DESIGN PATTERNS - CHAPITRE 16

Iterator en Python

Séparer les données de leur parcours avec le protocole d'itération Python.

Par Ali ZAINOUL · Cours gratuit

Ce que vous allez apprendre

- Séparer les données de leur parcours avec le protocole d'itération Python.
- Vérifier l'épuisement d'un itérateur et la création d'un nouveau parcours.

Prérequis. Savoir utiliser classes, héritage, composition, listes, dictionnaires et exceptions Python. Les mécanismes spécifiques sont expliqués avant leur emploi.

Iterator : intention et expérience

Iterator : séparer collection et parcours

Un itérateur Python fournit `__iter__()`, qui retourne cet itérateur, et `__next__()`, qui retourne la prochaine valeur ou lève `StopIteration`. La fonction standard `next()` appelle cette dernière méthode. Un itérateur consommé ne redémarre pas ; demandez-en un nouveau à la collection.

Python

```python
# example.py
class Numbers:
    """
    Aggregate class holding a collection of numbers.
    """
    def __init__(self, numbers):
        self._numbers = numbers

    def create_iterator(self):
        """
        Creates and returns an iterator for the numbers collection.
        """
        return NumberIterator(self._numbers)

class NumberIterator:
    """
    Iterator class for traversing the numbers collection.
    """
    def __init__(self, numbers):
        self._numbers = numbers
        self._index = 0

    def has_next(self):
        """
        Checks if there are more elements to iterate over.
        """
        return self._index < len(self._numbers)

    def __iter__(self):
        return self

    def __next__(self):
        """
        Returns the next element in the collection.
        """
        if self.has_next():
            value = self._numbers[self._index]
```

```
            self._index += 1
            return value
        else:
            raise StopIteration

# Client code
numbers = Numbers([1, 2, 3, 4, 5])
iterator = numbers.create_iterator()

try:
    while True:
        print(next(iterator))
except StopIteration:
    pass
```

Vérifier et interpréter

Consommez l'itérateur jusqu'à StopIteration, puis appelez encore next(). Il doit rester épuisé. Créez ensuite un nouvel itérateur pour repartir au début. Le parcours est indépendant du client, mais sa relation avec une collection modifiée pendant l'itération doit être définie ; cet exemple ne réalise pas de copie de la liste reçue.

Version web : https://www.clearcode.fr/courses/programming/design-patterns/comportement/iterator