Aula 05

Anotações

As anotações no Python são uma forma de indicar explicitamente o tipo de dado que uma variável ou parâmetro de uma função espera receber. As anotações são opcionais, mas podem ser úteis para documentar o código e para ajudar os editores de código e outras ferramentas a fornecer sugestões de autocompletar e de erros de tipo.

As anotações são definidas colocando-se o tipo de dado após o nome da variável ou do parâmetro, separados por dois-pontos.

Variáveis

Alguns exemplos usados nas variáveis:

```python
"""módulo main.py"""

nome: str = 'Thomas Cruise Mapother IV'
idade: int = 60
altura: float = 1.70
filhos: list = ['Suri', 'Connor', 'Isabella']
pais: tuple = (
    'Mary Lee Pfeiffer',
    'Thomas Cruise Mapother III',
    12)
casado: bool = False

dados: dict = {
    'nome': 'Thomas Cruise Mapother IV',
    'apelido': 'Tom Cruise',
    'idade': 60,
    'altura': 1.70}

print(f'__annotations__ : {__annotations__}')
```

As anotações de tipo não afetam o comportamento do código em si, elas são usadas principalmente para documentação e ferramentas de análise estática de código. No entanto, a partir do Python 3.5, é possível usar as anotações de tipo com ferramentas externas, como o Mypy, para verificar se o código está de acordo com as especificações de tipo.

É importante notar que as anotações de tipo não são obrigatórias no Python, e muitos projetos e bibliotecas populares não fazem uso delas. Além disso, o Python é uma linguagem dinamicamente tipada, o que significa que as variáveis não precisam ser declaradas com um tipo específico e podem mudar de tipo durante a execução do programa. As anotações de tipo não mudam esse comportamento fundamental da linguagem, elas apenas fornecem informações adicionais para ajudar a entender o código.

Podemos usar a variável build-in e buscar todas essas variáveis declaradas.

```python
for var, tip in __annotations__.items():
    print(f'{var}: {tip}')
```

```
PS C:\Users\gutoh\OneDrive\0.Python> python main.py
nome: <class 'str'>
idade: <class 'int'>
altura: <class 'float'>
filhos: <class 'list'>
pais: <class 'tuple'>
casado: <class 'bool'>
dados: <class 'dict'>
PS C:\Users\gutoh\OneDrive\0.Python>
```

Variáveis que não tiverem seus tipos explicitamente especificadas não serão armazenadas em __annotations__.

```python
"""módulo main.py"""

nome: str = 'Thomas Cruise Mapother IV'
idade: int = 60
altura = 1.70

print(__annotations__)
```

```
PS C:\Users\gutoh\OneDrive\0.Python> python main.py
{'nome': <class 'str'>, 'idade': <class 'int'>}
PS C:\Users\gutoh\OneDrive\0.Python>
```

Repare que a variável "altura" não teve seu tipo especificado, logo ela não estará no dicionário __annotations__.

Funções

As anotações nas funções são úteis para especificar que tipo de variável ela espera receber e retornar.

```python
"""módulo main.py"""


def mostra_dados(p_nome: str, p_idade: int) -> None:
    """Função que mostra os dados de uma pessoa"""
    print(f'Nome: {p_nome} | Idade: {p_idade}')


if __name__ == '__main__':
    nome: str = 'Tom Cruise'
    idade: int = 60
    altura: float = 1.70

    mostra_dados(nome, idade)
```

No exemplo acima, além das anotações nas variáveis, temos anotações quanto ao retorno da função através da notação "->", que indica a função retornando None.

Mais exemplos de retornos de outros tipos.

```python
"""módulo main.py"""


def soma_lista(numeros: list[int]) -> int:
    """Recebe uma lista de números e retorna a soma
    dos elementos usando a função sum do Python."""
    return sum(numeros)


def imc(peso: float, altura: float) -> float:
    """Calcula o IMC (Índice de Massa Corporal) de
    uma pessoa."""
    return peso / (altura ** 2)


if __name__ == "__main__":
    print(soma_lista(list(range(10))))
    print(imc(85, 1.7))
```

```python
def concatena_nome(nome: str, sobrenome: str) -> str:
    """Concatena o nome e o sobrenome de uma pessoa."""
    return f"{nome} {sobrenome}"


def e_par(numero: int) -> bool:
    """Retorna True se o número for par, senão False."""
    return numero % 2 == 0


if __name__ == "__main__":
    print(concatena_nome('Tom', 'Cruise'))
    print(e_par(42))
    print(e_par(13))
```

```python
def filtra_pares(lista: list[int]) -> list[int]:
    """Retorna uma lista com os números
    pares de uma lista de inteiros."""
    return [x for x in lista if x % 2 == 0]

if __name__ == "__main__":
    print(filtra_pares(list(range(100))))
```

Reparou que agora os tipos lista tem uma especificação de tipo dentro? Isso indica que ele espera ou retorna uma lista daquele tipo específico.

Caso a função, por exemplo, possa receber mais de um tipo, podemos especificar usando o pipe | para separar os tipos.

```python
def soma(prm_1: int | float, prm_2: int | float) -> int | float:
    """soma dois números do tipo int ou float"""
    return prm_1 + prm_2

if __name__ == "__main__":
    print(soma(40, 2))
    print(soma(40.0, 2.0))
```

A função acima espera receber valores do tipo inteiro ou do tipo float.

Repare que, mesmo que isso ajude na explicação, isso pode deixar o código mais difícil de compreender, já que pode haver um excesso de informações.

Antigamente, esse trabalho era realizado com a ajuda do módulo typing, que foi introduzido no Python 3.5 (e que realiza muitas outras operações). A partir da versão 3.9, essa verificação foi introduzida de maneira nativa no Python, sem a necessidade de chamar o módulo.

Dicas VS Code

No campo de busca, pesquise por "python inlay hints" e marque as 3 opções. Elas ajudam mostrando sugestões de tipos e retorno no seu código.