Programmation - WhatIsThisShape - 150 points

Kévin DUVERGER

Table des matières

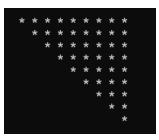
1 Résolution WhatIsThisShape :

Tentons encore de nous connecter au serveur pour voir ce qu'il nous envoie :

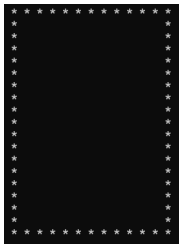
```
PS D:\ALL\Team CRYPTIS\2025\creationCTFInitiation\Programming\WhatIsThisShape> ncat 146.59.227.136 23408
libnsock ssl_init_helper(): OpenSSL legacy provider failed to load.

Welcome to WhatIsThisShape dear challenger !
I will send you 30 images and each time, you have to give me the name of the shape that appears as fast as possible
Make sure your terminal is at least 80 characters wide to do this challenge
Ready ?? 3.. 2.. 1.. Go !

Shape #1 :

                    * * * * * * * * * * * * * * * * * * * *
                    *                                     *
                    *                                     *
                    *                                     *
                    *                                     *
                    *                                     *
                    *                                     *
                    *                                     *
                    *                                     *
                    *                                     *
                    *                                     *
                    *                                     *
                    *                                     *
                    *                                     *
                    *                                     *
                    *                                     *
                    * * * * * * * * * * * * * * * * * * * *


Please tell me what shape it is : t
Your choice is wrong, this shape is a square
```

Il semblerait donc que nous allons devoir analyser le nom d'une forme à partir d'une image ASCII pour le retourner au serveur. Pour l'instant nous ne connaissons pas le nom des formes mais en envoyant une entrée invalide on peut les récupérer.

Voici quelques unes des autres formes que l'on peut recevoir :

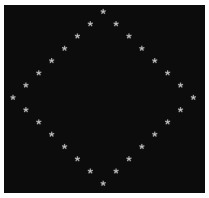


(a) `triangle`

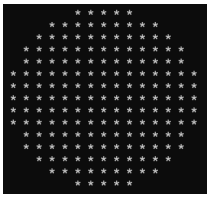


(b) `rectangle`

(c) `oval`



(d) `diamond`



(e) `circle`

Il existe aussi des versions remplies de ces formes auquel cas il faudra préfixer le nom par `filled` (ci-dessus, vous pouvez voir que le triangle rectangle et le cercle sont dans ce cas).

Format des "images" dans le programme. Pour stocker les "images" qui nous sont envoyées par le serveur, l'idée sera de créer un tableau bi-dimensionnel : pour le remplir il faudra splitter l'entrée par ligne, puis prendre un caractère sur 2 (car il y a des espaces entre chaque caractère pour avoir un look un peu plus carré). Voici le code qui permet de faire cela :

```python
def createBoardFromLineList(stringList) :
    boardSizeY = len(stringList)
    boardSizeX = (len(stringList[0]) + 1) // 2
    board = []
    for y in range(0, boardSizeY) :
        board.append([ ])
        for x in range(0, boardSizeX) :
            board[-1].append(stringList[y][x * 2])
    return board
```

Détecter si une forme est remplie. La première façon d'avoir moins de cas à considérer est de toujours se réduire au cas "forme vide". L'idée pour détecter si une forme est "remplie" est de voir s'il existe des * qui ont 4 voisins qui sont aussi des * : si c'est le cas la forme est remplie et tous les pixels vérifiant cela pourront être supprimés pour vider la forme :

```python
def unfillShape(board) :
    boardSizeY = len(board)
    boardSizeX = len(board[0])
    # First count the number of activated neighbours per cell
    neighborCountList = []
    for y in range(0, boardSizeY) :
        neighborCountList.append([])
        for x in range(0, boardSizeX) :
            neighborCountList[-1].append(0)
            if x > 0 and board[y][x - 1] == '*' : neighborCountList[-1][-1] += 1
            if y > 0 and board[y - 1][x] == '*' : neighborCountList[-1][-1] += 1
            if x < boardSizeX - 1 and board[y][x + 1] == '*' : neighborCountList[-1][-1] += 1
            if y < boardSizeY - 1 and board[y + 1][x] == '*' : neighborCountList[-1][-1] += 1
    # Then, remove the ones having 4
    unfilled = False
    for y in range(0, boardSizeY) :
        for x in range(0, boardSizeX) :
            if neighborCountList[y][x] == 4 :
                unfilled = True
                board[y][x] = ' '
    return unfilled
```

Reconnaître les formes. Nous pouvons maintenant passer à la partie la plus complexe qui sera de reconnaître la forme parmi les 6 possibilités qui ont été vues au départ. Pour débuter cette reconnaissance, on peut commence par récupérer les 4 "coins" d'une forme : les 2 pixels les plus hauts puis aux extrémités à gauche / à droite, les 2 pixels les plus bas puis aux extrémités à gauche / à droite :

```python
def findTopLeftActivePixel(board, boardSizeX, boardSizeY) :
    for y in range(0, boardSizeY) :
        for x in range(0, boardSizeX) :
            if board[y][x] == '*' :
                return [ x, y ]

def findTopRightActivePixel(board, boardSizeX, boardSizeY) :
    for y in range(0, boardSizeY) :
        for x in range(boardSizeX - 1, -1, -1) :
            if board[y][x] == '*' :
                return [ x, y ]

def findBottomLeftActivePixel(board, boardSizeX, boardSizeY) :
    for y in range(boardSizeY - 1, -1, -1) :
        for x in range(0, boardSizeX) :
            if board[y][x] == '*' :
                return [ x, y ]

def findBottomRightActivePixel(board, boardSizeX, boardSizeY) :
    for y in range(boardSizeY - 1, -1, -1) :
        for x in range(boardSizeX - 1, -1, -1) :
            if board[y][x] == '*' :
                return [ x, y ]
```

Si les 2 pixels de chaque cas sont confondus, ce sera un losange. Si les x des pixels associés en haut / en bas sont les mêmes, et que les y des pixels sur 2 lignes sont les mêmes c'est probablement un rectangle / carré, mais ça pourrait aussi être un ovale et pour faire la différence on va continuer les lignes (qui ne devraient pas être présentes dans l'ovale). Ensuite, pour savoir si c'est un triangle rectangle il faut que 2 des 4 points soient confondus, et c'est la seule condition. Pour finir et tester si c'est un cercle / un ovale, on va récupérer la largeur de la forme (en récupérant le pixel avec le plus petit X et celui avec le plus grand) : si elles sont identiques c'est un cercle, sinon un ovale. Voici le code :

```python
def recognizeShape(board) :
    # Unfilling and getting the size
    filled = unfillShape(board)
```

```
    boardSizeY = len(board)
    boardSizeX = len(board[0])

    # Finding some special points to help us
    topLeftActivePixel = findTopLeftActivePixel(board, boardSizeX, boardSizeY)
    bottomLeftActivePixel = findBottomLeftActivePixel(board, boardSizeX, boardSizeY)
    topRightActivePixel = findTopRightActivePixel(board, boardSizeX, boardSizeY)
    bottomRightActivePixel = findBottomRightActivePixel(board, boardSizeX, boardSizeY)

    # Testing whether it is a diamond
    if topLeftActivePixel == topRightActivePixel and bottomLeftActivePixel == bottomRightActivePixel :
        height = bottomLeftActivePixel[1] - topLeftActivePixel[1]
        # Only testing the top half
        isDiamond = True
        point1, point2 = [ topLeftActivePixel[0], topLeftActivePixel[1] ], [ topLeftActivePixel[0], topLeftActivePixel[1] ]
        for i in range(0, height // 2) :
            point1 = [ point1[0] - 1, point1[1] + 1 ]
            point2 = [ point2[0] + 1, point2[1] + 1 ]
            if board[point1[1]][point1[0]] != '*' or board[point2[1]][point2[0]] != '*' :
                isDiamond = False
                break
        # Returning
        if isDiamond :
            return ("filled " if filled else "") + "diamond"

    # Testing whether it is a rectangle / square
    if topLeftActivePixel[1] == topRightActivePixel[1] and bottomLeftActivePixel[1] == bottomRightActivePixel[1] : # Testing the y's
        if topLeftActivePixel[0] == bottomLeftActivePixel[0] and topRightActivePixel[0] == bottomRightActivePixel[0] : # Testing the x's
            width  = topRightActivePixel[0] - topLeftActivePixel[0]
            height = bottomLeftActivePixel[1] - topLeftActivePixel[1]
            # Testing the lines
            isRectangle = True
            for x in range(0, width) :
                if board[topLeftActivePixel[1]][topLeftActivePixel[0] + x] != '*' :
                    isRectangle = False
                if board[bottomLeftActivePixel[1]][topLeftActivePixel[0] + x] != '*' :
                    isRectangle = False
            for y in range(0, height) :
                if board[topLeftActivePixel[1] + y][topLeftActivePixel[0]] != '*' :
                    isRectangle = False
                if board[topLeftActivePixel[1] + y][topRightActivePixel[0]] != '*' :
                    isRectangle = False
            # Returning
            if isRectangle :
                if width == height :
                    return ("filled " if filled else "") + "square"
                else :
                    return ("filled " if filled else "") + "rectangle"

    # Testing whether it is a triangle rectangle
    triRectPoss1 = topLeftActivePixel == topRightActivePixel and topLeftActivePixel != bottomLeftActivePixel and topLeftActivePixel != bottomRightActivePixel and bottomLeftActivePixel != bottomRightActivePixel
    triRectPoss2 = topLeftActivePixel == bottomLeftActivePixel and topRightActivePixel != topLeftActivePixel and bottomRightActivePixel != topLeftActivePixel and topRightActivePixel != bottomRightActivePixel
    triRectPoss3 = topRightActivePixel == bottomRightActivePixel and topLeftActivePixel != topRightActivePixel and bottomLeftActivePixel != topRightActivePixel and topLeftActivePixel != bottomLeftActivePixel
    triRectPoss4 = bottomLeftActivePixel == bottomRightActivePixel and topLeftActivePixel != bottomLeftActivePixel and topRightActivePixel != bottomLeftActivePixel and topLeftActivePixel != topRightActivePixel
    if triRectPoss1 or triRectPoss2 or triRectPoss3 or triRectPoss4 :
        return ("filled " if filled else "") + "rectangle triangle"

    # Finally, it is either an oval or a circle
    minXActive, maxXActive = 100, 0
    for y in range(0, boardSizeY) :
        for x in range(0, boardSizeX) :
            if board[y][x] == '*' :
                if x < minXActive : minXActive = x
                if x > maxXActive : maxXActive = x
    width  = maxXActive - minXActive
    height = bottomLeftActivePixel[1] - topLeftActivePixel[1]
    if width == height :
```

```
        return ("filled " if filled else "") + "circle"
    else :
        return ("filled " if filled else "") + "oval"
```

Notez que ce n'est pas la seule façon de faire et qu'il y en a plein d'autres totalement valides !

Programme principal. Et pour finir, voici le programme principal qui gère la communication avec le serveur :

```
server = "146.59.227.136"
port = 23408

ma_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
try:
    ma_socket.connect((server, port))
except Exception as e:
    print("Problème de connexion", e.args)
    sys.exit(1)

for i in range(0, 30) :
    data = ma_socket.recv(16384).decode("utf-8")
    print(data)
    lineList = data.split("\n")
    board = createBoardFromLineList(lineList[6:-1] if i == 0 else lineList[1:-1])
    shapeName = recognizeShape(board)
    print(shapeName)
    ma_socket.sendall((shapeName + "\n").encode("utf-8"))
data = ma_socket.recv(8192).decode("utf-8")
print(data)

ma_socket.close()
```