QCM – Chapitre 5 : Fonctions et Lambdas

Question 1

Quelle est la bonne syntaxe pour définir une fonction en Python ?

A. `fonction nom(param1, param2):`
B. `define nom(param1, param2):`
C. `def nom(param1, param2):`
D. `function nom(param1, param2):`

Réponse attendue : C
Explication : Le mot-clé `def` est utilisé pour définir une fonction en Python.

Question 2

Que renvoie cette fonction ?

```
def f(x):
    x += 1
print(f(3))
```

A. 4 B. 3 C. None D. Erreur

Réponse attendue : C **Explication :** La fonction ne contient pas de `return`, elle affiche rien et retourne `None`.

Question 3

Quelle est la différence entre une variable locale et une variable globale ?

A. Aucune, elles sont identiques B. Une variable locale existe seulement dans la fonction où elle est définie C. Une variable globale est détruite à la fin de la fonction D. Une variable locale remplace toujours la variable globale

---

Réponse attendue : B

Question 4

Quel sera le résultat du code suivant ?

```
x = 5
def test():
    x = 10
    print(x)
test()
print(x)
```

A. `10` puis `10` B. `10` puis `5` C. `5` puis `10` D. `Erreur`

Réponse attendue : B **Explication :** `x = 10` est local à la fonction.

Question 5

Quelle est la bonne définition d'une fonction *lambda* ?

A. Une fonction sans arguments B. Une fonction anonyme (sans nom explicite) C. Une fonction récursive D. Une fonction définie dans une autre fonction

---

Réponse attendue : B

Question 6

Quelle est la sortie du code suivant ?

```
add = lambda a, b: a + b
print(add(2, 3))
```

A. `lambda(2,3)` B. `23` C. `5` D. `Erreur`

Réponse attendue : C

Question 7

Quelle différence entre une fonction `lambda` et une fonction définie avec `def` ?

A. Une lambda ne peut contenir qu'une seule expression B. Une lambda ne peut pas être assignée à une variable C. Une lambda est plus rapide à exécuter D. Elles sont identiques

---

Réponse attendue : A

Question 8

Quel est le résultat de ce code ?

```
numbers = [1, 2, 3, 4]
evens = list(filter(lambda x: x % 2 == 0, numbers))
print(evens)
```

A. `[1, 3]` B. `[2, 4]` C. `[1, 2, 3, 4]` D. `None`

Réponse attendue : B

Question 9

Quel est le rôle de la fonction `map()` ?

A. Appliquer une fonction sur chaque élément d'un itérable B. Filtrer les éléments d'un itérable C. Trier une liste D. Créer une copie d'une liste

---

Réponse attendue : A

Question 10

Quelle sortie produit ce code ?

```
names = ["alice", "bob"]
result = [(lambda n: n.upper())(name) for name in names]
print(result)
```

A. `['ALICE', 'BOB']` B. `['Alice', 'Bob']` C. `[None, None]` D. `Erreur`

Réponse attendue : A **Explication :** La lambda applique `upper()` sur chaque nom.

Question 11

Quelle instruction crée une *closure* valide ?

A. Définir une fonction dans une autre fonction B. Appeler une fonction sans arguments C. Déclarer une variable globale D. Utiliser `lambda` dans une boucle

---

Réponse attendue : A

Question 12

Que fera ce code ?

```
def make_counter():
    count = [0]
    def increment():
        count[0] += 1
        return count[0]
    return increment

c = make_counter()
print(c(), c())
```

A. `1 1` B. `1 2` C. `2 2` D. `Erreur`

Réponse attendue : B **Explication :** La liste `count` conserve l'état entre deux appels (closure).

Question 13

Que renvoie `sum(map(lambda x: x**2, [1, 2, 3]))` ?

A. `[1, 4, 9]` B. `14` C. `(1, 4, 9)` D. `Erreur`

Réponse attendue : B **Explication :** $1^2 + 2^2 + 3^2 = 14$.

Question 14

Quelle syntaxe permet d'utiliser `lambda` dans une compréhension de liste ?

A. `[lambda x: x+1 for x in range(3)]` B. `[(lambda x: x+1)(x) for x in range(3)]` C. `[x+1 for lambda x in range(3)]` D. `[(x) lambda x+1 for x in range(3)]`

Réponse attendue : B

Question 15

Que renverra ce code ?

```
pairs = list(map(lambda x: (x, x**2), range(3)))
print(pairs)
```

A. `[1, 4, 9]` B. `[(0, 0), (1, 1), (2, 4)]` C. `(0, 1, 2)` D. `Erreur`

Réponse attendue : B