Nom

Interrogation
Récursivité

Note

Exercice 1: Codes très simples

Cet exercice très simple va me permettre de voir si vous avez compris la récursivité.

Question 1: Créez une fonction récursive Prod(L) qui renvoie le produit des termes de la liste L non vide

```
>>> Prod([1,2,3,4])
24
```

1-1

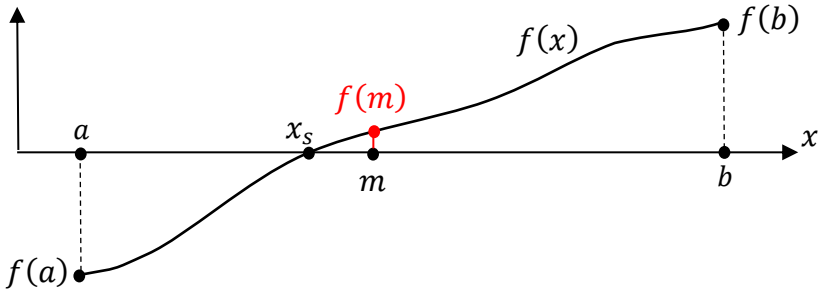
Question 2: Créez une fonction récursive Reverse(Char) qui renvoie l'inverse d'une chaine de caractères Char non vide

```
>>> Reverse('12345')
'54321'
```

1-2

Exercice 2: Exercices plus complexes

Soit une fonction f définie de $\mathbb{R}$ dans $\mathbb{R}$ dont on cherche la solution supposée unique et existante x_s sur $[a,b]$ telle que $f(x_s)=0$. On appelle x_n la solution numérique obtenue par dichotomie qui devra être précise à un ε près, c'est-à-dire $|x_s - x_n| \leq \varepsilon$.



Le principe de résolution consiste à calculer le milieu m de $[a,b]$ et d'étudier le signe de $f(a)f(m)$. Si $f(a)f(m) \leq 0$, le nouvel intervalle de recherche est $[a,m]$, sinon $[m,b]$.

Question 1: Proposer une fonction récursive Dichotomie(f,a,b,Eps) renvoyant x_n comme proposé ci-dessus

2-1

Question 2: Proposer une fonction récursive Insertion(t,L) qui renvoie une liste dans laquelle t est inséré à sa place dans L supposée triée par ordre croissant – L n'est pas modifiée – PAS de Dichotomie attendue

2-2

Exercice 3: Complexité d'algorithmes récursifs

Soit la suite définie par :

$$u_0 = 1, n \geq 1, u_{n+1} = \begin{cases} u_n + 1 \; si \; u_n < 1 \\ \frac{u_n}{2} \; sinon \end{cases}$$

On propose le code suivant :

```
def rec(n):
    if n==0:
        return 1
    else:
        Un_m1 = rec(n-1)
        if Un_m1 < 1:
            Un = Un_m1 + 1
        else:
            Un = Un_m1 / 2
        return Un
```

Question 1: Donner et démontrer la complexité de la fonction rec proposée

3-1

On propose maintenant le code que certains d'entre vous auraient pu réaliser :

```
def rec(n):
    if n==0:
        return 1
    else:
        if rec(n-1) < 1:
            Un = rec(n-1) + 1
        else:
            Un = rec(n-1) / 2
        return Un
```

Question 2: Donner et démontrer la complexité de la nouvelle fonction rec proposée

3-2

Question 3: Compléter le tableau suivant en précisant la complexité dans chaque cas

1	Auto-appel 1 fois au rang n-1 $C(n) = C(n-1) + O(n^{\alpha})$		
2	Auto-appel γ>1 fois au rang n-1 γ constant $C(n) = \gamma C(n-1) + O(n^{\alpha})$		
31	Auto-appel 1 fois au rang n/2 $C(n) = C\left(\frac{n}{2}\right) + O(n^{\alpha})$	$\alpha = 0$	
32		$\alpha \geq 1$	
41	Auto-appel γ>1 fois au rang n/γ γ constant $C(n) = \gamma C\left(\frac{n}{\gamma}\right) + O(n^{\alpha})$	$\alpha = 0$	
42		$\alpha = 1$	
43		$\alpha \geq 2$	
5	Auto-appel aux rangs n-1 et n-2 $C(n) = aC(n-1) + bC(n-2) + O(1)$		

3-3

Question 4: Pour chacun des algorithmes proposés, donner le cas du tableau précédent, la valeur des paramètres $(\alpha, \gamma \dots)$ et la complexité

Algorithme	Cas	Paramètres	Complexité $C(n)$

`def f(n):`
`  if n==1:`
`  return 1`
`  else:`
`  return f(n-1) + 2*f(n-2)`			

`def f(n):`
`  if n==1:`
`  return 1`
`  else:`
`  S = f(n-1)`
`  for i in range(n):`
`  S+= i`
`  return S`			

`def f(n):`
`  if n==1:`
`  return 1`
`  else:`
`  return f(n-1) + f(n-1)`			

`def f(n):`
`  if n==1:`
`  return 1`
`  else:`
`  return f(n-1)`			

`def f(n):`
`  if n==1:`
`  return 1`
`  else:`
`  S = f(n-1) + f(n-1)`
`  for i in range(n):`
`  S+= i`
`  return S`			

`def f(n):`
`  if n==1:`
`  return 1`
`  else:`
`  S = f(n-1) + f(n-1) + f(n-1)`
`  for i in range(n):`
`  S += i`
`  return S`			

`def f(n):`
`  if n==1:`
`  return 1`
`  else:`
`  n1 = n//2`
`  n2 = n-n//2`
`  S = f(n1) + 2*f(n2)`
`  for i in range(n):`
`  S += i`
`  return S`			

`def f(n):`
`  if n==1:`
`  return 1`
`  else:`
`  N = n//2`
`  return 2*f(N)`			

`def f(n):`
`  if n==1:`
`  return 1`
`  else:`
`  S = 0`
`  a = f(n-1)`
`  for i in range(10):`
`  S += a/n`
`  return S`			

3-4