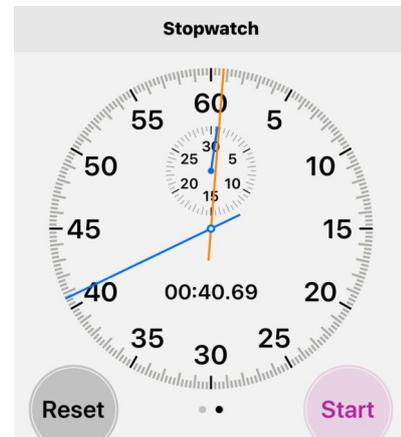

TP - CHRONOMÈTRE

1 Contexte

On souhaite réaliser un chronomètre de précision en Python capable d'afficher une mesure de durée selon le format suivant :

mm : ss . cc

- **mm** représentant les chiffres des minutes,
- **ss** les chiffres des secondes,
- **cc** les chiffres des centièmes de secondes.



Pour cela, nous allons utiliser la programmation objet qui permet de bien séparer des différentes fonctionnalités que devra implémenter le programme : c'est le **principe de séparation des responsabilités**.

- Une classe **Chronometre** se chargera d'effectuer les mesures de durée.
- Une classe **Afficheur** se chargera de produire une fenêtre graphique intégrant un affichage ainsi que des boutons de contrôle.

2 Construction de la classe Chronometre

2.1 La fonction `time.time`

Pour mesurer des durées, on utilisera la fonction `time`¹ du module python `time`, en plaçant en début de programme la directive d'import suivante :

```
from time import time
```

Après avoir lu la documentation officiel de cette fonction, écris un programme capable de mesurer la durée d'exécution d'une répétition de 10 000 000 d'additions.

2.2 Implémentation de la classe **Chronometre**

On donne ci-dessous l'interface de la classe, c'est à dire un ensemble de méthodes que tu devras implémenter dans la classe **Chronomètre** et qui permettra à un utilisateur de la classe **Chronometre** de ne pas manipuler directement ses attributs.

- `def demarrer() -> None` : déclanche la mesure de durée
- `def interrompre() -> None` : interrompt la mesure de durée
- `def remettre_a_zero() -> None` : réinitialise la mesure de durée à zéro
- `def mesure_en_cours() -> bool` : renvoie `True` si une mesure est en cours, `False` sinon.
- `def duree() -> float` : renvoie la durée en seconde mesurée au moment de l'appel, en prenant en compte les éventuelles interruptions ayant eu lieu.

1. Sur ton carnet de bord, analyse le fonctionnement du chronomètre de ton téléphone. Détermine alors les attributs dont la classe **Chronometre** a besoin puis ce que doit effectuer chacune des méthodes de l'interface sur ces attributs.
2. Définis la classe **Chronometre** et implémentes-y chacune de ces méthodes. Tu pourras tester le bon fonctionnement de ta classe en faisant la même mesure que dans la partie 2.1.

1. Consulte la documentation officielle de Python : <https://docs.python.org/3/library/time.html#time.time>

- Ajoute une méthode `__str__` qui permettra d'afficher la durée mesurée au format `mm:ss.cc`, par exemple lorsqu'un objet `chronometre` est passé à la fonction `print`.
Exemple de formatage d'un entier : `'{:02d}'.format(3)` renvoie "03".
- Teste ton implémentation en mesurant la durée d'exécution d'instructions.

3 Construction d'une interface graphique

3.1 Prise en main de Tkinter

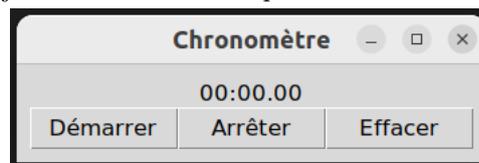
Pour réaliser l'interface graphique de notre programme, on utilisera la bibliothèque `Tkinter`. On trouve dans sa documentation officiel le programme de démonstration suivant :

```

1 from tkinter import *
2 from tkinter import ttk
3 root = Tk()
4 frm = ttk.Frame(root, padding=10)
5 frm.grid()
6 ttk.Label(frm, text="Hello World!").grid(column=0, row=0)
7 ttk.Button(frm, text="Quit", command=root.destroy).grid(column=1, row=0)
8 root.mainloop()

```

- Crée une classe `Afficheur` et ajoutes-y toutes les variables de l'exemple comme attributs, en les initialisant dans le constructeur de la même manière que dans l'exemple.
- Ajoute une méthode `afficher` dont la fonction sera d'afficher l'interface graphique en appelant la méthode `mainloop` sur l'attribut de type `Tk`.
- Modifier l'interface graphique en ajoutant deux boutons pour obtenir le rendu suivant :



On pourra passer comme argument `columnspan=3` à la méthode `grid` pour étendre l'affichage du texte à trois colonnes.

3.2 Implémentation de l'interface de la classe `Afficheur`

La dernière étape dans la construction du programme consiste à intégrer le chronomètre à l'interface graphique que tu viens de créer. Il faut pour cela :

- ajouter à la classe `Afficheur` un attribut de type `Chronometre` ;
- lier les méthodes de l'interface de `Chronomètre` aux boutons de l'interface graphique.

On donne ci-dessous l'interface de la classe `Afficheur` à implémenter :

- `def demarrer() -> None` : déclenche la mesure de durée. On voit l'affichage de la durée progresser dans la fenêtre.
- `def arreter() -> None` : interrompt la mesure de durée ainsi que l'affichage.
- `def effacer() -> None` : réinitialise la mesure de durée à zéro ainsi que l'affichage.
- `def rafraichir()` : met à jour l'affichage si une mesure est en cours.

Liaison des méthodes Pour lier l'appel d'une méthode à un clic de bouton, il suffit de passer comme argument au constructeur du bouton la référence de la méthode qu'il doit appeler ; `command=self.nom_methode`.

Fonction de rappel La méthode `rafraichir` devra utiliser une technique de la programmation événementielle pour mettre à jour l'affichage. Il s'agit de l'usage d'une fonction de rappel (ou callback) qui sera appeler toutes les `10 ms`. Concrètement il suffit d'appeler la méthode `after(10, self.rafraichir)` de l'objet de type `Label` dans la méthode `rafraichir` elle-même. L'effet obtenu sera que la méthode `rafraichir` s'appellera elle-même toute les `10 ms`. On en profitera donc pour mettre à jour l'affichage du chronomètre.