Année académique 2024-25

HAUTE ECOLE EN HAINAUT
Campus technique

Campus technique
8a, avenue V.Maistriau B-7000 Mons
Tél : +32 (0)65 33 81 54
Fax : +32 (0)65 31 30 51
E-mail : tech-mons@heh.be
www.heh.be

Les Décorateurs

Bachelier en informatique et systèmes – Finalité Télécommunications et réseaux – Bloc 1

1 Les Décorateurs en Python : c'est quoi ?

Un **décorateur** est une façon élégante de modifier ou **améliorer le comportement** d'une fonction sans changer son code original.
En gros, c'est comme mettre un **costume ou un déguisement** à une fonction pour lui ajouter de nouveaux pouvoirs !

- On les reconnaît facilement en Python grâce à la syntaxe @.

2 Analogie Fun : Le costume de super-héros

Imaginons que vous avez une **fonction Python** qui est une **personne normale** :

```
def dire_bonjour():
    print("Bonjour tout le monde !")
```

Maintenant, imaginons que vous voulez donner à cette personne des **super-pouvoirs** (ex. : voler, courir plus vite).
Vous pourriez bien sûr modifier directement la fonction. Mais ce serait **ennuyeux et risqué** (comme une chirurgie faites par Mr Depreter !)

À la place, on peut simplement lui mettre un **costume de super-héros** qui lui donnera immédiatement des super-pouvoirs, sans la modifier en profondeur !

Ce costume, c'est un **décorateur** .

3 Exemple concret avec un décorateur

Imaginons un décorateur nommé costume_superheros. Ce décorateur permet à la fonction dire_bonjour() de faire quelque chose de spécial avant et après son appel :

```
def costume_superheros(fonction_normale):
    def super_pouvoirs():
        print("Le héros décolle dans les airs !")
        fonction_normale()
        print("Le héros repart à super-vitesse !")
    return super_pouvoirs

@costume_superheros
def dire_bonjour():
    print("Bonjour tout le monde !")

dire_bonjour()
```

Résultat :

Le héros décolle dans les airs !

Bonjour tout le monde !

Le héros repart à super-vitesse !

4 Explications détaillées ligne par ligne :

① Création du décorateur :

def costume_superheros(fonction_normale):

- Le décorateur est une fonction prenant une autre fonction en paramètre.

② Création d'une fonction interne :

def super_pouvoirs():

- Cette fonction interne (appelée aussi wrapper) définit les nouveaux pouvoirs que nous voulons ajouter.

③ Utilisation de la fonction d'origine :

fonction_normale()

- On appelle la fonction originale (sans modification) au milieu du nouveau comportement.

④ Retour du nouveau comportement :

return super_pouvoirs

- Le décorateur retourne la nouvelle fonction (le costume amélioré).

⑤ Application du décorateur :

@costume_superheros

- On applique ce décorateur à dire_bonjour(), ce qui revient à faire :

dire_bonjour = costume_superheros(dire_bonjour)

5 Règles importantes des décorateurs

1. ☑ Un décorateur doit **prendre une fonction en paramètre**.
2. ☑ Il doit **retourner une nouvelle fonction (wrapper)**.
3. ☑ Il ne doit **pas modifier directement la fonction originale**, seulement l'enrober.
4. ☑ On peut cumuler plusieurs décorateurs sur une même fonction :

```
@decorateur1
@decorateur2
def ma_fonction():
    pass
```

6 Pourquoi utilise-t-on des décorateurs ?

- Pour **ajouter des fonctionnalités communes** (ex. : logs, vérifications, sécurité).
- Pour **éviter de répéter du code** (principe DRY : Don't Repeat Yourself).
- Pour rendre le code plus **propre**, **modulaire**, et **élégant**.

7 Exemple de décorateur courant : @patch (Mock)

Le décorateur @patch() est très utilisé dans les tests unitaires (mocking). Il **remplace temporairement** une fonction réelle par une fonction simulée (mock) :

Analogie :
Imaginez que votre fonction réelle est une voiture, mais lors d'un test, vous voulez éviter d'utiliser la vraie voiture.

Le décorateur @patch() vous permet de remplacer temporairement la vraie voiture par une **voiture miniature télécommandée** :

Exemple :

```
from unittest.mock import patch
import unittest
def voiture():
    return "Je suis la vraie voiture"
class TestVoiture(unittest.TestCase):
    @patch("__main__.voiture", return_value="Je suis une voiture miniature")
    def test_voiture(self, mock_voiture):
        resultat = voiture()
        self.assertEqual(resultat, "Je suis une voiture miniature")
if __name__ == "__main__":
    unittest.main()
```

Ici :

- Le décorateur @patch() remplace la fonction voiture() par une fonction simulée.
- Le mock retourne "Je suis une voiture miniature".

8 Exercices:

- **Exercice 1 : Le costume de pirate**

Créer un décorateur @pirate qui ajoute automatiquement "Arrrgh ! " avant d'appeler la fonction originale.

- **Exercice 2 : Décorateur de sécurité**

Créer un décorateur @verifier_age qui vérifie que l'utilisateur a plus de 18 ans avant d'exécuter la fonction protégée.

9 Conclusion : Pourquoi maîtriser les décorateurs ?

Les décorateurs sont essentiels pour écrire du code Python :

- ☑ **propre**,
- ☑ **réutilisable**,
- ☑ **élégant**,
- ☑ **facilement maintenable.**

Les comprendre vous permettra également d'utiliser efficacement d'autres décorateurs Python très courants (@patch, @staticmethod, @property, etc.).

10 Résumé :

« Un décorateur Python, c'est comme mettre une cape de super-héros à votre fonction pour lui offrir instantanément de nouveaux pouvoirs ! »