Applying evolutional algorithms to solve problems:

optimize_table.py

Clara: Okay, Karl, the guest list is final. Ten people, three tables. All set. Easy, right?

Karl: You say "easy" as if this isn't a diplomatic minefield. Remember last year?

Clara: How could I forget? Aunt Zia sat next to Cousin Ben, couldn't hear a word he said for the whole dinner—he's got the voice of a timid mouse.

Karl: And then there was the *ex-wife meets current wife* incident. Linda and Karen at the same table? I'm still recovering from the tension.

Clara: And don't even think about putting Aunt Sara next to a man. She'll glare all evening and then write us a passive-aggressive thank-you note.

Karl: We also need to corral the kids—Mia, Tom, and Zoe—preferably all at one table. But not without an adult to keep the spaghetti off the ceiling.

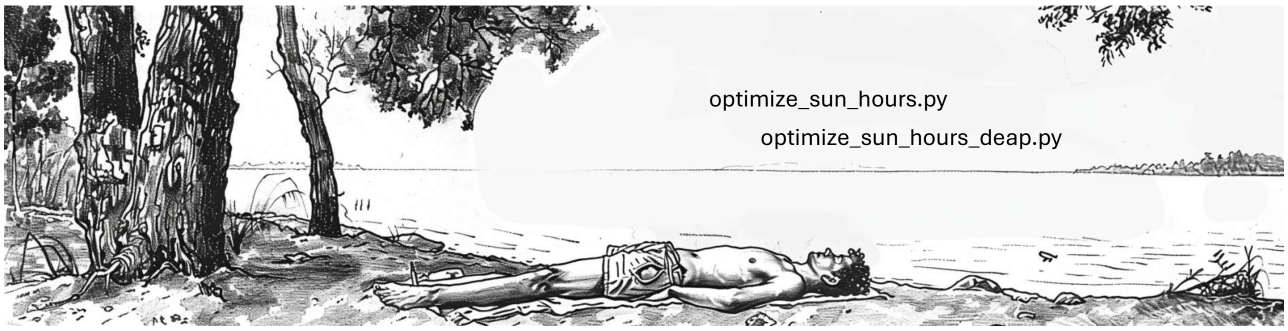
Clara: So... we need to seat everyone, respect fragile egos, prevent accidental skirmishes, and keep the children from staging a coup.

Karl: In short, we're planning the G20 Summit. But with cake.

- Which interpersonal or physical traits lead to constraint violations in the script, and how are these checked in the code?
  (Hint: e.g., "deaf_left", "loud_voice", "no_men_nearby", etc.)
- Why is randomly assigning guests to tables not sufficient, and how does the evolutionary algorithm help optimize the seating arrangement?
- How does the code ensure that children are seated together but with at least one adult at their table, and why is this an important condition according to the teaser dialogue?
- How does the concept of "fitness" in an evolutionary algorithm translate to the seating arrangement problem, and what does it aim to minimize?

Exploring optimize_table_deap.py
This version uses the **DEAP** library (Distributed Evolutionary Algorithms in Python).

- What are the main benefits of using DEAP over hand-written evolution?
  ☐ Simpler code ☐ More control ☐ Visual output ☐ All of the above
- What parts of the evolution process are handled by DEAP?
  ☐ Population creation
  ☐ Selection
  ☐ Mutation / Crossover
  ☐ Evaluation
- (Mark all that apply.)


optimize_sun_hours.py

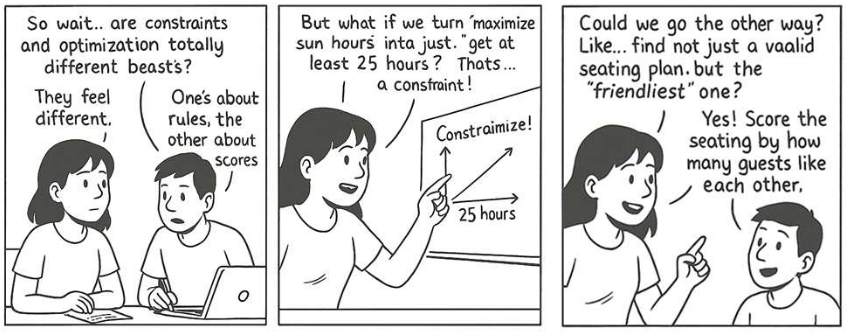
optimize_sun_hours_deap.py

Tim had one goal for his summer vacation: lie in the sun like a lazy lizard. Wake up, stroll to the lake, plant his towel, and soak up rays for eight blissful hours — minimal movement, maximum tan.
Then came the phone call.
“Timmy,” said his mom, “I booked us a day to go couch shopping, remember? Also, Aunt Zia is baking your favorite cake, so you *have* to stop by. Oh — and guess who wants to see you again? Anna from third grade! She has brunch every morning *after* the kids leave for school. And the attic really needs sorting… you said you’d help.”
Suddenly, Tim’s beach dream looked more like a bureaucratic nightmare. Can he still squeeze in some quality sunshine — between social duties, family nostalgia, and flat-pack furniture?
Only a clever plan will save his tan.

- How does the algorithm balance between time-consuming activities and the goal of maximizing Tim’s sun hours?
- Why is travel time modeled explicitly, and how does the location sequence (Strandbad → activities → Strandbad) affect the outcome?
- What role does the Boolean representation of an individual ([True, False, ...]) play in modeling a weekly schedule for Tim?
- What are the main differences in how population, mutation, and crossover are handled in the DEAP version compared to the manual version?
  (Hint: Look for toolbox.register() vs. custom mutate() and crossover() functions.)
- How does DEAP’s use of creator and toolbox simplify or complicate the process compared to writing everything from scratch?

Let’s compare the two kinds of Problems:

Feature	**Optimization Problem**	**Constraint Satisfaction Problem**
Objective	Maximize/minimize a numerical value	Satisfy all given constraints
Evaluation	Fitness score is continuous or scalar	Violations counted (want 0 violations)
Typical Solution	“Best” solution within constraints	“Feasible” solution that fulfills all rules
Example	Maximize sun hours during vacation	Arrange guests at tables without conflict
Handling infeasibility	May penalize constraint violation	Any violation makes a solution invalid or less desirable


So wait.. are constraints and optimization totally different beast's?
They feel different.
One's about rules, the other about scores
But what if we turn 'maximize sun hours' inta just. "get at least 25 hours? Thats... a constraint!
Constraimize!
25 hours
Could we go the other way? Like... find not just a vaalid seating plan. but the "friendliest" one?
Yes! Score the seating by how many guests like each other.

Optimization or Constraints Problem?

#	Task Description	Your Answer
1	Assign students to dorm rooms so **no enemies share a room**	
2	Find a diet plan with **maximum protein under 2000 calories**	
3	Assign people to project teams **so each team has at least one coder**	
4	Choose a set of lectures to attend that **fit within your schedule**	
5	Order deliveries to **minimize total driving time**	
6	Place exhibits in a museum **so loud ones aren’t next to quiet ones**	
7	Allocate ads in a layout to **maximize click-through rate**	
8	Organize a music festival so that **bands with rival members don’t overlap**	
9	Pick a study schedule that **balances workload and maximizes free evenings**	
10	Match mentors to mentees so that **no one mentors someone from their team**	

Lisa: So, here's my plan: collect as many ECTS as possible, get solid grades, and ideally… do as little work as humanly possible.

Max (grinning): Ah yes, quite a new strategy! How's that working out?

Lisa: Well, "Statistics for the Humanities" gives you 10 ECTS – but the syllabus is just walls of formulas and horrible proofs. You start dreaming of formulas trying to devour you.

Max: Nice. What about something easier?

Lisa: "Editorial Theory." Tons of reading, but the prof gives generous grades if you show up and say things like "materiality of the text."
Only 5 ECTS though, and your eyes melt after page 300.

Max: So you want max credits, minimum sweat, and a Degree your mom can put on the fridge?

Lisa: Exactly. It's basically a multi-objective optimization problem. And if I had done the evolutionary algorithm course last year, I knew how to solve it.

Multi-Objective Evolutionary Algorithm (MOEA)

- A **MOEA** is an algorithm that solves **optimization problems with more than one goal** (objective).
- It finds a set of **trade-off solutions** instead of a single "best" answer.
- These solutions form a **Pareto front** — where no solution is strictly better than another in all objectives.
- MOEAs are based on **evolutionary principles** like selection, crossover, and mutation.

Here are some use cases. Do you have more ideas, where MOEA might come in handy?

- **Product design** – balance cost, weight, and performance.
- **Layout design** - clarity vs. information density
- **Finance** – optimize return vs. risk in a portfolio.
- **Machine learning** – tune models for accuracy vs. complexity.
- **LLM-Tasks** – minimize time, reduce hallucinations, improve reliability
- **Scheduling** – minimize time and cost at the same time.