+1/1/60+

Bases de l'informatique 1 — SPUF100

Année 2025-2026 — Examen

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 méthodes autorisées
range(…) len(…) print(…)
L.append(…) c.lower(…) c.isalpha(…)

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

# Test de l'appartenance à une collection autorisé
x in L
```

```
# Sauf lorsque c'est précisé dans l'énonce, vous ne pouvez pas
# utiliser les compréhensions ou les slices
[ x for x in range(L) ]
chaine[début:fin:pas]

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

Malgré le sujet sur l'espéranto en novembre 2025, les différents chefs d'états continuent de faire et de déclarer des guerres. Afin d'encourager la paix dans le monde ainsi que votre niveau en programmation, ce sujet sera à nouveau dédié à l'espéranto, langue pacifiste cherchant à rapprocher les peuples.

Exercice 1 Utilitaires . . . . . 2 points

☐0 ☐0,5 ☐1 ☐1,5 ☐2

1. Écrire une fonction `formater(n, chaîne)` qui renvoie la chaîne obtenue en ajoutant des espaces à gauche jusqu'à obtenir une longueur `n`. Si la longueur de la chaîne initiale est plus grande que n, on renverra une chaîne identique à cette dernière.

```
>>> formater(10, "iu")
'        iu'
>>> formater(10, "iom")
'       iom'
```

```
>>> formater(10, "0123456789")
'0123456789'
>>> formater(10, "Aaaaaaaaaaah!!!")
'Aaaaaaaaaaah!!!'
```

```
def formater(n, chaîne):
    n0 = n - len(chaîne) # ou simplement
    espaces = ""         # return  (n - len(chaîne)) * " " + chaîne
    for i in range(n0):
        espaces = espaces + " "
    return espaces + chaîne
```

2. Écrire une fonction `raccourcir(chaîne, n)` qui renvoie la chaîne obtenue en supprimant les `n` derniers caractères. Si `n` est plus grand que la longueur de la chaîne, on renverra une chaîne vide.

```
>>> raccourcir("guillotiner",2)
'guillotin'
>>> raccourcir("python",4)
'py'
>>> raccourcir("python",146)
''
```

```
def raccourcir(chaîne, n):
    rés = ""
    for i in range(len(chaîne) - n):
        rés = rés+chaîne[i]
    return rés
```

Pour l'exercice suivant, on définit la fonction que vous pourrez utiliser :

```
def afficher(chaîne):
    print(formater(8, chaîne), end="")
```

Exercice 2 Grammaire de l'espéranto . . . . . 2,5 points

☐0 ☐0,5 ☐1 ☐1,5 ☐2 ☐2,5

```
liste_couples = [ ("présent", "as"), ("passé", "is"), ("futur", "os"),
                  ("impératif", "u"), ("conditionnel", "us")]
liste_pronoms =  [ "mi", "vi", "li/ŝi", "ni", "vi", "ili"]
```

1. On se donne les deux listes ci-dessus. Écrire la fonction `conjuger(verbe, temps)`, avec deux chaînes en paramètre, qui affiche la conjugaison du verbe au temps demandé. Il suffit d'afficher pour chaque pronom le verbe dans lequel le `i` final a été remplacé par la bonne terminaison (`"as"` pour le présent, etc.).

```
>>> conjuger("labori", "futur")
   mi laboros
   vi laboros
li/ŝi laboros
   ni laboros
   vi laboros
  ili laboros
```

```
>>> tableau() # pour la question 2
      iu     kiu     ĉiu   neniu     tiu
      io     kio     ĉio   nenio     tio
      ia     kia     ĉia   nenia     tia
      ie     kie     ĉie   nenie     tie
     ies    kies    ĉies  nenies    ties
     iel    kiel    ĉiel  neniel    tiel
     iam    kiam    ĉiam  neniam    tiam
     iom    kiom    ĉiom  neniom    tiom
```

```
def conjuger(verbe,temps):
    racine = raccourcir(verbe, 1)
    for couple in liste_couples:
        (ch,final) = couple
        if ch == temps:
            fin = final
            break # optionnel
    for p in liste_pronoms:
        print(formater(5,p) + " " + racine + fin)
```

2. En utilisant une double boucle `for` et en définissant deux listes, écrire la fonction `tableau()` qui affiche, selon l'exemple ci-dessus, la table des corrélatifs. Ce sont des petits mots constitués d'un préfixe et d'un suffixe. On pourra utiliser la fonction `afficher` de l'exercice 1 pour l'alignement des colonnes.

```
def tableau():
    for fin in ["u", "o", "a","e", "es", "el", "am", "om"]:
        for début in ["i", "ki", "ĉi", "neni", "ti"]:
            afficher(début+fin)
        print()
```

À ignorer sauf pour les curieux : le préfixe indique le rôle grammatical quand le suffixe en change le sens.
Préfixe : *where=**k**ie*, ***some**where=**i**e*, ***any**where=**ĉ**ie*, ***no**where=**neni**e here=**t**ie*) et
Suffixes *some**one**=i**u**/any**one**=ĉi**u***, *some**times**=i**am**/any**times**=ĉi**am***, *some**thing**=i**o**/any**things**=ĉi**o***.

Exercice 3 Découpage de texte . . . . . 4 points

☐0 ☐0,5 ☐1 ☐1,5 ☐2 ☐2,5 ☐3 ☐3,5 ☐4

L'objectif de cet exercice est d'écrire une fonction `split` améliorée pour découper une chaîne en une liste de mots. On pourra utiliser la méthode `lettre.isalpha()` pour tester si un symbole est une lettre et `lettre.lower()` pour mettre une lettre en minuscule.

1. On se donne un texte en espéranto constitué de lettres et de symboles divers (espace, chiffre, ponctuations, etc). Écrire une fonction `nettoyer(texte)` qui renvoie une nouvelle chaîne dans laquelle chaque lettre du texte a été remplacée par sa version minuscule et chaque autre symbole par un espace.

```
>>> nettoyer("C'était 12 petits ARC-EN-CIEL")
'c était  petits arc en ciel'
```

```
def nettoyer(texte):
    rés = ""
    for c in texte:
        if c.isalpha():
            rés = rés + c.lower()
        else:
            rés = rés + " "
    return rés
```

2. Écrire une fonction `indices_espaces(texte)` qui renvoie une liste contenant tous les indices de la chaîne `texte` correspondant à un espace.

```
>>> indice_espaces('c était  petits arc en ciel')
[1, 7, 8, 9, 10, 17, 21, 24]
```

```
def indice_espaces(ch):
    rés = []
    for i in range(len(ch)):
        if ch[i] in " ":
            rés.append(i)
    return rés
```

3. En déduire une fonction `découper` qui découpe un texte quelconque en une liste de mots, les mots étant les sous-chaînes **non vides** comprises entre deux espaces. On pourra exceptionnellement utiliser la syntaxe des *slices* : `chaîne[début:fin]`

```
>>> découper("  unu du tri    kvar kvin ses    sep ok naŭ  ")
['unu', 'du', 'tri', 'kvar', 'kvin', 'ses', 'sep', 'ok', 'naŭ']
>>> découper("C'était 12 petits ARC-EN-CIEL")
['c', 'était', 'petits', 'arc', 'en', 'ciel']
```

```
def découper(texte):
    ch = nettoyer(texte)
    L = indice_espaces(ch)
    début = 0
    rés = []
    for k in L:
        mot = ch[début:k] #
        if mot != "":
            rés.append(mot)
        début = k+1
    mot = ch[début:]
    if mot != "":
        rés.append(mot)
    return rés
```

On en profite pour définir deux listes de mots/suffixes qui seront utiles pour le prochain exercice.

```
suffixes = découper("as os is us u a an aj ajn o on ojn oj e en")

vortetoj = découper("""la al anstataŭ antaŭ apud ĉe ĉirkaŭ da de dum
    ekster el en ĝis inter je kontraŭ krom kun laŭ malgraŭ per po por
    post preter pri pro sen sub super sur tra trans mi ni vi ci li ŝi
    ĝi ili oni si kiu tiu iu ĉiu neniu kio tio io ĉio nenio kia tia ia
    ĉia nenia kies ties ies ĉies nenies ambaŭ unu du tri kvar kvin ses
    sep ok naŭ dek cent mil nul kaj aŭ sed plus minus nek ke ĉu se ĉar
    apenaŭ dum ĝis kvankam kvazaŭ ol kiel ol kie tie ie ĉie nenie ĉi
    for kiam tiam iam ĉiam neniam ankoraŭ baldaŭ hodiaŭ hieraŭ morgaŭ
    jam ĵus nun plu tuj kial tial ial ĉial nenial kiel tiel iel ĉiel
    neniel kiom tiom iom ĉiom neniom ajn almenaŭ ankaŭ apenaŭ des do
    eĉ ja jen jes ju kvazaŭ mem ne nur pli plej preskaŭ tamen tre tro
    adiaŭ bis fi ha he ho hura nu ve """)
```

Exercice 4 Analyse statistique des radicaux . . . . . 6 points

☐0 ☐0,5 ☐1 ☐1,5 ☐2 ☐2,5 ☐3 ☐3,5 ☐4 ☐4,5 ☐5 ☐5,5 ☐6

En espéranto il y a deux types de mots. Les premiers (la majorité) sont construits à partir d'un radical (par exemple *varm/*) et peuvent se décliner en nom avec une terminaison en `-o` (*varmo*=le chaud) en adjectif avec un `-a` (*varma*=chaud), en adverbe avec un `-e` (*varme*=chaudement) ou en verbe (*mi varmas*=j'ai chaud). La liste `suffixes` de toutes les terminaisons (en tenant compte du pluriel `-j` et de l'accusatif `-n`) est donnée page précédente.

Les autres mots ne sont pas décomposables et s'appellent des *vortetoj* (petits mots). Ce sont, les chiffres, les corrélatifs, les prépositions, les pronoms, etc. Leur liste complète est aussi donnée page précédente.

1. Écrire une fonction `est_suffixe(mot,fin)` qui renvoie `True` si la chaîne `fin` est un suffixe de `mot` et `False` sinon. On utilisera une boucle sans créer de nouvelles chaînes (pas de *slice* ni concaténation).

```
>>> est_suffixe("varma", "a")
True
>>> est_suffixe("vorteto", "ojn")
False
```

```
>>> est_suffixe("varmas", "a")
False
>>> est_suffixe("vortetojn", "ojn")
True
```

```
def est_suffixe(mot,fin):
    nf = len(fin)
    nm = len(mot)
    if nf > nm:
        return False
    for i in range(1, nf+1):
        if fin[nf-i] != mot[nm-i]:
            return False
    return True
```

2. En déduire une fonction d'une ligne `enlever_suffixe(mot,fin)` qui renvoie le radical obtenu en enlevant le suffixe `fin` et en le remplaçant par un `"/"`. On pourra utiliser la fonction `raccourcir` de l'exercice 1.

```
>>> enlever_suffixe("vortetoj", "oj")
'vortet/'
>>> enlever_suffixe("varma", "a")
'varm/'
```

```
def enlever_suffixe(mot,fin):
    return raccourcir(mot,len(fin)) + "/"
```

3. En déduire une fonction `radical(mot)` qui en utilisant la liste `suffixes` trouve la terminaison à supprimer et renvoie le radical du mot.

```
>>> radical("vortetoj")
'vortet/'
>>> radical("hejmen")
'hejm/'
```

```
def radical(mot):
    for fin in suffixes:
        if est_suffixe(mot,fin):
            return enlever_suffixe(mot,fin)
```

4. En déduire une fonction `radicaux(texte)` qui renvoie l'*ensemble* de tous les radicaux du texte. On ne tiendra pas compte des petits mots non décomposables de la liste `vortetoj` et on pourra utiliser la fonction `découper` de l'exercice 3.

```
>>> ch = "La bela instruisto varme parolas al la studentoj per belaj paroloj"
>>> radicaux(ch) # "la", "al" et "per" sont des vortetoj
{'instruist/', 'student/', 'bel/', 'parol/', 'varm/'}
```

```
def radicaux(texte):
    ensemble = set()
    liste = découper(texte)
    for mot in liste:
        if not (mot in vortetoj): # ou if mot not in vortetoj
            ensemble.add(radical(mot))
    return ensemble
```

5. Écrire une fonction `stat(texte)` qui renvoie le dictionnaire de tous les radicaux du texte avec leur nombre d'occurrences.

```
>>> stat(ch)
{'bel/': 2, 'instruist/': 1, 'varm/': 1, 'parol/': 2, 'student/': 1}
>>> stat("Li kantas tre belan kanton")
{'kant/': 2, 'bel/': 1}
```

```
def stat(texte):
    liste_mots = découper(texte)
    dico = dict()
    for mot in liste_mots:
        if not (mot in vortetoj):
            racine = radical(mot)
            if racine in dico:
                dico[racine] = dico[racine]+1
            else:
                dico[racine] = 1
    return dico
```

Exercice 5 Voyager en bicyclette . . . . . 5,5 points

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

Un espérantiste désire faire un grand voyage à travers l'Europe pour rencontrer d'autres membres de la communauté. Pour cela, il mesure des coordonnées cartésiennes sur une carte pour préparer son voyage et définit de manière globale un dictionnaire contenant pour chaque ville européenne ses coordonnées.

```
VILLES = { "Nice":(0, 0), "Monaco":(0.25, 0), "Milan":(3, 4), …}
```

1. Écrire une fonction `distance(v1,v2)` qui renvoie la distance euclidienne entre les deux villes (~~on rappelle que la Terre est plate~~ on négligera la courbure de la Terre, pour effectuer les calculs dans un repère cartésien plan selon la formule du cours).

```
>>> distance("Nice", "Milan")
5.0
>>> distance("Nice", "Monaco")
0.25
```

```
def distance(p,q):
    (x1, y1) = VILLES[p]
    (x2, y2) = VILLES[q]
    return ((x1-x2)**2 + (y1-y2)**2) **.5
```

2. Écrire une fonction `supprimer(ville, liste_villes)` qui renvoie une nouvelle liste correspondant au paramètre `listes_villes` à laquelle on a enlevé tous les éléments égaux à ville.

```
>>> villes = ["Nice",  "Lyon", "Milan", "Berlin", "Monaco", "Zagreb", "Nice"]
>>> supprimer("Nice", villes)
['Lyon', 'Milan', 'Berlin', 'Monaco', 'Zagreb']
```

```
def supprimer(ville, villes):
    rés = []
    for v in villes:
        if v != ville:
            rés.append(v)
    return rés
```

3. Écrire une fonction `plus_proche(départ, à_visiter)` qui parmi toutes les villes de la liste `à_visiter` renvoie celle qui est la plus proche de la ville de départ. On supposera `à_visiter` non vide et ne contenant pas la ville de départ.

```
>>> villes = ["Lyon", "Milan", "Berlin", "Monaco", "Zagreb"]
>>> plus_proche("Nice", villes)
'Monaco'
```

```
def plus_proche(départ, villes):
    proche = villes[0]
    for v in villes:
        if distance(départ, v) < distance(départ, proche):
            proche = v
    return proche
```

4. On se donne une liste de villes correspondant à un parcours. Écrire une fonction `longueur(parcours)` qui calcule la distance complète du parcours.

```
>>> distance("Nice", "Monaco") + distance("Monaco", "Milan")
5.1041219597369
>>> longueur_parcours(["Nice", "Monaco", "Milan"])
5.1041219597369
```

```
def longueur_parcours(parcours):
    lon = 0
    for i in range(1, len(parcours)):
        précédente = parcours[i-1]
        suivante  =  parcours[i]
        lon = lon+ distance(précédente, suivante)
    return lon
```

5. Notre espérantiste veut préparer un parcours en Europe en bicyclette pour visiter un maximum d'espérantistes. Pour cela il part de Nice et rejoint à chaque fois la ville la plus proche parmi toutes les villes à visiter. Il ne doit jamais passer deux fois par une même ville et décide de s'arrêter une fois la distance `objectif` dépassée. Écrire la fonction `construire_chemin(départ, à_visiter, objectif)` qui renvoie la liste des villes prévues pour le voyage.

```
>>> à_visiter = ["Berlin", "Helsinki", "Monaco", ...]
>>> trajet = construire_chemin("Nice", à_visiter, 50)
>>> trajet
['Nice', 'Monaco', 'Milan', 'Zurich', 'Munich', 'Vienne', 'Ljubljana', 'Zagreb']
>>> longueur(trajet)
51.3
```

```
def construire_chemin(départ, à_visiter, objectif):
    parcours = [ départ ]
    ici = départ
    while longueur(parcours) < objectif:
        suivante = plus_proche(ici,à_visiter)
        parcours.append(suivante)
        ici = suivante
        à_visiter = supprimer(suivante, à_visiter)
    return parcours
```