Chapitre :
Le langage python (3)

I. Introduction

Le langage python est un langage de programmation apparu vers 1990. Développé au départ par Guido van Rossum, un développeur néerlandais, il porte son nom en référence à la troupe d'humoristes britanniques des Monty Python.
Ce langage

- permet la **programmation impérative** : c'est celle qui décrit les opérations en termes de séquences d'instructions exécutées par la machine. Chaque étape peut donc modifier l'état des variables.
  C'est le type de programmation appris en lycée jusqu'à présent.
- permet également d'autres types de programmation, comme la **programmation fonctionnelle**. Dans ce cas le calcul est considéré comme évaluation de fonctions (mathématiques), et non comme changements successifs d'états.
- est **orienté objet** : il permet de définir des « objets » qui contiennent des attributs (paramètres, données) et des méthodes (des fonctions permettant de modifier ou de communiquer des attributs).
- est **typé** : chaque variable est associée à un type (nombre, chaîne de caractères, etc...). Cela permet de détecter des erreurs et s'assurer d'une certaine cohérence ; on n'ajoute pas un nombre à une chaîne de caractères.
- est **interprété** : l'exécution d'un code dans ce langage nécessite l'installation de l'interpréteur (c.à.d. du programme) python sur la machine. Bien que cela soit possible, il n'est pas fait pour produire des programmes exécutables sur des machines sur lesquelles python n'est pas installé.
  Un « programme » python reste donc un simple fichier texte contenant l'algorithme, lisible par un humain et donc modifiable pour qui connaît le langage.
  Cela s'oppose aux **langages compilés**, qui possèdent des programmes de compilation produisant, à partir du code, un programme exécutable ne nécessitant pas l'installation du langage utilisé sur la machine qui l'exécute.

Une des marques d'identité de ce langage est l'utilisation obligatoire de l'**indentation** (ajout d'espace en début de ligne).
Les avantages de ce langage sont qu'il possède beaucoup de bibliothèques (des algorithmes communs ou utiles déjà créés dans ce langage permettant de gagner du temps, car évitant d'avoir à les écrire de nouveau) et qu'il est assez simple à mettre en œuvre, contrairement à d'autres langages permettant un meilleur contrôle mais nécessitant plus de temps d'apprentissage pour les maîtriser suffisamment.
Le but de ce document sera tout d'abord d'expliquer la syntaxe utilisée pour les instructions de base déjà connues depuis la seconde.
Plus tard nous introduirons de nouveaux types de variables (booléens, chaînes de caractères, listes) et quelques fonctions utilisables sur ceux-ci.
Le cours sera ponctué d'exemples à tester ainsi que d'exercices à faire, tout ceci permettant de manipuler les notions et syntaxes introduites.
Il ne s'agit pas ici d'être exhaustif, mais bien d'introduire le langage. L'utilisation des objets n'est pas prévue et ne sera pas traitée ici.
Il est d'autre part possible avec python de créer des interfaces graphiques, ce que nous verrons plus tard.
Plus de renseignements pourront être trouvés sur la page officielle de documentation (en anglais) :

http://docs.python.org/3.2/index.html

ainsi que sur bien d'autres pages qui pourront être trouvées grâce à des requêtes précises sur des moteurs de recherche sur Internet.

II. Traduction des instructions déjà connues

0. Avant propos : utilisation de python

Pour exécuter des fichiers python ou de courtes instructions, nous utiliserons le programme IDLE. Celui-ci est disponible directement sous windows à l'installation de python. Il doit en principe être installé en supplément à python sous linux. IDLE permet d'interpréter les fichiers et les commandes python.

La fenêtre principale est le mode interactif : il permet de tester de courtes instructions.
C'est là que devront être tapés la plupart des exemples donnés dans ce cours.

Pour des algorithmes importants il est préférable de les écrire dans des fichiers. L'extension usuelle d'un fichier python est « .py », qu'il ne faut pas oublier d'ajouter lorsque l'on enregistre le fichier pour la première fois. Il est possible de créer et d'ouvrir ces fichiers dans IDLE. ils seront visibles dans de nouvelles fenêtres et pourront être exécutés (avec la touche F5). L'exécution standard sera visible dans la fenêtre principale.

Selon l'installation, il est possible d'exécuter directement un code python en double-cliquant sur le fichier.

1. Variables

Le nom d'une variable peut être composé de lettres et de chiffres, ainsi que du caractère « _ ».
Il ne doit cependant pas commencer par un chiffre.

 La casse des caractères (majuscule, minuscule) est importante.

On affecte une valeur à une variable avec l'instruction « = ». À gauche du signe égal se trouve le nom de la variable.

Exemple

```
1 X=5
```

Exercice 1

Exécuter le code suivant dans le mode interactif. Que fait-il ?

```
1 a=5
2 b=10
3 a,b
4 c=a
5 a=b
6 b=c
7 a,b
```

Remarque On peut faire des affectations simultanées, ce qui peut être astucieux.
Exécuter dans le mode interactif :

```
1 a=5
2 b=10
3 a,b=b,a
4 a,b
```

2. Affichage (sortie)

Pour afficher la valeur d'une variable, ou du texte, on peut se contenter dans le mode interactif d'écrire le nom de la variable.. Cependant cela ne fonctionne pas si l'exécution est faite à partir d'un fichier. On utilise alors la fonction **print**.

Exemple
Tester les lignes suivantes dans le mode interactif, puis les copier dans un fichier (ne pas oublier de l'enregistrer avec une extension .py) et observer les différences d'affichage à l'exécution.

```
1 a=5
2 print(a)
3 a
4 print(a+3)
5 print("Ceci est du texte")
6 "Ceci est du texte"
```

3. Demande (entrée)

La commande **input** permet de demander une réponse, qui sera considérée comme **chaîne de caractère**, puisque entrée au clavier (en terminant par la touche entrée). On peut alors affecter la réponse à une variable :

Exemple
Taper dans un fichier texte et exécuter :

```
1 text=input("Écrire du texte : ")
2 print("Vous avez écrit « ",text," ».")
```

Remarquer au passage que la fonction **print** permet d'afficher plusieurs choses les unes après les autres.
Si l'on est certain d'obtenir un nombre en réponse, on peut transformer la chaîne de caractère en nombre grâce à la fonction **int** :

Exemple
Taper dans un fichier texte et exécuter :

```
1 n=int(input("Donner un nombre :\n"))
2 print("Je le multiplie par 2 :",2*n)
```

Remarquer au passage que le code particulier « \n » dans une chaîne de caractère permet d'aller à la ligne.
D'autre part, on peut demander à afficher le résultat d'un calcul (`2*n`), et pas seulement une variable.

4. Conditionnelle

C'est ici que l'on introduit le point déjà évoqué plus haut : l'indentation.
L'instruction « Si ... Alors ... Sinon ... FinSi » impose en python une syntaxe particulière faisant intervenir de manière cruciale l'indentation. En effet, il n'y a pas de FinSi, et ce qui indique la fin est simplement le retour à l'indentation précédente.
Voici la syntaxe :

```
1 if condition:
2   Instructions si condition vérifiée
3 else:
4   Instructions si condition non vérifiée
5 Suite des instructions
```

L'indentation peut être faite au choix avec des espaces, ou avec la touche tabulation, mais doit toujours être la même pour deux instructions de même niveau.
L'utilisation du symbole « : » à la fin des lignes « if ... » et « else » est également nécessaire à la syntaxe.
Le else (sinon) n'est pas obligatoire après un if. Dans certains cas, pour éviter d'emboîter les conditionnelles, on peut utiliser des « elif *condition* : » (sinon si) ; il faut alors terminer par un else.

Exemple
Taper dans un fichier texte et exécuter :

```
x=int(input("Donner un nombre :\n"))
if x>10:
  print("nombre supérieur à 10")
elif x>4:
  print("nombre entre 5 et 10")
else:
  print("nombre inférieur à 5")
```

Exercice 2
Écrire un algorithme qui demande deux nombres et les compare (trois cas possibles)

5. Boucle Pour

Si la variable n est un entier, on peut effectuer une tâche n fois dans une boucle **Pour** avec les instructions suivantes :

```
for i in range(n):
  Instructions à effectuer
Suite des instructions
```

Là aussi, ne pas oublier les « : » et l'indentation, la boucle se terminant au retour à l'indentation précédente.
Il est possible de mettre autre chose à la place de « range(n) », en particulier des listes. Cela permet beaucoup de souplesse, mais nous verrons cela plus tard.

 Il est tout à fait possible d'utiliser la valeur du compteur i, mais il commence à 0 et termine à n-1.

Exemple
Exécuter dans le mode interactif :

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

Exercice 3
Faire un algorithme qui affiche les multiples d'un entier donné N, de 1×N à 10×N.

Remarque En fait, on peut indiquer à partir de quelle valeur on commence en rajoutant un argument à range, mais il faut alors faire attention à l'arrêt. Exécuter dans le mode interactif :

```
for i in range(10,15):
  print(i)
```

6. Boucle Tant que

Le principe est le même que pour les deux précédents. La syntaxe est la suivante :

```
while condition:
  Instructions si condition est vérifiée
Suite des instructions
```

Exemple
Taper dans un fichier texte et exécuter :

```
k=3
e=7
while k<e:
  k=k+1
print(k-e)
```

Exercice 4
Faire un algorithme qui donne le quotient et le reste de la division d'un entier n par un entier d, les deux entiers n et d étant donnés par l'utilisateur.
Rappel : le quotient est le plus grand entier q tel que $q \times d \leqslant n$. Le reste est alors $n - q \times d$.

III. Supplément : Nombres et opérations

Il y a deux types de nombres : int et float.
Les premiers sont les entiers (integers), les seconds les nombres décimaux (floating point).
La virgule des nombres décimaux est symbolisée par un point.
Les opérations usuelles peuvent être appliquées avec ces deux types de nombres, plus quelques autres (mais il peut y avoir des résultats surprenants !).
Il ne faut pas trop compter sur la fiabilité des calculs même les plus simples avec les nombres décimaux dans python.
Exécuter dans le mode interactif :

+,-,*,/ : addition, soustraction, multiplication, division

```
1 x=3.2
2 y=7
3 print(x+y)
4 print(x*y)
```

// : division entière (donne le quotient)

```
1 print(y//3)
```

% : reste d'une division entière (modulo)

```
1 print(y%3)
```

** : puissance (entière ou non)

```
1 print(y ** 3)
```