1.25 Les bases de la programmation avec Python

**Ahmed Ammar (ahmed.ammar@fst.utm.tn)**

Institut Préparatoire aux Études Scientifiques et Techniques, Université de Carthage.

Oct 7, 2020

# C’est quoi Python?

Le langage de programmation Python () a été créé en 1989 par Guido van Rossum, aux Pays-Bas. La première version publique de ce langage a été publiée en 1991.

Ce langage de programmation présente de nombreuses caractéristiques intéressantes :

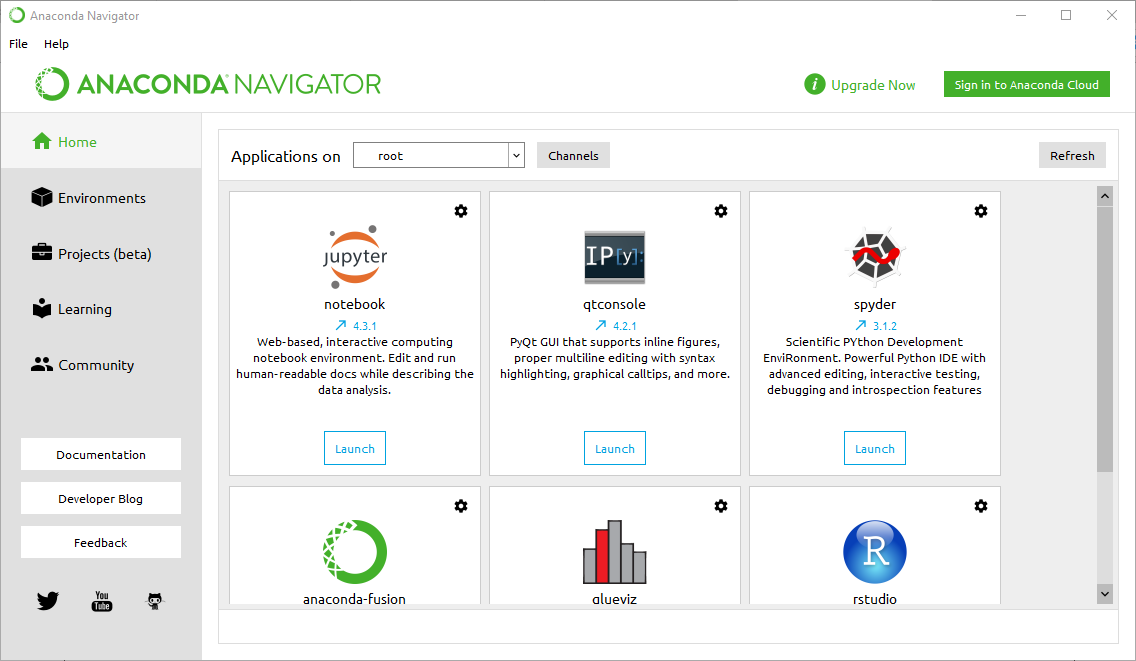
* Il est **multiplateforme**. C’est-à-dire qu’il fonctionne sur de nombreux systèmes d’exploitation : Windows, Mac OS X, Linux, Android, iOS, depuis les mini-ordinateurs Raspberry Pi jusqu’aux supercalculateurs.
* Il est **gratuit**. Vous pouvez l’installer sur autant d’ordinateurs que vous voulez (même sur votre téléphone!).
* C’est un **langage de haut niveau**. Il demande relativement peu de connaissance sur le fonctionnement d’un ordinateur pour être utilisé.
* C’est un **langage interprété**. Un script Python n’a pas besoin d’être compilé pour être exécuté, contrairement à des langages comme le C ou le C++.
* Il est **orienté objet**. C’est-à-dire qu’il est possible de concevoir en Python des entités qui miment celles du monde réel (une cellule, une protéine, un atome, etc.) avec un certain nombre de règles de fonctionnement et d’interactions.
* Il est **relativement simple** à prendre en main.
* Enfin, il est très utilisé en industrie technologique et plus généralement en data science et intelligence artificielle.

# Installation d’un environnement Python scientifique

## Qu’est ce que Anaconda ?

Anaconda () est une distribution Python. A son installation, Anaconda installera Python ainsi qu’une multitude de packages (voir [liste de packages anaconda]({https://docs.anaconda.com/anaconda/packages/pkg-docs#python-3-6})). Cela nous évite de nous ruer dans les problèmes d’incompatibilités entre les différents packages.

Finalement, Anaconda propose un outil de gestion de packages appelé [conda]({https://conda.io/docs/}). Ce dernier permettra de mettre à jour et installer facilement les librairies dont on aura besoin pour nos développements.



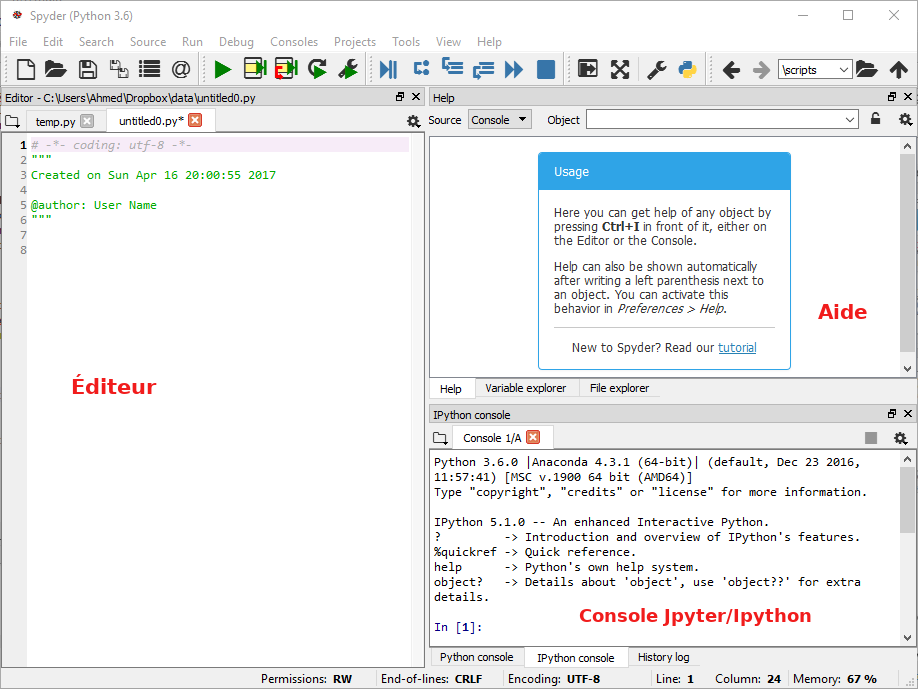
Interface graphique du navigateur Anaconda sur Windows

* Nous demandons à tous les étudiants de télécharger Anaconda. Pour cela, il faut télécharger un installeur à partir de , correspondant à votre système d’exploitation (Windows, Mac OS X, Linux). Il faut choisir entre 32 bits ou 64 bits (pour la version *Python 3*) selon que votre système d’exploitation est 32 bits ou 64 bits.
* Anaconda installe plusieurs exécutables pour développer en Python dans le répertoire *anaconda3/bin* (voir dans votre dossier personnel), sans toujours créer des raccourcis sur le bureau ou dans un menu.

## L’environnement Spyder

Pour le développement de programmes en langage Python, des applications spéciales appelées IDE (Integrated Development Environment) peuvent être utilisées. **Spyder** (Scientific PYthon Development EnviRonment) est un environnement de développement interactif gratuit inclus avec Anaconda. Il comprend des fonctionnalités d’édition, de test interactif, de débogage et d’introspection.

Après avoir installé Anaconda, vous pouvez démarrer Spyder sur macOS, Linux ou Windows en ouvrant une fenêtre de terminal (Ubuntu/macOS) ou d’invite de commande (Windows) et en exécutant la commande spyder.



Spyder sous Windows.

# Premier programme en Python : "Hello World!"

C’est devenu une tradition que lorsque vous apprenez un nouveau langage de programmation, vous démarrez avec un programme permettant à l’ordinateur d’imprimer le message *"Hello World!"*.

In [1]: print("Hello World!")  
Hello World!

Félicitation! tout à l’heure vous avez fait votre ordinateur saluer le monde en anglais! La fonction print() est utilisée pour imprimer l’instruction entre les parenthèses. De plus, l’utilisation de guillemets simples print('Hello World!') affichera le même résultat. Le délimiteur de début et de fin doit être le même.

In [2]: print('Hello World!')  
Hello World!

# Commentaires

Au fur et à mesure que vos programmes deviennent plus grands et plus compliqués, ils deviennent plus difficiles à lire et à regarder un morceau de code et à comprendre ce qu’il fait ou pourquoi. Pour cette raison, il est conseillé d’ajouter des notes à vos programmes pour expliquer en langage naturel ce qu’il fait. Ces notes s’appellent des commentaires et commencent par le symbole #.

Voyez ce qui se passe lorsque nous ajoutons un commentaire au code précédent:

In [3]: print('Hello World!') # Ceci est mon premier commentaire  
Hello World!

Rien ne change dans la sortie? Oui, et c’est très normal, l’interpréteur Python ignore cette ligne et ne renvoie rien. La raison en est que les commentaires sont écrits pour les humains, pour comprendre leurs codes, et non pour les machines.

# Expressions

## Opérations arithmétiques

L’interpréteur Python agit comme une simple calculatrice : vous pouvez y taper une expression et l’interpréteur restituera la valeur. La syntaxe d’expression est simple: les opérateurs +, -, \* et / fonctionnent comme dans la plupart des autres langages (par exemple, Pascal ou C); les parenthèses (()) peuvent être utilisées pour le regroupement. Par exemple:

In [4]: 5+3  
Out[4]: 8  
In [5]: 2 - 9 # les espaces sont optionnels  
Out[5]: -7  
In [6]: 7 + 3 \* 4 #la hiérarchie des opérations mathématique  
Out[6]: 19  
In [7]: (7 + 3) \* 4 # est-elle respectées?  
Out[7]: 40  
  
# en python3 la division retourne toujours un nombre en virgule flottante  
In [8]: 20 / 3  
Out[8]: 6.666666666666667  
In [9]: 7 // 2 # une division entière  
Out[9]: 3

On peut noter l’existence de l’opérateur % (appelé opérateur modulo). Cet opérateur fournit le reste de la division entière d’un nombre par un autre. Par exemple :

In [10]: 7 % 2 # donne le reste de la division  
Out[10]: 1  
In [11]: 6 % 2  
Out[11]: 0

Les exposants peuvent être calculés à l’aide de doubles astérisques \*\*.

In [12]: 3\*\*2  
Out[12]: 9

Les puissances de dix peuvent être calculées comme suit:

In [13]: 3 \* 2e3 # vaut 3 \* 2000  
Out[13]: 6000.0

## Opérateurs relationnels

|  |  |  |
| --- | --- | --- |
| < | strictement inférieur |  |
| <= | inférieur ou égal |  |
| > | strictement supérieur |  |
| >= | supérieur ou égal |  |
| == | égal | Attention : deux signes == |
| != | différent |  |

In [17]: b = 10  
 ...: b > 8  
Out[17]: True  
  
In [18]: b == 5  
Out[18]: False  
  
In [19]: b != 5  
Out[19]: True  
  
In [20]: 0 <= b <= 20  
Out[20]: True

## Opérateurs logiques

In [21]: note = 13.0  
  
In [22]: mention\_ab = note >= 12.0 and note < 14.0  
  
In [23]: # ou bien : mention\_ab = 12.0 <= note < 14.0  
  
In [24]: mention\_ab  
Out[24]: True

In [25]: not mention\_ab  
Out[25]: False  
  
In [26]: note == 20.0 or note == 0.0  
Out[26]: False

L’opérateur in s’utilise avec des chaînes (type str) ou des listes (type list).

Pour une chaînes:

In [30]: chaine = 'Bonsoir'  
 ...: #la sous-chaîne 'soir' fait-elle partie de la chaîne 'Bonsoir' ?  
  
In [31]: resultat = 'soir' in chaine  
 ...: resultat  
Out[31]: True

Pour une liste:

In [32]: maliste = [4, 8, 15]  
 ...: #le nombre entier 9 est-il dans la liste ?  
  
In [33]: 9 in maliste  
Out[33]: False  
  
In [34]: 8 in maliste  
Out[34]: True  
  
In [35]: 14 not in maliste  
Out[35]: True

# Variables et affectation

Dans presque tous les programmes Python que vous allez écrire, vous aurez des variables. Les variables agissent comme des espaces réservés pour les données. Ils peuvent aider à court terme, ainsi qu’à la logique, les variables pouvant changer, d’où leur nom. C’est beaucoup plus facile en Python car aucune déclaration de variables n’est requise. Les noms de variable (ou tout autre objet Python tel que fonction, classe, module, etc.) commencent par une lettre majuscule ou minuscule (A-Z ou a-z). Ils sont sensibles à la casse (VAR1 et var1 sont deux variables distinctes). Depuis Python, vous pouvez utiliser n’importe quel caractère Unicode, il est préférable d’ignorer les caractères ASCII (donc pas de caractères accentués).

Si une variable est nécessaire, pensez à un nom et commencez à l’utiliser comme une variable, comme dans l’exemple ci-dessous:

Pour calculer l’aire d’un rectangle par exemple: largeur x hauteur:

In [15]: largeur = 25  
 ...: hauteur = 40  
 ...: largeur # essayer d'accéder à la valeur de la variable largeur  
Out[15]: 25

on peut également utiliser la fonction print() pour afficher la valeur de la variable largeur

In [16]: print(largeur)  
25

Le produit de ces deux variables donne l’aire du rectangle:

In [17]: largeur \* hauteur # donne l'aire du rectangle  
Out[17]: 1000

Notez ici que le signe égal (=) dans l’affectation ne doit pas être considéré comme **"est égal à"**. Il doit être **"lu"** ou interprété comme **"est définie par"**, ce qui signifie dans notre exemple:

La variable largeur est définie par la valeur 25 et la variable hauteur est définie par la valeur 40.

Si une variable n’est pas *définie* (assignée à une valeur), son utilisation vous donnera une erreur:

In [18]: aire # essayer d'accéder à une variable non définie  
-----------------------------------------------------------------------  
NameError Traceback (most recent call last)  
<ipython-input-18-1b03529c1ce5> in <module>()  
----> 1 aire # essayer d'accéder à une variable non définie  
  
NameError: name 'aire' is not defined

Laissez-nous résoudre ce problème informatique (ou **bug** tout simplement)!. En d’autres termes, assignons la variable aire à sa valeur.

In [19]: aire = largeur \* hauteur  
 ...: aire # et voila!  
Out[19]: 1000

# Noms de variables réservés (keywords)

Certains noms de variables ne sont pas disponibles, ils sont réservés à python lui-même. Les mots-clés suivants (que vous pouvez afficher dans l’interpréteur avec la commande help("keywords")) sont réservés et ne peuvent pas être utilisés pour définir vos propres identifiants (variables, noms de fonctions, classes, etc.).

In [20]: help("keywords")  
  
Here is a list of the Python keywords. Enter any keyword to get more help.  
  
False def if raise  
None del import return  
True elif in try  
and else is while  
as except lambda with  
assert finally nonlocal yield  
break for not  
class from or  
continue global pass  
  
# par exemple pour éviter d'écraser le nom réservé lambda  
In [22]: lambda\_ = 630e-9  
 ...: lambda\_  
Out[22]: 6.3e-07

# Types simples

Les types utilisés dans Python sont: integers, long integers, floats (double prec.), complexes, strings, booleans. La fonction type() donne le type de son argument

## Le type int (integer : nombres entiers)

Pour affecter (on peut dire aussi assigner) la valeur 20 à la variable nommée age :

age = 20

La fonction print() affiche la valeur de la variable :

In [24]: print(age)  
20

La fonction type() retourne le type de la variable :

type(age)  
Out[25]: int

## Le type float (nombres en virgule flottante)

b = 17.0 # le séparateur décimal est un point (et non une virgule)  
b  
Out[26]: 17.0  
In [27]: type(b)  
Out[27]: float  
In [28]: c = 14.0/3.0  
 ...: c  
Out[28]: 4.666666666666667

Notation scientifique :

In [29]: a = -1.784892e4  
 ...: a  
Out[29]: -17848.92

## Le type complexe

Python possède par défaut un type pour manipuler les nombres complexes. La partie imaginaire est indiquée grâce à la lettre « j » ou « J ». La lettre mathématique utilisée habituellement, le « i », n’est pas utilisée en Python car la variable i est souvent utilisée dans les boucles.

In [37]: a = 2 + 3j  
 ...: type(a)  
Out[37]: complex  
In [38]: a  
Out[38]: (2+3j)

In [39]: b = 1 + j  
--------------------------------------------------------------  
NameError Traceback (most recent call last)  
<ipython-input-39-0f22d953f29e> in <module>()  
----> 1 b = 1 + j  
  
NameError: name 'j' is not defined

Dans ce cas, on doit écrire la variable b comme suit:

In [41]: b = 1 + 1j  
 ...: b  
Out[41]: (1+1j)

sinon Python va considérer j comme variable non définie.

On peut faire l’addition des variables complexes:

In [42]: a + b  
Out[42]: (3+4j)

## Le type bool (booléen)

Deux valeurs sont possibles : True et False

In [16]: choix = True # NOTE: "True" différent de "true"  
 ...: type(choix)  
Out[16]: bool

# Types composés

## Le type str (string : chaîne de caractères)

In [43]: nom = 'Tounsi' # entre apostrophes  
 ...: nom  
Out[43]: 'Tounsi'  
In [44]: type(nom)  
Out[44]: str  
In [45]: prenom = "Ali" # on peut aussi utiliser les guillemets  
 ...: prenom  
Out[45]: 'Ali'  
In [46]: print(nom, prenom) # ne pas oublier la virgule  
Tounsi Ali

La concaténation désigne la mise bout à bout de plusieurs chaînes de caractères. La concaténation utilise l’opérateur +:

In [47]: chaine = nom + prenom # concaténation de deux chaînes de caractères  
 ...: chaine  
Out[47]: 'TounsiAli'

Vous voyez dans cet exemple que le nom et le prénom sont collé. Pour ajouter une espace entre ces deux chaînes de caractères:

In [48]: chaine = prenom + ' ' + nom  
 ...: chaine # et voila  
Out[48]: 'Ali Tounsi'

On peut modifier/ajouter une nouvelle chaîne à notre variable chaine par:

In [49]: chaine = chaine + ' 22 ans' # en plus court : chaine += ' 22 ans'  
 ...: chaine  
Out[49]: 'Ali Tounsi 22 ans'

La fonction len() renvoie la longueur (*length*) de la chaîne de caractères :

In [53]: print(nom)  
 ...: len(nom)  
Tounsi  
Out[53]: 6

#### Indexage et slicing :

+---+---+---+---+---+---+  
|------------------------|  
 | T | o | u | n | s | i |  
 +---+---+---+---+---+---+  
 |------------------------|  
 0 1 2 3 4 5 6  
 --->  
-6 -5 -4 -3 -2 -1  
 <----

In [55]: nom[0] # premier caractère (indice 0)  
Out[55]: 'T'  
  
In [56]: nom[:] # toute la chaine  
Out[56]: 'Tounsi'  
  
In [57]: nom[1] # deuxième caractère (indice 1)  
Out[57]: 'o'  
  
In [58]: nom[1:4] # slicing  
Out[58]: 'oun'  
  
In [59]: nom[2:] # slicing  
Out[59]: 'unsi'  
  
In [60]: nom[-1] # dernier caractère (indice -1)  
Out[60]: 'i'  
  
In [61]: nom[-3:] # slicing  
Out[61]: 'nsi'

On ne peut pas mélanger le type str et type int.

Soit par exemple:

In [63]: chaine = '22'  
 ...: annee\_naissance = 2018 - chaine  
----------------------------------------------------------  
TypeError Traceback (most recent call last)  
<ipython-input-63-8607078f78d2> in <module>()  
 1 chaine = '22'  
----> 2 annee\_naissance = 2018 - chaine  
  
TypeError: unsupported operand type(s) for -: 'int' and 'str'

Pour corriger cette erreur, la fonction int() permet de convertir un type str en type int:

In [64]: nombre = int(chaine)  
 ...: type(nombre) # et voila!  
Out[64]: int

Maintenant on peut trouver annee\_naissance sans aucun problème:

In [65]: annee\_naissance = 2018 - nombre  
 ...: annee\_naissance  
Out[65]: 1996

#### Interaction avec l’utilisateur (la fonction input())

La fonction input() lance une case pour saisir une chaîne de caractères.

In [66]: prenom = input('Entrez votre prénom : ')  
 ...: age = input('Entrez votre age : ')  
  
Entrez votre prénom : Foulen  
  
Entrez votre age : 25

#### Formatage des chaînes

Un problème qui se retrouve souvent, c’est le besoin d’afficher un message qui contient des valeurs de variables.

Soit le message: Bonjour Mr/Mme prenom, votre age est age.

La solution est d’utiliser la méthode format() de l’objet chaîne str() et le {} pour définir la valeur à afficher.

print(" Bonjour Mr/Mme {}, votre age est {}.".format(prenom, age))

#### Le type list (liste)

Une liste est une structure de données.

Le premier élément d’une liste possède l’indice (l’index) 0.

Dans une liste, on peut avoir des éléments de plusieurs types.

In [1]: info = ['Tunisie', 'Afrique', 3000, 36.8, 10.08]  
  
In [2]: type(info)  
Out[2]: list

La liste info contient 5 éléments de types str, str, int, float et float

In [3]: info  
Out[3]: ['Tunisie', 'Afrique', 3000, 36.8, 10.08]  
  
In [4]: print('Pays : ', info[0]) # premier élément (indice 0)  
Pays : Tunisie  
  
In [5]: print('Age : ', info[2]) # le troisième élément a l'indice 2  
Age : 3000  
  
In [6]: print('Latitude : ', info[3]) # le quatrième élément a l'indice 3  
Latitude : 36.8

La fonction range() crée une liste d’entiers régulièrement espacés :

In [7]: maliste = range(10) # équivalent à range(0,10,1)  
 ...: type(maliste)  
Out[7]: range

Pour convertir une range en une liste, on applique la fonction list() à notre variable:

In [8]: list(maliste) # pour convertir range en une liste  
Out[8]: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

On peut spécifier le début, la fin et l’intervalle d’une range:

In [9]: maliste = range(1,10,2) # range(début,fin non comprise,intervalle)  
 ...: list(maliste)  
Out[9]: [1, 3, 5, 7, 9]  
  
In [10]: maliste[2] # le troisième élément a l'indice 2  
Out[10]: 5

On peut créer une liste de listes, qui s’apparente à un tableau à 2 dimensions (ligne, colonne) :

0 1 2  
10 11 12  
20 21 22

In [11]: maliste = [[0, 1, 2], [10, 11, 12], [20, 21, 22]]  
 ...: maliste[0]  
Out[11]: [0, 1, 2]  
  
In [12]: maliste[0][0]  
Out[12]: 0  
  
In [13]: maliste[2][1] # élément à la troisième ligne et deuxième colonne  
Out[13]: 21  
  
In [14]: maliste[2][1] = 78 # nouvelle affectation  
  
In [15]: maliste  
Out[15]: [[0, 1, 2], [10, 11, 12], [20, 78, 22]]

# Les conditions

## L’instruction if

En programmation, nous avons toujours besoin de la notion de condition pour permettre à un programme de s’adapter à différents cas de figure.

if expression: # ne pas oublier le signe de ponctuation ':'  
 "bloc d'instructions" # attention à l'indentation (1 Tab ou 4 \* Espaces)  
# suite du programme

* Si l’expression est vraie (True) alors le bloc d’instructions est exécuté.
* Si l’expression est fausse (False) on passe directement à la suite du programme.

#### Exemple 1: Note sur 20.

Dans cet exemple nous allons tester si la note entrée par l’utilisateur. Si la note est > 10 on doit recevoir le message: "J’ai la moyenne" sinon il va rien faire.

chaine = input("Note sur 20 : ")  
note = float(chaine)  
if note >= 10.0:  
 # ce bloc est exécuté si l'expression (note >= 10.0) est vraie  
 print("J'ai la moyenne")  
  
# suite du programme  
print("Fin du programme")

* Les blocs de code sont délimités par l’indentation.
* L’indentation est obligatoire dans les scripts.

## L’instruction else

Une instruction else est toujours associée à une instruction if.

if expression:  
 "bloc d'instructions 1" # attention à l'indentation (1 Tab ou 4 \* Espaces)  
else: # else est au même niveau que if  
 "bloc d'instructions 2" # attention à l'indentation  
# suite du programme

* Si l’expression est vraie (True) alors le bloc d’instructions 1 est exécuté.
* Si l’expression est fausse (False) alors c’est le bloc d’instructions 2 qui est exécuté.

#### Exemple 2 : moyenne.

Dans cet exemple nous allons tester si la note entrée par l’utilisateur. Si la note est > 10 on doit recevoir le message: "J’ai la moyenne" sinon il va afficher "C’est en dessous de la moyenne".

chaine = input("Note sur 20 : ")  
note = float(chaine)  
if note >= 10.0:  
 # ce bloc est exécuté si l'expression (note >= 10.0) est vraie  
 print("J'ai la moyenne")  
else:  
 # ce bloc est exécuté si l'expression (note >= 10.0) est fausse  
 print("C'est en dessous de la moyenne")  
print("Fin du programme")

Pour traiter le cas des notes invalides ( ou ), on peut imbriquer des instructions conditionnelles :

chaine = input("Note sur 20 : ")  
note = float(chaine)  
if note > 20.0 or note < 0.0:  
 # ce bloc est exécuté si l'expression (note > 20.0 or note < 0.0) est vraie  
 print("Note invalide !")  
else:  
 # ce bloc est exécuté si l'expression (note > 20.0 or note < 0.0) est fausse  
 if note >= 10.0:  
 # ce bloc est exécuté si l'expression (note >= 10.0) est vraie  
 print("J'ai la moyenne")  
 else:  
 # ce bloc est exécuté si l'expression (note >= 10.0) est fausse  
 print("C'est en dessous de la moyenne")  
print("Fin du programme")

Ou bien encore:

chaine = input("Note sur 20 : ")  
note = float(chaine)  
if note > 20.0 or note < 0.0:  
 print("Note invalide !")  
else:  
 if note >= 10.0:  
 print("J'ai la moyenne")  
 if note == 20.0:  
 # ce bloc est exécuté si l'expression (note == 20.0) est vraie  
 print("C'est même excellent !")  
 else:  
 print("C'est en dessous de la moyenne")  
 if note == 0.0:  
 # ce bloc est exécuté si l'expression (note == 0.0) est vraie  
 print("... lamentable !")  
print("Fin du programme")

## L’instruction elif

Une instruction elif (contraction de **else if**) est toujours associée à une instruction if.

if expression 1:  
 "bloc d'instructions 1"  
elif expression 2:  
 "bloc d'instructions 2"  
elif expression 3:  
 "bloc d'instructions 3" # ici deux instructions elif, mais il n'y a pas de limitation  
else:  
 "bloc d'instructions 4"  
# suite du programme

* Si l’expression 1 est vraie alors le bloc d’instructions 1 est exécuté, et on passe à la suite du programme.
* Si l’expression 1 est fausse alors on teste l’expression 2 :
* si l’expression 2 est vraie on exécute le bloc d’instructions 2, et on passe à la suite du programme.
* si l’expression 2 est fausse alors on teste l’expression 3, etc.

Le bloc d’instructions 4 est donc exécuté si toutes les expressions sont fausses (c’est le bloc "par défaut").

Parfois il n’y a rien à faire. Dans ce cas, on peut omettre l’instruction else :

if expression 1:  
 "bloc d'instructions 1"  
elif expression 2:  
 "bloc d'instructions 2"  
elif expression 3:  
 "bloc d'instructions 3"  
# suite du programme

L’instruction elif évite souvent l’utilisation de conditions imbriquées (et souvent compliquées).

#### Exemple 3 : moyenne-bis.

On peut tester plusieurs possibilités avec une syntaxe beaucoup plus propre avec les instructions if-elif-else:

note = float(input("Note sur 20 : "))  
if note == 0.0:  
 print("C'est en dessous de la moyenne")  
 print("... lamentable!")  
elif note == 20.0:  
 print("J'ai la moyenne")  
 print("C'est même excellent !")  
elif 0 < note < 10: # ou bien : elif 0.0 < note < 10.0:  
 print("C'est en dessous de la moyenne")  
elif note >= 10.0 and note < 20.0: # ou bien : elif 10.0 <= note < 20.0:  
 print("J'ai la moyenne")  
else:  
 print("Note invalide !")  
print("Fin du programme")

## Exercise : Condition sur le jour de travail

Si aujourd’hui est lundi alors je dois aller travailler, mais si c’est dimanche alors je peux rester faire la grasse matinée. Pour pouvoir accomplir ce genre de choses en Python, on fait appel à des expressions booléennes qui ne peuvent revêtir que deux possibilités - ou bien l’expression est vraie ou bien elle est fausse - et à la syntaxe if condition: qui permet de contrôler le flux du programme grâce à ces valeurs booléennes.

day\_week = ["Lundi", "Mardi", "Mercredi", "Jeudi", "Vendredi",  
 "Samedi", "Dimanche"]  
today = input("Aujourd'hui est: ")  
  
if \*condition vraie\*: # Quelle est la condition vraie dans ce cas?  
 print("Je dors le matin!")  
else:  
 print("Je travail le matin!")  
  
print("Fin du programme")

#### Hint.

Dans la **condition vraie**, utilisez l’opérateur logique in pour tester les éléments de la liste day\_week.

#### Solution.

day\_week = ["Lundi", "Mardi", "Mercredi", "Jeudi", "Vendredi", "Samedi", "Dimanche"]  
today = input("Aujourd'hui est: ")  
if today in day\_week:  
 if today == day\_week[-1]: # la condition vraie  
 print("Je dors le matin!")  
 else:  
 print("Je travail le matin!")  
  
print("Fin du programme")

# Les boucles

## L’instruction while

while expression: # ne pas oublier le signe de ponctuation ':'  
 "bloc d'instructions" # attention à l'indentation (1 Tab ou 4 \* Espaces)  
# suite du programme

* Si l’expression est vraie (True) le bloc d’instructions est exécuté, puis l’expression est à nouveau évaluée.
* Le cycle continue jusqu’à ce que l’expression soit fausse (False) : on passe alors à la suite du programme.

#### Exemple 1 : un script qui compte de 1 à 4.

# initialisation de la variable de comptage  
compteur = 0  
while compteur < 5:  
 # ce bloc est exécuté tant que la condition (compteur < 5) est vraie  
 print(compteur)  
 compteur += 1 # incrémentation du compteur, compteur = compteur + 1  
print(compteur)  
print("Fin de la boucle")

#### Exemple 2 : Table de multiplication par 8.

compteur = 1 # initialisation de la variable de comptage  
while compteur <= 10:  
 # ce bloc est exécuté tant que la condition (compteur <= 10) est vraie  
 print(compteur, '\* 8 =', compteur\*8)  
 compteur += 1 # incrémentation du compteur, compteur = compteur + 1  
print("Et voilà !")

#### Exemple 3 : Affichage de l’heure courante.

import time # importation du module time  
quitter = 'n' # initialisation  
while quitter != 'o':  
 # ce bloc est exécuté tant que la condition est vraie  
 # strftime() est une fonction du module time  
 print('Heure courante ', time.strftime('%H:%M:%S'))  
 quitter = input("Voulez-vous quitter le programme (o/n) ? ")  
print("A bientôt")

## L’instruction for

for élément in séquence : # ne pas oublier le signe de ponctuation ':'  
 "bloc d'instructions" # attention à l'indentation (1 Tab ou 4 \* Espaces)  
# suite du programme

Les éléments de la séquence sont issus d’une chaîne de caractères ou bien d’une liste.

#### Exemple 1 : séquence de caractères.

chaine = 'Bonsoir'  
for lettre in chaine: # lettre est la variable d'itération  
 print(lettre)  
print("Fin de la boucle")

La variable lettre est initialisée avec le premier élément de la séquence (’B’). Le bloc d’instructions est alors exécuté.

Puis la variable lettre est mise à jour avec le second élément de la séquence (’o’) et le bloc d’instructions à nouveau exécuté...

Le bloc d’instructions est exécuté une dernière fois lorsqu’on arrive au dernier élément de la séquence (’r’).

#### Fonction range().

L’association avec la fonction range() est très utile pour créer des séquences automatiques de nombres entiers :

for i in range(1, 5):  
 print(i)  
print("Fin de la boucle")

#### Exemple 2 : Table de multiplication.

La création d’une table de multiplication paraît plus simple avec une boucle for qu’avec une boucle while :

for compteur in range(1,11):  
 print(compteur, '\* 8 =', compteur\*8)  
print("Et voilà !")