Décorateurs en python

Les décorateurs sont des fonctions qui prennent une autre fonction ou méthode en argument et renvoient une nouvelle fonction ou méthode modifiée. En Python, les décorateurs sont largement utilisés pour ajouter des fonctionnalités à des fonctions ou des classes sans les modifier directement.

Voici quelques points clés sur les décorateurs en Python :

1. **Syntaxe des décorateurs** : En Python, les décorateurs sont définis en plaçant le symbole `@` suivi du nom du décorateur juste au-dessus de la fonction ou de la méthode à décorer. Par exemple :

```
@decorator
def my_function():
    pass
```

Cela équivaut à my_function = decorator(my_function).

2. **Fonctions comme objets de première classe** : En Python, les fonctions sont des objets de première classe, ce qui signifie qu'elles peuvent être assignées à des variables, passées comme arguments à d'autres fonctions et retournées comme valeurs à partir d'autres fonctions. Cela permet de manipuler facilement les fonctions dans les décorateurs.

3. **Décorateurs de fonctions et de méthodes** : Les décorateurs peuvent être utilisés pour modifier le comportement des fonctions et des méthodes. Ils peuvent être utilisés pour ajouter du logging, de la validation d'entrée, du caching, etc.

4. **Chainage de décorateurs** : Il est possible d'appliquer plusieurs décorateurs à une même fonction ou méthode en les empilant les uns au-dessus des autres avec la syntaxe `@decorator1`, `@decorator2`.

```
@decorator1
@decorator2
def my_function():
    pass
```

Cela équivaut à my_function = decorator1(decorator2(my_function)).

5. **Décorateurs de classe** : En plus des fonctions, les décorateurs peuvent également être utilisés pour décorer des classes. Les décorateurs de classe peuvent être utilisés pour ajouter des méthodes supplémentaires, des propriétés calculées, ou pour modifier le comportement de l'initialisation de la classe, entre autres choses.

6. **Utilisation de décorateurs prédéfinis** : Python fournit plusieurs décorateurs prédéfinis dans les modules standard tels que `functools`, `contextlib`, `abc`, etc. Par exemple, `functools.wraps` est souvent utilisé pour conserver les métadonnées des fonctions décorées.

7. **Création de vos propres décorateurs** : Vous pouvez créer vos propres décorateurs personnalisés en définissant une fonction qui prend une autre fonction en argument, modifie son comportement selon vos besoins, et renvoie la fonction modifiée.

Les décorateurs sont un outil puissant en Python, mais ils peuvent rendre le code difficile à comprendre s'ils sont mal utilisés. Il est donc important de les utiliser avec parcimonie et de les documenter clairement pour faciliter la compréhension du code par d'autres développeurs.

Décorateurs Fondamentaux

Les décorateurs sont des fonctions spéciales en Python qui permettent de modifier le comportement d'autres fonctions ou méthodes. Voici un aperçu des principaux décorateurs :

- `@staticmethod`: Ce décorateur est utilisé pour définir une méthode statique dans une classe. Une méthode statique peut être appelée à partir de la classe ou de l'instance de la classe, mais elle ne reçoit pas implicitement l'instance de la classe (le self).

> Ce décorateur est utilisé pour définir une méthode statique dans une classe. Les méthodes statiques sont des méthodes qui peuvent être appelées sur la classe elle-même plutôt que sur une instance de la classe. Lorsqu'un objet appelle une méthode statique, aucun paramètre d'instance (comme self) n'est passé automatiquement à la méthode.

- `@classmethod`: Ce décorateur est similaire à @staticmethod, mais il reçoit implicitement la classe elle-même en tant que premier argument plutôt que l'instance de la classe.

> Ce décorateur est utilisé pour définir une méthode de classe dans une classe. Les méthodes de classe prennent la classe elle-même comme premier argument, conventionnellement appelé cls, plutôt que l'instance de la classe. Les méthodes de classe sont souvent utilisées pour créer des instances de la classe ou pour effectuer des opérations qui affectent la classe dans son ensemble.

- `@abstractmethod`: Ce décorateur est utilisé pour déclarer une méthode comme étant une méthode abstraite dans une classe abstraite. Une classe est considérée comme abstraite si elle contient au moins une méthode abstraite. Les méthodes abstraites doivent être redéfinies dans les sous-classes concrètes.

> Ce décorateur est utilisé pour déclarer une méthode comme abstraite dans une classe abstraite. Une méthode abstraite est une méthode qui doit être redéfinie dans une sous-classe, mais qui n'a pas de définition dans la classe abstraite elle-même.

- `@override`: Ce n'est pas un décorateur standard en Python, mais il est souvent utilisé comme convention pour indiquer qu'une méthode redéfinit une méthode de sa classe mère. Ce décorateur n'est pas requis, mais il est souvent utile pour documenter le code et éviter les erreurs de frappe lors de la redéfinition de méthodes.

> Utilisé dans le contexte de la programmation orientée objet pour indiquer qu'une méthode dans une classe enfant redéfinit une méthode héritée de la classe parente.

Les décorateurs sont utilisés en plaçant simplement le nom du décorateur au-dessus de la fonction ou de la méthode à laquelle il s'applique. Voici un exemple d'utilisation des décorateurs :

```
from abc import ABC, abstractmethod

class MyClass:
    @staticmethod
    def static_method():
        print("This is a static method")

    @classmethod
    def class_method(cls):
        print(f"This is a class method of {cls}")

class MyAbstractClass(ABC):
    @abstractmethod
    def abstract_method(self):
        pass

class ConcreteClass(MyAbstractClass):
    def abstract_method(self):
        print("Implementation of abstract_method")

    @staticmethod
    def static_method():
        print("This is a static method")

    @classmethod
    def class_method(cls):
        print(f"This is a class method of {cls}")

ConcreteClass.static_method()
ConcreteClass.class_method()
```

Dans cet exemple, ConcreteClass implémente abstract_method, et elle peut également appeler les méthodes statiques et de classe définies dans MyClass.