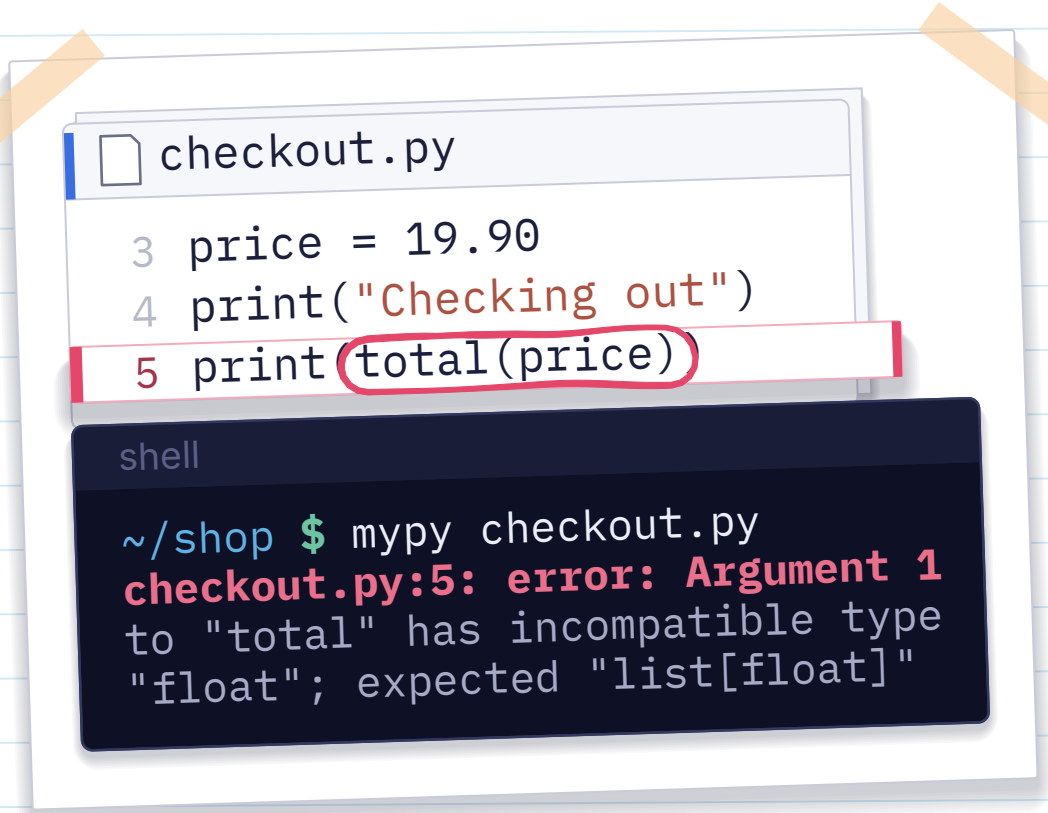
Prof. Aviv Zohar

Semester A, 2026–7

Type checking as early feedback

What a checker reports, how to run it, how it helps an agent, and what it misses



The same mistake, found while it runs or before

billing.py

```python
def total(prices: list[float]) -> float:
    return sum(prices)
```

checkout.py

```python
from billing import total

price = 19.90
print("Checking out")
print(total(price))
```

One price, where total expects a list of prices.

before it runs	while it runs
nothing has checked the code yet	*not run yet*

The same mistake, found while it runs or before

billing.py

```
1 def total(prices: list[float]) -> float:
2     return sum(prices)
```

Python checks types as each line runs.

checkout.py

```
1 from billing import total
⋮
3 price = 19.90
4 print("Checking out")
5 print(total(price))
```

before it runs

nothing has checked the code yet

while it runs

shell

```
~/shop $ python checkout.py
Checking out
Traceback (most recent call last):
⋮
TypeError: 'float' object is not iterable
```

It starts, then fails when line 5 runs.

The same mistake, found while it runs or before

billing.py

```python
def total(prices: list[float]) -> float:
    return sum(prices)
```

mypy reads the `list[float]`. Python ignores it.

checkout.py

```python
from billing import total

price = 19.90
print("Checking out")
print(total(price))
```

before it runs → while it runs

shell

```
~/shop $ mypy checkout.py
checkout.py:5: error: Argument 1 to
"total" has incompatible type
"float"; expected "list[float]"
```

Nothing has run yet.

shell

```
~/shop $ python checkout.py
Checking out
Traceback (most recent call last):
⋮
TypeError: 'float' object is not iterable
```

The same mistake, found while it runs or before

billing.py

```python
def total(prices: list[float]) -> float:
    return sum(prices)
```

checkout.py

```python
from billing import total

price = 19.90
print("Checking out")
print(total(price))
```

Run a checker and the mistake turns up before the program does.

before it runs → **while it runs**

shell

```
~/shop $ mypy checkout.py
checkout.py:5: error: Argument 1 to
"total" has incompatible type
"float"; expected "list[float]"
```

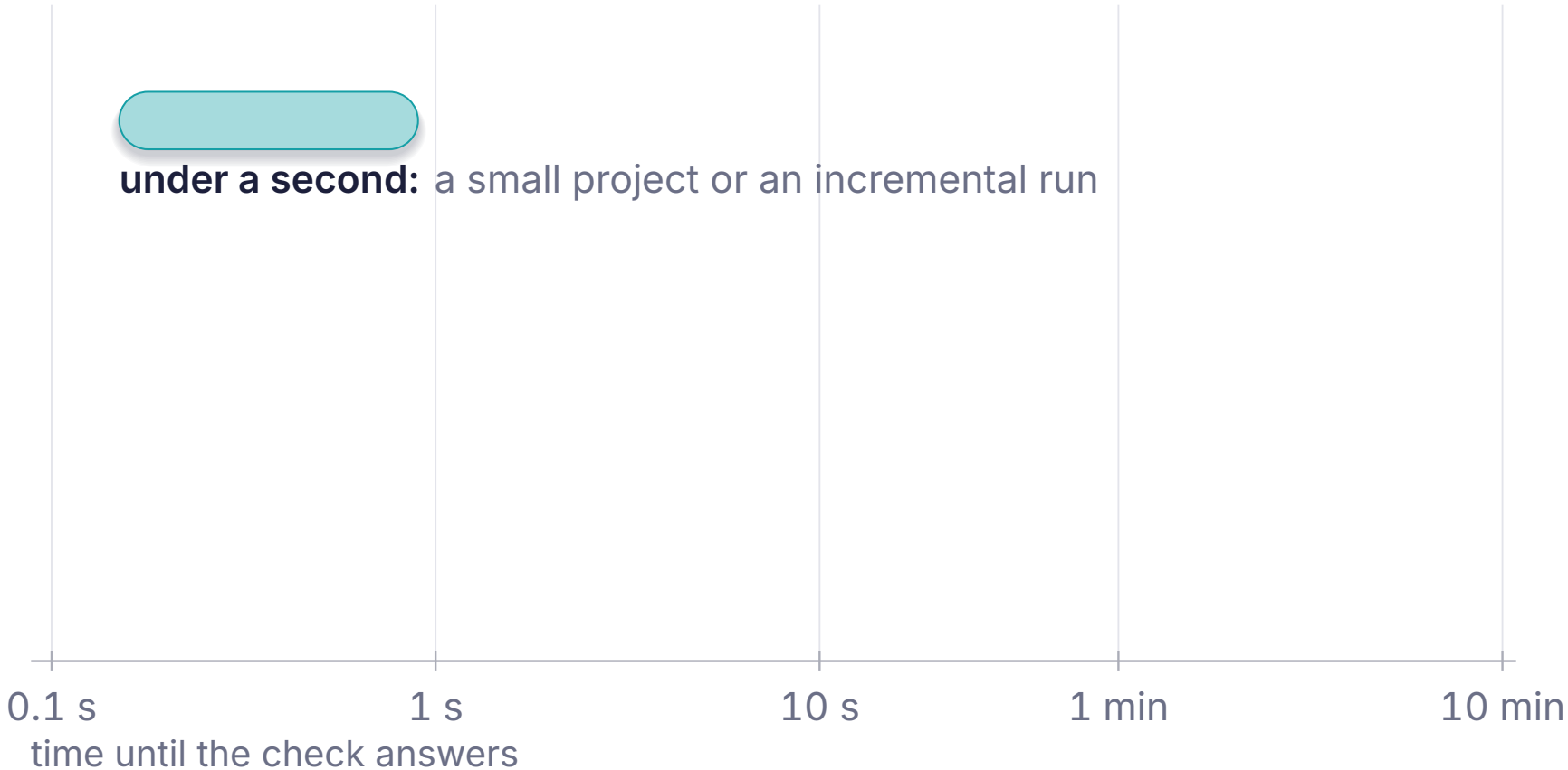
shell

```
~/shop $ python checkout.py
Checking out
Traceback (most recent call last):
⋮
TypeError: 'float' object is not iterable
```

How long each check takes to answer

type checker

Reads the code.
Runs nothing.



illustrative values

How long each check takes to answer

type checker

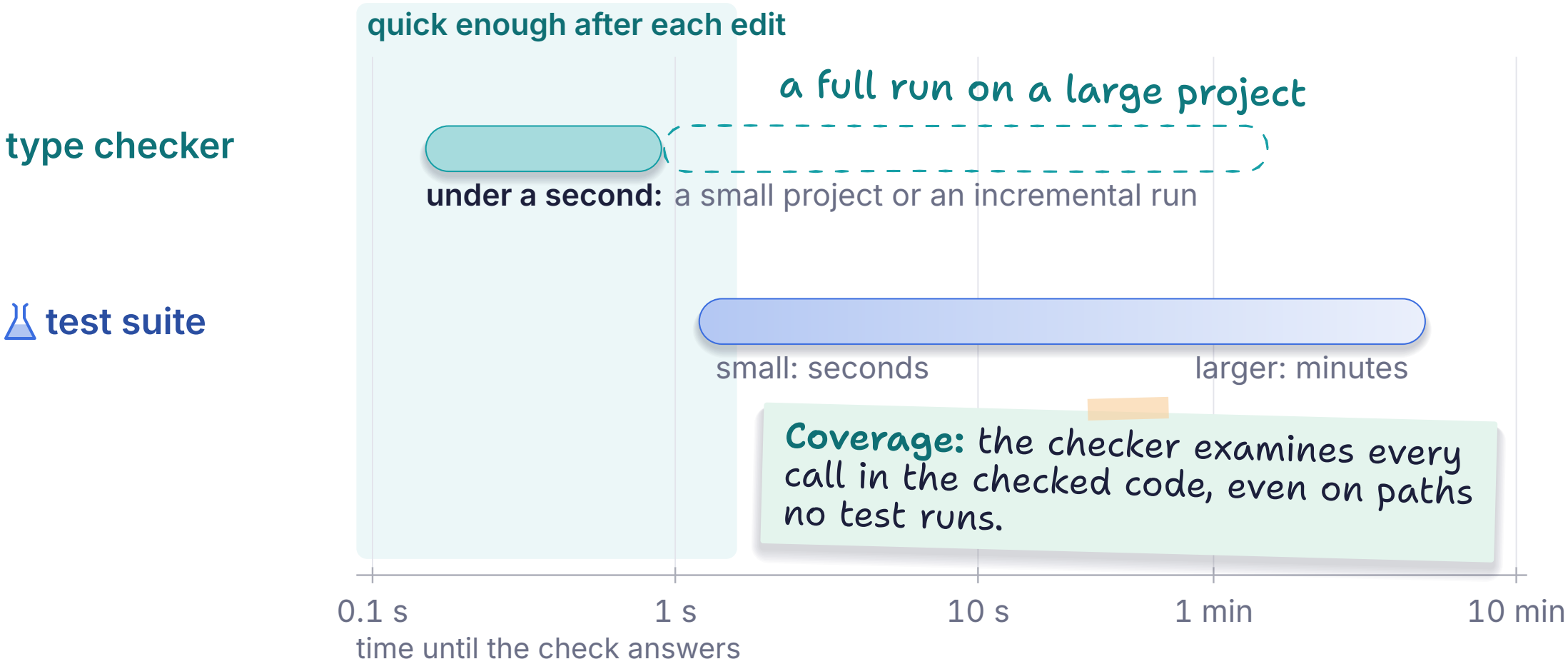
test suite

Executes the code.



illustrative values

How long each check takes to answer



illustrative values

Python's annotations are read by a separate checker

billing.py

```
1 def total(prices: list[float]) -> float:
2     return sum(prices)
```

A type annotation: `prices` is a list of floats.

one program

- billing.py — annotated
- checkout.py — annotated
- report.py — no annotations

Python's annotations are read by a separate checker

billing.py

```
1 def total(prices: list[float]) -> float:
2     return sum(prices)
```

one program

- billing.py — annotated
- checkout.py — annotated
- report.py — no annotations

Python runs it follows the value

```
total(price)
    ↓ 19.90
def total(prices: list[float])        ?
    ↓ 19.90
    return sum(prices)                ✗
```

? The annotation is not enforced: the number gets in.

✗ `sum` fails on it: `TypeError`.

Python's annotations are read by a separate checker

billing.py

```
1 def total(prices: list[float]) -> float:
2     return sum(prices)
```

one program

- billing.py — annotated
- checkout.py — annotated
- report.py — no annotations

Python runs it follows the value

```
total(price)
```

19.90

```
def total(prices: list[float])
```

?

19.90

```
    return sum(prices)
```

✖

mypy reads it follows the type

```
price = 19.90
```

float

```
total(price)
```

float

```
def total(prices: list[float])
```

✖

float is not list[float], and nothing ran.

Start checking one file at a time

report.py — no annotations

```python
from billing import total

def daily_report():
    print(total(19.90))
```

ⓐ default: the body is skipped

ⓑ strict: annotation missing

ⓒ strict: the body is checked

shell

```
~/shop $ mypy report.py
Success: no issues found in 1 source file      ⓐ
~/shop $ mypy --strict report.py
report.py:4: error: Function is missing a      ⓑ
return type annotation
report.py:5: error: Argument 1 to "total"      ⓒ
has incompatible type "float";
expected "list[float]"
```

TypeScript

billing.ts

```typescript
function total(prices: number[]): number {
```

tsc →

billing.js

```javascript
function total(prices) {
```

The types are checked, then removed.

Run the checker from the shell

Python

```shell
~/shop $ uv add --dev mypy                    ⓐ
~/shop $ mypy --strict .                       ⓑ
```

ⓐ Install it in the project first, as a development dependency.

ⓑ `--strict` suits a project that is already annotated. The dot : every Python file under this folder.

TypeScript

Run the checker from the shell

Python

```shell
~/shop $ uv add --dev mypy
~/shop $ mypy --strict .
```

TypeScript

```shell
~/web $ npx tsc --noEmit
```

tsconfig.json

```
1 {
2   "compilerOptions": {
3     "strict": true
```

`--noEmit`: check the project, write no JavaScript. How strict is set in the config.

Run the checker from the shell

Python

shell

```
~/shop $ uv add --dev mypy
~/shop $ mypy --strict .
checkout.py:5: error: Argument 1 to "total"
has incompatible type "float";
expected "list[float]"
Found 1 error in 1 file (checked 4 source files)
```

TypeScript

shell

```
~/web $ npx tsc --noEmit
```

tsconfig.json

```
{
  "compilerOptions": {
    "strict": true
  }
}
```

checkout.py:5: error: Argument 1 to "total"
file line which argument

has incompatible type "float";
the type it had

expected "list[float]"
what would fit

Run the checker from the shell

Python

shell

```
~/shop $ uv add --dev mypy
~/shop $ mypy --strict .
checkout.py:5: error: Argument 1 to "total"
has incompatible type "float";
expected "list[float]"
Found 1 error in 1 file (checked 4 source files)
```

TypeScript

shell

```
~/web $ npx tsc --noEmit
```

tsconfig.json

```
{
  "compilerOptions": {
    "strict": true
```

Neither command runs the application, so untested paths are checked too.

The checker tells the agent which calls to update

billing.py — edited by the agent

```
- def total(prices: list[float]) -> float:
+ def total(prices: list[float], tax_rate: float) -> float:
```

the agent has not run the checker yet

cart.py

```
5       return total(items)
```

invoice.py

```
5       return total(prices)
```

A new parameter. Each call that passes only the prices no longer matches.

The checker tells the agent which calls to update

billing.py — edited by the agent

```
- def total(prices: list[float]) -> float:
+ def total(prices: list[float], tax_rate: float) -> float:
```

agent session ~/shop

tool call asked by the model

```
Bash mypy --strict .
```

tool result run by the harness

```
invoice.py:5: error: Missing positional
argument "tax_rate" in call to "total"
cart.py:5: error: Missing positional
argument "tax_rate" in call to "total"
Found 2 errors in 2 files
```

ⓐ ⓑ

cart.py

```
5      return total(items)
```

ⓑ

invoice.py

```
5      return total(prices)
```

ⓐ

Each message names the file, the line and the missing argument.

The checker tells the agent which calls to update

billing.py — edited by the agent

```
- def total(prices: list[float]) -> float:
+ def total(prices: list[float], tax_rate: float) -> float:
```

agent session ~/shop

tool call asked by the model

```
Edit cart.py
```

tool call asked by the model

```
Bash mypy --strict .
```

tool result run by the harness

```
invoice.py:5: error: Missing positional
argument "tax_rate" in call to "total"
Found 1 error in 1 file
```

@

cart.py

```
5      return total(items, 0.17)
```

✓

invoice.py

```
5      return total(prices)
```

@

One error left: the caller that still needs the same change.

The checker tells the agent which calls to update

billing.py — edited by the agent

```
- def total(prices: list[float]) -> float:
+ def total(prices: list[float], tax_rate: float) -> float:
```

agent session ~/shop

tool call asked by the model

Edit cart.py

tool call asked by the model

Bash mypy --strict .

tool result run by the harness

```
invoice.py:5: error: Missing positional
argument "tax_rate" in call to "total"
Found 1 error in 1 file
```

cart.py

```
5      return total(items, 0.17)
```

invoice.py

```
5      return total(prices)
```

Without static checking, this call waits for a test or a user to run it. If nothing runs it, it stays hidden.

A signature is documentation the checker reads

described in a docstring

billing.py

```
1 def total(prices):
2     """prices: list of floats"""
3     return sum(prices)
```

Prose. Nothing checks it when the function changes.

described in the signature

A signature is documentation the checker reads

described in a docstring

billing.py

```
def total(prices):
    """prices: list of floats"""
    return sum(prices)
```

described in the signature

billing.py

```
def total(prices: list[float]) -> float:
    return sum(prices)
```

The same fact, in a form the checker uses.

A signature is documentation the checker reads

described in a docstring

billing.py

```
1 def total(prices):
2     """prices: list of floats"""
3     return f"{sum(prices):.2f}"
```

cart.py

```
5     print(total(prices) * 1.17)
```

no message at all

described in the signature

billing.py

```
1 def total(prices: list[float]) -> str:
2     return f"{sum(prices):.2f}"
```

cart.py

```
5     print(total(prices) * 1.17
```

shell

```
~/shop $ mypy --strict .
cart.py:5: error: Unsupported operand
types for * ("str" and "float")
```

A signature is documentation the checker reads

described in a docstring

billing.py

```
1 def total(prices):
2     """prices: list of floats"""
3     return f"{sum(prices):.2f}"
```

cart.py

```
5     print(total(prices) * 1.17)
```

no message at all

described in the signature

billing.py

```
1 def total(prices: list[float]) -> str:
2     return f"{sum(prices):.2f}"
```

cart.py

```
5     print(total(prices) * 1.17)
```

shell

```
~/shop $ mypy --strict .
cart.py:5: error: Unsupported operand
types for * ("str" and "float")
```

A signature is documentation that every type-checking run checks.

A new type can say what a value means

xy.py

```
1  from typing import NewType
3  X = NewType("X", float)
4  Y = NewType("Y", float)
7  def move_to(x: X, y: Y) -> None: ...
10 x, y = X(3.0), Y(4.0)
11 move_to(y, x)
```

Two new types. At run time each is still an ordinary `float`. To the checker they differ.

y where x belongs: reported, though both are floats.

shell

```
~/shop $ mypy --strict xy.py
xy.py:11: error: Argument 1 to
"move_to" has incompatible type "Y";
expected "X"
```

The same for IDs: a customer ID and an order ID, both `int`.

Type names can use the words of the domain

board.py

```
4 type Square = tuple[int, int] ⓐ
5 type Board = dict[Square, Piece] ⓑ
  ⋮
8 def piece_at(board: Board, square: Square) -> Piece | None: ...
```

ⓐ a square: a pair of numbers

ⓑ a board: squares to pieces

| a union: a `Piece` or `None`

board.py — the same signature, without the names

```
def piece_at(
    board: dict[tuple[int, int], Piece], square: tuple[int, int]
) -> Piece | None: ...
```

A reader has to work out that this is a chess board.

Type names can use the words of the domain

board.py

```
4 type Square = tuple[int, int]
5 type Board = dict[Square, Piece]
  ⋮
8 def piece_at(board: Board, square: Square) -> Piece | None: ...
```

Named types record intent. They tell a reader, and the agent, what each value stands for, and the checker keeps the code to it.

board.py — the same signature, without the names

```
def piece_at(
    board: dict[tuple[int, int], Piece], square: tuple[int, int]
) -> Piece | None: ...
```

Type feedback for code a model writes

Compile errors in TypeScript written by models

94%

were type errors, on average, in code written in one go

An agent that runs the checker sees these errors and repairs them before you do.

Mündler et al., 2025 · GitHub Octoverse 2025

Type feedback for code a model writes

Compile errors in TypeScript written by models

94%

were type errors, on average, in code written in one go

Start in a typed language

TypeScript

GitHub's most-used language by monthly contributors, August 2025

It measures popularity, not the reasons.

Mündler et al., 2025 · GitHub Octoverse 2025

Type feedback for code a model writes

Compile errors in TypeScript written by models

94%

were type errors, on average, in code written in one go

Start in a typed language

TypeScript

GitHub's most-used language by monthly contributors, August 2025

The checker gives concrete feedback, with no second model to judge the code.

Mündler et al., 2025 · GitHub Octoverse 2025

A wrong answer can still type-check

pricing.py

```python
def price_after_discount(price: float, pct: float) -> float:
    """Return the final price after a pct percent discount."""
    return price * pct / 100
```

the contract

the code

try it

`price_after_discount(100, 20)` → should be **80**

A wrong answer can still type-check

pricing.py

```python
def price_after_discount(price: float, pct: float) -> float:
    """Return the final price after a pct percent discount."""
    return price * pct / 100
```

try it

`price_after_discount(100, 20)` → should be **80** returns **20.0**

the discount, not the price after it

shell

```
~/shop $ mypy --strict pricing.py
Success: no issues found in 1 source file
```

types check

Floats in, a float out. Every caller gets the wrong number.

A wrong answer can still type-check

pricing.py

```python
def price_after_discount(price: float, pct: float) -> float:
    """Return the final price after a pct percent discount."""
    return price * pct / 100
```

try it

`price_after_discount(100, 20)` → should be **80** returns **20.0**

the discount, not the price after it

shell

```
~/shop $ mypy --strict pricing.py
Success: no issues found in 1 source file
```

types check

`assert price_after_discount(100, 20) == 80` fails

Types: values used compatibly. **Tests:** the right value.

Let the type checker take the first pass

1 · the type checker

Catches the mix-ups, in seconds, on every call it checks.



2 · the tests

Check that the answers are right.

See you in the next video.