Bases de l'informatique 1 — SPUF100

Année 2024-2025 — Examen terminal

Nom:
Prénom:
Numéro d'étudiant :

Durée: 2 heures.

Aucun document n'est autorisé. L'usage de la calculatrice ou de tout autre appareil électronique est interdit.

Les exercices sont indépendants. Au sein d'un même exercice, vous pouvez utiliser les variables et fonctions des questions précédentes, même si vous n'avez pas su les faire; chaque question est donc indépendante.

À part les méthodes et fonctions de base, vous n'avez pas le droit d'utiliser les fonctions et les méthodes « avancées », sauf si l'énoncé vous conseille l'utilisation de certaines d'entre elles.

```
# Fonctions autorisées
range(...) len(...) print(...)

# Boucles autorisées
Boucles while
Boucles for avec un range
Boucles for sans range (for x in L)

# Vous pouvez utiliser les compréhensions ou les slices
[ x for x in range(L) ]
chaine[début:fin:pas]

# Les méthodes suivantes sont autorisés :
L.append(...)
```

```
# Par exemple les méthodes et fonctions suivantes sont entre autres interdites
max(...) min(...) sum(...) abs(...)
s.split(...) s.index(...) L.extend(...)
```

	Exercice 1 Identifiants de code-barres
	0 0,5 1 1,5 2 2,5 3 3,5 4 4,5 5 5,5
	1. En utilisant les fonctions chr et ord, écrire la fonction entier (chaîne) qui transforme une chaîne de longueur 1 représentant un chiffre en l'entier correspondant. Écrire de même la fonction chiffre (n) transformant un nombre entier entre 0 et 9 (compris) en chaîne.
1 2 3	>>> entier("3") 3 >>> chiffre(7)
1	171
	2. Écrire une fonction tester_identifiant(ch) qui teste si la chaîne donnée en paramètre est bien une chaîne de 7 chiffres. On ne pourra évidemment pas utiliser la méthode ch.isdigit().
1	>>> tester_identifiant("1234567")
2	True 2 True >>> tester_identifiant("12345678") 3 >>> tester_identifiant("444X222")
4	False False

3. Un identifiant pour code barre (pour la norme EAN-8) est composé de 7 chiffres quelconques et d'un 8e chiffre (appelé la clé) calculé à partir des 7 premiers. Pour calculer la clé, on applique la méthode suivante : A) On ajoute les différents chiffres en les pondérant alternativement par 3 ou 1. B) Si le chiffre des unités du nombre obtenu est non nul, on soustrait ce chiffre à 10, sinon, la clé est zéro. Par exemple pour calculer la clé associée à l'identifiant "1564781". On commence par calculer la somme pondérée :

 $3 \times 1 + 1 \times 5 + 3 \times 6 + 1 \times 4 + 3 \times 7 + 1 \times 8 + 3 \times 1 = 62$ (le premier chiffre est multiplié par 3, le suivant par 1, celui d'après par 3 à nouveau, etc). Le chiffre des unités de 62 étant 2, la clé est donc 10 - 2 = 8.

Écrire une fonction ajouter_clé(ch) qui à partir d'un identifiant de 7 chiffres renvoie une nouvelle chaîne de 8 chiffres obtenue en ajoutant la clé à la fin.

1. Écrire une fonction chaînes_vers_liste(ch) qui prend en argument une chaîne composée de 10
groupes de quatre chiffres séparés par des tirets et la transforme en une liste de 10 chaînes (chaque chaîne
correspondant à un groupe de quatre chiffres). On ne pourra pas utiliser la méthode ch.split("-"),
mais on pourra utiliser les slices (chaîne[début:fin:pas]).

```
>>> valeurs = "3211-2221-2122-1411-1132-1231-1114-1312-1213-3112"
>>> chaînes_vers_liste(valeurs)
['3211', '2221', '2122', '1411', '1132', '1231', '1114', '1312', '1213', '3112']
```

Dorénavant, et pour la suite de l'exercice, on définit la variable encodage comme ci-dessous.

```
>>> encodage = chaînes_vers_liste(valeurs)
```

	• • • •	• • • • • • • • • • • • • • • • • • • •	
1). C'est la succession de ces barres qui nous donr On représente une barre noire par une suite de syr 0. Il y a en fait deux formats : le format A corresp format C correspond aux barres de couleurs Noir-	nera mbo ond -Bla f orn	oles 1 et une barre blanche par une suite de symbole l aux barres de couleurs Blanc-Noir-Blanc-Noir et l nc-Noir-Blanc. mat) qui prend en argument une chaîne de chiffre	s e
>>> traduire_vers_chaîne("3211", "A")		•	
		>>> Tradilira Ware chalha(" 3 7" "/")	
'0001101'	1	>>> traduire_vers_chaîne("1312", "A") '0111011'	
	1 2 3		
'0001101' >>> traduire_vers_chaîne("3211", "C")		'0111011' >>> traduire_vers_chaîne("1312", "C")	
'0001101' >>> traduire_vers_chaîne("3211", "C")		'0111011' >>> traduire_vers_chaîne("1312", "C")	٠
'0001101' >>> traduire_vers_chaîne("3211", "C")		'0111011' >>> traduire_vers_chaîne("1312", "C")	
'0001101' >>> traduire_vers_chaîne("3211", "C")		'0111011' >>> traduire_vers_chaîne("1312", "C")	
'0001101' >>> traduire_vers_chaîne("3211", "C")		'0111011' >>> traduire_vers_chaîne("1312", "C")	
'0001101' >>> traduire_vers_chaîne("3211", "C")		'0111011' >>> traduire_vers_chaîne("1312", "C")	
'0001101' >>> traduire_vers_chaîne("3211", "C")		'0111011' >>> traduire_vers_chaîne("1312", "C")	
'0001101' >>> traduire_vers_chaîne("3211", "C")		'0111011' >>> traduire_vers_chaîne("1312", "C")	
'0001101' >>> traduire_vers_chaîne("3211", "C")		'0111011' >>> traduire_vers_chaîne("1312", "C")	
'0001101' >>> traduire_vers_chaîne("3211", "C")		'0111011' >>> traduire_vers_chaîne("1312", "C")	
'0001101' >>> traduire_vers_chaîne("3211", "C")		'0111011' >>> traduire_vers_chaîne("1312", "C")	
'0001101' >>> traduire_vers_chaîne("3211", "C")		'0111011' >>> traduire_vers_chaîne("1312", "C")	
'0001101' >>> traduire_vers_chaîne("3211", "C")		'0111011' >>> traduire_vers_chaîne("1312", "C")	
'0001101' >>> traduire_vers_chaîne("3211", "C")		'0111011' >>> traduire_vers_chaîne("1312", "C")	

3. On peut enfin passer à l'encodage final. Notre code permet de coder les 8 chiffres par un long nombre binaire représenté sous forme de chaîne. Supposons que l'on souhaite coder 96333693.

chiffres		9	6	3	3		3	6	9	3	
encodage		3112	1114	1411	1411		1411	1114	3112	1411	
blocs	init.	C	à traduire d	ıu format 1	4	centre	C	à traduire d	ıu format (C	fin
séquences	101	0001011	0101111	0111101	0111101	01010	1000010	1010000	1110100	1000011	101

La représentation binaire finale se compose de 11 séquences réparties sur 5 blocs :

- (a) une séquence initiale 101
- (b) les quatre séquences correspondant aux 4 premiers chiffres au format A
- (c) une séquence centrale 01010
- (d) les quatre séquences correspondant aux 4 derniers chiffres au format C
- (e) une séquence finale 101

Écrire la fonction encoder (ch) qui renvoie l'écriture binaire correspondant à la représentation sous forme de code-barres de la chaîne ch. On pourra utiliser la liste encodage définie de manière globale à la question 1 qui contient les 10 codes associés (dans le bon ordre) aux dix chiffres de 0 à 9.

>>> encoder("96333693")	
'1010001011010111101111101011110101010000	
	· • •



La représentation d'un code-barre au format binaire n'est pas très lisible. Nous allons écrire avec Tkinter un petit programme qui affiche le motif associé à un code-barre comme dans l'image ci-contre.

Concrètement chaque 1 correspondra à une colonne noire et chaque 0 à une colonne blanche. Une bande noire épaisse est simplement une succession de plusieurs colonnes noires.

On commence à se donner le code ci-dessous qui crée et initialise la fenêtre Tk.



```
import tkinter as tk

# Variables globales

Hauteur = 500

Largeur = 700

Marge = 100

# On crée une fenêtre et un canevas (zone de dessin)

root = tk.Tk()

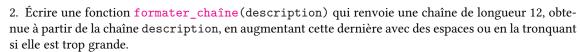
root.title("Générateur de code barre")

Dessin=tk.Canvas(root,height=Hauteur,width=Largeur,bg="white")

Dessin.pack()
```

Dessin.pack()
1. On cherche à calculer la largeur d'une colonne à partir de la largeur, des marges de chaque côté et du nombre n de colonnes donné en paramètre. Écrire la fonction calculer_delta(n) renvoyant la largeu d'une telle colonne.
2. Écrire alors une fonction tracer_colonne(n, d) qui trace en noir et en partant de la marge de gauche la nème colonne de largeur delta.
On utilisera la fonction vue en cours, Dessin.create_rectangle(x0,y0,x1,y1,fill="black") of (x0,y0) et (x1,y1) sont les coordonnées de deux sommets diagonaux du rectangle.

3. En déduire une fonction tracer_code_barre(numéro) qui affiche le motif associé au numéro donné en paramètre. On utilisera évidemment la fonction encoder de l'exercice précédent ainsi que la fonction tracer_colonne.
Exercice 4 Formatages divers
1. Écrire une fonction découper (ligne) qui découpe une ligne de trois éléments en un tuple de trois chaînes de caractères. On ne pourra pas utiliser la méthode split.
>>> découper("1234561,TABLETTE CHOCOLAT,10.00")
('1234561', 'TABLETTE CHOCOLAT', '10.00')



1	>>> formater_chaîne("CHAUSSETTE")
2	'CHAUSSETTE '
3	>>> formater_chaîne("CHARCUTAILLE")
4	'CHARCUTAILLE'
5	>>> formater_chaîne("BOXER EN POIL DE LAPIN")
6	'BOXER EN POI'

+1	/a	/52+

Exercice 5 Base de données et tickets de caisse	. 3,5 points
0 0,5 1 1,5 2 2,5 3 3,5	
On se donne un fichier texte intitulé bdd. txt qui contient l'ensemble des références d'un magas.	in. Chaque
ligne contient trois éléments séparés par des virgules : le code-barre, la description et le prix un ligne contient trois éléments séparés par des virgules : le code-barre, la description et le prix un ligne contient trois éléments séparés par des virgules : le code-barre, la description et le prix un ligne contient trois éléments séparés par des virgules : le code-barre, la description et le prix un ligne contient trois éléments séparés par des virgules : le code-barre, la description et le prix un ligne contient trois éléments séparés par des virgules : le code-barre, la description et le prix un ligne contient de la code-barre, la description et le prix un ligne contient de la code-barre, la description et le prix un ligne contient de la code-barre, la description et le prix un ligne contient de la code-barre, la co	nitaire.
1234561.TABLETTE CHOCOLAT.10.23	

```
1 1234561, TABLETTE CHOCOLAT, 10.23

2 1785436, SAUCISSON, 5.26

3 7895647, DÉODORANT, 3.25

4 7535417, SLIP DE LUXE, 57.75
```

1. En déduire une fonction lire_base_de_données (nom_fichier) qui à partir du contenu du fichier donné en paramètre renvoie un dictionnaire dont les clés sont les codes-barres et les valeurs, des couples de la forme (description, prix). On pourra utiliser la fonction float("12.5") pour convertir des chaînes en flottant ainsi que la fonction découper de l'exercice précédent.

```
>>> BDD = lire_base_de_données("bdd.txt")
>>> BDD['1785436']
('SAUCISSON', 5.26)
```

2. Écrire une fonction ticket_de_caisse(liste_codes, BDD, nom_fichier) qui à partir d'une liste de codes-barres construit le ticket de caisse correspondant. Le résultat ne sera pas affiché mais stocké dans un fichier texte dont le nom est fourni par la chaîne nom_fichier.

On fera attention de gérer proprement les alignements et on calculera deux totaux : le total des prix des produits alimentaires (ceux dont le code-barre commence par 1) et le total complet de tous les prix.

```
>>> ticket_de_caisse(["1785436","7535417","1785436","1234561"],BDD,"ticket.txt")
```

```
SAUCISSON 5.26€

SLIP DE LUXE 57.75€

SAUCISSON 5.26€

TABLETTE CHO 10.00€

TOTAL : 78.27€

DONT ALIM. : 20.52€
```