Corrigé du QCM n°1

1. Sous python, on peut :

   1. utiliser la console pour saisir des instructions exécutées dans la foulée
   2. utiliser la console pour écrire des scripts (fichier '*.py') élaborés
   3. travailler sans console
   4. utiliser l'éditeur pour enregistrer un script puis l'exécuter

La console permet la saisie et l'exécution des instructions tapées. Si l'on souhaite écrire un code élaboré, la console est inadaptée, il faut le faire dans l'éditeur qui permet d'enregistrer son script en vue de l'exécuter dans une console. Celle-ci est donc indispensable, même si on travaille dans l'éditeur.

2. L'instruction `2.0+3` renvoie :

   1. `5`
   2. une erreur
   3. `[5]`
   4. `5.0`

Le nombre `2.0` est flottant et le type flottant est absorbant d'où la réponse `5.0`.

3. Soit `a` une variable contenant un entier. L'instruction `a%2==0` renvoie :

   1. une erreur
   2. `True` si `a` est pair
   3. le quotient de `a` par 2
   4. `True` si `a` est impair

L'opération `a%2` renvoie le reste de la division euclidienne de `a` par 2 et le résultat du test est donc `True` si `a` est pair et `False` sinon.

4. L'instruction `len(str(2**2026))` renvoie :

   1. `"2**2026"`
   2. le nombre de chiffres dans l'écriture décimale de 2^{2026}
   3. une erreur
   4. 2026

On convertit le nombre 2^{2026} en chaîne et on renvoie la taille de cette chaîne ce qui correspond donc au nombre de chiffres dans son écriture décimale.

5. On exécute le script suivant :

```
a="bonjour"
b=a[::-1]
print(b)
```

L'exécution du script :

1. renvoie une erreur
2. renvoie `"ruojnob"`
3. affiche `ruojnob`
4. affiche `"bonjour"`

La variable `b` reçoit la chaîne renversée de `"bonjour"`, c'est-à-dire `"ruojnob"` et celle-ci est affichée par l'instruction `print`.

6. Dans la console on saisit `>>> a="bonjour"`. On constate que :

1. `a[3:]="soir"` provoque une erreur
2. `a[:3]+"soir"` renvoie `"bonsoir"`
3. `a[3:]="soir"` modifie `a` en `"bonsoir"`
4. `"soir" in a` provoque une erreur

Le test `"soir" in a` renvoie `False`. Puis, le slicing `a[:3]` renvoie les caractères indicés de 0 à 2, c'est-à-dire la chaîne `"bon"`. En concaténant, l'instruction `a[:3]+"soir"` renvoie `"bonsoir"` . En revanche, une chaîne n'est pas mutable et tenter de modifier `a` provoque une erreur.

7. Soit `a` une variable contenant un nombre à deux chiffres. Pour tester l'égalité des deux chiffres, on peut utiliser :

1. `a%11==0`
2. `a[0]==a[1]`
3. `a//10==a%10`
4. `str(a)==a[::-1]`

Si la variable `a` contient un nombre à deux chiffres égaux, alors celui-ci est divisible par 11 ou, de manière équivalente, son chiffre des dizaines donné par `a%10` est égal à son chiffre des centaines donné par `a//10`. Un entier est un type simple et l'indexation ou le slicing sur `a` provoque un message d'erreur.

8. Après l'instruction `a=list(range(10))`, on a :

1. la variable `a` contient `[1,2,...,10]`
2. la variable `a` contient `[0,1,...10]`
3. rien car l'instruction provoque une erreur
4. la variable `a` contient `[0,1,...,9]`

On construit la séquence de 0 à 10 exclu par pas de 1 et on la convertit en liste.

9. On exécute le script suivant :

```
a=[1,2,"bonjour"]
b=a.pop()
a.append(0)
```

Après l'exécution du script :

1. a contient [1,2,0] et b contient 1
2. a contient [2,"bonjour",0] et b contient 1
3. a contient [0,1,2] et b contient "bonjour"
4. a contient [1,2,0] et b contient "bonjour"

L'instruction b=a.pop() sort le dernier élément de a c'est-à-dire la chaîne "bonjour" et la stocke dans b. Enfin, on ajoute 0 en fin de liste dans a.

10. La saisie dans la console de >>> a=[[]]*3;a[0].append(1);a :

    1. renvoie [[],[],[],1]
    2. renvoie [[1],[],[]]
    3. renvoie [[1],[1],[1]]
    4. ne renvoie rien

La construction de a fait que les trois sous-listes de a sont la duplication d'une même liste vide : on a donc créé trois étiquettes sur la même liste. L'instruction append modifie cette sous-liste et les trois étiquettes posées dessus renvoient le même contenu.