2025A-420-SN1-RE Intra Solutionnaire

Nom : _____

Modalités

- Professeur : Vincent Archambault-Bouffard.
- Date : 2025-10-06.
- Durée : 3h00.
- Aucune documentation permise, sauf l'aide mémoire fournie.
- L'examen est sur 100 points. Il comporte 3 sections.
- Les questions sont indépendantes les unes des autres.

Section 1: Qu'affiche ce code ? (25 points)

Question 1.1 (6 points)

Écrivez ce que le code affiche. Ce code ne comporte pas d'erreur.

```
elements = ["H", "He", "Li", "Be"]
for i in range(len(elements)):
    if elements[i] == elements[2]:
        print(f"Trouvé: {elements[i]}")
        break
    print(f"Élément: {elements[i]}")
print("Fin")
```

Solution :

```
Élément: H
Élément: He
Trouvé: Li
Fin
```

Explication : La boucle parcourt les indices de 0 à 3. À l'indice 2, `elements[2]` vaut "Li", donc la condition est vraie, on affiche "Trouvé: Li" et le `break` sort de la boucle. Le "Fin" final s'affiche toujours.

Question 1.2 (6 points)

Écrivez ce que le code affiche. Ce code ne comporte pas d'erreur.

```
resultats = [2, 8, 3, 6, 4]
for i in resultats:
    print(f"Valeur: {i}")
    if i > 5:
        print("Trop grand")
        break
print(f"Index final: {i}")
```

Solution :

```
Valeur: 2
Valeur: 8
Trop grand
Index final: 8
```

Explication : La boucle commence avec i=2 (pas > 5), puis i=8 (> 5), donc on affiche "Trop grand" et on sort avec `break`. La variable `i` conserve la valeur 8 après la boucle.

Question 1.3 (6 points)

Écrivez ce que le code affiche. Ce code ne comporte pas d'erreur.

```
compteur = 10
while compteur > 0:
    if compteur % 3 == 0:
        print(f"Multiple de 3: {compteur}")
    compteur = compteur - 2
print(f"Fin: {compteur}")
```

Solution :

```
Multiple de 3: 6
Fin: 0
```

Explication : Le compteur commence à 10 et décrémente de 2 à chaque itération (10, 8, 6, 4, 2, 0). Seul 6 est un multiple de 3.

Question 1.4 (7 points)

Écrivez ce que le code affiche. Ce code ne comporte pas d'erreur.

```
limit = 5
i = 0
while i < limit:
    j = 0
    while j <= i:
        print("*" * (j + 1))
        j = j + 1
    print("-----")
    i = i + 1
print("Terminé")
```

Solution :

```
*
-----
*
**
-----
*
**
***
-----
*
**
***
****
-----
*
**
***
****
*****
-----
Terminé
```

Explication : Pour chaque valeur de i (0 à 4), on affiche une séquence d'étoiles croissante de 1 à i+1, suivie de "-----".

Section 2: Corrigez ce code (25 points)

Question 2.1 (8 points)

1. Encerclez la ou les erreur(s) dans le code suivant, c'est-à-dire là où le programme va planter.

```
s = "Bonjour"
for i in range(s):  # <-- ERREUR ICI
    print(s[i])
```

2. Corriger le code pour qu'il fonctionne comme prévu.

Solution :

L'erreur est `range(s)` car `range()` attend un nombre entier, pas une chaîne de caractères.

Code corrigé :

```
s = "Bonjour"
for i in range(len(s)):
    print(s[i])
```

Ou alternativement :

```
s = "Bonjour"
for i in s:
    print(i)
```

Question 2.2 (8 points)

Dans cette question, vous pouvez assumer que l'utilisateur entre un indice valide (0, 1, 2 ou 3).

1. Encerclez la ou les erreur(s) dans le code suivant, c'est-à-dire là où le programme va planter.

```
molecules = ["H2O", "CO2", "NH3", "CH4"]
choix = input("Entrez un indice (0-3) : ")
print(f"Molécule choisie : {molecules[choix]}")  # <-- ERREUR ICI
```

2. Corriger le code pour qu'il fonctionne comme prévu.

Solution :

L'erreur est que `choix` est une chaîne de caractères (retournée par `input()`), mais les indices de liste doivent être des entiers.

Code corrigé :

```
molecules = ["H2O", "CO2", "NH3", "CH4"]
choix = input("Entrez un indice (0-3) : ")
choix = int(choix)
print(f"Molécule choisie : {molecules[choix]}")
```

Ou en une ligne :

```
molecules = ["H2O", "CO2", "NH3", "CH4"]
choix = int(input("Entrez un indice (0-3) : "))
print(f"Molécule choisie : {molecules[choix]}")
```

Question 2.3 (9 points)

1. Encerclez la ou les erreur(s) dans le code suivant, c'est-à-dire là où le programme va planter.

```
vitesse = 105
if vitesse > 100 or < 60:  # <-- ERREUR 1 ICI
    print("Vitesse non acceptable !")
else  # <-- ERREUR 2 ICI
    print("Vitesse acceptable.")
```

2. Corriger le code pour qu'il fonctionne comme prévu.

Solution :

Deux erreurs :

1. `or < 60` est incomplet, il faut répéter la variable : `or vitesse < 60`
2. Il manque le `:` après `else`

Code corrigé :

```
vitesse = 105
if vitesse > 100 or vitesse < 60:
    print("Vitesse non acceptable !")
else:
    print("Vitesse acceptable.")
```

Section 3: Écrire un programme (50 points)

Question 3.1 : Suggestions de films (16 points)

Écrivez un programme qui demande à l'utilisateur des suggestions de films et les stocke dans une liste. Si l'utilisateur entre un film déjà suggéré, affichez le message "Film déjà suggéré !", ne l'ajoutez pas à la liste et redemandez une nouvelle suggestion. Si l'utilisateur entre "STOP", le programme se termine et affiche la liste finale des films suggérés si elle contient au moins trois films différents. Sinon, affichez "Pas assez de suggestions.".

Solution :

```
films = []

while True:
    film = input("Entrez un film (ou STOP pour terminer) : ")

    if film == "STOP":
        break

    if film in films:
        print("Film déjà suggéré !")
    else:
        films.append(film)

if len(films) >= 3:
    print("Liste finale des films suggérés :")
    for f in films:
        print(f"- {f}")
else:
    print("Pas assez de suggestions.")
```

Question 3.2 : Analyse de signaux (17 points)

Vous devez représenter des signaux périodiques à l'aide de Matplotlib. Les signaux suivent les équations suivantes :

- Signal 1 : f_1(t) = 3 * sin(2 * π * t)
- Signal 2 : f_2(t) = 2 * cos(2 * π * t + π / 4)

Sur un même graphique, affichez **quatre courbes** représentant :

- Le signal f_1(t)
- Le signal f_2(t)
- La somme des signaux : f_1(t) + f_2(t)
- La différence des signaux : f_1(t) - f_2(t)

Paramètres techniques :

- Générez 500 points de données pour que les courbes soient bien lisses
- Limitez l'affichage strictement entre 0 et 2 secondes
- Ajoutez un titre : "Analyse des signaux électromagnétiques"
- Étiquettes des axes : "Temps (s)" et "Amplitude (V)"
- Légende appropriée pour chaque courbe
- Grille visible
- Utilisez des couleurs différentes pour chaque courbe

Solution :

```
import matplotlib.pyplot as plt
import math

# Génération de 500 points entre 0 et 2 secondes
n = 500
t = []
for i in range(n):
    t.append(i * 2 / (n - 1))

# Calcul des signaux
f1 = []
f2 = []
somme = []
difference = []

for temps in t:
    valeur_f1 = 3 * math.sin(2 * math.pi * temps)
    valeur_f2 = 2 * math.cos(2 * math.pi * temps + math.pi / 4)

    f1.append(valeur_f1)
```

```
    f2.append(valeur_f2)
    somme.append(valeur_f1 + valeur_f2)
    difference.append(valeur_f1 - valeur_f2)

# Création du graphique
plt.plot(t, f1, label="f1(t) = 3*sin(2πt)", color='blue')
plt.plot(t, f2, label="f2(t) = 2*cos(2πt + π/4)", color='red')
plt.plot(t, somme, label="f1(t) + f2(t)", color='green')
plt.plot(t, difference, label="f1(t) - f2(t)", color='orange')

# Paramètres du graphique
plt.xlim(0, 2)
plt.title("Analyse des signaux électromagnétiques")
plt.xlabel("Temps (s)")
plt.ylabel("Amplitude (V)")
plt.legend()
plt.grid(True)

plt.show()
```

Question 3.3 : Répartition d'électrons (17 points)

Code de départ :

```
# Orbitales dans l'ordre de remplissage
orbitales = ["1s", "2s", "2p", "3s", "3p", "4s", "3d", "4p"]

# Capacités maximales
capacites = [2, 2, 6, 2, 6, 2, 10, 6]
```

Consignes :

1. Demander à l'utilisateur un nombre d'électrons à répartir (entre 1 et 36) et valider cette entrée.
2. Afficher ensuite la répartition des électrons dans les orbitales selon le principe de Aufbau, en respectant les capacités maximales de chaque orbitale. C'est-à-dire que vous devez remplir chaque orbitale jusqu'à sa capacité maximale avant de passer à la suivante. Pour chaque orbitale, affichez son nom suivi du nombre d'électrons qu'elle contient.

Exemples :

- Pour 8 électrons le code affiche : `1s2 2s2 2p4`
- Pour 11 électrons le code affiche : `1s2 2s2 2p6 3s1`

Solution :

```
# Orbitales dans l'ordre de remplissage
orbitales = ["1s", "2s", "2p", "3s", "3p", "4s", "3d", "4p"]

# Capacités maximales
capacites = [2, 2, 6, 2, 6, 2, 10, 6]

# Demander et valider le nombre d'électrons
while True:
    nb_electrons = int(input("Entrez le nombre d'électrons (1-36) : "))
    if 1 <= nb_electrons <= 36:
        break
    print("Nombre invalide. Entrez un nombre entre 1 et 36.")

# Répartition des électrons
configuration = ""
electrons_restants = nb_electrons

for i in range(len(orbitales)):
    if electrons_restants == 0:
        break
```

```
    # Nombre d'électrons à placer dans cette orbitale
    electrons_dans_orbitale = min(electrons_restants, capacites[i])

    # Ajouter à la configuration
    configuration += orbitales[i] + str(electrons_dans_orbitale) + " "

    # Réduire le nombre d'électrons restants
    electrons_restants -= electrons_dans_orbitale

# Afficher la configuration (sans l'espace final)
print(configuration.strip())
```

Solution alternative :

```
# Orbitales dans l'ordre de remplissage
orbitales = ["1s", "2s", "2p", "3s", "3p", "4s", "3d", "4p"]

# Capacités maximales
capacites = [2, 2, 6, 2, 6, 2, 10, 6]

# Demander et valider le nombre d'électrons
nb_electrons = int(input("Entrez le nombre d'électrons (1-36) : "))
while nb_electrons < 1 or nb_electrons > 36:
    nb_electrons = int(input("Nombre invalide. Entrez un nombre entre 1 et 36 : "))

# Répartition des électrons
configuration = []
electrons_restants = nb_electrons

for i in range(len(orbitales)):
    if electrons_restants == 0:
        break

    electrons_dans_orbitale = min(electrons_restants, capacites[i])
    configuration.append(orbitales[i] + str(electrons_dans_orbitale))
    electrons_restants -= electrons_dans_orbitale

print(" ".join(configuration))
```