supply & Demand Market Simulator

Description

A stylized market model is used to study how a price changes when supply and demand are not balanced. This is a programming model, not a calibrated economic forecast. At each iteration, demand decreases with price, supply increases with price, and the price moves in the direction of excess demand. The application must expose the whole trajectory and its stopping condition.

Input data

The group will receive a synthetic CSV dataset of stylized market scenarios, each specifying starting conditions and parameters controlling how supply, demand and prices respond over time. Some parameter combinations approach an equilibrium, while others may fail to converge within the simulation horizon. These are teaching scenarios for studying iterative adjustment, not estimates derived from a real market.

Expected result

Produce the iteration history (price, supply, demand, gap) and a convergence summary stating whether equilibrium tolerance was reached, after how many iterations, and at what approximate price.

Project-specific core tasks

- **M01 (3 PCP)** Implement the iterative update rule exactly as specified and record the market state at every iteration.
- **M02 (1 PCP)** Apply both stopping conditions correctly and distinguish converged from non-converged scenarios.
- **M03 (1 PCP)** Report final price, final supply-demand gap, number of iterations, and convergence status.
- **S01 (1 PCP)** Compare at least two scenarios and discuss how starting values/parameters affect the trajectory.

Full core: 10 PCP (including the 4-PCP common technical core).

Extensions

- **E01 (1 PCP)** Add a more detailed convergence warning/report, for example detecting slow or oscillatory behaviour using a documented rule.
- **E02 (2 PCP)** Introduce documented random supply/demand shocks and allow a random seed.
- **E03 (2 PCP)** Build a scenario generator that creates valid parameter combinations within configurable ranges and supports a user-specified random seed.
- **E04 (2 PCP)** Plot price and/or supply-demand gap across iterations.
- **E05 (3 PCP)** Implement and compare an alternative price-adjustment rule.
- **E06 (3 PCP)** Run repeated stochastic simulations and analyze the distribution of final prices or convergence times.

Suggested class (optional): `MarketScenario`.

Core programming focus: iterative simulation, stopping conditions, convergence, and scenario analysis.

Project 18 – Media Collection & Recommendation Tool

Description

A user wants to explore a movie catalogue and rating history, search for relevant titles, understand genre/rating patterns, and optionally obtain simple recommendations. The project uses the real MovieLens Latest Small dataset. Recommendation logic must remain transparent and understandable; machine-learning recommenders are not required.

Input data

The group will receive two CSV datasets from *MovieLens Latest Small*: a movie catalogue with titles and genres, and anonymized user ratings collected over time. The data support searches, descriptive analyses and interpretable recommendations; sparsely rated movies require care when constructing rankings. Original source and usage conditions: GroupLens Research, University of Minnesota, *MovieLens Latest Small*: `https://grouplens.org/datasets/movielens/latest/`.

Expected result

Produce searchable/filterable catalogue results, descriptive statistics, robust rankings and, when selected, a documented recommendation output.

Project-specific core tasks

- **M01 (2 PCP)** Search/filter the catalogue using at least two criteria, such as case-insensitive title substring, genre, and/or extracted release year.
- **M02 (2 PCP)** Combine movies and ratings to compute useful statistics such as rating count and average rating by movie and genre.
- **M03 (1 PCP)** Generate at least one robust ranking. To avoid ranking a movie from one isolated rating, use and document a minimum rating-count threshold.
- **S01 (1 PCP)** Implement a simple transparent non-ML recommendation rule, for example suggesting highly rated movies in user-selected genres that satisfy a minimum rating count.

Full core: 10 PCP (including the 4-PCP common technical core).

Extensions

- **E01 (1 PCP)** Add a rating or genre histogram.
- **E02 (2 PCP)** Build a user-specific preference profile from the user's highly rated movies and use it to produce personalized suggestions for unseen movies.
- **E03 (2 PCP)** Add Pandas-based catalogue/rating exploration across multiple genres or time periods.
- **E04 (2 PCP)** Extract release years from titles and analyze rating/popularity trends over time.
- **E05 (3 PCP)** Implement and compare two transparent recommendation rules, such as popularity-based and genre-profile-based.
- **E06 (3 PCP)** Implement a similarity-based recommendation rule using manually defined interpretable features rather than a machine-learning library.

Suggested class (optional): `MediaItem`.

Core programming focus: search, filtering, ranking, joining datasets, and transparent recommendation rules.

Project 19 – Population / Epidemic Dynamics Simulator

Description

A simplified three-state population model is used to practise discrete-time simulation. The provided scenarios use susceptible, infected and recovered states. The core model is a deterministic teaching version of an SIR-style process and is not intended for real epidemiological forecasting or parameter estimation. Students must implement the state transitions explicitly and verify that the population remains consistent.

Input data

The group will receive a synthetic CSV dataset of population scenarios containing initial numbers of susceptible, infected and recovered individuals, along with parameters governing infection and recovery over a simulated period. The scenarios are designed for a simplified educational model and are not epidemiological observations or forecasts. They allow changes in parameters and population trajectories to be compared.

Expected result

Produce a day-by-day state trajectory, consistency checks, peak/final indicators and scenario comparisons.

Project-specific core tasks

- **M01 (3 PCP)** Simulate the deterministic S-I-R state transitions for the requested number of days using the specified equations and record each state over time.
- **M02 (1 PCP)** Check after every update that states are non-negative within numerical tolerance and that `S + I + R` remains equal to the total population within floating-point tolerance.
- **M03 (1 PCP)** Report peak infected value/day and final state proportions.
- **S01 (1 PCP)** Compare at least two parameter scenarios using the same indicators.

Full core: 10 PCP (including the 4-PCP common technical core).

Extensions

- **E01 (1 PCP)** Add alerts when the infected state exceeds a configurable threshold.
- **E02 (2 PCP)** Implement a stochastic transition version using a documented random rule and allow a random seed.
- **E03 (2 PCP)** Build a scenario generator that creates valid initial states and parameters within configurable ranges and supports a user-specified random seed.
- **E04 (2 PCP)** Plot the state trajectories with Matplotlib.
- **E05 (3 PCP)** Model an intervention that changes one or more parameters after a selected day and compare trajectories.
- **E06 (3 PCP)** Run repeated stochastic simulations and report the distribution of peak or final outcomes.

Suggested class (optional): `PopulationState or Scenario`.

Core programming focus: state-transition simulation, consistency checks, peaks, and scenario comparison.

Project 20 – Tournament / Game Simulation Manager

Description

A league-style tournament contains teams and completed match results. The organizer wants a program that reconstructs the standings, applies ranking and tie-breaking rules, and summarizes team performance. The provided data use scores, so the core adopts a football-style points system; more elaborate tournament generation and repeated simulations are optional extensions.

Input data

The group will receive two synthetic CSV datasets: a list of tournament teams and a record of completed matches, including dates and scores. Teams vary in performance, and some rankings require explicit tie-breaking. The basic project reconstructs a league table from observed match results; more elaborate schedule generation and tournament simulations are optional.

Expected result

Produce a complete league table and team statistics, including the ranking order and enough information to explain tie-breaking.

Project-specific core tasks

- **M01 (2 PCP)** Process every match and compute, for each team, played, wins, draws, losses, goals/points scored, goals/points conceded, and league points.
- **M02 (2 PCP)** Sort teams into a final/current ranking according to league points and the documented tie-breaking sequence.
- **M03 (1 PCP)** Implement at least one explicit tie-breaker; the recommended sequence is goal difference followed by goals scored and then team name/ID for deterministic output.
- **S01 (1 PCP)** Produce additional performance statistics such as win rate, average goals scored, form over recent matches, or best/worst streaks.

Full core: 10 PCP (including the 4-PCP common technical core).

Extensions

- **E01 (1 PCP)** Add a textual histogram or compact visual representation of team performance.
- **E02 (2 PCP)** Build a match/tournament generator with configurable participants, number of fixtures and random seed.
- **E03 (2 PCP)** Add a knockout stage after the league/round-robin phase.
- **E04 (2 PCP)** Create Matplotlib performance visualizations.
- **E05 (3 PCP)** Run Monte Carlo tournament simulations and estimate winning/top-position probabilities.
- **E06 (3 PCP)** Compare two points/tie-breaking systems and quantify ranking changes.
- **E07 (3 PCP)** Generate a fair schedule using a transparent heuristic and verify simple fairness criteria.

Suggested class (optional): `Player, Team, or Match`.

Core programming focus: state updates, ranking algorithms, tie-breaking, and simulation.

Alternative project proposals

Students may propose a different project if it is comparable in scope and uses the programming concepts of the course. The proposal must identify the input data, mandatory core, secondary core and extensions so that an equivalent PCP configuration can be agreed before implementation. Approval is required before starting a project that is not in this catalogue.