Exercice 1 :

Écrire les réponses sur papier, puis utiliser l'interpréteur python pour vérifier et surtout comprendre les différences.
Quelle est la valeur de chacune des expressions suivantes une fois entrées dans l'interpréteur Python ?

code.py

```
>>> 4 // 5 * 3 + 2 ** 3
8
>>> 4 // 5 * (3 + 2) ** 3
0
>>> 2 == 1 + 1
True
>>> 2 == 1 + 1 + 1
False
>>> (2 == 1 + 1 + 1) and (2 == 1 + 1)
False
>>>2 == 1 + 1 + 1 or 2 == 1 + 1
True
>>>1 == 0 // 0
ZeroDivisionError
>>>(not (0 == 0)) and 1 == 0 // 0
False
>>> "2" == '1' + "1"
False
```

Exercice 2 :

Écrire les réponses puis utiliser l'interpréteur python pour vérifier et surtout comprendre les différences.
Écrire sur papier tout ce qui est affiché par ce code python, puis le vérifier en éxécutant le script.

code.py

```
hauteur = 50
print("Est-il vrai que hauteur = 50 ?")
print(hauteur == 50)
b=56
print(b != 5 and b != 6)
print(b <= 5 or b >= 6)
print(b == 50+6 or b >= 6)
print(b >= 6 and b < 50+6)
```

SHELL

```
>>>
Est-il vrai que hauteur = 50 ?
True
True
True
True
False
```

Exercice 3 : Opérations sur les entiers.

Un groupe d'élèves souhaite former des équipes pour un tournoi sportif. Le nombre total d'élèves et la taille de chaque équipe sont donnés.

1. Résoudre le problème de calcul ,comme au collège, sur votre cahier de brouillon. Le groupe est composé de 28 élèves, il faut 6 joueurs par équipe. Calculer le nombre d'équipes et le nombre de joueurs sans équipe pour avoir des équipes de 6 joueurs.

   $28 = 6 \times 4 + 4$ Il y a 4 équipes de 6 joueurs et 4 joueurs sans équipe.

2. Pas de correction à tester dans le shell

3. Question 3 :

EX3.py

```python
ChaineInput : str  = input ("Entrer un entier :")
NbEntier : int = int (ChaineInput)
print(f"Le double de {NbEntier} est {2*NbEntier}")
```

Ce code python affiche 20 si l'on entre 10. Mais génère une erreur "ValueError" si l'on entre la chaîne "dix". C'est la raison pour laquelle il faut s'assurrer du bon contenu de la chaîne pour la transformer sans erreur en entier.

4. Question 4 : Première possibilité, rédiger tous les tests, c'est un peu long.

EX3.py

```
def  est_un_chiffre_v1(c : str) -> bool :
    if c == '0' :
        return True
    if c == '1' :
        return True
    if c == '2' :
        return True
    if c == '3' :
        return True
    if c == '4' :
        return True
    if c == '5' :
        return True
    if c == '6' :
        return True
    if c == '7' :
        return True
    if c == '8' :
        return True
    if c == '9' :
        return True
    # Si aucun test précédent n'est vrai la fonction renvoie Faux.
    return False
```

Deuxième possibilité, utiiser des or pour avoir moins de lignes. Attention le code semble un peu moins lisible.

EX3.py

```
def  est_un_chiffre_v2(c : str) -> bool :
    # le même code que _v2 en moins de ligne, un peu moins lisible.
    if c == '0' or c == '1'  or c == '2' :
        return True
    if  c == '3' or c == '4'  or c == '5' :
        return True
    if  c == '6' or c == '7'  or c == '8' or c == '9'  :
        return True
    # Si aucun test précédent n'est vrai la fonction renvoie Faux.
    return False
```

La version la plus élégante et courte. L'idée est de comparer le caractère c aux caractères de la chaine qui contient tous les chiffres :

EX3.py

```
def est_un_chiffre(c : str) -> bool :
    # définition de la chaîne avec les caractères qui sont des chiffres
    chiffres : str ="0123456789"
    i : int =0
    while i< len(chiffres) :
        if c == chiffres[i] :
            return True    # "c" est un des caractères de chiffres.
        else :
            i=i+1
    return False # on sort de la boucle sans avoir trouvé de chiffre

    assert( est_un_chiffre_v3("0")==True)
    assert( est_un_chiffre_v3("1")==True)
    assert( est_un_chiffre_v3("2")==True)
    assert( est_un_chiffre_v3("3")==True)
    assert( est_un_chiffre_v3("4")==True)
    assert( est_un_chiffre_v3("5")==True)
    assert( est_un_chiffre_v3("6")==True)
    assert( est_un_chiffre_v3("7")==True)
    assert( est_un_chiffre_v3("8")==True)
    assert( est_un_chiffre_v3("9")==True)
    assert( est_un_chiffre_v3("z")==False)
```

La fonction `est_un_entier_positif()` lît une chaîne de caractère. Cette fonction vérifie chacun des caractères et renvoie True si il n'y a que des chiffres et renvoie False si un autre caractère est trouvé.

EX3.py

```
def est_un_entier_positif(chaineInput:str)-> bool :
     """Cette fonction a pour entrée une chaîne de caractère.
     Cette fonction vérifie que chacun des caractère est
     un chiffre en appellant la fonction est_un_chiffre().
     Si cest le cas, renvoie True sinon False """

    i : int =0
    if chaineInput=="" :  # le cas de la chaine vide renvoie False.
        return False
    while i< len(chaineInput) :
        # pour tous les caractères de la chaine, on les test un par un.
        # on appelle la fonction est_un_chiffre qui renvoie un booléen
        # si le caractère de la chaine est bien un chiffre, on passe au suivant.
        # si le caractère de chaîne n'est pas un chiffre, on déduit que ChaineInput ne contient pas
        if not( est_un_chiffre(chaineInput[i]) ) :
            return False
        else :
            i=i+1
        # la boucle while est terminée
    # Si on sort de la boucle while alors tous les caractères sont des chiffres
    # la fonction peut renvoyer True.
    return True

assert est_un_entier_positif("12") ==True
assert est_un_entier_positif("")   ==False
assert est_un_entier_positif("a")  ==False
assert est_un_entier_positif("-12")==False
assert est_un_entier_positif("3.14")==False
```

5. .

EX3.py

```python
def demander_entier(question : str) -> int:
    """Cette fonction reçoit la chaîne question.
    La question est affiché à l'aide de input().
        Cette fonction vérifie que la réponse entrée à l'aide
        de la commande input() est bien un entier positif
        en utilisant la fonction est_un_entier_positif().
        Tant que (while en anglais) la réponse n'est pas un entier,
        la question est reposée à l'utilisateur."""
    reponse : str  = input (question)
    while not (est_un_entier_positif(reponse) ) :
        print ("Vous n'avez pas saisi un entier. Recommencez.")
        reponse  = input (question)
    # après le while la reponse contient uniquement des chiffres
    # donc il possible de la convertir en type entier.
    NbEntier : int = int (reponse)
    return NbEntier
```

Etant donné que la sortie de cette fonction dépend d'une entrée de l'utilisateur, il n'est pas judicieux de tester la fonction avec assert. Vous devez alors la tester en l'appellant plusieurs fois et en passant différentes valeurs pour la valider.

6. En complétant les f-string (vues en cours) :

EX3.py

```python
NbTotalEleve : int = demander_entier("Nombre total d'élèves : ")
TailleEquipe : int = demander_entier("Nombre de joueurs par équipe : ")
Nombre_d_equipe_complète = NbTotalEleve // TailleEquipe
Nombre_d_eleve_restant =  NbTotalEleve % TailleEquipe
# affichage du résultat rappelant les entrées et présentant les résultats
print(f"Le nombre total d'élèves : {NbTotalEleve}")
print(f"La taille d'une équipe :  {TailleEquipe}")
print(f"Le nombre d'équipes complètes :{Nombre_d_equipe_complète}")
print(f"Nombre d'élève(s) sans équipe :{Nombre_d_eleve_restant}")
```

Exercice 4 : Écrire une fonction qui calcule le quotient et le reste de la division euclidienne de 2 entiers. Le programme ne doit pas utiliser les fonctions python % ou // . Remarque : vous pouvez utiliser la fonction **int()** si besoin et judicieusement. La signature de la fonction est la suivante :

Correction : La commande assert permet de tester le code. Cela permet en lisant le code de voir ce que renvoie la fonction sur des exemples donnés. Si le résultat est différent de ce qui est écrit, un message d'erreur apparaît.

code.py

```
def divisionEuclidienne(a: int , b:int)-> tuple[int,int] :
    quotient : int = int(a/b)
    reste : int    = a - quotient * b
    return quotient, reste

assert( divisionEuclidienne(8 , 4) == (2,0)  )
assert( divisionEuclidienne(9 , 4) == (2,1)  )
```

Exercice 5 : Puzzle de Parson.

Remettre les commandes dans un ordre permettant à cette fonction de renvoyer le maximum parmi 4 entiers en Python.

code.py

```
return m
m = b
m = c
m = d
m : int = a
def max4(a : int ,b : int ,c :int ,d : int ) -> int :
if b > m :
if c > m :
if d > m :
```

code.py

```
def max4(a : int ,b : int ,c :int ,d : int ) -> int :
    m : int = a
    if b > m :
        m = b
    if c > m :     # l'ordre des test peut être dans un autre ordre
       m = c
    if d > m :
        m = d
    return m
```

Exercice 6 : Examiner la série de commandes du programme ci-dessous et prédire (SANS TAPER LE CODE DANS UN PREMIER TEMPS, A FAIRE DE TETE!) le résultat de l'exécution du script.

code.py

```
x : int =1
print(x)
y: int = 2
x=x+y
y=x**y
print(y)
print(x,y)
```

SHELL

```
>>>
1
9
3 9
```

Exercice 7 : Rédigez un programme qui calcule le prix d'une commande de soda pour un festival. Les trois valeurs suivantes sont représentées par les variables :

- ***nbr*** : entier désignant le nombre de fûts commandés
- ***prix*** : prix unitaire d'un fût.
- ***reduc*** : coefficient (entre 0 et 1) représentant la réduction dont bénéficie le client.

Le programme affiche le montant de la facture : $\boxed{m = nbr \times prix \times reduc}$.

1. Affecter les variables comme indiqué ci-dessus dans le cas dune commande de 27 fûts dont le prix unitaire est de 22,95 euros pour un client bénéficiant de 5 % de réduction.

EX7.py

```
nbr : int = 27
prix : float = 22.95
reduc : float = 0.95
m : float = nbr * prix * reduc

# ecrire ci-après l'affichage du montant de la facture.
print(f"Le montant de la facture {m})
```

2. Compléter le programme pour demander à l'utilisateur le nombre de fûts (qui doit être un nombre entier) et le prix arrondi (également un nombre entier pour simplifier l'exercice). Vous pouvez bien sûr ré-utiliser les fonctions de l'EX3.

EX3.py

```
Recopier ICI la fonction   est_un_chiffre()
Recopier ICI la fonction  est_un_entier_positif()
Recopier ICI la fonction  demander-entier()
# résoudre l'exercice en appellant les fonctions pour définir les entrées.
nbr : int     = demander_entier("Entrer le nombre de fûts : ")
prix : float  = demander_entier("Entre le prix (nombre entier !!)  :")
# traitement des données:
reduc : float = 0.9
m : float = nbr * prix * reduc
# affichage des résultats.
print(f"Le montant de la facture pour {nbr} fûts à {prix} euros"
 "avec réduction de {reduc:.2f}  est de {montant:.2f} euros")
# l'affichage du print se fait sur 2 lignes dans ce code
# le code reste ainsi lisible dans l'éditeur.
```

Exercice 8 :

Initialise deux entiers : a = 0 et b = 10 .

1. Écrit une première boucle affichant et incrémentant la valeur de a tant quelle reste inférieure à celle de b.
2. Écrit une deuxième boucle décrémentant la valeur de b et affichant sa valeur si elle est impaire. Boucler tant que b nest pas nul.

code.py

```
a : int = 0
b : int = 10

while a < b:
    print(f"a = {a}")
    a =a + 1

while b > 0:
    b = b - 1
    if b % 2 != 0:  # Vérifie si b est impair
        print(f"b = {b}")
```

Exercice 9 :

Écrire un script qui demande un entier **n** et affiche ensuite à l'aide d'une boucle **while**, tous les entiers impairs inférieurs à **n**.

R

```
ecopier ICI la fonction  est_un_chiffre()
 Recopier ICI la fonction  est_un_entier_positif()
 Recopier ICI la fonction  demander-entier()

def demander_entier(nill):
    return int(input(nill))

n   : int  = demander_entier("Entrez un entier positif  n : ")

print(f"Les entiers impairs inférieurs à {n} sont :")
k : int = 1
while k < n:
    if k % 2 != 0:
        print(k , end=' - ') # ecriture en ligne avec - et sans retour ligne
    k=k+ 1
```