FOUNDATION LESSON

WEEK 2 · LESSON 1

Decisions in Python

A precise business rule selects the instructions to run.

A RULE WITH A PRECISE BOUNDARY

MENTAL MODEL

A rule with a precise boundary

Fictional rule
Extra review applies when the amount is greater than 2000

2000 itself does not pass this condition

THE CONDITIONAL BLOCK

CODE WALKTHROUGH W02-L01-S03

```
01  amount = 2500
02  if amount > 2000:
03      print("review")
04  print("checked")
```

WHAT TO TRACK

The condition controls the indented line

PREDICT THE RESULT

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

THE TRUE PATH

CODE WALKTHROUGH W02-L01-S04

```
01  amount = 2500
02  if amount > 2000:
03      print("review")
04  print("checked")
```

WHAT TO TRACK

2500 > 2000 is True

Python enters the indented block

VALIDATE THE RESULT

```
review
checked
```

THE FALSE PATH

CODE WALKTHROUGH W02-L01-S05

```
01  amount = 1500
02  if amount > 2000:
03      print("review")
04  print("checked")
```

WHAT TO TRACK

Only the input changes

Predict what happens when amount is

PREDICT THE RESULT

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

EXECUTION AFTER THE BLOCK

CODE WALKTHROUGH W02-L01-S06

```
01  amount = 1500
02  if amount > 2000:
03      print("review")
04  print("checked")
```

WHAT TO TRACK

The review message is skipped

The checked message still runs

VALIDATE THE RESULT

checked

BOUNDARY TESTS

MENTAL MODEL

Boundary tests

Amount 1999: checked
Amount 2000: checked
Amount 2001: review, then checked

TWO ALTERNATIVE OUTCOMES

CODE WALKTHROUGH W02-L01-S08

```
01  balance = -25
02  if balance >= 0:
03      print("non-negative")
04  else:
05      print("negative")
06  print("classification complete")
```

WHAT TO TRACK

One if–else statement

One branch for True and another for

PREDICT THE RESULT

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

THE ELSE BRANCH

CODE WALKTHROUGH W02-L01-S09

```
01  balance = -25
02  if balance >= 0:
03      print("non-negative")
04  else:
05      print("negative")
06  print("classification complete")
```

WHAT TO TRACK

–25 >= 0 is False

The else block supplies the

VALIDATE THE RESULT

```
negative
classification complete
```

ZERO BELONGS TO THE NON-NEGATIVE PATH

CODE WALKTHROUGH W02-L01-S10

```
01  balance = 0
02  if balance >= 0:
03      print("non-negative")
04  else:
05      print("negative")
06  print("classification complete")
```

WHAT TO TRACK

The operator >= includes equality

Zero is a valid numeric input here

VALIDATE THE RESULT

```
non-negative
classification complete
```

A DELIBERATE INDENTATION ERROR

CODE WALKTHROUGH W02-L01-S11

```
01  amount = 2500
02  if amount > 2000:
03  print("review")
```

WHAT TO TRACK

The following block is intentionally invalid

VALIDATE THE RESULT

```
IndentationError: expected
an indented block after 'if'
statement on line 2
```

INDEPENDENT PREDICTION

CODE WALKTHROUGH W02-L01-S12

```
01  stock = 0
02  if stock > 0:
03      print("available")
04  else:
05      print("out of stock")
06  print("inventory checked")
```

WHAT TO TRACK

Predict every output line

Then change stock to 3 and run again

PREDICT THE RESULT

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

PRACTICE SOLUTION

LEARNER PRACTICE

Practice solution

stock = 0
  out of stock
  inventory checked

stock = 3
  available
  inventory checked

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.

A SHORT SELF-CHECK

MENTAL MODEL

A short self-check

A condition is False

Does Python skip the indented block,
the rest of the script, or both?

LESSON RECAP

Recap and next step

A condition selects a path
Indentation identifies its block
else supplies an alternative
Boundary inputs check the intended rule

UNDERSTAND APPLY VALIDATE EXPLAIN

FOUNDATION LESSON

WEEK 2 · LESSON 2

Ordered business rules

if-elif-else chains / Condition ordering and overlapping rules

ORDERED BUSINESS RULES

MENTAL MODEL

Ordered business rules

An if–elif–else chain checks conditions from top to bottom and chooses the first true branch.
Overlapping thresholds make order consequential. Test the exact threshold and just below it, and make the fallback explicit.

SCORE THRESHOLDS

CODE WALKTHROUGH W02-L02-S03

```
01  score = 71
02  if score >= 85:
03      band = "A"
04  elif score >= 70:
05      band = "B"
06  elif score >= 55:
07      band = "C"
08  else:
09      band = "D"
10  print(band)
```

WHAT TO TRACK

Score thresholds

PREDICT THE RESULT

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

SCORE THRESHOLDS

CODE WALKTHROUGH W02-L02-S04

```
01  score = 71
02  if score >= 85:
03      band = "A"
04  elif score >= 70:
05      band = "B"
06  elif score >= 55:
07      band = "C"
08  else:
09      band = "D"
10  print(band)
```

WHAT TO TRACK

Score thresholds

VALIDATE THE RESULT

B

INCOME PRIORITY

CODE WALKTHROUGH W02-L02-S05

```
01  income = 90000
02  if income >= 80000:
03      result = "priority review"
04  elif income >= 40000:
05      result = "standard review"
06  else:
07      result = "additional review"
08  print(result)
```

WHAT TO TRACK

Income priority

PREDICT THE RESULT

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

INCOME PRIORITY

CODE WALKTHROUGH W02-L02-S06

```
01  income = 90000
02  if income >= 80000:
03      result = "priority review"
04  elif income >= 40000:
05      result = "standard review"
06  else:
07      result = "additional review"
08  print(result)
```

WHAT TO TRACK

Income priority

VALIDATE THE RESULT

priority review

BELOW THE BOUNDARY

CODE WALKTHROUGH W02-L02-S07

```
01  income = 39999
02  if income >= 80000:
03      result = "priority review"
04  elif income >= 40000:
05      result = "standard review"
06  else:
07      result = "additional review"
08  print(result)
```

WHAT TO TRACK

Below the boundary

PREDICT THE RESULT

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

BELOW THE BOUNDARY

CODE WALKTHROUGH W02-L02-S08

```
01  income = 39999
02  if income >= 80000:
03      result = "priority review"
04  elif income >= 40000:
05      result = "standard review"
06  else:
07      result = "additional review"
08  print(result)
```

WHAT TO TRACK

Below the boundary

VALIDATE THE RESULT

additional review

INDEPENDENT PREDICTION

LEARNER PRACTICE

Independent prediction

Change one input in ordered business rules and test a boundary or edge case.

01 Predict the exact output.
02 Change only one input.
03 Run in a fresh Python process.
04 Explain any 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.

ORDERED BUSINESS RULES

Recap and next step

An if–elif–else chain checks conditions from top to bottom and chooses the first true branch. Overlapping thresholds make order consequential. Test the exact threshold and just below it, and make the fallback explicit.

UNDERSTAND APPLY VALIDATE EXPLAIN

FOUNDATION LESSON

WEEK 2 · LESSON 3

Validate before processing

Nested decisions and validation before processing

VALIDATE BEFORE PROCESSING

MENTAL MODEL

Validate before processing

Separate invalid inputs from valid inputs that simply fail an eligibility rule.
Nested checks can express dependent conditions.
Printing an error does not end a top-level script. Place later processing inside the valid branch rather than assuming a print acts like return.

VALIDATE AGE AND INCOME

CODE WALKTHROUGH W02-L03-S03

```
01  age = 22
02  income = 30000
03  if age < 0:
04      print("invalid age")
05  elif income < 0:
06      print("invalid income")
07  elif age < 18:
08      print("not eligible: age")
09  elif income < 35000:
10      print("not eligible: income")
11  else:
12      print("eligible")
```

WHAT TO TRACK

Validate age and income

PREDICT THE RESULT

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

VALIDATE AGE AND INCOME

CODE WALKTHROUGH W02-L03-S04

```
01  age = 22
02  income = 30000
03  if age < 0:
04      print("invalid age")
05  elif income < 0:
06      print("invalid income")
07  elif age < 18:
08      print("not eligible: age")
09  elif income < 35000:
10      print("not eligible: income")
11  else:
12      print("eligible")
```

WHAT TO TRACK

Validate age and income

VALIDATE THE RESULT

```
not eligible: income
```

EARLY AGE CHECK

CODE WALKTHROUGH W02-L03-S05

```
01  age = 16
02  if age < 18:
03      print("reject: age")
04  else:
05      print("continue checks")
```

WHAT TO TRACK

Early age check

PREDICT THE RESULT

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

EARLY AGE CHECK

CODE WALKTHROUGH W02-L03-S06

```
01  age = 16
02  if age < 18:
03      print("reject: age")
04  else:
05      print("continue checks")
```

WHAT TO TRACK

Early age check

VALIDATE THE RESULT

```
reject: age
```

INDEPENDENT PREDICTION

LEARNER PRACTICE

Independent prediction

Change one input in validate before processing and test a boundary or edge case.

01 Predict the exact output.
02 Change only one input.
03 Run in a fresh Python process.
04 Explain any 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.

VALIDATE BEFORE PROCESSING

Recap and next step

Separate invalid inputs from valid inputs that simply fail an eligibility rule. Nested checks can express dependent conditions. Printing an error does not end a top-level script. Place later processing inside the valid branch rather than assuming a print acts like return.

UNDERSTAND APPLY VALIDATE EXPLAIN

FOUNDATION LESSON

WEEK 2 · LESSON 4

Membership and clean strings

Membership tests with in and not in / String validation and cleaning

MEMBERSHIP AND CLEAN STRINGS

MENTAL MODEL

Membership and clean strings

Membership checks whether a substring or item appears in a collection. Strip surrounding whitespace and normalise case before comparing text.
String methods return new strings. The original text remains unchanged unless you assign the result.

MEMBERSHIP CHECKS

CODE WALKTHROUGH W02-L04-S03

```
01  status = "PAID"
02  print("AI" in status)
03  print("US" not in status)
04  region = "GR"
05  approved = region == "GR" or region == "CY" or region == "IE"
06  print(approved)
```

WHAT TO TRACK

Membership checks

PREDICT THE RESULT

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

MEMBERSHIP CHECKS

CODE WALKTHROUGH W02-L04-S04

```
01  status = "PAID"
02  print("AI" in status)
03  print("US" not in status)
04  region = "GR"
05  approved = region == "GR" or region == "CY" or region == "IE"
06  print(approved)
```

WHAT TO TRACK

Membership checks

VALIDATE THE RESULT

```
True
True
True
```

CLEAN BEFORE COMPARISON

CODE WALKTHROUGH W02-L04-S05

```
01  raw = "  paid "
02  clean = raw.strip().upper()
03  print(clean)
04  print(clean.endswith("ID"))
05  print(raw)
```

WHAT TO TRACK

Clean before comparison

PREDICT THE RESULT

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

CLEAN BEFORE COMPARISON

CODE WALKTHROUGH W02-L04-S06

```
01  raw = "  paid "
02  clean = raw.strip().upper()
03  print(clean)
04  print(clean.endswith("ID"))
05  print(raw)
```

WHAT TO TRACK

Clean before comparison

VALIDATE THE RESULT

```
PAID
True
  paid
```

INDEPENDENT PREDICTION

LEARNER PRACTICE

Independent prediction

Change one input in membership and clean strings and test a boundary or edge case.

01 Predict the exact output.
02 Change only one input.
03 Run in a fresh Python process.
04 Explain any 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.

MEMBERSHIP AND CLEAN STRINGS

Recap and next step

Membership checks whether a substring or item appears in a collection. Strip surrounding whitespace and normalise case before comparing text. String methods return new strings. The original text remains unchanged unless you assign the result.

UNDERSTAND APPLY VALIDATE EXPLAIN

FOUNDATION LESSON

WEEK 2 · LESSON 5

Missing versus zero

Truthiness, None and empty values / Week 2 integrated practice: eligibility and transaction rules

MISSING VERSUS ZERO

MENTAL MODEL

Missing versus zero

None is a distinct missing-value marker; zero, empty text and False have different meanings despite all being falsey.
Use is None for missing data. A zero numeric balance may be supplied and valid even though bool(0) is False.

EMPTY AND MISSING

CODE WALKTHROUGH W02-L05-S03

```
01  value = ""
02  if value:
03      print("provided")
04  else:
05      print("empty")
06  missing = None
07  print(missing is None)
```

WHAT TO TRACK

Empty and missing

PREDICT THE RESULT

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

EMPTY AND MISSING

CODE WALKTHROUGH W02-L05-S04

```
01  value = ""
02  if value:
03      print("provided")
04  else:
05      print("empty")
06  missing = None
07  print(missing is None)
```

WHAT TO TRACK

Empty and missing

VALIDATE THE RESULT

```
empty
True
```

RULE PRECEDENCE

CODE WALKTHROUGH W02-L05-S05

```
01  amount = 3200
02  region = "GR"
03  if amount is None:
04      result = "missing amount"
05  elif amount <= 0:
06      result = "invalid"
07  elif region == "XX" or region == "YY":
08      result = "manual review"
09  elif amount >= 3000:
10      result = "enhanced check"
11  else:
12      result = "standard"
13  print(result)
```

WHAT TO TRACK

Rule precedence

PREDICT THE RESULT

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

RULE PRECEDENCE

CODE WALKTHROUGH W02-L05-S06

```
amount = 3200
region = "GR"
if amount is None:
    result = "missing amount"
elif amount <= 0:
    result = "invalid"
elif region == "XX" or region == "YY":
    result = "manual review"
elif amount >= 3000:
    result = "enhanced check"
else:
    result = "standard"
print(result)
```

WHAT TO TRACK

Rule precedence

VALIDATE THE RESULT

enhanced check

ZERO IS SUPPLIED

CODE WALKTHROUGH W02-L05-S07

```
01  balance = 0
02  print(bool(balance))
03  print(balance is not None)
```

WHAT TO TRACK

Zero is supplied

PREDICT THE RESULT

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

ZERO IS SUPPLIED

CODE WALKTHROUGH W02-L05-S08

```
01  balance = 0
02  print(bool(balance))
03  print(balance is not None)
```

WHAT TO TRACK

Zero is supplied

VALIDATE THE RESULT

```
False
True
```

INDEPENDENT PREDICTION

LEARNER PRACTICE

Independent prediction

Change one input in missing versus zero and test a boundary or edge case.

01 Predict the exact output.
02 Change only one input.
03 Run in a fresh Python process.
04 Explain any 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.

MISSING VERSUS ZERO

Recap and next step

None is a distinct missing-value marker; zero, empty text and False have different meanings despite all being falsey. Use is None for missing data. A zero numeric balance may be supplied and valid even though bool(0) is False.

UNDERSTAND APPLY VALIDATE EXPLAIN

FOUNDATION LESSON

WEEK 2 · LESSON 6

Lists and safe indexing

Lists: ordered mutable collections / Indexing and negative indexing

LISTS AND SAFE INDEXING

MENTAL MODEL

Lists and safe indexing

A list is an ordered mutable sequence. Indices start at zero, and negative indices count backward from the end.
Indexing an empty list raises IndexError. Check the sequence is non-empty before using the final element.

ORDERED RETURNS

CODE WALKTHROUGH W02-L06-S03

```
01  returns = [0.01, -0.02, 0.015]
02  print(returns)
03  print(len(returns))
```

WHAT TO TRACK

Ordered returns

PREDICT THE RESULT

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

ORDERED RETURNS

CODE WALKTHROUGH W02-L06-S04

```
01  returns = [0.01, -0.02, 0.015]
02  print(returns)
03  print(len(returns))
```

WHAT TO TRACK

Ordered returns

VALIDATE THE RESULT

```
[0.01, -0.02, 0.015]
3
```

FIRST AND LAST

CODE WALKTHROUGH W02-L06-S05

```
01  prices = [100, 102, 99, 105]
02  print(prices[0])
03  print(prices[-1])
```

WHAT TO TRACK

First and last

PREDICT THE RESULT

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

FIRST AND LAST

CODE WALKTHROUGH W02-L06-S06

```
01  prices = [100, 102, 99, 105]
02  print(prices[0])
03  print(prices[-1])
```

WHAT TO TRACK

First and last

VALIDATE THE RESULT

```
100
105
```

EMPTY INDEXING

CODE WALKTHROUGH W02-L06-S07

```
01  values = []
02  print(values[-1])
```

WHAT TO TRACK

Empty indexing

PREDICT THE RESULT

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

EMPTY INDEXING

CODE WALKTHROUGH W02-L06-S08

```
01  values = []
02  print(values[-1])
```

WHAT TO TRACK

Empty indexing

VALIDATE THE RESULT

```
IndexError: list index out
of range
```

INDEPENDENT PREDICTION

LEARNER PRACTICE

Independent prediction

Change one input in lists and safe indexing and test a boundary or edge case.

01 Predict the exact output.
02 Change only one input.
03 Run in a fresh Python process.
04 Explain any 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.

LISTS AND SAFE INDEXING

Recap and next step

A list is an ordered mutable sequence. Indices start at zero, and negative indices count backward from the end. Indexing an empty list raises IndexError. Check the sequence is non-empty before using the final element.

UNDERSTAND APPLY VALIDATE EXPLAIN

FOUNDATION LESSON

WEEK 2 · LESSON 7

Slicing and mutation

Slicing sequences / List methods and mutation

SLICING AND MUTATION

MENTAL MODEL

Slicing and mutation

A slice excludes its stop position. Append adds one object, extend adds multiple elements, and sort mutates a list.
sorted returns a new list; list.sort returns None. Appending a list is different from extending with its contents.

SLICES AND STEPS

CODE WALKTHROUGH W02-L07-S03

```
01  values = [10, 20, 30, 40, 50]
02  print(values[1:4])
03  print(values[::2])
```

WHAT TO TRACK

Slices and steps

PREDICT THE RESULT

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

SLICES AND STEPS

CODE WALKTHROUGH W02-L07-S04

```
01 values = [10, 20, 30, 40, 50]
02 print(values[1:4])
03 print(values[::2])
```

WHAT TO TRACK

Slices and steps

VALIDATE THE RESULT

```
[20, 30, 40]
[10, 30, 50]
```

ADD AND SORT

CODE WALKTHROUGH W02-L07-S05

```
01  transactions = [100, 250]
02  transactions.append(500)
03  transactions.extend([75, 125])
04  transactions.sort()
05  print(transactions)
```

WHAT TO TRACK

Add and sort

PREDICT THE RESULT

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

ADD AND SORT

CODE WALKTHROUGH W02-L07-S06

```
01  transactions = [100, 250]
02  transactions.append(500)
03  transactions.extend([75, 125])
04  transactions.sort()
05  print(transactions)
```

WHAT TO TRACK

Add and sort

VALIDATE THE RESULT

```
[75, 100, 125, 250, 500]
```

APPEND OR EXTEND

CODE WALKTHROUGH W02-L07-S07

```
01  first = [10]
02  first.append([20, 30])
03  second = [10]
04  second.extend([20, 30])
05  print(first)
06  print(second)
```

WHAT TO TRACK

Append or extend

PREDICT THE RESULT

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

APPEND OR EXTEND

CODE WALKTHROUGH W02-L07-S08

```
01  first = [10]
02  first.append([20, 30])
03  second = [10]
04  second.extend([20, 30])
05  print(first)
06  print(second)
```

WHAT TO TRACK

Append or extend

VALIDATE THE RESULT

```
[10, [20, 30]]
[10, 20, 30]
```

SORT OR SORTED

CODE WALKTHROUGH W02-L07-S09

```
01  amounts = [30, 10]
02  ordered = sorted(amounts)
03  print(amounts)
04  result = amounts.sort()
05  print(ordered)
06  print(amounts)
07  print(result)
```

WHAT TO TRACK

Sort or sorted

PREDICT THE RESULT

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

SORT OR SORTED

CODE WALKTHROUGH W02-L07-S10

```
01  amounts = [30, 10]
02  ordered = sorted(amounts)
03  print(amounts)
04  result = amounts.sort()
05  print(ordered)
06  print(amounts)
07  print(result)
```

WHAT TO TRACK

Sort or sorted

VALIDATE THE RESULT

```
[30, 10]
[10, 30]
[10, 30]
None
```

POP AND MUTATION

CODE WALKTHROUGH W02-L07-S11

```
01  values = [10, 20]
02  removed = values.pop()
03  print(removed)
04  print(values)
```

WHAT TO TRACK

Pop and mutation

PREDICT THE RESULT

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

POP AND MUTATION

CODE WALKTHROUGH W02-L07-S12

```
01  values = [10, 20]
02  removed = values.pop()
03  print(removed)
04  print(values)
```

WHAT TO TRACK

Pop and mutation

VALIDATE THE RESULT

```
20
[10]
```

INDEPENDENT PREDICTION

LEARNER PRACTICE

Independent prediction

Change one input in slicing and mutation and test a boundary or edge case.

01 Predict the exact output.
02 Change only one input.
03 Run in a fresh Python process.
04 Explain any 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.

SLICING AND MUTATION

Recap and next step

A slice excludes its stop position. Append adds one object, extend adds multiple elements, and sort mutates a list. sorted returns a new list; list.sort returns None. Appending a list is different from extending with its contents.

UNDERSTAND APPLY VALIDATE EXPLAIN

FOUNDATION LESSON

WEEK 2 · LESSON 8

Tuples and dictionaries

Tuples and immutable records / Dictionary fundamentals

TUPLES AND DICTIONARIES

MENTAL MODEL

Tuples and dictionaries

Tuples group positional fields without top-level reassignment. Dictionaries map keys to values for named fields.
A singleton tuple needs a comma. Dictionary get with a default avoids a missing-key error but does not override a present None.

UNPACK A RECORD

CODE WALKTHROUGH W02-L08-S03

```
01  record = ("AAPL", 10, 190.5)
02  ticker, quantity, price = record
03  print(ticker, quantity, price)
```

WHAT TO TRACK

Unpack a record

PREDICT THE RESULT

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

UNPACK A RECORD

CODE WALKTHROUGH W02-L08-S04

```
01  record = ("AAPL", 10, 190.5)
02  ticker, quantity, price = record
03  print(ticker, quantity, price)
```

WHAT TO TRACK

Unpack a record

VALIDATE THE RESULT

```
AAPL 10 190.5
```

NAMED FIELDS

CODE WALKTHROUGH W02-L08-S05

```
01  customer = {"id": 101, "region": "GR", "income": 42000}
02  print(customer["region"])
03  print(customer.get("debt", 0))
```

WHAT TO TRACK

Named fields

PREDICT THE RESULT

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

NAMED FIELDS

CODE WALKTHROUGH W02-L08-S06

```
01  customer = {"id": 101, "region": "GR", "income": 42000}
02  print(customer["region"])
03  print(customer.get("debt", 0))
```

WHAT TO TRACK

Named fields

VALIDATE THE RESULT

GR
0

SINGLETON TUPLE

CODE WALKTHROUGH W02-L08-S07

```
01  one = (100,)
02  print(type(one))
03  print(len(one))
```

WHAT TO TRACK

Singleton tuple

PREDICT THE RESULT

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

SINGLETON TUPLE

CODE WALKTHROUGH W02-L08-S08

```
01  one = (100,)
02  print(type(one))
03  print(len(one))
```

WHAT TO TRACK

Singleton tuple

VALIDATE THE RESULT

```
<class 'tuple'>
1
```

INDEPENDENT PREDICTION

LEARNER PRACTICE

Independent prediction

Change one input in tuples and dictionaries and test a boundary or edge case.

01 Predict the exact output.
02 Change only one input.
03 Run in a fresh Python process.
04 Explain any 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.

TUPLES AND DICTIONARIES

Recap and next step

Tuples group positional fields without top-level reassignment. Dictionaries map keys to values for named fields. A singleton tuple needs a comma. Dictionary get with a default avoids a missing-key error but does not override a present None.

UNDERSTAND APPLY VALIDATE EXPLAIN

FOUNDATION LESSON

WEEK 2 · LESSON 9

Updates and unique values

Dictionary methods and updates / Sets and uniqueness

UPDATES AND UNIQUE VALUES

MENTAL MODEL

Updates and unique values

Dictionary update adds or replaces key-value pairs. Sets represent distinct members and are useful for membership and deduplication.
Never promise the printed order of a set. Sort for a stable presentation, or compare the members semantically.

UPDATE FIELDS

CODE WALKTHROUGH W02-L09-S03

```
01  customer = {"id": 101, "income": 42000}
02  customer.update({"income": 45000, "active": True})
03  print(customer["id"])
04  print(customer["income"])
05  print(customer["active"])
06  print("debt" in customer)
07  print(customer.get("debt", 0))
```

WHAT TO TRACK

Update fields

PREDICT THE RESULT

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

UPDATE FIELDS

CODE WALKTHROUGH W02-L09-S04

```
01  customer = {"id": 101, "income": 42000}
02  customer.update({"income": 45000, "active": True})
03  print(customer["id"])
04  print(customer["income"])
05  print(customer["active"])
06  print("debt" in customer)
07  print(customer.get("debt", 0))
```

WHAT TO TRACK

Update fields

VALIDATE THE RESULT

```
101
45000
True
False
0
```

UNIQUE IDENTIFIERS

CODE WALKTHROUGH W02-L09-S05

```
01  ids = [1, 2, 2, 3, 3, 3]
02  unique_ids = set(ids)
03  print(sorted(unique_ids))
04  print(len(unique_ids))
```

WHAT TO TRACK

Unique identifiers

PREDICT THE RESULT

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

UNIQUE IDENTIFIERS

CODE WALKTHROUGH W02-L09-S06

```
01  ids = [1, 2, 2, 3, 3, 3]
02  unique_ids = set(ids)
03  print(sorted(unique_ids))
04  print(len(unique_ids))
```

WHAT TO TRACK

Unique identifiers

VALIDATE THE RESULT

```
[1, 2, 3]
3
```

PRESENT NONE

CODE WALKTHROUGH W02-L09-S07

```
01  customer = {"credit": None}
02  print("credit" in customer)
03  print(customer.get("credit", 0))
04  print(customer.get("balance", 0))
```

WHAT TO TRACK

Present None

PREDICT THE RESULT

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

PRESENT NONE

CODE WALKTHROUGH W02-L09-S08

```
01  customer = {"credit": None}
02  print("credit" in customer)
03  print(customer.get("credit", 0))
04  print(customer.get("balance", 0))
```

WHAT TO TRACK

Present None

VALIDATE THE RESULT

```
True
None
0
```

INDEPENDENT PREDICTION

LEARNER PRACTICE

Independent prediction

Change one input in updates and unique values and test a boundary or edge case.

01 Predict the exact output.
02 Change only one input.
03 Run in a fresh Python process.
04 Explain any 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.

UPDATES AND UNIQUE VALUES

Recap and next step

Dictionary update adds or replaces key-value pairs. Sets represent distinct members and are useful for membership and deduplication. Never promise the printed order of a set. Sort for a stable presentation, or compare the members semantically.

UNDERSTAND APPLY VALIDATE EXPLAIN

FOUNDATION LESSON

WEEK 2 · LESSON 10

Choose the right collection

Choosing the right collection / Nested collections

CHOOSE THE RIGHT COLLECTION

MENTAL MODEL

Choose the right collection

Lists preserve order and duplicates; tuples group a fixed record; dictionaries name fields; sets track unique members.
A tuple may contain a mutable nested list. Immutability of the tuple does not freeze the nested object.

COLLECTION SIZES

CODE WALKTHROUGH W02-L10-S03

```
01  ordered = [1, 2, 2]
02  unique = {1, 2, 2}
03  labelled = {"a": 1, "b": 2}
04  print(len(ordered), len(unique), labelled["a"])
```

WHAT TO TRACK

Collection sizes

PREDICT THE RESULT

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

COLLECTION SIZES

CODE WALKTHROUGH W02-L10-S04

```
01  ordered = [1, 2, 2]
02  unique = {1, 2, 2}
03  labelled = {"a": 1, "b": 2}
04  print(len(ordered), len(unique), labelled["a"])
```

WHAT TO TRACK

Collection sizes

VALIDATE THE RESULT

3 2 1

NESTED RECORDS

CODE WALKTHROUGH W02-L10-S05

```
records = [
    {"id": 1, "amount": 100},
    {"id": 2, "amount": 250},
]
print(records[0]["amount"])
print(records[1]["amount"])
total = records[0]["amount"] + records[1]["amount"]
print(total)
```

WHAT TO TRACK

Nested records

PREDICT THE RESULT

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

NESTED RECORDS

CODE WALKTHROUGH W02-L10-S06

```
records = [
    {"id": 1, "amount": 100},
    {"id": 2, "amount": 250},
]
print(records[0]["amount"])
print(records[1]["amount"])
total = records[0]["amount"] + records[1]["amount"]
print(total)
```

WHAT TO TRACK

Nested records

VALIDATE THE RESULT

```
100
250
350
```

NESTED MUTABILITY

CODE WALKTHROUGH W02-L10-S07

```
01  record = ("C001", [100])
02  record[1].append(200)
03  print(record)
```

WHAT TO TRACK

Nested mutability

PREDICT THE RESULT

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

NESTED MUTABILITY

CODE WALKTHROUGH W02-L10-S08

```
01  record = ("C001", [100])
02  record[1].append(200)
03  print(record)
```

WHAT TO TRACK

Nested mutability

VALIDATE THE RESULT

```
('C001', [100, 200])
```

INDEPENDENT PREDICTION

LEARNER PRACTICE

Independent prediction

Change one input in choose the right collection and test a boundary or edge case.

01 Predict the exact output.
02 Change only one input.
03 Run in a fresh Python process.
04 Explain any 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.

CHOOSE THE RIGHT COLLECTION

Recap and next step

Lists preserve order and duplicates; tuples group a fixed record; dictionaries name fields; sets track unique members. A tuple may contain a mutable nested list. Immutability of the tuple does not freeze the nested object.

UNDERSTAND APPLY VALIDATE EXPLAIN

FOUNDATION LESSON

WEEK 2 · LESSON 11

Aliasing and shallow copies

Copying, aliasing and Week 2 record practice

ALIASING AND SHALLOW COPIES

MENTAL MODEL

Aliasing and shallow copies

Assignment may create two names for the same list. A shallow copy creates a different outer list while sharing nested mutable objects.
Distinguish value equality (==) from identity (is). Test nested edits when claiming a copy is independent.

ALIAS OR COPY

CODE WALKTHROUGH W02-L11-S03

```
01  a = [1, 2]
02  b = a
03  c = a.copy()
04  a.append(3)
05  print(b)
06  print(c)
```

WHAT TO TRACK

Alias or copy

PREDICT THE RESULT

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

ALIAS OR COPY

CODE WALKTHROUGH W02-L11-S04

```
01  a = [1, 2]
02  b = a
03  c = a.copy()
04  a.append(3)
05  print(b)
06  print(c)
```

WHAT TO TRACK

Alias or copy

VALIDATE THE RESULT

```
[1, 2, 3]
[1, 2]
```

SHALLOW NESTED COPY

CODE WALKTHROUGH W02-L11-S05

```
01  original = [{"id": 1, "balance": 1200}]
02  outer_copy = original.copy()
03  outer_copy[0]["balance"] = 1500
04  print(original[0]["balance"])
05  print(outer_copy is original)
06  print(outer_copy[0] is original[0])
```

WHAT TO TRACK

Shallow nested copy

PREDICT THE RESULT

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

SHALLOW NESTED COPY

CODE WALKTHROUGH W02-L11-S06

```
01  original = [{"id": 1, "balance": 1200}]
02  outer_copy = original.copy()
03  outer_copy[0]["balance"] = 1500
04  print(original[0]["balance"])
05  print(outer_copy is original)
06  print(outer_copy[0] is original[0])
```

WHAT TO TRACK

Shallow nested copy

VALIDATE THE RESULT

```
1500
False
True
```

EQUALITY VERSUS IDENTITY

CODE WALKTHROUGH W02-L11-S07

```
01  first = [100]
02  second = [100]
03  print(first == second)
04  print(first is second)
```

WHAT TO TRACK

Equality versus identity

PREDICT THE RESULT

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

EQUALITY VERSUS IDENTITY

CODE WALKTHROUGH W02-L11-S08

```
01  first = [100]
02  second = [100]
03  print(first == second)
04  print(first is second)
```

WHAT TO TRACK

Equality versus identity

VALIDATE THE RESULT

```
True
False
```

INDEPENDENT PREDICTION

LEARNER PRACTICE

Independent prediction

Change one input in aliasing and shallow copies and test a boundary or edge case.

01 Predict the exact output.
02 Change only one input.
03 Run in a fresh Python process.
04 Explain any 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.

ALIASING AND SHALLOW COPIES

Recap and next step

Assignment may create two names for the same list. A shallow copy creates a different outer list while sharing nested mutable objects. Distinguish value equality (==) from identity (is). Test nested edits when claiming a copy is independent.

UNDERSTAND APPLY VALIDATE EXPLAIN

FOUNDATION LESSON

WEEK 2 · LESSON 12

Iterate with for and range

for loops over lists / range and counting loops

ITERATE WITH FOR AND RANGE

MENTAL MODEL

Iterate with for and range

A for loop visits the supplied items in order. range generates integers with an excluded stop.
Rebinding a loop variable does not mutate the original list. Build a separate result list when transforming values.

LOOP WITHOUT MUTATION

CODE WALKTHROUGH W02-L12-S03

```
01  values = [100, 200, 300]
02  for value in values:
03      print(value + 10)
04  print(values)
```

WHAT TO TRACK

Loop without mutation

PREDICT THE RESULT

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

LOOP WITHOUT MUTATION

CODE WALKTHROUGH W02-L12-S04

```
01  values = [100, 200, 300]
02  for value in values:
03      print(value + 10)
04  print(values)
```

WHAT TO TRACK

Loop without mutation

VALIDATE THE RESULT

```
110
210
310
[100, 200, 300]
```

RANGE STOP

CODE WALKTHROUGH W02-L12-S05

```
01  for month in range(1, 4):
02      print("Month", month)
```

WHAT TO TRACK

Range stop

PREDICT THE RESULT

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

RANGE STOP

CODE WALKTHROUGH W02-L12-S06

```
01  for month in range(1, 4):
02      print("Month", month)
```

WHAT TO TRACK

Range stop

VALIDATE THE RESULT

```
Month 1
Month 2
Month 3
```

BUILD A NEW LIST

CODE WALKTHROUGH W02-L12-S07

```
01  amounts = [100, -20, 0, 300]
02  positive = []
03  for amount in amounts:
04      if amount > 0:
05          positive.append(amount)
06  print(amounts)
07  print(positive)
```

WHAT TO TRACK

Build a new list

PREDICT THE RESULT

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

BUILD A NEW LIST

CODE WALKTHROUGH W02-L12-S08

```
01  amounts = [100, -20, 0, 300]
02  positive = []
03  for amount in amounts:
04      if amount > 0:
05          positive.append(amount)
06  print(amounts)
07  print(positive)
```

WHAT TO TRACK

Build a new list

VALIDATE THE RESULT

```
[100, -20, 0, 300]
[100, 300]
```

INDEPENDENT PREDICTION

LEARNER PRACTICE

Independent prediction

Change one input in iterate with for and range and test a boundary or edge case.

01 Predict the exact output.
02 Change only one input.
03 Run in a fresh Python process.
04 Explain any 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.

ITERATE WITH FOR AND RANGE

Recap and next step

A for loop visits the supplied items in order. range generates integers with an excluded stop. Rebinding a loop variable does not mutate the original list. Build a separate result list when transforming values.

UNDERSTAND APPLY VALIDATE EXPLAIN

FOUNDATION LESSON

WEEK 2 · LESSON 13

Index and pair sequences

enumerate for position and value / zip for parallel sequences

INDEX AND PAIR SEQUENCES

MENTAL MODEL

Index and pair sequences

enumerate gives a position and an item; zip pairs corresponding items from multiple sequences.
zip normally stops at the shortest sequence. Validate equal lengths before processing when truncation would lose data.

ENUMERATE TICKERS

CODE WALKTHROUGH W02-L13-S03

```
01  tickers = ["AAPL", "MSFT", "NVDA"]
02  for position, ticker in enumerate(tickers, start=1):
03      print(position, ticker)
```

WHAT TO TRACK

Enumerate tickers

PREDICT THE RESULT

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

ENUMERATE TICKERS

CODE WALKTHROUGH W02-L13-S04

```
01  tickers = ["AAPL", "MSFT", "NVDA"]
02  for position, ticker in enumerate(tickers, start=1):
03      print(position, ticker)
```

WHAT TO TRACK

Enumerate tickers

VALIDATE THE RESULT

```
1 AAPL
2 MSFT
3 NVDA
```

ZIP TWO SEQUENCES

CODE WALKTHROUGH W02-L13-S05

```
01  dates = ["Jan", "Feb", "Mar"]
02  sales = [100, 120, 90]
03  for date, value in zip(dates, sales):
04      print(date, value)
```

WHAT TO TRACK

Zip two sequences

PREDICT THE RESULT

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

ZIP TWO SEQUENCES

CODE WALKTHROUGH W02-L13-S06

```
01  dates = ["Jan", "Feb", "Mar"]
02  sales = [100, 120, 90]
03  for date, value in zip(dates, sales):
04      print(date, value)
```

WHAT TO TRACK

Zip two sequences

VALIDATE THE RESULT

```
Jan 100
Feb 120
Mar 90
```

LENGTH VALIDATION

CODE WALKTHROUGH W02-L13-S07

```
01  dates = ["Jan", "Feb", "Mar"]
02  sales = [100, 120]
03  if len(dates) != len(sales):
04      print("cannot process: different lengths")
05  else:
06      for date, value in zip(dates, sales):
07          print(date, value)
```

WHAT TO TRACK

Length validation

PREDICT THE RESULT

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

LENGTH VALIDATION

CODE WALKTHROUGH W02-L13-S08

```
01  dates = ["Jan", "Feb", "Mar"]
02  sales = [100, 120]
03  if len(dates) != len(sales):
04      print("cannot process: different lengths")
05  else:
06      for date, value in zip(dates, sales):
07          print(date, value)
```

WHAT TO TRACK

Length validation

VALIDATE THE RESULT

```
cannot process: different
lengths
```

INDEPENDENT PREDICTION

LEARNER PRACTICE

Independent prediction

Change one input in index and pair sequences and test a boundary or edge case.

01 Predict the exact output.
02 Change only one input.
03 Run in a fresh Python process.
04 Explain any 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.

INDEX AND PAIR SEQUENCES

Recap and next step

enumerate gives a position and an item; zip pairs corresponding items from multiple sequences. zip normally stops at the shortest sequence. Validate equal lengths before processing when truncation would lose data.

UNDERSTAND APPLY VALIDATE EXPLAIN

FOUNDATION LESSON

WEEK 2 · LESSON 14

Accumulate safely

Accumulation patterns / Conditional logic inside loops

ACCUMULATE SAFELY

MENTAL MODEL

Accumulate safely

Initialise totals and counts before the loop; update them for each qualifying item.
Guard operations that need a non-empty sequence.
Initialising a maximum to zero fails for an all-negative list. Seed it from the first item after an empty-data check.

TOTAL AND COUNT

CODE WALKTHROUGH W02-L14-S03

```
01  values = [10, 20, 30, 40]
02  total = 0
03  count = 0
04  for value in values:
05      total += value
06      count += 1
07  print(total, count)
```

WHAT TO TRACK

Total and count

PREDICT THE RESULT

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

TOTAL AND COUNT

CODE WALKTHROUGH W02-L14-S04

```
01  values = [10, 20, 30, 40]
02  total = 0
03  count = 0
04  for value in values:
05      total += value
06      count += 1
07  print(total, count)
```

WHAT TO TRACK

Total and count

VALIDATE THE RESULT

```
100 4
```

INCLUSIVE FLAG RULE

CODE WALKTHROUGH W02-L14-S05

```
01  amounts = [100, 2500, 500, 4000]
02  flagged = 0
03  for amount in amounts:
04      if amount >= 2000:
05          flagged += 1
06  print(flagged)
```

WHAT TO TRACK

Inclusive flag rule

PREDICT THE RESULT

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

INCLUSIVE FLAG RULE

CODE WALKTHROUGH W02-L14-S06

```
01  amounts = [100, 2500, 500, 4000]
02  flagged = 0
03  for amount in amounts:
04      if amount >= 2000:
05          flagged += 1
06  print(flagged)
```

WHAT TO TRACK

Inclusive flag rule

VALIDATE THE RESULT

2

NEGATIVE MAXIMUM

CODE WALKTHROUGH W02-L14-S07

```
01  values = [-5, -2, -9]
02  if values:
03      largest = values[0]
04      for value in values:
05          if value > largest:
06              largest = value
07      print(largest)
08  else:
09      print("no maximum: empty input")
```

WHAT TO TRACK

Negative maximum

PREDICT THE RESULT

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

NEGATIVE MAXIMUM

CODE WALKTHROUGH W02-L14-S08

```
values = [-5, -2, -9]
if values:
    largest = values[0]
    for value in values:
        if value > largest:
            largest = value
    print(largest)
else:
    print("no maximum: empty input")
```

WHAT TO TRACK

Negative maximum

VALIDATE THE RESULT

-2

EMPTY MAXIMUM

CODE WALKTHROUGH W02-L14-S09

```
01  values = []
02  if values:
03      largest = values[0]
04      for value in values:
05          if value > largest:
06              largest = value
07      print(largest)
08  else:
09      print("no maximum: empty input")
```

WHAT TO TRACK

Empty maximum

PREDICT THE RESULT

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

EMPTY MAXIMUM

CODE WALKTHROUGH W02-L14-S10

```
01  values = []
02  if values:
03      largest = values[0]
04      for value in values:
05          if value > largest:
06              largest = value
07      print(largest)
08  else:
09      print("no maximum: empty input")
```

WHAT TO TRACK

Empty maximum

VALIDATE THE RESULT

no maximum: empty input

GROUP TOTALS

CODE WALKTHROUGH W02-L14-S11

```
01  records = [{"route": "GR", "amount": 100}, {"route": "CY", "amo
↪   unt": 50}, {"route": "GR", "amount": 20}]
02  totals = {}
03  for record in records:
04      route = record["route"]
05      totals[route] = totals.get(route, 0) + record["amount"]
06  for route in sorted(totals):
07      print(route, totals[route])
```

WHAT TO TRACK

Group totals

PREDICT THE RESULT

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

GROUP TOTALS

CODE WALKTHROUGH W02-L14-S12

```
records = [{"route": "GR", "amount": 100}, {"route": "CY", "amo
unt": 50}, {"route": "GR", "amount": 20}]
totals = {}
for record in records:
    route = record["route"]
    totals[route] = totals.get(route, 0) + record["amount"]
for route in sorted(totals):
    print(route, totals[route])
```

WHAT TO TRACK

Group totals

VALIDATE THE RESULT

```
CY 50
GR 120
```

INDEPENDENT PREDICTION

LEARNER PRACTICE

Independent prediction

Change one input in accumulate safely and test a boundary or edge case.

01 Predict the exact output.
02 Change only one input.
03 Run in a fresh Python process.
04 Explain any 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.

ACCUMULATE SAFELY

Recap and next step

Initialise totals and counts before the loop; update them for each qualifying item. Guard operations that need a non-empty sequence. Initialising a maximum to zero fails for an all-negative list. Seed it from the first item after an empty-data check.

UNDERSTAND APPLY VALIDATE EXPLAIN

FOUNDATION LESSON

WEEK 2 · LESSON 15

Control and repeat loops

break and continue / while loops

CONTROL AND REPEAT LOOPS

MENTAL MODEL

Control and repeat loops

continue skips the remainder of one iteration; break leaves the nearest loop; while repeats while a condition remains true.
A while loop can execute zero times. Identify which statement makes progress toward its termination condition.

SKIP AND STOP

CODE WALKTHROUGH W02-L15-S03

```
01  values = [10, -1, 20, 0, 30]
02  for value in values:
03      if value < 0:
04          continue
05      if value == 0:
06          break
07      print(value)
```

WHAT TO TRACK

Skip and stop

PREDICT THE RESULT

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

SKIP AND STOP

CODE WALKTHROUGH W02-L15-S04

```
values = [10, -1, 20, 0, 30]
for value in values:
    if value < 0:
        continue
    if value == 0:
        break
    print(value)
```

WHAT TO TRACK

Skip and stop

VALIDATE THE RESULT

```
10
20
```

WHILE TERMINATION

CODE WALKTHROUGH W02-L15-S05

```
01  balance = 1000
02  months = 0
03  while balance > 0:
04      balance -= 250
05      months += 1
06  print(months)
```

WHAT TO TRACK

While termination

PREDICT THE RESULT

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

WHILE TERMINATION

CODE WALKTHROUGH W02-L15-S06

```
01  balance = 1000
02  months = 0
03  while balance > 0:
04      balance -= 250
05      months += 1
06  print(months)
```

WHAT TO TRACK

While termination

VALIDATE THE RESULT

4

INDEPENDENT PREDICTION

LEARNER PRACTICE

Independent prediction

Change one input in control and repeat loops and test a boundary or edge case.

01 Predict the exact output.
02 Change only one input.
03 Run in a fresh Python process.
04 Explain any 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.

CONTROL AND REPEAT LOOPS

Recap and next step

continue skips the remainder of one iteration; break leaves the nearest loop; while repeats while a condition remains true. A while loop can execute zero times. Identify which statement makes progress toward its termination condition.

UNDERSTAND APPLY VALIDATE EXPLAIN

FOUNDATION LESSON

WEEK 2 · LESSON 16

Terminate and comprehend

Avoiding infinite loops / List comprehensions

TERMINATE AND COMPREHEND

MENTAL MODEL

Terminate and comprehend

A bounded while loop needs a changing condition and a clear stopping explanation.
List comprehensions express a transformation and optional filter.
Explain the equivalent explicit loop first. A short comprehension is not necessarily better if it obscures validation.

BOUNDED REPETITION

CODE WALKTHROUGH W02-L16-S03

```
01  attempts = 0
02  value = 1
03  while value < 10 and attempts < 20:
04      value *= 2
05      attempts += 1
06  if value >= 10:
07      print("target reached")
08  else:
09      print("attempt limit reached")
10  print(value, attempts)
```

WHAT TO TRACK

Bounded repetition

PREDICT THE RESULT

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

BOUNDED REPETITION

CODE WALKTHROUGH W02-L16-S04

```
01 attempts = 0
02 value = 1
03 while value < 10 and attempts < 20:
04     value *= 2
05     attempts += 1
06 if value >= 10:
07     print("target reached")
08 else:
09     print("attempt limit reached")
10 print(value, attempts)
```

WHAT TO TRACK

Bounded repetition

VALIDATE THE RESULT

```
target reached
16 4
```

FILTER AND TRANSFORM

CODE WALKTHROUGH W02-L16-S05

```
01  amounts = [100, -20, 300]
02  positive_doubled = [amount * 2 for amount in amounts if amount
↪   > 0]
03  print(positive_doubled)
```

WHAT TO TRACK

Filter and transform

PREDICT THE RESULT

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

FILTER AND TRANSFORM

CODE WALKTHROUGH W02-L16-S06

```
01  amounts = [100, -20, 300]
02  positive_doubled = [amount * 2 for amount in amounts if amount
↪   > 0]
03  print(positive_doubled)
```

WHAT TO TRACK

Filter and transform

VALIDATE THE RESULT

[200, 600]

EQUIVALENT LOOP

CODE WALKTHROUGH W02-L16-S07

```
01  amounts = [100, -20, 300]
02  positive_doubled = []
03  for amount in amounts:
04      if amount > 0:
05          positive_doubled.append(amount * 2)
06  print(positive_doubled)
```

WHAT TO TRACK

Equivalent loop

PREDICT THE RESULT

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

EQUIVALENT LOOP

CODE WALKTHROUGH W02-L16-S08

```
01  amounts = [100, -20, 300]
02  positive_doubled = []
03  for amount in amounts:
04      if amount > 0:
05          positive_doubled.append(amount * 2)
06  print(positive_doubled)
```

WHAT TO TRACK

Equivalent loop

VALIDATE THE RESULT

[200, 600]

INDEPENDENT PREDICTION

LEARNER PRACTICE

Independent prediction

Change one input in terminate and comprehend and test a boundary or edge case.

01 Predict the exact output.
02 Change only one input.
03 Run in a fresh Python process.
04 Explain any 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.

TERMINATE AND COMPREHEND

Recap and next step

A bounded while loop needs a changing condition and a clear stopping explanation. List comprehensions express a transformation and optional filter. Explain the equivalent explicit loop first. A short comprehension is not necessarily better if it obscures validation.

UNDERSTAND APPLY VALIDATE EXPLAIN

FOUNDATION LESSON

WEEK 2 · LESSON 17

Batch processing and comprehensions

Dictionary and set comprehensions / Week 2 batch transaction processing

BATCH PROCESSING AND COMPREHENSIONS

MENTAL MODEL

Batch processing and comprehensions

Set comprehensions deduplicate; dictionary comprehensions create key-value mappings.
Combine validation, iteration and accumulation in a small transaction batch.
Do not rely on set iteration order. Exclude invalid amounts before adding to a total
and count flagged transactions with the intended boundary.

CLEAN UNIQUE NAMES

CODE WALKTHROUGH W02-L17-S03

```
01  names = [" alice ", "BOB", "Alice"]
02  clean = {name.strip().title() for name in names}
03  lookup = {name: len(name) for name in sorted(clean)}
04  print(sorted(clean))
05  print(lookup)
```

WHAT TO TRACK

Clean unique names

PREDICT THE RESULT

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

CLEAN UNIQUE NAMES

CODE WALKTHROUGH W02-L17-S04

```
01  names = [" alice ", "BOB", "Alice"]
02  clean = {name.strip().title() for name in names}
03  lookup = {name: len(name) for name in sorted(clean)}
04  print(sorted(clean))
05  print(lookup)
```

WHAT TO TRACK

Clean unique names

VALIDATE THE RESULT

```
['Alice', 'Bob']
{'Alice': 5, 'Bob': 3}
```

VALIDATE BATCH TOTALS

CODE WALKTHROUGH W02-L17-S05

```
transactions = [120, 2500, 0, -5, 75, 3200]
valid = []
flagged = 0
total = 0
for amount in transactions:
    if amount <= 0:
        continue
    valid.append(amount)
    total += amount
    if amount >= 2000:
        flagged += 1
print(valid)
print(total, flagged)
```

WHAT TO TRACK

Validate batch totals

PREDICT THE RESULT

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

VALIDATE BATCH TOTALS

CODE WALKTHROUGH W02-L17-S06

```
transactions = [120, 2500, 0, -5, 75, 3200]
valid = []
flagged = 0
total = 0
for amount in transactions:
    if amount <= 0:
        continue
    valid.append(amount)
    total += amount
    if amount >= 2000:
        flagged += 1
print(valid)
print(total, flagged)
```

WHAT TO TRACK

Validate batch totals

VALIDATE THE RESULT

```
[120, 2500, 75, 3200]
5895 2
```

SEMANTIC SET CHECK

CODE WALKTHROUGH W02-L17-S07

```
01  names = [" alice ", "BOB", "Alice"]
02  clean = {name.strip().title() for name in names}
03  lookup = {name: len(name) for name in sorted(clean)}
04  print(clean == {"Alice", "Bob"})
05  print(lookup == {"Alice": 5, "Bob": 3})
```

WHAT TO TRACK

Semantic set check

PREDICT THE RESULT

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

SEMANTIC SET CHECK

CODE WALKTHROUGH W02-L17-S08

```
01  names = [" alice ", "BOB", "Alice"]
02  clean = {name.strip().title() for name in names}
03  lookup = {name: len(name) for name in sorted(clean)}
04  print(clean == {"Alice", "Bob"})
05  print(lookup == {"Alice": 5, "Bob": 3})
```

WHAT TO TRACK

Semantic set check

VALIDATE THE RESULT

```
True
True
```

INDEPENDENT PREDICTION

LEARNER PRACTICE

Independent prediction

Change one input in batch processing and comprehensions and test a boundary or edge case.

01 Predict the exact output.
02 Change only one input.
03 Run in a fresh Python process.
04 Explain any 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.

BATCH PROCESSING AND COMPREHENSIONS

Recap and next step

Set comprehensions deduplicate; dictionary comprehensions create key-value mappings. Combine validation, iteration and accumulation in a small transaction batch. Do not rely on set iteration order. Exclude invalid amounts before adding to a total and count flagged transactions with the intended boundary.

UNDERSTAND APPLY VALIDATE EXPLAIN