



Bases de l'informatique 1 — SPUF100

Année 2024-2025 — Seconde session

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 autorisées
2 range(...) len(...) print(...)
3
4 # Boucles autorisées
5 Boucles while
6 Boucles for avec un range
7 Boucles for sans range (for x in L)
8
9 # Vous pouvez utiliser les compréhensions ou les slices
10 [ x for x in range(L) ]
11 chaine[début:fin:pas]
12
13 # Les méthodes suivantes sont autorisées :
14 L.append(...)
```

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



Exercice 1 Boucles while 4 points

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

1. Écrire une fonction **sr**(n) qui renvoie le plus petit nombre entier k vérifiant $n < \sqrt{1} + \sqrt{2} + \dots + \sqrt{k}$. On utilisera une boucle **while** et la fonction **sqrt** de la bibliothèque **math**. C'est à vous de faire l'import.

```
1 >>> sqrt(1) + sqrt(2) + sqrt(3) + sqrt(4) + sqrt(5)
2 8.382332347441762
3 >>> sqrt(1) + sqrt(2) + sqrt(3) + sqrt(4) + sqrt(5) + sqrt(6)
4 10.83182209022494
5 >>> sr(10)
6 6
```

.....

.....

.....

.....

.....

.....

.....

.....

.....

2. Écrire une fonction **intervalle**(a,b) qui à partir de deux entiers $a \leq b$ renvoie le tableau [a, ..., b]. Il est évidemment interdit d'utiliser la fonction **range** et il faudra créer le tableau de la bonne taille avant de le remplir (pas de **append** ou de **concatenation**).

```
1 >>> intervalle(5,10)
2 [5, 6, 7, 8, 9, 10]
3 >>> intervalle(2019,2025)
4 [2019, 2020, 2021, 2022, 2023, 2024, 2025]
```

.....

.....

.....

.....

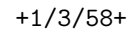
.....

.....

.....

.....

.....

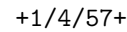


☐ 0 ☐ 0,5 ☐ 1 ☐ 1,5 ☐ 2 ☐ 2,5 ☐ 3 ☐ 3,5 ☐ 4 ☐ 4,5

- [illegible]

- [illegible]

- [illegible]

[illegible]

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

1. Une courbe est dite fonctionnelle si les différentes valeurs de x sont strictement croissantes. Écrire une fonction `est_fonctionnelle(courbe)` qui renvoie `True` si la courbe est fonctionnelle et `False` sinon.

[illegible]



2. Écrire une fonction **maximum**(courbe) qui renvoie le point de la courbe avec la plus grande ordonnée (si plusieurs points ont la même ordonnée, on renverra le point avec le plus petit abscisse).

```
1 >>> maximum([(2,3), (4,4), (7,-2), (8,0), (10, -33), (11,4)])
2 (4, 4)
```

.....

.....

.....

.....

.....

.....

.....

3. Écrire une fonction **nombre_positif**(courbe) qui renvoie le nombre de fois où l'ordonnée est positive (0 est considéré comme positif).

```
1 >>> nombre_positif([(2,3), (4,4), (7,-2), (8,0), (10, -33)])
2 3
```

.....

.....

.....

.....

.....

.....

.....

4. Écrire une fonction **flouter**(courbe) qui construit une liste de points à partir du paramètre courbe en sélectionnant un point sur deux.

```
1 >>> flouter([(-11,1), (-8,0), (0,3), (2,-3), (5,1), (11,12)])
2 [(-11, 1), (0, 3), (5, 1)]
```

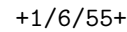
.....

.....

.....

.....

.....



☐ 0 ☐ 0,5 ☐ 1 ☐ 1,5 ☐ 2 ☐ 2,5 ☐ 3 ☐ 3,5 ☐ 4 ☐ 4,5 ☐ 5 ☐ 5,5 ☐ 6 ☐ 6,5 ☐ 7

```
1 >>> est_adn("GAT C")
2 False
3 >>> est_adn("Choucroute")
4 False
```

```
1 >>> complémentaire("AAGGTTCCAGTC")
2 'TTCCAAGGTCAG'
```



Un brin d'ADN est une séquence de nucléotides (A, C, T, G). Pour traduire l'ADN en protéines, on utilise le code génétique : à un codon (séquence de trois nucléotides) on associe une protéine. Chaque protéine sera représentée par une lettre. On se donne une liste traducteur qui donne les associations (codon, protéine) sous la forme d'un couple de chaînes de caractères. C'est le fameux code génétique.

```
1 >>> traducteur['CGT']
2 'R'
3 >>> traducteur['ATC']
4 'I'
```

```
1 >>> traducteur['TCT']
2 'S'
3 >>> traducteur['CCT']
4 'P'
```

Le dictionnaire traducteur est définie de manière globale. On pourra donc l'utiliser dans toutes les fonctions.

3. Écrire une fonction **traduire**(codon) qui prend en entrée une chaîne codon et qui renvoie la protéine associée dans le dictionnaire traducteur. Si la chaîne codon ne se trouve pas dans ce dictionnaire, la fonction renverra le caractère '_'.

```
1 >>> traduire("CGT")
2 'R'
3 >>> traduire("CFDT")
4 '_'
```

.....

.....

.....

.....

.....

4. On se donne une chaîne adn dont la longueur est un multiple de trois. Écrire la fonction **protéines**(adn) qui renvoie la chaîne des protéines associées à chacun des codons.

```
1 >>> protéines("TCTCCTATCAAAGAA") # TCT CCT ATC AAA GAA
2 'SPIKE'
```

.....

.....

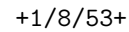
.....

.....

.....

.....

.....



Écrire une fonction `liste_protéines(adn)` qui prend une séquence ADN et renvoie la liste des gènes associés.

[illegible]