Asie– 2024 – sujet2 - Correction

Exercice 2 (6 points)

---

1. Effectifs possibles

Expression : `'[2*(i+1)-3) for i in range(3, 10)]'`

Analyse des parenthèses :

- `[` → ouvrante
- `(` → ouvrante
- `(` → ouvrante
- `)` → fermante (correspond au `(i+1)`) ✓
- `)` → fermante (correspond au `(` après `*`) ✓
- `]` → fermante

Mais attention :

La parenthèse `)` après `-3` ferme une parenthèse `(` alors que la dernière parenthèse ouvrante non fermée est `[`.

Donc les types ne correspondent pas.

Conclusion : L'expression **n'est pas bien parenthésée**.

👉*Commentaire : Il ne suffit pas d'avoir le même nombre de parenthèses ouvrantes et fermantes, il faut aussi respecter l'ordre et le type.*

Partie A

2. Fonction `compte_ouvrante`

```
def compte_ouvrante(txt):
    compteur = 0
    for c in txt:
        if c in "([{":
            compteur += 1
    return compteur
```

👉*Commentaire : On teste si le caractère appartient aux parenthèses ouvrantes.*

---

3. Fonction `compte_fermante`

```
def compte_fermante(txt):
    compteur = 0
    for c in txt:
        if c in ")]}":
            compteur += 1
    return compteur
```

👉*Commentaire : Même principe mais avec les parenthèses fermantes.*

---

4. Fonction `bon_compte`

```
def bon_compte(txt):
    return compte_ouvrante(txt) == compte_fermante(txt)
```

👉*Commentaire : Cette fonction vérifie seulement l'égalité des nombres.*

---

5. Exemple où `bon_compte` renvoie True mais l'expression est fausse

Exemple :

`"(]"`

Il y a :

- 1 parenthèse ouvrante
- 1 parenthèse fermante

Donc `bon_compte("(]")` renvoie `True`.

Mais l'expression est incorrecte car `(` ne correspond pas à `]`.

👉*Commentaire : Le comptage seul est insuffisant : il ne vérifie ni l'ordre ni le type.*

Partie B

6. Compléter la classe `Pile`

Méthode `empiler`

```
def empiler(self, elt):
    self.contenu.append(elt)
```

Méthode `depiler`

```
def depiler(self):
    if self.est_vide():
        return "La pile est vide."
    return self.contenu.pop()
```

👉*Commentaire :*

- *`append()` ajoute en fin de liste (sommet de pile).*
- *`pop()` enlève et renvoie le dernier élément (LIFO).*

---

7. Nombre de comparaisons

Expression : `'tab[2*(i + 4)] - tab[3]'`

Longueur : 24 caractères.

Pour chaque caractère :

- test si parenthèse ouvrante
- test si parenthèse fermante

Donc environ 2 comparaisons par caractère.

Pour chaque parenthèse fermante :

- test de correspondance (comparaison supplémentaire)

Il y a 6 parenthèses.

Donc, comparaisons ≈ 2 × 24 + 6 = 48 + 6 = **54 comparaisons environ**

👉*Commentaire : On effectue un nombre de comparaisons proportionnel à la taille de la chaîne.*

Cas général (taille n)

Pour une chaîne de taille n :

- 2 tests par caractère → 2n
- Au maximum n/2 comparaisons supplémentaires

Donc nombre total ∈ O(n)

👉 **Conclusion : complexité linéaire O(n)**

👉 *Commentaire : Même si plusieurs comparaisons sont faites, elles restent proportionnelles à n.*

---

8. Fonction `est_bien_parenthesee`

```
def est_bien_parenthesee(txt):
    p = Pile()
    couples = {')': '(', ']': '[', '}': '{'}

    for c in txt:
        if c in "([{":
            p.empiler(c)
        elif c in ")]}":
            if p.est_vide():
                return False
            sommet = p.depiler()
            if sommet != couples[c]:
                return False

    return p.est_vide()
```

👉 *Commentaire :*

- *On empile les ouvrantes.*
- *Lorsqu'on rencontre une fermante :*
  - *on vérifie que la pile n'est pas vide,*
  - *on vérifie que les types correspondent.*
- *À la fin, la pile doit être vide.*

La pile permet de vérifier l'ordre d'imbrication (structure LIFO).

Résumé pédagogique

Méthode	Suffisant ?	Pourquoi
Comptage simple	❌	Ne vérifie pas l'ordre
Algorithme avec pile	✅	Vérifie ordre + type
