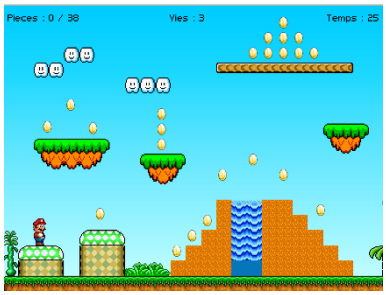
Toutes les réponses sont à faire sur le fichier : jeu_plateforme_elev.py



Dans un jeu de plate-forme à deux dimensions, un personnage se déplace sur un paysage parsemé d'obstacles modélisé par une succession de paliers sur lequel il doit sauter. Le personnage se déplace de la gauche vers la droite.

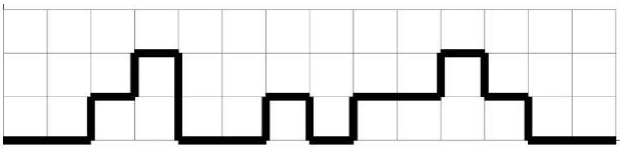
Pour cet exercice, le personnage peut **sauter d'une hauteur de 1** et il ne peut pas **tomber d'une hauteur de plus de 2**. Pour que le niveau soit faisable il doit donc respecter ces règles. Le niveau est modélisé par une liste représentant la hauteur des paliers successifs : niveau = [0, 1, …]

La difficulté d'un niveau se calcule en cumulant les valeurs absolues des écarts entre la hauteur d'un pallier et la hauteur du pallier suivant :

$$d = \sum_{i=0}^{i=len(niveau)-1} abs(niveau[i] - niveau[i+1])$$

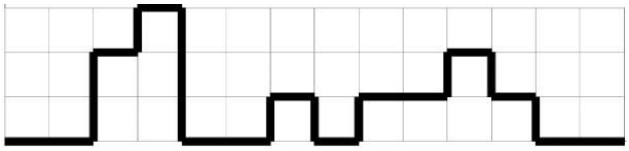
PARTIE A :

Exemple 1 : niveau 1 conforme



Le niveau ci-contre est conforme puisque les sauts sont d'une hauteur de 1 et les chutes n'excèdent pas une hauteur de 2. Sa difficulté est de : 0+1+1+2+0+1+1+1+0+1+1+1 soit 10. Il est modélisé par la liste : **niveau1 = [0, 0, 1, 2, 0, 0, 1, 0, 1, 1, 2, 1, 0, 0]**

Exemple 2 : niveau 2 non conforme



Le niveau ci-dessous est non-conforme, en effet on passe d'une hauteur zéro à une hauteur 2 sans intermédiaire et le personnage peut tomber d'une hauteur de 3 (donc supérieure à 2). Il est modélisé par la liste : **niveau2 = [0, 0, 2, 3, 0, 0, 1, 0, 1, 1, 2, 1, 0, 0]**

On souhaite modifier le niveau 2 pour qu'il soit conforme, pour cela il faut modifier la liste associée en exécutant la suite d'instructions suivante :

a. Remplacer les ..... par le code idoine pour rendre le niveau conforme aux exigences.

niveau2[.....] = ...

niveau2[.....] = ...

Avant de l'intégrer au jeu, on souhaite vérifier la conformité d'un niveau. Pour cela, on code une fonction **conforme(niveau)**, qui prend en paramètre une liste d'entiers qui représente un niveau et renvoie **True** si le niveau est conforme, **False** sinon.

b. Remplacer les ..... de la fonction suivante pour que les assertions passent.

```
def conforme(niveau):
    """ vérifie la conformité d'un niveau : un index et son suivant doivent
        avoir un écart >= -1 et <= 2 pour que le niveau soit conforme
        param : niveau(list) liste des hauteurs successives du niveau
        return : result(bool) résultat de la conformité
    """
    assert type(niveau) == list,"le type() de 'niveau' n'est pas conforme"
    result = True
    for ..... (len(niveau)-1) :
        ecart = niveau[i] - .....
        if  ecart < -1 ..... ecart .....:
            result = .....
    return result
assert conforme([0, 0, 1, 0, 1, 2, 2, 3, 3, 4, 2, 1, -1, -2, -2]) == True,"/!\ defaut conforme()"
assert conforme([0, -2, -1, -1, -1, 0, 0, -2, -2, -1, -1, 0, -2, -3, -2]) == True,"/!\ defaut conforme()"
assert conforme([0, -1, -1, -1, 0, 1, 3, 1, 2, 2, 0, -1, 0, 1, 1]) == False,"/!\ defaut conforme()"
assert conforme([0, 0, 1, 0, 1, 2, 2, 3, 3, 4, 1, 1, -1, -2, -2]) == False,"/!\ defaut conforme()"
```

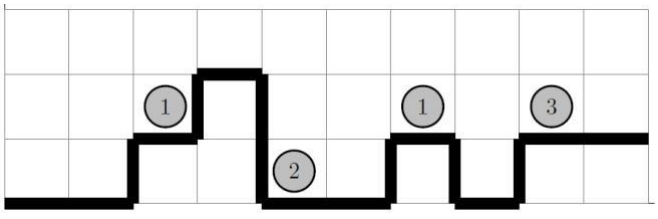
c. ***Expliquer*** *pourquoi dans la ligne "* for ..... (len(niveau)-1):*" on a mis len(niveau)-1 et pas len(niveau).*

d. ***Proposer*** *une fonction difficulte(niveau) qui prend en paramètre une liste représentant un niveau conforme, et renvoie la difficulté de ce niveau telle que définie plus haut.* ***Rédiger*** *une "doc string". Votre proposition de fonction doit passer les assertions suivantes :*

```
assert difficulte(niveau1) == 10, "/!\ defaut difficulte()"
assert difficulte([0, 1, 2, 0, 1, 2, 2, 3, 3, 4, 2, 1, -1, -2, -2]) == 14, "/!\ defaut difficulte()"
assert difficulte([0, -2, -3, -2, -1, 0, 0, -2, -3, -2, -1, 0, -2, -3, -2]) == 16, "/!\ defaut difficulte()"
```

PARTIE B

On améliore maintenant un peu le jeu en disposant des pièces sur le parcours. Chaque pièce a une valeur, le joueur marque les points correspondants lorsque le personnage ramasse une pièce. Chaque **case** du parcours est maintenant représentée par une liste de deux éléments : **(hauteur, valeur_piece)**. S'il n'y a pas de pièce sur la case, la valeur associée à la pièce sera nulle.



Exemple 3 : niveau avec pièces

Le niveau3 proposé ci-contre est conforme, de difficulté 7 et il est maintenant représenté par :

niveau3 = [(0,0), (0,0), (1,1), (2,0), (0,2), (0,0), (1,1), (0,0), (1,3), (1,0)]

e. *On veut modifier la valeur d'une pièce dans le niveau donné en exemple ci-dessus.* ***Compléter*** *l'instruction permettant de changer la pièce de valeur 3 en une pièce de valeur 1.*

niveau3[.....

On suppose que le joueur atteint la case **n** et que la position la plus à gauche correspond à **n = 0.**

f. ***Proposer*** *une fonction score(niveau, n) avec en paramètre le niveau3 ainsi qu'une position horizontale n et qui calcule le score maximum qu'il est possible d'atteindre si le joueur ne se fait pas éliminer et qu'il ramasse toutes les pièces.* ***Rédiger*** *une "doc string". Les assertions suivantes doivent passer.*

```
assert score(niveau3, 0) == 0, "/!\ defaut score()"
assert score(niveau3, 9) == 5, "/!\ defaut score()"
assert score(niveau3, 6) == 4, "/!\ defaut score()"
```

g. ***Proposer*** *une assertion en début de votre fonction qui vérifie que n n'excède pas la longueur du niveau.*

Les meilleurs scores des joueurs sont mémorisés dans un dictionnaire de la forme **meilleurs_scores** = {'Alice': 18, 'Bob': 16, 'Tima': 14 , 'Alan': 20}. Alice vient de battre son record, elle vient de réaliser un score de 32.

h. ***Donner*** *l'instruction permettant de mettre à jour le dictionnaire meilleurs_scores.*

On considère la fonction définie ci-dessous qui prend en paramètre un dictionnaire **records** comme **meilleurs_scores** cité en exemple ci-dessus. Les clés sont les noms et les valeurs associées les meilleurs scores de chacun.

i. ***Analyser*** *cette fonction et* ***dire*** *(sans interpréteur de code) ce que renvoie cette fonction si records est meilleurs_scores.*

```
>>> mystere(meilleurs_scores)
?
```

```
def mystere(records) :
    nbre1 = 0
    for nom in records :
        if records[nom] > nbre1 :
            nbre1 = records[nom]
            resultat1 = nom
    nbre2 = nbre1
    for nom in records :
        if records[nom] < nbre2 :
            nbre2 = records[nom]
            resultat2 = nom
    return resultat1, resultat2
```

j. ***Proposer*** *une fonction bon_niveau(records) qui prend en paramètre un dictionnaire records identique à meilleurs_scores et qui renvoie une liste constituée des noms des joueurs ayant un score supérieur ou égal à 20.* ***Rédiger*** *une "doc string". Les assertions suivantes doivent passer.*

```
assert bon_niveau({'Alice': 12, 'Bob': 18, 'Tima': 63, 'Alan': 20}) == ['Tima', 'Alan'], "/!\ defaut bon_niveau()"
assert bon_niveau({'Greg': 16, 'Bab': 26, 'Tino': 12, 'Gab': 17}) == ['Bab'], "/!\ defaut bon_niveau()"
```