Why Type Hints?

Making Python Code Safer and More Maintainable

Effective Programming Practices for Economists

Python is Dynamically Typed

```python
def add(a, b):
    return a + b

add(1, 2)        # Returns 3
add("hello", " world")  # Returns "hello world"
add([1, 2], [3, 4])     # Returns [1, 2, 3, 4]
```

Flexibility is great, but...

- What types does `add` actually expect?
- Will it work with my data?
- What does it return?

The Problem with Dynamic Typing

```python
def calculate_portfolio_return(prices, weights):
    # What are prices and weights?
    # Lists? NumPy arrays? DataFrames? Dicts?
    return sum(p * w for p, w in zip(prices, weights))
```

Questions your IDE can't answer:

- Does `prices` need to be the same length as `weights`?
- Can I pass a pandas Series?
- What if I accidentally swap the arguments?

Type Hints to the Rescue

```python
def calculate_portfolio_return(
    prices: list[float],
    weights: list[float],
) -> float:
    return sum(p * w for p, w in zip(prices, weights))
```

Now we know:

- Both arguments are lists of floats
- The function returns a float
- IDE can warn us about mistakes

Three Benefits of Type Hints

1. Documentation

Types are always up-to-date documentation (unlike comments)

2. IDE Support

Autocomplete, refactoring, and error detection

3. Bug Prevention

Type checkers catch errors before runtime

Type Hints are Optional

```python
# Python doesn't enforce types at runtime!

def greet(name: str) -> str:
    return f"Hello, {name}"

greet(2)  # Runs without error!
# Returns "Hello, 2"
```

Type hints are for:

- **Developers** reading the code
- **IDEs** providing assistance
- **Type checkers** (e.g., ty) finding bugs → great as guardrails for AI agents

Type Checkers: ty

```
# Install ty (from Astral, makers of ruff and uv)
pixi add ty

# Check your code
pixi run ty check my_module.py
```

```
# my_module.py
def greet(name: str) -> str:
    return f"Hello, {name}"

greet(3)  # ty error: Argument 1 has incompatible type "int"
```

IDE Integration

Modern IDEs like VS Code use type hints for:

- **Autocomplete** - knows what methods are available
- **Error highlighting** - red squiggles for type mismatches
- **Refactoring** - safely rename across codebase
- **Documentation** - hover to see types

Writing Python Like It’s Rust

> "Parse, don’t validate" - Make illegal states unrepresentable

The Rust philosophy applied to Python:

1. Use types to encode **what** something is
2. Make invalid inputs **impossible** to construct
3. Let the type system catch errors **before runtime**

Source: Writing Python Like It’s Rust

Real-World Impact: optimagic

Before (stringly-typed):

```
minimize(fun=f, params=x, algorithm="scipy_lbfgsb")
# Typo? No error until runtime!
```

After (strongly-typed):

```
minimize(fun=f, params=x, algorithm=om.algorithms.scipy_lbfgsb)
# IDE autocomplete, typos caught immediately
```

optimagic migrated its entire API for better user and developer experience.

When to Use Type Hints

Always use them for:

- Function signatures (arguments and return types)
- Class attributes
- Public APIs

Optional for:

- Local variables (often inferred)
- Quick scripts and notebooks
- Prototype code

Rule of thumb:

If someone else will read it, add types.

Summary

Type hints make Python code:

1. **Self-documenting** - Types explain intent
2. **Safer** - Catch bugs before runtime
3. **More maintainable** - Refactor with confidence

Caveat: Active area of progress in Python

Things have evolved a lot in recent years!