Function Annotation

Function Annotation

- **Function Annotation** means **attaching metadata** (extra information) to a **function's parameters** and **return value**.
- These **annotations** usually describe **what type of value** is **expected** for each parameter and **what type the function** should **return**.
- Annotations **do not enforce types** at runtime. They are only hints.
- Annotations = Hints for Humans and Tools, **NOT** Guards Against Wrong Inputs

Python

```
def function_name(parameter1: type1, parameter2: type2, ...) -> return_type:
    # function body
```

: after parameter: **type hint**
-> after closing parenthesis **return type**

Example of Function Annotation

Python	Output

`def add_numbers(a: int, b: int) -> int:`
`    return a + b`
 	
`print(add_numbers(5, 10))`
	15

Here:

- **a: int** means parameter **a** is expected to be an **integer**.
- **b: int** means parameter **b** is expected to be an **integer**.
- **-> int** means the function is expected to return an **integer**.

Key Points To Remember

- **Not Enforced:** Python will not stop you if you pass a wrong type. (No error unless you manually check types.)
- **For Documentation:** Helps developers.
- Stored in **__annotations__** dictionary: Python saves all annotation info internally.

Python

```
print(add_numbers.__annotations__)
```

Output

```
{'a': <class 'int'>, 'b': <class 'int'>, 'return': <class 'int'>}
```

Complex Annotations

- Python also supports annotating parameters and returns using **complex types**
- You need to import from the **typing module**

Data Type	Usage
List	List of specific type
Tuple	Tuple of types
Dict	Dictionary with key-value types
Optional	Either a type or None
Union	One of multiple types

Example

Python

```python
from typing import List

def total(numbers: List[int]) -> int:
    return sum(numbers)

print(total([1,2,3,4,5]))
```

Output

```
15
```

Summary

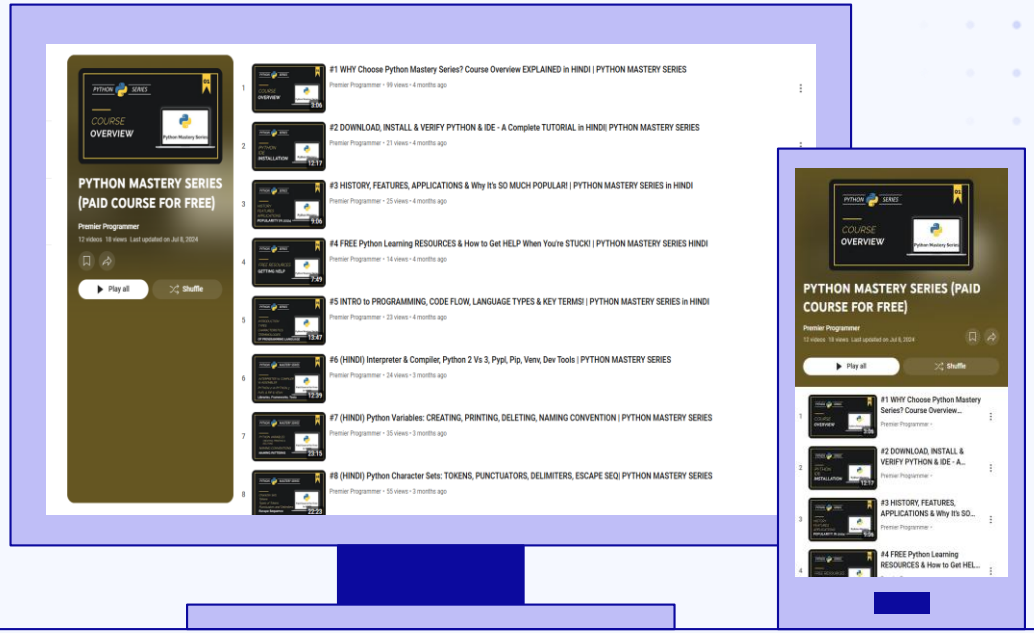
- Add **type hints** to **parameters** and **return values**.
- Python **does not enforce** types at runtime.
- Helpful for documentation, IDE support, and type checking tools.
- Use **typing module** for **complex types** (List, Tuple, Dict, etc.).
- Annotations are **optional**, but **highly recommended** for clean code!

Practice Set - 1

Download Link in **Description** and **Pinned Comment**

WATCH

Level up your coding with each episode in this focused Python series.



Next Video!

Practice Set - 1
Solution