

TD3 : Lecture de fonctions récursives

Compétences

- Lire et prévoir le résultat d'une fonction récursive
- Évaluer la terminaison d'une fonction récursive

Exercices

* **Ex. 1** — On considère les fonctions `foo1` et `foo2` suivantes :

```
1 def foo1(n):
2     if n == 0:
3         print(0)
4     else:
5         print(n)
6         foo1(n - 1)
```

```
1 def foo2(n):
2     if n == 0:
3         print(0)
4     else:
5         foo2(n - 1)
6         print(n)
```

Décrire précisément la pile d'exécution des fonctions `foo1` et `foo2` pour $n = 3$

* **Ex. 2** — On considère les fonctions `foo3` et `foo4` suivantes :

```
1 def foo3(n):
2     if n == 0:
3         print(0)
4     else:
5         print(n)
6         foo4(n - 1)
```

```
1 def foo4(n):
2     if n == 0:
3         print(0)
4     else:
5         foo3(n - 1)
6         print(n)
```

Décrire précisément la pile d'exécution des fonctions `foo3` et `foo4` pour $n = 3$

* **Ex. 3** — Un étudiant programme la fonction `fractionEgyptienne` suivante :

```
1 def fractionEgyptienne(a, b, t):
2     if a == 1:
3         t += [b]
4     fractionEgyptienne(1, b, t)
5     fractionEgyptienne(a - 1, b + 1, t)
6     fractionEgyptienne(a - 1, b*(b + 1), t)
```

1. L'appel `fractionEgyptienne(2, 5, [])` ne semble pas fonctionner et l'étudiant vous sollicite pour lui expliquer ce qui ne va pas. Qu'en pensez-vous?
2. Proposer à l'étudiant une correction possible.
3. Décrire précisément la pile d'exécution pour l'appel précédent.

* **Ex. 4** — On considère la fonction `powRec` suivante :

```
1 def powRec(a, n):
2     if ...
3         ...
4     return a * powRec(a, n-1)
```

1. Compléter la condition d'arrêt de cette fonction
2. Décrire précisément la pile d'exécution pour l'appel `powRec(2, 4)`

** **Ex. 5** — On considère la fonction `mystere` suivante :

```

1 def mystere(tab, g, d):
2     if g == d:
3         return tab[g]
4
5     else:
6         milieu = (g + d) // 2 - 1
7         if milieu % 2 == 1:
8             milieu += 1
9         if tab[milieu] != tab[milieu + 1] :
10            return mystere(tab, g, milieu)
11        else:
12            return mystere(tab, milieu + 2, d)

```

Décrire précisément la pile d'exécution de l'appel suivant : `mystere([1,1,6,6,7,9,9,3,3,0,0], 0, 10)`. Donner un nom explicite à cette fonction

**** Ex. 6 —** On considère la fonction `checkNumbers` suivante :

```

1 def checkNumbers(L, n, i, j) :
2     if i >= n - 1:
3         return True
4     else :
5         if j == n:
6             return checkNumbers(L, n, i+1, i+2)
7         else :
8             if L[i] == L[j]:
9                 return False
10            else:
11                return checkNumbers(L, n, i, j+1)

```

1. Décrire précisément la pile d'exécution pour l'appel de fonction `checkNumbers([2,3,6,2], 4, 0, 1)`
2. Décrire précisément la pile d'exécution pour l'appel de fonction `checkNumbers([2,3,6,1], 4, 0, 1)`
3. Que fait cette fonction?
4. Quelle est sa complexité?
5. Donner la version itérative de `checkNumbers`
6. Quelle est sa complexité