

# Éléments pour le langage de programmation Python™

Laurent Astier – `laurent.astier@ac-limoges.fr`

## Table des matières

<b>1</b>	<b>L'environnement de programmation pour Python</b>	<b>1</b>
1.1	Découverte . . . . .	1
1.2	Applications . . . . .	2
1.3	Questions . . . . .	2
1.4	À retenir . . . . .	2
<b>2</b>	<b>Les objets classiques en Python</b>	<b>2</b>
2.1	Les types des variables . . . . .	2
2.2	Cas des variables numériques . . . . .	3
2.3	Cas de la fonction <code>print</code> . . . . .	3
2.4	Cas des listes . . . . .	3
2.5	À retenir . . . . .	3
<b>3</b>	<b>Quelques éléments de programmation en Python</b>	<b>4</b>
3.1	Les conditions en Python . . . . .	4
3.1.1	Cas simple . . . . .	4
3.1.2	Multi-conditions . . . . .	4
3.1.3	Que mettre dans les tests de conditions ? . . . . .	5
3.1.4	Application : <code>reaction.py</code> . . . . .	5
3.2	Réaliser des boucles <code>for</code> . . . . .	6
3.2.1	Boucle <code>for</code> sur des objets . . . . .	6
3.2.2	Boucle <code>for</code> et fonction <code>range()</code> . . . . .	6
3.3	Réaliser des boucles <code>while</code> . . . . .	7
3.3.1	Fonctionnement . . . . .	7
3.3.2	Application . . . . .	7
3.4	Les fonctions en Python . . . . .	7
3.4.1	Première méthode : un peu longue . . . . .	8
3.4.2	Seconde méthode : avec une fonction . . . . .	8

## 1 L'environnement de programmation pour Python

### 1.1 Découverte

Selon le logiciel que vous utilisez pour Python, vous disposez à l'écran d'un *éditeur* où vous pouvez taper ou copier-coller des programmes assez longs, et d'une *console* qui permet :

- d'avoir les résultats de compilation de vos programmes
- de rentrer des commandes courtes (sur quelques lignes au maximum)

On repère la *console* par la présence des 3 chevrons : `>>>` .

## 1.2 Applications

Rendez-vous à l'adresse

<https://github.com/formationPythonPC/Agreg-Interne/blob/master/programme1.py>, cliquez sur Raw (brouillon) et copiez-coller le code disponible sur votre éditeur. Compilez-le pour en voir le résultat :

- Une courbe s'affiche
- sur la console, on voit : `>>> 0.15 -- 1.4534528761455452`

## 1.3 Questions

1. À quoi servent les lignes commençant par `#` ?
2. L'importation des bibliothèques est-elle obligatoire (lignes 2 et 3) ?
3. Que sont les « variables » (ligne 4) ?
4. En vous servant de la console, expliquez le rôle de la fonction `np.arange(...)` (ligne 7).
5. À quoi sert la fonction `print(...)` (ligne 11) ?
6. À quoi sert la fonction `plt.plot(...)` ?
7. Décrire de même le rôle des lignes suivantes du programme.
8. En vous aidant de la question 4, comment comprenez-vous `T[2]` dans la fonction `print` ligne 11 ?
9. Placez-vous dans la console ; vous pouvez y réaliser des opérations à partir des tableaux `T` ou (et) `A`.

## 1.4 À retenir

- l'environnement éditeur – console
- les bibliothèques `matplotlib` et `numpy`
- les commentaires `#`
- la notion de *variable*
- la fonction `print`
- la fonction `plt.plot`
- la numérotation de Python : à partir de 0

# 2 Les objets classiques en Python

## 2.1 Les types des variables

En Python, les objets sont « typés », ils appartiennent à des catégories :



CODE PYTHON™

```
(
)( >>> var1 = 3 # les entiers sont appelés "integers" en anglais
)( >>> var2 = 5.2 # les décimaux sont appelés "flottants" par Python
)( >>> ch = "chaîne de caractères" # une suite de caractères est une chaîne de
)( caractères en Python
)(
)( >>> L = [0,1,2,3,4,5] # ceci est une liste qui possède 6 éléments
)(
)( >>> var5 = True # ceci est un booléen : 2 valeurs possibles, True ou False
```

## 2.2 Cas des variables numériques

On peut sans problème changer le contenu de la "boîte" :



CODE PYTHON™

```
(
)( >>> a = 10
)( >>> a = a + 1 # ou a+=1
)( >>> print(a)
)( 11
)( >>> a = a**2 ; print(a)
)( 121
```

## 2.3 Cas de la fonction print

La fonction `print` peut retourner la valeur de variables, des chaînes de caractères ou les deux :



CODE PYTHON™

```
(
)( >>> v = 2
)( >>> print("la valeur est v = ", v, " m/s")
)( la valeur est v = 2 m/s
```

## 2.4 Cas des listes



CODE PYTHON™

```
(
)( >>> L = [] # création liste vide
)( >>> M = [2,3,4]
)( >>> L.append(1) ; print(L) # ajout d'éléments
)( >>> N = L + M ; print(N)
)( [1,2,3,4] # cette opération est une concaténation
)( >>> L = 3 * L ; print(L)
)( [1,1,1] # le calcul n'est pas "naturel"
)( >>> print(M[0]) # premier élément de la liste
)( >>> print(M[-1]) # dernier élément de la liste
```

REMARQUES :

- une liste peut accueillir tout type d'éléments, des entiers, des flottants, des chaînes de caractères, même des listes.
- C'est ce qui explique que les opérations sur les listes ne soient pas "naturelles" (voir ci-contre).
- → À votre avis, comment réussir à effectuer des opérations "naturelles" sur des listes ?

## 2.5 À retenir

- le contenu d'une variable peut être modifié (d'où le nom *variable*)
- le signe "=" en programmation ne signifie pas du tout "égal à" ; c'est un signe d'affectation :  
`a = 3` : on affecte la valeur "3" à la variable "a"  
`a = a + 1` : on affecte la valeur a+1 (4) à la variable "a"
- la fonction `print` peut retourner une chaîne de caractères passée en argument, une variable passée en argument et même les deux)
- la fonction `append` permet de rajouter un élément à la fin d'une liste
- il n'y a pas d'opération "naturelle" sur les listes (disons plutôt *linéaire*)
- pour effectuer des opérations "naturelles", on utilisera plutôt des *tableaux* de la bibliothèque `numpy`
- les tableaux `numpy` ne doivent donc contenir que des nombres
- À titre d'exemple, comparer les résultats des mêmes opérations sur une liste ou sur un tableau :



CODE PYTHON™

```

)(
)( >>> L = [1,2,3]
)(
)( >>> print(2*L)
)( [1,2,3,1,2,3] # opération non "naturelle" : concaténation
)(
)( >>> import numpy as np
)(
)( >>> T = np.array(L) # on transforme la liste en tableau
)(
)( >>> print(2*T)
)( [2,4,6] # opération "naturelle"

```

### 3 Quelques éléments de programmation en Python

#### 3.1 Les conditions en Python

##### 3.1.1 Cas simple



CODE PYTHON™

```

)(
)( if condition : # si la condition est vérifiée - notez la tabulation en dessous
)(     xxxxxxxxxxx # on fait xxxxxxxxxxx
)(
)( else : # sinon
)(     yyyyyyyyyy # on fait yyyyyyyyyy
)(

```

##### 3.1.2 Multi-conditions



CODE PYTHON™

```

)(
)( if condition1 : # si la condition1 est vérifiée - notez la tabulation en dessous
)(     xxxxxxxxxxx # on fait xxxxxxxxxxx
)(
)( elif condition2 : # sinon si condition2
)(     yyyyyyyyyy # on fait yyyyyyyyyy
)(
)( elif condition3 : # sinon si condition3
)(     wwwwwwwwww # on fait wwwwwwwwww
)(
)( ...
)(
)( else : # sinon
)(     zzzzzzzzzz # on fait zzzzzzzzzz
)(

```

### 3.1.3 Que mettre dans les tests de conditions ?

condition	signification	condition	signification
<code>==</code>	est égal à	<code>&lt;</code>	est inférieur à
<code>!=</code>	est différent de	<code>&gt;=</code>	est supérieur ou égal à
<code>&gt;</code>	est supérieur à	<code>&lt;=</code>	est inférieur ou égal à

On notera que la condition d'égalité est représentée par le double signe égal, **à ne pas confondre donc avec le symbole d'affectation**. Voir la remarque

On peut associer plusieurs conditions ensemble :

écriture	signification
<code>condition1 and condition2</code>	condition1 ET condition2
<code>condition1 or condition2</code>	condition1 OU condition2
...	...

### 3.1.4 Application : `reaction.py`

On considère la réaction totale entre l'hydrogène sulfureux ( $\text{H}_2\text{S}$ ) et le dioxyde de soufre ( $\text{SO}_2$ ) qui produit du soufre et de l'eau et modélisée par l'équation :  $2 \text{H}_2\text{S} + \text{SO}_2 \longrightarrow 3 \text{S} + 2 \text{H}_2\text{O}$

L'exercice peut être rédigé comme un programme. Vous pourrez le nommer `reaction.py`.

Allez chercher le code présent à ce lien. Copiez-coller le dans votre programme. Ce code est à compléter.

1. Créez 4 **variables** qui contiendront les valeurs des coefficients stoechiométriques des différentes espèces. On pourra nommer ces variables `coeff_xxx`.
2. Créez 4 variables, `n0_H2S`, `n0_SO2`, `n0_S`, `n0_H2O` qui contiendront les quantités de matière initiales de chacune des quatre espèces.

Le programme devra demander à l'utilisateur les valeurs de ces 4 quantités.

AIDES :

- la fonction `input()` permet d'interagir avec l'utilisateur du programme.
  - Attention : la fonction `input()` renvoie une chaîne de caractères ; il faudra penser à la transformer en nombre "flottant"...
3. Écrire les formules permettant de calculer les avancements maximaux possibles pour les deux réactifs ; on les notera `xmax_H2S` et `xmax_SO2`. Vous utiliserez les variables définies précédemment.
  4. En utilisant une **condition** ou par une autre méthode (voir ci-dessous), trouvez l'expression de l'avancement maximal `xmax` de cette réaction en fonction des résultats de la question précédente. Faites afficher sur la console la valeur de cet avancement maximal.

AIDES :

- Si vous ne souhaitez pas traiter la question par la condition, Python dispose des fonctions `min()` et `max()` qui retournent le minimum et le maximum d'une liste passée en paramètre.
  - Pour afficher quelque chose en console, on utilise la fonction `print()`.
5. Calculez ensuite les valeurs finales de quantité de matière dans les 4 espèces, que vous appellerez `nF_H2S`, `nF_SO2`, `nF_S` et `nF_H2O` à partir des variables définies précédemment.

Votre programme devra ensuite afficher sur la console les valeurs finales des quantités de matière des 4 espèces.

↪ [aide à la résolution : reaction-aide.py](#)

► [lien vers la correction de cet exercice : reaction-correction.py](#) ◀

## 3.2 Réaliser des boucles **for**

### 3.2.1 Boucle **for** sur des objets

La boucle **for** permet de parcourir des objets ; dans notre cas, les objets parcourus seront souvent des listes ou la valeur des *indices* d'une liste.

On a réalisé une chronophotographie du mouvement d'une balle sur un banc à coussins d'air.

Le logiciel Avimeca retourne comme données :  $\Delta t = 0,04$  s (intervalle de temps entre 2 mesures) ainsi que les coordonnées des points atteints à ces intervalles de temps.

Allez chercher le code présent à ce lien. Vous pouvez le copier-coller et le compléter au fur et à mesure des questions.

1. D'après vous, à quoi servent les lignes 5 et 6 ? Compilez le programme pour le vérifier.
2. On se sert de ce programme pour calculer la vitesse au 4<sup>ème</sup> point.  
Les consignes en provenance de l'I.G. recommandent pour le calcul de la vitesse au  $i^{\text{ème}}$  point de la trajectoire :

$$v(i) = \frac{v(i+1) - v(i)}{t(i+1) - t(i)}$$

Le calcul réalisé à la ligne 9 est-il cohérent avec cette définition ? Justifiez précisément.

3. Globalement, à quoi servent les lignes 15 à 17 ?
4. Ligne 15, quelles sont les valeurs parcourues par la variable "i". Comprenez-vous pourquoi ? Comment vérifier votre hypothèse ?
5. Complétez la ligne 20 pour faire afficher la liste VX.

► [lien vers la correction de cet exercice : programme2-correction.py](#) ◀

### 3.2.2 Boucle **for** et fonction **range()**

La boucle **for** permet de parcourir les éléments d'un objet. Ici, l'objet est la fonction **range**. Tapez les lignes suivantes pour comprendre le fonctionnement de cet élément :



CODE PYTHON™

```
((
)) >>> for i in range(1,12,2) : # début inclus, fin exclue, pas
((      print(i)
```



CODE PYTHON™

```
((
)) >>> for i in range(3,10) :
((      print(i)
```




CODE PYTHON™

```
((
)) >>> for i in range(10) :
((      print(i)
```

### 3.3 Réaliser des boucles **while**

#### 3.3.1 Fonctionnement

Tentez de comprendre ce que font les lignes suivantes :



CODE PYTHON™

```

)<
)<  >>> i = 0 # initialisation d'une variable
)<  >>> while i <= 5 :
)<      print(i)
)<      i+=1 # ou i=i+1 - incrémentation de la variable
)<

```

Vous pouvez tester ces quelques lignes pour vérifier vos hypothèses.

REMARQUES SUR LA BOUCLE **WHILE** :

- Une erreur très classique est d'oublier l'incrémentation ; vous pouvez le tester, dans ce cas le programme rentre dans une boucle infinie puisque  $i$  reste à 0.  
Pensez à la combinaison Ctrl + C pour arrêter le programme.
- Il faut bien penser que la boucle **while condition** signifie : tant que la condition est **vraie**.  
Ainsi, l'écriture **While True** débute en fait une boucle infinie.

#### 3.3.2 Application

### 3.4 Les fonctions en Python

En programmation, les fonctions (ou procédures) sont des blocs qui permettent de gagner parfois énormément de temps, et rendent aussi le code général plus lisible.

On considère un rayon de lumière passant d'un milieu d'indice  $n_1$  à un milieu d'indice  $n_2$  avec un angle d'incidence  $i$ .

On souhaite connaître l'angle de réfraction  $r$  que fait le rayon dans le second milieu par rapport à la normale au dioptre.

On rappelle la loi de la réfraction :  $n_1 \times \sin i = n_2 \times \sin r$

REMARQUES :

- Par défaut, Python ne connaît pas les fonctions trigonométriques
- Il faudra donc les importer, depuis la bibliothèque **math**
- Pour Python, les angles sont en radians

### 3.4.1 Première méthode : un peu longue



#### CODE PYTHON™

```

)(
)( from math import sin, asin, degrees, radians
)( i = 42 # angle d'incidence en degrés
)( n1 = 1 # indice de l'air
)( n2 = 1.33 # indice de l'eau
)( i = radians(i) # on transforme i en radians
)( r = asin(n1*sin(i)/n2) # on sort r en radians
)( r = degrees(r) # on transforme r en degrés
)( print(r) # on affiche r

```

On constate que le code est long ; si le programme nécessite plusieurs fois ce calcul avec des valeurs différentes, il faudra copier-coller le code qui occupera plusieurs lignes et deviendra illisible.

### 3.4.2 Seconde méthode : avec une fonction

On va définir une fonction (`refraction`) ; on peut voir une fonction en programmation comme le concept du même nom en math.

La fonction `refraction` prendra en *paramètres* les valeurs de  $i$ ,  $n_1$  et  $n_2$  et retournera la valeur de  $r$  ( $r = \text{refraction}(i, n_1, n_2)$ ).

Voici comment on *définit* une fonction : MOT-CLÉ `DEF`



#### CODE PYTHON™

```

)(
)( from math import sin, asin, degrees, radians
)( def refraction(i, n1, n2) : # cette fonction prend 3 paramètres
)(     i = radians(i)
)(     r = asin(n1*sin(i)/n2)
)(     r = degrees(r)
)(     return r # la fonction retourne la valeur de r en degrés

```

Si, en console, vous tapez à présent `refraction(1, 1.33, 35)`, vous aurez bien comme résultat environ 25,5, ce qui était attendu.

L'avantage est qu'à présent, vous pouvez réaliser un autre calcul du même type (`refraction(1, 1.5, 40)`) de façon très simple.