FOUNDATION LESSON

WEEK 3 · LESSON 4

Document a function’s contract

Docstrings, basic type hints and validation

UNDERSTAND THE IDEA FIRST

MENTAL MODEL

Understand the idea first

Think of the contract as a label on a measuring instrument: it says what goes in and what comes out. The label does not physically stop misuse. A runtime check is executable code that can reject an unsuitable value. A docstring is the first string in the function body. A type hint is an annotation describing an intended type.

A DOCUMENTED CALCULATION

CODE WALKTHROUGH W03-L04-S03

```
01  def revenue(quantity: int, price: int) -> int:
02      """Return quantity times price in whole teaching units."""
03      return quantity * price
04
05  print(revenue(3, 20))
```

WHAT TO TRACK

Parameters: quantity and price.
Result: integer revenue.

PREDICT THE RESULT

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

A DOCUMENTED CALCULATION — RESULT

CODE WALKTHROUGH W03-L04-S04

```
01  def revenue(quantity: int, price: int) -> int:
02      """Return quantity times price in whole teaching units."""
03      return quantity * price
04
05  print(revenue(3, 20))
```

WHAT TO TRACK

Parameters: quantity and price.
Result: integer revenue.

VALIDATE THE RESULT

60

A HINT DOES NOT ENFORCE THE INPUT

CODE WALKTHROUGH W03-L04-S05

```
01  def revenue(quantity: int, price: int) -> int:
02      """Return quantity times price in whole teaching units."""
03      return quantity * price
04
05  print(revenue("3", 2))
```

WHAT TO TRACK

Observe what Python actually does with a string and an integer.

PREDICT THE RESULT

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

A HINT DOES NOT ENFORCE THE INPUT — RESULT

CODE WALKTHROUGH W03-L04-S06

```
01  def revenue(quantity: int, price: int) -> int:
02      """Return quantity times price in whole teaching units."""
03      return quantity * price
04
05  print(revenue("3", 2))
```

WHAT TO TRACK

Observe what Python actually does with a string and an integer.

VALIDATE THE RESULT

33

VALIDATE A WHOLE-NUMBER QUANTITY

CODE WALKTHROUGH W03-L04-S07

```
01  def checked_revenue(quantity: int, price: int):
02      if type(quantity) is not int or type(price) is not int:
03          return None
04      if quantity < 0 or price < 0:
05          return None
06      return quantity * price
07
08  print(checked_revenue(3, 20))
09  print(checked_revenue("3", 20))
10  print(checked_revenue(0, 20))
```

WHAT TO TRACK

Policy: non-negative exact integers;
None marks invalid input.

PREDICT THE RESULT

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

VALIDATE A WHOLE-NUMBER QUANTITY — RESULT

CODE WALKTHROUGH W03-L04-S08

```
01  def checked_revenue(quantity: int, price: int):
02      if type(quantity) is not int or type(price) is not int:
03          return None
04      if quantity < 0 or price < 0:
05          return None
06      return quantity * price
07
08  print(checked_revenue(3, 20))
09  print(checked_revenue("3", 20))
10  print(checked_revenue(0, 20))
```

WHAT TO TRACK

Policy: non-negative exact integers; None marks invalid input.

VALIDATE THE RESULT

```
60
None
0
```

PAUSE AND APPLY

LEARNER PRACTICE

Pause and apply

Modify checked_revenue so a negative quantity is invalid but zero quantity is still valid. Test quantity zero, minus one and the string two. Explain why a type check comes before a numeric comparison.

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

A docstring explains the contract. Basic type hints describe intended types; they do not enforce them during ordinary execution. Runtime validation checks both type and value. In the next lesson, we make invalid inputs easier to diagnose using meaningful errors.

UNDERSTAND APPLY VALIDATE EXPLAIN