Interrogation d'ITC n°2A

Semaine du 27/11/2023

Durée : 35 minutes

Nom :		**Prénom :**	

Consignes

- Les codes doivent être présentés en mentionnant explicitement l'indentation, au moyen par exemple de barres verticales sur la gauche.
- Pour qu'un code soit compréhensible, il convient de choisir des noms de variables les plus explicites possibles.
- La performance du code proposé à chaque question entrera dans l'évaluation, de-même que l'utilisation de boucles appropriées.

/20

Présentation (écriture, propreté, marqueurs d'indentation du code, ...) : /1

Exercice 1 Pour démarrer

1. /3 On rappelle la définition de la factorielle d'un entier n, notée $n!$:

$$\begin{cases} 0! &= 1 \\ n! &= n(n-1)(n-2)\cdots 1 \quad \text{si } n > 0. \end{cases}$$

Écrire le script d'une fonction `fact(n:int)->int` qui renvoie $n!$, pour $n \geq 0$, de manière itérative.

2. Soit $a \in \mathbb{R}^{+\star}$. On considère la suite (u_n) définie par :

$$u_0 = a, \quad \forall n \geqslant 0, \quad u_{n+1} = \frac{1}{2}\left(u_n + \frac{a}{u_n}\right).$$

On désire écrire le script de la fonction `u(a:float, n:int)->float` qui prend en argument un flottant `a` et un entier naturel `n` et qui renvoie la valeur de u_n.

3. Un élève propose le script suivant :

```
def u(a:float, n:int)->float:
    if n == 0:
        return a
    else:
        return 0.5*(u(a, n-1)+a/u(a, n-1))
```

et s'étonne du temps élevé nécessaire à l'exécution pour $n \geq 25$.

3.1) /1 Soit $n \in \mathbb{N}$. On note a_n le nombre d'appels de la fonction `u` nécessaires au calcul du terme u_n, avec la convention $a_0 = 1$. Déterminer l'expression de (a_n) en fonction de n, en cherchant une relation de récurrence entre a_n et a_{n+1}. Expliquer le fonctionnement observé.

3.2) /2 Écrire ci-dessous une nouvelle version récursive `u_bis` de la fonction `u` qui élimine le défaut de la première version.

4. /2 On donne ci-dessous le script d'une fonction `f(L:list)` qui prend en argument une liste de nombres.

```
def f(L):
    if len(L) == 0:
        return 0
    else:
        return L[0] + f(L[1:])
```

Décrire en une phrase le rôle de cette fonction.

Exercice 2 **Recherche dichotomique du maximum d'une liste** Pour déterminer le plus grand élément d'une liste $L = [\ell_0, \ell_1, \dots, \ell_{n-1}]$ de n éléments, on peut utiliser la méthode dichotomique et récursive suivante, pour toute liste ayant au moins deux éléments :

- déterminer le maximum de la première moitié de la liste,
- déterminer le maximum de la seconde moitié de la liste,
- comparer les deux *maxima* et renvoyer le plus grand des deux.

Le cas de la liste à un élément est un cas trivial qui sert de cas terminal pour la programmation récursive.

Écrire une fonction `max_dicho` qui renvoie le maximum de la liste `L` selon ce principe. *La fonction ne devra pas se servir de la fonction `max` de Python* /3

Exercice 3 **Multiplication de grands nombres** Lorsqu'on multiple deux grands nombres n et m, le nombre d'opérations entre chiffres est assez important (en posant l'opération, on multiple chaque chiffre de n par chaque chiffre de m), ce qui rend l'opération assez couteuse en temps, et très souvent plus lente qu'une addition ou une soustraction entre nombres. Pour diminuer le temps d'exécution, on peut utiliser l'algorithme de KARATSUBA décrit ci dessous :

- si n ou m sont constitués d'un seul chiffre (c'est-à-dire $(m,n) \in [\![0,9]\!]^2$), on calcule directement le produit nm,
- sinon,
  - on écrit les nombres n et m sous la forme $\begin{cases} n = a \times 10^k + b \\ m = c \times 10^k + d \end{cases}$ où a, b, c, d sont des entiers, et où k est un entier choisi pour « couper en deux » le plus grand des entiers n ou m, plus précisément on prendra pour k le quotient de la division euclidienne de N par 2, où N désigne le nombre de chiffres du plus grand entier entre n et m ; a, c (resp. b, d) sont alors les quotients (resp. les restes) des divisions euclidiennes de n et m par 10^k,
  - on calcule alors récursivement les produits ac, bd et $(a-b)(c-d)$ selon cette même méthode,
  - on calcule le produit nm en utilisant la formule (qui découle d'un simple calcul du produit) :

$$nm = ac \times 10^{2k} + (ac + bd - (a-b)(c-d)) \times 10^k + bd.$$

Par exemple, si on cherche à multiplier $n = 123$ et $m = 45678$, m étant l'entier le plus grand, composé de 5 chiffres, on prendra $k = 2$, d'où $n = 1 \times 10^2 + 23$, $m = 456 \times 10^2 + 78$. Ainsi on aura $a = 1, b = 23, c = 456$ et $d = 78$.

La multiplication par 10^k ou 10^{2k} revenant à un simple décalage vers la gauche, on a transformé le calcul d'un produit de deux grands nombres en trois calculs de produits de nombres de taille moitié. On peut alors montrer que cette méthode est plus efficace que le calcul direct, à partir d'une certaine « taille » de n et m.

1. /2 On rappelle que pour n entier, l'instruction `str(n)` permet de transformer l'entier en sa chaine de caractère correspondante (`str(425)` renvoie `'425'`). En utilisant cette possibilité, écrire le script d'une fonction `taille_max(n:int,m:int)->int` qui renvoie le nombre de chiffres qui compose le plus grand des deux entiers n et m.

2. /3 On peut montrer que pour n entier, le nombre de chiffres qui compose n est le plus petit entier naturel ℓ tel que : $10^\ell > n$. En déduire le script d'une seconde fonction `taille_maxbis(n:int,m:int)->int` qui renvoie le nombre de chiffres qui compose le plus grand des deux entiers n et m.

3. /3 Écrire le script d'une fonction récursive `Karatsuba(n:int, m:int)->int` qui renvoie le produit des entiers n et m selon la méthode de KARATSUBA. *La fonction ne devra bien sûr pas utiliser le symbole $*$, sauf pour des nombres ne possédant qu'un seul chiffre. En revanche, les calculs des puissances de 10 en utilisant $**$ sont autorisés, ainsi que la multiplication par une puissance de* 10.

Correction
de l'Interrogation N°2A
MPSI

Solution 1

1.
```
def fact(n:int)->int:
    P = 1
    for k in range(1, n+1):
        P *= k
    return P
```

2. **2.1)** Dans le script proposé, on a deux appels récursifs, soit $a_{n+1} = 2a_n$. Avec $a_0 = 1$, on a donc $\boxed{a_n = 2^n}$. Le temps d'exécution croit de manière exponentielle avec n.

2.2) Pour limiter le nombre d'appels, on peut écrire :
```
def u_bis(a:float,n:int)->float:
    if n == 0:
        return 1
    else:
        v = u_bis(a, n-1)
        return 0.5*(v+a/v)
```

3. La fonction renvoie la somme des éléments de la liste `L` si `L` est non-vide, et 0 sinon.

Solution 2

```
def max_dicho(L:list):
    if len(L) == 1:
        return L[0]
    else:
        m = len(L)//2
        max_g = max_dicho(L[:m])
        max_d = max_dicho(L[m:])
        if max_g > max_d:
            return max_g
        else:
            return max_d
```

```
>>> max_dicho([-1, -3, 4, 5, 2])
5
```

Solution 3

1.
```
def taille_max(n:int,m:int):
    if n > m:
        return len(str(n))
    else:
        return len(str(m))
```
```
>>> taille_max(12, 123)
3
```

2.
```
def taille_maxbis(n:int,m:int)->int:
    M = max(n, m)
    l = 0
    while 10**l <= M:
        l += 1
    return l
```
```
>>> taille_maxbis(123, 12)
3
```

3.
```
def Karatsuba(n,m):
    if n < 10 or m < 10:
        return n*m
    else:
        N = taille_max(n, m)
        k = N // 2
        puiss = 10**k
        a = n // puiss
        b = n % puiss
        c = m // puiss
        d = m % puiss

        ac = Karatsuba(a, c)
        bd = Karatsuba(b, d)
        prod = Karatsuba(a-b, c-d)

        return ac*puiss**2+(ac+bd-prod)*puiss+bd
```
```
>>> Karatsuba(123, 45678)
5618394
```

Interrogation d'ITC n°2B

Semaine du 27/11/2023

Durée : 35 minutes

Nom :		Prénom :	

Consignes

- Les codes doivent être présentés en mentionnant explicitement l'indentation, au moyen par exemple de barres verticales sur la gauche.
- Pour qu'un code soit compréhensible, il convient de choisir des noms de variables les plus explicites possibles.
- La performance du code proposé à chaque question entrera dans l'évaluation, de-même que l'utilisation de boucles appropriées.

/20

Présentation (écriture, propreté, marqueurs d'indentation du code, ...) : /1

Exercice 4 Pour démarrer

1. /3 On rappelle la définition de la factorielle d'un entier n, notée $n!$:

$$\begin{cases} 0! &= 1 \\ n! &= n(n-1)(n-2)\cdots 1 \quad \text{si } n > 0. \end{cases}$$

Écrire le script d'une fonction `fact(n:int)->int` qui renvoie $n!$, pour $n \geq 0$, de manière récursive.

2. Soit $a \in \mathbb{R}^{+\star}$. On considère la suite (u_n) définie par :

$$u_0 = a, \quad \forall n \geqslant 0, \quad u_{n+1} = \frac{1}{2}\left(u_n + \frac{a}{u_n}\right).$$

On désire écrire le script de la fonction `u(a:float, n:int)->float` qui prend en argument un flottant `a` et un entier naturel `n` et qui renvoie la valeur de u_n.

3. Un élève propose le script suivant :

```
def u(a:float, n:int)->float:
    if n == 0:
        return a
    else:
        return 0.5*(u(a, n-1)+a/u(a,n-1))
```

et s'étonne du temps élevé nécessaire à l'exécution pour $n \geq 25$.

3.1) /1 Soit $n \in \mathbb{N}$. On note a_n le nombre d'appels de la fonction `u` nécessaires au calcul du terme u_n, avec la convention $a_0 = 1$. Déterminer l'expression de (a_n) en fonction de n, en cherchant une relation de récurrence entre a_n et a_{n+1}. Expliquer le fonctionnement observé.

3.2) /2 Écrire ci-dessous une nouvelle version `u_bis`, mais itérative cette fois, de la fonction `u`.

4. /2 On donne ci-dessous le script d'une fonction `f(L:list)` qui prend en argument une liste de nombres.

```
def f(L):
    if len(L) == 0:
        return 1
    else:
        return f(L[:-1])*L[-1]
```

Décrire en une phrase le rôle de cette fonction.

Exercice 5 **Recherche dichotomique du minimum d'une liste** Pour déterminer le plus petit élément d'une liste $L = [\ell_0, \ell_1, \dots, \ell_{n-1}]$ de n éléments, on peut utiliser la méthode dichotomique et récursive suivante, pour toute liste ayant au moins deux éléments :

- déterminer le minimum de la première moitié de la liste,
- déterminer le minimum de la seconde moitié de la liste,
- comparer les deux *minima* et renvoyer le plus petit des deux.

Le cas de la liste à un élément est un cas trivial qui sert de cas terminal pour la programmation récursive.

Écrire une fonction `min_dicho` qui renvoie le minimum de la liste `L` selon ce principe. *La fonction ne devra pas se servir de la fonction `min` de Python* /3

Exercice 6 **Multiplication de grands nombres** Lorsqu'on multiple deux grands nombres n et m, le nombre d'opérations entre chiffres est assez important (en posant l'opération, on multiple chaque chiffre de n par chaque chiffre de m), ce qui rend l'opération assez couteuse en temps, et très souvent plus lente qu'une addition ou une soustraction entre nombres. Pour diminuer le temps d'exécution, on peut utiliser l'algorithme de KARATSUBA décrit ci dessous :

- si n ou m sont constitués d'un seul chiffre (c'est-à-dire $(m,n) \in [\![0,9]\!]^2$), on calcule directement le produit nm,
- sinon,
  - on écrit les nombres n et m sous la forme $\begin{cases} n = a \times 10^k + b \\ m = c \times 10^k + d \end{cases}$ où a, b, c, d sont des entiers, et où k est un entier choisi pour « couper en deux » le plus grand des entiers n ou m, plus précisément on prendra pour k le quotient de la division euclidienne de N par 2, où N désigne le nombre de chiffres du plus grand entier entre n et m ; a, c (resp. b, d) sont alors les quotients (resp. les restes) des divisions euclidiennes de n et m par 10^k,
  - on calcule alors récursivement les produits ac, bd et $(a-b)(c-d)$ selon cette même méthode,
  - on calcule le produit nm en utilisant la formule (qui découle d'un simple calcul du produit) :

$$nm = ac \times 10^{2k} + (ac + bd - (a-b)(c-d)) \times 10^k + bd.$$

Par exemple, si on cherche à multiplier $n = 123$ et $m = 45678$, m étant l'entier le plus grand, composé de 5 chiffres, on prendra $k = 2$, d'où $n = 1 \times 10^2 + 23$, $m = 456 \times 10^2 + 78$. Ainsi on aura $a = 1, b = 23, c = 456$ et $d = 78$.

La multiplication par 10^k ou 10^{2k} revenant à un simple décalage vers la gauche, on a transformé le calcul d'un produit de deux grands nombres en trois calculs de produits de nombres de taille moitié. On peut alors montrer que cette méthode est plus efficace que le calcul direct, à partir d'une certaine « taille » de n et m.

1. /2 On rappelle que pour n entier, l'instruction `str(n)` permet de transformer l'entier en sa chaine de caractère correspondante (`str(425)` renvoie `'425'`). En utilisant cette possibilité, écrire le script d'une fonction `taille_max(n:int,m:int)->int` qui renvoie le nombre de chiffres qui compose le plus grand des deux entiers n et m.

2. /3 On peut montrer que pour n entier, le nombre de chiffres qui compose n est le plus petit entier naturel ℓ tel que : $10^\ell > n$. En déduire le script d'une seconde fonction `taille_maxbis(n:int,m:int)->int` qui renvoie le nombre de chiffres qui compose le plus grand des deux entiers n et m.

3. /3 Écrire le script d'une fonction récursive `Karatsuba(n:int, m:int)->int` qui renvoie le produit des entiers n et m selon la méthode de KARATSUBA. *La fonction ne devra bien sûr pas utiliser le symbole $*$, sauf pour des nombres ne possédant qu'un seul chiffre. En revanche, les calculs des puissances de 10 en utilisant $**$ sont autorisés, ainsi que la multiplication par une puissance de* 10.

Correction
de l'Interrogation N°2B
MPSI

Solution 4

1.
```
def fact(n:int)->int:
    if n == 0:
        return 1
    else:
        return n*fact(n-1)
```

2. **2.1)** Dans le script proposé, on a deux appels récursifs, soit $a_{n+1} = 2a_n$. Avec $a_0 = 1$, on a donc $a_n = 2^n$. Le temps d'exécution croit de manière exponentielle avec n.

2.2) Pour limiter le nombre d'appels, on peut écrire :
```
def u_bis(a:float,n:int)->float:
    u = a
    for _ in range(1, n+1):
        u = (1/2)*(u+a/u)
    return u
```

3. La fonction renvoie le produit des éléments de la liste `L` si `L` est non-vide, et 1 sinon.

Solution 5

```
def min_dicho(L:list):
    if len(L) == 1:
        return L[0]
    else:
        m = len(L)//2
        min_g = min_dicho(L[:m])
        min_d = min_dicho(L[m:])
        if min_g > min_d:
            return min_d
        else:
            return min_g
```

```
>>> min_dicho([-1, -3, 4, 5, 2])
-3
```

Solution 6

1.
```
def taille_max(n:int,m:int):
    if n > m:
        return len(str(n))
    else:
        return len(str(m))
>>> taille_max(12, 123)
3
```

2.
```
def taille_maxbis(n:int,m:int)->int:
    M = max(n, m)
    l = 0
    while 10**l <= M:
        l += 1
    return l
>>> taille_maxbis(123, 12)
3
```

3.
```
def Karatsuba(n,m):
    if n < 10 or m < 10:
        return n*m
    else:
        N = taille_max(n, m)
        k = N // 2
        puiss = 10**k
        a = n // puiss
        b = n % puiss
        c = m // puiss
        d = m % puiss

        ac = Karatsuba(a, c)
        bd = Karatsuba(b, d)
        prod = Karatsuba(a-b, c-d)

        return ac*puiss**2+(ac+bd-prod)*puiss+bd
>>> Karatsuba(123, 45678)
5618394
>>> 123*45678
```