OCI - Introduction à Python

David Da SILVA

Oct 14, 2020

CONTENTS

Cet ouvrage est un recueil de Jupyter Notebooks dédiés à l'introduction du langage Python Pour les élèves de l'Option Complémentaire Informatique au gymnase de Chamblandes.

Note: Ces Notebooks ont été rassemblés en ouvrage avec les outils Jupyter Book 2.0 basé sur Sphinx dans le cadre du projet ExecutableBookProject.

Ces éléments ont été rassemblés afin d'illustrer les possibilités des JupyterBook pour créer des ressources dans le cadre du projet DGEP-EPFL de création de contenu pour la Discipline Obligatoire Informatique.

CHAPTER

ONE

INTRODUCTION À PYTHON & IPYTHON NOTEBOOK

1.1 Introduction

Nous allons apprendre la langage de programmation *Python* ainsi que l'utilisation de différents modules de calculs numériques et de représentation graphique. *Python* est un langage de programmation facile à apprendre et très utilisé c'est pourquoi il a été choisi pour ce cours. Il y a de multiple façons de lancer des progammes *Python*, de la ligne de commandes aux interfaces graphiques.

Python est un langage haut niveau, orienté objet est très versatile.

Charactéristiques générales de *Python*:

- **langage simple et clair:** Le code est facile à lire et intuitif, la syntaxe est minimaliste et facile à apprendre.
- **langage expressif:** Mois de lignes de code, moins de bugs, plus facile à maintenir.

Détails techniques:

- **typage dynamique:** Pas besoin de définir le type des variables, des arguments des fonctions ou de ce qu'elles retournent.
- **gestion automatique de la mémoire:** Pas besoin d'allouer ou de désallouer de la mémoire pour les variables et les structures de données. Pas de problèmes de fuite de mémoire.
- **langage interprété:** Pas besoin de compiler le code. L'interpréteur *Python* lit et exécute le code à la volée.

Avantages:

- Le principal avantage est la facilité de programmation ce qui minimize le temps de développement, de débuggage et de maintien du code.
- La conception du langage encourage plusieurs bonnes pratiques de programmation:
- Programmation orientée objets et modulaire.
- Réutilisation du code.
- Intégration de la documentation dans le code.
- Une vaste collections de librairies et de modules additionnels.

Désavantages:

Python étant un langage interprété et à typage dynamique, l'exécution de code *Python* peut être plus longue que celle de code écrit dans des langage compilé forrtement typés tels que C++ ou Fortran.

```
print("Bonjour")
```

1.2 Les bases en Python

Ce cours se fera par le biais de *notebook ipython* comme celui-ci. Un notebook est interactif - vous pouvez modifier et faire tourner du code dans des cellules et sauvegarder le résultat. Les cellules de code commencent par "In [n]:" où n indique l'ordre dans lequel les cellules ont été interprétées. Le résultat de l'interprétation du code est normalement affichée dans une cellule directement sous-jacente commençant par "Out [n]:". Pour exécuter le contenu d'une cellule, utilisez *shift-enter* ou sélectionnez *Run* dans le menu *Cell*.

Il est conseillé d'exécuter chaque cellule de code et de s'assurer d'avoir compris pourquoi un certain résultat est produit. Certaines cellules de code dépendent de définitions faites dans des cellules précédentes, vous obtiendrez les meilleurs résultats en exécutant les cellules de manière séquentielles. Au fur et à mesure que vous avancerez dans un notebook, vous trouverez des exercices marqués "Exercice", faites-les !

Quelques instructions de base

- Cliquez sur les bouton `Play` pour exécuter et passer à la cellule suivante. Le raccourci clavier est `shift-enter`
- Pour ajouter une nouvelle cellule sélectionnez le menu `"Insert->Insert New Cell Below/Above"` ou cliquez sur le bouton '+'
- Vous pouvez changer le mode code d'une cellule vers le mode texte en utilisant le menu déroulant.
- Vous pouvez changer le contenu d'une cellule en double-cliquant dessus.
- Pour sauvegarder votre notebook, sélectionnez le menu `"File->Save and Checkpoint"` ou pressez `Ctrl-s` ou `Command-s` sur un Mac
- Pour annuler une opération, pressez `Ctrl-z` ou `Command-z` sur un Mac
- Pour annuler la supression d'une cellule faite par le menu `Edit->Delete Cell`, sélectionnez le menu `Edit->Undo Delete Cell`
- Le menu `Help->Keyboard Shortcuts` propose une liste de raccourcis clavier

Ce notebook est inspiré des notebooks suivants:

- Lab1-IntroductionToPython de Alexei Gilchrist
- A Crash Course in Python for Scientists de Rick Muller
- Lectures on Scientific Computing with Python de J.R. Johansson.

1.2.1 Arithmetique

Vous pouvez utiliser python comme une calculatrice, les priorités de calcul sont respectées. Par exemple, $9 \times (2 + 3) - 40 + 2^2$ s'écrit :

```
9 * (2 + 3) - 40 + 2
```

Dans un notebook, le résultat d'une cellule est celui de la dernière commande.

```
12
12 * 12
12 * 12 * 12
12 * 12 * 12 * 12
print("Toto")
```

Pour afficher les résultats intermédiaires, utiliser la fonction `print`

```
print(3 * 7)
```

Exercice

Modifiez le code ci-dessous afin d'afficher tous les résultats intermédiaires

```
12
12*12
12*12*12
12*12*12*12
```

Les nombres complexes sont définis de manières native en python. Un nombre imaginaire est suivi de la lettre "j":

```
2j
```

En général un nombre complexe et l'addition d'un réel et d'un nombre imaginaire, par exemple:

```
2 + 1j + 4 + 2j
```

et

```
(2 + 1j) * (4 + 2j)
```

Exercice

Étant donné le nombre imaginaire pur $i = \sqrt{-1}$, calculez i^i :

1.2.2 Variables

Les variables sont définies à l'aide du symbole égal (**=**) et la valeur peut être n'importe quel objet Python (nous reviendrons plus en détails sur la notion d'objet un peu plus tard). Le type d'objet affecté à une variable peut changer durant l'exécution d'un programme, il n'est pas définitif - c'est le typage dynamique. Le nom d'une variable doit commencer avec une lettre ou une underscore (**_**) et peut contenir des caractères alphanumériques et des underscores. **Prenez l'habitude de donner des noms significatifs à vos variables.**

```
A = 10
B_2 = 5
A_plus_B_2 = A + B_2
print(A**B_2)

A = "Five "
print(A * B_2)
```

Certains noms sont réservé pour le langage et ne peuvent pas être utilisés pour des variables:

```
and, as, assert, break, class, continue, def, del, elif, else, except,
exec, finally, for, from, global, if, import, in, is, lambda, not, or,
pass, print, raise, return, try, while, with, yield
```

Une fonctionalité très pratique de *iPython* est la complétion automatique. Dans la cellule suivante, tapez `A_` puis la touche TAB (->|).

Exercice

Affectez les valeurs suivantes aux variables correspondantes:

- `7` dans la variable `my_int`
- `2.21` dans la variable `my_float`
- `True` dans la variable `my_bool`

Effectuez les opérations suivantes:

- `my_int` divisé par `my_float`
- `my_float` à la puissance `my_int`
- `my_bool` plus `my_int`

Que se passe-t-il pour la troisième opération si on affecte `False` à `my_bool` ? Que peut-on en déduire au sujet des variables booléennes en *Python* ?

Ecrivez votre réponse dans une nouvelle cellule ci-dessous.

1.2.3 Strings (chaîne de caractères)

En *Python* les chaînes de caractères (*string*) sont défini en utilisant des paires de guillemets simple (‘) ou double (“). Cela permet d’inclure des apostrophes ou des citations dans des chaînes de caractères.

```
print("Albert O'Connor")
print('Je lui ai dit "Salut!"')
```

Certains opérateurs mathématiques ont été surchargés pour fonctionner avec des chaînes de caractères. Par exemple l’opérateur ‘**+**’ concatène les strings et ‘*****’ les répète.

```
"Good" + " " + "Morning!"
```

```
"=" * 30
```

Commentaires

Python ignore tout ce qui se trouve sur une ligne précédée du symbole #, ce qui permet de rajouter des commentaires à votre code ou d’en désactiver temporairement une partie. Si ce symbole est contenu dans une string, il est considéré comme un caractère normal et les caractères suivant ne sont pas ignorés. Prenez l’habitude de **toujours commenter votre code**, même si vous ne l’écrivez que pour vous. Quelques mots d’explications rendront votre code beaucoup plus lisible et vous feront gagner du temps lorsque vous y reviendrez plus tard.

```
# this is a comment and won't get evaluated
# a = 10
a = 15

a
```

Exercice

Corrigez l'erreur dans l'affectation suivante et ajoutez un commentaire qui explique la modification faite et pourquoi.

```
welcome_message = "Buenos dias! "'
```

1.2.4 Modules

La plupart des fonctionnalités de *Python* sont fourni par des *modules* qui peuvent être importés afin de fournir des constantes, des fonctions, des classe, etc. *Python* contient de base une grande quantité de ces modules sous la dénomination de la **Python Standard Library**. Cette librairie fournit des outils pour manipuler des fichiers, des dossiers, lire des données, explorer du XML, etc. Par exemple le module *math* contient de nombreuses fonctions et constantes mathématique qui peuvent être utilisées une fois le module *importé*.

```
import math
```

Nous pouvons maintenant utiliser la constante π

```
print(math.pi)
```

Pour utiliser un élément d'un module, celui-ci doit être précédé du nom du module qui le défini. Afin d'éviter d'avoir à écrire le nom du module trop souvent, il est possible d'importer certains éléments du module dans l'espace de nom (*namespace*) local.

```
from math import pi, cos

cos(pi)
```

Il est également possible d'importer tous les éléments d'un module en utilisant la commande suivante

```
from math import *

sin(pi / 2.0)
```

L'avantage de cette approche est qu'elle permet d'utiliser toutes les fonctions et constantes du module *math* sans avoir à taper le préfixe "math". Le problème est que maintenant le namespace local contient tout un tas d'éléments qui ne sont pas forcément souhaités et surtout qui risquent d'entrer en conflit avec d'autres éléments définis auparavant. C'est pourquoi il est fortement conseillé d'utiliser plutôt une des deux premières méthodes.

Exercice

Affectez à la variable `x` la valeur $\frac{3e}{\pi}$. Calculer l'expression suivante : $y = \sqrt{\log(7x^2 + 21x - cos(\frac{\pi}{4})}$

Un peu d'aide ?

Il existe plusieurs manières d'obtenir de l'aide ou de savoir ce qui est disponible sans pour autant avoir besoin de documentation extérieure ou de Google. La fonction intrinsèque *dir* affiche tous les éléments d'un module ou d'une classe et permet d'avoir un aperçu de ce qui est disponible.

```
print(dir(math))
```

La fonction *help* est elle aussi très utile:

```
help(math)
```

Ce Notebook est a été crée par David Da SILVA - 2020

CHAPTER

TWO

LES LISTES EN *PYTHON*

2.1 Définition et instanciation

La liste en *Python* (`List`) est le type de données le plus flexible.C'est une séquence d'éléments, qui peut être modifiée (suppression/ajout) et découpée (`Slice`).

Les listes en *Python* sont définies en utilisant des crochets [ ] et en plaçant des éléments à l'intérieur qui sont séparés par des virgules.Les éléments de la liste sont indexés (`index`) de gauche à droite, le premier index vaut 0.

Attention : Les *listes* définies à l'aide de parenthèses sont appelées des `tuples` et leur contenu n'est pas modifiable.

```
weekdays = ["Monday","Tuesday","Wednesday","Thursday","Friday","Saturday","Sunday"]
```

2.2 Accéder aux éléments d'une liste

Pour accéder à un élément en particulier de la liste on peut utiliser son index (le numéro de sa place dans la liste) noté entre [ ].**Attention :** en *Python* les index commence à 0. Le premier élément de la liste s'obtient donc avec la notation `maliste[0]`.

```
print( weekdays[3] )
```

2.2.1 Index négatifs

Vous pouvez également parcourir les éléments de la liste en commençant par la fin en utilisant des index négatifs. Le dernier élément se trouve à l'index -1, l'avant dernier à l'index -2, etc.

```
weekend = [ weekdays[-2], weekdays[-1] ]
print(weekend)
```

2.2.2 Sous-liste | `slicing`

Pour extraire une *tranche* de la liste on utilise le symbole **:** qui séparent l'index de début et l'index de fin.**Attention :** l'élément à l'index de fin est exclu de la liste (i.e. [*index_début ; index_fin*[ )

```
weekdays[0:5]
```

Exercice

Créez une sous-liste allant de la fraise au kiwi

```
fruitlist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
```

```
sublist = fruitlist[2:5]
print(sublist)
```

Remarques

Si l'index de début est omis, la *tranche* commencera au début de la liste. Si c'est l'index de fin qui est absent, la *tranche* finira à la fin de la liste.Une *tranche* peut également être définie avec des index négatifs.

Exercice

Créez la même sous-liste que précédement allant de la fraise au kiwi, mais cet fois avec des index négatifs

2.2.3 Listes imbriquées

Les listes ne sont pas forcément constituées d'éléments de même type. Vous pouvez faire tous les mélanges que vous voulez y compris mettre imbriquer des listes dans des listes ou des tuples dans des tuple et vice versa.

```
mixedlist = [ 1.0, 100, "Elephant", ("mouse", "rat"), weekend ]
mixedlist
```

Pour accéder aux éléments imbriqués on utilise autant de niveaux de [ ] qu'il y a de niveaux d'imbriquation. Par exemple pour accéder au premier élément de la liste `weekend` imbriquée dans `mixedlist` on utilisera la notation suivante :

```
mixedlist[-1][0]
```

2.3 Modification de liste

Les listes ont de nombreuses méthodes utiles qui sont prédéfinies comme par exemple pour ajouter ou supprimer des éléments. Des explications concises sont accessibles via la commande `help(list)`. Ignorez les méthodes qui commencent par un underscore (_) ; ces méthodes sont utilisées pour définir des opérations internes comme l'initialisation d'une liste, les surcharges d'opérateurs, etc.

2.3.1 Modification d'un élément

Il est possible de modifier un élément d'une liste en y accédant par son `index`.

```
fruitlist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
fruitlist[3] = "kumquat"
print(fruitlist)
```

Exercice

Modifiez *Wednesday* dans la liste `weekdays` en *Mercredi*.

Modifiez le deuxième élément de la liste `weekend` imbriquée dans `mixedlist` pour **Samedi**

```
mixedlist = [ 1.0, 100, "Elephant", ("mouse", "rat"), weekend ]
```

```
mixedlist[4][1] = "Samedi"
print(mixedlist)
```

2.3.2 Ajout d'un élément

En fin de liste

La méthode **`append(x)`** ajoute l'élément *x* en fin de liste.

```
fruitlist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
fruitlist.append("kumquat")
print(fruitlist)
```

À un endroit spécifique

La méthode **`insert(idx, x)`** insère l'élément *x* à l'`index` *idx*.

```
fruitlist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
fruitlist.insert(2,"kumquat")
print(fruitlist)
```

Concaténation de listes

Il y a plusieurs façons de concaténer 2 listes ou plus. La méthode la plus simple est sûrement l'utilisation de l'opérateur +.

```
list1 = ["a", "b" , "c"]
list2 = [1, 2, 3]

list3 = list1 + list2
print(list3)
```

On peut également utiliser la méthode **`extend(iter)`** et qui va rajouter les éléments d'une liste (ou une autre structure itérable) à une autre liste.

```
list1 = ["a", "b" , "c"]
list2 = [1, 2, 3]

list1.extend(list2)
print(list1)
```

Exercices

1. Ajoutez le papmplemousse (en anglais) à la fin de `fruitlist`.
2. Insérez le citron (toujours en anglais) entre l'orange et le kiwi.
3. Supprimez le **melon** de la liste.
4. Imprimez l'avant-dernier élément de la liste en utilisant un index négatif.
5. Créez une nouvelle liste `fruitday` qui est la concaténation de `weekdays` et `fruitlist`.

2.3.3 Suppression

Supprimer un élément spécifique

La méthode **`remove(elmt)`** supprime l'élément *elmt*.

```
fruitlist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
fruitlist.remove("orange")
print(fruitlist)
```

Supprimer selon une position

La méthode **`pop(idx)`** supprime l'élément se trouvant à la position *idx* et le retourne (`return`).Si *idx* n'est pas précisé, `pop()` supprime et retourne le dernier élément de la liste.

```
fruitlist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
fruit = fruitlist.pop(3)
print(fruit)
print(fruitlist)

lastfruit = fruitlist.pop()
print(lastfruit)
print(fruitlist)
```

Effacement

Le mot clé `del` peut aussi être utilisé pour effacer un élément dont l'`index` est connu.

```
fruitlist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
del fruitlist[2]
print(fruitlist)
```

Attention : le mot clé `del` efface aussi le contenu d'une variable.

```
fruitlist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
del fruitlist
print(fruitlist)
```

Pour vider le contenu d'une liste, il faut utiliser la méthode **`clear()`**.

```
fruitlist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
fruitlist.clear()
print(fruitlist)
```

Exercice

Créez une nouvelle liste `monovowelfruit` dans laquelle vous déplacerez les fruits de `fruitlist` dont le nom ne possède pas 2 voyelles différentes. Ces fruits ayant été déplacés, ils ne seront plus dans `fruitlist`.

```
fruitlist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]



print(fruitlist)
```

2.3.4 Copier une liste

L'affectation d'une liste existante à une nouvelle variable (`fruitcopy = fruitlist`) ne créé pas une copie de la liste, mais créé une référence. Toutes modifications de la liste initiale affectera la référence.

```
fruitlist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
fruitcopy = fruitlist

fruitlist[3] = "peach"

print(fruitcopy)
```

Pour créer une copie on peut utiliser la méthode **`copy()`** qui va générer une copie de la liste.

```
fruitlist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
fruitcopy = fruitlist.copy()

fruitlist[3] = "peach"

print(fruitcopy)
```

On peut également utiliser le *constructeur* de listes : **`list(elmt)`**, qui créer une liste à partir de `elmt`.

```
fruitlist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
fruitcopy = list(fruitlist)

fruitlist[2] = "peach"

print(fruitcopy)
```

2.4 Méthodes de liste

Complétez les descriptions des méthodes suivantes.Pour celles que vous ne connaissez pas, utilisez la fonction `help` afin de truover une description, puis essayez de les utiliser dans des exemples.

Méthode	Description
append()	
clear()	
copy()	
count()	
extend()	
index()	
insert()	
pop()	
remove()	
reverse()	
sort()	

Besoin d'aide ? : par ici

2.5 Fonctions supplémentaires

2.5.1 `len()`

Il existe des fonctions prédifinies qui peuvent opérer sur les liste telles que la fonction **`len`** qui permet d'obtenir la taille d'une liste.

```
len(fruitlist)
```

Exercice

Imprimez le dernier élément de la liste `weekdays` en utilisant la fonction `len` (pas d'index négatif).

Pourquoi la commande `weekdays[len(weekdays)]` ne fonctionne-t-elle pas ?

votre réponse ici

2.5.2 `range()`

Une commande très utile pour générer des séquences de nombres : **`range`**. Cette fonction peut prendre 1, 2 ou 3 paramètres.

Avec 1 paramètre elle permet de générer la liste des entiers inférieurs au paramètre.

```
list(range(10))
```

Avec 2 paramètres vous spécifiez la valeur de début et celle de fin (non inclue).

```
list(range(2,10))
```

Avec 3 paramètres vous spécifiez la valeur de début, de fin et le pas.

```
list(range(2,10,2))
```

Cette commande est particulièremet utile pour les *boucles* que nous aborderons par la suite.

Exercices

Utilisez la fonction `range()` pour générer la liste de tous les multiples de 7 inférieurs 4103.

En utilisant des listes imbriquées, créez une structure pour représenter la matrice 3x3 suivante : $A = \begin{pmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{pmatrix}$. Vos imbrications de listes devront permettre à la commande `A[l][c]` d'accéder à l'élément se trouvant à la ligne `l` et la colonne `c` ; e.g. `A[1][2]=8` et `A[2][0]=3` (Attention les index commencent à 0).

```
A = [[1,4,7],[2,5,8],[3,6,9]]
print(A)
print(A[1][2])
print(A[2][0])
```

CHAPTER

THREE

BOOLÉENS ET TESTS

3.1 Vrai ou Faux

En plus des différents types déjà vus, il existe un type un peu particulier appelé **Booléens** qui prend uniquement 2 valeurs, **vrai** ou **faux**. En *Python* un booléen est soit `True` soit `False`. Les booléens servent à donner le résultat d'une comparaison entre 2 objets. *Python* permet de comparer 2 objets de plusieurs manières :

- `A==B`: est-ce que `A` est égal à `B`?
- `A!=B`: est-ce que `A` est différent de `B`?
- `A>B`, `A>=B`, `A<B`, `A<=B`: est-ce que `A` est plus grand, plus grand ou égal, plus petit, plus petit ou égal à `B`?
- `A is B`: est-ce que `A` est le même objet que `B` ? (pas uniquement sa valeur)
- `A in B`: est-ce que `B` contient `A` ? (si `B` est un container comme une liste par exemple)

```
9**2 == 10+8*10-9
```

```
[1,2,3]==[1,2,3]
```

```
[1,2,3]==[1,3,2]
```

```
3.0 in ['Yay!', 'Python', 3.0]
```

Ces tests peuvent être combinés entre eux en utilisant des opérateurs logiques.

```
A = 30

( A >10 ) or ( A%3==0 )
```

3.2 Opérations et transformation

Les opérateurs logique opèrent sur des booléens et renvoient des booléens en fonction de leurs tables. Il existe 2 opérations, **ET** et **OU** et 1 transformation appelée **contraire**.

Les tables sont les suivantes:

and	True	False
True	True	False
False	False	False

or	True	False
True	True	True
False	True	False

-	not
True	False
False	True

En *Python* ces opérateurs sont notés **and**, **or** et **not** et les résultats des tables peuvent être obtenus en appliquant les fonctions sur des booléens.

```
print(True and False)
print(False or True)
print(not False)
```

3.3 Table de vérité

Une table de vérité est une table mathématique utilisée en logique pour représenter de manière sémantique des expressions logiques et calculer la valeur de leur fonction relativement à chaque combinaison de valeur assumée par leurs variables logiques.

Les tables de vérité peuvent être utilisées en particulier pour dire si une proposition est vraie pour toutes les valeurs légitimement imputées.

Considérons les 2 variables `A` et `B`, et pour simplifier la lecture nous associerons la valeur 0 à `False` et la valeur 1 à `True`.

Les tables de vérité pour les 3 opérateurs s'écrivent :

A	B	not A	not B	A and B	A or B
0	0	1	1	0	0
0	1	1	0	0	1
1	0	0	1	0	1
1	1	0	0	1	1

3.3.1 Exercice

En utilisant `Python` pour faire les tests, remplissez la table de vérité suivante, et conservez **tous** les blocs de code `Python` que vous avez utilisé.

A	B	not (A and B)	(not B) or (not A)
0	0	?	?
0	1	?	?
1	0	1	?
1	1	?	?

```
A = True
B = False
```

```
not (A and B)
```

3.3.2 Questions

Que remarquez-vous ?

Quelle relation pensez-vous qu'il y ait entre `not(A or B)` et `(not A) and (not B)` ?

Connaissez-vous les lois de *De Morgan*?

Vos réponses ici

3.4 Bloc de test

Une particularité de *Python* est l'utilisation des indentation afin de rassembler des informations au sein d'un bloc. Les indentations dans *Python* remplace les accolades **{ }** dans d'autres langages tels que **C++**.

Afin de définir un bloc de code en *Python* comme pour les tests, les fonctions ou les boucles, il faut utiliser le double point **:** suivi d'un niveau d'indentation.

Voyons par exemple la structure d'un bloc conditionnel de type *SI...ALORS...SINON* qui se traduit en *Python* par les commandes `if` et `else`.

```
if A==B:
  # les instructions de ce bloc sont exécutées
  # si A est égal à B
else:
  # sinon ce sont les instructions de ce bloc
  # qui seront exécutées
```

Un bloc conditionnel de manière général possède la structure légèrement plus complexe suivante:

```
if A==B:
  # les instructions de ce bloc sont exécutées
  # si A est égal à B
elif A==C:
  # celles de ce bloc sont exécutées
  # si A est égal C
  # ET si le premier test est faux (renvoi False)
elif test2:
  # celles de ce bloc sont exécutées
  # si test2 est vrai (renvoi True)
  # ET si les deux premiers tests sont faux (renvoi False)

.
.
.

else:
    # finalement ces instructions sont exécutées si
    # aucun des tests précédents n'est vérifié
```

Il peut y avoir autant de `elif` que vous voulez mais il ne peut y avoir qu'un seul `else` (c'est un Highlander!)

Tous les `elif` ainsi que le `else` peuvent être omis.

Modifiez les valeurs de `A` dans le code suivant et observez ce qui se passe.

```
A = 15

if A > 9:
    print("A est plus grand que 9")
elif A > 4:
    print("A est compris entre 5 et 9")
else:
    print("A est plus petit que 5")
```

3.4.1 Exercice

Créez un test afin de déterminer si un nombre est un carré (i.e 1, 4, 9, 16, …), et créez un bloc conditionnel qui affiche soit "carré" soit "pas un carré" en fonction du résultat du test.

Indice : `int(d)` retourne la partie entière du nombre décimal `d`.

Testez votre bloc pour différentes valeurs.

3.5 Interraction avec l'utilisateur

Afin de rendre vos script un peu plus interractifs, il est possible de demander à python d'utiliser une entrée clavier pour l'utiliser ensuite dans votre script.

La commande à utiliser est `input()`. Elle provoque l'ouverture d'un champ dans lequel l'utilisateur doit rentrer une donnée au clavier et attend l'appui de la touche `Entrée`.

3.5.1 Exemple

```
A = 20
toto = input()
print("Vous avez saisie : {0} et A vaut {1} ".format(toto,A))
```

La fonction `input()` peut contenir un paramètre qui sera affiché devant la fenêtre de saisie

```
import sys
reload(sys)
sys.setdefaultencoding("utf-8")
```

```
yr = input("Quel est votre année de naissance ?")
print("Je devine que vous avez environ {0} ans ! \nJe suis fort hein !?!".format(2020-
↪int(yr)))
```

3.5.2 Exercices

- Écrivez un programme qui demande un nombre à l'utilisateur, puis calcule et affiche le carré de ce nombre
- Écrivez un programme qui demande un nombre à l'utilisateur et l'informe ensuite si ce nombre est pair ou impair
- Écrivez un programme qui demande l'âge d'une personne et affiche ensuite sa catégorie :
    1. "Poussin" de 6 à 7 ans
    2. "Pupille" de 8 à 9 ans
    3. "Minime" de 10 à 11 ans
    4. "Cadet" après 12 ans
    5. "Junior" après 16 ans
    6. "Senior" après 18 ans

Ce Notebook est a été crée par David Da SILVA - 2020

CHAPTER

FOUR

BOUCLES EN *PYTHON*

Il existe 2 types de boucle en *Python*: le boucles `for` et les boucles `while`. Elles s'écrivent de la manière suivante :

```
for variable1 in list:
    # variable1 prend chacune des valeurs de la liste
    code
```

et

```
while test_est_True:
    # Le code est répété tant que le test retourne True
    code
```

La commande `continue` permet de sauter certaines itérations au sein d'une boucle. Essayez de deviner le comportement de la boucle suivante avant de l'exécuter et notez le ici :

what's your guess ?

4.1 Boucle `for`

Une boucle `for` va parcourir tous les éléments d'une séquence (quelque soit son type : liste, tuple, dictionnaire, ensemble, chaîne de caractères) et exécuter des instructions à chaque fois.Par exemple, `range(20)` crée la séquence des entiers de 0 à 19 (la borne de fin est exclue).

On peut *itérer* sur chacune de ces valeurs et effectuer des opérations:

```
for i in range(20):
    print("Le cube de {0:>2} est {1:>4}".format(i, i**3))
```

4.1.1 Itération

Une boucle `for` peut *itérer* sur une chaîne de caractère :

```
for x in "banana":
    print(x)
```

sur une liste :

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
    print(x)
```

sur une liste mixte :

```
for i in [4, 6, "asdf", "jkl"]:
    print(i)
```

Remarque : Les opérations ne sont pas forcément en lien avec le compteur de la boucle.

```
A = 21
print(A)

for i in range(5):
    print("Hello")
    A += 2

print(A)
```

Exercices

Changer le `print()` de la boucle ci-dessus afin que les éléments s'impriment tous de manière centrée sur une ligne de 30 caractères remplies de "=".

i.e. : "==========5=========="

Indice: utiliser le formating mini-language

```
for i in [4, 6, "asdf", "jkl"]:
    print(i)
```

```
for i in [4, 6, "asdf", "jkl"]:
    print("{0:=^30}".format(i))
```

Écrivez un programme qui affiche une suite de 12 nombres dont chaque terme est égal au triple du terme précédent

Modifiez ce programme afin que ce soit l'utilisateur qui choisisse le première valeur

4.1.2 Énumération

Un besoin assez courant quand on manipule une liste ou tout autre objet itérable est de récupérer en même temps l'élément et son indice à chaque itération.Pour cela, la méthode habituellement utilisée est simple : au lieu d'itérer sur notre liste, on va itérer sur une liste d'entiers partant de 0 et allant de 1 en 1 jusqu'au dernier indice valide de la liste, obtenue via la fonction `range()`.Cette méthode n'est pas du tout efficace : en effet, manipuler ainsi l'indice est totalement contre-intuitif et va à l'encontre du principe des itérateurs en Python.

Un exemple de code utilisant cette mauvaise méthode :

```
mylist = [4, 6, "asdf", "jkl"]

for indice in range(0, len(mylist)):
    print("mylist[%d] = %r" % (indice, mylist[indice])) #une autre manière de␣
↪formater des chaînes de caractère
```

Pour réaliser ce genre d'itérations, on va utiliser la fonction **`enumerate(iter)`** Elle permet en effet de récupérer une liste de tuples `(indice, valeur)` en fonction du contenu de la séquence et d'une manière très pratique.On l'utilise comme ceci :

```
for indice, valeur in enumerate(mylist):
    print("mylist[{0}] = {1}".format(indice, valeur)) #c'est ce mode de formatage qu
↪'il faut préférer
```

Cette manière de faire se rapproche beaucoup plus de ce qui doit être fait en *Python* si l'on veut utiliser correctement le langage : on itère directement sur les valeurs à la sortie d'un générateur, au lieu d'utiliser un indice (manière plus courante dans les langages comme le C n'ayant pas d'itérateurs comme ceux de *Python*).

Exercice

Affichez chaque lettre du mot suivant suivi de sa position dans le mot : *supercalifragilisticexpialidocious*

4.2 Boucle `while`

4.2.1 Définition

La boucle `while` ne parcourt pas de séquence, mais vérifie une condition à chaque tour. Si cette condition est vérifiée, les instructions du corps de la boucle sont effectuées. Dans le cas contraire, la boucle s'arrête et les instructions ne sont pas executées.

```
i = 0
while i < 20:
    print("Le cube de {0:>2} est {1:>4}".format(i, i**3))
    i = i+1
```

Attention : Une boucle `while` dont la condition est toujours vérifiée ne s'arrêtera jamais. On parle de boucle infinie. C'est une erreur classique qui peut avoir des erreurs dramatiques, programme ne s'arrête pas, saturation de la mémoire, crash…

```
#Si vous exécutez la boucle ci-dessous, il vous faudra appuyer sur le bouton Stop ( à␣
↪droite de >| Run)
#avant de pouvoir continuer à travailler
i = 1
while i > 0:
    print("Le cube de {0:>2} est {1:>4}".format(i, i**3))
    i = i + 1
```

4.2.2 Exercices

Utilisez une boucle `while` pour afficher tous les multiples de 21 inférieurs à 2538

Écrivez un programme qui affiche chaque lettre du mot *supercalifragilisticexpialidocious* précédant la lettre *x*.

4.2.3 Exercice

Écrivez un programme qui demande à l'utilisateur un nombre compris entre 21 et 42 jusqu'à ce que la réponse convienne en l'aiguillant (trop grand, trop petit…)

4.3 `break` et `continue`

Parfois il est nécessaire de sortir d'une boucle avant la fin de son exécution. Dans ces cas là, il faut utiliser le mot clé `break` qui fonctionne pour les 2 types de boucle `for` et `while`.

Par exemple, `while True:` crée une boucle infinie pusique le test est toujours vrai, mais l'instruction `break` permet d'en sortir :

```
i = 0
while True:
    i = i + 10
    if i > 95:
        break
print(i)
```

La commande `continue` permet de sauter certaines itérations au sein d'une boucle.Essayez de deviner le comportement de la boucle suivante avant de l'exécuter et notez le ici :

what's your guess ?

```
for x in range(30):
    if x%3!=0:
        continue
    print(x)
```

4.4 `for` vs. `while`

4.4.1 Questions

1. Quelles sonts les différences que vous notez entre les 2 types de boucle
2. Selon vous y a-t-il un boucle plus rapide que l'autre ? Pourquoi ?

Vos réponses ici

4.4.2 Testons

Le module `time` founit la fonction `time()` qui renvoit le temps courant depuis `Epoch`.En utilisant cette fonction, calculer la temps mis pour calculer les carrés des chiffres de 0 à 10^7 avec une boucle `for` et une boucle `while`.

Conseil : éviter de faire un `print` pour chaque ligne de calcul…

Au fait, quel est le `Epoch` de *Python* ?

```
import time

start = time.time()
```

(continues on next page)

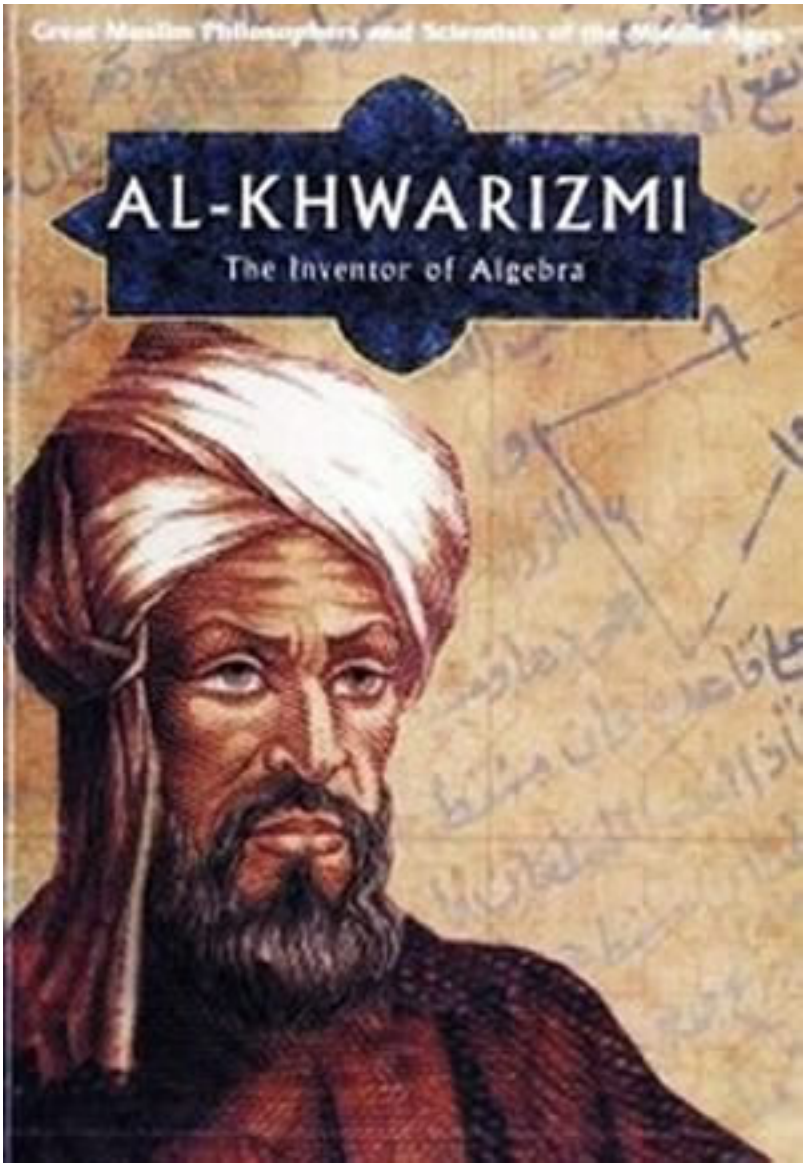
(continued from previous page)

```
#code de la boucle à évaluer
stop = time.time()

print(stop-start)
```

Ce Notebook est a été crée par David Da SILVA - 2020

Source: openclassrooms.com



CHAPTER

FIVE

MOHAMMED AL-KHWARIZMI

Le mot français « algorithme » provient du nom d'un savant arabe du IX^{me} siècle : **Mohammed al-Khwarizmi** (Khiva vers 788 — vers 850 Bagdad) qui fut l'un des inventeurs de l'algèbre et du système décimal. C'est également grâce à lui que se diffuseront les chiffres arabes en Occident.

Le premier ouvrage *al-Kitâb al-mukh- tasar fâ hisâb al-jabr w'al-muqâbala*, le Livre de l'explication du calcul de la remise en place et de la simplification, a donné son nom à l'algèbre.

Al-Khwarizmi y présente une exposition complète de la résolution des équations du premier et du second degré. L'inconnue, que nous notons `x`, s'appelle la *racine* et, comme il a éliminé tout nombre négatif, il distingue six cas et les traite sur des exemples qui se généralisent sans difficulté pour toute équation de même type.

Il considère ainsi :

- Carrés égaux aux racines, c'est-à-dire de la forme $ax^2 = bx$;
- Carrés égaux à un nombre, soit $ax^2 = c$;
- Racines égales à un nombre, soit $bx = c$;
- Carrés et racines égaux à un nombre, soit $ax^2 + bx = c$;
- Carrés et nombre égaux aux racines, soit $ax^2 + c = 6x$;
- Racines et nombres égaux aux carrés, soit $bx + c = ax^2$;

où `a`,`b` et `c` désignent des nombres positifs.

Contrairement aux mathématicien grecs, al-Khwarizmi détaille des méthodes effectives de résolution d'équations.

Historiquement liée au calcul, la notion d'algorithme s'est progressivement étendue à la manipulation de différents objets, des textes et des images par exemple.

CHAPTER

SIX

LES ALGORITHMES

Un algorithme est simplement une méthode qui sert à résoudre un problème en un nombre fini d'étapes : chercher un mot dans le dictionnaire, classer des mots par ordre alphabétique, classer des nombres par ordre de grandeur, chercher le meilleur parcours possible sur une carte, trouver une racine carrée, construire des listes de nombres premiers, etc.

On peut décrire un algorithme comme étant une suite d'actions à accomplir séquentiellement, dans un ordre fixé.

6.1 Définitions : Algorithme ≠ Programme

6.1.1 Algorithme

Un algorithme est la description abstraite des étapes *simples* conduisant à la résolution d'un problème. C'est la partie conceptuelle d'un programme.

6.1.2 Programme

Un programme est l'implémentation d'un algorithme dans un langage donné et sur un système particulier.

6.1.3 Exemple

Décrivez ci-dessous un algorithme pour trouver le maximum d'une *longue* liste de nombres entiers :L = (17, 23218, 543, 7, 1984, 2000000, 21, … , 3, 666, 69, 0, 42)

Ecrire votre algorithme ici

6.2 Ingrédients de base des algorithmes en pseudo-code

6.2.1 Données

Elles peuvent être de 3 types:

1. entrées
2. sorties
3. internes

6.2.2 Instructions

Affectations

Typiquement mettre une valeur ou un résultat dans une variable$x \leftarrow 4\Delta \leftarrow b^2 - 4ac$

Instructions de contrôle

1. branchements conditionnels ou test : *si …. alors …. sinon*
2. itérations ou boucles : *pour .. allant de .. à ..* ; *pour tous les éléments de … répéter …*
3. boucles conditionnelles : *tant que (test est vrai) répéter …*

Exemple : Que fait cet algorithme ?

Algorithmeentrée : N entier positifsortie : ??$i \leftarrow 0$**Tant que** $2^i \leq N$ $i \leftarrow i + 1$**Sortir :** i

La sortie de cette algorithme représente le nombre de bits nécessaires pour représenter N en binaire

6.2.3 Que font les algorithmes suivants

Algorithme 1

entrée : a, b deux entiers naturels non nulssortie : ??$x \leftarrow a y \leftarrow b z \leftarrow 0$**Tant que** $y \geq 1$ **Si** y est pair $x \leftarrow 2x$ $y \leftarrow y/2$ **Sinon** $z \leftarrow z + x$ $y \leftarrow y - 1$**Sortir :** z

Algorithme 1 :

Note : Les algorithmes existent depuis bien avant les ordinateurs! En particulier, l'algorithme ci-dessus nous vient de l'Egypte ancienne.

Algorithme 2

entrée : n entier naturelsortie : ??$m \leftarrow n i \leftarrow 1$**Tant que** $m \geq 0$ $i \leftarrow 2i$ $m \leftarrow m - 1$**Sortir :** i

Algorithme 2 :

Algorithme 3

entrée : a, b deux entiers naturels non nulssortie : ??$s \leftarrow 0$**Si** $a \leq b$ **Pour** i allant de 1 à a $s \leftarrow s + b$**Sinon** **Pour** i allant de 1 à b $s \leftarrow s + a$**Sortir :** s

Algorithme 3 :

6.2.4 Créations d'algorithme

Pour ces exercices, la syntax n'est pas primordiale, ce qui est important est de s'assurer que votre algorithme possède des entrées et des sorties, que les différentes étapes sont des opérations *simples* qui se suivent dans le bon ordre afin d'obtenir le résultat. Assurez-vous surtout que votre algorithme s'arrête bien !

Somme de multiples

Écrivez un algorithme qui calcule la somme des *n* premiers nombres entiers faisant partie de la liste de multiple de 5 et de 7.Pour n = 5, cette somme vaut 5+7+10+14+15 = 51.

Somme de multiples : votre algorithme ici

Comptage de mot

Soit A une chaîne de caractères formée uniquement de mots et d'espaces (uniques) entre les mots, et soit n sa longueur (exemple: *A* ="Le silence des agneaux" et donc *n* = 22). Écrivez un algorithme dont les entrées sont *A* et *n*, et dont la sortie est le nombre de mots de la chaîne (4 dans l'exemple).

Comptage de mot : votre algorithme ici

Les deux plus grands

Soit *L* une liste de nombres entiers positifs de taille *n* (exemple: *L* = {3,43,17,22,16} et donc *n* = 5). Ecrivez un algorithme dont l'entrée est *L* et *n*, et dont la sortie sont les deux plus grands nombres de la liste (dans l'exemple: 43 et 22).

Les deux plus grands : votre algorithme ici

6.2.5 La méthode al-Khwarizmi

Voici ci-dessous l'algorithme de al-Khwarizmi pour résoudre toutes les équations du type $x^2 + bx = c$, où $\mathtt{b}$ et $\mathtt{c}$ sont des nombres positifs.

> On prend la moitié des racines ; on la met au carré, que l'on additionne au nombre. Prenons alors la racine carrée de ce nombre et ôtons-lui la moitié des racines pour obtenir la solution.

Rappel : L'inconnue, que nous notons $\mathtt{x}$, s'appelle la *racine*.

Réécrivez cet algorithme en pseudo-code ci dessous:

Algorithme d'al-Kwharizmi : pseudo code à mettre ci-dessous

6.2.6 Implémentation

Programmer consiste à transmettre à un ordinateur, à l'aide des instructions d'un langage, l'algorithme qu'il doit appliquer pour parvenir au résultat qu'on lui demande d'établir.

Écrivez en *Python* l'implémentation des algortihmes des exercices précédents.

Pour pouvoir réaliser cet exercice, il vous faut d'abord avoir vu *OCI04_Boucles*

Ce Notebook est a été crée par David Da SILVA - 2020

Source: Tangente HS 37 & EPFL ICC - O. Lévêque