TP 3 : Fonctions ; Équations ; Boucles imbriquées

1. Retour sur les fonctions

Exercice 1. Calcul booléen

(1) Écrire une fonction `langue(n)` qui étant donné un entier `n` affiche un message dépendant de la la valeur de `n` :

Valeur de `n`	Message à afficher
1	`bonjour`
2	`good morning`
3	`buenos dias`

(2) Écrire une fonction `signe(x)` qui, étant donné un réel `x` renvoie le signe de `x`, i.e.[1], -1 si `x` est strictement négatif, 0 si `x` est nul et 1 si `x` est strictement positif.

(3) Écrire une fonction qui, étant donné un réel x donne l'image de x par l'unique fonction $\mathbb{R} \to \mathbb{R}$ impaire[2] f telle que $\forall x \in \mathbb{R}^+, f(x) = x^2$.

(4) Écrire une fonction qui, étant donné un entier n, renvoie `True` si n est pair[3], et `False` sinon.

(5) Écrire une fonction qui, étant donné deux entiers n et m, renvoie `True` si les deux entiers ont la même parité, et `False` sinon.

2. Équations polynômiales

Exercice 2. Équations du premier degré

(1) Écrire une fonction dont l'en-tête est def sol_premier_degre(a,b) qui renvoie la solution d'une équation du premier degré $ax + b = 0$ avec a et b réels

(2) Écrire une fonction dont l'en-tête est def est_solution_de_premier_degre(a,b,x) qui teste si un réel x est solution d'une équation $ax + b = 0$ avec a et b réels

(3) Tester ces fonctions pour $7x - 2.1 = 0$

(4) Tester ces fonctions pour $3x - 7 = 0$

Exercice 3. Équations du second degré

(1) Écrire un programme qui étant donné une équation du second degré à coefficients réels détermine le nombre de ses solutions réelles et leurs valeurs éventuelles

(2) Tester ce programme pour $x^2 + 6x + 9 = 0$

(3) Tester ce programme pour $x^2 + 10^{-10} = 0$ puis pour $x^2 - 10^{-10} = 0$

1. Abréviation du latin *id est* signifiant "c'est-à-dire".
2. Une fonction f définie sur $\mathbb{R}$ est impaire si et seulement si $\forall x \in \mathbb{R}, f(-x) = -f(x)$.
3. On pourra penser au reste par la division euclidienne par 2.

(4) Tester ce programme pour $0,1x^2 + 0,6x + 0,9 = 0$ Que constate-t-on ? Expliquer ce phénomène.

(5) Tester ce programme pour $x^2 + (1 + 2^{-50})x + 0,25 + 2^{-51} = 0$ et comparer les résultats obtenus avec la résolution à la main de cette équation. Expliquer les différences trouvées

(6) On travaille dans cette question sous l'hypothèse (raisonnable au vu des résultats précédents) que la norme IEEE754 assure les propriétés suivantes des arrondis :
   - si le discriminant est strictement positif, alors sa valeur approchée calculée est positive ou nulle
   - si le discriminant est strictement négatif, alors sa valeur approchée calculée est négative ou nulle
   - si le discriminant est nul sa valeur approchée est de signe quelconque.

   Adapter votre programme pour qu'il n'affiche que des affirmations certaines à propos des polynômes qui lui sont fournis.

3. Suites récurrentes

On appelle *suite récurrente simple*, une suite (u_n) définie par :

(1) son premier terme $u_0 = a$, où a est un réel donné,

(2) une relation entre deux termes consécutifs du type $u_{n+1} = f(u_n)$ où f est une fonction donnée appelée *fonction génératrice* de la suite (u_n)

Si tout va bien, en particulier si a appartient à un intervalle stable par f, les données de a et de f permettent de calculer, pour tout entier naturel n, le terme de rang n.

Si tout va encore mieux, une telle suite peut converger vers un *point fixe* de f, c'est-à-dire une solution de l'équation $f(x) = x$

Malheureusement, même dans une situation ou, a priori, tout devrait bien se passer, les choses ne sont pas si simples, comme en témoigne l'exemple ci-dessous.

Exercice 4. Suite récurrente instable

Si u_0 est un point fixe de f, une récurrence triviale prouve alors que la suite (u_n) est constante égale à u_0.

Dans ce cas, elle converge évidemment vers u_0. Par exemple la suite (u_n) définie par $u_0 = \dfrac{7}{3}$ et, pour tout n, $u_{n+1} = -2u_n + 7$ est constante (justifiez).

Et pourtant...

(1) Voici un programme utilisant une boucle `for` :

```
terme = 7./3
for indice in range(101) :
    print(terme)
    terme = -2*terme + 7
```

Recopier ce programme dans l'éditeur, expliquer à quoi il sert et commenter chacune de ses lignes.

Exécuter ce programme. Que peut-on constater ? Comment expliquer cela ?

(2) Voici un deuxième programme :

```
terme = 7./3
for i in range(121) :
    print(i*10, terme)
    for j in range(1,11) :
        terme = -2*terme + 7
```

Recopier ce programme dans l'éditeur, expliquer à quoi il sert et commenter chacune de ses lignes (à l'aide du symbole #). Exécuter ce programme.

(3) Le professeur vous fournit un script appelé `conversion.py`.

Utiliser la fonction `float2sem` qui convertit un flottant en binaire, selon la norme IEEE754, en faisant apparaître les bits affectés au signe (s), à l'exposant (e) et à la matisse (m).

Modifier alors les programmes précédents pour qu'ils affichent les représentations binaires des termes successifs de la suite (u_n).

Exercice 5. Résolution d'une équation par l'étude d'une suite récurrente

On cherche ici un encadrement de la solution de l'équation : $\ln(x) + 2 - x = 0(1)$.

On peut facilement prouver que cette équation possède une unique solution strictement supérieure à 1. Cette solution sera notée s_1.

(1) Définir la fonction f : $x - \ln(x) + 2$.

La fonction logarithme népérien se trouve, sous le nom `log`, dans la bibliothèque `math`.

Le plus simple est de mettre en début de script `from math import log`

(2) Écrire une boucle « for » permettant de calculer le terme de rang 10 de la suite récurrente (u_n) de premier terme 3 et de fonction génératrice f.

(3) Même question pour la suite récurrente (v_n) de premier terme 4 et de fonction génératrice f.

La suite (u_n) semble croissante, et (v_n) semble décroissante, et l'une et l'autre prennent, au fur et à mesure que l'indice augmente, des valeurs de plus en plus proches. On admet que leurs termes de rang n fournissent un encadrement de s_1.

(4) Écrire (à l'aide d'une boucle `while` un programme permettant d'obtenir un encadrement à 10^{-8} près de s_1.

4. Boucles imbriquées

Exercice 6. Tracé de rectangles

Écrire une fonction `rectangle(n,p)` qui affiche à l'écran un rectangle de n lignes et p colonnes remplies avec le caractère * comme ci-dessous :

```
* * * * *
* * * * *
* * * * *
```

Le retour à la ligne peut être obtenu à l'aide du caractère spécial `\n`

Exercice 7. Tracé de triangles

(1) Écrire une fonction `triangle(n)` qui affiche à l'écran un triangle de n lignes remplies avec le caractère * comme ci-dessous :

```
*
* *
* * *
* * * *
```

(2) Écrire une fonction `triangle(n)` qui affiche à l'écran un triangle de n lignes remplies avec le caractère * comme ci-dessous :

```
    *
   * * *
  * * * * *
```