IFT-1004 - Introduction à la programmation

Module 6.5 : Le paradigme orienté objet - Exemples supplémentaires

I

Table des matières

Exemples de modélisation orientée objet

Exemples de modélisation orientée objet

Objectif

Illustrer l'utilisation de la programmation orientée objet par de nouveaux exemples.

La classe `NombreEntier`

Définissons une classe `NombreEntier` qui a un seul attribut : une valeur entière. Par contre, nous pourrons définir plusieurs méthodes pour cette classe qui agiront sur cette valeur.

Par exemple, nous pouvons créer une méthode `ajouter` qui prend en argument une valeur à additionner à la valeur actuelle, ou bien une méthode `est_premier`, qui valide si oui ou non notre nombre entier est premier.

La classe NombreEntier

```python
class NombreEntier:
    def __init__(self, valeur):
        self.valeur = valeur

    def ajouter(self, autre_valeur):
        self.valeur += autre_valeur

    def est_premier(self):
        if self.valeur <= 1:
            return False

        for i in range(2, self.valeur // 2 + 1):
            if self.valeur % i == 0:
                return False

        return True
```

La classe NombreEntier

Utilisation :

```
>>> n = NombreEntier(10)
>>> print(n.valeur)
10

>>> n.ajouter(3)
>>> print(n.valeur)
13

>>> print(n.est_premier())
True
```

La classe `Document`

Idée : encapsuler une chaîne de caractères et fournir des méthodes reliées à cette chaîne. Nous redéfinissons également la *méthode spéciale* `__repr__` pour indiquer à Python comment représenter un objet `Document` en une chaîne de caractères

```
class Document:
    def __init__(self, le_texte):
        self.texte = le_texte

    def transformer_en_majuscules(self):
        self.texte = self.texte.upper()

    def __repr__(self):
        return self.texte
```

La classe `Document`

Exemple d'utilisation :

```
>>> d = Document("Ceci est un texte.")
>>> print(d.texte)
Ceci est un texte.

>>> print(d)
Ceci est un texte.

>>> d.transformer_en_majuscules()
>>> print(d)
CECI EST UN TEXTE.
```

La classe Document

On pourrait également imaginer vouloir chiffrer ou déchiffrer ce document à l'aide du chiffrement de César. Ici,on ne considère que les lettres minuscules.

```
class Document:
    [...]
    def chiffrer(self, cle):
        alphabet = "abcdefghijklmnopqrstuvwxyz"
        nouveau_texte = ""

        for c in self.texte:
            if c in alphabet:
                index = alphabet.index(c)
                nouvel_index = (index + cle) % 26
                nouveau_texte += alphabet[nouvel_index]
            else:
                nouveau_texte += c

        self.texte = nouveau_texte
```

La classe Document

```
class Document:
    [...]

    def dechiffrer(self, cle):
        self.chiffrer(-cle)
```

La classe Document

Utilisation :

```
>>> d = Document("ceci est un texte!")
>>> d.chiffrer(8)
>>> print(d)
kmkq mab cv bmfbm!

>>> d.dechiffrer(8)
>>> print(d)
ceci est un texte!
```

La classe Document

Finalement, nous pourrions vouloir sauvegarder un document.

```
class Document:
    [...]

    def enregistrer_sous(self, nom_fichier):
        f = open(nom_fichier, 'w')
        f.write(self.texte)
        f.close()
```

Utilisation :

```
>>> d = Document("Ceci est un texte!")
>>> d.enregistrer_sous("mon_fichier.txt")
```

La classe `DocumentAvecNom`

La classe `DocumentAvecNom` représente un document qui contient un nom unique qui lui est propre. Elle diffère de la classe `Document` en deux points : ce type de document contient un nom, et une méthode `enregistrer()` qui ne prend pas d'argument en entrée (comme le document possède déjà un nom).

La classe DocumentAvecNom

Cette classe reste tout de même un Document, nous allons donc hériter de la classe Document.

```
class DocumentAvecNom(Document):
    def __init__(self, le_texte, nom):
        # Appel du constructeur de la classe de base.
        super().__init__(le_texte)

        # Initialisation de l'attribut supplémentaire nom.
        self.nom = nom

    def enregistrer(self):
        # Réutilisation de la méthode enregistrer_sous
        # déjà définie (car on hérite de Document).
        self.enregistrer_sous(self.nom + '.txt')
```

La classe DocumentAvecNom

On pourrait vouloir redéfinir la méthode `enregistrer_sous`, pour que celle-ci modifie également le nom du document.

```
class DocumentAvecNom(Document):
    [...]

    def enregistrer_sous(self, nom_fichier):
        # On extrait du nom du fichier la partie avant
        # l'extension.txt.
        self.nom = nom_fichier[:-4]

        # On appelle la méthode enregistrer_sous de
        # la classe Document.
        super().enregistrer_sous(nom_fichier)
```

La classe `Dossier`

Un `Dossier` est un **regroupement de documents nommés**. Il contient un dictionnaire dont la clé est le nom du document, et la valeur est le document en question. Une méthode `ajouter_document` permet d'ajouter un document.

La classe Dossier

```
class Dossier:
    def __init__(self):
        self.documents = {}  # Initialisation de l'attribut 'documents'

    def ajouter_document(self, document):
        # Ajout du document dans l'attribut 'documents',
        # qui est un dictionnaire.
        self.documents[document.nom] = document

    def __repr__(self):
        # Conversion pour l'affichage.
        ma_chaine = "Contenu du dossier: \n|"
        for cle, valeur in self.documents.items():
            ma_chaine += ("\n|- {}.txt : {}".format(
                cle, valeur.texte))
        ma_chaine += "\n|"
        return ma_chaine
```

La classe `Dossier`

Exemple d'utilisation :

```
>>> d_1 = DocumentAvecNom("Ceci est un document", "document1")
>>> d_2 = DocumentAvecNom("Un deuxième document", "document2")

>>> mon_dossier = Dossier()
>>> mon_dossier.ajouter_document(d_1)
>>> mon_dossier.ajouter_document(d_2)

>>> print(mon_dossier.documents["document1"].texte)
Ceci est un document

>>> mon_dossier.documents["document2"].enregistrer()
```

Lectures et travaux dirigés

Lectures et travaux dirigés

- Chapitres 11 et 12 de G. Swinnen
- Travaux dirigés : Exercices sur la programmation orientée objet.