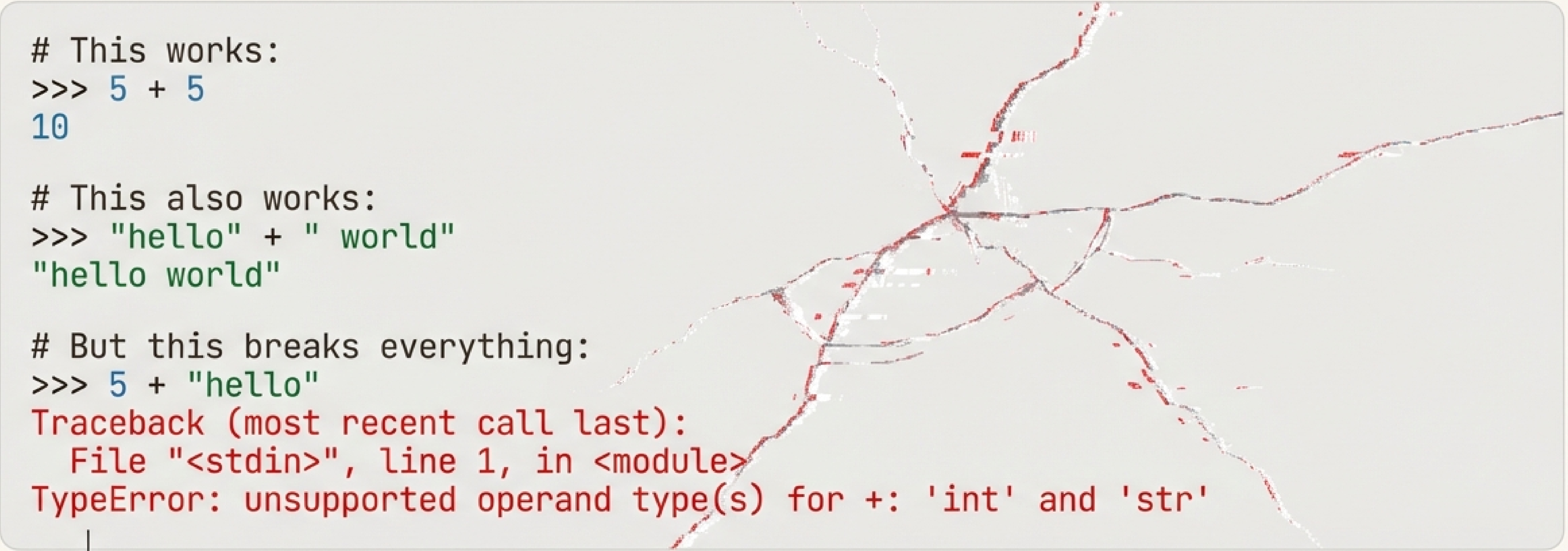
From Chaos to Clarity: Mastering Python’s Data Types

A visual guide to understanding what data types are, why they matter, and how to choose the right one every time.

Why does this code crash?



```
# This works:
>>> 5 + 5
10

# This also works:
>>> "hello" + " world"
"hello world"

# But this breaks everything:
>>> 5 + "hello"
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

Python stopped because it hit a fundamental rule: **Types determine what operations are valid.** You can't add a number to a word because it doesn't make sense.

To prevent this chaos, Python uses a classification system. Let's build the clarity you need.

A Data Type is Python’s Classification System

A data type tells Python two things: **“What kind of information is this?”** and **“What can I do with it?”**

You don’t throw flour, sugar, and salt into one jar. You label them because they behave differently.

Python does the same with data. It classifies data into categories, and each category has its own rules.

Types remove confusion. When Python knows an `age` is a number and an `email` is text, it knows you can do math with the age but not the email.

The Core Building Blocks: Numbers for Counting and Measuring

`int`

Whole numbers. No decimal points.
Positive, negative, or zero.

For things you count.

```
age = 25
student_count = 30
items_in_stock = 150
```

`float`

Numbers with a decimal point.
Represents fractions.

For things you measure.

```
price = 19.99
weight_kg = 2.5
temperature = 37.5
```

Ask yourself: **‘Does this value need a decimal point?’** If no, use `int`. If yes, use `float`.

Representing Text, Decisions, and a Fundamental Rule

str

A str is a sequence of characters—letters, numbers, and symbols. It's how Python stores any text data.

```
name: str = "Alice"
email: str = "alice@example.com"
message: str = "Your total is $19.99"
```

bool

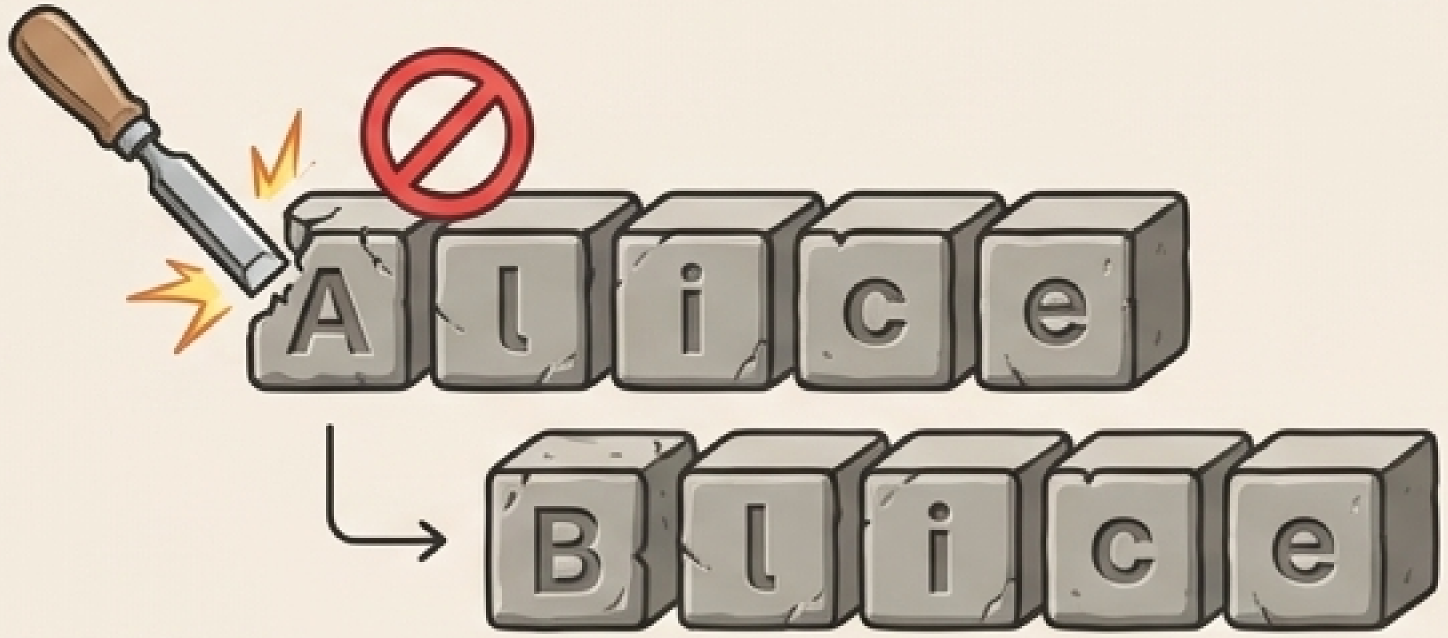
A bool has exactly two values: True or False (capitalization matters!). It's for any yes/no decision.

```
is_logged_in: bool = True
has_paid: bool = False
```

The Rule of Immutability

Immutable means **unchangeable.** Once a string is created, you cannot modify its individual characters.

```
name = "Alice"
# This will cause an error!
# name[0] = "B"

# To change it, you must create a new string:
new_name = "B" + name[1:] # "Blice"
```



It prevents accidental changes and makes your data predictable.

The Power of Nothing: Understanding `None`

`None` is a special value representing the **absence of a value**. It doesn't mean zero, and it doesn't mean an empty string.

int

`score = 0`

int

The score is zero. A value exists.

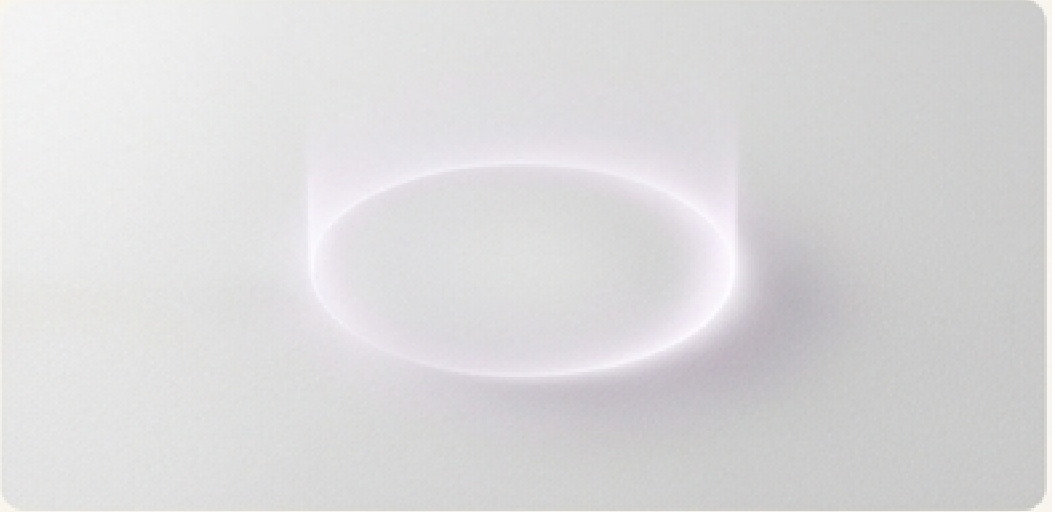
str

`message = ""`

str

The message is empty. A value exists.

None



`phone_number = None`

NoneType

There is no phone number. No value was provided.

When to Use `None`

- For optional data that might not be provided (`middle_name: str | None = None`).
- As a placeholder for a value you'll calculate later.
- When a function has no result to return.

Organizing Your Data: An Introduction to Collections

Collections are containers for grouping related data. The type you choose depends on the job.

list

A Shopping List

An **ordered** collection of items that you **can change**. Use when order matters and the contents might grow or shrink.

```
scores = [85, 90, 88]
```

tuple

A Sealed Envelope

An **ordered** collection of items that you **cannot change** (immutable). Use for fixed data, like coordinates.

```
rgb_color = (255, 99, 71)
```

dict (Dictionary)

A Phone Book

Stores **key-value pairs**. Look up values by a unique key, not by position.

```
user = {"name": "Alice", "age": 25}
```

set

A Collection of Unique Badges

An **unordered** collection of **unique** items. Duplicates are automatically removed.

```
tags = {"python", "data", "code"}
```

A deep dive into collection methods is coming in Chapter 20. For now, focus on choosing the right container for the job.

The Collection Decision Guide

Need	Collection	Why	Example
☰	**list**	Matter is manchment lɪver that shopping_ cart in coltient	`shopping_cart: list[str]`
✉	**tuple**	Sealer is some froat and seen order, aleraments, tuples	`coordinates: tuple[float, float]`
🔑	**dict**	Key-value pair unformed matent, items, and unique line	`phone_book: dict[str, str]`
◇	**set**	Unique_user_ids, set-bostent, and unique items	`unique_user_ids: set[int]`
→→	**range**	Sequence symptiom, with periameted sequences	`for i in range(10):`

Before you store multiple items, always ask: **Does order matter? Can it change? Do I need unique items? How will I look them up?**

Your Type Detective Toolkit: Inspecting Data

Python gives you tools to inspect and verify the type of any variable.

Tool 1: type(value)

Tells you the exact type of a value.

`type(42)` returns `<class 'int'>`

Good for simple inspection, but isinstance() is often better for checking.

Tool 2: isinstance(value, type)

The preferred way to check if a value is of a certain type. Returns True or False.

`isinstance(19.99, float)` returns `True`

Powerful You can check for multiple types at once: `isinstance(x, (int, float))` checks if x is any kind of number.

Tool 3: id(value)

Returns the unique memory address of an object.

`id(None)` will always be the same number in your program.

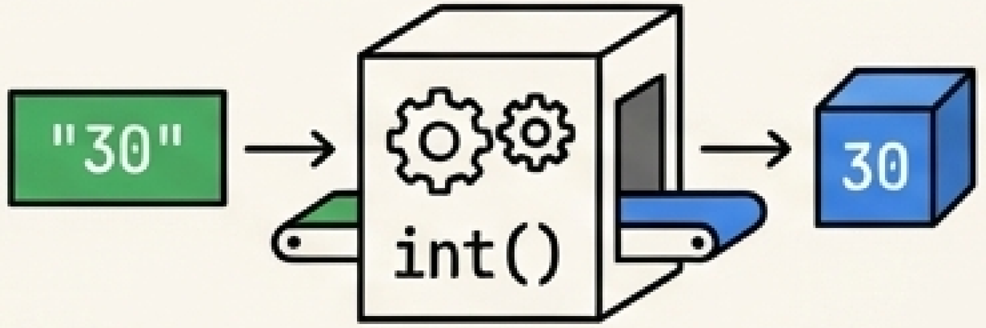
Advanced; used to understand if two variables point to the exact same object in memory.

Changing Hats: How to Convert Between Types

Converting data from one type to another.

Explicit Casting
(You are in control)

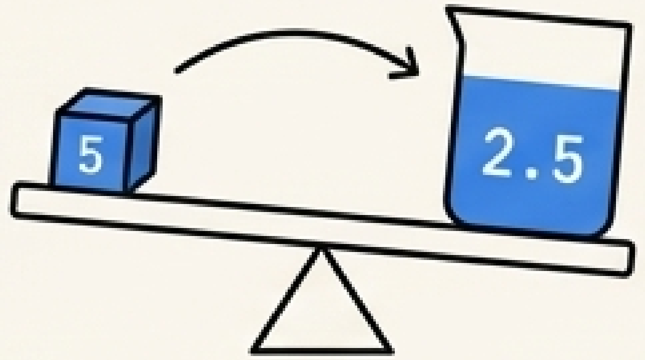
You use functions like `int()`, `float()`, `str()` to intentionally convert a value.



```
age_str = "30"
age_int = int(age_str)  # "30" becomes 30

price_float = 19.99
price_str = str(price_float) # 19.99 becomes "19.99"
```

Implicit Casting
(Python does it automatically)

Happens automatically in mixed-type math. Python promotes the “simpler” type to the “more complex” one to avoid losing data.



```
result = 5 + 2.5 # 5 (int) is implicitly converted to 5.0 (float)
# result is 7.5 (float)
```

Beware of Lossy Conversions!

When you cast a `float` to an `int`, the decimal part is **truncated** (chopped off), not rounded.

```
int(19.99) → 19
```

You lose the .99

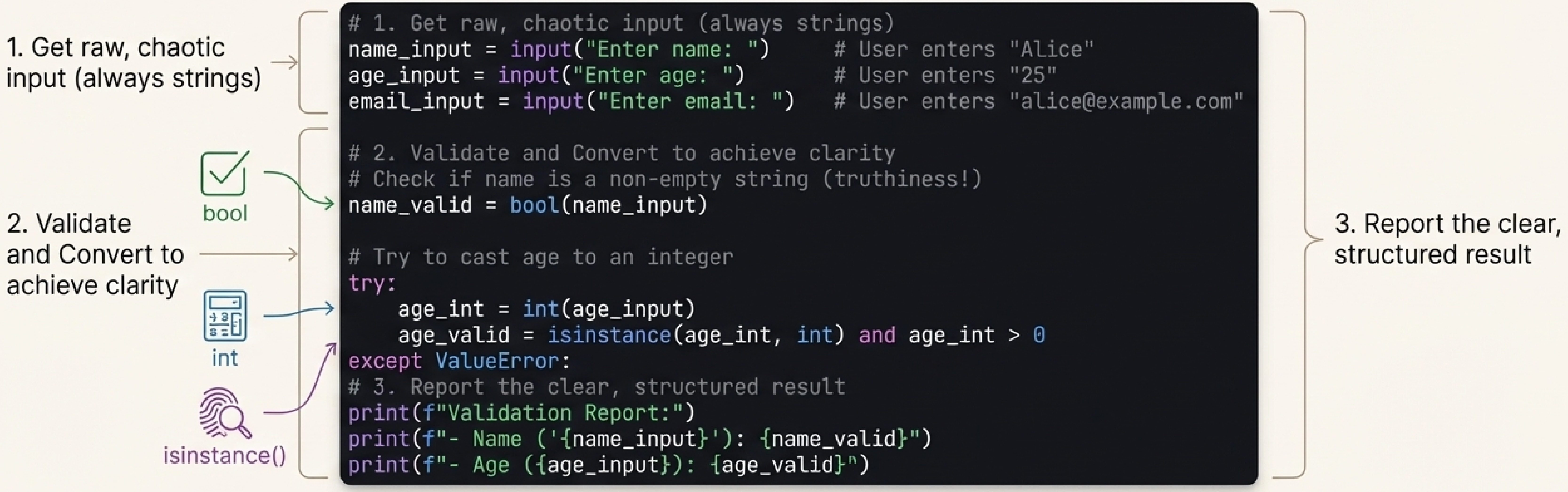
Capstone Project: From Chaotic Input to Clear Data

The Problem

User input is always text and can be unpredictable. How do we ensure our program gets the data types it needs?

The Solution

The User Profile Validator



1. Get raw, chaotic input (always strings)

2. Validate and Convert to achieve clarity

bool

int

isinstance()

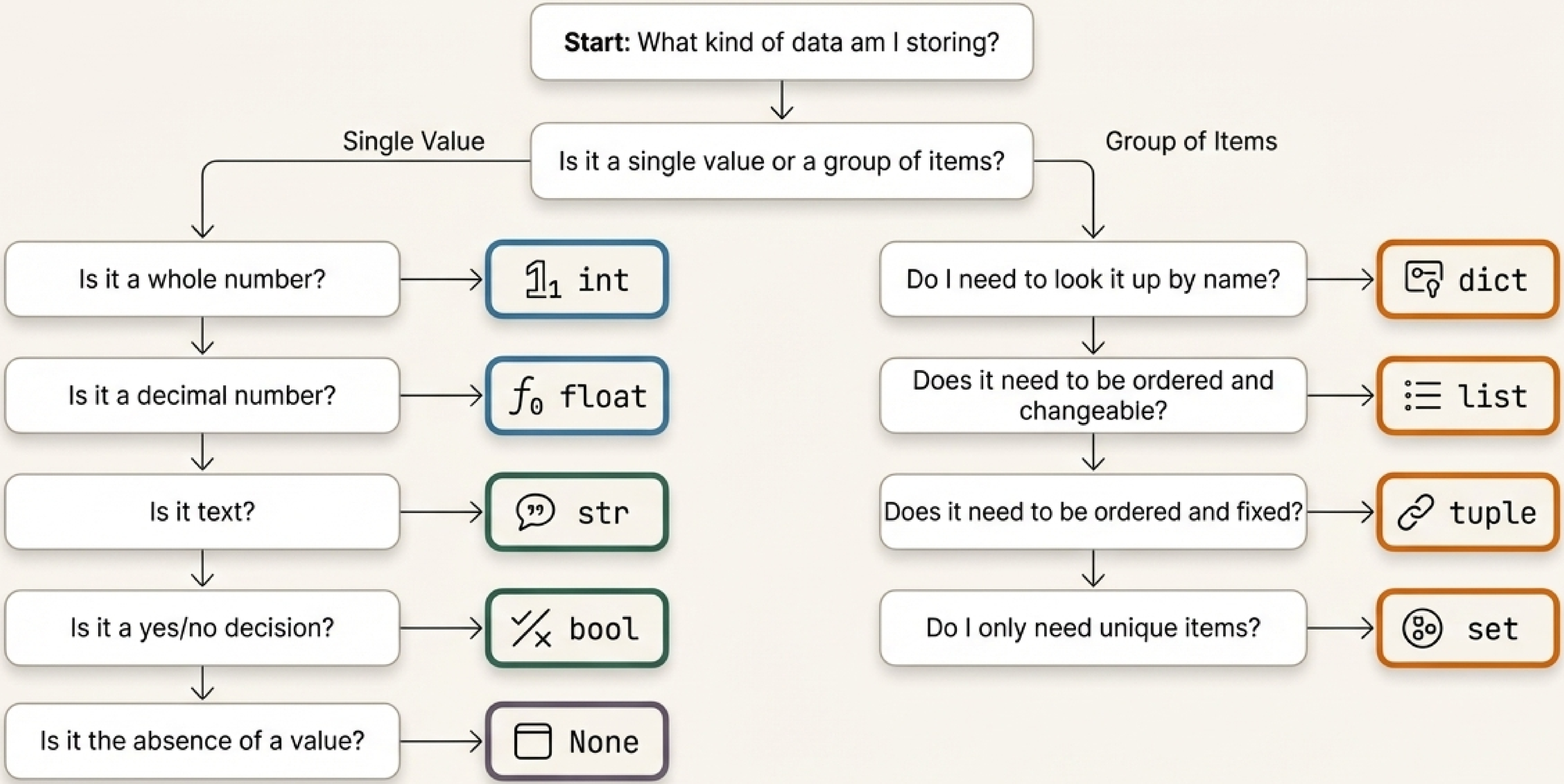
3. Report the clear, structured result

```python
# 1. Get raw, chaotic input (always strings)
name_input = input("Enter name: ")     # User enters "Alice"
age_input = input("Enter age: ")       # User enters "25"
email_input = input("Enter email: ")   # User enters "alice@example.com"

# 2. Validate and Convert to achieve clarity
# Check if name is a non-empty string (truthiness!)
name_valid = bool(name_input)

# Try to cast age to an integer
try:
    age_int = int(age_input)
    age_valid = isinstance(age_int, int) and age_int > 0
except ValueError:
# 3. Report the clear, structured result
print(f"Validation Report:")
print(f"- Name ('{name_input}'): {name_valid}")
print(f"- Age ({age_input}): {age_valid}")
```

This is how real applications achieve data integrity. They inspect, validate, and cast untrusted input into clean, reliable types.

NotebookLM

Your Type Decision Framework: The Ultimate Takeaway



Start: What kind of data am I storing?

Is it a single value or a group of items?

Single Value

Is it a whole number? → int

Is it a decimal number? → float

Is it text? → str

Is it a yes/no decision? → bool

Is it the absence of a value? → None

Group of Items

Do I need to look it up by name? → dict

Does it need to be ordered and changeable? → list

Does it need to be ordered and fixed? → tuple

Do I only need unique items? → set

Use this framework every time you create a variable. Clarity in types leads to clarity in code.

Test Your Mastery

For each scenario, which data type is the best fit?

You're storing a user's profile with their age, email address, and premium membership status.

Hint: `age: ?, email: ?, is_premium: ?`

Correct Answer: `int`, `str`, `bool`

You need to store the RGB values for a fixed color, like `(255, 99, 71)`. The values must not change accidentally.

Hint: A list or a tuple?

Correct Answer: `tuple`, because it's immutable.

A user's middle name is optional. What value should the variable have if they don't provide one?

Hint: "" or `0` or `None`?

Correct Answer: `None`, to represent the absence of data.

You have a list of user IDs for an event and need to remove any duplicates automatically.

Hint: `Which collection enforces uniqueness?`

Correct Answer: `set`