	Fiche activité	La programmation Python	STI2D SIN
		Premier pas : EduPython	

1. Notion de fonction

- Dans la fenêtre de script, recopier et exécuter le programme ci-contre. Prendre garde à respecter les espaces en début de ligne (l'indentation).

```
def discr(a,b,c):
    return b**2-4*a*c

discr(5,3,1)
```

  À quoi peut servir ce programme ?

- Effacer la dernière ligne du programme ; mettre à la place, en s'inspirant de la définition de la fonction discr, une fonction x_sommet prenant pour paramètres trois nombres a, b et c et renvoyant l'abscisse du sommet de la parabole d'équation y = ax2 + bx + c. exécuter le programme.

  Rien ne devrait s'afficher ; pourquoi ?

- Dans la console (voir présentation de l'éditeur), écrire : discr (5, 3, 1) et valider ; puis x_sommet (5, 3, 1) et valider. Recommencer cette étape en variant les paramètres.

À retenir :

Une fonction telle que discr :

• prend un certain nombre de paramètres (ici a, b et c), éventuellement aucun ;

• renvoie, via return, un ou plusieurs résultats.

L'objectif est que ces résultats soient exploitables par une autre fonction, ou un programme qui aurait par exemple pour but de les afficher.

Le return termine la fonction : les éventuelles instructions suivantes ne sont pas exécutées.

2. Appeler une fonction ; instruction conditionnelle

Retour à la fenêtre de script. À la suite des fonctions déjà mises en place, écrire la fonction ci-contre, en la modifiant et en la complétant de façon logique ; la fonction retournera une chaîne de caractères.

Exécuter le programme, puis tester la nouvelle fonction dans la console

```
def nombre_racines_trinome(a,b,c):
    if discr(a,b,c)<0 :
        message = "Premier message"
    if (...) : ( ou else: )
        ...
    return message
```

Instructions utiles :

- L'instruction si… alors… sinon… se code comme ci-contre. Il n'y a pas de fin si. Remarquer l'indentation des lignes après les « deux points ».
    - Opérateurs de comparaison : « == » « != » « < > » « <= » « > = »
    - Opérateurs logiques : and or
    - concaténer deux chaînes de caractères : essayer le code ci-contre dans la console.

```
if (condition):
    ...
else:
    ...
```

```
a = "bonjour "
b = "monde"
c = 4
a+b
a+str(c)
```

À retenir :

L'indentation, obtenue avec la touche de tabulation ou avec des espaces, est primordiale : tout ce qui est indenté après le if () : sera exécuté comme un bloc. Il ne faut pas que l'indentation varie (nombre d'espaces, passer de la tabulation à des espaces) en cours de bloc. Une fonction « retourne » aussi bien un résultat numérique qu'une chaîne de caractères.

3. Utiliser des bibliothèques

Modifier la fonction précédente, ou en créer une autre, de sorte que soit résolue dans R l'équation du second degré :

$Ax^2 + bx + c = 0$

À retenir :

La racine carrée s'écrit sqrt(), mais n'existe pas dans les instructions de base de Python. Il faut ajouter une bibliothèque spécifique, en début de programme. Ces bibliothèques sont simplement des catalogues de fonctions. La racine carrée est définie dans la bibliothèque math, que l'on charge en écrivant en début de script : *from math import **

Boucles Pour et Tant_que

Définition :

Syntaxe boucle Pour

```
for compteur in range(1,10):
    ...
```

La variable compteur va parcourir la liste des valeurs contenues dans range(1,10). Cette liste est : [1,2,3,. . .,9] : elle s'arrête juste avant 10.

Autre syntaxe : avec range(10), la variable compteur parcourt la liste [0,1,2,. . . ,9] : cette liste a 10 valeurs.

Syntaxe boucle Tant_que

```
while (condition):
    ...
```

L'indentation a la même importance ici que pour le if ; elle définit les "blocs" d'instructions qui s'exécutent à l'intérieur des boucles.

Application 1

Recopier et compléter la fonction fibo ci-contre, de sorte qu'elle renvoie, sous (u_n) forme de liste, les k premiers termes de la suite définie par :

$u_0 = 1, \quad u_1 = 1 \quad$ et $\quad u_{n+2} = u_n + u_{n+1}$

```
def fibo(k):
    # Initialisation de la liste
    L = [1,1]
    for compteur in range ( ... , ... ):
        ...
    return L
```

Modifier cette fonction de sorte qu'elle prenne pour arguments, outre le nomu_0 et u_1.rmes de la suite à afficher, les valeurs de

Instructions utiles :

Si L est la liste *[1,1,2,3]*, l'instruction *L.append(5)* ajoute le nombre 5 au bout de cette liste. L contient alors : *[1,1,2,3,5].*

Le terme de rang *i* de la liste L est : *L[i].*

L'indexation commence toujours à 0 : avec L = *[1,1,2,3]*, l'élément *L[0]* contient 1.

Application 2

La suite (u_n) est définie par $u_0 = 7$ pour tout *n* entier : $u_{n+1} = \frac{1}{2}\left(u_n + \frac{5}{u_n}\right)$

On peut prouver que cette suite est décroissante et converge vers √ 5.

Créer une fonction qui calcule et renvoie la liste de tous les termes de (u_n) jusqu'à obtenir une approximation à 10^{-6} près de la limite.

Améliorations possibles : la fonction pourrait s'adapter à n'importe quel nombre *a* dont on cherche une approximation de la racine carrée ; d'autres paramètres pourraient être le seuil, le nombre de départ …