+1/1/60+

Bases de l'informatique 1 — SPUF100

Année 2024-2025 — Examen terminal

Nom : .....

Prénom : .....

Numéro d'étudiant : .....

☐0 ☐1 ☐2 ☐3 ☐4 ☐5 ☐6 ☐7 ☐8 ☐9
☐0 ☐1 ☐2 ☐3 ☐4 ☐5 ☐6 ☐7 ☐8 ☐9
☐0 ☐1 ☐2 ☐3 ☐4 ☐5 ☐6 ☐7 ☐8 ☐9

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 . . . . . 5,5 points

☐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.

```
>>> entier("3")
3
>>> chiffre(7)
'7'
```

```
def entier(ch):
    return ord(ch) - ord('0')

def chiffre(i):
    return chr(i + ord('0'))
```

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()`.

```
>>> tester_identifiant("1234567")
True
>>> tester_identifiant("12345678")
False
```

```
>>> tester_identifiant("4442222")
True
>>> tester_identifiant("444X222")
False
```

```
def tester_identifiant(ch):
    if len(ch) != 7:
        return False
    for c in ch:
        if ord(c) < ord("0") or ord("9") < ord(c):
            return False
    return True
```

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 :

$3\times\mathbf{1} + 1\times\mathbf{5} + 3\times\mathbf{6} + 1\times\mathbf{4} + 3\times\mathbf{7} + 1\times\mathbf{8} + 3\times\mathbf{1} = 62$ (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.

```
>>> ajouter_clé("1564781") # la clé calculée avec la méthode précédente est 8
'15647818'
```

```
def ajouter_clé(ch):
    poids = 3
    somme = 0
    for c in ch:
        somme = somme + poids * entier(c)
        poids = 4 - poids # transforme 3 en 1 et 1 en 3
    nouveau = somme%10
    if nouveau != 0:
        nouveau = 10-nouveau
    return ch + chiffre(nouveau)
```

Exercice 2 Encodage . . . . . 5 points

☐0 ☐0,5 ☐1 ☐1,5 ☐2 ☐2,5 ☐3 ☐3,5 ☐4 ☐4,5 ☐5

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)
```

```
def chaînes_vers_liste(ch):
    L = []
    déb = 0
    fin = 4
    for i in range(10):
        L.append(ch[déb:fin])
        déb += 5
        fin += 5
    return L
```

2. Chaque quadruplet, comme "3211", représente 4 barres de largeurs variables (une première barre de largeur 3, suivie d'une barre de largeur 2 puis d'une autre de largeur 1 et enfin d'une dernière de largeur 1). C'est la succession de ces barres qui nous donnera le fameux code-barre.

On représente une barre noire par une suite de symboles 1 et une barre blanche par une suite de symboles 0. Il y a en fait deux formats : le format A correspond aux barres de couleurs Blanc-Noir-Blanc-Noir et le format C correspond aux barres de couleurs Noir-Blanc-Noir-Blanc.

Écrire la fonction `traduire_vers_chaîne(ch,format)` qui prend en argument une chaîne de chiffres et un format ("A" ou "C") et renvoie le code binaire correspondant.

```
>>> traduire_vers_chaîne("3211", "A")
'0001101'
>>> traduire_vers_chaîne("3211", "C")
'1110010'
```

```
>>> traduire_vers_chaîne("1312", "A")
'0111011'
>>> traduire_vers_chaîne("1312", "C")
'1000100'
```

```
def traduire_vers_chaîne(ch,type_format):
    symbole="0"
    suivant="1"
    if type_format == "C":
        symbole, suivant = suivant, symbole
    rés =""
    for valeur in ch:
        rés += symbole * entier(valeur)
        temp = symbole
        symbole = suivant
        suivant = temp
    return rés
```

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.*	*à traduire au format A*				*centre*	*à traduire au 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.

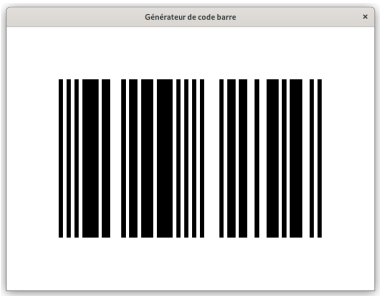
```
>>> encoder("96333693")
'10100010110101111011110101111010101000010101000011101001000010101'
```

```
def encoder(identifiant):
    bord = "101"
    milieu = "01010"
    rés = bord
    for i in range(0,4):
        c = entier(identifiant[i])
        code  = encodage[c]
        rés = rés + traduire_vers_chaîne(code, "A")
    rés = rés + milieu
    for i in range(4,8):
        c = entier(identifiant[i])
        code = encodage[c]
        rés = rés + traduire_vers_chaîne(code, "C")
    rés = rés + bord
    return rés
```

Exercice 3 Dessinons le code-barre . . . . . 3 points

☐0 ☐0,5 ☐1 ☐1,5 ☐2 ☐2,5 ☐3

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()
```

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 largeur d'une telle colonne.

```
def calculer_delta(n):
    return (Largeur - 2*Marge) / n
```

2. Écrire alors une fonction `tracer_colonne(n, d)` qui trace en noir et en partant de la marge de gauche la $n^{ème}$ colonne de largeur `delta`.
On utilisera la fonction vue en cours, `Dessin.create_rectangle(x0,y0,x1,y1,fill="black")` où `(x0,y0)` et `(x1,y1)` sont les coordonnées de deux sommets diagonaux du rectangle.

```
def tracer_colonne(n, delta):
    x0 = Marge + n * delta
    x1 = x0 + delta
    y0 = Marge
    y1 = Hauteur - Marge
    Dessin.create_rectangle(x0, y0, x1, y1, fill="black")
```

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`.

```
def tracer_code_barre(numéro):
    code = encoder(numéro)
    delta = calculer_delta(len(code))
    for i in range(len(code)):
        if code[i] == "1":
            tracer_colonne(i,delta)
```

Exercice 4 Formatages divers . . . . . 3 points

☐0 ☐0,5 ☐1 ☐1,5 ☐2 ☐2,5 ☐3

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')
```

```
def découper(chaîne):
    code = ""
    description = ""
    prix = ""
    élément=0 # 0:code 1:description 2:prix
    for c in chaîne:
        if c == ",":
            élément = élément + 1
        elif élément == 0:
            code += c
        elif élément == 1:
            description += c
        else:
            prix += c
    return (code, description,prix )
```

2. Écrire une fonction `formater_chaîne(description)` qui renvoie une chaîne de longueur 12, obtenue à partir de la chaîne `description`, en augmentant cette dernière avec des espaces ou en la tronquant si elle est trop grande.

```
>>> formater_chaîne("CHAUSSETTE")
'CHAUSSETTE  '
>>> formater_chaîne("CHARCUTAILLE")
'CHARCUTAILLE'
>>> formater_chaîne("BOXER EN POIL DE LAPIN")
'BOXER EN POI'
```

```python
def formater_chaîne(description):
    résultat = ""
    for i in range(12):
        if i >= len(description):
            résultat += " "
        else:
            résultat += description[i]
    return résultat
```

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 magasin. Chaque ligne contient trois éléments séparés par des virgules : le code-barre, la description et le prix unitaire.

```
1234561,TABLETTE CHOCOLAT,10.23
1785436,SAUCISSON,5.26
7895647,DÉODORANT,3.25
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)
```

```
def lire_base_de_données(nom_fichier):
    fichier = open(nom_fichier, 'r', encoding='UTF-8')
    dico = dict()
    for ligne in fichier:
        (identifiant, description, prix) = découper(ligne)
        dico[identifiant]  = (description, float(prix))
    fichier.close()
    return dico
```

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€
```

```
def formater_prix(prix):
    if prix < 10:
        return f" {prix:.2f}"
    else:
        return f"{prix:.2f}"

def ticket_de_caisse(listes_code, BDD, nom_fichier):
    fichier = open(nom_fichier, "w", encoding="UTF-8")
    prix_total = 0
    prix_nourriture = 0

    for code in listes_code:
        if code in BDD:
            (description, prix) = BDD[code]
            if code[0] == "1":
                prix_nourriture = prix_nourriture + prix
            prix_total = prix_total + prix
            ch = formater_chaîne(description) +" "+ formater_prix(prix) + "€\n"
            fichier.write(ch)
    fichier.write("-"*19 + "\n")
    fichier.write("TOTAL      : " + formater_prix(prix_total) + "€\n")
    fichier.write("DONT ALIM. : " + formater_prix(prix_nourriture)  + "€\n")
    fichier.close()
```