INF101 : examen final - 5 Janvier 2023

- **Durée 2 heures.** Le sujet fait 3 pages. Aucun document, aucun appareil électronique.
- Répondez **sur la fiche réponse** fournie.
- Toutes les réponses (booléens, fonctions...) sont attendues en **Python**. Respectez **strictement** les consignes de l'énoncé (noms de fonctions/variables, format d'affichage...). Indentez correctement votre code et incluez des commentaires pertinents.
- On interdit d'utiliser `break` et `continue`, ainsi que toute fonction, opérateur ou structure non listée dans le mémo.
- Vos programmes doivent être **LISIBLES**. Le barème est **indicatif**, la qualité de la présentation sera prise en compte.

1 EXERCICE 1 : BOOLEENS (2 points)

On suppose que *sa* et *sb* sont 2 chaînes de caractères, et x un entier.

1. Écrire des expressions booléennes représentant les affirmations suivantes :
   - *sa* contient une lettre minuscule et *sb* la même lettre en majuscule. Les **seules** fonctions autorisées sont `chr` et `ord`.
   - x est strictement inférieur à 100, et est un nombre premier. **Aucune** fonction autorisée.
2. Écrire la négation des booléens suivants en utilisant les lois de De Morgan (pas d'opérateur *not*, ni de comparaison avec `True` ou `False`) :
   - $2 <= len(sa) < 7$ or *sb* in ["*oui*","*Oui*"]
   - $x\%3 == 0$ and $1 < x < 30$

2 EXERCICE 2 : MOYENNE (3 points)

Dans cet exercice on veut lire les notes d'étudiants et calculer leur moyenne. **Aucune liste** ne doit être utilisée dans cet exercice. Exemples d'interaction pour un étudiant :

```
# exemple d'interaction
Veux-tu saisir une note ? oui
Note ? 27
Incorrect, note ? 2.7
Veux-tu saisir une note ? oui
Note ? 10.5
Veux-tu saisir une note ? oui
Note ? 12
Veux-tu saisir une note ? non
Moyenne de 3 notes = 8.4

# autre exemple
Veux-tu saisir une note ? non
Moyenne de 0 notes : -1
```

3. Écrire une fonction `lireNote` qui ne reçoit **aucun** argument, qui lit une note entrée par l'utilisateur (*par exemple 12.5*). La fonction doit **filtrer** le nombre entré par l'utilisateur **jusqu'à** ce qu'il s'agisse bien d'une note correcte (entre 0 et 20 inclus). A ce moment la fonction **renvoie** la note lue.
4. Écrire une fonction `moyenne` qui ne reçoit **aucun** argument. Elle propose **en boucle** à l'utilisateur d'entrer une note : tant qu'il accepte en répondant exactement "oui", elle lit une note en **appelant** la fonction précédente, puis lui propose d'en entrer une autre, etc. Quand il refuse (toute autre réponse que "oui") d'entrer plus de notes, la fonction **renvoie** le nombre de notes saisies (0 si aucune) et leur moyenne (ou -1 si aucune). On **interdit** de stocker toutes les notes dans une liste.
5. Écrire un programme principal qui demande le nombre d'étudiants, puis **appelle** la fonction précédente pour calculer la moyenne de chacun, et **affiche** chaque résultat avec le message suivant : "Moyenne de XX notes = YY" (en remplaçant XX et YY par les bonnes valeurs).

Tournez la page

3 EXERCICE 3 : ESTIMATION DU NOMBRE d'EULER (3 points)

On considère des banques qui fonctionnent de la manière suivante : la banque 1 offre un taux d'intérêts de 100% ($\frac{1}{1}$) tous les 1 ans ; la banque 2 offre un taux d'intérêts de 50% ($\frac{1}{2}$) tous les 6 mois ($\frac{1}{2}$ ans) ; ... ; la banque n offre un taux de $\frac{1}{n}$ tous les $\frac{1}{n}$ ans. Ainsi, si on place 1 euro dans la banque 1, au bout d'un an on aura 2 euros (1 + 100% d'intérêts après 1 an). Si on place 1 euro dans la banque 2, au bout d'un an on aura 2.25 euros (1 + 50% d'intérêts à 6 mois = 1.5, puis 50% d'intérêts sur 1.5 = 0.75 après les 6 mois suivants).

6. Écrire une fonction `banque(n)` qui renvoie ce qu'on obtient au bout d'un an en plaçant 1 euro dans la banque n ($\frac{1}{n}$ d'intérêts tous les $\frac{1}{n}$ ans). *Remarque : cette fonction calcule en fait des valeurs approchées du nombre d'Euler e, qui vaut environ 2,718281828459045. Le module math fournit une constante e qui donne sa valeur.*

7. Écrire une fonction `estimIncond(nmax)` qui reçoit un entier *nmax*, et **affiche** (au format de l'exemple) successivement toutes les approximations (calculées avec la fonction ci-dessus) pour toutes les valeurs de n entre 1 et *nmax* inclus. Elle ne renvoie rien.

```
# Exemple pour nmax = 10
Approx avec 1 = 2.0
Approx avec 2 = 2.25
Approx avec 3 = 2.3703703703703702
...
Approx avec 10 = 2.5937424601000023
```

8. Écrire une fonction `estimCond(prec)` qui reçoit un nombre **réel** *prec*, calcule et **affiche toutes** les estimations pour les valeurs successives de n, **jusqu'à** ce que l'estimation ait atteint la précision *prec* (c-à-d que la distance entre e, fourni par le module `math`, et son approximation ainsi calculée, doit être **inférieure ou égale** à *prec*). La fonction affiche alors un message final (cf exemple) indiquant la valeur finale de n.

```
# Exemple pour prec = 0.01
Approx avec 1 = 2.0
Approx avec 2 = 2.25
Approx avec 3 = 2.3703703703703702
...
Approx avec 133 = 2.7081326544338706
Approx avec 134 = 2.708207880225728
Approx avec 135 = 2.708281999071387
Precision 0.01 atteinte pour 135
```

4 EXERCICE 4 : DES MOTS, RIEN QUE DES MOTS (4 points)

On s'intéresse ici à des chaînes de caractères. On **interdit** de convertir la chaîne de caractères en liste de caractères. On interdit aussi d'utiliser les fonctions Python de manipulation de chaînes (`upper`, `lower`, `reverse`, etc). L'objectif est d'évaluer votre capacité à écrire l'algorithme de ces opérations, pas votre connaissance de Python.

9. On **impose** d'utiliser une boucle **for** pour itérer directement sur les **caractères** de s. (a) Écrire une fonction `reverse(s)` qui reçoit une chaîne s et renvoie une chaîne contenant les mêmes caractères mais à l'envers. ; (b) Écrire une fonction `saufVoyelles(s)` qui reçoit une chaîne s, et renvoie une chaîne contenant les mêmes caractères (y compris ponctuation) sauf les voyelles (minuscules comme majuscules).

10. Écrire une fonction `commence(mot,let)` qui reçoit une chaîne (éventuellement vide, attention !) et une lettre, et **renvoie** un booléen indiquant si le mot commence bien par cette lettre (avec la même casse). Par exemple "abricot" commence bien par "a" (mais pas par "A") ; la chaîne vide ne commence par aucune lettre.

11. Écrire une fonction `afficheSi(mots,let)` qui reçoit une liste de mots et une lettre ; elle affiche (un par ligne) **uniquement** les mots de la liste qui commencent par cette lettre (appeler la fonction `commence`). On **interdit** bien sûr de modifier la liste de mots.

12. Écrire une fonction `alphabetMelange` qui reçoit en paramètre **optionnel** un booléen *minusc* (par défaut vrai). Cette fonction **renvoie** une liste contenant toutes les lettres de l'alphabet, une et une seule fois chacune, en minuscules si *minusc* est vrai, en majuscules sinon. Vous **devez** utiliser une **boucle** et les fonctions `chr` et `ord` pour générer la liste alphabétique (**interdiction** de la remplir "à la main"). Vous pouvez utiliser la fonction `random.shuffle` pour la mélanger.

Tournez la page

5 EXERCICE 5 : TRIONS des LISTES (4+ points)

13. Écrire une fonction `echange(li,p1,p2)` qui reçoit une liste *li*, et la **modifie** en échangeant les éléments situés aux positions $p1$ et $p2$ (on suppose que $p1$ et $p2$ sont des positions correctes). La fonction ne renvoie **rien**. **Attention** on **interdit** de décaler inutilement des éléments, il faut modifier **uniquement** ce qui est nécessaire.

14. Écrire une fonction `indmaxApres(li,pos)` qui reçoit une liste *li* et un indice *pos* (supposé correct) ; la fonction **ne modifie pas** la liste, et considère seulement les éléments de *li* après l'indice *pos* (inclus). Elle **renvoie** l'**indice** du plus grand élément situé après *pos*. **Attention** il ne faut parcourir les éléments qu'une seule fois ! *Par exemple indmaxApres([9,2,7,3,9,0],1) renvoie 4 (l'indice du plus grand élément après la position 1 est 4, pour la valeur 9. On remarquera que 9 est aussi présent à l'indice 0 mais avant pos donc la fonction renvoie bien 4 et pas 0.).*

15. Écrire une fonction `triSelection(li)` qui reçoit une liste *li* et la **modifie** pour la trier en ordre décroissant selon l'algorithme du tri par sélection (**rappel** : le tri par sélection cherche le maximum après un indice donné, puis l'échange avec l'élément à cet indice ; l'indice se décale ensuite, ainsi la partie gauche de la liste est triée, alors que la partie droite ne l'est pas ; quand l'indice atteint la fin de la liste, celle-ci est entièrement triée). Vous **appellerez** vos fonctions `echange` et `indmaxApres`. La fonction ne doit manipuler **qu'une seule** liste et la modifier au fur et à mesure.

16. (Difficile) Écrire une fonction `triBulles(li)` qui reçoit une liste *li* et la **modifie** pour la trier en ordre décroissant suivant l'algorithme du tri à bulles (rappel: le tri à bulles échange successivement des paires d'éléments qui ne sont pas dans le bon ordre ; ainsi au premier passage le plus petit élément remonte comme une bulle en dernière position; au 2e passage le second plus petit élément remonte en avant-dernière position ; etc). Vous appellerez votre fonction `echange`.

6 EXERCICE 6 : STATISTIQUES SUR TEXTE (4 points)

17. Écrire une fonction `classer(txt)` qui reçoit un texte (on suppose que les mots sont tous séparés par des espaces, aucune autre ponctuation). La fonction "classe" les mots par longueur : elle **renvoie** un dictionnaire associant des clés entières (les longueurs possibles de mots) à la liste des mots du texte ayant cette longueur. *Par ex : sur la consigne de cette question (dont on a supprimé les accents et la ponctuation) la fonction renvoie le dictionnaire suivant. On remarque qu'il n'y a pas d'ordre entre les clés, et qu'un mot peut apparaître plusieurs fois. {8: ['fonction', 'fonction', 'longueur', 'entieres', 'longueur'], 7: ['classer', 'suppose', 'separes', 'espaces', 'renvoie'], 3: ['qui', 'que', 'les', 'par', 'des', 'les', 'par', 'des', 'les', 'des'], 6: ['recoit', 'aucune', 'classe'], 2: ['un', 'on', 'La', 'un', 'de', 'la', 'du'], 5: ['texte', 'autre', 'liste', 'texte', 'ayant', 'cette'], 4: ['mots', 'sont', 'tous', 'mots', 'elle', 'cles', 'mots', 'mots'], 11: ['ponctuation'], 12: ['dictionnaire'], 9: ['associant', 'longueurs', 'possibles'], 1: ['a']}* .

18. Écrire une fonction `compter(txt)` qui reçoit une chaîne de caractères *txt* et la parcourt **une seule fois** pour compter le nombre d'occurrences de chacun de ses caractères. La fonction crée et **renvoie** un dictionnaire associant chaque caractère **présent** dans le texte (et **uniquement** les caractères présents) avec son nombre d'occurrences. Par ex `compter("hello all")` renvoie `{"h":1,"e":1,"l":4,"o":1," ":1,"a":1}`

19. Écrire une fonction `caracPlus(txt)` qui reçoit un texte non vide, et cherche quel est le caractère le plus présent dans ce texte. Il faut d'abord **appeler** la fonction `compter` puis parcourir le dictionnaire ainsi calculé. La fonction **renvoie** le caractère le plus fréquent (le premier trouvé en cas d'égalité). *Par exemple pour le texte ci-dessus "hello all", la fonction renvoie le caractère "l" car il est présent 4 fois.*

20. Écrire une fonction `listeCaracsPlus(txt)` similaire, sauf qu'en cas d'égalité, on ne renvoie plus le premier caractère trouvé avec la fréquence maximale, mais la **liste** de **tous** les caractères qui ont cette même fréquence maximale. S'il n'y en a qu'un, on renvoie un singleton (liste à un seul élément). *Par exemple pour le texte "good luck to all", la fonction renvoie la liste ["o", " ", "l"], ces 3 caractères étant à égalité avec 3 occurrences.*

FIN

Mémo Python - UE INF101 / INF104 / INF131 - version 2023

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

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

`input(msg)` : affiche le message (optionnel), lit au clavier une chaîne de caractères, la renvoie

```
res = int(input(msg)) # lecture et conversion en entier
res = float(input(message)) # conversion en réel
```

Opérateurs booléens

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

Opérateurs de comparaison

```
== égalité              != différence
<  inférieur,           <= inférieur ou égal
>  supérieur,           >= supérieur ou égal
```

Opérateurs et fonctions arithmétiques

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

abs(x): valeur absolue
math.sqrt(x): racine carrée
```

Aléatoire: module random

`import random`: importer le module pour l'utiliser
`random.randint(inf,sup)`: entier aléatoire entre bornes inf et sup incluses
`random.shuffle(li)`: mélange la liste (effet de bord), ne renvoie rien
`random.choice(li)`: renvoie un élément au hasard de *li*

Instructions conditionnelles

```
if condition :
    instructions

if condition :
    instructions
else :
    instructions
```

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

Caractères et chaînes

`ord(c)` : renvoie le code ASCII du caractère c
`chr(a)` : renvoie le caractère de code ASCII a
`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"
`c in s` : vérifie que le caractère *c* apparaît dans la chaîne *s*
`s[i]` : gives character at index *i* in string *s*
(**error** if index out of range)

`ch.split(arg)` : retourne la liste des sous-chaînes de *ch* coupée à chaque occurrence de `arg` (par défaut `arg=" "`)
`ch.join(liste)` : renvoie la concaténation des chaînes de `liste`, en utilisant `ch` comme séparateur

Itération tant que

```
while condition :
    instructions
```

Définition d'une fonction

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

Fonction qui renvoie la valeur ou variable v.

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 ($i >= 0$).
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 élements 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)

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.

`len(monDico)`: longueur d'un dictionnaire.

`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)