Chapitre 1

Bases algorithmiques et Python

1.1 Bases algorithmes

Un **algorithme** est une suite finie d'instructions simples et non-ambigües dont l'exécution se fait en temps fini permettant de résoudre une classe de problèmes.

Il est constitué généralement de 4 briques de base :

- Des instructions. La plupart du temps, elles manipulent des variables. Une variable permet de stocker une valeur intermédiaire plus ou moins complexe. Attention, ce ne sont pas les mêmes que des variables en mathématiques : Une variable informatique a toujours une (et une seule) valeur bien définie, même si celle-ci peut changer au cours du temps.
- Des branchements. Ceci permet de sélectionner des instructions à exécuter ou au contraire d'en enlever. La décision est prise pendant l'exécution de l'algorithme. Dans cette catégorie, on trouve en particulier le `Si ... Alors ... Sinon ...`.
- Des boucles. Ceci permet de répéter des instructions un certains nombres de fois. Ce nombre peut être prédéfinie ou non, selon ce qu'on souhaite faire.
  On trouve ici `Tant que ... Faire ...` et `Pour ... Faire ...`.
- Une interface avec l'extérieur (un humain ou un autre programme). On les appelle aussi des entrées-sorties. Elles permettent, entre autres, de préciser le problème à résoudre (les entrées) et de communiquer le résultat (la sortie).

Ces 4 briques suffisent à écrire n'importe quel programme. Même le dernier jeu en 3D ultra-réaliste n'est constitué que de ces 4 briques. Il y en a simplement beaucoup et dans un ordre très précis...

Nous reviendrons en détail sur ces différentes briques avec Python.

Exercice 1.

On rappelle que le centre du cercle circonscrit est le point d'intersection des médiatrices des cotés du triangle.

Donner un algorithme permettant, à partir de 3 points distincts A, B, C, de construire le cercle circonscrit au triangle ABC. Les instructions admises sont `parallele a ... passant par ...`, `perpendiculaire a ... passant par ...`, `cercle de centre ... passant par ...`, `milieu de ...`, `droite passant par ... et ...`, `intersection de ... et ...`.

Exercice 2.

Donner un algorithme qui permet de déterminer, pour a et b réels quelconques, le nombre de solutions de l'équation $ax + b = 0$.
***Attention**, on ne cherche pas à déterminer les solutions, juste à savoir combien il y en a.*
***Attention**, a peut être nul !*

1.2 Python

1.2.1 Le langage

Python est un langage plutôt vieux (les premières versions datent de 1990). Il est très utilisé tant dans l'industrie informatique que par la communauté universitaire et les ingénieurs. La dernière version à ce jour est la 3.11.

C'est un **langage interprété**, c'est à dire que les instructions sont lues une à une pour être exécutées. Ceci est différent d'un **langage compilé** dans lequel les instructions sont d'abord lues en bloc et transformées en un langage directement compréhensible par le processeur (fichier exécutable).

Python est un langage à **typage dynamique**. Cela signifie que le type des variables (entier, chaîne de caractères, tableau, objet,...) n'est pas figée lors de l'écriture du programme mais déterminé lors de l'affectation. Ceci apporte un confort certain au niveau de la gestion de la mémoire : c'est Python qui gère ! Par contre, ceci peut être à l'origine de bugs difficiles à détecter.

1.2.2 Les outils

Nous utiliserons n'importe quelle version supérieure à 3.6 de Python. Les versions antérieures sont obsolètes.

Pour GNU/Linux : Dans la très grande majorité des cas, Python est déjà présent. Il faut simplement vérifier la version : dans un terminal : `python3 --version`
`https://docs.python-guide.org/starting/install3/linux/`
On utilise ensuite n'importe quel éditeur de texte (ou spyder).

Pour Mac : l'installation de python se fait via ce lien :
`https://www.python.org/downloads/macOS`.
Pour l'édition, spyder se trouve ici :
`https://github.com/spyder-ide/spyder`.

Pour windows : winpython se trouve ici :
`https://sourceforge.net/projects/winpython/files/latest/download`

Sous windows, nous utiliserons l'IDE spyder. IDE signifie Integrated Developement Environment. C'est un éditeur de texte qui permet aussi de lancer le programme en cours de rédaction via la touche **F5** ou la flèche verte.

1.2.3 Spécificités du langage python

L'une des grandes spécificités de Python est la délimitation des blocs de code à l'aide de l'indentation. Une augmentation d'indentation signifie la création d'un nouveau bloc (il devrait y avoir un `:` à la fin de la ligne au dessus) et une diminution d'indentation signifie la fin du bloc. La convention (PEP8) veut qu'on utilise 4 espaces pour indenter.

Sinon, pour les branchements et les boucles, vous les avez très certainement déjà vus :

```
if condition1:
    instruction1
    instruction2
    ...
elif condition2:
    instruction3
    ...
else:
    instruction4
    ...
instructions_apres_if
```

```
while condition:
    instruction1
    instruction2
    ...
instructions_apres_while
```

```
for i in range(n):
    instruction1
    ...
instructions_apres_for
```

Remarquez bien les indentations.
Le `elif` et le `else` sont optionnels.
Pour les conditions, on peut utiliser les comparaisons habituelles : `<`, `>`, `<=`, `>=`, `==`, `!=`. On peut aussi utiliser les connecteurs logiques `and`, `or`, `not` pour combiner plusieurs conditions.

Pour la boucle `for`, la variable `i` prend toutes les valeurs entières entre 0 (compris) et `n` (exclus). On parcourt bien `n` fois la boucle mais en commençant de 0 et en s'arrêtant à `n-1`.

Pour l'affectation, pas besoin de déclarer l'existence d'une variable. Si vous lui affectez une valeur, Python la crée tout seul et la détruit quand il est sûr que vous n'en avez plus besoin. L'opérateur d'affectation est un `=`. Par exemple `nombre = 42`.
Attention, l'opérateur n'est pas symétrique comme en mathématique. `5 = nombre` n'a aucun sens et produira une erreur !
Attention aussi à ne pas confondre `=` et `==`.

Les opérateurs mathématiques les plus courants : `+`, `-`, `*`, `/`.
L'opérateur `%` calcule le reste de la division euclidienne : `16 % 3` vaut 1.
L'opérateur `//` calcule le quotient de la division euclidienne : `16 // 3` vaut 5.
L'opérateur `**` permet de calculer une puissance entière : `3 ** 4` vaut 81.

Une chaîne de caractères est entourée de guillemets simples ou doubles : `'chaine'` ou `"chaine"`. Nous verrons plus en détail cette structure dans un prochain TD.

Enfin, le symbole `#` est utilisé pour écrire un commentaire : tout ce qui se trouve sur la ligne après est ignoré par Python. Ceci permet de faire une remarque à un éventuel relecteur humain permettant de mieux comprendre ce qui est écrit. Nous y reviendrons également...

1.3 Quelques exercices

Pour cette série d'exercice, on ajoute des fonctions du bloc d'entrées-sorties :
Pour afficher un résultat (une sortie), on utilise `print(valeur)` et pour demander un paramètre (une entrée), on utilise `valeur = int(input())`. Cette dernière paraît complexe. Elle sera expliquée ultérieurement. Ce qu'il faut retenir pour le moment c'est que `valeur` est une variable contenant un entier déterminé par l'utilisateur. Dans les deux cas, `valeur` est une variable dont le contenu sera affiché (pour le premier) ou qui sera affecté (pour le deuxième). Si on veut afficher plusieurs choses sur une même ligne, on pourra utiliser des virgules dans le `print`. Par exemple si la variable `age` contient mon age, on peut écrire `print("j'ai", age, "ans.")`.

Il ne faut pas trop s'appuyer sur ce couple `input`/`print` pour les entrées/sorties. Nous verrons par la suite que c'est loin d'être le plus pertinent... Mais ceci sera suffisant pour nous ici.

Exercice 3.
Dans chaque cas, écrire ce qu'affiche le programme.

```
n = 2**3
if n > 8:
    print("plus grand que 8")
elif n < 8:
    print("plus petit que 8")
else:
    print("egal 8")
print(n)
```

```
n = 0
while n < 5:
    print(n)
    n = n + 1
print("n=", n)
```

```
maxi = (3 * 11) // 6
for n in range(maxi):
    print(n * 5)
    print(n % 2)
print("maxi=", maxi)
```

Exercice 4.
Transcrire en Python l'exercice 2.

Exercice 5.
Écrire un programme qui demande deux nombres entiers et qui affiche le maximum des deux.

Exercice 6.
Écrire un programme qui demande un nombre entier naturel strictement positif `n` et affiche la somme des nombres entre 0 et `n` et le produit des nombres de 1 à `n`.

Exercice 7.

1. Écrire deux programmes Python qui demandent un entier `n` et affiche les `n` premières puissances de 2. Le premier en utilisant l'opérateur `**`, l'autre sans cet opérateur.
2. Écrire un programme Python qui demande un entier `barriere` et affiche toutes les puissances de 2 plus petites que `barriere`.
3. Modifier le dernier programme pour indiquer aussi le nombres de puissances affichées.