FOUNDATION LESSON

WEEK 3 · LESSON 5

Handle failures without hiding them

try/except and meaningful errors

UNDERSTAND THE IDEA FIRST

MENTAL MODEL

Understand the idea first

An exception interrupts the ordinary flow of a calculation. Raising an exception reports that the operation cannot fulfil its contract. A try block marks an operation whose known failure we can handle. An except clause names the specific failure. Catching everything and continuing silently is like covering a warning light: the underlying problem still exists.

CONVERT ONE SUPPLIED VALUE

CODE WALKTHROUGH W03-L05-S03

```
01  text = "twelve"
02  try:
03      quantity = int(text)
04  except ValueError as error:
05      print("Quantity must be whole-number text")
06      print(type(error).__name__)
```

WHAT TO TRACK

Catch the anticipated conversion failure only.

PREDICT THE RESULT

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

CONVERT ONE SUPPLIED VALUE — RESULT

CODE WALKTHROUGH W03-L05-S04

```
01  text = "twelve"
02  try:
03      quantity = int(text)
04  except ValueError as error:
05      print("Quantity must be whole-number text")
06      print(type(error).__name__)
```

WHAT TO TRACK

Catch the anticipated conversion failure only.

VALIDATE THE RESULT

```
Quantity must be
whole-number text
ValueError
```

RAISE ERRORS AT THE CONTRACT BOUNDARY

CODE WALKTHROUGH W03-L05-S05

```
01  def line_total(quantity, price):
02      if type(quantity) is not int or type(price) is not int:
03          raise TypeError("quantity and price must be integers")
04      if quantity < 0 or price < 0:
05          raise ValueError("quantity and price must be non-negati
↪   ve")
06      return quantity * price
07
08  print(line_total(3, 20))
09  print(line_total(-1, 20))
```

WHAT TO TRACK

Integers required; non-negative values required; zero allowed.

PREDICT THE RESULT

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

RAISE ERRORS AT THE CONTRACT BOUNDARY — RESULT

CODE WALKTHROUGH W03-L05-S06

```
01  def line_total(quantity, price):
02      if type(quantity) is not int or type(price) is not int:
03          raise TypeError("quantity and price must be integers")
04      if quantity < 0 or price < 0:
05          raise ValueError("quantity and price must be non-negati
↪   ve")
06      return quantity * price
07
08  print(line_total(3, 20))
09  print(line_total(-1, 20))
```

WHAT TO TRACK

Integers required; non-negative values required; zero allowed.

VALIDATE THE RESULT

```
60
ValueError: quantity and
price must be non-negative
```

HANDLE A KNOWN RECORD ERROR

CODE WALKTHROUGH W03-L05-S07

```
01  def positive_quantity(value):
02      quantity = int(value)
03      if quantity < 0:
04          raise ValueError("quantity must not be negative")
05      return quantity
06
07  for text in ["3", "bad", "-1"]:
08      try:
09          print("Accepted:", positive_quantity(text))
10      except ValueError as error:
11          print("Rejected:", str(error))
```

WHAT TO TRACK

Conversion and domain errors remain visible; records processed separately.

PREDICT THE RESULT

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

HANDLE A KNOWN RECORD ERROR — RESULT

CODE WALKTHROUGH W03-L05-S08

```
01  def positive_quantity(value):
02      quantity = int(value)
03      if quantity < 0:
04          raise ValueError("quantity must not be negative")
05      return quantity
06
07  for text in ["3", "bad", "-1"]:
08      try:
09          print("Accepted:", positive_quantity(text))
10      except ValueError as error:
11          print("Rejected:", str(error))
```

WHAT TO TRACK

Conversion and domain errors remain visible; records processed separately.

VALIDATE THE RESULT

```
Accepted: 3
Rejected: invalid literal
for int() with base 10:
'bad'
Rejected: quantity must not
be negative
```

PAUSE AND APPLY

LEARNER PRACTICE

Pause and apply

Write a function that converts supplied text to an integer and rejects a negative stock quantity. Process zero, minus two and the text unknown. Print a useful rejection message for ValueError. Do not use a bare except and do not replace invalid values with zero.

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

Raise explains why an operation cannot honour its contract. Catch specific anticipated errors where you can make a useful decision. Keep the cause visible, and avoid silently swallowing failures. Next, we use development checks and boundary cases to verify reusable calculations.

UNDERSTAND APPLY VALIDATE EXPLAIN