

INF101/INF131 : examen 2e session

Juin 2019

- **Durée 2 heures.**
- **Lisez toutes les questions avant de commencer !** Le sujet fait 3 pages.
- Respectez **strictement** les consignes de l'énoncé (noms des fonctions et variables, format d'affichage...)
- Les exercices sont indépendants et ne sont **pas** dans l'ordre de difficulté. Vous pouvez sauter des exercices et des questions. Vous pouvez supposer existantes les fonctions des questions précédentes et les utiliser, même si vous n'avez pas répondu à la question correspondante.
- Il est inutile de filtrer les entrées lorsque ce n'est pas explicitement demandé (elles seront supposées correctes).
- **Aucun document autorisé. L'utilisation d'une calculatrice, d'un téléphone, etc, est interdite.**
- **Chaque question vaut 1 point.** Le barème est indicatif. La qualité de la rédaction et de la présentation sera prise en compte. Pensez à commenter vos programmes pour les rendre plus lisibles. Attention à l'indentation. Répondez à **chaque exercice sur une feuille séparée.**

1 Booléens (3 points)

Écrire les expressions booléennes correspondant aux phrases suivantes :

1. L'entier q est le quotient de la division de l'entier a par l'entier b , et l'entier r est le reste de cette division.
2. Le nombre a est soit strictement positif et multiple du nombre p , soit négatif et multiple du nombre n .
3. Les chaînes de caractères c , d , e sont dans l'ordre alphabétique strict (la chaîne c est avant la chaîne d dans le dictionnaire, etc)
4. Les chaînes de caractères (supposées non vides) c et d commencent par la même lettre
5. La chaîne de caractères c ne contient aucune voyelle minuscule
6. L'entier naturel n a au maximum k chiffres

2 Codage de mots (4 points)

Dans cet exercice, on appelle **mot** une chaîne de caractères constituée uniquement de lettres minuscules de l'alphabet.

1. Écrire une fonction `codage1` qui reçoit en paramètre un mot, et renvoie la somme des positions de ses lettres dans l'alphabet. Par exemple pour le mot 'salut', on renverra 73 ($19+1+12+21+20$).
2. Écrire une fonction `codage2` qui reçoit en paramètre un mot et renvoie le mot obtenu en remplaçant chaque lettre par la suivante (le 'z' devient un 'a'). Par exemple 'salut' devient 'tbmvu'.
3. Écrire une fonction `codage3` qui reçoit en paramètre un mot, et renvoie une chaîne de caractères ne comportant plus que les consonnes de ce mot. Par exemple 'salut' devient 'slt', 'bonjour' devient 'bnjr'.
4. Écrire une fonction `codage4` qui reçoit en paramètre un mot, et renvoie un mot composé des mêmes lettres mais mélangées dans un ordre aléatoire. *Indice: on pourra utiliser la fonction `shuffle` du module `random`, sans oublier de l'importer.*

3 Triangle de Pascal et coefficient binomial (7 points)

1. Écrire une fonction `calculeTriangle(n)` qui calcule et renvoie le triangle de Pascal de taille n (nombre de lignes), pour n reçu en paramètre. Cette fonction renvoie une liste de n listes, chaque liste représentant une des lignes du triangle (les listes ne sont donc pas toutes de même taille).
Par exemple pour $n=4$, cette fonction renvoie la liste: $[[1],[1,1],[1,2,1],[1,3,3,1]]$.
2. Écrire une fonction `afficheTriangle(t)` qui affiche la liste t reçue en paramètre, qui contient un triangle de Pascal, sous la forme d'un triangle. *Par exemple la liste précédente est affichée sous la forme suivante:*

```

1
1 1
1 2 1
1 3 3 1

```

Le nombre situé à l'intersection de la ligne n et de la colonne k (lignes et colonnes sont numérotées à partir de 0) est appelé coefficient binomial C_n^k . Il représente le coefficient de rang k dans le développement de $(a+b)^n$. Par exemple: $(a+b)^3 = 1 * a^3 + 3 * a^2 * b + 3 * a * b^2 + 1 * b^3 = C_3^0 * a^3 + C_3^1 * a^2 b + C_3^2 * a b^2 + C_3^3 * b^3$

3. Écrire une fonction `coeffBinomial(n,k)` qui calcule et renvoie le coefficient binomial C_n^k en fonction des entiers n et k reçus en paramètre. *Par exemple pour $n = 3$ et $k = 1$ cette fonction renvoie C_3^1 qui vaut 3.* Attention les numérotations partent de 0.
4. Écrire une fonction `devPolynome(a,b,n)` qui développe $(a+b)^n$ en se servant du triangle de Pascal (attention à ne calculer le triangle qu'une seule fois). Cette fonction affiche le développement, et ne renvoie rien. *Par exemple: `devPolynome(5,7,3)` affiche: $(5+7)^3 = 1 * 5^3 * 7^0 + 3 * 5^2 * 7^1 + 3 * 5^1 * 7^2 + 1 * 5^0 * 7^3$*
On notera que cette fonction peut recevoir a et b soit comme des entiers soit comme des chaînes de caractères (par contre n doit être un entier). *Par exemple: `devPolynome("a","b",4)` affiche: $(a+b)^4 = 1 * a^4 * b^0 + 4 * a^3 * b^1 + 6 * a^2 * b^2 + 4 * a^1 * b^3 + 1 * a^0 * b^4$*
5. Écrire une fonction `calculPolynome(a,b,n)` qui reçoit des entiers a,b,n et calcule la valeur de $(a+b)^n$ en se servant du développement grâce au triangle de Pascal (qui sera calculé une seule fois grâce à la fonction ci-dessus). Cette fonction renvoie la valeur calculée.
6. Écrire une fonction `verifPascal(n)` qui calcule le triangle de Pascal à n lignes, puis tire au hasard 2 entiers entre 1 et 30, calcule la valeur de $(a+b)^n$ à partir de ce triangle de Pascal, et la compare avec la valeur calculée normalement. Cette fonction renvoie un booléen indiquant si les valeurs trouvées concordent.
7. Écrire un programme principal qui demande à l'utilisateur un entier n , calcule le triangle de Pascal correspondant, l'affiche, le vérifie et affiche le résultat ("valide" ou "non valide").

4 Dictionnaire de traduction (6 points)

1. Écrire une fonction `initListeMots` qui ne reçoit aucun paramètre, demande à l'utilisateur de saisir des mots français en terminant par 'stop', enregistre tous ces mots dans une liste (sauf 'stop'), et renvoie cette liste.
2. Écrire une fonction `initDicoAnglais` qui reçoit une liste de mots français, demande à l'utilisateur la traduction de chaque mot en anglais, et initialise un dictionnaire français-anglais (clé = mot français, valeur = mot anglais). Elle renvoie ce dictionnaire.
3. Écrire une fonction `compterHomonymes` qui reçoit un mot anglais et un dictionnaire français-anglais, et compte de combien de mots français il est la traduction dans ce dictionnaire. *Par exemple le mot anglais "row" est la traduction de "ligne", mais aussi de "dispute" ou encore de "ramer".* Il faut donc parcourir le dictionnaire entier pour chercher tous les homonymes. La fonction renvoie 2 valeurs : le compte, et la liste des homonymes. *Par exemple `compterHomonymes("row")` renvoie 3, ["ligne", "dispute", "ramer"], en supposant que ces mots (et aucun autre homonyme) sont présents dans le dictionnaire utilisé.*
4. Écrire une fonction `supprimerDoublons` qui reçoit en paramètres un dictionnaire français-anglais et une liste de mots français; elle parcourt les mots français pour vérifier qu'il n'y a pas plusieurs mots ayant la même traduction dans le dictionnaire reçu. Si c'est le cas, alors les doublons sont supprimés du dictionnaire (on ne garde que le premier mot ayant cette traduction). Cette fonction ne renvoie rien, elle modifie le dictionnaire reçu en paramètre (effet de bord). *Indice: on a besoin de parcourir la liste des mots plutôt que les clés du dictionnaire, car on n'a pas le droit de modifier le dictionnaire pendant qu'on le parcourt.*
5. Écrire une fonction `inverseDico` qui reçoit en paramètre un dictionnaire français-anglais, et l'inverse pour créer un dictionnaire anglais-français (*on suppose qu'il n'y a pas plusieurs mots ayant la même traduction*).
6. Écrire un programme principal qui utilise les fonctions précédentes pour :
 - Créer un dictionnaire nommé `fr2en` associant des mots français (saisis par l'utilisateur) à leur traduction anglaise (saisie par l'utilisateur).
 - Créer un dictionnaire nommé `en2fr` en inversant le dictionnaire ci-dessus. On prendra soin de supprimer les doublons avant l'inversion.
 - Demander à l'utilisateur de saisir un mot et une langue ('en' ou 'fr'), puis utiliser le bon dictionnaire pour traduire le mot dans l'autre langue, et afficher le résultat.
 - Lui proposer de rejouer avec un autre mot et langue, jusqu'à ce qu'il refuse de continuer.
 - Afficher 'fin du programme' avant de se terminer.

Mémo Python - UE INF101 / INF131 / INF204 - version 2018

Opérations sur les types

`type()` : pour connaître le type d'une variable
`int()` : transformation en entier
`float()` : transformation en flottant
`str()` : transformation en chaîne de caractères

Définition d'une fonction

```
def nomFonction(arg) :  
    instructions  
    return v
```

Fonction qui renvoie la valeur ou variable `v`.

Infini

`float('inf')` : valeur infinie positive ($+\infty$)
`float('-inf')` : valeur infinie négative ($-\infty$)

Écriture dans la console

```
print(a1,a2,...,an, sep=xx, end=yy)
```

- Pour imprimer une suite d'arguments de `a1` à `an`
- `sep` : permet de définir le séparateur affiché entre chaque argument (optionnel, par défaut " ")
- `end` : permet de définir ce qui sera affiché à la fin (optionnel, par défaut : saut de ligne)

Lecture dans la console

```
res = input(message)
```

- Pour lire une suite de caractères au clavier terminée par `< Enter >`
- Ne pas oublier de transformer la chaîne en entier (`int`) ou réel (`float`) si nécessaire.
- La chaîne de caractères résultante doit être affectée (ici à la variable `res`).
- L'argument est optionnel : c'est un message explicatif destiné à l'utilisateur

Opérateurs booléens

`and` : et logique
`or` : ou logique
`not` : négation

Opérateurs de comparaison

<code>==</code> égalité	<code>!=</code> différence
<code><</code> inférieur,	<code><=</code> inférieur ou égal
<code>></code> supérieur,	<code>>=</code> supérieur ou égal

Instructions conditionnelles

```
if condition :  
    instructions
```

```
if condition :  
    instructions  
else :  
    instructions
```

```
if condition1 :  
    instructions  
elif condition2 :  
    instructions  
else :  
    instructions
```

Opérateurs arithmétiques

<code>+</code> : addition,	<code>-</code> : soustraction
<code>*</code> : multiplication,	<code>**</code> : puissance,
<code>/</code> : division,	<code>//</code> : quotient div entière,
<code>%</code> : reste de la division entière (modulo)	

Caractères

`ord(c)` : renvoie le code ASCII du caractère `c`
`chr(a)` : renvoie le caractère de code ASCII `a`

Chaînes de caractères

`len(s)` : renvoie la longueur de la chaîne `s`
`s1+s2` : concatène les chaînes `s1` et `s2`
`s*n` : construit la répétition de `n` fois la chaîne `s`
exemple : `"ta"*3` donne `"tatata"`
`list(chaine)` : renvoie la liste des caractères de la chaîne
`ch.split(arg)` : retourne la liste des sous-chaînes de `ch`, en coupant à chaque occurrence de `arg` (par défaut `arg=" "`)
`ch.join(liste)` : concatène les chaînes de `liste`, en utilisant `ch` comme séparateur, et renvoie la chaîne résultante
`ch.upper()` : passe `ch` en majuscules
`ch.lower()` : passe `ch` en minuscules

Itération tant que

```
while condition :  
    instructions
```

Itération for, et range

```
for e in conteneur :  
    instructions
```

```
for var in range (deb, fin, pas) :  
    instructions
```

Itère les instructions avec **e** prenant chaque valeur dans le conteneur (liste, chaîne ou dictionnaire) ; ou avec **var** prenant les valeurs entre **deb** et **fin** avec un **pas** donné.

range(a) : séquence des valeurs [0, a[
range (b,c) : séquence des valeurs [b, c[(pas=1, $c > b$)
range (b, c, g) : idem avec un pas = g
range(b,c,-1): valeurs décroissantes de b (incl.) à c (excl.), pas=-1 ($c < b$)

Listes

maListe = []: création d'une liste vide
maListe = [e1,e2,e3] : création d'une liste, ici à 3 éléments e1, e2, et e3

maListe[i]: obtenir l'élément à l'index *i* ($i \geq 0$).
Les éléments sont indexés à partir de 0. Si $i < 0$, les éléments sont accédés à partir de la fin de la liste. Ex : **maListe[-1]** permet d'accéder au dernier élément de la liste

maListe.append(elem): ajoute un élément à la fin
maListe.extend(liste2): ajout de tous les éléments de la liste **liste2** à la fin de la liste **maListe**
maListe.insert(i,elem): ajout d'un élément à l'index *i*

res = maListe.pop(index): retire l'élément présent à la position **index** et le renvoie, ici dans la variable **res**
maListe.remove(element): retire l'élément donné (le premier trouvé)

len(maListe): nombre d'éléments d'une liste
elem in maListe: teste si un élément est dans une liste (renvoie True ou False)
maListe.index(elem): renvoie l'index (la position) d'un élément dans une liste (ValueError si absent)

l2 = maListe: crée un synonyme (2ème nom pour la liste)
l3 = list(maListe): crée une copie de surface (un clone)
l4 = copy.deepcopy(maListe): crée une copie profonde (récursive)

Aléatoire

random.randint(inf,sup): entier aléatoire entre bornes inf et sup incluses
random.shuffle(maListe): mélange la liste (effet de bord), ne renvoie rien
random.choice(maListe): renvoie un élément au hasard de la liste

Dictionnaires

monDico = {} : création d'un dictionnaire vide
monDico = { c1:v1, c2:v2, c3:v3 } : création d'un dictionnaire, ici à 3 entrées (clé c1 avec valeur v1, etc)

e = monDico[c1] : les valeurs du dictionnaire sont accessibles par leurs clés. Ici, **e** prendra la valeur v1. Provoque une erreur si la clé n'existe pas.

monDico[c3] = v3: ajoute une nouvelle valeur au dictionnaire (ici v3) avec une clé (ici c3). Si la clé existe déjà, la valeur associée est modifiée.

del monDico[C3]: supprime une association dans le dictionnaire. La clé doit exister.

c in monDico: vérifie l'existence d'une clé dans le dictionnaire, renvoie True ou False.

dic2 = monDico: crée un synonyme (2ème nom au dico)
dic3 = dict(monDico): crée une copie de surface (clone)
dic4 = copy.deepcopy(monDico): crée une copie profonde (récursive)

Gestion des fichiers

f=open('data.txt'): ouvrir un fichier en lecture seule
f=open('data.txt','w'): ouvre un fichier en écriture (attention s'il existe il est écrasé, sinon il est créé)
f=open('data.txt','a'): ouvre un fichier en écriture (ajoute le texte à la fin)

texte = f.read(): lire tout le fichier en une seule fois
lignes = f.readlines(): lire en 1 fois toutes les lignes du fichier et les stocker dans une liste (un élém=une ligne)

for ligne in f:
 instructions
Lire le fichier ligne par ligne dans une boucle for

f.write(texte): écrire dans un fichier (**texte** doit obligatoirement être une **string**).
Ne saute pas de ligne automatiquement à la fin du texte.
'\n' code un saut de ligne.

f.close(): ferme un fichier