COURS ACCELERE D'ALGORITHMIQUE

Objectifs : ce cours a pour but de présenter les fondamentaux en algorithmique au niveau du lycée : y seront traitées notamment les notions d'entrées et de sorties de données, de boucle et d'instruction conditionnelle. On utilisera le langage naturel pour décrire les algorithmes employés ; leur mise en œuvre s'effectuera via votre calculatrice ou sur ordinateur en utilisant le langage Python.

Sommaire

Pour la première année depuis la réforme de la classe de seconde GT en 2009, l'algorithmique est au programme du bac. Ne la négligez pas ! Vous aurez au moins une question cette année !

Bon, rappelons quand même une définition plausible de l'algorithmique :

Définitions : On peut définir un **algorithme** comme une suite d'instructions précises, à appliquer dans un ordre précis, pour arriver en un nombre fini d'étapes à un certain résultat. **L'algorithmique** est la science des algorithmes (création, amélioration...)

Exemple : préparer une recette de cuisine !

CHAPITRE 1 : VARIABLES INFORMATIQUES ENTREES ET SORTIE DE DONNEES

Considérons l'exemple simple suivant consistant à écrire un algorithme qui étant donné la longueur et la largeur d'un rectangle (disons exprimées en cm) renvoie son aire en cm².
Nous aurons besoin de 3 variables :

1. Une variable **longueur** saisie par l'utilisateur (donc *variable d'entrée*)
2. Une variable **largeur** saisie par l'utilisateur (donc également une *variable d'entrée*)
3. Enfin une variable **aire** renvoyée par l'ordinateur (donc *variable de sortie*)

L'algorithme pourrait se formuler ainsi :

écrire « Saisir la longueur du rectangle »
lire longueur
écrire « Saisir la largeur du rectangle »
lire largeur
aire ← longueur*largeur
écrire aire

La valeur affectée à la variable aire dépend des valeurs affectées aux variables longueur et largeur et est donnée par la formule bien connue longueur*largeur

Exercice 1 : écrire un algorithme en langage naturel qui étant donné un triangle ABC rectangle en A, demande à l'utilisateur de saisir les côtés AB et AC et renvoie la valeur de l'hypoténuse BC.

Exercice 2 : écrire un algorithme en langage naturel qui étant donné un triangle équilatéral ABC renvoie la mesure de sa hauteur.

SYNTHESE

Définitions

1. Dans un programme, une **variable** correspond à un emplacement de la mémoire de la calculatrice ou de l'ordinateur. Elle est repérée par un nom et contient une valeur.
2. La notion **d'affectation** (ou d'assignation) désigne l'opération par laquelle on établit un lien entre le nom de la variable et sa valeur (son contenu). On l'a noté ici avec le symbole ←
3. **L'entrée des données** (ou lecture des données) est l'opération qui consiste à saisir des valeurs pour qu'elles soient utilisées par le programme. On la note « **Saisir** valeur » ou « **lire** valeur ». *Remarque* : une valeur peut être un nombre entier, un nombre réel, une chaîne de caractères...
4. **La sortie des résultats** (ou écriture des résultats) permet d'afficher les valeurs des variables après traitement. On note cette instruction « **Afficher** valeur » ou « **écrire** valeur »

CHAPITRE 2 : NOTION DE BOUCLE

Premiers exercices

Dans les deux exercices qui suivent, vous pouvez utiliser les *instructions* suivantes :

1. **Avancer** (d'une longueur donnée)
2. **Tourner à gauche ou à droite** (d'un angle donné en degré)

On suppose que le « crayon » se déplace par défaut horizontalement et vers la droite au départ.

Exercice 1 : Écrire un programme permettant de construire un carré de côté 50 tout en revenant à la position de départ.

Exercice 2 : Écrire un programme permettant de construire un triangle équilatéral de côté 100 tout en revenant à la position de départ.

Remarque : Dans l'exercice 1 comme dans le suivant, un même *bloc d'instructions* a été répété plusieurs fois. Repérer lequel dans chacun des deux cas.

Cette répétition des mêmes instructions un certain nombre de fois peut être « résumée » en introduisant une notion très importante en programmation, et très économique au niveau du nombre de lignes à écrire.

Les instructions répétitives : notion de boucle

Il existe essentiellement deux méthodes pour écrire une **boucle**, c'est-à-dire **un procédé qui permet la répétition un certain nombre de fois d'un même processus** (addition, multiplication, etc...).

Afin de les mettre en pratique, nous allons résoudre l'exercice 1 de cette manière.

METHODE 1

Analysons le programme en langage naturel suivant :

i ← 0 **Tant que** i<4 (ou i<=3) faire avancer de 50 tourner à gauche de 90° i ← i + 1 Fin Tant que	**J'affecte** à i la valeur 0 Tant que la condition annoncée est vraie, on effectue les deux instructions qui suivent On réaffecte à i sa valeur précédente augmentée de 1

Quel est le résultat obtenu ?
On peut alors résumer la structure de boucle décrite ici de la manière suivante :

Tant que condition vérifiée faire

Bloc d'instructions
Fin Tant que

Remarques : Si la condition n'est pas vérifiée au début, alors le bloc d'instructions ne sera pas exécuté du tout.
ATTENTION ! Pour que l'algorithme soit correct, il est nécessaire que la condition cesse d'être vérifiée au bout d'un nombre fini de répétitions. Sinon, le programme « boucle indéfiniment ».

Exercice 3 : Que fait le programme suivant ?

S ← 80
Tant que S>20 faire
    S ← S + 10

METHODE 2

Une autre manière courante d'écrire une boucle est :

Pour i variant de 0 à 3 (ou de 1 à 4) faire avancer de 50 tourner à gauche de 90° Fin Pour	La variable i évolue comme indiqué, par défaut de 1 en 1. Les deux instructions suivantes sont alors effectuées

Le résultat obtenu est exactement le même : le dessin d'un carré de côté 50.

On peut alors résumer la structure de boucle décrite ici de la manière suivante :

Pour i variant de 0 à N (ou de 1 à N) faire

    Bloc d'instructions
Fin Pour

Problème type : on veut calculer la somme $S = 1+2+3+...+99+100$.

Analyse : Bien évidemment, on ne va pas saisir à la machine cette longue opération ! Il nous faut donc trouver un moyen pour que cette dernière l'exécute, mais en écrivant le moins de lignes possible.
On commence par remarquer que l'opération qui est sans cesse utilisée est l'addition. **D'où l'idée d'utiliser une boucle** : on répète plusieurs fois le même procédé : ici additionner.
Additionner oui, mais quoi ? 1 d'abord, puis 2, puis 3, … , jusqu'à 100. Autrement dit on additionne une suite de nombres qui « varient », dans le cas présent de 1 en 1.

L'idée essentielle est de calculer S petit à petit, comme on le ferait à la main en ajoutant peu à peu tous les termes contenus dans l'addition.
Comme S va évoluer et les termes qui la composent aussi, **on a l'idée d'introduire deux variables** : S elle-même et un « compteur » i, qui devra prendre successivement les valeurs 1, 2, 3, …, 100.

Nous allons résoudre ce problème en créant un algorithme efficace, utilisant (mais ce n'est pas une obligation), le premier type de boucle.

Mise en œuvre **:**

Instructions	Signification
S ← 0 i ← 1 **Tant que** i ≤ 100 faire S ← S + i i ← i + 1 Fin Tant que **Afficher** S	On **affecte** à S la valeur 0 (valeur initiale) On **affecte** à i la valeur 1 (valeur initiale) La condition est posée: i varie jusqu'à 100 On réaffecte à S sa valeur précédente plus la valeur actuelle de i On réaffecte à i sa valeur augmentée de 1 Une fois sorti de la boucle, on affiche S

Vous allez à présent « faire tourner l'algorithme à la main », c'est-à-dire regarder, étape par étape l'évolution des variables i et S, et enfin le résultat final. Pour cela, compléter le tableau suivant. Vous ne donnerez pas le résultat du calcul de S à chaque étape, mais seulement l'écriture de S sous la forme d'une somme.

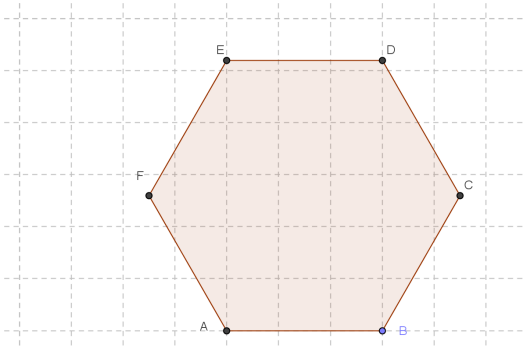
Après l'itération	**i**	**S**
0	1	0
1		
2		
3		
...		
99		
100		

Quelle est la dernière valeur prise par la variable S ? Quel est le résultat affiché par le programme ?

Exercice 5 : Écrire un programme utilisant une boucle **Tant que** ou une boucle **Pour** calculant et affichant le résultat de :

1. $S = 3+6+9+...+201$
2. $P = 2\times4\times6\times8\times...\times40$
3. $S = 1+\frac{1}{2}+\frac{1}{3}+...+\frac{1}{10}$

Exercice 6 : Construire la figure suivante en utilisant l'instruction **Tant que** (longueur d'un côté du quadrillage 50).On partira du point A.



<u>Exercice 7</u> : En utilisant une boucle **Tant que**, écrivez un programme permettant de calculer puis d'afficher le résultat de la somme suivante :

$$T = 1+\frac{1}{2}+\frac{1}{2\times 3}+\frac{1}{2\times 3\times 4}+\ldots\frac{1}{2\times 3\times 4\times \ldots\times 10}$$

Indication : bien analyser de combien de variables vous avez besoin.

<u>Exercice 8</u> :

En utilisant l'instruction **Tant que**, soit une fois soit deux fois dans le même programme, obtenir le dessin suivant :

Carré de côté 10 et intervalle entre les carrés 10.

Vous pouvez encore utiliser les *instructions* suivantes :

- **Avancer** (d'une longueur donnée)
- **Tourner à gauche ou à droite** (d'un angle donné en degré)

Vous disposez en plus de deux instructions :
lever le crayon (permet d'avancer sans que ceci dessine quoi que ce soit)
baisser le crayon (permet de redessiner à nouveau)

CHAPITRE 3 : NOTION D'INSTRUCTION CONDITIONNELLE

Nous avons vu à la séance précédente la notion de boucle, utilisée dans le cas où une même suite d'opérations est répétée un certain nombre de fois. Nous allons nous intéresser ici au cas d'instructions conditionnelles.
Cette notion permet à un algorithme d'effectuer certaines instructions seulement **si une certaine condition est remplie**.
Ceci prend la forme suivante en pseudo-langage :

```
Si (condition vérifiée) faire
        Bloc d'instructions
Fin Si
```

Dans le cas où la condition de départ n'est pas vérifiée, il est également possible d'indiquer à l'algorithme une alternative qu'il doit effectuer. On obtient alors la structure suivante :

```
Si (condition vérifiée) faire
        Bloc d'instructions n°1
Sinon faire
        Bloc d'instruction n°2
Fin Si
```

Exercice 1
Écrire dans la zone de texte adjacente le résultat affiché à l'écran par l'ordinateur pour chacun des programmes suivants :

Question 1

```
N ← 4
a ← 7
Si N<a faire
        écrire « Eh »
Sinon
        écrire « Oh »
Fin Si
```

Question 2

```
i ← 1
Tant que i<5 faire
        Si i est divisible par 2 faire
                écrire 3*i
        Sinon
                écrire i
        i ← i + 1
        Fin Si
Fin Tant que
```

Exercice 2

Écrire un programme qui demande à l'utilisateur d'écrire son sexe à l'écran : **M** pour masculin ; **F** pour féminin.

a) Si l'utilisateur saisit M, l'ordinateur affichera : Bonjour monsieur !

b) Si l'utilisateur saisit F, l'ordinateur affichera : Bonjour mademoiselle !

Le symbole de comparaison égalité se note =

Vous avez le droit aux instructions suivantes :

- **Si** … **Sinon**
- **écrire** (un texte apparaissant à l'écran)
- **lire** (un texte sur l'écran de l'ordinateur)

Notons le cas où **la condition de départ peut être constituée de plusieurs autres conditions qui doivent être vérifiées simultanément**. Nous obtenons alors une suite d'instructions conditionnelles imbriquées. Il y a essentiellement deux structures pour décrire ceci.

Méthode n°1	Méthode n°2
Si (condition 1 vérifiée) faire **Si (condition 2 vérifiée)** faire **Si (condition 3 vérifiée)** faire … **Si (condition k vérifiée)** faire Bloc d'instructions	**Si (condition 1 vérifiée ET condition 2 vérifiée ET… ET condition k vérifiée)** faire Bloc d'instructions Fin Si

Exercice 3

Un automobiliste veut louer une grosse cylindrée dans une agence. Ceci n'est possible que s'il a 25 ans ou plus **et** au minimum 5 ans de permis. Écrire un programme qui demande à l'utilisateur son âge, puis le nombre d'années qu'il possède son permis et renvoie le résultat « Location acceptée » ou « Location refusée » selon les réponses saisies.

Analyse : L'utilisateur doit vérifier deux conditions pour que sa location soit acceptée : « Avoir un âge au moins égal à 25 ans » **ET** « posséder le permis depuis au moins 5 ans ».

Vous avez le droit aux instructions suivantes :

- **Si** … **Sinon**
- **écrire** (un texte apparaissant à l'écran)
- **lire** (une valeur sur l'écran de l'ordinateur)

On dispose aussi de l'opérateur logique **OU** au sens mathématique du terme (ce n'est pas un OU exclusif).

Exercice 4 : Déterminer si une année (dont le millésime est introduit par l'utilisateur) est bissextile ou non. (Une année A est bissextile si A est divisible par 4. Elle ne l'est cependant pas si A est un multiple de 100, à moins que A ne soit multiple de 400).

Vous disposez des mêmes instructions qu'à l'exercice précédent ainsi que de l'instruction : **est divisible par** et de son contraire.

Dans certains cas, il se peut que plusieurs alternatives soient possibles si la condition initiale n'est pas vérifiée. On obtient alors la structure suivante :

Si (condition 1 vérifiée) faire
    Bloc d'instructions n°1
Sinon si (condition 2 vérifiée) faire
    Bloc d'instruction n°2
…etc...
Sinon si (condition N-1 vérifiée) faire
    Bloc d'instructions n° N-1
Sinon faire
    Bloc d'instruction n° N
Fin Si

Remarque : **L'écriture « Sinon si » signifie que les conditions 1, 2,..., N-1 ne sont jamais vérifiées simultanément deux à deux**. On a une **partition** des cas possibles.

Exemple : le fait que la condition 3 soit vérifiée implique nécessairement que les conditions 1 et 2 ne l'aient pas été.

Exercice 5 : Convertir une note scolaire N quelconque (sur 20), entrée par l'utilisateur sous forme de points (par exemple 12,75), en une note standardisée suivant le code suivant :

Note	**Appréciation**
N >= 16	A
16 > N >=12	B
12 > N >= 10	C
10 > N >= 8	D
N < 8	E

Vous avez le droit aux instructions suivantes :

- **Si** … **Sinon si … Sinon**
- **écrire** (un texte apparaissant à l'écran)
- **lire** (une valeur sur l'écran de l'ordinateur)

CHAPITRE 4 : PYTHON EN 15 MINUTES

Jusqu'à présent, nous avons conçu des algorithmes, c'est-à-dire une suite d'instructions bien précises pour résoudre un problème posé. Cependant un algorithme n'est pas un programme. Ce-dernier doit être écrit dans un langage de programmation que l'ordinateur interprétera. Nous utiliserons le langage Python et je vous invite fortement à traduire les scripts donnés dans celui de votre calculatrice.

En séance de TP, je détaillerai comment utiliser Python en mode interactif ou comment conserver vos scripts pour une utilisation future. Voici les principales commandes à connaître :

Affectation-réaffectation

Nous avons vu en cours l'opération **d'affectation**, qui consiste à donner à une *variable* une valeur (nombre entier, réel, chaîne de caractères, etc...).

En pseudo-langage, l'opération d'affectation a été symbolisée par le sigle "←"
Exemple : a←5 # J'affecte à la variable a la valeur 5

En langage Python l'opération d'affectation s'écrit « = ».

Ne croyez pas que cette affectation soit définitive. On peut **réaffecter** à la variable « a » une nouvelle valeur en réutilisant le symbole « = ».

Typage des variables

En Python, nous travaillerons avec 4 types de variables :

1. Les entiers relatifs : type **int** (comme integer)
   Exemple : en saisissant a=-4, vous définissez a comme une variable de type string
2. Les nombres flottants : type **float**
   Exemple : en saisissant b=5.32, vous définissez b comme une variable de type float
   Attention, en Python le séparateur décimal est un point et non une virgule !
3. Les chaînes de caractères : type **str** (string)
   Exemple : en saisissant c='salut', vous définissez c comme une variable de type str
4. Les listes : type **list**
   Exemple : en saisissant d=[12,3.14,'hello',100], vous définissez d comme une variable de type list.

Opérateurs de comparaison

<		strictement inférieur
>		strictement supérieur
<=		inférieur ou égal
>=		supérieur ou égal
<>	ou !=	différent
==		égale

Remarque : notez bien la différence entre le symbole d'affectation = et l'égalité mathématique ==

Écrire une boucle « Tant que » avec Python

Syntaxe en langage usuel	Syntaxe en Python
Tant que condition vérifiée faire **bloc d'instructions** Fin Tant que	**while condition vérifiée :** **bloc d'instructions**

Remarques :

- **Les : sont obligatoires après la condition associée à while.**
- **l'indentation du bloc d'instructions associé à while est obligatoire.**

Utilisation du module turtle

Toute utilisation des outils graphique du module turtle nécessite qu'on l'ait « appelé » par la commande : **from turtle import *** avant toute chose. On redonne les commandes usuelles pour tracer des figures :

reset()	On efface tout et on recommence
goto(x,y)	Aller à l'endroit(x,y)
forward(x)	Avancer de la distance x
backward(x)	Reculer de la distance x
up()	Relever le crayon (pour pouvoir avancer sans dessiner).
down()	Abaisser le crayon pour recommencer à dessiner.
left(x)	Tourner à gauche d'un angle en degrés égal à x.
right(x)	Tourner à droite d'un angle en degrés égal à x.

Écrire une instruction conditionnelle « Si...alors » avec Python

Syntaxe en langage usuel	Syntaxe en Python
Si condition vérifiée faire **bloc d'instructions** Fin Si	**if condition vérifiée:** **bloc d'instructions**

Remarques :

- **Les : sont obligatoires après la condition associée à if.**
- **l'indentation du bloc d'instructions associé à if est obligatoire.**

Écrire une instruction conditionnelle « Si...alors...sinon » avec Python

Syntaxe en langage usuel	Syntaxe en Python
Si condition vérifiée faire **bloc d'instructions 1** **Sinon** faire **bloc d'instructions 2** Fin Si	**if condition vérifiée:** **bloc d'instructions 1** **else:** **bloc d'instructions 2**

Remarques :

- **Les : sont obligatoires après la condition associée à if ainsi qu'après else.**
- **l'indentation du bloc d'instructions associé à if est obligatoire ; même chose pour celui associé à else.**
- **if et else sont indentés identiquement**

Écrire une instruction conditionnelle « Si...Sinon si...sinon » avec Python

Syntaxe en langage usuel	Syntaxe en Python
Si (condition 1 vérifiée) faire bloc d'instructions n°1 **Sinon si (condition 2 vérifiée)** faire bloc d'instructions n°2 …etc... **Sinon si (condition N-1 vérifiée)** faire bloc d'instructions n° N-1 **Sinon** faire bloc d'instruction n° N Fin Si	**if (condition 1 vérifiée):** bloc d'instructions n°1 **elif (condition 2 vérifiée):** bloc d'instructions n°2 ...etc... **elif (condition N-1 vérifiée):** bloc d'instructions n° N-1 **else:** bloc d'instruction n°N

Remarques :

- **elif signifie sinon si**
- **Les : sont obligatoires après la condition associée à if ainsi qu'après celles associées aux elif et à else.**
- **l'indentation du bloc d'instructions associé à if est obligatoire ; même chose pour celui associés aux elif et à else.**
- **if, elif et else sont indentés identiquement**
- **Toutes les conditions énoncées sont par construction incompatibles deux à deux.**

Entrées de données : un résumé

La commande de base est l'instruction input() qui attend que l'utilisateur saisisse une donnée à l'écran. Attention, en Python (version 3.1), tout ce qui est saisi sera automatiquement interprété comme une variable de type str (chaîne de caractères). Si nous avons à saisir un nombre (entier ou flottant, il faudra composer avec une autre instruction). On retiendra ceci :

Pour saisir une chaîne de caractères :
c=input("Saisir votre texte ")

Pour saisir un entier :
d=int(input("Saisir un entier "))

Pour saisir un nombre flottant :
e=float(input("Saisir un nombre réel "))

Remarque : la commande input() permet de saisir un message de votre choix.

Sorties de données : un résumé

Pour écrire la valeur d'une variable saisie ou traitée auparavant :
print(variable)

Pour écrire du texte :
print("Texte") Le texte est entre guillemets

Pour mixer les deux :
print("Texte 1",variable 1,"Texte 2",variable 2, etc...)

CHAPITRE 5 : APPLICATIONS

Exercice 1 : un peu de travail sur les vecteurs...

Question 1 : On considère un repère $(O\ ;\vec{i}\ ;\vec{j})$. On se donne deux points A, B repérés par leurs coordonnées.

Écrire un programme qui étant donné les coordonnées x_A et y_A d'un point A, puis x_B et y_B d'un point B, renvoie celles du vecteur $\overrightarrow{AB}$ sous la forme d'une liste.

Question 2 : modifier le script précédent pour saisir en plus les coordonnées x_C et y_C d'un point C, puis x_D et y_D d'un point D, renvoie celles du vecteur $\overrightarrow{CD}$ sous la forme d'une liste. Ce programme déterminera ensuite si les droites (AB) et (CD) sont parallèles ou non.

Exercice 2 : Résolution d'une équation du second degré

Le but de cet exercice est de déterminer le nombre et une valeur numérique des solutions de l'équation du second degré $a\,x^2+b\,x+c=0$.

Écrire un programme en langage naturel et à l'aide du logiciel Python qui :

1. demande à l'utilisateur de saisir les coefficients a,b et c
2. détermine le nombre de solutions de $a\,x^2+b\,x+c=0$
3. précise le nombre de solutions, et dans le cas où elles existent, écrive ces solutions.

Exercice 3 : Méthode de dichotomie

Données : nous disposons d'une fonction f définie sur un intervalle $[a\,;b]$ à valeurs dans $\mathbb{R}$ telle que :

- f est strictement monotone sur $[a\,;b]$
- f est continue sur [a ; b]
- $f(a)\,f(b)<0$

Alors (théorème de la bijection) il existe un unique réel *c* appartenant à]a; b[tel que $f(c)\ =\ 0$.

Principe de la dichotomie : Supposons que l'on sache qu'il y a une racine x_0 dans l'intervalle $[a\,;b]$. Pour préciser sa valeur, nous allons découper $[a\,;b]$ de manière plus fine de sorte à encadrer avec la précision voulue le réel x_0 .

Posons $m\ =\ \dfrac{a+b}{2}$.

- Si $f(a)\,f(m)<0$ alors $x_0\in[a\,;m]$. On pose alors $a_1=a$ et $b_1=m$.
- Si $f(a)\,f(m)>0$ alors $x_0\in[m\,;b]$. On pose alors $a_1=m$ et $b_1=b$

On recommence en prenant l'intervalle $[a_1\,;b_1]$ et ainsi de suite. On obtient alors des intervalles $[a_2\,;b_2]$,..., $[a_n\,;b_n]$. Lorsque $\dfrac{|b_n-a_n|}{2}<p$, on a $\dfrac{a_n+b_n}{2}\ =\ x_0$ à p près.

1. Écrire un programme en Python qui étant donnés deux réels a et b (a<b) et une fonction f continue et strictement monotone sur [a ; b], donne une approximation de x_0 à une précision p près (p choisie par l'utilisateur). Pour simplifier on supposera f strictement croissante sur $[a\,;b]$.
2. Testez ce programme pour les fonctions suivantes :
   a) f est définie sur $[-5\,;10]$ par $f(x)\ =\ x^3+2\,x+1$
   b) f est définie sur $[-0{,}5\,;1]$ par $f(x)\ =\ x-\cos x$

Exercice 4 : Un peu de probabilités conditionnelles...
On considère un dé à 6 faces parfait et 3 urnes numérotées de 1 à 3.
L'urne 1 contient 3 boules blanches, 4 boules noires et 3 boules vertes.
L'urne 2 contient 2 boules blanches, 2 boules noires et 6 boules vertes.
L'urne 3 contient 4 boules blanches, 4 boules noires et 2 boules vertes.
Toutes ces boules sont indiscernables au toucher.
On considère l'expérience aléatoire qui consiste à lancer lancer d'abord le dé puis :

- Si le numéro est 1 ou 2, on tire une boule dans l'urne n°1
- Si le numéro est 3, 4 ou 5 on tire une boule dans l'urne n°2
- Sinon, on tire une boule dans l'urne n°3.

1. Modéliser cette situation à l'aide d'un arbre de probabilité. On pourra prendre comme événements : « U_1 » : obtenir 1 ou 2 ; « U_2 » : obtenir 3,4 ou 5 ; « U_3 » : obtenir 6, puis « B » : tirer une boule blanche ; « N » : tirer une boule noire et « V » : tirer une boule verte.
2. Écrire un algorithme modélisant le lancer du dé, suivi du tirage d'une boule dans l'une des 3 urnes et le programmer sous Python.
3. Faire tourner le programme n fois (n choisi par l'utilisateur) et relever la fréquence d'apparition d'une boule blanche, d'une boule noire et d'une boule verte. On testera avec n=50, 100 et 1000.
4. Compléter le tableau donné en annexe avec n=50000
5. Calculer p(B), p(N) et p(V) et comparer avec les résultats trouvés à la question précédente.

Remarque : Pour simuler le tirage au sort d'un entier aléatoire entre 1 et N, on importe la fonction randint de la bibliothèque random via la commande from random import * ; randint(a,b) renvoie un entier aléatoire entre a et b. L'instruction break sert à sortir d'une boucle.

Indication : On pourra numéroter les boules de chaque urne pour modéliser le tirage dans les urnes ; par exemple dans l'urne 1 on notera : 1,2,3 pour les boules blanches ; 4, 5, 6, 7 pour les boules noires et 8, 9, 10 pour les boules vertes.

Fréquence Expérience n°	f_B	f_N	f_V
1			
2			
3			
4			
5			
Moyenne des fréquences			

f_B est la fréquence d'apparition de l'événement B, etc...

Exercice 5 : Travail sur une suite.
On définit la *suite de Collatz* de la manière suivante :

- On se donne un entier u_0 strictement positif.
- Pour tout entier $n \geqslant 1$, on calcule u_{n+1} en fonction de u_n de la manière suivante :

  a) Si u_n est pair, on a $u_{n+1} = \frac{u_n}{2}$

  b) Si u_n est impair, on a $u_{n+1} = 3u_n + 1$

On donne ci-dessous un exemple traité avec le tableur. On a choisi comme valeur initiale $u_0 = 5$ et il est affiché les 21 premiers termes de cette suite.

n	u_n
0	5
1	16
2	8
3	4
4	2
5	1
6	4
7	2
8	1
9	4
10	2
11	1
12	4
13	2
14	1
15	4
16	2
17	1
18	4
19	2
20	1

1. Que remarquez-vous ?
2. Écrire en langage naturel puis en Python un programme demandant à l'utilisateur de :
   a) saisir à l'écran un entier u_0 strictement positif,
   b) de saisir un entier n (l'indice maximum de calcul des u_k)
   c) et qui affiche tous les termes de u_0 à u_n .
3. Tester votre programme avec $u_0 = 3$ puis $u_0 = 7$. (faire varier n également)
4. Que remarquez-vous ?

Remarques : 1) La suite (u_n) définie de la manière précédente est dite définie ***par récurrence***. On calcule chacun de ses termes de proche en proche.

2) On ne sait toujours pas prouver ce que vous avez constaté ! C'est un ***problème ouvert***.

Problème **(niveau terminale S)**

Soit la fonction f définie sur [0 ; +∞[par $f(x) = x^2 - a$ où a est un réel strictement positif donné. Le but de cet exercice est de trouver une « bonne approximation » de la solution x^* de l'équation $f(x) = 0$.

Partie 1

On choisit a = 2.

Soit . A le point de C d'abscisse 1.

1. Donner une équation de la tangente T à C au point A. Cette dernière coupe l'axe des abscisses en un point d'abscisse x_0 .
2. Après avoir calculé x_0 , donner une équation de la tangente T_0 à C au point d'abscisse x_0 . Cette dernière coupe l'axe des abscisses en un point d'abscisse x_1 . Calculer x_1

et comparer x_1 à x_0 .

3. On définit ainsi une suite $(x_n)_{n\geqslant 0}$ de réels de proche en proche. (on justifiera proprement son existence un peu plus loin) Soit n un entier naturel quelconque. Donner l'expression littérale de x_{n+1} en fonction de x_n .
4. Écrire un programme à l'aide du logiciel Python qui demande à l'utilisateur :
   - de saisir la valeur de l'entier n
   - renvoie la valeur de x_n défini précédemment.
5. Le tester avec n=2, n=3, n=10.

Partie 2

On souhaite répondre à deux questions :

- La suite $(x_n)_{n\geqslant 0}$ converge-t-elle ? Si oui, est-ce bien vers x^* ?
- Évaluer la qualité de l'approximation de x^* par x_n , autrement dit, pour une précision arbitraire p>0 fixée par l'utilisateur, combien nous faut-il d'itérations pour avoir $|x^*-x_n|<p$? **On restreint désormais f à l'intervalle I = [1 ; 2].**

Sous-partie A)

1. Quelle est la valeur exacte de x^* ? Justifier que $x^*\in[1;2]$.
2. En se servant du résultat du I)3) démontrer que pour tout entier naturel n $x_{n+1}=\frac{1}{2}\left(x_n+\frac{2}{x_n}\right)$. On posera $g(x)=\frac{1}{2}(x+\frac{2}{x})$.
3. Démontrer que $g(x)=x$ si et seulement si $f(x)=0$. (on s'est ramené à un problème de point fixe).
4. On admet le résultat suivant : Soit $(u_n)_{n\geqslant 0}$ une suite d'éléments d'un intervalle fermé I telle que pour tout entier naturel n, $u_{n+1}=g(u_n)$. On suppose que :
   - $(u_n)_{n\geqslant 0}$ converge vers un réel l
   - g est continue sur I
   - ALORS on a $l=g(l)$.

   a) Étudier les variations de g sur l'intervalle [1 ; 2].

   b) Justifier que tous les termes de la suite $(x_n)_{n\geqslant 0}$ sont bien définis et appartiennent à $I=[1;2]$ (le prouver par récurrence).

   c) Prouver que la suite $(x_n)_{n\geqslant 0}$ est décroissante. En déduire que $(x_n)_{n\geqslant 0}$ converge, et ce vers x^* .

Sous-partie B)

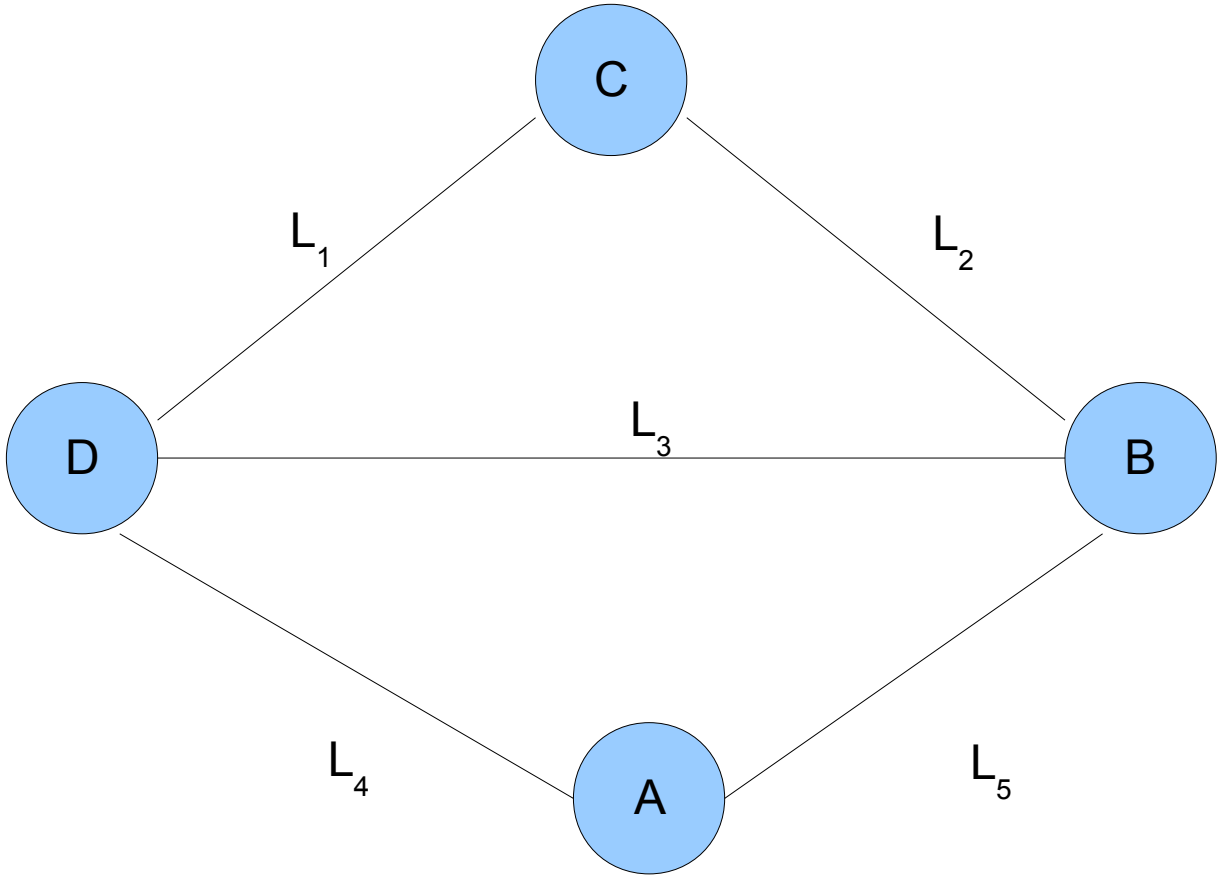
On pose pour tout entier naturel n, $e_n=x_n-x^*$.

5. Démontrer que $e_{n+1}=\frac{1}{2x_n}e_n^2$ puis que $e_{n+1}\leqslant\frac{1}{2}e_n^2$.
6. Démontrer par récurrence que pour tout entier naturel n que $e_n\leqslant\left(\frac{1}{2}\right)^{2^n}$
7. Soit p>0 la précision recherchée.. On supposera p<1. Déterminer un entier naturel N tel que si n>N, alors $e_n<p$.
8. Écrire un programme à l'aide du logiciel Python qui demande à l'utilisateur :
   - de saisir la précision recherchée p
   - renvoie la valeur de x^* à p près.

Remarque : Les résultats de la sous-partie B justifient à nouveau la convergence de la suite $(x_n)_{n\geqslant 0}$ vers x^* .

Exercice 6 : Modélisation d'un réseau d'informations ; variables aléatoires indépendantes, loi binomiale.

On considère un réseau de sommets ABCD (que l'on peut se représenter comme un réseau d'ordinateurs par exemple) reliés par des liaisons L_1 à L_5 comme représenté sur le schéma ci-dessous.



Ce réseau a pour objet de transmettre de l'information de A vers D. Ces informations transitent par n'importe quel chemin en fonctionnement : par exemple $L_5L_2L_1$ est un chemin possible et il fonctionne si et seulement si les trois liaisons L_5, L_2 et L_1 fonctionnent.

Chaque liaison a la même probabilité *p* de fonctionnement durant une unité de temps. Durant cette unité de temps, on considère qu'une liaison ne change pas d'état : ou bien elle reste en fonctionnement ou bien elle est en panne.

Les cinq liaisons sont **indépendantes** les unes des autres en ce qui concerne leurs pannes.

On dit que le réseau fonctionne durant une unité de temps si l'un des chemins possibles entre A et D a permis le passage de l'information entre ces points.

1. Déterminer en fonction de p la probabilité $f(p)$ que le réseau fonctionne durant une unité de temps. Vous raisonnerez formellement ou vous écrirez un script modélisant le fonctionnement du réseau.
2. Pour renforcer la fiabilité du réseau, on remplace chacune des liaisons simples par des liaisons multiples constituées de n brins en parallèle, ayant chacun la même probabilité β de fonctionnement durant une unité de temps. Une liaison multiple fonctionne si au moins l'un des brins la constituant fonctionne.
   a) On appelle A l'événement « une liaison multiple fonctionne » et pour une liaison multiple donnée on note X la variable aléatoire égale au nombre de brins qui la constitue qui fonctionnent.
   a) i) Justifier que X suit une loi binomiale dont on précisera les paramètres.

a) ii) En déduire p(A).

Remarque : comme A dépend de β et de n on écrira $p(A)=g(\beta,n)$.

b) En déduire la probabilité $h(\beta,n)$ de fonctionnement du réseau ainsi constitué.

c) Écrire un script qui demande à l'utilisateur :

- de saisir le nombre n de brins constituant chaque segment
- de saisir la probabilité β de fonctionnement de l'un des n brins d'un segment
- et qui renvoie la probabilité $h(\beta,n)$ de fonctionnement du réseau ainsi constitué.

Vous pouvez soit vous servir de la formule établie précédemment, soit modéliser le transit de l'information entre A et D. Il sera judicieux de créer une fonction **segment(beta,n)**.

Tester le programme avec quelques valeurs : $n=5$ et $\beta=0{,}25$, etc...

3. Posons α le coût unitaire des brins constituant les liaisons. Soit γ le gain par unité de temps de l'opérateur du réseau lorsque celui-ci fonctionne.

   a) Calculer l'espérance du gain de l'opérateur pour une unité de temps.

   b) Si $\alpha=5$, $\beta=0{,}2$ et $\gamma=500$, déterminer le nombre n de brins en parallèle nécessaire pour maximiser l'espérance de gain de l'opérateur. Vous pourrez vous aider d'un tableur au besoin.

Exercice7 : A l'aide d'un tableur ou de Python écrire l'algorithme de la méthode d'Euler permettant de résoudre l'équation différentielle : f est définie sur $[0;2]$ par $f'(x)=5f(x)+x^2$ et $f(0)=3$.