Lesson 5 – Combining Conditions, Loops, and Functions

Why combine everything?

We have three superpowers now:

- **Conditions** choose actions based on rules (*constraints*).
- **Loops** explore many options.
- **Functions** apply the same formula reliably to every option.

Optimization loop: Constraint → Evaluate → Keep Best

Example 1: Cheapest Shopping Within a Budget

A shopper has a list of items with different prices and a limited amount of money to spend. The goal is to find the most affordable item that still fits within the budget. This helps make smart purchasing decisions when money is tight.

- **Variables:** x_i = price of item i
- **Objective:** find the smallest x_i such that $x_i \leq$ budget
- **Constraint:** only choose items within the available budget

Budget tip: pick the lowest price that is still feasible.

```
prices = [12.5, 7.0, 9.5, 5.0, 14.0]
budget = 10.0

best_price = float('inf') # start with an impossibly large value
best_idx = -1

for i, p in enumerate(prices):
    if p <= budget and p < best_price: # feasible AND cheaper
        best_price = p
        best_idx = i

if best_idx == -1:
    print("Nothing affordable.")
else:
```

```
    print("Cheapest within budget is item", best_idx, "at", best_price)
```

Output

```
Cheapest within budget is item 3 at 5.0
```

Optimization Connection

This is a *filter + min* pattern: first enforce feasibility (**constraint**), then keep the best value (**objective**).

Example 2: Best route with time and transfers

A commuter is choosing how to get across town. There are several possible routes (A, B, C, D), and each one takes a different amount of time and may require changing buses or trains. Transfers are annoying and take extra effort, so we want a route that is not only fast but also has as few transfers as possible.
Suppose each route has `(time_min, transfers)`. We penalize transfers:

$$\text{score} = -\text{time} \; - \; 5 \times \text{transfers}$$

Higher score is better (less time, fewer transfers).

```
routes = [("A", 18, 0), ("B", 14, 1), ("C", 12, 2), ("D", 16, 0)]

def route_score(time_min, transfers):
    return -time_min - 5*transfers

best_name = None
best_score = -float('inf')

for name, t, k in routes:
    s = route_score(t, k)
    print("Route", name, "has score", s)
    if s > best_score:
        best_score = s
        best_name = name

print("Best route is", best_name, "with score", best_score)
```

Output

```
Route A has score -18
Route B has score -19
```

```
Route C has score -22
Route D has score -16
Best route is D with score -16
```

Optimization Connection

We combined a **function** (the scoring rule), a **loop** (try all routes), and a **comparison** (keep best-so-far). This mirrors multi-criteria optimization.

Example 3: Packing snacks (score = fun $-0.5\times$ weight)

Imagine you are packing snacks for a day trip. Each snack has a "fun" rating (how tasty or exciting it is) and a weight. You want a snack that is fun, but not too heavy to carry. We use a score that rewards fun and penalizes weight.
We choose the single snack with the best tradeoff score.

Tradeoff: higher fun vs. heavier to carry.

```
snacks = [
    ("chips", 7, 2.0), # (name, fun, weight)
    ("apple", 5, 0.4),
    ("cookie", 6, 0.8),
    ("soda", 4, 1.0),
]

def snack_score(fun, weight):
    return fun - 0.5*weight

best = None
best_s = -float('inf')

for name, fun, wt in snacks:
    s = snack_score(fun, wt)
    print(name, "score =", s)
    if s > best_s:
        best_s = s
        best = name

print("Pick:", best, "with score", best_s)
```

Output

```
chips score = 6.0 apple score = 4.8 cookie score = 5.6 soda score = 3.5
Pick: chips with score 6.0
```

Try a variation

Change the penalty to 1.0 × weight or 0.25 × weight and see which snack wins. Tuning the score changes the optimal choice.

Lesson 5 Activities

Practice

1. **Budget picker:** With `prices = [11, 4, 8, 3, 12]` and `budget = 9`, print the *cheapest* item within budget and its index. If none are feasible, say so.
2. **Two-attribute route:** Routes are `(name, time, transfers)` as in Example 2. Write `route_score(t, k) = -t - 4*k`. Loop to print each score and the best route.
3. **Snack tradeoff:** With `snacks = [(name, fun, weight), ...]`, write `snack_score(fun, w) = fun - 0.75*w`. Print the best snack.
4. **Filter then best:** Classroom chores `[("trash", 3), ("wipe", 2), ("sweep", 5), ("boards", 1)]` where the number is minutes. You only have `time_limit = 3`. Print the most time-consuming chore you can still finish.
5. **Challenge: General selector.** Write `best_choice(options, score_fn)` that returns the element in `options` with maximum `score_fn(item)`. Test on routes (time/transfers) and on snacks (price/fun) with different scoring rules.