POUR L'ÉCOLE
DE LA CONFIANCE

KIT DE DÉMARRAGE EN LANGAGE PYTHON

Journées de l'Inspection de Physique-Chimie 2019

académie Lyon
RÉGION ACADÉMIQUE
AUVERGNE-RHÔNE-ALPES
MINISTÈRE
DE L'ÉDUCATION NATIONALE
ET DE LA JEUNESSE
MINISTÈRE
DE L'ENSEIGNEMENT SUPÉRIEUR,
DE LA RECHERCHE
ET DE L'INNOVATION

KIT DE DÉMARRAGE EN LANGAGE PYTHON

Ce kit comprend 5 documents à destination des enseignants qui débutent avec la programmation en Python et dont le but est d'animer des séances de physique-chimie en Python.

Ces documents regroupent un panorama des commandes usuelles en programmation avec ce langage, des erreurs classiques, des méthodes couramment utilisées…

SOMMAIRE

1. PYTHON POUR LE PROFESSEUR DE PHYSIQUE-CHIMIE DÉBUTANT EN PYTHON…

Cette fiche a pour but de présenter le langage python à des professeurs étant peu familiers avec la programmation et dont le but est d'animer une séance au cours de laquelle les élèves modifient un programme python. Les propos sont donc volontairement simplifiés. Les « spécialistes » voudront bien m'en excuser (O.Chaumette, auteur de cette fiche)

0. Pourquoi PYTHON ?

Python est un langage de programmation simple à apprendre, intuitif et dont le code est très lisible. Il est « interprété » (c'est-à-dire traduit en réelles instructions compréhensibles par un ordinateur) par un logiciel (python.exe). Cela rend un code python plus lent à exécuter que dans le cas d'autres langages (java, C++…) mais a peu de conséquences pour un travail en lycée.
Au fil du temps, beaucoup de « modules » externes ont été développés pour Python dans tous les domaines (mathématiques, traitement d'images, de sons, transformées de Fourier, jeux etc…). Pour faire simple, un module est un mini-programme apportant des instructions nouvelles à Python. Pour utiliser un module, il suffit de « l'importer » au début d'un de nos programmes python pour pouvoir utiliser ses nouvelles instructions (inutile de savoir comment elles ont été développées !). L'existence de ces modules est un des gros avantages de Python.
Pour finir, Python est un langage qui peut tourner sur tous les systèmes d'exploitation (Windows, Mac OS X, Linux, Android…).

Python a bien quelques inconvénients tout de même. Comme dit précédemment, étant interprété, il est plutôt lent à s'exécuter. Cela n'est pas grave pour une utilisation scientifique mais on ne pourrait pas développer des jeux 3D modernes avec. Il est également peu commode de créer des fichiers « exécutables » autonomes. C'est-à-dire que pour exécuter un programme Python, il faut avoir Python installé sur son ordinateur.

1. Installation de PYTHON

Les collègues pressés pourront directement lire "l'essentiel" de la fin de ce paragraphe.

Quelle version de Python utiliser : la 2 ou la 3 ?

Il coexiste 2 versions de python : les versions 2 et 3. Elles diffèrent sur quelques points (en particulier l'utilisation de l'instruction « print » qui devient une fonction dans la version 3). Si pendant longtemps les modules complémentaires n'étaient compatibles qu'avec la version 2, il s'avère aujourd'hui (en 2019) que beaucoup ne sont mis à jour que pour la version 3 (en particulier numpy qui est LE module le plus utilisé pour les calculs scientifiques).
En conclusion, *il faut travailler en version 3.x* (version 3.7 en 2019. La version 2 sera obsolète le 31/12/2019).

IMPORTANT : on trouve sur internet de nombreux programmes développés en version 2 et qui pourraient ne pas fonctionner en version 3. Vous trouverez, en annexe de cette fiche, les plus fréquentes des modifications à apporter au code version 2 pour le faire tourner en version 3 (parfois, il suffit juste de changer une minuscule en majuscule !)

Python en version « Installable » ou en version « Portable » ?

La version « Installable » est celle téléchargée sur le site officiel et qu'il faut installer sur son ordinateur. Elle a l'avantage d'être toujours à jour. Mais les modules externes utiles (en particulier Matplotlib et Numpy pour les calculs et tracés de courbes mathématiques, Pygame pour faire des jeux) ne sont pas fournis avec. Il faut les installer à part (procédure peu évidente). Elle est disponible pour Windows, Mac OS, Linux, Android…
PRATIQUE : Il existe des packs tout faits pour faciliter l'installation de python et certains modules complémentaires : **Anaconda** est celui préconisé (le plus à jour actuellement, le plus maintenu et bénéficiant de la plus grande communauté)..

La version « Portable » est une version que l'on peut exécuter depuis une clé USB. Ses très gros avantages sont :
- de n'installer aucun fichier sur l'ordinateur hôte
- d'avoir toujours avec soi Python sur une clé USB
- de posséder déjà tous les modules essentiels (Tkinter, Matplotlib et Numpy, pygame …).

Pour information, cette version est utilisée par de nombreux professeurs d'ISN (Informatique et Sciences du Numériques) avec les élèves. Elle est conseillée si aucune autre possibilité n'est disponible sur le réseau de l'établissement.
ATTENTION, elle n'est disponible que pour Windows (Pour les personnes utilisant MacOS ou Linux, il faut la lancer à partir d'un émulateur Windows, « Wine » par exemple).

Télécharger Python

VERSION « CLASSIQUE » installable : (Windows, Linux, Mac OS, Android))

Le site officiel est : https://www.python.org/. Dans le menu **download** du site, choisir le système d'exploitation (Windows, Mac OS X etc…). Pour Windows, choisir Windows x86 executable installer si vous ne savez pas si votre système d'exploitation tourne sous 64 bits (sinon choisir Windows x86-64 executable installer). Attention, l'installation des modules complémentaires doit se faire à l'aide de la commande 'pip'.

VERSION "PACK" : pour chez soi ou pour les administrateurs réseau (Windows, Linux, Mac OS)
Le site d'Anaconda est : https://www.anaconda.com/distribution/
Il faut ensuite compléter l'installation pour installer pygame ou opencv.

VERSION « PORTABLE » : pour clé USB (Windows uniquement ou via l'émulateur Wine pour MacOS ou Linux)
Il en existe plusieurs (rien n'est simple !). Voici les 3 principales :
- **Portable Python 3.2 :** c'est la version « historique ». Très bien faite, elle n'est malheureusement plus mise à jour et ne contient pas pygame. Elle est encore beaucoup utilisée dans les lycées et on peut la télécharger ici : http://portablepython.com/wiki/Download/

- **WinPython :** c'est la version « portable » la plus complète à ce jour en termes de modules. Elle contient les modules scientifiques ainsi qu'un très grand nombre de modules permettant de faire de la programmation dans tous les domaines. En revanche, elle ne contient pas l'éditeur pyscripter (qui est celui préconisé dans cette fiche). On peut télécharger WinPython ici : http://winpython.github.io/

- **Edupython :** c'est une version développée (à partir de la version portable 3.2 et d'une version appelée « AmiensPython ») par Vincent MAILLE, professeur de mathématiques de l'Académie d'Amiens. Cette version contient tous les modules importants pour le travail au lycée (Matplotlib et Numpy, pygame…), l'éditeur simple « Pyscripter » ainsi qu'un module développé spécialement pour le lycée (francisant et simplifiant certaines instructions du python). C'est la version fortement conseillée qui a le mérite d'être clé en main pour l'utilisation au lycée. On peut la télécharger ici : https://edupython.tuxfamily.org/. Une fois téléchargé, il suffit d'exécuter « SetupEP26.exe » (ou bien extraire le contenu de « EP26.zip » dans un dossier de sa clé USB ou son ordinateur). Un grand merci à Vincent Maille et ses collègues pour leur travail.

L'essentiel

Au choix pour utiliser Python :

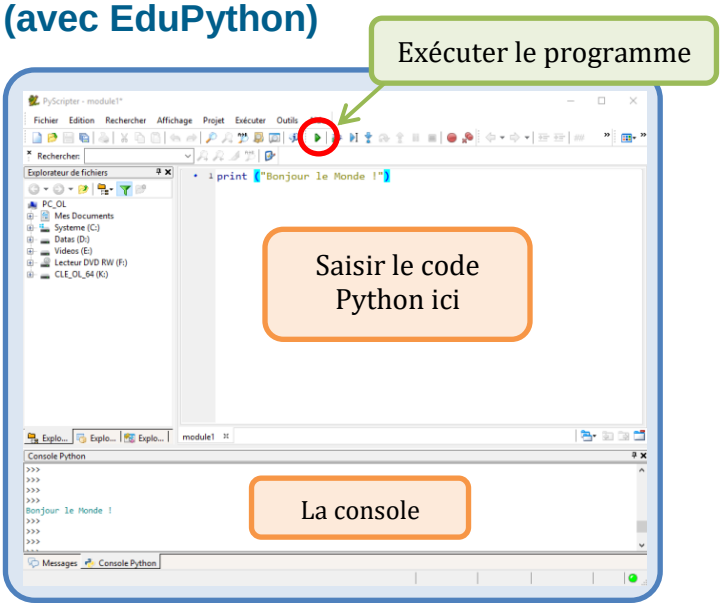
Version « Portable » : Fortement conseillée (et peut-être donnée aux élèves pour un usage « à la maison »)
EduPython peut être copié sur une clé USB et exécuté sans écrire de données sur l'ordinateur hôte. Complet, disposant des modules importants, il dispose également d'une version de pyscripter. Disponible pour Windows uniquement.
http://download.tuxfamily.org/edupython/Setup_EP26.exe

Version installable préconisée : (Windows, Linux, Mac OS)
Anaconda : https://www.anaconda.com/distribution/

2. Taper et exécuter un programme python (avec EduPython)

L'essentiel

1. Lancer **Py-scripter.exe** ou bien, si on utilise EduPython : **Edupython.exe**
2. Charger un programme (***Fichier >> Ouvrir***) ou bien taper le code dans la partie droite de Py-scripter
3. Lancer le programme en cliquant sur l'icône verte : ▶ (ou bien les touches Ctrl+F9)
4. Le résultat apparaît souvent dans la **console**, en bas de l'écran de Py-scripter



Remarque : il existe d'autres logiciels permettant de saisir du code Python (des IDE : eclipse, wing IDE…). **Py-scripter** a le mérite d'être simple à utiliser et fourni avec la distribution python de base (ou Edupython). **Spyder** est l'éditeur d'Anaconda.
Un simple éditeur de texte peut aussi suffire (**Bloc-Note** ou **Notepad++** par exemple)

Pyscripter n'est pas « beau » ?

Si le fond noir par défaut de Pyscripter ne vous convient pas, il est possible de changer l'aspect de Pyscripter par le menu « Affichage » puis « Style »

3. Structure d'un programme Python : quelques bases

Ce paragraphe a pour but de vous familiariser avec la structure d'un programme python. En revanche, il ne donne pas les bases du langage Python.

Pour rappel, dans un programme, on écrit une suite d'instructions (ce sont des ordres donnés à l'ordinateur).
Ces instructions utilisent des mots-clés propres à chaque langage de programmation et respectent une sorte de grammaire. Je renvoie à des formations Python pour connaitre cette grammaire et les mots clés. Des mémentos regroupant les mots clés les plus courants sont faciles à trouver sur internet.
En python, il est indispensable de maîtriser la notion d'**indentation** (c'est-à-dire décalage de l'écriture des instructions)

Voici un exemple classique (et totalement inutile !) de programme Python :

```
# Prog développé par Maître Yoda pour la classe

# Importation des modules
import tkinter
from mathplotlib import *

# Fonctions utilisées
def Affiche_Bonjour():
    print ("Bonjour")

# initialisation des variables
Ma_Variable = int(input("Entrez une valeur"))
Mon_Texte =""

# Programme principal
if Ma_Variable == 0 :
    print("Votre variable est nulle")
    Ma_Variable = 1
    print("Hello World")
    Affiche_Bonjour()
    print("Olivier")
else :
    print("Votre variable n'est pas nulle")
    Ma_Variable = 0
    Affiche_Bonjour()
    print("le monde")
    """ chers élèves, vous pouvez modifier
    les lignes suivantes """
    print("Mon prénom est Olivier")
    Affiche_Bonjour()
    print("Olivier")
    """ fin des modifications
    faites par les élèves """

print("Salut")

for i in range(0,4) :
    print i
    if i ==2 :
        print("i vaut 2")
        print ("et c'est très bien !")
    print ("La boucle tourne !")

print("On arrête le programme ici ")
print("Au revoir")
```

En bleu foncé, les mots clés du Python.

« def » définit une fonction

« if » permet d'écrire une condition

« print » affiche quelque chose dans la console

« for » créé une boucle. Ici « i » est le compteur

Les **commentaires** sont précédés d'un « dièse ». Ils sont là pour décrire une partie du programme. Ils sont inutiles pour l'exécution du programme.

Avant un « bloc » (donc à la suite d'un « if », d'un « else », d'un « for », d'un while ») **ne pas oublier les deux points**

« Bloc » d'instructions à réaliser si « Ma_Variable » vaut zéro. Les instructions du bloc doivent toutes être alignées. En python, on dit qu'elles sont **indentées**. Si une instruction n'est pas indentée correctement, elle ne fera pas partie du bloc (et indiquera la fin du bloc « if »)

« Bloc » d'instructions à réaliser si « Ma_Variable » ne vaut pas zéro (« else »=sinon). Les instructions correspondantes sont indentées.

Les **commentaires** sur plusieurs lignes sont encadrés par des triples guillemets. En rose, ils sont visibles et permettent de donner des consignes aux élèves au sein du programme.

Cette instruction ne fait pas partie du bloc « else » car elle n'est pas indentée

Bloc d'instructions de la **boucle** « for ».

Bloc d'instructions du « if ». Elles sont indentées par rapport au bloc du « for »

Cette instruction ne fait pas partie du bloc « if » mais fait partie du bloc « for »

Ces instructions ne font ni partie du bloc « if », ni du bloc « for »

Le « **égal** » d'une condition s'écrit avec un double signe égal : « **==** » (et « **différent** » avec «**!=**»)

4. Animation d'une séance en Python : les erreurs fréquentes

*Les élèves étant, dans l'ensemble, peu familiarisés avec Python, ils commettent des erreurs qui empêchent leur programme de fonctionner. Voici une liste **non exhaustive** des erreurs fréquemment rencontrées :*

Erreurs d'indentation : (la ligne est souvent précédée d'un soulignement en rouge)
- la ligne d'instruction n'est pas alignée correctement sur un bloc ou sur la racine

Erreurs de syntaxe : (l'instruction est souvent soulignée en rouge)
- le mot clé n'existe pas ou 2 lettres ont été inversées (« imput » au lieu de « input »)
- la casse d'un mot clé n'est pas respectée (« Print » au lieu de « print »)
- une parenthèse a été ouverte mais non refermée. Idem pour des guillemets.

Erreurs de variables :
- le nom de la variable ne doit pas contenir d'accents ou d'espace
- la casse du nom de variable doit être respectée (« A » est une variable différente de « a »)
- la variable ne doit pas avoir le nom d'un mot clé (ne pas appeler une variable « print »)
- une variable est de type « entier » et on veut l'afficher comme un texte : il manque «str » devant la variable.*
- une variable est de type « texte » et on veut l'additionner à un chiffre : il manque «int » devant la variable.*

** solution volontairement simplifiée : il faut se plonger dans la structure des variables en python pour aller plus loin*

Erreurs de conditions :
- oubli des 2 points à la fin de la ligne if ou else
- Pour une condition « égale », il faut mettre un double signe « égal »
- erreur d'indentation des instructions du bloc if ou else

Erreurs de boucles « for » :
- oubli des 2 points à la fin de la ligne
- erreur d'indentation des instructions du bloc « for »
- la valeur de fin du compteur de boucle n'est pas bonne : il faut penser que le « **in range(debut, fin)** » fait varier le compteur de boucle de début à fin-1.*(solution volontairement simplifiée : il faut se plonger dans la structure des variables en python pour aller plus loin)*

Erreurs de boucles « while » («tant que »):
- oubli des 2 points à la fin de la ligne
- erreur d'indentation des instructions du bloc « while »
- la condition du while n'a pas été modifiée dans le bloc donc la boucle tourne à l'infini…

Erreurs de listes (ou tableaux) : Le célèbre « out of range »
- si un index dépasse le nombre d'élément d'un tableau, l'errer « out of range » apparaît : il faut vérifier les bornes si on utilise une boucle ou bien l'index utilisé.

Exemple : si une liste « L » contient 4 élements, « print L[8] » renverra l'erreur « out of range ».

5. Annexe : adaptations fréquentes à réaliser pour passer de Python 2 à 3

1. « print » devient une fonction : Concrètement, cela signifie qu'il faudra systématiquement mettre des parenthèses après print en Python 3 ce qui n'était pas nécessaire en version 2

Exemple : en Python 2 : `print "bonjour"` devient en Python 3 : `print("bonjour")`

2. « input » renvoie systématiquement un texte : le `raw_input` du Python 2x n'existe plus. Pour saisir un **nombre**, il faut maintenant taper : `int(input("Question"))` et pour saisir un **texte**, il faut taper : `input("Question")`

3. Tkinter perd sa majuscule : `import Tkinter` (en Python 2) devient : `import tkinter` (en Python 3)

4. Les objets de Tkinter sont renommés : par exemple, `tkMessageBox` (en python 2) devient `tkinter.messagebox`
`tkFont` (en python 2) devient `tkinter.font` etc…

Plus de détails sur les correspondances Python 2-Python 3 ici : http://apprendre-python.com/page-syntaxe-differente-python2-python3-python-differences

6. Annexe 2 : indications pour le référent TICE de l'établissement

Solution SCRIBE : un paquet wpkg avec python 3 a été développé. Les modules supplémentaires sont à télécharger et installer par des "wheel". (pygame, opencv etc.)

Solution CIBLE : lors de l'édition des images, faire installer anaconda. Vérifier la présence des modules pygame et opencv. Si pas disponibles, les faire installer : **"conda install -c cogsci pygame"** dans la console Anaconda.

Si la solution CIBLE est déjà installée : extraire un exécutable **Edupython** sur un partage réseau. (*Version de la fiche :* 2 de mars 2019)

2. PYTHON : L'ESSENTIEL POUR COMMENCER À PROGRAMMER…

Cette fiche a pour but de présenter les bases de la programmation appliquées au langage python à des professeurs peu familiers avec la programmation et dont le but est d'animer une séance au cours de laquelle les élèves modifient un programme python.

1. Types de variable

VARIABLE

Une **variable** est une sorte de « case » dans laquelle un programme stocke une donnée.
Une variable peut contenir un nombre, un texte, une liste et bien d'autres choses.

En Python, il existe plusieurs types de variables. Les plus fréquentes sont

int : variable de type entier relatif
float : variable de type réel
str : variable de type texte

En python, contrairement à de nombreux langages, **il n'est pas nécessaire d'indiquer le type d'une variable avant de l'utiliser**. Il faut cependant que le programmeur l'ait en tête. C'est une des difficultés des élèves.

Exemple de déclaration de variables :

Algorithme en langage naturel	Equivalent en Python
On met le nombre 28 dans la variable appelée « A » On met le texte « Bonjour » dans la variable appelée « Texte »	A = 28 Texte = "Bonjour"

2. Bases de programmation

INSTRUCTIONS

Une **instruction** est un ordre donné à l'ordinateur. Un **programme** est un ensemble d'instructions.
Une instruction ne renvoie pas de valeur mais exécute une action.

Exemples :

Algorithme en langage naturel	**Instructions en Python**	**Résultat à l'écran**

Afficher le texte « bonjour »
Afficher le nombre 28
Additionner 2 et 4
 	
Additionner 2 et 4 et mettre le résultat dans la variable A
Afficher la valeur de A
	`print ("Bonjour")`
	`print (28)`
	`2+4`
	
	`A = 2+4`
	`print(A)`
		`Bonjour`
		`28`
		(rien)
		
		(rien)
		`6`

Le programme a bien additionné les 2 chiffres mais comme on ne lui a pas demandé d'afficher le résultat, on ne voit rien…

BOUCLES

Une **boucle** permet de répéter plusieurs fois une suite d'instructions (appelée **bloc**).
Il existe deux types de boucles : les boucles **for*** et les boucles **while****
Les boucles **for** sont utilisées quand on connaît le nombre de fois que l'on doit répéter le bloc.
Les boucles **while** sont utilisées dans le cas contraire.

* **for** signifie **pour** ** **while** signifie **tant que**

*Exemple de boucle **for** simple :*

Algorithme en langage naturel	**Instructions en Python**	**Résultat à l'écran**

Mettre la valeur 0 dans la variable A
Faire 10 fois :
Ajouter 1 à A et remettre cette valeur dans A
Afficher A
	`A=0`
	`for i in range(10):`
	`A=A+1`
	`print (A)`
		`1`
		`2`
		`3`
		(Etc)…
		`9`

Ces 2 instructions forment un bloc. Les instructions d'un même bloc doivent être décalées par rapport au « for » et alignées. On dit que ces instructions sont **indentées** (=décalées)

*Exemple de boucle **for** utilisant une variable compteur de boucle :*

Algorithme en langage naturel	Instructions en Python	Résultat à l'écran
Pour la variable i prenant des valeurs de 1 à 9 : Afficher i	`for i in range(1,10):` `  print (i)`	1 2 *(Etc)…* 9

Dans une boucle **for**, le **compteur de boucle** (ici la variable i) prend des valeurs jusqu'à la borne supérieure de « range » **-1**. On peut aussi rajouter un pas d'itération (exemple : for i in range (1,10,2) pour compter de 2 en 2)

*Exemple de boucle **while** :*

Algorithme en langage naturel	Instructions en Python	Résultat à l'écran

Mettre la valeur 0 dans A
Tant que A est inférieur à 9 :
Ajouter 1 à A et remettre cette valeur dans A
Afficher A
	`A=0`
	`while A<9:`
	`  A= A+1`
	`  print (A)`
		1
		2
		(Etc)…
		9

Bloc de la boucle **while**

CONDITIONS

Une **condition** permet d'exécuter un bloc d'instructions en fonction du résultat (VRAI ou FAUX) d'une comparaison.
Les conditions commencent par **if***.
Si la condition n'est pas respectée (Résultat FAUX), il est possible d'exécuter un autre bloc en utilisant **else****.

* **if** signifie **si** ** **else** signifie **sinon**

OPÉRATEURS de COMPARAISON

Algorithme en langage naturel	Instruction en Python
Si A = B	`if A==B:`
Si A différent de B	`if A!=B:`
	`if not (A==B):`
Si A > B	`if A>B:`
Si A supérieur ou égal à B	`if A>=B:`
Si A compris entre 0 et 2	`if 0<A<2:`
Si A = 2 et B = 4	`if A==2 and B==4:`
Si A = 2 ou B = 4	`if A==2 or B==4:`

Attention au double « égal »

Dans tous les cas, bien penser à mettre les **deux points**

L'instruction **input** permet à l'utilisateur du programme d'entrer un texte. Pour transformer ce texte en entier, on utilise **int**.

Exemple de condition :

Algorithme en langage naturel	Instruction en Python

Demander une valeur entière à l'utilisateur et la mettre dans A
Si A = 2 alors :
Afficher A
Afficher (« A vaut bien 2 »)
Sinon :
Afficher A
Afficher (« A ne vaut pas 2 »)
	`A=int(input("Entrez A"))`
	`if A==2 :`
	`  print (A)`
	`  print("A vaut 2")`
	`else :`
	`  print (A)`
	`  print("A ne vaut pas 2")`

Bien indenter (= décaler et aligner) les instructions de chaque bloc

3. Les listes en Python

Une **liste** est un tableau contenant des données (de type int ou float ou str).
Une liste peut avoir 1, 2 ou 3 dimensions (on a alors un tableau à 1,2,3 dimensions)

Une liste se définit comme une variable à laquelle on assigne les éléments de la liste entre crochets et séparés par des virgules. Le 1er élément de la liste se trouve à la position (appelée index) : 0

Quelques exemples de manipulation de listes et d'instructions utiles:

Algorithme en langage naturel	**Instructions en Python**	**Résultat à l'écran**
Mettre dans une liste appelée X les valeurs 4,2,6,8	`X=[0,2,6,8]`	*(rien)*
Afficher le 1er élément de la liste (donc 4)	`print (X[0])`	4 *(1ère position : index 0)*
Afficher le 2nd élément de la liste (donc 2)	`print (X[1])`	2
Afficher le dernier élément de la liste (donc 8)	`print (X[-1])`	8
Mettre dans la variable L le nombre d'éléments de X Afficher L	`L=len(X)` `print (L)`	*(rien)* 4
Ajouter 12 à la liste	`X.append(12)`	*(rien)*
Afficher la liste X	`print (X)`	`[0,2,6,8,12]`
Afficher la position de 6 dans la liste X	`print (X.index(6))`	2 *(donc 3ème position)*
Créer une liste Y à 2 dimensions avec : 2 5 8 4 1 3	`Y=[[2,5,8],` `  [4,1,3]]`	
Afficher l'élément de Y à la 1ère ligne et 2nde colonne	`print (Y[0][1])`	5

Numéro de ligne (commençant à 0)

Numéro de colonne (commençant à 0)

Remarque : on rencontre, dans certains programmes, des instructions du type : T = (1,5,"bonjour", 28).
Il ne s'agit pas d'une définition de liste mais de **tuple**. Pour faire simple, un tuple est une liste non modifiable pouvant contenir des données de type différent. On les utilise rarement dans les programmes simples.

4. Les modules (ou bibliothèques) Python

Les **modules** sont des programmes qui rajoutent des fonctionnalités (donc des instructions) à Python.
Il n'est pas nécessaire de savoir comment sont programmées les fonctions contenues dans un module. Il suffit juste de savoir les utiliser (chaque module possède une notice listant ses fonctions et comment les utiliser).

Il existe un très grand nombre de modules qui permettent, par exemple :
De créer une interface graphique (module *tkinter*), d'utiliser des fonctions mathématiques (module *math*), de tracer des graphiques (module *matplotlib*), de faire des calculs (module *numpy*), de jouer des sons (module *wav*) etc…

Pour utiliser les fonctions d'un module dans Python, il faut **importer le module** au début de notre programme.

Exemples :

`import math`	On importe tout le module **math** et ses fonctions
`from math import *`	Autre manière d'importer toutes les fonctions de **math**
`from math import sin(),cos()`	On importe seulement sin() et cos() du module **math.**
`import numpy as np`	On importe tout le module **numpy** sous l'**alias** (c'est-à-dire le nom) « **np** »

Remarque : Certaines fonctions sont présentes dans plusieurs modules (comme sinus ou cosinus qui existent dans les modules math et numpy). Pour préciser quel module on utilise, on met son nom (ou son alias) devant la fonction.

Exemple :

`print(math.sin(0.2))`	Python utilisera la fonction sinus du module math
`print(np.sin(0.2))`	Python utilisera sinus du module numpy (que l'on a importé sous l'alias np)

5. Pour aller plus loin…

Sitographie utile : https://python.developpez.com/cours/apprendre-python3/

Version 1 du document – janvier 2019

3. PYTHON : QUELQUES MÉTHODES UTILES À RETENIR

Cette fiche a pour but de présenter quelques réponses à des problèmes classiques de programmation Python pour les professeurs animant des activités de physique-chimie. Le mot « méthode » a un sens bien précis en programmation orientée objet mais il est employé ici dans son sens commun (et non orienté objet).

Chaque code ci-dessous est suivi d'un commentaire (respectant la syntaxe Python, c'est-à-dire précédé d'un dièse)

CONVERTIR DES TYPES DE VARIABLES

```
T = str(N)                          # transforme un entier (int) ou réel (float) N en un texte (string)
N = int(T)   ou   N = float(T)      # transforme un texte T (string)  en entier (int) ou réel (float)
```

AFFICHER AU SEIN D'UNE MÊME PHRASE DES VARIABLES « texte » (string) et « nombre » (int, float)

```
A = 28              # A est une variable entière (int)
T ="bonjour"        # T est une variable texte (string)
print (T,A)         # bonjour (string) et 28 (int) seront écrit sur la même ligne séparés par un espace: bonjour 28
```

ou bien

```
A = 28                  # A est une variable entière (int)
T ="bonjour"            # T est une variable texte (string)
print (T+str(A))        # bonjour et 28 seront écrit sur la même ligne sans espace : bonjour28
```

DEMANDER UN NOMBRE ou UN TEXTE À L'UTILISATEUR

On est amené souvent à demander à l'utilisateur du programme de saisir une valeur (qui sera stockée dans une variable). L'instruction à utiliser diffère un peu selon que l'on veut demander un texte, un entier ou un réel.

```
T = input("Entrez un texte")                    # la variable T sera de type texte (string)
N = int(input("Entrez un nombre entier"))       # la variable N sera de type entier (int)
R = float(input("Entrez un nombre réel"))       # la variable R sera de type réel (float)
```

DEMANDER UNE VALEUR À L'UTILISATEUR TANT QU'ELLE NE CONVIENT PAS

On veut parfois demander à l'utilisateur un nombre (un numéro atomique par exemple). Si Z n'est pas compris entre 0 et 18, il faut redemander le nombre. Voici la méthode à utiliser :

```
Z = 0                                   # toujours initialiser Z de manière à ce que la boucle « while » se fasse
while not (0<Z<=18):                                                # tant que Z n'est pas entre 1 et 18
      Z = int(input("Entrez un Z compris entre 1 et 18"))           # on (re)demande Z
```

INVERSER LE CONTENU DE 2 VARIABLES

Dans la plupart des langages de programmation, pour inverser le contenu de A et B, il faut taper :

```
TEMP = A        # on met dans une variable temporaire le contenu de A
A = B           # son contenu étant sauvé dans TEMP, A peut prendre la valeur de B
B = TEMP        # on met dans B la valeur de TEMP (l'ancien contenu de A)
```

En Python, on peut faire encore plus simple :

```
B,A = A,B
```

PARCOURIR UNE LISTE

```
L = [2,4,6,8]                   # L est une liste d'entiers
for i in range (len(L)) :       # i va prendre la valeur de l'index de chaque élément (0 puis 1 puis 2..)
      print(L[i])               # on affiche l'élément à l'index « i » (par exemple)
```

ou bien

```
L = [2,4,6,8]                    # L est une liste d'entiers
for N in L:                      # N va prendre la valeur de chaque élément de la liste (2 puis 4 puis 6…)
      print(N)                   # on affiche l'élément (par exemple)
```

TESTER RAPIDEMENT LA PRÉSENCE D'UN ÉLÉMENT DANS UNE LISTE

```
L = ["bonjour","salut","hello"]          # L est une liste de textes (string)
MOT = "Ciao"                             # MOT est le texte à tester (string)
if MOT in L:                             # Teste si MOT est contenu dans L
      print("le mot est présent dans L")
```

ou bien

```
if not (MOT in L):                       # Teste s'il n'y a pas MOT dans la liste L
      print("le mot n'est pas présent dans L")
```

AFFICHER LES ELEMENTS D'UNE LISTE SANS SAUT À LA LIGNE

Pour afficher les éléments d'une liste « L », il faut éviter d'écrire : print(L)
Voici quelques méthodes pour afficher proprement à la suite les éléments d'une liste pour tout type de variable :

```
L = [2,4,6,8]                    # On crée une liste
for N in L:                      # N va prendre la valeur de chaque élément de la liste (2 puis 4 puis 6…)
      print(N, end=' ')          # affichage de N et d'un espace (celui après « end ») sans retour à la ligne
                                 # Résultat : 2 4 6 8
```

```
Phrase = ["Bonjour","le","monde"]    # On crée une liste
for Mot in Phrase:                   # Mot va prendre chaque élément de la liste (Bonjour puis le ….)
      print(Mot, end=',')            # affichage de Mot et d'une virgule (celle après « end ») sans retour à la ligne
                                     # Résultat : Bonjour,le,monde
```

Attention, les prochains « print » seront encore écrits à la suite. Pour sauter une ligne, taper `print()` *(sans rien)*

Pour écrire les éléments à l'envers (ou mettre dans une variable texte tous les éléments) :

```
Texte=""                             # On crée une variable de type texte vide (double guillemets)
Phrase = ["Bonjour","le","monde"]    # On crée une liste
for Mot in Phrase:                   # Mot va prendre chaque élément de la liste (Bonjour puis le ….)
      Texte = Mot + " "+ Texte       # A chaque tour de boucle, Texte est modifié: on place Mot devant lui
print(Texte)                         # affichage de la variable Texte
                                     # Résultat : monde le Bonjour
```

L'ARRONDI ET L'ÉCRITURE SCIENTIFIQUE en PYTHON

Pour écrire un résultat en écriture scientifique, il faut taper `"%.e"%` *suivi du nom de la variable (ou* `"%.E"%`*)*

```
N = 1988412
print("%e"%N)                    # va afficher 1.988412e+06
print("%E"%N)                    # va afficher 1.988412E+06
```

Pour écrire un résultat en écriture scientifique avec n chiffres significatifs, il faut taper `"%.(n-1)e"%` *suivi du nom de la variable (ou* `"%.(n-1) E"%`*) (en fait, n-1 représente le nombre de décimales)*

```
N = 1988412
print("%.1e"%N)                  # va afficher 2.0e+06    soit 2 chiffres significatifs (CS)
print("%.1E"%N)                  # va afficher 2.0E+06    soit 2 CS
print("%.2e"%N)                  # va afficher 1.99e+06    soit 3 CS
print("%.3e"%N)                  # va afficher 1.998e+06    soit 4 CS
```

4. PYTHON ET LE MODULE NUMPY

Cette fiche présente les bases de Numpy, module qui permet de faire des calculs et créer des tableaux utilisables avec Matplotlib.

IMPORTER NUMPY

```
import numpy as np
```

« **Alias** » (sorte de surnom) de numpy dans le programme : tous les objets ou fonctions de numpy (sinus, cosinus, tableaux…) commenceront par « **np** » pour indiquer qu'ils relèvent du module numpy

FONCTIONS COURANTES DE NUMPY

Fonctions trigonométriques			Fonctions trigonométriques inverses			π	$\sqrt{x}$
`np.sin(x)`	`np.cos(x)`	`np.tan(x)`	`np.arcsin(x)`	`np.arccos(x)`	`np.arctan(x)`	`np.pi`	`np.sqrt(x)`
ln(x) et e^x	log(x) et 10^x	Val absolue	Signe (renvoie 1 si positif et -1 si négatif)		arrondi à n décimales		y^x
`np.log(x)` `np.exp(x)`	`np.log10(x)` `10**x`	`np.abs(x)`	`np.sign(x)`		`np.around(x,n)`		`y**x`

Exemple:

Algorithme en langage naturel	Instructions en Python	Résultat à l'écran
Afficher la valeur de pi Afficher le sinus de 3.10^{-2} rad	`print(np.pi)` `print(np.sin(3e-2))`	`3.14159265…` `0.029995500`

Création d'un tableau Numpy (à 1 dimension) « à la main »

Cette fonction permet de créer un tableau (= une liste) à une dimension d'éléments que l'on saisit « à la main » :

`T = np.array([])`	créé un tableau vide
`T = np.array([`*valeur1*`,` *valeur2*`,` *valeur3* `,` *etc…*`])`	créé un tableau contenant *valeur1, valeur2, etc…*

Insertion et suppression d'éléments dans un tableau

`T = np.append(T,` *Element*`)`	ajoute *Elément* à la fin du tableau.
`T = np.delete(T,` *index*`)`	Supprime l'élément situé à la position *index* (1er : 0 ; dernier : len(T)-1)

La fonction LINSPACE()

Cette fonction permet d'obtenir simplement un tableau (= une liste) à une dimension **d'un nombre choisi de valeurs** :

`T = np.linspace(`*début*`,`*fin*`,`*nbre_val*`)`	créé une liste de *nbre_val* valeurs entre *début* et *fin* (inclus)

*Exemple de tableau créé avec **linspace()** :*

Algorithme en langage naturel	Instructions en Python	Résultat à l'écran

Créer un tableau de 7 valeurs entre 1 et 4
Afficher ce tableau
Afficher le 1er élément de ce tableau
Afficher le dernier élément de ce tableau
	`T = np.linspace(1,4,7)`
	`print(T)`
	`print(T[0])`
	`print(T[-1])`
		(rien, création)
		`[1 1.5 2 2.5 3 3.5 4]`
		`1.0`
		`4.0`

La fonction ARANGE()

Cette fonction permet d'obtenir une liste d'éléments entre deux bornes. Elle est équivalente à la fonction range() de Python.

`Liste = np.arange(`*début*`,`*fin*`,`*pas*`)`	crée une liste d'int de *début* et ***fin-1*** par pas de *pas*

*Exemple de tableau créé avec **arange()** :*

Algorithme en langage naturel	Instructions en Python	Résultat à l'écran

Créer un tableau entre 1 et 7 par pas de 2
Afficher ce tableau
Créer un tableau entre 1 et 7 par pas de 0,5
Afficher ce tableau
	`T = np.arange(1,8,2)`
	`print(T)`
	`T2 = np.arange(1,8,0.5)`
	`print(T2)`
		(rien, création)
		`[1 3 5 7]`
		(rien, création)
		`[1 1.5 2 2.5 … 7]`

Pour mieux comprendre : les tableaux de NUMPY et la différence LINSPACE/ARANGE

Les listes de Numpy semblent faire doublon avec les listes de Python. **Il est préférable d'utiliser les tableaux Numpy** si l'on souhaite ensuite tracer des courbes avec Matplotlib et créer des tableaux à partir d'autres tableaux.

Exemple : si « **t** » est un tableau, l'instruction : **x=2*t** va créer un tableau « x » dont chaque élément vaut le double de ceux de t.

En résumé, pour créer des tableaux Numpy, on peut utiliser :

- soit **linspace()** quand on veut **un nombre choisi de valeurs** réelles entre 2 bornes
- soit **arange()** quand on veut un nombre de valeurs entre 2 bornes **en choisissant un pas d'itération**
- soit **array()** si on a des valeurs expérimentales à saisir « à la main »
- soit écrire : **Nouveau_Tableau = fonction d'un Tableau_existant**

Version 2 du document – mars 2019

5. PYTHON ET LE MODULE MATPLOTLIB

Cette fiche a pour but de présenter les bases de Matplotlib, module qui permet de tracer des courbes avec Python.

IMPORTER MATPLOTLIB

```
import matplotlib.pyplot as plt
```

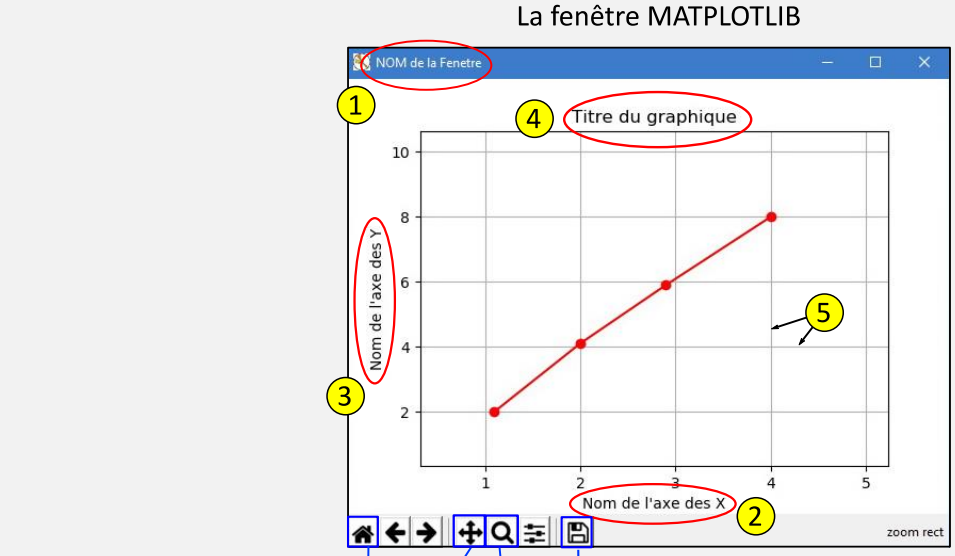
« **Alias** » de MATPLOTLIB

Création d'une fenêtre MATPLOTLIB

`plt.figure("NOM de la Fenetre")` créé une fenêtre dont le nom est *«Nom de la fenêtre» :* voir 1 ci-dessous

Remarque : cette fenêtre ne s'affichera uniquement lorsque l'on aura saisi, à la fin de tout le code : `plt.show()`

La fenêtre MATPLOTLIB



Configuration des axes et du graphique

Toutes les configurations sont **optionnelles** (MATPLOTLIB tracera les courbes même si les lignes ci-dessous ne sont pas présentes)

`plt.xlabel("NOM de l'axe des X")`	Donne un nom à l'axe des abscisses (ici : `"NOM de l'axe des X"`) voir 2
`plt.ylabel("NOM de l'axe des Y")`	Donne un nom à l'axe des ordonnées (ici : `"NOM de l'axe des Y"`) voir 3
`plt.axis([Xmin, Xmax, Ymin, Ymax])`	Définit les valeurs min et max des abscisses et ordonnées

`plt.title("TITRE du graphique")`	Donne un titre au graphique (ici : `" TITRE du graphique "`) voir 4
`plt.grid()`	Affiche la grille : voir 5

Tracé de points et segments

`plt.plot(X, Y, style)` Place un point en ***X*** et ***Y*** avec un certain ***style***. X et Y peuvent être des tableaux Numpy

`plt.plot([X1,Y1],[X2,Y2],style)` Trace un segment entre les points de coordonnées (***X1,Y1***) et (***X2,Y2***)

Styles disponibles : à placer entre guillemets *dans l'ordre* : **1. Couleur** **2. Type de point** **3. Tracé**

Couleurs						Type de points tracés					Tracé	
r	b	g	k	m	c	o	.	x	+	v	-	--
Rouge	Bleu	vert	noir	magenta	cyan	Gros point	Petit point	Croix	Croix +	Triangle	Points reliés	Points reliés en pointillé

Exemples : `"rx-"` affiche des croix, rouges et reliées `"ko"` affiche des gros points, noirs et non reliés

Plus d'informations et d'options de tracé à l'adresse : https://matplotlib.org/api/_as_gen/matplotlib.pyplot.plot.html

Exemples

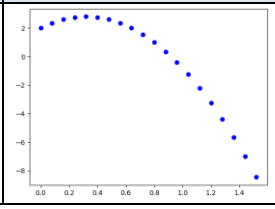
Exemple de courbe tracée à partir de données expérimentales (saisies « à la main ») :

Algorithme en langage naturel	Instructions en Python	Résultat à l'écran

Après avoir créé une fenêtre Matplotlib :
Créer un tableau de val expérimentales I
Créer un tableau de val expérimentales U
Tracer U en fonction de I (croix rouges reliées)
Et pour finir, afficher la fenêtre Matplolib
	`I = np.array([1,2,3])`
	`U = np.array([2,4,6])`
	`plt.plot(I,U,"rx-")`
	`plt.show()`
		

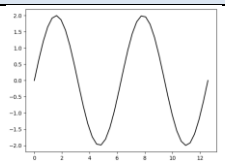
Exemple de tracé de courbe paramétrée (équations horaires par exemple) :

Algorithme en langage naturel	Instructions en Python	Résultat à l'écran

Après avoir créé une fenêtre Matplotlib :
Créer un tableau « t » de 0 à 2 s par pas de 0,1 s
Créer un tableau « x » qui vaut 0,8t
Créer un tableau « y » qui vaut $-5t^2+4t+2$
Tracer Y en fonction de X (en bleu, gros points)
Et pour finir, afficher la fenêtre Matplolib
	`t = np.arange(0,2,0.1)`
	`X = 0.8*t`
	`Y = -5*t**2 + 4*t + 2`
	`plt.plot(X,Y,"bo")`
	`plt.show()`
		

Exemple de tracé de fonction :

Algorithme en langage naturel	Instructions en Python	Résultat à l'écran

Après avoir créé une fenêtre Matplotlib :
Créer un tableau « x » de 40 valeurs de 0 à 4π
Créer un tableau « y » qui vaut 2sin(X)
Tracer Y en fonction de X (pas de points, juste reliés noir)
Et pour finir, afficher la fenêtre Matplolib
	`X = np.linspace(0,4*np.pi,40)`
	`Y = 2*np.sin(X)`
	`plt.plot(X,Y,"k-")`
	`plt.show()`
		

Ajout de commentaires sur le graphique

Il est possible d'ajouter un texte dans la fenêtre Matplotlib. Les codes couleurs sont les mêmes que pour « plot »

`plt.text (X,Y,"Texte à afficher",color='C')` Affiche le texte *Texte à afficher* aux coordonnées X,Y de couleur *C*

Tracé de vecteurs

`plt.quiver(X,Y,Vx,Vy,color='C',scale=20)` Dessine un vecteur en (*X*, *Y*) de coordonnées (*Vx*, *Vy*) de couleur *C*

Le nombre après « scale » dépend de l'échelle du graphique. On peut être amené à le modifier si le vecteur dessiné est trop petit ou sort de la fenêtre. « 20 » est une échelle habituelle.

Il peut être nécessaire d'imposer que le repère soit **orthonormé** en tapant les 2 lignes suivantes :

```
axes = plt.gca()
axes.set_aspect('equal','box')
```

Et pour finir…

À la fin du programme Python, on doit saisir :

`plt.show()` Affiche la fenêtre Matplotlib (**indispensable**)

`plt.get_current_fig_manager().window.state('zoomed')` Affiche la fenêtre plein écran sous Windows (**optionnel**)

Remarque au sujet de ce document :

Il a pour vocation à présenter les fonctions les plus fréquemment utilisées par le professeur de physique lors de l'animation de séances. Le module Matplotlib est très riche et de nombreuses autres fonctions/options sont disponibles à l'adresse suivante : https://matplotlib.org/api/index.html

Version 1.1 .
du document – février 2019

ANNEXE
MÉMENTO PYTHON 3

Le mémento des deux pages suivantes, réunit de manière non exhaustive
les fonctions les plus couramment utilisées en Python 3.x

MEMENTO PYTHON 3

Version 2.00 - O. Chaumette
Lycée JP SARTRE – 69500 BRON

TYPE et CONVERSIONS DE VARIABLES

Types courants : *Entier* (**int**) *Réel* (**float**) *Texte* (**str**) *Booléen* (**bool**)

Déclaration de variables :

```
T = "Bonjour"
N = 2
```
Assigne à la variable texte (string) *T* le texte « *Bonjour* »
Assigne à la variable entière (int) *N* la valeur *2*

Conversion de variables :

```
T = str(N)
N = int(T)   ou   N = float(T)
```
transforme **un entier** (int) ou réel (float) N **en texte** (string)
transforme **un texte** T (string) **en entier** (int) ou réel (float)

ENTREES et SORTIES : demander une valeur à l'utilisateur et afficher à l'écran.

Entrée au clavier :

```
Un_Texte = input("question")
Un_Entier = int(input("question"))
```
Pose « *question* » à l'utilisateur. Réponse dans la **variable « texte »** : Un_Texte
Réponse dans la **variable « entière »** : un_entier

Sortie écran :

```
print("Texte",variable)
print("Bonjour"+texte)
print("Bonjour"+str(Nombre))
print("Bonjour", end=' ')
print("Bonjour", end=',')
print("Bonjour\nle\nmonde")
print("%.2e"%N)
```
écrit sur l'écran *Texte* suivi du contenu de *Variable* séparés par un espace
écrit sur l'écran *Bonjour* suivi du contenu de *texte.* ***Réservé aux textes***
écrit sur l'écran *Bonjour* suivi de *Nombre* transformé en texte
écrit *Bonjour* suivi d'un espace sans retour à la ligne (le prochain print sera collé)
écrit *Bonjour* suivi d'une virgule sans retour à la ligne (le prochain print sera collé)
écrit *Bonjour* suivi d'un saut de ligne (**\n**) puis *le* puis saut de ligne puis *monde*
Ecrit le réel (ou entier) *N* en écriture scientifique avec 2 décimales (donc 3 chiffres significatifs)

TESTS et CONDITIONS :

Test simple:

```
if Condition:
    Instructions si « Condition » est vraie
```
bien penser aux « : »!
bien penser à l'indentation !!!

Test avec SINON (else):

```
if Condition:
    Instructions si « Condition » est vraie
else:
    Instructions si « Condition » est fausse
```
bien penser à l'indentation et aux « : »

Test avec SINON SI (else if):

```
If Condition1:
    Instructions si « Condition1 » est vraie
elif Condition2:
    Instructions si « Condition2 » est vraie
else:
    Instructions si « Condition1 et 2 » sont fausses
```
bien penser à l'indentation et aux « : »

Test avec conditions multiples:

```
if Condition1 and/or Condition2:
    Instructions
```
and: condition 1 **ET** 2 respectées
or: condition 1 **OU** 2 respectées

Opérateurs dans les conditions: **ATTENTION le signe = est réservé à l'affectation de variables**

`==` :égal `!=` :différent `not` : contraire de la condition
`>` (ou `<`):supérieur (ou inférieur) `>=` (ou `<=`):sup (ou inf) ou égal

On peut utiliser un intervalle : exple : `if 2<x<3:` *Instructions*

Variable Booléenne

```
Mon_Booleen = (A==B)
```
la variable *Mon_Booleen* prendra la valeur *True* si A égal B, *False* sinon.

INCREMENTATION d'une variable : Changer sa valeur en fonction de sa valeur initiale.

`Var += 1` incrémente la variable de 1 : cela remplace `Var = Var +1.` (Cas général : `Var+=`*Valeur*)

Cela marche aussi avec une chaine (`Chaine +=` *Texte* revient à écrire `Chaine = Chaine +` *Texte*)

Il existe aussi : `Var -=` *Valeur* (pour soustraire *Valeur* à Var), `Var *=` *Valeur* (pour multiplier Var par *Valeur*)

LISTES : Ce sont des tableaux.

Déclaration :

```
Liste = ["a","b","c"]
Liste = [2,4,8]
Liste = []
Liste2D = [[1,2,3],[4,5,6]]
```
Crée une liste de textes (type ***string*** car entre guillemets)
Crée une liste d'entiers (type ***int*** car pas de guillemets)
Crée une liste vide
Crée une liste d'entiers à 2 dimensions (ici 2 lignes et 3 colonnes)

Accès aux éléments de la liste :

```
Liste[index]
Liste[index1:index2]
Liste[index1:]
Liste[:index2]
Liste[-1]
Liste2D[n°_ligne][n°_col]
```
Renvoie l'élément situé à l'emplacement *index* (qui commence à 0)
Renvoie les éléments entre *index1* et *index2*
Renvoie les éléments à partir d'*index1* jusqu'à la fin
Renvoie les éléments du début jusqu'à *index2*
Renvoie le dernier élément (si -2 : l'avant dernier etc…)
Renvoie l'élément situé à *n°_ligne* (en commençant à 0) et *n°_col* (début 0)

Manipulation de listes :

```
len(Liste)
Liste.append(Elément)
Liste.remove(Elément)
Liste [pos] = "a"
Liste.insert(pos,Elément)
Liste.index(Elément)
Liste.count(Elément)
Elément in Liste
```
Renvoie la longueur (= **nbre d'éléments**) de la liste. C'est un **entier**.
Ajoute *Elément* **à la fin** de la liste
Enlève la 1ère occurrence de *Elément*
met 'a' à la position *pos* (début=0) (**en écrasant** l'él[t] qui s'y trouve)
Insère *Elément* à la position *pos* (**sans écraser.** Cela décale les autres)
Renvoie l'emplacement de *Elément* dans la liste (**1ère position = index 0**).
Renvoie le nombre de fois qu'*Elément* est présent dans la liste
Renvoie si *Elément* est présent dans la liste (True/False). Utile dans un `if` :
Exemple : `if Lettre in Mot` (si *Lettre* est présent dans *Mot*)

Parcourir une liste :

```
for Element in Liste:
```
Element prend successivement le contenu de chaque élément de la liste

ASTUCE : *Afficher les éléments d'une liste les uns à la suite des autres (séparés par un espace):*

```
T = ""
for Element in Liste:
    T = T +" "+ Element
print (T)
```
← remplacer ici `Element` par `str(Element)` si la liste contient des entiers

CHAINES (ou TEXTES): Ce sont des listes de caractères donc ont les mêmes fonctions + des fonctions suppl. :

Mise en forme :

```
Texte = "Bonjour \nSalut"
Texte = "Bonjour \"Salut\""
```
Saute une ligne après le \n
Saute des guillemets à la place de \"

Manipulations fréquentes de chaines :

```
Texte.lower()
Texte.upper()
Texte.split()
Texte.split(Caractère)
Texte.find(Mot)
Texte.replace(MotOld,MotNew)
```
Met le texte en minuscule
Met le texte en majuscule
Renvoie **une liste** contenant les mots du texte (s'ils st séparés par « **espace** »)
Renvoie **une liste** contenant les mots du texte (s'ils st séparés par ***Caractère***)
Renvoie le + petit index de *Mot* dans Mon_Texte (ou **-1 si pas trouvé**)
Remplace *MotOld* par *MotNew* dans Mon_Texte

```
Texte [:n]
Texte [Len(Mon_Texte)- n:]
Texte [n1:n2]
```
Renvoie les *n* premiers caractères en partant de la gauche
Renvoie les *n* premiers caractères en partant de la droite
Renvoie les caractères entre les rangs *n1* et *n2*

```
Texte.startswith(Mot)
Texte.endswith(Mot)
```
Renvoie True si Mon_Texte commence par *Mot*
Renvoie True si Mon_Texte finit par *Mot*

Utilisation des codes ASCII (des caractères d'un texte) :

```
Caractere = chr(Code_Ascii)
Code_ASCII = ord(Caractère)
```
Renvoie le caractère (type **str**) dont le code ascii est *Code_Ascii*
Renvoie le code ascii de *Caractère*.

Codes ASCII utiles : `"A"`:65 `"Z"`:90 `"a"`:97 `"z"`:122 `"0"`:48 `"9"`:57 espace :32

BOUCLES :

Boucle FOR générale: Dans le cas où on connaît le nombre de répétitions

```
for Variable in Ensemble de valeurs:
        Instructions
```

Variable va prendre toutes les valeurs de « ensemble de valeurs »
bien penser à l'indentation !!!

Boucle FOR avec des nombres:

```
for Compteur in range(Nombre):
```
Compteur varie de ***0*** à ***Nombre-1***

```
for Compteur in range(début,fin):
```
Compteur varie de ***début*** à ***fin-1***

```
for Compteur in range(début,fin,pas):
```
Compteur varie de ***début*** à ***fin-1*** par sauts de ***pas***

```
for Compteur in range(fin,début,-1):
```
Compteur varie de ***fin*** à ***début+1*** (donc **à l'envers**)

Boucle FOR avec des listes/chaines:

```
for Variable in Liste:
```
Variable prend la valeur de chaque élément de la liste *Liste*

Boucle WHILE (tant que) : Dans le cas où on ne connaît **pas** le nombre de répétitions

```
while Condition:
```
bien penser aux « : » et à l'indentation

Instructions tant que « Condition » est vraie

Modifier la variable intervenant dans la condition sinon la boucle serait infinie !

Sortie « artificielle » de boucle :

`break` arrête la boucle

OPERATEURS et FONCTIONS MATHÉMATIQUES:

Division:

`Nbre1 // Nbre2`	Renvoie la **partie entière** de la division de *Nbre1* par *Nbre2*
`Nbre1 % Nbre2`	Renvoie le **reste de la division** de *Nbre1* par *Nbre2*
`divmod(Nbre1, Nbre2)`	Renvoie un tuple (Résultat Division entière,Reste)

Fonctions:

`round(x)`	arrondit un "réel" *x* vers l'entier le plus proche
`round(x, n)`	arrondit un "réel" à la décimale *n*. *n* négatif permet un arrondi à la dizaine, centaine... près.
`**`	marque l'**exposant** et a la priorité sur +, -, *, /
`pow(x,y)`	renvoie *x* à la puissance *y*, équivaut à *x* ** *y*
`max(x, y)`	renvoie la plus grande des deux valeurs
`min(x, y)`	renvoie la plus petite des deux valeurs
`abs()`	renvoie la valeur absolue d'un nombre (sans le signe)

MODULE MATH : en complément des fonctions précédente. Pour les réels.

Déclaration :

```
import math
```

Fonctions :

`math.pi`	retourne une approximation de la constante pi: 3.1415926535897931
`math.degrees()` et `radians()`	transforment en degrés ou en radians
`math.cos()`,`.sin()`,`.tan()`	fonctions trigonométriques usuels
`math.acos()`,`.asin()`,`.atan()`	fonctions trigonométriques inverses
`math.exp()`	exponentielle
`math.log()` et `math.log10()`	logarithme népérien et décimal
`math.sqrt()`	renvoie la racine carrée

MODULE RANDOM : hazard

Déclaration :

```
import random
```

Fonctions :

`random.choice(Ma_Liste)` choisit un élément de la liste `Ma_Liste`
`random.sample(Ma_Liste,n)` renvoie une liste de *n* éléments choisis dans `Ma_Liste`
`random.shuffle(Ma_Liste)` mélange les éléments de `Ma_Liste`
`random.randrange(borne1,borne2)` renvoie un entier au hasard entre *borne1* (incluse) et *borne2* (exclue)

FONCTIONS : Ce sont des parties de programme que l'on peut appeler régulièrement selon les besoins.

Définition d'une fonction SANS return (=procédure) : c'est un morceau de programme qu'on peut appeler

```
def Ma_fonction():
    Liste d'instructions
```
Attention à l'indentation

Appel de la fonction:

```
Ma_fonction()
```
à placer dans le prg principal : Exécute le code de « Ma_Fonction »

Définition d'une fonction AVEC return:

```
def Ma_fonction():
    Liste d'instructions qui crée une «Variable_Résultat »
    return Variable_Résultat
```

Appel de la fonction :

```
Variable = Ma_fonction()
```
met dans *Variable* le résultat des instructions de *Ma_fonction*

Définition d'une fonction avec des ARGUMENTS (ou paramètres):

```
def Ma_fonction(paramètre1, paramètre2,…):
    Liste d'instructions qui crée une «Variable_Résultat »
    en utilisant paramètre1, paramètre2
    return Variable_Résultat
```

Appel de la fonction AVEC paramètres :

```
Variable = Ma_fonction(p1,p2…)
```
met dans *Variable* le résultat des instructions de *Ma_fonction*
(ayant utilisé les paramètres p1, p2 etc…)

Utilisation d'une variable du programme principal dans une fonction :

Les Variables d'une fonction et celles du programme principal sont indépendantes. Si on peut utiliser une variable du programme principal dans une fonction, il faut ajouter dans la fonction :

```
global Nom_de_la_Variable_du_prg_Principal
```

FICHIERS : LECTURE/ECRITURE simple de textes

Création/ouverture d'un fichier « Nom_de_Fichier.ext » :

`Mon_fichier = open('Nom_de_Fichier.ext','w')`	ouverture en écriture (efface l'existant si déjà présent)
`Mon_fichier = open('Nom_de_Fichier.ext','a')`	ouverture en modification (pour ajouter des données)
`Mon_fichier = open('Nom_de_Fichier.ext','r')`	ouverture en lecture

Fermeture d'un fichier : ***OBLIGATOIRE s'il a été ouvert !!***

```
Mon_fichier.close()
```

Ecriture de données dans « Mon_Fichier » : *il faut que « Mon_fichier » ait été ouvert (en écriture ou modif)*

```
Mon_fichier.write(Texte)
```
Ecrit « Texte » à la suite du fichier ; sans saut de ligne !

Pour ajouter une saut de ligne : *Texte* doit être : `'\n'`
Pour ajouter une tabulation : *Texte* doit être : `'\t'`

Lecture de données dans « Mon_Fichier » : *il faut que « Mon_fichier » ait été ouvert (en lecture)*

`Variable = Mon_fichier.read()`	Met dans « Variable » tout le contenu du fichier
`Variable = Mon_fichier.read(N)`	Met dans « Variable » les *N* caractères du fichier (à partir de la position atteinte dans le fichier)
`Variable = Mon_fichier.readline()`	Met dans « Variable » une ligne du fichier (à partir de la position atteinte dans le fichier)
`Liste = Mon_fichier.readlines()`	Met toutes les lignes du fichier dans une liste.

Changement du dossier dans lequel on lit/écrit le fichier :

```
import os
os.chrdir(Dossier_Voulu)
```
Dossier_Voulu peut être absolu: **"C:/Mon/Dossier"** ou relatif: **"/Mon/Dossier"**

RÉGION ACADÉMIQUE
AUVERGNE-RHÔNE-ALPES

MINISTÈRE
DE L'ÉDUCATION NATIONALE
ET DE LA JEUNESSE

MINISTÈRE
DE L'ENSEIGNEMENT SUPÉRIEUR,
DE LA RECHERCHE
ET DE L'INNOVATION

Document réalisé pour le GRD Physique-Chimie de l'Académie de Lyon par
Olivier CHAUMETTE (lycée JP Sartre – 69500 Bron)
Relecture et corrections :
Mathilde DIDIER-GLENAT, Jacques VINCE, Jean-Baptiste BUTET et Isabelle BERNARD

Avril 2019