+1/1/60+

Programmation impérative en Python — SPUF21

Année 2023-2024 — 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.

```python
# Fonctions autorisées
range(…) len(…)
print(…) int(…)

# Méthodes et mots-clés autorisés
L.append(x)
x in L
```

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

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

Exercice 1 Écrire et tester son code . . . . . 3,5 points

☐0 ☐0,5 ☐1 ☐1,5 ☐2 ☐2,5 ☐3 ☐3,5

1. Écrire une fonction `est_strictement_positif(L)` qui renvoie `True` si tous les éléments de `L` sont strictement positifs. On supposera que `L` est une liste d'entiers et on renverra `True` lorsque `L` est vide.

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

2. Écrire (au moins) trois tests avec `assert` pour la fonction précédente. Vos tests devront être suffisamment diversifiés.

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

3. On cherche à afficher tous les multiples de 10 jusqu'à 50. La fonction suivante est-elle correcte ? Justifiez et en cas d'erreur, proposez une correction.

```
1 def étrange():
2     i=0
3     while i<=50:
4         if i%10 == 0:
5             print(i)
6         else:
7             i = i+1
```

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

Exercice 2 Un joli tapis ! . . . . . 2 points

☐ 0 ☐ 0,5 ☐ 1 ☐ 1,5 ☐ 2

1. Dans cet exercice, **interdiction** d'utiliser la fonction `print`. Vous devez utiliser les trois fonctions ci-dessous.

```
def dièse():
    print('#',end='')

def point():
    print('.', end="")
```

```
def nouvelle_ligne():
    print()
```

2. Écrire la fonction `tapis_écossais(largeur, hauteur)` qui affiche un magnifique tapis à carreau. Si vous n'y arrivez pas, vous pouvez choisir d'écrire à la place la fonction `tapis_incomplet` mais vous n'aurez que la moitié des points...

```
>>> tapis_écossais(15,6) # À faire
###############
#.#.#.#.#.#.#.#
###############
#.#.#.#.#.#.#.#
###############
#.#.#.#.#.#.#.#
>>> tapis_incomplet(10,4) # pour ceux qui ne réussissent pas tapis_écossais
#.#.#.#.#.
#.#.#.#.#.
#.#.#.#.#.
#.#.#.#.#.
```

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

Exercice 3 Des suites et des boucles ..... 4 points

☐0 ☐0,5 ☐1 ☐1,5 ☐2 ☐2,5 ☐3 ☐3,5 ☐4

1. Écrire de manière récursive la fonction `u(n)` correspondant à la suite récursive définie par :

$$\begin{cases} u_0 &= 1 \\ u_{n+1} &= 0,9 \times u_n + 2 \end{cases}$$

.....

.....

.....

.....

.....

.....

On se propose dans les trois questions suivantes de calculer la suite de manière itérative afin de trouver sa limite. On ne devra pas utiliser la fonction `u(n)` précédemment définie.

2. Écrire la fonction `suivant(x)` qui calcule et renvoie le prochain terme de la suite. Par exemple, si $x = u_3$, alors `suivant(x)` sera égal à u_4.

```
>>> suivant(1)  # 0,9 × 1 + 2
2.9
>>> suivant(10) # 0,9 × 10 + 2
11.0
```

.....

.....

.....

3. Écrire la fonction `absolue(x)` qui calcule et renvoie la valeur absolue du nombre x.

```
>>> absolue(3)
3
>>> absolue(-5)
5
```

.....

.....

.....

.....

.....

.....

.....

4. Écrire une fonction `limite(début, suivant, epsilon)` qui calcule et renvoie la limite de la suite définie par $u_0 =$ `début` et $u_{n+1} =$ `suivant`(u_n). On calculera les termes de la suite avec une boucle `while` et on s'arrêtera lorsque $|u_{n+1} - u_n| < \varepsilon$ (le symbole ε correspondant au paramètre `epsilon`) Dans ce cas on considèrera que u_n est la limite de la suite.

Exercice 4 Recherche de chaînes dans un texte ..... 5,5 points

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

1. Écrire une fonction `extraire_mot(texte, indice, n)` qui renvoie la sous-chaîne de la chaîne `texte` en commençant à l'indice donné en paramètre et de longueur `n`. On supposera l'indice valide (inutile de gérer le cas où l'indice n'est pas valide) et si la longueur dépasse du texte, on renverra la sous-chaîne commençant à l'indice et terminant avec le texte.

```
>>> extraire_mot("ABCDEFGHIJKLM", 3,5)
'DEFGH'
>>> extraire_mot("Salutation à toi", 2,3) # indice 2 et longueur 3
'lut'
>>> extraire_mot("bref", 1,500) # longueur trop grand
'ref'
```

2. Écrire une fonction `compatible(mot1,mot2)` qui renvoie `True` si les deux chaînes sont de même longueur et que leurs caractères sont comptatibles deux à deux. Deux caractères sont compatibles s'ils sont égaux ou si l'un des deux est égal à `'_'`

```
>>> compatible("abc", "a_c")
True
>>> compatible("a_cd", "abc")
False
```

```
>>> compatible("_be", "a_c")
False
>>> compatible("_b_d", "a__d")
True
```

3. Écrire une fonction `recherche(mot, texte)` qui renvoie la liste de toutes les sous-chaînes de texte compatibles avec `mot` ainsi que leurs indices. Le résultat sera donc une liste de couples de la forme `(indice, sous-chaîne)`.

```
>>> recherche("s_x_", "Alice doit désoxyder son sixième saxophone")
[(13, 'soxy'), (25, 'sixi'), (33, 'saxo')]
```

Exercice 5 Reconversion chez les traders . . . . . 5 points

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

Prenant conscience que les études à la fac ne sont pas faîtes pour vous et souhaitant vous enrichir rapidement, vous décidez de monter une start-up spécialisée dans la bourse des crypto-monnaies et dans l'IA.

Vous avez trouvé sur internet une liste de triplets de la forme `(date, prix, produit)` donnant l'historique en temps réel des cours de la bourse des crypto-monnaies. Par exemple, le triplet `("2024-03-01", 0.5, "#cc")` indique que le cours (le prix) de l'action de la cryptomonnaie `"#cc"`=coincoin (un successeur de bitcoin) est de 0,5€ le premier mars 2024.

```
historique = [("2023-10-01", 0.4, "#cc"), #  0,4€ en octobre
              ("2023-11-01", 0.5, "#cc"), #  0,5€ en novembre
              ("2023-12-01", 1,   "#cc"), #    1€ en décembre
              ("2024-02-01", 0.5, "#bc"), #
              ("2024-01-01", 0.3, "#cc"), #  0,3€ en janvier
              ("2024-02-01", 0.3, "#cc"), #  0,3€ en février
              ("2024-03-01", 0.5, "#bc"), #
              ("2024-03-01", 0.7, "#cc")] #  0.7€ en mars
```

1. Écrire une fonction `filtrer(historique, produit)` qui renvoie une liste composée de tous les éléments de la liste `historique` correspondant au produit donné en paramètre. On renverra une liste de tuples de la forme `(prix,date)`.

```
>>> filtrer(historique,"bc") # le bitcoin
[(0.5, '2024-02-01'), (0.5, '2024-03-01')]
>>> L = filtrer(historique,"cc") # le coincoin
>>> for x in L:
...     print(x)
(0.4, '2023-10-01')
(0.5, '2023-11-01')
(1, '2023-12-01')
(0.3, '2024-01-01')
(0.3, '2024-02-01')
(0.7, '2024-03-01')
```

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

2. Vous souhaitez seulement avoir la fin de liste. Écrire une fonction `tail(liste, n)` qui renvoie les `n` derniers élements de la liste `liste`.

```
>>> L = [1,2,3,4,5,6,7,8,9]
>>> tail(L,3)
[7, 8, 9]
>>> tail(L,0)
[]
>>> tail(L,1000)
[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Il est temps maintenant de créer une IA commentant le cours de la bourse et donnant des conseils pertinents d'investissement.

3. Écrire une fonction `journaliste_ia(produit,historique,n)` qui prend l'historique du cours des crypto-monnaies et affiche un commentaire pour les `n` dernières valeurs concernant le produit donné en paramètre. Il y aura trois types de message : un quand le prix augmente, un autre quand le prix diminue et un dernier quand le prix stagne. On respectera l'affichage proposé dans l'exemple.

```
historique = [("2023-10-01", 0.4, "#cc"), # 0,4€ en octobre
              ("2023-11-01", 0.5, "#cc"), # 0,5€ en novembre --> Ça monte
              ("2023-12-01", 1,   "#cc"), #   1€ en décembre --> Ça monte
              ("2024-02-01", 0.5, "#bc"), #
              ("2024-01-01", 0.3, "#cc"), # 0,3€ en janvier ---> Ça baisse
              ("2024-02-01", 0.3, "#cc"), # 0,3€ en février ---> Ça ne bouge pas
              ("2024-03-01", 0.5, "#bc"), #
              ("2024-03-01", 0.7, "#cc")] # 0.7€ en mars ------>Ça monte
```

```
>>> journaliste_ia("cc",historique, 4)
2023-12-01 Ça monte, il faut acheter !
2024-01-01 Ça chute, vendez tout !
2024-02-01 Ça ne bouge pas, investissez ailleurs
2024-03-01 Ça monte, il faut acheter !
```