CPGE – TSI1
Les fonctions et les procédures en python

1. Cas d'utilisation d'une fonction d'une fonction (ou d'une procédure)

- Pour diviser la complexité d'un problème :
  On écrit plusieurs fonctions qui effectuent chacune un traitement élémentaire. On pourra tester séparément ces fonctions pour les utiliser ensuite dans la résolution du problème.

- Quand un traitement est répétitif :
  On écrit une fonction (ou une procédure) qu'on appellera chaque fois que cela est nécessaire. On gagne ainsi en lisibilité et en compacité du code.

- Quand on veut réutiliser ultérieurement un traitement :
  Une fonction (ou une procédure) testée et mise au point peut être intégrée à une bibliothèque (collection des fonctions et/ou de procédures).
  Les exemples sont très nombreux en python (math…).

2. Différence entre fonction et procédure

- Une fonction effectue un calcul et renvoie un résultat.

- Une procédure effectue un traitement (affichage, modification de données…) et ne renvoie pas de valeur.

3. Déclaration

- Une fonction ou une procédure se déclare avec le mot clé **def**, avec le caractère « **:** » en fin de ligne

- Une fonction ou une procédure peut recevoir un ou plusieurs arguments,

- Une fonction retourne une ou plusieurs valeurs avec le mot clé **return**.

- Bonnes pratiques : Mettre un commentaire d'entête pour décrire ce que fait la fonction (documentation du code python)

```
"""
Fonction calculant le discriminant d'une équation du 2nd degré
ax2 + bx + c = 0
"""
def delta(a,b,c):
    return b**2 - 4*a*c
```

4. Utilisation

- Une fonction s'utilise (appel) par son nom et la valeur des paramètres dans les parenthèses, comme ici dans une autre fonction :

```
"""
Fonction calculant les racines d'une équation du 2nd degré
ax2 + bx + c = 0
"""
from math import sqrt
def racines(a,b,c):
    d = delta(a,b,c)
    if d > 0 :
        # On a 2 solutions
        return (-b -sqrt(d)) / (2*a), (b -sqrt(d)) / (2*a)
        # On a 1 solution
    elif d == 0:
        return (-b) / (2*a)
    else:
        # On a 0 solution
        return()
```

- Ou par la console Python :

```
In [9]: racines (1,5,4)
Out[9]: (-4.0, 1.0)

In [12]: racines (4,4,1)
Out[12]: -0.5

In [5]: racines (5,3,1)
Out[5]: ()

In [13]: delta (1,5,4)
Out[13]: 9

In [14]: delta (4,4,1)
Out[14]: 0

In [15]: delta (5,3,1)
Out[15]: -11
```

5. Exercices

- Taper et tester le code Python des fonctions **delta(a,b,c)** et **racines(a,b,c)** dans un fichier **Eq2degre.py** (partie gauche de Spyder)

- Compléter le code avec ces fonctions ou procédures :

```
29 def saisie(parametre):
30     print("Entrez la valeur de " + parametre + " :")
31     return(int(input()))

37 def afficheRacines(a,b,c):
38     print("Racines : ")
39     print(racines(a,b,c))
```

- **saisie(parametre)** est-elle une fonction ou une procédure ?

- **afficheRacines(a,b,c)** est-elle une fonction ou une procédure ?

- Créer les commentaires d'entête de ces fonctions ou procédures.

- Créer un programme principal à la suite permettant de saisir les 3 paramètres a, b, et c et d'afficher les racines de l'équation $ax^2 + bx + c = 0$

- Tester votre programme. Voici un exemple d'affichage :

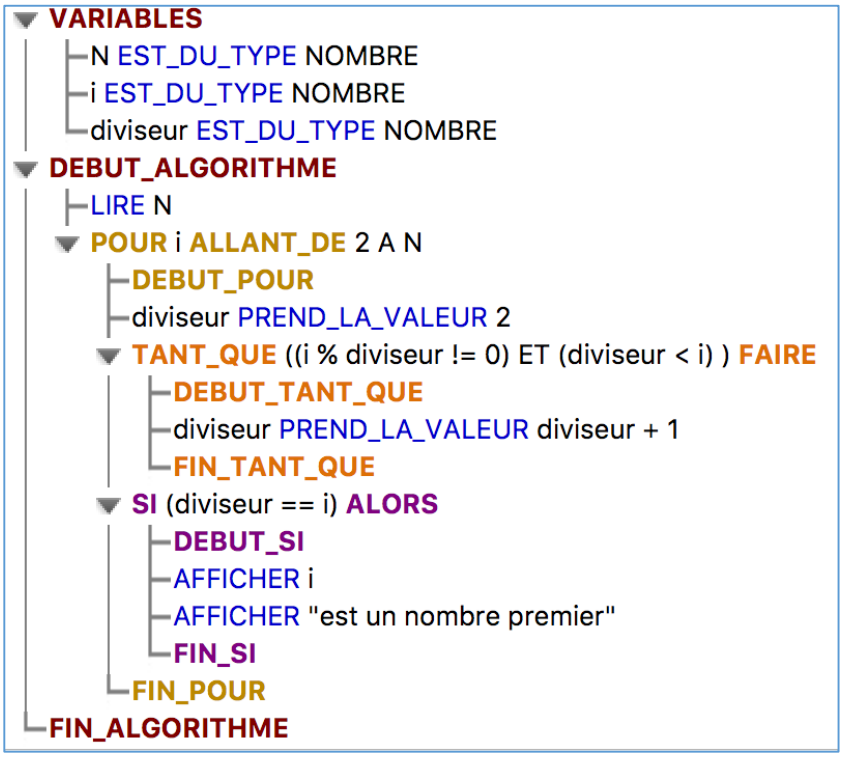
```
Entrez la valeur de a :

1
Entrez la valeur de b :

5
Entrez la valeur de c :

4
Racines :
(-4.0, 1.0)
```

6. Exercices d'application

- Créer et tester une fonction qui saisit une borne (nombre entier).
- Créer et tester une fonction qui renvoie une liste de nombres premiers inférieurs à une borne passée en paramètre.
- Créer et tester un programme principal qui saisit une borne et affiche la liste des nombres premier inférieurs à cette borne.

Rappels :

Algorithme :



Programme en Python :

```
1  # Entrée des données
2  print ("Entrez le nombre N : ")
3  N = int(input())
4  # Traitement répété N fois
5  for i in range(2, N):
6      diviseur = 2
7      while i % diviseur != 0 and diviseur < i :
8          diviseur = diviseur + 1
9      if diviseur == i:
10         # affichage du résultat
11         print (i)
12         print ("est un nombre premier")
```