Exercice 1 :

1. Écrire une fonction **applique(f, t : tuple) tuple** qui applique la fonction f sur tous les éléments du tuple t et renvoie le nouvel ensemble.
2. Écrire une fonction **double(x : int) int** qui renvoie le double de x.
3. Tester *applique* avec la fonction *double* et un tuple d'entiers.

Exercice 2 :

1. Écrire une fonction **repeter_delai(f, n : int, t : int) None** qui effectue n appels *f(0,...,f(n-1)* où chaque appel est suivi d'une pause de t secondes. La documentation Python ci-après peut être utile pour effectuer la pause : https://docs.python.org/fr/3/library/time.html
2. Écrire une fonction **dessine(n : int) → None** qui trace un trait de longueur $n * 50$ avec la bibliothèque *turtle* puis qui tourne à gauche d'un angle de 90°.
3. Tester *repeter_delai* avec la fonction *dessine*.

Exercice 3 :

1. Écrire une fonction **trouve(p, t : tuple) → object** qui reçoit une fonction p et un tuple t et renvoie le premier élément x de t tel que $p(x) = True$. Si aucun élément ne satisfait p alors la fonction renvoie *None*.
2. Écrire une fonction **est_positif(x : int) → bool** qui renvoie True si x est positif.
3. Utiliser la fonction *trouve* sur un tuple d'entiers et la fonction *est_positif*.
4. Écrire une fonction **est_premier(n : int) → bool** qui renvoie True si n est un nombre premier.
5. Utiliser la fonction *trouve* sur un tuple d'entiers et la fonction *est_premier*.

Exercice 4 : Notion d'invariant

```python
def tri_insertion(tab):
    taille = len(tab)
    for i in range(1,taille):
        en_cours = tab[i]
        j = i-1
        while j >= 0 and tab[j] > en_cours:
            tab[j+1] = tab[j]
            j -= 1
        tab[j+1] = en_cours
    return tab
```

L'invariant du tri par insertion peut être : « Le tableau tab[0:i] est trié. »

1. Rappeler la définition d'un invariant. À quoi sert-il ?
2. Quelle est la valeur de i avant la première itération du tri ? L'invariant est-il vérifié ?
3. Appliquer un raisonnement par récurrence pour montrer que l'invariant est vrai pour tout i. Conclure.

Exercice 5 : L'expression Python **lambda** crée une fonction anonyme (https://docs.python.org/fr/3/reference/expressions.html?#lambda).

1. Dans la console, tester le code ci-après :

```python
f = lambda x: x*2
f(5)
```


Christophe Viroulaud

2. Écrire une fonction g telle que pour tout x, $g(x) = 2x + 3$.
3. Écrire une fonction **h(f,g)** qui reçoit deux fonctions f et g en arguments et renvoie leur composition, c'est à dire une fonction h telle que pour tout x, $h(x) = f(g(x))$.
4. Tester *h(5)*.

Exercice 6 :

1. Écrire une fonction **calcul(operation, l : tuple)** $\rightarrow$ **int** qui effectue le calcul suivant :

$$t[0] \; operation \; t[1] \; operation \; ... \; operation \; t[n-1]$$

2. Écrire une fonction anonyme **addition** qui additionne deux entiers x et y.
3. Utiliser la fonction *calcul* avec un tuple d'entiers et la fonction *addition*.
4. Tester la fonction *calcul* avec d'autres opérations.

Christophe Viroulaud