Calculatrices et **documents non autorisés**.

Reportez le numéro de votre groupe sur votre copie (-1 point si pas fait). Ce sujet de **3 pages** comprend des exercices indépendants. Au sein d'un même exercice, il est demandé d'utiliser des appels aux fonctions écrites dans les questions précédentes. Ceci même si vous n'avez pas donné leur code et uniquement lorsque c'est pertinent. Toute fonction doit être réalisée en Python. **Les annotations de type doivent être données**, mais **la documentation n'est pas demandée**, sauf si la question le précise explicitement. **Le barème est donné à titre indicatif.**

Vous **ne devez pas** utiliser les fonctions prédéfinies Python `count`, `split`, `join`, `del`, `pop`, `map`, `sum`, `min`, `max`. D'une manière générale vous ne devez utiliser aucune fonction ou instruction qui n'aurait pas été vue pendant les enseignements.

Vous ne devez pas utiliser de boucle while, ni de récursivité. L'utilisation de range n'est, de même, pas nécessaire.

1 Jeu vidéo (13 pts)

On s'intéresse aux fonctionnalités d'un jeu dans lequel le joueur pilote un personnage qui doit attraper divers objets.

Le jeu est organisé en niveaux. Dans le niveau le plus simple, le joueur doit attraper des étoiles et trouver un diamant. Le jeu calcule un nombre de points en fonction du nombre d'étoiles attrapées. Trouver le diamant apporte un bonus. Le calcul est le suivant :

- entre 0 et 10 étoiles trouvées : 2 points par étoile, bonus de 20 points si le diamant est trouvé ;
- au delà de 11 étoiles trouvées : 3 points par étoile, bonus de 10 points si le diamant est trouvé ;
- nombre d'étoiles multiple de 9 — cas particulier — : 10 points par étoile, pas de bonus.

1. Écrire une fonction `nb_points` qui prend en paramètre un entier positif ou nul `nb_et` (nombre d'étoiles) et un booléen `diamant_trouve` (diamant trouvé) et qui renvoie le nombre de points attribué par le jeu. Par exemple : `nb_points(15, False)` vaut 45, `nb_points(15, True)` vaut 55 et `nb_points(18, True)` vaut 180.

Dans un niveau supérieur, le joueur doit attraper des étoiles et des pièces de monnaie. Le jeu calcule le ratio entre les pièces et les étoiles attrapées (ou entre les étoiles et les pièces attrapées) pour faire des statistiques.

On souhaite écrire un prédicat `est_ratio_sup_seuil` qui prend en paramètre 2 entiers positifs `x` et `y` tels que `x` est inférieur ou égal à `y` et `y` est non nul, ainsi qu'un paramètre flottant `seuil` compris entre 0 et 1 (bornes exclues). Ce prédicat renvoie vrai si et seulement si le ratio $\frac{x}{y}$ est supérieur strictement au seuil.

Par exemple `est_ratio_sup_seuil(9, 10, 0.8)` vaut vrai et `est_ratio_sup_seuil(2, 10, 0.8)` vaut faux.

2. Donner :
   - la précondition de ce prédicat ;
   - le code du prédicat `est_ratio_sup_seuil`.

En utilisant le prédicat précédent, on souhaite écrire un prédicat `est_ratio_proche_1` qui prend en paramètre 2 entiers positifs ou nuls `nb_et` (nombre d'étoiles) et `nb_pc` (nombre de pièces) et qui renvoie vrai pour chacun des cas suivants et faux sinon :

- soit le nombre d'étoiles est strictement plus grand que le nombre de pièces et le ratio $\frac{nbpieces}{nbetoiles}$ est strictement supérieur à 0.8,
- soit le nombre de pièces est strictement plus grand que le nombre d'étoiles et le ratio $\frac{nbetoiles}{nbpieces}$ est strictement supérieur à 0.8,
- soit le nombre d'étoiles et de pièces sont égaux.

3. Écrire :
   - la définition d'une constante globale qui représente un seuil valant `0.8` ;
   - le prédicat `est_ratio_proche_1`, en utilisant cette constante.

Dans une version du jeu à 2 joueurs, chaque joueur a un pseudo :

- le premier joueur de pseudo `pseudo1` doit attraper les étoiles,
- le second joueur de pseudo `pseudo2` doit attraper les pièces,
- le pseudo du binôme est : `pseudo1` avec un caractère `*` avant et après, suivi de `pseudo2` avec un caractères `$` avant et après.

4. Écrire une fonction `pseudos_binomes` qui prend en paramètre 2 listes de pseudos représentant 2 groupes de joueurs, et renvoie la liste des pseudos des binômes possibles, le premier groupe attrapant les étoiles et le second les pièces. Par exemple `pseudos_binomes(["lou", "hey"], ["mi", "don"])` vaut `["*lou*$mi$", "*lou*$don$", "*hey*$mi$", "*hey*$don$"]`.

Le temps mis pour réaliser les niveaux est chronométré.

5. Écrire une fonction `meilleur_temps` qui prend en paramètre une liste non vide de temps (entiers) et renvoie le meilleur temps (c'est-à-dire le plus petit). Par exemple : `meilleur_temps([20223, 502, 1067])` vaut `502`.

Les actions du joueur sont représentées par des caractères. Par exemple sauter est représenté par le caractère `|`, avancer vers la droite est représenté par le caractère `>`. La séquence des actions est stockée dans une chaîne.

6. On souhaite écrire une fonction `supprimer_actions` qui prend en paramètre une chaîne de caractères `actions_joueur` et une chaîne de caractères non vide `actions_suppr`. Cette fonction renvoie une nouvelle chaîne de caractères qui contient les caractères de `actions_joueur` dans le même ordre, en supprimant ceux qui apparaissent dans `actions_suppr`. Par exemple `supprimer_actions(">>>**|>**|$|$o", ">|")` vaut `"****$$o"`.
   a. Donner des tests pertinents pour cette fonction,
   b. Écrire le code de cette fonction.

On souhaite analyser les actions effectuées par un joueur entre 2 sauts (caractère `|`).

7. On souhaite écrire une fonction qui prend en paramètre une chaîne de caractères non vide représentant les actions du joueur et renvoie une liste de chaînes. Cette liste contient les actions réalisées par le joueur de part et d'autre d'un saut. Par exemple on obtiendra `['', '***','', '>>', '$']` pour les actions `'|***||>>|$'`. Donner le code **et la documentation** de cette fonction, **en incluant comme seul test l'exemple précédemment donné**.

2 Barre de progression (4.5 pts)

On veut construire une bibliothèque de barres de progression en mode texte, comme celles qui indiquent dans un terminal la part réalisée et la part restant à réaliser du téléchargement d'un fichier.

Ces barres de progression reposent sur le calcul d'une chaîne de caractères représentant la progression, comme suit :

- on connaît le pourcentage `pourcentage` (compris entre 0 et 100 inclus) de la tâche déjà réalisée et le nombre de caractères total `longueur` (strictement positif) que peut occuper la chaîne représentant la progression ;
- on utilise un caractère `car_realise` pour représenter la tâche déjà réalisée, et un autre caractère `car_restant` pour représenter la tâche restant à réaliser ;
- la chaîne représentant la progression contient `n` caractères `car_realise` puis `m` caractères `car_restant` avec `n + m` valant `longueur` ;
- `n` est la partie entière[1] de `pourcentage`% de `longueur`. Par exemple 39% de 20 vaut approximativement `7.8`, partie entière 7. Par exemple 40% de 20 vaut 8, partie entière 8.

On aura par exemple :

`pourcentage`	`longueur`	`car_realise`	`car_restant`	`n`	`m`	chaîne représentant la progression
40	20	`'='`	`'.'`	8	12	`"========............"`
39	20	`'+'`	`'-'`	7	13	`"+++++++-------------"`

On utilisera la fonction `floor` du module `math` pour le calcul de partie entière. Par exemple `floor(7.8)` vaut `7`.

On souhaite programmer deux fonctions qui calculent 2 barres de progression basées sur le principe ci-dessus. Mais ces barres utilisent des caractères et des formats différents pour décorer la chaîne de progression. Pour respecter les bonnes pratiques de programmation vues en cours **vous devez écrire une fonction intermédiaire** (ou auxilliaire), donner son code mais pas sa documentation. Si vous ne voyez pas comment écrire cette fonction, faire la question suivante sans.

8. Écrire le code nécessaire aux deux fonctions `barre1` et `barre2` qui ont le comportement suivant :

[1] Dans cette phrase le symbole % désigne le pourcentage mathématique et non l'opérateur modulo de Python.

```
>>> barre1(40, 20)
'[40%][========............]'
>>> barre1(39, 20)
'[39%][=======.............]'
>>> barre2(40, 20)
'Progress: |++++++++------------| 40% complete'
>>> barre2(39, 20)
'Progress: |+++++++-------------| 39% complete'
```

On souhaite écrire un programme principal qui permet d'afficher un exemple de barre de progression calculée par la fonction `barre1` pour 60%. Ce `main` demande de saisir au clavier une longueur puis calcule et affiche la barre de progression.

On ne vérifiera pas la saisie de l'utilisateur. On aura par exemple :

```
Entrer une longueur : 20
[60%][============........]
```

9. Sans écrire d'autres fonctions, écrire une fonction principale `main` qui code le comportement décrit ci-dessus.

3 Lecture de code (2.5 pts)

On donne les 2 fonctions mystère suivantes :

```
def foo(x:int, y:int) -> int:
    """ Précondition : x et y > 0"""
    return x//y + x%y

def bar(liste:list[int]) -> list[int]:
    """ Précondition : liste non vide et contient des entiers > 0
    """
    p = liste[0]        #1
    res = [p]           #2
    for e in liste[1:]: #3
        f = foo(p, e)   #4
        res.append(f)   #5
        res.append(e)   #6
        p = e           #7
    return res          #8
```

10. Donner la valeur de :
    a. `foo(6, 2)`
    b. `foo(15, 6)`
11. On considère l'appel de fonction `bar([15, 6, 2])`.
    a. Donner la valeur de `liste[0]` et `liste[1:]`.
    b. Recopier et remplir le tableau de la mémoire ci-dessous de tel sorte qu'il reflète l'exécution de `bar([15, 6, 2])`. **Le nombre de colonnes est donné à titre indicatif**.

e							
p							
f							
res							
liste							

    c. Entourer clairement dans le tableau le résultat de l'appel.