Formation générale Python
Les Bases

Table des matières

1 Introduction

1.1 Qu'est-ce que Python ?

Python est un langage de programmation sous licence libre mis au point par Guido van Rossum au tout début des années 90. Le nom du langage vient du fait que son concepteur était un inconditionnel de la série "Monty Python".

Ce langage connaît un fort développement car son caractère ouvert permet à chacun de contribuer au code et de créer de nouvelles bibliothèques qui enrichissent le langage.

Jusqu'à l'année dernière, la version 2 de Python était encore bien utilisée notamment par des OS comme Linux ; ces derniers temps, il est fortement conseillé d'installer la version 3 qui unifie certaines caractéristiques du langage.

Python implémente un nouveau paradigme de programmation appelé POO (Programmation Orientée Objet) ; pour les non-initiés, cela ne veut absolument rien dire. Retenons simplement qu'en Python, « tout est objet » (forte parole), qu'un objet appartient à une classe d'objets, et que les objets disposent de plusieurs méthodes (ou fonctions) que l'on identifie dans le code par cette écriture : `objet.methode()`.

A priori, nous n'aurons pas à utiliser cette caractéristique du langage, destinée aux utilisateurs avancés ; néanmoins, ce dernier paragraphe pour incompréhensible qu'il soit, trouvera parfois un certain écho dans les morceaux de codes que vous trouverez ici ou là.

1.2 Pourquoi se mettre à Python ?

Pour les personnes n'ayant jamais fait de programmation, Python propose un langage relativement intuitif ; de plus la mise en forme du code est bien plus claire que dans d'autres langages de programmation comme C ou Java.

Pour notre discipline, la programmation fait une entrée très remarquée dans les programmes, en classe de Seconde, de Première (spécialité), dans l'enseignement scientifique, dans l'enseignement technologique optionnel Sciences et Laboratoire. On voit en effet apparaître des termes comme "Programmation", "codage" ou "microcontrôleurs".

Le langage de programmation recommandé est Python, que les élèves utilisent aussi en Mathématiques et sans doute dans les nouveaux enseignements Sciences Numériques et Technologie (Seconde) ou Numérique et Sciences Informatiques (spécialité Première).

Enfin, et c'est un avis personnel du rédacteur de ce tutoriel qui ne connaissait absolument rien à ce domaine il y a encore 3 ans, la programmation oblige à changer sa manière de réfléchir aux problèmes ; en ce sens, c'est intellectuellement très enrichissant. Et on se rend assez vite compte que c'est un outil extrêmement puissant.

1.3 Installation de la base

1.3.1 Sous Windows

Il existe plusieurs possibilités d'installation sous Windows ; je détaille plus bas une installation que j'ai pu tester ; il en existe d'autres.

Je signale notamment une distribution Python "clé en main", avec toutes les bibliothèques intéressantes pour les Sciences appelée EduPython, disponible à cette adresse : `https://edupython.tuxfamily.org/`.

Elle contient donc tout le nécessaire pour pratiquer Python, des éditeurs francisés et elle est bien documentée. Elle est notamment souvent utilisée par nos collègues de Maths (plus grande facilité d'installation en réseau). Il est probable que ce soit la distribution déjà présente dans vos établissements.

La suite de cette section s'intéresse à l'installation de la base de Python et de son éditeur nommé IDLE sous Windows par une autre méthode (celle que j'ai pu tester), et au rajout de bibliothèques externes.

Python n'est pas présent dans les distributions Windows. Le tutoriel que je donne ici semble valable pour Windows 7, 8 et 10.

- rendez-vous à la page `www.python.org`
- dans "Downloads", sélectionnez le lien vers `Python3.7.xx`
- Enregistrez le fichier `xxx.exe`
- Rendez-vous ensuite dans le dossier de Téléchargements
- Double-cliquez sur le fichier téléchargé pour l'installer
- Dans la fenêtre qui s'ouvre :
  notez le répertoire d'installation de Python qui est inscrit en dessous de `Install Now`, ça peut être utile.

!!!!!! Ne cliquez pas encore sur `Install Now` !!!!!!

!!!!!! Assurez-vous de cocher la case "Add Python3.7 to PATH en bas de la page !!!!!!

- choisissez "Install Now"
  Python s'installe alors ainsi que certaines bibliothèques intéressantes, comme on le verra plus tard.
- Une fois l'installation terminée, vous pouvez fermer la fenêtre.

Vous pouvez déjà vérifier que Python est bien installé ; comme vous brûlez de voir la bête, rendez-vous dans votre menu sur le lien nommé IDLE intégré à Python (c'est l'éditeur de base de Python) et testez-le pour :

— des maths : 2+3 renvoie le bon résultat... ouf !

— une première boucle : (n'écrivez pas ce qui suit #, ce sont des commentaires pour vous guider, ils ne sont pas lus par Python). Une fois compilé, tâchez de comprendre ce que fait ce bout de code.

code Python

```
>>> for i in range(5) : # taper un retour à la ligne, une indentation se crée à la ligne suivante
        print(i**2) # taper 2 retours à la ligne
```

On peut déjà faire énormément de choses avec la base que nous venons d'installer mais nous aurons besoin d'autres bibliothèques qui ne sont pas incluses dans la version de base de Python.

1.3.2 Sous Linux

Python est déjà installé par défaut sur Linux.
Vous pouvez ainsi tester les exemples donnés ci-dessus pour Windows.
Je parle ici des versions de Linux découlant de Debian, Ubuntu.
Nous verrons plus loin que d'autres bibliothèques que celles présentes dans la base sont utiles ; les commandes à rentrer en console pour pouvoir les installer sont :

```
sudo apt-get install python3-pip
sudo apt-get install idle-python3.x
sudo python3.x -m pip install numpy
sudo apt-get install python3.xxx-matplotlib
sudo python3.x -m pip install scipy pygame sympy vpython
sudo python3.x -m pip install pyserial (pour la gestion du port série // arduino)
```

1.3.3 Sous MacOS

La base de Python est déjà installée dans MacOS ; je ne connais pas par contre le processus pour installer PIP (s'il n'est pas installé) afin d'obtenir de nouvelles bibliothèques.
La mise à jour de PIP se fait par la commande suivante : `pip3 install -upgrade pip`.
Vous pouvez tester les exemples donnés ci-dessus pour Windows.

1.4 Installer des bibliothèques supplémentaires

Avec la base de Python, il est possible de faire déjà beaucoup de choses, malheureusement certaines choses sont impossibles (tracer des courbes, créer des objets en 3D, communiquer via un port série avec une carte d'acquisition telle ArduinoTM, faire du calcul formel...).

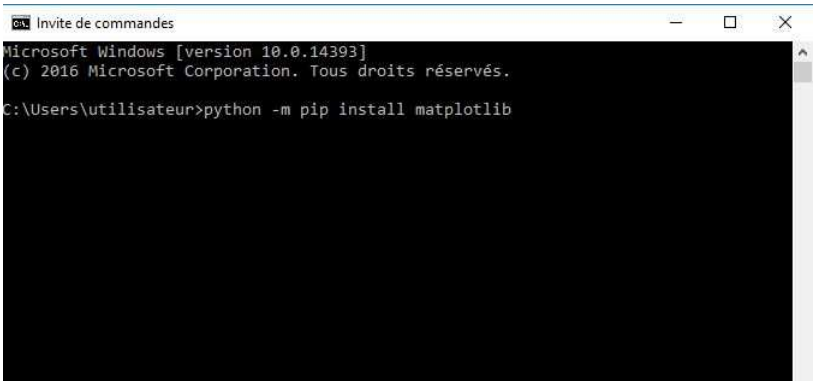
Heureusement, dans la base installée précédemment, se trouve un paquet nommé PIP qui est un installateur de bibliothèques.
Il va nous suffire de l'utiliser.

1.4.1 Sous Windows

— "Entrez" dans la console (l'écran tout noir qui fait peur)

Si vous ne la trouvez pas, taper "console" ou "cmd.exe" dans la barre de recherche de Windows.

— Ensuite c'est très simple ; pour tous les paquets qu'on voudra installer, il suffira de taper
`python -m pip install ...`
Par exemple, pour la bibliothèque `matplotlib` qui permet de tracer des courbes, il suffira de taper ceci avant de faire un "retour chariot"



— De la même manière, installez les bibliothèques :

— numpy (peut-être déjà installé)	— sympy	— csv
— scipy	— vpython	— pandas
— pygame	— pyserial	

REMARQUES :

— Même si vous n'en avez aucune utilité, installez-les tout de même : elles sont vraiment légères pour les disques durs actuels et ça ne coûte rien.

— Sur l'ordinateur où j'ai pu tester une installation Python sous Windows, la bibliothèque vpython ne s'installe pas ; c'est dommage car elle me semble très utile pour notre discipline. Cela dit, on peut accéder à un site : `www.glowscript.org` qui utilise directement en ligne le langage Python et notamment la bibliothèque vpython. On peut ainsi tester et utiliser cette bibliothèque même sans l'installer.

— À quoi servent ces bibliothèques ?
Pour ce que j'ai pu faire avec :
  — matplotlib permet de tracer des courbes, des diagrammes, de créer des animations ...
  — numpy permet de gérer des tableaux de valeurs, et permet de calculer simplement l'image d'un tableau de valeurs par une fonction (voir exemple plus bas).
  — scipy est une bibliothèque scientifique permettant beaucoup de choses...j'ai pu tester l'analyse spectrale de fonctions ou de fichiers audio notamment.
  — pygame est une bibliothèque pour créer des jeux évolués sous Python ; comme elle dispose d'un moteur physique très performant, j'ai pensé qu'on pourrait peut-être l'utiliser dans notre discipline. Je ne l'ai jamais testée.
  — sympy permet de faire notamment du calcul formel.
  — vpython permet de créer des simulations en 3D
  — pyserial permet à Python de communiquer via le port série (par exemple avec une carte d'acquisition comme ArduinoTM...)
  — csv et pandas vont permettre d'importer des fichiers de données et de les traiter par Python. La bibliothèque pandas est notamment très puissante (voir le vade-mecum sur l'importation de données) et rappelle le traitement qu'on peut faire sous tableur, la puissance de Python pour gérer de très grosses bases de données en plus.

1.4.2 Sous Linux

Cas déjà évoqué plus haut

1.4.3 Sous MacOS

L'installation de bibliothèques supplémentaires se réalise par la commande suivante :
`pip install nom_du_paquet` ; par exemple pour installer scipy, on tapera `pip install scipy`.

1.4.4 Test

1. Pour tester si les bibliothèques se sont bien installées :
   — nous allons demander à Python d'importer les bibliothèques (nous ne les utiliserons pas mais si il y a eu un problème lors de l'installation, Python renverra alors une erreur)
   — Pour cela il suffit de taper :
   `>>>import nom_de_la_bibliotheque`
   Par exemple :
   `>>>import scipy`
   — Si Python ne se met pas à crier tout rouge, c'est que la bibliothèque est bien installée.
   REMARQUE : Pour la bibliothèque pyserial, la commande est :
   `>>>import serial`

2. Testons les bibliothèques matplotlib et pyplot
   On souhaite tracer la courbe représentative de la fonction $f : x \longmapsto f(x) = \sin(x)$ entre -4 et 4
   le principe :
   — nous utiliserons la bibliothèque matplotlib pour le tracé et la bibliothèque numpy pour créer des tableaux de coordonnées $(x; y)$
   — on va appeler le paquet pyplot de matplotlib et nous l'utiliserons sous l'appellation "plt"
   — on va appeler le paquet numpy et nous l'utiliserons sous l'appellation "np"
   — nous allons créer un tableau X de valeurs d'abscisses entre -4 et 4 ; nous choisissons de prendre 200 points dans cet intervalle pour le calcul.
   — nous utilisons la fonction sinus du paquet numpy pour créer un tableau Y image de notre tableau X (terme à terme)
   — on construit la courbe en reliant point à point avec la fonction "plot" du paquet pyplot
   — enfin, on affiche la courbe avec la fonction "show" du même paquet.

code Python

```
>>> import matplotlib.pyplot as plt
>>> import numpy as np
>>> X = np.linspace(-4,4,200)
>>> Y = np.sin(X)
>>> plt.plot(X,Y)
>>> plt.show()
```

2 IDLE : console et éditeur

2.1 L'éditeur d'IDLE

Vous avez dû vous rendre compte que l'utilisation de la console IDLE pour taper du code n'est pas très pratique : il suffit de faire une erreur pour devoir tout retaper, on ne peut pas remonter dans les lignes précédentes pour corriger.

IDLE propose en plus de la **console** que vous avez utilisée et identifiée par les 3 chevrons (>>>) un **éditeur** qui permet de créer réellement des programmes.

Pour y accéder, quand vous êtes dans la console, taper `Ctrl+N` ou `File→New File`.

À présent nous appellerons `Console` l'interface avec les 3 chevrons (>>>) et `Éditeur` l'interface sans les chevrons.

Placez-vous dans l'**éditeur**. Vous pouvez à présent taper plusieurs lignes de code, les modifier ...

Pour exemple, vous pouvez retaper les codes vus précédemment. **Mais comment voir le résultat ?**

Il faut pour cela lancer la compilation : on le réalise en tapant `F5` ou `Run→Run Module`.

Python vous demandera alors d'enregistrer votre fichier, vous pourrez l'appeler par exemple `test.py`, l'extension `.py` permettant d'identifier un fichier python.

La compilation permettra de voir ou d'afficher le rendu du programme (par exemple sur la console).

code Python

```
print("Python pour la Physique-Chimie") #affiche sur la console la chaîne de caractères
a = 3
print(a) #affiche sur la console la valeur de la variable "a"
```

Comparatif Console / Éditeur

la console : avec les chevrons >>>	**l'éditeur : sans les chevrons**
— permet de faire des tests rapides — un simple appui sur `Entrée` donne le résultat de la compilation — aucun enregistrement n'est nécessaire — donne le résultat de compilations de programmes plus longs travaillés sur l'éditeur — ne permet pas de remonter dans les commandes pour correction	— permet de rentrer des codes longs — le programme doit être enregistré — on compile en tapant `F5` — on peut se déplacer dans le document pour corriger, modifier le code...

2.2 Écran de travail

On tape le code ci-dessous sur l'éditeur :

code Python

```
for x in range(20) : # pour x variant entre les valeurs entières 0 et 19 – Noter l'indentation qui s'est créée en dessous
    y = x**2 # on calcule y : carré de x
    print("valeur de x : ", x, " — valeur de y : ", y) #on affiche les valeurs de x et y
```

La console va nous afficher les valeurs de x et de y. Il peut être intéressant à l'écran de réduire vos fenêtres pour voir afficher l'**éditeur**, qui contient votre code, et la **console** qui renvoie les résultats de la compilation.

2.3 Les "astuces" d'IDLE

1. IDLE ne propose pas à côté de chaque ligne, le numéro de chacune. Mais celui-ci est visible en bas à droite : `Ln : ...`
2. IDLE (console ou éditeur) propose une petite aide sur les fonctions utilisées : par exemple, si je tape `print(` il me propose une série de paramètres à placer après la fonction rentrée.
3. IDLE dispose aussi de l'autocomplétion : si je tape `pr` suivi de la touche `Tab`, il me propose une série de commandes proposant cette série de touches. Je peux choisir celle qui me convient en me déplaçant avec les flèches directionnelles du clavier.
4. La console IDLE nous permettra de faire des tests et de visualiser les résultats de nos programmes ; nos programmes plus longs seront à taper dans l'éditeur.

2.4 Et si ça plante ?

L'nevironnement IDLE est très robuste car il est lui-même implémenté par du code Python. Lorsque votre programme plante, tourne sans fin, le problème se situe généralement entre l'ordinateur et la chaise ...

Pour arrêter un programme qui plante, il suffit de taper `Ctrl + C` dans la console.

Pour exemple, vous pouvez tester ce programme qui entre dans une boucle infinie et l'arrêter avec la combinaison de touches citée plus haut :

code Python

```
>>> while True : # Notez encore l'indentation qui se crée en dessous
        print("boucle sans fin")
```

Maintenant que notre base est installée et que nous disposons d'une console de tests et d'un éditeur, nous allons pouvoir commencer à rentrer dans le monde des commandes Python.

3 Variables, modules supplémentaires, fonctions importantes, listes

3.1 Variables et types de variables

3.1.1 Variables

Les lignes suivantes peuvent être tapées indifféremment sous l'éditeur IDLE ou sous la console.

Une variable peut être vue comme une "boîte" dans laquelle on met quelque chose :

code Python

```
>>> truc = 5 # On affecte la valeur 5 à la variable "truc"
>>> print(truc) # on affiche la valeur de truc
```

Cette instruction affecte le chiffre "5" dans la variable nommée `truc`.

!!! Il est important de comprendre la signification du signe "=" : celui-ci n'a rien à voir avec la notion d'égalité ; il veut juste dire `"affecte"` et l'instruction se lit par la gauche :

```
On affecte la valeur 5 à la variable "truc"
```

Si ensuite on tape :

code Python

```
>>> truc = 2
>>> print(truc)
```

la valeur de la variable a changé.

De même, on peut écrire la ligne suivante qui serait incohérente d'un point de vue mathématique, mais pas en programmation.

code Python

```
>>> truc = truc + 1 # ou aussi : truc += 1
>>> print(truc)
```

(On affecte la valeur de "truc" augmentée de 1 à la variable "truc").
Noter l'équivalence de signification entre : `truc=truc+1` et `truc+=1`

3.1.2 Types des variables

On peut mettre tout ce qu'on veut dans une variable :

code Python

```
>>> var1 = 3 # les entiers sont appelés "integers" en anglais
>>> var2 = 5.2 # les décimaux sont appelés "flottants" par Python
>>> var3 = "chaîne de caractères" # une suite de caractères est une chaîne de caractères en Python
>>> var4 = [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
```

Le dernier type de variables sera souvent utilisé dans des boucles ou pour fixer certaines conditions.

Si on veut connaître le "type" d'une variable, il existe la fonction `type()` qui renvoie le type de la variable :

code Python

```
>>> type(var1) # renvoie 'int' pour entier
>>> type(var2) # renvoie 'float' pour flottant
>>> type(var3) # renvoie 'str' pour string, chaîne de caractères
>>> type(var4) # renvoie 'list' pour liste
>>> type(var5) # renvoie 'bool' pour booléen
```

3.2 Et pour une utilisation plus avancée ? Importons des bibliothèques (ou modules)

Vérifiez que le code suivant renvoie une erreur :

code Python
```
>>> sin(1.5)
```

Avec la base Python, de nombreuses opérations mathématiques sont impossibles. Il faut pour les réaliser utiliser les bibliothèques qui contiennent ces fonctions.

La bibliothèque `math` contient de nombreuses fonctions mathématiques ; nous allons l'importer pour l'utiliser :

code Python
```
>>> import math # on importe ma bibliothèque math
>>> math.sin(1.5) # on utilise la fonction sinus de la bibliothèque math
```

Cette écriture `math.sin()` ne vous rappelle-t-elle pas le passage ésotérique de l'introduction ? (lien cliquable1.1)

Pour voir toutes les fonctions de la bibliothèque `math` :

code Python
```
>>> dir(math)
```

La console affiche l'ensemble des fonctions de la bibliothèque `math`.

On peut aussi importer une bibliothèque de façon plus légère :

code Python
```
>>> import math as m # on va utiliser l'alias 'm' pour la bibliothèque math
>>> m.sin(1.5) # on utilise la fonction sinus de la bibliothèque math
```

Et pour ne pas s'embêter avec les alias on peut aussi importer uniquement les fonctions qui nous intéressent :

code Python
```
>>> from math import sin # on importe la fonction sinus de la bibli math
>>> sin(1.5)
```

Ou alors on importe carrément l'ensemble des fonctions de la bibliothèque (* sous Linux et Python signifie "all")

code Python
```
>>> from math import * # on importe toutes les fonctions de la bibli math
>>> sin(1.5)
>>> cos(3)
```

Le problème de cette dernière méthode est que lors de l'importation de différentes bibliothèques dans un programme, certaines fonctions pourront porter le même nom et Python renverra alors une erreur, ne sachant de quelle bibliothèque il s'agit.

La méthode qui semble la plus conseillée dans ce cas est donc de se choisir un alias.

3.3 Des fonctions importantes

3.3.1 La fonction `print()`

Nous l'avons déjà rencontrée ; elle permet d'afficher à l'écran ce qu'on lui passe comme paramètre :

code Python

```
>>> print("chaîne de caractères") # affiche le texte passé en paramètre
>>> a = 2
>>> print(a) # affiche la valeur de la variable 'a'
>>> print("a") # affiche la chaîne de caractères "a"
>>> print("la valeur de la variable a est : ", a) # on peut afficher texte et valeur de variable
>>> print("texte 1") ; print("   texte 2") # print revient à la ligne par défaut
>>> print("texte 1, end=' ') ; print("   texte 2") # on peut afficher sur la même ligne
```

REMARQUE : il faut noter ici une différence entre **console** et **éditeur** ; pour voir la valeur d'une variable `a`, il suffit en fait de taper `a` suivi d'un retour chariot dans la console, alors qu'il faudra écrire `print(a)` dans l'éditeur pour que s'affiche à la console la valeur de `a`.

Je choisis généralement de présenter la forme la plus longue dans les exemples, même sur la console, sauf dans le 3.4 pour simplifier les commandes.

3.3.2 Interagir avec l'utilisateur : la fonction `input()`

On aura souvent besoin que l'utilisateur du programme rentre une valeur (par exemple pour un tableau d'avancement ou autre) ; on utilise alors la fonction `input()`.

code Python

```
>>> n1 = input("Quelle quantité de matière de H2O au début (en mol) ? ")
```

On constate que Python nous retourne la phrase passée en paramètre et attend ; nous pouvons alors rentrer la quantité que nous désirons, par exemple `0.3`. La variable `n1` contiendra donc `0.3`

Si nous réalisons un calcul sur `n1`, par exemple `n1 - 0.1`, Python nous renvoie une erreur.

Comprenez-vous pourquoi ?

La fonction `input()` retourne forcément une chaîne de caractères et pas un nombre.

Il faudra donc que nous convertissions cette chaîne de caractères en décimal avant de faire le calcul.

Cela se réalise très simplement :

code Python

```
>>> n1 = input("Quelle quantité de matière de H2O au début (en mol) ? ")
>>> n1 = float(n1) # cette ligne transforme la chaîne de caractères 'n1' en flottants et la place dans la variable 'n1'
>>> print(n1-0.1) # on peut faire le calcul
```

Une manière plus rapide de le réaliser est d'écrire :

code Python

```
>>> n1 = float(input("Quelle quantité de matière de H2O au début (en mol) ? "))
>>> print(n1-0.1) # on peut faire le calcul
```

Vous comprenez alors qu'on peut convertir une variable dans un autre type à l'aide des fonctions `int()`, `str()` ...

3.3.3 La fonction `range()`

On rencontre la fonction `range()` très souvent dans les programmes. Sa forme de base est la suivante :
`range(début inclus, fin exclue, pas)`

Le code suivant `range(1,17,2)` renverra donc tous les nombres impairs entre 1 inclus et 17 exclus.

Lorsqu'on le tape sous Python, cela ne renvoie pas la liste de ces nombres.
Pour l'avoir, nous allons parcourir cet objet `range()` avec une boucle (nous verrons les boucles plus tard) :

code Python

```
>>> for i in range(1,17,2) : #on définit une variable i qui va parcourir la liste de valeurs
        # Notez l'indentation qui s'est créée lors du retour à la ligne après les " :"
        print(i) # on affiche i sur la console
```

On peut ne définir que le nombre de départ et celui de fin, dans ce cas, le pas sera de 1 :

code Python

```
>>> for i in range(3,12) :
        print(i)
```

La plupart du temps, on ne précise que la valeur finale de la fonction `range()`. La valeur initiale est alors de 0 et le pas de 1.

code Python

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

REMARQUE : Cette dernière écriture nous permettra surtout de **répéter quelque chose plusieurs fois** ; en effet, à la compilation, la variable `i` va parcourir la liste à chaque tour de boucle.

En conséquence, la ligne `for i in range(10)` peut-être traduite par : `"Fais 10 fois"`

Vous pouvez vous en convaincre avec le code suivant :

code Python

```
>>> for i in range(10) :
        print("je répète")
```

REMARQUE BIS : On note, et nous l'avons déjà constaté, que lors d'un retour à la ligne après " :", Python indente la ligne suivante avec une tabulation. Cela permet de visualiser dans le code ce qui est dans la boucle `for` pour les exemples ci-dessus.

Cela permet d'avoir une vision plus claire du code mais peut entraîner aussi de nombreuses erreurs : attention à ne pas oublier la tabulation et à ne pas se perdre dans des tabulations multiples quand plusieurs boucles `for` sont imbriquées.

3.4 Un objet fondamental, la liste

Les listes sont des objets fondamentaux en Python et nous aurons sans doute à nous en servir dans notre discipline (pour tracer des courbes, classer des valeurs, les conserver, représenter des vecteurs, des coordonnées ...).

Je donne ici les commandes de base des listes, qui nous seront utiles :

code Python

```
>>> L1 = [ ] # création d'une liste vide
>>> L2 = [1, 2.45, 3, 4.72, "dernier élément"] # une liste avec 5 éléments
>>> L2.append(6) # fonction qui ajoute un élément à la fin de la liste L2
>>> len(L2) # affiche la longueur de la liste L2
>>> L2[0] # affiche le premier élément de L2
>>> L2[1] # affiche le second élément de L2
>>> L2[-1] # affiche le dernier élément de L2
>>> L2[-2] # affiche l'avant-dernier élément de L2
```

REMARQUES :

— Il faudra donc bien faire attention à la numérotation qu'utilise Python : le premier élément d'une liste est le `numéro 0`.

— Dans l'**éditeur**, il faudrait rajouter la fonction `print()` pour voir afficher les résultats ci-dessus (cf remarque du 3.3.1).
Par exemple, pour voir afficher après compilation la longueur d'une liste L travaillée dans l'**éditeur**, il faudrait taper `print(len(L))` dans l'**éditeur**.

— l'écriture `L2.append()` doit à présent clairement vous évoquer l'introduction 1.1

Tâchez de comprendre ce que fait le code suivant tapé dans l'éditeur et sauvegardé sous le nom `courbe.py` par exemple ; testez-le.

code Python

```
import matplotlib.pyplot as plt
X = [ ]
Y = [ ]
for i in range(-100,100) :
    x = i
    y = i**2
    X.append(x)
    Y.append(y)
    print(x,"   ;   ", y)
plt.plot(X,Y) # rajoutez ensuite dans la parenthèse : color = "red", marker = "+", linewidth = 0.5
plt.show()
```

4 Conditions en Python

4.1 Une bibliothèque pour le hasard

Cette partie va nous permettre d'introduire une bibliothèque bien pratique de Python qui gère le hasard, c'est la bibliothèque `random`.

Importons la bibliothèque dans un fichier que nous utiliserons ensuite, par exemple par la commande :

code Python
```
>>> import random as r
```

(On a choisi l'alias "r" pour la bibliothèque ; on aurait aussi pu bien écrire : `from random import *` , ce qui nous aurait permis d'alléger l'écriture du code).

La bibliothèque va nous permettre notamment de choisir un nombre aléatoire entre 2 valeurs données (incluses dans l'intervalle) :

code Python
```
>>> r.randint(-10,10) # retourne un nombre aléatoire entre -10 et 10
```

4.2 Les conditions en programmation

4.2.1 Principe

On souhaite souvent réaliser des actions en fonction d'un test préalable.
Les commandes à retenir pour les conditions sont :

code Python
```
if condition : #si la condition est vérifiée – noter la tabulation en dessous
    xxxxxx # on fait xxxxxx
else  : # sinon – noter la tabulation en dessous
    yyyyyy # on fait yyyyy
```

on peut enchâsser d'autres conditions en utilisant le mot-clé `elif` mais je ne l'ai que rarement utilisé.
les contraintes sur les conditions peuvent être :

condition	signification	condition	signification
==	est égal à	<	est inférieur à
!=	est différent de	>=	est supérieur ou égal à
>	est supérieur à	<=	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 en gras du 3.1.1

On peut associer plusieurs conditions ensemble :

écriture	signification
`condition1 and condition2`	condition1 ET condition2
`condition1 or condition2`	condition1 OU condition2
...	...

4.3 Exemples d'utilisation

4.3.1 Exercice 1 :

— Le programme demande un nombre à l'utilisateur (négatif, positif ou nul)
— Si le nombre est positif ou nul, il le place dans une liste L1 qu'on aura au préalable créée vide,
— Sinon, il le place dans une liste L2 qu'on aura au préalable créée vide.
— ce processus doit se répéter 10 fois, et on affiche les 2 listes à la fin.
Tâchez d'écrire ce programme, il contient beaucoup de choses que nous avons déjà vues.

UNE CORRECTION POSSIBLE

```
L1 = [ ] # on créée 2 listes vides
L2 = [ ]
for i in range(10) : #ce qui suit sera réalisé 10 fois – noter la tabulation qui suit
    a = float(input("Choisis un nombre : ")) # demande un nombre + le transforme en flottant
    if a >= 0 : # noter la tabulation qui suit
        L1.append(a) # on ajoute l'élément à la liste L1
    else  : # noter la tabulation qui suit
        L2.append(a) # on ajoute l'élément à la liste L1
print(L1) # on affiche les 2 listes
print(L2)
```

Si vous avez eu des difficultés sur le code précédent, c'est bien normal ; comprenez bien chaque ligne et essayez-vous à l'exercice suivant.

4.3.2 Exercice 2 :

Même exercice mais cette fois-ci c'est Python qui choisit un nombre au hasard entre -100 et 100 ; on range le nombre dans les listes L1 ou L2 selon son signe.
Le processus doit se répéter 10 000 fois.
À la fin, Python affiche la longueur de la liste L1 et la longueur de la liste L2.

UNE CORRECTION POSSIBLE

```
import random as r # on importe la bibliothèque random sous l'alias 'r'
L1, L2 = [ ], [ ]
for i in range(10000) :
    a = r.randint(-100,100)
    if (a >= 0) :
        L1.append(a)
    else  :
        L2.append(a)
print(len(L1)) # on affiche la longueur des 2 listes
print(len(L2))
```

4.3.3 Exercice 3

Même chose mais cette fois-ci, on classifie selon que :
— le nombre retourné est positif ou nul ET plus petit strictement à 50 ; dans ce cas-là on le range dans la liste L1
— le nombre retourné est strictement négatif ET plus grand ou égal à -50 ; dans ce cas-là on le range dans la liste L2

UNE CORRECTION POSSIBLE

```
from random import randint # on importe la fonction randint de la bibliothèque random
L1, L2 = [ ], [ ]
for i in range(10000) :
    a = randint(-100,100)
    if (a >= 0) and (a < 50) :
        L1.append(a)
    else  :
        if (a < 0) and (a >= -50) :
            L2.append(a)
print(len(L1)) # on affiche la longueur des 2 listes
print(len(L2))
```

Beaucoup de choses dans ces exercices : importation de bibliothèques sous différentes formes, boucles, conditions multiples et quelques fonctions classiques déjà rencontrées.

Il existe d'autres types de mots-clés dans les structures conditionnelles (`not` ou `is`), de même les conditions peuvent être déclinées à travers de nombreux tests avec le mot-clé `elif` qui enrichit la structure classique, mais pour notre pratique je pense qu'ils sont peu utiles. Pour cette raison, je ne les présente pas.

5 Les boucles

Nous avons déjà rencontré les boucles avec la fonction `range()`. Nous allons généraliser ceci.

Une boucle consiste en la répétition d'une série de commandes ; soit on connaît d'avance le nombre de répétitions, on parle de boucles bornées, soit on ne connaît pas d'avance le nombre de répétitions et on fixe une condition pour arrêter la boucle, on parle alors de boucles non bornées.

Sorti de ces considérations théoriques, voyons les commandes permettant de les réaliser.

5.1 Boucles bornées

5.1.1 Principe

Ce type de boucle est caractérisé par le mot-clé `for`.

Le principe consiste à parcourir un objet, et à répéter une action chaque fois qu'on passe à un nouvel élément de l'objet.

Prenons un exemple :

code Python

```
>>> L = [1,5,12,27,274]
>>> for valeur in L : # on passe sur les 5 éléments de L, et pour chacun :
        print("je répète") # on affiche ...
```

Nous savons déjà que ce code est équivalent à :

code Python

```
>>> for i in range(5) : # on passe sur les 5 éléments (0,1,2,3,4), et pour chacun :
        print("je répète") # on affiche ...
```

L'objet sur lequel on fait passer la variable peut être aussi une chaîne de caractères :

code Python

```
>>> C = "abcdefg"
>>> for lettre in C : # on passe sur les 7 éléments de C, et pour chacun :
        print("je répète") # on affiche ...
```

5.1.2 Exercices :

1. Une classe, 35 copies

   Il est déconseillé, dans la profession, d'évaluer des copies à partir de leur place respective sur les marches d'un escalier après le jet du paquet de copies sur ce dernier ...

   C'est pourtant ce que vous allez faire ici :

   — Créez une liste vide
   — Pour chaque copie, attribuez-lui une note aléatoire entre 0 et 20, et stockez cette note dans la liste
   — affichez la liste

   UNE CORRECTION POSSIBLE

   ```
   from random import randint
   L = [ ] # liste qui contiendra la suite de notes
   for i in range(35) : # Fais 35 fois ...
       note = randint(0,20) # la variable note est affectée d'un nombre aléatoire entre 0 et 20
       L.append(note) # on ajoute cette valeur à la liste... puis on reboucle
   print(L) # quand la boucle est finie, on affiche la liste de notes
   ```

2. 35 copies, une moyenne
   Poursuivez votre code pour qu'il calcule la moyenne de ces copies

UNE CORRECTION POSSIBLE

```
...
somme = 0 # on intialise une variable qui contiendra la somme des notes
for i in L : # pour chaque élément de la liste
    somme += i # ou somme=somme+i ; on ajoute chaque note à 'somme'
moyenne = somme/len(L) # on est sorti de la boucle ; on calcule la moyenne
print(moyenne) # et on l'affiche
```

On retrouve dans cet exemple que l'instruction `variable = variable + 1` peut aussi s'écrire `variable += 1`

3. Écrivez un programme qui :
   — crée une liste L vide
   — remplit cette liste vide de 20 nombres aléatoires compris entre 0 et 200
   Puis,
   — parcourt la liste L et classe les nombres pairs de la liste dans une nouvelle liste L1 et les nombres impairs dans une nouvelle liste L2
   — Enfin, le programme affiche les 3 listes.

   DONNÉE :

   Python connaît l'arithmétique : il sait calculer quotient et reste d'une division euclidienne.

code Python

```
>>> n = 13
>>> n/5 # renvoie 2.6
>>> n // 5 # renvoie 2, le quotient de la division euclidienne de n par 5
>>> n % 5 # renvoie 3, le reste de la division euclidienne de n par 5
```

⟶ En particulier, si n est pair, $n\%2 = 0$; si n est impair, ce n'est pas le cas.

UNE CORRECTION POSSIBLE

```
from random import randint
L, L1, L2 = [ ], [ ], [ ]
for i in range(20) :
    a = randint(0,200)
    L.append(a)
for i in L :
    if (i % 2 == 0) :
        L1.append(i)
    else  :
        L2.append(i)
print(L) ; print(L1) ; print(L2)
```

5.2 Boucles non bornées

Il arrive fréquemment que l'on ne sache pas combien de fois on doit réaliser une boucle.
Imaginons l'exercice suivant : LA SUITE DE SYRACUSE
— Choisir un nombre entier positif noté n
— Si n est pair, on le divise par 2
— Si n est impair, on réalise le calcul suivant : $3 \times n + 1$

Et on répète cela . . . jusqu'à ce qu'on arrive à une suite récurrente (4,2,1,4,2,1,4,2,1. . .)
Testez dans votre tête avec quelques nombres pour vous convaincre qu'on obtient toujours ce résultat.
On désire que Python construise une liste avec les nombres rencontrés au cours de cette suite de Syracuse.
On comprend bien qu'on effectue une boucle, mais on ne sait pas jusqu'à quand. . .
Le problème, on le voit, c'est qu'on ne sait pas combien de tours de boucles sont à réaliser.
Il faudrait pouvoir dire : Tant qu'on n'arrive pas à 4,2,1...
ça tombe plutôt bien : il existe le mot-clé `while`.
la structure est la suivante :

code Python

```
while condition : # tant que la condition est vraie – noter la tabulation en dessous
    xxxxxx # on réalise la suite de commandes xxxxxx
```

Trouver le programme qui met dans une liste tous les membres de la suite de Syracuse d'un nombre rentré par l'utilisateur jusqu'à arriver à 4,2,1 et affiche cette liste.

UNE CORRECTION POSSIBLE

```
n = int(input("Donne ton nombre : "))
L = [n]
while n != 1 : # le programme reste dans la boucle tant que n différent de 1
    if n % 2 == 0 : # si n est pair
        n = n / 2 # on remplace n par sa moitié. . . et on sort du if...else...
    else  :
        n= 3*n + 1 # on remplace n par 3xn + 1. . . et on sort du if...else...
    L.append(n) # on ajoute la valeur de n à la liste et on recommence la boucle
print(L) # on affiche la liste, une fois sorti de la boucle
```

5.3 Utiliser une boucle while à la place d'une boucle for

On peut utiliser le mot-clé `while` dans une boucle bornée ; cela se rencontre assez souvent.
Exemple :

code Python

```
>>> i = 0 # initialisation d'une variable
>>> while i <= 5 :
        print(i)
        i += 1 # ou i=i+1 ; incrémentation de la variable à chaque tour de boucle
```

Cette boucle prend la structure très classique : initialisation – condition sur la variable – commandes – incrémentation de la variable.

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.

5.4 Exercices :

5.4.1 Exercices du 5.1.2

Reprenez les 3 exercices en remplaçant les boucles `for` par des boucles `while`.
Ceci est très classique et va vous obliger à jongler entre la longueur de la liste et le numéro des indices.

1. Exercice 1 : Une classe, 35 copies

UNE CORRECTION POSSIBLE

```
from random import randint
L = [ ]
i = 1 # initialisation d'une variable qui va se déplacer sur les copies
while i <= 35 :
    note = randint(0,20)
    L.append(note)
    i+= 1 # incrémentation de la variable pour le tour de boucle suivant
print(L)
```

2. Exercice 2 : 35 copies, une moyenne

CONTRAINTE :

Interdit d'utiliser les nombres 34 ou 35 dans votre programme. Il doit pouvoir s'adapter à la donnée de n'importe quelle liste, quelle que soit sa longueur.

UNE CORRECTION POSSIBLE

```
...
somme = 0
i = 0 # initialisation de la variable qui va parcourir les éléments de la liste
while i < len(L) : # donc on s'arrête à len(L)-1
    sommme = somme + L[i] # L[34] est bien le dernier terme de la liste
    i += 1 # on incrémente la variable pour le tour de boucle suivant
moyenne = somme/len(L)
print(moyenne)
```

3. Exercice 3 :

UNE CORRECTION POSSIBLE

```
from random import randint
L, L1, L2 = [ ], [ ], [ ]
i = 1
while i <= 20 :
    a = randint(0,200)
    L.append(a)
    i += 1 # on pense bien à incrémenter ! ! !
i = 0
while i < len(L) : # donc on s'arrête à len(L)-1
    if (L[i] % 2 == 0) :
        L1.append(L[i])
    else  :
        L2.append(L[i])
    i += 1 # on pense bien à incrémenter ! ! !
print(L) ; print(L1) ; print(L2)
```

5.4.2 Exercice supplémentaire : se servir de l'aide contextuelle d'IDLE

Votre programme doit :

— créer une liste contenant un nombre aléatoire (que vous ne connaissez pas) entre 0 et 10

— la liste doit se compléter avec des nombres aléatoires pris entre 0 et 100 tant que la différence entre le dernier élément de la liste et le premier reste inférieure à 80.

— Quand la condition n'est plus remplie, la liste est pleine, et Python nous l'affiche.

CONTRAINTE :

Quel que soit le terme de la liste, la différence avec le premier terme doit être inférieure à 80.

DONNÉE :

Pensez à utiliser la touche Tab pour voir les fonctions proposées par Python. Si je tape `L.` suivi d'un appui sur la touche `Tab`, cela donne les différentes fonctions qu'on peut appliquer aux listes. Il en existe sans doute une pour supprimer un élément d'une liste ...

UNE CORRECTION POSSIBLE

```
from random import randint
L = [ ]
L.append(randint(0,10))
while L[-1] - L[0] <= 80 :
    L.append(randint(0,100))
L.pop() # cette fonction supprime le dernier élément de la liste
print(L)
```

À l'essai, mon code gardait le dernier terme dont la différence avec le premier était plus grande que 80 ; il a donc fallu le supprimer et les listes possèdent une fonction qui effectue ce travail. En utilisant la touche `Tab`, on peut faire défiler les fonctions associées aux listes. Un test rapide sur la fonction `pop()` prouve qu'elle supprime bien le dernier élément d'une liste.

Cette méthode de suppression d'un élément d'une liste pour coller à la consigne est un peu faite « à la hache » ; comme toujours, il existe sans doute d'autres méthodes, plus élégantes.

6 Fonctions, procédures en Python

On a déjà pu réaliser des choses évoluées et vous avez dû vous rendre compte que les structures rencontrées permettent de gagner beaucoup de temps.

Il nous reste à voir la notion de fonctions qui, elle aussi, va permettre un gain de temps énorme ; de plus, les codes obtenus en définissant et utilisant ces fonctions sont plus lisibles.

6.1 Une nouvelle bibliothèque : `turtle`

6.1.1 Présentation

Pour introduire les fonctions, nous utiliserons un paquet de base chez Python : `turtle`.

Commençons par l'importer, pour l'instant en console et voyons les fonctions que propose cette bibliothèque avec la fonction `dir()`.

code Python

```
>>> import turtle
>>> dir(turtle)
```

Nous voyons qu'une des fonctions s'appelle `forward` ; nos bases, très limitées mais existantes en anglais, nous indiquent que cela signifie "vers l'avant".

Tapons alors dans la console `turtle.forward(` et une indication nous donne l'information salvatrice : il faut rentrer une distance dans les parenthèses.

Tapons alors dans la console `turtle.forward(50)`, validons et là ... nous voyons une petite flèche qui s'est déplacée de 50 pixels.

Dès lors, tout est clair, `turtle` va nous permettre de coder de petits dessins.

Le paquet "turtle" permet de faire sortir vos résultats de programmes de la console ; pour l'avoir testé avec les élèves, ils l'apprécient.

6.1.2 Commandes de base

À l'utilisation de `turtle`, certaines commandes sont pratiques à connaître :
— on peut imaginer que le curseur (la tortue) tient un stylo
— l'origine du repère utilisé est (0,0) au centre de la fenêtre

Commande	Traduction
forward(xxx) ou fd(xxx)	avance de xxx px
backward(xxx) ou bd(xxx)	recule de xxx px
up()	lève le stylo pour ne plus écrire
down()	baisse le stylo pour écrire
goto(x,y)	aller au point de coordonnées (x ;y)
left(α)	tourne vers la gauche d'un angle α (en degrés)
right(α)	tourne vers la droite d'un angle α (en degrés)
write("truc")	écris le texte à l'endroit où est placé le curseur
reset()	efface tout et revient à l'origine
shape("turtle")	change l'apparence du curseur en tortue
speed(nombre)	gère la vitesse (1 - 3 - 6 - 10 - 0 de la plus petite à la plus grande)
...	...

Pour plus de renseignements, un tuto bien fait sur `turtle` est ici : (référence cliquable)
`https://zestedesavoir.com/tutoriels/944/a-la-decouverte-de-turtle/`

Exercice : Dessiner un carré de côté 100 pixels.

méthode longue :

UNE CORRECTION POSSIBLE

```
from turtle import *
fd(100)
left(90)
fd(100)
left(90)
fd(100)
left(90)
fd(100)
left(90)
```

méthode courte :

UNE CORRECTION POSSIBLE

```
from turtle import *
for i in range(4) :
    fd(100)
    left(90)
```

On privilégiera bien sûr la seconde méthode.

6.2 Première approche des fonctions

6.2.1 Un carré

Le problème avec les méthodes précédentes, c'est que si on change la valeur du côté du carré, il faut tout recommencer.

L'idée est alors de créer une fonction qu'on appelle `carre()` par exemple (Python n'aime pas les accents) et à laquelle on pourra indiquer la longueur du côté. Quand on écrira cette fonction, Python dessinera un carré avec la bonne valeur du côté :

`carre(200)` provoquera le dessin d'un carré de 200 pixels de côté.

le mot-clé pour définir une fonction est : `def`, et ensuite on indique la définition de la fonction (la suite de commandes que turtle effectuera quand il lira la nouvelle fonction `carre`).

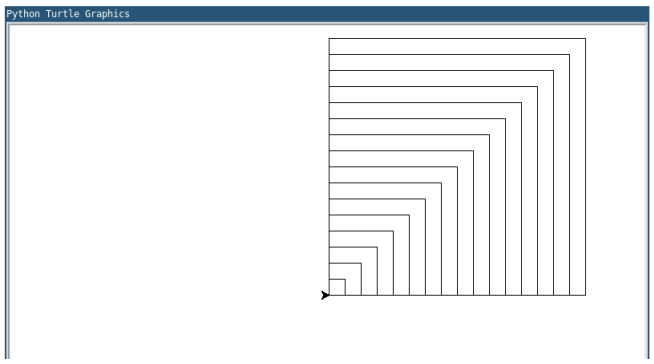
code Python

```
def carre(longueur) : # le paramètre de la fonction est la variable 'longueur'
    for i in range(4) : # noter les indentations, avant et après
        fd(longueur) # c'est ici que le paramètre 'longueur' va jouer son rôle
        left(90)
```

Et si à présent on tape `carre(200)`, nous aurons l'affichage d'un carré de côté 200 pixels.

6.2.2 Des carrés

Tentez d'obtenir, en 2 lignes, cette figure :



UNE CORRECTION POSSIBLE

```
>>> for i in range(20,360,20) :
>>>         carre(i)
```

EXERCICE : Construire alors une autre fonction appelée `figure` pour tracer cette figure ; cette fonction prend en paramètres la longueur du côté du plus petit carré, la longueur du côté du plus grand carré, et le pas de variation entre chaque longueur de côté des carrés successifs. Testez-la.

UNE CORRECTION POSSIBLE

```
def figure(mini, maxi, pas) : # on a pris mini et maxi car min et max sont des mots-clés pour Python
    for i in range(mini, maxi+pas, pas) : # Pourquoi maxi+pas pour la borne supérieure de l'intervalle ?
        carre(i)
```

CONCLUSIONS :

— L'appel de la fonction par : `figure(30,180,10)` tracera 16 carrés dont les côtés mesureront 30, 40, 50, ... 170, 180 pixels ;
— Une fonction peut donc avoir de nombreux paramètres ;
— Cet exercice nous fait souvenir que `range(0,5)` contient les valeurs 0, 1, 2, 3, 4, soit l'intervalle $[0\,;\,5[$; ainsi, si on veut que notre plus grand carré ait un côté de longueur `maxi`, il faut donc que la borne supérieure soit supérieure d'une longueur `pas`.
— Mais surtout : on peut utiliser une fonction dans la définition d'une autre fonction. (Cela va même plus loin puisqu'on peut définir une fonction f en se servant de cette même fonction f : très utile en math pour calculer des récurrences).

6.3 Une autre fonction : calcul d'angle réfracté

Prenons l'exemple de la réfraction ; nous souhaitons construire une fonction qui lorsqu'on lui passe en paramètres :
— l'indice n_1 du premier milieu
— l'indice n_2 du second milieu
— l'angle d'incidence i (rentré en degrés)
nous donne l'angle de réfraction r (en degrés aussi).

6.3.1 Exercice sans fonction

Pour l'instant tentez de trouver, en utilisant le module `math` de Python et ce qu'il contient (`import math` puis `dir(math)`), comment réaliser ce calcul.
Prenez des valeurs numériques pour toutes les grandeurs (mais pas pour r forcément).

UNE CORRECTION POSSIBLE

```
from math import sin, asin, degrees, radians
i = 45 # 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 et on le met dans la variable i
r = asin(n1*sin(i)/n2) # formule pour avoir r en radians
r = degrees(r) # place la valeur de r en degrés dans la variable r
print(r) # on affiche le résultat
```

On voit que le code est long, si notre programme nécessite plusieurs fois ce calcul, il faudra copier-coller le code qui occupera plusieurs lignes et deviendra illisible.

6.3.2 Une solution avec une fonction

On va définir la fonction `refraction` qui, lorsqu'on lui passe en paramètres i, n_1 et n_2 nous retourne la valeur de r :
Les fonctions utiles (sin, asin, degrees, radians) auront été au préalable importées.

code Python

```
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 va retourner 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.

Mais on voit apparaître ici un mot-clé encore inconnu : `return`. Pourtant il n'existait pas lors de notre fonction `carre`. Dès lors une question se pose :

6.4 `return` or not `return` : fonction ou procédure

Quand devrons-nous indiquer `return` ?
La réponse est la suivante :

Quand ce que retourne def . . . peut être mis dans une variable, il faut utiliser `return`, on parlera alors de **fonction**.

Dans le cas contraire, on parlera de **procédure**.

En d'autres termes, il faudra utiliser `return` si le résultat de notre fonction est **typé** (entier, flottant, chaîne de caractères, liste . . .) (référence cliquable : 3.1.2).

Par exemple : si on tape à partir des définitions de fonctions précédentes :

- `type(refraction(1,1.33,35))`, Python nous retourne `float`. Le résultat de la **fonction** est donc une variable de type flottant.
- Par contre, `type(carre(100))` renvoie une erreur : le résultat ne peut-être mis dans une variable, donc pas de type. Et on s'en doutait, on imagine mal le type de la variable qui contiendrait le dessin du carré. On parle alors de **procédure**.

Si vous n'avez pas compris les subtilités cachées derrière ces dernières lignes, ce n'est pas très grave. Souvenez-vous que si votre fonction retourne une valeur dont vous avez besoin plus tard, pour d'autres calculs par exemple, vous devez utiliser `return`.

6.5 Variables locales et globales

Je n'entrerai pas dans le détail de cette distinction ; mais signalons tout de même le problème :
Utilisons à nouveau la fonction `refraction`. Dans celle-ci, nous avons défini une variable r qui est retournée par Python. Cette variable est définie à l'intérieur de la fonction.
On ne peut se servir de cette variable en dehors de la fonction ; dans le reste du programme, Python ne la connaît pas.

Si donc par hasard, Python vous renvoie une erreur à propos d'une de vos variables (... is not defined), pensez que cela peut venir de là ; vous utilisez une variable définie dans une fonction mais pas dans le corps principal du programme.

6.6 Quelques exercices

6.6.1 De l'utilité des fonctions pour faire des maths

On considère la fonction $f : x \longmapsto f(x) = x^2 + 3$.
Construire une fonction Python prenant en paramètre l'abscisse d'un point de la courbe et retournant son ordonnée.

UNE CORRECTION POSSIBLE

```
>>> def f(x) :
        return (x**2+3)
>>> f(5) # renvoie bien 28
```

6.6.2 . . . mais aussi des maths

Écrire une fonction `syracuse(n)` qui renvoie la liste de la suite de Syracuse qu'initie n (référence cliquable : 5.2). On prend $n > 1$, notre suite s'arrêtera quand on arrive à 4,2,1.

UNE CORRECTION POSSIBLE

```
def syracuse(n) :
    S = [n]
    while n > 1 :
        if n % 2 == 0 :
            n = n / 2
        else  :
            n = 3*n + 1
        S.append(n)
    return (S)
```

`syra(5)` (ou `print(syra(5))` sur l'éditeur) renvoie bien la liste attendue.

6.6.3 . . . ou encore des maths

Définir la fonction `factorielle(n)` ($n!$ en maths) définie pour tout $n \geqslant 1$ par :
— si $n = 1$: $n! = 1$
— sinon : $n! = n \times (n-1) \times (n-2) \times \ldots \times 3 \times 2 \times 1$

Solution de base :

UNE CORRECTION POSSIBLE

```
def factorielle(n) :
    produit = 1 # on initialise ce qu'on va retourner
    for i in range(1,n+1) : # on s'arrêtera bien à n
        produit = produit * i
    return (produit)
```

Solution par récurrence :

UNE CORRECTION POSSIBLE

```
def factorielle(n) :
    if n == 1 :
        return 1
    else  :
        return (n * factorielle(n-1))
```

6.6.4 Un peu de Botanique pour changer

Les fleurs sont généralement constituées de 3, 5, 8, 13, 21, 34 ou 55 organes, par exemple : 3 pétales pour les lis , 5 pour les boutons d'or, 21 pour les chicorées, 34 ou 55 pour les marguerites, . . .

Mais, mais . . . il s'agit d'une suite de Fibonacci ! Zut, encore des Maths.

Chaque terme d'une suite de Fibonacci est la somme des 2 termes précédents.
En Maths, on écrirait :
— $u_0 = 0$ et $u_1 = 1$
— $\forall n > 1 : u_n = u_{n-1} + u_{n-2}$
Définir une fonction permettant de calculer n'importe quel terme de la suite pour n donné.

UNE CORRECTION POSSIBLE

```
def fibo(n) :
    if n == 0 :
        return 0
    elif n == 1 : # la commande elif est équivalente à else if
        return 1
    else  :
        return fibo(n-1)+fibo(n-2)
```

7 Conclusion

À l'heure où ces lignes sont écrites, nous disposons des projets de programmes de lycée en classes de Seconde et de Première. Il apparaît que les notions que les élèves connaîtront en programmation en Seconde (à travers les enseignements de Mathématiques ou de Sciences Numériques et Technologie) sont : variables, boucles, instructions conditionnelles, fonctions. En Première, avec la spécialité Mathématiques, se rajoute la notion de listes. Le programme de la spécialité Numérique et Sciences Informatiques de Première quant à lui, va fort logiquement beaucoup plus loin.

Le présent tutoriel s'attache à expliciter ces différentes notions, bases de tout langage de programmation. Le choix de présentation qui n'utilise que très peu notre champ disciplinaire a été voulu dès le départ pour trois raisons :

— d'une part pour ne pas enfermer le lecteur dans un mode de raisonnement quant à notre discipline qui est le mien et qui, dès lors, ne lui conviendra pas forcément ;
— d'autre part, les applications que je peux anticiper en Seconde comme en Première demandent déjà un certain recul sur cet outil et une certaine avancée dans l'apprentissage des bases de Python ;
— Enfin la programmation se prête remarquablement bien aux conjectures mathématiques (voir suite de Syracuse ou de Fibonacci) et cela très rapidement dans l'apprentissage.

Pour ce qui concerne notre discipline, en Seconde, l'étude de la Mécanique utilise Python ; il me semble que nous aurons besoin des listes (listes à 2 éléments pour définir les coordonnées des points ou des vecteurs par exemple ou extraction de listes à partir des données enregistrées dans un fichier `.csv` pour traitement). Cette notion n'aura pas encore été traitée par les élèves dans d'autres disciplines, ainsi, si mon hypothèse quant à leur utilisation en Mécanique est confirmée, ce sera à nous d'introduire cette notion.

Le programme de l'Enseignement Scientifique de Première demande lors du Projet Expérimental et Numérique de faire une « acquisition numérique de données » et gérer leur « traitement [et] représentation ». Python peut être un outil valable pour cette tâche. On retrouvera des fonctionnalités de Python vues dans la partie « Signaux et capteurs » de Seconde (« représenter un nuage de points [...] modéliser la caractéristique [...] à l'aide d'un langage de programmation »).

En Première, spécialité PC, les passages du Programme où on utilise Python sont assez nombreux (Chimie : tableau d'avancement, Mécanique, Ondes et Signaux). Ce sera sans doute l'occasion d'utiliser des bibliothèques "effleurées" dans ce tutoriel (Matplotlib) et aussi pourquoi pas d'utiliser des bibliothèques de visualisation 3D.

La poursuite de la formation Python en fin d'année sera dédiée à l'application de la programmation et notamment de Python dans notre discipline ; nous aurons l'occasion de faire un focus sur les activités que l'on peut proposer dans les parties du programme de Physique-Chimie proposant cet outil, mais peut-être aussi d'imaginer de nouvelles applications ailleurs dans notre enseignement.

Bibliographie, sitographie

Tous les liens ci-dessous sont cliquables.

— Une distribution Python clé en main avec tout le nécessaire déjà installé (éditeurs, installateurs de paquets, paquets classiques pour les sciences : `https://edupython.tuxfamily.org/`
— un tuto sur Python sur la base de points du programme : `http://infoforall.fr/physique/physique-accueil.html`
— le cours d'Openclassrooms sur Python en PDF : `http://user.oc-static.com/pdf/223267-apprenez-a-programmer-en-python.pdf`
— "The Holy Bible" pour Python `https://inforef.be/swi/download/apprendre_python3_5.pdf`
— le tutoriel en ligne sur l'utilisation de la bibliothèque `turtle` `https://zestedesavoir.com/tutoriels/944/a-la-decouverte-de-turtle/`
— Un lien vers une formation en ligne de l'académie de Bordeaux mêlant Python et ArduinoTM. `https://ent2d.ac-bordeaux.fr/disciplines/sciences-physiques/physique-et-chimie-computationnelle/`
— Un lien vers un PDF rassemblant beaucoup d'applications Python en Physique-Chimie en CPGE : `http://pcsi.kleber.free.fr/IPT/doc/py4phys.pdf`
— Il existe pléthore de tutoriels en ligne pour les bibliothèques les plus courantes de Python. Je ne vais pas les recenser ici, une simple recherche Internet vous permettra d'y accéder.