Les Classes

Les classes sont un concept fondamental en programmation orientée objet (POO) utilisées dans de nombreux langages, y compris Python et MicroPython. Elles sont particulièrement utiles lors de la programmation de robots, car elles permettent d'organiser et de structurer le code de manière modulaire. Dans cet article, nous allons explorer ce qu'est une classe, comment la créer en Python/MicroPython, et comment l'utiliser pour créer des objets robotiques.

3GMS Découverte

Qu'est-ce qu'une classe ?

Une classe est un **modèle** ou un plan pour créer des objets. Elle définit les **attributs** (variables) et les **méthodes** (fonctions) qui seront associés aux objets créés à partir de cette classe. En d'autres termes, une classe est une sorte de "**moule**" à partir duquel des objets spécifiques peuvent être créés.

Une classe peut être comparée à un plan ou à un **modèle** conceptuel **pour créer des objets**. Elle définit la **structure** et le **comportement** que chaque objet créé à partir de cette classe devra suivre. Dans notre exemple de classe Robot, la classe elle-même spécifie qu'un robot aura un nom et pourra effectuer des actions telles que "avancer" ou "tourner à gauche". Cependant, une classe seule n'est qu'une abstraction, c'est-à-dire une idée théorique de ce à quoi ressemble un objet.

Les objets, en revanche, sont des **instances** concrètes de ces classes. Ils sont créés à partir du modèle fourni par la classe et portent avec eux toutes les caractéristiques et les fonctionnalités définies par cette classe. Prenons l'exemple des robots r1 et r2 que nous avons créés à partir de la classe Robot. Chacun de ces robots a son propre nom distinct (r1 et r2) et peut effectuer les mêmes actions, mais ils le font de manière indépendante. Les objets sont essentiellement des "réalisations" de la classe, où la classe est le plan et les objets sont les instances spécifiques conformes à ce plan. Cette approche de modélisation par classes et objets rend la programmation plus structurée, réutilisable et facile à maintenir, ce qui est particulièrement bénéfique dans des domaines tels que la robotique où la complexité peut être élevée.

Création d'une classe en Python/MicroPython

Pour créer une classe en Python/MicroPython, vous utilisez le mot-clé `class`, suivi du nom de la classe (généralement écrit en CamelCase) et de deux points. Ensuite, vous définissez les attributs et les méthodes de la classe à l'intérieur de blocs d'indentation. Voici un exemple simple de classe en Python :

```python
class Robot:
    def __init__(self, name):
        self.name = name

    def move_forward(self):
        print(f"{self.name} avance.")

    def turn_left(self):
        print(f"{self.name} tourne à gauche.")
```

Dans cet exemple, nous avons créé une classe `Robot` avec trois méthodes : `__init__()`, `move_forward()`, et `turn_left()`. La méthode `__init__()` est spéciale, elle est appelée lorsque nous créons un nouvel objet de la classe et nous permet d'initialiser ses attributs.

Création d'objets à partir d'une classe

Une fois que nous avons défini une classe, nous pouvons créer des objets (ou instances) de cette classe. Voici comment créer un robot à partir de notre classe `Robot` :

```python
r1 = Robot("Robot1")
r2 = Robot("Robot2")
```

Maintenant, nous avons deux robots, `r1` et `r2`, chacun avec son propre nom.

Utilisation des méthodes de la classe

Les méthodes d'une classe sont des fonctions qui peuvent être appelées sur les objets de cette classe. Par exemple, pour faire avancer notre robot `r1`, nous pouvons appeler la méthode `move_forward()` de la manière suivante :

```python
r1.move_forward()
```

Cela affichera "Robot1 avance." sur la console. De même, pour faire tourner à gauche `r2`, nous utilisons la méthode `turn_left()` :

```python
r2.turn_left()
```

Cela affichera "Robot2 tourne à gauche.".

`__init__`

La méthode `__init__()` (prononcée "init double souligné" ou "init") est une méthode spéciale en Python et MicroPython qui **est automatiquement appelée lors de la création d'un nouvel objet à partir d'une classe**. Elle est

également connue sous le nom de **constructeur**. La principale fonction de la méthode `__init__()` est **d'initialiser les attributs de l'objet nouvellement créé**.

Voici comment définir et utiliser la méthode `__init__()` dans une classe :

```
class MaClasse:
    def __init__(self, parametre1, parametre2):
        self.attribut1 = parametre1
        self.attribut2 = parametre2
```

Dans cet exemple, la méthode `__init__()` est définie avec trois paramètres : `self`, `parametre1`, et `parametre2`. Le paramètre `self` est automatiquement passé lorsque vous créez un objet à partir de la classe, et il fait référence à l'instance de l'objet lui-même.

Lorsqu'un nouvel objet est créé à partir de cette classe, la méthode `__init__()` est appelée automatiquement et les valeurs fournies pour `parametre1` et `parametre2` sont utilisées pour initialiser les attributs `attribut1` et `attribut2` de l'objet. Par exemple :

```
mon_objet = MaClasse("Valeur1", "Valeur2")
```

Dans cet exemple, `mon_objet` est créé à partir de la classe `MaClasse`, et la méthode `__init__()` est appelée avec les valeurs "Valeur1" et "Valeur2". En conséquence, `attribut1` de `mon_objet` sera égal à "Valeur1" et `attribut2` sera égal à "Valeur2".

Dans notre classe "Robot", la méthode `__init__()` est appelée avec la valeur "Robot1". En conséquence, l'attribut `name` de cet objet aura pour valeur `Robot1`.

La méthode `__init__()` est essentielle car elle permet de garantir que les objets créés à partir de la classe ont un état initial cohérent. Elle peut également être utilisée pour effectuer d'autres opérations d'initialisation complexes, telles que l'ouverture de fichiers, la configuration de connexions réseau, ou d'autres tâches nécessaires pour préparer l'objet à être utilisé dans le reste du programme.

Self

Le paramètre "`self`" est un concept fondamental en Python et MicroPython lors de la création de méthodes dans une classe. Il représente l'**instance actuelle de la classe**. En d'autres termes, "`self`" est utilisé pour faire référence à l'objet lui-même à l'intérieur des méthodes de la classe.

Lorsque vous définissez des **méthodes dans une classe**, vous devez inclure le paramètre "`self`" comme premier paramètre dans la liste des paramètres de la méthode. Par convention, le nom "`self`" est utilisé, mais vous pourriez techniquement utiliser n'importe quel autre nom, bien que cela soit déconseillé pour des raisons de clarté.

Voici pourquoi le paramètre "`self`" est important :

1. **Accès aux attributs de l'objet** : En utilisant "`self`", vous pouvez **accéder aux attributs spécifiques de l'objet** créé à partir de la classe. Par exemple, dans la méthode `move_forward()` de notre classe `Robot`, nous pouvons accéder à l'attribut `name` de l'objet `self` pour afficher le nom du robot actuel.

```
def move_forward(self):
    print(f"{self.name} avance.")
```

2. **Appel d'autres méthodes de la classe** : Vous pouvez également utiliser "`self`" pour appeler d'autres méthodes de la classe à l'intérieur de la méthode actuelle. Cela permet aux méthodes de collaborer entre elles et de partager des informations.

```
def move_and_turn(self):
    self.move_forward()
    self.turn_left()
```

3. **Création et manipulation d'attributs** : "`self`" est utilisé pour créer et manipuler des attributs de l'objet. Par exemple, dans la méthode `set_speed()` de la classe `RobotMotor`, nous utilisons "`self`" pour mettre à jour l'attribut `speed` de l'objet.

```
def set_speed(self, speed):
    self.speed = speed
    print(f"{self.name} a maintenant une vitesse de {speed}.")
```

En résumé, le paramètre "`self`" est essentiel pour que les méthodes d'une classe puissent interagir avec les données et le comportement de l'objet auquel elles sont associées. Il permet de créer des objets avec leurs propres caractéristiques uniques et de gérer leur état et leur comportement de manière encapsulée, ce qui est un concept clé de la programmation orientée objet.

Exemple concret : Utilisation de classes en robotique

Supposons que vous travaillez avec des robots équipés de moteurs et de capteurs. Vous pouvez utiliser des classes pour modéliser ces robots de manière efficace. Voici un exemple simplifié d'une classe `RobotMotor` en Python/MicroPython qui pourrait être utilisée pour contrôler les moteurs d'un robot :

```
class RobotMotor:
    def __init__(self, motor_name):
        self.name = motor_name
        self.speed = 0

    def set_speed(self, speed):
        self.speed = speed
        print(f"{self.name} a maintenant une vitesse de {speed}.")

    def stop(self):
        self.speed = 0
        print(f"{self.name} s'est arrêté.")
```

Dans cet exemple, `RobotMotor` représente un moteur de robot avec des méthodes pour régler la vitesse et l'arrêter. Vous pourriez créer plusieurs objets `RobotMotor` pour chaque moteur de votre robot et les contrôler individuellement.

En conclusion, les classes sont un outil puissant en Python et MicroPython pour organiser le code et modéliser des objets de manière efficace, ce qui est particulièrement utile en robotique. Vous pouvez les utiliser pour créer des objets robotiques modulaires et réutilisables, ce qui facilite la programmation et la maintenance de vos projets.