DBSF - Préformation

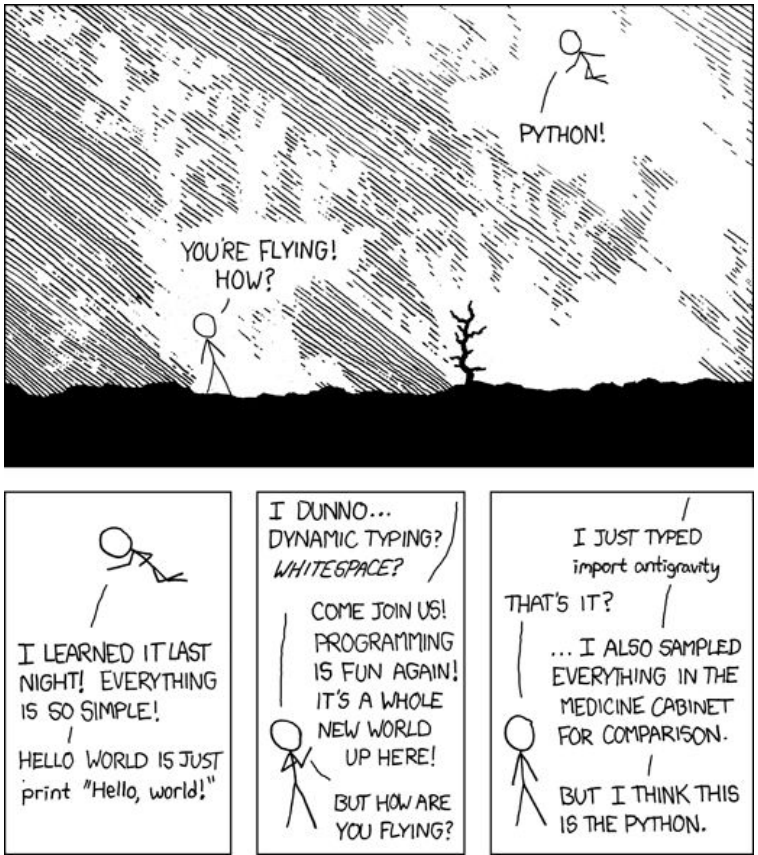
Algorithme

Bastien Gorissen & Thomas Stassin

Prologue

“Est-ce que vous avez bien dormi ?”

Je pense que j'ai rêvé en Python, c'est normal ?



Simple, simple...

Bon ok, on a vu beaucoup de choses la semaine passée…

- Exécuter du code
- **print()**
- Les variables
- Manipuler des **string**
- Utiliser **+**, **<**, **>=**, **and**, **or**, **not**, **...**
- Appeler des fonctions
- **str()**, **int()**, **float()**, **input()**, …
- **If … elif … else**

Et aujourd'hui ?

Le menu est plus léger *en apparence*.

- Terminer le sujet des nombres aléatoires
- Écrire des boucles pour répéter du code
- Apprendre à manipuler des listes
- Utiliser **range()**

Et nos objectifs du jour :

- Faire un jeu de devinette
- Résoudre un labyrinthe !
- Réaliser un petit jeu de pendu

Level 2-0

“Flashback”

NOMDRA

Beaucoup de jeux mettent à profit la notion de hasard pour offrir au joueur une expérience toujours renouvelée.

Bien entendu, derrière, ne se cache rien d'autre qu'une instruction spécifique destinée à générer des valeurs aléatoires.

En Python, ces fonctions sont reprises dans la librairie **random**, et nous allons surtout nous intéresser à la fonction **randint()**, qui sert à générer un entier aléatoire.

MARDON

Pour pouvoir utiliser **randint()**, nous allons devoir demander à Python de l'importer.

Pour ce faire, il faut placer en début de script une ligne telle que:

```
from random import randint
```

Ensuite, vous pouvez utiliser **randint()** comme toute autre fonction de Python:

```
d6 = randint(1, 6)
```

Rondma

d6 = randint(1, 6)

La fonction **randint()** accepte 2 arguments:

- Un minimum (ici, 1)
- Un maximum (ici, 6)

...et génère un nombre entier compris entre le minimum et le maximum (compris).

Il existe bien d’autres fonctions dans la librairie **random**, mais nous aurons l’occasion de les voir plus tard !

Exercice - Guess the Number

Cette fois, vous allez partir d'un script vierge, et créer un jeu de devinette simple.

- Le joueur devra deviner un nombre choisi aléatoirement entre 1 et 10.
- Le jeu lui dira après chaque essai si sa proposition était trop haute, trop basse, ou s'il a deviné juste.
- Il aura 3 chances pour découvrir le nombre, après quoi il aura perdu, et le programme lui annoncera la bonne réponse.

Exercice - Guess the Number

Comment procéder :

- Choisir et enregistrer un nombre entre 1 et 10.
- Demander une proposition au joueur.
- Comparer la réponse du joueur et le nombre choisi, et afficher le message correspondant.
- Si le joueur n'a pas trouvé la réponse, refaire les 2 points précédents.
- Si c'était le troisième essai, dire au joueur qu'il a perdu, et afficher quel était le nombre à deviner.

Level 2-1

“While E. Coyote.”

L'ordinateur, cet esclave incompris…

Les ordinateurs sont particulièrement doués pour faire des tâches répétitives.

Nous allons voir une façon économique de demander à l'ordinateur de répéter un morceau de code un nombre de fois donné.

On appelle ça une *boucle*.

La boucle logique ou boucle while

Il y a 2 types de boucles:

- Les boucles *logiques*
- Les boucles *arithmétiques*

Nous allons commencer par le premier type.

On utilise le nom “logique”, car c’est une boucle qui va s’exécuter “*tant que*” (**while**) une certaine condition est vérifiée.

Tant que…

Comme son nom l'indique (en anglais du moins), la boucle **while** bouclera tant que… mais "tant que" quoi?

Et bien tant que la condition qui la suit est *vraie*.

```
while a < 3:
    # best code in the world
```

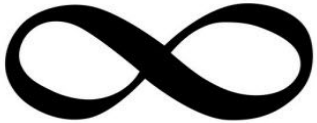
Dans cet exemple, la boucle bouclera tant que a est strictement plus petit que 3.

Vers l'infini et au delà...

Evidemment, comme pour un **if**, il y a un bloc de code indenté, et contrôlé par cette boucle.

```
hit_points = 3
while hit_points >= 0:
    # big_punch retourne un chiffre entre 0 et 2
    hit_points = hit_points - big_punch()
```

A votre avis, que se passe-t-il si **big_punch()** renvoie toujours 0 ?



La condition **hit_points >= 0** sera toujours vraie et donc la boucle continuera, encore…

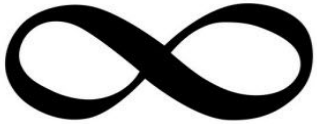
et encore…

et encore…

…

…

et encore...



et encore…

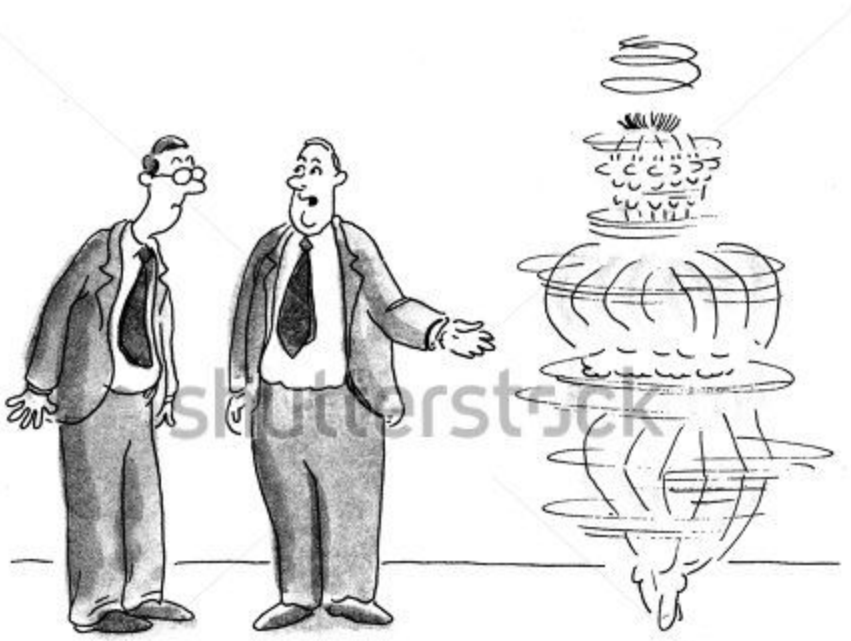
et encore…

…

…

Jusqu’à l’infini…

Le cas de la boucle infinie est bien connu des programmeurs et a déjà dû causer des crises de nerfs.



"Bob is our infinite loop specialist."

Et si jamais ça arrive?

Comme écrit sur la couverture du Guide du Routard Intergalactique : “**Don’t panic!**”

Il y a une échappatoire.

En Python, on peut interrompre l’exécution d’un code en pressant les touches **Ctrl + C**

Dans ce cas, une erreur ***KeyInterrupt*** apparaît, mais au moins le code s’arrête :)

D'ailleurs soyons fous

Dans la console Python, testez le code suivant:

```
while True:
    pass
```

Et testez Ctrl+C.

Donc, maintenant qu'on sait tout ça?

A vous de jouer!

Nous allons améliorer notre jeu de devinette !

Utilisez une boucle **while** pour éviter de répéter votre code 3 fois.

Changez ensuite le nombre d’essais autorisés, et imprimez après chaque tentative le nombre d’essais restant au joueur.

Level 2-2

“Todo: listes”

Listes

Nous avons beaucoup utilisé des variables qui contiennent une valeur, mais en Python, on peut faire bien plus !

On peut par exemple affecter toute une liste de valeurs à une seule variable.

Pour faire celà, il nous faut créer un objet de type “liste”.

Listes

Comme premier exemple, créons une liste de nombres :

my_list = [4, 4, 4, 7, 1, 9]

La variable **my_list** contient donc maintenant une liste de 6 nombres entiers.

Mais on peut mettre ce qu’on veut !

other_list = [“I”, “really”, “love”, “Python”]

Listes

Vous pouvez considérer chaque élément de la liste comme une espèce de variable.

Et on peut mixer les types:

```
three = 3
my_list = [4, 5, 6]
weird_list = [1, “two”, three, my_list]
```

Ce code générera la liste:

```
[ 1, “two”, 3, [4, 5, 6] ]
```

Mise à l'index

Outre créer des listes, reste à voir comment accéder à leur divers éléments.

On utilise pour ce faire un “index”. C’est simplement un nombre qui désigne la position d’un élément dans la liste.

```
my_list = [“I”, “really”, “love”, “Python”]
print(my_list[2])
```

...va imprimer “love” à la console.

Euh...

…”love” n’était pas à l’index 3 ?

Non ! En Python, on commence à compter à partir de zéro !

my_list[0] -> premier élément

my_list[1] -> deuxième élément

…

Euh...

On peut également utiliser des indexes négatifs !

my_list[-1] -> Dernier élément

my_list[-2] -> Avant-dernier élément…

Ca peut être utile pour nous éviter de devoir calculer la longueur de la liste.

Slicing and dicing

Il est également possible d'accéder à plusieurs éléments d'une liste en même temps.

Pour ce faire, on donne un index de départ, et un index de fin, et Python va extraire une sous-partie de la liste.

```
my_list = [“I”, “really”, “love”, “Python”]
print(my_list[1:3])
```

Ceci va renvoyer une sous-liste : **[“really”, “love”]**
On reçoit donc les éléments à partir de l'index 1 (compris) jusqu'à l'index 3 (non-compris).

Slicing and dicing

On peut aussi omettre un des deux indexes pour aller jusqu'à la fin de la liste:

```
my_list[:3]
```

```
my_list[1:]
```

Rappel : Quand on spécifie une limite, Python va chercher les éléments depuis le premier index (compris), jusqu'au dernier index (non-compris).

Petite parenthèse de "caractère"

Les indexes, ça fonctionne aussi pour les chaînes de caractères !

```
name = “TheName”

print(name[3:])
```

Autres opérations

On peut rajouter un élément à la fin d'une liste:

the_list.append(new_element)

Ou demander la longueur d'une liste (ou d'une string):

len(the_list)

Ou encore insérer un élément à une position donnée:

the_list.insert(index, element)

Autres opérations

Pour retirer un élément, deux options :

the_list.remove(element)

...qui retire la première copie de **element** de la liste

del the_list[index]

...qui retire ce qui se trouve à l'index **index**.

the_list.pop(index)

...qui renvoie et retire ce qui se trouve à l'index **index**.

Tupleware

Dans le code, nous avons vu une “liste”, mais définie avec des parenthèses.

dungeon_size = (5, 5)

Il s’agit d’un “tuple”. Vous pouvez voir ça comme une liste dont on ne peut pas modifier le contenu directement.

Mais ils fonctionnent toujours avec les indexes, etc...

Level 2-3

“Plus de bouclettes.”

Les boucles FOR

Nous avons vu les boucles **while**.

Il existe un autre type de boucle, qui s’exécute un nombre donné de fois.

Mais avant, un dernier petit détour...

Un détour par où ?

Un élément presque indispensable en Python est la fonction qui permet de générer une séquence de nombres entiers.

C’est **range()**

Dans son incarnation la plus simple :

range(10)

va générer **[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]**

Enfin presque...

On peut faire mieux ?

range() créé un type d'objet un peu spécial, mais on peut convertir ça en liste pour y voir plus clair :

```
my_sequence = range(10)
my_list = list(my_sequence)
print(my_list)
```

Ou encore, en condensant (gare aux parenthèses) :

```
print(list(range(10)))
```

D'autres infos ?

On peut donner un début et une fin :

range(2, 7)

Ou encore, donner le “pas”

range(0, 10, 2)

Question bonus : comment faire pour générer une liste de 10 à 1 ?

C'est mignon, mais ça ne nous fait pas une boucle...

Effectivement, mais retenez ceci :

range() est une façon de produire une *séquence*, tout comme une liste ou un tuple.

Voyons voir ce que nous pouvons en faire...

Level 2-4

“Le réveil de la ***for***ce”

Enfin !

Et oui !

En Python, il existe donc plusieurs types de boucles, nous allons voir la boucle dite arithmétique, ou encore “boucle **for**”.

Un exemple vaut mieux qu’un long discours :

```
for i in range(10):
    print('Compteur : ' + str(i))
```

Donc, il se passe quoi là exactement ?

```
for i in range(10):
    print('Compteur : ' + str(i))
```

La boucle for va parcourir les éléments d'une *séquence*, et va exécuter le code autant de fois qu'il y a d'éléments dans la séquence.

A chaque fois, l'élément de la séquence est mis dans une variable spéciale, ici **i**, qu'on peut utiliser à l'intérieur de la boucle. (Ca change donc à chaque passage dans la boucle !)

Ca semble bien, tout ça !

A vous de jouer !

Ecrivez une boucle **for** qui écrit les nombres impairs de 21 à 11 (compris).

LEVEL 2-5

"BOSS BATTLE"

A vous de jouer!

Nous allons réaliser un petit jeu de “pendu”.

Pour cela, il va falloir:

- Une liste de mots à trouver
- Une liste des lettres proposées par l’utilisateur
- Imprimer le mot à trouver avec uniquement les lettres devinées, et des “-” pour remplacer les autres.
- Détecter quand le joueur a trouvé toutes les lettres.

Homework

Pirate vs. Ninja

Un combat à mort…

Cette fois, à vous de faire un micro jeu qui se joue tout seul. Il simulera un combat entre un pirate et un ninja. Qui en sortira vainqueur ?

Le “jeu” consistera donc à définir les points de vie des deux personnages, et à les faire se battre tant que l’un des deux n’a pas été réduit à 0 point de vie.

Le slide suivant vous donne une idée de la structure à suivre (on peut appeler ça du “pseudo-code”), il y a évidemment pleins de façons d’organiser ça :)

Un combat à mort...

```
Mettre les points de vie du pirate à 20
Mettre les points de vie du ninja à 25
TANT QUE les points de vie du pirate et du ninja sont supérieurs à 0:
    Le pirate inflige entre 1 et 6 points de dommage au ninja
    Le ninja inflige entre 2 et 4 points de dommage au pirate
    Afficher les points de vie restants des deux personnages

SI le pirate ET le ninja ont leurs points de vie à 0 ou moins:
    Afficher un message d'égalité
SINON SI le pirate a ses points de vie à 0 ou moins:
    Afficher la victoire du ninja
SINON:
    Afficher la victoire du pirate
```