Type Hinting avec Python

Le type hinting est une pratique consistant à annoter les types de données dans le code Python à l'aide de commentaires spéciaux ou de déclarations syntaxiques. Ces annotations ne sont pas nécessaires pour que le code fonctionne, mais elles permettent de spécifier les types attendus pour les variables, les arguments de fonction et les valeurs de retour.

Pourquoi utiliser le type hinting ?

1. **Documentation améliorée**: Les annotations de type fournissent une documentation claire sur les types de données attendus, ce qui facilite la compréhension du code pour les autres développeurs.
2. **Détection précoce des erreurs**: Les outils statiques tels que Mypy peuvent analyser les annotations de type et détecter les incohérences de type potentielles, ce qui permet de repérer les erreurs de manière précoce dans le processus de développement.
3. **Meilleure maintenance du code**: Les annotations de type rendent le code plus explicite et réduisent les risques de bugs, ce qui facilite la maintenance du code à long terme.

Utilisation de l'import *typing*:

Le module typing fournit des outils pour annoter les types de données de manière plus expressives. Voici comment l'utiliser :

```
from typing import List, Tuple, Dict, Union, Optional, Callable, Literal
```

- List, Tuple, Dict : Annoter des conteneurs comme des listes, des tuples et des dictionnaires.
- Union : Annoter des variables avec plusieurs types possibles.
- Optional : Annoter des variables qui peuvent être de type défini ou None.
- Callable : Annoter des fonctions avec des types de fonctions attendus comme arguments ou retours.
- Literal : Limiter les valeurs d'une variable à un ensemble prédéfini de valeurs littérales.
- Any : type spécial qui indique que la variable peut être de n'importe quel type.

```
from typing import Any

a: Any = 1
a = 3.14
a = "Bonjour"
```

En utilisant ces outils, vous pouvez rendre vos annotations de type plus expressives et plus précises, ce qui améliore la qualité et la maintenabilité de votre code.

Utilisation des Type Hints:

```
vit: float = 1.5  # Annotation de type
```

- Les annotations de type permettent de spécifier le type des variables, des arguments de fonction et des valeurs de retour.
- Ils sont facultatifs et n'affectent pas l'exécution du code.
- Même si vous n'initialisez pas la variable avec une valeur float, le code peut toujours être exécuté.

Syntaxe des Annotations de Type:

```
def f(v: int, w: str = '') -> None:  # Types hints pour les paramètres et le
retour
    ...
```

- Les annotations de type sont placées après les noms de variables, de paramètres ou de valeurs de retour, suivies de :.
- L'utilisation de None comme type de retour indique qu'aucune valeur n'est retournée.

Types de Base:

```
from types import NoneType

a: int = 1
a = 3.14
a = "Bonjour"

value: NoneType = None
```

- Les types de base tels que int, float, str, NoneType, etc., peuvent être annotés directement.

Containers:

```
from typing import List, Tuple, Dict

n1: List[int] = [1, 2, 3]
n2: Tuple[int, str, float] = (1, "allo", 3.14)
n3: Dict[str, int] = {"allo": 1}
```

- Pour les conteneurs comme les listes, tuples et dictionnaires, utilisez des annotations de type spécifiques (List, Tuple, Dict, etc.).

Annotations Avancées:

```
def sum_moy(values: list[float]) -> tuple[float, float]:
    ...
```

- Vous pouvez annoter des fonctions avec des paramètres et des valeurs de retour plus complexes.

Union Types:

```
from typing import Union

nombre: Union[int, float] = 3
```

- Les types union permettent à une variable d'accepter plusieurs types de données.

Optional Types:

```
from typing import Optional

value: Optional[int] = None
```

- Les types optionnels permettent à une variable d'être de type défini ou None.

Callable Types

```
from typing import Callable

def f(i: int, r: float) -> complex:
    ...

task: Callable[[int, float], complex] = f
```

- Vous pouvez annoter les fonctions avec des types de fonctions attendues comme arguments ou retours.

Literal Types

```
from typing import Literal

var: Literal['rouge', 'vert', 'bleu'] = 'rouge'
```

- Les types littéraux limitent les valeurs d'une variable à un ensemble prédéfini de valeurs littérales.

Forward Declaration

```
class Tower:
    def compare(other: 'Tower') -> bool:
        ...
```

- Utilisez des chaînes de caractères pour indiquer un type qui sera défini plus tard dans le code.