DBSF - Préformation

Algorithme

Bastien Gorissen & Thomas Stassin

Ready ?

Press start!

LEVEL 6-1

“Combat de légumes!”

VEGGIE FIGHTER

Nous allons créer un jeu de combat de légumes:

Une carotte hargneuse va s’en prendre à une tomate tueuse, et nous allons simuler leurs combats sans merci.

Le jeu fonctionnera comme suit:

- Chaque combattant a trois attaques (poing, pied, morsure)
- Chaque attaque a une certaine force
- Le combat se joue en plusieurs rounds, au cours desquels les légumes vont choisir un coup chacun (au hasard)
- Si l’un des légumes est tombé à 0 points de vie, le combat se termine et un vainqueur (ou une égalité) est déclaré.

VEGGIE FIGHTER

Nous allons utiliser import pour séparer le jeu en lui-même des données des combattants.

Commençons par créer deux dictionnaire dans un module “**fighters.py**”:

Un pour la carotte hargneuse (**fighter1**), un autre pour la tomate tueuse (**fighter2**).

Dans chacun de ces dictionnaires, mettez une donnée pour le nom, une pour l’attaque au poing, une autre pour l’attaque au pied et une quatrième pour la morsure.

Pour chacune de ces données (ou clé), affectez une valeur (à votre convenance).

VEGGIE FIGHTER

Ajoutez ensuite encore une clé pour les points de vie et une autre pour la défense.

Une fois que c’est fait nous avons les données pour nos légumes combattants (ou notre robot tueur et notre t-rex laser).

Nous pouvons donc les importer dans notre script principal.

VEGGIE FIGHTER

Ensuite tant que l'un des deux n'est pas mort (c'est à dire que ses point de vie ne sont pas à 0 ou moins).

Faites-les combattre:

Pour le combat c'est très simple:

Pour l'un des deux combattants, le programme choisit une attaque (poing, pied, morsure)

En fonction de l'attaque choisie, récupérez la valeur dans le dictionnaire.

Calculez les dégâts qui sont égaux à l'attaque - la défense du perso attaqué.

Enlevez les dégâts aux points de vie du personnage attaqué.

VEGGIE FIGHTER

Refaites exactement la même opération pour l’autre personnage.

Pour chaque attaque, affichez l’attaque choisie ainsi que le nombre de dégâts infligés.

A la fin du combat (donc quand au moins l’un des deux combattants est mort), affichez le vainqueur.

Trois possibilités:

- Soit fighter1 gagne
- Soit fighter2 gagne
- Soit c’est un double KO

Level 6-2

“Fonction” (Warning Blague Star Wars incoming)

Fonction 101

Une fonction est un outil qui permet d'abstraire une partie du code afin d'augmenter la clarté et la lisibilité de celui-ci.

Nous avons déjà rencontré des fonctions dans les cours précédents: **len**, **print**, **input**, etc.

Nous allons voir comment créer nos propres fonctions

La recette d'une fonction

Le mot magique pour définir une fonction est **def**

Ce mot doit être suivi du nom de la fonction, suivi lui-même de parenthèses et de “:”.

```
def mega_function_of_death():

    # code qui arrache du steak de poney
```

Notez bien que les parenthèses ne resteront pas vides, on verra plus tard comment les remplir.

Appeler une fonction

Appeler une fonction est bête comme choux.

Il suffit de faire comme suit:

mega_function_of_death()

Encore une fois, c’est similaire à **print, len** et compagnie, puisque ce sont aussi des fonctions.

Les noms de vos fonctions doivent **TOUJOURS être EXPLICITES**.

Ceci afin qu’on ne doive pas lire l'intégralité du code de la fonction pour connaitre son utilité.

Maintenant que ceci est dit...

Guess the number, le retour...

Dans le “guess the number” , ajoutez une fonction **“bingo”** qui affiche "Bingo!" dans la console.

Dans le code principal du jeu, remplacez la partie qui dit au joueur qu'il a trouvé la bonne réponse par un appel à votre fonction.

Level 6-3

“Le retour”

```
def the_return_of_the_jedi():
    the_jedi = Jedi
    return the_jedi
```

Le retour...

Le retour d'une fonction est la valeur que renvoie la fonction au code appelant.

```
def dice_simulation():
    dice = randint(1, 6)
    return dice

result = dice_simulation()
print("Résultat du dé: " + str(result))
```

Ici la fonction renvoie la valeur contenue dans la variable **dice**.

Le retour...

C’est le mot clé **return** qui renvoie la valeur.

Noter bien que le fait de retourner une valeur arrête immédiatement le traitement de la fonction, pour *retourner* à l’appelant.

```
def weird_function():
    result = randint(1,6)
    return result

    print(“Texte qui ne sera jamais afficher”)
```

Dans ce script la dernière ligne ne sera jamais affichée car elle est située juste après le return.

Le BinGo Contre-Attaque

Faites en sorte de bouger dans une fonction la partie du code qui demande à l'utilisateur de choisir un chiffre.

Le retour de cette fonction sera le nombre choisi par l'utilisateur. (cette phrase vous est offerte par Yavanna)

Level 6-4

“Entre parenthèses”

Paramètre...

Toute fonction peut avoir un ou plusieurs paramètres.

```
from random import randint

dice = randint(1, 6)
```

Dans la fonction **randint** 1 et 6 sont les paramètres de la fonction.

...argument

Pour pouvoir utiliser ces paramètres une fonction a besoin d’arguments

```
def double(number):
    return number * 2
```

Ici **number** est l’argument de la fonction **double**

```
def list_multiplication(array, number):
    clone = []

    for n in array:
        clone.append(n * number)

    return clone
```

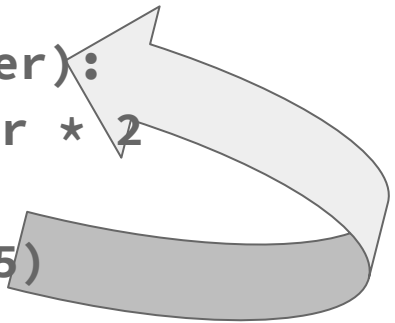
Et ici **array** et **number** sont les arguments de la fonction **list_multiplication**

Argument... Paramètres ...?

Lors de l'appel d'une fonction, le compilateur passe le valeur des paramètres dans les arguments respectifs:

```
def double(number):
    return number * 2

value = double(5)
```



Et bien sûr, il faut récupérer la valeur renvoyée dans une variable ou autre, sinon elle est perdue.

La menace Bingo

Dans le bingo, faites en sorte de sortir le code qui vérifie le nombre rentré par le joueur dans une fonction.

Le code affichera si le nombre est plus petit ou plus grand, et renverra **True** si le joueur à deviné juste et **False** dans les autres cas.

La fonction prendra en paramètres le nombre donné par le joueur ainsi que le nombre à deviner.

Level 6-5

“Valeur par défaut… ”

Courage...

Il y a moyen de donner une valeur par défaut à un argument.

Cette valeur lui sera donnée si le code appelant ne le fait pas.

```
def dice_simulation(faces=6):
    dice = random.randint(1, faces)
    return dice
```

Dans ce code l'argument **faces** vaudra 6 si l'appelant ne le définit pas.

Dernière chose nouvelle pour aujourd'hui...

result = dice_simulation(10)

Ici **faces** vaut 10

result = dice_simulation()

et ici **faces** vaut 6

Il est donc possible d'omettre des paramètres s'ils ont une valeur par défaut.

On peut bien-sûr mélanger

Il est évidemment possible de mélanger argument normal et argument avec valeur par défaut.

Mais, les arguments avec valeurs par défaut doivent toujours être en dernier dans la liste des arguments.

```
def dices_simulation(dice_number, faces=6):
    result = 0

    for n in range(dice_number):
        dice = random.randint(1, faces)
        result = result + dice
    return result
```

La revanche du Bingo

Dans le code du “guess the number”, modifiez la détermination des bornes du jeu à l'aide d'une fonction avec une valeur par défaut.

La première fois, le nombre est choisi de manière normale, mais à chaque fois que le joueur recommence de jouer, la borne maximum est modifiée de manière à valoir **born_max + (7 - tentatives)**.

La toute première fois où elle est appelée, la fonction n'aura pas de paramètre, et les autres fois elle aura le nombre de tentatives comme paramètre.