(I) ÉTUDE PAS À PAS D'UN ALGORITHME DONNÉ

1. Dichotomie. Cf cours et TP7 et son corrigé.

2. print algorithme (carre , 0. , 2. , 0.15)
(Rq: attention carre(x) est une valeur, la fonction c'est carre.

3. Le critère d'arrêt de la boucle while est:
$(b-a) > 2 * epr$ donc $(b-a) > 0{,}30$.

	début		test $(b-a)>0{,}30$	calculs dans la boucle			test conditionnel: $f(a)\,f(c)<0$		fin	
itération	a	b	résultat	c	f(c)	f(a)	résultat	conséquence	a	b
0	0	2	vrai	1	-2	-3	faux	$a \leftarrow c$	1	2
1	1	2	vrai	1,5	-0,75	-2	faux	$a \leftarrow c$	1,5	2
2	1,5	2	vrai	1,75	0,0625	-0,75	vrai	$b \leftarrow c$	1,5	1,75
3	1,5	1,75	faux	→ On sort de la boucle while						

(Rq: * Vu l'appel de fonction au 2.: $f(x) = carre(x) = x^2 - 3$

* $\sqrt{3} \approx 1{,}73$, pour s'autocorriger on peut vérifier que l'on a à chaque étape $a < 1{,}73 < b$ ...

4. * la valeur affichée à l'écran est celle renvoyée par l'algorithme, c'est-à-dire la dernière valeur de $\frac{a+b}{2}$.

* la valeur affichée est donc $\frac{1,5+1,75}{2} = 1{,}625$

5. * Quel que soit l'intervalle de départ, l'algorithme renvoie une valeur (tant que f est définie sur l'intervalle bien sûr...), et la boucle n'est jamais infinie. Le programme affiche donc toujours un résultat.

* Mais, puisque $\sqrt{3}$ n'appartient pas à $[2;3]$, la fonction carré ne s'annule pas sur cet intervalle et on a toujours, $\forall c \in [a;b]$, $f(a)f(c) > 0$.
Dans l'algorithme on fait donc toujours : $a \leftarrow c$, et c converge vers b, qui n'est par une approximation pertinente de la racine cherchée.

* En l'occurrence, le programme affiche 2,875.

6. On insère entre les lignes 1 et 2 :

```
if f(a) * f(b) > 0 :
    return " nous ne savons pas si f s'annule entre a et b"
```

On privilégie un return qui fait sortir de la fonction plutôt :
- qu'un print seul qui laisse la boucle infinie être stoppée alors qu'elle est absurde
- qu'un print puis un else, qui oblige à indenter la totalité du code suivant.

(II) QUESTION DE COURS - MÉTHODE DE NEWTON

Cf TP 10 et son corrigé.

Quelques remarques :

* on connait le nombre d'itérations, donc on privilégie une boucle for, la boucle while étant préférée lorsque le nombre d'itérations est difficile à anticiper.

* un enchainement du type :

$$\left.\begin{aligned} x_0 &= x_n \\ x_n &= x_0 - f(x_0)/df(x_0) \end{aligned}\right) \text{ ou l'inverse}$$

n'est utile que si x_0 ou x_n doivent être réutilisés à une itération ultérieure, sinon cela ne sert à rien.

C'est utile, par exemple, si on utilise une boucle while avec comme critère d'arrêt quelque chose utilisant x_0 ou x_n.

* si on fait le calcul $x \leftarrow x - f(x)/df(x)$ avant la boucle for, on risque de faire une itération de trop.

(III) EXERCICE – MÉTHODE DE LA FAUSSE POSITION

(Rq : Il s'agit de la méthode de la sécante : TP10 – Exercice 6

1. * Soit $y = px + q$ l'équation de la droite passant par $(a, f(a))$ et $(b, f(b))$.

* On a donc : $p = \dfrac{f(b) - f(a)}{b - a}$

* On trouve aussi : $y(a) = f(a) = pa + q$ d'où $q = \dfrac{f(a)b - f(b)a}{b - a}$

* On cherche x^* tel que $y(x^*) = 0$, d'où :

$$y(x^*) = 0 = px^* + q \text{ et}$$

$$\boxed{x^* = -\frac{q}{p} = a - f(a)\frac{b-a}{f(b)-f(a)}}$$

(Rq : relation analogue à celle utilisée dans la méthode de Newton.

2. Cf code en Annexe.

(Rq : le critère d'arrêt est une distance minimale entre 2 positions de découpe (c_prec - c > eps)

et non pas une longueur minimale de l'intervalle $[a; b]$.

```
# -*- coding: utf-8 -*-
"""
Created on Wed Mar 22 12:28:16 2017

@author: willie
"""

# Exemple d'équation non linéaire à résoudre
def f(x):
    return (x-2)**2-1

#Recherche de la solution par la méthode de la fausse position
def fausse_position(f,a,b,epsilon):
    #on vérifie si une solution existe dans l'intervalle [a;b]
    if f(a)*f(b)>0:
        print("Il n'est pas sûr qu'il y ait une solution dans l'intervalle.")
        return None
    i=0 #initialisation compteur du nombre d'iteration

    #on note c la position de découpe
    distance = b-a #initialisation de la distance entre deux positions de découpe
    c=b #initialisation de la première position de découpe

    while abs(distance)>epsilon and i<1000:
    #on teste si la distance entre deux positions de découpe
    #est supérieure à la précision cherchée
    #et si le nombre d'iteration n'est pas trop grand
        c_prec=c #on sauvegarde la position de découpe précédente
        c=a-f(a)*(b-a)/(f(b)-f(a)) #on calcule la nouvelle position de découpe
        if f(a)*f(c)<0: #le zéro est dans l'intervalle [a;nouvelle position]
            print 'b'
            b=c #le nouvel intervalle [a;b] est donc [a;nouvelle position]
        else: #le zéro est dans l'intervalle [nouvelle position;b]
            a=c #le nouvel intervalle [a;b] est donc [nouvelle position;b]
        distance=c_prec-c #on recalcule la distance entre deux positions de découpe
        i=i+1 #on incémente le nombre d'iterations
    return i,c #on renvoie le nombre d'itérations total et la dernière position
calculée

#on affiche la dernière position calculée après appel de la fonction
print fausse_position(f,0.,2.,1e-12)
```