CONSIGNE POUR L'ENSEMBLE DU TP

1) **Dans le répertoire lp2 créer un répertoire TP02.**
2) **Dans le répertoire TP02 utiliser un fichier pour chaque exercice.**
3) **nommer vos fichiers logiquement ex1.py , ex2.py ... ou EX1.py EX2.py ... ou exercice_1.py etc. Bref quelque chose de logique.**
4) **Vérifier que vous pouvez sauvegarder votre travail. Si ce n'est pas le cas, signalez vous à la chargée ou au chargé de TP (sinon cocher la case sous OUTILS / OPTION / case:"Utiliser boite de dialogue fichier de Tk au lieu de zenity")**

ECRIVEZ systématiquement des tests de vos fonctions ! Testez vos procédures. !

Pour réutiliser les fonctions écrites lors du TP1 pour entrer des entiers, télécharger le fichier module_EX_3_TP1.py dans votre répertoire de travail TP02 !

Exercice 10 :

Écrire une fonction **entrelacement(s1,s2)** qui prend en paramètres deux chaînes de caractères **s1** et **s2** de **même** longueur et qui renvoie la chaîne qui contient en alternance un caractère de **s1** suivi d'un caractère de **s2**.

Par exemple, **entrelacement('abc' , '123')** renvoie **a1b2c3**.

Rappel Vous devez toujours tester votre code

travail à faire :

1) Ecrire la fonction entrelacement dont la signature est définie ci-après.

code

```
def entrelacement(s1 : str ,s2 : str ) -> str :
        """ Cette fonction renvoie la chaine entrelacée
        le premier caractère de s1 puis le premier de s2
        le deuxième caratère de s1 puis le deuxième de s2
        ...
        le dernier caractère de s1 puis le dernier de S2"""
```

2) Pour tester votre fonction, compléter le code ci-dessous. Ce code :

- vérifie que les chaînes ont la même longueur pour continuer.
- appelle la fonction entrelacement.
- affiche le résultat.

code

```
def entrelacement(s1 : str ,s2 : str ) -> str
        # votre de la fonction ICI
        # votre code

s1 : str ="abc"   # définition de s1
s2 : str ="123"   # définition de s2
if len(s1)!=len(s2): # le programme s'arrête dans ce cas.
        print('Erreur de saisie, le programme se termine') #
else : #le programme continue et appelle la fonction entrelacement.
        chaineResultat = entrelacement(s1,s2) # appel de la fonction
        print(f"La chaine {...} et la chaîne {...} se transforment en {...}"
              f"après l'appel de la fonction entrelacement()" )
```

3) Modifier les chaines s1 et s2 à votre convenance, tester des chaînes de même longueur ou pas.

Exercice 11 :

1) Écrire une fonction **nombre_apparitions(c , s)** qui prend en paramètres un caractère **c** et une chaîne de caractères **s** et qui renvoie le nombre de fois où **c** apparait dans **s**. Par exemple :

code.py

```
>>> nombre_apparitions('e','les merveilleuses')
5
```

Tester votre fonction sur plusieurs cas que vous définirez pour vérifier que le résultat est bien celui que vous attendez.

2) En utilisant la fonction précédente, écrivez une fonction **absence_de_e(s)** qui prend en paramètre une chaîne de caractères **s** et renvoie :

- **True** si **s** ne contient ni le caractère **e** ni **E**.
- **False** si **s** contient le caractère **e** ou **E**.

3) Pour tester votre fonction, écrire des tests de la fonction **absence_de_e(s)**.

Exercice 12 :

1) Écrire une fonction **affiche_miroir(s)** qui prend une chaîne de caractères **s** et qui affiche la chaîne **s** et son image miroir (**s** à l'envers).

SHELL

```
>>> affiche_miroir('abc')
abc cba
```

2) Écrire une fonction **miroir(s)** qui cette fois-ci retourne la chaîne de caractères **s** à l'envers, de sorte que l'on puisse répondre à la question 1 par une nouvelle fonction affiche_miroir_v2() :

code.py

```
def miroir(s : str) -> str :
        """Cette fonction renvoie la chaîne miroir,
         le premier caractère étant le dernier de la chaîne s
         le second étant l'avant dernier etc."""


def affiche_miroir_v2(s : str)-> None: # cette fonction est en fait
                                       #une procédure(voir le cours).
        print(s,miroir(s))
```

3) Écrire une fonction **est_un_palindrome(s)** qui :

- retourne **True** si **s** est un palindrome
- retourne **False** sinon

autrement dit un mot qui est égal à son image miroir. Proposez une solution qui n'utilise pas la fonction **miroir** et qui ne crée aucune nouvelle chaîne de caractères. Écrire quatre tests avec **assert**.

Exercice 13 :

Créer dans votre répertoire de travail le fichier texte (**message.txt**)de 3 lignes comme ci-dessous. En cas de problème pour créer le fichier, appeler la ou le chargé(e) de TP.

Contenu de message.txt :

```
Bonjour
Comment ça va ?
Bonne journée.
```

1) Ecrire une fonction `lecture_fichier()` qui renvoie une liste de chaine de caractères contenant les lignes du fichier.

Remarque : l'ouverture la lecture ou écriture ont été vues en IP1 et sont également rappelées dans le CM1 de IP2.

code

```
def lecture_fichier(nom : str) -> list[str] :
        """Cette fonction lit de fichier dont le nom est en paramêtres
        et renvoie une liste, chaque élément de la liste
        est une ligne du fichier texte"""
```

Dans l'interpréteur de commande python taper :

SHELL

```
>>> print (lecture_fichier("message.txt"))
```

Avez vous repérez et compris l'affichage obtenu ? Demander à votre voisinage ou au chargé(e) de TP.

Rappel : cette affichage a été obtenu lors des TP en IP1 !!

2) Ecrire une fonction (en fait une procédure) qui affiche la liste obtenue à la question 1. Le contenu est à afficher ligne par ligne avec la précision du numéro de ligne (Ligne 1 : etc) comme dans l'exemple ci-après.

code

```
def affiche_liste(contenuFichier : list[str] ) -> None :
        """Affiche les lignes en ajoutant  Ligne 1 :  puis Ligne 2 etc"""
```

Pour tester et valider votre code vous pouvez utiliser l'interpréteur par exemple.

SHELL

```
>>> affiche_liste( contenuFichier )
Ligne 1 : Bonjour
Ligne 2 : Comment ça va?
Ligne 3 : Bonne journée
```

Exercice 14 :

Écrire un programme qui lit un fichier texte et crée un nouveau fichier où toutes les lignes vides sont supprimées. Remarque : Pour valider ce programme, vous pouvez créer votre fichier texte de votre choix en le créant dans le même répertoire de travail et en insérant des lignes vides avec une ou plusieurs frappe de la touche entrée.

Ci dessous voici un exemple de fichier qui comprend des lignes vides. Vous pouvez voir que les lignes numéro 2,3 et 5,6 et 7 sont vides.

messageAvecLignesVides.txt

```
1 Bonjour
2
3
4 Comment ça va ?
5
6
7
8 Bonne journée.
```

Votre programme doit afficher le contenu sans lignes vides. Votre programme doit écrire le fichier "FichierSansLignes.txt".

D'après l'exemple précédent le fichier résultat correspondant est :

messageSansLignesVides.txt

```
1 Bonjour
2 Comment ça va ?
3 Bonne journée.
```

Remarque : le cours CM1 IP2 contient les lignes de codes pour lire un fichier texte ou écrire ce même type de fichier.

Exercice 15 :

Écrire (et tester !) un programme qui demande à l'utilisateur de saisir son prénom, son âge et sa ville, puis affiche ces informations sous forme de phrase formatée avec une **f-string**.

Exercice 16 :

Objectif de l'éxercice :Écrire un programme qui demande à l'utilisateur de saisir la longueur et la largeur d'un rectangle, calcule sa surface, et affiche un message contenant le résultat formaté avec une **f-string**. La longueur et la largeur sont des valeurs entières, ce qui vous permettra de ré-utiliser les fonctions de l'EX3 du TP1. Votre programme doit commencer par importer la fonction `demander_entier()` contenue dans le code `module_EX_3_TP1.py` et que vous pouvez renommer en `entrer_entier` par exemple

1) Pour ré utiliser la fonction écrite dans le TP1 EX3 , recopier en début de votre programme les instructions suivantes :

code

```
        # cette instruction lit la fonctions écrite et validée précédemment.
        # vous disposez de la fonction demander_entier directement renommée ici en entrer_entier()
        # pour savoir ce que fait cette fonction taper help (demander\_entier)
        # et vous aurez l'affichage du contenu du docstring.
        from module_EX_3_TP1 import demander_entier as entrer_entier
        # faite un test :
        nombre=entrer_entier("Taper un entier")
        print(f"le nombres saisi est {nombre}")
```

2) Tester ce code en l'éxécutant. Vous disposez maintenant d'une nouvelle fonction sur laquelle le programme ne peut faire d'arrêt suite à une erreur.

3) Ecrire et tester un programme à l'aide d'une ou plusieurs fonctions pour :

- Demander la longueur à l'utilisateur.
- Demander la largeur du rectangle à l'utilisateur.
- Calculer la surface du rectangle.
- Afficher le résultat (avec une **f-string**)

Exercice 17 :

Écrire un programme qui demande à l'utilisateur un entier **"n"**, puis affiche la table de multiplication correspondante jusqu'à 10, avec un affichage formaté en utilisant les **f-string**.

Vous pouvez si vous le souhaitez, ré-utiliser la fonction du module du TP1. Pour tester ces 2 lignes de code, voir EX 16

code

```
from module_EX_3_TP1 import demander_entier as entrer_entier
# faite un test :
nombre=demander_entier("Taper un entier")
```

L'affichage dans l'interpréteur python doit être de ce style :

$2 \times 5 = 10$

Exercice 18 :

Écrire un programme qui demande à l'utilisateur de saisir un nombre entier et indique s'il est pair ou impair. Vous pouvez ré-utiliser le fichier module_EX_3_TP1.py ou utiliser la méthode `isnumeric()`. Si vous utilisez la méthode, lire la documentation ou faire des tests pour bien comprendre ce qu'elle fait.

Exercice 19 :

Écrire (et tester !) un programme qui demande à l'utilisateur de saisir une note entière (entre 0 et 20) et affiche un message basé sur cette note :

- Si la note est supérieure ou égale à 16, le programme affiche "Très bien".
- Si elle est entre 12 (inclus) et 16 (exclus), le programme affiche "Bien".
- Si elle est entre 8 (inclus) et 12 (exclus), le programme affiche "Passable".
- Sinon, le programme affiche "Insuffisant".

Votre programme doit vérifier que la note est bien entre 0 et 20. Si ce n'est pas le cas, l'utilisateur est invité à nouveau à saisir la note.

Vous pouvez ré-utiliser les fonctions de module_EX_3_TP1.py.

Exercice 20 : Écrire (et tester !) un programme qui demande à l'utilisateur de saisir uniquement un entier. Le programme vérifie si cet entier satisfait une ou plusieurs des conditions suivantes et affiche un message approprié. Si aucune condition n'est remplie, le programme l'indique toujours par un message ad'hoc.

Les conditions à tester sont :

- L'entier est divisible par 3.
- L'entier est pair.
- L'entier est strictement positif.
- L'entier est compris entre 10 et 50 inclus.

Vous pouvez ré-utiliser les fonctions de module_EX_3_TP1.py.

Exercice 21 : Écrire un programme qui demande à l'utilisateur de saisir uniquement un entier. Le programme vérifie si cet entier satisfait la condition suivante et affiche un message approprié.

- Le nombre est un multiple de 5 mais pas de 10.

Si aucune condition n'est remplie, le programme l'indique toujours par un message ad'hoc. Le programme doit demander à l'utilisateur s'il souhaite tester un autre nombre après chaque exécution.

Vous pouvez ré-utiliser les fonctions de module_EX_3_TP1.py.

Exercice 22 : Conjecture de Syracuse.

Une suite de Syracuse est une suite d'entiers naturels définie de la manière suivante : on part d'un nombre entier strictement positif. S'il est pair, on le divise par 2. Sil est impair, on le multiplie par 3 et l'on ajoute 1. En répétant lopération, on obtient une suite d'entiers strictement positifs dont chacun ne dépend que de son prédécesseur.

$$u_{n+1} = \begin{cases} \frac{u_n}{2}, & \text{si } u_n \text{ est pair,} \\ 3 \times u_n + 1, & \text{si } u_n \text{ est impair} \end{cases}$$

Par exemple, à partir de 10, on construit la suite des nombres : 10, 5, 16, 8, **4, 2, 1**, **4, 2, 1** et ainsi de suite. Après que le nombre 1 a été atteint, la suite des valeurs 1, 4, 2, 1, 4, 2 se répète indéfiniment en un cycle de longueur 3.

Le **temps de vol** d'un entier u_0 est le nombre n d'étapes du calcul pour arriver la première fois à $u_n = 1$.. Le temps de vol de la suite dont le premier terme est 10 est de 6.

Travail à faire :

1) Rédiger et tester une fonction calculant le terme suivant de la suite de syracuse. Ecrire plusieurs tests de votre choix en complément des 2 proposés.

code

```
def elementSuivant( n : int  ) -> int  :
        """cette fonction renvoie l'élément de rang n+1
        de la suite de Syracuse."""

assert(elementSuivant(1) == 4  )
assert(elementSuivant(3) == 10  )
```

2) Rédiger et tester une fonction qui calcule tous les termes de la suite de Syracuse pour un u_0 donné.Ecrire plusieurs tests de votre choix en complément des 3 proposés.

code

```
def CalculElementsDeLaSuite (u0 : int) -> list[int] :
    """cette renvoie tous les éléments de la suite
    dont le premier terme est passé en paramêtre"""

assert( CalculElementsDeLaSuite ( u0=1 ) == [1] )
assert( CalculElementsDeLaSuite ( u0=2 ) == [2 , 1] )
assert( CalculElementsDeLaSuite ( u0=5 ) == [5, 16, 8, 4, 2, 1] )
```

3) Rédiger et tester une fonction calculant le temps de vol d'une suite de Syracuse. Ecrire plusieurs tests de votre choix..

code

```
1
2 def TempsDeVol (listeTermes : list[int] ) -> int:
3     """ Cette fonction renvoie le temps de vol
4     qui est le nombre d'étapes pour arriver à 1."""
5
6 assert( TempsDeVol( CalculElementsDeLaSuite ( u0=1 ) ) == 0)
7 assert( TempsDeVol( CalculElementsDeLaSuite ( u0=2 ) ) == 1)
```

4) Rédiger et tester la fonction qui recherche le plus petit entier ayant un temps de vol supérieur ou égal à 100. Et compléter la partie "# programme principal" du code ci-dessous pour appeller les fonctions précédentes.

code

```
1 def recherche_nombre(tdvMin : int) -> tuple[int ,int, list[int] ] :
2     """ tdvMin est le temps de vol minimum voulu.
3     Cette fonction itére sur la valeur de u0.
4     Au début u0=1  d'où  TdV=0
5     Ensuite u0=2  d'où TdV =1
6     ...
7     Ensuite u0=10 d'où TdV = 6
8     ...
9     La fonction incrémente la valeur de u0 tant que TdV est inférieur à TdVMin.
10    Cette fonction renvoie renvoie un tuple (u0 , tdV , listeDeTerme )."""
11 assert(recherche_nombre(0)==(1 ,0 , [1] ) )
12 assert(recherche_nombre(1)==(2 ,1 , [2 , 1] ) )
13 assert(recherche_nombre(5)==(3 ,7 , [3, 10, 5, 16, 8, 4, 2, 1] ) )
14 # programme principal
15 tdvMin=100
16 (u0,tdv, termes) = recherche_nombre(tdvMin)
17 print( f"Pour un temps de vols minimum de {tdvMin}")
18 print( f"Le premier entier correspondant est {u0} ")
19 print( f"Le temps de vol correspondant est {tdv}")
20 print( f"La liste des termes de la suite est {termes}")
```

Le saviez vous ?

Actuellement en 2025, aucun chercheur n'a trouvé de nombre de départ qui n'aboutisse pas à 1, puis au cycle trivial. Pour information un prix d'environ 1 million de dollar sera décerné en cas de démonstration de la conjecture :

https://mathprize.net/posts/collatz-conjecture/.