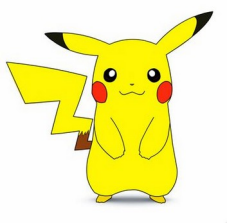
La programmation orientée objet

Introduction

Mise en situation

Simulation de combats de Pokemon



Caractéristiques d'un Pokemon

- Nom :
- Points de vie :
- Attaque
- Défense
- Niveau

Que fait un Pokemon ?

- Attaquer
- Subir des dégâts

Mise en situation

Choix de la structure de données

Les structures de données ?

Les structures de données permettent d'organiser les données dans un format adapté aux machines afin que les informations puissent être organisées, traitées, stockées et récupérées efficacement.

Quelle structure de donnée utiliser pour stocker les caractéristiques des Pokemon ?

Comment implémenter en Python ce que peut faire un Pokemon (comportements ou actions) ?

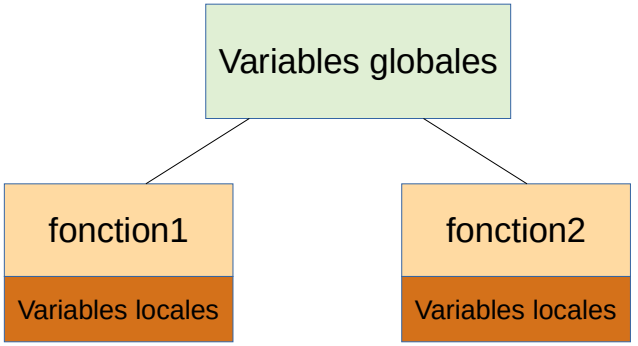
Paradigmes de programmation

Programmation procédurale

La programmation procédurale consiste à définir des procédures et des fonctions qui sont des regroupements d'instructions. Les fonctions et procédures s'exécutent dans un ordre spécifique pour résoudre un problème.

On utilise :

- des variables (globales) qui contiennent les données
- des fonctions qui peuvent modifier une variable



Paradigmes de programmation

Objectifs :

- **une meilleure organisation du code**
- **cacher des données et des détails d'implémentations.**

Solution : Regrouper les données et ce qu'on peut faire avec dans des entités nommées objets.

objet

Notion d’objet

Un **objet** est une entité qui regroupe **les données** et **ce qu’on peut faire avec ces données**.

Il est inutile de connaître le fonctionnement interne de l’objet pour l’utiliser.

Un objet est une entité informatique comprenant :

- un ensemble d'**attributs** (ou **variables d’instances**)

  des variables propres à l'objet

- des méthodes

  des fonctions propres à l’objet

On appelle **encapsulation** le regroupement des variables et des fonctions au sein d'une même entité.

La programmation orientée objet, c'est créer du code source mais qui sera encapsulé en le plaçant dans un conteneur (un objet).

Notion d’objet

L'accès aux données et méthodes peut être réglementé :

Un objet possède un état :

L’état correspond à la valeur de ses attributs à un à instant donné.

Il peut varier au cours du temps.

L’état actuel est lié aux comportements passés.

Un objet possède une identité :

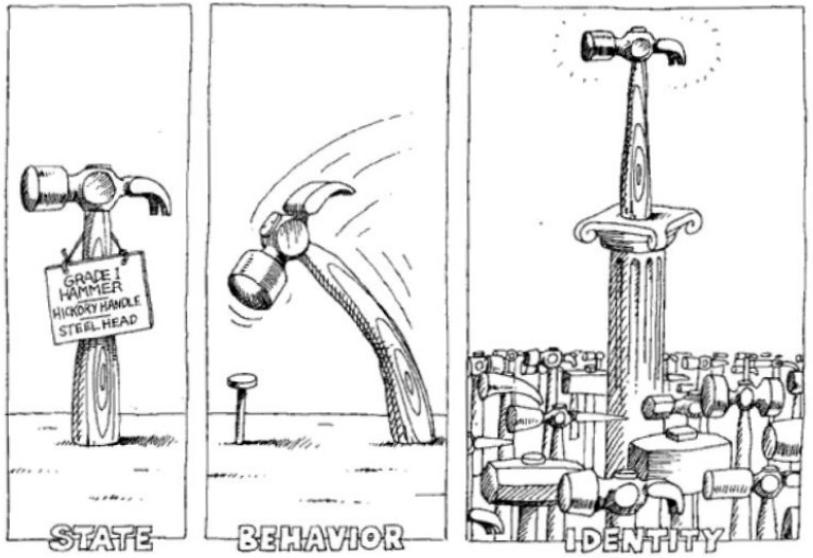
Les objets peuvent être distingués même si tous leurs attributs ont des valeurs identiques.

Deux objets sont distincts même si tous leurs attributs ont des valeurs identiques.

Notion d’objet

Comportement d’un objet :

- Décrit par les méthodes (fonctions membres)
- Actions & réactions aux sollicitations extérieures
- Comportement dépend de l’état (attributs)



Notion de classe

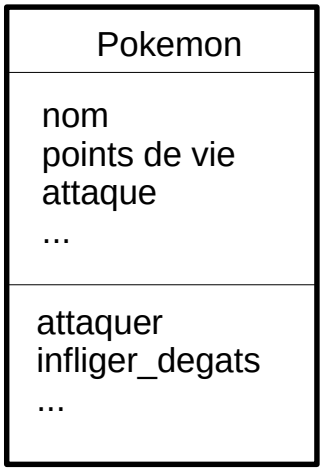
La classe c’est 'le moule' qui permet de construire des objets.

Pour construire une maison, on utilise un plan d’architecte.
La classe correspond au plan, l’objet à la maison.

Un objet est décrit par une classe :

- Une classe est un **prototype** qui définit des attributs et des méthodes des objets.
- C’est donc un modèle utilisé pour créer plusieurs objets présentant des caractéristiques communes.

Classe Pokemon



Une classe permet de définir ce qu’est un Pokemon.

Classe et instance de classe

La classe Pokemon permet de construire le pokemon pikachu.

On dit que pikachu est une instance de la classe Pokemon.

La méthode qui permet de créer l'objet est appelée constructeur.

Ne pas confondre **instance de classe** et **classe**

Une instance de classe fait référence à une chose précise
Une classe désigne un groupe de choses similaires

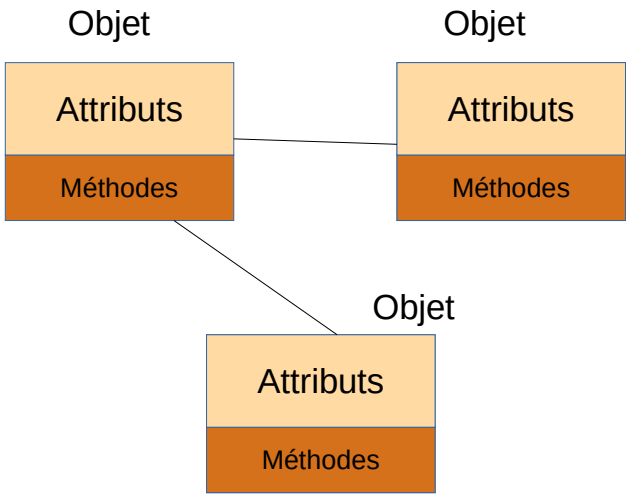
Le vélo de mon voisin et le mien sont deux instances de

la classe vélo, même si ils sont identiques.

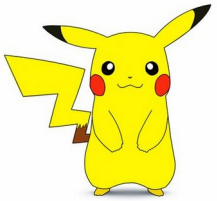
Paradigmes de programmation

Programmation orientée objet

La POO se concentre sur la représentation des problèmes à l'aide d'objets. Un objet représente un concept ou une entité du monde réel et définit ses caractéristiques et ses comportements.



Application :
Simuler des combats de Pokemons

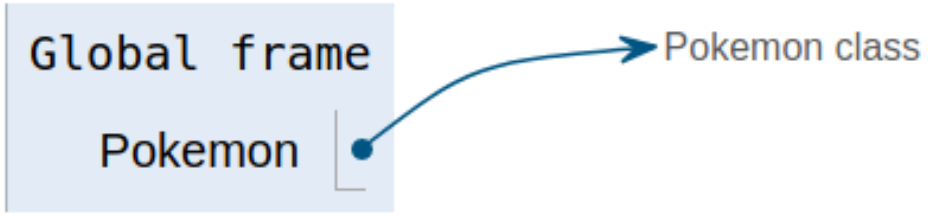


Définir la classe Pokemon

```
class Pokemon:
    pass

print(Pokemon)
```

```
<class '__main__.Pokemon'>
```



Application :
Simuler des combats de Pokemons

La classe Pokemon et l'objet Pikachu

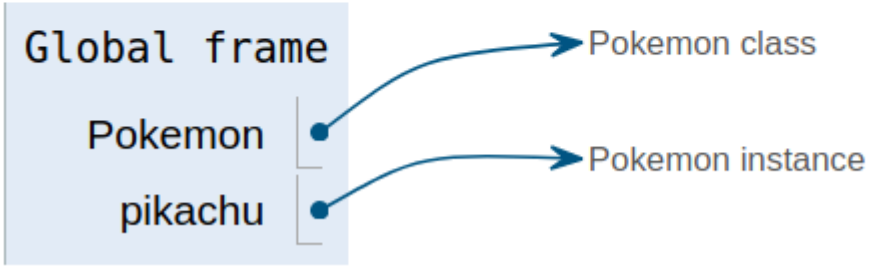
```
class Pokemon:
    pass

print(Pokemon)

pikachu = Pokemon()
```

pikachu est une instance de la classe Pokemon

```
<class '__main__.Pokemon'>
```



Application :
Simuler des combats de Pokemons

La méthode __init__() permet d'initialiser les attributs de l'objet. Elle est appelée juste après la création de l'objet par la méthode spéciale __new__().

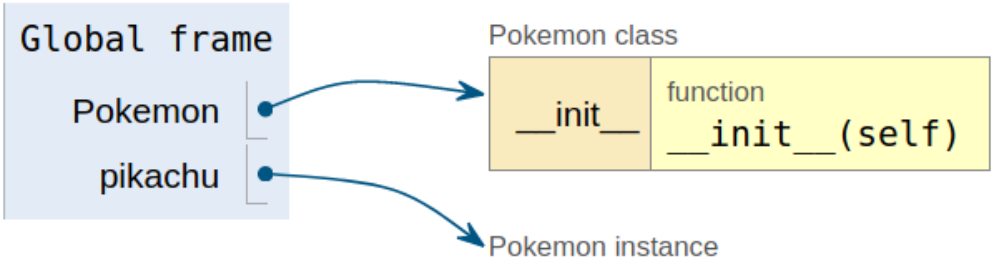
L'instance (l'objet) est représenté par le mot clé self.

```
class Pokemon:
    def __init__(self):
        print('Création d\'une instance')

print(Pokemon)

pikachu = Pokemon()
```

```
<class '__main__.Pokemon'>
Création d'une instance
```



Application : Simuler des combats de Pokemons

Initialiser les attributs

On peut passer la valeur initiale des attributs comme argument de la fonction __init__()

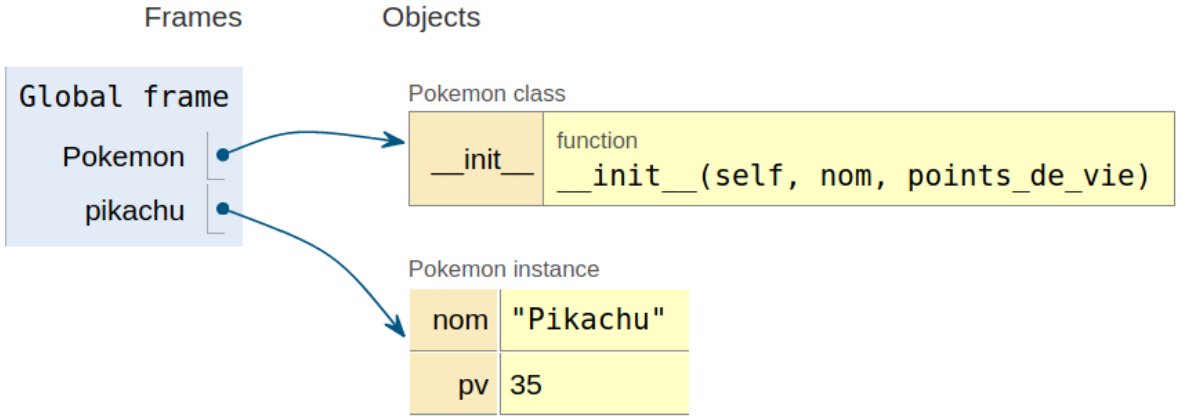
```
class Pokemon:
    def __init__(self, nom, points_de_vie):
        self.nom = nom
        self.pv = points_de_vie

pikachu = Pokemon('Pikachu', 35)
print(pikachu.nom)
print(pikachu.pv)
```

Mauvaise pratique

Néanmoins en Python dans certains cas ce sera toléré.

```
Pikachu
35
```



Application :
Simuler des combats de Pokemons

Indiquer que les attributs sont privés.

```
class Pokemon:
    def __init__(self, nom, points_de_vie):
        self._nom = nom
        self._pv = points_de_vie

pikachu = Pokemon('Pikachu', 35)
print(pikachu._nom)
print(pikachu._pv)
```

Il s'agit d'une simple indication

En Python les attributs non publics sont des attributs dont le nom commence par un ou deux caractère(s) underscore. Par exemple, _attribut, ou __attribut.

Application : Simuler des combats de Pokemons

Les assesseurs : ajouter des méthodes pour accéder aux attributs

```
class Pokemon:
    def __init__(self, nom, points_de_vie):
        self._nom = nom
        self._pv = points_de_vie

    def get_nom(self):
        return self._nom

    def get_pv(self):
        return self._pv


Pikachu = Pokemon('Pikachu', 35)
print(Pikachu.get_nom())
print(Pikachu.get_pv())
```

```
Pikachu
35
```

Application : Simuler des combats de Pokemons

Les mutateurs : ajouter des méthodes pour modifier les attributs

```
class Pokemon:
    def __init__(self, nom, points_de_vie):
        self._nom = nom
        self._pv = points_de_vie

    def get_nom(self):
        return self._nom

    def get_pv(self):
        return self._pv

    def set_degats(self, val):
        self._pv = self._pv - val
```

Et si val > self.pv ?

```
pikachu = Pokemon('Pikachu', 35)
print(pikachu.get_pv())
pikachu.set_degats(5)
print(pikachu.get_pv())
```

Affichage

```
35
30
```

Application :
Simuler des combats de Pokemons

La fonction print

```
pikachu = Pokemon('pikachu', 35)
print(pikachu)
```

→ <__main__.Pokemon object at 0xb5f27810>

Modifier le comportement de la fonction print

Ajout de la méthode :

```
def __str__(self):
    return str(self.get_nom()) + '\n\tPoints de vie:' + str(self.get_pv())
```

→ pikachu
Points de vie:35