Correction du Partiel

Jeudi 9 novembre 2017

Exercice 1 [2 points]

1. Soient les fonctions Python suivantes :

```
def calculPerimetre(largeur, longueur):
    perimetre = 2*(largeur+longueur)

def fonctionPerimetre(largeur, longueur):
    return 2*(largeur+longueur)

def trouverPerimetre(largeur, longueur):
    print(2*(largeur+longueur))
```

Quels vont être les affichages des instructions suivantes ?

(a) `print(calculPerimetre(4,5))`

Solution :
```
None
```

(b) `print(fonctionPerimetre(4,5))`

Solution :
```
18
```

(c) `print(trouverPerimetre(4,5))`

Solution :
```
18
None
```

(d)
```
if (calculPerimetre(4,5) == 18):
    print("Le périmètre vaut 18")
```

Solution :
Cela n'affiche rien.

2. Soit le programme Python suivant :

```
def maximum(x,y):
    if x>y:    return x
    elif x==y: return "Les deux nombres sont égaux"
    else:      return y

a=int(input("Saisir la valeur de a :"))
b=int(input("Saisir la valeur de b :"))
c=int(input("Saisir la valeur de c :"))
print("Le maximum est :", a+maximum(b,c))
```

(a) Qu'affiche le programme si l'utilisateur saisit la valeur `3` pour `a`, `2` pour `b` et `4` pour `c` ?

Solution :

```
Le maximum est : 7
```

(b) Qu'affiche le programme si l'utilisateur saisit la valeur `3` pour `a`, `4` pour `b` et `4` pour `c` ?

Solution :

```
Erreur car int+str impossible
```

3. Soient les fonctions suivantes :

```
def calculSuivant(n):
    if (n%2==0):
        return n//2
    else:
        return (3*n)+1

def syracuse(n):
    while (n!=1):
        n = calculSuivant(n)
        print(n)
```

Quels vont être les affichages des instructions suivantes :

(a) `syracuse(2)`

Solution :

```
1
```

(b) `syracuse(8)`

Solution :

```
4
2
1
```

Exercice 2 [4 points]

On cherche à calculer e^x à l'aide des développements limités. On sait que lorsque K tend vers l'infini :

$$e^x = \sum_{n=0}^{K} \frac{x^n}{n!}$$

où $n! = 1 * 2 * \ldots * (n-1) * n$ et $0! = 1$.

Le programme ci-dessous lit un nombre saisi au clavier par l'utilisateur, et affiche la plus petite valeur de K qui permet de calculer l'exponentiel de ce nombre avec une précision de $0{,}01$. Mais ce programme comporte 11 erreurs ! Il s'agit de le corriger avec un stylo de couleur directement sur le programme ci-dessous : utiliser les lignes vides pour mettre une version corrigée de la ligne précédente comportant une (ou des) erreur(s) ou ajouter d'éventuelles instructions manquantes.

```
from math import exp
x=input(Donner x)
somme=1
puissance=x
epsilon='0.01'
n=2
while abs(math.exp(x)-somme)<epsilon
    puissance=puissance*x
    fact=fact*n
    somme=puissance/fact
print('K='n)
```

Solution :

```
import math
x=float(input('Donner x '))
somme=1+x
puissance=x
fact=1
epsilon=0.01
n=2
while abs(math.exp(x)-somme)>epsilon:
    puissance=puissance*x
    fact=fact*n
    somme=somme+puissance/fact
    n=n+1
print('K=',n)
```

Exercice 3 [3 points]

On joue à lancer 3 dés : un dé rouge à 6 faces (numérotées de 1 à 6), un dé vert à 8 faces (numérotées de 1 à 8) et un dé bleu à 10 faces (numérotées de 1 à 10). On veut déterminer combien il y a de façons d'obtenir un nombre n donné en additionnant les trois faces des dés. Par exemple, il y a une seule façon d'obtenir 3 qui est de faire 1 avec le dé vert, 1 avec le dé rouge et 1 avec le dé bleu. Il y a 3 façons d'obtenir 4 :

— 1 avec le dé vert, 1 avec le dé rouge et 2 avec le dé bleu, ou bien,
— 1 avec le dé vert, 2 avec le dé rouge et 1 avec le dé bleu, ou bien,
— 2 avec le dé vert, 1 avec le dé rouge et 1 avec le dé bleu.

Écrire le programme **Python** qui implémente l'algorithme suivant :

1. demander à l'utilisateur de saisir une valeur entière ;
2. à l'aide de boucles imbriquées, compter le nombre de possibilités permettant d'obtenir ce nombre comme somme des dés rouge, vert et bleu ;
3. afficher le résultat.

Exemple d'exécution :

affichage Entrez un nombre entier :

saisie 5

affichage Il y a 6 façon(s) de faire 5 avec ces trois dés.

Solution :

```
n = int(input("Entrez un nombre :"))
compteur = 0
deRouge = 1
while deRouge<7:
    deVert = 1
    while deVert<9:
        deBleu = 1
        while deBleu<11:
            if deVert + deRouge + deBleu == n:
                compteur += 1
            deBleu += 1
        deVert += 1
    deRouge += 1
print("Il y a", compteur, "façon(s) de faire", n,
      "avec ces trois dés.")
```

Exercice 4 [5 points]

1. Étant donnée une séquence $a_1, a_2, \ldots, a_m$ de m nombres stockée dans un tableau `tab` de taille n, avec $m < n$, écrire en **pseudo-code une fonction** qui permet d'ajouter un nouvel élément x dans cette séquence. Cette fonction prendra en argument le tableau, l'élément x à insérer et la position `p` (cela signifiant qu'après l'insertion, l'élément se trouve en position `p`). Elle retournera le tableau modifié. **On pourra supposer que la position passée en argument est toujours valide.**

2. Écrire un algorithme permettant à l'utilisateur de saisir un nombre quelconque $m < n$ de valeurs qui initialiseront le tableau `tab`. Pour simplifier, nous supposerons que le tableau est de taille $n = 2m$. L'utilisateur doit d'abord saisir le nombre m de valeurs souhaitées puis saisir toutes ces valeurs. Ensuite l'algo demandera à l'utilisateur quel est l'élément `x` à ajouter et quelle est la position `p` à laquelle il faut insérer le nouvel élément et utilisera la fonction de la question précédente pour réaliser l'insertion. Une fois l'insertion réalisée, le programme affichera la nouvelle séquence et sa longueur. L'utilisateur doit pouvoir insérer autant d'éléments qu'il le souhaite tant que la taille de la séquence reste inférieure à $2m$ (dans le cas contraire, l'algorithme devra préciser à l'utilisateur que le tableau est plein).

Solution :

NB : On ne sait pas combien il y a d'éléments significatifs dans le tableau. On décale donc tous les éléments de la position p jusqu'à la fin.

```
FONCTION insertion(T TYPE TABLEAU DE NOMBRE, x TYPE NOMBRE,
                   p TYPE NOMBRE) TYPE TABLEAU DE NOMBRE
    VARIABLES_LOCALES
        i TYPE NOMBRE
    DEBUT
        i <- TAILLE(T) - 1
        TANT_QUE i>=p FAIRE :
            DEBUT
                T[i] <- T[i-1]
                i <- i-1
            FIN
        T[p-1] <- x
        RENVOYER T
    FIN
```

```
ALGORITHME
VARIABLES
    tab TYPE TABLEAU DE NOMBRE
    x, n, m, p TYPE NOMBRE
    rep TYPE CHAINE
DEBUT
    ECRIRE "Combien de valeurs à entrer ?"
    LIRE m
    n <- 2*m
    tab <- CREER_TABLEAU(n)
    POUR i ALLANT_DE 0 A m-1
        DEBUT
            ECRIRE "Elément en position ", i+1, "?"
            LIRE tab[i]
        FIN
    rep <- "oui"
    TANT_QUE rep=="oui" ET m<n FAIRE
        DEBUT
            ECRIRE "Quel est l'élément à insérer ?"
            LIRE x
            ECRIRE "A quelle position ?"
            LIRE p
            TANT_QUE p>m+1 ou p<=0 FAIRE
                DEBUT
                ECRIRE "A quelle position ?"
                LIRE p
                FIN
            tab <- insertion(tab, x, p)
            m <- m+1
            ECRIRE "La nouvelle sequence de longueur ",
                   m, " est :"
            POUR i ALLANT_DE 0 à m-1
                DEBUT
                    ECRIRE tab[i]
                FIN
            SI m<n ALORS
                DEBUT
                    ECRIRE "Voulez-vous ajouter à nouveau ?",
                           "Répondre par oui ou non"
                    LIRE rep
                FIN
            SINON
                DEBUT
                    ECRIRE "le tableau est plein"
                FIN
        FIN
FIN
```

Exercice 5 [6 points]

Il pîaart que puor la lctreue, l'orrde des lrttees à l'iétunreir des mots n'a acnuue itnpocmare. La sulee chose qui cptmoe est que la pemièrre et la dneèirre lttree seonit à leur pclae.

L'objectif de cet exercice est de concevoir un programme en **Python** permettant de tester cette théorie. Étant donnée une chaîne de caractères saisie par l'utilisateur, il mélangera aléatoirement les lettres à l'intérieur des mots et affichera la phrase modifiée. On supposera que la chaîne de caractères ne comporte pas de signe de ponctuation et que deux mots sont séparés par exactement une espace.

Attention : L'utilisation du type `list` est interdite dans cet exercice.

1. Écrire une fonction `permuteMot()` qui prend en argument un mot et permute aléatoirement les lettres à l'intérieur du mot en gardant les première et dernière lettres inchangées.
2. Écrire une fonction `permute()` qui prend en argument une phrase et applique la première fonction sur chacun des mots.
3. Écrire le programme demandé en utilisant les fonctions précédentes.

Exemple : L'utilisation saisit la phrase `"Je vais avoir une bonne note"`, alors le programme peut afficher `"Je vias avior une bnone ntoe"`.

Solution :

```
from random import randint

def permuteMot(ch):
    if len(ch) < 3:
        return ch
    ch1 = ch[1:len(ch)-1]
    ch2 = ''
    while len(ch1)>0:
        i = randint(0,len(ch1)-1)
        ch2 = ch2 + ch1[i]
        ch1 = ch1[:i] + ch1[i+1:]
    return ch[0] + ch2 + ch[len(ch)-1]

def permute(phrase):
    cpt = 0
    resul = ''
    while cpt<len(phrase):
        mot = ''
        while cpt<len(phrase) and phrase[cpt]!=' ':
            mot = mot + phrase[cpt]
            cpt += 1
        resul = resul + permuteMot(mot) +  ' '
        cpt += 1
    return resul

phrase = input('Saisir une phrase ')
print(permute(phrase))
```