Python pour les data scientistes

Chapitre 2 : Eléments du langage Python

1. Les commentaires sur une seule ligne s'introduisent en insérant le **symbol #** avant le texte.
2. Les commentaires sur plusieurs lignes s'introduisent au sein des triples quotes : """ ...."""

Exemple. Commentaires en python

```
1 # Voici un commentaire sur une seule ligne
2 """ Voici un commentaire
3 sur plusieurs
4 lignes . . . .
5 """
```

Les variables

Une variable est une zone de mémoire dans laquelle une valeur est stockée. En programmation, cette variable possède un identificateur. En Python, la déclaration d'une variable et son initialisation se font en même temps.

```
>>> x=0
>>> x
0
```

Dans cet exemple, nous avons déclaré, puis initialisé la variable x avec la valeur 0. On peut interpréter cette instruction comme suit :

- Python a « deviné » que la variable était un entier. On dit que Python est un langage au typage dynamique.
- Python a alloué (réservé) l'espace en mémoire pour y accueillir un entier. Chaque type de variable prend plus ou moins d'espace en mémoire. Python a aussi fait en sorte qu'on puisse retrouver la variable sous le nom x.
- Enfin, Python a assigné la valeur 0 à la variable x.

L'instruction print Python (on verra qu'il s'agit d'une fonction) est souvent utilisée pour générer la sortie des variables.

Exemple. affichage variable

```
1 x = 5
2 print (x) # affiche 5
```

Types de données

En Python, il n'est pas nécessaire de déclarer les variables avant de pouvoir leur affecter une valeur, le type de données est affecté dynamiquement. La valeur que l'on affecte à une variable possède un type qui dépend de la nature des données (entier, réel, chaîne de caractères, etc.). Le type du contenu d'une variable peut donc changer si on change sa valeur. Pour connaitre le type d'une valeur, Python dispose de la fonction **type()** qui revoie le type d'une valeur introduite, par exemple :

type(2) renvoie int
chaine="bonjour"
type(chaine) revoie str

Dans ce qui suit, nous présenterons les types les plus connus dans Python.

- int : Permet de stocker une valeur entière.
- float : Ce type est utilisé pour stocker des nombres à virgule flottante.
- str : ce type représente une suite quelconque de caractères délimitée soit par des simple quottes, soit par des double quottes.
- bool : ce type est utilisé pour les valeurs True ou False (valeurs booléennes).
- list : ce type est utilisé sous Python une collection d'éléments séparés par des virgules et délimitée par des crochets. Les éléments de la liste ne sont pas nécessairement du même type de donnée. Voici quelques exemples:

```
maliste = ["bonjour", "monde", 16, "dimanche", 2022]
type(maliste)
```

Ce code renvoie list

```
>>> maliste = ["bonjour", "monde", 16, "dimanche", 2022]
>>> type(maliste)
<class 'list'>
```

Vous pouvez appeler un élément d'une liste avec sa position qu'on appelle indice qui commence par la valeur 0.

Si on revient à la liste :

```
maliste = ["bonjour", "monde", 16, "dimanche", 2022]
```
maliste[0] renvoie la valeur "bonjour"
maliste[2] renvoie la valeur 16

Comme les chaînes de caractères, les listes supportent l'opérateur + qui permet la concaténation de deux listes, ainsi que l'opérateur * pour la duplication.

Examinons l'exemple suivant :
l1=["bonjour"," les amis"]
l2 = [22,06,2015]

Après l'opération l1+l2 on obtient la liste ["bonjour"," les amis", 22,06,2015]
L'opération l1*2 renvoie ["bonjour"," les amis","bonjour"," les amis"] (la liste est dupliquée 2 fois).

Vous pouvez aussi utiliser la méthode ".append()" lorsque vous souhaitez ajouter un seul élément à la fin d'une liste.

Par exemple : 11.append(1) renvoie la liste ["bonjour"," les amis",1], la valeur 1 a été ajoutée à la fin de la liste.

Une fonctionnalité importante dans Python est la manipulation des nombres complexes. La partie imaginaire est indiquée par la lettre j (ou J).

Voilà un exemple, qui renvoie le type d'un nombre complexe :

```
x = 1 - 2j
type(x)
```

Ce code renvoie complex

```
>>> x = 1 - 2j
>>> type(x)
<class 'complex'>
>>>
```

Le tableau suivant représente une liste des opérateurs arithmétiques usuelles :

Opérateur	Signification
+	Addition
-	Soustraction
*	Multiplication
/	Division
**	Puissance
%	Reste de la division Euclidienne
//	Quotient de la division Euclidienne

La fonction **input()** prend en argument un message (sous la forme d'une chaîne de caractères), demande à l'utilisateur d'entrer une valeur et renvoie celle-ci sous forme d'une chaîne de caractères. Il faut ensuite convertir cette dernière en entier (avec la fonction int()).

Examinons cet exemple:
Ecrire un programme en Python qui demande à l'utilisateur d'introduire son nom et de lui afficher le message : Bonjour <nom>

```
nom = input("Tapez votre nom : ")
print("Bonjour  : " , nom)
```

La fonction **print()** affiche l'argument qu'on lui passe entre parenthèses et un retour à ligne. Ce retour à ligne supplémentaire est ajouté par défaut. Si toutefois, on ne veut pas afficher ce retour à la ligne, on peut utiliser l'argument par « mot-clé » end :

```
print (" Hello world ")
print (" Hello world !", end ="")
```

Bloc d'instruction

Un bloc regroupe une instruction ou une série d'instructions d'un même enchainement comme les tests, les boucles...etc. Chaque bloc est défini par une indentation en décalant le début des instructions vers la droite grâce à des espaces (4 espaces habituellement mais même nombre pour le même bloc d'instructions) en début de ligne.

Toutes les instructions d'un même bloc doivent être indentées exactement au même niveau (c'est-à-dire décalées à droite d'un même nombre d'espaces).

Un bloc peut contenir une ou plusieurs instructions, et notamment des instructions composées (tests, boucles, etc.).

Exemple2

Ecrire un programme en Python qui demande à l'utilisateur de saisir deux nombres entiers x et y et de lui afficher leur somme.

Correction

```
x = input("Tapez la valeur du premier nombre x : ")
y = input("Tapez la valeur du deuxième nombre y : ")
x = int(x)
y = int(y)
somme = x+y
print("La somme " , somme)
```

Les tests

Pour tout langage de programmation, Python dispose des structures alternatives pour faire des tests comme l'instruction "if".

L'instruction if

L'instruction "if" est la structure alternative la plus simple. La syntaxe de "if" fait intervenir la notion de bloc :

```
if condition:
    Instruction
```

Où `condition` est une expression booléenne (True ou False)

`Instruction` n'est exécutée que **si** la condition est vérifiée.
Le : (deux-points) à la fin de la ligne contenant le "if" introduit le bloc d'instructions à exécuter si la condition est vérifiée.

Voici un exemple illustratif :

```
x = input("Saisir un nombre")
x = int(x)
if x > 0:
    print("Vous avez saisi un nombre positif")
```

Ce code demande à l'utilisateur de saisir un nombre entier, il teste sa valeur, si elle est supérieure à 0, un message s'affiche lui indiquant qu'il s'agit bien d'un nombre positif. Noter que le message s'affiche uniquement si la condition est vérifiée (vraie).

L'instruction if … else

L'instruction **if … else** peut être utilisée si l'enchaînement d'exécution suit deux cycles différents après la vérification de certaines conditions. La syntaxe de cette instruction est la suivante :

```
if condition:
    Instruction 1
else:
    Instruction 2
```

Si la condition est vérifiée, on exécute Instruction 1, sinon, on exécute `Instruction 2`. On rappelle que `Instruction 1` ou `Instruction 2` regroupe ou plusieurs instructions.

Voici un exemple illustratif :

```
x = input("Saisir un nombre")
x = int(x)
if x > 0:
    print("Vous avez saisi un nombre positif")
else: print("Vous avez saisi un nombre négatif")
```

Ce code demande à l'utilisateur de saisir un nombre entier, il teste sa valeur, si elle est supérieure à 0, un message s'affiche lui indiquant qu'il s'agit bien d'un nombre positif, sinon, un autre message s'affiche indiquant que le nombre est négatif.

Il est à noter qu'une expression booléenne peut contenir des opérateurs de comparaison. Le tableau suivant représente une liste des opérateurs logiques.

Opérateur	Signification
x == y	x est égal à y
x != y	x est différent de y
x > y	x est plus grand que y
x < y	x est plus petit que y
x >= y	x est plus grand ou égal à y
x <= y	x est plus petit ou égal à y

Exemple3

Ecrire un programme en Python qui demande à l'utilisateur de saisir deux nombres a et b et afficher le nombre le plus grand de ces deux nombres.

Correction

```
a = int(input("Tapez la valeur du nombre a : "))
b = int(input("Tapez la valeur du nombre b : "))
if (a > b):
    print("Le maximum  de a et de b est : ", a)
else:
    print("Le maximum  de a et de b ", b)
```

Les boucles

En programmation, les boucles sont utilisées pour répéter plusieurs fois l'exécution d'une partie du programme. Python dispose de deux types de boucles "for" et "while". Le choix de la boucle dépend du nombre d'itérations, si on connaît avant de démarrer la boucle le nombre d'itérations à exécuter, on choisit une boucle "for". Au contraire, si la décision d'arrêter la boucle ne peut se faire que par un test, on choisit une boucle "while". L'instruction **do…while** existe dans plusieurs langages de programmation tel que le java, PHP mais elle n'existe pas en Python, seulement, on peut reproduire son fonctionnement.

La boucle for

Ce type de boucle est utilisé quand on connait le nombre de répétitions. "for" est une instruction composée d'un en-tête se terminant par deux points :, suivie d'un bloc d'instructions indenté. Sa syntaxe :

```
for i in [0, 1, 2, 3]:
    Instruction
```

On appelle i compteur de boucle, il permet de calculer le nombre d'itérations. Voilà un exemple illustratif qui affiche une liste de valeur :

```
for i in [0, 1, 2, 3]:
    print("On affiche la valeur", i)
```

Ce code affiche :

On affiche la valeur 0
On affiche la valeur 1
On affiche la valeur 2
On affiche la valeur 3

On peut obtenir le même résultat sans spécifier une liste de valeurs, en utilisant la fonction range().

```
for i in range(4):
    print("On affiche la valeur", i)
```

La boucle while

Ce type de boucle est utilisé quand on ne connait pas à l'avance le nombre de répétitions. Sa syntaxe :

```
while condition:
    Instruction
```

Si la condition est fausse au départ, le corps de la boucle n'est jamais exécuté. Si la condition reste toujours vraie, alors le corps de la boucle est répété indéfiniment.

Voici un exemple illustratif, il s'agit du même exemple de la boucle "for" :

```
i=0
while i<4:
    print("On affiche la valeur", i)
    i=i+1
```

L'instruction break permet de « casser » l'exécution d'une boucle (while ou for). Elle fait sortir de la boucle et passer à l'instruction suivante.

```
for i in range(10):
    print("debut iteration", i)
    print("bonjour")
    if i == 2:
        break
    print("fin iteration", i)
```

```
print("apres la boucle")
```

L'instruction continue permet de passer prématurément au tour de boucle suivant. Elle fait continuer sur la prochaine itération de la boucle.

```
for i in range(4):
    print("debut iteration", i)
    print("bonjour")
    if i < 2:
        continue
    print("fin iteration", i)
print("apres la boucle")
```

La clause else dans un boucle permet de définir un bloc d'instructions qui sera exécuté à la fin seulement si la boucle s'est déroulée complétement sans être interrompue par un break.

Contrairement aux instructions présentes après la boucle, qui s'exécutent dans tous les cas (avec ou sans interruption par un break), le bloc d'instructions défini dans la clause else ne s'exécutera pas lors de l'interruption par un break. Après l'interruption, on passera directement aux instructions après la boucle.

Autrement dit, le bloc de la clause else est exécuté lorsque la boucle se termine par épuisement de la liste (avec for) ou quand la condition devient fausse (avec while), mais pas quand la boucle est interrompue par un break. Ceci est illustré dans la boucle suivante, qui recherche des nombres premiers :

```
for n in range(2, 8):
    for x in range(2, n):
        if n % x == 0:
            print(n, "egale", x, "*", n/x)
            break
    else:
        print(n, "est un nombre premier")
```

Exemple4

Ecrire un programme en langage Python qui affiche les 100 premiers nombres entiers.

Correction

```
for i in range(0,101):
    print(i)
```

Les fonctions

Le langage Python possède déjà des fonctions prédéfinies comme print() pour afficher du texte ou une variable, input() pour lire une saisie clavier... Mais il offre à l'utilisateur la possibilité de créer ses propres fonctions :

La syntaxe Python pour la définition d'une fonction est la suivante :

```
def nom_fonction(liste de paramètres):
      bloc d'instructions
```

Vous pouvez choisir n'importe quel nom pour la fonction que vous créez, à l'exception des mots-clés réservés du langage, et à la condition de n'utiliser aucun caractère spécial ou accentué (le caractère souligné « _ » est permis). Comme c'est le cas pour les noms de variables, on utilise par convention des minuscules,

notamment au début du nom (les noms commençant par une majuscule seront réservés aux classes).

Exemple

fonction qui renvoie le double d'un nombre

```
1 def maFonction(x) :
2       return 2*x
3 print ( "Le double de 5 est : " , maFonction(5) )

4 # affiche : Le double de 5 est : 10
```