1 Programmer un calcul

1.1 Version initiale

1. Reproduire le programme ci-contre dans un script (à gauche) du logiciel **Pyzo**.
2. Exécutez ce programme : onglet `Exécuter` ou `ctrl+shift+Entrée`.
3. Tester ce programme avec `x1=8`, `x2=13`, `x3=11`. Que renvoie le programme en sortie ?
4. Que calcule ce programme pour les trois nombres saisis en entrée ?

Code Python :

```
x1=eval(input("Note 1 : "))
x2=eval(input("Note 2 : "))
x3=eval(input("Note 3 : "))
m=(x1+x2+x3)/3
print(m)
```

1.2 Avec une entrée globale : affectation multiple

1. Reproduire le programme ci-contre.
2. Exécutez ce programme : onglet `Exécuter` ou `ctrl+shift+Entrée`
3. Tester ce programme avec `x1=8`, `x2=13`, `x3=11`. Que renvoie le programme en sortie ?

Code Python :

```
x1,x2,x3=eval(input("Entrer les
    trois notes separees par une
    virgule : "))
m=(x1+x2+x3)/3
print(m)
```

1.3 Avec une fonction

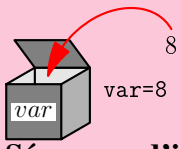
1. Reproduire le programme ci-contre.
2. Exécutez ce programme.
3. Tester ce programme en saisissant dans la console à droite : **`moyenne`**`(8,13,11)`.

Code Python :

```
def moyenne(x1,x2,x3):
    """calcul d'une moyenne"""
    return (x1+x2+x3)/3
```

Trois instructions élémentaires :

- **Affectation d'une variable :** Une variable informatique peut être considérée comme une étiquette collée sur une boîte qui peut contenir successivement plusieurs valeurs. Le contenu de la boîte peut varier au cours de l'exécution d'un programme (contrairement à une variable mathématique).



Dans Python, le symbole **=** n'est pas celui de l'égalité mathématique. Sa fonction est **d'affecter** une valeur à une variable : on stocke une valeur numérique ou du texte dans une boîte de la mémoire. Ici `var` reçoit la valeur 8 (on peut écrire `var` ← `8`).

- **Séquence d'instructions :** en Python, une séquence d'instructions s'obtient en écrivant simplement à la suite, dans un ordre déterminé, différentes instructions, chacune sur une ligne, avec la même indentation (décalage) (on reviendra sur l'utilisation et la place essentielle de l'indentation).
- **Fonction :** une fonction est une suite d'instructions que l'on peut appeler avec un nom. Sa déclaration est structurée ainsi :
  - ➢ la déclaration commence par le mot-clé **`def`** suivi du nom de la fonction, puis des paramètres et enfin deux points **`:`** pour déclarer le bloc d'instruction indenté correspondant à la fonction ;
  - ➢ dès que le programme rencontre l'instruction **`return`**, il quitte la fonction et renvoie la ou les valeurs indiquées.

Définitions : input() et print() :

- **entrées :** la fonction `input()` dont la syntaxe est : `var=input("question")` affiche une invite où figure le texte `question` et un espace dans lequel on entrera ce qui est demandé. La réponse est alors affectée à la **variable** `var` qui est alors considérée comme **une chaîne de caractères**, ce qui peut poser un problème si on veut utiliser la réponse comme un nombre par la suite.
  Dans ce cas, l'instruction `var=eval(input("question"))` cherche à convertir la chaîne de caractère de la réponse en un nombre.
- **sorties :** la fonction `print(var)` affiche le contenu de la variable `var`.
  - ➢ `print(a,b)` affiche à la suite sans passer à la ligne les éléments `a` et `b`.
  - ➢ `print("Texte",var)` affiche le texte `Texte` suivi de la valeur de la variable `var`.

2 Une quatrième instruction élémentaire : le test

On veut modifier la fonction **`moyenne`** en la complétant par un commentaire :

— **Si** la moyenne est supérieure ou égale à 16, alors on affiche `"Très bien"` ;

— **Si** elle est comprise entre 13 (inclus) et 16 (exclus), alors on affiche `"Satisfaisant"` ;

— **Si** elle est comprise entre 10 (inclus) et 13 (exclus), alors on affiche `"Convenable"`

— **sinon**, on affiche `"Insuffisant"`

Compléter le programme ci-dessous pour qu'il réponde à la demande.

Code Python :

```python
def moyenne(x1,x2,x3):
    m=(x1+x2+x3)/3
    if ..... :
        print(m," : Très bien")
    elif ..... :
        print(m," : Satisfaisant")
    elif ..... :
        print(m," : Convenable")
    else:
        print(m," : Insuffisant")
```

Le test et la structure alternative :

Selon qu'une certaine condition est vérifiée ou non, on fera un certain traitement (constitué d'une ou plusieurs instructions). Cela correspond à la structure `SI...ALORS ....`

- `if test :`
  - ➢ en Python, le test `SI`, va se traduire par l'instruction **`if`**. Si le `test` est vérifié, il effectue l'ensemble des instructions **indentées** (c'est-à-dire décalées). L'indentation peut-être obtenue à l'aide de la touche de tabulation ou se fait automatiquement avec la touche "Entrée" ;
  - ➢ le `ALORS` n'apparaît pas en Python, mais est remplacé par deux points **`:`**
  - ➢ il n'y a pas de mot clé `"end"` pour signaler la fin du bloc de lignes concernées par les instructions conditionnelles. Il faut revenir à l'indentation précédente pour sortir du **`if`**.
- `if test :... else ...` cette structure effectue les instructions indentées lorsque le *`test`* est vérifié, **sinon** elle effectue les instructions alternatives indentées (ce qu'il y a après le `else`). Le `else` est aligné avec le `if` qui lui correspond.
- `if test1 : instructions1 elif test2 : instructions2` Le terme `elif` est une contraction de "else if" (sinon, si...) et permet de traiter par disjonction de cas.
  Cette structure effectue les *`instructions1`* indentées lorsque le *`test1`* est vérifié, **sinon** elle effectue le *`test2`* et, **si** celui-ci est vérifié, effectue les *`instructions2`* indentées.
  - ➢ on peut enchaîner autant d'instructions `elif` que nécessaire ;
  - ➢ on peut éventuellement terminer la série de `elif` par un `else` pour traiter tous les cas restants.
- Opérateurs de test :

Symbole	Signification
==	Opérateur de test d'égalité
> et <	Opérateur de tests d'inégalités strictes habituelles
<= et >=	Opérateur de tests d'inégalités strictes larges ⩽ et ⩾
!=	Opérateur de test de non-égalité (« différent de »).

3 Manipulation d'un grand nombre de valeurs : les listes

On veut modifier la fonction **moyenne** vue précédemment pour qu'elle calcule la moyenne de n'importe quelle série de valeurs.

1. Reproduire le programme ci-contre.
2. Quelle est la différence avec le programme précédent ?
3. Exécutez ce programme : onglet `Exécuter` ou `ctrl+shift+Entrée`.
4. Tester ce programme en saisissant dans la console à droite : **moyenne**`([14,17,18,8,6,11,15])`.

Code Python :

```python
def moyenne(L):
    m=sum(L)/len(L)
    if m>=16 :
        print(m," : Très bien")
    elif m>=13 :
        print(m," : Satisfaisant")
    elif m>=10 :
        print(m," : Convenable")
    else:
        print(m," : Insuffisant")
```

Les listes

Une liste est une suite d'éléments numérotés dont **le premier indice est 0**. Une liste n'a donc (presque) pas de limite de taille. Python affiche la liste comme un tableau d'éléments de cette liste séparés par **une virgule** et délimité par des **crochets**.

- l'instruction `L=[]` créé une liste `L` vide ;
- Pour atteindre l'élément d'indice `i` de la liste `L`, il suffit d'écrire `L[i]` ;
- la fonction `len(L)` renvoie le nombre d'éléments de la liste `L` ;
- la fonction `sum(L)` renvoie la somme des éléments de la liste `L` ;
- la fonction `sorted(L)` renvoie la liste `L` triée par ordre croissant.
  La fonction `sorted(L,reverse=True)` renvoie la liste triée par ordre décroissant) ;
- l'instruction `e in L` teste si l'élément `e` est dans la liste `L` et renvoie `True` ou `False` ;
- la méthode `L.append(e)` ajoute l'élément `e` à la fin de la liste `L`.
- la méthode `L.insert(i,e)` insère l'élément `e` au rang `i` de la liste `L`.
- la méthode `L.pop(i)` supprime l'élément d'indice `i` de la liste `L`.

4 Une cinquième instruction élémentaire : la boucle

On veut désormais calculer la moyenne de la classe ci-dessous :

Note	6	7	8	9	10	11	12	13	14	15	16	17
Nombre d'élèves	3	1	2	1	2	5	3	3	3	2	1	1

1. Rappeler le calcul mathématique à faire pour le calcul de la moyenne de la classe.
2. Dans le programme ci-dessous, quelle est la nature des variables `"valeurs"`, `"effectifs"` et `"produits"` ?
3. Que fait la commande `produits.append(valeurs[i]*effectifs[i])` ?
4. Saisir, exécuter ce programme puis le tester en saisissant dans la console à droite :
   moyenne`([6,7,8,9,10,11,12,13,14,15,16,17],[3,1,2,1,2,5,3,3,3,2,1,1])`.

Code Python :

```python
def moyenne(valeurs,effectifs):
    """calcule la moyenne pondérée d'une série, affectée de coefficients"""
    produits=[]
    for i in range(len(valeurs)):
        produits.append(valeurs[i]*effectifs[i])
    m=sum(produits)/sum(effectifs)
    return m
```

5. Appliquer la fonction `moyenne` à la liste des notes qu'Amélie a eu au cours du trimestre en mathématiques :

Note	12	8	14	11	6,5
coefficient	1	2	2	1	3

6. Déterminer la plus petite note (à 0,5 point près) qu'elle aurait dû avoir au dernier contrôle pour obtenir une moyenne supérieure ou égale à 13.
7. La fonction `seuil(valeurs,effectifs,objectif)` recherche la valeur minimale de la dernière note (élément d'indice 4 de la liste `valeurs` : `valeurs[4]`, à 0,5 point près) pour que la moyenne trimestrielle atteigne ou dépasse la note `objectif` :

Code Python :

```
8  def seuil(valeurs,effectifs,objectif):
9      while moyenne(valeurs,effectifs)................:
10         valeurs[4]=..............
11     return valeurs[4],moyenne(valeurs,effectifs)
12
```

Compléter cette fonction et tester la avec un objectif de 13 : `seuil([12,8,14,11,6.5],[1,2,2,1,3],13)`

8. Reprendre les mêmes notes initiales avec cette fois-ci, un objectif de 17. Que se passe-t-il ? Que pourrait-on rajouter à la condition d'arrêt pour que la 5ème ne dépasse pas 20 ?

Les boucles

Comme dans la plupart des langages, il existe en Python principalement deux manières de réaliser une boucle, c'est-à-dire une répétition d'un bloc d'instructions. Comme pour la structure **`si`**, la partie à répéter sera indentée vers la droite, ce qui permet en plus une bonne visibilité de l'algorithme.

- **Boucle bornée avec compteur POUR :** La boucle POUR est utilisée lorsque l'on souhaite répéter plusieurs fois une même instruction. Il faut connaitre le nombre de répétitions (itérations) souhaité de cette instruction.
  On aura besoin d'une variable permettant de compter le nombre de fois que l'on répète l'instruction. En Python, on indique les valeurs que doit prendre cette variable "compteur" dans une liste.
  L'instruction `for var in range(debut,fin,pas) :` réalise une boucle en faisant parcourir à la variable *var* toute la liste **`range()`**.
  Les paramètres *debut* et *pas* sont optionnels. Cela permet de réaliser une boucle en faisant parcourir à la variable *var* :
  - ➢ tous les entiers de l'intervalle `[0;fin[` si un seul paramètre est renseigné : **`range`**`(fin)`
  - ➢ tous les entiers de l'intervalle `[debut;fin[` si on a deux paramètres : **`range`**`(debut,fin)`
  - ➢ les entiers de l'intervalle `[debut;fin[` situés à une distance *pas* l'un de l'autre si trois paramètres sont renseignés : **`range`**`(debut,fin,pas)`

  On peut aussi faire parcourir à la variable *var* tous les éléments d'une liste quelconque en utilisant la syntaxe `for var in liste :`
- **boucle non bornée TANT QUE :** On utilise généralement la boucle TANT QUE lorsqu'on souhaite répéter un nombre de fois une même instruction et qu'on ne sait pas combien de fois cette instruction va être répétée.
  On connait alors une condition d'arrêt, c'est-à-dire un test qui permet de savoir si l'instruction va être répétée ou non.
  L'instruction `while condition :` exécute une instruction ou un bloc d'instructions tant que la *condition* est vérifiée. (La boucle peut donc ne jamais être exécutée si, d'entrée la *condition* n'est pas remplie).
  - ➢ il est possible d'utiliser un compteur `i`dans une boucle while mais il faut l'initialiser auparavant `i=`*debut* et l'incrémenter à l'intérieur des instructions indentées (`i=i+`*pas*).
  - ➢ certaines fois, la condition est toujours vérifiée et la boucle tourne à l'infini. Dans ce cas, vous pouvez interrompre le programme en tapant simultanément Ctrl + F2 ou Ctrl + I .

5 Applications

5.1 Médiane et quartiles d'une série

On donne l'algorithme décrivant la détermination de la médiane d'une série de valeurs :

def mediane*(liste)*:
    """cette fonction détermine la médiane d'une série donnée sous forme d'une liste"""
    Trier la *liste* par ordre croissant
    $n \leftarrow$ nombre d'éléments de la *liste*
    si n est impair **:**
        renvoie le terme de la *liste* d'indice (n+1)/2
    sinon :
        renvoie la moyenne entre les termes d'indices centraux

Code Python :

```python
def mediane(liste):
    """cette fonction détermine la médiane d'une série donnée sous forme d'une liste"""
    ...
    ...
    if ... :
        return ...
    else:
        return ...

```

1. Ouvrir le fichier `mediane_vide.py`, le compléter et tester la fonction `mediane` avec les deux listes proposées.
2. Dans ce même fichier, définir une fonction `Q1` qui donnera le premier quartile de la liste puis le tester.
3. Même question avec une fonction `Q3` qui donnera le troisième quartile d'une liste.

5.2 Pente d'une droite et parallélisme

1. On souhaite programmer une fonction `pente(xA,xB,yA,yB)` qui calcule la pente d'une droite (non parallèle à l'axe des ordonnées) passant par les points $A(x_A\,;\,y_A)$ et $B(x_B\,;\,y_B)$.
   a. Compléter le script Python permettant le calcul de la pente :

Code Python :

```python
def pente(xA,yA,xB,yB):
    """cette fonction détermine la pente d'une droite (AB) dans une repère"""
    return ...

```

   b. Tester la fonction `pente` en calculant la pente de (AB) avec $A(-1\,;\,4)$ et $B(7\,;\,-2)$.
   c. Essayer d'améliorer la fonction précédente en proposant un message d'erreur si la droite est parallèle à l'axe des ordonnées. Tester la fonction avec (AB) puis (CD) où $C(-0{,}5\,;\,4)$ et $D(-0{,}5\,;\,11)$.

Code Python :

```python
def pente(xA,yA,xB,yB):
    """cette fonction détermine la pente d'une droite (AB) dans une repère"""
    if ............ :
        return ...
    else :
        print(.........................)
```

2. On souhaite désormais construire une fonction `parallele(xA,yA,xB,yB,xC,yC,xD,yD)` qui teste si deux droites (AB) et (CD) sont parallèles.

   a. Utiliser la fonction `pente` pour définir la fonction `parallele` :

Code Python :

```python
def parallele(xA,yA,xB,yB,xC,yC,xD,yD):
    if ..........== .......... :
        return True
    else :
        return False
```

   b. Tester la fonction avec (AB) ($A(-1\,;\,4)$ et $B(7\,;\,-2)$) et (CD)($C(5\,;\,2)$ et $B(-11\,;\,14)$.

3. On souhaite pour finir construire une fonction `parallelogramme(xA,yA,xB,yB,xC,yC,xD,yD)` qui teste si un quadrilatère $ABCD$ est un parallélogramme ou non.

   a. Utiliser la fonction `parallele` pour définir la fonction `parallelogramme` :

Code Python :

```python
def parallelogramme(xA,yA,xB,yB,xC,yC,xD,yD):
    if ..........== .......... and ..........== .......... :
        return True
    else :
        return False
```

   b. Tester la fonction avec `parallelogramme(-1.5,-3,4,-1,5.5,4,0,2)`.

5.3 Milieu d'un segment

1. Dans le même fichier, construire une fonction `milieu(xA,yA,xB,yB)` qui calcule les coordonnées du milieu d'un segment $[AB]$. *Indication : renvoyer une liste de deux éléments `[...,...]`.*
2. Redéfinir une fonction `parallelogramme2(xA,yA,xB,yB,xC,yC,xD,yD)` qui teste la propriété de parallélogramme en faisant appel à la fonction `distance`.

5.4 Distance dans un repère orthonormé

1. Dans le même fichier, construire une fonction `distance(xA,yA,xB,yB)` qui calcule la longueur du segment $[AB]$ (dans un repère orthonormé).
2. Redéfinir une fonction `parallelogramme2(xA,yA,xB,yB,xC,yC,xD,yD)` qui teste la propriété de parallélogramme en faisant appel à la fonction `milieu`.

Opérateurs mathématiques et logiques

Opérateurs mathématiques		Opérateurs logiques	
Symbole	Signification	Opérateur	Signification
`+,-,*,/`	opérations arithmétiques	A **and** B	renvoie VRAI si les conditions A **et** B sont vérifiées.
`a//b`	quotient entier de a par b.	A **or** B	renvoie VRAI si au moins une des conditions A **ou** B est vérifiée.
`a%b`	reste entier dans la division de a par b.	A^B	renvoie VRAI si une seule des deux conditions est vérifiée.
`a**b`	calcule a^b. (équivalent `pow(a,b)`)		

5.5 Python et les courbes de fonctions

5.6 Utilisation de Matplotlib

Pour tracer des courbes de fonctions, **Python** seul n'est pas suffisant et nous avons besoin des bibliothèques (ou modules) **numpy**, **matplotlib**, et **math**. Pour les utiliser dans un script, il faut les importer en les renommant éventuellement à l'aide d'un alias :

Code Python :

```python
from math import * # on importe toutes les fonctions du module math
import numpy as np # utilisation de l'alias np pour appeler une fonction du module numpy
import matplotlib.pyplot as plt # on importe l'interface pyplot du module matplotlib avec le surnom (alias) plt
```

Le tracé d'une courbe dans Python se fait à l'aide de l'instruction **plot()**. Son rôle consiste à placer des points dont les coordonnées sont dans un tableau et à relier ces points par des segments.

1. Ouvrir le fichier `courbe_1.py` et l'exécuter. Compléter ensuite la copie du script en expliquant le rôle de chaque ligne d'instruction (voir les premiers commentaires) :

Code Python :

```python
from math import *
import numpy as np
import matplotlib.pyplot as plt
def f(x):
    return 8*sqrt(x**2+1)-x**2-7 #définit une fonction mathématique
antecedents=np.linspace(-8,10,100)# définit une liste de 100 valeurs d'antécédents, régulièrement répartis entre -8 et 10.
images=[f(t) for t in antecedents]#..........
plt.plot(antecedents,images)#..........
plt.grid()#..........
plt.title("Courbe de la fonction")#..........
plt.xlabel("abscisses")#..........
plt.ylabel("ordonnées")#..........
plt.show()#..........
```

2. Quelle est l'expression, en notation habituelle, de la fonction f tracée ? $f(x) = \ldots\ldots\ldots\ldots$
3. Modifier le script pour qu'il affiche successivement les courbes des fonctions suivantes :
   - $g(x) = x^2 - x + 2x \in [-4\,;\,7]$;
   - $h(x) = 0{,}5x^3 - 2x^2 - x + 4$ pour $x \in [-10\,;\,15]$;
   - $k(x) = \dfrac{x^2 + 5x - 3}{x^2 + 1}$ pour $x \in [-5\,;\,6]$;

5.7 Longueur d'un arc de courbe

On considère une fonction f définie sur un intervalle $[a\,;\,b]$ et on cherche une valeur approchée de la longueur ℓ de la courbe représentative de f.

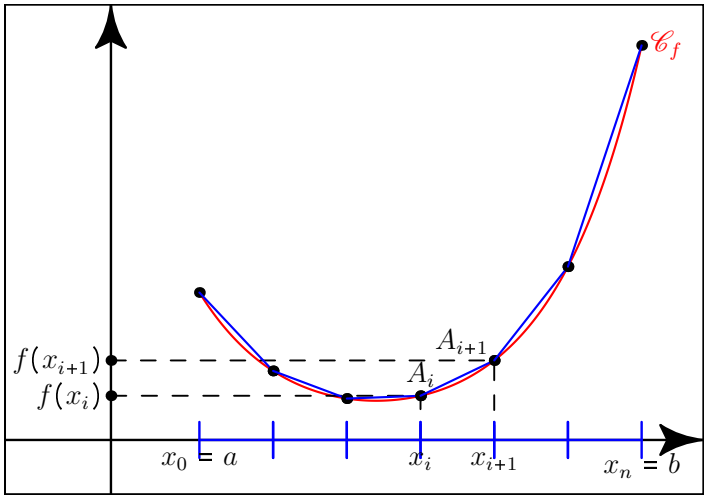
Méthode : on subdivise (on partage) l'intervalle $[a\,;\,b]$ en n petits intervalles $[x_i\,;\,x_{i+1}]$ de même amplitude $x_{i+1} - x_i = \dfrac{b-a}{n}$ avec $x_0 = a$, $x_n = b$. On note A_i le point de coordonnée $(x_i\,;\,f(x_i))$ de la courbe.

On approche alors sur $[x_i\,;\,x_{i+1}]$ la courbe par le segment $[A_i\,;\,A_{i+1}]$.

On prend alors comme valeur approchée de ℓ la somme des longueurs de ces segments.

On va donc réutiliser la fonction `distance(xA,yA,xB,yB)` vue dans les scripts précédents et qui calcule la longueur d'un segment $[AB]$ (dans un repère orthonormé).

```
def longueurcourbe(f,a,b,n):
    long ← 0
    xA ← a
    yA ← f(a)
    h ← (b-a)/n
    pour i allant de 0 à n − 1 :
        xB ← xA+h
        yB ← f(xB)
        long ← long+distance(xA,yA,xB,yB)
        xA ← xB
        yA ← yB
    renvoie long
```



1. Ouvrez le fichier `longueur_courbe.py` et complétez la fonction `longueurcourbe(f,a,b,n)` en vous inspirant de l'algorithme ci-dessus.
2. Testez cette fonction sur la fonction `8*sqrt(x**2+1)-x**2-7` entre 1 et 6, avec une subdivision de 10 puis de 100 (on utilise une fonction **`lambda`**) :

Code Python :

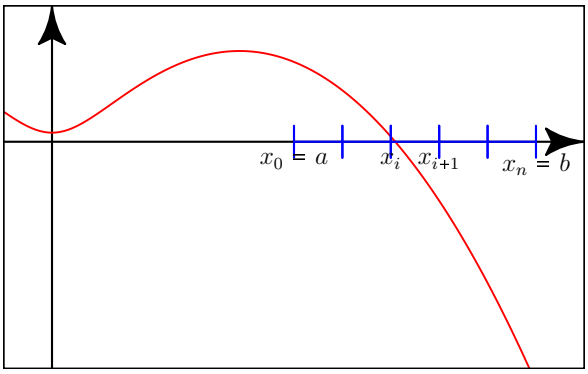
```
>>> longueurcourbe(lambda x:8*sqrt(x**2+1)-x**2-7,1,6,10)
```

3. Calculer la longueur des arcs de courbe suivants :
   - $h(x) = 0{,}5x^3 - 2x^2 - x + 4$ pour $x \in [-10\,;\,15]$;
   - $k(x) = \dfrac{x^2+5x-3}{x^2+1}$ pour $x \in [-5\,;\,6]$;

5.8 Résolution d'une équation par balayage

1. Reprendre le script `courbe_1.py` et déterminer un encadrement des solutions de l'équation $f(x) = 0$.
2. On va créer une fonction `tableau(f,a,b,n)` qui renvoie un tableau de valeurs, c'est-à-dire une liste de couples $(x,f(x))$ pour x parcourant une subdivision de l'intervalle $[a\,;\,b]$: l'intervalle $[a\,;\,b]$ est de nouveau partagé en intervalles $[x_i\,;\,x_{i+1}]$ de même amplitude $x_{i+1} - x_i = \dfrac{b-a}{n} = h$ avec $x_0 = a$, $x_n = b$.
   a. Traduire l'algorithme ci-dessous en une fonction Python :

```
def tableau(f,a,b,n):
    h ← (b-a)/n
    L est une liste qui reçoit les valeurs
      de la subdivision a+i*h, pour i
      variant entre 0 et n
    fL est une liste qui reçoit les couples
      (x,f(x)) pour x parcourant la liste L
      renvoie fL
```



: on peut aussi reprendre la fonction `longueurcourbe` et la remanier...

   b. Tester cette fonction avec la fonction du script :

Code Python :

```
>>> tableau(lambda x:f(x),5,10,5)
```

3. Réutiliser la fonction pour trouver un encadrement de cette solution au dixième, puis au centième et enfin au millième.