Proposition de corrigé sur les bogues

Explications sur la nature des bogues et les corrections éventuelles

bogue00.py

Les types ne sont pas vérifiés dans doubler() et dans la saisie.
Comme l'opérateur * fonctionne aussi bien avec des string qu'avec des int, il ne retourne aucune erreur mais un résultat erroné.

```
>>> "3" * 2
'33'
>>> 3 * 2
6
```

On peut corriger le programme ainsi :

```
def doubler(x):
    return x * 2

def tripler(y):
    return y * 3

nombre = int(input("Saisir un nombre : "))  # Ajout de int() pour avoir un int
print( tripler(doubler(nombre)) )
```

bogue01.py

L'erreur vient d'un mauvais choix d'opérateur.
Ici on utilise = au lieu de += pour faire la concaténation

```
nouveau_mot = mot[i] + "_"
```

Il suffit donc de modifier cette ligne ainsi :

```
nouveau_mot += mot[i] + "_"
```

bogue02.py

Cette ligne est fautive, car le résultat de la suppression de la voyelle ne fonctionnera que pour la dernière voyelle du mot.

```
nouveau_mot = mot.replace(lettre, "")
```

Pour pouvoir fonctionner sur toutes les voyelles, il faut que le résultat du remplacement soit réaffecté à nouveau pour la prochaine voyelle

```
    nouveau_mot = mot
    les_voyelles = ('a', 'e', 'i', 'o', 'u')
    for lettre in mot.lower():
        if lettre in les_voyelles:
            nouveau_mot = nouveau_mot.replace(lettre, "")
    return nouveau_mot
```

bogue03.py

Le problème vient ici de la méthode cloner().

```
def cloner(notes):
    """Clone un tableau de notes passé en paramètre"""
    nouvelles_notes = notes  # notes et nouvelles_notes sont en fait le même tablea
    return nouvelles_notes
```

En écrivant...

```
nouvelles_notes = notes
```

...on pense copier le tableau note, mais en fait on ne copie que sa référence, ce qui fait que les 2 tableaux sont en fait le même.

Pour effectuer une vraie copie de tableau, il faut utiliser le module copy

```
import copy
nouvelles_notes = copy.deepcopy(notes)
```

bogue04.py

Le calcul ne s'effectue pas correctement car l'opérateur utilisé est celui de la comparaison == plutôt que celui de l'affectation =

```
total == total + farine     # Mauvaise version
total = total + farine      # Bonne version
```

La fonction malaxer() devra être réécrite comme ci-dessous :

```
def malaxer(farine, oeufs, chocolat):
    """Malaxe les 3 ingrédients dans certaines quantités exprimées en grammes"""
    # Poids total de la pate
    total = 0

    # Ajout et malaxage de tous les ingrédients
    total == total + farine
    total == total + oeufs
    total == total + chocolat

    # On retourne le poids total de la pate
    return total
```

bogue05.py

Ici, il s'agit juste d'un problème d'indentation

```
if age < 18:
    pouvoir_voter = False
else:
    pouvoir_voter = True
pouvoir_parier = True   # Mal indentée
```

On pourra le corriger ainsi

```
if age < 18:
    pouvoir_voter = False
else:
    pouvoir_voter = True
    pouvoir_parier = True   # bien indentée
```

bogue06.py

Il faut déclarer la variable poids en global dans la fonction modifier_poids() avec : global poids

```
def modifier_poids(nouveau_poids):
    global poids
    poids = nouveau_poids
    print("Mon nouveau poids est de %d kg" % (poids))
```

Dans une fonction, si il y a une affectation de variable comme dans modifier_poids(), alors Python créé une nouvelle variable locale
Par contre dans la fonction afficher_poids(), comme il n'y a pas d'affectation, Python utilise la variable globale (sans besoin de préciser global poids)

https://stackoverflow.com/questions/929777/why-does-assigning-to-my-global-variables-not-work-in-python

bogue07.py

La condition du while != est trop restrictive et la limite jamais atteinte => on obtient une boucle infinie
Mieux vaut utiliser l'opérateur < comme ci-dessous :

```
while nombre < limite:
    print(f"{nombre} ", end="")
    nombre += 2
```

bogue08.py

L'affichage arrondi à 2 digits après la virgule ci-dessous cache l'erreur de calcul

```
print(f"J'accélère : acceleration_fusee = {acceleration_fusee:.2f}")
```

En le modifiant ainsi, on peut voir le vrai calcul effectué et pourquoi les if de la fonction verifier() passent à côté

```
print(f"J'accélère : acceleration_fusee = {acceleration_fusee}")
```

En fait il faut éviter l'opérateur == avec les float à cause des erreurs d'arrondis dans les calculs
Mieux vaudrait écrire la fonction verifier() ainsi :

```
def verifier_et_conseiller(acceleration):
    delta = 0.001
    if acceleration > (1.0 - delta) and acceleration < (1.0 + delta):
        print("\033[30;32mOK, accélération de 1G\033[00m")
    if acceleration > (2.0 - delta) and acceleration < (2.0 + delta):
        print("\033[30;33mAttention, accélération de 2G \033[00m")
    if acceleration > (3.0 - delta) and acceleration < (3.0 + delta):
        print("\033[30;31mUrgence, accélération de 3G, veuillez freiner\033[00m")
```