L'Informatique Tronc Commun en PCSI

Lycée Pasteur – Neuilly sur Seine

L'utilisation du langage de codage Python que vous avez déjà vu au lycée fera l'objet d'une matière, l'Informatique Tronc Commun (ITC) en MPSI et PCSI.

Afin de (re)voir les notions qui sont au programme du lycée et éviter d'être submergé(e) en septembre par une grosse dose de travail si vous ne maitrisez pas les rudiments de Python, voici quelques supports de travail :

- Doc1, qui aborde les notions de base du langage python et l'utilisation de l'interface Pyzo. Une vidéo est proposée en complément de ce poly ainsi que deux fichiers .py , l'un à compléter lors des exercices et l'autre contenant la correction des exercices proposés dans le poly, sont également disponible sur le site de Pasteur.
- Doc2 permet de se familiariser avec les messages d'erreurs que vous pourrez rencontrer lors de l'exécution de vos programmes. Vous devez absolument apprendre à les utiliser pour vous auto-corriger.

Un test, basé sur les notions du Doc1, sera effectué en début d'année pour évaluer le niveau de chacun.

Bel été à tous,

L'équipe d'ITC des classes de PCSI et MPSI.

Doc 1 : Pour débuter avec Python et Pyzo

En complément de ce document, vous pouvez

- visionner la vidéo suivante : https://youtu.be/ZccGT6Ja7dg

Le but n'est pas forcément de lire ce poly et visionner la vidéo en une seule fois. Des chapitres ont été créés dans la vidéo pour faciliter son visionnage.

- vous tester sur les exercices présents à la fin de ce document. Un fichier à compléter (.py) et un fichier de correction (.py) sont également à votre disposition sur le site de Pasteur.

I. Prise en main de Python/Pyzo

On peut écrire des codes de programmes Python dans n'importe quel éditeur de texte et enregistrer ces scripts au format .py. Il ne reste alors qu'à les exécuter.

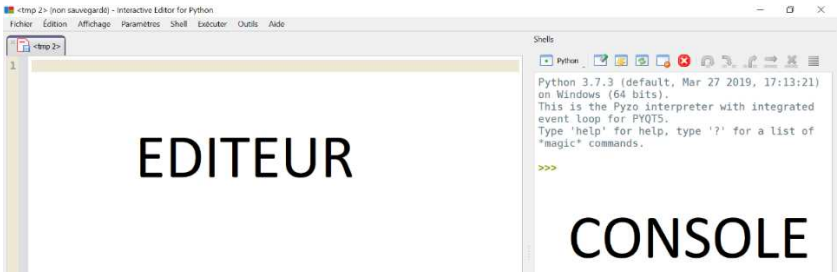
Il est plus simple d'utiliser des interfaces graphiques qui permettent à la fois d'écrire des codes, mais aussi de les exécuter. Nous utiliserons le logiciel **Pyzo**, qui est un logiciel portable. On peut le télécharger à partir du lien suivant :
https://pyzo.org/start.html

Il suffit alors de suivre les étapes « Step 1 » pour installer le logiciel Pyzo et « Step 2 » pour installer l'environnement Python, en choisissant la suite *Anaconda*.

On lance le fichier exécutable nommé Pyzo. La fenêtre qui s'est ouverte est divisée en plusieurs parties.

Les deux plus importantes :

- Un **éditeur texte** : permet d'écrire et de sauvegarder les programmes.
- Un **interpréteur** (ou **console**) : comme une calculette, permet d'exécuter et utiliser les programmes, ou pour écrire des programmes courts qui n'ont pas besoin d'être sauvegardés.



On va rapidement s'apercevoir que d'utiliser uniquement la console est peu commode : on ne peut pas modifier une ligne déjà écrite.
L'utilisation de l'éditeur est plus commode et permet de sauvegarder le travail effectuer.
On peut structurer les programmes en créant **des cellules** en commençant une ligne par deux dièses **##,** le texte suivant est alors en gras.
On peut également créer des **commentaires**, il s'agit de remarques destinées aux personnes qui lisent le code écrit dans l'éditeur et qui ne sont pas exécutées. Les commentaires commencent par un dièse #.

Nous vous conseillons d'enregistrer un TP par fichier .py et de prendre l'habitude de créer au moins une cellule par question pour faciliter leurs exécutions.

Il est nécessaire **d'éxécuter** les programmes contenus dans l'éditeur afin de les tester dans la console et ainsi effectuer une autocorrection dans le cas où un message d'erreur s'affiche ou si le résultat obtenu n'est pas celui attendu.
Il y a plusieurs façons d'exécuter du code écrit dans l'éditeur.
— avec la touche F5 pour exécuter tout le contenu de l'éditeur (très peu utilisé)
— avec la touche Alt+Enter pour exécuter uniquement les lignes que l'on a sélectionnées.

— en se plaçant dans une cellule puis Ctrl+Enter, afin d'exécuter le contenu d'une cellule.

II. Les variables

Les variables, qui sont des objets manipulés dans les programmes python, peuvent être de différents types.

Pour afficher dans la console, le résultat d'une expression écrite dans l'éditeur, on utilisera la fonction *print(expression)*.

1. Les entiers

Type : *int* pour *interger* qui correspond aux entiers relatifs

Opérateurs sur les entiers et exemples :

Opérateur	Opération
+	Addition
-	Soustraction
*	Multiplication
/	Quotient usuel
a%b	Reste de la division euclidienne de *a* par *b*
a//b	Quotient de la division euclidienne de *a* par *b*
a**n	*a* puissance n

```
>>> 2021-18
2003

>>> 2021+18
2039

>>> 18*2
36

>>> 18**2
324

>>> 18//7
2

>>> 18%7
4
```

2. Les flottants

Type : *float* qui correspond aux nombres à virgules.

La précision étant limitée, les calculs sur les flottants, contrairement aux calculs sur les entiers, ne sont approchés et les propriétés usuelles de + et x (commutativité et associativité par exemple) ne sont plus vérifiées en général :

```
>>> 1-1/3-1/3-1/3
1.1102230246251565e-16

>>> 1/3+1/3+1/3
1.0
```

3. Les booléens

Type : *bool* qui correspond aux deux valeurs de vérité : *True* et *False*

Opérateurs : Pour combiner des booléens booléennes, on dispose des trois opérateurs logiques, la disjonction, la conjonction et la négation : *or* (ou), *and* (et), *not* (non). Ils sont définis par les tables de vérité suivantes :

b1	***b2***	***b1 or b2***	***b1 and b2***	***not(b1)***
True	*True*	*True*	*True*	*False*
True	*False*	*True*	*False*	*False*
False	*True*	*True*	*False*	*True*
False	*False*	*False*	*False*	*True*

Les opérateurs de comparaison permettent de construire des valeurs booléennes à partir d'autres expressions. On peut appliquer à des couples d'expressions les opérateurs de comparaison suivants :

Opérateur	Sens
==	= (égalité)
!=	≠
<, >	<, >
<= , >=	≤, ≥

```
>>> 1<2
True

>>> 2>=2
True

>>> 15==45-3*10
True

>>> 15!=45-3*10
False
```

Exercice 1

4. Les chaines de caractères

Type : *str* pour *string* ce qui correspond à une séquence de caractères sur laquelle les opérations mathématiques ne sont pas possibles.

Opérateur	Opération
apostrophes ’, guillemets " ou des triples apostrophes ”’	Permettent de délimiter des chaines de caractères
\n	Permet d’afficher des retours à la ligne dans une chaine de caractère (signifie new line)
str + str	Permet de former une plus grande chaine de caractère en accolant deux chaines de caractères. Par exemple *‘abc’ + ‘def’* donne *‘abcdef’*
*str * int*	Permet de former une plus grande chaine de caractère en répétant une chaine de caractère. Par exemple *‘abc’*3* donne *‘abcabcabc’*

Exercice 2

III. Le noyau impératif d'un langage de programmation

On peut dégager, dans la plupart des langages, qu'ils soient impératifs ou non, un groupe de cinq constructions élémentaires qui constituent ce qu'on appelle le noyau impératif du langage :

- LA DÉCLARATION et L'AFFECTATION
- LA SÉQUENCE : exécution d'une première instruction *p1* puis d'une seconde *p2*
- LE TEST : selon l'état des variables, on exécute une instruction *p1* ou une autre instruction *p2*
- LA BOUCLE : répéter une instruction *p* un nombre de fois dépendant de l'état des variables.

Il est primordial de bien comprendre et de distinguer les deux notions suivantes :

- Une **expression** est une suite d'opérations sur des objets ayant une valeur, dont le résultat est une valeur.
- Une **instruction** est une commande qui doit être exécutée par la machine.

1. Déclaration et affectation

Une **variable** est un espace mémoire dans lequel il est possible de mettre une valeur, cette opération étant appelée **affectation**. L'utilisateur manipule cette variable par l'intermédiaire d'un nom qu'il choisit.

Ainsi une **affectation** est une instruction qui lie un nom, déjà existants ou nouveau, à une valeur.

Symbole de l'affectation :	=
Nom de variable :	Formé de lettres sans accent, de chiffres, du caractère _ et commençant toujours par une lettre

Code	Action de l'instruction
x=exp	L'expression *exp* est évaluée puis le résultat est stocké dans la variable *x*.
x+=exp	A le même effet que *x=x+exp* (valable aussi pour *-=, *=, /=,//=, %=* et ***=* L'expression de *exp* est évaluée puis sa valeur est ajoutée (pour +=) à celle de *x* et finalement le résultat est stocké dans la variable *x*.
x,y=exp1,exp2	Les expressions *exp1* et *exp2* sont évaluées puis les résultats sont respectivement affectés aux variables *x1* et *x2*.
Fonction	**Opération**
print(argument)	Affiche la (ou les) variable(s) contenue(s) dans l'argument dans la console.
type(argument)	Affiche la nature de la variable *argument*

 Exercices 3 et 4

2. Le test : instruction conditionnelle

```
if condition :
    instructions1
else :
    instructions2
```

Soient *condition* une expression booléenne, *instructions1* et *instructions2* deux instructions ou blocs d'instructions.

L'instruction ci-contre correspond à :

- si, après évaluation, *condition* vaut *True*, alors *instructions1* est exécutée.
- Sinon (donc si *condition* vaut *False),* Python exécute *instructions2.*

La commande *else* n'est pas obligatoire si une action doit être faite seulement si *condition* est vérifiée.
La commande *elif* permet de distinguer plusieurs cas dans **un même test** qui est terminé dès qu'une expression booléenne est vérifiée.

A retenir :	if/else/elif **sans majuscule**. : après les conditions ou le else. **Indentation** après les :

```
x,a=12,0
if x%2==0 :
    a+=1
elif x%4==0:
    a+=1
print(a)
```

```
x,a=12,0
if x%2==0 :
    a+=1
if x%4==0:
    a+=1
print(a)
```

Dans le premier cas, a = 1

Dans le deuxième cas, a = 2

Exercices 5, 6 et 7

3. La boucle while ou boucle conditionnelle

Si on souhaite répéter une séquence d'instructions un nombre de fois qui n'est pas connu à l'avance car dépendant de l'état des variables, on utilise une boucle while ou boucle « *tant que* ».

```
while condition :
    instructions
```

Soient *condition* une expression boolénne et *instruction* une instruction ou un bloc d'instructions.
Le bloc *instructions* est exécuté si *condition* a pour valeur *True.* Si la condition est encore vérifiée après l'exécution d'*instructions*, le bloc *instructions* est de nouveau exécuté, et ainsi de suite jusqu'à ce que *condition* prenne la valeur *False.*

A retenir :	while **sans majuscule**. : après la condition et **l'indentation** après les : *Instructions* doit obligatoirement modifier la valeur de *condition* après un nombre fini d'exécution du bloc ; sinon le programme tournera de manière infinie.

Exercice 8

4. La boucle for ou boucle inconditionnelle

```
for i in range(a,b,pas):
    action
```

Quand une instruction est itérée un nombre de fois connu à l'avance, on utilise une boucle for.
Soient *a* et *b* des **entiers** et *action* une instruction.
La variable *i* parcourt l'ensemble des entiers de la forme a+k*pas, compris entre *a* et *b-1* **inclus** avec *k* un entier positif.
i peut être une variable utilisée ou non dans le programme.
Par défaut, le début vaut 0 et le pas vaut 1. On peut alors préciser uniquement la variable *b*.

A retenir :	La variable i **ne prend pas la valeur de b** car elle s'arrête à b-1. : après la condition et **l'indentation** après les : Dans le cas où l'intervalle est décroissant, a>b, le pas vaut -1 et les trois arguments doivent être précisés.

Exercice 9

IV. Les fonctions

On repère rapidement que si on veut tester plusieurs fois de suite un même script avec les mêmes valeurs pour les variables d'entrée, il faut l'exécuter plusieurs fois et entrer à chaque fois les valeurs souhaitées. On peut contourner ce problème en écrivant des fonctions.

Une fonction est un programme qui prend en argument un ensemble de variables et qui renvoie un résultat utilisable tel quel par un autre programme.

```
def nom_de_la_fonction(argument1,argument2):
    ''' Aide sur la fonction'''
    corps de la fonction
```

Pour définir une fonction en Python, on utilise la syntaxe suivante :

- Sur la première ligne :
  - Le mot ***def*** est une clé de langage qui initie la création d'une fonction.
  - Le nom de la fonction suit les mêmes règles que le nom d'une variable.
  - Les *arguments* sont présents entre parenthèses et séparés par des virgules. Il peut ne pas avoir d'argument dans la fonction.
  - Cette première ligne se termine par le signe « ***:*** », qui entraine la création d'une indentation lors du passage à la ligne suivante.
- Sur la deuxième ligne, la partie entre les triples guillemets est une aide avec des indications sur ce qu'on attend des arguments et du résultat. Elle n'est pas obligatoire mais recommandée.
- Le corps de la fonction est un bloc de code indenté par rapport à la ligne de définition.

La fonction peut **renvoyer** un résultat si le corps de la fonction contient l'instruction ***return*** qui est une clé de langage. Dans ce cas, l'appel de la fonction créer une valeur, qui peut être ensuite affectée à une variable et/ou réutilisée. Ce n'est pas le cas de la procédure ***print()***

!! Une instruction *return* interrompt l'exécution d'une fonction. Il faut donc être vigilant dans le placement de(s) instruction(s) *return* dans la fonction !!

Pour qu'une variable puisse être utilisable dans le corps d'une fonction, elle doit donc soit être en argument, soit être définie au sein du corps de la fonction. Dans ce deuxième cas, c'est alors une variable **locale** qui n'a plus d'existence à la fin de l'exécution de la fonction.

Un avantage notable des fonctions est qu'on peut découper une tâche complexe en plusieurs sous-tâches distinctes résolues par des fonctions.
Enfin, une fonction peut prendre un, plusieurs, voire aucun argument et renvoyer un résultat composé d'une ou de plusieurs variables.

 Exercice 10

Exercices

Niveau de difficulté des questions mentionnées par * (facile), ** ou *** (niveau plus élevé)

Exercice 1 :

On peut utiliser aussi des fonctions mathématiques.
Pour ce faire, il faut d'abord charger la bibliothèque math en tapant : from math import *
(soit sur une ligne de l'éditeur qu'il faudra exécuter, soit directement sur la console puis Enter)
L'étoile signifie que l'on charge toutes les fonctions de la bibliothèque math. On a alors accès, notamment, aux fonctions sin, cos, sqrt, log, log10, exp ou aux constantes pi et e.
Attention, la fonction log correspond à la fonction logarithme népérien ln introduite en mathématiques. Pour utiliser la fonction logarithme décimal log, il faudra utiliser la fonction log10 sous python.

1.(*) Prévoir le résultat des calculs suivants et les effectuer dans la console :
100//3 100/3 100%3 10**3

2.(**) Comparer les résultats des calculs suivants :
14.0//7.0 14//7 14//7.0 1-3*(1/3) 1-1/3-1/3-1/3

3.(**) Prévoir le résultat des calculs suivants et les effectuer dans la console :
sin(5π/2) sin(3π)

4.(*) Prévoir le résultat des expressions suivantes et les effectuer dans la console :
1 - 3*(1/3) != 1 - 1/3 - 1/3 - 1/3 (2**4%3) == 2 (4%2)+1-2**3 > (12//4)*(-2**4)

Exercice 2 :

1.(*) Afficher les chaines de caractères suivantes dans la console, à l'aide d'une seule fonction *print*.

a. Tant en Maths qu'en Physique, Chimie et SI, il est très important d'être à l'aise en calculs : le calcul est un outil intermédiaire entre les premières lignes écrites qui ne sont que la traduction du problème, et la conclusion, c'est donc fondamental.
On ne peut s'améliorer en calculs qu'en en faisant, et certainement pas en se contentant de recopier le tableau.
Rappelons enfin que l'usage de la calculatrice est le plus souvent interdit à l'écrit des concours en Mathématiques.

b. Le campanile du Lycée Pasteur de Neuilly-sur-Seine est revêtu des quatre inscriptions suivantes : "L'heure revient, l'homme ne revient pas " ; "Chaque heure blesse, la dernière tue " ; "L'heure française sonnera toujours " ; " Quand l'heure sonne, homme soit debout ".
Elles sont gravées lors de la construction du lycée et sont antérieures à la Première Guerre mondiale.

2.(*) Prévoir le résultat des opérations suivantes et les effectuer dans la console.
'3'*3 '43'+'56' «informatique»*4 «l'informatique»+ «c'est» + «chouette»

Exercice 3 :

1.(*) Tester les instructions ci-dessous dans une console et expliquer ce qui se passe.

```
>> a = 3
>> a
>> print(a)
>> print('a')
>> print('a = ', a)
```

2.(*) Prévoir la valeur des variables à l'issue des séries d'instructions suivantes. Prévoir les messages d'erreur éventuels. Vérifier les résultats dans la console et réinitialiser la console après chaque série.

>> a=4	>> a=2	>> a=4	>> a=4	>> i=3	>> b=2	>> a=3	>> a = 2
>> b=2	>> b=4	>> b=2	>> b=2	>> ai=3*i	>> b=a	>> b=4	>> b = 2*A
>> c=b	>> c=a	>> 2=a	>> c=b	>> b=2*a3		>> x=a	
>> b=a	>> b=a	>> b=3	>> b=a			>> a=b	
	>> c=b-a		>> a=c			>> c=x	

Exercice 4 : Interagir avec l'utilisateur

Pour demander une chaîne de caractères à l'utilisateur, on peut écrire l'instruction suivante dans l'éditeur :
x = input('Une explication.')
Une fois cette ligne de code exécutée, l'utilisateur est invité à taper quelque chose dans la console. Lorsqu'il aura appuyé sur la touche Entrer, ce qu'il aura tapé sera stocké dans la variable x, qui sera alors de type *str.*
Pour demander un nombre à l'utilisateur, il faut lui demander une chaîne de caractères puis convertir cette chaîne de caractères en nombre à l'aide de la fonction *int(arg)* ou *float(arg)* qui transforme une variable *arg* soit en entier ou en flottant. L'opération inverse peut être faite grâce à la fonction *str(arg)* qui transforme une variable *arg* en chaine de caractère.

1.(*) Recopier le code ci-dessous tel quel dans un éditeur.

```
121 ##Changement du prénom
122 prenom = input("Entrer votre prénom : \n")
123
124 ##Calcul de l'âge
125 année = int(input("Entrer votre année de naissance :"))
126
127 ##Piège
128 année = 2000
129 prenom = "Super élève"
130 print("Etes-vous vraiment",prenom,"agé(e) de ",2022-année,'ans ?')
```

2.(*) Exécuter les 3 cellules dans l'ordre, sans oublier de répondre aux sollicitations dans la console. Qu'observe-t-on lors de l'exécution de la ligne 130 ?
3. (**) Exécuter ce code pour qu'il affiche en fin d'exécution la phrase « Etes-vous vraiment X âgé(e) de N ans ? » avec X et N le prénom et l'âge de l'utilisateur.
4.(**) Quel est le rôle de la fonction *int()* ligne 125 ? Pourquoi est-elle nécessaire pour afficher « Etes-vous vraiment X âgé(e) de N ans ? » avec X et N le prénom et l'âge de l'utilisateur ?
5.(**) Rajouter des lignes pour afficher le type de la variable *prenom* définie ligne 122 et de la variable *année* définie ligne 125.

Exercice 5

On considère le code ci-contre :

```
x=0
if 17%3 == 1 :
    x = x + 1
else :
    x-=2
```

1.(*) Expliquer son fonctionnement en détail sans vérifier le résultat.
2.(*) Ajouter une ligne à un endroit bien choisi du code qui permet de vérifier vos affirmations.
3.(*) Vérifier enfin en exécutant le code modifié.

Exercice 6

Ne pas oublier de tester les programmes écrits dans l'éditeur en les exécutant, afin de vérifier leur bon fonctionnement dans les différents cas de figures.
1.(*) Ecrire un programme qui permet de demander deux nombres à l'utilisateur et d'afficher le plus grand des deux. Dans le cas d'égalité, rien n'est affiché.
2.(**) Demander à l'utilisateur deux entiers et afficher dans la console la moyenne s'ils sont tous les deux pairs.
3.(**) Demander à l'utilisateur de renseigner deux variables numériques *a* et *b* et échanger leurs contenus si *a* est divisible par 5. Afficher ensuite les valeurs de *a* et de *b*.

Exercice 7

(**) Expliquer le fonctionnement du programme suivant. Comment le modifier pour séparer les deux cas du « else » ?

```
a = float(input("Entrer a :"))
b = float(input("Entrer b :"))
if a!=0 :
    print('''L'unique solution de ax+b=0 vaut environ''',-b/a)
else :
    print(''' a est nul donc l'équation ax+b=0 admet soit une infinité de
solutions ou bien aucune solution''')
```

Exercice 8 :

Écrire des programmes réalisant les tâches suivantes en utilisant une boucle conditionnelle :

1.(*) Afficher les entiers naturels de la forme 3k + 1 inférieurs ou égaux à 100.

2.(*) Demander en boucle à l'utilisateur de taper quelque chose jusqu'à ce qu'il tape : "J'adore les cours d'ITC !"

3.(***) Tirer des entiers entre 1 et 10 au hasard jusqu'à ce que leur somme dépasse strictement 16 et les afficher.
On importera pour cela la fonction randint de la bibliothèque random avec la ligne de code suivante :
>> from random import randint
La fonction randint prend deux entiers en arguments et renvoie un entier compris entre ces deux entiers inclus.

4.(**) Déterminer le plus petit entier n ≥ 2 tel que $\left|\frac{\sin(n)}{n}\right| \leq 10^{-3}$

5.(**) Trouver le plus petit multiple de 13 supérieur ou égal à *n*, pour *n* choisi par l'utilisateur.

6.(**) Réaliser une boucle infinie. Il ne faudra pas oublier de réinitialiser la console pour sortir de cette boucle infinie.

7.(***) On considère la suite (u_n) définie par $u_0 = 5$ et la relation : $\forall n \in \mathbb{N}, u_{n+1} = \frac{u_n}{2}$. Afficher le plus grand rang *k* tel que $u_k \geq 0.1$ et le terme u_k correspondant.

Exercice 9 :

Écrire des programmes réalisant les tâches suivantes en utilisant une boucle inconditionnelle :

1.(*) Générer une par une, les 4 colonnes ci-dessous.

```
1          0          100        1
2          3          99         2
3          6          98         2
...        ..         ...        3
1000       1002       0          3
                                 3
                                 ...
                                 10 (10 fois)
```

2.(**) Afficher les 5 premiers termes de la suite définie par $u_0 = 1$ et $\forall n \in \mathbb{N}, u_{n+1} = \frac{u_n+1}{u_n-n}$

3.(**) Afficher le terme de rang *n* de la suite définie par $u_0 = 2$ et $\forall n \in \mathbb{N}, u_{n+1} = u_n - n + 1$ pour *n* choisi par l'utilisateur.

4.(**) Calculer la somme $\sum_{k=1}^{1000} \ln(k)$ en faisant varier *k* dans l'ordre croissant puis recommencer en faisant varier *k* dans l'ordre décroissant.

5.(***) Calculer la somme $\sum_{k=0}^{n} \sum_{i=1}^{k} \frac{1}{i+k}$ pour *n* choisi par l'utilisateur.

Exercice 10 :

Écrire des fonctions qui satisfont les spécifications suivantes. Ne pas oublier de tester les fonctions pour vérifier leur fonctionnement.

1.(*) Fonction ***somme(a,b)***, avec *a* et *b* deux entiers, qui retourne la somme *a+b* sous la forme d'un entier.

2.(*) Fonction ***multiple(a,b)***, avec *a* et *b* deux entiers**,** qui retourne un booléen dont la valeur indique si l'un est multiple de l'autre.

3.(**) Fonction ***calcule(x,y),*** avec *x* et *y* deux réels, qui retourne si possible, $\frac{\sqrt{x}}{\ln(y)}$. Dans le cas inverse, elle ne retourne rien.

4.(**) Fonction ***puissance(x)***, avec *x* un réel > 1, qui retourne la plus petite de ses puissances supérieures à 100.

5.(***) Fonction ***parfait(x)***, avec x un entier naturel non nul, et qui renvoie un booléen dont la valeur indiquer s'il est parfait ou non.
Rappel : un entier naturel non nul est dit parfait s'il est la somme de ses diviseurs positifs stricts. Par exemple, 6 est parfait car ses diviseurs positifs stricts sont 1, 2 et 3 et on a 1 + 2 + 3 = 6.

6.(***) Fonction ***bissextile(n)***, avec *n* un entier naturel, et qui renvoie un booléen dont la valeur indiquer si l'année n est bissextile.
Rappel : une annéeest bissextile si elle est soit divisible par 4 mais pas par 100, soit divisible par 400

7.(***) On rappelle que la fonction *randint* issu de la bibliothèque *random* renvoie un entier tiré au hasard entre ses deux arguments (bornes incluses).
Fonction ***multiple3()***, qui renvoie le plus petit multiple de 3 strictement supérieur à un entier tiré au hasard entre 1 et 10. La fonction affichera également l'entier tiré au hasard.

Doc 2 : Erreurs classiques rectifiables par auto-correction

Il y a des erreurs classiques dans le script des étudiants débutants qui empêchent leur programme de fonctionner. Voici une liste **non exhaustive** des erreurs fréquemment rencontrées et qui peuvent en général être facilement corrigée en autonomie après exécution du script puisque la ligne supposée erronée est indiquée :

Erreurs d'indentation :

- la ligne d'instructions n'est pas alignée correctement sur un bloc

```
IndentationError: expected an indented block
```

Erreurs de syntaxe :

- le mot clé n'existe pas ou 2 lettres ont été inversées (« imput » au lieu de « input »)

```
...is not defined
```

- la casse d'un mot clé n'est pas respectée (« Print » au lieu de « print »)

```
...is not defined
```

- une parenthèse a été ouverte mais non refermée. Idem pour des guillemets.

```
SyntaxError: invalid syntax
```

Astuce : si dans une ligne, une erreur de syntaxe est détectée mais que visiblement aucune erreur est présente, cela signifie que l'erreur se situe à la ligne précédente avec l'oubli de fermer une parenthèse, un crochet…

Erreurs de variables :

- le nom de la variable ne doit pas contenir d'espace.

```
SyntaxError: invalid syntax
```

- la casse du nom de variable doit être respectée (« A » est une variable différente de « a »)

```
...is not defined
```

- la variable doit être préalablement déclarée avant d'être utilisée.

```
...is not defined
```

- la variable ne doit pas avoir le nom d'un mot clé (ne pas appeler une variable « print », « def », « True »…)

- un nombre décimal s'écrit avec un « point » à la place d'une virgule.

- une variable associée à une autre par une opération ou une concaténation peut provoquer une erreur liée à la compatibilité de type. Il est nécessaire d'opérer pour une des variables la conversion appropriée (int, str, float, list, …).

```
TypeError: unsupported operand
```

Erreurs de conditions :

- oubli de « : » à la fin de la ligne if, elif ou else

```
SyntaxError: invalid syntax
```

- Pour une condition d'égalité, il faut mettre «==» (ne confondre test d'égalité avec « == » et une déclaration/affectation avec « = »)

```
SyntaxError: invalid syntax
```

- erreur d'indentation des instructions du bloc « if, elif ou else »

```
IndentationError: expected an indented block
```

Erreurs de boucles « for » :

- oubli de « : » à la fin de la ligne

```
SyntaxError: invalid syntax
```

- erreur d'indentation des instructions du bloc « for »

```
SyntaxError: invalid syntax
```

- 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.

Astuce : vérifier la première valeur obtenue à l'issue de la première itération et vérifier la dernière valeur obtenue à l'issue de la dernière itération. Il faut vérifier les bornes si on utilise une boucle.

Erreurs de boucles « while » («tant que »):

- oubli de « : » à la fin de la ligne

```
SyntaxError: invalid syntax
```

- erreur d'indentation des instructions du bloc « while »

```
SyntaxError: invalid syntax
```

- la condition du while n'a pas été modifiée dans le bloc donc la boucle tourne à l'infini…(recherche d'une terminaison de boucle)

Erreurs de listes (ou tableaux) :

- si un index dépasse le nombre d'élément d'une liste, l'erreur « out of range » apparaît : il faut vérifier les bornes si on utilise une boucle ou bien l'index utilisé.

```
IndexError: list index out of range
```

Astuce : l'erreur peut exister avec une chaîne de caractères.

```
IndexError: string index out of range
```