Calculatrice scientifique en utilisant l'abstraction, l'héritage et le polymorphisme

Objectif :

Créer une calculatrice scientifique avec les fonctionnalités suivantes :

- Une classe abstraite **Calculatrice** qui définit les méthodes de calcul de base.
- Une classe **CalculatriceScientifique** qui étend Calculatrice et ajoute des fonctions scientifiques comme le sinus, le cosinus, la tangente, etc.
- Une interface graphique avec **Tkinter** pour interagir avec l'utilisateur.

Structure :

1. **Classe Abstraite Calculatrice** :
   - Une méthode abstraite calculer() qui sera redéfinie dans les classes dérivées.
   - Une méthode afficher_resultat() pour afficher le résultat sur l'interface.
2. **Classe CalculatriceScientifique** :
   - Surcharge la méthode calculer() pour inclure des calculs scientifiques.
   - Ajoute des méthodes pour des calculs comme sin(), cos(), tan(), etc.
3. **Interface graphique avec Tkinter** :
   - Des boutons pour les opérations basiques et scientifiques.
   - Un champ d'entrée pour l'affichage des opérations.

Exercice :

Étape 1 : Créer la classe Calculatrice abstraite

```
from abc import ABC, abstractmethod
import math

class Calculatrice(ABC):
    def __init__(self):
        self.operande = ""
        self.resultat = 0

    @abstractmethod
    def calculer(self):
        pass

    def afficher_resultat(self):
        return self.resultat
```

Étape 2 : Créer la classe CalculatriceScientifique

```
class CalculatriceScientifique(Calculatrice):
    def __init__(self):
```

```
        super().__init__()

    def calculer(self, expression):
        try:
            self.operande = expression
            # Ici on peut ajouter un calcul simple ou scientifique
            if "sin" in expression:
                self.resultat = math.sin(math.radians(float(expression[3:])))
            elif "cos" in expression:
                self.resultat = math.cos(math.radians(float(expression[3:])))
            elif "tan" in expression:
                self.resultat = math.tan(math.radians(float(expression[3:])))
            else:
                self.resultat = eval(expression)  # Utilisation de eval pour les autres calculs
        except Exception as e:
            self.resultat = "Erreur"
```

Étape 3 : Créer l'interface avec Tkinter

```
import tkinter as tk

class CalculatriceApp(tk.Tk):
    def __init__(self):
        super().__init__()
        self.title("Calculatrice Scientifique")
        self.geometry("400x600")

        self.calculatrice = CalculatriceScientifique()

        # Interface graphique
        self.creer_widgets()

    def creer_widgets(self):
        # Champ de texte pour afficher les opérations
        self.texte = tk.Entry(self, width=20, font=("Arial", 24), borderwidth=2, relief="solid")
        self.texte.grid(row=0, column=0, columnspan=4)

        # Boutons de la calculatrice
        boutons = [
            ('7', 1, 0), ('8', 1, 1), ('9', 1, 2), ('/', 1, 3),
            ('4', 2, 0), ('5', 2, 1), ('6', 2, 2), ('*', 2, 3),
            ('1', 3, 0), ('2', 3, 1), ('3', 3, 2), ('-', 3, 3),
            ('0', 4, 0), ('C', 4, 1), ('=', 4, 2), ('+', 4, 3),
            ('sin', 5, 0), ('cos', 5, 1), ('tan', 5, 2), ('sqrt', 5, 3),
        ]

        for (text, row, col) in boutons:
            bouton = tk.Button(self, text=text, width=5, height=2, command=lambda t=text:
self.on_button_click(t))
            bouton.grid(row=row, column=col)
```

```
    def on_button_click(self, texte):
        if texte == "=":
            expression = self.texte.get()
            self.calculatrice.calculer(expression)
            self.texte.delete(0, tk.END)
            self.texte.insert(tk.END, self.calculatrice.afficher_resultat())
        elif texte == "C":
            self.texte.delete(0, tk.END)
        else:
            current = self.texte.get()
            self.texte.delete(0, tk.END)
            self.texte.insert(tk.END, current + texte)

app = CalculatriceApp()
app.mainloop()
```

Explication de l'exercice :

1. **Héritage** :
    - CalculatriceScientifique hérite de Calculatrice et surcharge la méthode calculer() pour effectuer des calculs scientifiques spécifiques.
2. **Abstraction** :
    - La classe Calculatrice est une classe abstraite qui définit l'interface de calcul (méthode calculer()). Les classes dérivées comme CalculatriceScientifique doivent implémenter cette méthode.
3. **Surcharge de Méthode** :
    - La méthode calculer() dans CalculatriceScientifique surcharge la méthode de la classe parente pour inclure des calculs scientifiques comme le sinus, cosinus, tangente, etc.
4. **Interface Graphique avec Tkinter** :
    - L'application permet à l'utilisateur de saisir une expression dans un champ de texte, d'appuyer sur un bouton pour effectuer des calculs basiques ou scientifiques, et d'afficher le résultat à l'écran.
5. **Fonctionnalités supplémentaires** :
    - La calculatrice peut effectuer des calculs de base (+, -, *, /) ainsi que des calculs scientifiques (sin, cos, tan, sqrt).

Exercice pour aller plus loin :

- Ajouter des fonctionnalités comme le calcul de logarithmes (log), de puissances (^), ou d'autres fonctions trigonométriques.
- Ajouter des validations pour gérer les erreurs, par exemple, lorsque l'utilisateur entre une expression invalide.
- Créer d'autres classes comme CalculatriceBasique pour les calculs simples, en héritant de Calculatrice.

Cela vous permettra d'améliorer vos compétences en abstraction, héritage et surcharge de méthodes tout en travaillant avec Tkinter pour créer des interfaces graphiques.