FOUNDATION LESSON

WEEK 3 · LESSON 6

Check reusable KPI calculations

assert, edge cases and a small KPI library

UNDERSTAND THE IDEA FIRST

MENTAL MODEL

Understand the idea first

Think of validation as the gate that checks incoming values, and a development assertion as a check on the result of your own reasoning. assert condition raises AssertionError if the condition is false during ordinary execution. Python can remove assertions under optimisation. Therefore, required input validation must use ordinary executable checks and explicit errors.

A PASSING DEVELOPMENT CHECK

CODE WALKTHROUGH W03-L06-S03

```
01  def profit(revenue, cost):
02      return revenue - cost
03
04  assert profit(1200, 800) == 400
05  assert profit(0, 0) == 0
06  print("Two checks passed")
```

WHAT TO TRACK

Normal case and zero boundary; assertions do not print on success.

PREDICT THE RESULT

Predict the output.
Explain each operation.
Pause before continuing.

A PASSING DEVELOPMENT CHECK — RESULT

CODE WALKTHROUGH W03-L06-S04

```
01  def profit(revenue, cost):
02      return revenue - cost
03
04  assert profit(1200, 800) == 400
05  assert profit(0, 0) == 0
06  print("Two checks passed")
```

WHAT TO TRACK

Normal case and zero boundary; assertions do not print on success.

VALIDATE THE RESULT

```
Two checks passed
```

A KPI WITH AN EMPTY-INPUT POLICY

CODE WALKTHROUGH W03-L06-S05

```
01  def average_sale(amounts):
02      if not amounts:
03          raise ValueError("at least one sale is required")
04      for amount in amounts:
05          if amount < 0:
06              raise ValueError("sales must be non-negative")
07      return sum(amounts) / len(amounts)
08
09  print(average_sale([100, 200]))
10  print(average_sale([]))
```

WHAT TO TRACK

Policy: non-negative numeric sales; empty batch has no average.

PREDICT THE RESULT

Predict the output.
Explain each operation.
Pause before continuing.

A KPI WITH AN EMPTY-INPUT POLICY — RESULT

CODE WALKTHROUGH W03-L06-S06

```
01  def average_sale(amounts):
02      if not amounts:
03          raise ValueError("at least one sale is required")
04      for amount in amounts:
05          if amount < 0:
06              raise ValueError("sales must be non-negative")
07      return sum(amounts) / len(amounts)
08
09  print(average_sale([100, 200]))
10  print(average_sale([]))
```

WHAT TO TRACK

Policy: non-negative numeric sales; empty batch has no average.

VALIDATE THE RESULT

```
150.0
ValueError: at least one
sale is required
```

A DELIBERATELY WRONG EXPECTATION

CODE WALKTHROUGH W03-L06-S07

```
01  def profit(revenue, cost):
02      return revenue - cost
03
04  assert profit(100, 40) == 70, "check the expected profit"
```

WHAT TO TRACK

Prediction: distinguish a failing test from proof the function is wrong.

PREDICT THE RESULT

Predict the output.
Explain each operation.
Pause before continuing.

A DELIBERATELY WRONG EXPECTATION — RESULT

CODE WALKTHROUGH W03-L06-S08

```
01  def profit(revenue, cost):
02      return revenue - cost
03
04  assert profit(100, 40) == 70, "check the expected profit"
```

WHAT TO TRACK

Prediction: distinguish a failing test from proof the function is wrong.

VALIDATE THE RESULT

```
AssertionError: check the expected profit
```

PAUSE AND APPLY

LEARNER PRACTICE

Pause and apply

Create a total-positive-sales function. Return zero for an empty list and include only positive numeric amounts. Check a mixed batch, an empty batch and an all-zero batch with independently calculated expectations. Keep this code together as a small KPI collection; module imports come next.

PREDICT · MODIFY · RUN

01 Restate the supplied rules.

02 Predict the result.

03 Modify and run the code.

04 Compare actual output.

05 Explain any difference.

RECAP AND NEXT STEP

Recap and next step

Check normal values, boundaries, empty input and invalid input. Assertions support development and may be disabled; required validation uses explicit checks. A failed test requires investigation of both logic and expectation. Next, we organise reusable functions into modules.

UNDERSTAND APPLY VALIDATE EXPLAIN