# Переменные

## Статическая типизация переменных

1. Библиотека которая содержит инструменты для статиеской типизации – **typing**
2. Инструменты этой библиотеки не влияеют на выполнение кода напрямую, а нужены:

* для проверки типов линтерами (mypy, pyright, pylance),
* для автодополнения в IDE,
* для библиотек (например, FastAPI, Pydantic), которые читают типы в runtime + создают автоматически на их основе документацию.

### Инструменты typing

1. Annonated
2. Annotated — это конструкция из модуля typing (Python 3.9+), которая позволяет "оборачивать" обычный тип и прикреплять к нему метаданные.

1. Общая идея

from typing import Annotated

T = Annotated[int, "метаданные"]

* Первый аргумент: тип (int).
* Все последующие аргументы: метаданные (любой объект).

Для Python это значит: "T — это int, у которого есть дополнительные данные, которые может использовать сторонний код".

1. Зачем это нужно

Обычно аннотации в Python дают только тип.  
x: int → просто информация о том, что x — целое.

Но иногда нужно хранить ещё что-то:

* описание поля,
* ограничения (например, ≥ 1),
* откуда брать данные (query, path, header).

Annotated позволяет это делать.

1. В контексте FastAPI

FastAPI использует Annotated, чтобы в одном месте объединить:

1. Тип данных (для Python и валидации).
2. Метаданные (для документации и правил извлечения из HTTP).

Пример:

from typing import Annotated

from fastapi import Path

user\_id: Annotated[int, Path(title="ID пользователя", ge=1)]

* Базовый тип: int.
* Метаданные: Path(...), которые говорят FastAPI:
  + взять значение из URL (/users/{user\_id}),
  + применить валидацию (ge=1),
  + добавить описание в OpenAPI ("ID пользователя").

1. Ключевые моменты

* Annotated не заменяет тип → он оборачивает его.
* Метаданные никак не влияют на сам Python-интерпретатор, они используются только библиотеками (например, FastAPI, Pydantic).
* Можно прикреплять несколько метаданных:

Annotated[int, "meta1", "meta2"]

# Операции с файлами

1. Лучше в программах использовать путь не относительный или абсолютный – он зависит от того, в какой директории открыт терминал  
     
   А лучше вычислять путь данного файла и использовать его:

from pathlib import Path

import oracledb

*# Получаем путь к директории, где находится текущий файл (db\_caller)*

current\_dir = Path(\_\_file\_\_).parent

*# Строим путь к instantclient относительно текущего файла*

lib\_dir\_path = current\_dir / "instantclient\_23\_8"

*# Передаем этот путь в init\_oracle\_client*

oracledb.init\_oracle\_client(lib\_dir=str(lib\_dir\_path))

# Декораторы

Пример

def add\_sprinkles (func):

def wrapper(\*args, \*\*kwargs):

print("\*You add sprinkles \*")

func(\*args, \*\*kwargs)

return wrapper

def add\_fudge (func):

def wrapper(\*args, \*\*kwargs):

print("\*You add fudge \*")

func(\*args, \*\*kwargs)

return wrapper

@add\_sprinkles

@add\_fudge

def get\_ice\_cream(flavor):

print(f"Here is your {flavor} ice cream ')

get\_ice\_cream("vanilla")

from pathlib import Path