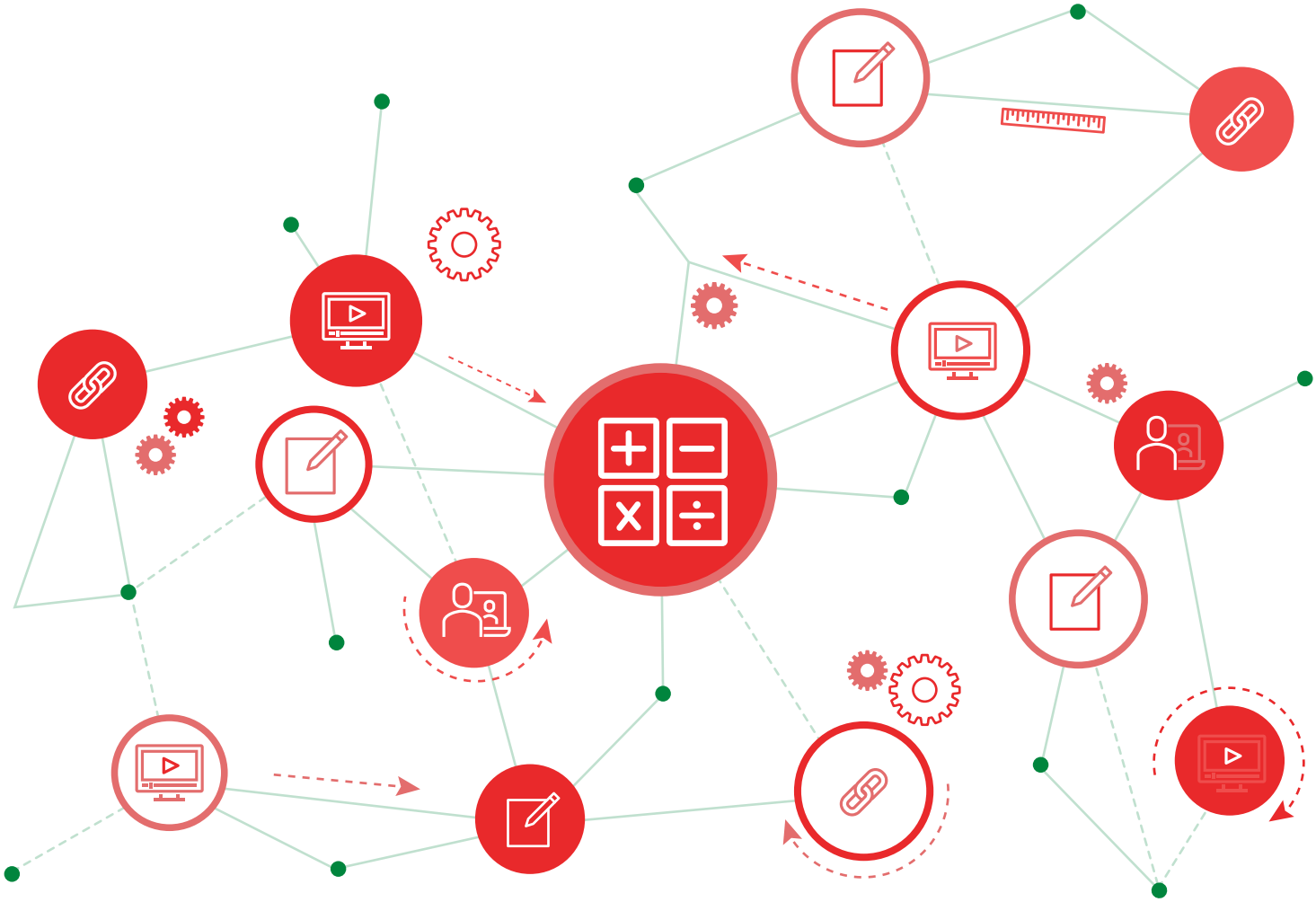
T³ FRANCE

Teachers Teaching with Technology™



Laurent Didier, Florent Girod, Boris Hanuš

Livret d'activités en mathématiques
Lycée général

www.t3france.fr

python™

Sommaire

1. Somme des chiffres d'un entier. **3**

2. Diviseurs, nombres premiers et pgcd. **5**

3. Divisibilité et nombres premiers Divisibilité et nombres premiers **10**

4. Le crible d'Eratostène **14**

5. Multiples et nombres premiers **16**

6. Conjecture de Goldbach **19**

7. Calcul de la longueur d'une courbe **22**

8. Calcul approché de longueur d'une portion de courbe **24**

9. Approximation de racine de 2 par balayage **28**

10. Recherche d'extremum d'une fonction **31**

11. Approximation d'extremums par balayage **33**

12. Rayon d'un cylindre minimisant sa surface **36**

13. Calcul d'une mensualité d'un prêt **38**

14. Bibliothèque Random, médiane... **41**

15. En attendant 1, 2, 3 sigma ... **47**

16. Moyenne, écart-type **52**

17. Fréquence d'une lettre dans un texte. **55**

18. Lancer de pièce et prise de décision. **58**

19. Une somme de hasards - Lancer de 4 dés. **60**

20. Tirage sans remise et recherche d'une proba **64**

21. Le lièvre et la tortue **66**

22. Suite de Fibonacci - Le problème des chevaliers **70**

23. Equation réduite de droite. **73**

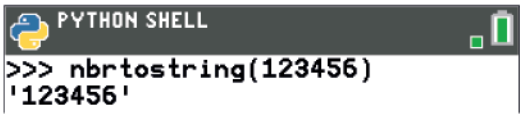
24. Des formules dans un repère - Milieu,colinéarité… **76**

Somme des chiffres d'un entier.

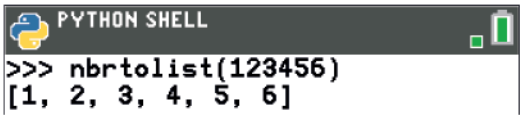
Le but de cette activité est de trouver la somme des chiffres de 2222^{333} ainsi que la somme des chiffres de la somme des chiffres du nombre 1234^{123}.

Dans un script SOMMECH

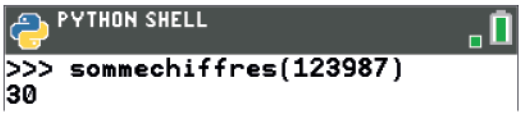
1°) Créer une fonction `nbrtostring` qui prend comme argument un entier et qui renvoie une chaîne de caractères composée des chiffres de cet entier.



2°) Ecrire une fonction `nbrtoliste` qui prend comme argument un entier et qui renvoie la liste composée des chiffres de cet entier (au format `int`).



3°) Ecrire une fonction `sommechiffres` qui prend comme argument un entier naturel et qui renvoie la somme de ses chiffres.

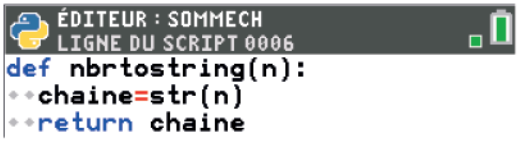


4°) Applications :

- Quelle est la somme des chiffres du nombre 2222^{333} ?
- Quelle est la somme des chiffres du nombre obtenue en faisant la somme des chiffres de 1234^{123} ?

Fonction `nbrtostring`

1°) Cette fonction se code facilement en utilisant la fonction `str` de Python qui permet de convertir un nombre (de type `int` ou `float`) en une chaîne de caractères.



On n'était pas obligé de coder cette fonction, elle renomme seulement la fonction `str` de Python, mais dans le contexte de notre exercice elle lui donne un nom « plus parlant ».

Somme des chiffres d'un entier.

Fonction `nbrtoliste`

2°) On peut coder cette fonction de deux façons :

Une première façon qui n'utilise pas les définitions condensées des listes en Python :

On convertit l'entier `n` en chaîne de caractères `h`.

Puis à partir de la liste `m` (vide initialement), on parcourt tous les caractères de la chaîne `h` dans une boucle `for` et à chaque tour de boucle on ajoute le caractère courant (en le convertissant en nombre grâce à l'instruction `int`) dans la liste `m`.

On peut utiliser aussi une seconde façon beaucoup plus courte et équivalente à la précédente :

ÉDITEUR : SOMMECH
LIGNE DU SCRIPT 0015

```
def nbrtolist(n):
  h=nbrtostring(n)
  m=[]
  for i in h:
    m.append(int(i))
  return m
```

ÉDITEUR : SOMMECH
LIGNE DU SCRIPT 0022

```
def nbrtolist(n):
  h=nbrtostring(n)
  m=[int(i) for i in h]
  return m
```

Fonction `sommechiffres`

3°) On va commencer par convertir l'entier `n` en une liste dont les éléments sont les chiffres de `n` (dans son écriture en base 10) grâce à notre fonction précédente `nbrtolist`.

Puis, pour faire la somme des éléments de la liste on utilise la fonction native de Python `sum`.

ÉDITEUR : SOMMECH
LIGNE DU SCRIPT 0026

```
def sommechiffres(n):
  return sum(nbrtolist(n))
```

Application

4°) On trouve 4958 pour la somme des chiffres de 2222^{333}, et 19 pour la somme des chiffres de la somme des chiffres de 1234^{123}.

```
>>> from SOMMECH import *
>>> sommechiffres(2222**333)
4958
>>> sommechiffres(sommechiffres(
1234**123))
19
```

Diviseurs, nombres premiers et pgcd.

On va voir dans cette activité quelques fonctions classiques en arithmétique.

Dans un script ARITHM

1°) Ecrire une fonction `divise` qui prend comme argument `a`$\in \mathbb{Z}^*$ et `b`$\in \mathbb{Z}$ et qui renvoie `True` si `a` divise `b` et `False` sinon.

```
PYTHON SHELL
>>> divise(5,7)
False
>>> divise(5,10)
True
```

2°) Ecrire une fonction `nbdiviseur` qui prend comme argument $n \in \mathbb{N}^*$ et qui renvoie le nombre de diviseurs positifs de n.

```
PYTHON SHELL
>>> nbdiviseur(10)
4
>>> nbdiviseur(11)
2
```

3°) Ecrire une fonction `listediviseur` qui prend comme argument $n \in \mathbb{N}^*$ et qui renvoie la liste des diviseurs positifs de n.

```
PYTHON SHELL
>>> listediviseur(10)
[1, 2, 5, 10]
>>> listediviseur(4569878)
[1, 2, 29, 58, 78791, 157582, 22
84939, 4569878]
```

4°) Ecrire une fonction `listedivcommun` qui prend comme argument a et b deux entiers naturels non nuls et qui renvoie la liste des diviseurs positifs communs à a et b.

```
PYTHON SHELL
>>> listedivcommun(6,8)
[1, 2]
>>> listedivcommun(30,75)
[1, 3, 5, 15]
```

5°) Ecrire une fonction `pgcd` qui prend comme arguments a et b deux entiers naturels non nuls et qui renvoie $pgcd(a,b)$ en utilisant la fonction `listedivcommun`.

```
PYTHON SHELL
>>> pgcd(6,8)
2
>>> pgcd(30,75)
15
```

6°) Ecrire une fonction `ppcm` qui prend comme arguments a et b deux entiers naturels non nuls et qui renvoie $ppcm(a,b)$.

```
PYTHON SHELL
>>> ppcm(6,8)
24
>>> ppcm(30,75)
150
```

7°) Ecrire une fonction `premier` qui prend comme argument $n \in \mathbb{N}^*$ et qui renvoie `True` si n est premier et `False` sinon.

```
PYTHON SHELL
>>> premier(1)
False
>>> premier(2)
True
>>> premier(10)
False
```

Diviseurs, nombres premiers et pgcd.

8°) Ecrire une fonction `npremier` qui prend comme argument $n \in \mathbb{N}^*$ et qui renvoie le n-ième nombre premier.

```
PYTHON SHELL
>>> npremier(1)
2
>>> npremier(5)
11
```

9°) Ecrire une fonction `listepremier` qui prend comme argument $n \in \mathbb{N}^*$ et qui renvoie la liste des nombres premiers inférieurs ou égaux à n.

```
PYTHON SHELL
>>> listepremier(25)
[2, 3, 5, 7, 11, 13, 17, 19, 23,
 29, 31, 37, 41, 43, 47, 53, 59,
 61, 67, 71, 73, 79, 83, 89, 97]
```

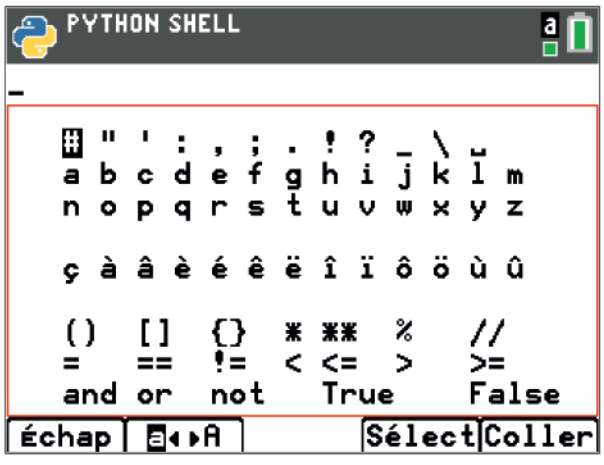
10°) Ecrire une fonction `premiersup` qui prend comme argument $n \in \mathbb{N}^*$ et qui renvoie le plus petit nombre premier supérieur strict à n.

```
PYTHON SHELL
>>> premiersup(1)
2
>>> premiersup(9)
11
>>> premiersup(24)
29
```

Fonction `divise`

1°) Pour obtenir le reste de la division euclidienne de b par a on utilise le symbole %.

Il est accessible dans [a A #], puis à l'aide des flèches de direction de la calculatrice, on va chercher le symbole % qu'il faut sélectionner en appuyant sur [Sélect] puis sur [Coller] pour l'insérer dans le texte du script.




```
ÉDITEUR : ARITHM
LIGNE DU SCRIPT 0012
def divise(a,b):
  if b%a==0:
    return True
  else:
    return False
```

Attention : Pour faire le test si le reste est nul, il ne faut pas oublier le double égal == qui correspond à la comparaison (un seul = correspond à l'affectation).

Diviseurs, nombres premiers et pgcd.

Fonction `nbiviseur`

2°) Pour compter les diviseurs de l'entier `n`$\in \mathbb{N}^*$, on commence par introduire un compteur k qu'on initialise à 1. En effet tous les entiers `n`$\in \mathbb{N}^*$ admettent 1 comme diviseur, donc on démarre notre compteur à 1.

Puis on effectue pour tous les entiers de 2 à `n` le test de divisibilité.
Afin de parcourir tous les entiers de 2 à `n` on utilise une boucle `for`, la variable choisie ici est `i`.

`range(2,n+1)` correspond aux entiers compris entre 2 (au sens large) et `n+1` (au sens strict). Autrement dit :

`range(2,n+1)`$=\{i \in \mathbb{N} \mid 2 \leq i < n+1\} = \{i \in \mathbb{N} \mid 2 \leq i \leq n\}$

Ce qui explique la présence de `n+1` dans l'instruction `range`.

A l'intérieur de la boucle, on effectue le test de divisibilité de `i` par `n` en réutilisant la fonction `divise` précédente. On sait que cette fonction renvoie le booléen `True` ou `False`.

Juste après l'écriture de l'instruction `if` on doit écrire un booléen. Etant donné que la fonction `divise` renvoie un booléen on n'a pas eu besoin d'écrire : `if divise(i,n)==True:`
Mais on pouvait le faire car `divise(i,n)==True` est aussi un booléen !

ÉDITEUR : ARITHM
LIGNE DU SCRIPT 0018

```
def nbdiviseur(n):
  k=1
  for i in range(2,n+1):
    if divise(i,n):
      k=k+1
  return k
```

Fonction `listediviseurs`

3°) La liste des diviseurs de l'entier `n`$\in \mathbb{N}^*$ est initialisée avec la valeur 1 car 1 est un diviseur de `n`.

Puis on parcourt tous les entiers de 2 à `n` avec l'instruction :

`for i in range(2,n+1):`

A chaque fois que `i` divise `n` on ajoute `i` à la liste `liste` avec l'instruction `liste.append(i)`. Lorsque la boucle est achevée, `liste` est renvoyée.

ÉDITEUR : ARITHM
LIGNE DU SCRIPT 0025

```
def listediviseur(n):
  liste=[1]
  for i in range(2,n+1):
    if divise(i,n):
      liste.append(i)
  return liste
```

Fonction `listedivcommun`

4°) `la` et `lb` représentent respectivement la liste de diviseurs de `a` et `b`.

On initialise la liste `liste` contenant les diviseurs communs de `a` et `b` comme une liste vide.

`for i in la:` signifie que `i` va parcourir tous les éléments de la liste `la`. Donc `i` va prendre comme valeur successivement tous les diviseurs de `a`.

On teste très facilement avec Python la présence de `i` dans la liste `lb` des diviseurs de `b` en écrivant tout simplement `if i in lb`.

Si ce dernier test est positif alors `i` est un diviseur commun à `a` et `b`, on l'ajoute à la liste `liste` des diviseurs communs en écrivant :

`liste.append(liste)` . A la fin de la fonction on renvoie `liste`.

ÉDITEUR : ARITHM
LIGNE DU SCRIPT 0032

```
def listedivcommun(a,b):
  la=listediviseur(a)
  lb=listediviseur(b)
  liste=[]
  for i in la:
    if i in lb:
      liste.append(i)
  return liste
```

Diviseurs, nombres premiers et pgcd.

Fonction `pgcd`

5°) Pour trouver le pgcd il suffit d'utiliser les fonctions que nous avons créées précédemment. Il correspond au plus grand diviseur commun, le script de la fonction tient donc en 4 lignes !

ÉDITEUR : ARITHM
LIGNE DU SCRIPT 0041

```
def pgcd(a,b):
  liste=listedivcommun(a,b)
  d=max(liste)
  return d
```

Fonction `ppcm`

6°) En utilisant la propriété $\forall (a,b) \in \mathbb{N}^{*2}\ pgcd(a,b) \times ppcm(a,b) = ab$

On peut définir le ppcm comme étant égal à $\frac{ab}{pgcd(a,b)}$.

On n'oubliera pas d'écrire // pour que le résultat de la division soit un entier.

ÉDITEUR : ARITHM
LIGNE DU SCRIPT 0046

```
def ppcm(a,b):
  d=pgcd(a,b)
  m=a*b//d
  return m
```

Fonction `premier`

7°) On peut aborder cette fonction de façon très simple à l'aide des fonctions précédentes. En effet $n \in \mathbb{N}^*$ est premier lorsqu'il admet exactement 2 diviseurs (1 et lui-même). On peut donc compter le nombre de diviseurs de n et tester s'il y en a 2 :

On peut aussi s'affranchir des fonctions précédentes et utiliser une propriété classique :

$n \in \mathbb{N}, n \geq 2$ est premier lorsqu'il n'est divisible par aucun entier compris entre 2 et $\sqrt{n}$.

On prend la précaution de tester si n vaut 1 et de renvoyer `False` si c'est le cas, car notre propriété est valable pour $n \geq 2$, il faut donc traiter le cas $n = 1$ à part.

`m` va représenter la partie entière de $\sqrt{n}$. La fonction partie entière est `floor`, qui se trouve dans la bibliothèque `math`, il faudra bien penser à la lire avant en écrivant : `from math import *`.

Puis on écrit une boucle `for` avec une variable `i` qui prendra toutes les valeurs entières entre 2 et `m` (ne pas oublier d'écrire `range(2,m+1)`). A chaque tour de boucle on teste si `i` divise `n` et lorsque c'est le cas, on renvoie `False`.

Finalement si la boucle se termine c'est que l'instruction `return False` ne s'est jamais exécutée (l'instruction `return` arrête l'exécution de la fonction et donc de la boucle) cela signifie qu'aucun des nombres testés n'est un diviseur de `n` donc `n` est premier, il faut donc renvoyer `True`.

ÉDITEUR : ARITHM
LIGNE DU SCRIPT 0051

```
def premier(n):
  k=nbdiviseur(n)
  if k==2:
    return True
  else:
    return False
```

ÉDITEUR : ARITHM
LIGNE DU SCRIPT 0058

```
from math import *
def premier(n):
  if n==1:
    return False
  else:
    m=floor(sqrt(n))
    for i in range(2,m+1):
      if divise(i,n):
        return False
    return True
```

Diviseurs, nombres premiers et pgcd.

Fonction `npremier`

8°) `npremier(n)` doit renvoyer le n-ième nombre premier. On va introduire deux variables `k` et `i` : `k` va dénombrer le nombre de nombres premiers trouvés et `i` est un entier qui est initialisé à 2 et qui va augmenter de 1 en 1 afin de parcourir tous les entiers jusqu'à en avoir rencontré `n` premiers. Dans ce cas l'algorithme s'arrête (puisque `k` et `n` sont égaux).

On remarque qu'on renvoie `i-1` et non pas `i`, en effet, pour `k=n-1` la boucle continue de s'exécuter et lorsque `i` est premier alors la valeur de `k` passe à `n` (donc le résultat à renvoyer est `i`), mais `i` est incrémenté de 1… C'est donc bien `i-1` qu'il faut renvoyer.

On peut aussi incrémenter `i` de 2 en 2 pour accélérer la boucle. Il faudra pour cela modifier le début du script de la fonction en initialisant `k` et `i` respectivement à 1 et 3. Il faudra aussi envisager le cas `n=1` à part.

ÉDITEUR : ARITHM
LIGNE DU SCRIPT 0072

```
def npremier(n):
  k=0
  i=2
  while k<n:
    if premier(i):
      k=k+1
    i=i+1
  return i-1
```

Fonction `listepremier`

9°) Pour obtenir la liste des `n` premiers nombres premiers, on commence par initialiser la liste `liste` qui va les contenir en écrivant `liste=[]`

Puis dans une boucle `for` la variable `i` va prendre toutes les valeurs entières de 1 à `n`. A chaque tour de boucle on ajoute à la liste `liste` le nombre premier n°`i` grâce à la fonction `npremier` précédente.

ÉDITEUR : ARITHM
LIGNE DU SCRIPT 0081

```
def listepremier(n):
  liste=[]
  for i in range(1,n+1):
    liste.append(npremier(i))
  return liste
```

Fonction `premiersup`

10°) Pour trouver le premier nombre premier supérieur strict à `n`, commençons par définir une variable `k` initialisée à `n+1` (puisqu'on cherche un nombre premier supérieur strict à `n`) et tant que `k` n'est pas premier on l'incrémente. La boucle va s'arrêter au premier nombre premier trouvé.

ÉDITEUR : ARITHM
LIGNE DU SCRIPT 0087

```
def premiersup(n):
  k=n+1
  while premier(k)==False:
    k=k+1
  return k
```

Thème: Arithmétique

Niveau: 2nde

TI-83 Premium CE

python™

Divisibilité et nombres premiers

Compétences visées

- **chercher**, expérimenter – en particulier à l'aide d'outils logiciels ;
- **modéliser**, faire une simulation, valider ou invalider un modèle ;
- **représenter**, choisir un cadre (numérique, algébrique, géométrique...), changer de registre ;
- **calculer**, appliquer des techniques et mettre en œuvre des algorithmes.

Ces compétences sont mises en œuvre dans le cadre de l'extrait du programme de 2^{nde} GT ci-dessous :

- « Déterminer si un entier naturel a non nul est multiple d'un entier naturel b non nul. »
- « Déterminer si un entier naturel est premier. »

Situation déclenchante

L'os d'Ishango

Les entailles retrouvées sur l'os d'Ishango daté à plus de 20 000 ans avant notre ère, semblent isoler quatre nombres premiers 11, 13, 17 et 19. Certains archéologues l'interprètent comme la preuve de la connaissance des nombres premiers. Toutefois, il existe trop peu de découvertes permettant de cerner les connaissances réelles de cette période ancienne.

Problématique

Ecrire un script qui permet de vérifier si un nombre est premier ou non.

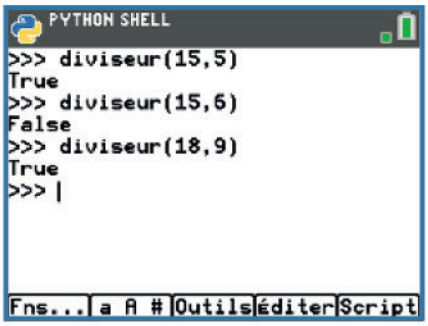
On pourra ensuite écrire un script qui fournit la décomposition en produit de facteurs premiers d'un nombre entier.

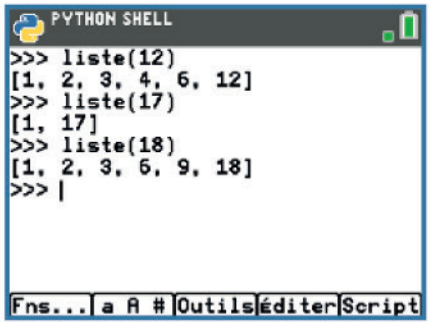
Fiche méthode

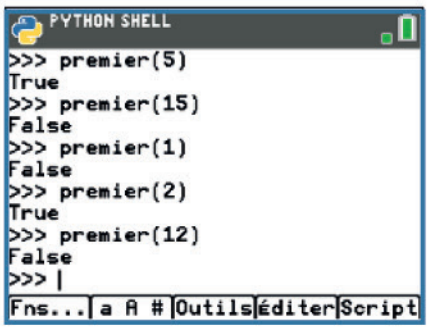
Proposition de résolution

On **crée trois fonctions dans ce script** :

- Une fonction **diviseur** qui prend comme arguments deux entiers naturels a et b et qui renvoie vrai si b divise a et faux sinon.
- Une fonction **liste** qui prend comme argument un entier naturel a et qui renvoie la liste des diviseurs positifs de a.
- Une fonction **premier** qui prend comme argument un entier naturel p et qui renvoie vrai si p est un nombre premier et faux sinon.

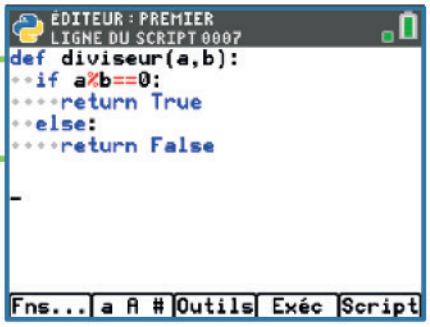






Etapes de résolution

La fonction **diviseur** prend comme arguments deux entiers naturels a et b et renvoie vrai ou faux si b divise a ou non. L'instruction `a%b` permet de déterminer le reste dans la division euclidienne de a par b.



Pour profiter de tutoriels vidéos, Flasher le QRCode ou cliquer dessus

Fiche méthode

Etapes de résolution

C'est un principe à retenir : On peut appeler une fonction (ici : la fonction **diviseur**) à l'intérieur d'une autre fonction (ici : **liste)**. L'utilisation successive de fonctions en python rend le script dans son ensemble plus lisible.

La fonction **liste** prend comme argument un entier naturel a et renvoie la liste des diviseurs de a. L'instruction `l=[]` permet d'initialiser la liste `l` avec une liste vide. L'instruction `l.append(i)` permet de rajouter le nombre `i` à la liste `l`.

```
ÉDITEUR : PREMIER
LIGNE DU SCRIPT 0016
def liste(a):
  l=[]
  for i in range(1,a+1):
    if diviseur(a,i)==True:
      l.append(i)
  return l
Fns... | a A # | Outils | Exéc | Script
```

La fonction **premier** prend comme argument un entier naturel p et renvoie vrai si p est un nombre premier et faux sinon. On fait appel à la fonction **liste** pour vérifier si p admet exactement deux diviseurs 1 et p.

```
ÉDITEUR : PREMIER
LIGNE DU SCRIPT 0023
def premier(p):
  if liste(p)==[1,p]:
    return True
  else:
    return False
Fns... | a A # | Outils | Exéc | Script
```

```
PYTHON SHELL
>>> premier(5)
True
>>> premier(15)
False
>>> premier(1)
False
>>> premier(2)
True
>>> premier(12)
False
>>>
Fns... | a A # | Outils | Éditer | Script
```

Deuxième méthode

Sans utiliser les listes, on peut créer d'une autre manière une fonction **premier2** qui prend comme argument un entier naturel p et qui renvoie vrai si p est premier et faux sinon.

```
ÉDITEUR : PREMIER
LIGNE DU SCRIPT 0046
def premier2(p):
  if p==1:
    return False
  for i in range(2,p):
    if p%i==0:
      return False
  return True
Fns... | a A # | Outils | Exéc | Script
```

Pour profiter de tutoriels vidéos, Flasher le QRCode ou cliquer dessus

Fiche méthode

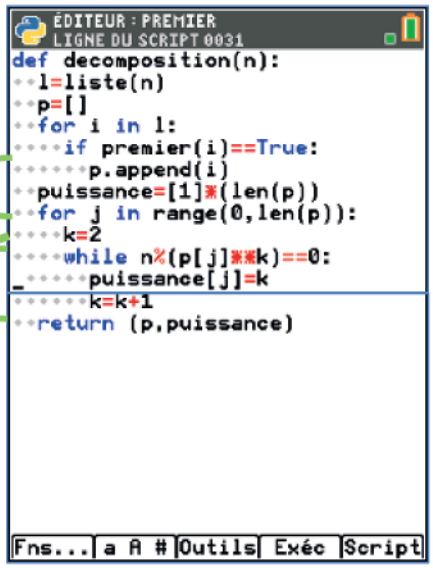
Approfondissement possible

En approfondissement, on peut travailler sur la décomposition en produit de facteurs premiers d'un entier naturel. La fonction **decomposition** qui prend comme argument un entier naturel n et qui renvoie la liste des nombres premiers dans la décomposition en facteur premier de n, ainsi que la liste des puissances associées dans cette décomposition de n.

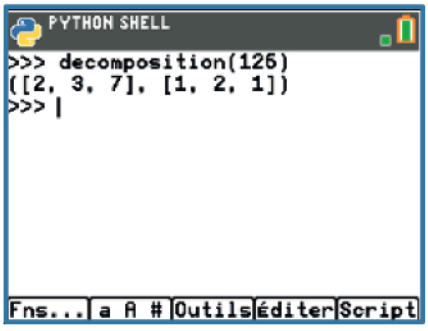
Cette boucle permet de créer la liste des diviseurs premiers de n.

Initialisation de la liste des puissances à 1.
L'instruction `len(p)` permet de renvoyer la longueur de la liste p.

Cette boucle permet de tester la divisibilité de n par les puissances des diviseurs premiers de n et de stocker cette puissance dans la liste puissance.



Remarque : L'instruction `range(0,len(p))` aurait pu être remplacée par `range(len(p))` car par défaut l'instruction range commence à 0.



Pour profiter de tutoriels vidéos, Flasher le QRCode ou cliquer dessus

Le crible d'Eratosthène

On se propose dans ce script d'utiliser l'algorithme d'Eratosthène pour déterminer une liste de nombres premiers.

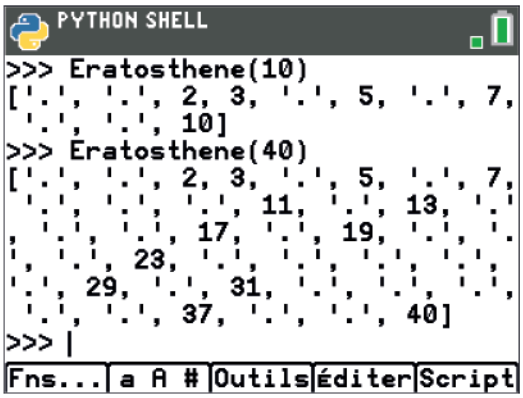
Dans un script ERATOS

Ecrire une fonction `Eratosthene` qui prend comme argument $n \in \mathbb{N}^*$ et qui, à partir de la liste `[0,1,2,3,4,5,…,n]` élimine tous les entiers non premiers (on les remplacera par un `.`) de la liste selon l'algorithme d'Eratosthène et renvoie cette liste (contenant donc uniquement des points et des nombres premiers).

On rappelle l'algorithme d'Eratosthène :

```
Fonction Eratosthene(n)
nb est la liste contenant tous les nombres de 0 à  n
Affecter un . au deux premiers éléments de la liste nb
Pour i allant de 0 à n
Si nb[i] est différent d'un .  #comme c'est un nombre premier on va effacer tous ses multiples
Pour j allant de 2i à n avec un pas de i
Affecter un . à nb[j]
FinPour
FinSi
FinPour
Renvoyer la liste nb
```

Ecrire cette fonction en Python en respectant la bonne indentation !



Le crible d'Eratosthène

Fonction `Eratosthene`

Traduisons l'algorithme en Python :

Fonction `Eratosthene(n)`
    `nb` est la liste contenant tous les nombres de `0` à n
    Affecter un `.` au deux premiers éléments de la liste `nb`
    Pour `i` allant de `0` à `n`
        Si `nb[i]` est différent d'un `.` *#comme c'est un nombre premier on va effacer tous ses multiples*
            Pour j allant de `2*i` à `n` avec un pas de `i`
                Affecter un `.` à `nb[j]`
            FinPour
        FinSi
    FinPour

La fonction en Python ne pose pas de difficulté particulière.

Notons cependant deux points :

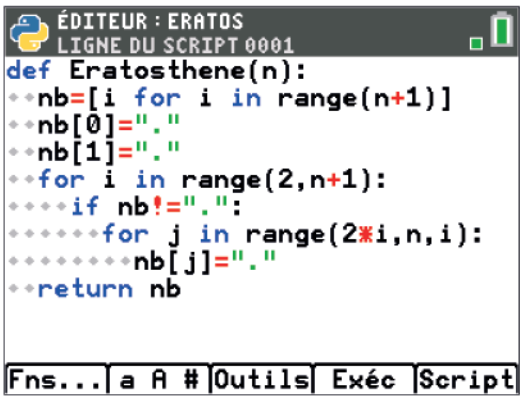
- On définit la liste `nb` de façon concise grâce à la syntaxe Python :

```
nb=[i for i in range(n)]
```

On aurait pu utiliser classiquement :

```
nb=[]
for i in range(n)
  nb.append(i)
```

- On rappelle que le symbole différent en Python est `!=`



Thème: Arithmétique

Niveau: 2nde

TI-83 Premium CE

python™

Multiples et nombres premiers

Compétences visées

- **chercher**, expérimenter – en particulier à l'aide d'outils logiciels ;
- **modéliser**, faire une simulation, valider ou invalider un modèle ;
- **représenter**, choisir un cadre (numérique, algébrique, géométrique...), changer de registre ;
- **calculer**, appliquer des techniques et mettre en œuvre des algorithmes.

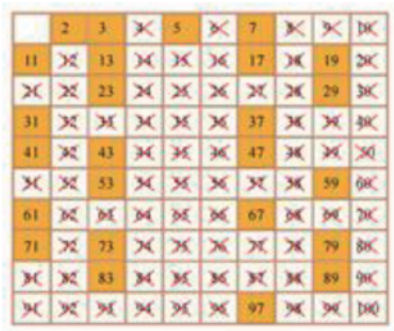
Ces compétences sont mises en œuvre dans le cadre de l'extrait du programme de 2nde GT ci-dessous :

- « Pour des entiers a et b donnés, déterminer le plus grand multiple de a inférieur ou égal à b. »
- « Déterminer si un entier naturel est premier. »

Situation déclenchante

Le crible d'Eratosthène

Ce crible, qui permet de trouver tous les entiers premiers jusqu'à un entier spécifié.



Voici le principe du crible :

1) Écrire tous les entiers de 2 à n.

2) Enlever (ou barrer) les multiples de 2 sauf 2.

3) Récupérer le plus petit nombre non barré, c'est à dire 3, et barrer les multiples de 3 sauf 3, etc...

4) Test d'arrêt : On s'arrête dès qu'on a dépassé la racine carrée de n.

5) Les nombres restants sont les nombres premiers inférieurs ou égaux à n.

.

Problématique

Ecrire un script qui simule le fonctionnement du crible d'Eratosthène et qui permet de trouver les nombres premiers inférieurs ou égaux à un entier spécifié.

Fiche méthode

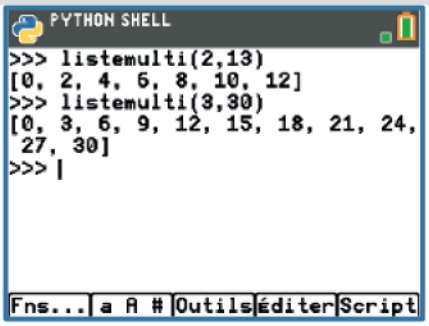
Proposition de résolution

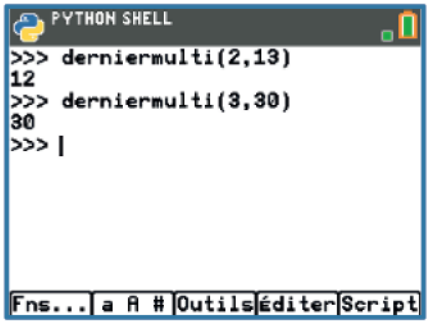
On **crée trois fonctions dans ce script**:

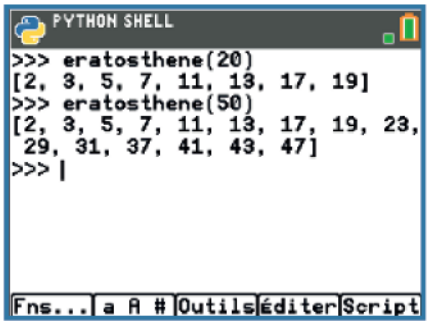
- Une fonction **listemulti** qui prend comme arguments deux entiers naturels a et n et qui renvoie la liste des multiples de a inférieurs ou égaux à n.

- Une fonction **derniermulti** qui prend comme arguments deux entiers naturels a et n et qui renvoie le plus grand multiple de a inférieur ou égal à n.

- Une fonction **eratosthene** qui prend comme argument un entier naturel n et qui renvoie la liste des entiers premiers inférieurs ou égaux à n en utilisant le principe du crible d'Eratosthène.





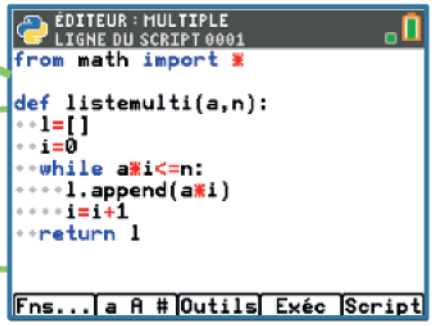


Etapes de résolution

L'instruction **from math import** * permet d'ajouter les fonctions de la bibliothèque `math` (on aura besoin de la fonction racine carré dans le programme suivant)

Fonction **listemulti** qui prend comme arguments deux entiers naturels a et n et qui renvoie la liste des multiples de a inférieurs ou égaux à n.

L'instruction `l=[]` permet d'initialiser la liste `l` avec la liste vide. L'instruction `l.append(a*i)` permet de rajouter `a*i` à la liste `l`.



Pour profiter de tutoriels vidéos, Flasher le QRCode ou cliquer dessus

Fiche méthode

Etapes de résolution

C'est un principe à retenir : On peut appeler une fonction (ici : la fonction **listemulti**) à l'intérieur d'une autre fonction (ici: **derniermulti**). L'utilisation successive de fonctions en python rend le script dans son ensemble plus lisible.

La fonction **derniermulti** qui prend comme arguments deux entiers naturels a et n et qui renvoie le plus grand multiple de a inférieur ou égal à n.

```
ÉDITEUR : MULTIPLE
LIGNE DU SCRIPT 0022
def derniermulti(a,n):
  l=listemulti(a,n)
  a=len(l)-1
  return l[a]
Fns... | a A # | Outils | Exéc | Script
```

La fonction **eratosthene** qui prend comme argument un entier naturel n et qui renvoie la liste des entiers premiers inférieurs ou égaux à n.

On va créer une liste nommée multiple dans laquelle on va stocker tous les multiples des premiers entiers inférieurs ou égaux à la partie entière de $\sqrt{n}$.Pour cela, on va faire appel à la l'instruction **listemulti(i,n)** définie précédemment. Il faudra cependant enlever les 2 premiers éléments de la liste (instruction `del l[0]` suivi de `del l[0]`).

L'instruction `multiple=multiple+l` permet de fusionner les deux listes.

```
ÉDITEUR : MULTIPLE
LIGNE DU SCRIPT 0033
def eratosthene(n):
  a=floor(sqrt(n))
  multiple=[]
  premier=[]
  for i in range (2,a+1):
    l= listemulti(i,n)
    del l[0]
    del l[0]
    multiple=multiple+l
  for i in range (2,n+1):
    if i not in multiple:
      premier.append(i)
  return premier
Fns... | a A # | Outils | Exéc | Script
```

Une fois la liste de multiples créée, d'après le crible d'Eratosthène, les entiers premiers sont ceux qui ne font pas partie de la liste. On détermine donc à l'aide d'une boucle les éléments non présents dans la liste et on les ajoute à la liste des nombres premiers. L'instruction `premier.append(i)` permet d'ajouter l'entier `i` à la liste `premier`.

Pour profiter de tutoriels vidéos, Flasher le QRCode ou cliquer dessus

Nombre premier : Conjecture de Goldbach

Compétences visées

- **chercher**, expérimenter – en particulier à l'aide d'outils logiciels ;
- **modéliser**, faire une simulation, valider ou invalider un modèle ;
- **représenter**, choisir un cadre (numérique, algébrique, géométrique...), changer de registre ;
- **calculer**, appliquer des techniques et mettre en œuvre des algorithmes.

Ces compétences sont mises en œuvre dans le cadre de l'extrait du programme de 2nde GT ci-dessous :

« Déterminer si un entier naturel est premier. » ou dans le cadre de l'extrait du programme de spécialité mathématiques en terminale S : « questionnement sur les nombres premiers »

Situation déclenchante

La conjecture de Goldbach affirme que tout nombre pair supérieur ou égal à 4 est la somme de 2 nombres premiers. Peut-on la tester sur les premiers entiers naturels ?

Problématique

Ecrire un script qui permet de vérifier cette conjecture pour un entier naturel n donné. Le script devra fournir tous les couples d'entiers premiers dont la somme fait n.

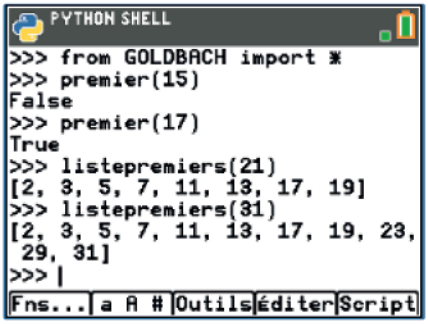
Fiche méthode

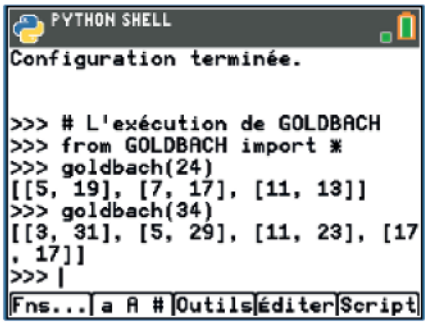
Proposition de résolution

A partir d'une liste de nombres premiers, on teste toutes les sommes possibles de manière à obtenir le nombre souhaité.

Ainsi, **on crée trois fonctions dans ce script** :

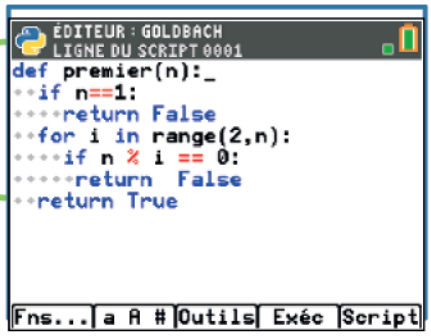
- Une fonction **premier** qui prend comme paramètre un entier naturel n et qui renvoie vrai si n est premier et faux sinon.
- Une fonction **listepremiers** qui prend comme paramètre un entier naturel n et qui renvoie la liste des nombres premiers inférieurs ou égaux à n.
- Une fonction **goldbach** qui prend comme argument un entier naturel n et qui renvoie une liste de couples d'entiers premiers dont la somme fait n.





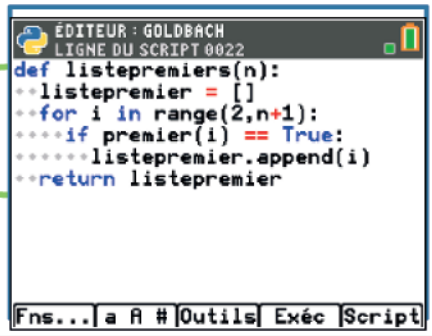
Etapes de résolution

Fonction **premier** :
L'instruction `for i in range(2,n)` permet de créer une boucle avec `i` variant de 2 à `n-1`.
Pour avoir un script plus rapide on aurait pu s'arrêter à $\text{floor}(\sqrt{n}) + 1$ dans la boucle.
L'instruction `n%i` permet de renvoyer le reste dans la division euclidienne de `n` par `i`.



C'est un principe à retenir : On peut appeler une fonction (ici : la fonction **premier**) à l'intérieur d'une autre fonction (ici : **listepremier**). L'utilisation successive de fonctions en python rend le script dans son ensemble plus lisible.

La fonction **listepremiers :**
L'instruction `listepremier.append(i)` permet de rajouter le nombre i à la liste `listepremier`.



Pour profiter de tutoriels vidéos, Flasher le QRCode ou cliquer dessus

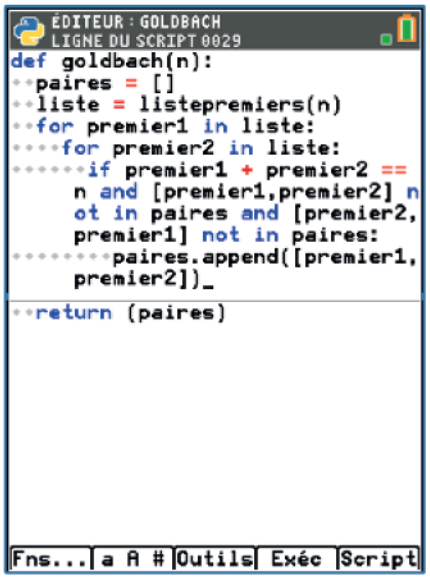
Fiche méthode

Etapes de résolution

La fonction **goldbach** :
On fait appel à la fonction **listepremiers** pour obtenir une liste de nombres premiers.

On utilise une double boucle pour générer toutes les sommes possibles de nombres premiers à partir de la liste obtenue grâce à la fonction **listepremiers.**

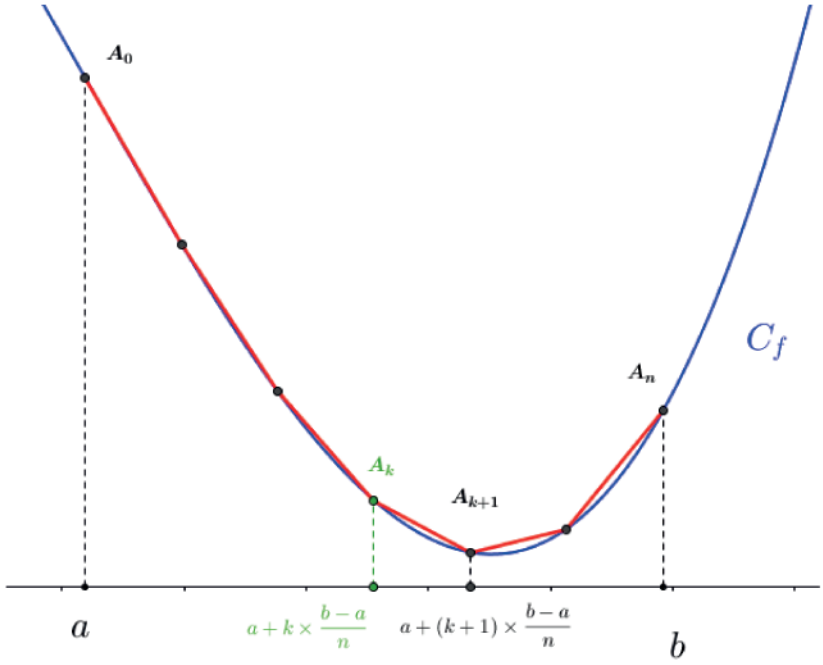
La structure conditionnelle permet de tester si la somme est bien égale au nombre recherché et si la paire n'a pas déjà été sélectionnée.



Pour profiter de tutoriels vidéos, Flasher le QRCode ou cliquer dessus

Calcul de la longueur d'une courbe.

En se plaçant dans tout l'exercice dans un repère orthonormé $(O,\vec{i},\vec{j})$, on va approcher la longueur d'une courbe d'un arc rectifiable à l'aide d'une ligne polygonale dont les extrémités sont des points de la courbe.



Dans un script LCOURBE

1°) Ecrire une fonction `d` qui prend comme paramètres les réels x_A, y_A, x_B, y_B et qui renvoie la distance AB.

Application : Calculer AB et CD avec $A\begin{vmatrix}1\\2\end{vmatrix}$, $B\begin{vmatrix}-1\\3\end{vmatrix}$, $C\begin{vmatrix}-2\\4\end{vmatrix}$ et $D\begin{vmatrix}1\\0\end{vmatrix}$

2°) Ecrire une fonction `f` qui prend comme paramètre le réel `x` et qui renvoie x^2+2x+3 .

Application : Calculer $f(-1)$ et $f(2)$.

3°) Ecrire une fonction `lcourbe` qui prend comme paramètres les réels a, b et n un entier non nul et qui renvoie le calcul approché de la longueur de la courbe représentant la fonction `f` sur $[a,b]$ en partageant $[a,b]$ en n parties égales.

On rappelle :

$$l_{C_f}=\sum_{k=0}^{n-1}A_kA_{k+1} \quad \text{avec} \quad A_k\begin{vmatrix}x_k\\f(x_k)\end{vmatrix} \text{ et } x_k=a+\frac{k(b-a)}{n}$$

Application avec $f(x)=x^2+2x+3$, $x\in[-1,2]$ et $n=10$ puis $n=100$ et $n=1000$

PYTHON SHELL
```
>>> distance(1,2,-1,3)
2.23606797749979
>>> distance(-2,4,1,0)
5.0
```

PYTHON SHELL
```
>>> f(-1)
2
>>> f(2)
11
```

PYTHON SHELL
```
>>> # Shell Reinitialized
>>>
>>>
>>> from LCOURBE import *
>>> lcourbe(-1,2,10)
9.73969171859895
```

Calcul de la longueur d'une courbe.

Fonction `d`

1°) On va utiliser la fonction racine carrée, il faut donc lire la bibliothèque math auparavant.

On rappelle que puissance 2 s'écrit `**2`.

ÉDITEUR : LCOURBE
LIGNE DU SCRIPT 0008

```
from math import *
def distance(xa,ya,xb,yb):
  d=sqrt((xb-xa)**2+(yb-ya)**2)
  return d
```

Fonction `f`

2°) On pouvait gagner une ligne de code en ne définissant pas `y` et en écrivant directement return `x**2+2*x+3`.

ÉDITEUR : LCOURBE
LIGNE DU SCRIPT 0015

```
def f(x):
  y=x**2+2*x+3
  return y
```

Fonction `lcourbe`

3°) On souhaite calculer :

$$\sum_{k=0}^{n-1} A_k A_{k+1} \text{ avec } A_k \left|_{f(x_k)}^{x_k} \right. \text{ et } x_k = a + \frac{k(b-a)}{n}$$

Il s'agit de calculer une somme qu'on va stocker dans la variable `s`.

On initialise `s` à 0.

Il y a `n` termes dans cette somme, on va donc utiliser une boucle `for i in range(n)` qui va tourner n fois.

A chaque tour de boucle il faut :

- Calculer x_k et x_{k+1}. On a nommé ces deux variables `x1` et `x2` dans le script.
- Calculer leurs images par f. On appelle `y1` et `y2` ces images.
- Ajouter à `s` la distance entre les deux points de coordonnées (`x1,y1`) et (`x2,y2`).

On renvoie la somme `s` à la fin du script.

On vérifie notre calcul en définissant la fonction dans l'application fonction de la TI-83 Premium CE :

Et à l'aide d'un calcul d'intégrale classique on vérifie notre résultat :

ÉDITEUR : LCOURBE
LIGNE DU SCRIPT 0019

```
def lcourbe(a,b,n):
  s=0
  for k in range(n):
    x1=a+k*(b-a)/n
    x2=a+(k+1)*(b-a)/n
    y1=f(x1)
    y2=f(x2)
    s=s+distance(x1,y1,x2,y2)
  return s
```

Fns... | a A # | Outils | Exéc | Script

PYTHON SHELL

```
>>> # Shell Reinitialized
>>>
>>>
>>> from LCOURBE import *
>>> lcourbe(-1,2,10)
9.73969171859895
>>> lcourbe(-1,2,100)
9.747014779047248
>>> lcourbe(-1,2,10000)
9.747088751210628
>>>
```

Fns... | a A # | Outils | éditer | Script

NORMAL FLOTT AUTO RÉEL RAD MP

Graph1 Graph2 Graph3

$Y_1 = X^2+2X+3$

NORMAL FLOTT AUTO RÉEL RAD MP

$$\int_{-1}^{2} \left(\sqrt{1+\left(\frac{d}{dX}(Y_1(X))|_{X=X} \right)^2} \right) dX$$

9.74708876

Thème: fonctions

Niveau: 2nde

TI-83 Premium CE

python™

Calcul approché de longueur d'une portion de courbe

Compétences visées

- **chercher**, expérimenter – en particulier à l'aide d'outils logiciels ;
- **modéliser**, faire une simulation, valider ou invalider un modèle ;
- **représenter**, choisir un cadre (numérique, algébrique, géométrique...), changer de registre ;
- **calculer**, appliquer des techniques et mettre en œuvre des algorithmes.

Ces compétences sont mises en œuvre dans le cadre de l'extrait du programme de 2nde GT ci-dessous :

« Algorithme de calcul approché de longueur d'une portion de courbe représentative de fonction. »

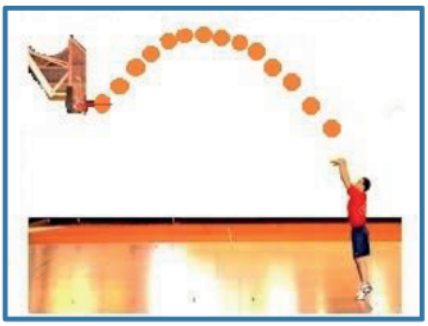
Situation déclenchante

Estimation de la longueur d'une courbe

On sait que la trajectoire d'un ballon est une portion de parabole, représentant une fonction polynôme du second degré.

On souhaite déterminer ***la longueur du trajet du ballon*** entre le moment où il quitte les mains du joueur et l'arrivée au panier.

Dans un souci de simplification, on traitera dans un premier temps une fonction de référence (ici, la fonction carré sur l'intervalle [0 ; 1] et on reviendra à ce problème dans un second temps).



Problématique

Soit f la fonction définie sur l'intervalle [0 ; 1] par : $f(x) = x^2$ et C sa courbe représentative dans un repère orthonormé.

Quelle est la longueur de la courbe C ?

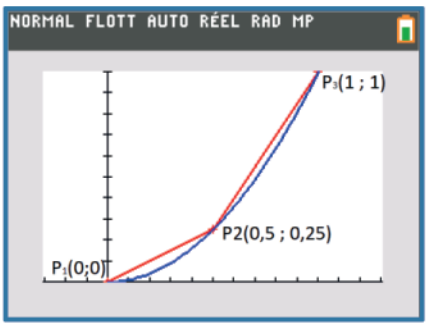
Fiche méthode

Proposition de résolution

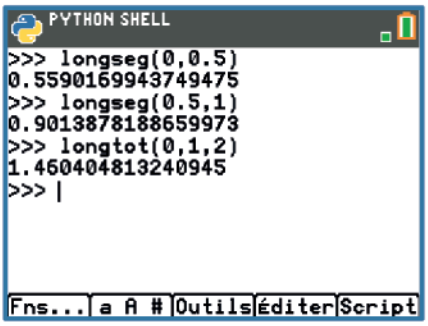
Le principe est d'approcher la courbe par une ligne brisée composée de segments dont on ajoutera les longueurs.

- Écrire la fonction **longseg** qui a comme paramètres xA et xB et qui renvoie la longueur du segment [AB], avec A(xA ; f(xA)) et B(xB ; f(xB)).
- Ci-contre, on approche la courbe par deux segments. Leurs longueurs sont estimées par la fonction **longseg**, et la longueur totale par **longtot** : P_1P_2 = longseg(0,0.5) ~ 0,559

  P_2P_3 = longseg(0.5,1) ~ 0,901



La longueur totale est la somme de ces longueurs, donnée par **longtot**(0,1,2). Dans cette instruction, les paramètres 0, 1, 2 signifient que l'on s'est placé entre les abscisses 0 et 1 avec 2 segments dont les abscisses des extrémités sont 0 et 0,5 pour le premier segment, 0,5 et 1 pour le second segment.

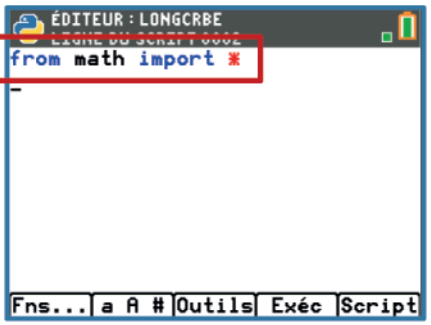


Ainsi, **on crée trois fonctions dans ce programme** :

- Une fonction **f** qui a pour paramètre x et qui renvoie l'image de x par la fonction utilisée dans le contexte, la fonction carré dans ce cas ;
- Une fonction **longseg** qui a pour paramètres xA et xB et qui renvoie la longueur du segment [AB] avec A (xA ; f(xA)) et B(xB ; f(xB)) ;
- Une fonction **longtot** qui a pour paramètres xA, xB et n ; elle renvoie la somme des longueurs des n segments qui approchent la courbe.

Remarques

Importation en préambule du code de la bibliothèque « math » par « **from math import ✶** » pour pouvoir utiliser la fonction **sqrt** (racine carrée)



Pour profiter de tutoriels vidéos, Flasher le QRCode ou cliquer dessus

Fiche méthode

Etapes de résolution

Création de la fonction **f** dont l'expression est $f(x)=x^2$

```
ÉDITEUR : LONGCRBE
LIGNE DU SCRIPT 0010
from math import *

def f(x):
  return x**2

def longseg(xA,xB):
  return sqrt((xB-xA)**2+(f(xB)-
    f(xA))**2)

_
Fns... | a A # | Outils | Exéc | Script
```

Fonction **longseg** donnant la longueur d'un segment [AB] avec A(xA ; f(xA)) et B(xB ; f(xB)).
On notera l'utilisation de la fonction **f** précédemment définie qui rend l'écriture de cette fonction simple et proche des notations usuelles.

C'est un principe à retenir : l'utilisation successive de fonctions en python rend le programme dans son ensemble plus lisible et plus efficace.

```
ÉDITEUR : LONGCRBE
LIGNE DU SCRIPT 0014
def longseg(xA,xB):
  return sqrt((xB-xA)**2+(f(xB)-
    f(xA))**2)

def longtot(xA,xB,n):
  pas=(xB-xA)/n
  long=0
  for i in range(n):
    long=long+longseg(xA+i*pas,x
      A+(i+1)*pas)
  return long
Fns... | a A # | Outils | Exéc | Script
```

Fonction **longtot** donnant une valeur approchée de la longueur de la courbe.
Dans cette fonction, on va faire appel à la fonction **longseg,** ce qui allège le script.

Pour cela, on définit un pas : il sera constant et égal à $\frac{x_B-x_A}{n}$.

Sur chaque intervalle du type $[x_A + i \times pas\ ;\ x_A + (i+1) \times pas]$, on approche la longueur de la partie de courbe comprise entre ces abscisses en utilisant la fonction **longseg.**

On a créé une variable **long** (initialisée à 0) qui cumulera la longueur de chaque segment pour approximer la longueur totale de la courbe, valeur retournée par la fonction.

Retour au problème

Ainsi, ce programme permet de répondre au problème de départ en donnant une valeur approchée de la longueur de la courbe représentative de la fonction carré sur l'intervalle [0 ; 1].

On trouve ici une longueur d'environ 1,47894.

```
PYTHON SHELL
>>>
>>> longtot(0,1,10)
1.478197397487329
>>> longtot(0,1,100)
1.478935403974238
>>> longtot(0,1,1000)
1.478942783008997
>>> longtot(0,1,10000)
1.478942856799243
>>> longtot(0,1,50000)
1.478942857514788
Fns... | a A # | Outils | éditer | Script
```

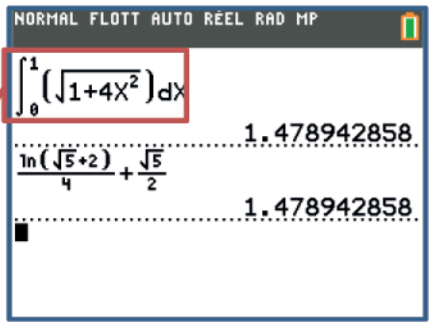
Pour profiter de tutoriels vidéos, Flasher le QRCode ou cliquer dessus

Fiche méthode

Un peu de théorie

Sur le plan théorique, la longueur de la courbe représentative d'une fonction f sur un intervalle [a ; b] (sur lequel elle est continûment dérivable) est donnée par : $\int_a^b \sqrt{1+f'(x)^2}\,dx$

Cela permet d'obtenir la valeur exacte par rapport au problème initial ainsi qu'une valeur approchée et d'apprécier la précision de l'algorithme selon la valeur de n.

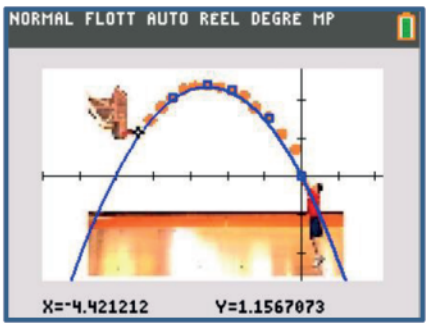


Retour à la situation déclenchante

En important l'image du ballon de basket dans la calculatrice et en la mettant en arrière-plan, on peut approcher la trajectoire du ballon par une courbe représentant une fonction du second degré.

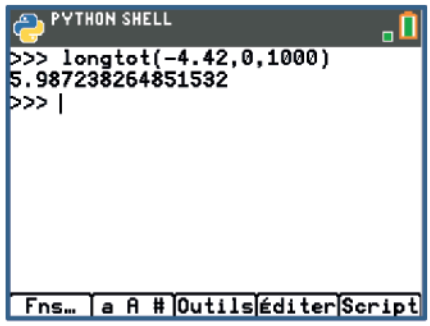
Remarque : on réalise cette importation par le biais de TI-Connect CE; on affiche ensuite l'image en tant qu'« Arrière-plan » (passer par « Format » / « Arrière-plan » et les flèches directionnelles pour faire défiler les images enregistrées dans la calculatrice).

Il faudra adapter le repère pour que les coordonnées soient réalistes (taille du joueur et distance au panier).



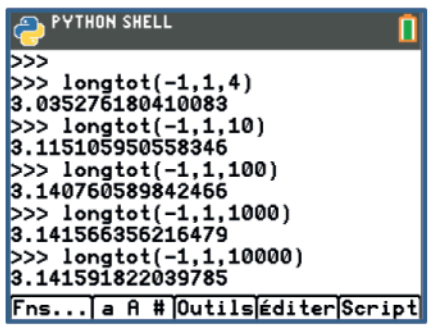
On obtient comme équation $y = -0{,}3661x^2 - 1{,}8538x$ en ayant choisi Xmin=-7 / Xmax=2.2 / Ymin=-2.7 et Ymax=2.8

On a pour cela utilisé une fonctionnalité de la TI 83 Premium CE permettant de déterminer une fonction approchant des points (menu stats, puis CALC, puis E:TracéAjust-éq ; il faut alors placer les points qui seront interpolés. On termine par éQRég et le choix RégDeg2).



Les abscisses sont à prendre entre -4,42 (abscisse du panier) et 0 (abscisse des mains du joueur ; on a calé l'origine volontairement au niveau des mains du joueur).

On obtient une trajectoire - entre les mains du joueur et le panier - d'une longueur d'environ 6 m en ayant remplacé dans la fonction **f** du script `return x**2` par : `return -0.3661*x**2-1.8538*x`



Une application intéressante

En appliquant la méthode précédente à la fonction $f(x) = \sqrt{1-x^2}$ sur l'intervalle [-1 ;1], on obtient une approximation de π

Pour profiter de tutoriels vidéos, Flasher le QRCode ou cliquer dessus

Thème: Arithmétique

Niveau: 2nde

TI-83 Premium CE

python™

Approximation de $\sqrt{2}$ par balayage.

Compétences visées

- **chercher**, expérimenter – en particulier à l'aide d'outils logiciels ;
- **modéliser**, faire une simulation, valider ou invalider un modèle ;
- **représenter**, choisir un cadre (numérique, algébrique, géométrique...), changer de registre ;
- **calculer**, appliquer des techniques et mettre en œuvre des algorithmes.

Ces compétences sont mises en œuvre dans le cadre de l'extrait du programme de 2^{nde} GT ci-dessous :

" Déterminer par balayage un encadrement de $\sqrt{2}$ d'amplitude inférieure ou égale à10^{-n}."

Situation déclenchante

Le nombre $\sqrt{2}$ n'est pas un nombre rationnel : il ne peut pas s'écrire sous la forme d'une fraction d'entier. Comment déterminer une valeur approchée de ce nombre ? Comme $1 \leq 2 \leq 4$, par croissance de la fonction racine carrée sur son domaine de définition, on peut encadrer $\sqrt{2}$, par les entiers 1 et 2. Comment obtenir en encadrement plus précis ?

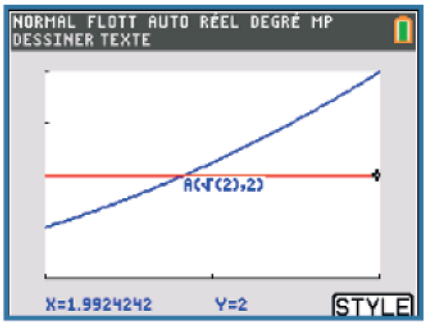
Problématique

Ecrire un script qui permet de déterminer un encadrement d'amplitude 10^{-n}de $\sqrt{2}$ .Les bornes de l'intervalle seront arrondies à 10^{-n}.

Fiche méthode

Proposition de résolution

On trace la représentation graphique de la fonction f définie sur [1 ; 2] par : $f(x) = x^2$.

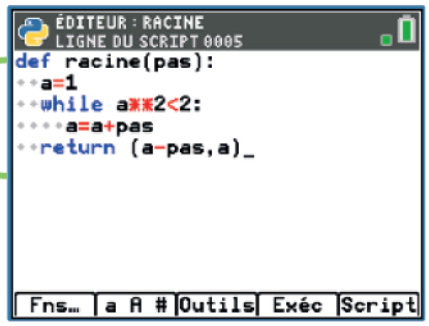


En traçant le tableau de variation de la fonction f sur [1; 2] par le corollaire du théorème des valeurs intermédiaires l'équation $f(x) = 2$ possède une unique solution sur [1 ;2] .

Ainsi, on crée une fonction **racine** qui prend comme argument n un entier naturel et qui renvoie deux réels avec n décimales encadrant $\sqrt{2}$ et ayant un écart de 10^{-n}. Le principe de la fonction est d'incrémenter les abscisses de 10^{-n} tant que les images restent inférieures à 2.

Première idée non satisfaisante

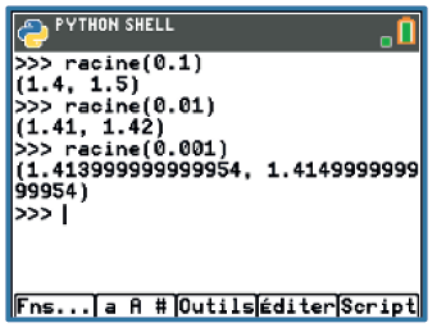
On pourrait naïvement proposer le programme ci contre.



La fonction **racine** permet de retourner 2 nombres dont l'écart est égal au `pas`.

La variable a représente la valeur des abscisses. Tant que l'image de a par la fonction carrée est inférieure à 2 on augmente l'abscisse de la valeur du pas. La boucle `while` s'arrête donc lorsqu'on vient juste de dépasser la valeur recherché.

Observons l'exécution du script pour différentes valeurs du pas.



Lors de l'exécution du script, avec un pas de 0.1 ou de 0.01 , les résultats restent cohérents. Mais avec un pas de 0.001 ou un pas plus précis, les résultats ne correspondent pas aux attentes. D'où provient ce phénomène ?

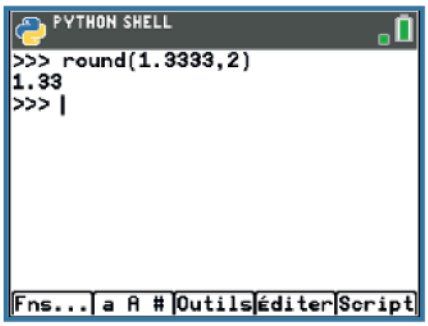
Pour profiter de tutoriels vidéos, Flasher le QRCode ou cliquer dessus

Fiche méthode

Problèmes liés aux nombres à virgule

Les nombres à virgule flottante sont représentés, au niveau matériel, en fractions de nombres binaires (base 2). Malheureusement, la plupart des fractions décimales ne peuvent pas avoir de représentation exacte en fractions binaires. Par conséquent, en général, les nombres à virgule flottante qui sont saisies **sont seulement approximés** en fractions binaires pour être stockés dans la machine.

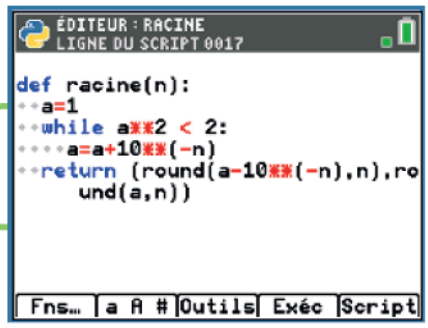
Pour éviter ce type de problème nous aurons recours dans le script à la commande `round(a,b)` qui arrondi le nombre a avec b chiffres après la virgule.



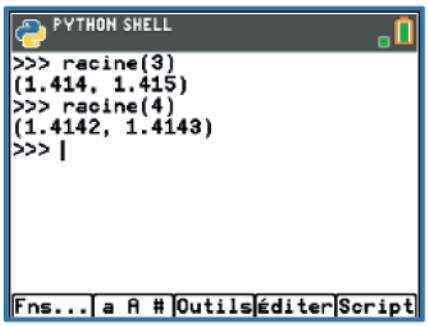
Etapes de résolution

La fonction **racine** prend comme argument n un entier naturel et renvoie deux réels avec n décimales, encadrant $\sqrt{2}$, ayant un écart de 10^{-n}. Round permet de maitriser la précision.

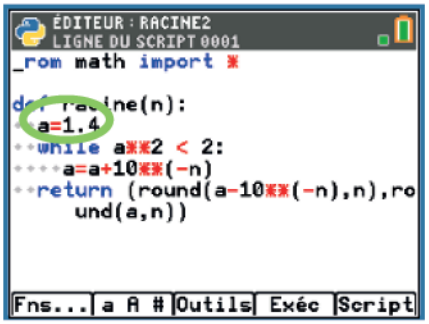
La variable a représente la valeur des abscisses. Tant que l'image de a par la fonction carré est inférieure à 2 on augmente l'abscisse de la valeur du pas (10^{-n}). La boucle while s'arrête donc lorsqu'on vient juste de dépasser le point recherché.



On obtient les résultats ci-contre lors de l'exécution de la fonction.



Pour obtenir un temps de réponse satisfaisant si l'on souhaite une précision plus grande (à partir de `n=6`) il est préférable de changer la valeur de départ dans le script : on peut par exemple saisir `a=1.4` (résultat obtenu précédemment à l'aide du script).



Remarque : Une amélioration possible de la fonction **racine** est de transformer la variable a en paramètre.

Pour profiter de tutoriels vidéos, Flasher le QRCode ou cliquer dessus

Recherche d'extremum d'une fonction.

Soit f la fonction définie sur $\mathbb{R}$ par $f(x) = x^3 - x^2 - 5x + 3$.

Soit a et b deux réels, $a < b$ et $d \in \mathbb{R}_+^*$. Soit $A = \{a + kd | k \in \mathbb{N} \; et \; a + kd \leq b\}$, donc $A = \{a, a + d, a + 2d, .., a + nd\}$ avec n le plus grand entier tel que $a + nd < b$.

On se propose dans cette activité de rechercher les réels $ymin$ et $ymax$ vérifiant :

$ymin = min\{f(x) | x \in A\}$ ainsi que $ymax = max\{f(x) | x \in A\}$.

Cela correspond à la recherche d'une valeur approchée des extrema par balayage de f sur $[a, b]$ (on ne prend en compte que les valeurs de $x \in A$).

Pour que ces valeurs $(ymin, ymax)$ correspondent à des valeurs approchées convenables des extrema de f sur $[a, b]$ il faudra prendre un pas assez petit et une fonction assez régulière (ce qui est le cas dans la plupart des fonctions étudiées au lycée).

Dans un script MINMAX

1°) Ecrire une fonction f qui prend en argument le réel x et qui renvoie $x^3 - x^2 - 5x + 3$.

On testera cette fonction en calculant les images ci-contre.

```
>>> f(-2)
1
>>> f(1)
-2
>>> f(3/2)
-3.375
```

2°) Ecrire une fonction `mini` qui prend comme arguments les réels a, b et d et qui, en incrémentant x avec un pas de d, renvoie le couple $(xmin, ymin)$ que nous avons défini ci-dessus.

Quel est alors le minimum de f sur $[-2; 2]$ obtenu par cette méthode ?

```
PYTHON SHELL
>>> mini(-2,2,0.01)
(1.670000000000003, -3.481437)
```

3°) Ecrire une fonction maxi qui prend comme arguments les réels a, b et d et qui, en incrémentant x avec un pas de d, renvoie le couple $(xmax, ymax)$.

Quel est alors le maximum de f sur $[-2; 2]$ obtenu par cette méthode ?

```
PYTHON SHELL
>>> maxi(-2,2,0.01)
(-1.0, 6.0)
```

Recherche d'extremum d'une fonction.

Fonction `f`

1°) Le fait de définir la fonction f dans le script permettra de la modifier facilement sans avoir à modifier les fonctions qui vont suivre et qui font appel à cette fonction f.

```
ÉDITEUR : MINMAX
LIGNE DU SCRIPT 0002
def f(x):
  y=x**3-x**2-5*x+3
  return y
```

Fonction `mini`

2°) On va utiliser 4 variables : `x`, `y=f(x)`, `xmin` et `ymin`.

`x` va être initialisé à `a` et incrémenté de `d` à chaque tour de boucle. La boucle se poursuit tant que `x` est inférieur ou égal à `b`.

On initialise `xmin` à a et `ymin` à `f(a)`.

Dès que `y` (qui vaut `f(x)`) sera inférieur à `ymin` alors on trouve une valeur plus petite que `ymin`, il faut remplacer `ymin` par cette valeur `y`.

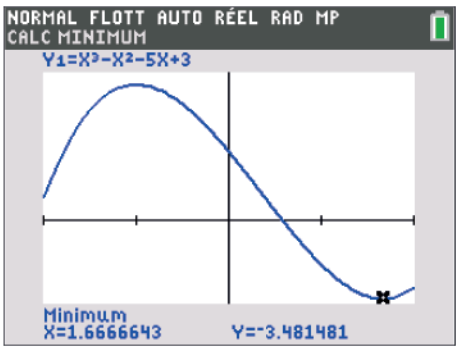
A la fin on renvoie le couple `xmin,ymin`.

A l'exécution de la fonction `mini` avec une précision (sur `x`) de 0,01 on trouve que le minimum de f sur $[-2;2]$ vaut $-3{,}48$ et il est atteint en $1{,}67$.

Les résultats correspondent bien à ce qu'on peut trouver à l'aide du module recherche de minimum et maximum de la TI-83 Premium CE.

```
ÉDITEUR : MINMAX
LIGNE DU SCRIPT 0016
def mini(a,b,d):
  x=a
  xmin=a
  ymin=f(xmin)
  while x<=b:
    x=x+d
    y=f(x)
    if y<ymin:
      xmin=x
      ymin=y
  return xmin,ymin
```

```
PYTHON SHELL
>>> mini(-2,2,0.01)
(1.6700000000000003, -3.481437)
>>> f(1.66)
-3.481304
>>> f(1.67)
-3.481437000000001
>>> f(1.68)
-3.480768000000001
```

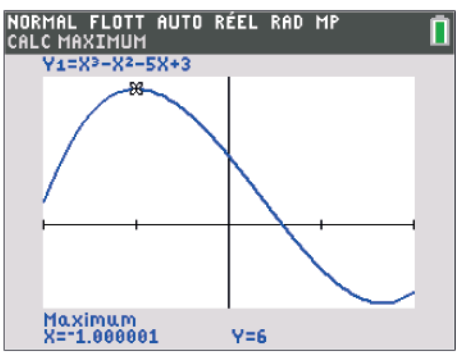


Fonction `maxi`

3°) Il suffit de changer la condition du `if`. On a changé le nom des variables, mais cela n'était pas nécessaire en soit.

```
ÉDITEUR : MINMAX
LIGNE DU SCRIPT 0028
def maxi(a,b,d):
  x=a
  xmax=a
  ymax=f(xmax)
  while x<=b:
    x=x+d
    y=f(x)
    if y>ymax:
      xmax=x
      ymax=y
  return xmax,ymax
```

```
PYTHON SHELL
>>> maxi(-2,2,0.01)
(-1.0, 6.0)
>>> f(-0.99)
5.999601
>>> f(-1)
6
>>> f(-1.01)
5.999599
```



Approximation d'extremums par balayage

Compétences visées

- **chercher**, expérimenter – en particulier à l'aide d'outils logiciels ;
- **modéliser**, faire une simulation, valider ou invalider un modèle ;
- **représenter**, choisir un cadre (numérique, algébrique, géométrique...), changer de registre ;
- **calculer**, appliquer des techniques et mettre en œuvre des algorithmes.

Ces compétences sont mises en œuvre dans le cadre de l'extrait du programme de 2nde GT ci-dessous :

Pour une fonction dont le tableau de variations est donné, algorithmes d'approximation numérique d'un extrémum par balayage.

Situation déclenchante

L'étude de fonctions a pour principal objectif de déterminer des extrema. En effet, dans la vie de tous les jours, cela peut par exemple correspondre à minimiser des coûts de production ou maximiser des bénéfices. Cependant il est assez rare de pouvoir déterminer précisément les coordonnées de ces extrema. Comment faire alors pour obtenir au moins une valeur approchée ?

Problématique

Ecrire un script qui permet de déterminer une approximation par balayage du maximum de la fonction $f(x) = 4x^3 - 20x^2 + 25x$ sur l'intervalle $[\,0, 2.5\,]$.

Ecrire un second script qui permet de déterminer une approximation par balayage du minimum de la fonction $g(x) = -3x^3 + 4x + 1$ sur l'intervalle $[\,-2\,,1\,]$.

Fiche méthode

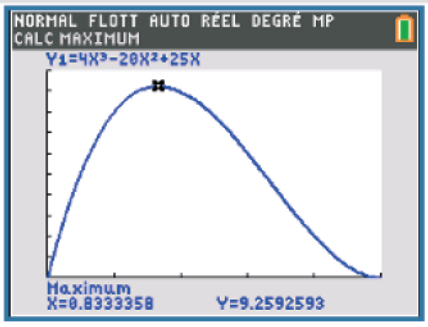
Proposition de résolution

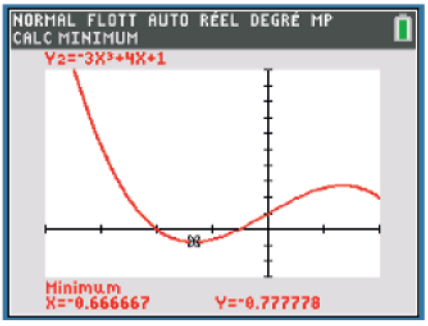
On trace les représentations graphiques des fonctions f et g définies par : $f(x) = 4x^3 - 20x^2 + 25x$ sur l'intervalle $[\,0, 2.5\,]$ et $g(x) = -3x^3 + 4x + 1$ sur l'intervalle $[\,-2\,, 1\,]$.

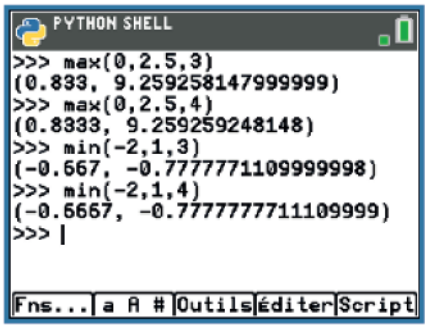
On observe les extrema dont on peut obtenir une approximation à l'aide des fonctions de la calculatrice. Comment fait elle pour les déterminer ? Elle possède un algorithme (similaire à celui proposé) qui permettra d'obtenir une approximation des coordonnées de l'extremum.

Ainsi, **on crée quatre fonctions dans ce script** :

- Une fonction **f** qui prend comme paramètre un réel x et qui renvoie l'image du réel x par la fonction **f**.
- Une fonction **max** qui prend comme paramètres a,b,n , avec a,b des réels et n un entier naturel et qui renvoie une approximation de l'abscisse du maximum de la fonction **f** sur l'intervalle [a,b] avec une précision de n chiffres après la virgule.
- Une fonction **g** qui prend comme paramètre x un réel et qui renvoie l'image du réel x par la fonction **g**.
- Une fonction **min** qui prend comme paramètres (a,b,n) , avec a,b des réels et n un entier naturel et qui permet de renvoyer une approximation du minimum de la fonction **g** sur l'intervalle [a,b] avec une précision de n chiffres après la virgule.



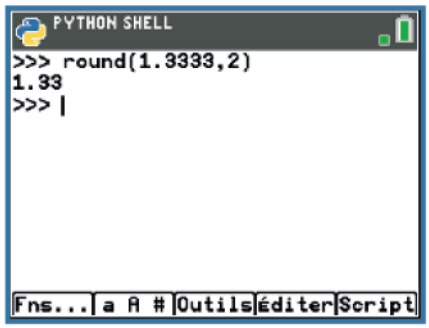




Problèmes liés aux nombres à virgule

Les nombres à virgule flottante sont représentés, au niveau matériel, en fractions de nombres binaires (base 2). Malheureusement, la plupart des fractions décimales ne peuvent pas avoir de représentation exacte en fractions binaires. Par conséquent, en général, les nombres à virgule flottante qui sont saisies **sont seulement approximés** en fractions binaires pour être stockés dans la machine.

Pour éviter ce type de problème nous aurons recours dans le script à la commande round(a,b) qui arrondi le nombre a avec b chiffres après la virgule.

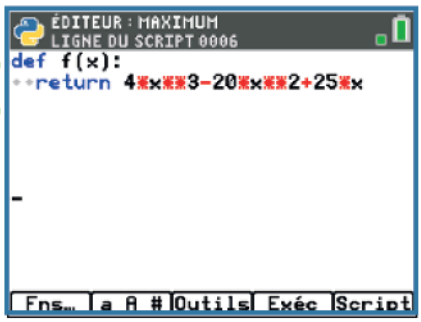


Pour profiter de tutoriels vidéos, Flasher le QRCode ou cliquer dessus

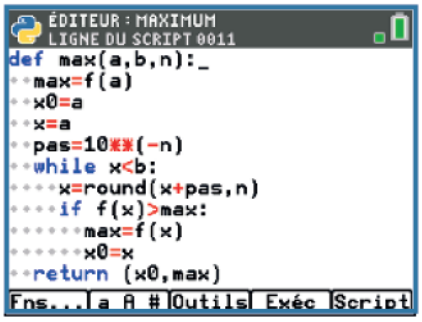
Fiche méthode

Etapes de résolution

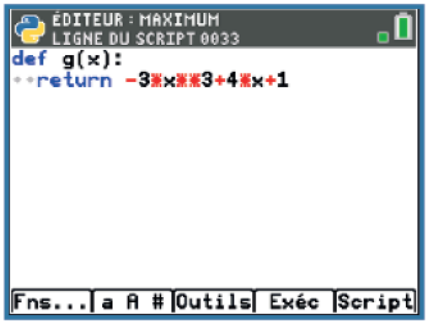
Fonction **f** qui prend comme paramètre un réel x et qui renvoie l'image du réel x par la fonction **f**.



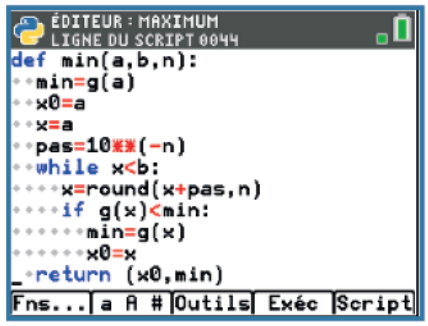
Fonction **max** qui prend comme paramètres a,b,n , avec a,b des réels et n un entier naturel et qui renvoie une approximation de l'abscisse du maximum de la fonction **f** sur l'intervalle [a,b] avec une précision de n chiffres après la virgule.
L'instruction `x=round(x+pas,n)` permet de conserver le pas souhaité.



C'est un principe à retenir : On peut appeler une fonction (ici : la fonction **f**) à l'intérieur d'une autre fonction (ici: **max**).
L'utilisation successive de fonctions en Python rend le programme dans son ensemble plus lisible.



Fonction **min** qui prend comme paramètres (a,b,n) , avec a,b des réels et n un entier naturel et qui permet de renvoyer une approximation de l'abscisse du minimum pour la fonction **g** sur l'intervalle [a,b] avec une précision de n chiffres après la virgule.
L'instruction `x=round(x+pas,n)` permet de conserver le pas souhaité.



Remarque pour gagner du temps : au lieu de chercher le minimum de la fonction g on pouvait chercher l'opposé du maximum de la fonction $-g$ et ainsi éviter le deuxième algorithme.

Pour profiter de tutoriels vidéos, Flasher le QRCode ou cliquer dessus

Rayon d'un cylindre minimisant sa surface

On cherche à minimiser la surface de métal nécessaire à la fabrication d'une boîte de conserve ayant un volume donné v.

Dans un script CONSERVE

1°) Ecrire une fonction `hauteur` qui prend comme arguments le rayon `r` (en cm) de la boîte de conserve et son volume `v` (en cm³) et qui renvoie la hauteur (en cm) de la boîte.

```
>>> hauteur(10,1200)
3.819718634205488
```

2°) Ecrire une fonction `surface` qui prend comme arguments le rayon `r` (en cm) de la boîte de conserve et son volume `v` (en cm³) et qui renvoie la surface (en cm²) de la boîte.

```
>>> surface(10,1200)
868.3185307179587
```

On suppose que $r \in [2,10]$ (en cm).

3°) Ecrire une fonction `dimensions` qui prend comme arguments le volume `v` (en cm³) et qui renvoie le rayon (en cm) et la hauteur (en cm) de la boîte de conserve en utilisant le moins de métal possible.

4°) Application : prendre `v`= $850\ cm^3$ puis `v`= $212 cm^3$.

```
>>> from CONSERVE import *
>>> dimensions(850)
(5.129999999999933, 10.280975466
57202)
>>> dimensions(212)
(3.229999999999974, 6.4681628186
75988)
```

Fonction `hauteur`

1°) Pour un cylindre de rayon r et de hauteur h son volume est :

$$v = \pi r^2 \times h$$

On en déduit que $h = \frac{v}{\pi r^2}$ car $r \neq 0$.

Remarque : `pi` est un nombre issue de la bibliothèque `math`.

```
ÉDITEUR : CONSERVE
LIGNE DU SCRIPT 0010
from math import *
def hauteur(r,v):
  h=v/(pi*r**2)
  return h
```

Fonction `surface`

2°) La surface S de la boîte est constituée de la somme de la surface S_1 du cylindre :

$$S_1 = 2\pi r \times h = 2\pi r \times \frac{v}{\pi r^2} = \frac{2v}{r}$$

Et de la surface S_2 des deux disques supérieur et inférieur :

$$S_2 = 2 \times \pi r^2$$

Ainsi $S = S_1 + S_2 = \frac{2v}{r} + 2 \times \pi r^2$

```
def surface(r,v):
  s=(2*v)/r+2*pi*r**2
  return s
```

Rayon d'un cylindre minimisant sa surface

python

Fonction `dimensions`

3°) On connait le volume v et on cherche la valeur de `r` qui minimise la surface de la boîte S.

On sait que $r \in [2,10]$.

On va faire varier le rayon à l'aide d'une variable `i` qui va s'incrémenter de 0,01 à chaque tour de boucle.

`i` va être initialisé à 2 et la boucle continue tant que `i<10`.

`s` est la surface, initialisée à la surface correspondant à un rayon de 2 (pour le volume v donné).

A chaque tour de boucle, on calcule la surface pour un rayon `i` et si cette surface est plus petite on l'affecte à `s` et on retient le rayon correspondant dans la variable `r`.

A la fin de la boucle on renvoie le couple `r,s` le rayon obtenu pour la surface `s` la plus petite (parmi celles calculées).

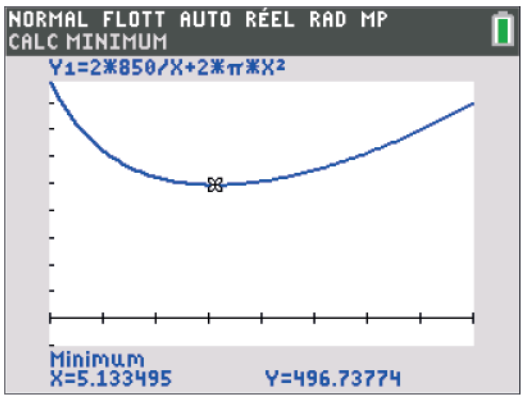
```
ÉDITEUR : CONSERVE
LIGNE DU SCRIPT 0020
def dimensions(v):
  i=2
  r=2
  s=surface(2,v)
  while i<10:
    if surface(i,v)<s:
      s=surface(i,v)
      r=i
    i=i+0.01
  return r,hauteur(r,v)
Fns... | a A # | Outils | Exéc | Script
```

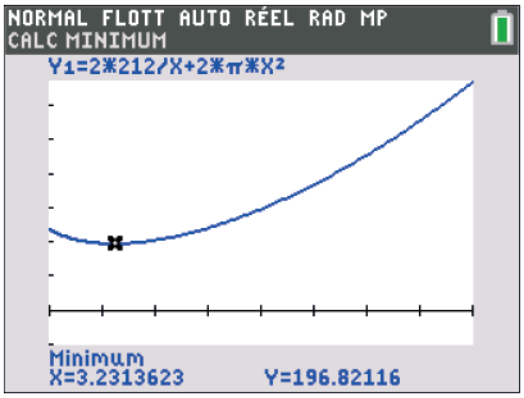
4°) Dans l'application numérique on obtient que pour un volume de 850 cm^3 le rayon qui minimise la surface est $r = 5{,}13$ cm pour une hauteur de $h = 10{,}28$ cm.

De même, pour un volume de 212 cm^3 on obtient un rayon de 3,22 cm pour une hauteur de 6,47 cm.

On peut aussi vérifier ces valeurs à l'aide des outils d'analyse numérique de la TI-83 premium CE :

```
>>> from CONSERVE import *
>>> dimensions(850)
(5.129999999999933, 10.280975466
57202)
>>> dimensions(212)
(3.229999999999974, 6.4681628186
75988)
```





Remarque : Cet exercice correspond à la recherche d'un extremum d'une fonction par balayage.

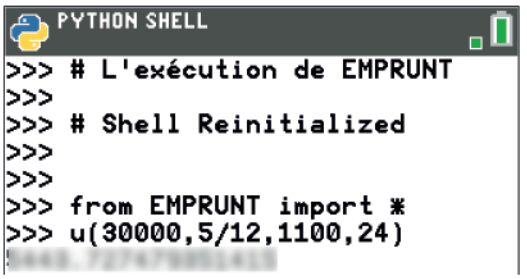
Calcul d'une mensualité d'un prêt

Dans un script EMPRUNT

1°) On emprunte une somme `C` à un taux d'intérêt mensuel de `t`% avec une mensualité de `M`.

Ecrire une fonction `u` qui prend comme arguments `C`, `t`, `M` et `n` et qui renvoie le capital restant dû après `n` mensualités.

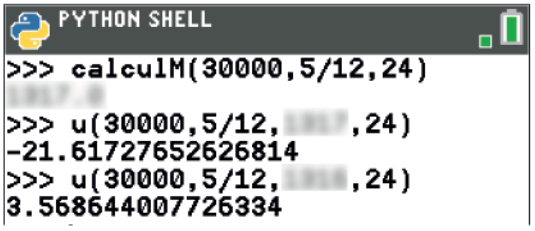
Application : Calculer le capital restant dû si on emprunte 30000€ au taux annuel de 5% avec une mensualité de 1100€ après 24 mois de remboursement.



2°) Ecrire une fonction `calculM` qui prend comme arguments `C`, `t` et `n` et qui renvoie la mensualité (à l'euro près) correspondant à un emprunt d'une somme `C` à un taux d'intérêt mensuel de `t`% avec `n` mensualités.

Indication : On pourra initialiser `M` à `C/n` et incrémenter `M` de 1€ tant que le capital restant dû (après `n` mensualités) est positif.

Application : Calculer la mensualité d'un emprunt de 30.000€ sur 24 mois au taux annuel de 5%.



3°) Ecrire une fonction `cout` qui prend comme arguments `C`, `t` et `n` et qui renvoie le coût du crédit.

Application : Calculer le coût du crédit pour un capital de 30.000€ sur 24 mois au taux annuel de 5%.

4°) Ecrire une fonction `pourcent` qui prend comme arguments `C`, `t` et `n` et qui renvoie le pourcentage du coût du crédit par rapport à la somme empruntée.

Application : Quel est le pourcentage du coût du crédit par rapport au capital emprunté pour le crédit suivant :

30.000€ à 5% annuel sur 24 mois.

Qu'en est-il si on emprunte sur 60 mois ?

5°) Question subsidiaire :

Les banques utilisent la règle (fausse) suivante :

Le taux mensuel = taux annuel / 12

Pour en emprunt de 250.000€ à 3% annuel sur 240 mois, calculer le coût du crédit avec la méthode de calcul de la banque puis avec la méthode mathématiquement correcte (qui consiste à calculer le taux mensuel équivalent).

Quelle est la somme dont l'emprunteur s'est fait spolier ?

Calcul d'une mensualité d'un prêt

Fonction `u`

On initialise la variable `d` à `c` le montant dû initialement.

Pour chacun des `n` mois, le capital dû augmente de `t`% moins la mensualité remboursée.

Il suffit de faire ce calcul `n` fois à l'aide d'une boucle `for`.

Application : Calculer le capital restant dû si on emprunte 30.000€ au taux annuel de 5% avec une mensualité de 1100€ après 24 mois de remboursement.

On trouve un capital restant dû de 5443,73€.

ÉDITEUR : EMPRUNT
LIGNE DU SCRIPT 0001

```
def u(c,t,m,n):
  d=c
  for i in range(n):
    d=d*(1+t/100)-m
  return d
```

```
>>> from EMPRUNT import *
>>> u(30000,5/12,1100,24)
5443.727479351415
```

Fonction `calculM`

2°) On initialise la variable `m` qui représente la mensualité recherchée à `c/n` (qui correspond à la mensualité d'un prêt à taux 0%). On pouvait aussi initialiser `m` à 0 mais cela va prendre plus de temps à la fonction pour trouver `m`.

Tant que le capital restant dû à la fin des `n` mois est positif on incrémente `m` de 1.

A la fin de cette boucle on renvoie `m` qui va correspondre à la plus petite mensualité pour laquelle le capital restant dû est négatif. On pourrait aussi renvoyer `m-1` la plus grande mensualité pour laquelle le capital restant dû est positif.

Application : Calculer la mensualité d'un emprunt de 30.000€ sur 24 mois au taux annuel de 5%.

On trouve 1.317€ qu'on peut vérifier en calculant le capital restant dû pour une mensualité de 1.316€ et 1.317€.

Remarque 1 : On pourrait améliorer le script de cette fonction pour obtenir une valeur approchée de la mensualité au centime près en procédant par approximations successives : à l'euro près comme nous venons de le faire, puis au dixième d'euro près (en initialisant `d` à la mensualité obtenue à l'euro près précédente `-1`) et en incrémentant `m` de 0,1 à chaque tour de boucle et recommencer avec un incrément pour `d` de 0,01€.

Remarque 2 : Si on dispose de la somme des termes d'une suite géométrique ou du raisonnement par récurrence on démontre que

$$m = c \times \frac{\frac{t}{100}}{1-\left(1+\frac{t}{100}\right)^{-n}}$$

ÉDITEUR : EMPRUNT
LIGNE DU SCRIPT 0016

```
def calculM(c,t,n):
  m=c/n
  while u(c,t,m,n)>0:
    m=m+1
  return m
```

PYTHON SHELL

```
>>> calculM(30000,5/12,24)
1317.0
>>> u(30000,5/12,1317,24)
-21.61727652626814
>>> u(30000,5/12,1316,24)
3.568644007726334
```

Calcul d'une mensualité d'un prêt

Fonction `cout`

Grâce à la fonction `calculM` qui permet de calculer la mensualité, le coût est rapidement obtenu en faisant `n*m` (qui correspond à ce que l'emprunteur a remboursé à la fin des `n` mois) moins `c` le capital emprunté.

```
def cout(c,t,n):
  m=calculM(c,t,n)
  return m*n-c
```

```
>>> cout(30000,5/12,24)
1608.0
```

Fonction `pourcent`

Le pourcentage recherché étant `cout/c*100` la fonction s'écrit facilement :

```
def pourcent(c,t,n):
  a=cout(c,t,n)
  return a/c*100
```

Application : Quel est le pourcentage du coût du crédit par rapport au capital emprunté pour le crédit suivant :

30.000€ à 5% annuel sur 24 mois.

On trouve 5,36%.

Qu'en est-il si on emprunte sur 60 mois ?

On trouve 13,4%.

```
>>> pourcent(30000,5/12,24)
5.36
>>> pourcent(30000,5/12,60)
13.4
```

Question subsidiaire

On calcule tout d'abord le coût avec un taux mensuel de $t = \frac{3}{12}\%$ et on obtient 82.800€.

Le taux mensuel équivalent à un taux annuel de 5% est

$$t = \left(\left(1 + \frac{5}{100}\right)^{\frac{1}{12}} - 1\right) \times 100$$

Ce qui donne $t \approx 0{,}4074\%$ (contre $\frac{5}{12} = 0{,}4167\%$)

Ainsi le coût de l'emprunt pour ce taux est 81.600€.

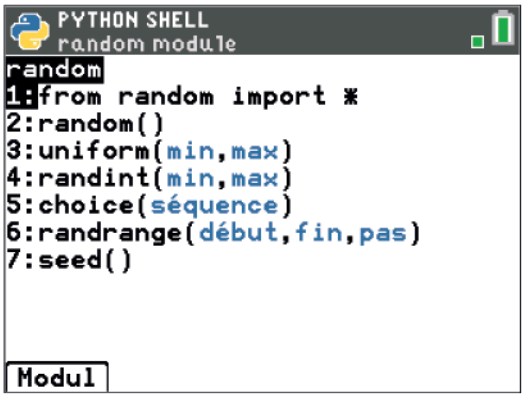
La banque a donc gagné 1.200€ avec ce « petit détail technique ».

```
PYTHON SHELL
>>>
>>> # L'exécution de EMPRUNT
>>>
>>> # Shell Reinitialized
>>> from EMPRUNT import *
>>> cout(250000,3/12,240)
82800.00000000001
>>> t=((1+3/100)**(1/12)-1)*100
>>> cout(250000,t,240)
81600.0
>>>
Fns... | a A # | Outils | éditer | Script
```

Bibliothèque Random, médiane, quartiles…

Bibliothèque RANDOM

On a accès à la bibliothèque RANDOM en appuyant sur **Fns...** `Modul` puis `random`.



Voici un résumé des 7 instructions disponibles :

`from random import *` permet de mettre en mémoire les instructions de cette bibliothèque afin de pouvoir les utiliser.

`random()` renvoie un `float` appartenant à l'intervalle $[0;1]$. Cela permet de simuler la loi uniforme sur $[0,1]$.

`uniform(min,max)` permet de simuler la loi uniforme sur [`min`,`max`].

`randint(min,max)` renvoie un entier aléatoire compris entre `min` et `max` en simulant un tirage équiprobable.

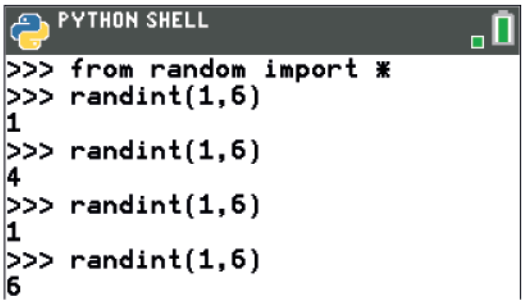
`choice` choisit de façon équiprobable un élément dans une liste ou une chaîne de caractères.

`randrange(début,fin,pas)` choisit de façon équiprobable un élément dans range (`début`,`fin`,`pas`).

`seed(entier)`. C'est à partir d'une graine (seed) qu'est engendrée une suite de nombres pseudo-aléatoires. Définir un seed c'est donc définir cette suite de nombres. Afin d'obtenir des tirages différents, on prend souvent comme argument l'heure de la machine en millisecondes.

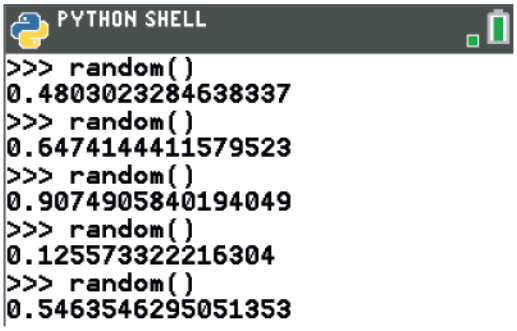
randint

On peut utiliser cette instruction, par exemple, pour simuler un lancer de dé à 6 faces :



random

`random` permet de simuler la loi uniforme sur $[0,1]$:



Bibliothèque Random, médiane, quartiles…

uniform

Dans un cadre plus général, `uniform` va permettre de simuler une loi uniforme sur un intervalle $[a, b]$, par exemple sur $[5; 10]$:

```
PYTHON SHELL
>>> uniform(5,10)
6.105354418939879
>>> uniform(5,10)
8.69246292524293
>>> uniform(5,10)
9.336148744726751
>>> uniform(5,10)
8.714289669842675
>>> uniform(5,10)
8.278188485752734
```

choice

Cette instruction peut prendre en argument une liste ou une chaîne de caractères. Elle permet par exemple :

- Choisir un nombre de façon équiprobable parmi les nombres 8 ; 10 et 15 :

```
PYTHON SHELL
>>> choice([8,10,15])
8
>>> choice([8,10,15])
15
>>> choice([8,10,15])
15
>>> choice([8,10,15])
10
```

- Choisir une lettre de façon équiprobable parmi les lettres du mot « abcde » :

```
PYTHON SHELL
>>> choice("abcde")
'd'
>>> choice("abcde")
'a'
>>> choice("abcde")
'a'
>>> choice("abcde")
'e'
>>> choice("abcde")
'b'
```

- Choisir un mot de façon équiprobable parmi les mots « vin », « jambon », « fromage », « pain » :

```
PYTHON SHELL
>>> choice(["vin","jambon","from
age","pain"])
'fromage'
>>> choice(["vin","jambon","from
age","pain"])
'vin'
>>> choice(["vin","jambon","from
age","pain"])
'vin'
```

Remarque : On peut mélanger nombres et chaînes de caractères dans la liste.

Bibliothèque Random, médiane, quartiles…

Il est possible de faire un choix dans une liste de nombres un peu plus sophistiquée :

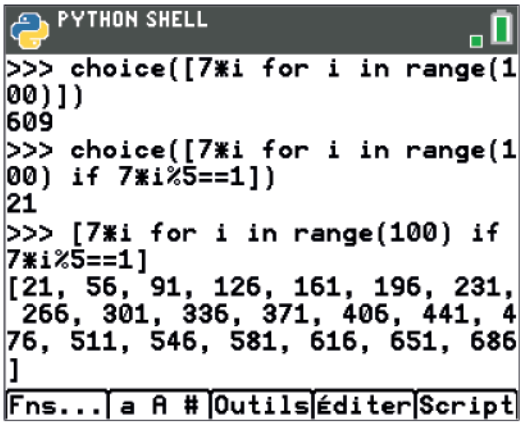
Pour choisir de façon équiprobable un multiple de 7 inférieurs à 700 on écrira :

`choice([7*i for i in range(100)])`

Si en plus on veut imposer à ce multiple de 7 d'avoir un reste de 1 lors de sa division euclidienne par 5 il faut écrire :

`choice([7*i for i in range(100) if 7*i%5==1])`

On peut aussi afficher la liste des multiples de 7 inférieurs à 700 dont le reste de la division par 5 vaut 1 :



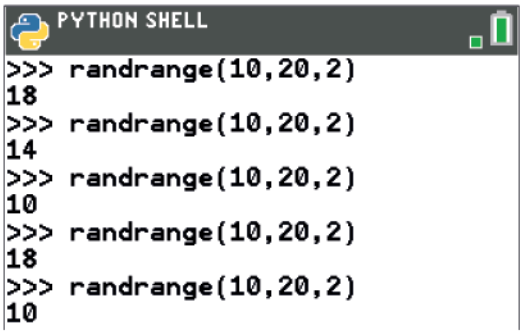
randrange

`randrange` est la version plus détaillée de `randint`. En effet cette instruction permet de choisir de façon équiprobable un entier dans une liste engendrée par l'instruction `range` (très utilisée dans les boucles `for`) :

`range(10,20,2)` correspond aux nombres de 10 (au sens large) à 20 (au sens strict) avec un pas de 2, soit :

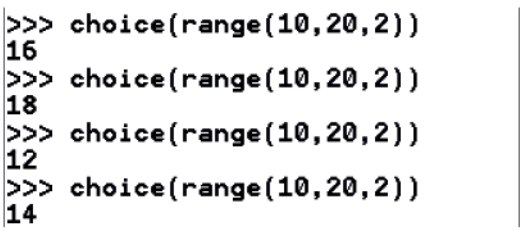
10,12,14,16,18.

`randrange(10,20,2)` va donc choisir équiprobablement un de ces nombres :



Remarque : Il y a équivalence entre les deux instructions suivantes :

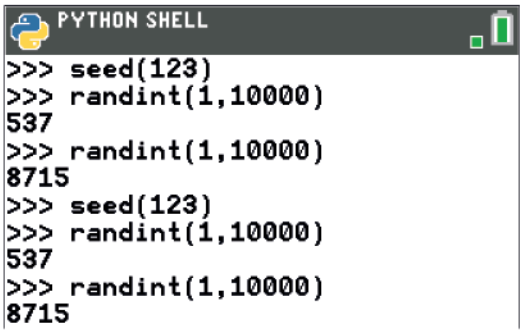
`randrange(10,20,2)` et `choice(range(10,20,2))` :



seed

Les nombres aléatoires engendrés à l'aide des instructions précédentes sont basés sur un générateur de type Mersenne Twister. On parle d'ailleurs de nombres pseudo-aléatoires… Il est très répandu dans les différents langages informatiques, même s'il est à proscrire quand on veut engendrer des clefs aléatoires en cryptographie (une étude plus fine du fonctionnement du générateur de Mersenne Twister permet de deviner les nombres aléatoires si on connait le seed ou bien une série de valeurs obtenues par cet algorithme).

Le seed qui se traduit par graine en français, représente le terme initial d'une suite qui est donc complètement déterminée à partir de ce premier terme. Ainsi pour un seed donné, les simulations aléatoires seront les mêmes :



Bibliothèque Random, médiane, quartiles…

Dans un script STAT

1°) Ecrire une fonction alea qui prend $n \in \mathbb{N}^*$ en paramètre et qui renvoie une liste de `n` entiers aléatoires compris entre 0 (au sens large) et 1000 (au sens strict).

```
>>> alea(10)
[236, 848, 627, 87, 736, 701, 84
9, 503, 975, 80]
```

2°) Ecrire une fonction `etendue` qui prend comme argument une liste et qui renvoie l'étendue des valeurs de cette liste (on pourra utiliser les fonctions `min` et `max` de Python)

```
>>> a
[236, 848, 627, 87, 736, 701, 84
9, 503, 975, 80]
>>> etendue(a)
895
```

3°) Ecrire une fonction `med` qui prend comme argument une liste et qui renvoie la médiane des valeurs de cette liste.

Rappel : Dans une liste de n valeurs ordonnées (ordre croissant),

si n est pair la médiane et la demi-somme des valeurs n° : $\frac{n}{2}$ et $\frac{n}{2}+1$

si n est impair la médiane est la valeur n° : $\frac{n+1}{2}$

Attention : La première valeur de la liste est liste[0] (et non pas liste[1])…

```
>>> med([1,2])
1.5
>>> med([1,2,3])
2
>>> med([1,2,3,5])
2.5
>>> med([1,2,3,5,10])
3
```

4°) Ecrire deux fonctions `q1` et `q3` qui prennent comme argument une liste et qui renvoient respectivement le 1er et le 3ème quartile des valeurs de cette liste.

Rappel : Dans une liste de n valeurs ordonnées (ordre croissant) , si n est divisible par 4 :

Q_1 est la valeur n° $\frac{n}{4}$ et Q_3 est la valeur n° $\frac{3n}{4}$

Si n n'est pas divisible par 4 :

Q_1 est la valeur n°p+1 avec p le quotient de la division de n par 4.

Q_3 est la valeur n°p+1 avec p le quotient de la division de 3n par 4.

```
PYTHON SHELL
>>>
>>> from STAT import *
>>> q1([6,1,9,5])
1
>>> q1([6,1,9,5,10])
5
>>> q3([6,1,9,5])
6
>>> q3([6,1,9,5,10])
9
```

5°) Ecrire une fonction `ecartinter` qui prend comme argument une liste et qui renvoie l'écart interquartile des valeurs de cette liste.

```
>>> ecratinter([7,9,1,10,13])
3
```

6°) Ecrire une fonction `moustache` qui prend comme argument une liste et qui renvoie le tuple : `min,q1,med,q3,max`.

```
>>> moustache([7,9,1,10,13])
(1, 7, 9, 10, 13)
```

Bibliothèque Random, médiane, quartiles…

Fonction `alea`

1°) Ecrire une fonction alea qui prend `n`$\in \mathbb{N}^*$ en paramètre et qui renvoie une liste de n entiers aléatoires compris entre 0 et 1000.

A partir de la liste vide `liste`, on va répéter `n` fois l'instruction suivante :

Ajouter un entier aléatoire compris entre 0 et 1000 à la liste `liste`.

La fonction renvoie `liste` à la fin de la boucle :

```
ÉDITEUR : STAT
LIGNE DU SCRIPT 0009
from random import *
def alea(n):
  liste=[]
  for i in range(n):
    liste.append(randint(0,1000)
      )
  return liste
```

On peut noter au passage une façon plus concise que Python permet d'utiliser pour générer une telle liste :

On recommandera cependant aux débutants le script précédent.

```
ÉDITEUR : STAT
LIGNE DU SCRIPT 0016
def alea(n):
  liste=[randint(0,1000) for i i
      n range(n)]
  return liste
```

Fonction `etendue`

2°) L'étendue étant la différence entre la plus grande et la plus petite valeur de la liste, il suffit d'utiliser les instructions `min` et `max` de Python.

```
ÉDITEUR : STAT
LIGNE DU SCRIPT 0020
def etendue(liste):
  return max(liste)-min(liste)
```

Fonction `med`

3°) On calcule la taille de la liste grâce à l'instruction `len`.

Puis on utilise la méthode `sort` qui permet d'ordonner notre liste dans l'ordre croissant.

On considère alors 2 cas pour suivre la définition de la médiane:

Si n est divisible par 2 (c'est-à-dire si `n%2==0`) ou non.

Il y a **deux pièges** : Il faut bien faire attention au décalage : le terme n° $\frac{n}{2}$ est liste[n//2-1] et il ne faut pas oublier d'écrire n//2 et non pas n/2.

En effet, n//2 renvoie le quotient de la division de n par 2 qui est un entier (donc de type `int`) et n/2 divise n par 2 et renvoie un `float`.

Le terme attendu entre crochets ne peut pas être un `float`... il doit obligatoirement être un entier.
Même si 4 est pair 4/2 est de type float :

```
ÉDITEUR : STAT
LIGNE DU SCRIPT 0014
def med(liste):
  n=len(liste)
  liste.sort()
  if n%2==0:
    m=(liste[n//2-1]+liste[n//2]
      )/2
  else:
    m=liste[n//2]
  return m
```

```
PYTHON SHELL
>>> 4/2
2.0
```

On constate bien l'erreur si on utilise le symbole / et non pas //

Le message est clair : L'indice d'une liste doit être un entier et non pas un `float`.

```
PYTHON SHELL
>>> liste=[7,6,0,1]
>>> n=4
>>> liste[n/2]
Traceback (most recent call last
):
  File "<stdin>", line 1, in <mo
dule>
TypeError: list indices must be
integers, not float
```

Bibliothèque Random, médiane, quartiles…

Fonctions `q1` et `q3`

4°) Cette question est très semblable à la question 3°).

ÉDITEUR : STAT
LIGNE DU SCRIPT 0033

```
def q1(liste):
    n=len(liste)
    liste.sort()
    if n%4==0:
        q=liste[n//4-1]
    else:
        q=liste[n//4]
    return q
```

ÉDITEUR : STAT
LIGNE DU SCRIPT 0043

```
def q3(liste):
    n=len(liste)
    liste.sort()
    if n%4==0:
        q=liste[3*n//4-1]
    else:
        q=liste[3*n//4]
    return q
```

Fonctions `ecartinter`

5°) Il suffit d'utiliser les fonction `q1` et `q3` définies précédemment :

ÉDITEUR : STAT
LIGNE DU SCRIPT 0053

```
def ecratinter(liste):
    e=q3(liste)-q1(liste)
    return e
```

Fonctions `moustache`

6°) On utilise les fonctions `min`, `max` de Python puis les fonctions `med`, `q1` et `q3` que nous avons créées dans les questions précédentes.

Dans le `return`, on n'est pas obligé d'écrire le tuple avec des parenthèses.

```
def moustache(liste):
    mi=min(liste)
    ma=max(liste)
    m=med(liste)
    qu1=q1(liste)
    qu3=q3(liste)
    return mi,qu1,m,qu3,ma
```

Fns... | a A # | Outils | Exéc | Script

En attendant 1, 2, 3 sigma ...

Compétences visées

- **chercher**, expérimenter – en particulier à l'aide d'outils logiciels ;
- **calculer**, appliquer des techniques et mettre en œuvre des algorithmes ;
- **communiquer** un résultat par oral ou par écrit, expliquer une démarche.

Ces compétences sont mises en œuvre au regard de l'extrait du programme de 2nde GT ci-dessous :
« Pour des données réelles ou issues d'une simulation, lire et comprendre une fonction écrite en Python renvoyant la moyenne m, l'écart type s, et la proportion d'éléments appartenant à $[m - 2s\ ,\ m + 2s]$. »

Situation déclenchante

Pour bien comprendre les formules statistiques et prendre un peu de recul ...

Comment distinguer dans une série numérique de grande taille représentant des données (a priori réelles) les valeurs aberrantes des valeurs extrêmes ?

Problématique

Créer des fonctions en python permettant de :

- calculer la moyenne d'une série de nombres ;
- calculer l'écart type d'une série de nombres ;
- calculer la proportion de valeurs dont la distance à la moyenne est inférieure à deux écarts types.

Fiche méthode

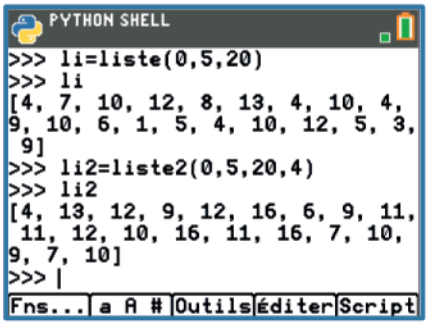
Proposition de résolution

Pour éviter d'avoir à saisir des listes de nombres à la main, on crée au préalable une fonction permettant de générer une liste de *n* nombres aléatoires entiers à partir de la fonction **randint**. Cependant, une liste obtenue directement à partir **randint** donnerait un résultat peu intéressant comme cela est montré à la fin de la fiche. C'est pourquoi, pour rendre le travail pertinent, on est amené à créer des listes de nombres de manière plus fine, comme décrit dans les étapes de résolution qui suivent.

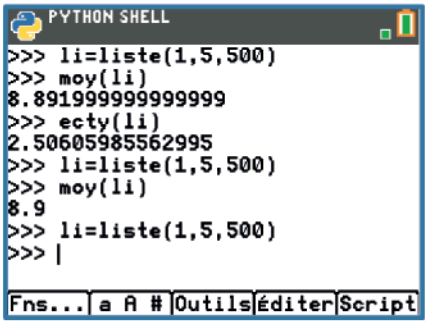
- Sur la copie d'écran ci-contre, l'instruction **liste**(0,5,20) renvoie une liste de 20 nombres entiers compris entre 3x0 et 3x5 ; cette liste est stockée dans la variable `li`.

  `li` est ensuite affichée (c'est à déconseiller par la suite lorsqu'on aura beaucoup de valeurs).

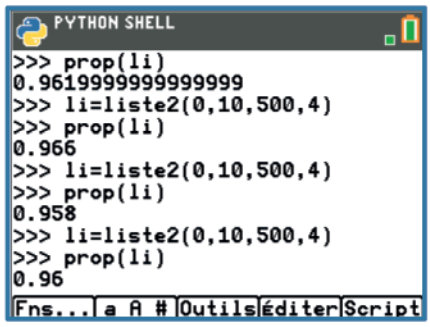
  liste2(0,5,20,4) génère quant à elle une liste de 20 nombres entiers avec un mode plus « élaboré » décrit plus tard.



- Écrire la fonction **moy** et la fonction **ecty** qui prennent toutes deux comme paramètre une liste et qui renvoient respectivement la valeur moyenne et l'écart type de des valeurs de la liste



- Écrire une fonction **prop** qui prend en paramètre une liste et qui renvoie la proportion de valeurs de la liste qui sont à une distance inférieure à deux écarts types de la moyenne.

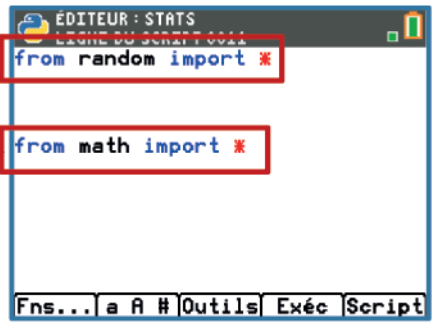


- Pour effectuer plusieurs essais, il est commode d'utiliser la flèche directionnelle vers le haut : un appui réécrit la dernière instruction utilisée (deux appuis donne l'instruction écrite deux lignes plus haut et ainsi de suite) ; en appuyant sur `entrer`, on exécute à nouveau l'instruction.

Remarque

Importation en préambule du code de la bibliothèque « random » par « **from random import *** » pour pouvoir utiliser la fonction **randint**

Importation de la bibliothèque « math » par « **from math import *** » pour pouvoir utiliser les fonctions **sqrt** (racine carrée) et **fabs** (valeur absolue)



Pour profiter de tutoriels vidéos, Flasher le QRCode ou cliquer dessus

Fiche méthode

Etapes de résolution

L'instruction **liste**(a,b,n) génère une liste de n valeurs entières comprises entre 3a et 3b.

```
ÉDITEUR : STATS
LIGNE DU SCRIPT 0010
from random import *
from math import *

def liste(a,b,n):
  li=[]
  for i in range(n):
    li.append(randint(a,b)+randi
      nt(a,b)+randint(a,b))
  return li

Fns... a A # Outils Exéc Script
```

`li = [ ]` définit une variable de type list et l'initialise en une liste vide.

`li.append()` permet d'ajouter à la liste `li` la valeur passée en paramètre : ici, on ajoute `randint(a,b)+ randint(a,b) +randint(a,b)` à la liste `li`.

La fonction **moy** a comme paramètre une liste et renvoie la moyenne des valeurs contenues dans la liste.

```
ÉDITEUR : STATS
LIGNE DU SCRIPT 0010
def moy(liste):
  return sum(liste)/len(liste)

def ecty(liste):
  v=0
  n=len(liste)
  m=moy(liste)
  for x in liste:
    v=v+(x-m)**2
  v=v/n
  return sqrt(v)

Fns… a A # Outils Exéc Script
```

sum est une fonction native qui prend une liste en paramètre et renvoie la somme des valeurs contenues dans la liste.

len est une fonction native qui prend une liste en paramètre et renvoie le nombre de valeurs de la liste.

La fonction **ecty** a pour paramètre une liste et renvoie l'écart type de la série de valeurs contenues dans la liste.

La fonction **ecty** utilise la fonction **moy** précédemment définie, ce qui rend son écriture proche de la formule du cours.

`for x in liste` permet à `x` de parcourir les valeurs contenue dans la liste nommée `liste`.

Il faut comprendre le rôle joué par la variable `v` (v pour variance) qui ajoute au fur et à mesure que l'on parcourt la liste les valeurs notées usuellement $(x_i - m)^2$.

```
ÉDITEUR : STATS
LIGNE DU SCRIPT 0030
def prop(liste):
  c=0
  n=len(liste)
  m=moy(liste)
  s=ecty(liste)
  for x in liste:
    if fabs(x-m)<=2*s:
      c=c+1
  return c/n

Fns… a A # Outils Exéc Script
```

La fonction **prop** a comme paramètre une liste et renvoie la proportion des valeurs de la liste situées à une distance supérieure à deux écarts type de la moyenne.

Pour des raisons de lisibilité, on a choisi de noter : `n=len(liste)`, `m=moy(liste)`, `s=ecty(liste)`, ce qui permet d'avoir un test écrit de manière proche de l'écriture naturelle :

« si | valeur – *m* | <= 2*s*, alors ajouter 1 à *c* »

Pour profiter de tutoriels vidéos, Flasher le QRCode ou cliquer dessus

Fiche méthode

Retour à la situation déclenchante

Lorsqu'une liste est issue d'une situation réelle (valeurs données par un capteur, valeurs issues d'un sondage, etc.), il se peut que certaines soient erronées.

S'il est possible de remonter à la source, on pourra éventuellement corriger la valeur (erreur de saisie par exemple), sinon, il sera pertinent de l'éliminer.

Comment examiner ces valeurs ?

Il faut les traiter au cas par cas, et le travail précédent va permettre de le faire.

En effet, on examinera les valeurs inférieures à $m-2s$ et celles supérieures à $m+2s$ et on fera fonctionner son esprit critique ! Pour distinguer des valeurs acceptables qui seraient extrêmes des valeurs aberrantes.

Si par exemple un sondage traite de la taille des individus (en cm), la valeur 1.78 aura sans doute été saisie par quelqu'un répondant en m. On pourra la modifier.

liste(50,70,100) génère 100 nombres entiers compris entre 150 et 210 : ils peuvent simuler un échantillon de tailles d'adultes. On observe les valeurs extrêmes, hors de l'intervalle $[m-2s;m+2s]$.

```
ÉDITEUR : STATS
LIGNE DU SCRIPT 0031
def test(liste):
  examine=[]
  n=len(liste)
  m=moy(liste)
  s=ecty(liste)
  for x in liste:
    if fabs(x-m)>2*s:
      examine.append(x)
  return examine
Fns… | a A # | Outils | Exéc | Script
```

```
PYTHON SHELL
>>>
>>> li=liste(50,70,100)
>>> li.append(1.78)
>>> test(li)
[1.78]
>>> li=liste(50,70,100)
>>> test(li)
[158, 157, 202, 203, 158]
>>>
Fns... | a A # | Outils | éditer | Script
```

Remarque

On peut améliorer le processus permettant de générer une liste à partir de la fonction **randint** par la fonction ci-contre qui va ajouter r fois des nombres aléatoires entiers compris entre les entiers a et b.

La fonction **liste2**(a,b,n,r) génère une liste de n valeurs entières comprises entre a×r et b×r en ajoutant r fois un nombre issu de randint(a,b).

```
ÉDITEUR : STATS
LIGNE DU SCRIPT 0019
def liste2(a,b,n,r):
  li=[]
  for i in range(n):
    val=0
    for i in range(r):
      val=val+randint(a,b)
    li.append(val)
  return li
Fns... | a A # | Outils | Exéc | Script
```

Pour profiter de tutoriels vidéos, Flasher le QRCode ou cliquer dessus

Fiche méthode

Un peu de théorie

Pourquoi avoir créé des fonctions pour générer des listes de valeurs ?

On aurait pu écrire le code ci-contre, générant une liste directement à partir de **randint** : on obtient n nombres entiers répartis 'uniformément' entre a et b.
Mais alors, toutes les valeurs appartiennent à l'intervalle $[m-2s;m+2s]$. On va le montrer ici.

```
EDITEUR : STATS
LIGNE DU SCRIPT 0029

def liste3(a,b,n):
  li=[]
  for i in range(n):
    li.append(randint(a,b))
  return li

Fns... | a A # | Outils | Exéc | Script
```

Si X est une variable aléatoire qui suit la loi de probabilité suivante :

a_i	1	2	…	n
$P(X=a_i)$	$\frac{1}{n}$	$\frac{1}{n}$		$\frac{1}{n}$

Son espérance mathématique est : $E(X)=\frac{1}{n}\sum_{k=1}^{n}k=\frac{1}{n}\cdot\frac{n(n+1)}{2}=\frac{n+1}{2}$
La variance se calcule par :

$$V(X)=\frac{1}{n}\sum_{k=1}^{n}\left(k-\frac{n+1}{2}\right)^2=\frac{1}{n}\left(\sum_{k=1}^{n}k^2-(n+1)\sum_{k=1}^{n}k+\sum_{k=1}^{n}\frac{(n+1)^2}{4}\right)$$

$$V(X)=\frac{1}{n}\left(\frac{n(n+1)(2n+1)}{6}-2(n+1)\frac{n(n+1)}{4}+n\frac{(n+1)^2}{4}\right)=\frac{n^2-1}{12}$$

Au final, l'écart type est égal à : $\sigma(X)=\sqrt{V(X)}=\sqrt{\frac{n^2-1}{12}}$

```
PYTHON SHELL
>>> prop(li)
0.0
>>> li=liste3(1,10,5)
>>> prop(li)
1.0
>>> li=liste3(1,10,500)
>>> prop(li)
1.0
>>> li=liste3(1,10,500)
>>> prop(li)
1.0
Fns... | a A # | Outils | Éditer | Script
```

On peut confronter ces résultats à la moyenne et l'écart type obtenu à partir de **liste3**(1,10).
Théoriquement, la moyenne vaut 5,5 et l'écart type $\sqrt{\frac{99}{12}}\approx 2,87$

```
PYTHON SHELL
>>> li=liste3(1,10,500)
>>> moy(li)
5.328
>>> ecty(li)
2.8327400163093
>>> li=liste3(1,10,500)
>>> moy(li)
5.546
>>> ecty(li)
2.874697201445743
>>> |
Fns... | a A # | Outils | Éditer | Script
```

Ainsi, une telle liste ne présente pas d'intérêt quant à la proportion d'éléments distants de moins de deux écarts-types de la moyenne.
En effet, pour $n\in\mathbb{N}^*$: $E(X)-2\sigma(X)=\frac{n+1}{2}-2\sqrt{\frac{n^2-1}{12}}\leqslant 1$

$$E(X)+2\sigma(X)=\frac{n+1}{2}+2\sqrt{\frac{n^2-1}{12}}\geqslant n$$

Ce qui montre que toutes les valeurs comprises entre 1 et n sont dans l'intervalle $[E(X)-2\sigma(X)\,;\,E(X)+2\sigma(X)]$

```
PYTHON SHELL
>>> l1=liste(1,10,500)
>>> prop(l1)
0.9719999999999999
>>> l1=liste(1,10,500)
>>> prop(l1)
0.952
>>> l1=liste(1,10,500)
>>> prop(l1)
0.968
>>> |
Fns... | a A # | Outils | Éditer | Script
```

C'est pourquoi on a créé les fonctions **liste** et **liste2** qui génèrent des listes qui n'auront pas systématiquement toutes leurs valeurs comprises entre $m-2s$ et $m+2s$.

Pour profiter de tutoriels vidéos, Flasher le QRCode ou cliquer dessus

Moyenne, écart-type,…

Dans cet exercice, on va calculer la moyenne, l'écart-type d'une série statistique, ainsi que la proportion d'une partie de cette série.

Dans un script STAT2

1°) Ecrire une fonction `moyenne` qui prend comme argument une liste et qui renvoie la moyenne des valeurs de cette liste.

```
PYTHON SHELL
>>> moyenne([8,10,20,9,8])
11.0
```

2°) Ecrire deux fonctions `variance` et `ecart` qui prennent comme argument une liste et qui renvoient respectivement la variance et l'écart type des valeurs de cette liste.

```
PYTHON SHELL
>>> ecart([8,10,20,9,8])
4.560701700396553
```

```
PYTHON SHELL
>>> variance([8,10,20,9,8])
20.80000000000001
```

3°) Ecrire une fonction `pop` qui prend comme arguments une liste et deux réels a et b et qui renvoie le nombre de valeurs de la liste qui sont comprises entre a et b.

```
PYTHON SHELL
>>> pop([8,10,20,9,8],8,10)
4
>>> pop([8,10,20,9,8],11,15)
0
>>> pop([8,10,20,9,8],9,15)
2
```

4°) Ecrire une fonction `sdd` qui prend comme argument un entier naturel $n \in \mathbb{N}^*$ et qui renvoie une liste de n termes correspond chacun à la somme des faces d'un lancer aléatoire de 2 dés à 6 faces.

Application : Quelle est la proportion de valeurs comprises entre $\bar{x} - 2\sigma$ et $\bar{x} + 2\sigma$ pour une simulation de 500 lancers ?

```
PYTHON SHELL
>>> sdd(10)
[9, 5, 5, 7, 4, 4, 2, 7, 7, 9]
>>> sdd(10)
[6, 6, 5, 9, 2, 10, 12, 5, 9, 11
]
>>> sdd(10)
[9, 2, 9, 10, 7, 8, 9, 5, 5, 6]
>>> sdd(15)
[6, 5, 2, 11, 5, 8, 8, 7, 11, 7,
 11, 6, 7, 10, 11]
```

Moyenne, écart-type,…

Fonction `moyenne`

1°) Pour calculer la moyenne on va utiliser la fonction Python `sum` pour obtenir la somme `s` de toutes les valeurs et `len` pour obtenir `n` l'effectif total. Ainsi la moyenne sera `s/n`.

```
ÉDITEUR : STAT2
LIGNE DU SCRIPT 0009
from math import *
from random import *
def moyenne(liste):
  s=sum(liste)
  n=len(liste)
  return s/n
```

Fonction `variance`

2°) On propose deux façons de coder cette fonction :

- On calcule la moyenne `m`, l'effectif total `n` et `s` la somme des carrés des valeurs de la série. Puis on renvoie `s/n-m**2`. Cette méthode est basée sur la relation $V(X) = E(X^2) - E(X)^2$. On remarquera qu'on a utilisé une façon très concise pour définir `s`.

```
ÉDITEUR : STAT2
LIGNE DU SCRIPT 0016

def variance(liste):
  m=moyenne(liste)
  n=len(liste)
  s=sum([i**2 for i in liste])
  return s/n-m**2
```

- La deuxième fonction est appelée `variance2` et elle commence par calculer la somme suivante (avec les notations habituelles) :

$$\sum_{i=1}^{n}(x_i - m)^2$$

```
ÉDITEUR : STAT2
LIGNE DU SCRIPT 0023

def variance2(liste):
  m=moyenne(liste)
  n=len(liste)
  s=0
  for i in liste:
    s=s+(i-m)**2
  return s/n
```

On remarquera que dans la boucle `for i` prend toutes les valeurs des éléments de la liste (qui contient les valeurs de notre série statistique).

On divise `s` par `n` à la fin pour obtenir la formule de la variance.

Pas de difficulté particulière pour la fonction `ecart`.

```
ÉDITEUR : STAT2
LIGNE DU SCRIPT 0031

def ecart(liste):
  v=variance(liste)
  return sqrt(v)
```

Fonction `pop`

3°) Il s'agit ici de dénombrer tous les éléments de la liste dont les valeurs sont comprises entre a et b.

On parcourt tous les éléments de la liste à l'aide d'une boucle `for` et un simple test permet de déterminer s'il faut ajouter 1 à notre compteur `s`.

```
ÉDITEUR : STAT2
LIGNE DU SCRIPT 0034

def pop(liste,a,b):
  s=0
  for i in liste:
    if a<=i and i<=b:
      s=s+1
  return s
```

On propose une fonction `pop2` qui permet de créer une liste à partir des éléments de la liste `liste` en imposant une condition. L'effectif de cette nouvelle liste est la valeur recherchée.

```
ÉDITEUR : STAT2
LIGNE DU SCRIPT 0040

def pop2(liste,a,b):
  l=[i for i in liste if a<=i an
      d i<=b]
  return len(l)
```

Moyenne, écart-type,…

Fonction `sdd`

4°) A partir d'une liste vide, on répète `n` fois l'ajout de la somme de deux dés à notre liste.

ÉDITEUR : STAT2
LIGNE DU SCRIPT 0044

```
def sdd(n):
  liste=[]
  for i in range(n):
    liste.append(randint(1,6)+ra
      ndint(1,6))
  return liste
```

On propose aussi une fonction `sdd2` dont la syntaxe est plus concise.

ÉDITEUR : STAT2
LIGNE DU SCRIPT 0051

```
def sdd2(n):
  liste=[randint(1,6)+randint(1,
      6) for i in range(n)]
  return liste
```

Application

Quelle est la proportion de valeurs comprises entre $\bar{x} - 2\sigma$ et $\bar{x} + 2\sigma$ pour une simulation de 500 lancers ?

On simule le lancer de 500 fois deux dés. La liste des résultats est stockée dans `l` en utilisant la fonction `sdd`.

On calcule la moyenne `m` et l'écart type `sig`.

Puis grâce à la fonction `pop` on trouve le nombre de valeurs de la liste `l` comprises entre `m-2*sig` et `m+2*sig`.

Ce dernier résultat est divisé par l'effectif de la série qui est de 500.

On obtient ici une fréquence de 0,96.

PYTHON SHELL

```
>>> l=sdd(500)
>>> m=moyenne(l)
>>> sig=ecart(l)
>>> m-2*sig
2.301124231183145
>>> m+2*sig
11.71087576881686
>>> pop(l,m-2*sig,m+2*sig)
480
>>> 480/500
0.96
>>>
```

Fns... | a A # | Outils | éditer | Script

Fréquence d'une lettre dans un texte.

On va préparer ici la partie statistique du déchiffrement d'un texte crypté.

A partir d'un texte donné, on va relever les fréquences d'apparition des lettres dans ce texte.

On utilisera deux types de présentation des résultats : sous forme de liste et sous forme de dictionnaire.

Dans un script FREQLET

On considère le texte suivant : `chaine="stage ti python"` (ou tout autre texte, cela peut-être un texte beaucoup plus long téléchargé).

1°) Ecrire une fonction `espace` qui prend comme argument une chaîne de caractères `ch` et qui renvoie le nombre d'espace dans cette chaîne.

```
PYTHON SHELL
>>> from FREQLET import *
>>> espace(chaine)
2
```

2°) Ecrire une fonction `flettre` qui prend comme arguments une chaîne de caractères `chaine` et un caractère `c` et qui renvoie la fréquence d'apparition du caractère `c` dans le texte `chaine`.

```
PYTHON SHELL
>>> flettre(chaine,"a")
0.07692307692307693
>>> flettre(chaine,"t")
0.2307692307692308
>>> flettre(chaine,"z")
0.0
```

3°) Ecrire une fonction `freq` qui prend comme argument une chaîne de caractères `chaine` et qui renvoie la liste des fréquences des lettres "a", "b", ..., "z" dans `chaine`.

```
PYTHON SHELL
>>> freq(chaine)
[0.07692307692307693, 0.0, 0.0,
0.0, 0.07692307692307693, 0.0, 0
.07692307692307693, 0.0769230769
2307693, 0.07692307692307693, 0.
0, 0.0, 0.0, 0.0, 0.076923076923
07693, 0.07692307692307693, 0.07
692307692307693, 0.0, 0.0, 0.076
92307692307693, 0.23076923076923
08, 0.0, 0.0, 0.0, 0.0, 0.076923
07692307693, 0.0]
```

4°) Ecrire une fonction `pourcent` qui prend comme argument une chaîne de caractères `chaine` et qui renvoie le dictionnaire des pourcentages d'apparition des lettres "a", "b", ..., "z" dans `chaine` (on écrira seulement les lettres dont la fréquence associée est non nulle).

```
PYTHON SHELL
>>> pourcent(chaine)
{'e': 7.69, 'i': 7.69, 'g': 7.69
, 'h': 7.69, 'a': 7.69, 't': 23.
08, 'o': 7.69, 'y': 7.69, 'p': 7
.69, 's': 7.69, 'n': 7.69}
```

Fréquence d'une lettre dans un texte.

Fonction `espace`

1°) On va initialiser un compteur `s` à 0.

Puis on va parcourir toutes les lettres de `chaine` à l'aide d'une boucle `for`.

A chaque fois qu'une lettre est un espace alors on incrémente le compteur `s` de 1.

Fonction `flettre`

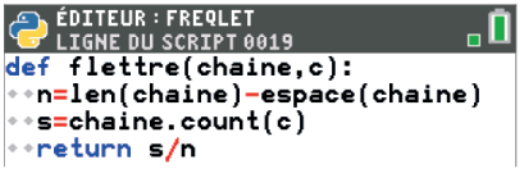
2°) On peut envisager 2 façons d'écrire cette fonction :

- En utilisant les fonctions et méthodes de Python

  `len` permet d'obtenir la taille d'une liste ou le nombre de caractères d'une chaîne (il faudra lui retirer le nombre d'espaces dans le texte).

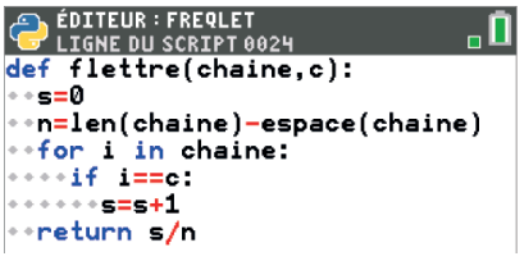
  `chaine.count(c)` renvoie le nombre de fois où la chaîne `c` est présente dans `chaine`.

  Cette façon de faire est très rapide.



- On peut aussi utiliser un compteur `s` et, dans une boucle `for`, parcourir tous les caractères de `chaine` et ajouter 1 au compteur dès que le caractère rencontré est égal à `c`.

  Cette méthode est plus longue mais elle utilise des notions de programmation classiques.



Fréquence d'une lettre dans un texte.

Fonction `freq`

3°) `liste` contiendra la liste des fréquences des lettres a, b, c, …, z dans notre `chaine`.

On va donc initialiser `liste` comme une liste vide. Cette fois on n'utilisera pas `liste=[]` mais `liste=list()` qui sont deux instructions équivalentes.

`alpha` est la chaîne de caractère contenant toutes les lettres de l'alphabet.

`for i,c in enumerate(alpha)` signifie que `c` va parcourir tous les caractères de la chaîne `alpha` et `i` sera l'indice associé.

La première valeur de `c` sera "a" et `i` vaudra 0.

La seconde valeur de `c` sera "b" et `i` vaudra 1, etc…

A l'aide de la fonction `flettre` on calcule à chaque tour de boucle `f` la fréquence d'apparition de la lettre `c` (qui vaut "a" au début puis "b",…) dans le texte `chaine`.

Et enfin on ajoute cette fréquence dans la liste des fréquences qu'on a appelé `liste`.

```
ÉDITEUR : FREQLET
LIGNE DU SCRIPT 0031
def freq(chaine):
  liste=list()
  alpha="abcdefghijklmnopqrstuvw
    xyz"
  for i,c in enumerate(alpha):
    f=flettre(chaine,c)
    liste.append(f)
  return liste

Fns… | a A # | Outils | Exéc | Script
```

Notion de dictionnaire en Python.

Un dictionnaire est un objet de type `dict`.

Un élément du dictionnaire est caractérisé par sa **clef** et sa **valeur** :

La **clef** aussi bien que sa **valeur** associée peut-être une chaîne de caractères ou un nombre.

Cela ressemble un peu aux listes, sauf que la clef peut-être une chaîne de caractères ou même un `float` !

```
PYTHON SHELL
>>> dico=dict()
>>> dico["pomme"]=17
>>> dico["poire"]="comice"
>>> dico[8]="huit"
>>> dico[10]=20
>>> dico
{'poire': 'comice', 8: 'huit', 1
0: 20, 'pomme': 17}
```

Fonction `pourcent`

4°) Cette fonction est sensiblement identique à `freq`.

On a initialisé notre dictionnaire `dico` en écrivant `dico=dict()`

On aurait pu aussi écrire `dico={ }` mais c'était moins parlant et on pouvait confondre avec le type `set` (ensemble).

On a seulement besoin de parcourir tous les caractères de l'alphabet, calculer la fréquence et la multiplier par 100 pour obtenir un pourcentage.

On a utilisé l'instruction `round` pour obtenir 2 décimales de précision.

A la fin on a testé si `f` était non nul afin de l'ajouter dans le dictionnaire `dico` avec une **clef** correspondant à la lettre de l'alphabet et comme **valeur** le pourcentage associé.

```
ÉDITEUR : FREQLET
LIGNE DU SCRIPT 0039
def pourcent(chaine):
  dico=dict()
  alpha="abcdefghijklmnopqrstuvw
    xyz"
  for c in alpha:
    f=flettre(chaine,c)
    f=round(100*f,2)
    if f!=0:
      dico[c]=f
  return dico
```

Lancer de pièce et prise de décision.

Sur 35 lancers de pièce on a obtenu plus de 25 fois pile. Est-ce normal ? (On utilisera un niveau de confiance de 95%)

Dans un script SIMUL1

1°) Ecrire une fonction `pf` qui prend comme argument `n` $\in \mathbb{N}^*$ et qui renvoie une liste de `n` termes représentant `n` lancer d'une pièce.

On représentera face par 0 et pile par 1.

```
PYTHON SHELL
>>>
>>> from SIMUL1 import *
>>> pf(10)
[0, 0, 0, 1, 1, 0, 0, 1, 1, 1]
>>> pf(20)
[1, 0, 0, 1, 1, 1, 0, 1, 1, 0, 1
, 1, 0, 0, 0, 1, 0, 0, 0, 0]
```

2°) Ecrire une fonction `simul` qui prend comme argument les entiers naturels non nuls `n` et `p`. On simulera `n` lancers d'une pièce et la fonction renvoie `True` si on a obtenu p (ou plus) fois pile et `False` sinon.

```
PYTHON SHELL
>>>
>>> from SIMUL1 import *
>>> simul(10,6)
False
>>> simul(10,6)
True
```

3°) Ecrire une fonction `frequence` qui prend comme arguments `q,n,p` des entiers naturels non nuls et qui effectue `q` fois `n` lancers de la pièce et renvoie la fréquence de l'événement « on a obtenu au moins `p` fois pile » lors des `q` simulations.

```
PYTHON SHELL
>>> frequence(100,30,17)
0.31
>>> frequence(1000,30,17)
0.29
>>> frequence(10000,30,17)
0.2936
```

4°) Conclure s'il est normal d'obtenir plus de 25 fois pile lors de 35 lancers d'une pièce.

Lancer de pièce et prise de décision.

Fonction `pf`

1°) On commence par initialiser la liste `liste` en écrivant :

`liste=[ ]`. On aurait pu aussi écrire `liste=list()`.

Dans une boucle `for` qui boucle `n` fois, on ajoute à la liste `liste` un entier aléatoire : 0 pour face et 1 pour pile.

On propose un seconde script plus concis réalisable grâce à la syntaxe Python.

ÉDITEUR : SIMUL1
LIGNE DU SCRIPT 0015

```
def pf(n):
  liste=[]
  for i in range(n):
    liste.append(randint(0,1))
  return liste
```

ÉDITEUR : SIMUL1
LIGNE DU SCRIPT 0008

```
from random import *
def pf(n):
  liste=[randint(0,1) for i in r
      ange(n)]
  return liste
```

Fonction `simul`

2°) A partir de la liste des faces et piles représentés par des 0 et des 1 dans `liste`, on additionne tous les termes de liste, ce qui correspondra au nombre de 1 présent dans la liste soit le nombre de fois où on a obtenu pile.

Si cette somme est supérieure ou égale à `p` on renvoie `True` et `False` sinon.

ÉDITEUR : SIMUL1
LIGNE DU SCRIPT 0021

```
def simul(n,p):
  liste=pf(n)
  if sum(liste)>=p:
    return True
  else:
    return False
```

Fonction `frequence`

3°) On recommence l'expérience précédente q fois. On va compter à chaque fois si on a obtenu plus de p piles ou non.

k est la variable qui correspond à ce nombre. Si simul(n,p) est vrai alors k est incrémenté de 1.

On remarque qu'on a écrit `if simul(n,p)`. On aurait pu aussi écrire `if simul(n,p)==True` de façon équivalente car `simul(n,p)` est un booléen.

On renvoie `k/q` qui correspond à la fréquence recherchée.

ÉDITEUR : SIMUL1
LIGNE DU SCRIPT 0028

```
def frequence(q,n,p):
  k=0
  for i in range(q):
    if simul(n,p):
      k=k+1
  return k/q
```

Conclusion

4°) Conclusion : Il y a environ moins d'un pourcent de chance d'obtenir 25 fois la face pile lors de 35 lancers.

Au niveau de confiance de 95% cela n'est pas normal d'obtenir 25 fois la face pile.

PYTHON SHELL

```
>>> frequence(1000,35,25)
0.008
>>> frequence(1000,35,25)
0.009
>>> frequence(10000,35,25)
0.009299999999999999
>>> frequence(100000,35,25)
0.008710000000000001
>>> frequence(100000,35,25)
0.00808
```

Thème: simulation ; loi des grands nombres

Niveau: 2nde

TI-83 Premium CE

python

Une somme de hasards

Compétences visées

- **modéliser**, faire une simulation, valider ou invalider un modèle ;
- **raisonner**, démontrer, trouver des résultats partiels et les mettre en perspective ;
- **calculer**, appliquer des techniques et mettre en œuvre des algorithmes.

« En classe de seconde, on formalise la notion de loi (ou distribution) de probabilité dans le cas fini en s'appuyant sur le langage des ensembles et on précise les premiers éléments de calcul de probabilités. On insiste sur le fait qu'une loi de probabilité (par exemple une équiprobabilité) est une hypothèse du modèle choisi, et ne se démontre pas. Le choix du modèle peut résulter d'hypothèses implicites d'équiprobabilité (par exemple, lancers de pièces ou dés équilibrés, tirage au hasard dans une population) qu'il est recommandable d'expliciter : **il peut aussi résulter d'une application d'une version vulgarisée de la loi des grands nombres, où un modèle est construit à partir de fréquences observées pour un phénomène réel** (par exemple : lancer de punaise, sexe d'un enfant à la naissance). Dans tous les cas, on distingue nettement le modèle probabiliste abstrait et la situation réelle. »

Il est précisé dans cet extrait de programme pour la classe de 2^{nde} que les fréquences sont observées sur un phénomène réel ; on l'élargit ici à une simulation numérique dans le cas où la probabilité de réalisation de l'événement attendu ne se calcule pas facilement. Si le fait de réaliser des essais 'physiques' est important, une simulation numérique est un complément intéressant comme illustration de « la loi des grands nombres ».

Situation déclenchante

Somme de quatre dés ...

On lance quatre dés à six faces (dés bien équilibrés) et on s'intéresse à la somme des valeurs des faces supérieures.

Quelle est la probabilité que cette somme soit inférieure ou égale à 18 ?

Problématique

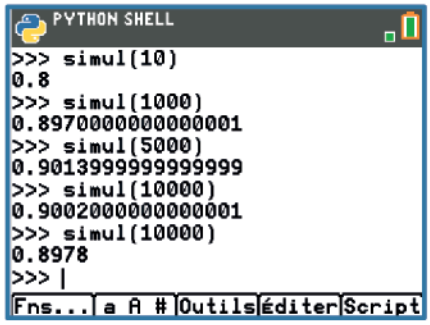
Comment traiter cette situation : modèle probabiliste ou simulation ?

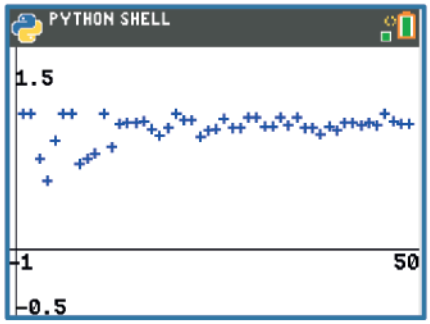
Fiche méthode

Proposition de résolution

On pourra effectuer des simulations grâce à un script en langage Python.

- Écrire une fonction **simul** de paramètre *n* (le nombre fois où on lance les quatre dés) qui renvoie la fréquence où la somme des valeurs des faces supérieures des quatre dés est inférieure ou égale à 18.
- Créer un compteur qui est incrémenté chaque fois que la somme des valeurs des faces supérieures des quatre dés est inférieure ou égale à 18.
- On peut d'une part observer l'évolution de la fréquence en fonction du nombre d'essais. On visualise alors une stabilisation de cette fréquence vers une certaine valeur, illustrant ainsi la « loi des grands nombres ».
- Cette observation peut être numérique (première copie d'écran) ou graphique : le script permettant de générer le graphique ci-contre est donné à la fin de cette fiche.
- Par ailleurs, on peut dès à présent observer la fluctuation d'échantillonnage en testant plusieurs fois la fonction **simul** avec la même valeur de *n*.





Remarque

Quelle que soit la méthode utilisée, il faudra un générateur de nombres (pseudo) aléatoires et donc importer la bibliothèque « random » par « **from random import ✶** »
Cette importation est en préambule du code.

On peut aussi choisir d'importer seulement la fonction randint issue de la bibliothèque random en saisissant « **from random import randint** ».

Lorsque l'on saisit « from random import ✶ », cela sous-entend que l'on importe toutes les fonctions de cette bibliothèque.

Pour profiter de tutoriels vidéos, Flasher le QRCode ou cliquer dessus

Fiche méthode

Etapes de résolution

La fonction **simul** simule le lancer de quatre dés. Elle a pour paramètre *n* qui représente le nombre de répétitions et renvoie la fréquence de réussite de l'expérience aléatoire.

```
EDITEUR : LOIGDNBR
LIGNE DU SCRIPT 0009
from random import randint

def simul(n):
  c=0
  for i in range(n):
    if randint(1,6)+randint(1,6)
      +randint(1,6)+randint(1,6)<
      =18:
        c=c+1
  return c/n

Fns... | a A # | Outils | Exéc | Script
```

- On initialise un compteur (représenté par la variable *c*) à 0.
- Chaque fois que la somme des quatre nombres entiers aléatoires compris entre 1 et 6 est inférieure ou égale à 18, ce compteur est incrémenté.
- Au final, le compteur est égal au nombre de cas favorables.
- En le divisant par *n*, le nombre d'essais, la fonction retourne la fréquence de réussite de l'expérience aléatoire.

Remarque

Pour un élève de 2nde, il n'est peut-être pas évident que :

- randint(1,6) + randint(1,6) + randint(1,6) + randint(1,6)
- randint(1,6) x 4
- randint(4,24)

ne représentent pas la même simulation.

Il peut être intéressant pour convaincre les élèves de lancer chacune de ces fonctions et de comparer les résultats comme l'illustre la copie d'écran ci-contre.

On observe des valeurs significativement différentes qui confirment que ces simulations ne sont pas équivalentes.

```
ÉDITEUR : LOIGDNBR
LIGNE DU SCRIPT 0020
def sim2(n):
  c=0
  for i in range(n):
    if randint(1,6)*4<=18:
      c=c+1
  return c/n

def sim3(n):
  c=0
  for i in range(n):
    if randint(4,24)<=18:

Fns… | a A # | Outils | Exéc | Script
```

```
PYTHON SHELL
>>> simul(1000)
0.9
>>> sim2(1000)
0.64
>>> sim3(1000)
0.706
>>>

Fns... | a A # | Outils | éditer | Script
```

Pour profiter de tutoriels vidéos, Flasher le QRCode ou cliquer dessus

Fiche méthode

Approche probabiliste

On peut avoir une approche 'probabiliste' en dénombrant le nombre de cas favorables.

Lorsqu'on lance quatre dés à six faces, dans combien de cas la somme des valeurs des faces supérieures des dés sortis est-elle inférieure ou égale à 18 ?

Si on se lance dans un dénombrement « à la main », cela risque d'être long et fastidieux, et le risque de se tromper n'est pas négligeable.

Un programme peut dénombrer le nombre de cas favorables ; la calculatrice est en effet adaptée pour effectuer ce type de tâche longue et fastidieuse.

On va donc tester toutes les sommes de quatre entiers compris entre 1 et 6 et comptabiliser le nombre de cas où la somme est inférieure ou égale à 18.

Ceci permet d'obtenir la probabilité de réalisation de l'événement voulu, et de comparer a posteriori les fréquences obtenues avec cette probabilité.

La fonction **test** ci-contre retourne le résultat sous la forme du couple (numérateur, dénominateur) soit ici (1170,1296) et on pourra comparer cette valeur à celles obtenues par simulations.

On obtient $p = \frac{1170}{1296} \approx 0{,}90278$.

```
ÉDITEUR : LOIGDNBR
LIGNE DU SCRIPT 0033
def test():
  c=0
  for i in range(1,7):
    for j in range(1,7):
      for k in range(1,7):
        for l in range(1,7):
          if i+j+k+l<=18:
            c=c+1
  return c,6**4
Fns… | a A # | Outils | Exéc | Script
```

```
PYTHON SHELL
>>> sim(1000)
0.8859999999999999
>>> sim(1000)
0.9140000000000001
>>> sim(1000)
0.8890000000000001
>>> test()
(1170, 1296)
>>> 1170/1296
0.9027777777777779
>>>
Fns… | a A # | Outils | éditer | Script
```

Complément : graphique par ti_plotlib

On peut réaliser le graphique présenté au début de cette fiche (fréquence de réussite en fonction du nombre d'essais), ce qui permet d'illustrer la stabilisation de la fréquence autour d'une valeur lorsque *n* augmente.

Pour cela, on utilise des fonctions de la bibliothèque `ti_plotlib` comme présenté ci-contre.

```
ÉDITEUR : LOIGDNBR
LIGNE DU SCRIPT 0045
import ti_plotlib as plt

def g(n):
  x,y=[],[]
  for i in range(1,n):
    x.append(i)
    y.append(sim(i))
  plt.cls()
  plt.window(-1,n,-0.5,1.5)
  plt.axes("on")
  plt.color(0,0,255)
  plt.scatter(x,y,"+")
  plt.color(0,0,0)
  plt.show_plot()
Fns… | a A # | Outils | Exéc | Script
```

Pour profiter de tutoriels vidéos, Flasher le QRCode ou cliquer dessus

Tirage sans remise et recherche d'une probabilité.

Dans cette activité, on va simuler le tirage d'une boule puis de deux boules sans remise dans une urne contenant des boules blanches, vertes et bleues. Puis on déterminera une valeur approchée de l'événement « tirer 2 boules blanches » dans une urne contenant 5 boules blanches, 3 vertes et 1 bleue.

Dans un script LC

1°) Ecrire une fonction `urne1` qui prend comme paramètres a, b, c des entiers naturels (tous non nuls), qui simule le tirage d'une boule dans une urne contenant a boules blanches, b boules vertes et c boules bleues et qui renvoie « blanc », « vert » ou « bleu ».

```
PYTHON SHELL
>>> urne1(5,6,7)
'blanc'
>>> urne1(5,6,7)
'vert'
>>> urne1(5,6,7)
'bleu'
```

2°) Ecrire une fonction `urne2` qui prend comme paramètres a, b, c des entiers naturels (tous plus grands ou égaux à 2), qui simule le tirage de 2 boules sans remise dans une urne contenant a boules blanches, b boules vertes et c boules bleues et qui renvoie le tuple des couleurs des boules tirées.

```
PYTHON SHELL
>>> urne2(5,6,7)
('bleu', 'bleu')
>>> urne2(5,6,7)
('bleu', 'bleu')
>>> urne2(5,6,7)
('blanc', 'vert')
```

3°) Ecrire une fonction `evalproba` qui prend comme argument un entier naturel non nul et qui renvoie la fréquence de l'événement « tirer 2 boules blanches » dans une urne contenant 5 boules blanches, 3 vertes et 1 bleue.

```
PYTHON SHELL
>>> evalproba(10)
0.2
>>> evalproba(100)
0.21
```

Tirage sans remise et recherche d'une probabilité.

Fonction `urne1`

1°) `h` représente un entier aléatoire compris entre 1 et `a+b+c`.

Si `h<=a` alors on a tiré une boule blanche.

Sinon si `h<=b` on a tiré une boule verte

Sinon on a tiré une boule bleue.

On n'oubliera pas d'importer la bibliothèque `random` afin d'utiliser la fonction `randint`.

Fonction `urne2`

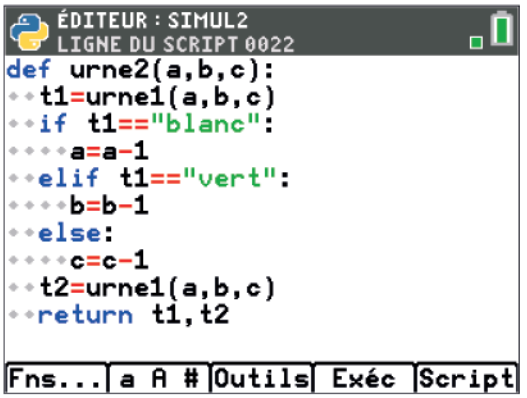
2°) On va commencer par tirer une boule de l'urne et on affecter sa couleur dans la variable `t1`.

Selon la couleur de la boule on retire 1 à l'effectif des boules de la couleur correspondante.

On a choisi de ne pas faire le test si ce nouveau nombre est positif car on a supposé que l'urne contient suffisamment de boules de chaque couleur pour pouvoir en tirer 2 de la même couleur.

Puis on tire une seconde boule dans l'urne dont les effectifs sont à jour. On stocke le résultat dans la variable `t2`.

On retourne le couple `t1,t2`.



Fonction `evalproba`

3°) Pour déterminer la fréquence recherchée on va effectuer `n` fois l'expérience « tirer 2 boules sans remise de l'urne ».

On stocke le résultat (qui est un couple « bleu », »blanc » par exemple) dans la variable `tirage`.

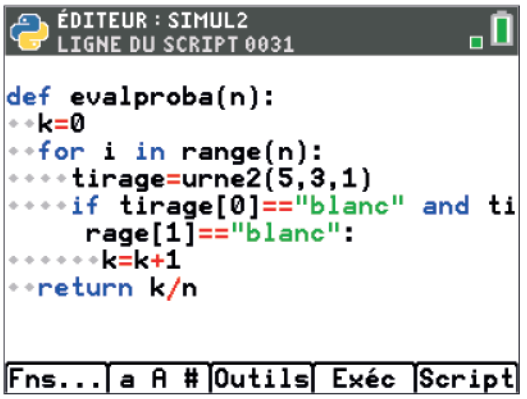
Si les deux valeurs `tirage[0]` et `tirage[1]` sont égales à « blanc » alors on ajoute 1 à la valeur `k` initialisée à 0 au début de la fonction.

`k/n` correspond à la fréquence recherchée.

On le vérifie par un calcul : la probabilité recherchée est

$$\frac{\binom{5}{2}}{\binom{9}{2}} = \frac{10}{36} = \frac{5}{18} \approx 0{,}2777\ldots$$

qui est proche de la fréquence obtenue pour de grandes valeurs de `n`.




```
>>> evalproba(10000)
0.2755
>>> evalproba(100000)
0.27778
```

Le lièvre et la tortue

Compétences visées

- **chercher**, expérimenter – en particulier à l'aide d'outils logiciels ;
- **modéliser**, faire une simulation, valider ou invalider un modèle ;
- **représenter**, choisir un cadre (numérique, algébrique, géométrique...), changer de registre ;
- **calculer**, appliquer des techniques et mettre en œuvre des algorithmes ;
- **communiquer** un résultat par oral ou par écrit, expliquer une démarche.

Ces compétences sont mises en œuvre dans le cadre de l'extrait du programme de 2nde GT ci-dessous :

« Simuler N échantillons de taille n d'une expérience aléatoire à deux issues. Si p est la probabilité d'une issue et f sa fréquence observée dans un échantillon, calculer la proportion des cas où l'écart entre p et f est inférieur ou égal à $\frac{1}{\sqrt{n}}$ »

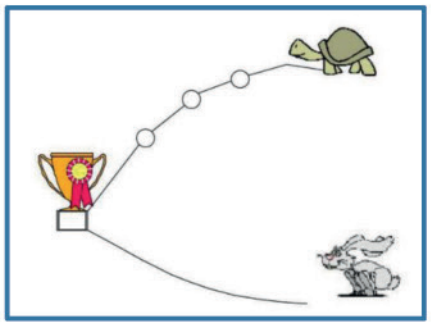
Situation déclenchante

Le lièvre et la tortue :

On lance un dé bien équilibré à six faces :

- à chaque lancer de dé, la tortue avance d'une case si le dé affiche les numéros 1, 2, 3, 4 ou 5 ;
- le lièvre gagne si le dé affiche le numéro 6.

Le but de la partie est d'atteindre la coupe : qui a la situation la plus favorable ?



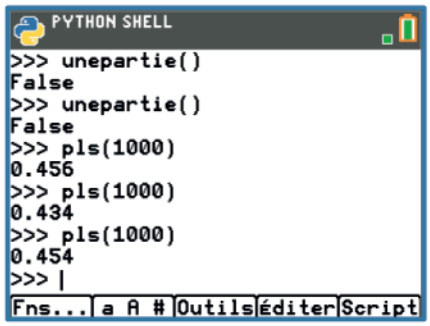
Problématique

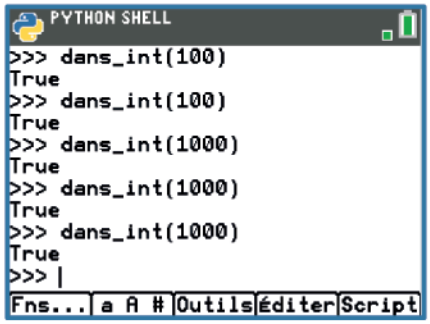
Comment modéliser la situation pour déterminer qui du lièvre ou de la tortue a la situation la plus favorable ?

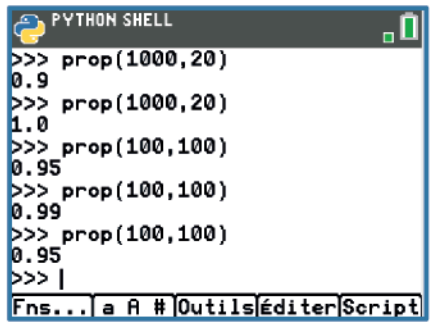
Fiche méthode

Proposition de résolution

- Écrire la fonction **unepartie** (sans argument) simulant une partie ; elle renvoie :
  - True pour une victoire de la tortue ;
  - False pour une victoire du lièvre.
- Écrire la fonction **pls** qui prend pour argument *n*, le nombre de répétitions voulues. Elle renvoie la fréquence de victoires de la tortue pour *n* parties.
- Écrire la fonction **dans_int** qui prend pour paramètre *n*, le nombre de parties simulées. Elle renvoie le booléen True si la fréquence de victoires de la tortue est à une distance inférieure ou égale à $\frac{1}{\sqrt{n}}$ de *p* (la probabilité de victoire de la tortue calculée par ailleurs égale à 0,482). Elle renvoie False si ce n'est pas le cas.
- Écrire la fonction **prop** qui a pour premier paramètre *n*, la taille de l'échantillon, et comme second paramètre *N*, le nombre d'échantillons ; elle renvoie la proportion des *N* échantillons de taille *n* testés qui sont tels que l'écart entre *p* et *f* est inférieur ou égal à $\frac{1}{\sqrt{n}}$.



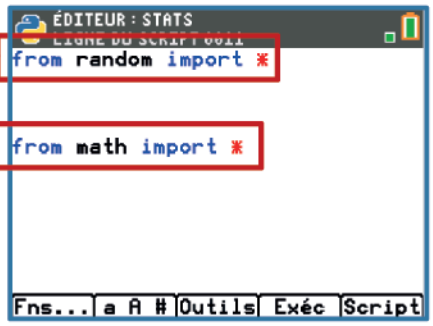




Remarque

Importation en préambule du code de la bibliothèque « random » par « **from random import ✶** » pour pouvoir utiliser la fonction **randint()**

importation de la bibliothèque « math » par « **from math import ✶** » pour pouvoir utiliser la fonction **fabs** (valeur absolue)



Pour profiter de tutoriels vidéos, Flasher le QRCode ou cliquer dessus

Fiche méthode

Etapes de résolution

La fonction **unepartie** n'a pas de paramètre et renvoie un booléen (True ou False) selon que la simulation donne une victoire de la tortue ou non.

```
ÉDITEUR : TORTUE
LIGNE DU SCRIPT 0011
from random import*
from math import *

def unepartie():
  for i in range(4):
    if randint(1,6)==6:
      return False
  return True
```

Elle simule au plus près le jeu : on répète jusqu'à quatre lancers de dé à six faces, et dès que la valeur 6 est sortie, **la fonction renvoie False.** En effet, dès que le script rencontre « **return** », la fonction s'arrête ; c'est aussi un gain de temps, les répétitions seront moins nombreuses.

La fonction **pls** prend pour paramètre `n`, le nombre de parties simulées. Elle renvoie la fréquence de parties gagnées par la tortue.

Elle utilise la fonction **unepartie** et un compteur (variable `c`) comptabilise le nombre de parties simulées donnant une victoire de la tortue.

```
ÉDITEUR : TORTUE
LIGNE DU SCRIPT 0023

def pls(n):
  c=0
  for i in range(n):
    if unepartie():
      c+=1
  return c/n
```

C'est l'utilisation successive de ces fonctions qui rend le programme très lisible et efficace.

A noter le raccourci **c+=1** qui signifie **c=c+1**

La fonction **dans_int** a pour paramètre `n` ; elle renvoie le booléen True ou False selon que la distance entre la fréquence de victoires de la tortue après `n` parties (donnée par **pls**(`n`)) et p est inférieure ou égale à $\frac{1}{\sqrt{n}}$ ou pas.

```
ÉDITEUR : TORTUE
LIGNE DU SCRIPT 0030

def dans_int(n):
  return fabs(pls(n)-0.482)<=1/s
      qrt(n)
```

Bien noter que « $|\text{pls}(n) - 0{,}482| \leqslant \frac{1}{\sqrt{n}}$ » est un booléen ; il prend la valeur True si le test est vérifié, la valeur False sinon. Cela permet d'avoir une syntaxe très concise.

La fonction **prop** a pour paramètres `n` (taille d'un échantillon de parties) et N (nombre d'échantillons) ; elle renvoie la proportion des N échantillons de taille `n` qui ont une fréquence de victoires de la tortue proche de p (distance inférieure ou égale à $\frac{1}{\sqrt{n}}$).

```
ÉDITEUR : TORTUE
LIGNE DU SCRIPT 0037

def prop(n,n_rep):
  c=0
  for i in range(n_rep):
    if dans_int(n) :
      c+=1
  return c/n_rep
```

Elle utilise efficacement la fonction **dans_int**.

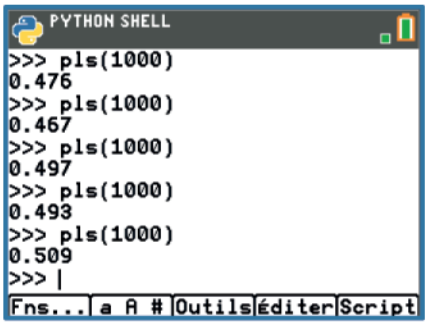
Pour profiter de tutoriels vidéos, Flasher le QRCode ou cliquer dessus

Fiche méthode

Remarques

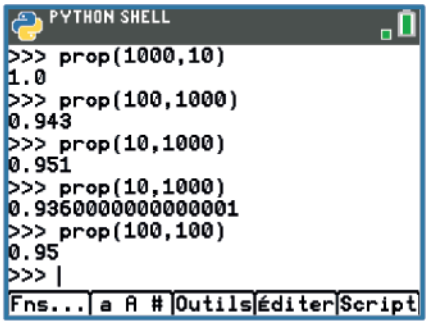
Il peut être intéressant d'exécuter plusieurs fois de suite l'instruction **`pls(1000)`** afin de constater la fluctuation d'échantillonnage sur des échantillons de même taille.

L'appel de la même instruction par la flèche directionnelle vers le haut est particulièrement utile ici : en effet, un appui sur la flèche directionnelle vers le haut appelle l'instruction tapée précédemment (deux appuis successifs rappellent l'instruction écrite deux lignes au-dessus, et ainsi de suite) ; en appuyant sur **`entrer`**, on exécute à nouveau l'instruction écrite dans la console.



L'instruction **prop**(n,N) est à manier avec précaution ! Dans la mesure où **prop**(1000,100) demande 100 répétitions de tests sur des échantillons de taille 1000 ... le calcul prendra une vingtaine de secondes.

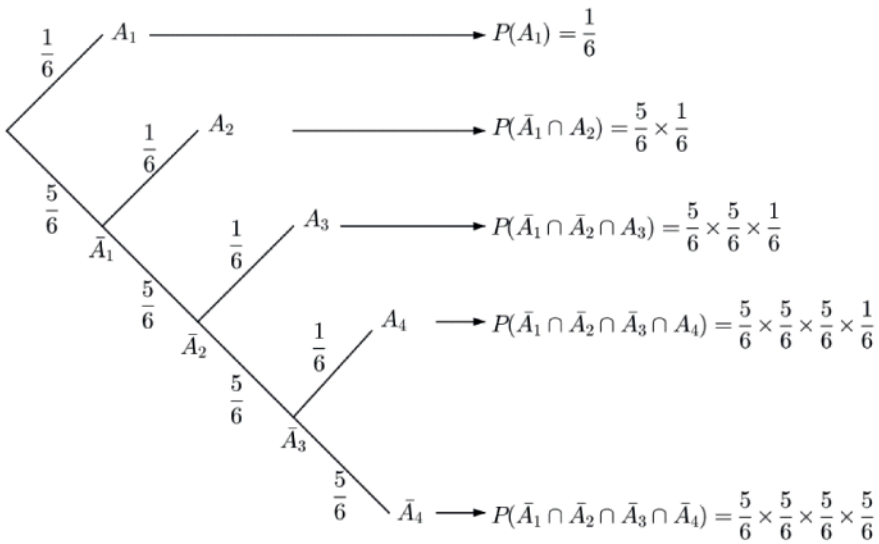
Le calcul de **prop**(1000,1000) est à déconseiller !



Pour aller plus loin

Pour ce qui est du calcul de la probabilité de victoire de la tortue, on peut proposer à un groupe d'élèves d'effectuer cet approfondissement / recherche, par exemple à l'aide d'un arbre comme celui présenté ci-contre :

On obtient ainsi la probabilité de victoire de la tortue : $p = \left(\frac{5}{6}\right)^4 \approx 0{,}482$



on note A_i l'événement « obtenir un numéro 6 au $i^{ème}$ lancer »

Pour profiter de tutoriels vidéos, Flasher le QRCode ou cliquer dessus

Fibonacci - Le problème des chevaliers.

Introduction

On peut approcher la récursivité en prenant l'exemple d'une suite définie par récurrence :

Soit (u_n) la suite définie par $u_0 = 4$ et $\forall n \in \mathbb{N}\ u_{n+1} = 2u_n - 3$

Pour calculer u_n à l'aide d'une fonction Python on peut procéder ainsi :

ÉDITEUR : RECURSI
LIGNE DU SCRIPT 0001

```
def u(n):
  if n==0:
    return 4
  else:
    return 2*u(n-1)-3
```

PYTHON SHELL

```
>>>
>>> from RECURSI import *
>>> u(0)
4
>>> u(1)
5
>>> u(2)
7
>>> u(3)
11
```

On peut aussi utiliser un algorithme itératif :

ÉDITEUR : RECURSI
LIGNE DU SCRIPT 0017

```
def u2(n):
  u=4
  for i in range(n):
    u=2*u-3
  return u
```

PYTHON SHELL

```
>>>
>>> from RECURSI import *
>>> u2(0)
4
>>> u2(1)
5
>>> u2(2)
7
>>> u2(3)
11
```

Les fonctions définies de façon récursive ont un coup spatial qu'il ne faut pas ignorer.

Ainsi, sur la TI-83 Premium CE le nombre de récursions est limité à 25 :

PYTHON SHELL

```
>>> u(24)
16777219
>>> u(25)
33554435
>>> u(26)
  File "RECURSI.py", line 6, in
u
  File "RECURSI.py", line 6, in
u
  File "RECURSI.py", line 6, in
u
  File "RECURSI.py", line 6, in
u
RuntimeError: maximum recursion
depth exceeded
>>>
```

Fns... | a A # | Outils | éditer | Script

Pas de problème pour la définition avec itérations

PYTHON SHELL

```
>>> u2(700)
526013590154837350724098988288 01
286655503398028231738594982809 03
068732154297080822113666536277 58
845122698296885617821771301943 22
501838038631278147706518808499 55
223671128444598191663757884322 71
7271293251735781379
>>>
```

Fns... | a A # | Outils | éditer | Script

Fibonacci - Le problème des chevaliers.

Dans une script RECURSI

1°) Programmer le calcul de u_n par récursivité avec (u_n) la suite de Fibonacci.

On appellera `fibo` la fonction qui prend comme argument $n \in \mathbb{N}$ et qui renvoie u_n.

On rappelle que $u_0 = 0$ et $u_1 = 1$ et $\forall n \in \mathbb{N},\ n \geq 2 : u_n = u_{n-1} + u_{n-2}$

Application : Calculer u_{20}.

Dans un script KNIGHT

2°) N chevaliers sont disposés en cercle, portant chacun un numéro, 1,...,N. Le numéro 1 porte une épée avec laquelle il tue le chevalier à sa gauche (le numéro 2) et passe l'épée au numéro 3 qui tue le chevalier à sa gauche etc. L'épée tourne ainsi entre les chevaliers restants jusqu'à ce qu'il n'y ait plus qu'un seul chevalier debout.

Si on commence avec la liste suivante des chevaliers :

[1,2,3,4,5] (en rouge celui avec l'épée)

Les tours successifs seront :

[1,3,4,5]

[1,3,5]

[3,5]

[3]

La programmation de cet algorithme est plus simple si on place en premier le chevalier qui tient l'épée :

[1,2,3,4,5]

[3,4,5,1]

[5,1,3]

[3,5]

[3]

Soit `tue` la fonction qui prend en argument une liste de chevaliers et qui renvoie le chevalier survivant.

Compléter le script de la fonction `tue`.

Application : Quel est le chevalier vivant lorsqu'il y en a 5 au début ? Et s'il y en a 12 ?

Fibonacci - Le problème des chevaliers.

Fonction `fibo`

1°) Il faudra faire 3 cas : Les deux premiers étant utilisés pour le calcul de u_0 et de u_1.

L'autre cas fait un appel à `fibo(n-1)` et `fibo(n-2)`.

On trouve $u_{20} = 6765$.

ÉDITEUR : RECURSI
LIGNE DU SCRIPT 0023

```
def fibo(n):
  if n==0:
    return 0
  elif n==1:
    return 1
  else:
    return fibo(n-1)+fibo(n-2)
```

PYTHON SHELL

```
>>> fibo(20)
6765
```

Fonction `tue`

2°) En utilisant l'astuce de garder le chevalier à l'épée en tête de la liste, la récursivité est beaucoup plus facile.

Lorsqu'il reste un seul élément dans la liste des chevaliers, on le renvoie, c'est le vainqueur.

Sinon on reprend la liste de l'élément d'indice 2 jusqu'à la fin (car le chevalier d'indice 1 est mort). Et on recopie l'assassin, c'est-à-dire `liste[0]` à la fin de la liste.

ÉDITEUR : KNIGHT
LIGNE DU SCRIPT 0029

```
def tue(liste):
  if len(liste)==1:
    return liste[0]
  else:
    return tue(liste[2:]+[liste[
      0]])
```

PYTHON SHELL

```
>>> from KNIGHT import *
>>> tue([1,2,3,4,5])
3
>>> tue([1,2,3,4,5,6,7,8,9,10,11
,12])
9
```

On peut aussi programmer cette fonction de façon itérative, c'est un peu plus long, mais dans ce cas on peut prendre autant de chevaliers que l'on souhaite… (En Python le nombre de récursions est toujours limité).

ÉDITEUR : KNIGHT
LIGNE DU SCRIPT 0001

```
from math import *
def uncoup(liste,i):
  n=len(liste)
  if i==n-1:
    liste.pop(0)
    j=0
  else:
    liste.pop(i+1)
    j=(i+1)%(n-1)
  return liste,j
```

Fns... | a A # | Outils | Exéc | Script

ÉDITEUR : KNIGHT
LIGNE DU SCRIPT 0011

```
def chevalier(n):
  liste=list(range(1,n+1))
  i=0
  while len(liste)>1:
    tour=uncoup(liste,i)
    liste=tour[0]
    i=tour[1]
  return liste[0]

def tue(liste):
```

Fns... | a A # | Outils | Exéc | Script

On obtient les mêmes résultats que la fonction précédente.

PYTHON SHELL

```
>>> chevalier(5)
3
>>> chevalier(12)
9
```

Equation réduite de droite.

Soit $(O,\vec{\imath},\vec{\jmath})$ un repère du plan. Soient $A\begin{vmatrix}x_A\\y_A\end{vmatrix}$, $B\begin{vmatrix}x_B\\y_B\end{vmatrix}$ et $C\begin{vmatrix}x_C\\y_C\end{vmatrix}$ où x_A, y_A, x_B, y_B, x_C et y_C sont des réels. On suppose que $A \neq B$.

Le but de cette activité est de donner l'équation réduite de la droite (AB)

Dans un script DROITE

1°) Ecrire une fonction `oy` qui prend comme arguments x_A, x_B et qui renvoie : `True` si (AB) est parallèle à l'axe des ordonnées et `False` sinon.

```
PYTHON SHELL
>>> oy(8,10)
False
>>> oy(7,7)
True
```

2°) Ecrire une fonction `reduite` qui prend comme arguments x_A, y_A, x_B, y_B et qui renvoie :

c si l'équation réduite de (AB) est de la forme $x = c$

m,p si l'équation réduite de (AB) est de la forme $y = mx + p$

Application : Prendre $A\begin{vmatrix}4\\5\end{vmatrix}$ et $B\begin{vmatrix}8\\-1\end{vmatrix}$ puis $A\begin{vmatrix}4\\5\end{vmatrix}$ et $B\begin{vmatrix}4\\9\end{vmatrix}$.

```
PYTHON SHELL
>>> reduite(4,5,8,-1)
(-1.5, 11.0)
>>> reduite(4,5,4,9)
4
```

3°) On souhaite avoir un affichage de ce type (voir ci-contre) :

On va supposer que x_A, y_A, x_B, y_B sont des entiers.

Ecrire une fonction `reduite2` qui prend comme argument x_A, y_A, x_B, y_B et qui renvoie l'affichage (au format `string`) de l'équation réduite de (AB).

Application : Reprendre les coordonnées des points A et B de la question 3.

```
PYTHON SHELL
>>> reduite2(4,5,8,-1)
'y = -1.5 * x +11.0'
>>> reduite2(4,5,4,9)
'x = 4'
```

4°) Comment faire pour afficher une fraction irréductible pour m et p ?

```
PYTHON SHELL
>>> reduite3(4,5,8,15)
'y = 5/2 * x -5'
```

Equation réduite de droite.

Fonction `oy`

1°) On renvoie `True` lorsque `xa` et `xb` sont égaux. On rappelle que la comparaison est symbolisée par `==`.

ÉDITEUR : DROITE
LIGNE DU SCRIPT 0009

```
def oy(xa,xb):
  if xa==xb:
    return True
  else:
    return False
```

Fonction `reduite`

2°) Si la droite est parallèle à l'axe des ordonnées, c'est-à-dire si la fonction `oy` renvoie `True` alors on retourne `c` qui vaut `xa` (ou `xb` au choix).

Sinon on calcule le coefficient directeur et l'ordonnée à l'origine et on renvoie le couple `m,p`.

ÉDITEUR : DROITE
LIGNE DU SCRIPT 0016

```
def reduite(xa,ya,xb,yb):
  if oy(xa,xb):
    return xa
  else:
    m=(yb-ya)/(xb-xa)
    p=yb-m*xb
    return m,p
```

Fonction `reduite2`

3°) On reprend la même structure que la fonction précédente.

Cette fois on renvoie du texte qu'on formate pour faire apparaître l'écriture habituelle d'une équation réduite.

On utilise la fonction Python `str` qui permet de convertir un nombre en chaîne de caractères.

ÉDITEUR : DROITE
LIGNE DU SCRIPT 0023

```
def reduite2(xa,ya,xb,yb):
  if oy(xa,xb):
    return "x = "+str(xa)
  else:
    m=(yb-ya)/(xb-xa)
    p=yb-m*xb
    return "y = "+str(m)+" * x +
      "+str(p)
```

Fns... | a A # | Outils | Exéc | Script

Dans certains cas l'affichage est agréable :

PYTHON SHELL

```
>>> reduite2(4,5,8,-1)
'y = -1.5 * x +11.0'
>>> reduite2(4,5,4,9)
'x = 4'
```

Mais dans d'autres cas, il est difficilement lisible :

PYTHON SHELL

```
>>> reduite2(3,4,14,12)
'y = 0.7272727272727273 * x +1.8
18181818181818'
```

On va résoudre ce problème dans la question suivante…

Equation réduite de droite.

Fonction `reduite3`

4°) La particularité de ce script est qu'on va utiliser la fonction `pgcd` que nous avons construit précédemment. Pour cela on indique qu'on souhaite utiliser les fonctions du script ARITHM :

```
from ARITHM import *
```

Il suffit après de simplifier numérateur et dénominateur de chaque fraction par leur pgcd.

On a aussi envisagé le cas où le dénominateur vaut 1 pour ne pas afficher $\frac{7}{1}$ mais 7…

Et, pour le coefficient p, on a évité l'affichage +-5 lorsque que p était négatif. Cela fait beaucoup de cas à considérer ce qui rend le script un peu plus complexe.

Mais nous n'avons pas envisager tous les cas particuliers…

En reprenant l'exemple précédent on obtient :

PYTHON SHELL

```
>>> reduite3(3,4,14,12)
'y = 8/11 * x + 20/11'
```

ÉDITEUR : DROITE
LIGNE DU SCRIPT 0033

```
from ARITHM import *
def reduite3(xa,ya,xb,yb):
  if oy(xa,xb):
    return "x = "+str(xa)
  else:
    d=pgcd(abs(yb-ya),abs(xb-xa)
      )
    mnum=(yb-ya)//d
    mdenom=(xb-xa)//d
    pnum=yb*mdenom-mnum*xb
    pdenom=mdenom
    d2=pgcd(abs(pnum),abs(pdenom
      ))
    pnum=pnum//d2
    pdenom=pdenom//d2
    if mdenom!=1:
      maff=str(mnum)+"/"+str(mde
      nom)
    else:
      maff=str(mnum)
    if pdenom!=1:
      paff=str(pnum)+"/"+str(pde
      nom)
    else:
      paff=str(pnum)
    if pnum/pdenom>0:
      paff="+ "+paff
    return "y = "+maff+" * x "+p
      aff
```

Des formules dans un repère

F. GIROD

Compétences visées

- **chercher**, expérimenter – en particulier à l'aide d'outils logiciels ;
- **représenter**, choisir un cadre (numérique, algébrique, géométrique...), changer de registre ;
- **raisonner**, démontrer, trouver des résultats partiels et les mettre en perspective ;
- **calculer**, appliquer des techniques et mettre en œuvre des algorithmes.

Le programme de la classe de 2nde GT propose explicitement une approche algorithmique pour déterminer les coordonnées du milieu d'un segment, la distance entre deux points, la caractérisation de la colinéarité de deux vecteurs et l'alignement de trois points.

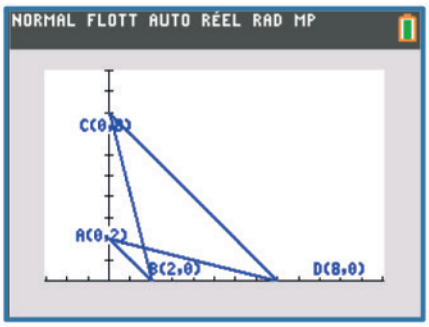
Situation déclenchante

Géométrie dans un repère

Sur la figure ci-contre : A (0,2), B (2,0), C (0,8) et D (8,0) sont quatre points du plan dont on donne les coordonnées.

On note E le milieu de [AB], F le milieu de [DC] et G l'intersection des segments [AD] et [BC].

Les points E, F et G sont-ils alignés ?



Problématique

Soient A (xA,yA), B (xB,yB), C (xC,yC) et D (xD,yD) ; créer des programmes permettant de :

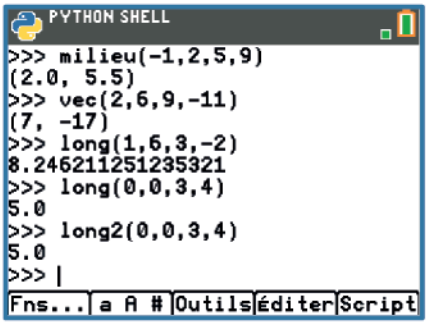
1. Déterminer les coordonnées de I, milieu de [AB] ;
2. Déterminer la longueur du segment [AB] ;
3. Déterminer les coordonnés du vecteur $\overrightarrow{AB}$;
4. Déterminer si les vecteur $\overrightarrow{AB}$ et $\overrightarrow{CD}$ sont colinéaires ;
5. Déterminer si les points A, B et C sont alignés.

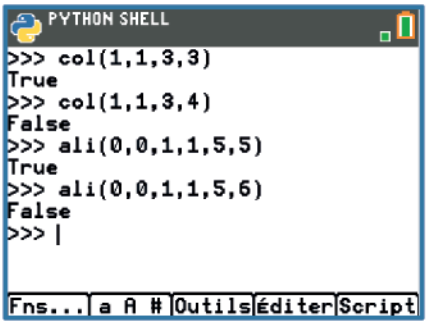
Fiche méthode

Proposition de résolution

On va créer plusieurs fonctions correspondantes à chaque situation.

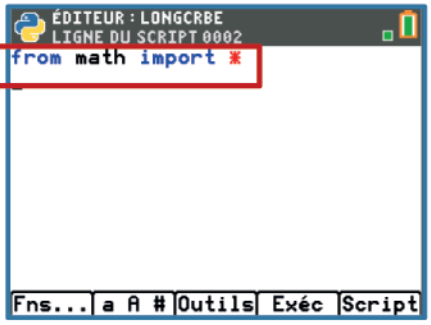
- Écrire la fonction **milieu** dont les quatre paramètres sont les coordonnées des extrémités du segment et renvoie les coordonnées du milieu de ce segment. Le résultat est rendu sous forme d'un tuple de deux valeurs. Son affichage est très proche de celui de coordonnées de points, ce qui en facilité la lecture.
- Écrire la fonction **vec** dont les quatre paramètres sont les coordonnées des points A et B et qui renvoie les coordonnées du vecteur $\overrightarrow{AB}$.
- Écrire la fonction **long** dont les quatre paramètres sont les coordonnées des extrémités du segment dont on veut connaître la longueur et qui renvoie une valeur approchée de la longueur de ce segment (la fonction **long2** en est une variante et donne évidemment le même résultat).
- Écrire la fonction **col** qui a pour paramètres les coordonnées des deux vecteurs dont on teste la colinéarité. Elle retourne un booléen : True si les deux vecteurs sont colinéaires, False dans le cas contraire.
- Écrire la fonction **ali** (on proposera une variante en terme de syntaxe nommée **ali2**) qui a pour paramètres les coordonnées des trois points dont on teste l'alignement. Elle retourne un booléen : True si les trois points sont alignés, False dans le cas contraire.





Remarque

Importation en préambule du code de la bibliothèque « math » par « **from math import ✶** » pour pouvoir utiliser la fonction **sqrt** (racine carrée)

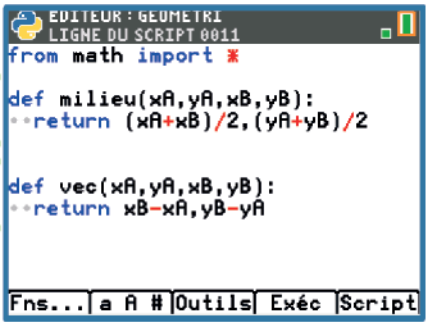


Pour profiter de tutoriels vidéos, Flasher le QRCode ou cliquer dessus

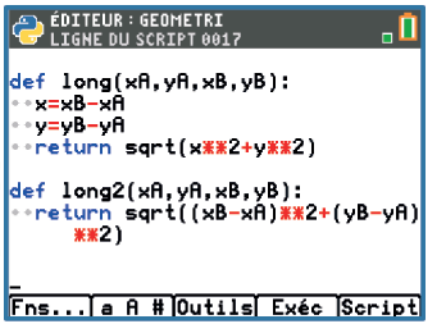
Fiche méthode

Etapes de résolution

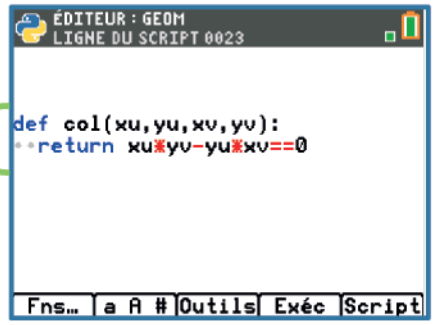
- La fonction **milieu** retourne un tuple de deux valeurs qui correspondent aux coordonnées du milieu du segment dont on a saisi en paramètres les coordonnées des extrémités.
- La fonction **vec** est similaire.



- La fonction **long** donne la longueur d'un segment ; les paramètres sont les coordonnées des extrémités du segment considéré.
- On en propose une seconde version, **long2** qui a une syntaxe plus concise.



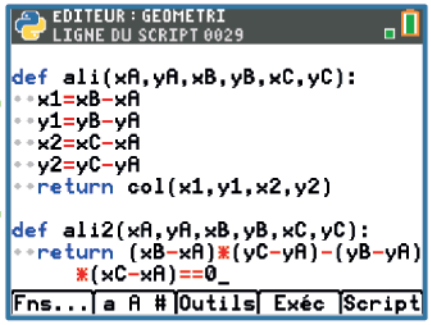
- La fonction **col** ainsi rédigée est efficace car elle retourne un booléen (True dans le cas où les vecteurs considérés sont colinéaires, False dans le cas contraire). « xu*yv-yu*xv==0 » est en effet un booléen.
  == signifie que l'on compare les deux quantités xu*yv-yu*xv et 0.



- On utilise la propriété suivante : A, B, C sont alignés si et seulement si $\overrightarrow{AB}$ et $\overrightarrow{AC}$ sont colinéaires. On propose deux syntaxes pour tester l'alignement de trois points.

  La première, **ali** utilise la fonction **col** et illustre bien la démarche consistant à étudier la colinéarité de deux vecteurs construits à partir de ces trois points.

  La seconde syntaxe, **ali2** reprend le principe utilisé pour la fonction **col** en retournant directement un booléen.



Pour profiter de tutoriels vidéos, Flasher le QRCode ou cliquer dessus

Fiche méthode

Retour à la situation déclenchante

On peut utiliser la fonction **milieu** pour déterminer les coordonnées des points E et F.

Pour déterminer les coordonnées de G, on peut chercher l'équation des droites (AD) et (BC) ; cela donne :

- (AD) : $y = -0{,}25x + 2$
- (BC) : $y = -4x + 8$

On peut utiliser le solveur de systèmes d'équations de la calculatrice pour déterminer les coordonnées de leur point d'intersection.

Pour cela, taper sur **résol** puis valider la touche 2 : PlySmlt2 ; on choisit à nouveau la touche 2 : SOLVEUR SYST D'ÉQUATIONS

On choisit de résoudre un système de deux équations à deux inconnues en validant les bons paramètres et on saisit les coefficients comme ci-contre.

Une fois RÉSOL validé, la solution est proposée (résultat donné ici sous forme fractionnaire).

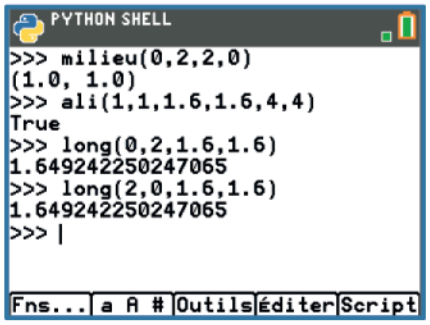
On trouve ainsi : G (1.6,1.6)

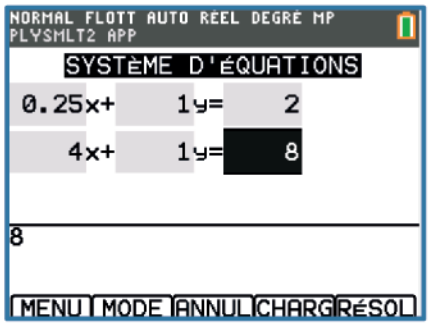
On peut à présent tester l'alignement des points E, G et F par **ali**(1,1,1.6,1.6,4,4).

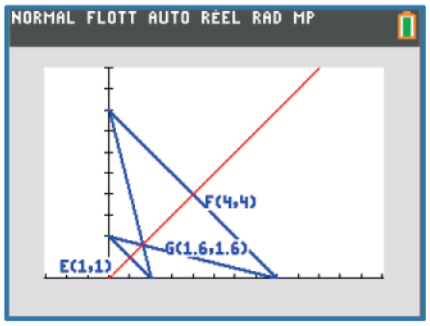
On peut également évaluer les longueurs des segments [AF] et [BF] par **long**(0,2,1.6,1.6) et **long**(2,0,1.6,1.6).

Ces programmes une fois exécutés permettent de lancer deux pistes prouvant l'alignement de ces points :

- En montrant que les vecteurs $\overrightarrow{EG}$ et $\overrightarrow{EF}$ sont colinéaires.
- En montrant que AG=GB et AF=FB, prouvant ainsi que les points E, F et G appartiennent tous trois à la médiatrice de [AB].







Pour profiter de tutoriels vidéos, Flasher le QRCode ou cliquer dessus

Plus d'informations sur le site internet de T³ France :

- Des ressources pédagogiques pour votre classe
- Un programme de formations gratuites sur site et en ligne
- Des vidéos d'aide à la prise en main de la technologie

t3france.fr

Un service après-vente est également accessible depuis le site **education.ti.com/fr/csc**