Chapitre 1
Algorithmique et programmation Partie 1

I. Introduction

1. Définition

À l'ère du numérique, des smartphones et des réseaux sociaux, qui n'a pas entendu parler des algorithmes ? Et pour cause, ils sont omniprésents dans notre quotidien. Ils sont à la base de nos applis et de nos ordinateurs, de la télécommunication et d'internet. Sans eux, la technologie contemporaine ne pourrait exister.

Pourtant, si tout le monde a entendu parler des algorithmes, très peu sont capables de dire ce qu'est un algorithme. Cette méconnaissance entraîne tantôt de la fascination, tantôt de la crainte, alors que la notion d'algorithme n'est peut-être pas aussi complexe qu'on pourrait le penser. D'ailleurs, beaucoup de personnes utilisent directement des algorithmes sans le savoir. Peut-être seriez vous capable de citer des exemples d'algorithmes que vous utilisez vous-même régulièrement ?

Si les algorithmes sont à la base de la programmation informatique, et donc du fonctionnement de nos applis, leur première utilisation est bien antérieure à l'invention de l'ordinateur. D'ailleurs, celle-ci précède même la vie du mathématicien persan Muhammad Ibn Mūsā al-Khuwārizmī (circa 780-850) dont le nom sous sa forme latinisée, Algorizmi, est à l'origine du mot algorithme. En effet, la plus ancienne trace retrouvée de la description d'un algorithme date de plus de 4500 ans. Il s'agit d'une tablette d'argile sumérienne qui détaille un algorithme de division.

Définition 1.1.
Un ***algorithme*** est une séquence ***finie*** et ***univoque***[a] d'instructions permettant de résoudre une classe de problèmes[b].

a. Qui ne souffre pas d'ambiguïté.
b. N'importe quel problème d'un certain type.

Exemple 1.1.
L'addition avec retenue est un algorithme que vous connaissez et utilisez depuis le primaire ! Il permet d'additionner deux entiers positifs quels qu'ils soient. ■

2. Le code

Les algorithmes sont importants en mathématiques comme en informatique parce qu'ils définissent des protocoles très précis permettant d'accomplir une tâche particulière autant de fois que nécessaire, sans jamais se tromper, ni avoir à se demander comment faire : il suffit de suivre les instructions. C'est un peu comme une recette de

cuisine mais pour laquelle on est certain que le plat sera réussi à chaque fois si l'on suit correctement les étapes. Pour garantir cette réussite systématique, l'algorithme doit être présenté dans un langage simplifié qui ne tolère pas les double-sens ou les approximations : le ***code***. C'est en cela qu'un algorithme diffère d'une recette de cuisine ou d'un mode d'emploi qui sont eux écrits en langage naturel.

Il existe de très nombreux langages différents pour écrire du code, notamment en informatique, où l'on parle de ***langage de programmation***. Lorsqu'un algorithme est écrit dans un langage de programmation particulier, on dit qu'il est ***implémenté*** dans ce langage (on utilise aussi le terme ***implémentation***). Dans ce cours, nous utiliserons le langage de programmation Python qui est au programme du lycée. Nous utiliserons également le ***pseudo-code*** qui est un langage intermédiaire entre le langage naturel et le code à proprement parler.

▪ Exemple 1.2.
Voici un exemple d'une séquence d'instructions en pseudo-code et de son implémentation en Python.

```
Pour k allant de 0 à 9
    Faire
    |Afficher k
Fin pour
```

```python
for k in range(10):
    print(k)
```

Pour bien distinguer entre les deux, la séquence en pseudo-code et la séquence en Python sont présentées avec des styles graphiques différents :
— le pseudo-code dans un rectangle blanc à bords droits ;
— le code Python dans un rectangle gris à bords ronds.
Cette convention sera maintenue tout au long de ce cours afin d'éviter toute confusion entre les deux. ▪

3. Objectifs du chapitre

Dans ce chapitre, vous apprendrez les principes de base pour l'élaboration d'un algorithme, son écriture en pseudo-code ainsi que son implémentation en Python. Vous serez également amenés à écrire vos propres algorithmes. Enfin, la notion de simulation sera introduite en fin de chapitre **??**.

Quelques exercices très simples ont été intégrées dans le cours, et vous êtes fortement encouragées à vérifier les réponses de ces exercices par vous mêmes, en exécutant les instructions correspondantes sur votre ordinateur. Ainsi vous pourrez pleinement appréhender les nouveaux objets que vous découvrirez au fur et à mesure de votre lecture de ce cours.

4. Exécuter un code Python en Console

Pour exécuter un code Python, il faut d'abord installer Python sur votre machine. Aujourd'hui, on peut installer Python sur un ordinateur, une tablette, un téléphone portable et même une calculatrice programmable. En fonction de votre machine et de votre système d'exploitation (Ubuntu/Linux, Windows, Mac OS X, ...), il existe de nombreuses façons d'installer Python et de nombreux ***environnements de développement***

différents qui permettent de visualiser le code que l'on programme avant de l'exécuter. Si vous vous y connaissez déjà en programmation, vous êtes libres de choisir l'environnement qui vous convient le mieux. Pour les autres, nous vous conseillons l'utilisation de l'environnement de développement Pyzo installé sur votre ordinateur via la distribution de base Python3 [1].

Pour installer Python, vous pouvez télécharger l'installateur à partir de la page web : https://www.python.org/downloads/. En principe, votre système d'exploitation sera reconnu automatiquement par le site internet, si ce n'est pas le cas choisissez l'onglet correspondant à votre système d'exploitation (Windows, macOs ou Linux). Ensuite, cliquez sur le lien de téléchargement correspondant à la version de Python 3 la plus récente. Une fois l'installateur téléchargé, exécutez le fichier et suivez les instructions correspondantes. Il vous suffira ensuite de lancer l'application Pyzo pour programmer et exécuter du code Python. Vous pourrez télécharger Pyzo à partir de la page web : https://pyzo.org/start.html.

Pour une prise en main rapide de l'environnement Pyzo, vous pouvez visionner l'un des nombreux tutoriels disponibles sur internet.

II. Variables et instructions élémentaires

1. Types de variables

En informatique, pour stocker les données, on utilise des ***variables***. Une variable est définie par un nom qui permet de l'identifier de manière unique. Ce nom permet à l'ordinateur de retrouver la donnée contenue dans sa mémoire qui correspond à la ***valeur*** de la variable. Chaque variable est caractérisée par un ***type*** qui correspond au type de donnée qu'elle contient. Il existe de nombreux types de variables différents en informatique mais nous n'en étudierons que quatre à ce stade.

Type booléen

Une variable de ***type booléen*** (ou ***variable booléenne***) est une variable qui prend l'une des deux valeurs : **vrai** ou **faux**. Le terme booléen est un hommage au grand mathématicien et logicien anglais George Boole (1815-1864) considéré comme le père de la logique moderne. En Python, le type booléen se note «`bool`» et les valeurs associées «`True`» et «`False`» respectivement.

Type entier

Le ***type entier*** désigne, comme son nom l'indique, des variables entières (qui appartiennent donc à l'ensemble $\mathbb{N}$) [2]. En Python, le type entier se note «`int`» (contraction de «*integer*» qui signifie entier en anglais).

1. Une distribution contient à la fois Python et un ensemble de bibliothèques Python (voir section **??**)

2. Attention toutefois, un ordinateur est un système fini et ne peut donc stocker des variables sans limite de taille. En pratique, la taille maximale d'une variable de type entière varie d'un langage de programmation à un autre.

Type flottant

Une variable de ***type flottant*** (ou ***variable flottante***) contient un nombre qui s'écrit avec un nombre fini de chiffres après la virgule[3]. Le type flottant se note «`float`» en Python (qui signifie flotter en anglais). Attention, la virgule est remplacée par un point dans l'écriture anglo-saxonne des nombres et donc, en particulier, dans le langage Python.

Type chaîne de caractères

Une variable de ***type chaîne de caractères*** contient du texte. Le terme ***caractère*** désigne les caractères typographiques que sont les lettres (minuscules ou majuscules, avec ou sans accents), les chiffres, les signes de ponctuation, etc. Dans le langage Python, ce type est noté «`str`» (contraction de «*string*» qui signifie fil ou enchaînement en anglais). Les chaînes de caractères s'écrivent en Python par du texte entre guillemets simples (par exemple : `'Ceci est un string en Python.'`) ou guillemets doubles (par exemple : `"Ceci est également un string en Python!"`).

2. Affectation

Définition 1.2.
L'***affectation*** d'une valeur à une variable est l'action de donner une valeur à cette variable.

Pour indiquer que l'on affecte la valeur `x` à la variable `a` en pseudo-code, on note :

```
a ← x
```

En Python, on note :

```
a = x
```

Attention, le signe « = » n'a donc pas le même sens en Python qu'en mathématiques ! En particulier, en Python, « `a = x` » n'a pas le même sens que « `x = a` ».

En programmation, il est possible d'affecter et de ***réaffecter*** des variables. Dans ce cas, c'est la dernière affectation qui prévaut sur les autres. On peut également mettre à jour une variable en lui réaffectant une valeur qui dépend d'elle-même.

▪ **Exercice 1.1.**
Le code Python affecte plusieurs valeurs successives à `a` en réalisant plusieurs affichages (la fonction «`print`» permet de réaliser un affichage en Python).

3. L'adjectif flottant qualifie en réalité la virgule dont la position n'est pas fixée (et peut donc «*flotter*») en mémoire pour ce type de variables.

```python
a = -3
a = 1
print(a)
a = a+1
print(a)
```

Que va renvoyer la Console Python lorsque l'on exécute ce code ? ■

3. Opérations élémentaires

À chaque type est associé un ensemble d'opérations. Vous trouverez ici la liste des opérations les plus élémentaires associées aux quatre types décrits précédemment. Tous les types admettent également deux opérations universelles qui sont l'opération de ***test d'égalité*** et l'opération de ***test de différence***. L'opération de test d'égalité, notée « = » en pseudo-code et « == » en Python, permet de vérifier si deux variables sont égales. L'opération de test de différence, notée « ≠ » en pseudo-code et « != » en Python, permet de vérifier si deux variables sont différentes.

■ **Exercice 1.2.**
Le code Python suivant affecte la valeur 2 à la variable `a` et la valeur −3 à la variable `b`.

```python
a = 2
b = -3
print(a == b)
print(a != b)
```

Que va renvoyer la Console Python lorsque l'on exécute ce code ? ■

Opérations sur les flottants

Pour les flottants, on peut utiliser les opérations classiques entre deux nombres : l'addition, la soustraction, la multiplication, la division, l'élévation à la puissance ; ainsi que les comparateurs d'ordre : inférieur ou égal, inférieur strict, supérieur ou égal, supérieur strict. La liste des symboles utilisés en Python figure dans le tableau ci-dessous :

Opération mathématique	Expression Python
$a+b$	`a + b`
$a-b$	`a - b`
$a\times b$	`a * b`
$a\div b$	`a / b`
a^b	`a ** b`
$a\leq b$	`a <= b`
$a<b$	`a < b`
$a\geq b$	`a >= b`
$a>b$	`a > b`

Opérations sur les entiers

Parmi les opérations standards entre deux entiers, on retrouve toutes les opérations décrites pour les flottants auxquelles on peut ajouter le quotient et le reste par la division euclidienne (c'est-à-dire la division entière).

Opération mathématique	Expression Python
Quotient de la division entière de a par b	`a // b`
Reste de la division entière de a par b	`a % b`

▪ **Exercice 1.3.**
On considère le code Python suivant.

```
a = 3 - 1
b = 1 + 2
c = a ** 3
d = c / 5
e = c // 5
f = (a <= b)
```

Que valent les variables `a`, `b`, `c`, `d`, `e` et `f` ? ▪

Opérations sur les booléens

Les opérations standards sur les booléens sont :
— la ***négation*** associée au mot-clé «**non**» ;
— la ***conjonction*** associée au mot-clé «**et**» ;
— la ***disjonction*** associée au mot-clé «**ou**».
Elles sont implémentées en Python à l'aide des opérateurs «`not`», «`and`» et «`or`» respectivement. Si `a` et `b` sont deux variables booléennes, on a :
— **non**`(a)` vaut **vrai** si et seulement si `a` vaut **faux** ;
— `a` **et** `b` vaut **vrai** si et seulement si les deux variables `a` et `b` sont égales à **vrai** ;
— `a` **ou** `b` vaut **vrai** si et seulement si l'une au moins des deux variables `a` ou `b` vaut **vrai**.

▪ **Exercice 1.4.**
Analysez le code Python qui suit.

```
a = (0 < 1) or (0 > 2)
b = not (1 < 2)
c = a and b
```

Quelles valeurs prennent les variables booléennes `a`, `b` et `c` ? ▪

Opérations sur les chaînes de caractère

Il existe un certain nombre d'opérations sur les chaînes de caractère en Python mais nous n'en considérons qu'une ici : la ***concaténation***. La concaténation permet de créer une nouvelle chaîne de caractère à partir de deux chaînes de caractère `a` et `b`, en mettant les caractères de `b` à la suite des caractères de `a`. La concaténation de `a` et `b` en Python se note `a + b`.

▪ Exercice 1.5.
On considère le code Python suivant.

```
a = "math"
b = a + "ematiques"
c = "ha"
c = c + c
```

Que valent les variables a, b et c ? ▪

4. Instructions conditionnelles

R Un père dit à son enfant : "**Si** tu finis tes légumes, **Alors** tu auras un dessert". L'enfant ne finit pas ses légumes mais le père donne quand même à son enfant un dessert. Le père a-t-il menti ?

La réponse est non. Et ce n'est pas pour des raisons d'éducation ou de morale. Le père a dit ce qu'il ferait si son enfant finit son dessert mais il n'a rien dit si il ne finissait pas son dessert.

Dans un algorithme, on peut choisir qu'une instruction ne s'exécute que si certaines conditions sont remplies. En pseudo-code, on peut utiliser les mots-clés **si** et **alors** pour indiquer une condition et la séquence d'instructions à exécuter si la condition est remplie. On peut également utiliser le mot-clé **Fin si** pour indiquer la fin de la séquence d'instructions conditionnelles. Une instruction conditionnelle peut donc s'écrire en pseudo-code sous la forme suivante :

```
Si la condition est vraie
    Alors faire
    |instructions
Fin Si
```

En Python, on utilise la commande «`if`» associé à l'utilisation des deux points «`:`» et d'une ***indentation*** (décalage vers la droite des lignes d'instructions [4]), selon la forme suivante :

```
if condition :
    instructions
```

Attention, les règles pour les deux points et l'indentation ne sont pas facultatives et le code ne fonctionnera pas correctement si elles ne sont pas respectées. Par ailleurs, l'indentation doit courir sur l'ensemble des instructions couvertes par la condition (donc éventuellement sur plusieurs lignes).

4. L'indentation en Python peut correspondre à une tabulation ou à un nombre fixe d'espaces. Toutefois, il est important qu'une même indentation soit utilisée pour un même groupe d'instructions. Dans l'usage, une indentation formée de 4 espaces est priviligiée.

On peut également différencier entre un certain nombre de cas en utilisant les mots-clés « **sinon si** » et « **sinon** » en pseudo-code ou les commandes « `elif` » et « `else` » en Python, selon le schéma qui suit :

```
Si la condition1 est vraie
    Alors faire
    |instructions1
Si la condition2 est vraie
    Alors faire
    |instructions2
Si la condition3 est vraie
    Alors faire
    |instructions3
Sinon
    Faire
    |instructions4
Fin si
```

```
if condition1 :
    instructions1
elif condition2 :
    instruction2
elif condition3 :
    instructions3
else :
    instructions4
```

▪ Exemple 1.3.
Deux joueurs s'affrontent aux dés. Le joueur qui obtient la plus grande valeur a gagné. Le programme suivant récupère la valeur obtenue par chacun des joueurs puis annonce qui a gagné la partie (on utilise pour cela les fonctions «`int`» et «`input`» de Python qui seront présentées en détails lors de la semaine **??**).

```
d1 = int(input("Entrez la valeur obtenue par le joueur 1."))
d2 = int(input("Entrez la valeur obtenue par le joueur 2."))
if d1 > d2 :
    print("Le joueur 1 a gagne.")
elif d2 > d1 :
    print("Le joueur 2 a gagne.")
else:
    print("Match nul!")
```

▪

5. Boucles

En algorithmique, les ***boucles*** permettent de répéter une séquence d'instructions sans avoir à réécrire la séquence. On distingue deux formes de boucles :

— les ***boucles bornées*** «**pour**», pour lesquelles la répétition de la séquence d'instruction correspond au parcourt de tous les éléments d'un ensemble fini par une variable ;
— les ***boucles non bornées*** «**tant que**», pour lesquelles la répétition de la séquence d'instruction est soumise à condition de répétition à chaque tour de la boucle.

Les boucles bornées sont appelées ainsi parce qu'elles s'arrêtent quand on arrive au bout de l'ensemble fini correspondant. Les boucles non bornées sont appelées ainsi parce qu'elles ne s'arrêtent que si la condition de répétition est fausse.

La boucle bornée «pour»

Dans ce cours, on ne considère que les boucles «**pour**» où la répétition de la séquence correspond au parcours par une variable d'un intervalle d'entiers (par exemple, $\{4,5,6,7,8,9\}$) et, le plus souvent, un intervalle d'entiers débutant en 0 (par exemple, $\{0,1,2,3,4,5,\}$). En pseudo-code, on pourra utiliser le mot-clé «**pour**» en début de boucle et indiquer la fin de cette boucle par un «**fin pour**».

```
1 Pour i allant de 1 à n
2     Faire
3     |instructions
4 Fin pour
```

En Python, on utilise la commande «`for`», associée à une instruction de la forme «`i in range(n)`» qui signifie que `i` parcourt l'intervalle $\{0,1,2,...,n-1\}$, et suivie d'un deux points. Une indentation permet ensuite d'indiquer les instructions correspondant à la séquence d'instructions à répéter.

```
1 for i in range(n) :
2     instructions
```

La boucle non bornée «tant que»

Dans la boucle «**tant que**», la boucle se répète tant qu'une condition de répétition est vraie. En pseudo-code, on peut indiquer le commencement d'une telle boucle par «**tant que**» et la fin des instructions de la boucle par «**fin tant que**».

```
1 Tant que la condition est vraie
2     Faire
3     |instructions
4 Fin tant que
```

En Python, la boucle «**tant que**» passe par la commande «`while`», associée à une condition, et suivie d'un deux points. Comme usuellement en Python, une indentation permet d'indiquer les instructions correspondant à la séquence d'instructions à répéter.

```
1 while condition :
2     instructions
```

Attention, si la condition est toujours vérifiée, la boucle ne s'arrêtera jamais. On dit que le programme tourne en ***boucle infinie***. Généralement, ce n'est pas souhaitable. Il faut donc s'assurer que la condition de répétition puisse passer de la valeur **vrai** à la valeur **faux** lors de la séquence d'instructions de la boucle.

▪ Exercice 1.6.
Dans le programme Python qui suit, l'une des deux boucles tourne en boucle infinie. Laquelle ?

```
1 i = 0
2 while i >= 0 :
3     i = i + 1
4     print(i)
```

```
1 i = 10
2 while i >= 0 :
3     i = i - 1
4     print(i)
```

Décrire pas-à-pas les opérations qui sont effectuées lors de l'exécution de chacune de ces boucles. ▪

On remarquera que dans l'exercice précédent, la variable `i` est affectée à une valeur avant le début de la boucle. Ceci est un schéma classique en programmation qui s'appelle l'***initialisation***. On dit, par exemple, que la variable `i` est initialisée à 0 avant la première boucle et initialisée à 10 avant la deuxième boucle.

6. Erreurs et bugs

— *SyntaxError* : Les erreurs de Syntaxe sont dues au non-respect des règles d'écriture de Python, comme un oubli de parenthèse ou un manque de " :"
— *NameError* : Les erreurs de définition sont dues à l'usage d'un nom qui n'a pas été défini, par exemple une variable, fonction qui n'a pas été définie.
— *TypeError* : Les erreurs de type sont dues à des valeurs dont le type n'est pas compatible avec l'expression dans laquelle elles apparaissent.
— **Erreur d'exécution** : les erreurs sont dues à des instructions que l'ordinateur ne sait pas exécuter comme une division par zéro.
— **Erreur de logique** : Ces dernières n'occasionnent pas en général pas de message d'erreur car il s'agit d'erreurs dans logique du programme, qui n'enfreignent pas les règles de Python. Ce sont les erreurs les plus difficiles à résoudre. Une erreur de logique courante est la **boucle infinie** : le programme ne s'arrête jamais car la condition d'une boucle non bornée ne devient jamais fausse. Dans ce cas, il faut en général interrompre le programme "de force" en tapant **Control-C**.

Exercices non à soumettre

■ **Exercice 1.7.**
On considère les trois séquences d'instructions suivantes en pseudo-code.

```
1 x ← x+1
2 b ← x^2
3 a ← x-1
4 c ← a^2
5 x ← b-c
```

```
1 x ← x-1
2 a ← x^2
3 x ← x+2
4 b ← x^2
5 x ← a+b
```

```
1 a ← x-1
2 b ← a^2
3 c ← x+1
4 d ← c^2
5 x ← b+d
```

1. On suppose que la variable `x` contient la valeur 2 avant l'exécution de la séquence. Dans chacun des cas, déterminez la valeur dans `x` après exécution de la séquence.
2. Identifiez la ou les séquences pour lesquelles, si `x` est initialisée à `a` avant l'exécution de la séquence, alors `x` contient $(a-1)^2 + (a+1)^2$ après exécution de la séquence.

■

■ **Exercice 1.8.**
Aladin propose le code Python suivant pour échanger la valeur de deux variables `a` et `b` :

```
1 a = 42
2 b = 23
3 a = b
4 b = a
```

1. Expliquer pourquoi le code fourni par Aladin ne remplit pas ses objectifs.
2. Proposer une autre solution qui échange effectivement les deux variables.

■

▪ Exercice 1.9.
On considère le code Python suivant :

```
x = int(input("Veuillez saisir un nombre entier."))
if x<100:
    print("Votre nombre est bien faible!")
else:
    print("Quel beau nombre!")
print("Au revoir!")
```

1. Que fait le programme si l'utilisateur rentre la valeur 15 ?
2. Et la valeur 1515 ?

▪

▪ Exercice 1.10.
À la fête foraine, l'accès aux montagnes russes est réservé aux personnes mesurant au moins 1m20 (compris) et au plus 2m10 (compris). Faites un programme Python qui demande à un utilisateur de rentrer sa taille en mètres et l'informe, en fonction de sa réponse, s'il :
— peut monter dans l'attraction ;
— est trop petit ;
— est trop grand.

▪

▪ Exercice 1.11.
Algorithme : Équation d'une droite passant par deux points donnés. L'objectif ici est de développer un programme Python qui renvoie l'équation d'une droite passant par deux points $A(x_A; y_A)$ et $B(x_B; y_B)$ dont les coordonnées dans un repère sont données.
1. Que se passe-t-il si A = B ?
2. Quelle est l'équation de la droite dans le cas où $x_A = x_B$ et $y_A \neq y_B$?
3. Donnez l'équation de la droite dans le cas où $x_A \neq x_B$.
4. Utilisez une instruction conditionnelle pour réaliser un programme Python qui affiche l'équation de la droite passant par A et par B.

▪

■ **Exercice 1.12.**
On considère le code Python suivant.

```
a = "A"
for k in range(2):
    a = a + "ha"
a = a +"!"
print(a)
```

1. Quelle valeur est contenu dans la variable a après exécution du programme.
2. Comment modifier le code pour produire un fou rire.

■

■ **Exercice 1.13.**
Pour chacun des programmes suivants, dire quelle valeur prend la variable d après exécution du programme.

```
a = 1
b = 1
c = 1
d = 0
for k in range(2):
    a = a+b
    b = b+a
    c = c+b
    d = d+c
```

```
a = 1
b = 1
c = 1
d = 0
for k in range(2):
    a = a+b
    b = b+a
    c = c+b
d = d+c
```

```
a = 1
b = 1
c = 1
d = 0
for k in range(2):
    a = a+b
    b = b+a
c = c+b
d = d+c
```

```
a = 1
b = 1
c = 1
d = 0
for k in range(2):
     a = a+b
b = b+a
c = c+b
d = d+c
```

■

▪ **Exercice 1.14.**
On considère la séquence d'instructions en pseudo-code qui suit.

```
1  x ← 3
2  y ← 11
3  k ← 1
4  Tant que x < y
5       Faire
6       | x ← 3x + 2
7       | y ← 2y + 1
8       | k ←  k + 1
9  Fin tant que
10 Afficher k
```

1. Quelle valeur est affichée lorsque l'on exécute cette séquence ?
2. Implémentez cette séquence en Python et exécutez votre code afin de vérifier votre réponse.

▪

▪ **Exercice 1.15.**
Un étudiant fauché place 10€ sur son livret A en 2020, rémunéré 0,5% d'intérêt par an. À partir de quelle année cet étudiant aura-t-il au moins 20€ sur son livret ? Réalisez un code Python qui permette de répondre à la question. ▪

▪ **Exercice 1.16.**

1. En utilisant une boucle «**pour**», rédigez un code Python qui vous permettra de remplir le tableau ci-dessus.
2. Même question avec une boucle «**tant que**».

i	-5	-4	-3	-2	-1	0	1	2	3	4	5
$\frac{i^3-4}{2}$											

▪

▪ **Exercice 1.17.**
Algorithme : Test de divisibilité. On se propose ici de définir un algorithme pour tester si un nombre naturel b est divisible par un nombre naturel non nul a.

1. Expliquez que b est divisible par a si et seulement si b est un multiple de a.
2. Donnez les quatre premiers multiples positifs de a dans l'ordre croissant.
3. Décrivez un algorithme qui utilise une boucle pour vérifier tous les multiples

de a dans l'ordre croissant, jusqu'à un certain point, pour savoir si b est un multiple de a . À partir de quand la boucle peut-elle s'arrêter pour conclure sur la divisibilité de b par a.
4. Implémentez votre algorithme en Python et testez le pour voir s'il fonctionne correctement.
5. Proposez un second programme Python qui aboutit au même résultat, sans utiliser de boucle, en utilisant l'opérateur %.

■

■ Exercice 1.18.

Algorithme : Plus grand multiple de a inférieur ou égal à b.

1. En vous inspirant de l'algorithme dans l'exercice précédent, définissez un algorithme qui permet de trouver le plus grand multiple d'un nombre naturel a inférieur ou égal à un nombre naturel b.
2. Généralisez votre algorithme au cas où a et b sont des nombres relatifs.
3. Implémentez votre algorithme en Python.
4. Proposez un second programme Python qui aboutit au même résultat, sans utiliser de boucle, en vous servant de l'opérateur //.

■

■ Exercice 1.19.

texte En utilisant le package random (taper *import random*) et la commande random.randint(a,b) qui tire un nombre au hasard en a et b. Faites une boucle conditionnelle tirant au hasard un nombre entre 0 et 50 jusqu'à être le numéro 18. On affiche le numéro dans la boucle. (Pour les NSI, cette boucle est elle infini et pourquoi ?) Faire une boucle qui tire des noms au hasard jusqu'à avoir un nombre pair. Faire une boucle où le nombre augmente de 3 en 3 jusqu'à dépasser le numéro 1802.

■

■ Exercice 1.20.

tt

1. Faire une boucle qui affiche les noms de toute votre famille
2. Faire une boucle qui compte tous les nombre de 1 à 1000
3. Faire une boucle qui dit le nombre de lettres pour le nom de chaque personne de votre famille (utiliser la fonction len())

■