Consulter le cours à la page allophysique.com//docs/NSI/langages/page2/

Part 1

Une fonction x + 1/x

Considérons la suite u_n

$$n \in \mathbb{N}$$

définie par

$$u_0 = 1$$

et la relation de recurrence :

$$u_{n+1} = u_n + \frac{1}{u_n}$$

1.1 Compléter le script récursif de cette fonction que l'on nommera `u_rec`. Ecrire egalement sont prototypage.

```
def u_rec(n):
    """...
          ...
          ...
          ...
          ...
          ...
    """
    ...
    ...
    ...
    ...
```

On utilisera les conventions suivantes pour le calcul de la complexité :

Opérations	Poids
+, −, ×, ÷	1 unité de temps
Affectation	1 unité de temps
Appel de fonction	1 unité de temps
Comparaison	1 unité de temps

1.1.1 Quelle est la loi de recurence sur le nombre d'instructions T(n) en fonction de T(n-1) pour cette fonction.

1.1.2 En déduire la complexité asymptotique O(g(n)).

1.2 Dans votre script, où pourrait-on ajouter un test d'assertion pour protéger la fonction d'une entrée non conforme (par exemple avec n<0) ? Ecrire l'instruction de ce test d'assertion.

1.3 On propose un autre script pour cette fonction :

```
def u_rec (n) :
    if n==0:
        return 1
    else :

        x=u_rec (n-1) # variable locale
```

```
7           return x+1/x
```

1.3.1 Cette fonction, est-elle plus efficace ? C'est à dire, est-elle de complexité inferieure ? Justifiez rapidement.

Part 2

Les tours de Hanoï

Voir le cours en ligne sur la compléxité. L'exemple y est longuement traité.

2.1 Principe

On considère trois tiges plantées dans une base. Au départ, sur la première tige sont enfilées N disques de plus en plus étroits. Le but du jeu est de transférer les N disques sur la troisième tige en conservant la configuration initiale.

2.2 algorithme récursif

L'algorithme récursif pour ce problème est étonnament réduit :

```
1  def hanoi(N,d,i,a):
2      """N disques doivent être déplacés de d vers a
3      Params:
4      N : int
5          nombre de disques
6      d: int
7          depart (vaut 1 au debut)
8      i: int
9          intermediaire (vaut 2 au debut)
10     a: int
11         fin (vaut 3 au debut)
12     Exemple:
13     lancer avec
14     >>> hanoi(3,1,2,3)
15     """
16     if N==1 :
17         print('deplacement de {} vers {}'.format(d,a))
18     else:
19         hanoi(N-1,d,a,i)
20         hanoi(1,d,i,a)
21         hanoi(N-1,i,d,a)
```

Résultat

```
1  >>> hanoi(3,1,2,3)
2  deplacement de 1 vers 3
3  deplacement de 1 vers 2
4  deplacement de 3 vers 2
5  deplacement de 1 vers 3
6  deplacement de 2 vers 1
7  deplacement de 2 vers 3
8  deplacement de 1 vers 3
```

2.2.1 Vérifier que pour N = 2 disques, il y a 3 déplacements, que pour 3 disques, il y en a 7, et que pour 4 disques, il y en a 15.

2.2.2 Proposez une loi de recurence entre le nombre de déplacements T(N) pour N disques, et le nombre de déplacements T(N-1) pour N-1 disques.

2.2.3 Cette loi, est-elle conforme à celle que l'on aurait déduite de l'etude de la complexité pour l'algorithme recursif?

Part 3

Extrait du sujet de Bac 2022 Polynesie : Ex 1

Cet exercice traite du thème «programmation», et principalement de la récursivité.

On rappelle qu'une chaîne de caractères peut être représentée en Python par un texte entre guillemets "" et que :

- la fonction `len` renvoie la longueur de la chaîne de caractères passée en paramètre ;
- si une variable `ch` désigne une chaîne de caractères, alors `ch[0]` renvoie son premier caractère, `ch[1]` le deuxième, etc. ;
- l'opérateur + permet de concaténer deux chaînes de caractères.

Exemples :

```
>>> texte = "bricot"
>>> len(texte)
6
>>> texte[0]
"b"
>>> texte[1]
"r"
>>> "a" + texte
"abricot"
```

On s'intéresse dans cet exercice à la construction de chaînes de caractères suivant certaines règles de construction.

Règle A : une chaîne est construite suivant la règle A dans les deux cas suivants :

- soit elle est égale à "a" ;
- soit elle est de la forme "a"+chaine+"a", où chaine est une chaîne de caractères construite suivant la règle A.

Règle B : une chaîne est construite suivant la règle B dans les deux cas suivants :

- soit elle est de la forme "b"+chaine+"b", où chaine est une chaîne de caractères construite suivant la règle A ;
- soit elle est de la forme "b"+chaine+"b", où chaine est une chaîne de caractères construite suivant la règle B.

On a reproduit ci-dessous l'aide de la fonction `choice` du module `random`.

```
>>>from random import choice
>>>help(choice)
Help on method choice in module random:
choice(seq) method of random.Random instance
 Choose a random element from a non-empty sequence.
```

La fonction `A()` ci-dessous renvoie une chaîne de caractères construite suivant la règle A, en choisissant aléatoirement entre les deux cas de figure de cette règle.

```
def A():
  if choice([True, False]):
    return "a"
  else:
    return "a" + A() + "a"
```

3.1 Question 1

A. Cette fonction est-elle récursive ? Justifier.

B. La fonction `choice([True, False])` peut renvoyer `False` un très grand nombre de fois consécutives. Expliquer pourquoi ce cas de figure amènerait à une erreur d'exécution. Dans la suite, on considère une deuxième version de la fonction A. À présent, la fonction prend en paramètre un entier `n` tel que, si la valeur de `n` est négative ou nulle, la fonction renvoie `"a"`. Si la valeur de `n` est strictement positive, elle renvoie une chaîne de caractères construite suivant la règle A avec un `n` décrémenté de 1, en choisissant aléatoirement entre les deux cas de figure de cette règle.

```
def A(n):
  if ... or choice([True, False]) :
    return "a"
  else:
    return "a" + ... + "a"
```

3.2 Question 2

A. Recopier sur la copie et compléter aux emplacements des points de suspension `...` le code de cette nouvelle fonction `A`.

B. Justifier le fait qu'un appel de la forme `A(n)` avec `n` un nombre entier positif inférieur à 50, termine toujours.

On donne ci-après le code de la fonction récursive `B` qui prend en paramètre un entier `n` et qui renvoie une chaîne de caractères construite suivant la règle B.

```
def B(n):
  if n <= 0 or choice([True, False]):
    return "b" + A(n-1) + "b"
  else:
    return "b" + B(n-1) + "b"
```

On admet que :

- les appels `A(-1)` et `A(0)` renvoient la chaîne `"a"` ;
- l'appel `A(1)` renvoie la chaîne `"a"` ou la chaîne `"aaa"` ;
- l'appel `A(2)` renvoie la chaîne `"a"`, la chaîne `"aaa"` ou la chaîne `"aaaaa"`.

3.3 Question 3

Donner toutes les chaînes possibles renvoyées par les appels `B(0)`, `B(1)` et `B(2)`.

Part 4

Extrait du Sujet Métropole Sept 1 : 2021 - Exercice 2

Principaux thèmes abordés : algorithmique (recherche dichotomique) et langages et programmation (récursivité)

On veillera à mettre sur la copie toutes les réponses.

4.1 Partie A : La recherche dichotomique

1. La recherche d'un élément dans un tableau avec une méthode dichotomique ne peut se faire que si le tableau est trié.

a. Vrai

b. Faux

2. Le coût d'un algorithme de recherche dichotomique est :

a. Constant : Complexité O(1)

b. Linéaire : Complexité O(n)

c. Logarithmique : Complexité O(log(n))

3. Justifier pourquoi l'entier `findeb` est un *variant de boucle* qui montre la terminaison du programme de recherche dichotomique de l'annexe 1 de l'exercice 2.

4.2 Partie B : La recherche dichotomique itérative

Le programme de recherche dichotomique de l'annexe 1 de l'exercice 2 est utilisé pour effectuer des recherches dans une liste.

Dans l'ensemble de cette partie, on considère la liste :

`Lnoms = ["alice", "bob","etienne", "hector", "lea", "nathan", "paul"]`.

2. En Python, l'opérateur `//` donne le quotient de la division euclidienne de deux nombres entiers. Proposer un algorithme pour obtenir ce quotient.

3. Donner la trace complète de l'exécution `rechercheDicho("lea", Lnoms)` en complétant le tableau ci-dessous sur votre copie :

Debut	Fin	M	condition deb <= fin	valeur renvoyée

4.3 Partie C : La recherche dichotomique récursive

1. Donner la définition d'une fonction récursive en programmation.

2. Écrire en langage naturel ou en python, l'algorithme de recherche dichotomique d'un élément dans une liste, triée de façon croissante, en utilisant une méthode récursive. Il renverra `True` si l'objet a été trouvé, `False` sinon.

4.4 ANNEXE 1

On considère la fonction de recherche dichotomique suivante :

```
1 def rechercheDicho (elem, liste):
2   """
3   Cette fonction indique si un élément se trouve dans un
4   tableau.
5   Elle utilise la méthode de recherche dichotomique.
6   Elle prend en arguments :
7   - elem : élément à rechercher de type string
```

```
  - liste : liste d'éléments de type string triée
  par ordre croissant
  Elle renvoie un booléen correspondant à la présence ou
  non de l'élément
  """
  deb = 0
  fin = len(liste)-1
  m = (deb+fin)//2
  while deb <= fin :
    if liste[m] == elem :
      return True
    elif liste[m] > elem :
      fin = m-1
    else :
      deb = m+1
    m = (deb+fin)//2
  return False
```

Part 5

Autres exercices

5.1 Exponentiation

Etudions l'exponentiation à travers deux exemples.

```
def exp1(n,x) :
  """
  programme qui donne x^n en sortie sans utiliser **
  n : entier
  x : reel
  exp1 : reel
  """
  acc=1
  for i in range(1,n+1):
    acc*=x
  return acc

def exp2(n,x):
    """
    n : entier
    x : reel
    exp1 : reel
    """
    if n==0 : return 1
    else : return exp2(n-1,x)*x
```

1. Combien de produits sont necessaires pour calculer une puissance n-ième avec la fonction `exp1` ?
2. Pour la fonction `exp2` : Soit u_n le nombre de produits nécessaires pour calculer une puissance n-ième. Quelle est la relation de récurrence vérifiée par u~n+1 ?

$$u_{n+1} = u_n + \ldots$$

3. En déduire la complexité pour ces 2 fonctions.