



Classes et méthodes abstraites

Gestion des exceptions

Programmation Python Avancée


Concepts de P00 - suite

- Classes et méthodes abstraites.
- Variables de classe et d'instance.
- Méthodes d'instance, de classe et statiques.
- Gestion d'exceptions



Abstraction

Classe abstraite :

- Elle sert de classe mère où l'on peut définir des méthodes dont les classes filles hériteront pour les implémenter.
 - Une classe abstraite n'est pas instanciable, elle ne contient pas de `__init__()`. Elle n'est utilisable que par le biais de l'héritage.
 - La classe abstraite contient des méthodes d'instance et de classes abstraites.
 - On définit une classe abstraite en la faisant hériter de la classe `ABC` du module `abc` de python.
- 

Abstraction

Déclaration d'une classe abstraite :

```
from abc import ABC
```

```
class Person(ABC):
```

```
    pass
```



Abstraction

Méthode abstraite :

- Une méthode abstraite définit seulement la signature d'une méthode mais n'a pas de corps.
- On ajoute le décorateur `@abstractmethod` pour obliger les classes qui héritent de la classe abstraite à implémenter la méthode et donc définir son corps.



Abstraction

Déclaration d'une méthode abstraite :

```
from abc import ABC
```

```
class Person(ABC):
```

```
    def manger():
```

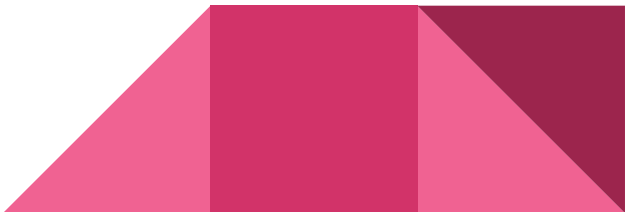
```
        pass
```

```
# Le décorateur va forcer l'implémentation de cette méthode par la classe fille
```

```
@abstractmethod
```

```
def marcher():
```

```
    pass
```



Variable d'instance VS variable de classe

- Variable d'instance : variable liée à une instance. Exemple, dans la classe `Circle`, un rayon est un attribut d'instance car il diffère en fonction de l'objet qui instancie la classe.
- Variable de classe : variable liée à une classe. Exemple, dans la classe `Circle`, `PI` est une variable de classe car c'est la même pour tous les objets de type `Circle`, elle est seulement spécifique à la classe `Circle` et non pas à une instance donnée.



Méthode d'instance, de classe et statique

Méthode d'instance : modifie l'état de l'objet.

- Exemple, la classe `Circle` contient une méthode `get_area(self)` qui permet de calculer la propriété surface de l'objet cercle avec un rayon d'une valeur donnée.
- Les méthodes d'instances prennent en paramètre `self` pour indiquer l'instance actuelle.
- Par défaut, toute méthode créée dans une classe est une méthode d'instance.
- Appel : `nom objet.nom methode()` ->
`circle_obj.get_area()`



Méthode d'instance, de classe et statique

Méthode de classe : modifie l'état de la classe et non pas de l'instance.

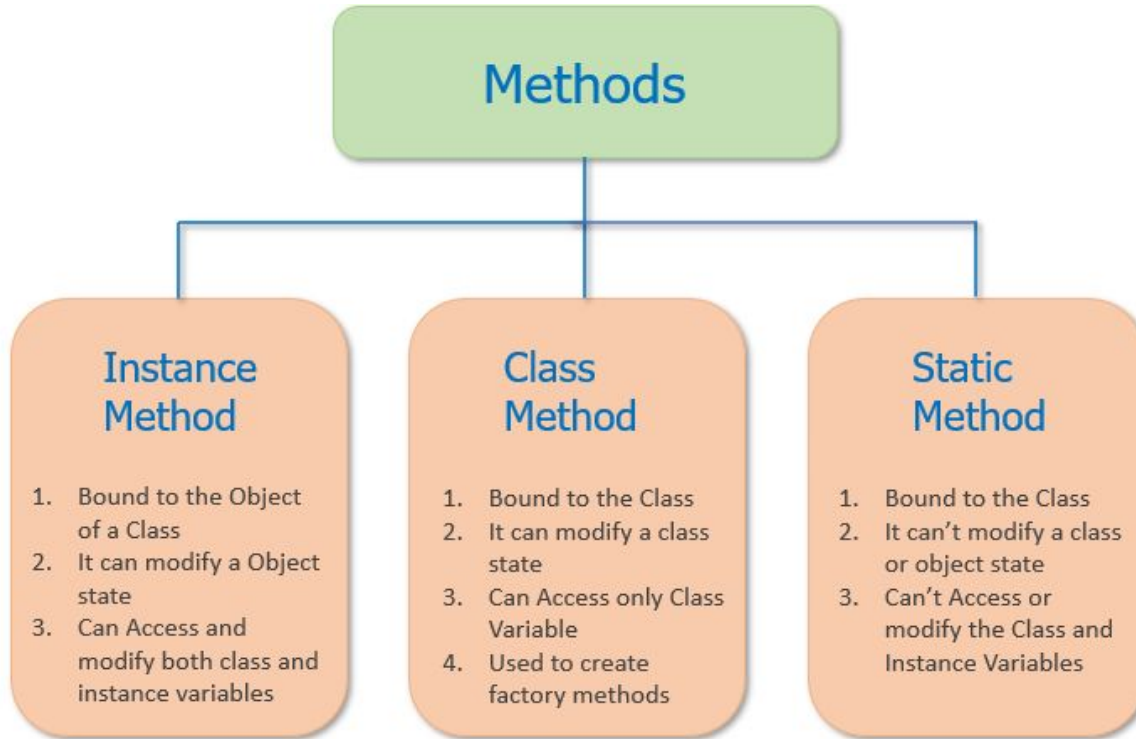
- Exemple, dans la classe `Circle` la méthode `modifier_pi()` permet de modifier la valeur `PI` pour la classe `Circle` et ainsi toutes les instances de cette classe seront affectées.
- Les méthodes de classe prennent en paramètre `cls` pour indiquer la classe actuelle.
- On indique à Python qu'une méthode est une méthode de classe en ajoutant le décorateur : `@classmethod`
- Appel : `nom_objet.nom_methode()` **ou** `nom_classe.nom_methode()`
-> `circle_obj.modifier_pi()` **ou** `Circle.modifier_pi()`

Méthode d'instance, de classe et statique

Méthode statique : permet de faire un calcul ou autre indépendamment des attributs d'instance et de classe. Mais qui reste liée à la classe conceptuellement.

- Exemple, pour la classe Circle on pourrait avoir une méthode statique qui permet de vérifier si deux rayons sont égaux `egal(rayon1: float, rayon2: float)`.
- Les méthodes statiques ne prennent pas en paramètre `self` ou `cls` car elles en sont indépendantes.
- On indique à Python qu'une méthode est statique en ajoutant le décorateur : `@staticmethod`
- **Appel :** `nom_objet.nom_methode()` ou `nom_classe.nom_methode()` -> `circle_obj.egal(val1, val2)` ou `Circle.egal(val1, val2)`

Méthode d'instance, de classe et statique



Source :
<https://pynative.com/python-class-method-vs-static-method-vs-instance-method/>

Gestion des exceptions

- Déclencher une exception :


```
def set_radius(value):  
    if value <= 0 :  
        raise Exception("Please enter a positive number")
```



Gestion des exceptions

- Gérer une exception :

```
try: # Tentative d'exécution de la ligne de code ci-dessous
    set_radius(value=-1)
except Exception as e: # Exécuté quand l'exception est déclenchée
    print(e)
finally : # Exécuté tout le temps
    print("End of script")
```



Gestion des exceptions

- Créer sa propre exception :

```
class MyException(Exception):  
    def __init__(self, message):  
        super() . __init__(message)
```

