INFO-H-100 - Programmation
TP 7 et 8 — Boucles `while`

Lorsque l'exercice demande d'écrire une fonction, écrivez la fonction demandée et testez-la avec plusieurs valeurs pertinentes.

Cette séance portant sur les boucles `while`, veillez à ne pas utiliser de boucle `for`.

Ex. 1. Qu'affiche le programme suivant ?

```
a = 1
while a <= 5:
    print a,
    a += 1
print a
```

Ex. 2. Qu'affiche le programme suivant ?

```
a = 0
while a < 5:
    a += 1
    print a,
print a
```

Ex. 3. Qu'affiche le programme suivant ?

```
a = 0
while a <= 5:
    a -= 1
    print a,
print a
```

Ex. 4. Ecrire une fonction `my_range(a,b)` qui produit une liste ordonnée contenant tous les entiers dans $[a, b[$.

Ex. 5. Ecrire une fonction `my_range_step(a,b,step)` qui produit une liste ordonnée contenant tous les entiers dans $[a, b[$ séparés de `step`. On considère que $a < b$ et que `step` est strictement positif.

Ex. 6. Ecrire une fonction `my_range_step(a,b,step)` qui produit une liste ordonnée contenant tous les entiers dans $[a, b[$ séparés de `step`. a et b ne sont pas nécessairement dans l'ordre croissant et `step` peut être négatif.

```
print my_range_step(1,10,2)     # [1, 3, 5, 7, 9]
print my_range_step(10,1,-2)    # [10, 8, 6, 4, 2]
print my_range_step(1,10,-2)    # []
print my_range_step(10,1,2)     # []
```

Ex. 7. Ecrire une fonction qui renvoie l'indice de la première apparition d'un caractère dans une chaîne, ou `-1` si le caractère n'apparaît pas.

Avec une boucle `for`, cette fonction peut s'écrire de la manière suivante :

```
def find1(string, char):
    for i in range(len(string)):
        if string[i] == char:
            return i
    return -1
```

Réécrivez-là en utilisant une boucle `while` et en ne faisant qu'un seul `return`.

Ex. 8. Ecrire une fonction qui reçoit en argument une chaîne de caractères et deux caractères, et qui renvoie une nouvelle chaîne de caractères où la première occurrence du premier caractère a été remplacée par le second caractère.

Ex. 9. Écrire une fonction qui détermine si une séquence est croissante.

```
>>> croissante([1, 2, 5, 8]) # -> True
>>> croissante([1, 8, 5, 2]) # -> False
>>> croissante([1, 2, 2, 5]) # -> False
>>> croissante([5])          # -> True
>>> croissante([])           # -> True
```

Ex. 10. Ecrire une fonction qui détecte si un mot passé en argument est un palindrome. Pour rappel, un palindrome est une phrase qui peut se lire dans les deux sens : l'ordre des lettres est symétrique.

```
>>> palindrome("hello")
False
>>> palindrome("radar")
True
```

Ex. 11. Ecrire une fonction qui calcule la valeur approchée de π base de la série suivante (due à Euler). Les calculs s'arrêtent lorsque la valeur du dernier terme ajouté est inférieure à 10^{-6} ou lorsque le nombre de termes considérés atteint 10000.

$$\frac{\pi^2}{6} = 1 + \frac{1}{2^2} + \frac{1}{3^2} + \frac{1}{4^2} + ...$$

Ex. 12. Ecrire une fonction qui calcule la valeur approchée de π base de la série suivante. Les calculs s'arrêtent lorsque la valeur du dernier terme ajouté est inférieure à un ϵ donné (c'est à dire en variable globale) ou lorsque le nombre de termes considérés atteint une borne également donnée.

$$\frac{\pi}{8} = \frac{1}{1 \times 3} + \frac{1}{5 \times 7} + \frac{1}{9 \times 11} + ...$$

Ex. 13. Ecrire une fonction qui calcule la valeur approchée de π base de la série suivante (due à Leibniz). Les calculs s'arrêtent lorsque la valeur du dernier terme ajouté est inférieure à un ϵ donné (c'est à dire en variable globale) ou lorsque le nombre de termes considérés atteint une borne également donnée.

$$\frac{\pi}{4} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - ...$$

Ex. 14. Ecrire une fonction qui renvoie la liste ordonnée de tous les nombres de Fibonacci inférieurs à n. Le n^e nombre de Fibonacci est défini de la manière suivante : $\forall n \geq 2 : F_n = F_{n-1} + F_{n-2}$ avec $F_0 = 0$ et $F_1 = 1$.

Ex. 15. Ecrire une fonction qui calcule la valeur approchée de π sur base de la série suivante. Les calculs s'arrêtent lorsque la valeur du dernier terme ajouté est inférieure à un ε donné, ou lorsque le nombre de termes considérés atteint une borne également donnée.

$$\frac{\pi}{4} = 4 \cdot arctan\left(\frac{1}{5}\right) - arctan\left(\frac{1}{239}\right)$$

avec la série :

$$arctan(x) = \frac{x}{1+x^2}\left(1 + \frac{2}{3} \cdot \frac{x^2}{1+x^2} + \frac{2}{3} \cdot \frac{4}{5} \cdot (\frac{x^2}{1+x^2})^2 + \cdots\right) = \frac{x}{1+x^2}\sum_{i=0}^{\infty} c_i (\frac{x^2}{1+x^2})^i$$

où $c_0 = 1$
et pour $i > 0, c_i = \frac{2 \cdot i}{2 \cdot i + 1} c_{i-1}$

Ex. 16. Deux suites (x_i) et (y_i) sont définies par

$$\begin{aligned} y_0 &= 2 \\ x_0 &= 1 \\ x_{i+1} &= \frac{2}{y_{i+1}} \\ y_{i+1} &= \frac{y_i + x_i}{2} \end{aligned}$$

Nous admettons que les suites convergent toutes les deux vers $\sqrt{2}$, avec de plus $x_i < \sqrt{2} < y_i$. Ecrire une fonction `rac2` qui calcule deux approximations supérieure et inférieure de $\sqrt{2}$, de différence inférieure à ε (prévoyez aussi d'arrêter les calculs si le nombre d'étapes dépasse une borne fixée).

Ex. 17. Soit r un nombre réel positif. La suite de nombres réels x_n est spécifiée par :

$x_1 = 1$,
$x_{n+1} = \frac{1}{2}\left(x_n + \frac{r}{x_n}\right)$,

Cette suite, obtenue par une méthode générale, due à Newton, converge très rapidement vers $\sqrt{r}$.

Ecrire une fonction qui calcule la valeur approchée de $\sqrt{r}$. L'exécution s'arrêtera lorsque la différence entre deux approximations successives sera inférieure à un ε donné.

Ex. 18. Soit r un nombre réel positif. Deux suites de nombres réels x_n et y_n sont spécifiées par :

$x_1 = 1$,
$y_1 = 1$,

et pour $n \geq 1$:

$$\begin{cases} x_{n+1} = x_n + r.y_n \\ y_{n+1} = x_n + y_n \end{cases}$$

Il n'est pas difficile de prouver $y_n \neq 0$ pour $n \geq 1$, et aussi que le quotient $\frac{x_n}{y_n}$ converge vers $\sqrt{r}$.

Ecrire une fonction qui calcule la valeur approchée $\frac{x_n}{y_n}$ de $\sqrt{r}$, où n est la plus petite valeur pour laquelle la condition suivante est vraie :
$(x_n)^2 < (r+\varepsilon)(y_n)^2$ **et** $(x_n)^2 > (r-\varepsilon)(y_n)^2$.
(ε étant comme d'habitude une valeur donnée). L'exécution sera interrompue si le nombre d'étapes devient trop grand.

INFO-H-100 - Programmation
TP 7 et 8 — Boucles `while`
Corrections

Solution de l'exercice 1:

Le programme affiche `1 2 3 4 5 6`

Solution de l'exercice 2:

Le programme affiche `1 2 3 4 5 5`

Solution de l'exercice 3:

Le programme affiche `-1 -2 -3 -4 -5 -6 -7` ... et ne s'arrête jamais (boucle infinie).

Solution de l'exercice 4:

```
def my_range(a,b):
        liste = []
        while (a < b):
                liste.append(a)
                a += 1

        return liste
```

Solution de l'exercice 5:

```
def my_range_step(a,b,step):
        liste = []
        while (a < b):
                liste.append(a)
                a += step

        return liste
```

Solution de l'exercice 6:

```
def my_range_step(a,b,step):
    liste = []
    if a<b and step>0:
        while a<b:
            liste.append(a)
            a += step
    elif a>b and step<0:
        while a>b:
            liste.append(a)
            a += step
    return liste

print my_range_step(1,10,2)     # [1, 3, 5, 7, 9]
print my_range_step(10,1,-2)    # [10, 8, 6, 4, 2]
print my_range_step(1,10,-2)    # []
print my_range_step(10,1,2)     # []
```

Solution de l'exercice 7:

```
def find1(ls, c):
        i = 0
        while i < len(ls) and ls[i] != c:
                i += 1
        if i == len(ls):
                i = -1
        return i
```

Solution de l'exercice 8:

Voir slides...

```
def replace1(st, old, new):
    i = 0
    while i < len(st) and st[i] != old:
        i += 1
    #i == lg or st[i] == old
    if i < len(st):
        st = st[:i] + new + st[i+1:]
    return st
```

Solution de l'exercice 9:

```
def croissante(ls):
    result = True
    if len(ls) >= 2:
        i = 1
        while i < len(ls) and result:
            result = ls[i] > ls[i - 1]
            i += 1
    return result
```

Solution de l'exercice 10:

```
def palindrome(mot):
    g = 0
    d = len(mot)-1
    while g<d and mot[g] == mot[d]:
        g+=1
        d-=1
    return mot[g] == mot[d]
```

Solution alternative...

```
def palindrome(mot):
    i = 0
    ok = True
    while i < len(mot)/2 and ok:
        if mot[i] != mot[len(mot)-i-1]:
            ok = False
        i+=1

    return ok

print palindrome('sos')
print palindrome('abba')
print palindrome('thierry')
```

Solution de l'exercice 11:

```
EPS        = 1E-6
NB_MAX = 10000

import math

def piEuler():
    nbTermes = 1
    terme = 1
    somme = 1
    while terme >= EPS and nbTermes < NB_MAX:
        nbTermes += 1
        terme = 1.0 / nbTermes ** 2
        somme += terme
    return math.sqrt(6*somme)
```

Solution de l'exercice 12:

```
def pi11():
    i = 1
    terme = 1.0/3
    somme = terme
    while terme >= EPS and i < NB_MAX:
        i += 1
        terme = 1.0/(4*i-3)/(4*i-1)
        somme += terme
    return 8*somme
```

Solution de l'exercice 13:

```
def piLeibnitz():
    nbTermes = 1
    signe = 1
    terme = 1
    somme = 1
    while terme >= EPS and nbTermes < NB_MAX:
        nbTermes += 1
        signe = -signe
        terme = 1.0 / (2 * nbTermes - 1)
        somme += signe*terme
    return 4*somme
```

Solution de l'exercice 14:

```
def listFib(n):
    ls = []
    if n >= 1:
        ls.append(0)
        (a, b) = (0, 1)
        while b < n:
            ls.append(b)
            (a, b) = (b, a+b)
    return ls
```

Solution de l'exercice 15:

```
def arctan(x):
    facteur0 = x**2 / (1 + x**2)
    i = 1
    facteur1 = 1
    facteur2 = 1
    terme = 1
    somme = 1
    while terme >= EPS and i < NB_MAX:
        facteur1 *= 2.0*i/(2*i + 1)
        i += 1
        facteur2 *= facteur0
        terme = facteur1*facteur2
        somme += terme
    return facteur0/x*somme

def piArctan():
    return 4*(4*arctan(1.0/5) - arctan(1.0/239))
```

Solution de l'exercice 16:

```
def rac2():
    (x, y) = (1, 2)
    i = 0
    while y - x >= EPS and i < NB_MAX:
        y = (x + y)/2.0
        x = 2.0/y
        i += 1
    return (x, y)
```

Solution de l'exercice 17:

```
def sqrtNewton(r):
    old = 0
    x = 1.0
```

```
    while abs(x - old) >= EPS:
        old = x
        x = (x + r/x)/2.0
    return x
```

Solution de l'exercice 18:

```
def proche(x, y, r):
    x2 = x * x
    y2 = y * y
    return x2 < ((r + EPS)*y2) and x2 > ((r - EPS)*y2)

def racine(r):
    i = 1
    (x, y) = (1, 1)
    while not proche(x, y, r) and i < NB_MAX:
        (x, y) = (x + r*y, x + y)
        i += 1
    return float(x)/y
```