Introduction

Ce tutoriel a pour but d'apprendre les rudiments du langage Python à un enfant ou un adolescent, en construisant petit à petit un petit jeu.

Nous utiliserons :

- le langage *Python* (Python3 pour être plus précis)

- la bibliothèque *Pygame* qui facilite la création du jeud

- un éditeur de texte (*Notepad++*, *Visual Studio Code*) ou un Environnement de Développement Intégré (IDE en anglais) comme par exemple *PyCharm*.

- le logiciel de gestion de version *git*.

Sommaire

Table des matières

Chapitre 1 – Découverte de Python

Leçon 1 : mon premier programme Python3

Python est un langage de programmation très courant. Il peut être utilisé pour, par exemple, programmer un serveur web, contrôler l'installation de programmes sur un ordinateur, apprendre à un robot à reconnaître des formes ou encore faire de la recherche scientifique.
Nous utiliserons dans ce tutoriel la version 3 de python, qui est la version courante actuelle.

Exemple de programme Python :

```
# mon premier programme
print("coucou c'est moi")
```

Ce programme affiche le texte « coucou c'est moi ».
La première ligne qui commence par le caractère **#** est un *commentaire*. Python considère que cette ligne ne fait pas partie du code du programme et ne va pas essayer de l'exécuter. A quoi ça sert alors ? Les commentaires sont écrits pour les humains : les autres développeurs ou soi-même !

On écrit les programmes Python dans des fichiers avec l'extension *.py*

Pour lancer le programme *monprogramme.py*, il faut taper dans un terminal de commandes :

```
python3  monprogramme.py
```

Exercice

- Ouvrir le répertoire lecon1
- Ouvrir le fichier *programme.py*
- Copier et coller les deux lignes de l'exemple de programme ci-dessus à l'emplacement de la ligne « A FAIRE » du fichier
- Lancer *programme.py* avec la commande : **`python3 programme.py`**
- Vous pouvez vous amuser à afficher le texte qui vous plaît.

Leçon 2 : la console Python

Au lieu d'écrire et de lancer un programme, on peut également utiliser python en mode console.

Ce qu'on appelle console ici n'a **rien à voir avec une console de jeu** ! C'est en fait une interface en ligne de commande, que l'on appelle souvent *console* en informatique : on peut taper des instructions qui sont immédiatement exécutées (on tape une instruction, Python l'exécute et affiche le résultat, on tape la suivante, etc.)

On n'a pas besoin d'écrire le code dans un fichier *.py*. Le mode console peut être très pratique pour effectuer des calculs (comme une calculatrice) ou tester des commandes.

Pour lancer la console taper dans un terminal de commande :

```
python3
```

La console s'affiche alors :

```
$ python3
Python 3.6.9 (default, Nov  7 2019, 10:44:02)
[GCC 8.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

On peut taper des commandes python :

```
>>> print("salut la console")
salut la console
>>> 1+1
2
```

Pour en sortir, il faut taper la commande :

```
exit()
```

(bien noter les parenthèses, sans quoi ça ne marche pas !)

Exercice

- Lancer la console python, effectuer quelques calculs.
- Lancer une commande *print* :

```
print("coucou")
```

- Quitter la console Python avec la commande **`exit()`**

Leçon 3 : les erreurs en Python

Les programmeurs sont loin d'être parfaits, et ils font souvent des erreurs en tapant leur code ! L'interpréteur Python (le programme qui exécute le code que le programmeur a écrit) essaie donc en général d'expliquer ce qui ne va pas.

Ces messages d'erreur en anglais sont parfois difficiles à comprendre, mais ils nous aident à savoir où est le problème.

Exercice

- Remplacer la ligne **`print("coucou c'est moi")`** par :

```
pXrint("coucou c'est moi")
```

(noter le **`X`** en trop)

- exécuter le programme
- Lire le message d'erreur. Trouvez-vous le nom du fichier dans lequel est l'erreur ? La ligne dans le fichier ? Est-ce que d'autres informations dans le message d'erreur vous paraissent utile ?
- Vous pouvez corriger l'erreur, relancer et vérifier que le programme se comporte maintenant correctement.
- Ajouter des espaces devant *print* (par exemple en appuyant sur la touche tabulation de l'ordinateur. Lancer le programme. Que se passe-t-il ? Comment corriger l'erreur ?

- Après avoir supprimé ces espaces devant *print*, mettre le mot *print* en majuscules et lancer le programme. Que se passe-t-il ? Comment corriger l'erreur ?

Leçon 4 : les variables

Une *variable* permettent de stocker temporairement une information dans la mémoire de l'ordinateur. On peut alors la modifier (au moyen d'une opération mathématique par exemple), afficher sa valeur ou la recopier dans une autre variable.

Exemple :

```
taille = 180
print(taille)
```

Dans cette exemple le nom de la variable est *taille*, et on lui fixe la valeur 180. La seconde ligne permet de l'afficher.

Ce programme affiche donc :

```
180
```

On peut aussi modifier la valeur de notre variable :

```
taille = 180
print(taille)
taille = taille + 7
print(taille)
```

La deuxième ligne lit la valeur de la variable taille et y ajoute 7. Ce court programme affiche:

```
180
187
```

Les variables peuvent être de différents types. Elles peuvent contenir par exemple : un nombre entier (comme dans l'exemple ci dessus), un nombre à virgule (par exemple 1.999, notez le « . » utilisé comme séparateur décimal), un texte (qu'on appelle *chaîne de caractères*, ou *string* en anglais).

Exemple :

```
prenom = "jean"  # variable de type string (chaîne de caractères)
taille = 100  # variable de type int (nombre entier)
print(prenom)
print(taille)
```

Exercice

- Insérer les quatre lignes de l'exemple ci-dessus dans *programme.py,* et l'exécuter.
- Essayer d'inverser l'ordre des lignes. Que-se passe-t-il si l'on place les lignes **`print(…)`** avant les lignes **`prenom=…`** et **`taille=…`** ?

- Essayer diverses opérations sur les deux variables, par exemple :

```
taille = 100
print(taille)
taille = taille * 2
print(taille)
taille = taille / 3
print(taille)
```

  - ou encore :

```
prenom = "jean"
print(prenom)
prenom = prenom + "AZERTY"
print(prenom)
```

  - ou encore :

```
prenom = "jean"
print(prenom)
taille = 100
print(taille)
prenom = prenom + taille
print(prenom)
```

  - Quelles opérations fonctionnent ? Lesquelles ne fonctionnent pas ? Comprenez-vous pourquoi ?

- Afficher le type des variables avec la fonction *type()*, par exemple :

```
print(type(taille))
```

Python peut donner comme réponse **`<class 'int'>`** (par exemple), ce qui correspond à un type *nombre entier* (**int**eger en anglais).

- De quels type sont nos variables ? Est-ce qu'elles peuvent changer de type après certaines opérations ?

Leçon 5 : les conditions

L'*exécution conditionnelle* est très utile en programmation : c'est ce qui permet au programme de se comporter différemment selon son état. En résumé, on définit un comportement du style :

si (une condition est vraie) **alors** (faire quelque chose) **sinon** (faire autre chose)

On n'est pas obligé d'utiliser la branche *sinon.*

par exemple :

si (on appuie sur la touche espace) **alors** (afficher « espace »)

En langage Python on écrit cela comme dans l'exemple suivant :

```
note = 15
if note > 10:
    print("tu as la moyenne")
else:
    print("tu n'as pas la moyenne")
```

notez :

- les mots-clés **`if`** et **`else`** et l'utilisation des « **`:`** »
- la condition **`> 10`**, qui signifie *strictement supérieur à 10*.
- les espaces avant les *print()*

Exercice

- Copier le code ci-dessus dans programme.py. Que se passe-t-il si on initialise la variable note à
  - **`18`**
  - **`5`**
  - **`"jean"`**
- Remettez l'initialisation de **`note`** à 15, et amusez-vous à change la condition, examinez le comportement. Que se passe-t-il si on écrit :
  - **`if note == 15:`**
  - **`if note >= 15:`**
  - **`if note != 15:`**
  - **`if note is 15:`**

Leçon 6 : afficher une variable et les chaînes formatées

On a vu plus haut comment afficher une variable numérique (un nombre, comme notre variable **`taille`**) et comment afficher une chaîne de caractère (comme notre variable **`prenom`** ou comme **`"tu as la moyenne"`**).

Il est possible facilement d'insérer la valeur de variables (numériques, texte, ou autres) dans une chaîne de caractère. Nous utiliserons pour cela la technique *f-print*.

Exemple :

```
temps = 16
prenom = "Gabriel"
print(f"{prenom} a couru le 100m en {temps}s")
```

Ce programme affiche :

```
Gabriel a couru le 100m en 16s
```

Notez bien :

- la lettre **`f`** minuscule avant les guillemets
- les accolades **`{`** et **`}`** autour du nom des variables dans la chaîne de caractère

Exercice

- Enlever le **`f`** minuscule. Qu'affiche alors le programme ?
- Remplacer la valeur de **`prenom`** par **`1234`**. Est-ce que cela cause une erreur ? Qu'affiche alors le programme ?
- Remplacezr**`{temps}`**par **`{temps+10}`**. Est-ce que cela cause une erreur ? Qu'affiche alors le programme ?
- Repartir de l'exemple de la *Leçon 5 : les conditions* et ajouter un prénom aux chaînes de caractères affichées dans les deux branches du if/else.

Leçon 7 : les listes

Les listes sont une façon d'assembler plusieurs valeurs comme dans un tableau.

Par exemple si nous voulons afficher le tableau suivant :

Vitesse en km/h
10
167
33.1

Créons une variable liste :

```
vitesse=[10, 167, 33.1]
```

On note que :

- Notre liste est définie par des crochets **`[`** et **`]`**
- Les valeurs sont séparées par des virgules **`,`**

Pour aller chercher l'un des éléments de cette liste, on utilise l'indice de liste, autrement dit le numéro de la case dans notre liste.

Important : la première case porte le numéro **`0`**. Dans notre exemple de liste à 3 valeurs, le dernier élément possède l'indice **`2`**.

Exemple :

```
print(vitesse[0])
print(vitesse[2])
```

Ce programme affiche :

```
10
33.1
```

Pour obtenir la longueur d'une liste (le nombre de cases), on peut utiliser la fonction **`len()`** :

```
cases = len(vitesse)
print(f"il y a {cases} cases dans cette liste")
```

Exercice

- Créer une liste de prénoms, la remplir avec au moins 3 prénoms
- Afficher le deuxième prénom de cette liste
- afficher la longueur de cette liste
- Pouvez-vous afficher toute la liste d'un coup ?

- Essayer des indices inattendus, par exemple un nombre plus grand que la longueur de la liste, ou encore un indice négatif. Que se passe-t-il ?
- Utiliser la fonction **`type()`** pour demander le type de la variable liste. Quel est le résultat ?

Leçon 8 : les boucles

Une **boucle** permet en programmation de répéter une instruction ou un bloc d'instructions. Nous allons utiliser d'abord la boucle *for* (*pour* en français). En python cette boucle effectue un bloc d'instruction pour chaque valeur successive prise dans une liste.

Exemple :

```
dossards = [1, 4, 2, 8]
for numero in dossards:
     print(f"dossard {numero}")
     print("---")
```

Bien noter :

- les mots-clé **`for`** et **`in`**
- le caractère **`:`** à la fin de la ligne **`for`**
- les espaces avant **`print(...)`**

ce programme affiche :

```
dossard 1
---
dossard 4
---
dossard 2
---
dossard 8
---
```

Explication : l'instruction **`for`** créée une variable que nous avons appelée **`numero`**. Cette variable va prendre successivement toutes les valeurs possibles dans la liste **`dossards`**, et effectuer les deux instructions *print* pour chacune de ces valeurs.

Pour faire une boucle **`for`** sans créer de liste auparavant, nous pouvons utiliser la fonction **`range()`**

Exemple :

```
for numero in range(5):
    print(numero)
```

Ce programme affiche :

```
0
1
2
3
4
```

Grâce à la fonction **`range()`**, la variable **`numero`** prend toutes successivement toutes les valeurs de 0 à 4, et exécute l'instruction **`print(numero)`** à chaque fois.

Enfin, il existe une autre sorte de boucle plus simple : la boucle while.
Exemple :

```
numero = 0
while numero < 10:
    print(numero)
    numero = numero + 3
```

Ce programme créée une variable appelée **`numero`**, l'initialise à la valeur **`0`**. Le bloc de code dans la boucle **`while`** est exécuté tant que **`numero`** est strictement inférieur à **`10`**. ce bloc de code affiche la valeur de **`numero`** et lui ajoute **3**.

Notez bien :

- le mot clé **`while`**
- le caractère « **`:`** »
- la condition **`numero < 10`**, qui doit vous rappeler la condition du **`if`**
- les espaces devant les deux lignes du bloc d'instructions de la boucle.

Le programme affiche :

```
0
3
6
9
```

Exercice

- Ouvrir le répertoire *lecon1*
- modifier le script *programme.py* pour afficher :

```
coucou c'est moi – numero 0
coucou c'est moi – numero 1
coucou c'est moi – numero 2
```

- modifier le script *programme.py* pour afficher :

```
coucou c'est gabriel
coucou c'est jean
coucou c'est paul
```

(conseil : créer une liste avec les trois prénoms, puis une boucle **for** qui parcours cette liste)

- vous pouvez consulter les corrigés dans les fichiers du *chapitre1*

Leçon 9 : les fonctions

Nous avons déjà utilisé des fonctions dans les leçons précédentes :

- la fonction **`print(texte)`** affiche la chaîne de caractère **`texte`**
- la fonction **`type(z)`** donne le type de la variable **`z`**
- la fonction **`len(l)`** donne la longueur de la liste l

On peut utiliser les fonctions Python « *toutes-faites* » (en anglais : *built-in*), qui font partie de Python, faire soi-même ses propres fonctions ou encore utiliser des *librairies* (en Python, on dit des *modules*) contenant des fonctions prêtes à être utilisées.

Une fonction peut comporter :

- des **arguments** : les informations qu'on lui donne, en les mettant dans les parenthèses
- des **valeurs de retour** : le résultat donné par la fonction

Par exemple, la fonction **`print()`** :

- prend en argument une chaîne de caractères
- ne renvoie rien

la fonction **`len()`** :

- prend en argument une liste
- renvoie un nombre entier : le nombre d'éléments dans la liste

Utiliser ses propres fonctions permet d'exécuter des actions sans avoir à réécrire leur code à chaque fois. Pour créer une fonction, il faut utiliser le mot-clé **`def`** (début de *define*, qui veut dire *définir* en anglais). Par exemple :

```
def affiche_note(prenom, note, matiere):
    print(f"{prenom} a eu {note} en {matiere}")
```

Exemple d'utilisation :

```
affiche_note("michael", 12, "maths")
affiche_note("john", 9, "anglais")
affiche_note("alice", 19.5, "sciences")
```

Ce programme affiche :

```
michael a eu 12 en maths
john a eu 9 en anglais
alice a eu 19.5 en sciences
```

Exercice

On a demandé à plusieurs personnes le nombre de jeux vidéo qu'ils ont à la maison :

Nom	**Nombre de jeux**
Anna	1
Eric	5
Julie	8
Terry	0

Ecrire un programme qui affiche le nombre de jeux vidéos qu'a chaque personne.
Si la personne a au moins 2 jeux vidéos, écrire :

```
Bob a 5 jeux vidéos
```

Si elle a 0 ou 1 jeu, écrire la phrase au singulier :

```
Bob a 1 jeu vidéo
```

(remplacer *Bob* et le nombre par le prénom et le nombre de jeu de la personne)

Consignes supplémentaires :

- créer deux listes de 4 éléments, l'une pour les prénoms et l'autre pour le nombre de jeux
- utiliser une boucle *for* avec un indice (de 0 à 3) pour parcourir les listes
- définir une fonction qui prend comme arguments un prénom et un nombre pour afficher la phrase voulue.

Un corrigé est disponible dans le répertoire *chapitre1*.

Chapitre 2 – Pygame : la boucle principale et la grille

Nous allons commencer à utiliser Pygame, mettre en place la structure de base du programme et dessiner notre monde.

Utiliser Pygame dans son programme

Il suffit d'écrire au début du fichier :

```
import pygame
```

Cette instruction dit à Python d'aller chercher le *module* Python *pygame*. Si le module n'est pas installé dans l'ordinateur, alors notre programme Python ne peut pas s'exécuter et affiche une erreur.

La boucle principale

Un jeu Pygame comprends toujours 2 parties :

a. l'initialisation (par exemple on dit au programme quels fichiers image on va utiliser, quel est la taille du monde, etc.)

b. la boucle principale: une fois que le jeu démarre, le programme va, en boucle, effectuer des actions si une touche du clavier est pressée, calculer les déplacements des personnages et des éléments du jeu et afficher les graphismes ;

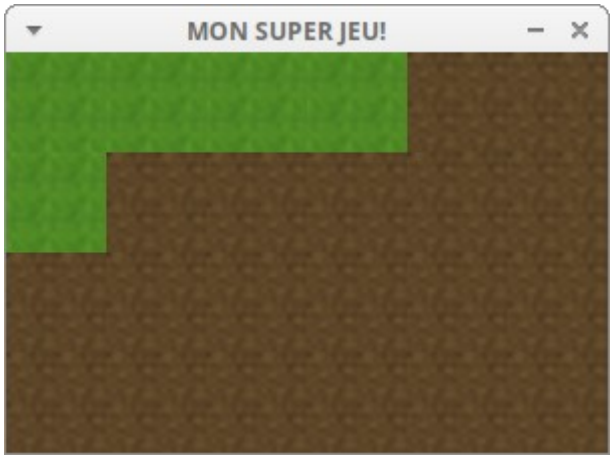
En Python, ça peut ressembler à :

```
# initialisation
pygame.init()
# on définit le titre de la fenêtre
pygame.display.set_caption("MON SUPER JEU!")
# initialiser l'affichage ici (voir le paragraphe suivant)
# ...

# boucle principale
game_over = False
while not game_over:
    # déplacer les personnages ici, etc.
    # ...

    # dire à pygame de mettre à jour l'écran
    pygame.display.update()
```

Se rappeler que les lignes qui commencent par **#** sont des commentaires (ce n'est pas du code exécuté par l'ordinateur)

La grille

La grille est la carte du jeu. Elle s'affichera dans toute la fenêtre. C'est une grille rectangulaire, où chaque case est peinte (la couleur représente le type de terrain). Pour ne pas peindre chaque case de manière uniforme, on y colle une texture.



Exemple de grille 6 colonnes x 4 lignes avec 2 textures différentes

Pour afficher cette grille, nous devons effectuer tout d'abord les étapes suivantes :

- charger chaque texture (fichier image) en utilisant la fonction **`pygame.image.load()`**
- définir le contenu de notre grille, en attribuant une texture par case. Pour cela on créé un *tableau* Python (liste de listes, voir l'exemple ci dessous)

Ces deux étapes donne un code ressemblant à :

```
# Définition des textures pour les tuiles de la grille
TX_N = pygame.image.load('./textures/dirt.png')  # sol nu
TX_H = pygame.image.load('./textures/grass.png')  # herbe

# Remplissage de la grille (6*4)
GRILLE = [
    [TX_H, TX_H, TX_H, TX_H, TX_N, TX_N],
    [TX_H, TX_N, TX_N, TX_N, TX_N, TX_N],
    [TX_N, TX_N, TX_N, TX_N, TX_N, TX_N],
    [TX_N, TX_N, TX_N, TX_N, TX_N, TX_N],
]  # Cette grille représente un tableau. Dans chacune des 4 lignes on a mis 6
textures (une par colonne)
```

Ensuite nous devons initialiser l'affichage en utilisant la fonction **`pygame.display.set_mode()`** :

```
TAILLE = 50  # Taille qu'on veut donner à chaque case de la grille, en pixels
taille_ecran = (6*TAILLE, 4*TAILLE)
surface = pygame.display.set_mode(taille_ecran)
```

Et enfin dans la boucle principale, il faut à chaque tour de boucle redessiner toutes les tuiles de la grille. Pour cela on imbrique deux boucles **`for`**, ce qui nous permet de parcourir toutes les cases (colonnes) de toutes les lignes. On dessine la texture de chaque case en utilisant la fonction **`SurfaceType.blit()`** de pygame :

```
    # Remplir la grille
    for indice_ligne in range(4):
        for indice_col in range(6):
            # calculer la position où doit s'afficher la texture
            position_texture = ( indice_col * TAILLE, indice_ligne * TAILLE )
            # dessiner la bonne texture au bon endroit
            surface.blit(GRILLE[indice_ligne][indice_col], position_texture)
```

pour calculer le position où doit s'afficher la texture, il faut considérer que :

- la coordonnée x (partant de la gauche de l'écran) sera 0 pour les textures de la première colonne (**`indice_col == 0`**), puis TAILLE, puis 2xTAILLE, et ainsi de suite.

- la coordonnée y (partant du haut de l'écran) sera 0 pour les textures de la première ligne (**`indice_ligne == 0`**), puis TAILLE, puis 2xTAILLE, et ainsi de suite.

Exercice

- Aller dans le répertoire chapitre*2*
- lancer le script *jeu.py* avec la commande : **`python3 jeu.py`**
- changer la taille de la grille en 20 x 10
- charger les textures supplémentaires « water » (eau) et « wall » (mur), et modifier la grille pour y mettre ces textures (libre à vous de dessiner la carte qui vous plaira)

Un corrigé est disponible dans le répertoire *chapitre2*

Chapitre 3