TEST DE PRÉREQUIS – PYTHON

Question 1 : Importation de bibliothèques

Quelles sont les façons correctes d'importer une bibliothèque ?

a) import turtle

b) import turtle as forward, backward

c) from turtle import forward, backward

d) import turtle as tortue

e) import turtle.forward, turtle.backward

Question 2 : Utilisation après un import spécifique

Après l'instruction suivante :

from turtle import forward as avance

Quelles expressions sont **correctes** ?

a) turtle.avance(10)

b) turtle.forward(10)

c) avance(10)

d) forward(10)

Question 3 : Expressions sans erreur

Quelles expressions **ne provoquent pas d'erreur** en Python ?

a) 10 + 20

b) 10 + '20'

c) '10' + '20'

d) 10 * 20

e) '10' * 20

f) '10' * '20'

Question 4 : Test sans erreur sur un tableau

Quelles expressions permettent de tester **sans risque d'erreur** que l'élément i du tableau t vaut x, sachant que i est un entier et t un tableau ?

a) t[i] == x and i >= 0 and i < len(t)
b) t[i] == x and (i >= 0 or i < len(t))
c) i >= 0 and i < len(t) and t[i] == x
d) i in range(len(t)) and t[i] == x

Question 5 : Affichage en ordre inverse

Quel programme affiche le contenu du tableau t **en ordre inverse** sans erreur ?

a)

```
for i in range(len(t), 0, -1):
   print(t[i])
```

b)

```
for i in range(1, len(t) + 1):
   print(t[len(t) - i])
```

c)

```
for i in range(len(t), 0, -1):
   print(t[i - 1])
```

d)

```
for i in range(len(t)):
   print(t[len(t) - i])
```

Question 6 : Préconditions d'une fonction

Quelles **préconditions** sont nécessaires pour éviter que la fonction suivante provoque une erreur ?

```
def f(p, n):
   for i in range(n):
      p.depiler()
   return p.sommet()
```

a) p doit être une pile

b) n doit être un entier

c) p ne doit pas être vide

d) n doit être un entier positif ou nul

e) p doit contenir au moins n éléments

f) p doit contenir au moins n + 1 éléments

📄 PAGE 2 : RÉPONSES ET EXPLICATIONS

✅ 1. Importation

Réponses correctes : a), c), d)

Explications :

- a) ✔ import turtle : importation correcte.
- b) ❌ Mauvaise syntaxe (as ne peut pas renommer plusieurs éléments).
- c) ✔ from turtle import forward, backward : valide, importe deux fonctions.
- d) ✔ import turtle as tortue : valide, on renomme le module.
- e) ❌ import turtle.forward est invalide — on ne peut pas importer une sous-fonction directement ainsi.

✅ 2. Utilisation après import

Réponse correcte : c)

Explications :

- from turtle import forward as avance crée **un alias local** avance.
- a) ❌ turtle.avance(10) : turtle n'a pas d'attribut avance.
- b) ❌ turtle.forward(10) : possible seulement si turtle a été importé, ce qui n'est pas le cas ici.
- c) ✔ avance(10) : c'est exactement l'alias importé.
- d) ❌ forward(10) : non défini (on l'a renommé en avance).

✅ 3. Expressions sans erreur

Réponses correctes : a), c), d), e)

Explications :

- a) ✔ addition entre deux entiers.
- b) ❌ 10 + '20' : erreur de type (int + str).
- c) ✔ concaténation entre deux chaînes.
- d) ✔ multiplication entre deux entiers.
- e) ✔ '10' * 20 : répète la chaîne '10' 20 fois.

- f) ❌ '10' * '20' : multiplication entre deux chaînes → erreur.

✅ 4. Test sans risque d'erreur

Réponses correctes : c), d)

Explications :

- a) ❌ risque d'erreur : t[i] est évalué avant la vérification de i >= 0.
- b) ❌ mauvaise condition : le or permet des indices hors bornes.
- c) ✔ ordre sûr : on vérifie d'abord que i est valide, puis on accède à t[i].
- d) ✔ i in range(len(t)) garantit que i est valide avant d'accéder à t[i].

✅ 5. Affichage en ordre inverse

Réponses correctes : b), c)

Explications :

- a) ❌ t[i] est hors limites, car range(len(t), 0, -1) commence à len(t) (indice invalide).
- b) ✔ len(t) - i donne les indices inversés valides.
- c) ✔ t[i-1] fonctionne avec la boucle allant de len(t) à 1.
- d) ❌ t[len(t)-i] produit un indice hors bornes quand i = 0.

✅ 6. Préconditions

Réponses correctes : a), b), d), f)

Explications :

- a) ✔ p doit être une **pile** (avoir depiler() et sommet()).
- b) ✔ n doit être un **entier**.
- c) ❌ peut être vide au début si n = 0.
- d) ✔ n doit être un entier **positif ou nul**.
- e) ❌ insuffisant : après dépilement de n éléments, on accède encore à p.sommet(), donc il faut **au moins n+1 éléments**.
- f) ✔ condition correcte : p doit contenir au moins n + 1 éléments pour que sommet() soit valide après les dépilements.