INF101 et INF104 : contrôle continu

Mardi 8 Novembre 2022

- **Durée 2 heures.** Le sujet fait 8 pages. Aucun document, ni calculette/téléphone/...
- **Lisez bien ces consignes avant de commencer !**
- Répondez sur le sujet, dans les cases prévues. Remplissez votre nom sur l'entête de chaque feuillet. Aucune feuille volante ni aucune réponse hors des cases ne sera considérée. Utilisez un brouillon si nécessaire.
- 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. Les questions étoilées sont plus difficiles et valent plus de points.
- Répondez sur le sujet dans les espaces prévus à cet effet. Aucune autre copie ne sera considérée.

1 EXERCICE 1 : BOOLEENS (4 points)

1.1 Expressions booléennes

On suppose que les variables suivantes sont correctement initialisées: a,b,c des entiers positifs, x,y des réels, s1,s2,s des chaînes de caractères (on suppose qu'elles contiennent uniquement des lettres minuscules), b1,b2 des booléens. La **seule** fonction autorisée dans cet exercice est `len()`. Écrire en Python les expressions booléennes suivantes:

1. $s1$ est strictement plus courte que $s2$, et est située après $s2$ dans l'ordre alphabétique

2. s est une lettre (minuscule ou majuscule) mais pas la lettre G majuscule.

3. a secondes équivalent à exactement b minutes et c secondes. *(Il est entendu que b doit être le plus grand nombre de minutes possibles dans a, par exemple 127 secondes équivaut à 2 minutes et 7 secondes, et pas à 1 minute 67.)*

Tournez la page

4. Soit b1 est vrai, soit b2, mais pas les 2 (ou exclusif, mais sans utiliser l'opérateur 'ou exclusif')

5. s1 est une seule lettre minuscule, s2 est une seule lettre majuscule, et ces 2 lettres apparaissent consécutivement dans la chaîne s, dans un ordre ou dans l'autre

6. l'entier n a entre b et c chiffres inclus (on suppose $b < c$ et on interdit ici d'utiliser n'importe quelle fonction, en particulier `str()` est interdite)

1.2 Négations

Écrire la négation des expressions booléennes suivantes, simplifiée au maximum (pas d'opérateur `not`) :

7. $0 <= a <= 20$ or $a\%3 == 0$

8. s1 in ["oui","Oui"]

2 EXERCICE 2 : BOUCLES (6 points)

2.1 Filtrage (1.5 points)

Écrire un programme qui lit une lettre x, puis lit un mot m en le filtrant pour qu'il commence par la lettre x. Le mot est alors affiché avec un message. On inclura un commentaire de sortie de boucle qui indique quelle condition booléenne est vraie à ce moment. Exemple d'exécution (respecter les mêmes affichages à l'**espace** près) :

```
lettre? a
mot qui commence par a? bateau
mot qui commence par a? abricot
Ton mot : abricot
```

1. .....
2. .....
3. .....
4. .....
5. .....
6. .....
7. .....
8. .....
9. .....

2.2 Analyse de code (1.5 points)

Dire ce qui est affiché par les boucles suivantes, en expliquant votre réponse.

```
i = 10
while i <= 0:
    print("*")
    i = i - 1
```

```
i = 1
while i<12 and i%2==1:
    print("*")
    i = i + 1
```

```
i = 0
while i < 10:
    print("*")
    if i%2 != 0:
        i = i + 2
```

2.3 Devine mon nombre, chaud ou froid (3 points)

Écrire un programme qui tire un nombre *secret* au hasard entre 1 et 100 (inclus), et demande à l'utilisateur de le deviner. À chaque essai, le programme indique si le joueur se rapproche ou s'éloigne de la solution comparé à son essai précédent, en affichant "tu chauffes" ou "tu refroidis" (si même distance, il "refroidit"). On considère que le premier essai "chauffe". Le programme se termine en affichant "victoire" et le nombre d'essais quand l'utilisateur trouve *secret*.

Exemple d'exécution avec secret = 50 :

```
Devine mon nombre? 17
Tu chauffes, essaie encore? 57
Tu chauffes, essaie encore? 70
Tu refroidis, essaie encore? 59
Tu chauffes, essaie encore? 42
Tu chauffes, essaie encore? 40
Tu refroidis, essaie encore? 50
Victoire en 7 essais
```

```
1. import random
2. .....
3. .....
4. .....
5. .....
6. .....
7. .....
8. .....
9. .....
10. .....
11. .....
12. .....
13. .....
14. .....
15. .....
16. .....
17. .....
18. .....
```

3 EXERCICE 3 : FECONDITE d'UN ENTIER (4 points)

Soit n un entier naturel (≥ 0). On définit la suite $(u_i)_{i\in\mathbb{N}}$ par :

$$\begin{cases} u_0 &= n \\ u_{i+1} &= u_i + \text{produit des chiffres de } u_i \end{cases}$$

A un moment, le terme u_i contiendra un 0, et donc u_{i+1} et tous les termes suivants resteront égaux à u_i (ajout d'un produit nul). L'indice i de ce premier terme contenant le chiffre 0 est appelé la fécondité de l'entier n de départ. Par exemple avec $u_0 = 7$, on a la suite $u_1 = 14, u_2 = 18, u_3 = 26, u_4 = 38, u_5 = 62, u_6 = 74, u_7 = 102$, donc la fécondité de 7 est 7.

1. Écrire une fonction `zero(n)` qui renvoie un booléen indiquant si l'entier n contient un 0.

Tournez la page

1. .....
2. .....
3. .....
4. .....
5. .....

2. Écrire une fonction `prod(n)` qui utilise une boucle pour calculer le produit des chiffres de l'entier n, et qui renvoie ce produit.

1. .....
2. .....
3. .....
4. .....
5. .....
6. .....
7. .....

3. Écrire une fonction `fecondite(n)` qui utilise les fonctions précédentes dans une boucle, pour calculer et renvoyer la fécondité de l'entier n.

1. .....
2. .....
3. .....
4. .....
5. .....
6. .....
7. .....

4. Écrire un programme principal qui boucle en demandant un entier n à l'utilisateur : tant que l'entier saisi est strictement positif le programme calcule et affiche sa fécondité puis redemande un nouvel entier ; dès que l'entier saisi n'est pas strictement positif, le programme s'arrête.

Tournez la page

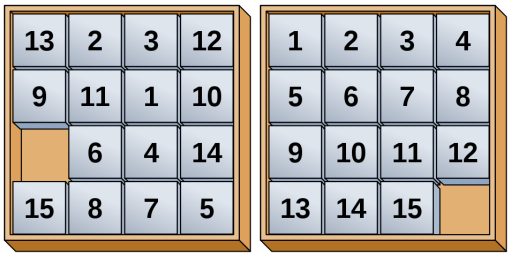
1. .....
2. .....
3. .....
4. .....
5. .....

4 EXERCICE 4 : JEU DU TAQUIN (7 points)

Un taquin d'ordre $n >= 2$ est un jeu solitaire en forme de damier carré de n lignes et n colonnes. Il comprend une seule case vide, les autres cases sont appelées tuiles et sont numérotées de 1 à $n^2 - 1$. La case vide permet d'y faire coulisser une tuile voisine. Le but du jeu est de déplacer les tuiles jusqu'à ce qu'elles soient dans l'ordre croissant, avec la case vide en bas à droite. La figure montre un taquin d'ordre 4, mélangé (à gauche) et résolu (à droite).

1. Écrire une fonction `taquinAlea(n)` qui génère et renvoie une liste de taille n^2 contenant les numéros dans le désordre, de 0 (représentant la case vide) à $n^2 - 1$. On ne s'occupe pas de savoir si le puzzle est possible. On rappelle la fonction `random.shuffle` permettant de mélanger une liste. *Par exemple pour $n = 3$ une liste possible est* `[7, 6, 8, 2, 4, 5, 1, 0, 3]`

1. .....
2. .....
3. .....
4. .....
5. .....
6. .....
7. .....
8. .....




```
13.2.3.12.
9.11.1.10.
 .6.4.14.
15.8.7.5.
```

Tournez la page

2. Écrire une fonction `affiche(taq)` qui reçoit une liste représentant un taquin carré, et l'affiche textuellement sous la forme d'un carré de côté n, avec un **espace** pour la case vide, et un point après chaque case. On ne demande pas d'aligner le texte *(cf exemple à côté de l'image.)*

1. .....
2. .....
3. .....
4. .....
5. .....
6. .....
7. .....
8. .....
9. .....
10. .....

3. Écrire une fonction `coordXY(numero, taq)` qui reçoit le taquin et un numéro (on **suppose** que ce numéro est **présent** dans le taquin), et renvoie une liste de 2 entiers contenant ses coordonnées dans le puzzle. *Par ex. pour le taquin mélangé de l'image, les coordonnées du numéro 0 sont [2,0]: ligne 2, colonne 0.*

1. .....
2. .....
3. .....
4. .....
5. .....
6. .....

4. Écrire une fonction `deplacable(numero,taq)` qui reçoit le taquin et un numéro, et renvoie un booléen indiquant si la tuile portant ce numéro peut se déplacer. *Dans l'ex. ci-dessus, 1, 4 et 3 sont déplaçables, les autres chiffres non.* On utilisera la fonction `coordXY` ci-dessus.

1. .....
2. .....
3. .....
4. .....

Tournez la page

5. Écrire une fonction `lireNumero(taq)` qui reçoit la liste représentant le taquin, interroge l'utilisateur pour connaître le numéro à déplacer, le filtre jusqu'à ce qu'il s'agisse d'un numéro valide (faisant partie du puzzle et pouvant être déplacé), et le renvoie une fois valide.

```
1. .....
2. .....
3. .....
4. .....
5. .....
6. .....
```

6. Écrire une fonction `deplacer(numero,taquin)` qui reçoit le taquin et un numéro à déplacer (qu'on **suppose valide**). La fonction **modifie** la liste *taquin* en déplaçant la tuile choisie vers la case vide, et ne renvoie **rien**.

```
1. .....
2. .....
3. .....
4. .....
5. .....
```

7. Écrire une fonction `jouerUn()` qui demande à l'utilisateur la taille du puzzle, génère un taquin aléatoire, l'affiche, puis lit et exécute 1 mouvement, et affiche le taquin ainsi modifié.

```
1. .....
2. .....
3. .....
4. .....
5. .....
6. .....
7. .....
```

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.

`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)