Введение в аннотации типов Python

Python имеет поддержку необязательных аннотаций типов.

Аннотации типов являются специальным синтаксисом, который позволяет определять тип переменной.

Объявление типов для переменных позволяет улучшить поддержку вашего кода редакторами и различными инструментами.

Это просто **краткое руководство / напоминание** об аннотациях типов в Python. Оно охватывает только минимум, необходимый для их использования с **FastAPI**... что на самом деле очень мало.

FastAPI целиком основан на аннотациях типов, у них много выгод и преимуществ.

Но даже если вы никогда не используете **FastAPI**, вам будет полезно немного узнать о них.

!!! note Если вы являетесь экспертом в Python и уже знаете всё об аннотациях типов, переходите к следующему разделу.

Мотивация

Давайте начнем с простого примера:

```
{!../../../docs_src/python_types/tutorial001.py!}
```

Вызов этой программы выводит:

```
John Doe
```

Функция делает следующее:

- Принимает `first_name` и `last_name`.
- Преобразует первую букву содержимого каждой переменной в верхний регистр с `title()`.
- Соединяет их через пробел.

```
{!../../../docs_src/python_types/tutorial001.py!}
```

Отредактируем пример

Это очень простая программа.

А теперь представьте, что вы пишете её с нуля.

В какой-то момент вы бы начали определение функции, у вас были бы готовы параметры...

Но затем вы должны вызвать «тот метод, который преобразует первую букву в верхний регистр».

Было это `upper`? Или `uppercase`? `first_uppercase`? `capitalize`?

Тогда вы попробуете с давним другом программиста: автодополнением редактора.

Вы вводите первый параметр функции, `first_name`, затем точку (`.`), а затем нажимаете `Ctrl+Space`, чтобы запустить дополнение.

Но, к сожалению, ничего полезного не выходит:

Добавим типы

Давайте изменим одну строчку в предыдущей версии.

Мы изменим именно этот фрагмент, параметры функции, с:

```
    first_name, last_name
```

на:

```
    first_name: str, last_name: str
```

Вот и все.

Это аннотации типов:

```
{!../../../docs_src/python_types/tutorial002.py!}
```

Это не то же самое, что объявление значений по умолчанию, например:

```
    first_name="john", last_name="doe"
```

Это другая вещь.

Мы используем двоеточия (`:`), а не равно (`=`).

И добавление аннотаций типов обычно не меняет происходящего по сравнению с тем, что произошло бы без неё.

Но теперь представьте, что вы снова находитесь в процессе создания этой функции, но уже с аннотациями типов.

В тот же момент вы пытаетесь запустить автодополнение с помощью `Ctrl+Space` и вы видите:

При этом вы можете просматривать варианты, пока не найдёте подходящий:

Больше мотивации

Проверьте эту функцию, она уже имеет аннотации типов:

```
{!../../../docs_src/python_types/tutorial003.py!}
```

Поскольку редактор знает типы переменных, вы получаете не только дополнение, но и проверки ошибок:

Теперь вы знаете, что вам нужно исправить, преобразовав `age` в строку с `str(age)`:

```
{!../../../docs_src/python_types/tutorial004.py!}
```

Объявление типов

Вы только что видели основное место для объявления подсказок типов. В качестве параметров функции.

Это также основное место, где вы можете использовать их с **FastAPI**.

Простые типы

Вы можете объявить все стандартные типы Python, а не только `str`.

Вы можете использовать, к примеру:

- `int`
- `float`
- `bool`
- `bytes`

```
{!../../../docs_src/python_types/tutorial005.py!}
```

Generic-типы с параметрами типов

Существуют некоторые структуры данных, которые могут содержать другие значения, например, `dict`, `list`, `set` и `tuple`. И внутренние значения тоже могут иметь свой тип.

Чтобы объявить эти типы и внутренние типы, вы можете использовать стандартный Python-модуль `typing`.

Он существует специально для поддержки подсказок этих типов.

`List`

Например, давайте определим переменную как `list`, состоящий из `str`.

Импортируйте `List` из `typing` (с заглавной `L`):

```
{!../../../docs_src/python_types/tutorial006.py!}
```

Объявите переменную с тем же синтаксисом двоеточия (`:`).

В качестве типа укажите `List`.

Поскольку список является типом, содержащим некоторые внутренние типы, вы помещаете их в квадратные скобки:

```
{!../../../docs_src/python_types/tutorial006.py!}
```

!!! tip Эти внутренние типы в квадратных скобках называются «параметрами типов».

```
В этом случае `str` является параметром типа, передаваемым в `List`.
```

Это означает: "переменная `items` является `list`, и каждый из элементов этого списка является `str`".

Если вы будете так поступать, редактор может оказывать поддержку даже при обработке элементов списка:

Без типов добиться этого практически невозможно.

Обратите внимание, что переменная `item` является одним из элементов списка `items`.

И все же редактор знает, что это `str`, и поддерживает это.

`Tuple` и `Set`

Вы бы сделали то же самое, чтобы объявить `tuple` и `set`:

```
{!../../../docs_src/python_types/tutorial007.py!}
```

Это означает:

- Переменная `items_t` является `tuple` с 3 элементами: `int`, другим `int` и `str`.
- Переменная `items_s` является `set` и каждый элемент имеет тип `bytes`.

`Dict`

Чтобы определить `dict`, вы передаёте 2 параметра типов, разделённых запятыми.

Первый параметр типа предназначен для ключей `dict`.

Второй параметр типа предназначен для значений `dict`:

```
{!../../../docs_src/python_types/tutorial008.py!}
```

Это означает:

- Переменная `prices` является `dict`:
    - Ключи этого `dict` имеют тип `str` (скажем, название каждого элемента).
    - Значения этого `dict` имеют тип `float` (скажем, цена каждой позиции).

`Optional`

Вы также можете использовать `Optional`, чтобы объявить, что переменная имеет тип, например, `str`, но это является «необязательным», что означает, что она также может быть `None`:

```
{!../../../docs_src/python_types/tutorial009.py!}
```

Использование `Optional[str]` вместо просто `str` позволит редактору помочь вам в обнаружении ошибок, в которых вы могли бы предположить, что значение всегда является `str`, хотя на самом деле это может быть и `None`.

Generic-типы

Эти типы принимают параметры в квадратных скобках:

- `List`
- `Tuple`
- `Set`
- `Dict`
- `Optional`
- ...и др.

называются **Generic-типами** или **Generics**.

Классы как типы

Вы также можете объявить класс как тип переменной.

Допустим, у вас есть класс `Person` с полем `name`:

```
{!../../../docs_src/python_types/tutorial010.py!}
```

Тогда вы можете объявить переменную типа `Person`:

```
{!../../../docs_src/python_types/tutorial010.py!}
```

И снова вы получаете полную поддержку редактора:

Pydantic-модели

Pydantic является Python-библиотекой для выполнения валидации данных.

Вы объявляете «форму» данных как классы с атрибутами.

И каждый атрибут имеет тип.

Затем вы создаете экземпляр этого класса с некоторыми значениями, и он проверяет значения, преобразует их в соответствующий тип (если все верно) и предоставляет вам объект со всеми данными.

И вы получаете полную поддержку редактора для этого итогового объекта.

Взято из официальной документации Pydantic:

```
{!../../../docs_src/python_types/tutorial011.py!}
```

!!! info Чтобы узнать больше о Pydantic, читайте его документацию.

FastAPI целиком основан на Pydantic.

Вы увидите намного больше всего этого на практике в Руководстве пользователя{.internal-link target=_blank}.

Аннотации типов в FastAPI

FastAPI получает преимущества аннотаций типов для выполнения определённых задач.

С **FastAPI** вы объявляете параметры с аннотациями типов и получаете:

- **Поддержку редактора**.
- **Проверки типов**.

...и **FastAPI** использует тот же механизм для:

- **Определения требований**: из параметров пути запроса, параметров запроса, заголовков, зависимостей и т.д.
- **Преобразования данных**: от запроса к нужному типу.
- **Валидации данных**: исходя из каждого запроса:
    - Генерации **автоматических ошибок**, возвращаемых клиенту, когда данные не являются корректными.
- **Документирования** API с использованием OpenAPI:
    - который затем используется пользовательскими интерфейсами автоматической интерактивной документации.

Всё это может показаться абстрактным. Не волнуйтесь. Вы увидите всё это в действии в Руководстве пользователя{.internal-link target=_blank}.

Важно то, что при использовании стандартных типов Python в одном месте (вместо добавления дополнительных классов, декораторов и т.д.) **FastAPI** сделает за вас большую часть работы.

!!! info Если вы уже прошли всё руководство и вернулись, чтобы узнать больше о типах, хорошим ресурсом является «шпаргалка» от `mypy` .