# Memory Puzzle

Part 1

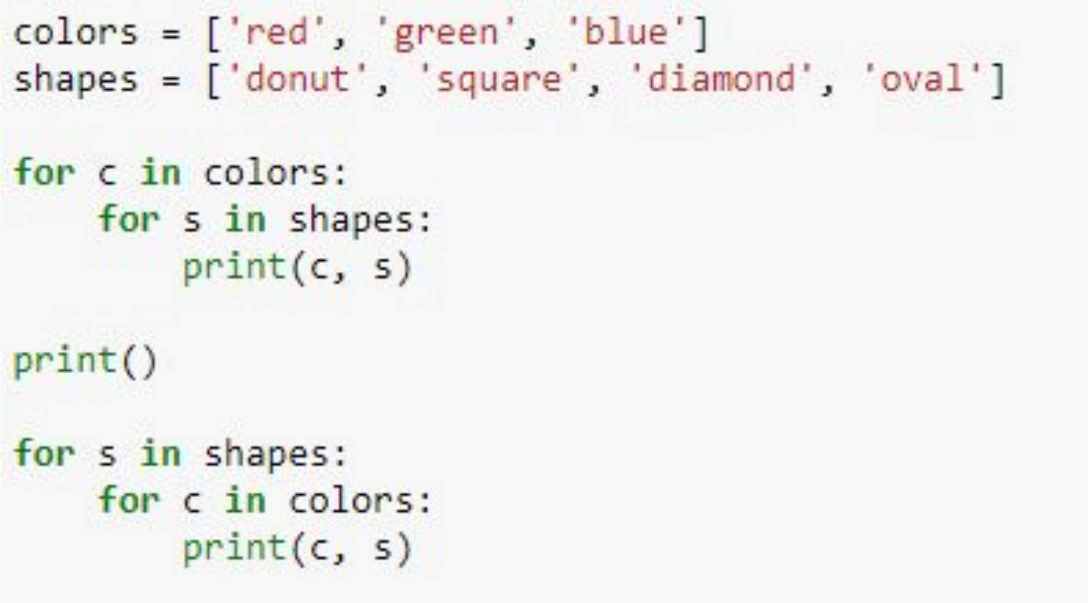
# Last time

- Pygame Primitive Drawing Functions
- Drawing code

# Today

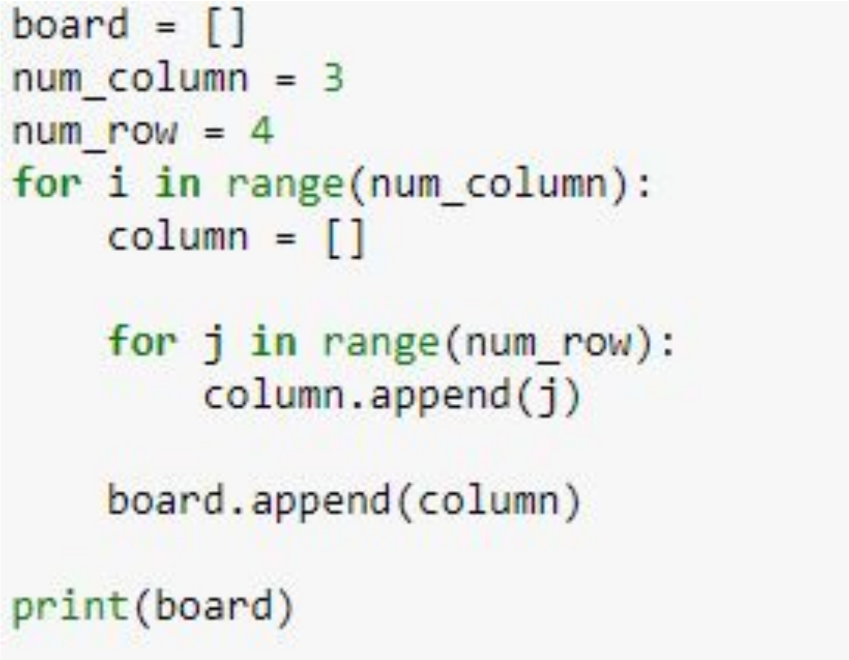
- Python advance
  - Nested for loop
  - List of list
  - Random shuffle and list operations
- Memory puzzle code
  - Main function

# Nested for loop



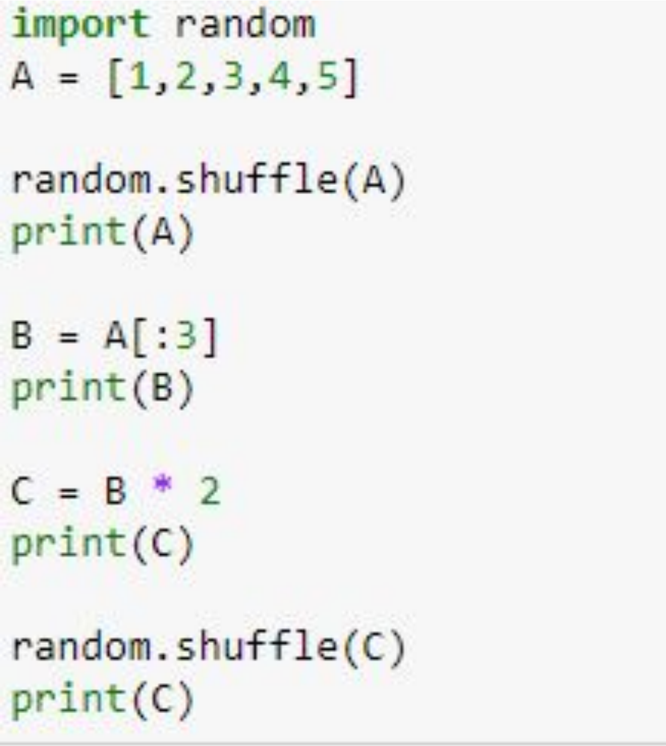
```
colors = ['red', 'green', 'blue']
shapes = ['donut', 'square', 'diamond', 'oval']

for c in colors:
    for s in shapes:
        print(c, s)

print()

for s in shapes:
    for c in colors:
        print(c, s)
```

# List of lists



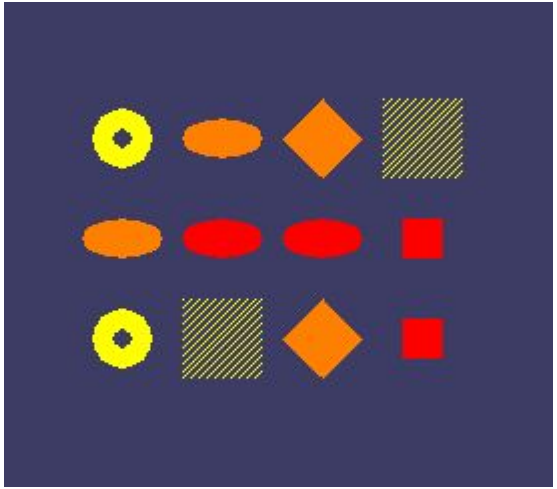
```
board = []
num_column = 3
num_row = 4
for i in range(num_column):
    column = []

    for j in range(num_row):
        column.append(j)

    board.append(column)

print(board)
```

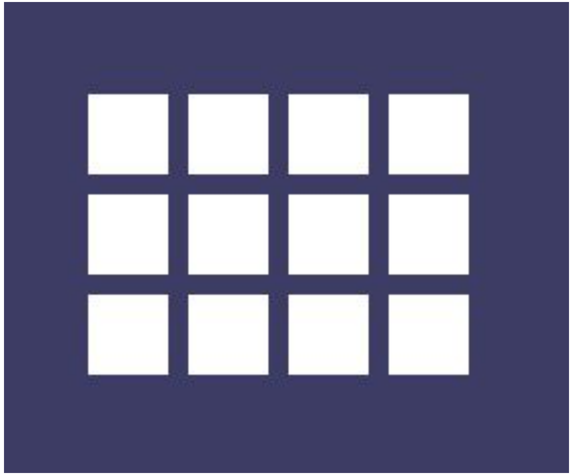
# Random Shuffle and list operations



```
import random
A = [1,2,3,4,5]

random.shuffle(A)
print(A)

B = A[:3]
print(B)

C = B * 2
print(C)

random.shuffle(C)
print(C)
```

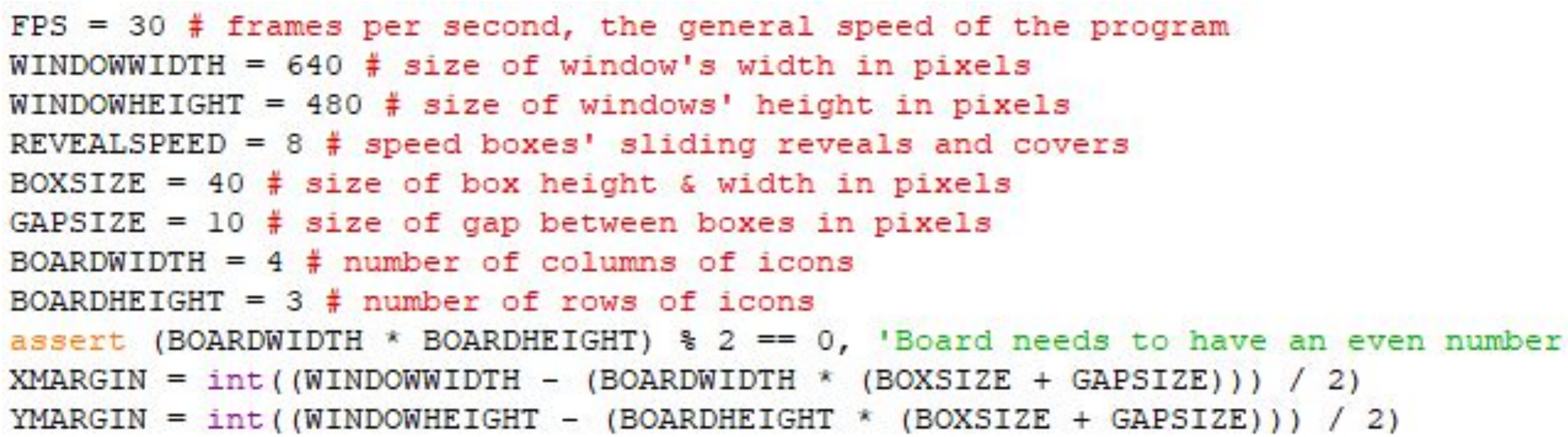
# Memory Puzzle Game



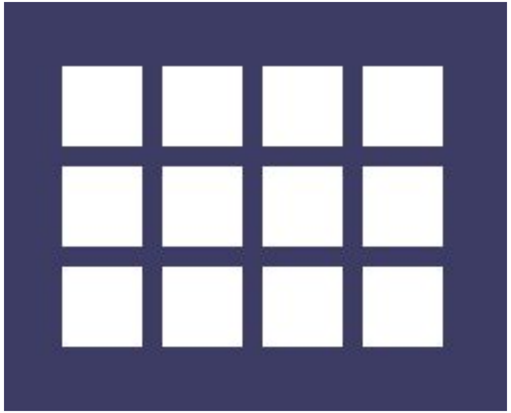


https://github.com/zhihongzeng2002/pythongame

# Game Parameters



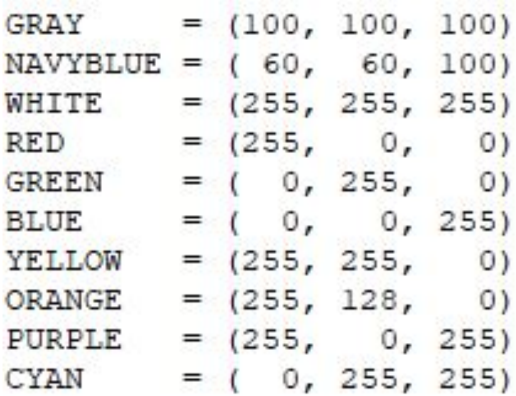
```
FPS = 30 # frames per second, the general speed of the program
WINDOWWIDTH = 640 # size of window's width in pixels
WINDOWHEIGHT = 480 # size of windows' height in pixels
REVEALSPEED = 8 # speed boxes' sliding reveals and covers
BOXSIZE = 40 # size of box height & width in pixels
GAPSIZE = 10 # size of gap between boxes in pixels
BOARDWIDTH = 4 # number of columns of icons
BOARDHEIGHT = 3 # number of rows of icons
assert (BOARDWIDTH * BOARDHEIGHT) % 2 == 0, 'Board needs to have an even number
XMARGIN = int((WINDOWWIDTH - (BOARDWIDTH * (BOXSIZE + GAPSIZE))) / 2)
YMARGIN = int((WINDOWHEIGHT - (BOARDHEIGHT * (BOXSIZE + GAPSIZE))) / 2)
```



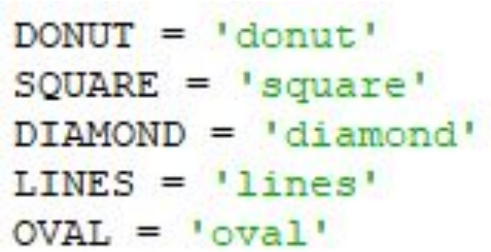
# Memory Puzzle

Part 2

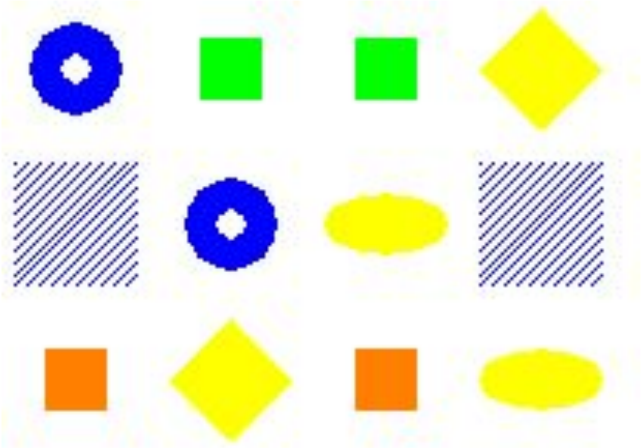
# Game Parameters



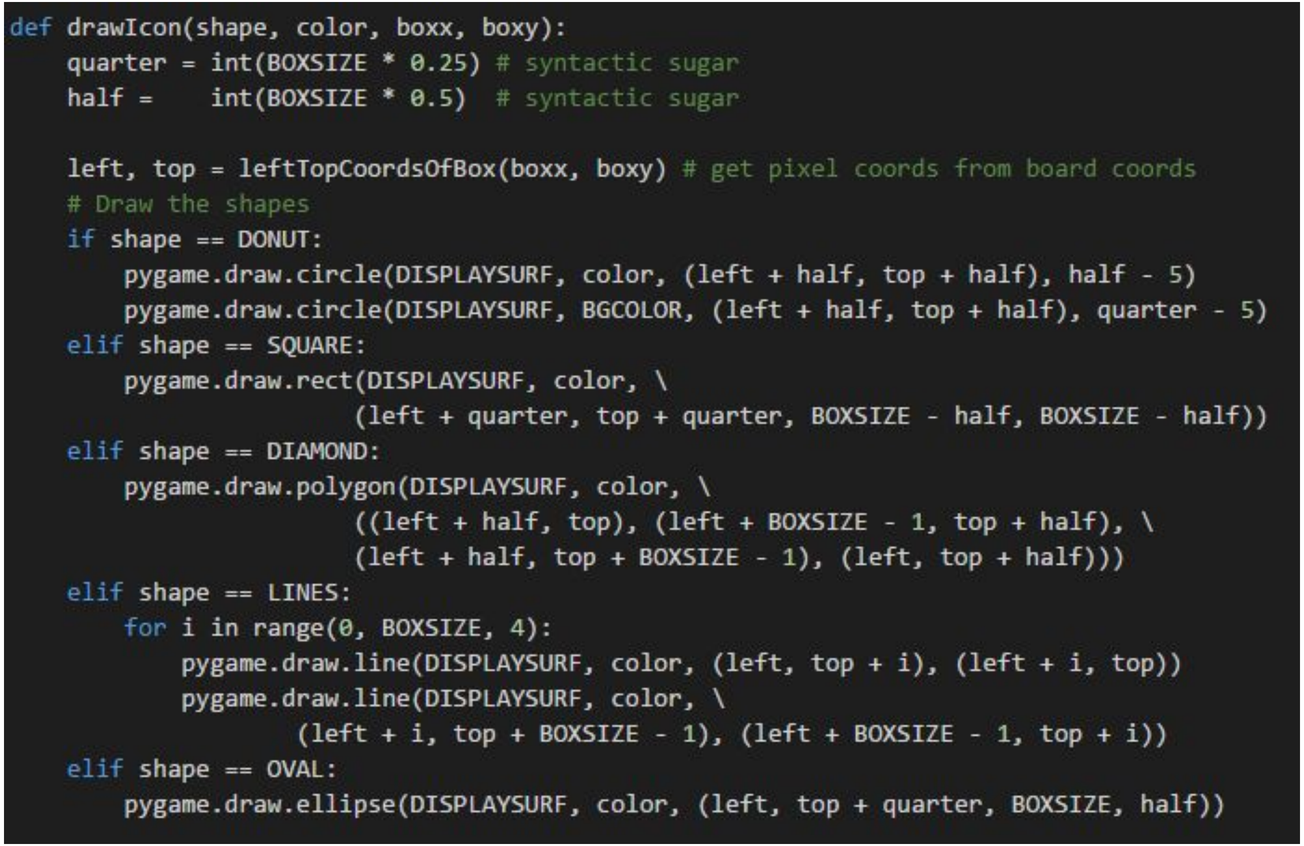
```
GRAY     = (100, 100, 100)
NAVYBLUE = ( 60,  60, 100)
WHITE    = (255, 255, 255)
RED      = (255,   0,   0)
GREEN    = (  0, 255,   0)
BLUE     = (  0,   0, 255)
YELLOW   = (255, 255,   0)
ORANGE   = (255, 128,   0)
PURPLE   = (255,   0, 255)
CYAN     = (  0, 255, 255)
```



```
DONUT = 'donut'
SQUARE = 'square'
DIAMOND = 'diamond'
LINES = 'lines'
OVAL = 'oval'
```

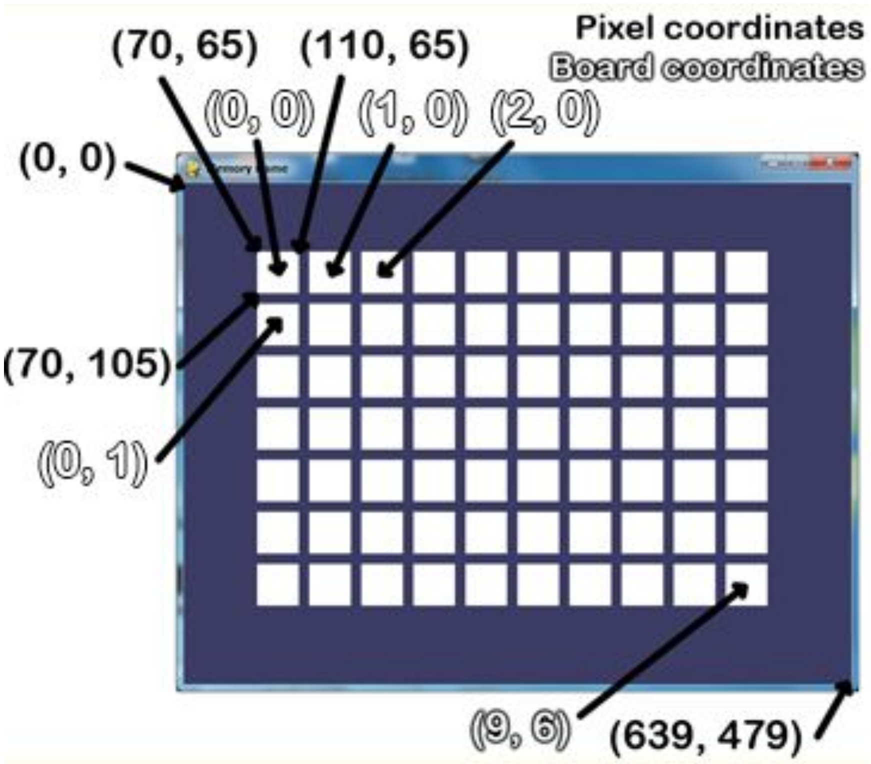


# Draw Icon

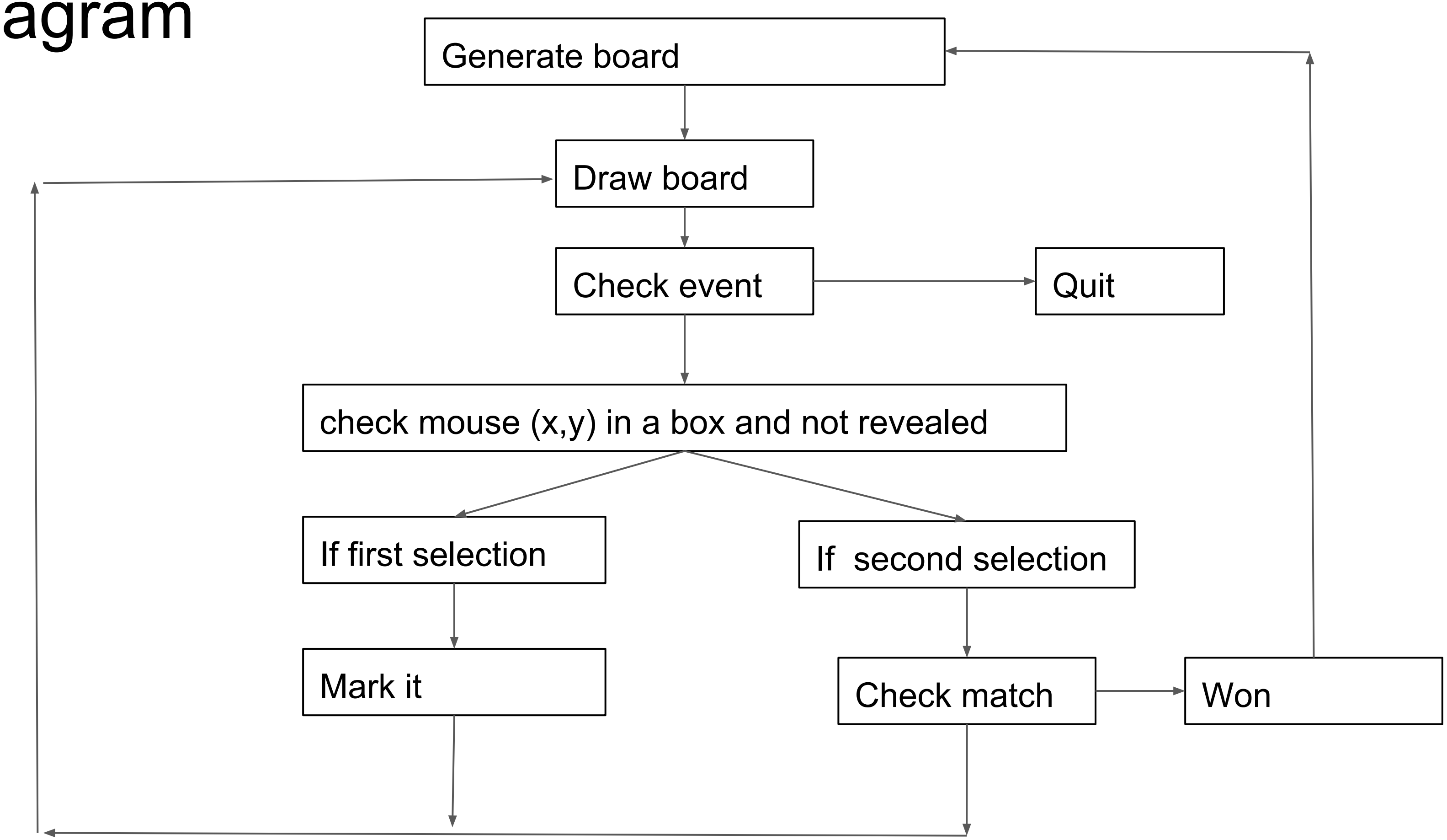


```python
def drawIcon(shape, color, boxx, boxy):
    quarter = int(BOXSIZE * 0.25) # syntactic sugar
    half =    int(BOXSIZE * 0.5)  # syntactic sugar

    left, top = leftTopCoordsOfBox(boxx, boxy) # get pixel coords from board coords
    # Draw the shapes
    if shape == DONUT:
        pygame.draw.circle(DISPLAYSURF, color, (left + half, top + half), half - 5)
        pygame.draw.circle(DISPLAYSURF, BGCOLOR, (left + half, top + half), quarter - 5)
    elif shape == SQUARE:
        pygame.draw.rect(DISPLAYSURF, color, \
                        (left + quarter, top + quarter, BOXSIZE - half, BOXSIZE - half))
    elif shape == DIAMOND:
        pygame.draw.polygon(DISPLAYSURF, color, \
                        ((left + half, top), (left + BOXSIZE - 1, top + half), \
                        (left + half, top + BOXSIZE - 1), (left, top + half)))
    elif shape == LINES:
        for i in range(0, BOXSIZE, 4):
            pygame.draw.line(DISPLAYSURF, color, (left, top + i), (left + i, top))
            pygame.draw.line(DISPLAYSURF, color, \
                    (left + i, top + BOXSIZE - 1), (left + BOXSIZE - 1, top + i))
    elif shape == OVAL:
        pygame.draw.ellipse(DISPLAYSURF, color, (left, top + quarter, BOXSIZE, half))
```

# Box Coordinates

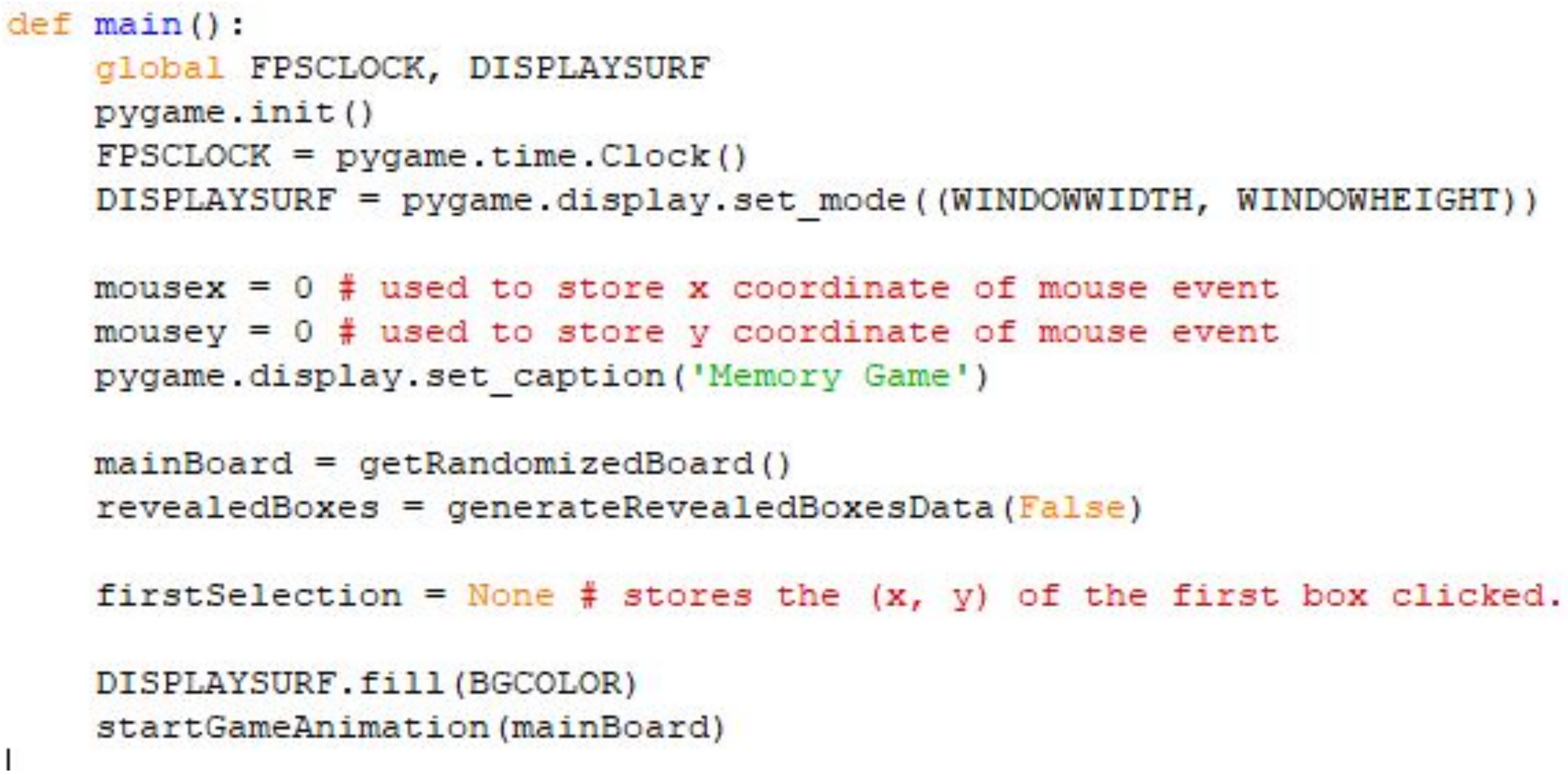


# Diagram

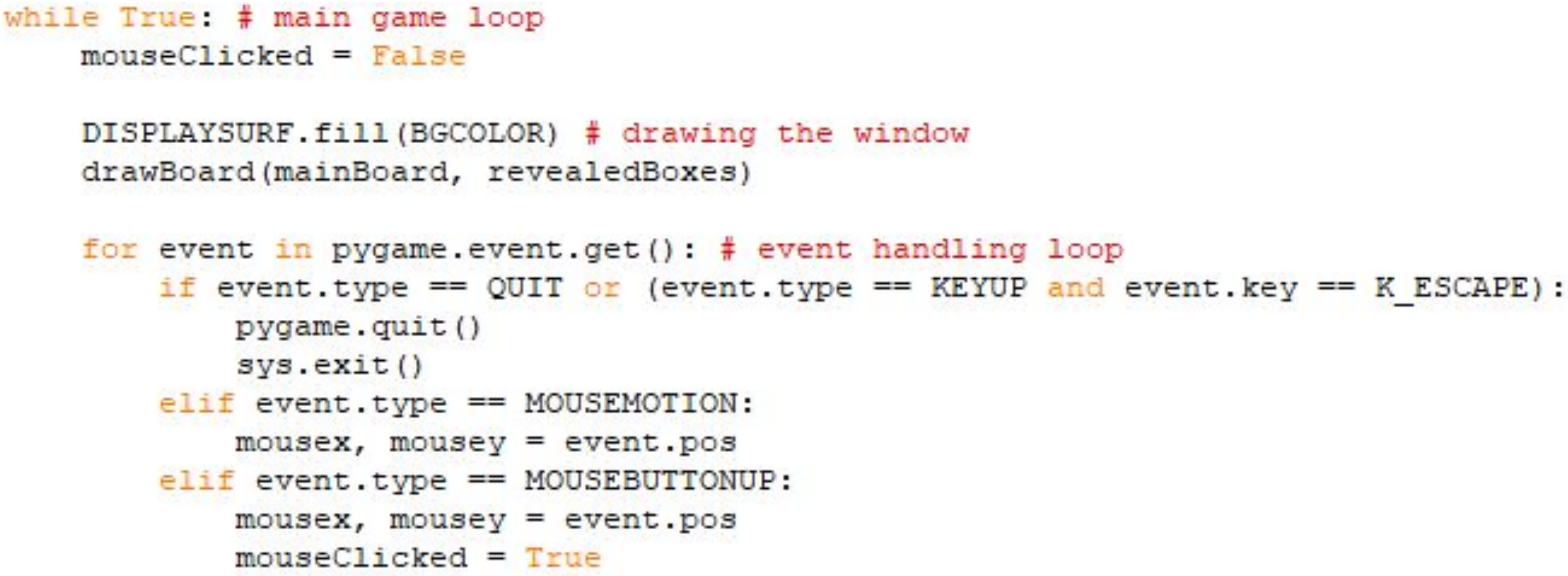


Generate board
Draw board
Check event
Quit
check mouse (x,y) in a box and not revealed
If first selection
If  second selection
Mark it
Check match
Won

# Main function

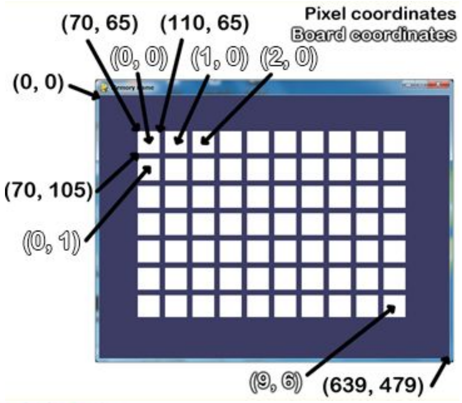


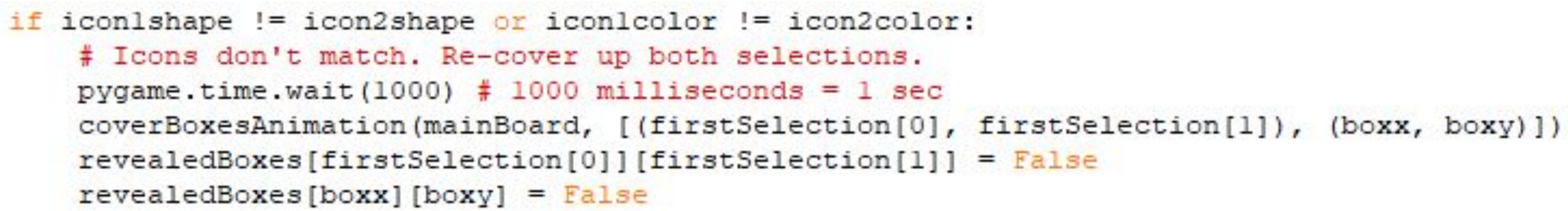
```
def main():
    global FPSCLOCK, DISPLAYSURF
    pygame.init()
    FPSCLOCK = pygame.time.Clock()
    DISPLAYSURF = pygame.display.set_mode((WINDOWWIDTH, WINDOWHEIGHT))

    mousex = 0 # used to store x coordinate of mouse event
    mousey = 0 # used to store y coordinate of mouse event
    pygame.display.set_caption('Memory Game')

    mainBoard = getRandomizedBoard()
    revealedBoxes = generateRevealedBoxesData(False)

    firstSelection = None # stores the (x, y) of the first box clicked.

    DISPLAYSURF.fill(BGCOLOR)
    startGameAnimation(mainBoard)
```



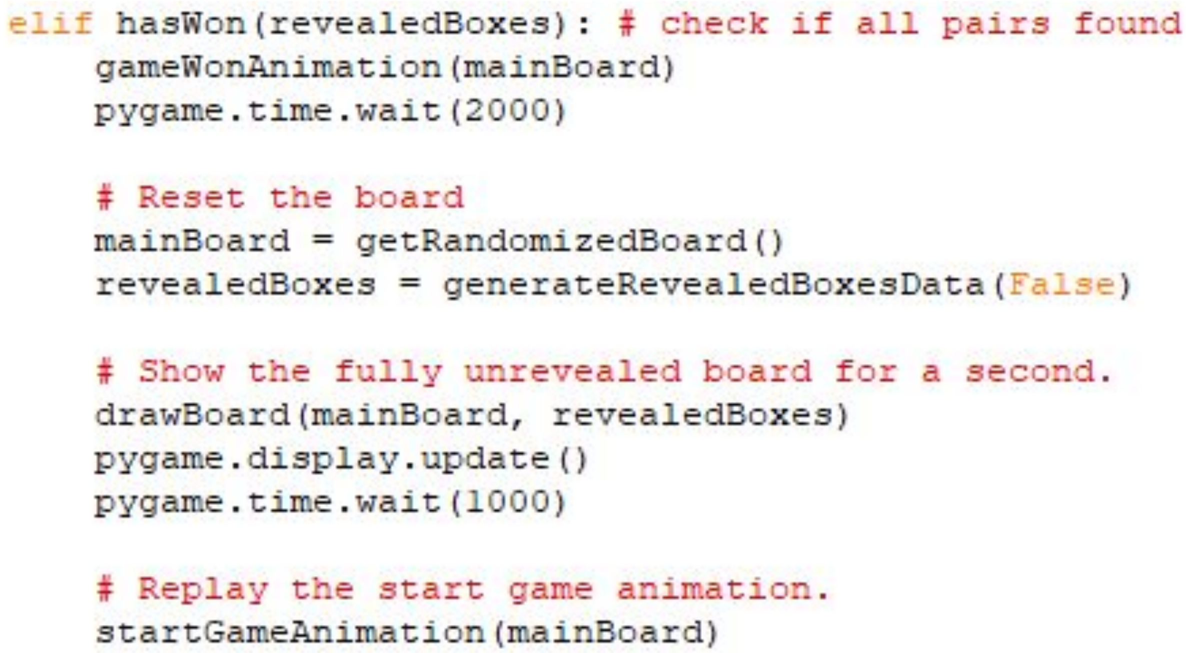
```
while True: # main game loop
    mouseClicked = False

    DISPLAYSURF.fill(BGCOLOR) # drawing the window
    drawBoard(mainBoard, revealedBoxes)

    for event in pygame.event.get(): # event handling loop
        if event.type == QUIT or (event.type == KEYUP and event.key == K_ESCAPE):
            pygame.quit()
            sys.exit()
        elif event.type == MOUSEMOTION:
            mousex, mousey = event.pos
        elif event.type == MOUSEBUTTONUP:
            mousex, mousey = event.pos
            mouseClicked = True
```

```
boxx, boxy = getBoxAtPixel(mousex, mousey)
if boxx != None and boxy != None:
    # The mouse is currently over a box.
    if not revealedBoxes[boxx][boxy]:
        drawHighlightBox(boxx, boxy)
    if not revealedBoxes[boxx][boxy] and mouseClicked:
        revealBoxesAnimation(mainBoard, [(boxx, boxy)])
        revealedBoxes[boxx][boxy] = True # set the box as "revealed"
        if firstSelection == None: # the current box was the first box clicked
            firstSelection = (boxx, boxy)
        else: # the current box was the second box clicked
            # Check if there is a match between the two icons.
            icon1shape, icon1color = getShapeAndColor(mainBoard, firstSelection[0], firstSelection[1])
            icon2shape, icon2color = getShapeAndColor(mainBoard, boxx, boxy)
```

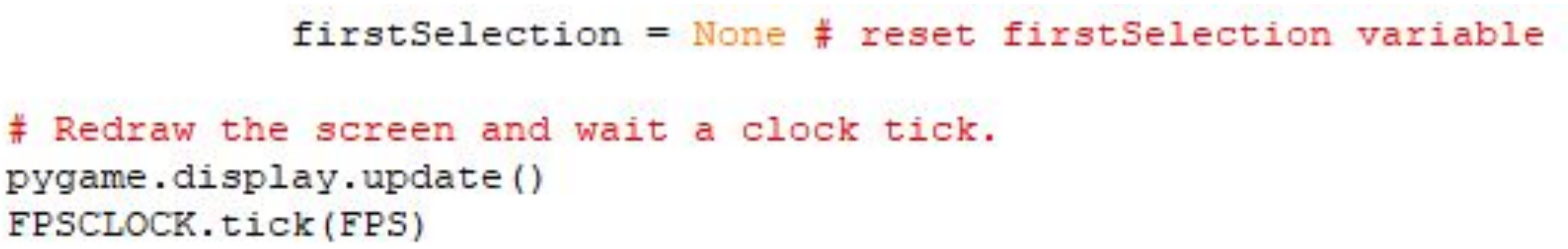




```
if icon1shape != icon2shape or icon1color != icon2color:
    # Icons don't match. Re-cover up both selections.
    pygame.time.wait(1000) # 1000 milliseconds = 1 sec
    coverBoxesAnimation(mainBoard, [(firstSelection[0], firstSelection[1]), (boxx, boxy)])
    revealedBoxes[firstSelection[0]][firstSelection[1]] = False
    revealedBoxes[boxx][boxy] = False
```

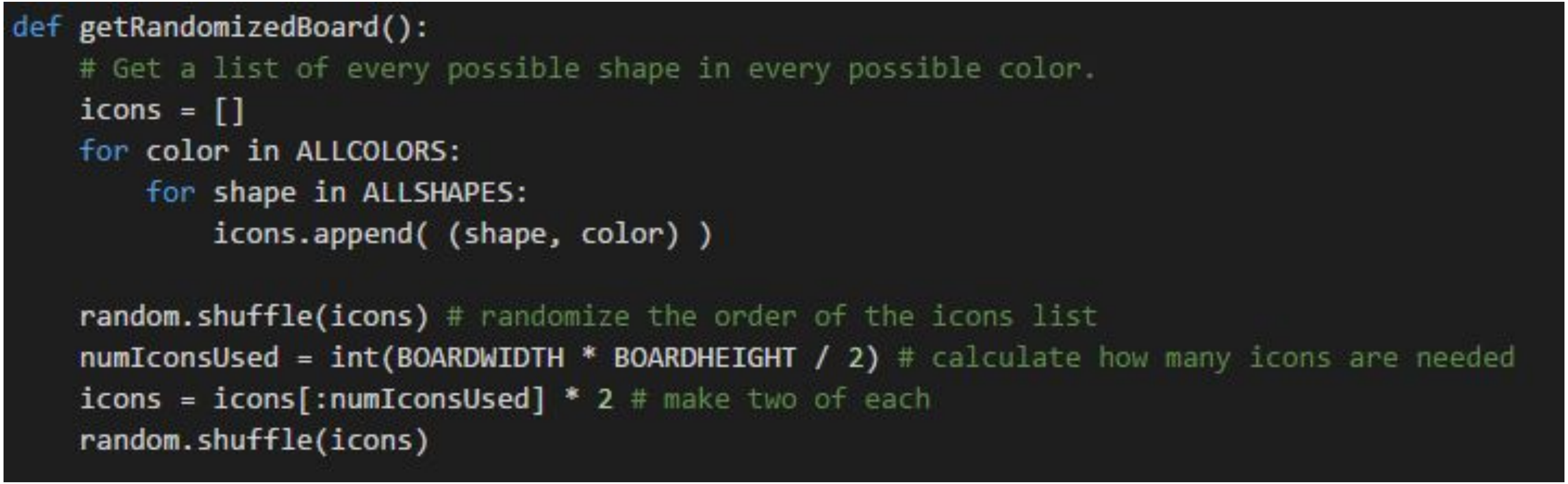


```
elif hasWon(revealedBoxes): # check if all pairs found
    gameWonAnimation(mainBoard)
    pygame.time.wait(2000)

    # Reset the board
    mainBoard = getRandomizedBoard()
    revealedBoxes = generateRevealedBoxesData(False)

    # Show the fully unrevealed board for a second.
    drawBoard(mainBoard, revealedBoxes)
    pygame.display.update()
    pygame.time.wait(1000)

    # Replay the start game animation.
    startGameAnimation(mainBoard)
```



```
            firstSelection = None # reset firstSelection variable

# Redraw the screen and wait a clock tick.
pygame.display.update()
FPSCLOCK.tick(FPS)
```

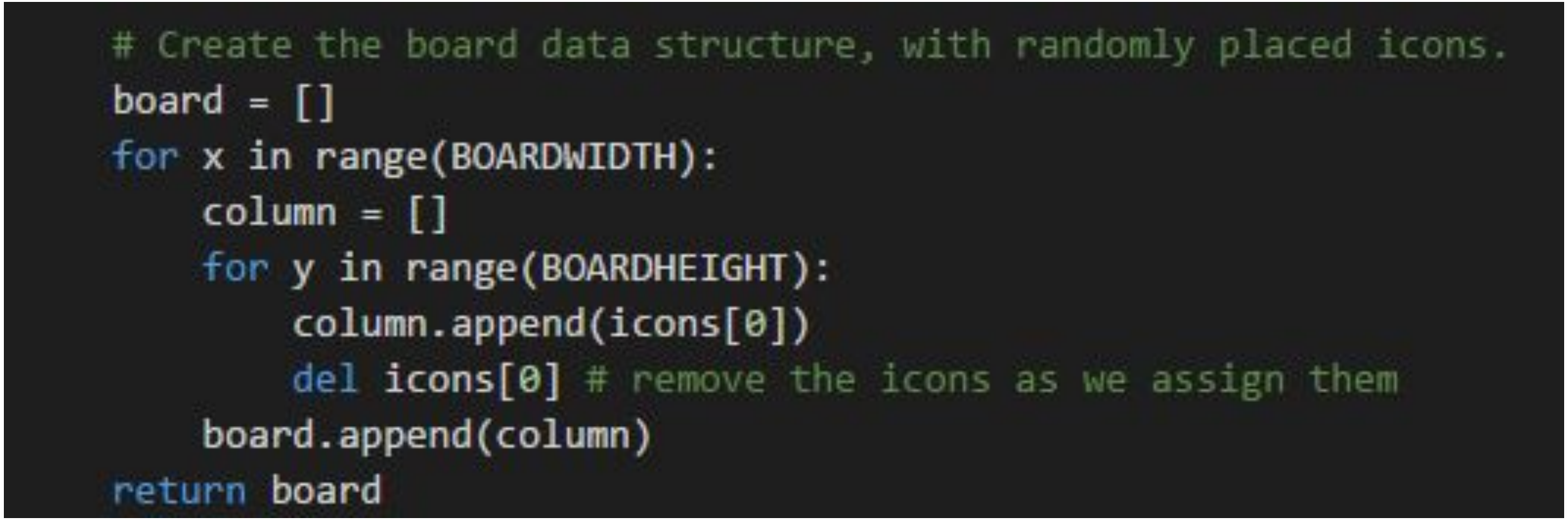
# Generate Board (1)



```
def getRandomizedBoard():
    # Get a list of every possible shape in every possible color.
    icons = []
    for color in ALLCOLORS:
        for shape in ALLSHAPES:
            icons.append( (shape, color) )

    random.shuffle(icons) # randomize the order of the icons list
    numIconsUsed = int(BOARDWIDTH * BOARDHEIGHT / 2) # calculate how many icons are needed
    icons = icons[:numIconsUsed] * 2 # make two of each
    random.shuffle(icons)
```

Exercise:
A = [‘a’, ‘b’, ‘c’, ‘d’]
print(A[:3])
print(A[:3]*2)

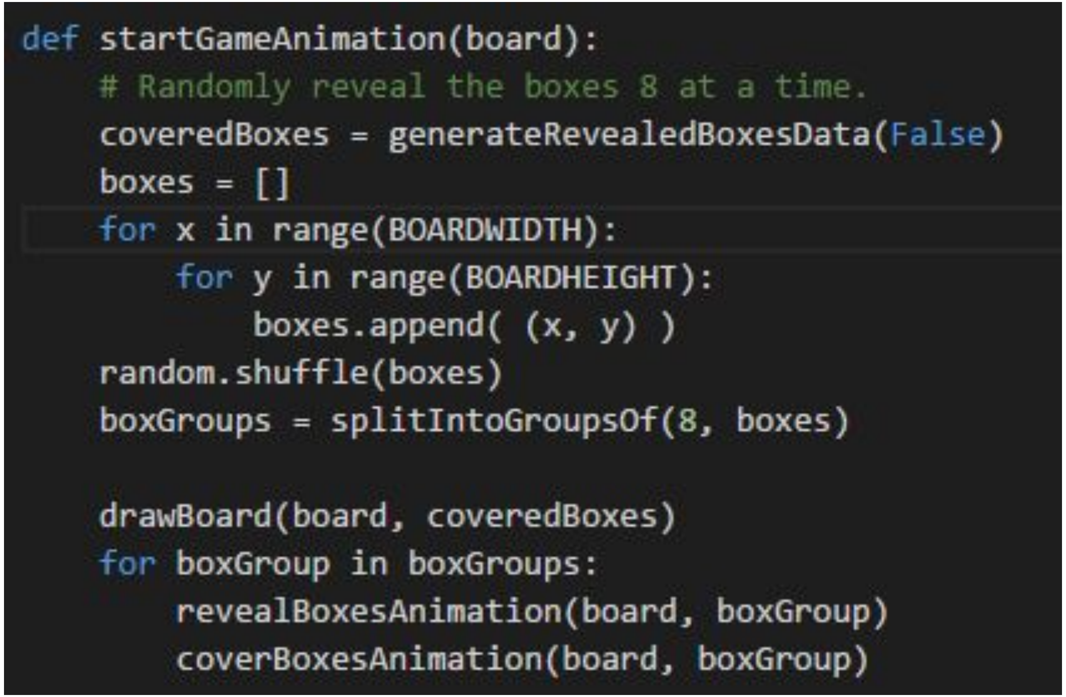
# Generate Board (2)



```
# Create the board data structure, with randomly placed icons.
board = []
for x in range(BOARDWIDTH):
    column = []
    for y in range(BOARDHEIGHT):
        column.append(icons[0])
        del icons[0] # remove the icons as we assign them
    board.append(column)
return board
```

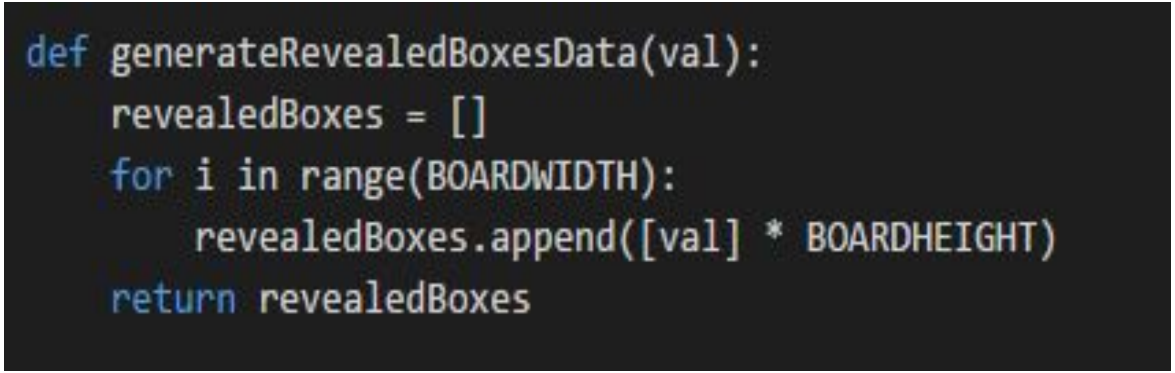
```
board = [
        [(‘square’, ‘red’) , (‘diamond’, ‘yellow’), …  (‘donut’, ‘blue’)],
        [(‘lines’, ‘green’) , (‘square’, ‘red’’), …  (‘lines’, ‘green’)],
        …
        [(‘oval’, ‘orange’) , (‘diamond’, ‘blue’), …  (‘donut’, ‘blue’)]
        ]
```

# Start Game Animation



```
def startGameAnimation(board):
    # Randomly reveal the boxes 8 at a time.
    coveredBoxes = generateRevealedBoxesData(False)
    boxes = []
    for x in range(BOARDWIDTH):
        for y in range(BOARDHEIGHT):
            boxes.append( (x, y) )
    random.shuffle(boxes)
    boxGroups = splitIntoGroupsOf(8, boxes)

    drawBoard(board, coveredBoxes)
    for boxGroup in boxGroups:
        revealBoxesAnimation(board, boxGroup)
        coverBoxesAnimation(board, boxGroup)
```

# Revealed Box Data

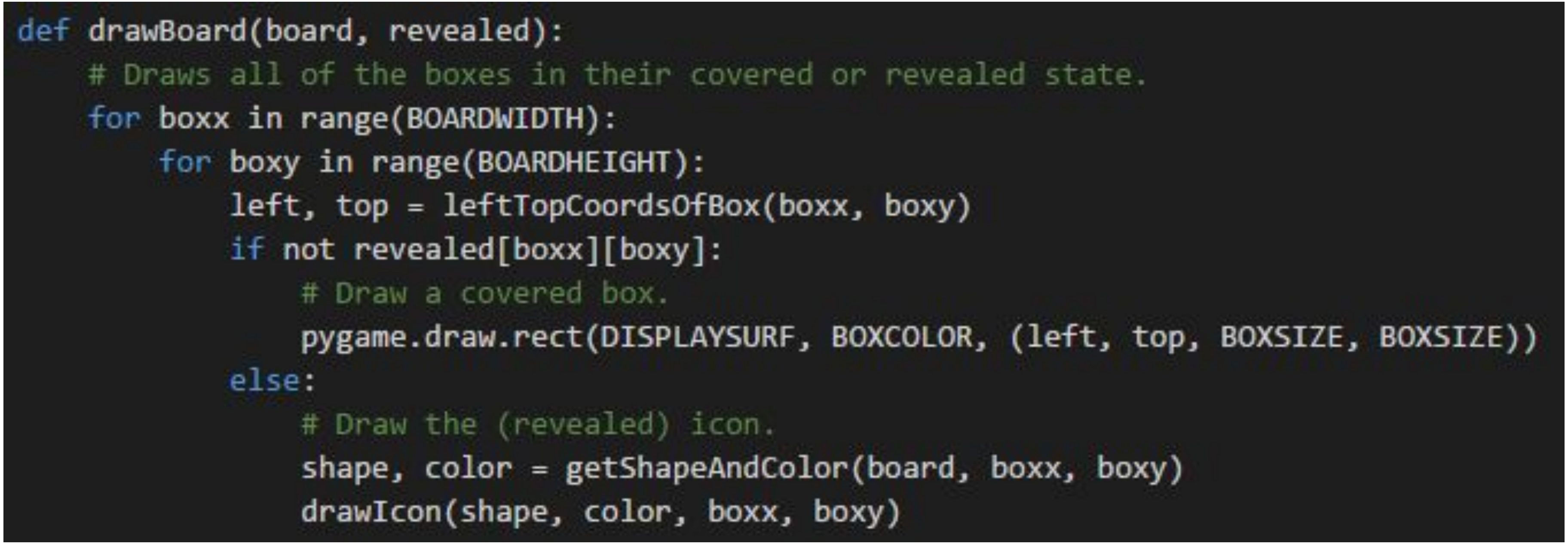


```
def generateRevealedBoxesData(val):
    revealedBoxes = []
    for i in range(BOARDWIDTH):
        revealedBoxes.append([val] * BOARDHEIGHT)
    return revealedBoxes
```

```
#Exercise:
A = [1]
B = [1] *3
print(B)

C = []
C.append([1]*3)
C.append([2]*3)
print(C)

print(C[0])
print(C[0][1])
print(C[1])
print(C[1][1])
```

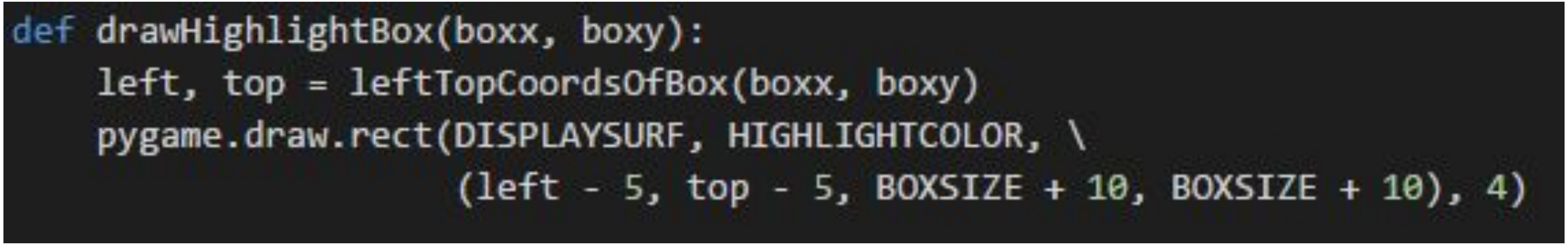
```
revealedBoxes = [
        [false, false, … false],
        [false, false, … false],
        …
        [false, false, … false]
        ]
```

# Draw Board



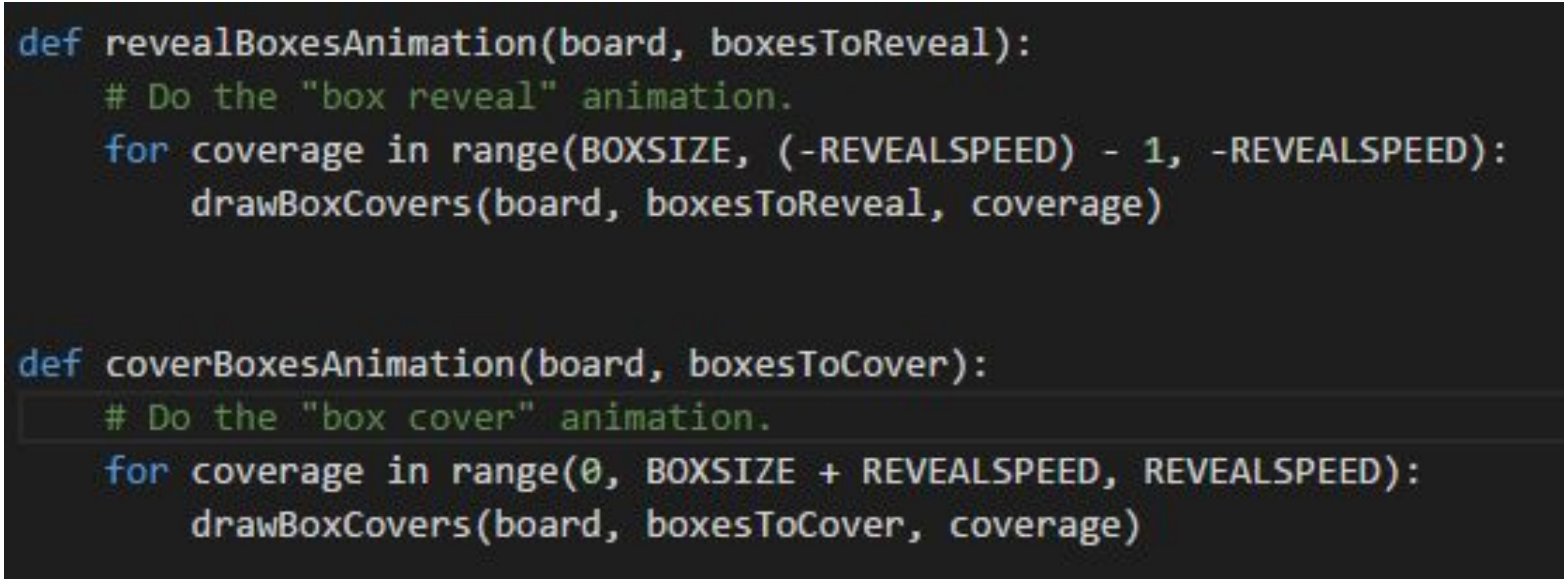
```python
def drawBoard(board, revealed):
    # Draws all of the boxes in their covered or revealed state.
    for boxx in range(BOARDWIDTH):
        for boxy in range(BOARDHEIGHT):
            left, top = leftTopCoordsOfBox(boxx, boxy)
            if not revealed[boxx][boxy]:
                # Draw a covered box.
                pygame.draw.rect(DISPLAYSURF, BOXCOLOR, (left, top, BOXSIZE, BOXSIZE))
            else:
                # Draw the (revealed) icon.
                shape, color = getShapeAndColor(board, boxx, boxy)
                drawIcon(shape, color, boxx, boxy)
```

# Draw Highlight Box



```
def drawHighlightBox(boxx, boxy):
    left, top = leftTopCoordsOfBox(boxx, boxy)
    pygame.draw.rect(DISPLAYSURF, HIGHLIGHTCOLOR, \
                     (left - 5, top - 5, BOXSIZE + 10, BOXSIZE + 10), 4)
```

# Animation

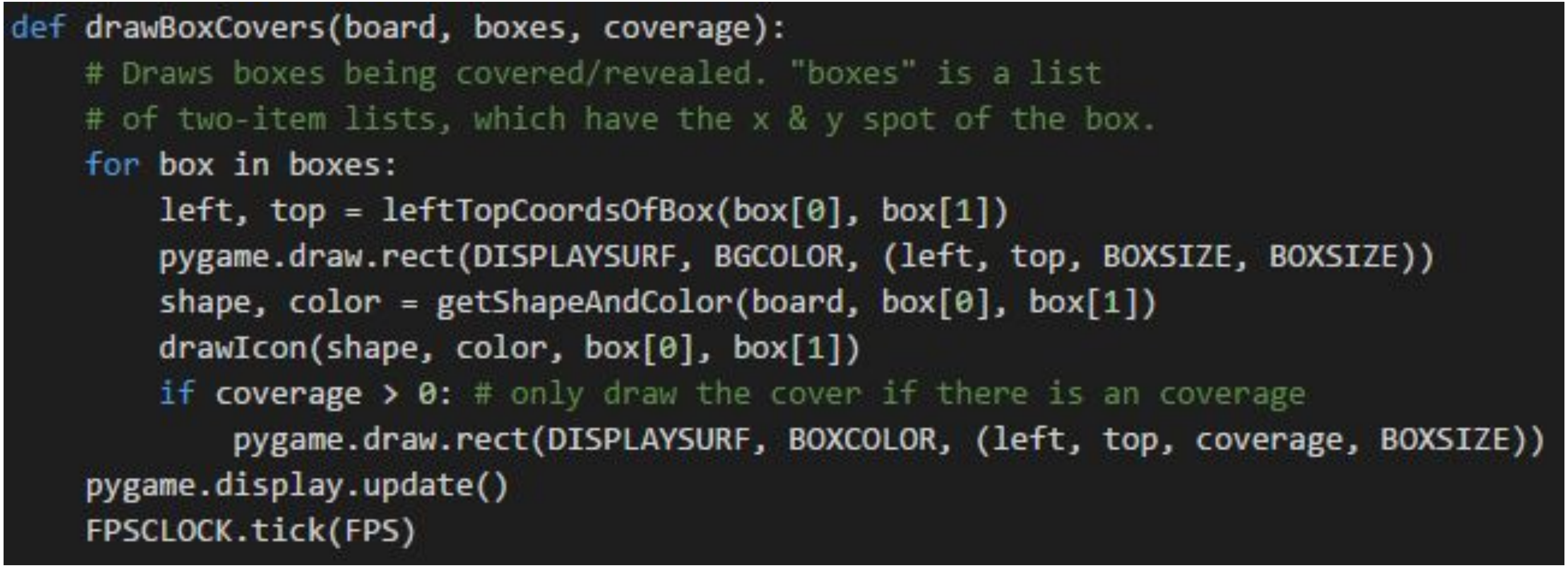


```
def revealBoxesAnimation(board, boxesToReveal):
    # Do the "box reveal" animation.
    for coverage in range(BOXSIZE, (-REVEALSPEED) - 1, -REVEALSPEED):
        drawBoxCovers(board, boxesToReveal, coverage)


def coverBoxesAnimation(board, boxesToCover):
    # Do the "box cover" animation.
    for coverage in range(0, BOXSIZE + REVEALSPEED, REVEALSPEED):
        drawBoxCovers(board, boxesToCover, coverage)
```

Reveal Animation

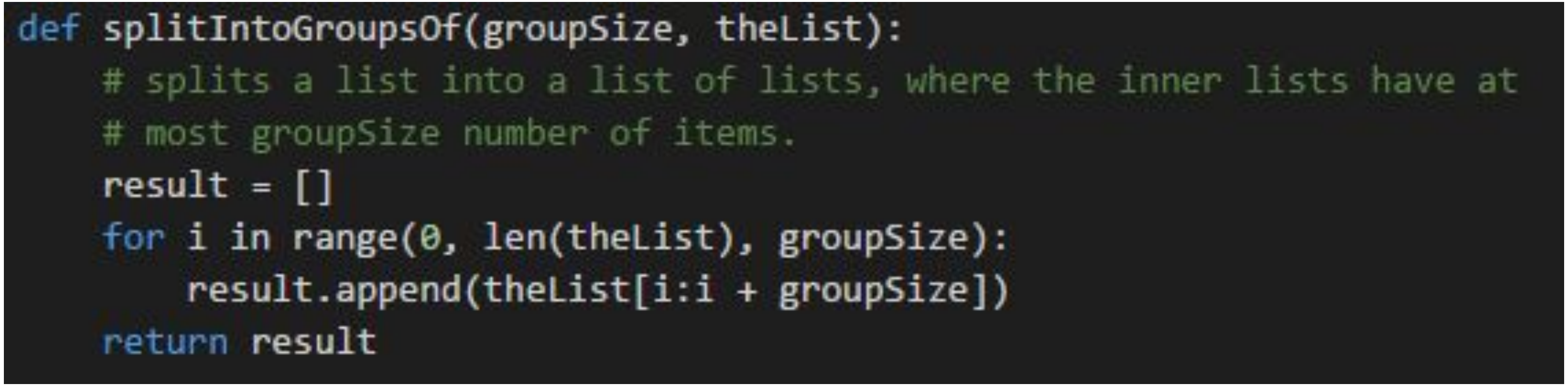
Cover Animation

# Draw Box Covers



```
def drawBoxCovers(board, boxes, coverage):
    # Draws boxes being covered/revealed. "boxes" is a list
    # of two-item lists, which have the x & y spot of the box.
    for box in boxes:
        left, top = leftTopCoordsOfBox(box[0], box[1])
        pygame.draw.rect(DISPLAYSURF, BGCOLOR, (left, top, BOXSIZE, BOXSIZE))
        shape, color = getShapeAndColor(board, box[0], box[1])
        drawIcon(shape, color, box[0], box[1])
        if coverage > 0: # only draw the cover if there is an coverage
            pygame.draw.rect(DISPLAYSURF, BOXCOLOR, (left, top, coverage, BOXSIZE))
    pygame.display.update()
    FPSCLOCK.tick(FPS)
```

# Split Into Groups



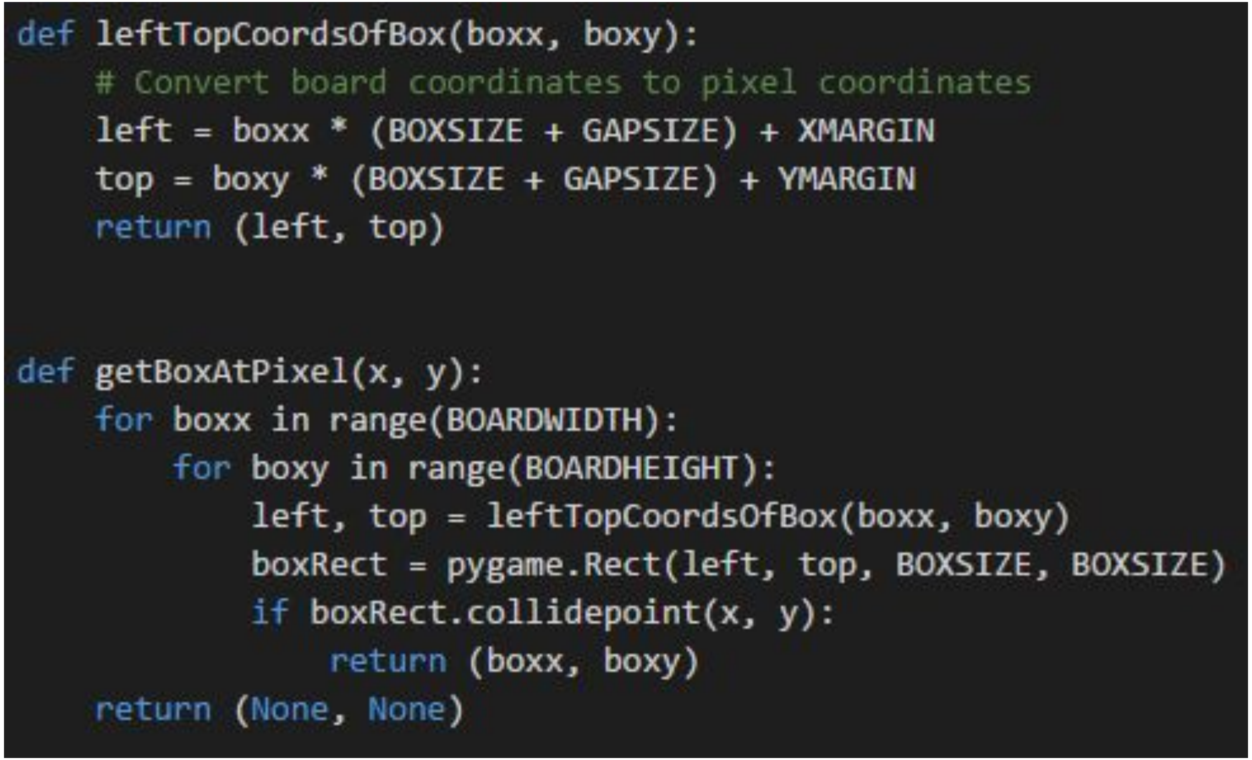
```python
def splitIntoGroupsOf(groupSize, theList):
    # splits a list into a list of lists, where the inner lists have at
    # most groupSize number of items.
    result = []
    for i in range(0, len(theList), groupSize):
        result.append(theList[i:i + groupSize])
    return result
```

Exercise:

X = range(0, 10, 2 )
print(X)

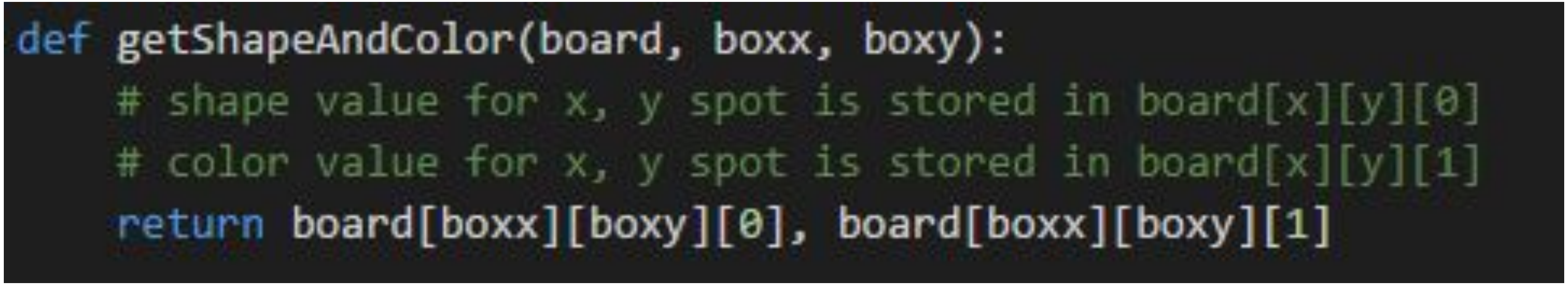
X = range(0, 10, 3)
print(X)

# Get Box Coordinate



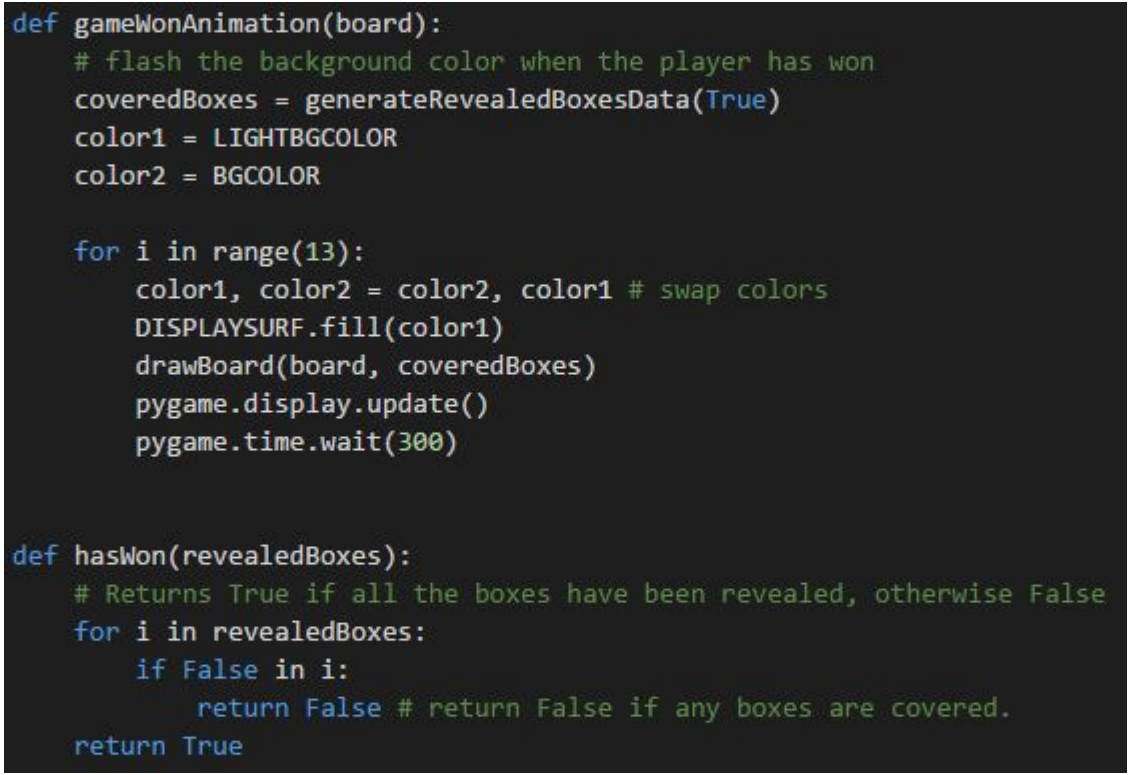
```python
def leftTopCoordsOfBox(boxx, boxy):
    # Convert board coordinates to pixel coordinates
    left = boxx * (BOXSIZE + GAPSIZE) + XMARGIN
    top = boxy * (BOXSIZE + GAPSIZE) + YMARGIN
    return (left, top)


def getBoxAtPixel(x, y):
    for boxx in range(BOARDWIDTH):
        for boxy in range(BOARDHEIGHT):
            left, top = leftTopCoordsOfBox(boxx, boxy)
            boxRect = pygame.Rect(left, top, BOXSIZE, BOXSIZE)
            if boxRect.collidepoint(x, y):
                return (boxx, boxy)
    return (None, None)
```

# Get Box Shape and Color



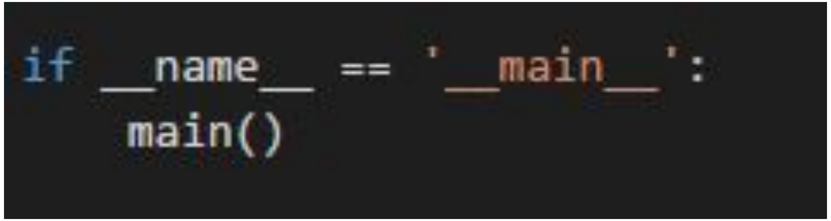
```python
def getShapeAndColor(board, boxx, boxy):
    # shape value for x, y spot is stored in board[x][y][0]
    # color value for x, y spot is stored in board[x][y][1]
    return board[boxx][boxy][0], board[boxx][boxy][1]
```

# Game Won and Animation



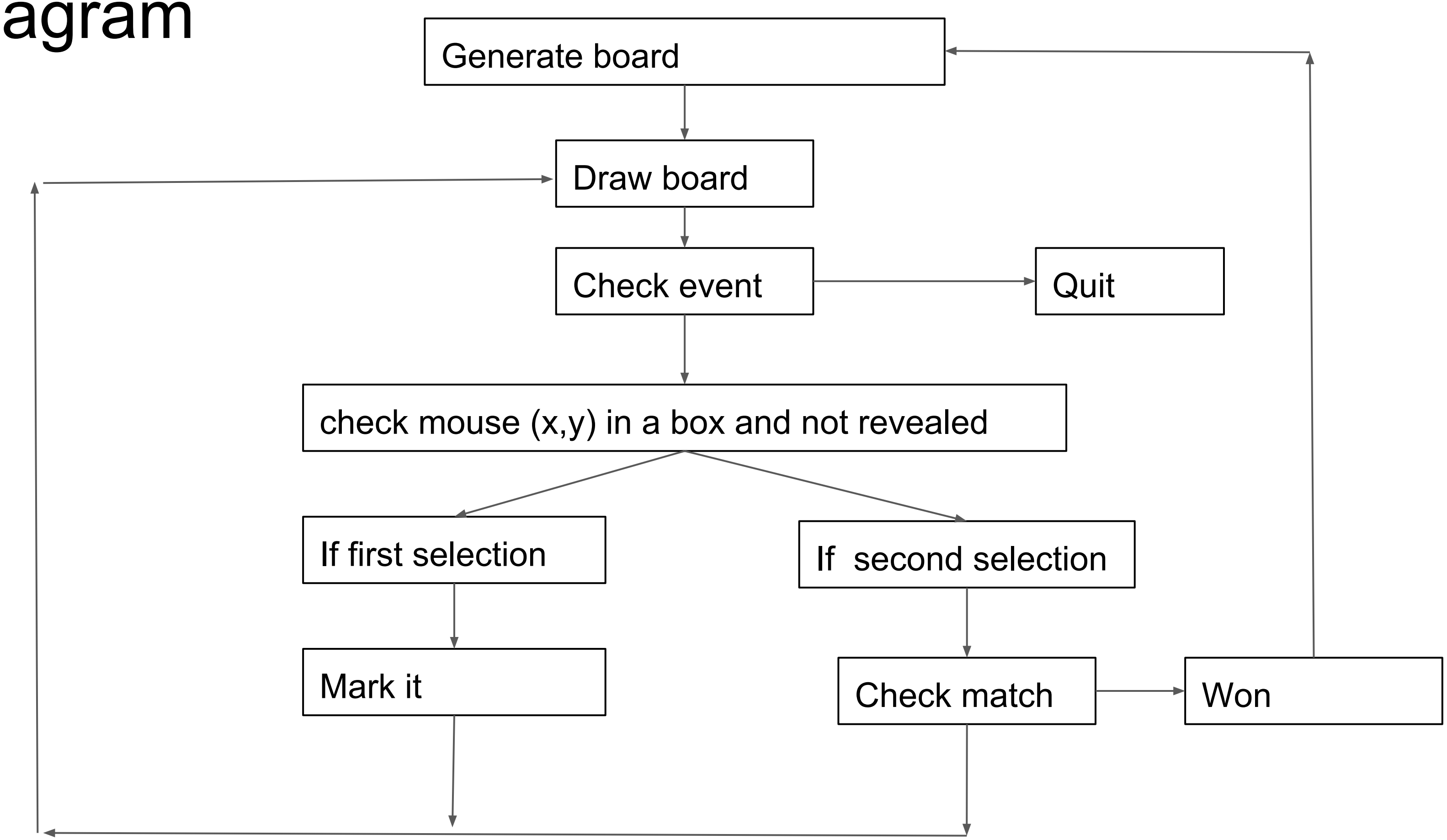
```
def gameWonAnimation(board):
    # flash the background color when the player has won
    coveredBoxes = generateRevealedBoxesData(True)
    color1 = LIGHTBGCOLOR
    color2 = BGCOLOR

    for i in range(13):
        color1, color2 = color2, color1 # swap colors
        DISPLAYSURF.fill(color1)
        drawBoard(board, coveredBoxes)
        pygame.display.update()
        pygame.time.wait(300)


def hasWon(revealedBoxes):
    # Returns True if all the boxes have been revealed, otherwise False
    for i in revealedBoxes:
        if False in i:
            return False # return False if any boxes are covered.
    return True
```

# Main function



```
if __name__ == '__main__':
    main()
```

# Diagram



Generate board

Draw board

Check event

Quit

check mouse (x,y) in a box and not revealed

If first selection

If  second selection

Mark it

Check match

Won