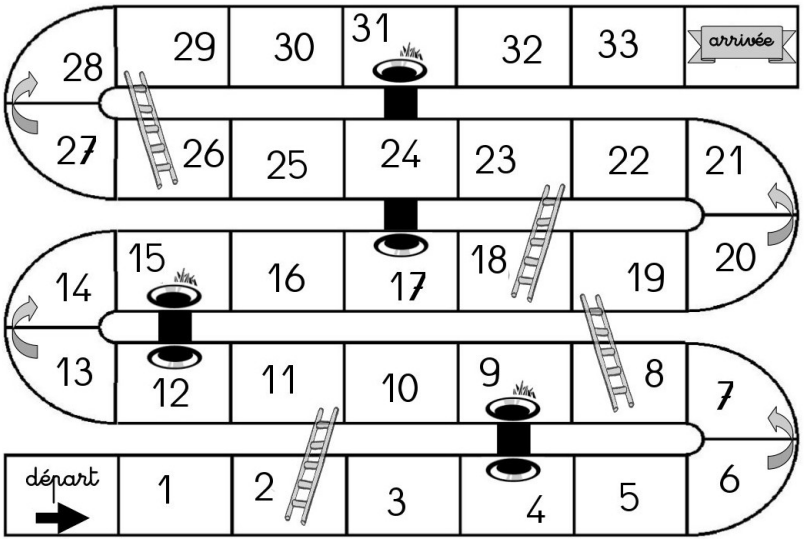
Initiation à Python : modéliser un jeu d'échelles et de serpents

L'objectif est construire sur le logiciel Python *un algorithme qui simule l'avancement d'un pion dans un jeu de société simple.*

On considère le plateau de jeu suivant :



Les joueurs jouent à tour de rôle. Lorsque c'est son tour, le joueur lance un dé à six faces et fait avancer son pion du nombre de cases indiqué par le dé puis applique les règles suivantes :

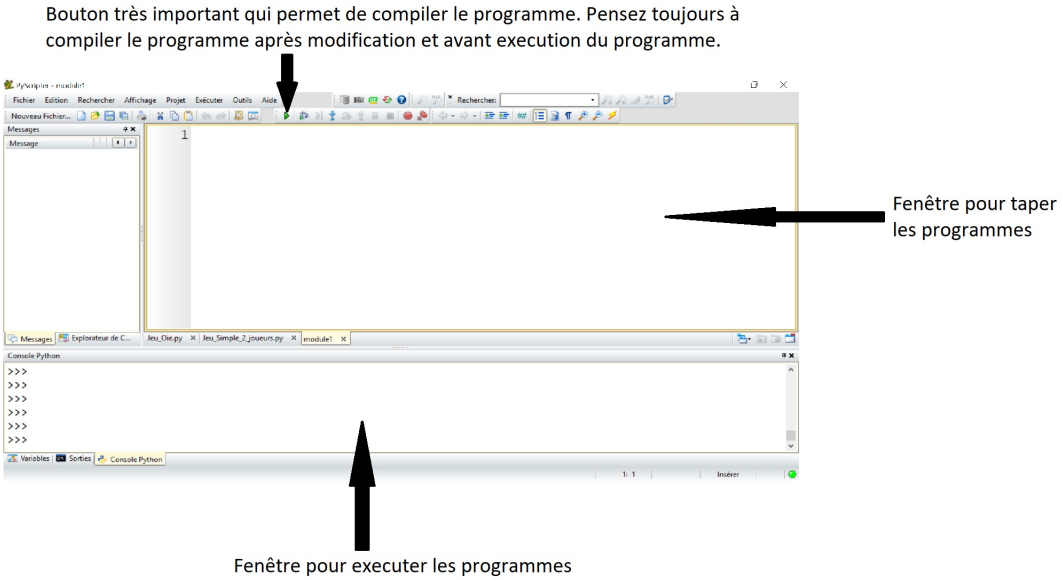
- si le pion atterrit sur une case au pied d'une échelle il suit l'échelle et va directement à la case indiquée par le haut de l'échelle.
- si le pion atterrit sur une case en haut d'un tunnel (équivalent du serpent dans le jeu classique) il suit le tunnel et recule à la case indiquée par le bas du tunnel.

Le gagnant est le premier joueur à dépasser la case 33.

Note : il est toujours possible de complexifier les règles, notamment faire en sorte que le joueur recule s'il ne tombe pas exactement sur la case arrivée.

I Apprivoiser Python

Dans votre session, ouvrir le logiciel **EduPython** (sur le bureau, dans le dossier MATHS). La fenêtre ci-dessous s'ouvre alors :



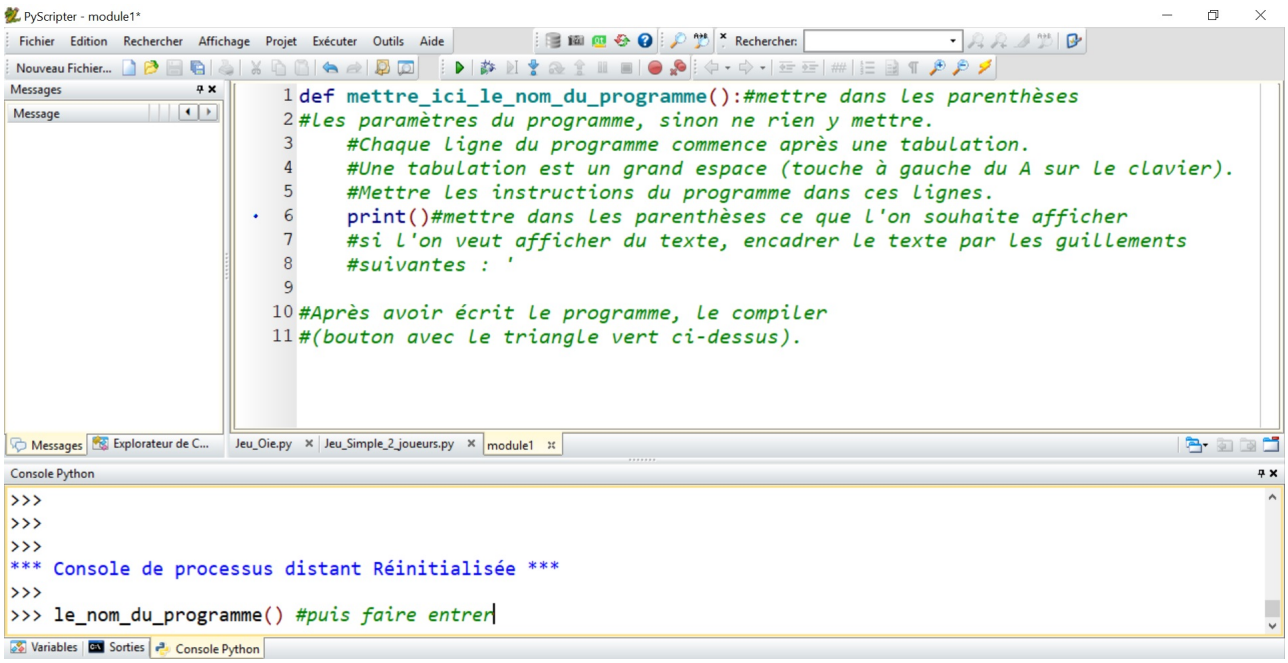
Exercice 1. L'interface des commandes d'exécutions.

La fenêtre du bas permet déjà de faire quelques opérations élémentaires.

1. *Du calcul :* taper devants les chevrons (le truc qui ressemble à $>>>$) une opération élémentaire comme $5+9$ puis faire entrer.
2. *Faire une affectation :* c'est-à-dire rentrer une valeur numérique (par exemple 5) dans une mémoire de l'ordinateur (par exemple n). Affecter 5 à la variable n.
3. **En une seule ligne**, ajouter 3 à la mémoire n et enregistrer cette valeur dans une nouvelle mémoire m.

Exercice 2. Le premier programme.

Dans la fenêtre du haut faire un premier programme affichant « Hello world ! » en vous aidant du squelette et des indications (après le signe #) suivants :



Exercice 3. Faire un nouveau programme qui demande un paramètre n, le multiplie par 5 et lui ajoute 3, puis retourne le nouveau nombre obtenu. La commande *return*, puis la variable considérée sans parenthèse autour, permet de retourner une valeur à la fin d'un programme.

II Faire lancer le dé à Python

Générer un nombre aléatoire nécessite de dire à Python de télécharger une *bibliothèque* qui lui apportera le savoir suffisant pour générer un nombre aléatoire. Dans ce but, insérer une ligne tout en haut de la fenêtre des programmes et taper dans cette ligne la commande

import random

La commande pour générer un nombre aléatoire entier entre deux entiers $n_{\min}$ et $n_{\max}$ est alors :

rand.randint($n_{\min}$,$n_{\max}$)

Exercice 4.

1. Ecrire un programme qui tire puis retourne un nombre aléatoire en 1 et 100.
2. Modifier ce programme pour qu'il modélise le lancer d'un dé à 6 faces.

III Faire avancer un joueur

On notera n la mémoire contenant la position du joueur. On suppose qu'au départ le joueur est à la case $n = 0$. Pour faire avancer le joueur de 5 case par exemple, quelque soit sa position précédente la commande est :

$$n = n + 3$$

Exercice 5. Ecrire un programme qui

1. fait partir le joueur de la case $n = 0$,
2. lance un dé à six faces,
3. fait avancer le joueur du nombre de cases indiqué par le dé,
4. retourne la case d'arrivée du joueur.

On souhaite maintenant écrire un programme qui fasse avancer le joueur un certain nombre de coups. On note p le nombre de fois où l'on veut faire avancer le joueur.

Exercice 6. Ecrire un programme qui

1. demande en paramètre la valeur de p,
2. fait partir le joueur de la case $n = 0$,
3. débute un boucle **Pour** à l'aide de la commande suivante :

for i in range(*le nombre de fois où l'on parcourt la boucle*) :

4. dans cette boucle *pour*, le programme doit lancer le dé et faire avancer le joueur,
5. à la fin le programme retourne la position finale du joueur.

Notez qu'à l'intérieur d'une boucle, toutes les lignes doivent avoir une tabulation supplémentaire : une pour dire que la ligne de commande se trouve dans le programme et une seconde pour dire que la ligne de commande se trouve dans la boucle. Il n'y a pas de commande de fin de boucle (du style end ou endfor). L'absence d'une seconde tabulation signifie à l'ordinateur que la boucle est terminée.

IV Jouons !

Dans cette partie, nous allons vouloir faire avancer notre joueur sur le plateau donné dans l'introduction. On ne tient pas compte pour le moment des échelles ou des trous. Notre joueur devra avancer jusqu'à avoir dépasser la case 33.

Exercice 7. A l'aide d'une boucle **tant que**, écrire un programme qui fait partir le joueur de la case $n = 0$ et le fait avancer tant qu'il n'a pas dépasser la case 33. Faire retourner la position finale du joueur.

Une boucle *tant que* s'écrit comme suit :

while *mettre ici l'instruction conditionnelle* :

De même que pour une boucle *pour* toutes les lignes contenues dans la boucle *tant que* doivent être écrites avec deux tabulations.

Exercice 8. Modifier le programme pour mettre un compteur :

1. Avant la boucle *tant que* initialiser le compteur à 0 :

$$cpt = 0$$

2. Dans la boucle *incrémenter le compter* c'est-à-dire augmenter sa valeur de 1 :

$$cpt = cpt + 1$$

3. Faire retourner au programme la valeur du compteur (à la place de la position finale du joueur).

V Avec des échelles et des trous

On veut désormais tenir compte des règles concernant les échelles et les trous du plateau de jeu. Par exemple le joueur qui atterrit à la case 2 sera directement propulsé à la case 11.

Exercice 9. Ecrire un algorithme qui

1. fait partir le joueur de la case $n = 0$,
2. met le compteur du nombre de tours à 0,
3. débute une boucle *tant que* avec l'instruction conditionnelle tant que le joueur n'a pas dépasser la case 33,
4. augmente le compteur d'une valeur,
5. lance le dé,
6. fait avancer le joueur du nombre de case indiqué par le dé,
7. bouge à nouveau le joueur si celui-ci se trouve en bas d'une échelle ou en haut d'un trou à l'aide d'instructions conditionnelles. Par exemple :

```
if n==2:
    n=11
```

8. retourne le compteur.

VI Le mode multijoueur

VI.1 Réécriture du programme précédent

Afin de pouvoir faire jouer plusieurs joueurs en même temps, nous allons dans un premier temps créer un sous-programme qui, à partir de n'importe quelle case de départ, fera avancer une fois un joueur.

Exercice 10. Ecrire un programme nommé *un_tour* qui

1. prend la case de départ n en paramètre,
2. lance le dé,
3. fait avancer le joueur du nombre de case affichée par le dé,
4. fait une suite d'instructions conditionnelles pour faire avancer le joueur s'il est en bas d'une échelle ou le reculer s'il est en haut d'un trou,
5. retourne la nouvelle position n du joueur

Exercice 11. Ecrire un programme qui fasse jouer un seul joueur (jusqu'à ce qu'il ait dépassé la case 33) en utilisant le programme de l'exercice 10 grâce à la commande suivante :

```
n=un_tour(n)
```

Faire retourner le nombre de tours.

VI.2 Niveau 2

Exercice 12. Ecrire un programme qui fait avancer deux joueurs. On notera $n1$, respectivement $n2$, la position du joueur 1, respectivement du joueur 2. On fera avancer les joueurs tant que le maximum entre $n1$ et $n2$ est plus petit que 33. Faire retourner le nombre de tours.

Exercice 13. Améliorer le programme précédent pour que le programme affiche « Le joueur 1 a gagné ! » lorsque c'est la cas et « Le joueur 2 a gagné ! » sinon. Pour afficher du texte, on pourra se référer à l'exercice 2.

VI.3 A la folie !

On veut écrire un programme qui prend en paramètre le nombre de joueurs. Pour ce faire, nous allons devoir apprendre à créer et manipuler une liste.

Exercice 14. Ecrire un programme qui

1. prend en paramètre le nombre de joueur (noté p),
2. crée une liste de 0, de taille celle des joueurs à l'aide de la commande suivante :

```
n=[0]*p
```

3. retourne cette liste.

Chaque terme de cette liste représentera la position de chaque joueur :
le premier terme, $n[0]$ sera la position du joueur 1,
le deuxième terme, $n[1]$ sera la position du joueur 2,
⋮
le dernier terme, $n[p-1]$ sera la position du joueur p.
Vous aurez noté que dans la notation Python $n[i]$ correspond au terme de la liste à la position $i+1$!

Exercice 15. Améliorer le programme précédent pour faire avancer tous les joueurs trois fois, à l'aide d'une boucle *pour* et du programme *un_tour*. Le programme retournera la liste des positions finales.

Nous allons maintenant créer le programme qui donnera le vainqueur. Pour ce faire nous allons avoir besoin de la position maximum des joueurs à chaque tour.

Exercice 16. Ecrire un programme

1. prend le nombre de joueurs p en paramètre,
2. met le compteur *cpt* à 0,
3. crée une liste des positions de tous les joueurs à 0 (voir l'exercice 14),
4. crée le maximum des positions de ces joueurs (ce sera ici 0 bien entendu) :

```
M=max(n)
```

5. débute une boucle *tant que* la valeur de M est plus petite ou égale à 33,
6. dans cette boucle,
   (a) incrémente le compteur *cpt* (voir l'exercice 8),
   (b) fait jouer tous les joueurs les uns après les autres (grâce à une boucle *pour*),
   (c) rentre la nouvelle valeur de M,
7. retourne le compteur.

Exercice 17. Améliorer le programme précédent pour qu'il affiche quel est le joueur qui a gagné.