



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.

```
1 # Fonctions et méthodes autorisées
2 range(...) len(...) print(...)
3 L.append(...) c.lower(...) c.isalpha(...)
4
5 # Boucles autorisées
6 Boucles while, Boucles for avec ou sans range (for x in L)
7
8 # Test de l'appartenance à une collection autorisé
9 x in L
```

```
1 # Sauf lorsque c'est précisé dans l'énoncé, vous ne pouvez pas
2 # utiliser les compréhensions ou les slices
3 [ x for x in range(L) ]
4 chaine[début:fin:pas]
5
6 # Les méthodes et fonctions suivantes sont entre autres interdites
7 max(...) min(...) sum(...) abs(...)
8 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.

```
1 >>> formater(10, "iu")
2 '      iu'
3 >>> formater(10, "iom")
4 '     iom'
```

```
1 >>> formater(10, "0123456789")
2 '0123456789'
3 >>> formater(10, "Aaaaaaaaaah!!!")
4 'Aaaaaaaaaah!!!'
```

.....

.....

.....

.....

.....

.....

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.

```
1 >>> raccourcir("guillotiner",2)
2 'guillotin'
3 >>> raccourcir("python",4)
4 'py'
5 >>> raccourcir("python",146)
6 ''
```

.....

.....

.....

.....

.....

.....

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

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

**Exercice 2** Grammaire de l'espéranto 2,5 points☐ 0 ☐ 0,5 ☐ 1 ☐ 1,5 ☐ 2 ☐ 2,5

```
1 liste_couples = [ ("présent", "as"), ("passé", "is"), ("futur", "os"),  
2                  ("impératif", "u"), ("conditionnel", "us")]  
3 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.).

```
1 >>> conjuger("labori", "futur")  
2     mi laboros  
3     vi laboros  
4     li/ŝi laboros  
5     ni laboros  
6     vi laboros  
7     ili laboros
```

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

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.

À ignorer sauf pour les curieux : le préfixe indique le rôle grammatical quand le suffixe en change le sens.

Préfixe : *where*=**kie**, *somewhere*=**ie**, *anywhere*=**ĉie**, *nowhere*=**nenie** *here*=**tie**) et

Suffixes *someone*=**iu**/*anyone*=**ĉiu**, *sometimes*=**iam**/*anytimes*=**ĉiam**, *something*=**io**/*anything*=**ĉio**.



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.

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

.....

.....

.....

.....

.....

.....

.....

.....

.....

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

```
1 >>> indice_espaces('c était      petits arc en ciel')
2 [1, 7, 8, 9, 10, 17, 21, 24]
```

.....

.....

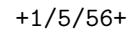
.....

.....

.....

.....

.....



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

This image shows a full page of white paper with horizontal dashed lines, typical of primary school writing paper. The lines are evenly spaced and run across the entire width of the page. There are no margins, text, or other markings present.

```
1 suffixes = découper("as os is us u a an aj ajn o on ojn oj e en")
2
3 vortetoj = découper("""la al anstataŭ antaŭ apud ĉe ĉirkaŭ da de dum
4 ekster el en ĝis inter je kontraŭ krom kun laŭ malgraŭ per po por
5 post preter pri pro sen sub super sur tra trans mi ni vi ci li ŝi
6 ĝi ili oni si kiu tiu iu ĉiu neniu kio tio io ĉio nenio kia tia ia
7 ĉia nenia kies ties ies ĉies nenies ambaŭ unu du tri kvar kvin ses
8 sep ok naŭ dek cent mil nul kaj aŭ sed plus minus nek ke ĉu se ĉar
9 apenaŭ dum ĝis kvankam kvazaŭ ol kiel ol kie tie ie ĉie nenie ĉi
10 for kiam tiam iam ĉiam neniam ankoraŭ baldaŭ hodiaŭ hieraŭ morgaŭ
11 jam ĵus nun plu tuj kial tial ial ĉial nenial kiel tiel iel ĉiel
12 neniel kiom tiom iom ĉiom neniom ajn almenaŭ ankaŭ apenaŭ des do
13 eĉ ja jen jes ju kvazaŭ mem ne nur pli plej preskaŭ tamen tre tro
14 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).

```
1 >>> est_suffixe("varma", "a")
2 True
3 >>> est_suffixe("vorteto", "ojn")
4 False
```

```
1 >>> est_suffixe("varmas", "a")
2 False
3 >>> est_suffixe("vortetojn", "ojn")
4 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.

```
1 >>> enlever_suffixe("vortetoj", "oj")
2 'vortet/'
3 >>> enlever_suffixe("varma", "a")
4 'varm/'
```

.....

.....

.....



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

```
1 >>> radical("vortetoj")
2 'vortet/'
3 >>> radical("hejmen")
4 'hejm/'
```

.....

.....

.....

.....

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.

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

.....

.....

.....

.....

.....

.....

.....



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

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

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

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.

```
1 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).

```
1 >>> distance("Nice", "Milan")
2 5.0
3 >>> distance("Nice", "Monaco")
4 0.25
```

.....

.....

.....

.....

.....



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`.

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

.....

.....

.....

.....

.....

.....

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.

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

.....

.....

.....

.....

.....

.....

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

```
1 >>> distance("Nice", "Monaco") + distance("Monaco", "Milan")
2 5.1041219597369
3 >>> longueur_parcours(["Nice", "Monaco", "Milan"])
4 5.1041219597369
```

.....

.....

.....

.....

.....

.....

[illegible]