

2025



Lesson 03

Python Type Hints & Pydantic

Introduction to Python Type Hints

- Python is a dynamically-typed language where the type of a variable is determined at runtime.
- **PEP 484** (Python Enhancement Proposal) introduces type hints to Python, laying the foundation for static type checking.
- Type hints introduce a layer of optional static typing allowing developer to check for type consistency without enforcing types at runtime.
- Static type checking verifies the type safety of a program based on analysis of a source code. Static type checking is done at compile-time.
- Python type hints do not enforce type checking at runtime but provide guidance for developers.
- It can be applied to function parameters and return types, which aids in understanding what is expected when calling a function.
- It supports complex types, generic and user-defined type.

Introduction to Python Type Hints

- Features like Optional, Union and Any enable more flexible type annotations while Generics allow coding functions and classes that work with multiple types.
- Static type checking like `pyright` analyses code for type consistency before execution.
- It helps to catch potential bugs early by using the provided type hints to check for mismatches in types during code development.
- Type hints are an invaluable tool for adding readability and reliability to your Python code, especially in larger projects or collaborative environment.
- Introduced in Python starting from version 3.5, type hints don't enforce types at runtime. They act as optional metadata for development tool.

Why Use Type Hints?

1. Enhance Code Readability

- Easier to understand what the type of data each function/variable expects and each function returns.

2. Debugging Facilitation

- Aids in identifying type related errors.

3. Development Experience

- IDEs (Visual Studio Code/PyCharm etc.) use type hints for better code completion and error detection.

4. Static Type Checking

Tools like mypy use type hints to perform type checking.

Python Type Checkers

- `mypy`
 - First PEP 484 type checker and is best for bulk type checking entire directories. General purpose type checking.
- `pyre`
 - Another type checker for bulk checking with a special focus on speed.
- `pyright`
 - Developed by Microsoft.
 - Powers in-IDE type checking for VS Code.
 - Real-time checking, integrated development Visual Studio Code.
- `pytype`
 - Developed by Google.
 - Fast, designed for larger codebases.

* The list above is not exhaustive.

Dataclass

- Dataclasses and type hints are both features in Python that enhance the way developers define and manage data structures, but they serve different purposes and have distinct characteristics.
- Dataclass was introduced in Python 3.7. It is a way to create classes that primarily store data with minimal boilerplate code. They are defined using the `@dataclass` decorator from the `dataclass` module.
- Dataclasses require type hints for their fields, which indicate the expected types of the attributes.
- However, these type hints do not enforce type checking at runtime; they serve mainly as documentation and for static analysis by tools like `mypy`.

Automatic Method Generation

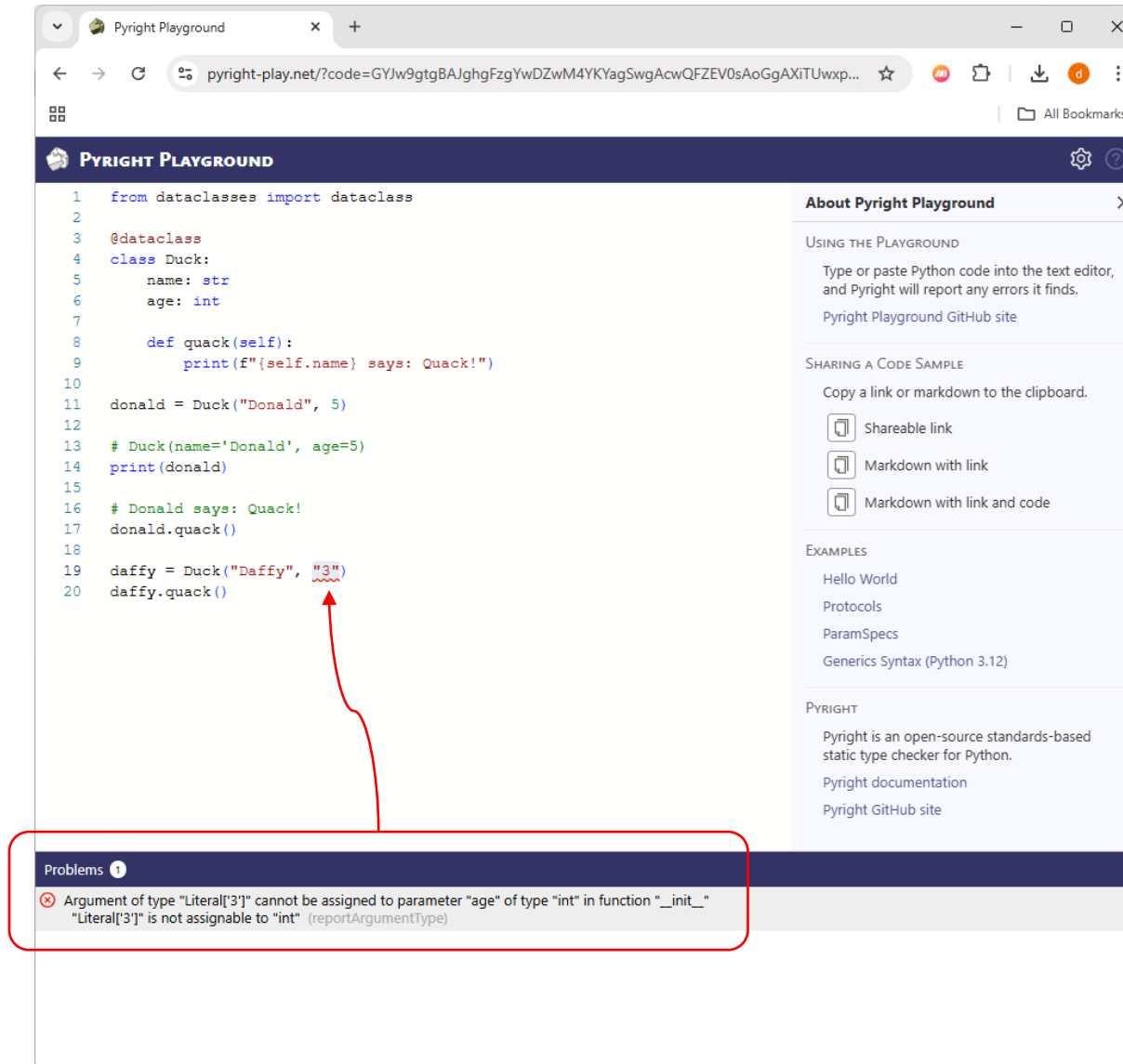
- When define a class as a dataclass, Python automatically generates several special methods, including:
 - `__init__()`: Initializes instance attributes.
 - `__repr__()`: Provides a string representation of the instance.
 - `__eq__()`: Enables comparison of instances based on their attributes.

Example

```
from dataclasses import dataclass
```

```
@dataclass
class Exercise:
    name: str
    reps: int
    sets: int
    weight: float
```

Pyright Playground



The screenshot shows the Pyright Playground web application. The main editor contains the following Python code:

```
1 from dataclasses import dataclass
2
3 @dataclass
4 class Duck:
5     name: str
6     age: int
7
8     def quack(self):
9         print(f"{self.name} says: Quack!")
10
11 donald = Duck("Donald", 5)
12
13 # Duck(name='Donald', age=5)
14 print(donald)
15
16 # Donald says: Quack!
17 donald.quack()
18
19 daffy = Duck("Daffy", "3")
20 daffy.quack()
```

A red arrow points from the error message in the Problems panel to the string "3" in the code on line 19. The error message is:

Problems 1

- Argument of type "Literal[3]" cannot be assigned to parameter "age" of type "int" in function "__init__"
"Literal[3]" is not assignable to "int" (reportArgumentType)

The right sidebar contains information about Pyright Playground, including instructions on how to use it, how to share a code sample, and examples of code snippets.

Dataclass Vs Python Type Hints

Feature	Dataclass	Type Hints
Availability	Python 3.7	Python 3.5
Purposes	Simplifies creation of data-centric classes	Provides optional type information
Automatic Method Generation	Yes (e.g., <code>__init__</code> , <code>__repr__</code>)	No
Type Enforcement	No (only for documentation)	No (only for documentation)
Usage	Primarily for data storage classes	For functions, variables, and class attributes

Both dataclasses and type hints enhance code readability and maintainability in Python, dataclasses focus on simplifying the creation of classes that hold data, whereas type hints provide a way to specify expected data types across various parts of your code without enforcing them at runtime.

References

- PEP 484 – Type hints
<https://peps.python.org/pep-0484/>
- PEP 557 – Data classes
<https://peps.python.org/pep-0557/>
- Real Python: Python type checking (Guide)
<https://realpython.com/python-type-checking/>
- Pyright
 - <https://microsoft.github.io/pyright/#/>
 - Playground: <https://pyright-play.net/>

Pydantic – An Introduction

- Pydantic and Python type hints share similarities but serve different purposes in Python development.
- A data validation library that extends Python type hints with **runtime** type-checking and data validation.
- Define models with types, and it ensures that the data matches these types by performing validation when data is passed in.
- Performs both type hints and automatic type enforcement and validation, which can raise errors if the data doesn't match the specified structure.
- Performs runtime validation based on the types you specify, ensuring that input data matches the expected types and constraints.
- Pydantic is widely used across various domains in the Python ecosystem, particularly for data validation and settings management.

Why Pydantic?

Learning Pydantic for LLM applications with frameworks like **LangChain** or **LlamaIndex** can bring substantial benefits, particularly for handling and validating data efficiently.

Here are some key reasons why Pydantic is valuable in these contexts:

- Large Language Model (LLM) applications often involve complex data pipelines, including inputs from users, external APIs, and databases.
- Pydantic makes it easy to define clear data models with strict validation, ensuring that inputs to your LLMs are accurate and reducing the risk of runtime errors.
- With Pydantic, you can define exact data structures for LLM inputs, intermediate processing, and outputs. This is especially helpful when managing prompts, responses, and metadata in frameworks like LangChain, where data consistency is crucial.

Pydantic Benefits

- Pydantic's type-based modeling encourages clearer and more maintainable code by using Python type hints. This makes data structures in LangChain or LlamaIndex pipelines easier to read and manage.
- It allows you to model prompt templates and response schemas explicitly. Pydantic can ensure each template and prompt step conforms to an expected structure.
- For response parsing, Pydantic's schema validation can help ensure that outputs from LLMs match a predictable structure, which is critical when chaining multiple LLM calls or interacting with other services.
- As LLM applications grow in complexity, Pydantic helps to create scalable, reusable code components. The well-defined data models can act as clear documentation for how different parts of your application communicate, making it easier to onboard new developers or extend existing workflows.

Pydantic Vs Python Type Hints

Feature	Python Type Hints	Pydantic
Validation Time	Static type checking only	Runtime validate
Enforcement Type	No runtime enforcement	Strict runtime enforcement
Dependencies	Built into Python 3.5 and later versions	Requires external package
Serialization	Not supported	Built0in serialization (dict, JSON)
Error Handling	Relies on static analysis	Raises detailed errors at runtime
Data coercion	None	Automatic type conversion when possible

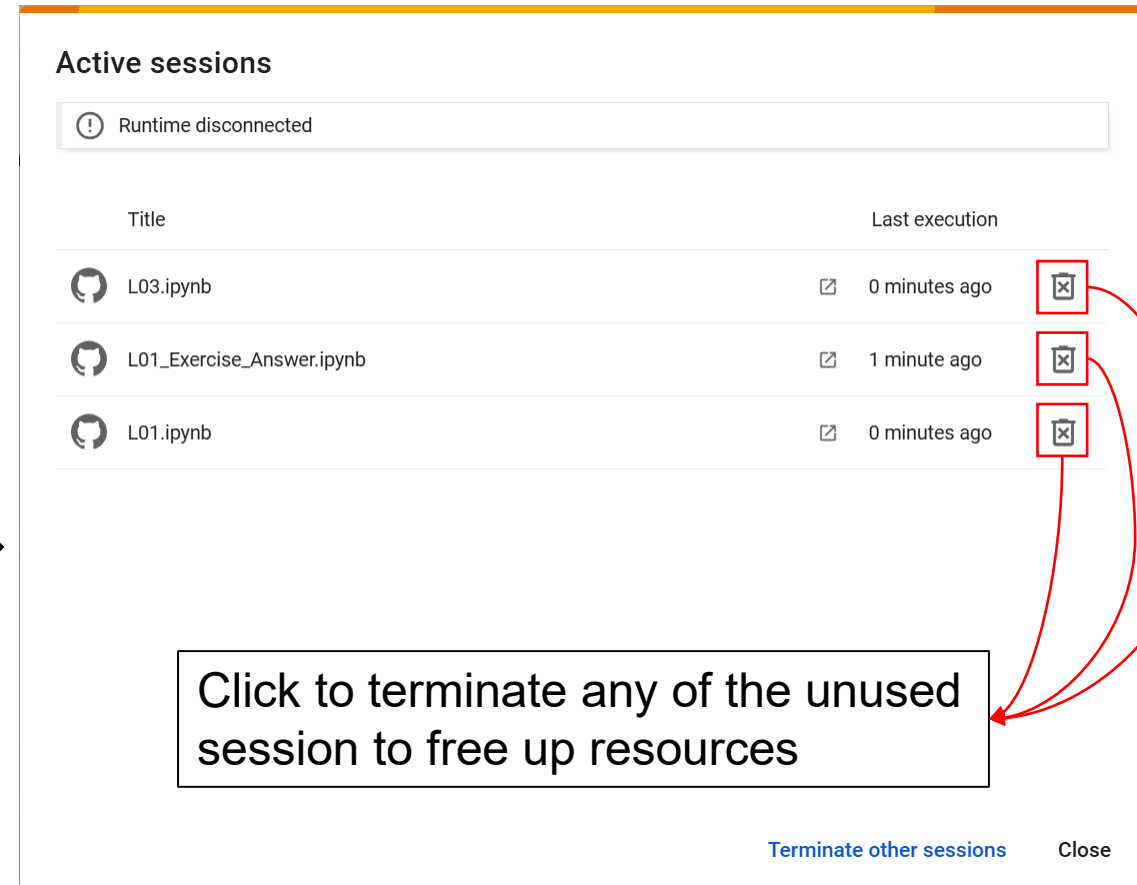
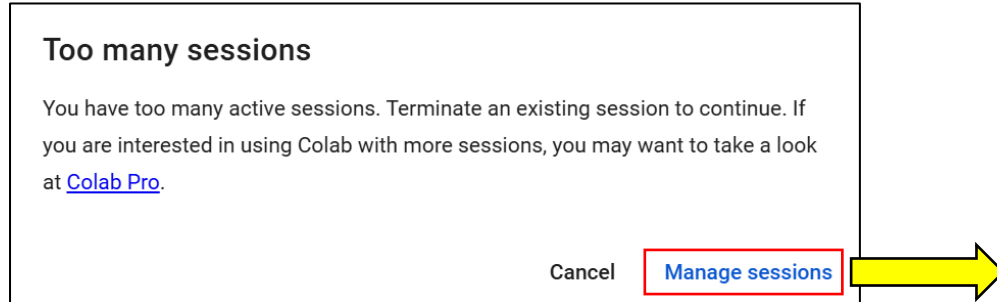
Python type hints are great for type-checking during development, while Pydantic provides powerful runtime validation and coercion, making it ideal for situations where data consistency and integrity are crucial, such as with external data sources or complex LLM applications.

Summary

- Integrating Pydantic into LangChain or LlamaIndex workflows will allow you to streamline data handling, enhance robustness, and ensure type-safe operations across your LLM application pipeline.
- This structured approach can be invaluable for both development efficiency and long-term maintainability.




Google Colab – A Reminder

- If you choose to run the code in Google Colab (wise choice!), sometimes you may encounter error message like:



Active sessions

⚠ Runtime disconnected

Title	Last execution	
L03.ipynb	0 minutes ago	
L01_Exercise_Answer.ipynb	1 minute ago	
L01.ipynb	0 minutes ago	

Click to terminate any of the unused session to free up resources

[Terminate other sessions](#) Close

Red arrows point from the termination buttons in the table to the text box below.

References

- Pydantic
<https://docs.pydantic.dev/latest/>
- Pydantic (v2) – In-depth Starter Guide
<https://www.youtube.com/watch?v=ok8bF8M7gjk>
- Pydantic V2 – Full course – Learn the BEST library for Data Validation and Parsing
https://www.youtube.com/watch?v=7aBRk_JP-qY
- Python Pydantic Tutorial – Learn how to write advanced classes in Python
<https://www.youtube.com/watch?v=pK6us8n9cLk>
- Pydantic – Validators (Root Validators, Pre-Item Validators, Each-Item Validators)
<https://www.youtube.com/watch?v=nQJKkY8XnRg>