DESIGN PATTERNS - CHAPITRE 24

Interpreter en Python

Représenter les règles d'une petite grammaire et évaluer leur arbre dans un contexte.

Par Ali ZAINOUL · Cours gratuit

Ce que vous allez apprendre

- Représenter les règles d'une petite grammaire et évaluer leur arbre dans un contexte.
- Vérifier une table de vérité, les erreurs de variable et le court-circuit.

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

Interpreter : intention et expérience

Définir une petite grammaire

Une application doit évaluer de petites règles répétitives. Le patron **Interpreter** représente les constructions d'un langage par des objets qui évaluent une expression dans un contexte.

Notre grammaire est `expression = variable | NOT expression | (expression AND expression)`. Les variables cherchent une valeur booléenne dans un dictionnaire ; `Not` nie son opérande ; `And` combine les résultats avec le court-circuit Python. On construit l'arbre à la main : cet exemple ne transforme pas un texte en arbre et n'utilise pas `eval`.

Le contrat est un booléen ; une variable évaluée absente lève `KeyError`, une valeur évaluée non booléenne lève `TypeError`. Le court-circuit peut laisser la branche droite non évaluée. Écrivez la table de vérité des deux variables et vérifiez les quatre lignes. Une grammaire complexe ou de très grands arbres justifie un analyseur dédié plutôt qu'une multiplication de classes récursives.

Python

```python
# example.py
from dataclasses import dataclass

@dataclass(frozen=True)
class Variable:
    name: str

    def interpret(self, context):
        value = context[self.name]
        if type(value) is not bool:
            raise TypeError("Rule values must be booleans")
        return value

@dataclass(frozen=True)
class And:
    left: object
    right: object

    def interpret(self, context):
        return self.left.interpret(context) and self.right.interpret(context)

@dataclass(frozen=True)
class Not:
    expression: object

    def interpret(self, context):
        return not self.expression.interpret(context)

rule = And(Variable("has_ticket"), Not(Variable("is_blocked")))
print(rule.interpret({"has_ticket": True, "is_blocked": False}))  # True
print(rule.interpret({"has_ticket": True, "is_blocked": True}))   # False
```

Pour approfondir, consultez **les opérations booléennes Python**.

Vérifier et interpréter

Évaluez les quatre paires de booléens puis une variable absente dans une branche évaluée. Vérifiez la table de vérité et KeyError. Le contexte ne valide pas toutes les variables à l'avance : le court-circuit peut éviter une branche. Un langage plus complexe demande de séparer analyse du texte, validation et évaluation.

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