+1/1/60+

Bases de l'informatique 1 — SPUF100

Année 2025-2026 — Partiel

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 et boucles autorisées
range(…) len(…) print(…)
Boucles while et boucles for avec un range

# Déstructuration autorisée
(x,y) = p # où p est un tuple de longueur 2
```

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

# Les boucles itérant directement sur les séquences sont interdites
for x in L:

# Vous n'avez pas le droit d'utiliser des compréhensions ou des slices
# À la place vous devez utiliser des boucles.
x in L
[ x for x in range(L) ]
chaine[début:fin:pas]
```

Exercice 1 Calculs binaires . . . . . 3,5 points

☐0 ☐0,5 ☐1 ☐1,5 ☐2 ☐2,5 ☐3 ☐3,5

1. En utilisant l'algorithme vu en cours, écrire 39 en binaire.

```
39 = 2 × 19 + 1
19 = 2 ×  9 + 1 # J'enleverai des points pour ceux
 9 = 2 ×  4 + 1 # utilisant une autre présentation que :
 4 = 2 ×  2 + 0 # … = 2 × … + …
 2 = 2 ×  1 + 0
 1 = 2 ×  0 + 1
39 est donc représenté par 10 0111 en binaire
```

2. En déduire la représentation binaire signée sur 8 bits de -39.

```
On part de 39 sur 8 bits   : 0010 0111
On prend le complément à 2 : 1101 1000
et on ajoute 1             : 1101 1001
-39 est donc représenté par 1101 1001  en binaire sur 8 bits
```

3. En utilisant l'algorithme vu en cours, donnez l'écriture décimale du nombre positif dont la représentation binaire est 1100110.

```
Sur 8 bits 110 0110 commence par 0, c'est un nombre positif
1        = 1
11       = 2 ×  1 + 1 = 3
110      = 2 ×  3 + 0 = 6
110 0    = 2 ×  6 + 0 = 12
110 01   = 2 × 12 + 1 = 25
110 011  = 2 × 25 + 1 = 51
110 0110 = 2 × 51 + 0 = 102

110 0110  vaut donc 102
```

4. Donnez l'écriture décimale du nombre dont la représentation signée sur 8 bits binaire est `10011001`.

```
Le nombre commençant par 1 sur 8 bits, c'est un négatif
On a la représentation sur 8 bits : 1001 1001
On prend le complément à 2        : 0110 0110 (tiens ? mais c'est 102 !)
et on ajoute 1                    : 0110 0111
On reconnaît 102 + 1 = 103, donc le nombre de départ est -103.
```

Exercice 2 Organiser un congrès espérantiste . . . . . 4 points

☐0 ☐0,5 ☐1 ☐1,5 ☐2 ☐2,5 ☐3 ☐3,5 ☐4

Dans ce monde barbare où chaque jour de nouvelles guerres sont déclarées, où de grand défis doivent être résolus à l'échelle du globe, il est nécessaire de répandre un outil de paix et de compréhension mutuelle entre les peuples, une véritable langue internationale qui ne s'impose pas par la loi du plus fort : l'*espéranto*. À défaut de changer la face du monde et de participer à la création d'un avenir meilleur, l'espéranto servira au moins de prétexte à vous évaluer en python pour ce partiel ; c'est toujours ça de pris.

Les espérantistes aiment se réunir en congrès. On se décide à faire des statistiques sur le pays d'origine de chacun des participants du congrès. On donne ci-dessous un exemple de telle statistique sous forme de liste donnant le nombre de participants par pays : il y a ici, entre autres, 7 finlandais et 3 iraniens.

```
1 stat = [("Finlande",7),("France",20),("Hongrie", 15),("Japon", 5), ("Iran",3)]
```

1. Écrire une fonction `liste_pays(stat)` qui à partir d'une statistique `stat` renvoie un nouveau tableau ne contenant que le nom des pays.

```
1 >>> liste_pays(stat)
2 ['Finlande', 'France', 'Hongrie', 'Japon', 'Iran']
```

```
def liste_pays(stat):
    tab = [0] * len(stat)
    for i in range(len(stat)):
        (p,n) = stat[i]
        tab[i] = p
    return tab
```

2. Écrire une fonction `plus_représenté(stat)` qui renvoie le nom du pays contenant le plus de participants. On supposera la liste non vide (l'organisateur du congrès étant forcément présent dans les statistiques) et en cas d'égalité entre pays, on renverra le premier parmi eux dans la liste.

```
1 >>> plus_représenté(stat)
2 'France'
3 >>> plus_représenté([("Brésil",15),("Togo",2),("Espagne",15)])
4 'Brésil'
```

```
def plus_représenté(stat):
    pmax = None
    nmax = 0
    for i in range(len(stat)):
        (p,n) = stat[i]
        if nmax<n:
            pmax = p
            nmax = n
    return pmax
```

3. Afin d'encourager la diversité du mouvement espérantiste, l'organisateur du congrès décide d'offir un livre à tous les membres du congrès appartenant à un pays avec peu de représentants. Écrire une fonction `pays_peu_représentés(stat)` qui renvoie la liste des pays ayant strictement moins que 10 participants. On commencera à compter le nombre de pays concernés afin de créer un tableau de la bonne taille avant de le remplir avec le nom des pays (rappel : la méthode append et les compréhensions sont interdites).

```
>>> pays_peu_représentés(stat)
['Finlande', 'Japon', 'Iran']
>>> pays_peu_représentés([("Brésil",15),("Togo",2),("Espagne",15)])
['Togo']
```

```
def pays_peu_représentés(stat):
    taille = 0
    for i in range(len(stat)):
        (p,n) = stat[i]
        if n<10:
            taille = taille + 1
    tab = [0] * taille
    j=0
    for i in range(len(stat)):
        (p,n) = stat[i]
        if n<10:
            tab[j] = p
            j = j+1
    return tab
```

Exercice 3 L'alphabet de l'espéranto . . . . . 4 points

☐0 ☐0,5 ☐1 ☐1,5 ☐2 ☐2,5 ☐3 ☐3,5 ☐4

On s'intéresse à détecter si un texte est en espéranto ou non. Pour cela, on va se baser sur l'alphabet de l'espéranto qui contient certaines lettres non présentes dans la table ASCII (les 128 premiers caractères d'Unicode) et qui au contraire interdit certaines lettres. On définit les deux chaînes ci-dessous, nécessaires pour répondre à la question 2.

```
en_plus = "ĉĝĥĵŝŭĈĜĤĴŜŬ" # Lettres accentués de l'espéranto
en_moins = "qwxyQWXY"     # Lettres non présentes en espéranto
```

1. Écrire une fonction `appartient(symbole, chaîne)` qui renvoie `True` si le caractère `symbole` est présent dans la chaîne et `False` sinon.

```
def appartient(symbole, chaîne):
    for i in range(len(chaîne)):
        if chaîne[i] == symbole:
            return True
    return False
```

2. En déduire une fonction `symbole_espérantiste(car)` qui renvoie `True` si et seulement si une des deux conditions suivantes est vérifiée (et renverra `False` sinon).
   (a) `car` est un symbole de la table ASCII (on utilisera la fonction `ord`) mais absent de la chaîne `en_moins`
   (b) `car` est un caractère de la chaîne `en_plus`

Pour avoir tous les points, votre réponse devra tenir sur une unique ligne et utiliser la fonction précédente.

```
def symbole_espérantiste(c):
    return … # À compléter avec une expression booléenne
```

```
def symbole_espérantiste(c):
    return (ord(c)<=127 or appartient(c,en_plus)) and not appartient(c,en_moins)
```

3. En déduire une fonction `nombre_caractères_invalides(texte)` qui renvoie le nombre de caractères invalides dans un texte en espéranto. On utilisera la fonction précédente.

```
>>> nombre_caractères_invalides("3€ en été") # caractères invalides : "€éé"
3
>>> nombre_caractères_invalides("How does owl fly ? Quoi ?") # invalides "wwyQ"
4
```

```
def nombre_caractères_invalides(texte):
    rés = 0
    for i in range(len(texte)):
        if not symbole_espérantiste(texte[i]):
            rés = rés + 1
    return rés
```

Exercice 4 Compter en espéranto . . . . . 4,5 points

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

En espéranto, la règle pour prononcer les nombres est particulièrement simple. Par exemple en français 876 se prononce $8 \times 100 + 60 + 16$ (huit cent soixante seize). En espéranto on aura simplement $8 \times 100 + 7 \times 10 + 6$ (ok cent sep dek ses). L'objectif de l'exercice est de traduire un nombre entre 0 et 9999 en espéranto. Pour cela on se donne les deux listes suivantes

```
#           0      1      2     3      4       5      6      7      8     9
chiffres=["nulo", "unu", "du", "tri", "kvar", "kvin", "ses", "sep", "ok", "naŭ"]
#                 1      10    100     1000
puissances = [ "unu", "dek", "cent", "mil" ]
```

1. Écrire une fonction `concat(a,b)` qui renvoie la chaîne obtenue en concatènant les deux chaînes `a` et `b` avec un espace entre les deux si les deux chaînes sont non vides. Si par contre une des deux chaînes est vide, on renverra l'autre.

```
>>> concat("ok","cent")
'ok cent'
>>> concat("","cent")
'cent'
```

```
>>> concat("ok","")
'ok'
>>> concat("","")
''
```

```
def concat(a,b):
    if a=="" or b=="":
        return a+b
    else:
        return a+" "+b
```

2. Écrire une fonction `nombre_simple(n,k)` qui affiche en espéranto le nombre $n \times 10^k$. On suivra, par ordre de priorité, les quatre règles ci-dessous.

   (a) Si `n` est nul (égal à 0) on renverra la chaîne vide
   (b) Si `k` est nul on renverra directement le chiffre correspondant à `n`
   (c) Si `n` vaut 1 on renvoie la puissance sans son coefficient
   (d) Sinon on renvoie la chaîne contenant le chiffre correspondant à `n` suivi de sa puissance.

```
>>> nombre_simple(0,3) # régle (a)
''
>>> nombre_simple(0,9) # régle (a)
''
>>> nombre_simple(1,0) # régle (b)
'unu'
>>> nombre_simple(3,0) # régle (b)
'tri'
```

```
>>> nombre_simple(1,2) # régle (c)
'cent'
>>> nombre_simple(1,3) # régle (c)
'mil'
>>> nombre_simple(2,3) # régle (d)
'du mil'
>>> nombre_simple(5,1) # régle (d)
'kvin dek'
```

```
def nombre_simple(n,k):
    if n==0:
        return ""
    elif k==0: # C'est un chiffre des unités
        return chiffres[n]
    elif n==1: # pas de multiplicité
        return puissances[k]
    else:
        return chiffres[n] + " " + puissances[k]
```

3. En déduire une fonction `nombro(n)` qui traduit le nombre `n` en espéranto. Pour avoir tous les points, il faudra lire les chiffres avec une boucle `while` en appliquant l'algorithme de l'exercice 1 mais appliqué à la base 10. On fera attention à vérifier que l'algorithme fonctionne dans les cas ci-dessous et en particulier le zéro.

```
>>> nombro(9106)
'naŭ mil cent ses'
>>> nombro(1111)
'mil cent dek unu'
>>> nombro(1010)
'mil dek'
```

```
>>> nombro(0)
'nulo'
>>> nombro(7)
'sep'
>>> nombro(21)
'du dek unu'
```

```
def nombro(n):
    rés=""
    if n== 0:
        return "nulo"
    else:
        k=0
        while n>0:
            r = n%10
            q = n//10
            rés = concat( nombre_simple(r,k), rés)
            k=k+1
            n=q
    return rés
```

Exercice 5 Encodage et système X . . . . . 4 points

☐0 ☐0,5 ☐1 ☐1,5 ☐2 ☐2,5 ☐3 ☐3,5 ☐4

L'espéranto possède la particularité d'utiliser des lettres accentuées. À l'époque pré-unicode, dans une communauté répartie autour du globe dans des pays utilisant des codages divers et variés, les espérantistes ont développé un système permettant de représenter leur langue sans perdre d'information : le système X (en langue originale : *la X-sistemo*).

```
traduction = [ ("ĥ","hx"), ("Ĥ","Hx"),
               ("ĝ","gx"), ("Ĝ","Gx"),
               ("ĵ","jx"), ("Ĵ","Jx"),
               ("ŭ","ux"), ("Ŭ","Ux"),
               ("ĉ","cx"), ("Ĉ","Cx"),
               ("ŝ","sx"), ("Ŝ","Sx") ]
```

1. On se donne le dictionnaire ci-dessus qui contient les lettres accentuées ainsi que la chaîne que l'on va leur substituer. Écrire une fonction `encoder_lettre(c)` qui renvoie la nouvelle chaîne correspondante à `c` si `c` apparaît en première position dans un couple de la liste. Si `c` n'apparaît nulle part dans la liste précédente, on renverra alors directement `c`.

```
>>> encoder_lettre("ĥ") # car ("ĥ", "hx") est dans traduction
'hx'
>>> encoder_lettre("Ŭ") # car ("Ŭ", "Ux") est dans traduction
'Ux'
>>> encoder_lettre("a") # il n'y a pas de couple ('a', _) dans traduction
'a'
>>> encoder_lettre("!") # il n'y a pas de couple ('!', _) dans traduction
'!'
```

```
def encoder_lettre(lettre):
    for i in range(len(traduction)):
        (avant, après) = traduction[i]
        if avant == lettre:
            return après
    return lettre
```

2. En déduire une fonction `encoder(texte)` qui renvoie la chaîne de caractères obtenue en remplaçant tous les symboles de la chaîne `texte` avec la fonction précédente.

```
>>> encoder("Unu lingvo por ŝanĝi la mondon kontraŭ la ĥaoso")
'Unu lingvo por sxangxi la mondon kontraux la hxaoso'
```

```
def encoder(texte):
    rés = ""
    for i in range(len(texte)):
        rés += encoder_lettre(texte[i])
    return rés
```

3. Dans le monde moderne où nous sommes, dans lequel règne Unicode, inutile de s'embêter avec le système X. Écrire la fonction inverse pour décoder les anciens textes.

```
>>> décoder("Unu lingvo por sxangxi la mondon kontraux la hxaoso")
'Unu lingvo por ŝanĝi la mondon kontraŭ la ĥaoso'
```

```
def décoder_lettre(ch):
    for i in range(len(traduction)):
        (avant, après) = traduction[i]
        if après == ch:
            return avant
    return ch

def décoder(chaîne):
    rés = ""
    for i in range(len(chaîne)):
        a = chaîne[i]
        if i+1<len(chaîne):
            b = chaîne[i+1]
            if b=="x":
                rés = rés + décoder_lettre(a+b)
            elif a!="x":
                rés = rés + a
        elif a!="x":
            rés = rés+a
    return rés
```