INF101 / INF104 : devoir surveillé

Mardi 9 Novembre 2021

- **Durée 2 heures.** Le sujet fait 4 pages. Aucun document, ni calculette/téléphone/...
- **Lisez bien ces consignes avant de commencer !**
- Respectez **strictement** les consignes de l'énoncé (noms de fonctions/variables, format d'affichage...)
- On interdit d'utiliser `break` et `continue`, ainsi que toutes fonctions non vues en cours.
- Un programme qui fonctionne mais ne respecte pas toutes les consignes ne rapporte **pas** de points. Un programme qui 'fonctionne' mais de manière manifestement sous-optimale ne rapporte pas tous les points.
- Vos programmes doivent être **LISIBLES**: indentation correcte, commentaires, pas de raccourcis...
- Les exercices sont indépendants, les questions dans chaque exercice aussi. Vous pouvez toujours utiliser une fonction d'une question précédente même sans l'avoir écrite.
- Le barème est **indicatif**. La qualité de la rédaction et de la présentation sera prise en compte.
- Répondez aux exercices **DANS l'ORDRE**. Laissez de la place si vous voulez revenir à un exercice plus tard. Évitez absolument les réponses éparpillées et les exercices mélangés entre eux.

1 EXERCICE 1 : BOOLÉENS (4 points)

On suppose que toutes les variables nécessaires sont bien initialisées (a,b sont des entiers, x et y sont des chaînes de caractères).

1. Écrire les expressions booléennes vraies si et seulement si :
   (a) L'entier a est pair et compris entre 0 inclus et 20 exclu.
   (b) Les caractères x et y correspondent à la même lettre, l'une en minuscule, l'autre en majuscule (attention on ne sait pas dans quel ordre).
   (c) Parmi les entiers a et b, au moins un est multiple de 2, et aucun n'est multiple de 3. *Indice :* il y a une astuce pour éviter de devoir lister tous les cas !
   (d) x est un mot de 6 caractères commençant par une voyelle majuscule. On interdit ici d'utiliser l'opérateur `in`.
2. Écrire la négation de ces expressions booléennes, simplifiée au maximum (pas d'opérateur `not`).

2 EXERCICE 2 : DES MOTS (4 points)

Écrire un programme qui lit des mots tapés au clavier par l'utilisateur, jusqu'à lire le mot "stop" (qui arrête la lecture et n'est pas compté dans la suite). Ce programme doit alors afficher le mot le plus long qu'il a lu (en cas d'égalité, le **dernier** mot lu de cette longueur; dans l'exemple ci-dessous c'est "bureaux" qui l'emporte sur "voiture") et le mot le plus court qu'il a lu (si plusieurs mots à égalité de longueur, c'est le **premier** lu qui l'emporte; dans l'exemple "chat" l'emporte sur "velo"). Si aucun mot n'a été lu (l'utilisateur tape "stop" tout de suite) alors le programme n'affiche rien.

Attention: on interdit d'utiliser des listes dans cet exercice.

```
Exemple d'exécution :
>
Tape des mots (stop pour arrêter)
? bateau
? chat
? voiture
? avion
? velo
? bureaux
? stop
Mot le plus long: bureaux
Mot le plus court: chat
>
```

Tournez la page

3 EXERCICE 3 : SUITE DE HERON (5 points)

La suite de Heron permet d'approximer la racine carrée d'un nombre a. Le premier terme u_0 est strictement positif, initialisé avec une valeur x "suffisamment proche" de $\sqrt{a}$. En pratique on utilise souvent la partie entière de $\sqrt{a}$ comme u_0 : on imposera donc $a \geq 1$, et donc $u_0 > 0$, pour éviter une division par 0 lors du calcul de u_1. La suite de Héron est donc définie ainsi :

$$\begin{cases} u_0 = x \quad , \quad avec \quad x \geq 1 \\ u_{n+1} = \frac{u_n + \frac{a}{u_n}}{2} \end{cases}$$

1. Écrire une fonction `int_sqrt(a)` qui utilise une boucle pour trouver la partie entière de la racine carrée du réel a (c-à-d le dernier entier dont le carré est inférieur à a), et renvoie cette valeur. On **interdit** ici d'utiliser les fonctions `sqrt`, `int`, ou toute autre méthode de calcul direct.
   Par exemple ***`int_sqrt(5)`*** *renvoie 2,* ***`int_sqrt(9)`*** *renvoie 3.*

2. Écrire une fonction `heron_sqrt(a,n)` qui reçoit 1 réel a et 1 entier n, et qui calcule et renvoie le terme u_n de la suite. Attention, il faut commencer par initialiser $u_0 = x$ grâce à la fonction `int_sqrt`.
   Par exemple ***`heron_sqrt(5,3)`*** *commence par initialiser u_0 à 2 (partie entière de la racine carrée de 5), puis calcule et renvoie u_3.*

3. Écrire une fonction `heron_rank(a,p)` qui calcule et renvoie le rang n auquel la suite de Héron donne l'approximation u_n de $\sqrt{a}$ avec une précision p (c'est-à-dire avec un écart maximal de p, entre le carré de la valeur estimée, et la valeur de a). Attention, il ne faut **surtout pas** appeler `heron_sqrt` pour calculer chaque terme ! (sinon on redémarre le calcul depuis u_0 à chaque fois.)
   Par exemple ***`heron_rank(5,0.01)`*** *renvoie 2 car ${u_2}^2$ vaut environ 5.0002 (la différence est donc inférieure à 0.01), mais ${u_1}^2 = 5.0625$ n'est pas suffisamment proche de 5.*

4. Écrire un programme principal qui répète les étapes suivantes en boucle :
   (a) Demande à l'utilisateur un réel a supérieur ou égal à 1, et le filtre jusqu'à ce que ce soit bien le cas.
   (b) Demande à l'utilisateur un écart réel p, puis utilise `heron_rank` pour calculer et afficher le rang n nécessaire pour obtenir la précision p.
   (c) Vérifie le calcul en affichant le carré du terme u_n, calculé avec `heron_sqrt`.
   (d) Propose de rejouer. L'utilisateur doit répondre par "oui" ou "non". Le programme recommence si la réponse est exactement "oui", sinon s'arrête en affichant le message "c'est fini".

Exemple d'exécution (respecter **exactement** le format d'affichage):

```
Estimation de la racine carree
a>=1? : 7
Precision? : 0.01
Precision de 0.01 obtenue au rang 3
Carre de l'estimation de rang 3 = 7.000003901511219
Veux-tu rejouer (oui/non)? : oui
a>=1? : 9
Precision? : 0.001
Precision de 0.001 obtenue au rang 0
Carre de l'estimation de rang 0 = 9
Veux-tu rejouer (oui/non)? : non
C'est fini
>>>
```

4 EXERCICE 4 : PENDU (7 points)

On veut coder un jeu du pendu, qui consiste à faire deviner un mot secret, lettre par lettre. A chaque essai, le jeu indique si la lettre proposée apparaît bien dans le mot, ou pas (auquel cas il incrémente le compteur d'erreurs, qui est limité). Le jeu affiche le mot secret en remplaçant les lettres inconnues par un tiret. Il affiche aussi les essais passés, pour mémoire.
Voici 2 exemples d'exécution (une victoire et une défaite):

```
************
Jeu du pendu
************
Niveau de difficulté? : 8
Tu as droit à 7 erreurs

* Essai numero 1
______
Lettres erronnées:
lettre = ? E
E apparait 1 fois

* Essai numero 2
___E__
Lettres erronnées:
lettre = ? AB
AB n'est pas une lettre
lettre = ? A
A apparait 2 fois

* Essai numero 3
_A_EA_
Lettres erronnées:
lettre = ? B
B apparait 1 fois

* Essai numero 4
BA_EA_
Lettres erronnées:
lettre = ? R
R apparait 0 fois
Encore 6 erreurs autorisees

* Essai numero 5
BA_EA_
Lettres erronnées:  R
lettre = ? S
S apparait 0 fois
Encore 5 erreurs autorisees

* Essai numero 6
BA_EA_
Lettres erronnées:  RS
lettre = ? T
T apparait 1 fois

* Essai numero 7
BATEA_
Lettres erronnées:  RS
lettre = ? U
U apparait 1 fois

Bravo, tu as devine le mot BATEAU
en 7 essais dont 2 erreurs !
```

```
************
Jeu du pendu
************
Niveau de difficulté? : 9
Tu as droit à 7 erreurs

* Essai numero 1
_____
Lettres erronnées:
lettre = ? A
A apparait 0 fois
Encore 6 erreurs autorisees

* Essai numero 2
_____
Lettres erronnées:  A
lettre = ? Z
Z apparait 0 fois
Encore 5 erreurs autorisees

* Essai numero 3
_____
Lettres erronnées:  AZ
lettre = ? X
X apparait 0 fois
Encore 4 erreurs autorisees

* Essai numero 4
_____
Lettres erronnées:  AZX
lettre = ? W
W apparait 0 fois
Encore 3 erreurs autorisees

* Essai numero 5
_____
Lettres erronnées:  AZXW
lettre = ? Q
Q apparait 0 fois
Encore 2 erreurs autorisees

* Essai numero 6
_____
Lettres erronnées:  AZXWQ
lettre = ? R
R apparait 0 fois
Encore 1 erreurs autorisees

* Essai numero 7
_____
Lettres erronnées:  AZXWQR
lettre = ? E
E apparait 1 fois

* Essai numero 8
____E
Lettres erronnées:  AZXWQR
lettre = ? I
I apparait 1 fois

* Essai numero 9
__I_E
Lettres erronnées:  AZXWQR
lettre = ? F
F apparait 0 fois
Encore 0 erreurs autorisees

Perdu, tu as epuise tes 7 essais,
il fallait trouver BOITE
```

1. Écrire une fonction `lit_lettre()` sans paramètre, qui demande à l'utilisateur de saisir une lettre, filtre la saisie jusqu'à ce qu'il s'agisse bien d'une seule lettre de l'alphabet (non accentuée) en majuscule, et **renvoie** alors la lettre lue.

2. Écrire une fonction `compte_lettre(mot,lettre)` qui reçoit deux chaînes de caractères (un mot et une lettre), et qui utilise une boucle pour compter puis **renvoyer** le nombre d'occurrences (nombre d'apparitions) de cette lettre dans ce mot (on **interdit** d'utiliser la fonction `count`).

3. Écrire une fonction `affiche_mot(mot,connues)` qui reçoit une chaîne de caractères (le mot secret) et une liste de chaînes de caractères (les lettres déjà proposées), et qui **affiche** le mot pour le joueur, avec les lettres connues à leur place, et les lettres non encore connues remplacées par le caractère "-". La fonction ne renvoie rien.
Par exemple `affiche_mot("BONJOUR",["B","O"])` doit afficher "BO- -O- -".

4. Écrire une fonction booléenne `victoire(secret,essais)` qui reçoit le mot secret et la liste des lettres déjà proposées par le joueur, et qui **renvoie** un booléen indiquant si toutes les lettres du mot ont bien été trouvées. Attention à bien arrêter la boucle dès que possible.

5. On suppose qu'on dispose d'une fonction `liste_mots()` qui renvoie une liste de mots générés automatiquement. Écrire une fonction `mot_secret(diff)` qui reçoit un entier $diff$, et qui **renvoie** un mot au hasard parmi ceux de la liste de mots (générée avec `liste_mots`) qui sont de longueur inférieure ou égale au niveau de difficulté donné par $diff$. Si $diff$ est trop faible, on renvoie un mot au hasard parmi tous les mots. On **interdit** d'utiliser la fonction `random.choice` mais vous pouvez utiliser `random.randint`.

6. Écrire une fonction `erreurs(secret,essais)` qui reçoit une chaîne de caractères (le mot secret) et une liste de chaînes de caractères (la liste des lettres déjà proposées par le joueur), et qui **renvoie** les erreurs (c-à-d les lettres proposées qui ne sont pas dans le mot secret), sous la forme au choix d'une liste de caractères ou d'une chaîne de caractères.
Par exemple `erreurs("CHATEAU",["E","A","X","R","T","S"])` doit renvoyer les lettres X, R et S, soit dans une chaîne `"XRS"`, soit dans une liste `["X", "R", "S"]`.

7. Écrire un programme principal qui demande le niveau de difficulté souhaité, génère un mot secret du bon niveau, puis demande au joueur de deviner ce mot en un maximum de 7 **erreurs** (et pas 7 essais !), **en respectant exactement les exemples** ci-dessus :

   - A chaque essai, le programme affiche l'état courant du mot deviné, et la liste des propositions erronées
   - Le programme lit et filtre la lettre proposée par le joueur, puis affiche son nombre d'apparitions dans le mot secret
   - A chaque erreur (et seulement en cas d'erreur) il affiche le nombre d'erreurs encore autorisées
   - Le programme se termine si le joueur fait 7 erreurs, et affiche alors la solution ; ou bien si le joueur trouve le bon mot, et affiche alors son nombre d'essais et d'erreurs.

FIN

Mémo Python - UE INF101 / INF104 / INF131 - version 2021

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

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

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

Opérateurs arithmétiques

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

Fonctions arithmétiques

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

Aléatoire: module random

`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

Instructions conditionnelles

```
if condition :
    instructions

if condition :
    instructions
else :
    instructions
```

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

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 ($i >= 0$).
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)

Définition d'une fonction

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

Fonction qui renvoie la valeur ou variable v.

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