Hochschule
München
University of
Applied Sciences

Fakultät für
Wirtschaftsingenieurwesen

Technical Drawing with Draftsman

Warm Up: Finde den Unterschied

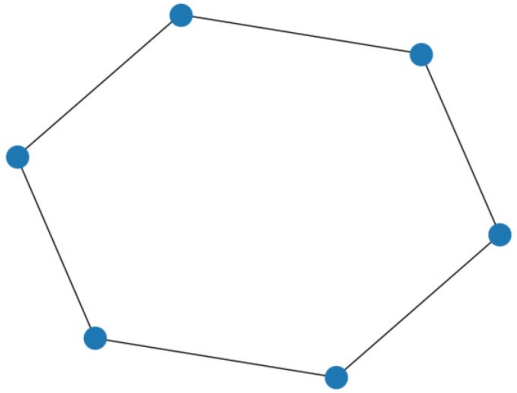
Frage:
Auf der rechten Seite sind zwei 6-Ecke („Hexagon“) dargestellt: einmal als Graph, einmal als Polygon.
Welche der folgenden Aussagen sind falsch?

a) Graph und Polygon bezeichnen dasselbe, d.h. sind identisch.

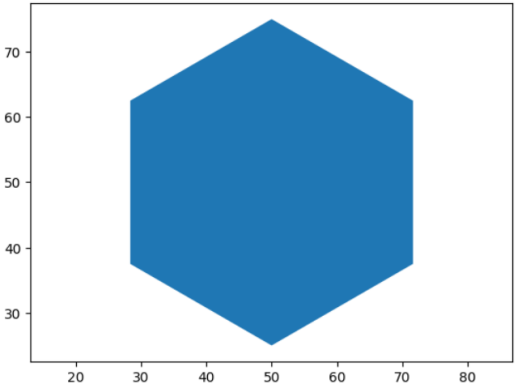
b) Ein Graph beschreibt die Topologie, ein Polygon beschreibt die Geometrie.

c) Ein Polygon ist auch ein Graph.

d) Ein Graph kann auch Informationen über die Geometrie haben.



Graph



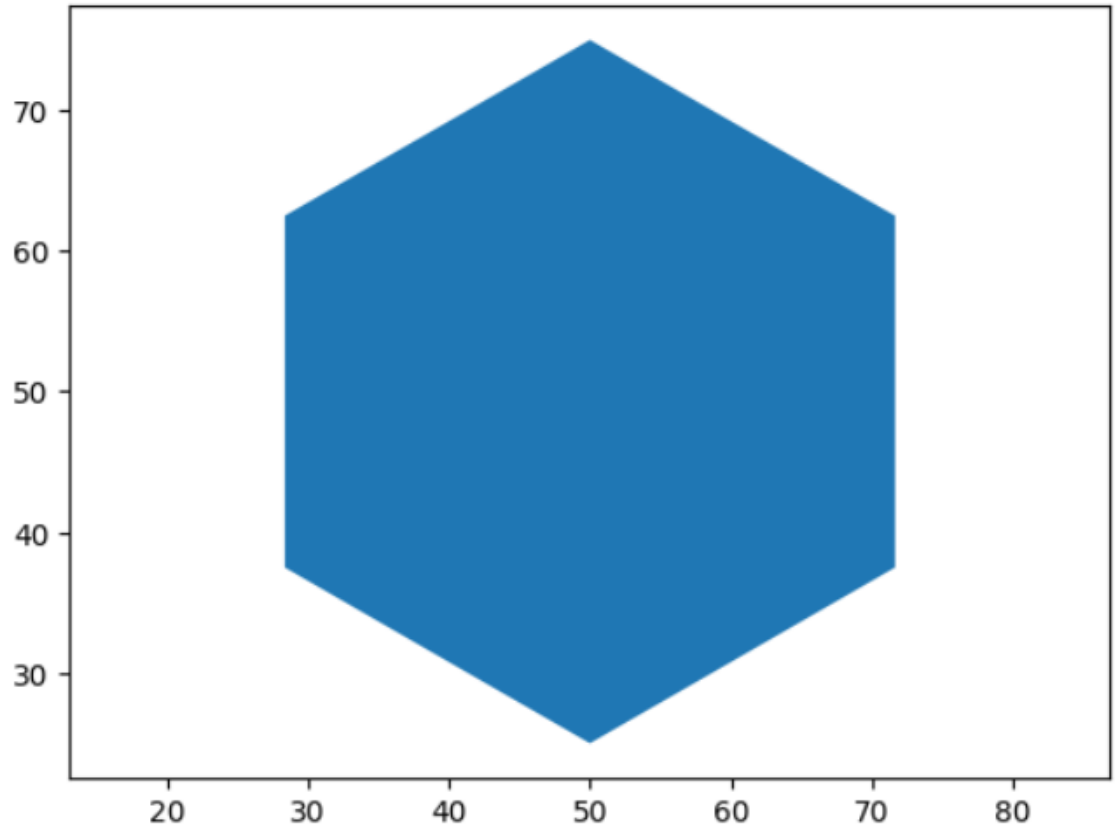
Polygon

(Technical) Drawing with matplotlib: Patches

- Matplotlib is a very strong toolset.
- It actually allows some kind of „drawing (geometric) shapes“.
- Instead of the expresion shape matplotlib is using the expression patch. Patches are
  - Rectangle
  - Circles
  - Polygon

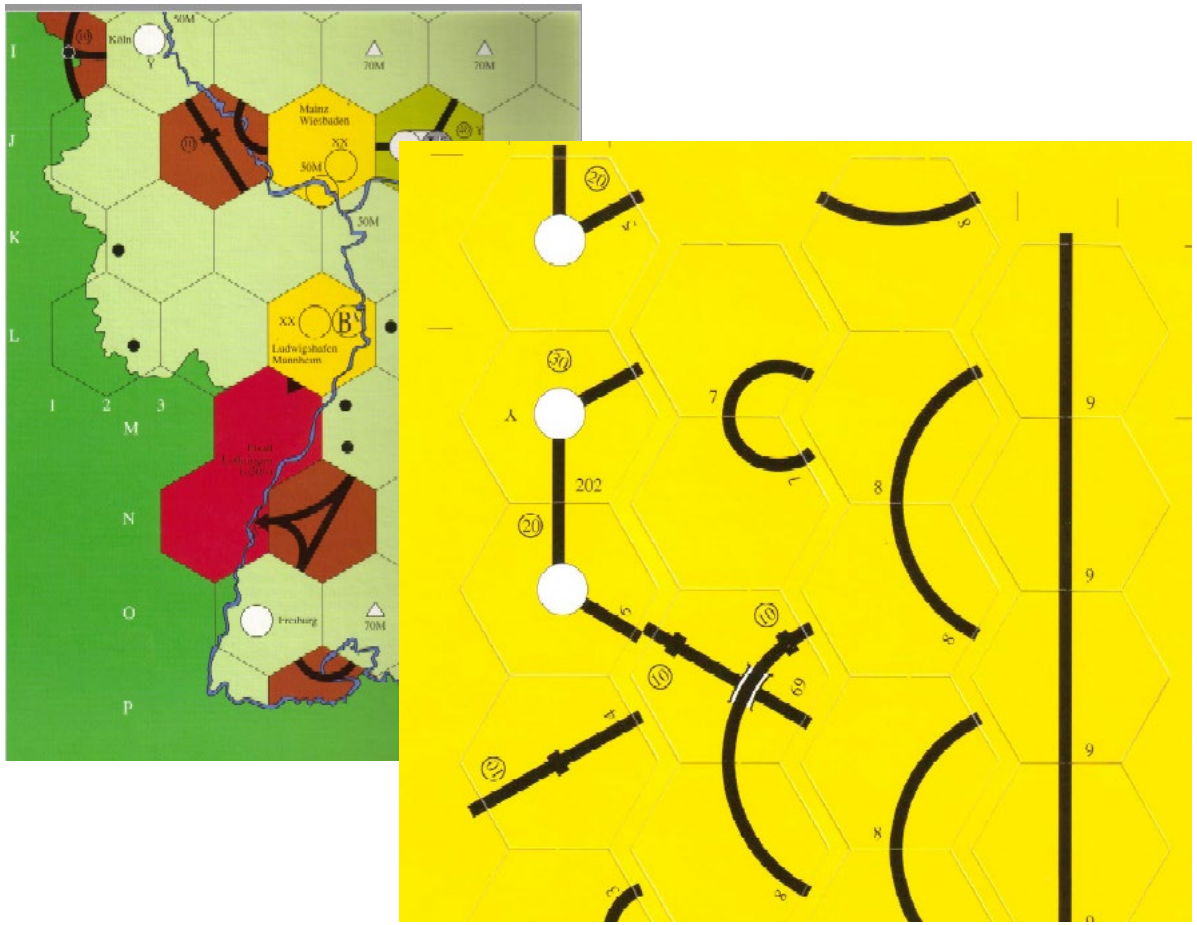
- Remark: there are better libraries on the market than matplotlib for drawing shapes. Matplotlib is best choice when using Jupyter Notebooks on DataHub.

- Alternative: Qt
  - https://www.qt.io/ (general information)
  - https://doc.qt.io/qtforpython-6/index.html (Python library)



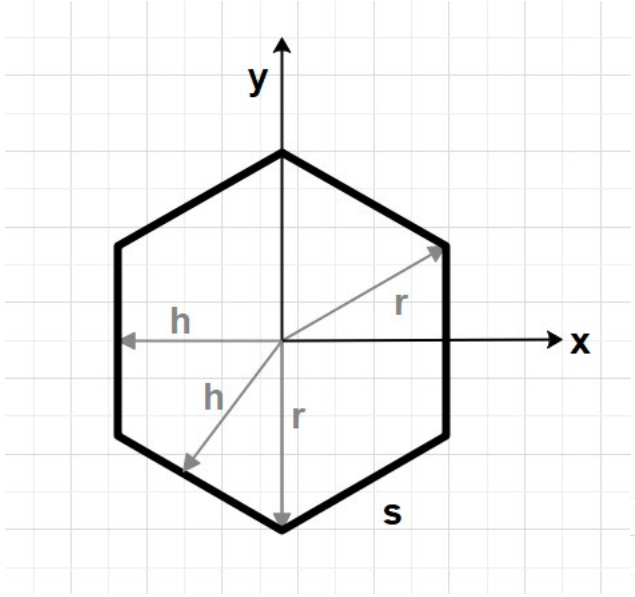
The 18xx Games

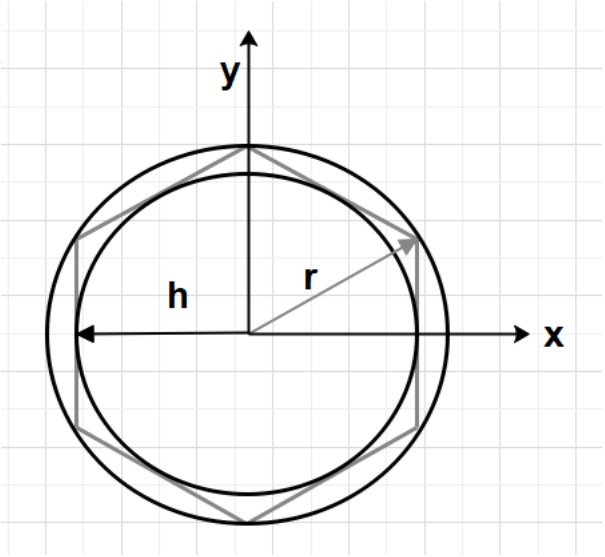
- Series of strategy games topic „railroad management“
- Created in the 1990s
- Many versions
  - 1830 US
  - 1835 Germany
  - Many more
- Based on history
- Lot of calculations
- Board is based on hexagon elements with tracks and stations on it.
- Average time for a single game: 10h



Geometry of a hex-Tile

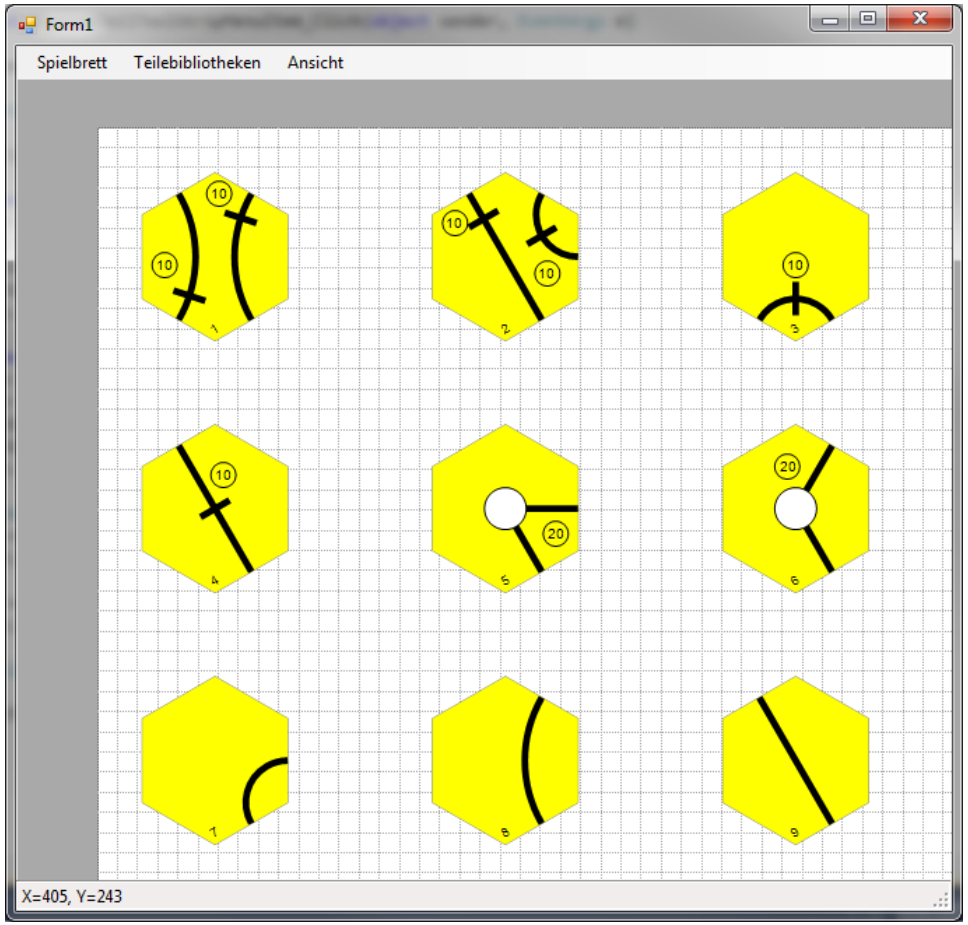
- The size of the tile is defined by the length of the side: s
- From basic geometry (circle, triangular) we are able to calculate r and h out of s:
  – r = s
  – h = r / 2 * sqrt(3)
- The values for r and h are the radians of the inner and outer circle of the hexagon.
- In case we „know“ the coordinates of the center of the hexagon, we are able to calculate the coordinates of the „describing“ elements
  – the corners
  – the middles of the edges
- We need these values for drawing the shape as well drawing the tracks.





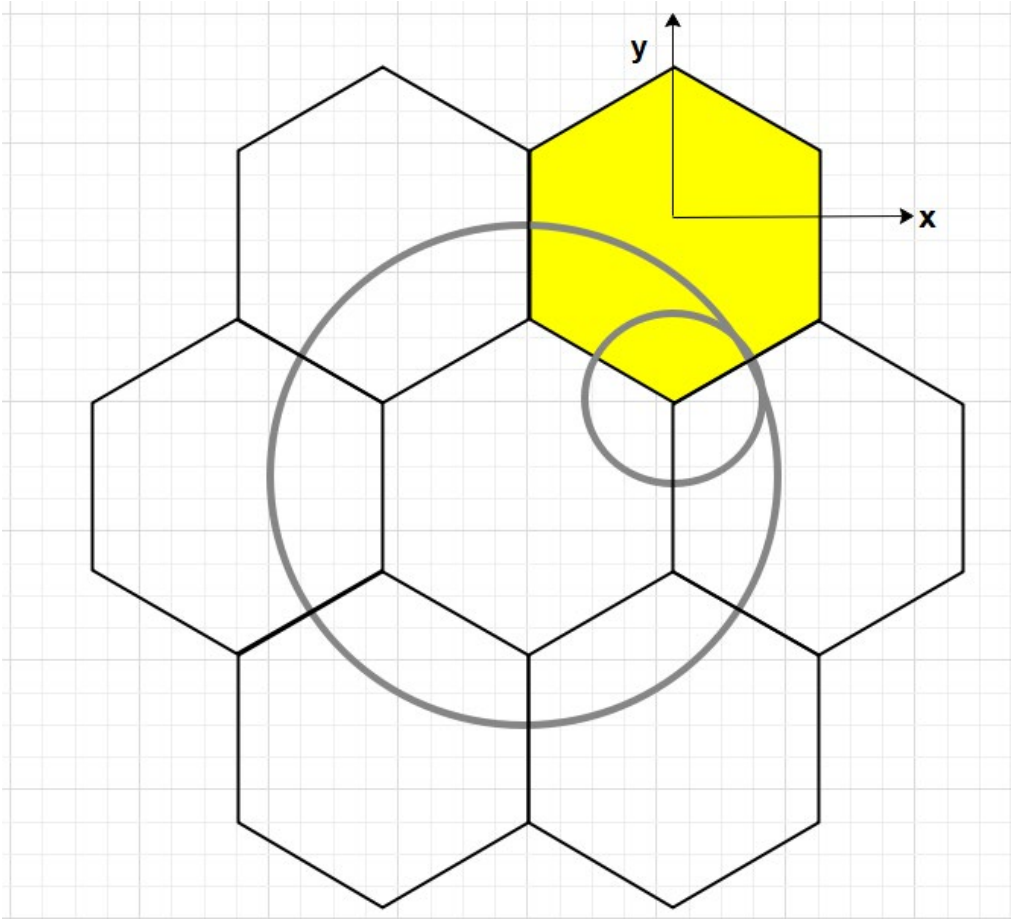
Examples

- From another program: some „track tiles“



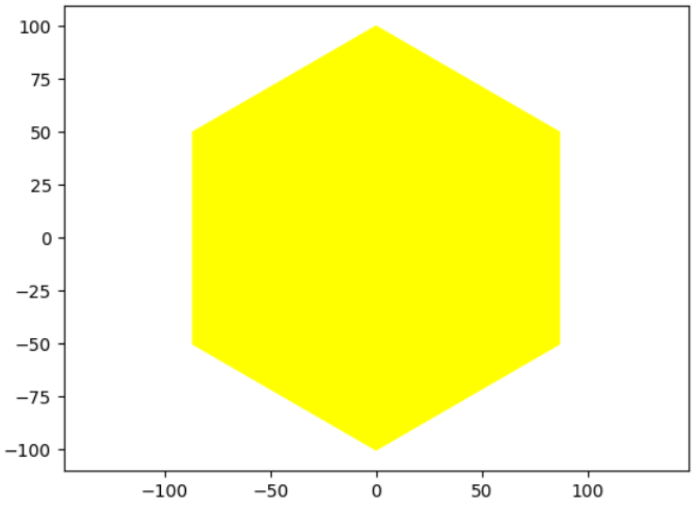
Construction of the tracks

- As we can see on the right: haxagons are beautiful for drawing.
- Again, all we need can be described from size s
- The radian of the smaller circle: r = s / 2
- The radian of the larger circle: r = 1.5 * s
- The center of the smaller circle = (0, -s)
- The center of the larger circle = (-s, -1.5 * s)
- All we have to do is „putting the elements together“



The basic polygon – made out of the corners

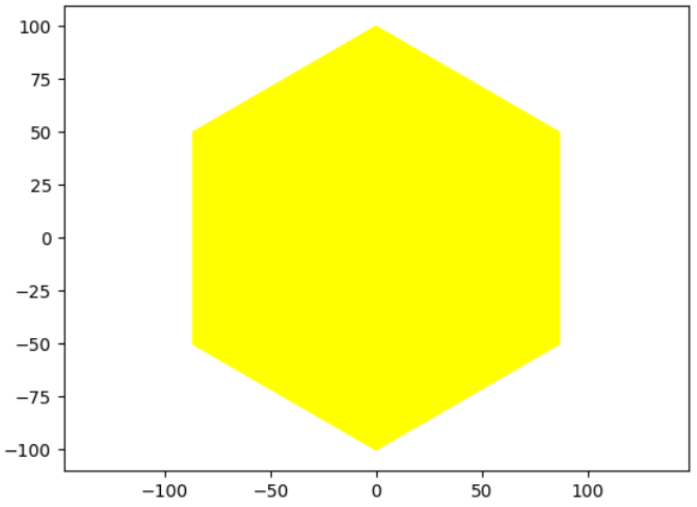
- Line 5 – 7: the „inputs“
- Line 9 – 11: calculating the values of the describing elements
- Line 14 -15: calculating the values of the coordinates of the corners
- Line 18: creating the polygon
- Line 19: drawing the polygon as patch




```
1   import matplotlib.pyplot as plt
2   from math import sqrt
3   from matplotlib.patches import Polygon
4
5   s = 100                 # size of the elements (no unit!)
6   mx = 0                  # center of the tile
7   my = 0                  # center of the tile
8
9   r = s                   # Radian outer circle
10  r2 = 0.5 * r            #
11  h = r2 * sqrt(3)        # radian inner circle
12
13  # corners starting at the top, clockwise
14  corners = [(mx + 0, my + r), (mx + h, my + r2), (mx + h, my - r2),
15          (mx + 0, my - r),  (mx - h, my - r2), (mx - h, my + r2)]
16
17  fig, ax = plt.subplots()
18  p1 = Polygon(corners, color="yellow")
19  ax.add_patch(p1)
20
21  plt.axis('equal')
22  plt.show()
23
```

The basic polygon – made out of the corners

- Line 5 – 7: the „inputs“
- Line 9 – 11: calculating the values of the describing elements
- Line 14 -15: calculating the values of the coordinates of the corners
- Line 18: creating the polygon
- Line 19: drawing the polygon as patch




```
1   import matplotlib.pyplot as plt
2   from math import sqrt
3   from matplotlib.patches import Polygon
4
5   s = 100                 # size of the elements (no unit!)
6   mx = 0                  # center of the tile
7   my = 0                  # center of the tile
8
9   r = s                   # Radian outer circle
10  r2 = 0.5 * r            #
11  h = r2 * sqrt(3)        # radian inner circle
12
13  # corners starting at the top, clockwise
14  corners = [(mx + 0, my + r), (mx + h, my + r2), (mx + h, my - r2),
15             (mx + 0, my - r),  (mx - h, my - r2), (mx - h, my + r2)]
16
17  fig, ax = plt.subplots()
18  p1 = Polygon(corners, color="yellow")
19  ax.add_patch(p1)
20
21  plt.axis('equal')
22  plt.show()
23
```

Learning programming = Learning decoupleling problems

- Question: in case I would like to draw a lot of polygons, what are the lines of code I need to do only this?

- Answer
  - Line 9 to 15 for really „calculating“ the corners
  - Line 18 für creating the Polygon

- The other lines?
  - Line 5 to 7 are just defining size and position of a single Polygon
  - Line 17 and 18 can be switched

```
1   import matplotlib.pyplot as plt
2   from math import sqrt
3   from matplotlib.patches import Polygon
4
5   s = 100                 # size of the elements (no unit!)
6   mx = 0                  # center of the tile
7   my = 0                  # center of the tile
8
9   r = s                   # Radian outer circle
10  r2 = 0.5 * r            #
11  h = r2 * sqrt(3)        # radian inner circle
12
13  # corners starting at the top, clockwise
14  corners = [(mx + 0, my + r), (mx + h, my + r2), (mx + h, my - r2),
15          (mx + 0, my - r),  (mx - h, my - r2), (mx - h, my + r2)]
16
17  fig, ax = plt.subplots()
18  p1 = Polygon(corners, color="yellow")
19  ax.add_patch(p1)
20
21  plt.axis('equal')
22  plt.show()
23
```

Introducing functions

- Defining a function requires three decisions
  - What is the name of the function?
  - What are the parameter of the function for giving information into the function (from the caller)
  - What is tthe function returning (to the caller)

- The function has a general purpose. It is working for very values of mx, my, and s

- Line 5: name and parameter

- Line 15: returning the polygon

- Line 17: begin of the „main“ programm

- Line 19: Calling the function drawTile

- Remark:
  the code of the function can be placed in an external file or module. It is then imported like the other libs in line 1 - 3

```
1  import matplotlib.pyplot as plt
2  from math import sqrt
3  from matplotlib.patches import Polygon
4
5  def drawTile(s, mx, my):
6      r = s                    # Radian outer circle
7      r2 = 0.5 * r             #
8      h = r2 * sqrt(3)         # radian inner circle
9
10     # corners starting at the top, clockwise
11     corners = [(mx + 0, my + r), (mx + h, my + r2), (mx + h, my - r2),
12                (mx + 0, my - r),  (mx - h, my - r2), (mx - h, my + r2)]
13
14     p = Polygon(corners, color="yellow")
15     return p
16
17 fig, ax = plt.subplots()
18
19 p1 = drawTile(100, 0, 0)
20 ax.add_patch(p1)
21
22 plt.axis('equal')
23 plt.show()
24
```

Functions are somekind of sustainable: we are reusing code

- Calling a function again and again with different values for the parameter.
- Creating a „complex“ picture with a simple program.
- Can we make it more simple?
- How to use the code for drawing a playing board?

```
16
17  fig, ax = plt.subplots()
18
19  p1 = drawTile(100, 0, 0)
20  ax.add_patch(p1)
21
22  p2 = drawTile(100, 200, 0)
23  ax.add_patch(p2)
24
25  p3 = drawTile(100, 100, 200)
26  ax.add_patch(p3)
27
28  plt.axis('equal')
29  plt.show()
30
```

A playing board

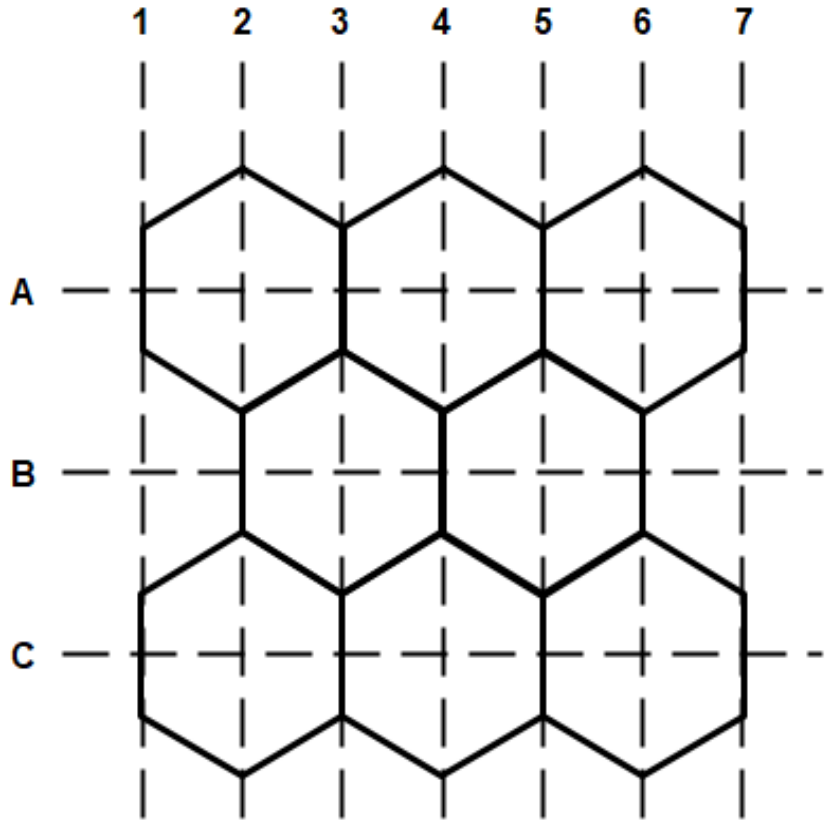
- Horizontal gap between axis 1. 2. 3. etc

  dx = 0.5 * sqrt(3) * s

- Vertical gap between axis A, B, C

  dy = 1.5 * s

- Within a row the middle of tiles have the distance of 2 * dx
- The middles for two rows have the distance of dy
- In odd rows (B) there is an
  - Offset of dx
  - The number of tiles is one less than in even rows (A, C)



Drawing a board

- Line 20 – 22: defining the size of the board
- Line 25 – 29: calculating the row specific parameter
- Line 22 – 23: calculating the coordinates of the middles

- A board is constructed out of rows and columns => 2 loops for iterating through all the positions

- Question: why is the a factor 0.9 in line 34???

```
18  fig, ax = plt.subplots()
19
20  nRows = 3       # number of rows
21  nCols = 3       # number of columns
22  size = 100      # size of a tile
23
24  for i in range(0, nRows,1):  # every row
25      offset = 0
26      cols = nCols
27      if i % 2 != 0:          # odd rows are different
28          offset = 0.5 * sqrt(3) * size
29          cols = nCols - 1
30
31      for j in range(0, cols, 1):     # every column in the row
32          center_x = j * 0.5 * sqrt(3) * size * 2 + offset
33          center_y = i * 1.5 * size
34          p = drawTile(size * 0.9, center_x, center_y)
35          ax.add_patch(p)
36
37  plt.axis('equal')
38  plt.show()
```

Drawing a playing board – the result