Exercice 10 :

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

1) La fonction entrelacement() peut s'écrire :

entralecement.py

```python
def entrelacement(s1, s2):
        """ 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"""
    resultat : str  = ""
    for i in range(len(s1)):
        resultat = resultat + s1[i] + s2[i]
    return resultat
```

2) Dans le code ci-dessous, la fonction entrelacement() avec son code est écrite en premier, les chaînes s1 et s2 sont définies "en dur". Le test sur la longueur suit, et enfin les résultats sont affichés.

code

```python
def entrelacement(s1 : str ,s2 : str ) -> str :
    resultat : str  = ""
    for i in range(len(s1)):
        resultat = resultat + s1[i] + s2[i]
    return resultat

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 {s1} et la chaîne {s2} se transforment "
        f"en {chaineResultat} "
        f"après l'appel de la fonction entrelacement()" )
    # veuillez noter la coupure de la chaine f-string
```

3) Pour tester le code vous pouvez saisir des chaînes de votre

code

```
# pour voir une branche du test
s1 : str ="abcd"   # définition de s1
s2 : str ="123"   # définition de s2
#
s1 : str ="tt"   # définition de s1
s2 : str ="oo"   # définition de s2
# à l'utilisateur de faire varier les paramètres.
```

Exercice 11 :

code.py

```python
def nombre_apparitions(c : str ,s : str ) -> int :
    res = 0
    for i in range(len(s)) :
        if s[i] == c :
            res = res + 1
    return res

# pour tester la fonction en écrivant ceci le programme ne renvoit rien car il ne détecte
assert(nombre_apparitions('a','aaa') == 3)
assert(nombre_apparitions('b','aaa') == 0)
assert(nombre_apparitions('b','') == 0)

# essayer de taper une prévisions fausse et vous verrez
# le message assertion error apparaître
# assert(nombre_apparitions('a','aaa') == 4) # renvoie une erreur

def absence_de_e_if(s :str) -> bool :
        if nombre_apparitions('e',s) == 0 and nombre_apparitions('E',s) == 0 :
                return True
        else :
                return False

def absence_de_e(s : str ) -> bool :  # cette version est plus compacte
                                #       et se passe du if
        return nombre_apparitions('e',s) == 0 and nombre_apparitions('E',s) == 0

assert absence_de_e("")==True        # On essaye la chaîne vide
assert absence_de_e("Salut")==True   # On essaye un chaîne sans e
assert absence_de_e("eSalut")==False # On essaye e/E au début au mileu et à la fin
assert absence_de_e("SalEut")==False
assert absence_de_e("Salute")==False
```

L'utilisation de la fonction assert est pratique pour les fonctions renvoyant un résultat que vous pouvez

connaître à l'avance. La fonction assert reste écrite dans le code, c'est utile pour vous indiquer que les fonctions produisent le résultat attendu.

Maintenant pour vous convaincre , vous pouvez aussi en phase de développement utiliser l'interpréteur Python. Cela permet de voir les éventuelles modifications à faire.

SHELL

```
>>>>> %Run EX11.py    #il faut lancer le script pour que l'interpréteur connaissent les fo
>>> nombre_apparitions('a','aaa')
3
>>>
>>> absence_de_e("Salut")
True
>>>
```

Exercice 12 :

1) La fonction affiche_miroir() est une procédure car elle ne fait que de l'affichage, elle renvoie None .

SCRIPT

```
def affiche_miroir(s : str) -> None : # c'est une procédure, donc pas d'assert !
        print(s , end=' ') # pas de retour à la ligne, juste un espace
        # i prend les valeurs 1, 2, 3, ... , len(s)-1
        i = 0
        while i < len(s) :
                print(s[len(s) - i - 1] , end='') 1# on pourrait faire aussi
                i=i+1           # un while avec l'indice variant de -1 en -1
        print()
```

Pour tester la procédure il faut utiliser l'interpréteur et essayer plusieurs chaînes.

SHELL

```
>>> %Run EX12.py      # éxecuter une fois le code.
>>> affiche_miroir('abc')
abc cba

>>> affiche_miroir('er')
er re
>>> affiche_miroir('')  # châine vide

>>>
```

2) La fonction miroir peut s'écrire comme ci-dessous, elle est ensuite appelée par la procédure affiche_miroir_v2().

SCRIPT

```
def miroir(s : str) -> bool :
    res = ''
    for i in range(len(s)) :
        res = s[i] + res      # attention à l'ordre de la concaténation !
    return res
assert(miroir('titi')=='itit')


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

3) Ci après se trouve la fonction est_un_palindrome() qui fait appelle à la fonction miroir. Vous trouverez également la fonction est_un_palindrome_SANS_miroir() qui n'appelle pas la fonction miroir() mais qui est plus longue à coder.

EX12.py SUITE

```
def est_un_palindrome(s : str) -> bool :   # utilise la fonction miroir !!!!
        if s == miroir(s) :
                return True
        else :
                return False

assert est_un_palindrome("toto") == False # cas faux
assert est_un_palindrome("essayasse") == True # longueur impaire
assert est_un_palindrome("ANNA") == True # longueur paire
assert est_un_palindrome("") == True  # chaîne vide

def est_un_palindrome_SANS_miroir(s : str ) -> bool:
    nbLettres    : int = len(s)
    nbAcomparer : int = len(s)//2
    # si len(s)=7 alors on compare les 7//2 = 3 premieres valeurs aux 3 dernières.
    # si len(s)=6 alors on compare les 6//2 = 3 premieres valeurs aux 3 dernières.
    i : int = 0
    while  i< nbAcomparer :
        if s[i] != s[nbLettres -i -1] :    # on compare en miroir.
            return False
        i=i+1
    return True

assert est_un_palindrome_SANS_miroir("toto") == False # cas faux
assert est_un_palindrome_SANS_miroir("essayasse") == True # longueur impaire
assert est_un_palindrome_SANS_miroir("ANNA") == True # longueur paire
assert est_un_palindrome_SANS_miroir("") == True  # chaîne vide
```

Exercice 13 :

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

1) Vous trouverez si besoin (non demandé dans l'énoncé) la procédure pour créer un fichier texte pour le charger par la suite.

```
def cree_fichier_txt(nomFichier)-> None :
    # Création du fichier message.txt (Ce n'était pas demandé dans l'énoncé)
    file = open(nomFichier, "w", encoding="utf-8")   # ouverture
    file.write("Bonjour\n")                            # ecriture
    file.write("Comment ça va ?\n")
    file.write("Bonne journée.\n")       # \n pour avoir le retour à la ligne.
    file.close()                                        # fermeture
cree_fichier_txt("message2.txt")   # le fichier message2.txt existe après cela.
```

La fonction lit_fichier() (qui est une procédure) s'écrit :

```
def lecture_fichier(nomFichier: 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"""
    fichier  = open( nomFichier , "r" , encoding="utf-8")# ouverture
    lignes = fichier.readlines() #chargement de toutes les lignes
    fichier.close()#fermeture.
    return lignes

ListeChaine :str = lecture_fichier("message2.txt")

print(ListeChaine)
```

Dans l'interpréteur, en tapant la commande demandée on obtient :

SHELL

```
>>> print (lecture_fichier("message.txt"))
['Bonjour\n', 'Comment ça va ?\n', 'Bonne journée.\n']
```

le caractère "**n**" correspond au retour à la ligne (vu en IP1)

2) La fonction afficher_liste(() (qui est ici une procédure) s'écrit :

d

```
ef afficher_liste(contenuFichier : list[str] )-> None :
    """Affiche les lignes en ajoutant  Ligne 1 :  puis Ligne 2 etc"""
    k: int =0
    while k< len(contenuFichier) :
        print(f"Ligne {k+1} : {contenuFichier[k]}", end="") # affiche
        #le contenu du fichier en ajoutant Ligne 1 : etc
         # le  end="" évite le saut de ligne supplémentaire
         # on ajoute end="" car le caractère \n fait passer à la ligne.
        k=k+1

afficher_liste(ListeChaine)
```

La procédure se teste avec l'interpréteur par exemple :

SHELL

```
>>> afficher_liste(ListeChaine)
Ligne 1 : Bonjour
Ligne 2 : Comment ça va ?
Ligne 3 : Bonne journée.
```

Exercice 14 :

Un programme lisant un fichier texte et supprimant les lignes vides (ou plus exactement contenant uniquement "\n" est le suivant :

SCRIPT

```
fichier   = open("fichier_avec_lignes_vides.txt","r")# ouverture
lignes = fichier.readlines() #chargement de toutes les lignes
fichier.close()  #fermeture.


fichierSortie  =  open("fichier_sans_lignes_vides.txt", 'w')
index = 0
while index < len(lignes):
    # Si la ligne n'est pas vide (différente de "\n"), on l'écrit dans le fichier de sorti
    if lignes[index] != "\n":
        fichierSortie.write(lignes[index])
    index = index + 1
fichierSortie.close()  #fermeture.
```

Pour vérifier la bonne éxécution, soit vous rechargez rigoureusement le fichier avec une fonction python et un affichage ou alors directement en utilisant le gestionnaire de fichier de votre ordinateur pour ouvrir le fichier et faire les vérifications.

Exercice 15 :

Le programme effectue les entrées nécessaires sous forme de chaînes de caractères, et les affiche. Pour tester le programme il suffit de lancer plusieurs fois et de changer les valeurs tout en vérifiant que cela correspond aux attentes.

SCRIPT

```
# Demander à l'utilisateur de saisir son prénom, son âge et sa ville
prenom = input("Quel est votre prénom ? ")
age = input("Quel est votre âge ? ")
ville = input("Dans quelle ville vivez-vous ? ")

# Afficher les informations sous forme de phrase formatée avec une f-string
print(f"Bonjour {prenom}, vous avez {age} ans et vous vivez à {ville}.")
```

Exercice 16 :

1) Facile.

2) Facile.

3) Le code répondant à la question est le suivant. L'utilisation de la fonction écrite au TP1 vous évite d'avoir à refaire les tests pour vérifier la validité des entrées de l'utilisateur.

SCRIPT

```
# cette instruction lit la fonctions écrite et validée précédemment.
# vous disposez de la fonction demander_entier directement
# 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

longueur : int = entrer_entier("Quelle est la longueur du rectangle ? ")
largeur  : int = entrer_entier("Quelle est la largeur du rectangle ?  ")

surface : int  = longueur * largeur

print(f"La surface du rectangle est de {surface} unités carrées.")
```

Exercice 17 :

Écrire un programme qui demande à l'utilisateur un entier **"n"**, puis affiche sa table de multiplication correspondante jusqu'à 10 $(1 \times n, 2 \times n, ..., 10 \times n)$, avec un affichage formaté en utilisant les **f-string**.

SCRIPT

```
from module_EX_3_TP1 import demander_entier as entrer_entier

n : int = entrer_entier("Entrez un entier n pour afficher la table de multiplication : ")

for i in range(1, 11):
    print(f"{n} x {i} = {n * i}")
```

Exercice 18 :

La méthode `isnumeric()` permet de voir si l'entrée saisie est correcte pour être convertie dans le type entier ou int :

EX18.py

```
def EntreEntier(question : str) -> int :
    data_ok : bool = False
    while not(data_ok):
        n_input = input(question)
        if n_input.isnumeric() and int(n_input) > 0:
            n = int(n_input)
            return n
        else:
            print("Veuillez entrer un entier positif valide.")


n = EntreEntier("Entrez un entier positif : ")
if n % 2 == 0:
    print(f" {n} est pair.")
else :
    print(f" {n} est impair.")
```

Deuxième possibilité, en réutilisant le code du module du TP1...

code

```
from module_EX_3_TP1 import demander_entier as entrer_entier

n = entrer_entier("(utilise entrer_entier() ). Entrez un entier positif  : ")
if n % 2 == 0:
    print(f" {n} est pair.")
else :
    print(f" {n} est impair.")
```

Exercice 19 :

Écrire un programme qui demande à '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".

EX19.py

```python
from module_EX_3_TP1 import demander_entier as entrer_entier

def AnalyseNote( note : float ) -> None :

    if   16 <= note and note <= 20 :
        print("Très bien.")
    elif  12<=note  and note <16 :
        print("Bien.")
    elif 8<=note  and note <12 :
        print("Passable.")
    else :
        print("Insuffisant.")

# veuillez noter comment éviter d'avoir des lignes qui sortent de l'éditeur.
# il suffit de rajoute un  " à la fin de la ligne à couper et un " au début
# de la nouvelle ligne
note : int  = entrer_entier(" Veuillez saisir une note entière entre 0 et 20"
                            "pour avoir une mention : ")
while note>20:
    print("La note saisie est supérieure à 20 ! Veuillez recommencer.")
    note : int  = entrer_entier(" Veuillez saisir une note entière entre"
                                "0 et 20 pour avoir une mention : ")

AnalyseNote(note)
```

Exercice 20 :

EX20.py

```python
entier : int  = 12      # on simplifie l'exercice

c1 : bool=False   # booléen pour retenir si les conditions sont vérifiées
c2 : bool=False   # pour le test final.
c3 : bool=False
c4 : bool=False
if entier % 3 == 0:
    c1 = True
    print(f"L'entier {entier} est divisible par 3.")
# pas de elif car un nombre peut remplir plusieurs conditions.
if entier % 2 == 0:
    c2 = True
    print(f"L'entier {entier} est pair.")
if entier > 0:
    c3 = True
    print(f"L'entier {entier} est  strictement positif. ")
if 10 <= entier and entier <= 50:
    c4 = True
    print(f"L'entier {entier} est   compris entre 10 et 50 inclus. ")

# Si aucune des conditions n'est vraie, on affiche un message par défaut
if not (c1 or c2 or c3 or c4):
    print("Aucune des conditions n'est vérifiées.")
```

Exercice 21 :

EX21.py

```python
from module_EX_3_TP1 import demander_entier as entrer_entier

calcul : bool =True
while calcul :
    entier : int  = entrer_entier("Veuillez saisir un entier : ")
    if  entier%10!=0 :
        if entier%5==0  :
            print(f"Le nombre {entier} est divisible par 5 mais pas par 10")
        else :
            print(f"Le nombre {entier} n'est pas divisible par 5 ou par 10")
    else :
        print(f"Le nombre {entier} est divisible par 10 (...et par 5 aussi) !")
    reponse = input ('Entrer o  pour effectuer un autre calcul :   ')
    if reponse !='o' :
        calcul = False
print("Programme terminé.Merci.")
```