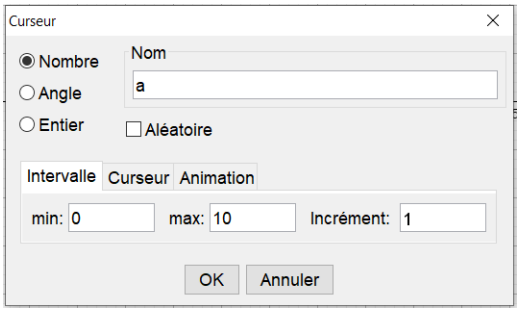
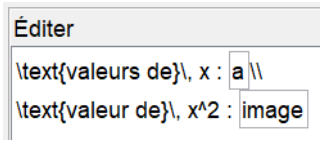
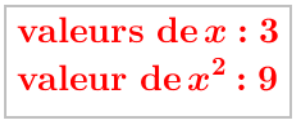
1 Une première approche avec GeoGebra

1. Définir la fonction carré f dans le champ de saisie : Saisie: f(x)=x^2
2. Créer un curseur (outil `curseur`) `a` allant de 0 à 10 avec un incrément de 1 :



3. Définir le nombre `image` défini comme le carré de `a` : Saisie: image = a^2
4. Créer ensuite un texte près de la courbe (outil `texte`) affichant les valeurs de `a` et a^2 (utiliser l'onglet `Objets` pour insérer des objets déjà créés dans GeoGebra) :





5. En partant de 0, faire varier le curseur pour que la valeur de x^2 dépasse 2. En déduire l'encadrement à l'unité de $\sqrt{2}$: ..... $< \sqrt{2} <$ .....
6. Régler l'incrément du curseur `a` à 0,1 : Incrément: 0.1 et déterminer l'encadrement de $\sqrt{2}$ au dixième : ..... $< \sqrt{2} <$ ..... Combien de "pas" de 0,1 ont été nécessaires (depuis 0) pour atteindre la première borne de l'encadrement ?
7. Même question pour un encadrement au centième : ..... $< \sqrt{2} <$ .....

2 Algorithme de balayage et programmation en Python

La procédure vue avec GeoGebra peut se répéter autant de fois que nécessaire pour atteindre un encadrement de $\sqrt{2}$ d'une précision donnée. Cette méthode, dite *« par balayage »*, peut être traduite sous la forme d'un algorithme, avec la fonction balayage :

racine_balayage

```
1 Fonction
    racine_balayage(nb_decimales) :
2     pas ← ......
3     x ← ......
4     Tant que ..... faire :
5         x ← ......
6     renvoie ....., x
```

Code Python :

```
1 def racine_balayage(nb_decimales):
2     """détermine  par balayage un
      encadrement de racine de 2
      d'amplitude 10^(-nb_decimales)"""
3     pas = ...
4     x = ....
5     while .......... :
6         x = ......
7     return ....., x
```

1. Complétez l'algorithme puis le programme Python associé.
2. Implémentez le programme Python et testez-le avec nb_decimales= 3. Que constate-t-on? Interprétation?
3. Modifiez le programme pour qu'il renvoie des valeurs avec la bonne précision décimale (on utilisera la fonction `round(x,n)` qui renvoie un arrondi de `x` avec `n` chiffres après la virgule).
4. a. Modifiez la fonction pour qu'elle donne un encadrement à une précision donnée de n'importe quel réel positif `nombre`.

racine_balayage

```
1 Fonction racine_balayage(nombre,
    nb_decimales) :
2     pas ← ......
3     x ← ......
4     Tant que ..... faire :
5         x ← ......
6     renvoie arrondi(.....,
        nb_decimales),
        arrondi(.....,
        nb_decimales)
```

Code Python :

```
1 def
  racine_balayage(nombre,nb_decimales):
2     """détermine  par balayage un
      encadrement de racine de nombre
      d'amplitude 10^(-nb_decimales)"""
3     pas = ...
4     x = ....
5     while ..... :
6         x = ......
7     return round(.......,nb_decimales),
      round(.......,nb_decimales)
```

b. Testez votre fonction avec les appels suivants :

Console Python :

```
>>> racine_balayage(2,nb_decimales = 7) # le nombre dont on veut encadrer la
racine carrée est passé en paramètre dans la fonction
(1.4142135, 1.4142136)
>>> racine_balayage(5,6) # encadrement de √5 à 10^-6
(2.236067, 2.236068)
```

c. Testez `racine_balayage(2,9)`. Que se passe-t-il?

d. Mettez en place un `compteur` qui comptera le nombre de tours de boucles. Pour cela il faudra placer dans le bloc d'instructions de la fonction les deux instructions suivantes, sans oublier de mettre la valeur de `compteur` dans le résultat renvoyé par la fonction :

Instruction 2 :

```
compteur = 0
```

Instruction 2 :

```
compteur = compteur + 1
```

e. Donner le nombre de tours de boucles effectués pour obtenir le résultat de l'appel :

Console Python :

```
>>> racine_balayage(2,nb_decimales = 7)
.....
```

f. Donnez un ordre de grandeur du nombre de tours de boucle nécessaires pour calculer `racine_balayage(2,9)`.

3 Balayage avec départ optimisé

1. Quel est le problème de l'algorithme précédent ? Comment l'améliorer pour qu'il puisse fournir un encadrement plus précis ?

L'idée est donc de recommencer le balayage à chaque niveau de précision mais en **partant d'une valeur `depart`, optimisée car obtenue au niveau précédent**.
On met alors en place une **deuxième structure répétitive** : une boucle `Pour` qui va s'exécuter autant de fois qu'il y a de niveaux de précision : 1 fois pour un encadrement à l'unité, deux fois pour un encadrement au dixième, 3 fois pour un encadrement au centième,...
À chaque tour de boucle, on détermine l'encadrement par balayage en partant de la borne inférieure de l'encadrement obtenu au tour précédent.

racine_balayage_opt

```
1 Fonction racine_balayage_opt(nombre, nb_decimales) :
2     debut ← 0
3     Pour i allant de 0 à ..... faire :
4         pas ← .....
5         x ← debut
6         Tant que x² < ..... faire :
7             x ← x+pas
8         debut ← .....
9     renvoie arrondi(....., nb_decimales), arrondi(....., nb_decimales)
```

2. Compléter cet algorithme et construire la fonction `racine_balayage_opt` dans un script Python.
3. Effectuer le test de l'encadrement de $\sqrt{2}$ avec une précision de 9 décimales, puis 15 décimales. Vous devez obtenir les valeurs ci-dessous :

Console Python :
```
>>> racine_balayage_opt(2,9)
(1.414213562, 1.414213563)
```

Console Python :
```
>>> racine_balayage_opt(2,15)
(1.414213562373094, 1.414213562373095)
```

4. Insérez un **compteur** dans votre fonction pour connaître le nombre de calculs de carrés nécessaires pour obtenir l'encadrement de $\sqrt{2}$ avec une précision de 9 décimales. Conclusion ?

Code Python :
```
def balayage_racine_opt(nombre, nb_decimales):
    """détermine  par balayage optimisé un encadrement de la racine carrée d'un
    nombre d'amplitude 10**(-nb_decimales)"""
    debut = 0 # variable qui recevra les bornes inférieures de chaque encadrement
    compteur = 0
    for i in range(0,.............) :
        pas = .........
        x = debut # balayage commencé à début avec l'algorithme de balayage initial
        while x**2 < ..........:
            x = x  + pas
            compteur = ...........
        debut = ........ # pour le prochain balayage, on repartira avec la borne
        inférieure de l'encadrement obtenu
    return round(debut, nb_decimales), round(debut + pas, nb_decimales), compteur
```

4 Scripts corrigés

Code Python :

```python
import math
def racine_deux_balayage(nb_decimales):
    """détermine  par balayage un encadrement de racine carrée de 2 d'amplitude 10^(-n)"""
    pas = 10**(-nb_decimales)
    x = 0
    while x**2 < 2:
        x = x + pas
    return round(x - pas, nb_decimales), round(x, nb_decimales)

def racine_balayage(nombre,nb_decimales):
    """détermine  par balayage un encadrement de racine carrée de a d'amplitude 10^(-n)"""
    pas = 10**(-nb_decimales)
    x = 0
    compteur = 0
    while x**2 < nombre:
        x = x + pas
        compteur = compteur + 1
    return round(x-pas,nb_decimales),round(x,nb_decimales),compteur


def racine_balayage_opt(nombre, nb_decimales):
    """détermine  par balayage un encadrement de la racine carrée d'un nombre d'amplitude 10**(-nb_decimales), en optimisant la borne inférieure de l'encadrement à chaque tour de boucle"""
    debut = 0 # on crée une variable début qui recevra tour à tour les bornes inférieures de chaque encadrement
    compteur = 0
    for i in range(0,nb_decimales + 1) :
        pas = 10**(-i)
        x = debut # on commence le balayage à début et on reprend l'algorithme de balayage initial
        while x**2 < nombre :
            x = x  + pas
            compteur = compteur + 1
        debut = x - pas # pour le prochain balayage, on repartira avec la borne inférieure de l'encadrement obtenu
    return round(debut, nb_decimales), round(debut + pas, nb_decimales), compteur # on renvoie le dernier encadrement obtenu

def racine_dicho(nombre,nb_decimales):
    """détermine par dichotomie un encadrement de la racine carrée d'un nombre d'amplitude 10**(-nb_decimales) en partant d'un encadrement à l'unité a et b"""
    x = 0 # on reprend l'algorithme de balayage pour trouver un encadrement à l'unité de la racine carrée
```

```
    while x**2 < nombre :
        x = x + 1
    debut = x - 1
    fin = x
    compteur = 0
    while abs(fin-debut) > 10**(-nb_decimales):
        milieu = (debut + fin) / 2
        if milieu**2 <= nombre :
            debut = milieu
        else :
            fin = milieu
        compteur = compteur + 1
    return round(debut, nb_decimales), round(fin, nb_decimales), compteur

def racine_heron(a, nb_decimales) :
    """cette fonction calcule un encadrement de la racine carrée de a avec le nombre de décimales spécifié"""
    longueur = a
    largeur = 1
    compteur = 0
    while abs(longueur - largeur) > 10**(-nb_decimales) :
        longueur = (longueur + largeur) / 2
        largeur = a / longueur
        compteur = compteur + 1
    return round(largeur,nb_decimales), round(longueur,nb_decimales),compteur


def decoupage_tranches(nombre):
    """découpe un entier en blocs de 2 chiffres en partant de la droite"""
    chaine_nombre = str(nombre)
    liste_tranches = []
    while len(chaine_nombre) >= 2 :
        tranche = chaine_nombre[-2]+ chaine_nombre[-1] # extraction de la tranche la plus à droite
        liste_tranches = [tranche] + liste_tranches # ajout de la tranche en tête de liste
        chaine_nombre = chaine_nombre[:-2] # on prend la liste tronquée de ses deux derniers éléments
    if chaine_nombre != '': # on rajoute un éventuel reste si le nombre de chiffres est impair
        liste_tranches = [chaine_nombre] + liste_tranches
    return liste_tranches


def racine_potence(nombre,nb_decimales):
    """renvoie la racine carrée d'un nombre entier, avec une précision spécifiée par nb_decimales"""
    nombre = int(str(nombre)+'00'*nb_decimales) # reconstitution du nombre en l'y adjoignant des tranches de '00' pour aller jusqu'à la précision souhaitée
```

```
    liste_tranches = decoupage_tranches(nombre)
    # gestion de la première tranche
    dividende = int(liste_tranches.pop(0)) # on enlève la première tranche de
    la liste
    chiffre = 0
    while chiffre**2 <= dividende : # recherche de la racine carrée de la
    première tranche
        chiffre = chiffre + 1
    chiffre = chiffre - 1
    racine = chiffre
    reste = dividende - chiffre**2
    # gestion des autres tranches
    while liste_tranches != []:
        dividende = int(str(reste) + liste_tranches.pop(0))
        double = 2 * racine
        chiffre = 0
        while (double*10 + chiffre)*chiffre <= dividende :
            chiffre = chiffre + 1
        chiffre = chiffre - 1
        racine = 10*racine + chiffre
        reste = dividende - (double*10 + chiffre)*chiffre
    return racine/10**(nb_decimales)
```