Des TICE pour faire des maths

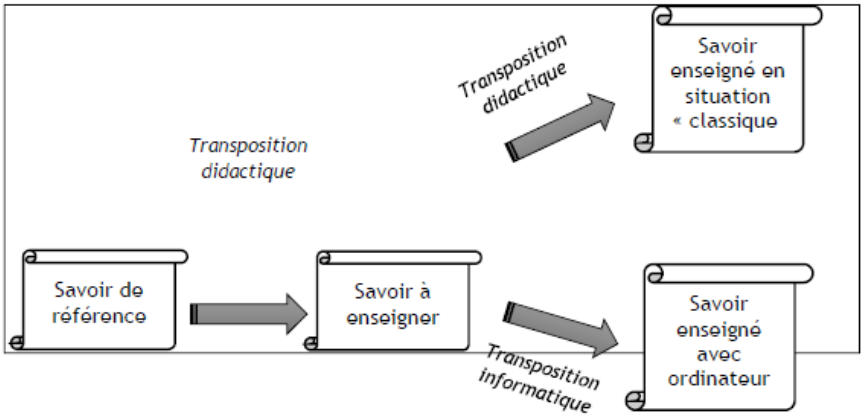
Table des matières

Introduction....2
L'anniversaire de Chloé....4
La marche du petit robot....8
Le jeu du lièvre et de la tortue....12
le ruban lumineux....16
L'escargot de Pythagore....19
Un classique : la suite de Syracuse....27

Introduction

Notre objectif, au travers de ces situations, n'est pas d'apprendre aux élèves à manipuler le tableur ni d'apprendre à coder en Python mais d'utiliser le tableur ou l'algorithmique pour aider à la résolution de problèmes mathématiques.

Nous sommes convaincus que vouloir introduire l'algorithmique et la programmation par le biais de la résolution de problèmes mathématiques double les difficultés puisqu'on sollicite alors deux expertises : l'expertise mathématique et l'expertise informatique. En effet, la résolution d'un problème mathématique par un programme informatique nécessite une double transposition didactique[1] :

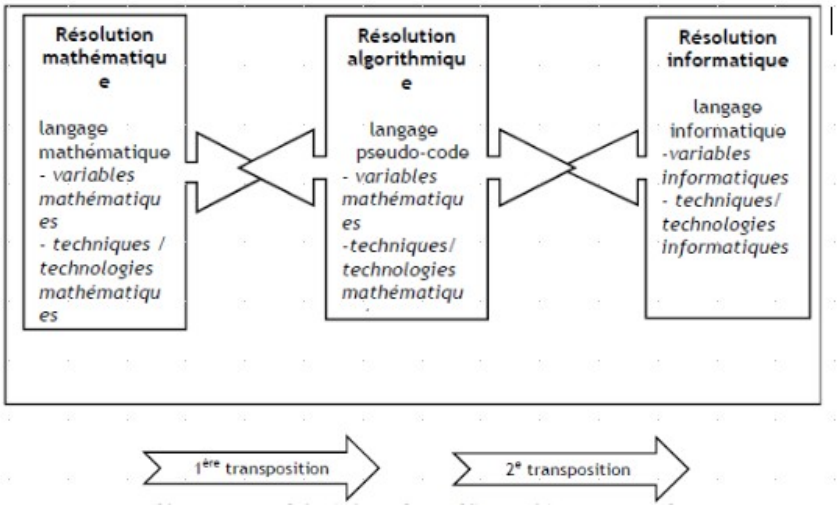


Transpositions didactique et informatique (Chevallard 1982, Balacheff 1994)

Balacheff (1994) explique qu'*aux contraintes de la transposition didactique s'ajoutent, ou plutôt se combinent, celles de modélisation et d'implémentation informatiques.*

Lors du colloque Espace Mathématique Francophone 2015, Nathalie Briant et Alain Bronner ont repris le concept de transposition informatique de Balacheff mais avec une adaptation, tenant compte de la singularité de l'algorithmique.
Ils voient émerger une double transposition, associée à des techniques différentes, justifiées par des technologies relevant du domaine mathématique, du domaine informatique, ou des deux conjointement.

1 La transposition didactique est l'activité qui consiste à transformer un objet de savoir savant en un objet de savoir à enseigner.



Double transposition de la résolution d'un problème mathématique en vue de sa programmation
(Nathalie Briant et Alain Bronner 2015)

Il nous semble donc nécessaire de distinguer deux phases dans l'apprentissage des TICE : la phase où l'élève s'approprie un nouveau langage et la phase où l'élève utilise l'outil pour résoudre un problème mathématique. Vouloir traiter les deux à la fois ne fait qu'accroître les difficultés.
Des séances spécifiques à l'utilisation du tableur, à l'algorithmique et à l'apprentissage du langage Python doivent être prévues avant de vouloir en faire une alternative à la résolution d'un problème mathématique.

Nous proposons dans ce document plusieurs situations où les TICE aident à la résolution du problème mathématique mais ces situations ne peuvent pas être utilisées pour introduire les notions algorithmiques qu'elles utilisent. Les élèves doivent avoir déjà abordé ces notions et les maîtriser correctement grâce à un travail fait en amont.

Sources :

- étude d'une transposition didactique de l'algorithmique au lycée : une pensée algorithmique comme un versant de la pensée mathématique (Nathalie BRIANT – Alain BRONNER, EMF 2015)
- Balacheff N. (1994) Didactique et intelligence artificielle. *Recherches en didactique des mathématiques, 14*(1), 9-42.
- Chevallard Y. (1985) *La transposition didactique*. Grenoble : La pensée sauvage.
- Modeste S. (2012) *Enseigner l'algorithme pour quoi ? Quelles nouvelles questions pour les mathématiques ? Quels apports pour l'apprentissage de la preuve ?* (Thèse de doctorat, Université de Grenoble).
- LAGRANGE, J.B., ROGALSKI, J. (2017) Savoirs, concepts et situations dans les premiers apprentissages en programmation et en algorithmique, in ANNALES de DIDACTIQUE et de SCIENCES COGNITIVES

L'anniversaire de Chloé

Notions mathématiques : multiple et diviseur, raisonnement logique

Notions d'algorithmiques : instructions conditionnelles, fonction

Niveau : à partir de la seconde

Résumé : écrire un algorithme qui permet de déterminer si une année est bissextile ou pas.

Chloé est née le 29 Février: Pas de chance !!!

Quand va-t-elle fêter son anniversaire ???

Consigne :

Chloé est née le 29 Février : Pas de chance !!!
Quand va-t-elle fêter son anniversaire ???

A la sortie de l'antiquité, il existait un calendrier qui ressemblait à celui que nous connaissons ***avec 365 jours et une année bissextile tous les quatre ans****. Mais la rotation de la Terre n'est pas tout à fait de 365,25 jours . En fait elle est de 365,24220 jours. Ce qui fait qu'il faut ajouter 2422 jours tous les 10 000 ans et non 2500.*

- *Il y a donc 78 jours de trop sur 10 000 ans.*
- *Le Pape Grégoire XIII ajouta alors un nouveau calcul des années bissextiles, en* ***enlevant les siècles pleins****. On avait donc sur 10 000 ans, 100 jours de moins, soit cette fois 22 jours de retard.*
- *Pour combler ce retard, on décida en plus* ***d'ajouter les années divisibles par 400****, ce qui donne 22 jours de retard plus 25, soit maintenant 3 jours d'avance sur 10000 ans....*

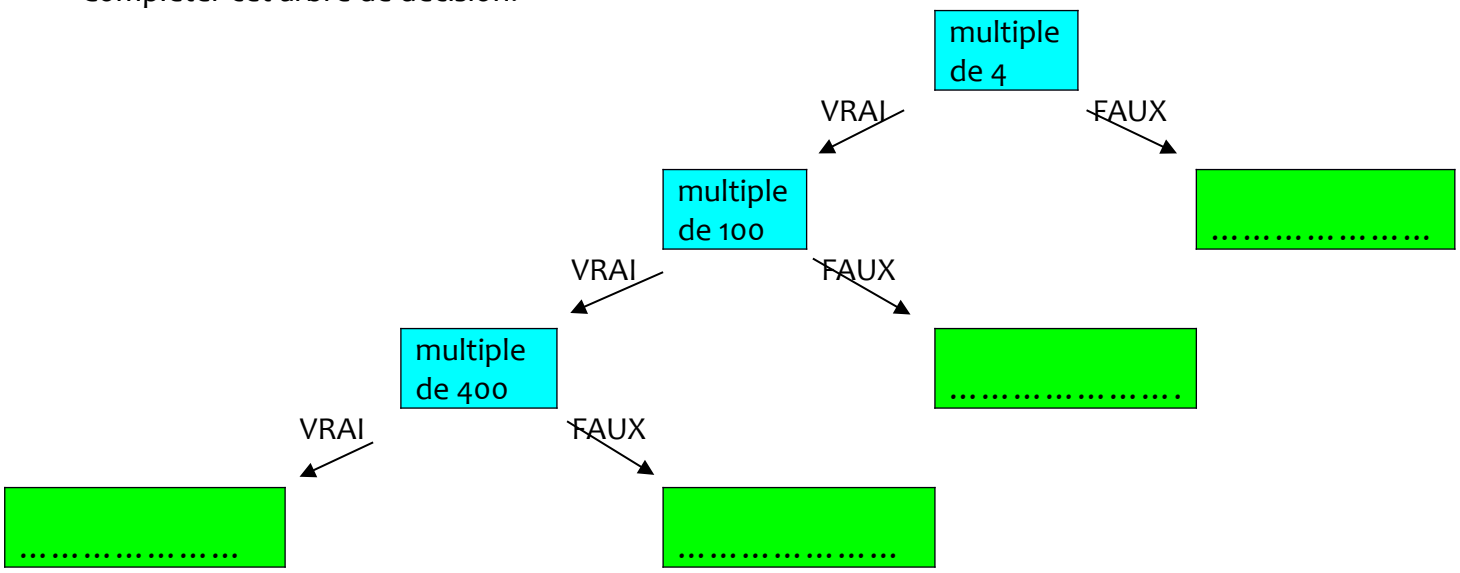
Il faut savoir qu'une année est donc bissextile si c'est un multiple de 4 qui n'est pas un multiple de 100, à l'exception des années multiples de 400 qui restent bissextiles

Exemple: "bissextile ou pas bissextile ?" Compléter:

2016 ……
1900 ……
2000 ……
2018 ……
2100 ……
2024 ……

- **Un schéma pour comprendre :**

On pense à une année, on lui fait subir les différents tests et selon la réponse aux tests on peut savoir si elle est **BISSEXTILE** ou **NON BISSEXTILE:**
Compléter cet arbre de décision:



- **Écriture d'un programme en langage Python :**

Programme 1 : "multiple ou pas multiple ?"
Le symbole % donne le reste de la division euclidienne :
ex 232 % 4 = ……
41 % 4 = ……

```
from lycee import *

def div(n):
    if n%4!=0:
        return ("n  ....................................")
    else:
        return (" n ....................................")
```

Compléter ce programme. Que permet-t-il de savoir sur le nombre *n* ?
……
Tester ce programme avec 2016, 2018, 2100
2016 : ……
2018 : ……
2100 : ……

Que peut-on conclure pour l'année 2018 ?.....
Peut-on immédiatement conclure pour les deux autres? ……

Programme 2: "bissextile ou pas bissextile ?"
On va utiliser l'arbre précédent et modifier le programme précédent pour savoir si l'année *n* est une année bissextile

```
from lycee import *
def bissextile(n):
    if n%4!=0:
        return (                                    )
    else:
```

<u>Complément : Pour aller plus loin……</u>

Programme 3
Améliorer le programme en utilisant le moins de tests logiques possibles pour déterminer si l'année *n* est bissextile…

Réponse pour le programme 2

```
from lycee import *

def bissextile(n):
    if n%4!=0:
        return ("l'année",n,"n'est pas bissextile")
    else:
        if n%100!=0:
            return ("l'année",n,"est bissextile")
        else:
            if n%400==0:
                return ("l'année",n,"est bissextile")
            else:
                return ("l'année",n,"n'est pas bissextile")

```

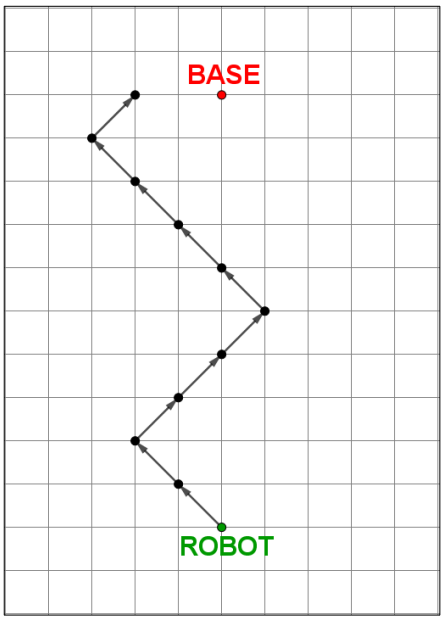
La marche du petit robot

Notions mathématiques : Réalisation d'une simulation, calcul de fréquences, probabilité, raisonnement logique

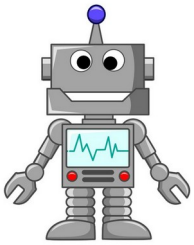
Notions d'algorithmiques : boucles bornées, instructions conditionnelles, fonction

Niveau : à partir de la seconde

Résumé : Un petit robot doit rentrer à sa base. Il fonctionne de façon aléatoire en diagonale vers la droite ou vers la gauche avec la même probabilité. On suppose que le robot fait exactement 10 déplacements et que la distance parcourue à chaque déplacement est identique.



Consigne :



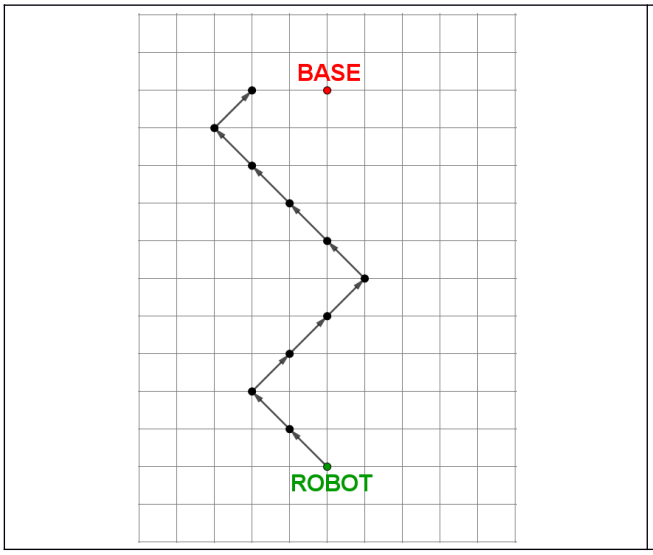
Un petit robot doit rentrer à sa base.

Il chemine de façon aléatoire en suivant les diagonales des carreaux du quadrillage sans jamais revenir en arrière.

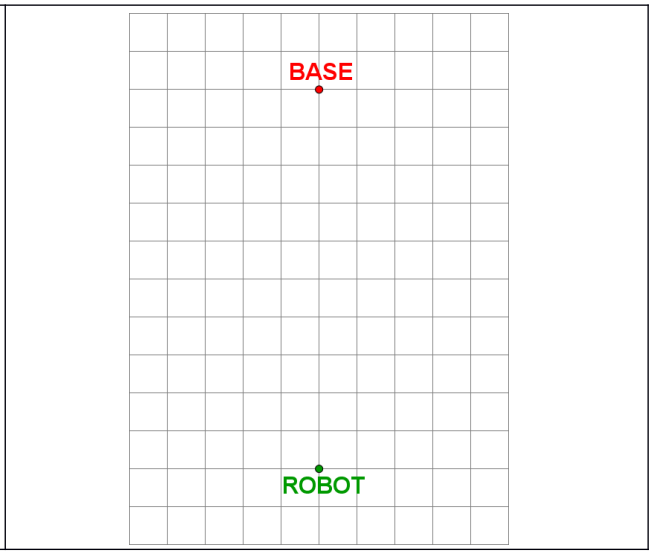
On pourra admettre que tous les chemins le ramenant à sa base contiennent toujours 10 déplacements élémentaires, et que le robot s'arrête automatiquement après 10 déplacements.

Voici un exemple (EX 1) où le robot n'atteint pas sa base.
Réaliser un exemple où il réussit à rentrer (EX2).

EX 1



EX 2



On a programmé une telle marche avec le langage « python »

```
2 from random import randint
3 def marche():
4     D=0
5     G=0
6     for i in range (10):
7         P=randint(0,1)
8         if P==0:
9             D=D+1
10        else :
11            G=G+1
12    return (D,G)
```

`randint(0,1)` génère un nombre entier aléatoire 0 ou 1

Première partie

- Expliquez comment est simulé le déplacement du robot. En particulier expliquez le rôle de la variable P.

……

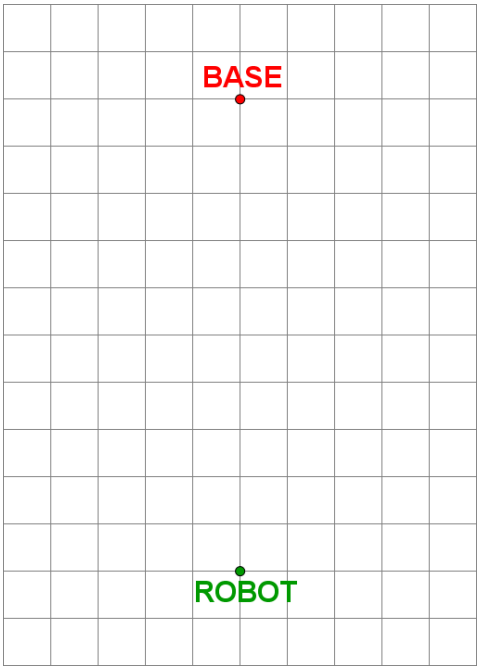
……

- Que représentent les variables G et D ? Quelles valeurs peuvent-elles prendre ?

……

……

- Exécutez le script : Qu'obtenez-vous ? ……
- Modifiez le script pour avoir un chemin détaillé du petit robot du type 0001010110 ou DDDGDGDDG.
- Testez le script modifié, donnez le résultat et tracez le chemin du robot

.. .. D= ……. G =……	

- On suppose qu'une fois le programme exécuté, la variable D a la valeur 5. Que peut-on en déduire sur la position finale du robot ?

……

- En déduire une instruction à ajouter pour contrôler si le robot est rentré à sa base

……

- Complétez le programme ci-dessous pour qu'il renvoie 1 si le robot a rejoint sa base et 0 sinon.

```
def marche():
    D=0
    G=0
    for i in range (10):
        P=randint(0,1)
        if P==0:
            D=D+1
        else :
            G=G+1
    if     :
        return
    else:
        return
```

- Exécutez le programme plusieurs fois et notez le nombre de fois où il a réussi à rentrer à la base. En déduire la fréquence obtenue de réussite .

f = ……

Deuxième partie

- Écrivez une fonction **`marchedurobot`**`(n)` qui simule un nombre de fois *n* la marche précédente et retourne la fréquence obtenue de réussite.

- Estimez la probabilité de réussite du robot.

Le jeu du lièvre et de la tortue

Notions mathématiques : Réalisation d'une simulation, calcul de fréquences, probabilité, raisonnement logique

Notions d'algorithmiques : boucles non bornées, instructions conditionnelles, fonction

Niveau : à partir de la seconde

Résumé : A l'aide d'un algorithme, on va modéliser le jeu suivant : « A chaque tour, on lance un dé. Si le 6 sort , alors le lièvre gagne la partie, sinon la tortue avance d'une case. La tortue a 5 cases à franchir avant d'atteindre l'arrivée. Dans ce cas, c'est la tortue qui gagne. »
L'objectif est de déterminer si le jeu est à l'avantage du lièvre ou de la tortue.

Consigne :

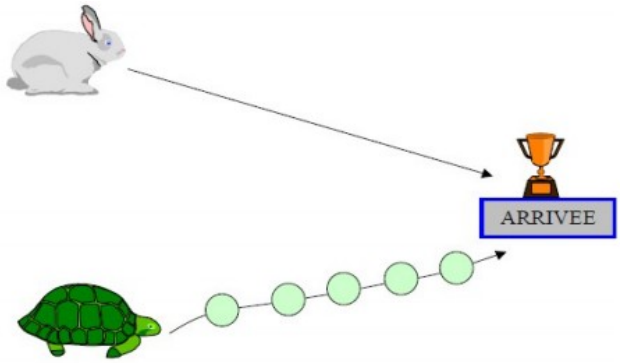
Il y a deux façons de produire des nombres aléatoires avec Python : avec randint ou avec random.

```
>>> from random import random
>>> random()
0.5844946455624939
```

On importe le module « random » de la libraire « random ».
random() donne un nombre aléatoire entre 0 et 1.

```
>>> from random import randint
>>> randint(1,6)
6
```

On importe le module « randint » de la libraire « random »
randint(1,6) donne un nombre aléatoire entier entre 1 et 6.



Règle du jeu : A chaque tour, on lance un dé. Si le 6 sort , alors le lièvre gagne la partie, sinon la tortue avance d'une case. La tortue a 5 cases à franchir avant d'atteindre l'arrivée. Dans ce cas, c'est la tortue qui gagne.

L'objectif est de déterminer si le jeu est à l'avantage du lièvre ou de la tortue.

1. Compléter l'algorithme ci-dessous :

```
position_tortue ← 0
position_lièvre ← 0
Tant que position_tortue < ….... et ….... faire
        dé = entier aléatoire entre 1 et 6
        Si dé = ….... alors
                position_lièvre ← ….....
        sinon
                position_tortue ← ….....…….....
        Fin Si
Fin Tant que
Si position_lièvre = 6 alors
        Afficher ….....…
sinon
        Afficher ….....…
Fin Si
```

2. On a utilisé cet algorithme pour créer en langage Python une fonction qui renvoie la valeur 1 si le lièvre gagne et 0 sinon. Compléter le script suivant.

```
def partie():
    from random import randint
    position_tortue=0
    position_lièvre=0
    while position_tortue<6 and ..........:
        dé=randint(1,6)
        if dé==6:
            position_lièvre=.....
        else:
            ..........
    if position_lièvre==6:
        return.....
    else:
        return.....
```

3. Compléter la fonction suivante qui permet de renvoyer la fréquence de parties gagnées par le lièvre parmi p jouées.

Aide : utiliser la fonction **`partie`**

```
def simul(p):
    n=0
    for i in range(...):
        n=.....
    return .....
```

Complément :

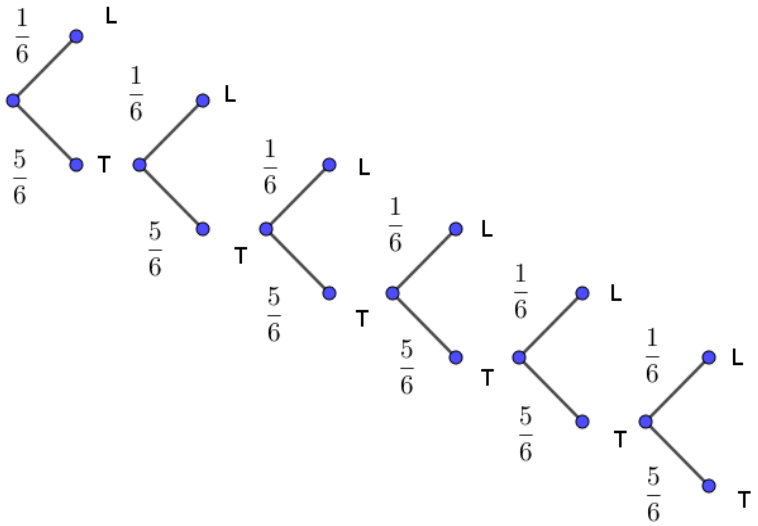
réponse :

```
def partie():
    from random import randint
    position_tortue=0
    position_lièvre=0
    while position_tortue<6 and position_lièvre<6:
        dé=randint(1,6)
        if dé==6:
            position_lièvre=6
        else:
            position_tortue=position_tortue+1
    if position_lièvre==6:
        return(1)
    else:
        return(0)

def simul(p):
    n=0
    for i in range(p):
        n=n+partie()
    return n/p

```

On peut modéliser cette situation à l'aide d'un arbre pondéré.



$$P(L)=\frac{1}{6}+\frac{1}{6}\times\frac{5}{6}+\frac{1}{6}\times\left(\frac{5}{6}\right)^2+\frac{1}{6}\times\left(\frac{5}{6}\right)^3+\frac{1}{6}\times\left(\frac{5}{6}\right)^4+\frac{1}{6}\times\left(\frac{5}{6}\right)^5=1-\left(\frac{5}{6}\right)^6\approx 0{,}665$$

Après cette modélisation, on peut demander aux élèves comment modifier la règle du jeu sur le nombre de cases pour que la probabilité que la tortue gagne soit plus grande que celle du lièvre.

le ruban lumineux

Notions mathématiques : distance dans un repère orthonormé

Notions d'algorithmiques : boucles bornées, fonction

Niveau : seconde

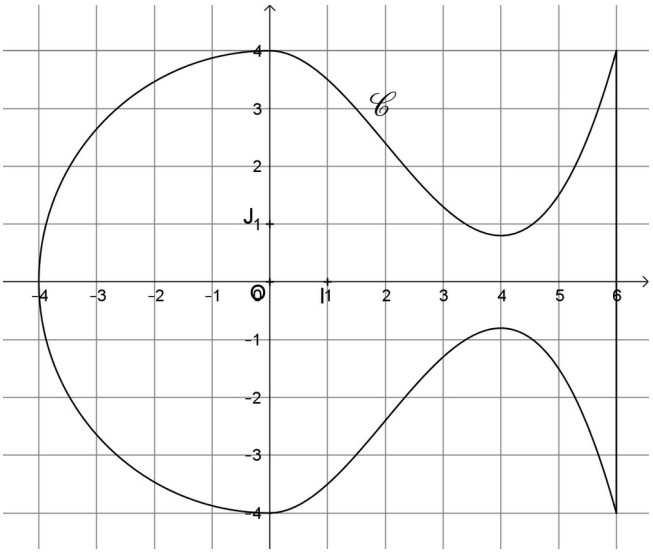
Résumé :

La société Dinfus a commandé le dessin d'un logo, représenté ci-dessous dans un repère orthonormé d'unité 1 m.

Ce logo, qui est symétrique par rapport à l'axe des abscisses (OI), comporte notamment un demi-cercle de centre O et de rayon 4 ainsi que la courbe représentative C d'une fonction *f* sur [0 ; 6].

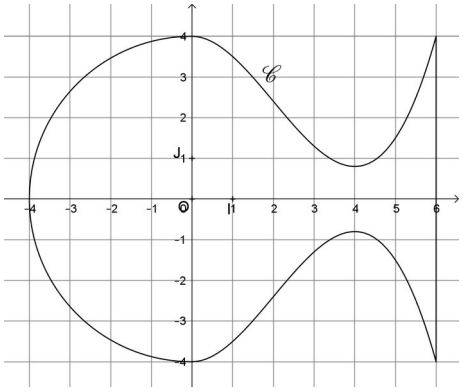
La société Dinfus souhaite entourer ce logo d'un ruban lumineux.

L'objectif de ce TP est d'estimer la longueur de ce ruban.



Consigne :

La société Dinfus a commandé le dessin d'un logo, représenté ci-contre dans un repère orthonormé d'unité 1 m.



Ce logo, qui est symétrique par rapport à l'axe des abscisses (OI), comporte notamment un demi-cercle de centre O et de rayon 4 ainsi que la courbe représentative C d'une fonction f sur [0 ; 6].

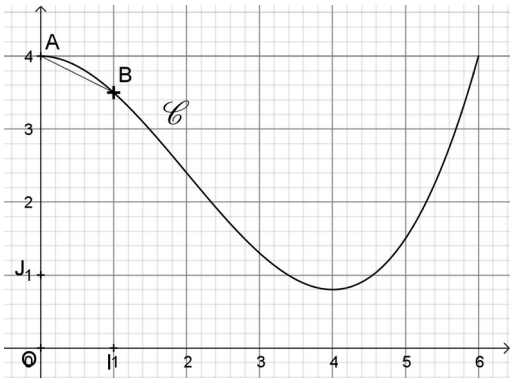
La société Dinfus souhaite entourer ce logo d'un ruban lumineux.

L'objectif de ce TP est d'estimer la longueur de ce ruban.

Distance entre deux points :

La courbe C représente la fonction f définie sur l'intervalle [0 ; 6] par $f(x) = 0{,}1x^3 - 0{,}6x^2 + 4$.

On a placé sur la courbe C les points A et B d'abscisses respectives $x_A = 0$ et $x_B = 1$.



Calculer à la main :

- les ordonnées des points A et B :
  y_A = ……
  y_B = ……
- les coordonnées du vecteur $\overrightarrow{AB}$:
- la longueur AB :

Appeler le professeur pour vérification □

On donne le script Python suivant comportant deux fonctions **`f`** et **`distance`**. Expliquer l'objectif de chacune.

```
from lycee import *

def f(x):
    return 0.1*x**3-0.6*x**2+4

def distance(xA,yA,xB,yB):
    return sqrt((xB-xA)**2+(yB-yA)**2)
```

……

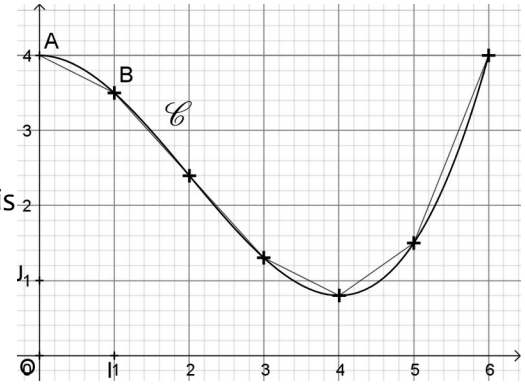
……

……

Écrire et tester la fonction **`distance`** avec les points A et B ci-dessus.

Appeler le professeur pour vérification □

Une première approximation de la longueur de la courbe $\mathcal{C}$



À l'aide d'un script Python, on va calculer la somme des longueurs des segments [AB], où A et B sont deux points de C ayant un écart en abscisse égal à 1.

Les points A et B ont initialement pour abscisses 0 et 1 puis se déplacent au fur et à mesure de l'exécution de l'algorithme ; leurs positions successives sont représentées sur la figure ci-contre.

On souhaite évaluer la longueur totale de la courbe C.

Compléter le script ci-contre en marquant sur la figure ci-dessus les différentes positions des points A et B.

Donner une estimation de la longueur de la courbe C, au mètre près.

Appeler le professeur pour vérification □

```
xA=0
yA=...
longueur=0
for n in range (0,...):
    xB=xA+1
    yB=...
    d=distance(xA,yA,xB,yB)
    longueur=longueur+...
    xA=xB
    yA=yB
print(...        )
```

Proposer une méthode pour obtenir une approximation plus précise puis la mettre en œuvre.

Appeler le professeur pour vérification □

Retour vers le ruban lumineux

Un ruban de 40 m suffira-t-il pour faire le tour de ce logo ?

L'escargot de Pythagore

Notions mathématiques : Modéliser à l'aide d'une suite, amorce de la notion de limite et du raisonnement par récurrence.

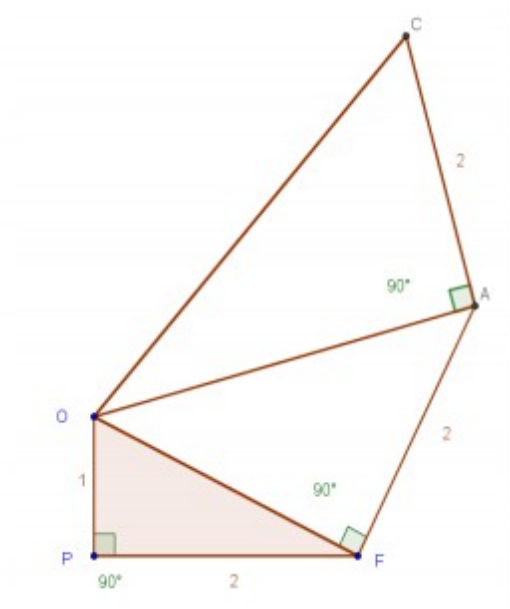
Notions d'algorithmiques : boucles non bornées, fonction

Niveau : à partir de la première

Résumé : Soit le triangle OPF rectangle en P tel que OP = 1 cm et PF = 2 cm.

On considère le procédé de construction amorcé ci-dessous.

L'hypoténuse peut-elle dépasser 1 m ? 1 km ?



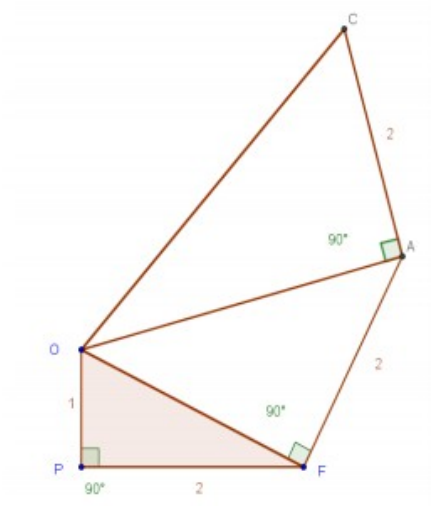
CONSIGNE

Soit le triangle OPF rectangle en P tel que OP = 1 cm et

PF = 2 cm.

On considère le procédé de construction amorcé ci-contre.

L'hypoténuse peut-elle dépasser 1 m ? 1 km ?



MODALITÉS

Cette activité a été testée en fin de première pour amorcer le travail sur la limite et en début de terminale pour le finaliser et mettre en place la définition de la limite infinie en l'infini.

Les élèves ont travaillé en groupe de 2 ou 3 et avaient à leur disposition des ordinateurs et des calculatrices.

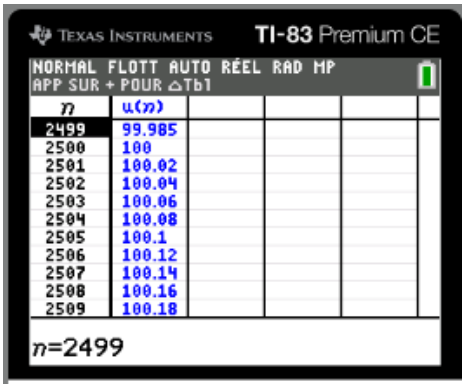
DESCRIPTIF

Rapidement, les élèves utilisent une suite pour modéliser la longueur des hypoténuses avec la suite *h* définie par $h_1=\sqrt{5}$ et $h_{n+1}=\sqrt{h_n^2+4}$ pour tout entier naturel $n\geqslant 1$.

Il s'agit de déterminer la plus petite valeur de *n* tel que $h_n\geqslant 100$.

1. avec le mode suite de la calculatrice

Des groupes utilisent le mode suite de leur calculatrice et avec un peu de patience obtiennent le tableau de valeurs :



Ils concluent que le 2500 ème triangle aura une hypoténuse de longueur supérieure à 1 m.

Par contre ils n'obtiennent pas de résultat pour 1 km, la valeur dépassant la capacité de la calculatrice.

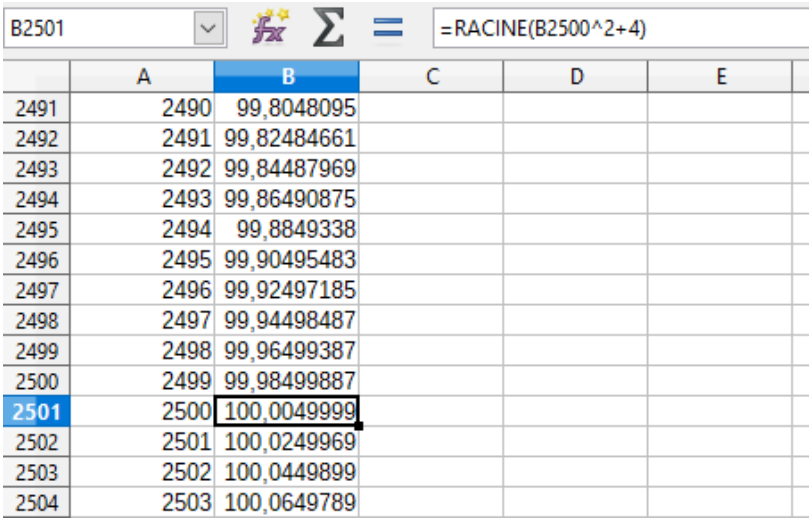
2. avec un algorithme :

```
from math import sqrt
def h(p):
    n=0
    u=1
    while u<p:
        u=sqrt(u**2+4)
        n=n+1
    return n
```

Pour h=100 cm, on trouve n=2500.

ensuite on dépasse vite la capacité du logiciel, le programme met beaucoup de temps pour trouver la valeur de n pour laquelle $h_n \geqslant 1 km$

3. avec un tableur



	A	B	C	D	E
2491	2490	99,8048095			
2492	2491	99,82484661			
2493	2492	99,84487969			
2494	2493	99,86490875			
2495	2494	99,8849338			
2496	2495	99,90495483			
2497	2496	99,92497185			
2498	2497	99,94498487			
2499	2498	99,96499387			
2500	2499	99,98499887			
2501	2500	100,0049999			
2502	2501	100,0249969			
2503	2502	100,0449899			
2504	2503	100,0649789			

On trouve la réponse pour dépasser 1 m mais le nombre maximum de lignes étant de 2^{20}, on ne trouve pas le nombre de triangles nécessaires pour dépasser 1 km.

Selon le niveau des classes, il arrive que n'apparaissent que ces trois propositions.

Le professeur s'en contente et fait le bilan des différentes solutions.

Le professeur peut alors demander aux élèves s'ils pensent qu'on pourra dépasser 1 km. Certains disent que la longueur des hypoténuses augmente donc finira par dépasser 1 km, d'autres sont moins convaincus et disent que la longueur pourrait finir par « stagner » et atteindre une valeur limite.

Cette dernière conjecture arrive plus facilement en terminale qu'en première surtout lorsque cet exercice est le premier où les élèves rencontrent la notion de limite.

La plupart sont convaincus que l'hypoténuse finira par dépasser 1 km avec en tête l'idée fausse qu'une suite croissante tend vers +∞.

On va donc essayer de trouver la forme explicite de la suite h pour répondre au problème.

Il arrive fréquemment que des groupes ont cherché à déterminer la forme explicite :

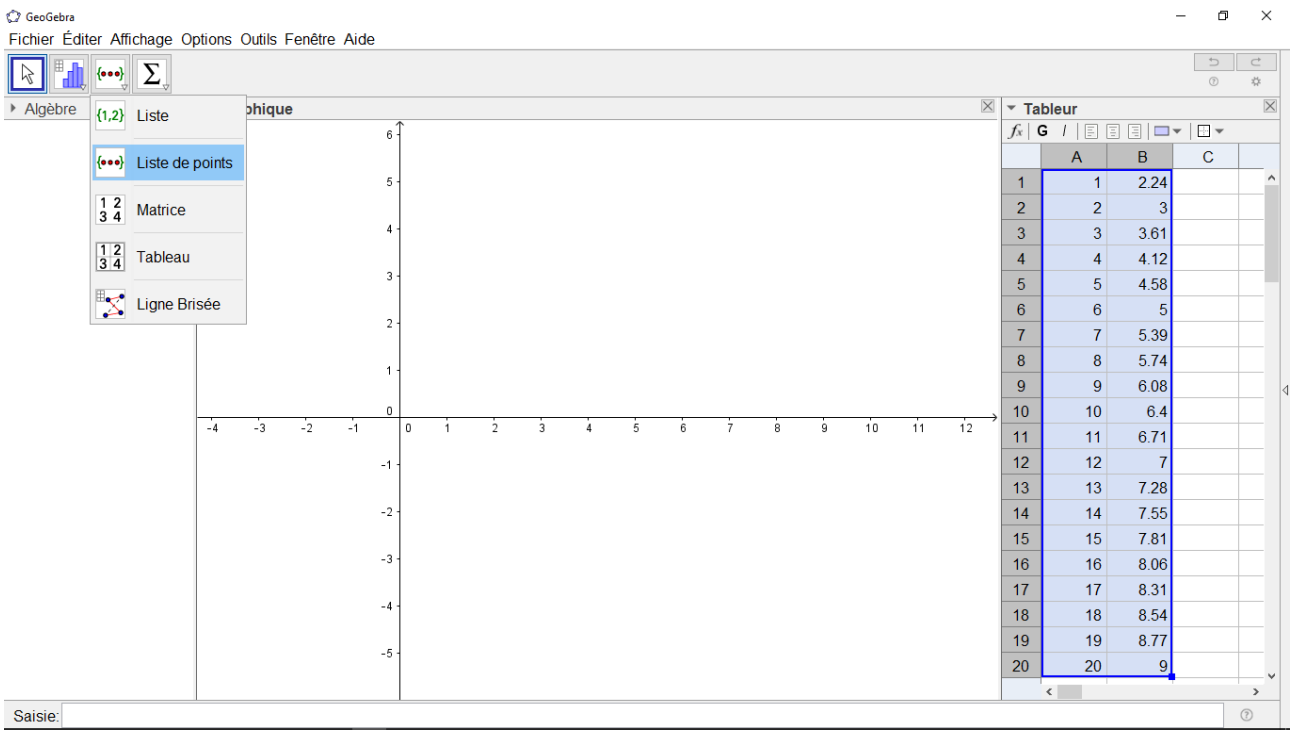
4. avec la forme explicite

- en calculant les premiers termes :

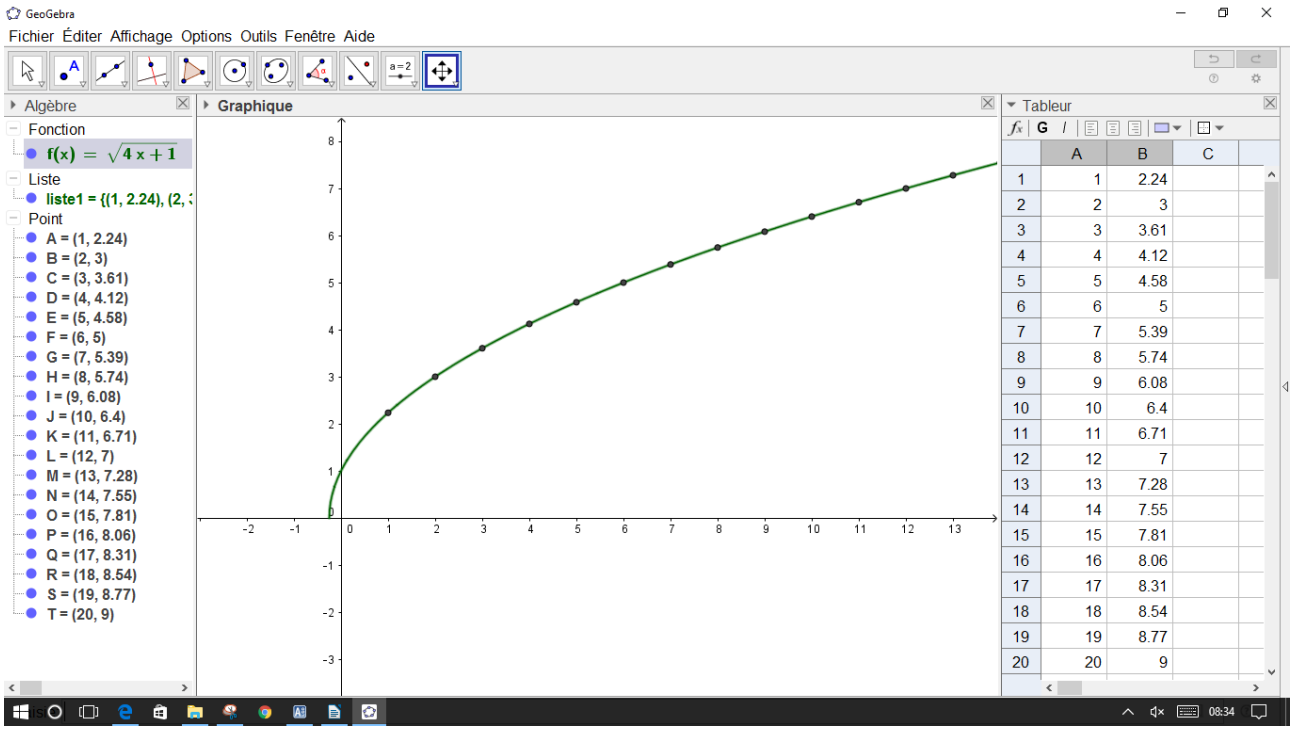
$h_1=\sqrt{5}$, $h_2=\sqrt{9}=3$, $h_3=\sqrt{13}$

Les élèves conjecturent la forme explicite : $h_n=\sqrt{4n+1}$ pour tout entier naturel $n\geqslant 1$.

- en utilisant GeoGebra

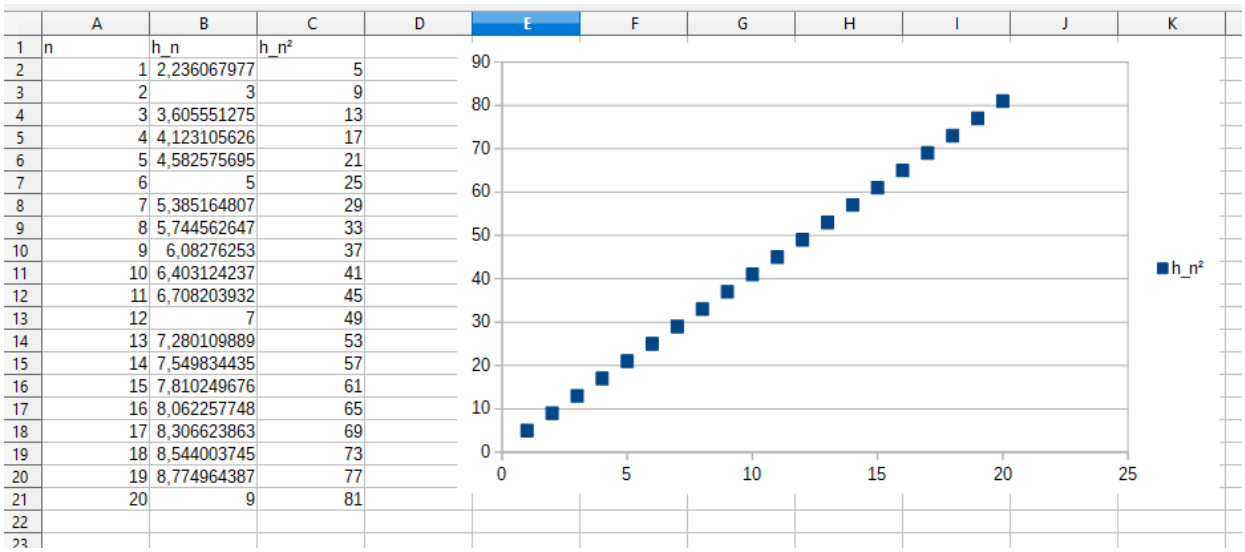


Avec le tableur de GeoGebra, on crée la suite puis la liste de points ,



Si les élèves ont conjecturé la forme explicite, ils tracent la représentation graphique de la fonction f définie sur $\left]\frac{-1}{4};+\infty\right[$ par $f(x)=\sqrt{4x+1}$ pour conforter leur conjecture.

- Dans le cas où aucun élève ne conjecture la forme explicite, le professeur propose d'utiliser le tableur de faire afficher dans la colonne B les termes de la suite h et dans la colonne C la valeur de h². Il demande ensuite aux élèves de placer le nuage de points de coordonnées (n, h_n^2).



	A	B	C
1	n	h_n	h_n²
2	1	2,236067977	5
3	2	3	9
4	3	3,605551275	13
5	4	4,123105626	17
6	5	4,582575695	21
7	6	5	25
8	7	5,385164807	29
9	8	5,744562647	33
10	9	6,08276253	37
11	10	6,403124237	41
12	11	6,708203932	45
13	12	7	49
14	13	7,280109889	53
15	14	7,549834435	57
16	15	7,810249676	61
17	16	8,062257748	65
18	17	8,306623863	69
19	18	8,544003745	73
20	19	8,774964387	77
21	20	9	81
22			
23			

les élèves conjecturent alors facilement que $h_n^2=4n+1$ et donc que $h_n=\sqrt{4n+1}$.

En première, on va admettre que pour tout entier $n \geqslant 1$, $h_n = \sqrt{4n+1}$.

En terminale, on utilise le raisonnement par récurrence pour le démontrer.

Il ne reste qu'à résoudre les inéquations :

$\sqrt{4n+1} \geqslant 100 \Leftrightarrow 4n+1 \geqslant 10000$ puisque $4n+1>0$

$\Leftrightarrow 4n \geqslant 9999 \Leftrightarrow n \geqslant \dfrac{9999}{4}$ or $\dfrac{9999}{4} = 2499{,}75$ et donc le plus petit entier n tel que $n \geqslant 2499{,}75$ est 2500.

On peut donc maintenant résoudre le problème du 1 km …

$\sqrt{4n+1} \geqslant 10^5 \Leftrightarrow 4n+1 \geqslant 10^{10} \Leftrightarrow 4n \geqslant 10^{10}-1 \Leftrightarrow n \geqslant \dfrac{10^{10}-1}{4}$

or $\dfrac{10^{10}-1}{4} = \dfrac{10^{10}}{4} - \dfrac{1}{4}$

$10^{10} = 100 \times 10^8$ donc $\dfrac{10^{10}-1}{4} = 25 \times 10^8 - \dfrac{1}{4}$, le plus petit entier n tel que $n \geqslant 25 \times 10^8 - \dfrac{1}{4}$ est donc 25×10^8 .

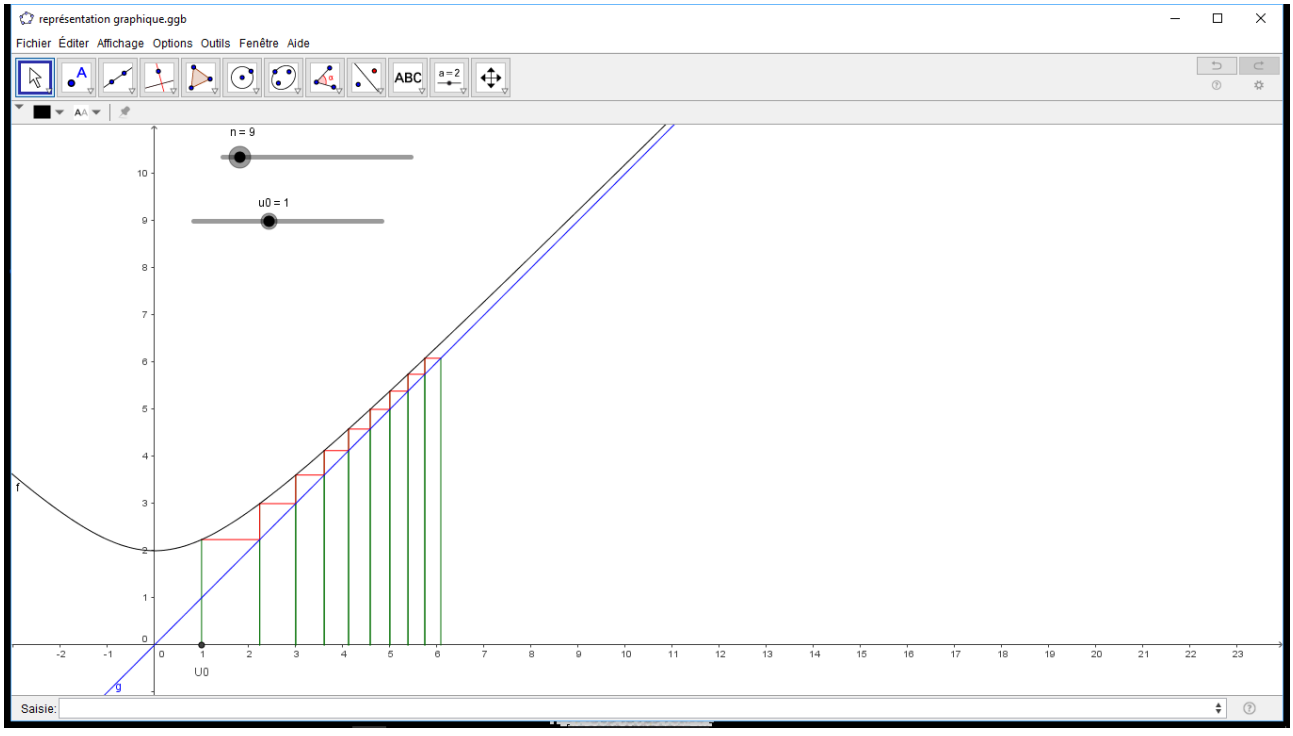
Et on peut montrer qu'on peut rendre h_n aussi grand qu'on veut .

Soit A un réel positif, $\sqrt{4n+1} > A \Leftrightarrow 4n+1 > A^2 \Leftrightarrow n > \dfrac{A^2-1}{4}$

on pose n_0 le plus petit entier strictement supérieur à $\dfrac{A^2-1}{4}$, pour tout $n \geqslant n_0$, $u_n \geqslant u_{n_0} > A$.

Par définition , la suite (h_n) tend vers $+\infty$.

remarque : les élèves pourraient utiliser la représentation graphique en escalier pour conjecturer que la suite h tend vers l'infini mais cette représentation étant peu vue en première, ils ne pensent pas à l'utiliser.



On peut écrire en bilan :

On peut rendre h_n aussi grand que l'on veut dès que *n* est suffisamment grand, on dit que la suite (h_n) tend vers $+\infty$ quand n tend vers $+\infty$.

Cette activité permet de mettre en avant les « limites » des outils numériques, ils ne permettent pas ici de répondre à la totalité du problème. Cependant, c'est une aide à la conjecture.

COMPLÉMENT

Après ce travail, on peut proposer un exercice où une suite croissante converge vers un réel l :

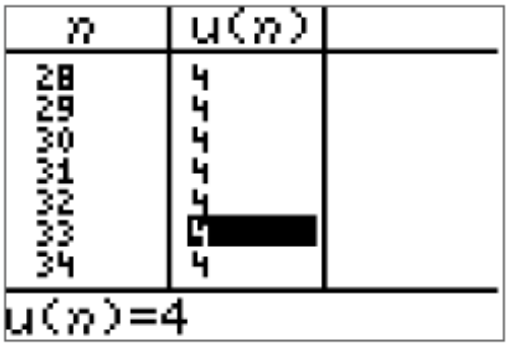
Un escargot décide d'escalader un mur de 4 m de haut . Chaque jour il arrive à grimper de 2 mètres mais la nuit lorsqu'il se repose il redescend de la moitié de la hauteur qu'il a atteinte. Au bout de combien de jours l'escargot arrivera-t-il à monter le mur ? (d'après Indice TS)

On modélise la situation par une suite :

Soit h_n la hauteur atteinte à la fin de la nième journée, on a pour tout entier $n \geqslant 1$,

$h_{n+1} = h_n + 2 - \frac{1}{2} h_n = \frac{1}{2} h_n + 2$ et $h_1 = 2$.

En utilisant, le mode suite de la calculatrice, on obtient :



les élèves conjecturent que l'escargot atteindra le mur le 33ème jour.

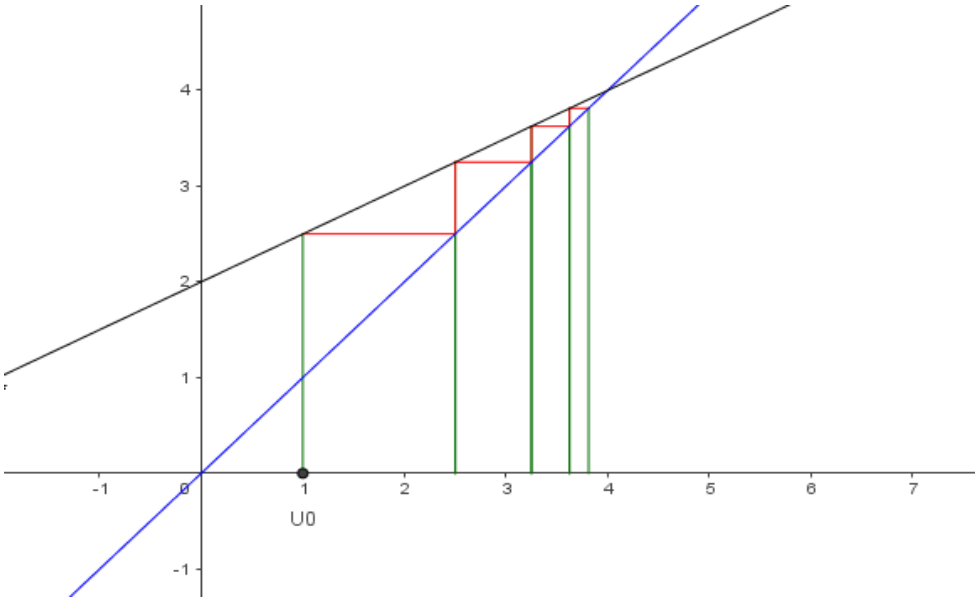
Mais en utilisant un algorithme sous Python, on obtient :

```
def escargot():
    h=2
    n=1
    while h<4:
        h=2+0.5*h
        n=n+1
    return h,n
```

et on obtient que $h_{54} = 4$.

Alors 54 ou 33 jours ? Finalement on montre par récurrence que $h_n < 4$ pour tout entier naturel $n \geq 1$.

On peut utiliser la représentation en escalier pour conjecturer les variations et la limite de la suite :



On a donc ici une suite croissante mais qui ne diverge pas vers +∞.

En terminale, on peut utiliser la suite auxiliaire : $u_n = h_n - 4$ pour tout entier naturel $n \geqslant 1$ et démontrer que $h_n = -2 \times \left(\frac{1}{2}\right)^{n-1} + 4$ pour tout entier naturel $n \geqslant 1$.

Un classique : la suite de Syracuse

Notions mathématiques : Modéliser à l'aide d'une suite,

Notions d'algorithmiques : boucles, listes, structure conditionnelle, fonction

Niveau : à partir de la première

Résumé :

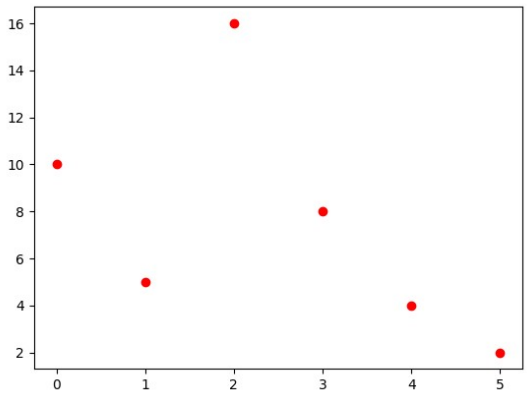
Voici un programme de calcul :

Étape 1 : Choisir un nombre entier naturel différent de 0
Étape 2 : S'il est pair, on le divise par 2
Sinon on le multiplie par 3 et on ajoute 1 au résultat
Étape 3 : On applique l'étape 2 au résultat

Lothar Collatz, un mathématicien allemand, conjectura dans les années 1930 que quel que soit le nombre choisi au départ on finira par tomber sur 1 et alors la suite de valeurs (1,4,2,1,4,2…) se répète indéfiniment. A la suite d'un exposé à l'université de Syracuse à New York, elle a acquis son surnom le plus connu : conjecture de Syracuse. Cette conjecture n'est toujours pas démontrée. En 2009, elle a été vérifiée numériquement pour tout entier inférieur à 2^{62}. En septembre 2019, Terence Tao , mathématicien australien, a montré qu'elle est « presque » vraie pour « presque » tous les entiers.

On va d'abord écrire le script d'une fonction qui renvoie la liste des termes de la suite définie par ce programme de calcul.

On peut représenter graphiquement ces suites, pour un premier terme de 10 :
Les mathématiciens assimilent ces graphiques à un vol.
Le temps de vol de *N* est le nombre d'opérations nécessaires pour atteindre 1 en partant de *N*.
L'altitude maximale est la valeur maximale de la suite obtenue en partant de *N*.
Ainsi pour *N* = 10, on a la liste : 5, 16, 8, 4, 2, 1. Le nombre 1 a été obtenu au bout de 6 étapes. Le temps de vol est de 6.
L'altitude est de 16.
On va ensuite écrire des scripts qui renvoient le temps de vol et l'altitude.



Consigne :

Un peu de cours sur les listes en informatique avant de démarrer :
Une liste est une collection ordonnée d'éléments. En Python on définit une liste avec des crochets []. Les éléments sont séparés par une virgule.

Exemple :

```
>>> L=[2,7,8]
>>> L
[2, 7, 8]
>>> L[0]
2
>>> L[2]
8
>>> len(L)
3
>>> L.append(9)
>>> L
[2, 7, 8, 9]
```

Attention les indices commencent à 0.

L[0] désigne le 1er élément de la liste

L[2] désigne le 3ème élément de la liste

len(L) renvoie le nombre d'éléments dans la liste.

« append » signifie ajouter en anglais, la fonction L.append(x) permet d'ajouter x à la fin de la liste.

Partie 1 : en débranché

Voici un script en Python où x désigne un entier naturel différent de 0.

```
1  from math import *
2
3  def f(x):
4      L=[x]
5      while x!=1:
6          if x%2==0:
7              L.append(x/2)
8              x=x/2
9          else:
10             x=1
11     return L
```

Aide : != signifie ≠ en Python

Aide : cette instruction signifie que le reste de la division euclidienne de x par 2 vaut 0.

1. Quel est le résultat de ce programme si on tape `f(4)` dans la console ?

2. Quel est le résultat de ce programme si on tape `f(5)` dans la console ?

3. Que renvoie cette fonction pour x=12 ? Compléter la liste ci-dessous sans faire tourner l'algorithme sur l'ordinateur.

Appeler le professeur

Partie 2 : la suite de Syracuse

Voici un programme de calcul :

Étape 1 : Choisir un nombre entier naturel différent de 0

Étape 2 : S'il est pair, on le divise par 2

Sinon on le multiplie par 3 et on ajoute 1 au résultat

Étape 3 : On applique l'étape 2 au résultat

1. Donner les nombres que l'on obtient si l'on choisit le nombre 4.

2. Même question avec 5.

3. Même question avec 6.

Lothar Collatz, un mathématicien allemand, conjectura dans les années 1930 que quel que soit le nombre choisi au départ on finira par tomber sur 1 et alors la suite de valeurs (1,4,2,1,4,2…) se répète indéfiniment.

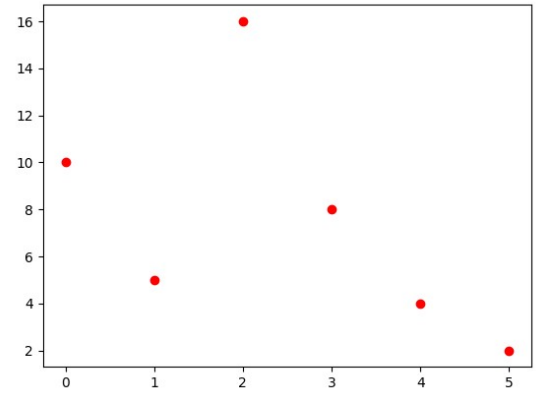
A la suite d'un exposé à l'université de Syracuse à New York, elle a acquis son surnom le plus connu : conjecture de Syracuse.

Cette conjecture n'est toujours pas démontrée. En 2009, elle a été vérifiée numériquement pour tout entier inférieur à 2^{62}. En septembre 2019, Terence Tao , mathématicien australien, a montré qu'elle est « presque » vraie pour « presque » tous les entiers.

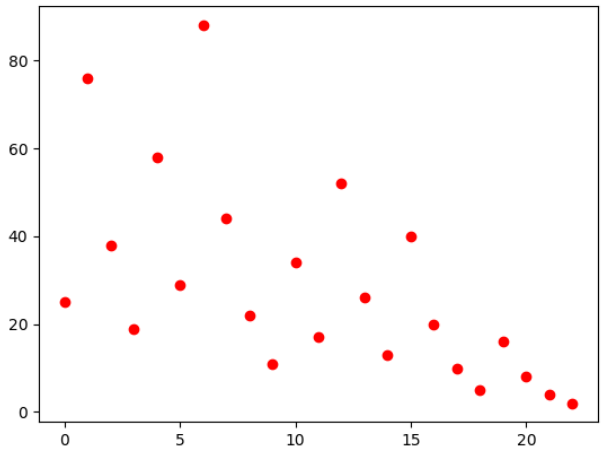
Partie 3 : programmation

1. En vous inspirant de la partie 1, écrire une fonction en langage Python qui renvoie la liste des nombres obtenus jusqu'à 1 où l'argument est le nombre de départ N.

2. On peut représenter graphiquement ces suites.



Pour N=10



Pour N=25

Les mathématiciens assimilent ces graphiques à un vol.

Le temps de vol de N est le nombre d'opérations nécessaires pour atteindre 1 en partant de N.

L'altitude maximale est la valeur maximale de la suite obtenue en partant de N

Ainsi pour N=10, on a la liste : 5, 16, 8, 4, 2, 1. Le nombre 1 a été obtenu au bout de 6 étapes. Le temps de vol est de 6. L'altitude est de 16.

Écrire deux fonctions qui renvoient l'une le temps de vol de la suite de Syracuse pour une valeur de N donnée et l'autre son altitude maximale.

3. Écrire des fonctions en Python qui permettent de répondre aux questions suivantes

a) Parmi les nombres entiers de 1 à 1000 quels sont ceux qui donnent la plus grande des altitudes maximales ?

b) Parmi les nombres entiers de 1 à 1000 quels sont ceux qui donnent le temps de vol le plus long ?

Le temps de vol record calculé actuellement est 2284 obtenu pour le nombre 2 361 235 441 021 745 907 775

sources : pourlascience.fr , https://www.pedagogie.ac-nantes.fr/

Analyse a posteriori :

Cette séance est à traiter lorsque les élèves maîtrisent les boucles et la structure conditionnelle.

La durée est de 1h30, tous les élèves réussissent à programmer la suite de Syracuse, la plupart fait le travail sur le le temps de vol et l'altitude et les plus à l'aise d'entre eux en algorithmique feront la question 3) de la partie 3 qui est facultative.

On démarre par un aperçu rapide des listes, le professeur est au vidéo-projecteur. Cette séance a été testée avec des élèves qui rencontraient pour la première fois la notion de liste.

La partie 1 permet d'amorcer le travail à partir d'un premier script dans lequel on définit une fonction où x est un entier différent de 0 qui va servir de base à la suite.

```
from math import *

def f(x):
    L=[x]
    while x!=1:
        if x%2==0:
            L.append(x/2)
            x=x/2
        else:
            x=1
    return L
```

Cette partie permet aux élèves d'analyser le script, elle se déroule en débranché. On fait le choix délibéré de ne pas mettre les élèves devant les ordinateurs afin qu'ils n'analysent pas le

programme de manière empirique. Fréquemment, par mimétisme sans doute, les élèves donnent la liste [5;1] et non pas [5] en réponse à la question 2.

Cette partie permet aussi d'expliciter la consigne de la ligne 6 et de demander aux élèves ce qu'on pourrait écrire en langage naturelle pour arriver à « si x est un nombre pair ».

La partie 2 se déroule encore une fois en débranché. On a fait le choix de ne pas définir le problème de Syracuse avec les notations des suites pour ne pas cumuler les difficultés. Rien n'empêche le professeur de demander à la fin de cette partie d'écrire ce programme de calcul en utilisant les notations des suites.

Beaucoup d'élèves s'arrêtent à l'étape 2, il faut les inciter à bien lire ce qui est écrit à l'étape 3…

Voici un script possible pour répondre à la question 1) de la partie 3 :

```
def s(x):
    L=[x]
    while x!=1:
        if x%2==0:
            L.append(x/2)
            x=x/2
        else:
            L.append(3*x+1)
            x=3*x+1
    return L
```

Le professeur peut mettre le programme de la partie 1 dans le répertoire de données de la classe, cela permet de gagner du temps et d'éviter les problèmes de syntaxe.

On rencontre plusieurs erreurs dans le script des élèves :

- *certains écrivent :* while x!=0 *au lieu de* while x!=1 *et se retrouvent avec une boucle qui ne s'arrêtent pas ...La confusion vient certainement de l'étape 1 du programme de calcul.*
- *D'autres (parfois les mêmes) oublient de rajouter le terme «* 3x+1 *» en écrivant :*

```
13 def s(x):
14     L=[x]
15     while x!=1:
16         if x%2==0:
17             L.append(x/2)
18             x=x/2
19         else:
20             x=3*x+1
21     return L
```

- Enfin on trouve des inversions d'ordre d'instructions comme :

```
13 def s(x):
14       L=[x]
15       while x!=1:
16         if x%2==0:
17             L.append(x/2)
18             x=x/2
19         else:
20             x=3*x+1
21             L.append(3*x+1)
22
23       return L
```

Pour la question 2) de la partie 3, l'objectif est d'utiliser la fonction s :

Pour le temps de vol :

```
def vol(x):
    return len(s(x))-1
```

Pour l'altitude :

```
def alt(x):
    m=0
    for i in s(x):
        if  i>m:
            m=i
    return m
```

Certains élèves utilisent aussi la fonction **max.**

Voici un script qui permet de donner la plus grande des altitudes maximales parmi les nombres entiers de 1 à i :

```
def listalt(i):
    L=[0]
    for k in range(1,i+1):
        if alt(k)>L[0]:
            L=[alt(k),k]
        elif alt(k)==L[0]:
            L.append(k)
    return L
```

Voici un script qui renvoie le premier terme, parmi les nombres entiers de 1 à i qui donnent le temps de vol le plus long.

```
def listvol(i):
    L=[0]
    for k in range(1,i+1):
        if vol(k)>L[0]:
            L=[vol(k),k]
        elif vol(k)==L[0]:
            L.append(k)
    return L
```