University of Exeter

Type definitions in Python Research Software Engineering

Introduction

- What is type hinting & why is it useful?
- What does this look like in Python 3?
- Tooling & setup
- Examples (live coding)
- More concepts, cheat sheets and NumPy support
- Circular import problems
- Python development in 2023

What is type hinting? Why is it useful?

Type hints: a formal solution for dynamically typed languages to document and analyse the type of objects.

- Statically typed
  - Object/variable types are *static*.
  - Variable types are declared, or sometimes inferred. These are known at compile time, and cannot be changed.
  - Compiler can perform checks based on types, helping to catch bugs at an early stage.
  - Higher start up overhead, but can pay off in speed.
  - C++, C, Rust, TypeScript etc.

- Dynamically typed
  - Object/variable types are *dynamic*.
  - Interpreter assigns variables a type based on their value at runtime.
  - Common in scripting languages where there is no compile stage.
  - Can be faster to write, but type assignment occurs at runtime and performance suffers.
  - Python, Ruby etc.

University of Exeter

What is type hinting? Why is it useful?

Type hints: a formal solution for dynamically typed languages to document and analyse the type of objects.

Keeping track of types can help improve:

- Developer experience: lower cognitive load and improved readability when object types are declared.
- Group working: helps communicate design intentions of co-workers.
- Identification of bugs: edge cases, such as return None, or hard to find bugs on legacy code bases.
- Documentation: clarity and accuracy improved when types are known.
- Data validation: helps control types of data being used in a system.

University of Exeter

What does this look like in Python 3?

Python is dynamically typed, and not compiled. The solution to obtain the listed benefits has been implemented as **type annotations,** now **type hints.**

- Type annotations (or hints) have been supported since Python 3.5.
- Improvements and features to this type system are being added each release: i.e. in Python 3.10 typing.Union can be typed with pipe symbol.
- Beyond being read by a developer, type annotations can be used by linters, static type checkers, IDEs, etc.
  - A special constant, typing.TYPE_CHECKING, is False at runtime, and True otherwise

Variable and function types are not enforced at runtime, whether type hints are used or not, and never will be!

University of Exeter

Tooling and setup

You can start writing type hints in Python without any additional software. However, for live analysis/type checking, additional software is required.

There are two main options:

- Mypy, been around the longest, probably the most used. Uses built-in Python parser.
- Pyright (included with Pylance extension), by Microsoft. Uses own parser.

In 2023, Pyright is many times faster than Mypy, and has excellent VSCode integration. The Pylance extension is trivial to install and setup in VSCode. Updates and PEP support have been implemented extremely fast.

For a detailed comparison, please see here.

University of Exeter

Live coding demo

Type annotations of variables

```python
# We can annotate types outside function definitions

car_models: list[str] = ["bmw", "mercedes", "ferrari"]
car_counts: list[int] = [1, 2, 54]
inventory: dict[str, int] = dict(zip(car_models, car_counts))

inventory: dict[str, int] = {
    "citroen": 5,
    "audi": 5,
    "kia": 20,
}

average_car_prices: dict[str, float] = {
    "citroen": 2043.54,
    "audi": 5673.5,
    "kia": 3265.78,
}
```

University of Exeter

Type annotations of functions

```python
# We can use type annotations for functions

def count_stock(inventory: dict[str, int]) -> int:
    """
    Function to count cars in inventory.

    Args:
        inventory: dictionary of inventory

    Returns:
        stock_count: the number of cars in inventory
    """

    stock_count = sum(inventory.values())

    return stock_count
```

University of Exeter

Another example of a function

```python
def remove_from_stock(inventory: dict[str, int], car_model_to_remove: str):
    """
    Removes a car from the inventory.

    Args:
        inventory: dictionary of inventory
        car_model_to_remove: the model of car to remove
    """

    if car_model_to_remove not in inventory:
        raise ValueError(f"No cars of model {car_model_to_remove} in inventory.")

    current_stock = inventory[car_model_to_remove]

    if current_stock > 1:
        inventory[car_model_to_remove] -= 1

    else:
        del inventory[car_model_to_remove]
```

University of Exeter

Custom class as a type annotation

```
def open_showrooms(showroom_list: list[Showroom]):
    """
    Open a number of showrooms.

    Args:
        showroom_list: list of showrooms to open
    """

    for showroom in showroom_list:
        showroom.is_open = True

    print("All showrooms are now open!")
```

More concepts

Other key annotations that are often used. Please see typing docs for more information & rules:

- str, list, tuple, dict etc: any built in types
- typing.Any: an unconstrained type
- typing.NoReturn: function never returns
- typing.Self: the current enclosed class
- typing.Union[a, b]: equivalent to a | b, means either a or b
- typing.Optional[a]: equivalent to a | None, means either a or None

Note that other abstract types, such as typing.Sequence, or typing.Iterable, can be used.

University of Exeter

More concepts

Dataclasses make heavy use of type hints.

- These help to reduce boilerplate code.
- They include lots of functionality/convenience methods out the box.
- You can also make these immutable.
- We can use the @dataclass decorator to define these.

TypedDicts do as well.

- TypedDicts must have the right keys and the right types.
- These can be used for data validation, and are really handy when returning data structures.

University of Exeter

Cheat sheets and NumPy

- The MyPy documentation hosts a good type hint cheat sheet, which can be found here.

- Information about type hinting in NumPy can be found here. Support is not yet mature, but works, with a generic np.typing.NDArray annotation. See the repository for an example.

- Libraries such as nptyping here allow more control over shape and dtype parameters for numpy type hints.

University of Exeter

Dataclasses

```python
from dataclasses import dataclass


# Dataclasses make use of type hints, and generate a number of methods for you
@dataclass
class InventoryItem:
    """Class for an inventory item in a showroom."""

    vehicle_type: str
    model: str
    unit_price: float
    stock: int = 0

    def total_model_cost(self) -> float:
        """Calculate total cost of model in stock."""

        return self.unit_price * self.stock
```

University of Exeter

TypedDicts

```python
from typing import TypedDict

class EngineAttributes(TypedDict):
    engine_type: str
    engine_size: str
    cylinders: float
    turbo: bool
    engine_location: str

class Car:

.......

    def get_engine_attributes(self) -> EngineAttributes:
        """Return a dictionary of engine attributes."""

        return {
            "engine_type": self.engine_type,
            "engine_size": self.engine_size,
            "cylinders": self.cylinders,
            "turbo": self.turbo,
            "engine_location": self.engine_location,
        }
```

University of Exeter

A gotcha: Circular import problems

```
import typing

if typing.TYPE_CHECKING:
    import CustomClass
```

Circular imports

- With a complex project containing many custom classes, it might be necessary to import these to use only as type annotations.
- Regular import statements can lead to circular import errors, where one module is imported into another.
- We can use the Boolean typing.TYPE_CHECKING to help resolve this issue in our import block. This is a method for type checkers to help break import cycles.
- It evaluates to True when type checking, and False at runtime.

University of Exeter

Python development in 2023

type hinting

- Pyright
- mypy

version control

- git
- GitHub

virtual environments & dependency managers

- Pipenv
- Poetry
- Conda
- Virtualenv

linting & auto-formatting

- black
- flake8
- isort
- Pylint
- a spell checker!

*(all on save)

CI/CD

- GitHub Actions

testing & auto-docs

- Pytest
- Sphinx

University of Exeter