



PYTHON & ALGORITHMIQUE

Bastien Gorissen & Thomas Stassin

READY ?

Press start!

LEVEL 1-9

Rappel

LES DÉFINITIONS IMPORTANTES

Classe: Description d'un type d'objet.

Objet: Représentant d'une classe.

Instance: Un objet est une instance d'une classe.

Attribut: Variable de classe.

Méthode: Fonction de classe.

self: Référence à l'objet lui-même (à usage interne à la classe)

OU ENCORE, NIVEAU SYNTAXE :

```
class MyClass:  
    def __init__(self, arg1):  
        self.attribute = arg1  
  
    def my_method(self):  
        print(self.attribute)
```

Et création d'un objet:

```
my_obj = MyClass("test")
```

LEVEL 1-10

Comme un relent de déjà vu :)

CLASSE ET JEU DE RÔLE, LE RETOUR

On peut faire le parallèle entre les **classes** d'un langage de programmation et celles d'un jeu de rôle, et pareil entre les **objets** et les personnages.

CLASSE ET JEU DE RÔLE, LE RETOUR

Une **classe** regroupe toutes les caractéristiques qui définissent un **type**, de la même façon que dans un RPG, la classe "mage" définit ce qu'est un mage, quels types de magie il peut utiliser et quelles sont ses caractéristiques.



CLASSE ET JEU DE RÔLE, LE RETOUR

Et de la même façon que dans un groupe de personnages de RPG vous pouvez avoir plusieurs mages étant différents les uns des autres, dans votre code, vous aurez plusieurs **objets** partageant la même **classe**.



LEVEL 1-11

Encore une couche ?

```
class Carrot:

    def __init__(self):

        self.sale_price = 5

        self.growth_max = 7

        self.growth = 0

    def grow(self):

        self.growth += 1
```

```
carrot = Carrot()
```

```
carrot.grow()
```

```
class Carrot:
```

```
    def __init__(self):
```

```
        self.sale_price = 5
```

```
        self.growth_max = 7
```

```
        self.growth = 0
```

```
    def grow(self):
```

```
        self.growth += 1
```

Classe

Une classe est un type d'objet.

Dans le code d'une classe on trouve tout ce qui définit un type, comme les méthodes et les attributs.

```
carrot = Carrot()
```

```
carrot.grow()
```

Instance
de la
classe

Une instance de la class est un objet (ou variable) du type de la classe.

L'instance possède les attributs et les méthodes de sa classe.

```
class carrot:
```

```
    def __init__(self):
```

```
        self.sale_price = 5.
```

```
        self.growth_max = 7.
```

```
        self.growth = 0.
```

```
    def grow(self):
```

```
        self.growth += 1.
```

```
carrot = Carrot()
```

```
carrot.grow()
```

Attribut
de la
classe

**Un attribut est une
variable de classe.**

Les attributs sont les
variables propres à la
classe.

```
class carrot:
```

```
    def __init__(self):
```

```
        self.sale_price = 5.
```

```
        self.growth_max = 7.
```

```
        self.growth = 0.
```

```
    def grow(self):
```

```
        self.growth += 1.
```

Méthode de
la classe

```
carrot = Carrot()
```

```
carrot.grow()
```

Appel de la
méthode

**Une méthode est une
fonction de classe.**

La méthode est une
fonction propre à la
classe n'agissant que sur
la classe et ses attributs

**Une instance de classe peut
faire appel aux méthodes de
sa classe.**

```
class carrot:
```

```
def __init__(self):
```

```
    self.sale_price = 5.
```

```
    self.growth_max = 7.
```

```
    self.growth = 0.
```

```
def grow(self):
```

```
    self.growth += 1.
```

```
carrot = Carrot()
```

```
carrot.grow()
```

Constructeur
de la classe

**Le constructeur est la
méthode qui crée la classe**

Cette méthode est appelée
lors de la l'instanciation
de la class
e.

Instanciation de
la classe

```
class carrot:
```

```
    def __init__(self):
```

```
        self.sale_price = 5.
```

```
        self.growth_max = 7.
```

```
        self.growth = 0.
```

```
    def grow(self):
```

```
        self.growth += 1.
```

“self” le
paramètre
fantôme

**self est le paramètre
obligatoire qui contient
l'instance de la classe**

Ce paramètre fait le lien
entre la méthode et le
reste de la classe.

```
carrot = Carrot()
```

```
carrot.grow()
```


LEVEL 1-12

Fighting Vegetables II - Electric Boogaloo

LÉGUMES BAGARREURS

Vous allez reprogrammer un combat de légumes (si, si, on y tiens VACHEMENT à notre thème...)

Chaque légume fera l'objet d'une classe différente:

Nous avons :

- la tomate tueuse
- le brocoli cogneur
- la carotte castagneuse

LÉGUMES BAGARREURS

Les légumes se battent sur un ring. Celui-ci sera aussi représenté par une classe.

On va y aller étape par étape, afin d'être sûrs de ne rien louper.

LÉGUMES BAGARREURS

La tomate tueuse:

Elle a:

- 10 points de vie
- 1 point de défense



Elle peut attaquer un adversaire, lui retirant entre 4 et 6 points de vie moins la défense de son opposant (minimum 0).

Faisons ensemble le code de la classe Tomate Tueuse ou KillerTomato...

```
class KillerTomato:
```

```
    def __init__(self):
```

```
        self.hp = 10
```

```
        self.hp_max = 10
```

```
        self.defense = 1
```

Pour commencer, écrivons
la classe et son
constructeur.

La Tomate tueuse:

Elle a:

- 10 points de vie
- 1 point de défense

```
class KillerTomato:

    def __init__(self):

        self.hp = 10

        self.hp_max = 10

        self.defense = 1
```

```
killer_tomato = KillerTomato()

print (killer_tomato.hp)
```

A ce stade on peut
afficher les points de vie
de la tomate tueuse pour
constater qu'elle est bien
en vie!

```
from random import randint
```

```
class KillerTomato:
```

```
    def __init__(self):
```

```
        self.hp = 10
```

```
        self.hp_max = 10
```

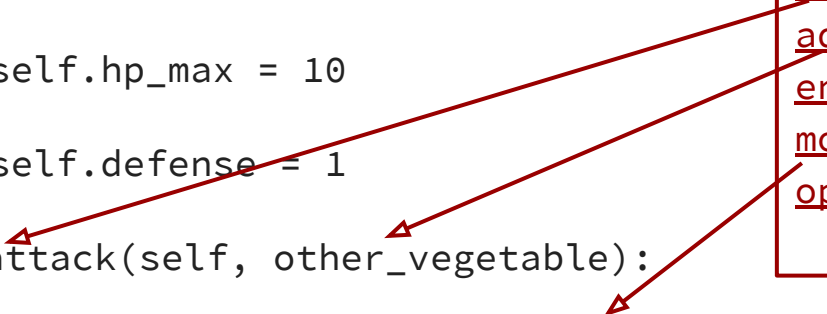
```
        self.defense = 1
```

```
    def attack(self, other_vegetable):
```

```
        damage = max(0, randint( 4, 6) - other_vegetable.defense)
```

```
        other_vegetable.hp -= damage
```

Elle peut attaquer un
adversaire lui retirant
entre 4 et 6 points de vie
moins la défense de son
opposant (minimum 0).



...

```
tomato_1 = KillerTomato()
```

```
tomato_2 = KillerTomato()
```

```
tomato_1.attack(tomato_2)
```

```
print(tomato_2.hp)
```

Si on fait attaquer une tomate tueuse avec une congénère, on constate que les points de vie de la tomate victime ont baissé.

VOILÀ LE CODE DE LA CLASSE DE LA TOMATE TUEUSE

```
from random import randint
```

```
class KillerTomato:
```

```
    def __init__(self):
```

```
        self.hp = 10
```

```
        self.hp_max = 10
```

```
        self.defense = 1
```

```
    def attack(self, other_vegetable):
```

```
        damage = max(0, randint( 4, 6) - other_vegatable.defense)
```

```
        other_vegetable.hp -= damage
```



LÉGUMES BAGARREURS

Le brocoli cogneur:

Il a:

- 10 points de vie
- 2 points de défense



Il peut attaquer un adversaire, lui retirant entre 1 et 4 points de vie moins la défense de son opposant (minimum 0).

Il retire deux fois plus de points de vie si il est en dessous de la moitié de ses points de vie originaux.

Je vous laisse faire le code du brocoli cogneur ou PuncherBroccoli

```
class PuncherBroccoli:
```

```
    def __init__(self):
```

```
        self.hp = 10
```

```
        self.hp_max = 10
```

```
        self.defense = 2
```

```
    def attacks(self, other_vegetable):
```

```
        factor = 1
```

```
        if self.hp < self.hp_max/2:
```

```
            factor = 2
```

```
        damage = max(0, randint(1, 4)) * factor
```

```
        damage -= other_vegetable.defense
```

```
        other_vegetable.hp -= damage
```

Le brocoli cogneur:

Il a:

- 10 points de vie
- 2 point de défense

Il peut attaquer un adversaire, lui retirant entre 1 et 4 points de vie moins la défense de son opposant (minimum 0).

Il retire deux fois plus de points de vie si il est en dessous de la moitié de ses points de vie originaux.

LÉGUMES BAGARREURS



La carotte castagneuse:

elle a:

- 8 points de vie
- 1 point de défense

Elle peut attaquer un adversaire lui retirant entre 3 et 5 points de vie moins la défense de son opposant (minimum 0).

De plus, si elle fait le maximum de dégâts (5 donc), elle récupère 1 point de vie (en ne dépassant pas 8).

Je vous laisse faire le code du carotte castagneuse ou BrawlerCarrot

```
class BrawlerCarrot:
```

```
    def __init__(self):
```

```
        self.hp = 8
```

```
        self.hp_max = 8
```

```
        self.defense = 1
```

```
    def attacks(self, other_vegetable):
```

```
        damage = randint(2, 5)
```

```
        if damage == 5 and self.hp < self.hp_max:
```

```
            self.hp += 1
```

```
        damage -= other_vegetable.defense
```

```
        damage = max(0, damage)
```

```
        other_vegetable.hp -= damage
```

La carotte castagneuse:

elle a:

- 8 points de vie
- 1 point de défense

Elle peut attaquer un adversaire lui retirant entre 3 et 5 points de vie moins la défense de son opposant (minimum 0).

De plus, si elle fait le maximum de dégâts (5 donc), elle récupère 1 point de vie (en ne dépassant pas 8).

LÉGUMES BAGARREURS

Normalement à ce stade
vous avez les classes des
trois légumes.



Il ne manque plus que le ring pour qu'ils se battent.

LÉGUMES BAGARREURS

Le ring permet à deux légumes de se battre.

Le combat ne concerne que deux légumes.

Chaque round, le premier légume attaque le second et vis-versa.

A la fin de chaque round, on affiche les points de vie de chaque combattant.

A la fin d'un round, le ring donne le gagnant si il y en a un. Le gagnant étant le premier qui met l'autre à 0 point de vie.

Voici à quoi devrait représenter la structure de la classe.

```
class Ring:
```

```
    def __init__(self, veg_1, veg_2):
```

```
        pass
```

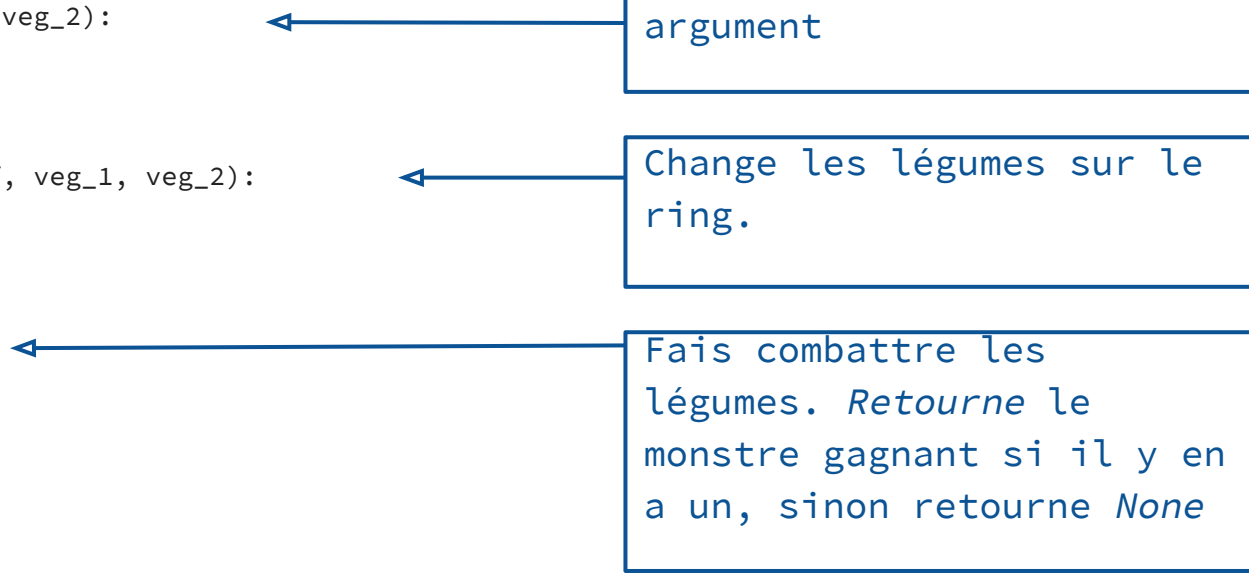
```
    def change_vegetables(self, veg_1, veg_2):
```

```
        pass
```

```
    def fight_round(self):
```

```
        pass
```

Constructeur.
Il prend 2 légumes en
argument



```
graph LR
    subgraph Box1 [ ]
        direction TB
        B1_1[Constructeur.]
        B1_2[Il prend 2 légumes en argument]
    end
    subgraph Box2 [ ]
        direction TB
        B2_1[Change les légumes sur le ring.]
    end
    subgraph Box3 [ ]
        direction TB
        B3_1[Fais combattre les légumes. Retourne le monstre gagnant si il y en a un, sinon retourne None]
    end
    Box1 --> L1[def __init__(self, veg_1, veg_2):]
    Box2 --> L2[def change_vegetables(self, veg_1, veg_2):]
    Box3 --> L3[def fight_round(self):]
```

Change les légumes sur le
ring.

Fais combattre les
légumes. *Retourne* le
monstre gagnant si il y en
a un, sinon retourne *None*


```
class Ring:
```

```
    def __init__(self, veg_1, veg_2):
```

```
        self.veg_1 = veg_1
```

```
        self.veg_2 = veg_2
```

```
    def change_vegetables(self, veg_1, veg_2):
```

```
        self.veg_1 = veg_1
```

```
        self.veg_2 = veg_2
```

Constructeur.

Il prend 2 légumes en arguments

Change les légumes sur le ring.

```
class Ring:
```

```
...
```

```
def fight_round(self):
```

```
    self.veg_1.attack(self.veg_2)
```

```
    if self.veg_2.hp <= 0:
```


```
        return self.veg_1
```

```
    self.veg_2.attacks(self.veg_1)
```

```
    if self.veg_1.hp <= 0:
```

```
        return self.veg_2
```

```
    return None
```



Fais combattre les légumes. *Retourne* le légume gagnant si il y en a un, sinon retourne *None*

LÉGUMES BAGARREURS

Il ne reste plus qu'à faire le script qui fait combattre les légumes entre eux.

Créez un légume de chaque type.

Faites deux combats par paire de légumes (un match aller et match retour) et comptez les points. N'oubliez pas de remettre les points de vie aux légumes avant de les refaire combattre.

Le légume qui a gagné le plus de matchs a gagné.

A la fin, affichez le gagnant.