Medical Images
Session 8

Lab: more on rasterization

D. Tost

1. Create the class method and make sure that you are able to create a uniforme image of the given size and color with the name of the vim.

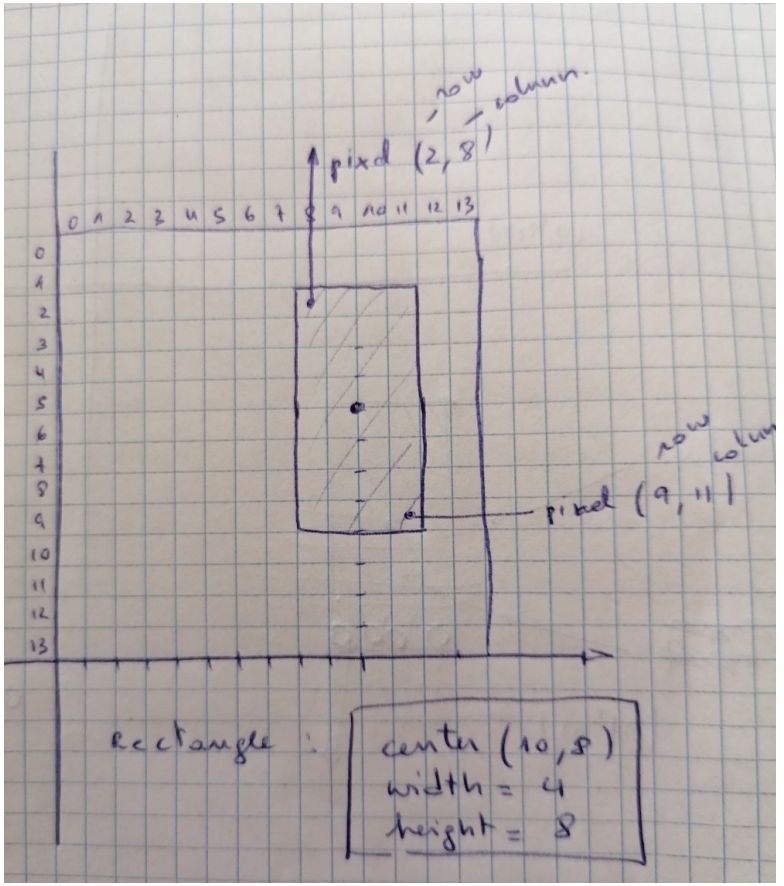
```
$ python3
Python 3.8.10 (default, Jun 22 2022, 20:18:18)
[GCC 9.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> from rimage import RImage
>>> from vimage import VImage
>>> vi = VImage.from_json('vim1.json')
>>> ri = RImage.rasterization(vi, 200, 300, (255, 255, 0))
>>> import matplotlib.pyplot as plt
>>> fig, ax = plt.subplots(1)
>>> ri.render_matplotlib(ax)
>>> plt.show()
```

```
class RImage:
    """
    Class representing a raster image
    """

    __dic_procedural = {
        "coffee": data.coffee,
        "astronaut": data.astronaut,
        "moon": data.moon,
    }

    @classmethod
    def rasterization(cls, vim, w, h, color):
        return RImage(vim.name, w, h, color)
```

To be improved in the next steps!!

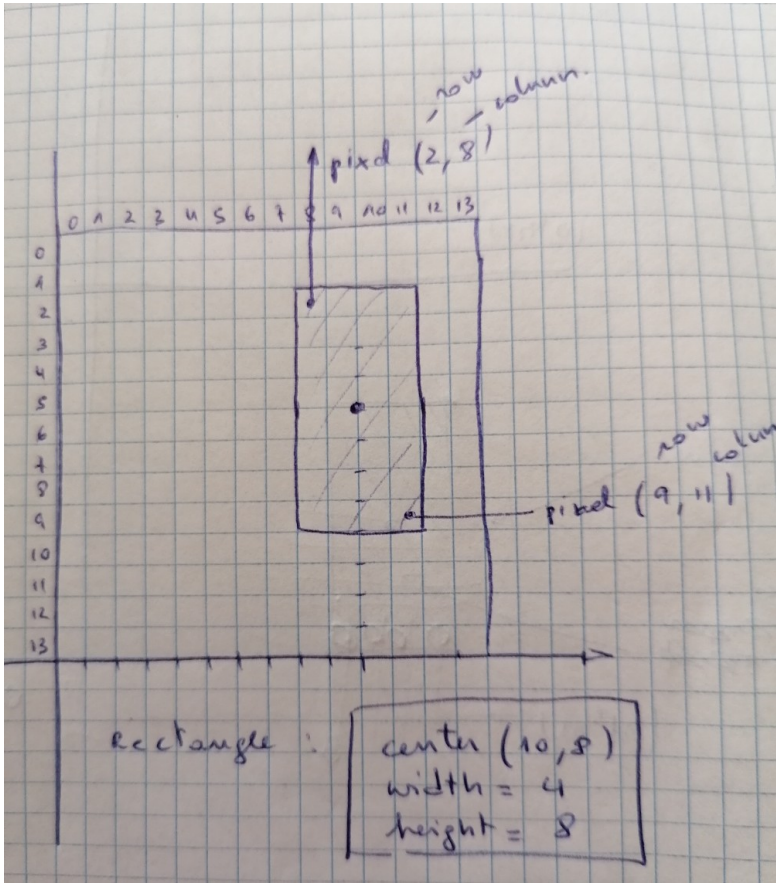
2. Understand rasterization. Focus on rectangles, first. Draw a case on paper with a pencil and try to rasterize “manually” in the terminal.

```
>>> from skimage import draw
>>> rr, cc = draw.rectangle((2, 8), (9, 11))
>>> rr
array([[2, 3, 4, 5, 6, 7, 8, 9],
       [2, 3, 4, 5, 6, 7, 8, 9],
       [2, 3, 4, 5, 6, 7, 8, 9],
       [2, 3, 4, 5, 6, 7, 8, 9]])
>>> cc
array([[ 8,  8,  8,  8,  8,  8,  8,  8],
       [ 9,  9,  9,  9,  9,  9,  9,  9],
       [10, 10, 10, 10, 10, 10, 10, 10],
       [11, 11, 11, 11, 11, 11, 11, 11]])
```



2. Understand rasterization. Focus on rectangles, first. Draw a case on paper with a pencil and try to rasterize “manually” in the terminal.

In this drawing, the viewport has width 14 and height 14. The window has origin (0, 0) and width 14 and height 14, therefor center(7, 7)
Do you see that we have to transform manually the rectangle and specify (2, 8) as the starting vertex?



3. Implement the method rasterization in class Rectangle.

```
def rasterization(self, matx):
    """
    To be implemented: it returns a list of 2D points corresponding to the
    rasterization of the rectangle having applied the window-to -viewport
    transformation matrix matx
    """
    pmin = self.vmin().rasterization(matx)
    pmax = self.vmax().rasterization(matx)
    rr, cc = draw.rectangle((pmax.y, pmin.x), (pmin.y-1, pmax.x-1))
    lpixels = []
    for i in range(len(rr)):
        for j in range(len(rr[i])):
            lpixels.append(Point(cc[i][j], rr[i][j]))
    return lpixels
```

We put -1 to avoid the enlargement of the rectangle

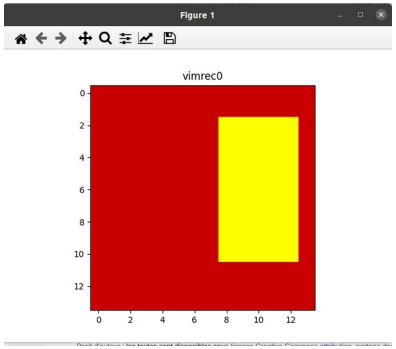
You just have to compute the extreme vertices of the rectangle, transform them specify the coordinates that rasdraw.rectangle requires. Then, create and return the list of pixels.

4. Try a first partial implementation. Specify the window **manually.**

```
[
   {
        "type": "rectangle",
        "value": {
           "center": {"x":10, "y": 8},
           "width": 4,
           "height": 8,
           "color": [255, 255, 0]
         }
   }
]
```

```
>>> import matplotlib.pyplot as plt
>>> from rimage import RImage
>>> from vimage import VImage
>>> vi = VImage.from_json('vimrec0.json')
>>> ri = RImage.rasterization(vi, 14, 14, (200, 0, 0))
>>> f, ax = plt.subplots(1)
>>> ri.render_matplotlib(ax)
>>> plt.show()
```

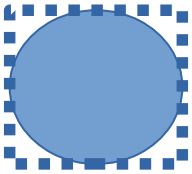
```
@classmethod
def rasterization(cls, vim, w, h, color):
    rim = RImage(vim.name, w, h, color)
    window = Rectangle(Point(7, 7), 14, 14, color)
    matx = window.viewport_transform(w, h)
    for fig in vim:
        lpixels = fig.rasterization(matx)
        for pix in lpixels:
            rim[int(pix.y), int(pix.x)] = fig.color
    return rim
```



The window is specified manually just to test!! This will work only for the scene that we draw by hand

4. Now, you just have to compute the window as the bounding box of the scene.

Implement the bounding box as an instance to Rectangle. Compute the bounding box of each figure and return its union. By now, start only the bounding box of the rectangle!!



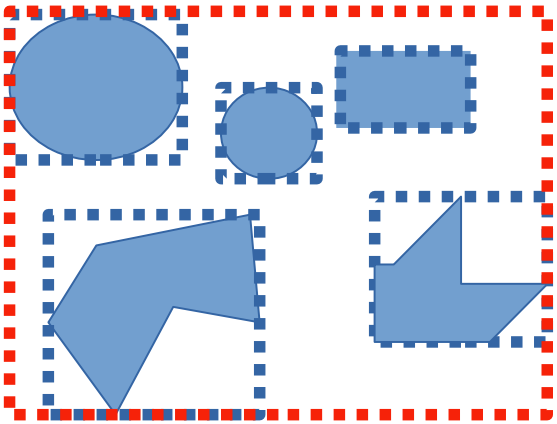
Bounding box of a cercle:
center = a copy of circle center
width and height = 2* radius

Bounding box of a polygon:
pmin = (min coord x of all vertices, min coord y of all vertices)
pmax = (max coord x of all vertices, ma coord y of all vertices)
center = the center point between pmin and pmax
width= pmax.x -pmin.x
Height= pmax.y-pmin.y

Bounding box of a rectangle:
A copy of itself

4- Now, you just have to compute the window as the bounding box of the scene.

Compute the bounding box of each figure and return its union.

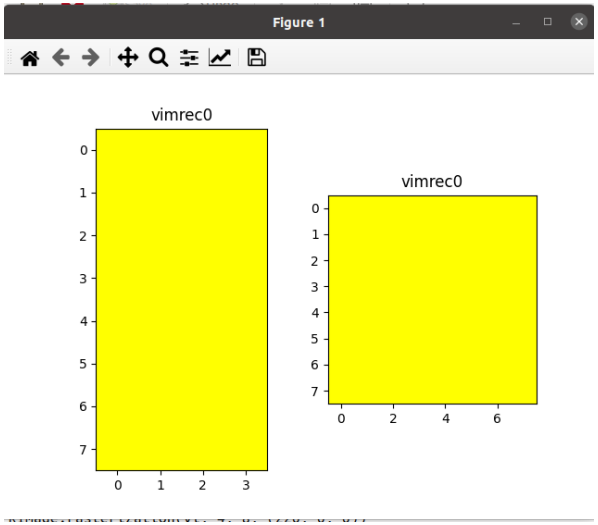


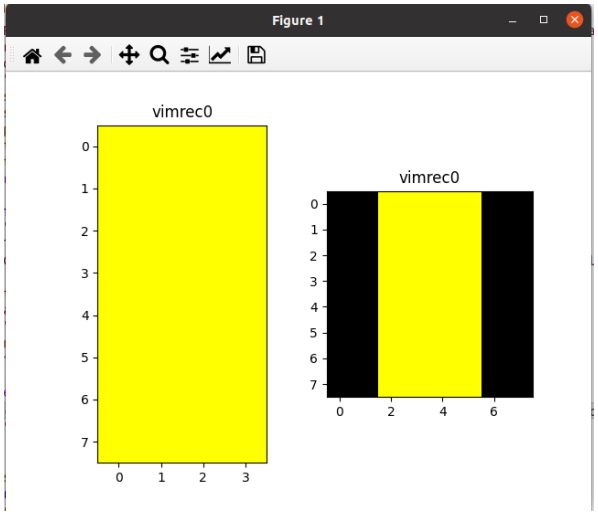
Bounding box:

xmin: min(bb.vmin().x) for all bounding boxes
xmax: max(bb.vmax().x) for all bounding boxes
xmin: min(bb.vmin().y) for all bounding boxes
xmax: max(bb.vmax().y) for all bounding boxes

center: midpoint (xmin, ymin) and (xmax, ymax)
width: xmax-xmin
height: ymax-ymin

4- Now, try :





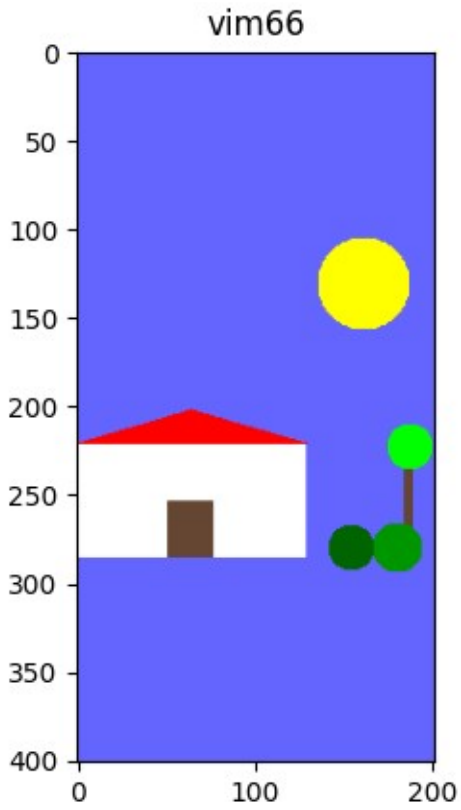

```
>>> import matplotlib.pyplot as plt
>>> from rimage import RImage
>>> from vimage import VImage
>>> vi = VImage.from_json('vimrec0.json')
>>> ri1 = RImage.rasterization(vi, 4, 8, (0, 0, 0))
>>> ri2 = RImage.rasterization(vi, 8, 8, (0, 0, 0))
>>> f, ax = plt.subplots(1,2)
>>> ri1.render_matplotlib(ax[0])
>>> ri2.render_matplotlib(ax[1])
>>> plt.show()
```

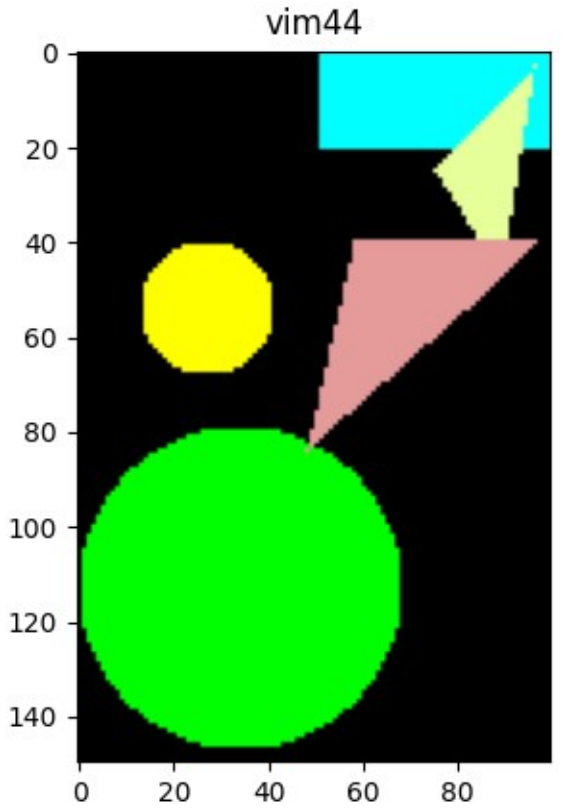
Do you see that the rectangle is deformed in the second plot?
You need to recompute the window to avoid deformations.

Implement the method fitting_window in class Rectanfle that given the boundingbox will return a window that contains the bounding box but has the same aspect ratio as the viewport.
You should get the second image.

5. Generalize to other types of figures and invent your own json files (find more in the docs)

- implement bounding_box in Circle and Polygon
- implement rasterization in Circle and Polygon





Problems you may find

When rasterizing the polygon, it happens as with the rectangle, the rasterization occupies 1 pixel more, and this is not as easy to solve as for the polygons. Therefore, my advice is to compute the transformations using the width and height of the raster image -1.

```
@classmethod
def rasterization(cls, vim, w, h, color):
    """
    It creates a rimage of the given size and background color on which the figures of the
    vectorial image have been rasterized.
    To avoid problems of testing with vectorial images defined in pixel coordinates, all
    transformations are computed using 1 pixel less.
    """
    rim = RImage(vim.name, w, h, color)
    window = vim.bounding_box()
    window = window.fitting_window(float(w-1)/(h-1))
    matx = window.viewport_transform(w-1, h-1)
    for fig in vim:
        lpixels = fig.rasterization(matx)
        for pix in lpixels:
            rim[int(pix.y), int(pix.x)] = fig.color
    return rim
```