II - PLANCHE 3 Étude de fonctions récursives

1 On suppose implémentée la fonction `autre(c1, c2)`, qui prend en argument deux caractères différents `c1` et `c2` parmi `"A"`, `"B"` et `"C"`, et qui renvoie le caractère différent de `c1` et de `c2`.

1. Compléter la fonction `hanoi` ci-dessous pour que celle-ci affiche la liste des déplacements à effectuer pour résoudre le problème de Hanoï de difficulté `n`.

Code python

```
def hanoi(n, depart, arrivee):
    """ int, str, str -> None """
    if n == 0:
        pass
    else:
        autre_pilier = autre(depart, arrivee)
        ....................................
        ....................................
        ....................................
```

2. a. Écrire sur votre feuille l'arbre d'appel de l'instruction `hanoi(4)`.

b. Combien d'appels récursifs réalise-t-on pour résoudre le problème de Hanoï de difficulté 4 ?

c. Conjecturer le nombre d'appels nécessaires pour résoudre le problème de Hanoï de difficulté n.

d. On suppose que l'on peut effectuer un déplacement en une seconde. Combien de temps met cet algorithme pour résoudre le problème de difficulté 64 ?

2 On définit la fonction `f` par le code python donné ci-dessous.

Code Python

```
def f(x, y):
    """ int, int -> int """
    if x == y:
        return x
    elif x < y:
        return f(x, y - x)
    else:
        return f(x - y, y)
```

1. Sans exécuter le code, indiquer ce que renvoient `f(5, 15)` et `f(8, 20)`.

2. Indiquer plus généralement ce que calcule `f(x, y)` en fonction des paramètres x et y.

3 Dans chacun des cas suivants, expliquer ce que calcule la fonction.

```
def f1(n):
  if n <= 1:
    return 1
  else:
    return 2*f1(n - 1)
```

```
def f2(n):
 if n <= 1:
  return 1
 else:
  return 2**f2(n - 1)
```

```
def f3(n):
 if n <= 1:
  return 1
 else:
  return f3(n - 1)**2
```

4 Expliquer pourquoi le code python ci-dessous, censé calculer la factorielle, est incorrect. En proposer une correction.

Code Python

```
def f(n):
    """ int -> int"""
    if n == 0:
        return 1
    if n > 1:
        return n*f(n - 1)
```

5 **1.** On considère la fonction `f` définie par le code python suivant :

Code Python

```
def f(n):
    """ int -> None """
    if n == 0:
        print('--')
    else:
        print(n)
        f(n - 1)
```

Déterminer l'affichage réalisé lorsque l'on exécute l'instruction `f(3)`.

2. On considère la fonction g définie par le code python suivant :

Code Python

```
def g(n):
    """ int -> None """
    if n == 0:
        print('--')
    else:
        g(n - 1)
        print(n)
```

Déterminer l'affichage réalisé lorsque l'on exécute l'instruction `g(3)`.

6 On considère la fonction `f` suivante :

Code Python

```
def f(n):
    """ int -> int """
    if n == 1:
        print(f"f(1) renvoie 1")
        return 1
    else:
        print(f"Appel récursif de f({n - 1})")
        res = f(n - 1) + 2**n
        print(f"f({n}) renvoie {res}")
        return res
```

1. Déterminer ce qui sera affiché à l'écran lors de l'exécution de l'instruction `f(4)`. Combien d'appels récursifs ont été réalisés ?

2. Expliquer ce que la fonction calcule en fonction du paramètre n. Exprimer le nombre d'appels récursifs réalisés par `f(n)` en fonction de n.

7 On considère la fonction `f` suivante :

Code Python

```
def f(n):
    """ int -> int """
    print(f"Appel f({n})")
    if n == 1:
        print(f"f(1) renvoie 1")
        return 1
    else:
        res = f(n - 1) + f(n - 1)
        print(f"f({n}) renvoie {res}")
        return res
```

1. Déterminer ce qui sera affiché à l'écran lors de l'exécution de l'instruction `f(3)`. Combien d'appels récursifs ont été réalisés ?

2. Expliquer ce que la fonction calcule en fonction du paramètre n. Exprimer le nombre d'appels récursifs réalisés en fonction de n.

3. Proposer une modification de la fonction `f` afin que l'instruction `f(n)` réalise un nombre d'appel récursifs proportionnel à `n`.

8 **1.** Soient x et n deux nombres entiers. On note `pow(x, n)` le nombre x^n.

a. Pour quelle valeur de n le nombre `pow(x, n)` est-il indépendant de x ?

b. Exprimer `pow(x, n)` en fonction de `pow(x, n-1)`. Expliquer votre réponse.

c. En déduire le code d'une fonction récursive `pow` qui étant donné deux nombres entiers x et n renvoie x^n.

d. Exprimer le nombre d'appels récursifs réalisés par l'exécution de l'instruction `pow(x, n)` en fonction de n.

2. On donne le code de la fonction `pow_fast` :

Code Python

```
def pow_fast(x, n):
    """ int, int -> int
    Détermine x^n à l'aide de l'algorithme d'exponentiation rapide. """
    if n == 0:
        return 1
    else:
        p = pow_fast(x, n//2)
        if n%2 == 0:
            return p*p
        else:
            return p*p*x
```

a. Dresser l'arbre d'appel de l'instruction `pow_fast(3, 19)`.

b. Conjecturer l'expression du nombre d'appels récursifs réalisés par l'exécution de l'instruction `pow(x, n)` en fonction de n.

9 On définit la suite de Syracuse (u_n) de la manière suivante :

- le premier terme u_0 est un entier strictement positif ;
- pour tout $n \geq 1$:
  - si u_{n-1} est pair, alors $u_n = u_{n-1}/2$,
  - sinon $u_n = 3u_n + 1$.

1. Soit $u_0 = 10$. Calculer $u_1, \ldots, u_7$. Que constate-t-on ?

2. Définir une fonction python récursive `u` qui prend en entrée deux paramètres u_0 et n et qui renvoie u_n.

3. La conjecture de Syracuse affirme que pour tout entier u_0, il existe un entier n tel que $u_n = 1$.

a. Vérifier que cette conjecture est vraie pour les nombres $0 < u_0 \leq 10$.

b. On appelle **temps de vol** du nombre u_0 le premier indice n tel que $u_n = 1$. Sans utiliser de boucle `while`, écrire une fonction python récursive `temps_vol` qui prend en entrée un nombre positif u_0 et qui détermine le temps de vol de la suite de Syracuse de premier terme u_0.

c. Écrire une fonction python qui renvoie la liste des temps de vol des entiers compris entre 1 et 100.