

Dans cette feuille, l'objectif est :

- ★ de voir quelques principes de base de l'algorithmique, dont certains sont applicables sur le programme de seconde ;
- ★ d'appliquer quelques algorithmes en utilisant le langage Python (version 2.6).

## I Présentation et premiers pas.

### 1 L'interface graphique.

L'interface graphique de Python se compose de deux parties :

- ✓ Une fenêtre appelée **Python Shell**, c'est dans cette fenêtre que Python s'exécute. Une ligne commence par `>>>` , c'est à la suite de ces trois signes qu'on peut écrire des commandes pour une utilisation immédiate, un peu comme sur une calculatrice avec l'écran de calcul.
- ✓ Un éditeur de programme. Il suffit, pour l'appeler, de demander une nouvelle fenêtre : **File, New window**.

Pour la suite de ce paragraphe, nous utiliserons uniquement la fenêtre **Python Shell**. Nous écrirons des programmes par la suite.

### 2 Les opérations algébriques.

Instruction	Opération
<code>&gt;&gt;&gt; 7+5.789</code> 12.789	l'addition
<code>&gt;&gt;&gt; 8.89-9.999</code> -1.109	la soustraction
<code>&gt;&gt;&gt; 7*3</code> 21	la multiplication
<code>&gt;&gt;&gt; 45/7</code> 6	le quotient de la division euclidienne de deux nombres entiers
<code>&gt;&gt;&gt; 45.36/9.655</code> 4.698083894355257	le quotient de deux nombres « réels »
<code>&gt;&gt;&gt; 45%7</code> 3	le reste de la division « euclidienne » de deux nombres, c'est l'opérateur modulo
<code>&gt;&gt;&gt; 6**2</code> 36	la puissance est symbolisée par deux signes <b>**</b>

Vous pouvez utiliser directement ces commandes dans la fenêtre **Python Shell**, pour voir ce que ça donne. En particulier, on peut essayer l'opérateur `%` avec des nombres « à virgule ».

### 3 Les types de nombres, l'arrondi.

Vous avez vu sur l'exemple de la division que **Python** connaît deux types de nombres. Il y a les entiers et les nombres en virgule flottante qui sont donnés avec le point décimal. On peut connaître le type d'un nombre avec la commande **type** :

```

>>> type(3.1415)
<type 'float'>
>>> type(6)
<type 'int'>
>>> type(6.0)
<type 'float'>
```

Vous pouvez essayer là aussi directement dans la fenêtre **Python Shell**. Essayer même avec un nombre grand nombre entier.

**Remarque :** les nombres en virgule flottante ont parfois un comportement auquel nous ne sommes pas habitués. Par exemple :

```
>>> 9.4142%3.557
2.3001999999999994
```

On peut regarder ce que ça donne avec la calculette ... Si on demande d'afficher le résultat :

```
>>> print(9.4142%3.557)
2.3002
```

Au passage, on note que la commande **print** permet l'affichage.

#### 4 Variables et affectation.

Pour affecter une valeur à une variable, il suffit d'utiliser le signe « = ».

```
>>> variable = 3.4142
>>> variable
3.4142000000000001
>>> print(variable)
3.4142
```

LE SIGNE « = » N'EST DONC PAS UN SIGNE D'ÉGALITÉ POUR  
PYTHON MAIS UN SIGNE D'AFFECTION.

Vous pouvez faire vous même quelques affectations et des opérations, utiliser les commandes **print** et **type** avec ces variables. Par exemple «définir le nombre  $\pi$ », donner un rayon et calculer le périmètre d'un cercle, l'aire d'un disque. Ou bien l'aire d'un carré, comme il vous plaira.

**Remarque :** on peut affecter une nouvelle valeur à la même variable, il s'agit alors d'une réaffectation :

```
>>> a = 7
>>> print(a)
7
>>> a=2*a
>>> print(a)
14
```

Que se passe-t-il avec  $2*a=a$  ?

#### 5 Fonctions mathématiques.

Les fonctions mathématiques ne sont pas comprises dans le Python « de base ». Pour les utiliser, il faut **importer** le module de mathématique, avec ses fonctions prédéfinies.

```
>>> from math import *
>>> sqrt(4)
2.0
>>> sin(pi/6)
0.49999999999999994
>>> log(2)
0.69314718055994529
>>> log(e**2)
2.0
```

## II Premiers programmes en Python.

Dans cette partie comme dans les suivantes, nous allons utiliser la seconde fenêtre de **Python**, celle qui sert à écrire des programmes.

### 1 Préparer l'écriture.

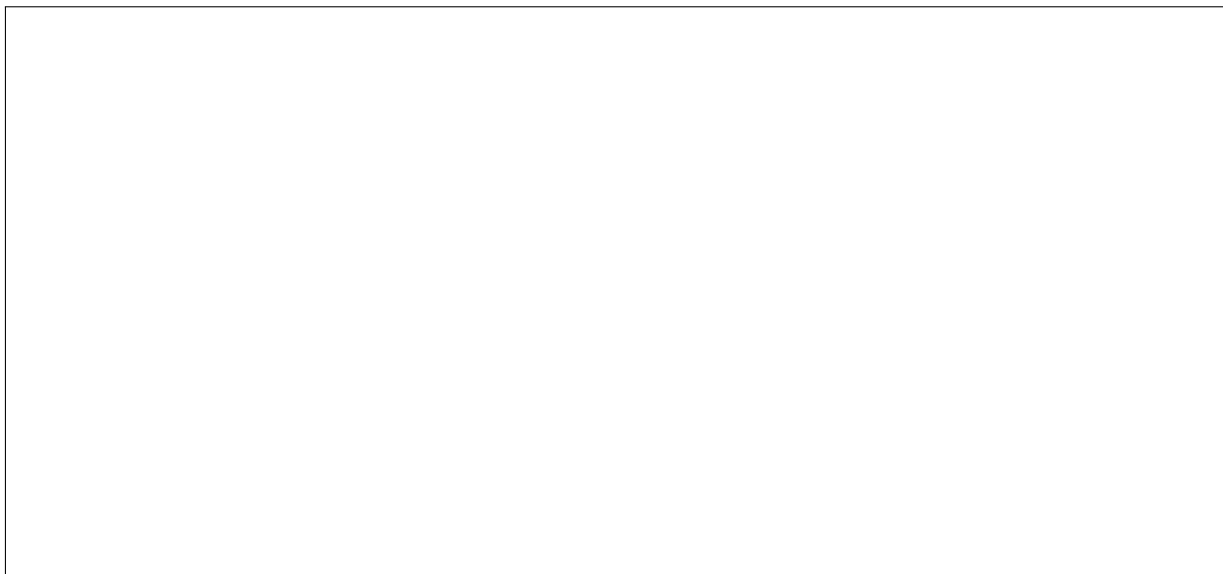
- Il faut donc, si ce n'est pas fait, commencer par demander une nouvelle fenêtre : **Files, New window**.
- Ensuite, il faut enregistrer ce qui sera le premier programme : **File, Save As...** . Vous pouvez donner le nom qui vous convient, sans espace, sans accents, et finissant par **.py**, c'est l'extension de nom de fichier pour les programmes écrits en Python.
- la toute première ligne (obligatoire) à écrire est `# -*- coding :Latin-1 -*-` . Cette ligne indique le codage des caractères (ici avec les accents français) et n'a aucun rapport avec les algorithmes.

### 2 La commande input.

la commande `input` sert à demander une variable, c'est une entrée.

Nous allons écrire un programme qui :

- ▶ demande un nombre (avec la commande `input`), la syntaxe est `a=input()` ;
- ▶ demande un second nombre ;
- ▶ affiche la somme la différence, le produit de ces nombres.



Quand le programme est écrit, on l'enregistre et on le lance en utilisant la touche **F5**. Il s'exécute dans l'autre fenêtre, **Python Shell**. Vous voyez le curseur à la ligne, il attend la valeur de la première variable.

```
>>> ===== RESTART =====  
>>>  
|
```

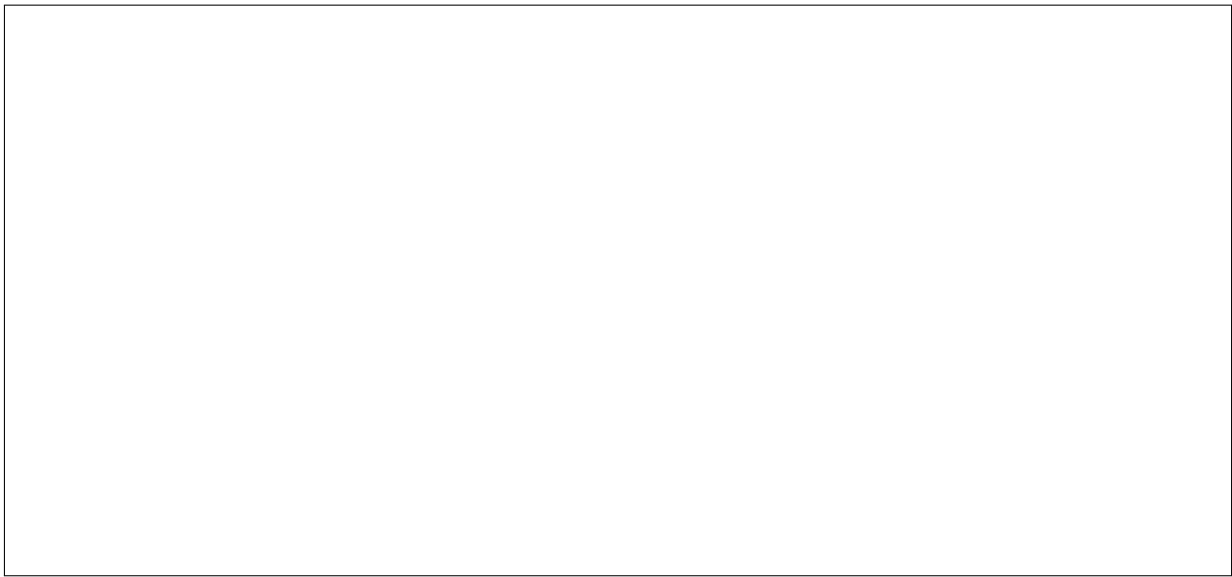
#### Remarque :

Il est possible d'améliorer la «présentation» des questions et des réponses. Par exemple :

```
a=input('entrer un nombre a :')  
print 'la somme des nombres ',a,' et ',b,' est : ', a+b
```

#### Remarque : que se passerait-il si on voulait calculer le quotient des nombres ?

**Exercice 1.** Écrire de même un programme qui demande un nombre  $a$ , puis affiche le nombre  $e^a$ , le nombre  $\sin(a)$ , le nombre  $f(a) = a^2 - 5a + 6$ . Utiliser la remarque précédente pour l'affichage, penser à importer le module mathématique.



### 3 Avec une condition.

Dans l'exercice précédent, nous avons utilisé des fonctions sinus et exponentielles qui sont définies sur  $\mathbb{R}$ . Prenons l'exemple de la fonction racine.

- Il s'agit donc de faire calculer, pour un nombre réel positif,  $\sqrt{a}$ . Quand le nombre donné en entrée est strictement négatif, il faut alors indiquer que le calcul n'est pas possible.
- En **Python**, ce sont les instructions **if** et **else** qui vont se charger de ce travail, sur le modèle suivant :

en langage courant	avec Python
si $a$ est un nombre positif ou nul afficher la racine carrée de $a$	<code>if a&gt;=0 :</code>  <code>    print .....</code>
sinon, afficher une phrase pour indiquer que le calcul n'est pas possible	<code>else :</code>  <code>    print .....</code>

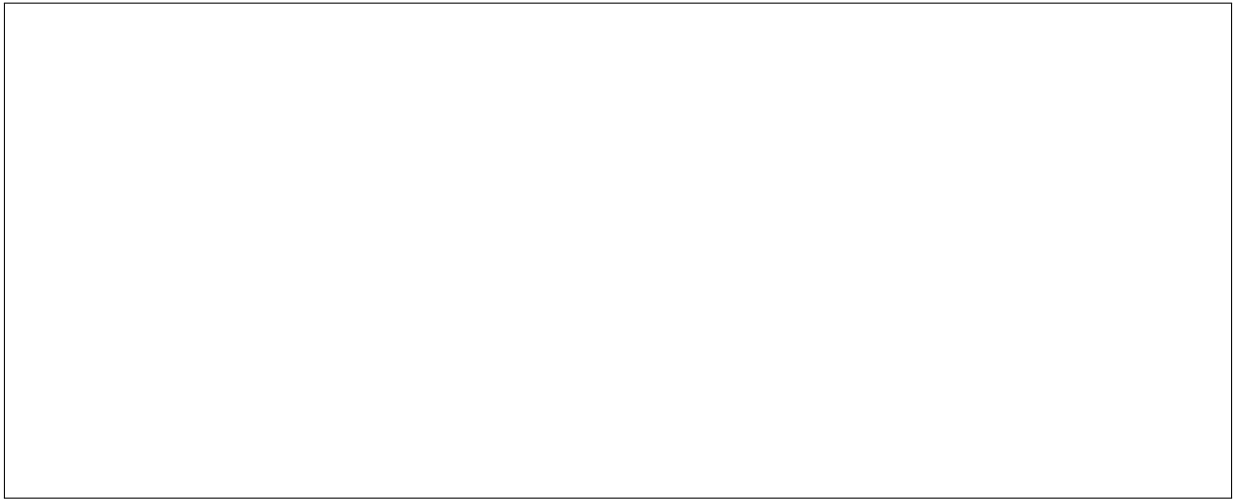
On note deux caractéristiques importantes de **Python** :

- ▶ les deux instructions **if** et **else** se terminent par deux points, « : » ;
- ▶ les lignes qui sont concernées par les instructions **if** et **else** sont en retrait (on dit indentées).

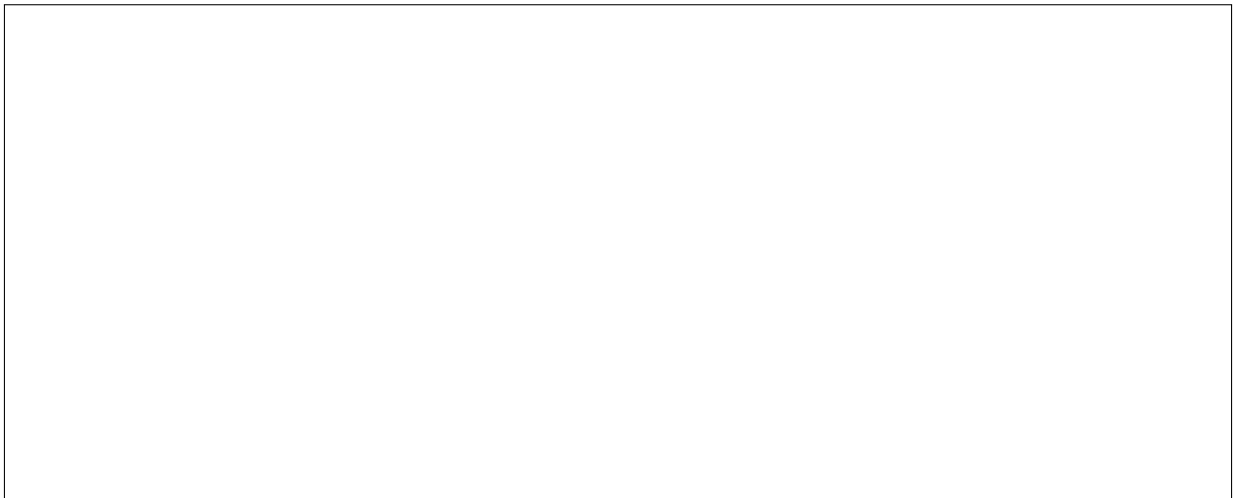
Cette dernière propriété sera utile pour avoir des *programmes lisible, faciles à corriger*.

**Exercice 2.** On veut maintenant faire afficher simultanément, pour un nombre réel  $a$  donné, et quand c'est possible, le logarithme népérien et la racine carrée d'un nombre  $a$ . Il s'agit de modifier le programme précédent pour que ce soit possible.

- ▶ Il faut pour cela utiliser l'instruction **elif**, qui est une abréviation de **else if** (sinon, si . . .). On la place entre **if** et **else**, avec la même syntaxe (sans oublier l'indentation) que **if**.
- ▶ On a vu que le symbole « = » désigne une affectation. **L'égalité dans un test** est donnée par le symbole « == » (deux fois le signe égal).



**Exercice 3.** Faire un petit programme Pythagore qui demande trois longueurs de côtés et qui indique si le triangle est rectangle ou pas (ici, on ne sera pas trop ambitieux et on indiquera quel est le plus grand côté) ou bien qui résout une équation du second degré.



#### Les différents symboles utilisés dans les tests

Symbole	signification
==	égal à
!=	différent de
<	inférieur strictement à
>	supérieur strictement à
<=	inférieur ou égal à
>=	supérieur ou égal à

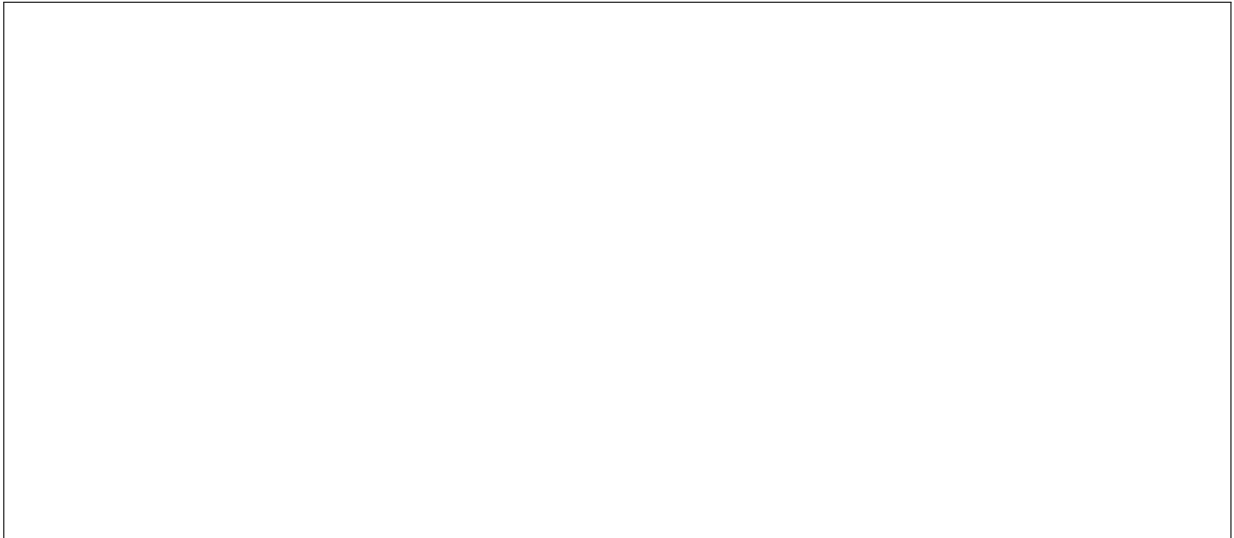
### III Les boucles dans Python.

#### 1 Les instructions conditionnelles dans Python, boucle while.

##### a. Exemple : fabriquer un compteur.

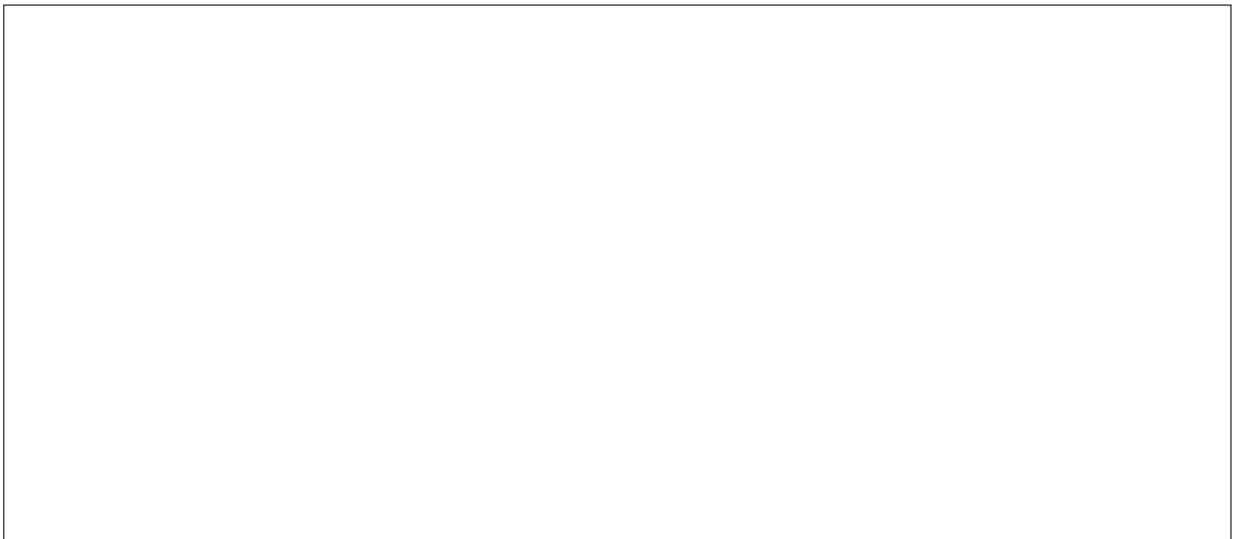
1. Il s'agit de demander l'affichage des nombres de 0 à 20. Pour cela, on utilise l'instruction `while`, (tant que).

en langage courant	instructions de Python
L'initialisation, on commence par 0	<code>k=0</code>
tant que <code>k</code> est inférieur ou égal à 20	<code>while k &lt;= 20 :</code>
on écrit le nombre <code>k</code>	<code>    print k</code>
on incrémente, c'est à dire qu'on affecte <code>k+1</code> à <code>k</code>	<code>    k=k+1</code>

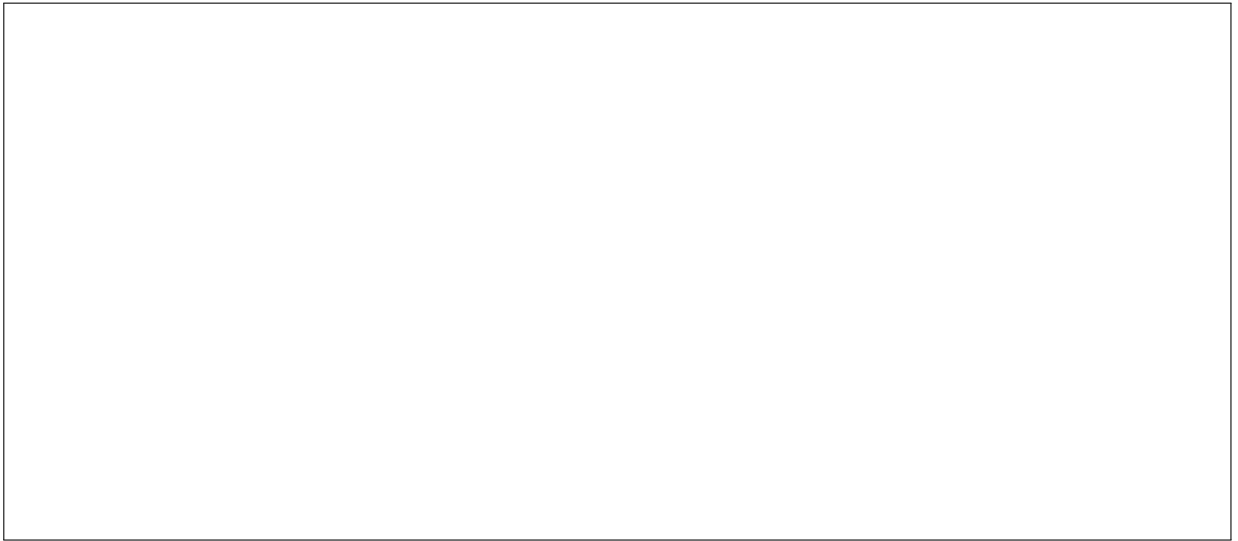


On remarque la même syntaxe que pour la commande `if`, avec les deux points et l'indentation obligatoire.

2. On complique un peu, faire un programme qui demande le début et la fin du compteur, puis qui affiche tous les nombres entiers entre ces deux là.

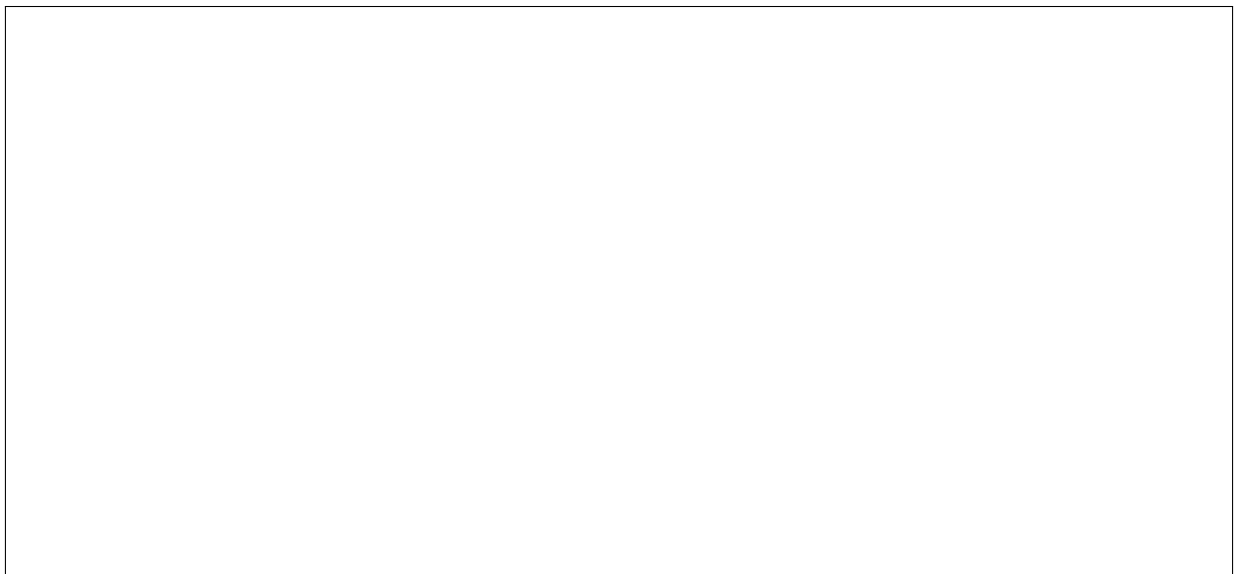


3. On peut aussi prévoir un test qui vérifie que le début est bien plus petit que la fin.
4. On peut aussi changer le pas (compter de deux en deux, tous les dixième, etc.). Il faut alors prévoir de demander le pas.

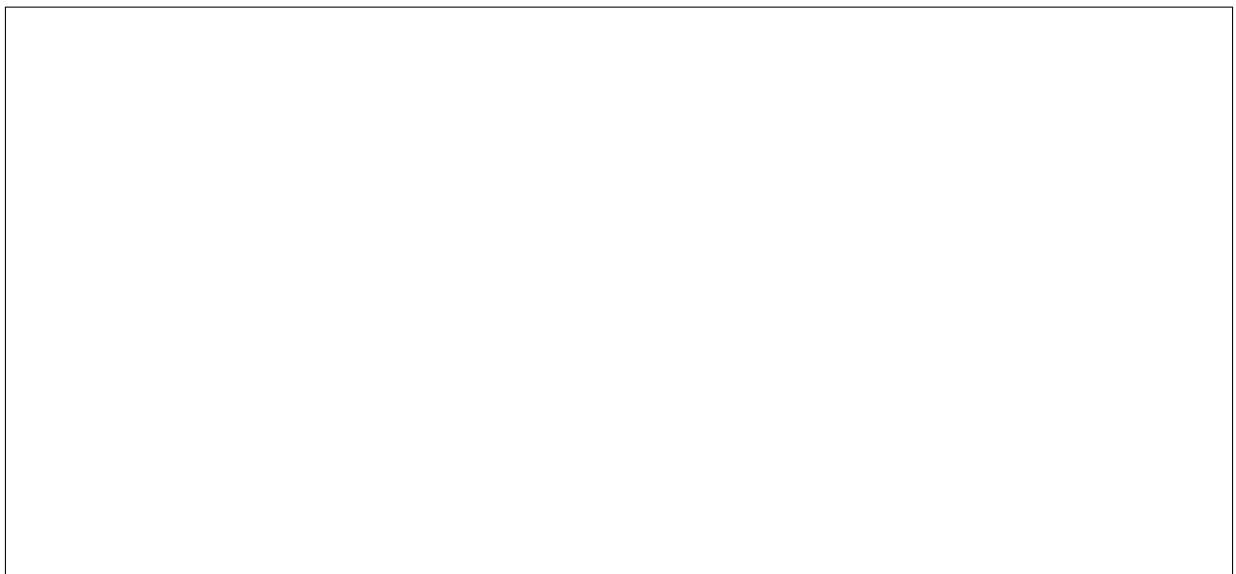


**b. Écrire la table d'une fonction.**

Modifier un des programmes précédents pour afficher la table d'une fonction avec un pas donné.



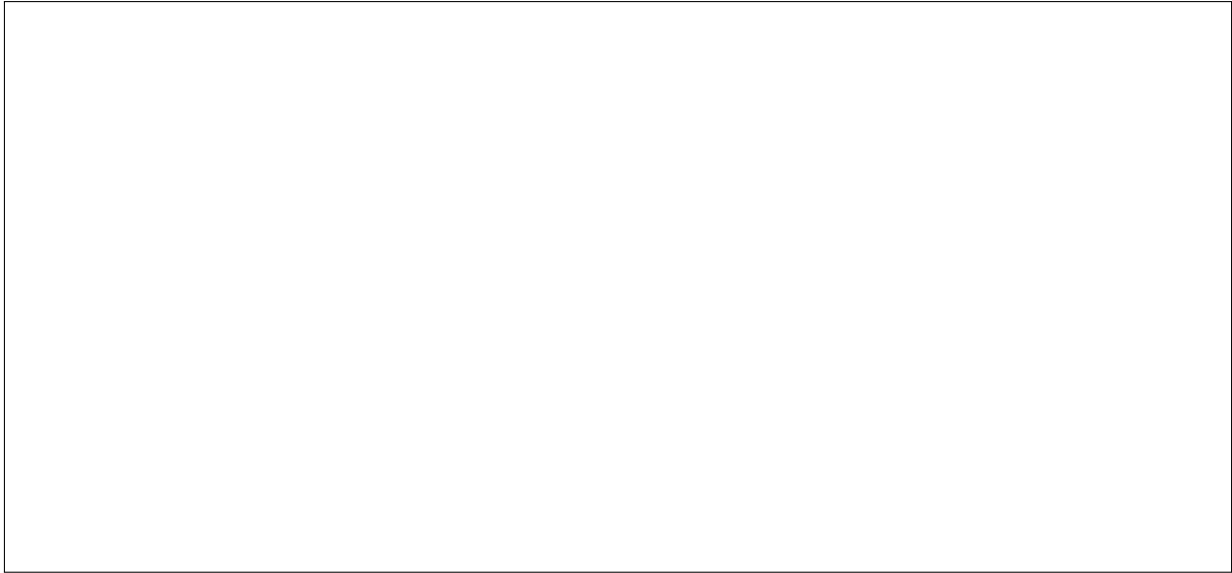
De même, en combinant les instructions `if` et `while`, donner un tableau de valeurs pour une fonction définie par morceaux .



### c. La méthode de balayage.

On prend une fonction continue qui change de signe sur un intervalle fermé de la forme  $[a,b]$  et tel que  $f(a) \times f(b) < 0$ . Par exemple la fonction  $f : x \mapsto x^3 + x - 1$  sur l'intervalle  $[-3;8]$ . On cherche à donner une approximation de la racine avec une précision donnée.

- ★ On part de la borne inférieure  $x_1 = -3$  de l'intervalle et on regarde quand  $f$  change de signe ou s'annule sur les intervalles  $[x_1; \underbrace{x_1 + 1}_{=x_2}]$ ,  $[x_2; x_2 + 1]$ , etc.
- ★ quand  $f$  change de signe, on prend pour nouvelle borne inférieure le dernier  $x_1$  obtenu, on divise le pas par 10 et on recommence, jusqu'à ce que le pas soit égal à la précision.
- ★ Il reste à afficher le résultat.



### d. Une fonction est-elle monotone sur un intervalle borné ?

C'est un des item du document d'accompagnement du ministère (page 23).

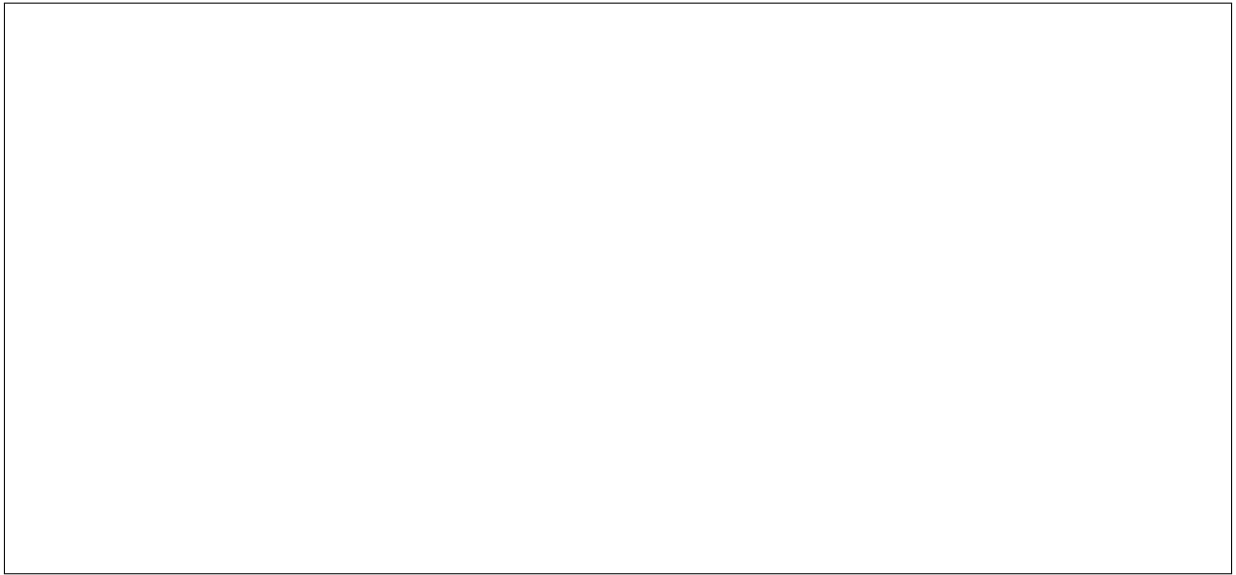
Prenons une fonction, par exemple  $f : x \mapsto x^2 + x - 1$ . Définie sur un intervalle fermé  $I$  borné (par exemple  $[-5; 8]$ , ou  $[2; 7]$ ). Le problème est le suivant :

- ★ on découpe l'intervalle  $I$  par une subdivision  $x_1, x_2$ , etc  $\dots$  ;
- ★ on regarde si les images successives sont rangées dans le même ordre ;
- ★ si c'est le cas, on indique que la fonction  $f$  semble monotone sur  $I$ , sinon, qu'elle n'est pas monotone sur  $I$ .

1. Il s'agit de voir si la fonction change de sens de variation ou pas. Il faut donc prévoir de calculer et conserver le sens de variation «initial», c'est à dire pour le premier intervalle de la subdivision
2. Il faudra, dans une boucle, changer d'intervalle dans la subdivision, déterminer le sens de variation pour cet intervalle et de la comparer au sens de variation initial.
3. Il faut livrer la conclusion à la fin.

**Remarque :** le sens de variation peut être symbolisé par un nombre (-1 ou 1 par exemple), mais aussi par une chaîne de caractères. (croissante ou décroissante par exemple).





**Remarque importante :** le langage **Python** possède la notion de fonction. Nous pourrions transformer ces derniers programmes pour qu'ils soient plus lisibles, plus facile d'utilisation, ou de ré-utilisation, dans le paragraphe consacré aux fonctions sous **Python**.

## 2 Les boucles sans condition, sans liste pour commencer.

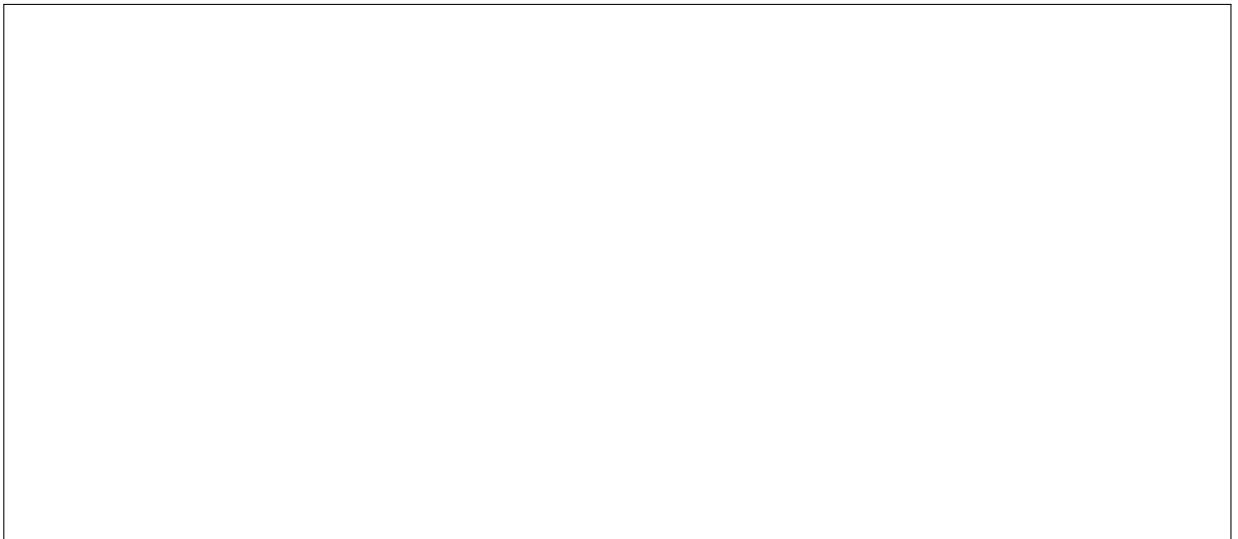
On peut faire l'équivalent d'un compteur en utilisant la commande **for**. Par exemple :

instructions	action
<code>for k in range(20) :</code> <code>print k</code>	affiche les nombres entiers entre 0 et 20 exclu, 20 étant exclu (il y a 20 nombres).
<code>for k in range(20,2) :</code> <code>print k</code>	affiche les nombres entiers pairs entre 0 et 20, c'est à dire avec un pas de 2.
<code>for k in range(5,20,3) :</code> <code>print k</code>	affiche les nombres entiers entre 5 et 20, avec un pas de 3 (20 est toujours exclu).

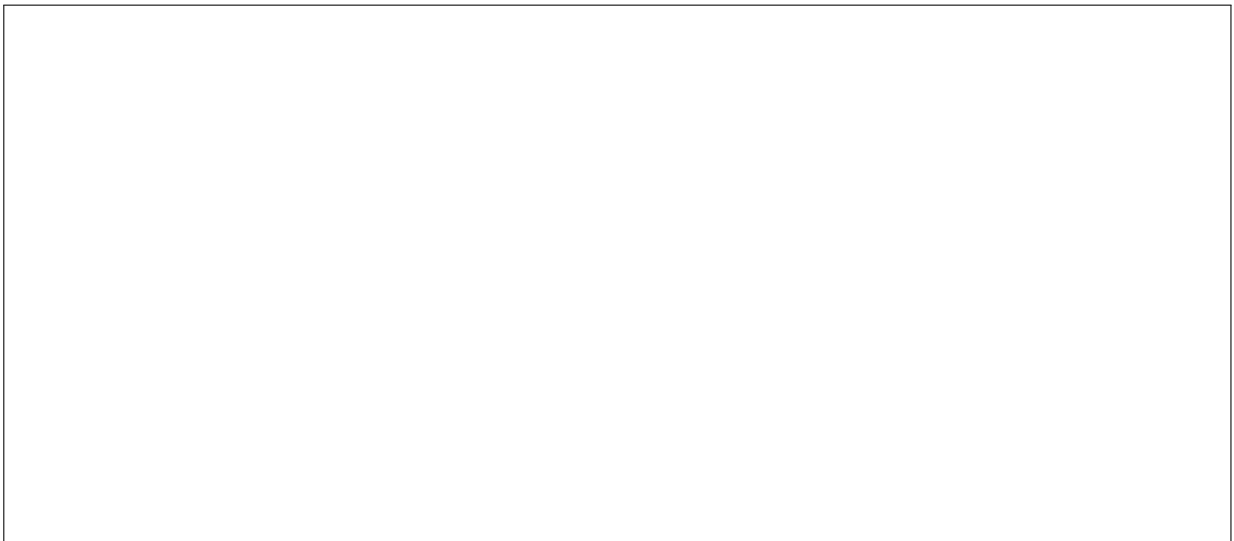
**Remarque :** Les nombres qui sont dans la commande **range** sont tous des nombres entiers (relatifs).

### Exercice 4.

1. Écrire un programme avec une boucle **for** qui donne la liste des nombres entiers entre 0 et 15 et la liste des images de ces nombres par la fonction  $f : x \mapsto 2x^2 - \sqrt{x} - 2$ .



2. Modifier le programme précédent pour afficher les images des nombres entre 2 et 5, avec un pas de 0,1.
3. Modifier ce dernier programme pour pouvoir choisir vous même le pas, la borne inférieure et la borne supérieure de l'affichage.



### 3 Les listes et les boucles avec une liste.

#### a. Listes et premières manipulations.

Exemple :

```
>>> liste=[12,2.36,7,8,'coucou',-3.2658]
>>> liste
[12, 2.3599999999999999, 7, 8, 'coucou', -3.2658]
>>> liste[4]
'coucou'
>>> liste[0]
12
```

On voit sur ce petit exemple la syntaxe qu'il faut utiliser et quelques propriétés des listes en **Python**.

- ★ On déclare une liste avec « = », comme pour les autres variables. Les éléments de la liste sont entre crochets et séparés par des virgules.
- ★ Une liste n'est pas forcément homogène, elle peut contenir les entiers, des flottants, des « strings » (chaînes de caractères), etc ...
- ★ On peut appeler un élément de la liste avec son numéro d'indexage. Attention, le premier élément de la liste est indexé par zéro.

**Remarque :** il y a des fonctions qui permettent de manipuler les listes.

- La fonction `len()` permet de connaître le nombre d'éléments (la longueur) d'une liste. Par exemple avec la liste précédente :

```
>>> len(liste)
6
```

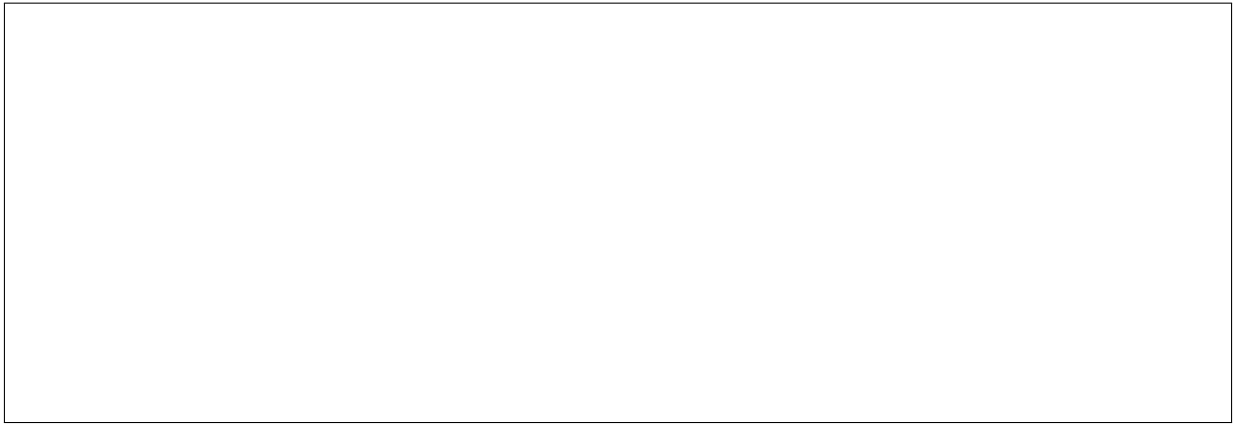
Notre liste possède bien 6 éléments.

- La fonction `del()` permet de supprimer un élément d'une liste :

```
>>> del(liste[2])
>>> liste
[12, 2.3599999999999999, 8, 'coucou', -3.2658]
```

Nous avons supprimé l'élément n°2 de la liste (c'est à dire le troisième élément, 7). Notre liste ne possède plus que 5 éléments.

**Exercice 5.** Écrire un petit programme où une liste est donnée, et où une boucle `for` permet l'affichage de tous les éléments de la liste.



### **b. Les boucles for avec les listes.**

**Remarque :** on peut utiliser les listes directement dans les boucles `for` par exemple :

```
>>> liste_classes=['seconde 10', 'première ES 2','terminale STL 1', 'BTS mava 1']
>>> for classe in liste_classes :
    print classe
```

```
seconde 10
première ES 2
terminale STL 1
BTS mava 1
```

### **c. La fonction append().**

Cette fonction permet d'ajouter un élément à la fin d'une liste. Par exemple :

```
>>> liste=[1,2,3,4,5]
>>> liste.append(6)
>>> liste
[1, 2, 3, 4, 5, 6]
```

Ceci permet, par exemple de fabriquer un à un les termes d'une suite, en commençant par initialiser la liste bien sûr. Commençons par exemple par la suite géométrique de premier terme 2 et de raison  $\frac{1}{2}$ .

```
# -*- coding :Latin-1 -*-
#-----#
suite=[2.0]          #initialisation, attention au nombre qui doit être un flottant
#-----#
for i in range(10) :      # on calcule et on stocke les termes jusqu'à u(10)
n=len(suite)           #n est le rang du terme à calculer
suite.append(suite[n-1]/2)      #on ajoute à la liste le terme suivant
print suite
```

**Exercice 6.** Calculer et exprimer dans une liste les premiers termes de la suite de Fibonacci.

**Exercice 7.** Construire une liste qui contiendra tous les nombres de -3 à 5 avec un pas de 0,1 et une autre qui contiendra toutes les images de ces nombres par la fonction  $f; x \mapsto x^2 - 3x + 2$ . Faire ensuite afficher côte à côte les nombres et leurs images.

**Exercice 8.** Réécrire le programme sur le théorème de Pythagore en utilisant une liste, les longueurs des côtés pouvant être données dans un ordre quelconque. la commande `nomdelaliste.sort()` permettra de trier (du plus petit au plus grand) les éléments de la liste.

**Remarque :** la commande `range` peut aussi être utilisée, avec la même syntaxe que pour les boucles `for` pour générer une liste :

```
>>> liste = range(8)
>>> print liste
[0, 1, 2, 3, 4, 5, 6, 7]
>>> liste = range(-13,9,2)
>>> print liste
[-13, -11, -9, -7, -5, -3, -1, 1, 3, 5, 7]
```

## IV Les fonctions dans Python.

La notion de fonction dans **Python** permet d'alléger la lecture des algorithmes des tâches «répétitives». Prenons l'exemple de l'algorithme qui permet de calculer les coordonnées du milieu d'un segment.

**Exercice 9.** Écrire un petit programme qui demande le nom, l'abscisse et l'ordonnée de deux points, puis qui calcule les coordonnées du milieu de ce segment.



Ici, nous avons besoin de faire deux fois la même chose, demander les deux extrémités du segment. Une alternative est de confier cette tâche répétitive (ici, c'est une petite répétition!) à une fonction. Ici, cette fonction va être la fonction `entrer_point()`.

```
#-----#  
  
# cette fonction sert à entrer un point du plan comme une liste :  
# le premier élément est le nom, les deux autres l'abscisse et l'ordonnée du point  
  
def entrer_point() :  
  
    nom=raw_input( 'Donner le nom du point : ' ) # entrée d'une chaine de caractères  
                                                raw est indispensable.  
  
    point=[nom] # initialisation de la liste du point  
  
    abscisse=input( 'entrer l\'abscisse du point : ' )  
    point.append(float(abscisse)) # on ajoute l'abscisse à la liste du point  
    ordonnee=input( 'entrer l\'ordonnée du point : ' )  
    point.append(float(ordonnee)) # on ajoute l'ordonnée à la liste.  
  
    return point # L'instruction return indique ce qu'il faut transmettre  
                au programme principal  
#-----#
```

### Remarque :

- ★ Noter la définition de la fonction qui commence par la commande `def` suivie du nom de la fonction . Entre parenthèses, il faut indiquer les arguments que doit transmettre le programme principal. Ici, il ne transmet rien, il y a des parenthèses avec rien dedans.
- ★ Tous à la fin, l'instruction `return` indique ce qu'il faut transmettre au programme principal. Ici, la fonction transmet au programme principal la liste avec le nom et les coordonnées, c'est la liste `point`.

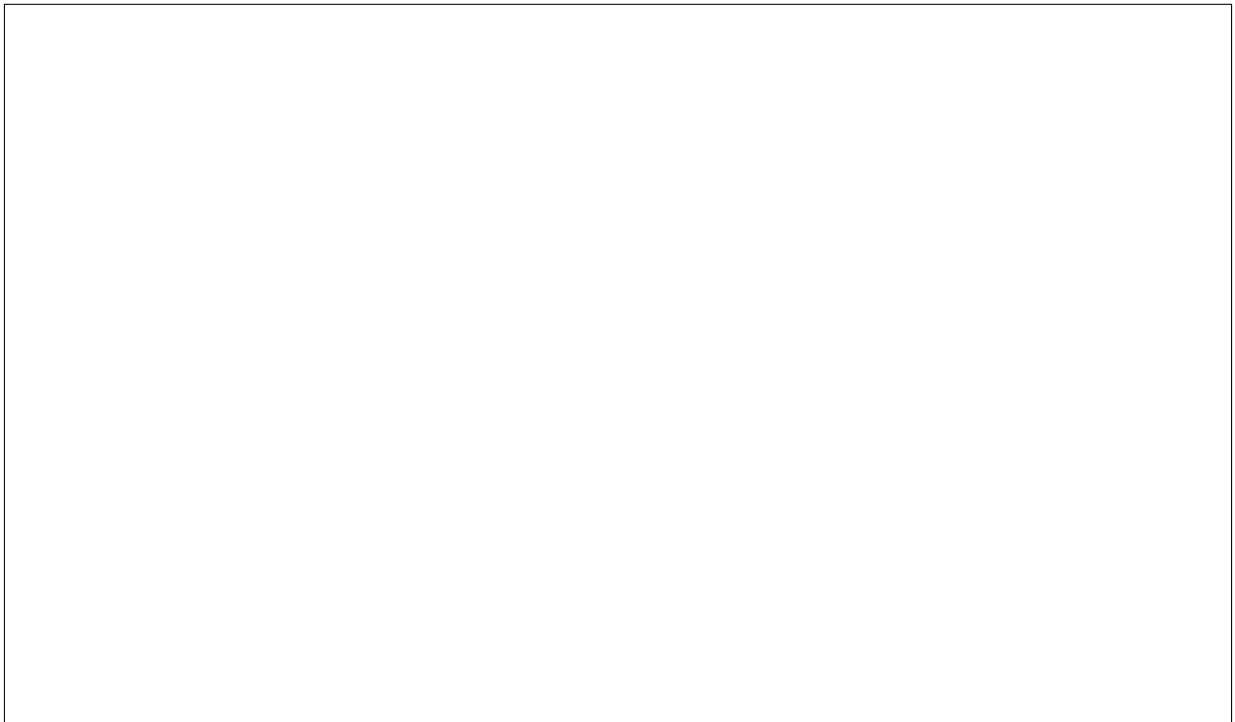
★ La présentation semblable à celle d'une boucle avec les deux points et l'indentation.

**Exercice 10.** Écrire un programme qui calcule, avec une fonction, les coordonnées du milieu d'un segment.



**Exercice 11.** Écrire un programme qui demande quatre points du plan, qui indique si le quadrilatère est un parallélogramme. Le programme comportera deux fonctions :

- ▶ une fonction pour entrer les points ;
- ▶ une fonction pour calculer les milieux.



**Exercice 12.** Écrire un programme qui demande trois points du plan et qui calcule le quatrième point de manière à obtenir un parallélogramme. Toujours avec les deux fonctions.

**Exercice 13.** Que fait ce programme ?

```

# -*- coding :Latin-1 -*-
#-----#
from math import *
#-----#
def f(x) :
    "c\'est la fonction dont on cherche à évaluer la monotonie"
    image=x**2+x-1
    return image
#-----#
    #c\'est l\'initialisation du programme
debut=1
fin=8
pas=1
#-----#
x1=debut
x2=x1+pas
sens_initial=(f(x2) - f(x1))/abs(f(x2) - f(x1))
sens=sens_initial
while sens == sens_initial and x2<fin :
    x1,x2=x2,x2+pas
    if (f(x2) - f(x1)) !=0 :
        sens = (f(x2) - f(x1))/abs(f(x2) - f(x1))
if sens==sens_initial :
    print ('la fonction semble monotone sur l\'intervalle [',debut,',';',fin,']')
else :
    print ('la fonction n\'est pas monotone sur l\'intervalle [',debut,',';',fin,']')

```

Quelle autre fonction aurait-on pu introduire ?

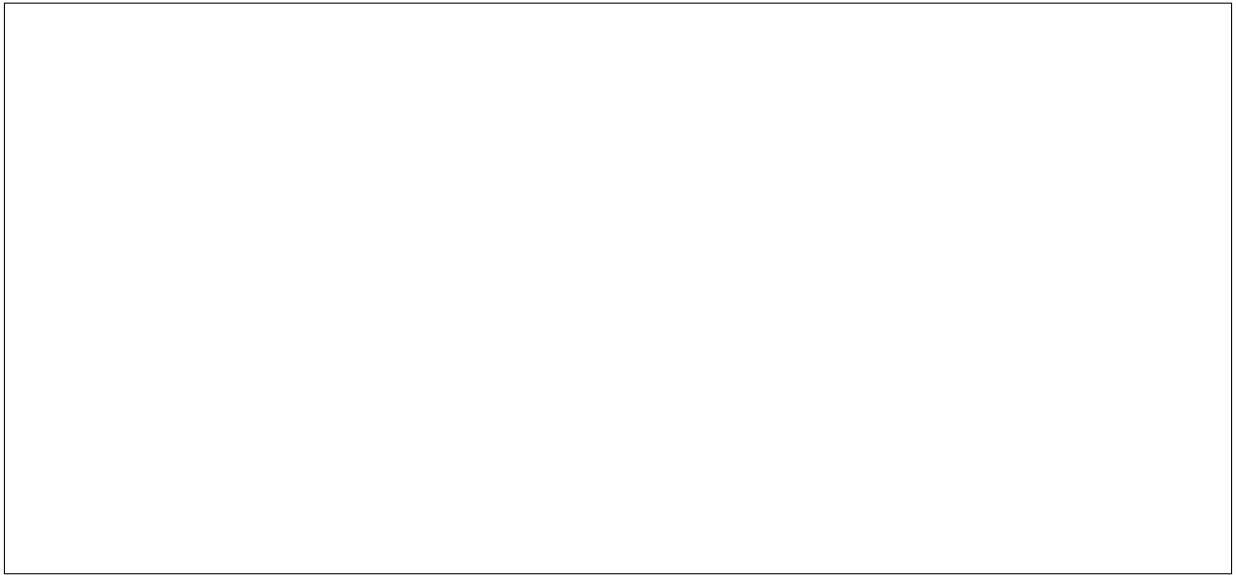
## V Les modules que nous pouvons fabriquer pour Python.

Reprenons l'exemple du parallélogramme (exercices n°10 et n°11). Dans ces deux programmes, les fonctions que nous utilisons sont les mêmes. **Python** permet d'écrire à part ces fonctions, dans un autre fichier. Donc :

- demander une nouvelle fenêtre ;
- copier dans ce fichier les fonctions qui calculent le milieu d'un segment, celle qui fabrique un point à la demande ;
- sauvegarder **dans le dossier où vous travaillez** ce fichier, avec l'extension `.py`, par exemple `geometrie.py`.

Ce faisant, vous venez de créer un nouveau module de **Python**. Vous pouvez l'utiliser comme le module `math`, avec la même syntaxe : `from geometrie import *`.

**Exercice 14.** Refaire un programme qui contrôle si un quadrilatère est ou pas un parallélogramme, en utilisant le module qui vient d'être créé.



**Exercice 15.** Pour l'exercice n°11 (on demande trois points puis on calcule les coordonnées du quatrième pour avoir un parallélogramme) quelle fonction supplémentaire pourrait-on écrire dans `geometrie.py` pour résoudre le problème ?

Quelles autres fonctions peut-on fabriquer pour le module `geometrie.py` ? Imaginer une liste, on ne va pas tout écrire tout de suite.

**Exercice 16.** Faire, en utilisant une fonction dans le module `geometrie.py`, un programme qui contrôle si un triangle est isocèle, et qui indique en quel sommet le cas échéant.

## VI Une idée de parcours en seconde.

### 1 Entrer deux points et calculer le milieu d'un segment.

- On utilise les commandes `input` et `print`
- On fait exécuter un calcul, il faut faire attention à la différence entre nombres entiers et nombres flottants.

**Remarque :** on peut aussi proposer d'afficher la distance des deux points, les coordonnées du vecteur, etc ...

### 2 Entrer quatre points et voir si ces points définissent un parallélogramme.

- On utilise les commandes `input` et `print`
- On fait exécuter un calcul.
- On utilise l'instruction conditionnelle `if ... else`

**Remarque :** on peut mener le calcul de deux manières différentes : avec les coordonnées de vecteurs ou avec les milieux de segments.

### 3 Entrer quatre points et voir si ces points définissent un parallélogramme, un losange, un rectangle, un carré.

- On va utiliser le programme précédent.
- Il s'agit cette fois de construire la fonction distance de deux points.
- Des instruction `if` seront imbriquées, il faudra bien respecter l'indentation.