Data Types and Type Annotations in Python

What Are Data Types?

- Every value in Python has a type: `int`, `str`, `list`, etc.
- Example: `3` is an `int`, `'hello'` is a `str`, `[1, 2, 3]` is a `list` of `int`s.

Why Do Data Types Matter?

- They help us reason about what code is supposed to do.
- They help us catch mistakes early (e.g., trying to add a string to a number).
- Many programming languages use types to prevent errors before running the code.

Type Annotations in Python

- Python is "dynamically typed", meaning that a named variable can be an integer and then later in the execution be a string.
- We can add type hints to our code.
  - Syntax: `def add(x: int, y: int) -> int:`
- Type annotations aren't enforced in Python
  - you can invoke the add function on non-integers
- You can use a type checker (like mypy) to do some checking of your code to identify places where expectations may be violated.
- We have vscode running the pylance linter, which also catches many type errors.

Running the Type Checker

To run the type checker, you can use the command line tool `mypy`. Simply navigate to the directory containing your Python files and run:

```
mypy <your_file.py>
```

This will check the specified file for type errors based on the annotations you've added.

Or you can run it against all files in a directory, such as the current directory:

```
mypy .
```

Example: Type Annotations in `advent.py`

Let's look at the code in `advent/advent.py` and see type annotations in action.

- What types do the function arguments and return values have?
- Are there places where using types could help us avoid mistakes?
    - Try changing the return type of the take function (method) to `int`.
    - What indication do you get in your IDE?
    - Notice the "Problems" tab (bottom panel, with Terminal)
    - Try pasting the errors from there into the chat window and ask it to fix the problem in the code.

Discussion

- Why bother with type checking?