FOUNDATION LESSON

WEEK 4 · LESSON 1

Create numerical arrays

One dtype describes an array; values still need business meaning

THE IDEA BEHIND THE OPERATION

MENTAL MODEL

The idea behind the operation

One dtype describes an array; values still need business meaning

LIST VALUES AND ARRAY VALUES

CODE WALKTHROUGH W04-L01-S03

```
01  import numpy as np
02  counts = [10, 20, 30]
03  array = np.array(counts)
04  print("List:", counts)
05  print("Array:", array)
```

WHAT TO TRACK

Three supplied counts, in units.

PREDICT THE RESULT

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

LIST VALUES AND ARRAY VALUES — RESULT

CODE WALKTHROUGH W04-L01-S04

```
01  import numpy as np
02  counts = [10, 20, 30]
03  array = np.array(counts)
04  print("List:", counts)
05  print("Array:", array)
```

WHAT TO TRACK

Three supplied counts, in units.

VALIDATE THE RESULT

```
List: [10, 20, 30]
Array: [10 20 30]
```

AN INITIAL STOCK PLAN

CODE WALKTHROUGH W04-L01-S05

```
01  import numpy as np
02  current = np.array([12, 0, 7])
03  planned = np.zeros(3)
04  print("Current:", current)
05  print("Planned:", planned)
06  print("Baseline:", np.ones(3))
07  print("Fixed fee:", np.full(3, 5.0))
08  print("Month labels:", np.arange(1, 4))
```

WHAT TO TRACK

Zero planned units means no extra units currently planned.

PREDICT THE RESULT

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

AN INITIAL STOCK PLAN — RESULT

CODE WALKTHROUGH W04-L01-S06

```
01  import numpy as np
02  current = np.array([12, 0, 7])
03  planned = np.zeros(3)
04  print("Current:", current)
05  print("Planned:", planned)
06  print("Baseline:", np.ones(3))
07  print("Fixed fee:", np.full(3, 5.0))
08  print("Month labels:", np.arange(1, 4))
```

WHAT TO TRACK

Zero planned units means no extra units currently planned.

VALIDATE THE RESULT

```
Current: [12  0  7]
Planned: [0. 0. 0.]
Baseline: [1. 1. 1.]
Fixed fee: [5. 5. 5.]
Month labels: [1 2 3]
```

EMPTY DATA AND A RAGGED TABLE

CODE WALKTHROUGH W04-L01-S07

```
01  import numpy as np
02  empty = np.array([])
03  print("Empty:", empty)
04  try:
05      bad = np.array([[10, 20], [30]])
06  except ValueError as error:
07      print(type(error).__name__)
08      print("Rows must have matching lengths")
```

WHAT TO TRACK

An empty array is permitted; this numeric table must be rectangular.

PREDICT THE RESULT

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

EMPTY DATA AND A RAGGED TABLE — RESULT

CODE WALKTHROUGH W04-L01-S08

```
01  import numpy as np
02  empty = np.array([])
03  print("Empty:", empty)
04  try:
05      bad = np.array([[10, 20], [30]])
06  except ValueError as error:
07      print(type(error).__name__)
08      print("Rows must have matching lengths")
```

WHAT TO TRACK

An empty array is permitted; this numeric table must be rectangular.

VALIDATE THE RESULT

```
Empty: []
ValueError
Rows must have matching
lengths
```

GUIDED MODIFICATION

LEARNER PRACTICE

Guided modification

Create current stock [5, 0, 9, 2] and a four-entry zero plan. Print both. Confirm that zero stock and zero planned additions are two different supplied business facts.

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.

INDEPENDENT APPLICATION

LEARNER PRACTICE

Independent application

Create three month labels with arange and five evenly spaced planning rates from 0.0 to 0.2 with linspace. Print them. Explain the excluded arange stop and included linspace endpoint.

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

An array gives values a consistent representation. Create it from a collection or an initialisation function, and keep business meaning explicit. Empty input and ragged rows are different cases. Next, we describe array dimensions, shape, size and dtype.

UNDERSTAND APPLY VALIDATE EXPLAIN

FOUNDATION LESSON

WEEK 4 · LESSON 2

Read shape, dimensions and dtype

ndim counts axes; shape gives axis lengths; size counts elements

THE IDEA BEHIND THE OPERATION

MENTAL MODEL

The idea behind the operation

ndim counts axes; shape gives axis lengths; size counts elements

SAME VALUES, DIFFERENT STRUCTURES

CODE WALKTHROUGH W04-L02-S03

```
01  import numpy as np
02  vector = np.array([10, 20, 30], dtype=np.int64)
03  row = np.array([[10, 20, 30]], dtype=np.int64)
04  print(vector.ndim, vector.shape, vector.size)
05  print(row.ndim, row.shape, row.size)
06  print(vector.dtype)
```

WHAT TO TRACK

Compare (3,) with (1, 3).

PREDICT THE RESULT

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

SAME VALUES, DIFFERENT STRUCTURES — RESULT

CODE WALKTHROUGH W04-L02-S04

```
01  import numpy as np
02  vector = np.array([10, 20, 30], dtype=np.int64)
03  row = np.array([[10, 20, 30]], dtype=np.int64)
04  print(vector.ndim, vector.shape, vector.size)
05  print(row.ndim, row.shape, row.size)
06  print(vector.dtype)
```

WHAT TO TRACK

Compare (3,) with (1, 3).

VALIDATE THE RESULT

```
1 (3,) 3
2 (1, 3) 3
int64
```

A WAREHOUSE STOCK GRID

CODE WALKTHROUGH W04-L02-S05

```
01  import numpy as np
02  units = np.array([[12, 8, 0], [5, 9, 4]], dtype=np.int64)
03  print("Axes:", units.ndim)
04  print("Shape:", units.shape)
05  print("Elements:", units.size)
06  print("Rows:", len(units))
```

WHAT TO TRACK

Rows are two warehouses; columns are three products.

PREDICT THE RESULT

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

A WAREHOUSE STOCK GRID — RESULT

CODE WALKTHROUGH W04-L02-S06

```
01  import numpy as np
02  units = np.array([[12, 8, 0], [5, 9, 4]], dtype=np.int64)
03  print("Axes:", units.ndim)
04  print("Shape:", units.shape)
05  print("Elements:", units.size)
06  print("Rows:", len(units))
```

WHAT TO TRACK

Rows are two warehouses; columns are three products.

VALIDATE THE RESULT

```
Axes: 2
Shape: (2, 3)
Elements: 6
Rows: 2
```

A CONVERSION CAN LOSE DECIMALS

CODE WALKTHROUGH W04-L02-S07

```
01  import numpy as np
02  prices = np.array([12.75, 8.25])
03  whole = prices.astype(np.int64)
04  empty = np.array([])
05  print("Original:", prices)
06  print("Converted:", whole)
07  print("Empty:", empty.shape, empty.size)
```

WHAT TO TRACK

astype creates converted data;
truncation is not rounding.

PREDICT THE RESULT

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

A CONVERSION CAN LOSE DECIMALS — RESULT

CODE WALKTHROUGH W04-L02-S08

```
01  import numpy as np
02  prices = np.array([12.75, 8.25])
03  whole = prices.astype(np.int64)
04  empty = np.array([])
05  print("Original:", prices)
06  print("Converted:", whole)
07  print("Empty:", empty.shape, empty.size)
```

WHAT TO TRACK

astype creates converted data; truncation is not rounding.

VALIDATE THE RESULT

```
Original: [12.75  8.25]
Converted: [12  8]
Empty: (0,) 0
```

GUIDED MODIFICATION

LEARNER PRACTICE

Guided modification

Create a two-by-two integer stock array. Print ndim, shape, size and dtype. Confirm size is four and explain why len is two.

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.

INDEPENDENT APPLICATION

LEARNER PRACTICE

Independent application

Compare np.array(7) with np.array([7]).
Print ndim, shape and size for each.
Explain why one value does not necessarily mean one dimension.

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

Read ndim, shape, size and dtype as separate properties. Business labels give axes their meaning. Conversion may discard information. Next, we select positions and slices, including the important difference between a view and a copy.

UNDERSTAND APPLY VALIDATE EXPLAIN

FOUNDATION LESSON

WEEK 4 · LESSON 3

Select slices and protect the original

Basic slices share numeric storage; copy creates independent numeric data

THE IDEA BEHIND THE OPERATION

MENTAL MODEL

The idea behind the operation

Basic slices share numeric storage; copy creates independent numeric data

POSITIONS AND WINDOWS

CODE WALKTHROUGH W04-L03-S03

```
01  import numpy as np
02  units = np.array([10, 20, 30, 40, 50])
03  print("First:", units[0])
04  print("Last:", units[-1])
05  print("Middle:", units[1:4])
06  print("Step:", units[::2])
```

WHAT TO TRACK

Indices start at zero; slice stop is excluded.

PREDICT THE RESULT

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

POSITIONS AND WINDOWS — RESULT

CODE WALKTHROUGH W04-L03-S04

```
01  import numpy as np
02  units = np.array([10, 20, 30, 40, 50])
03  print("First:", units[0])
04  print("Last:", units[-1])
05  print("Middle:", units[1:4])
06  print("Step:", units[::2])
```

WHAT TO TRACK

Indices start at zero; slice stop is excluded.

VALIDATE THE RESULT

```
First: 10
Last: 50
Middle: [20 30 40]
Step: [10 30 50]
```

AN EDITABLE DRAFT NEEDS A COPY

CODE WALKTHROUGH W04-L03-S05

```
01  import numpy as np
02  source = np.array([100, 200, 300, 400])
03  view = source[1:3]
04  view[0] = 999
05  draft = source[1:3].copy()
06  draft[0] = 50
07  print("Source:", source)
08  print("Draft:", draft)
```

WHAT TO TRACK

Compare shared data with independent numeric storage.

PREDICT THE RESULT

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

AN EDITABLE DRAFT NEEDS A COPY — RESULT

CODE WALKTHROUGH W04-L03-S06

```
01  import numpy as np
02  source = np.array([100, 200, 300, 400])
03  view = source[1:3]
04  view[0] = 999
05  draft = source[1:3].copy()
06  draft[0] = 50
07  print("Source:", source)
08  print("Draft:", draft)
```

WHAT TO TRACK

Compare shared data with independent numeric storage.

VALIDATE THE RESULT

```
Source: [100 999 300 400]
Draft: [ 50 300]
```

AN INDEX CAN FAIL WHEN A SLICE SUCCEEDS

CODE WALKTHROUGH W04-L03-S07

```
01  import numpy as np
02  values = np.array([10, 20, 30])
03  print("Beyond-end slice:", values[10:20])
04  try:
05      print(values[3])
06  except IndexError as error:
07      print(type(error).__name__)
08      print("Valid positions are 0, 1 and 2")
```

WHAT TO TRACK

Oversized slices can be empty; missing single positions raise.

PREDICT THE RESULT

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

AN INDEX CAN FAIL WHEN A SLICE SUCCEEDS — RESULT

CODE WALKTHROUGH W04-L03-S08

```
01  import numpy as np
02  values = np.array([10, 20, 30])
03  print("Beyond-end slice:", values[10:20])
04  try:
05      print(values[3])
06  except IndexError as error:
07      print(type(error).__name__)
08      print("Valid positions are 0, 1 and 2")
```

WHAT TO TRACK

Oversized slices can be empty; missing single positions raise.

VALIDATE THE RESULT

```
Beyond-end slice: []
IndexError
Valid positions are 0, 1 and
2
```

GUIDED MODIFICATION

LEARNER PRACTICE

Guided modification

Select the first three quantities from [5, 10, 15, 20, 25]. Make a copy, replace its first value with zero and verify the original remains unchanged.

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.

INDEPENDENT APPLICATION

LEARNER PRACTICE

Independent application

From seven daily quantities [4, 6, 8, 10, 12, 14, 16], extract the first five and the final two. Reverse the full series. Print each selection without changing the source.

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

Trace indices from zero and exclude the slice stop. Basic slices share data; a deliberate numeric copy protects independent editing. Check empty results explicitly. Next, we select two-dimensional cells, rows, columns and Boolean masks.

UNDERSTAND APPLY VALIDATE EXPLAIN

FOUNDATION LESSON

WEEK 4 · LESSON 4

Select cells, rows and matching values

Two indices select axes; a Boolean mask selects matching entries

THE IDEA BEHIND THE OPERATION

MENTAL MODEL

The idea behind the operation

Two indices select axes; a Boolean mask selects matching entries

A CELL, A COLUMN AND A ROW

CODE WALKTHROUGH W04-L04-S03

```
01  import numpy as np
02  stock = np.array([[12, 0, 7], [5, 9, 4]])
03  print("Cell:", stock[1, 2])
04  print("Column:", stock[:, 1])
05  print("Row:", stock[0, :])
```

WHAT TO TRACK

Rows: warehouses. Columns: products.

PREDICT THE RESULT

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

A CELL, A COLUMN AND A ROW — RESULT

CODE WALKTHROUGH W04-L04-S04

```
01  import numpy as np
02  stock = np.array([[12, 0, 7], [5, 9, 4]])
03  print("Cell:", stock[1, 2])
04  print("Column:", stock[:, 1])
05  print("Row:", stock[0, :])
```

WHAT TO TRACK

Rows: warehouses. Columns: products.

VALIDATE THE RESULT

```
Cell: 4
Column: [0 9]
Row: [12  0  7]
```

QUANTITIES INSIDE AN ALLOWED BAND

CODE WALKTHROUGH W04-L04-S05

```
01  import numpy as np
02  units = np.array([0, 5, 12, 20])
03  mask = (units >= 5) & (units < 20)
04  selected = units[mask]
05  selected[0] = 99
06  print("Mask:", mask)
07  print("Selected:", selected)
08  print("Source:", units)
```

WHAT TO TRACK

Include 5; exclude 20. Boolean selection creates a copy.

PREDICT THE RESULT

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

QUANTITIES INSIDE AN ALLOWED BAND — RESULT

CODE WALKTHROUGH W04-L04-S06

```
01  import numpy as np
02  units = np.array([0, 5, 12, 20])
03  mask = (units >= 5) & (units < 20)
04  selected = units[mask]
05  selected[0] = 99
06  print("Mask:", mask)
07  print("Selected:", selected)
08  print("Source:", units)
```

WHAT TO TRACK

Include 5; exclude 20. Boolean selection creates a copy.

VALIDATE THE RESULT

```
Mask: [False  True  True
False]
Selected: [99 12]
Source: [ 0  5 12 20]
```

A MASK MUST FIT ITS AXIS

CODE WALKTHROUGH W04-L04-S07

```
01  import numpy as np
02  values = np.array([10, 20, 30])
03  mask = np.array([True, False])
04  print(values[mask])
```

WHAT TO TRACK

Three positions cannot be filtered by two Boolean decisions.

PREDICT THE RESULT

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

A MASK MUST FIT ITS AXIS — RESULT

CODE WALKTHROUGH W04-L04-S08

```
01  import numpy as np
02  values = np.array([10, 20, 30])
03  mask = np.array([True, False])
04  print(values[mask])
```

WHAT TO TRACK

Three positions cannot be filtered by two Boolean decisions.

VALIDATE THE RESULT

```
IndexError: boolean index
did not match indexed array
along axis 0; size of axis
is 3 but size of
corresponding boolean axis
is 2
```

GUIDED MODIFICATION

LEARNER PRACTICE

Guided modification

Filter [2, 6, 10, 14] to keep values at least six and at most ten. Print the mask and result. Explain both inclusive boundaries.

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.

INDEPENDENT APPLICATION

LEARNER PRACTICE

Independent application

For stock [[3, 0], [8, 5], [0, 2]], select whole rows where the first product is positive. Separately select every positive cell. Print both shapes and explain their difference.

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

Choose selectors for each axis, match mask shape to the task and combine comparisons element by element. Boolean selections create copies. Next, we calculate across arrays and predict broadcasting from shapes.

UNDERSTAND APPLY VALIDATE EXPLAIN

FOUNDATION LESSON

WEEK 4 · LESSON 5

Calculate with arrays and broadcasting

Arithmetic is elementwise; shapes must be compatible

THE IDEA BEHIND THE OPERATION

MENTAL MODEL

The idea behind the operation

Arithmetic is elementwise; shapes must be compatible

REPETITION AND ELEMENTWISE ARITHMETIC

CODE WALKTHROUGH W04-L05-S03

```
01  import numpy as np
02  values = [10, 20, 30]
03  array = np.array(values)
04  print("List:", values * 2)
05  print("Array:", array * 2)
06  print("Add:", array + 5)
```

WHAT TO TRACK

The same operator has type-specific behaviour.

PREDICT THE RESULT

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

REPETITION AND ELEMENTWISE ARITHMETIC — RESULT

CODE WALKTHROUGH W04-L05-S04

```
01  import numpy as np
02  values = [10, 20, 30]
03  array = np.array(values)
04  print("List:", values * 2)
05  print("Array:", array * 2)
06  print("Add:", array + 5)
```

WHAT TO TRACK

The same operator has type-specific behaviour.

VALIDATE THE RESULT

```
List: [10, 20, 30, 10, 20,
30]
Array: [20 40 60]
Add: [15 25 35]
```

PRODUCT PRICES ACROSS TWO STORES

CODE WALKTHROUGH W04-L05-S05

```
01  import numpy as np
02  quantity = np.array([[2, 3, 1], [4, 0, 2]])
03  price = np.array([10.0, 20.0, 5.0])
04  revenue = quantity * price
05  print("Shapes:", quantity.shape, price.shape)
06  print("Revenue:")
07  print(revenue)
08  print("Result shape:", revenue.shape)
```

WHAT TO TRACK

Rows: stores. Final-axis product order agrees.

PREDICT THE RESULT

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

PRODUCT PRICES ACROSS TWO STORES — RESULT

CODE WALKTHROUGH W04-L05-S06

```
01  import numpy as np
02  quantity = np.array([[2, 3, 1], [4, 0, 2]])
03  price = np.array([10.0, 20.0, 5.0])
04  revenue = quantity * price
05  print("Shapes:", quantity.shape, price.shape)
06  print("Revenue:")
07  print(revenue)
08  print("Result shape:", revenue.shape)
```

WHAT TO TRACK

Rows: stores. Final-axis product order agrees.

VALIDATE THE RESULT

```
Shapes: (2, 3) (3,)
Revenue:
[[20. 60.  5.]
 [40.  0. 10.]]
Result shape: (2, 3)
```

REJECT ZERO BEFORE DIVISION

CODE WALKTHROUGH W04-L05-S07

```
01  import numpy as np
02  revenue = np.array([100.0, 50.0])
03  units = np.array([5, 0])
04  if np.any(units == 0):
05      raise ValueError("unit price requires positive quantities")
06  print(revenue / units)
```

WHAT TO TRACK

This task requires strictly positive quantities.

PREDICT THE RESULT

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

REJECT ZERO BEFORE DIVISION — RESULT

CODE WALKTHROUGH W04-L05-S08

```
01  import numpy as np
02  revenue = np.array([100.0, 50.0])
03  units = np.array([5, 0])
04  if np.any(units == 0):
05      raise ValueError("unit price requires positive quantities")
06  print(revenue / units)
```

WHAT TO TRACK

This task requires strictly positive quantities.

VALIDATE THE RESULT

```
ValueError: unit price
requires positive quantities
```

GUIDED MODIFICATION

LEARNER PRACTICE

Guided modification

Compute revenues for quantities [2, 0, 5] and prices [12.0, 8.0, 3.0]. Apply a fictional 10 percent discount to each revenue. Print both arrays; zero quantity remains valid.

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.

INDEPENDENT APPLICATION

LEARNER PRACTICE

Independent application

Try multiplying a two-by-three table by a two-entry vector. Catch ValueError and explain why the rightmost dimensions conflict. Then repair the business task using three product prices.

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

Array arithmetic works element by element. Predict broadcasting from the rightmost dimensions and verify business alignment separately. Guard invalid denominators before division. Next, we reduce arrays into totals and summaries along a stated axis.

UNDERSTAND APPLY VALIDATE EXPLAIN

FOUNDATION LESSON

WEEK 4 · LESSON 6

Summarise arrays along an axis

A reduction removes the selected axis and retains the others

THE IDEA BEHIND THE OPERATION

MENTAL MODEL

The idea behind the operation

A reduction removes the selected axis and retains the others

TOTALS WITH EXPLICIT OUTPUT SHAPES

CODE WALKTHROUGH W04-L06-S03

```
01  import numpy as np
02  sales = np.array([[20, 60, 5], [40, 0, 10]])
03  print("Product totals:", sales.sum(axis=0))
04  print("Store totals:", sales.sum(axis=1))
05  print("Grand total:", sales.sum())
```

WHAT TO TRACK

Rows: two stores. Columns: three products.

PREDICT THE RESULT

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

TOTALS WITH EXPLICIT OUTPUT SHAPES — RESULT

CODE WALKTHROUGH W04-L06-S04

```
01  import numpy as np
02  sales = np.array([[20, 60, 5], [40, 0, 10]])
03  print("Product totals:", sales.sum(axis=0))
04  print("Store totals:", sales.sum(axis=1))
05  print("Grand total:", sales.sum())
```

WHAT TO TRACK

Rows: two stores. Columns: three products.

VALIDATE THE RESULT

```
Product totals: [60 60 15]
Store totals: [85 50]
Grand total: 135
```

SUPPLIED ONE-PERIOD RETURNS

CODE WALKTHROUGH W04-L06-S05

```
01  import numpy as np
02  returns = np.array([0.02, -0.01, 0.03])
03  print("Count:", returns.size)
04  print("Mean:", round(float(returns.mean()), 4))
05  print("Min:", float(returns.min()))
06  print("Max:", float(returns.max()))
07  print("Population SD:", round(float(returns.std(ddof=0)), 4))
```

WHAT TO TRACK

Three fictional same-period decimal returns; describe this supplied set.

PREDICT THE RESULT

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

SUPPLIED ONE-PERIOD RETURNS — RESULT

CODE WALKTHROUGH W04-L06-S06

```
01  import numpy as np
02  returns = np.array([0.02, -0.01, 0.03])
03  print("Count:", returns.size)
04  print("Mean:", round(float(returns.mean()), 4))
05  print("Min:", float(returns.min()))
06  print("Max:", float(returns.max()))
07  print("Population SD:", round(float(returns.std(ddof=0)), 4))
```

WHAT TO TRACK

Three fictional same-period decimal returns; describe this supplied set.

VALIDATE THE RESULT

```
Count: 3
Mean: 0.0133
Min: -0.01
Max: 0.03
Population SD: 0.017
```

AN EMPTY BATCH HAS NO MEAN HERE

CODE WALKTHROUGH W04-L06-S07

```
01  import numpy as np
02  values = np.array([])
03  print("Empty total:", values.sum())
04  if values.size == 0:
05      raise ValueError("at least one observation is required")
06  print(values.mean())
```

WHAT TO TRACK

Empty sum is zero; an average requires observations.

PREDICT THE RESULT

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

AN EMPTY BATCH HAS NO MEAN HERE — RESULT

CODE WALKTHROUGH W04-L06-S08

```
01  import numpy as np
02  values = np.array([])
03  print("Empty total:", values.sum())
04  if values.size == 0:
05      raise ValueError("at least one observation is required")
06  print(values.mean())
```

WHAT TO TRACK

Empty sum is zero; an average requires observations.

VALIDATE THE RESULT

```
Empty total: 0.0
ValueError: at least one
observation is required
```

GUIDED MODIFICATION

LEARNER PRACTICE

Guided modification

For [[10, 20], [30, 40]], print axis-zero totals, axis-one totals and the grand total. Reconcile both summary arrays with the same grand total.

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.

INDEPENDENT APPLICATION

LEARNER PRACTICE

Independent application

For [10.0, NaN, 30.0], report the valid count, ordinary mean and nanmean. The supplied policy permits omitting missing observations only when the count is reported. Explain the denominator.

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

Predict the shape remaining after a reduction and reconcile totals. State policies for empty and missing inputs. One-period summaries stay separate from compounding. Next, pandas adds labels to our numerical data.

UNDERSTAND APPLY VALIDATE EXPLAIN

FOUNDATION LESSON

WEEK 4 · LESSON 7

Create labelled Series and DataFrames

Series labels align values; a DataFrame organises labelled columns

THE IDEA BEHIND THE OPERATION

MENTAL MODEL

The idea behind the operation

Series labels align values; a DataFrame organises labelled columns

A LABELLED QUANTITY SERIES

CODE WALKTHROUGH W04-L07-S03

```
01  import pandas as pd
02  units = pd.Series([5, 8], index=["A", "B"], name="units")
03  print(units.to_string())
04  print("A:", units.loc["A"])
05  print("Labels:", units.index.tolist())
```

WHAT TO TRACK

Product codes A and B label their quantities.

PREDICT THE RESULT

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

A LABELLED QUANTITY SERIES — RESULT

CODE WALKTHROUGH W04-L07-S04

```
01  import pandas as pd
02  units = pd.Series([5, 8], index=["A", "B"], name="units")
03  print(units.to_string())
04  print("A:", units.loc["A"])
05  print("Labels:", units.index.tolist())
```

WHAT TO TRACK

Product codes A and B label their quantities.

VALIDATE THE RESULT

```
A    5
B    8
A: 5
Labels: ['A', 'B']
```

CREATE A MIXED-COLUMN TABLE

CODE WALKTHROUGH W04-L07-S05

```
01  import pandas as pd
02  df = pd.DataFrame({
03      "product": ["A", "B"],
04      "units": [5, 8],
05      "price": [10.0, 4.5]
06  })
07  print(df.to_string(index=False))
08  print("Shape:", df.shape)
09  print(df.dtypes.to_string())
```

WHAT TO TRACK

Text identifiers, whole quantities and decimal prices.

PREDICT THE RESULT

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

CREATE A MIXED-COLUMN TABLE — RESULT

CODE WALKTHROUGH W04-L07-S06

```
01  import pandas as pd
02  df = pd.DataFrame({
03      "product": ["A", "B"],
04      "units": [5, 8],
05      "price": [10.0, 4.5]
06  })
07  print(df.to_string(index=False))
08  print("Shape:", df.shape)
09  print(df.dtypes.to_string())
```

WHAT TO TRACK

Text identifiers, whole quantities and decimal prices.

VALIDATE THE RESULT

```
product  units  price
      A      5   10.0
      B      8    4.5
Shape: (2, 3)
product     object
units        int64
price      float64
```

ARITHMETIC MATCHES LABELS

CODE WALKTHROUGH W04-L07-S07

```
01  import pandas as pd
02  a = pd.Series([100, 200], index=["A", "B"])
03  b = pd.Series([10, 20], index=["B", "C"])
04  result = a + b
05  print(result.to_string())
```

WHAT TO TRACK

Only label B is present in both inputs.

PREDICT THE RESULT

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

ARITHMETIC MATCHES LABELS — RESULT

CODE WALKTHROUGH W04-L07-S08

```
01  import pandas as pd
02  a = pd.Series([100, 200], index=["A", "B"])
03  b = pd.Series([10, 20], index=["B", "C"])
04  result = a + b
05  print(result.to_string())
```

WHAT TO TRACK

Only label B is present in both inputs.

VALIDATE THE RESULT

```
A      NaN
B    210.0
C      NaN
```

GUIDED MODIFICATION

LEARNER PRACTICE

Guided modification

Create a Series for product codes P and Q with quantities three and seven. Print it, select Q by label and list its index.

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.

INDEPENDENT APPLICATION

LEARNER PRACTICE

Independent application

Add Series [4, 6] labelled [A, B] and Series [10, 20] labelled [B, A]. Predict the aligned result before running. Explain why positional addition would answer a different question.

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

Use Series for labelled values and DataFrames for labelled columns. Arithmetic aligns labels, and missing partners matter. Next, we load a supplied CSV and inspect its structure before trusting it.

UNDERSTAND APPLY VALIDATE EXPLAIN

FOUNDATION LESSON

WEEK 4 · LESSON 8

Load and inspect a CSV table

Specify important types and verify required columns before analysis

THE IDEA BEHIND THE OPERATION

MENTAL MODEL

The idea behind the operation

Specify important types and verify required columns before analysis

PRESERVE INVOICE IDENTIFIERS

CODE WALKTHROUGH W04-L08-S03

```
01  import pandas as pd
02  df = pd.read_csv("data/invoices.csv", encoding="utf-8",
03                   dtype={"invoice_id": "string", "units": "int64
↪   "})
04  print(df.head(2).to_string(index=False))
05  print("Shape:", df.shape)
06  print("IDs:", df["invoice_id"].tolist())
```

WHAT TO TRACK

Invoice identifiers are text, including leading zeros.

PREDICT THE RESULT

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

PRESERVE INVOICE IDENTIFIERS — RESULT

CODE WALKTHROUGH W04-L08-S04

```
01  import pandas as pd
02  df = pd.read_csv("data/invoices.csv", encoding="utf-8",
03                   dtype={"invoice_id": "string", "units": "int64
↪   "})
04  print(df.head(2).to_string(index=False))
05  print("Shape:", df.shape)
06  print("IDs:", df["invoice_id"].tolist())
```

WHAT TO TRACK

Invoice identifiers are text, including leading zeros.

VALIDATE THE RESULT

```
invoice_id product  units
price       date
       001       A      2
12.0 2026-01-01
       002       B      5
3.0 2026-01-02
Shape: (3, 5)
IDs: ['001', '002', '003']
```

INSPECT TYPES AND NUMERICAL SUMMARIES

CODE WALKTHROUGH W04-L08-S05

```
01  import pandas as pd
02  df = pd.read_csv("data/invoices.csv", encoding="utf-8",
03                   dtype={"invoice_id": "string"})
04  print(df.dtypes.to_string())
05  print(df["units"].describe().to_string())
06  # Run df.info() in the notebook for the full structural report.
```

WHAT TO TRACK

Describe summarises; it does not validate the business rules.

PREDICT THE RESULT

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

INSPECT TYPES AND NUMERICAL SUMMARIES — RESULT

CODE WALKTHROUGH W04-L08-S06 1/2

```
01  import pandas as pd
02  df = pd.read_csv("data/invoices.csv", encoding="utf-8",
03                   dtype={"invoice_id": "string"})
04  print(df.dtypes.to_string())
05  print(df["units"].describe().to_string())
06  # Run df.info() in the notebook for the full structural report.
```

WHAT TO TRACK

Describe summarises; it does not validate the business rules.

VALIDATE THE RESULT

```
invoice_id    string[python]
product               object
units                  int64
price                float64
date                  object
count    3.000000
mean     2.333333
std      2.516611
min      0.000000
25%      1.000000
50%      2.000000
75%      3.500000
```

INSPECT TYPES AND NUMERICAL SUMMARIES — RESULT

CODE WALKTHROUGH W04-L08-S06 2/2

WHAT TO TRACK

Describe summarises; it does not validate the business rules.

VALIDATE THE RESULT

```
max       5.000000
```

A REQUIRED COLUMN IS ABSENT

CODE WALKTHROUGH W04-L08-S07

```
01  import pandas as pd
02  df = pd.read_csv("data/missing_column.csv", encoding="utf-8")
03  required = {"invoice_id", "units", "price"}
04  missing = required - set(df.columns)
05  if missing:
06      raise ValueError("missing required columns: " + ", ".join(s
↪   orted(missing)))
07  print("Schema accepted")
```

WHAT TO TRACK

Require invoice_id, units and price before calculations.

PREDICT THE RESULT

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

A REQUIRED COLUMN IS ABSENT — RESULT

CODE WALKTHROUGH W04-L08-S08

```
01  import pandas as pd
02  df = pd.read_csv("data/missing_column.csv", encoding="utf-8")
03  required = {"invoice_id", "units", "price"}
04  missing = required - set(df.columns)
05  if missing:
06      raise ValueError("missing required columns: " + ", ".join(s
↪   orted(missing)))
07  print("Schema accepted")
```

WHAT TO TRACK

Require invoice_id, units and price before calculations.

VALIDATE THE RESULT

```
ValueError: missing required
columns: price
```

GUIDED MODIFICATION

LEARNER PRACTICE

Guided modification

Load invoices.csv with invoice_id as string. Print the first two rows, the shape and column names. Check that units and price are present.

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.

INDEPENDENT APPLICATION

LEARNER PRACTICE

Independent application

Run info on the supplied invoices and explain why it returns None. Then load missing_column.csv and report the missing price column explicitly without fabricating values.

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

Protect identifier meaning, inspect structure and apply explicit schema checks. A preview and summary support understanding but do not certify data quality. Next, we select labelled rows and positional rows deliberately.

UNDERSTAND APPLY VALIDATE EXPLAIN

FOUNDATION LESSON

WEEK 4 · LESSON 9

Select with loc and iloc

loc selects labels; iloc selects integer positions

THE IDEA BEHIND THE OPERATION

MENTAL MODEL

The idea behind the operation

loc selects labels; iloc selects integer positions

LABEL THIRTY AND POSITION ONE

CODE WALKTHROUGH W04-L09-S03

```
01  import pandas as pd
02  df = pd.DataFrame({"units": [5, 8, 2]}, index=[10, 30, 50])
03  print("Label:", df.loc[30, "units"])
04  print("Position:", df.iloc[1, 0])
05  print("Label slice:", df.loc[10:30, "units"].tolist())
06  print("Position slice:", df.iloc[0:1, 0].tolist())
```

WHAT TO TRACK

Index labels are 10, 30 and 50.

PREDICT THE RESULT

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

LABEL THIRTY AND POSITION ONE — RESULT

CODE WALKTHROUGH W04-L09-S04

```
01  import pandas as pd
02  df = pd.DataFrame({"units": [5, 8, 2]}, index=[10, 30, 50])
03  print("Label:", df.loc[30, "units"])
04  print("Position:", df.iloc[1, 0])
05  print("Label slice:", df.loc[10:30, "units"].tolist())
06  print("Position slice:", df.iloc[0:1, 0].tolist())
```

WHAT TO TRACK

Index labels are 10, 30 and 50.

VALIDATE THE RESULT

```
Label: 8
Position: 8
Label slice: [5, 8]
Position slice: [5]
```

ONE COLUMN OR A ONE-COLUMN TABLE

CODE WALKTHROUGH W04-L09-S05

```
01  import pandas as pd
02  df = pd.DataFrame({"product": ["A", "B"], "units": [5, 8]})
03  column = df["units"]
04  table = df[["units"]]
05  print(type(column).__name__, column.shape)
06  print(type(table).__name__, table.shape)
07  print(df.loc[0:1, ["product", "units"]].to_string(index=False))
```

WHAT TO TRACK

A column-name list preserves a DataFrame result.

PREDICT THE RESULT

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

ONE COLUMN OR A ONE-COLUMN TABLE — RESULT

CODE WALKTHROUGH W04-L09-S06

```
01  import pandas as pd
02  df = pd.DataFrame({"product": ["A", "B"], "units": [5, 8]})
03  column = df["units"]
04  table = df[["units"]]
05  print(type(column).__name__, column.shape)
06  print(type(table).__name__, table.shape)
07  print(df.loc[0:1, ["product", "units"]].to_string(index=False))
```

WHAT TO TRACK

A column-name list preserves a DataFrame result.

VALIDATE THE RESULT

```
Series (2,)
DataFrame (2, 1)
product  units
      A      5
      B      8
```

A MISSING LABEL DOES NOT MEAN FIRST POSITION

CODE WALKTHROUGH W04-L09-S07

```
01  import pandas as pd
02  df = pd.DataFrame({"units": [5, 8]}, index=[10, 30])
03  print(df.loc[0, "units"])
```

WHAT TO TRACK

Zero is not an index label in this table.

PREDICT THE RESULT

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

A MISSING LABEL DOES NOT MEAN FIRST POSITION — RESULT

CODE WALKTHROUGH W04-L09-S08

```
01  import pandas as pd
02  df = pd.DataFrame({"units": [5, 8]}, index=[10, 30])
03  print(df.loc[0, "units"])
```

WHAT TO TRACK

Zero is not an index label in this table.

VALIDATE THE RESULT

```
KeyError: 0
```

GUIDED MODIFICATION

LEARNER PRACTICE

Guided modification

Create values [7, 9, 11] with labels [100, 200, 300]. Select label 200 and position one. Compare loc[100:200] with iloc[0:1].

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.

INDEPENDENT APPLICATION

LEARNER PRACTICE

Independent application

Load invoices.csv, then select invoice_id and units as a DataFrame. Select the final row by position and explain why its default label should not be treated as the invoice identifier.

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

Separate labels from positions and predict result type and shape. A missing label needs an intentional repair. Next, filtering and sorting will change row order while often retaining old index labels.

UNDERSTAND APPLY VALIDATE EXPLAIN

FOUNDATION LESSON

WEEK 4 · LESSON 10

Filter and sort an explicit report

Filtering retains labels; sorting and resetting are separate choices

THE IDEA BEHIND THE OPERATION

MENTAL MODEL

The idea behind the operation

Filtering retains labels; sorting and resetting are separate choices

COMBINE BUSINESS CONDITIONS

CODE WALKTHROUGH W04-L10-S03

```
01  import pandas as pd
02  df = pd.DataFrame({"product":["A","B","C"],
03                     "units":[5,0,8], "price":[10.0,4.0,12.0]})
04  mask = (df["units"] > 0) & (df["price"] <= 10)
05  print(df.loc[mask, ["product","units"]].to_string(index=False))
```

WHAT TO TRACK

Require positive units and price at most EUR 10.

PREDICT THE RESULT

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

COMBINE BUSINESS CONDITIONS — RESULT

CODE WALKTHROUGH W04-L10-S04

```
01  import pandas as pd
02  df = pd.DataFrame({"product":["A","B","C"],
03                     "units":[5,0,8], "price":[10.0,4.0,12.0]})
04  mask = (df["units"] > 0) & (df["price"] <= 10)
05  print(df.loc[mask, ["product","units"]].to_string(index=False))
```

WHAT TO TRACK

Require positive units and price at most EUR 10.

VALIDATE THE RESULT

```
product  units
      A      5
```

SORT, RESET AND ASSIGN DELIBERATELY

CODE WALKTHROUGH W04-L10-S05

```
01  import pandas as pd
02  df = pd.DataFrame({"product":["B","A","C"], "units":[5,5,2]}, i
↪   ndex=[7,3,9])
03  report = df.sort_values(["units","product"], ascending=[False,T
↪   rue])
04  print("Original labels:", report.index.tolist())
05  report = report.reset_index(drop=True)
06  report.loc[report["units"] == 2, "units"] = 0
07  print(report.to_string(index=False))
```

WHAT TO TRACK

Quantity descending; product ascending breaks ties.

PREDICT THE RESULT

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

SORT, RESET AND ASSIGN DELIBERATELY — RESULT

CODE WALKTHROUGH W04-L10-S06

```
01  import pandas as pd
02  df = pd.DataFrame({"product":["B","A","C"], "units":[5,5,2]}, i
↪   ndex=[7,3,9])
03  report = df.sort_values(["units","product"], ascending=[False,T
↪   rue])
04  print("Original labels:", report.index.tolist())
05  report = report.reset_index(drop=True)
06  report.loc[report["units"] == 2, "units"] = 0
07  print(report.to_string(index=False))
```

WHAT TO TRACK

Quantity descending; product ascending breaks ties.

VALIDATE THE RESULT

```
Original labels: [3, 7, 9]
product  units
      A      5
      B      5
      C      0
```

UNKNOWN FILTER DECISIONS NEED A POLICY

CODE WALKTHROUGH W04-L10-S07

```
01  import pandas as pd
02  units = pd.Series([5, None, 0], dtype="Int64")
03  mask = (units > 0).fillna(False)
04  print("Missing quantities:", int(units.isna().sum()))
05  print("Kept:", units[mask].tolist())
```

WHAT TO TRACK

Unknown quantities are excluded from this report and counted.

PREDICT THE RESULT

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

UNKNOWN FILTER DECISIONS NEED A POLICY — RESULT

CODE WALKTHROUGH W04-L10-S08

```
01  import pandas as pd
02  units = pd.Series([5, None, 0], dtype="Int64")
03  mask = (units > 0).fillna(False)
04  print("Missing quantities:", int(units.isna().sum()))
05  print("Kept:", units[mask].tolist())
```

WHAT TO TRACK

Unknown quantities are excluded from this report and counted.

VALIDATE THE RESULT

```
Missing quantities: 1
Kept: [5]
```

GUIDED MODIFICATION

LEARNER PRACTICE

Guided modification

Filter products with units at least three, sort units descending and break ties by product code ascending. Reset the display index and print the result.

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.

INDEPENDENT APPLICATION

LEARNER PRACTICE

Independent application

Create an independent copy of rows with positive units. Set units below three to three inside the copy using loc. Print both original and adjusted tables to prove the original is unchanged.

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

State filter boundaries and missing policies, sort with tie-breakers and reset labels deliberately. Use one-step assignment. Next, we create calculated columns and parse dates with an explicit format.

UNDERSTAND APPLY VALIDATE EXPLAIN

FOUNDATION LESSON

WEEK 4 · LESSON 11

Create columns and parse dates

Derived columns retain row meaning; date parsing follows an explicit format

THE IDEA BEHIND THE OPERATION

MENTAL MODEL

The idea behind the operation

Derived columns retain row meaning; date parsing follows an explicit format

CALCULATE EACH INVOICE REVENUE

CODE WALKTHROUGH W04-L11-S03

```
01  import pandas as pd
02  df = pd.DataFrame({"units":[2,0,5], "price":[12.0,8.0,3.0]})
03  df["revenue"] = df["units"] * df["price"]
04  print(df.to_string(index=False))
05  print("Total:", df["revenue"].sum())
```

WHAT TO TRACK

Revenue is quantity times unit price, in EUR.

PREDICT THE RESULT

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

CALCULATE EACH INVOICE REVENUE — RESULT

CODE WALKTHROUGH W04-L11-S04

```
01  import pandas as pd
02  df = pd.DataFrame({"units":[2,0,5], "price":[12.0,8.0,3.0]})
03  df["revenue"] = df["units"] * df["price"]
04  print(df.to_string(index=False))
05  print("Total:", df["revenue"].sum())
```

WHAT TO TRACK

Revenue is quantity times unit price, in EUR.

VALIDATE THE RESULT

```
 units  price  revenue
     2   12.0     24.0
     0    8.0      0.0
     5    3.0     15.0
Total: 39.0
```

ASSIGNMENT ALIGNS SERIES LABELS

CODE WALKTHROUGH W04-L11-S05

```
01  import pandas as pd
02  df = pd.DataFrame({"product":["A","B"]}, index=[10,20])
03  df["wrong"] = pd.Series([5,8])
04  df["right"] = pd.Series([5,8], index=df.index)
05  print(df.to_string())
```

WHAT TO TRACK

The external Series must use the intended row labels.

PREDICT THE RESULT

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

ASSIGNMENT ALIGNS SERIES LABELS — RESULT

CODE WALKTHROUGH W04-L11-S06

```
01  import pandas as pd
02  df = pd.DataFrame({"product":["A","B"]}, index=[10,20])
03  df["wrong"] = pd.Series([5,8])
04  df["right"] = pd.Series([5,8], index=df.index)
05  print(df.to_string())
```

WHAT TO TRACK

The external Series must use the intended row labels.

VALIDATE THE RESULT

```
   product  wrong  right
10       A    NaN      5
20       B    NaN      8
```

INVALID DATES REMAIN VISIBLE

CODE WALKTHROUGH W04-L11-S07

```
01  import pandas as pd
02  text = pd.Series(["2026-01-02", "2026-02-30", "bad"])
03  dates = pd.to_datetime(text, format="%Y-%m-%d", errors="coerce"
↪   )
04  print(dates.dt.strftime("%Y-%m-%d").fillna("INVALID").tolist())
05  print("Invalid:", int(dates.isna().sum()))
```

WHAT TO TRACK

Use year-month-day; invalid dates are flagged, not accepted.

PREDICT THE RESULT

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

INVALID DATES REMAIN VISIBLE — RESULT

CODE WALKTHROUGH W04-L11-S08

```
01  import pandas as pd
02  text = pd.Series(["2026-01-02", "2026-02-30", "bad"])
03  dates = pd.to_datetime(text, format="%Y-%m-%d", errors="coerce"
↪   )
04  print(dates.dt.strftime("%Y-%m-%d").fillna("INVALID").tolist())
05  print("Invalid:", int(dates.isna().sum()))
```

WHAT TO TRACK

Use year-month-day; invalid dates are flagged, not accepted.

VALIDATE THE RESULT

```
['2026-01-02', 'INVALID',
'INVALID']
Invalid: 2
```

GUIDED MODIFICATION

LEARNER PRACTICE

Guided modification

Calculate revenue for units [3, 4] and prices [5.0, 2.5]. Parse dates 2026-03-01 and 2026-03-02 with an explicit format. Print the two revenues and dates.

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.

INDEPENDENT APPLICATION

LEARNER PRACTICE

Independent application

Parse [2026-01-01, 2026-13-01] under the year-month-day rule. Report the invalid count and retain only valid dates in a separate report. Preserve the original text for correction.

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

Derived columns implement stated row rules. Series assignment aligns labels, and explicit date formats prevent ambiguity. Count parsing failures rather than hiding them. Next, we combine string operations with a complete offline DataFrame report.

UNDERSTAND APPLY VALIDATE EXPLAIN

FOUNDATION LESSON

WEEK 4 · LESSON 12

Build a validated DataFrame report

Clean text deliberately, validate records and reconcile the final report

THE IDEA BEHIND THE OPERATION

MENTAL MODEL

The idea behind the operation

Clean text deliberately, validate records and reconcile the final report

NORMALISE TEXT WITHOUT INVENTING MISSING CODES

CODE WALKTHROUGH W04-L12-S03

```
01  import pandas as pd
02  codes = pd.Series([" a ", None, "b"], dtype="string")
03  clean = codes.str.strip().str.upper()
04  print(clean.fillna("MISSING").tolist())
05  print("Missing:", int(clean.isna().sum()))
```

WHAT TO TRACK

Codes are uppercase; absent codes remain absent.

PREDICT THE RESULT

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

NORMALISE TEXT WITHOUT INVENTING MISSING CODES — RESULT

CODE WALKTHROUGH W04-L12-S04

```
01  import pandas as pd
02  codes = pd.Series([" a ", None, "b"], dtype="string")
03  clean = codes.str.strip().str.upper()
04  print(clean.fillna("MISSING").tolist())
05  print("Missing:", int(clean.isna().sum()))
```

WHAT TO TRACK

Codes are uppercase; absent codes remain absent.

VALIDATE THE RESULT

```
['A', 'MISSING', 'B']
Missing: 1
```

VALIDATE AND RECONCILE THE SUPPLIED REPORT

CODE WALKTHROUGH W04-L12-S05

```
01  import pandas as pd
02  df = pd.read_csv("data/sales.csv", encoding="utf-8", dtype={"pr
↪   oduct":"string"})
03  df["product"] = df["product"].str.strip().str.upper()
04  df["date"] = pd.to_datetime(df["date"], format="%Y-%m-%d", erro
↪   rs="coerce")
05  valid = (df["product"].notna() & df["product"].ne("")
06           & df["date"].notna() & df["units"].ge(0) & df["price"]
↪   .ge(0))
07  accepted = df.loc[valid].copy()
08  accepted["revenue"] = accepted["units"] * accepted["price"]
09  print("Accepted:", len(accepted), "Rejected:", int((~valid).sum
↪   ()))
10  print(accepted[["product","revenue"]].to_string(index=False))
11  print("Total EUR:", accepted["revenue"].sum())
```

WHAT TO TRACK

Missing/empty codes, invalid dates and negative values are rejected.

PREDICT THE RESULT

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

VALIDATE AND RECONCILE THE SUPPLIED REPORT — RESULT

CODE WALKTHROUGH W04-L12-S06

```
01  import pandas as pd
02  df = pd.read_csv("data/sales.csv", encoding="utf-8", dtype={"pr
↪   oduct":"string"})
03  df["product"] = df["product"].str.strip().str.upper()
04  df["date"] = pd.to_datetime(df["date"], format="%Y-%m-%d", erro
↪   rs="coerce")
05  valid = (df["product"].notna() & df["product"].ne("")
06           & df["date"].notna() & df["units"].ge(0) & df["price"]
↪   .ge(0))
07  accepted = df.loc[valid].copy()
08  accepted["revenue"] = accepted["units"] * accepted["price"]
09  print("Accepted:", len(accepted), "Rejected:", int((~valid).sum
↪   ()))
10  print(accepted[["product","revenue"]].to_string(index=False))
11  print("Total EUR:", accepted["revenue"].sum())
```

WHAT TO TRACK

Missing/empty codes, invalid dates and negative values are rejected.

VALIDATE THE RESULT

```
Accepted: 3 Rejected: 1
product  revenue
      A     24.0
      B     15.0
      D      0.0
Total EUR: 39.0
```

ALL RECORDS CAN BE REJECTED

CODE WALKTHROUGH W04-L12-S07

```
01  import pandas as pd
02  df = pd.DataFrame({"product":["",None], "units":[-1,2]})
03  df["product"] = df["product"].astype("string")
04  valid = (df["product"].notna() & df["product"].ne("")
05           & df["units"].ge(0)).fillna(False)
06  accepted = df.loc[valid]
07  print("Accepted:", len(accepted))
08  if accepted.empty:
09      raise ValueError("no valid records for this report")
```

WHAT TO TRACK

A completed filter is not proof that useful data remains.

PREDICT THE RESULT

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

ALL RECORDS CAN BE REJECTED — RESULT

CODE WALKTHROUGH W04-L12-S08

```
01  import pandas as pd
02  df = pd.DataFrame({"product":["",None], "units":[-1,2]})
03  df["product"] = df["product"].astype("string")
04  valid = (df["product"].notna() & df["product"].ne("")
05           & df["units"].ge(0)).fillna(False)
06  accepted = df.loc[valid]
07  print("Accepted:", len(accepted))
08  if accepted.empty:
09      raise ValueError("no valid records for this report")
```

WHAT TO TRACK

A completed filter is not proof that useful data remains.

VALIDATE THE RESULT

```
Accepted: 0
ValueError: no valid records
for this report
```

GUIDED MODIFICATION

LEARNER PRACTICE

Guided modification

Normalise product codes [a , B, missing] using string dtype, strip and upper. Report one missing code. Filter to present non-empty codes without replacing the missing code with a product.

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.

INDEPENDENT APPLICATION

LEARNER PRACTICE

Independent application

Rebuild the sales.csv report, preserve raw date text, export accepted_report.csv and rejected_report.csv, and verify accepted plus rejected equals input rows. State the accepted revenue total and test the all-rejected boundary.

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

You can now create and inspect numerical arrays, reason about selection, broadcasting and reductions, and build labelled DataFrame reports. Keep units, labels and validation policies explicit. Complete the required weekly quiz and ninety-minute project. Lesson practice remains formative.

UNDERSTAND APPLY VALIDATE EXPLAIN