

The background of the slide is a pixelated image of a Sonic the Hedgehog game level. It features a green grassy field with a blue sky at the top and a grey stone path at the bottom. Scattered throughout the field are several small, pixelated objects: blue and red rings, a red and white checkered flag, a small blue car-like object, and a small green bush with a yellow center. The title text is centered over the grassy area.

# INITIATION À LA PROGRAMMATION EN PYTHON

**Bastien Gorissen & Thomas Stassin**

# READY ?

Press start!

# PROLOGUE

“And another bite of Python...”

# VOUS ALLEZ COMMENCER À VOUS SENTIR DE PLUS EN PLUS À L'AISE AVEC LA PROGRAMMATION...



# JEU DU JOUR

“Transistor”



A stylized illustration of a female character with red hair and blue eyes, wearing a black jacket and a yellow dress. She is holding a large, glowing blue and yellow weapon with a red circular lens. The background is a futuristic cityscape with tall buildings and a large, glowing blue and yellow structure. The word "TRANSISTOR" is written in a glowing blue font.

TRANSISTOR





# LORS DE L'ÉPISODE PRÉCÉDENT...

La fois dernière:

- Nous avons revu les boucles **for**
- Réalisé un jeu de pendu
- Parcouru un labyrinthe



# AU MENU:

- Les dictionnaires
- Les imports de modules personnels
- Les imports de modules Python
- La documentation

# LEVEL 4-1

“Le dictionnaire de A à Z”

# DICT...

Un dictionnaire est un type de variable connu en Python sous le nom de **dict**.

Ce type de variable ressemble à une liste car il sert aussi à stocker plusieurs valeurs dans une même variable.

La différence avec sa cousine la liste c'est que le dictionnaire n'utilise pas des index qui se basent sur la position dans la liste, mais des clés.

# LE DICTIONNAIRE CLÉ EN MAIN

Chaque valeur du dictionnaire est liée à une clé qui est unique et chacune est liée à une valeur

Si, par exemple, je veux stocker dans un dictionnaire les données d'une recette de potion avec en clés les ingrédients et valeurs la quantité dont j'en ai besoin, j'écrirai:

```
recipe = {'toad bubble': 1, 'rat tongue': 2, 'spider jaw': 1}
```

# LE DICTIONNAIRE CLÉ EN MAIN

Donc oui, pour écrire un dictionnaire dans le code, on écrit les données entre accolades de la manière suivante: “*clé: valeur*”, chacune des données étant séparée par une virgule.

Dans l'exemple de la potion:

On retrouve bien les **clés** et les **valeurs**

```
{'toad bubble': 1, 'rat tongue': 2, 'spider jaw': 1}
```

NB: un dictionnaire vide s'écrit comme ceci: {}

# LE DICTIONNAIRE CLÉ EN MAIN

Les clés et les valeurs peuvent être de tout type:

Par exemple on pourrait décider d'utiliser des entiers comme clés qui représentent un numéro d'identification et comme valeur le nom de la personne derrière ce numéro d'identification.

De la même façon, on peut mettre comme valeur n'importe quel type de données, par exemple un **str**, un **int**, une **list** ou même... un autre **dict**

# LIRE UN DICTIONNAIRE

Si on veut lire une donnée particulière du dictionnaire on, procède comme pour la **list** à la différence près que l'on met entre les crochets la clé et non l'index.

Pour l'exemple de la potion, si je veux afficher la quantité de langues de rat j'écrirais:

```
print(recipe['rat tongue'])
```



# ÉCRITURE D'UN DICTIONNAIRE

De la même manière, si je veux changer la quantité de langues de rat j'écrirais:

```
recipe['rat tongue'] = 5
```

Attention, si vous tentez d'écrire une donnée pour une clé qui n'existe pas, elle sera créée:

```
recipe['dragon bone'] = 2
```

Cette ligne crée la clé 'dragon bone' et y associe la valeur 2.

Le dictionnaire ressemblera ensuite à:

```
{ 'toad bubble': 1, 'rat tongue': 5, 'spider jaw': 1, 'dragon bone': 2 }
```

# CLÉ INCONNUE

Par contre, si vous tentez d'accéder à une clé qui n'existe pas sans passer par l'affectation (par exemple en tentant de l'imprimer), Python renverra une erreur.

Et maintenant à vous de jouer...

# LE JEU DU DICTIONNAIRE

Imaginons que l'on veuille coder les prémisses du nouveau Dragon Age.

Dans un script vierge :

Entrez dans un dictionnaire par saisie clavier, le nom du héros sous la clé 'name' et sa puissance sous la clé 'power'.  
Ensuite, affichez ces données à l'écran.

# LEVEL 4-2

“Boucle et dictionnaire”

# ON BOUCLE TOUT :D

A l'instar de sa cousine la **list**, on peut boucler sur un dictionnaire à l'aide d'une boucle **for**.

Quelques différences toutefois.

# ON BOUCLE TOUT :D

Si j'écris:

```
inventory = {'potion ': 1, 'scroll ': 2, 'holy sword ': 0}  
  
for key in inventory :  
  
    print (key)
```

A l'écran va apparaître:

**potion**

**holy sword**

**scroll**

# ON BOUCLE TOUT :D

Lorsqu'on boucle sur un dictionnaire, ce qui va dans la variable d'itération (dans ce cas-ci *key*), ce sont les clés du dictionnaire.

Notez que celles-ci n'arrivent pas dans l'ordre dans lequel on les a inscrites.

De fait, le dictionnaire ne mémorise pas l'ordre d'entrée des données.

Et si je veux les valeurs alors?



# ON BOUCLE TOUT :D

Si l'on veut boucler sur les valeurs, on passera par la *méthode* `values()` propre au dictionnaire.

```
inventory = {'potion ': 1, 'scroll ': 2, 'holy sword ': 0}

for value in inventory.values():

    print( value )
```

Dans ce cas là j'obtiens les valeurs à la place des clés.

Et si je veux les deux?

# ON BOUCLE TOUT :D

Si l'on veut boucler sur les clé et les valeurs, on passera par la *méthode* `items()` elle aussi propre au dictionnaire.

```
inventory = {'potion ': 1, 'scroll ': 2, 'holy sword ': 0}
```

```
for key, value in inventory.items():
```

```
    print(key + ' : ' + str(value))
```

Notez que dans ce cas là j'ai besoin de deux variables d'itération une pour la clé une pour la valeur

And now... Exercice!!

# DRAGON AGE DICTIONARY

Toujours dans l'idée de faire le nouveau Dragon Age.

Stockez les caractéristiques du héros dans un dictionnaire :

La puissance vaut 10, la magie vaut 5 et la défense vaut 3.

Ensuite demandez à l'utilisateur le nom du héros et stockez-le dans le dictionnaire.

Enfin bouclez sur ce même dictionnaire et affichez toutes les caractéristiques et leurs valeurs (donc clé et valeur).

# LEVEL 4-3

“Autant en import le vent...”

# IMPORT...

On l'a déjà vu un bon nombre de fois, la commande **import** permet d'importer un autre module.

**Module:** c'est le nom que l'on donne au fichier “.py” que l'on va importer.

```
import world
```

Importe le contenu du fichier (module) *world.py* dans le script en cours et le rend disponible pour celui-ci.

# OK, MAIS... J'EN FAIT QUOI?

A partir de ce moment, vous pouvez utiliser le contenu du module `world` de cette manière:

```
image_ground = world.GROUND_IMAGE_PATH
```

Le “.” dans ce cas sert de caractère de traversée, pour passer du module `world` à la variable `GROUND_IMAGE_PATH`.

C'est un peu comme lorsqu'on utilise la méthode **run** de **game**.

# UNE AUTRE MÉTHODE...

Il est aussi possible de ne rendre disponible qu'une partie d'un module

```
from world import GROUND_IMAGE_PATH
```

Dans ce cas seule la variable `GROUND_IMAGE_PATH` est importée, par contre, pour l'utiliser, pas besoin de préfixer de “`world.`”

```
image_ground = GROUND_IMAGE_PATH
```



# ET PLUSIEURS À LA FOIS?

Il est possible de charger plusieurs “morceaux” d’un module à la fois.

```
from world import GROUND_IMAGE_PATH, OBSTACLE_PATH
```

Encore une fois, fortes de votre expérience, c’est à vous de jouer!

# PETITE EXPÉRIENCE

Vous allez créer 2 scripts:

**main\_import.py** et **interface3.py**

Dans `interface3.py`, nous allons stocker quelques données:

```
current_class = {  
    "title": "Intro to Programming",  
    "teachers": ("Bastien", "Thomas")  
}  
formation_name = "Game Developer"
```

# PETITE EXPÉRIENCE

Dans le script `main_import.py`, nous allons importer le module `interface3.py` :

```
import interface3

print (interface3.formation_name)

from interface3 import current_class

print(current_class["title"])
print(current_class["teachers"])
```

# LEVEL 4-4

“N’Import quoi!”

# ENCORE IMPORT?

Il n'y a pas que les modules personnels de votre programme Python qui sont *importables*.

Python vient aussi avec toute une série de *modules*.

L'un d'eux est **random**.

# ET COMMENT ON L'IMPORTE?

Pour importer un module Python, c'est comme pour les modules personnels.

```
import random
```

ou bien

```
from random import choice
```

si on ne veut que **choice**

# EN PARLANT DE CHOICE, ÇA FAIT QUOI?

Essayez, dans la console Python, d'utiliser *choice*.

Cette fonction prend comme argument un objets séquençable (comme une **list**, un **dict** ou un **str**)

Et donc, malgré le nom hyper “obvious” de cette fonction, déterminez ce qu'elle fait.

# LEVEL 4-5


“Nom de Zeus!”



# ET DONC, JE DOIS EXPÉRIMENTER CHAQUE FONCTION?

Il n'est évidemment pas nécessaire d'expérimenter chaque fonction de Python pour savoir ce qu'elle fait.

Plusieurs solutions s'offre à vous.

1. “oogle est ton ami...”
2. Il existe une documentation python en ligne:  
<https://docs.python.org/3/>
3. Vous pouvez utiliser la fonction **help** pour connaître l'utilité d'une fonction

# HELP?

Si je tape `help(random.randint)` dans la console, ceci s'affiche:

Help on method randint in module random:

`randint(a, b)` method of `random.Random` instance

Return random integer in range `[a, b]`, including both end points.

# LEVEL 4-6

“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