Informatique
Préparation à l'oral.

Divers - Sujets

2017

1. Centrale PC ♠

(Planche Officiel de la Taupe Gyroscope 2017 N°158)

On note d_n le nombre de permutations sans points fixes de $[\![1;n]\!]$. Par convention $d_0 = 1$.

1. Calculer d_1, d_2, d_3 et montrer que $\sum_{k=0}^{n} \binom{n}{k} d_{n-k} = n!$.
2. Montrer que $f(x) = \sum_{n \geq 0} \frac{d_n}{n!} x^n$ est définie sur $]-1;1[$ et que $e^x f(x) = \frac{1}{1-x}$.
3. Montrer que $d_n = n! \sum_{k=0}^{n} \frac{(-1)^k}{k!}$.
4. On note D_n la matrice carrée de taille $n+1$ de coefficients $d_{ij} = d_{i+j-2}$ pour $1 \leq i \leq j \leq n+1$.

   Montrer à l'aide de Python que son déterminant vaut : $H_n = \det(M_n) = \left(\prod_{k=0}^{n} k!\right)^2$.

2. D'après ENSAM PSI ♠

(Planche Revue de Mathématiques Spéciales 2017 N°1129)

1. Soit $n \in \mathbb{N}$. Que renvoie l'instruction `list(str(n))` ?

2. Ecrire une fonction `facto` qui reçoit en argument un entier naturel et renvoie la factorielle de cet entier.

3. Soit Σ l'ensemble des entiers naturels égaux à la somme des factorielles de leurs chiffres (par exemple, $145 \in \Sigma$ car $145 = 1! + 4! + 5!$). On note $\sigma(n)$ la somme des factorielles des chiffres de *n*. On a donc : $\Sigma = \{n \in \mathbb{N} / \sigma(n) = n\}$.

    a. Soit *n* un entier naturel à *p* chiffres. Montrer que l'on a $\sigma(n) \leq p \times 9!$ et $n \geq 10^{p-1}$.

    b. Soit la suite *u* définie par : $\forall p \in \mathbb{N}^*, u(p) = \dfrac{10^{p-1}}{p \times 9!}$.

    Montrer que *u* est strictement croissante et déterminer, à l'aide d'une fonction Python, le plus petit entier naturel *p* tel que $u(p) > 1$

    c. Déterminer Σ.

3. E3A PSI ♠

1. Que fait le code suivant :

```
nfin = 65 + 26

for i in range(65,nfin):
    print(i,"<->",chr(i))

from random import randint

chaine = ""
for i in range(30):
    chaine += chr(randint(65,nfin-1))
print(chaine)
```

2. Ecrire une fonction Python `tirer` qui reçoit comme arguments deux entiers `n` et `k` et renvoie une chaîne de `n` caractères pris au hasard parmi les `k` premières lettres majuscules de l'alphabet.

3. Ecrire une fonction `mot` qui reçoit comme arguments deux chaînes de caractères `m` et `c` et renvoie le nombre d'occurrences du mot `m` dans `c`.

4. Ecrire une fonction `compositions` qui reçoit comme arguments une chaîne de caractères `c` et un entier non nul `k` et renvoie une liste de doublets (m_i, n_i) où m_i est un mot de longueur `k` dans c et n_i le nombre d'occurrence de m_i dans `c`.

2016

1. ENSAM (PSI) ♠♠

Un nombre palindrome est un entier qui peut être lu indifféremment de gauche à droite ou de droite à gauche. Par exemple, 12321 en est un. On dit qu'un entier *n* est un palindrome à retard de degré 1 si *n* n'est pas un palindrome et si, lorsqu'on somme *n* avec le nombre n' obtenu en renversant l'ordre des chiffres de *n*, on obtient un nombre $p = n + n'$ qui est un nombre palindrome.
Par récurrence, l'entier *n* est à retard de degré *k* s'il faut effectuer *k* fois la sommation précédente pour obtenir un palindrome (à chaque étape, on somme donc le nombre obtenu avec son « permuté »).
Exemple : le nombre 4786 n'est pas un palindrome (degré 0), $4786 + 6874 = 11660$ n'en est pas un non plus (degré 1), $11660 + 6611 = 18271$ n'en est toujours pas un (degré 2), etc. On s'arrête lorsqu'on a trouvé un palindrome (degré *k*).

Ecrire une fonction Python `degree(n)` qui prend en arguments un entier *n* et renvoie son degré de retard.

2. ENSAM (PSI) ♠

(BEOS - 2016 – PSI – Epreuve orale N°2833)

On s'intéresse ici aux entiers naturels égaux à la somme des factorielles de leurs chiffres.
Par exemple : $1! = 1$, $2! = 2$ et $145! = 1! + 4! + 5!$.

On admet qu'il en existe un unique autre que les trois ci-dessus. Quel est-il ?

3. ENSAM (PSI) ♠

Ecrire une fonction `Binaire(n)` qui renvoie la liste de la représentation de *n* en base 2 avec le bit de poids fort le plus à gauche. Par exemple `Binaire(23)` renvoie 10111 sous forme de liste : `[1,0,1,1,1]`.

Ecrire une fonction `NombreDeUn(n)` qui renvoie le nombre de 1 dans la représentation de *n* en base 2.

Une liste est un palindrome si la lecture de cette liste de la gauche vers la droite est identique à la lecture de la droite vers la gauche et *n* est un 2-palindrome si `Binaire(n)` est un palindrome.

Ecrire une fonction `Palindrome(n)` qui indique si la représentation de *n* en base 2 est un palindrome.

Afficher tous les 2-palindromes inférieurs à 100.

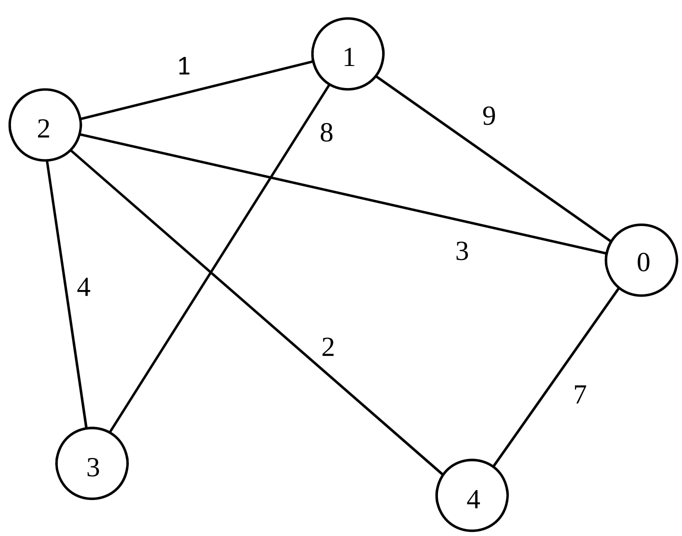
2015

1. D'après ENSAM (PSI) ♠

1) Pour un nombre entier naturel *n*, que renvoie l'instruction `list(str(n))` ?
2) Ecrire une fonction `somme` qui à un entier naturel *n* associe la somme de ses chiffres.
3) Un entier naturel *n* est dit « adéquat » si la somme de ses chiffres est un multiple de 10. Ecrire une fonction `test` qui renvoie le booléen `True` si le nombre *n*, passé en argument, est adéquat et `False` sinon.
4) Ecrire une fonction `modification` qui reçoit un entier naturel *n* en argument et change le chiffre des unités de *n* pour obtenir un nouvel entier n' qui est lui adéquat. Si *n* est adéquat, la fonction le renvoie inchangé.
5) Tester la fonction `modification` avec 10 entiers entre 10 000 et 100 000 grâce à la fonction `randint` du module `random`.
   La fonction affichera pour chacun des dix entiers :
   - L'entier lui-même.
   - La somme de ses chiffres.
   - L'entier modifié (éventuellement égal à l'entier lui-même).
   - La somme des chiffres de l'entier modifié.

2. E3A – Exercice type N°2 (PSI) ♠

On considère le graphe **G** suivant, où le nombre situé sur l'arête joignant deux sommets est leur distance, supposée entière :



1. Construire la matrice $\left(M_{ij}\right)0 \le i,\, j \le 4$, matrice de distances du graphe **G**, définie par :

   Pour tous les indices *i*, *j*, M_{ij} représente la distance entre les sommets *i* et *j*,
   ou encore la longueur de l'arête reliant les sommets *i* et *j*.

   On convient que, lorsque les sommets ne sont pas reliés, cette distance vaut -1. La distance du sommet *i* à lui-même est, bien sûr, égale à 0.

Dans les questions suivantes, la matrice *M* sera déclarée globale.

2. Ecrire une fonction **voisins**, d'argument un sommet *i*, renvoyant la liste des voisins du sommet *i* (la matrice *M* sera déclarée globale).
3. Ecrire une fonction **degre**, d'argument un sommet *i*, renvoyant le nombre de voisins du sommet *i*, c'est-à-dire le nombre d'arêtes issues de *i* (la matrice *M* sera déclarée globale).
4. Ecrire une fonction **longueur**, d'argument une liste L de sommets de G, renvoyant la longueur du trajet décrit par cette liste L, c'est-à-dire la somme des longueurs des arêtes empruntées. Si le trajet n'est pas possible, la fonction renverra -1.

3. D'après ENSAM (PT) ♠

1. Ecrire une fonction « Binaire » qui prend en entrée un entier naturel *n* et renvoie, sous forme de liste, l'écriture de cet entier en base 2 (par exemple, on aura « Binaire(5) = [1,0,1] »).
2. Ecrire une fonction « NombreDeUn » qui prend en entrée un entier naturel *n* et renvoie le nombre de 1 contenus dans son écriture binaire (par exemple, on aura « NombreDeUn(5) = 2 »).
3. Un nombre est un 2-palindrome lorsque son écriture en base 2 se lit de la même manière de gauche à droite que de droite à gauche (par exemple 5 est un 2-palindrome).
   Ecrire une fonction Palindrome qui prend en entrée un entier naturel *n* et détermine si cet entier est un 2-palindrome ou non.
4. Déterminer tous les 2-palindromes inférieurs à un entier N passé en argument (à chaque fois, on affichera le nombre et son écriture en base 2 sous forme d'une liste).

4. E3A – Exercice type N°7 (PSI) ♠

Soit *n* un entier naturel $n \le 26$. On souhaite écrire un programme qui code un mot en décalant chaque lettre de l'alphabet de *n* lettres.
Par exemple pour $n = 3$, le décalage sera le suivant :

Avant décalage	*a*	*b*	*c*	*d*	…	…	*x*	*y*	*z*
Après décalage	*d*	*e*	*f*	*g*	…	…	*a*	*b*	*c*

Le mot `oralensam` devient `rudohqvdp`.

1. Définir une chaîne de caractères contenant toutes les lettres dans l'ordre alphabétique (caractères en minuscule).
2. Ecrire une fonction **decalage**, d'argument un entier `n`, renvoyant une chaîne de caractères contenant toutes les lettres dans l'ordre alphabétique, décalées de `n`, comme indiqué ci-dessus.
3. Ecrire une fonction **indices**, d'arguments un caractère `x` et une chaîne de caractères `phrase`, renvoyant une liste contenant les indices de `x` dans `phrase` si `x` est une lettre de `phrase` et une liste vide sinon.
4. Ecrire une fonction **codage** d'arguments un entier `n` et une chaîne de caractères `phrase`, renvoyant `phrase` codée avec un décalage de `n` lettres.
5. Comment peut-on décoder un mot codé ?

5. D'après ENSAM (PSI) ♠♠

Le code creux consiste à coder un vecteur sans les zéros qu'il comporte et qui constituent une information inutile pour les calculs. Ce code comporte une première liste des coordonnées non nulles et une seconde qui indique leurs positions. Par exemple, le vecteur $(1, 2, 0, 5, 8, 0, 0, 0, 7)$ donne, en code creux $\left[9, [1, 2, 5, 8, 7], [1, 2, 4, 5, 9]\right]$ (le premier élément de la liste correspond à la taille du vecteur).

En Python :

1. Ecrire une fonction « `Encode` » permettant de coder un vecteur en code creux. Le vecteur initial sera fourni en argument de la fonction sous la forme d'une liste de coordonnées.
2. Ecrire une fonction « `Decode` » permettant de décoder un vecteur codé en code creux.

Pour chacune des opérations suivantes, les vecteurs initiaux doivent être conservés et le résultat fourni sous la forme d'un vecteur codée en code creux (lorsqu'il s'agit, bien sûr, d'un vecteur…).

3. Ecrire une fonction « `ScalMult` » permettant de multiplier un vecteur codé en code creux par un scalaire.
4. Ecrire une fonction « `ScalProd` » permettant d'effectuer le produit scalaire de deux vecteurs codés en code creux.
5. Ecrire une fonction « `Add` » permettant d'effectuer la somme de deux vecteurs codés en code creux.