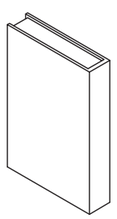
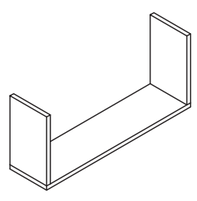
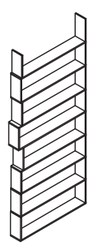


```
class Book:
def __init__(self, name, colour, origi
height, orientation):
    if name == None:
        self.name = random.choice(stri
    else:
        self.name = name
    self.colour = colour
    self.origin = origin
    self.width = width
    self.height = height
    self.depth = depth
    self.orientation =
    self.convertOrientation(orientatio
    self.box = self.getBox()

def getBox(self):
    a = self.origin
    b = a.translated(self.width * self
                     self.width * self.
                     self.width * self.
    d = a.translated(self.depth * -sel
                     self.depth * self.
                     self.depth * self.
    c = b.translated(self.depth * -sel
```



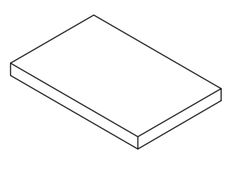
```
class Shelf:
def __init__(self, books):
    self.colour = pygame.Color(200, 20(
    self.width = 0
    self.highestBook = 0
    self.height = 20
    self.depth = 200
    self.orientation = books[0].orienta
    self.walls = []
    bookDeq = deque()
    for b in books:
        b.orientation = self.orientatio
        r = random.randint(0,1)
        if r == 0:
            bookDeq.appendleft(b)
        else:
            bookDeq.append(b)
        self.width += b.width
        if b.height > self.highestBook:
            self.highestBook = b.height
    bookDeq[0].moveTo(Point(0,0,0))
    for i in range(1, len(bookDeq)):
        bookDeq[i].moveTo(copy.deepcopy
        1].box.pB))
    self.books = bookDeq
```



```
class ShelvingColumn:
def __init__(self, hierarchy, origin, d
shelfLevels):
    self.origin = origin
    self.dir = dir
    self.width = 0
    self.height = 0
    self.depth = 0
    self.box = None
    self.orientation =
    self.convertOrientation(orientatio
    self.shelves = []
    self.hierarchy = hierarchy
    self.colour = pygame.Color(200, 20(
    self.walls = None
    self.maxLevels = shelfLevels
    self.bannedDirs = []

def convertOrientation(self, d):
    l = math.sqrt(d[0]**2 + d[1]**2)
    return [d[0]/l, d[1]/l, 0]

def getBox(self):
    if self.height == 0 and self.depth
    self.width ==0:
```

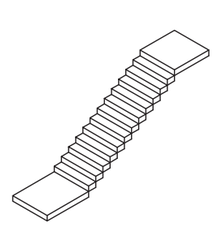


```
class Walkway:
def __init__(self, col, walkWidth, hei
    self.walkWidth = walkWidth
    self.pWalkway = pWalkway
    self.height = height
    self.col = col
    self.orientation = col.orientation
    self.v = v
    self.pA = col.box.pA.translated(0,
    self.pB = col.box.pB.translated(0,
    self.pC = col.box.pC.translated(se
    col.orientation[1],
                                    self.walkWidth
                                    col.orientatio
                                    self.walkWidth
                                    col.orientatio
    self.pD = col.box.pD.translated(se
    col.orientation[1],
                                    self.walkWidth
                                    col.orientatio
                                    self.walkWidth
                                    col.orientatio
    self.box = Box(pygame.Color(0, 0,
    self.pB, self.pC, self.pD, 100)
    self.pillar = []
```



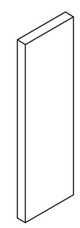
```
class Pillars:
def __init__(self, collections):
    self.collections = collections
    self.walkways = []
    self.pillars = self.getPillars()
    self.floorHeights = self.getFloorH
    self.envelopes = self.getEnvelopes

def extendPillars(self):
    pillars = []
    for f in range(len(self.floorHeigh
        lengthenPillars = []
        for p in self.pillars[f]:
            p.height += 100
            if not
            (self.walkwayCollision(sel
            p) or
            self.envelopeCollision(sel
            p)):
                lengthenPillars.append
        for p in pillars:
            if not
            (self.walkwayCollision(sel
            p) or
            self.envelopeCollision(sel
```



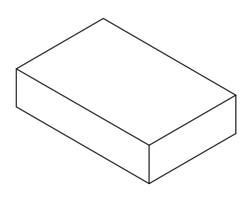
```
class Stairs:
def __init__(self, clusterCreator, wid
    self.clusterCreator = clusterCreat
    self.width = width
    self.depth = depth
    self.height = height
    self.clusters = self.clusterCreato
    self.floors = self.clusterCreator.
    self.stairs = []

def makeStairs(self):
    for n in range(self.floors):
        cluster = self.clusters[n]
        staircase = []
        floorheight = cluster.floorHei
        col = cluster.shelfColumns[-1]
        col.width = self.width
        col.getBox()
        walkway = Step(col.box.colour,
        col.box.pB.translated(0,0,-100
                col.depth, 100, self.
                col.orientation)
        staircase.append(walkway)
        while ((len(staircase)-1) * se
        floorheight - self.height):
```



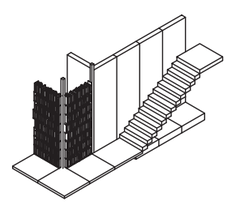
```
class WallPanels:
def __init__(self, clusterCreator, wid
maxSteps):
    self.clusterCreator = clusterCreat
    self.width = width
    self.depth = depth
    self.clusters = self.clusterCreato
    self.floors = self.clusterCreator.
    self.panels = []
    self.maxSteps = maxSteps
    self.oneFloorNr = 0

def initiatePanels(self):
    for n in range(self.floors):
        envelope = self.clusterCreator
        height = self.clusters[n].floo
        z = self.clusters[n].origin.z
        lines = envelope.getLines()
        oneFloorNr = 0
        for x in lines:
            l = Line(None, Point(x[0][
            Point(x[1][0], x[1][1], 0)
            row = []
            oX = l.pointB.x - l.pointA
            oY = l.pointB.y - l.pointA
```



```
class FloorBoard:
def __init__(self, colour, origin, widt
orientation, floor):
    self.colour = colour
    self.origin = origin
    self.width = width
    self.height = height
    self.depth = depth
    self.orientation =
    self.convertOrientation(orientation
    self.box = self.getBox()
    self.floor = floor

def getBox(self):
    a = self.origin
    b = a.translated(self.width * self.
                     self.width * self.o
                     self.width * self.o
    d = a.translated(self.depth * -self
                     self.depth * self.o
                     self.depth * self.o
    c = b.translated(self.depth * -self
                     self.depth * self.o
                     self.depth * self.o
```



```
class ClusterCreator:
def __init__(self, n, envelopes, origi
size, i, shelfLevels):
    self.n = n
    self.i = i
    self.envelopes = envelopes
    self.clusters = []
    self.size = size
    self.otherClusters = []
    self.origin = origin
    self.orientation = orientation
    #self.floorHeight = floorHeight
    self.shelfLevels = shelfLevels
    self.walkWidth = 1200
    self.wallPanels = None

def create(self):
    prevCollection = None
    for i in range(self.n):
        books = BookCollection(2**self
        shelves = []
        start = self.origin
        startOrientation = self.orient
        for b in range(2**(self.size-5
            set = books.books[b*32:b*3
```