TP n° 2 : fonctions récursives

1. Vrai / Faux

1. La récursivité peut s'utiliser pour remplacer une boucle *while* mais ne peut pas remplacer une boucle *for*.
2. On peut parler d'un programme récursif par opposition à un programme itératif.
3. Les valeurs des paramètres d'une fonction récursive doivent être identiques à chaque appel pour assurer la terminaison.
4. Dans une fonction récursive, il peut y avoir plusieurs conditions d'arrêt.
5. Les valeurs passées en paramètres pour les appels récursifs doivent constituer une suite décroissante.

2. Comprendre un programme récursif

2.1. Premier dessin

Avant de le tester, prédire ce que va faire le programme ci-dessous. Ensuite, le tester pour vérifier votre prédiction.

```
import turtle as tu

def dessine(n):
    if n > 0:
        tu.forward(n)
        tu.right(90)
        dessine(n - 5)

dessine(200)
tu.ht()
tu.mainloop()
```

2.2. Second dessin

Avant de le tester, prédire ce que va faire le programme ci-dessous. Ensuite, le tester pour vérifier votre prédiction.

```
import turtle as tu

def dessin(taille):
    for i in (-1, 1):
        tu.up()
        tu.goto(- taille // 2, i * taille)
        tu.down()
        tu.forward(taille)

def trace(taille):
    if taille > 0:
        dessin(taille)
        trace(taille - 10)

trace(200)
tu.ht()
tu.mainloop()
```

3. Boucles ou récursivité

Soit la fonction *compteur* suivante qui prend en paramètre un entier n et qui affiche à l'écran les entiers croissant de 0 à n inclus.

```
def compteur(n):
    for i in range(n + 1):
        print(i)
```

1. Écrire une version de cette fonction utilisant une boucle *while*.
2. Écrire une version récursive de cette fonction.

4. Rendre une fonction récursive

4.1. Fonction *pair*

Écrire une version récursive de la fonction *pair*, vue en cours, permettant de savoir si un entier est pair.

```python
def pair(n):
    while n > 0:
        n -= 2
    return n == 0
```

4.2. Algorithme d'Euclide

La valeur du PGCD (plus grand commun diviseur) de deux entiers naturels a et b peut être obtenue à l'aide de l'algorithme d'Euclide : $pgcd(a, b) = pgcd(b, r)$ où r est le reste de la division euclidienne de a par b. En voici une version itérative :

```python
def pgcd(a, b):
    if b == 0:
        return a
    while b != 0:
        a, b = b, a % b
    return a
```

4.3. Dessin avec Turtle

Tester le programme ci-dessous, puis en écrire une version récursive.

```python
import turtle as tu

couleurs = ['purple', 'blue', 'cyan', 'green', 'yellow', 'orange'
    , 'red']
tu.bgcolor('black')

def dessin():
    for i in range(210):
        tu.color(couleurs[i % 7])
        tu.forward(i)
        tu.right(50.8)
    tu.ht()
    tu.mainloop()

dessin()
```

5. Écrire un programme récursif

5.1. Puissance, première version

1. Écrire une fonction récursive *puissance* qui prend en paramètres un flottant x non nul et un entier naturel n et qui renvoie x^n. On se base sur la définition mathématique $x^0 = 1$ et $x^n = x \times x^{n-1}$ pour $n > 0$.
2. Modifier le code pour que la fonction soit récursive terminale (il est possible de s'inspirer de l'exemple du cours concernant la fonction factorielle).

5.2. Puissance, seconde version

Le but est toujours d'écrire une fonction puissance qui prend en paramètres un flottant x non nul et un entier naturel n et qui renvoie x^n, mais la méthode est différente.

On note que $x^0 = 1$ et on remarque que si $n = 2^k$ alors $x^n = x^{2k} = \left(x^2\right)^K$ et si $n = 2^{k+1}$ alors $x^n = x^{2k+1} = x\left(x^2\right)^k$. Le problème est divisé en deux sous-problèmes.

1. Quel est l'intérêt de cet algorithme par rapport au précédent ?
2. Écrire la fonction récursive *puissance* basée sur cet algorithme.
3. Modifier le code pour que la fonction soit récursive terminale.
4. Écrire une version itérative de cette fonction.

Capacités exigibles

- Écrire un programme récursif.
- Analyser le fonctionnement d'un programme récursif.