Introdução aos tipos Python

Python 3.6 + tem suporte para "type hints" opcionais.

Esses **"type hints"** são uma nova sintaxe (desde Python 3.6+) que permite declarar o tipo de uma variável.

Ao declarar tipos para suas variáveis, editores e ferramentas podem oferecer um melhor suporte.

Este é apenas um **tutorial rápido / atualização** sobre type hints Python. Ele cobre apenas o mínimo necessário para usá-los com o **FastAPI** ... que é realmente muito pouco.

O **FastAPI** é baseado nesses type hints, eles oferecem muitas vantagens e benefícios.

Mas mesmo que você nunca use o **FastAPI**, você se beneficiaria de aprender um pouco sobre eles.

!!! note "Nota" Se você é um especialista em Python e já sabe tudo sobre type hints, pule para o próximo capítulo.

Motivação

Vamos começar com um exemplo simples:

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

A chamada deste programa gera:

```
John Doe
```

A função faz o seguinte:

- Pega um `first_name` e `last_name`.
- Converte a primeira letra de cada uma em maiúsculas com `title()`.
- Concatena com um espaço no meio.

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

Edite-o

É um programa muito simples.

Mas agora imagine que você estava escrevendo do zero.

Em algum momento você teria iniciado a definição da função, já tinha os parâmetros prontos ...

Mas então você deve chamar "esse método que converte a primeira letra em maiúscula".

Era `upper`? Era `uppercase`? `first_uppercase`? `capitalize`?

Em seguida, tente com o velho amigo do programador, o preenchimento automático do editor.

Você digita o primeiro parâmetro da função, `first_name`, depois um ponto (`.`) e, em seguida, pressiona `Ctrl + Space` para acionar a conclusão.

Mas, infelizmente, você não obtém nada útil:

Adicionar tipos

Vamos modificar uma única linha da versão anterior.

Vamos mudar exatamente esse fragmento, os parâmetros da função, de:

```
    first_name, last_name
```

para:

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

É isso aí.

Esses são os "type hints":

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

Isso não é o mesmo que declarar valores padrão como seria com:

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

É uma coisa diferente.

Estamos usando dois pontos (`:`), não é igual a (`=`).

E adicionar type hints normalmente não muda o que acontece do que aconteceria sem elas.

Mas agora, imagine que você está novamente no meio da criação dessa função, mas com type hints.

No mesmo ponto, você tenta acionar o preenchimento automático com o `Ctrl Space` e vê:

Com isso, você pode rolar, vendo as opções, até encontrar o que "toca uma campainha":

Mais motivação

Marque esta função, ela já possui type hints:

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

Como o editor conhece os tipos de variáveis, você não apenas obtém a conclusão, mas também as verificações de erro:

Agora você sabe que precisa corrigí-lo, converta `age` em uma string com `str (age)`:

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

Tipos de declaração

Você acabou de ver o local principal para declarar type hints. Como parâmetros de função.

Este também é o principal local em que você os usaria com o **FastAPI**.

Tipos simples

Você pode declarar todos os tipos padrão de Python, não apenas `str`.

Você pode usar, por exemplo:

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

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

Tipos genéricos com parâmetros de tipo

Existem algumas estruturas de dados que podem conter outros valores, como `dict`, `list`, `set` e `tuple`. E os valores internos também podem ter seu próprio tipo.

Para declarar esses tipos e os tipos internos, você pode usar o módulo Python padrão `typing`.

Ele existe especificamente para suportar esses type hints.

`List`

Por exemplo, vamos definir uma variável para ser uma `lista` de `str`.

Em `typing`, importe `List` (com um `L` maiúsculo):

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

Declare a variável com a mesma sintaxe de dois pontos (`:`).

Como o tipo, coloque a `List`.

Como a lista é um tipo que contém alguns tipos internos, você os coloca entre colchetes:

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

!!! tip "Dica" Esses tipos internos entre colchetes são chamados de "parâmetros de tipo".

```
Nesse caso, `str` é o parâmetro de tipo passado para `List`.
```

Isso significa que: "a variável `items` é uma `list`, e cada um dos itens desta lista é uma `str`".

Ao fazer isso, seu editor pode fornecer suporte mesmo durante o processamento de itens da lista:

Sem tipos, isso é quase impossível de alcançar.

Observe que a variável `item` é um dos elementos da lista `items`.

E, ainda assim, o editor sabe que é um `str` e fornece suporte para isso.

`Tuple` e `Set`

Você faria o mesmo para declarar `tuple`s e `set`s:

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

Isso significa que:

- A variável `items_t` é uma `tuple` com 3 itens, um `int`, outro `int` e uma `str`.
- A variável `items_s` é um `set`, e cada um de seus itens é do tipo `bytes`.

`Dict`

Para definir um `dict`, você passa 2 parâmetros de tipo, separados por vírgulas.

O primeiro parâmetro de tipo é para as chaves do `dict`.

O segundo parâmetro de tipo é para os valores do `dict`:

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

Isso significa que:

- A variável `prices` é um dict`:
  - As chaves deste `dict` são do tipo `str` (digamos, o nome de cada item).
  - Os valores deste `dict` são do tipo `float` (digamos, o preço de cada item).

`Opcional`

Você também pode usar o `Opcional` para declarar que uma variável tem um tipo, como `str`, mas que é "opcional", o que significa que também pode ser `None`:

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

O uso de `Opcional [str]` em vez de apenas `str` permitirá que o editor o ajude a detectar erros, onde você pode estar assumindo que um valor é sempre um `str`, quando na verdade também pode ser `None`.

Tipos genéricos

Esses tipos que usam parâmetros de tipo entre colchetes, como:

- `List`
- `Tuple`
- `Set`
- `Dict`
- `Opcional`
- ...e outros.

são chamados **tipos genéricos** ou **genéricos**.

Classes como tipos

Você também pode declarar uma classe como o tipo de uma variável.

Digamos que você tenha uma classe `Person`, com um nome:

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

Então você pode declarar que uma variável é do tipo `Person`:

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

E então, novamente, você recebe todo o suporte do editor:

Modelos Pydantic

Pydantic é uma biblioteca Python para executar a validação de dados.

Você declara a "forma" dos dados como classes com atributos.

E cada atributo tem um tipo.

Em seguida, você cria uma instância dessa classe com alguns valores e ela os validará, os converterá para o tipo apropriado (se for esse o caso) e fornecerá um objeto com todos os dados.

E você recebe todo o suporte do editor com esse objeto resultante.

Retirado dos documentos oficiais dos Pydantic:

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

!!! info "Informação" Para saber mais sobre o Pydantic, verifique seus documentos .

FastAPI é todo baseado em Pydantic.

Você verá muito mais disso na prática no Tutorial - Guia do usuário{.internal-link target=_blank}.

Type hints em FastAPI

O **FastAPI** aproveita esses type hints para fazer várias coisas.

Com o **FastAPI**, você declara parâmetros com type hints e obtém:

- **Suporte ao editor**.
- **Verificações de tipo**.

... e **FastAPI** usa as mesmas declarações para:

- **Definir requisitos**: dos parâmetros do caminho da solicitação, parâmetros da consulta, cabeçalhos, corpos, dependências, etc.
- **Converter dados**: da solicitação para o tipo necessário.
- **Validar dados**: provenientes de cada solicitação:
    - A geração de **erros automáticos** retornou ao cliente quando os dados são inválidos.
- **Documente** a API usando OpenAPI:
    - que é usado pelas interfaces de usuário da documentação interativa automática.

Tudo isso pode parecer abstrato. Não se preocupe. Você verá tudo isso em ação no Tutorial - Guia do usuário{.internal-link target=_blank}.

O importante é que, usando tipos padrão de Python, em um único local (em vez de adicionar mais classes, decoradores, etc.), o **FastAPI** fará muito trabalho para você.

!!! info "Informação" Se você já passou por todo o tutorial e voltou para ver mais sobre os tipos, um bom recurso é <a href = "https://mypy.readthedocs.io/en/latest/cheat_sheet_py3.html" class = "external-link "target =" _ blank "> a "cheat sheet" do `mypy` .