

Performs the symmetry analysis of a Flex Tile.

# Flex Tile Analyzer

Takes any correctly formatted .csv or .json files as input. You can download the newest version of the application in the [Downloads](#) section.

## Facts

- The input matrix has to be quadratic (NxN), N can be any number starting with 1 (up to the limit of [Time Complexity](#)).
- Only diagonally oriented tiles are supported!
- Do not edit the CSV data with Excel or OpenOffice. This edition changes the number of commas and disables a successful read-out. If there is a need of manual edition use a simple text editor instead ([notepad](#), [Vi](#), [nano](#)...)
- Prepare data accordingly, not-a-number values will be substituted with zeroes. Invalid data can cause errors.
- both binary and by-90-rotations are supported. Change the binaryFlag in the [FlexTileAnalyzer](#) on [line 28](#).
- one prespecified tile makers symmetry is supported. It is the (0, 90, 180, 270) and its rotations.

## theDict

All types of file inputs are converted into a dictionary *theDict*. Each flextile is labelled with a unique dictionary index value based on its: filename; tile Nr; and the tile repetition Nr.; The whole flextile is converted into an array of integer values (for the angles) and saved in the dictionary. The analyzer then processes the dictionary.

## Input

### JSON

This is the older version of the [FlexTiles](#) output. [FlexTileAnalyzer](#) works with the "F" file. The data has to be pre-processed accordingly. "I" and "S" files are ignored.

Logic:

1. open the json file and read it into a dictionary. For further details read <http://docs.python.org/library/json.html>
2. [delete](#) "totalClicks", this is not a number and therefore cannot be sorted correctly. This variable also adds no value to the analysis process.
3. sort the values in the json dictionary in an ascending order
4. fill theDict with the angle values from the json dictionary
5. return *theDict*

### CSV

Optimized for the output of the [FlexTiles](#) process.

Logic:

1. Search for a line containing *Final* and starts with a #. All whitespace is trimmed, therefore any indentation or spacing can be ignored.
2. create a dictionary index based on the numbers of the *Final* csv line.
3. Skip the header line below and continue with reading the angles.
4. Read only lines with three entries. TotalClicks, initiation state or clicks have a different number of columns (entries). Skip those. If the CSV is edited with an application such as [OpenOffice](#) or MS Excel, all lines have a fixed number of columns. The reader will therefore fail.
5. fill theDict with the angle values from the selected column.
6. return *theDict*

## Print

- Parsed list of angles. Enables you to check whether the parsed data corresponds with the image.
- Shannon Entropy *S* of the data. 0= one perfectly dominant orientation of the tiles. 2 = perfect distribution of all four

orientations.

- Tile Makers Symmetry. 0= no blocks of the prespecified type. 1= perfect division into 2x2 blocks.
- Ratio of orientations. Ratio= SumForTheGivenOrientation/SumOfAllTiles. 0= no flextile is oriented towards this direction. 1 = all tiles show this orientation.
- Translational Symmetry:  $1 - (S/2)$ ; where  $0 < S < 2$ . Translational symmetry of 0 equals perfect distribution of tile orientations .Could also be denoted as 0.25, because at least 25% of the data will always have the dominant orientation. Yet the result has been normalized to 0,1. 1 = one perfectly dominant tile orientation. Which one it is can be read from the *Orientation Ratio*.
- Symmetry Axis
- Rotational Symmetry
- Draw the matrix

## Output

Creates a new [CommaSeparatedValues](#) (csv) file named *analyzedFlexTiles* with a date suffix. Example: analyzedFlexTiles\_28Sep2011-10-08-25.csv

Each input data file is identified by its own unique number in the first column of the file.

Order of Values:

- Unique ID
- [Shannon Entropy](#)
- [Tile Maker Symmetry](#)
- [Symmetry Axis](#)
- [Rotational Symmetry](#)
- [Translational Symmetry](#)

A set of functions for the FlexTileAnalyzer.

## Flex Block

A set of functions for the FlexTileAnalyzer.

## Overview

- [Symmetry Axis](#)
- [Tile Makers Symmetry](#)
- [Rotational Symmetry](#)
- [Translational Symmetry](#)
- [getOrientationCountsBy90Increment](#)
- [getRatioOfEquivalenceByRotation](#)
- [getDifferenceInPalindromeString](#)

---

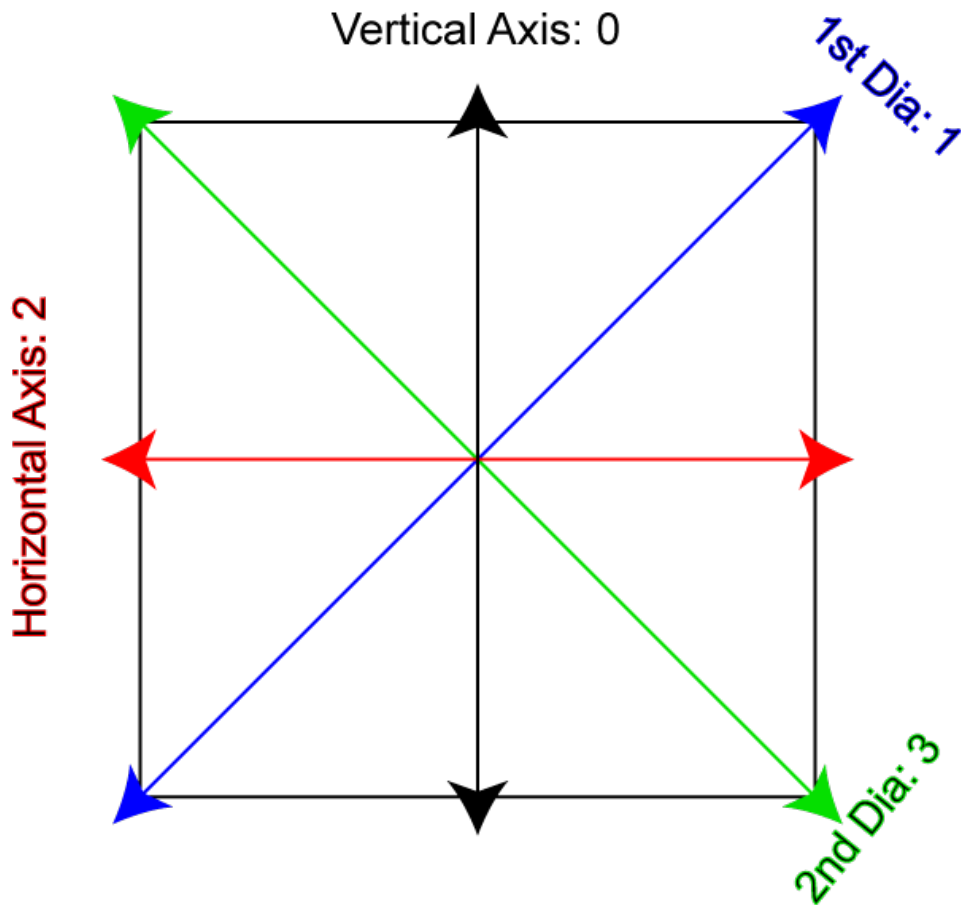
[FlexTileAnalyzer](#)

Axes Conventions

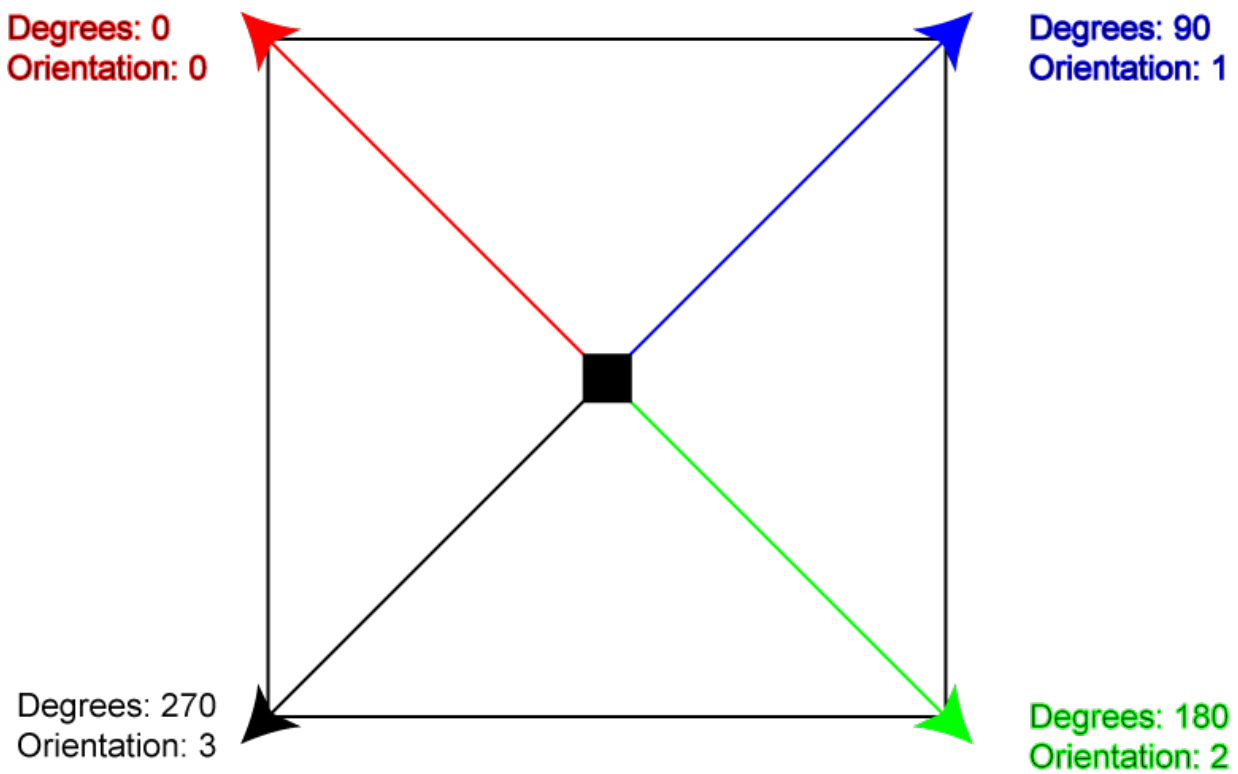
## Axes of Symmetry

Definition of the convention of axis numbering.

- 0/4: Vertical
- 1: 1st Diagonal
- 2: Horizontal
- 3: 2nd Diagonal



## Orientation of Flex Tiles



Definition of the [AngleAware](#) term

## Terminology

The term Angle Aware is used in this application to specify the change of values rather than position of flex tile orientations in the matrix.

**Position** is changed by the *rotate* function. This shifts the values in the matrix towards the right. The values themselves are not changed.

**Value** is changed by the angle aware function. The specification of the axis that the values are compared along is important in this case. Currently all values are uniformly rotated towards the right (0 becomes 90; 270 becomes 0).

Both functions are required to detect the rotational symmetry of the block.

In the application logic, if no axis is specified for the *rotate* function, it will not apply the angle aware rotation.

---

[RotSym](#) - [FlexBlock](#) - [FlexTileAnalyzer](#)

Computes values for 4 different axes of symmetry

= [SymAxis](#) =

This is a guide to the getXSymmetry Functions from the [FlexBlock](#) file.

## Details

### General

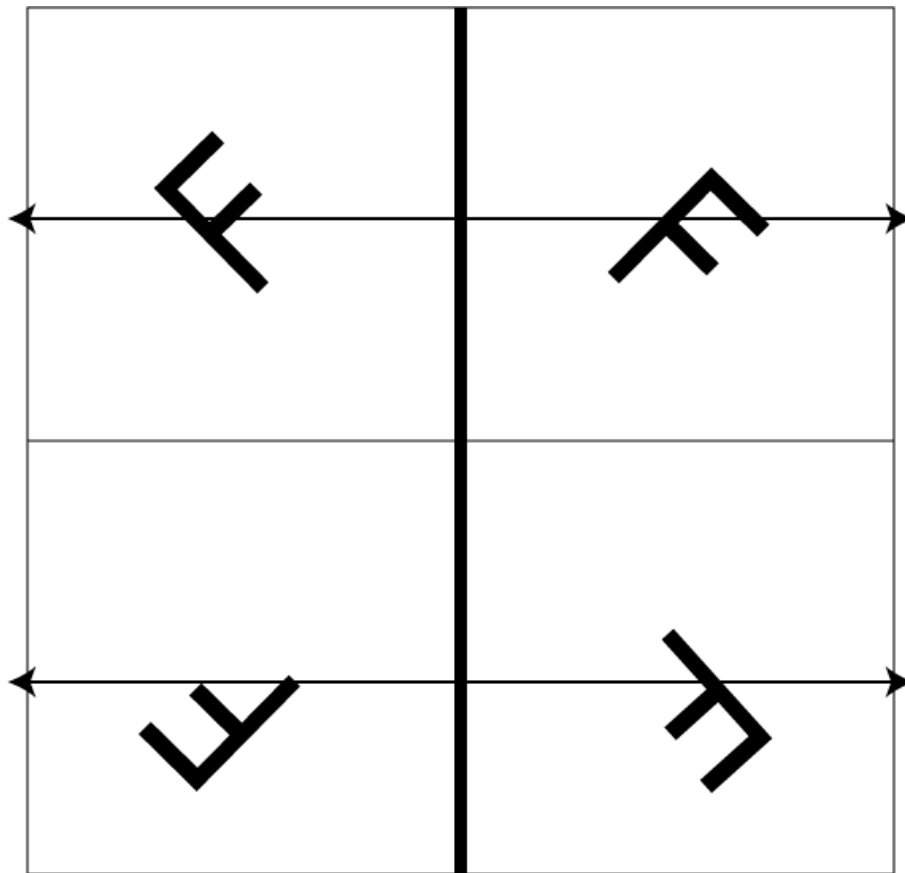
- All four functions only take a square matrix.
- The input is a list of tile orientations.
- The output is a number between 0 and 1.
  - Output = 1 - (sum of errors in palindromes)/(half the amount of tiles)
  - 0 denotes maximal error and therefore no symmetry.
  - 1 denotes minimal error and therefore a perfectly symmetrical matrix.
- Is based on the [getDifferenceInPalindromeString](#)
- For axis specification refer to [Axes](#)

### Vertical (0)

Every row must be a palindrome.

Mirroring:

- 0 to 90
- 180 to 270

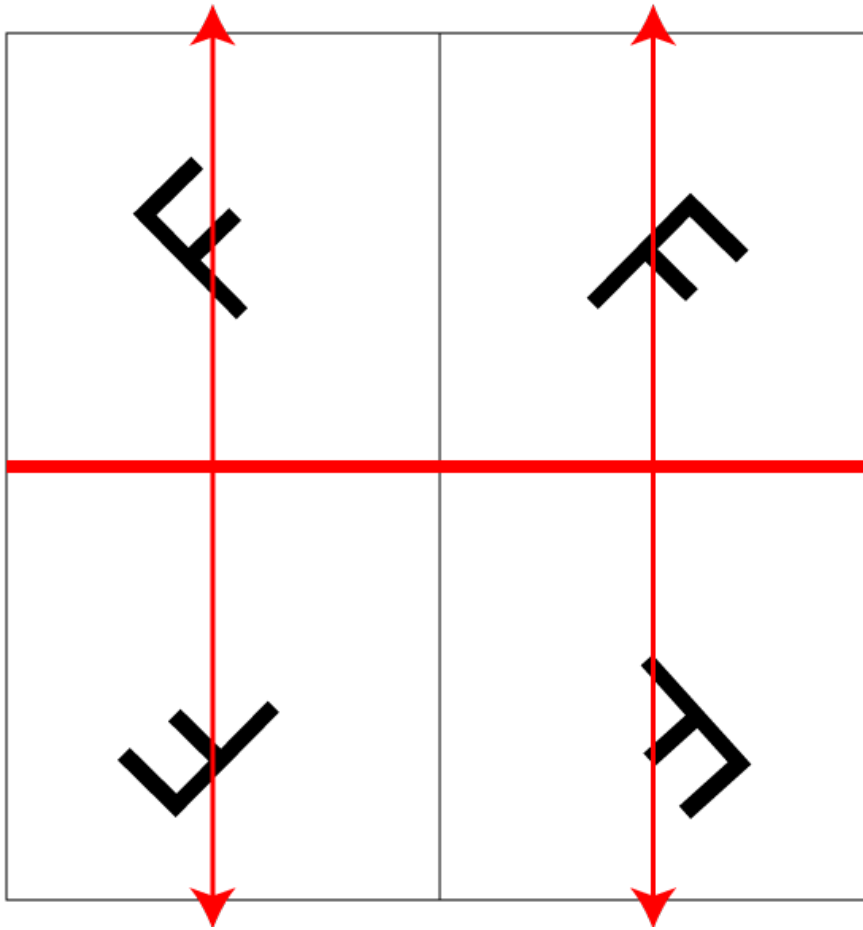


## Horizontal (2)

Every column must be a palindrome.

Mirroring:

- 0 to 270
- 90 to 180



## 1st Diagonal (1)

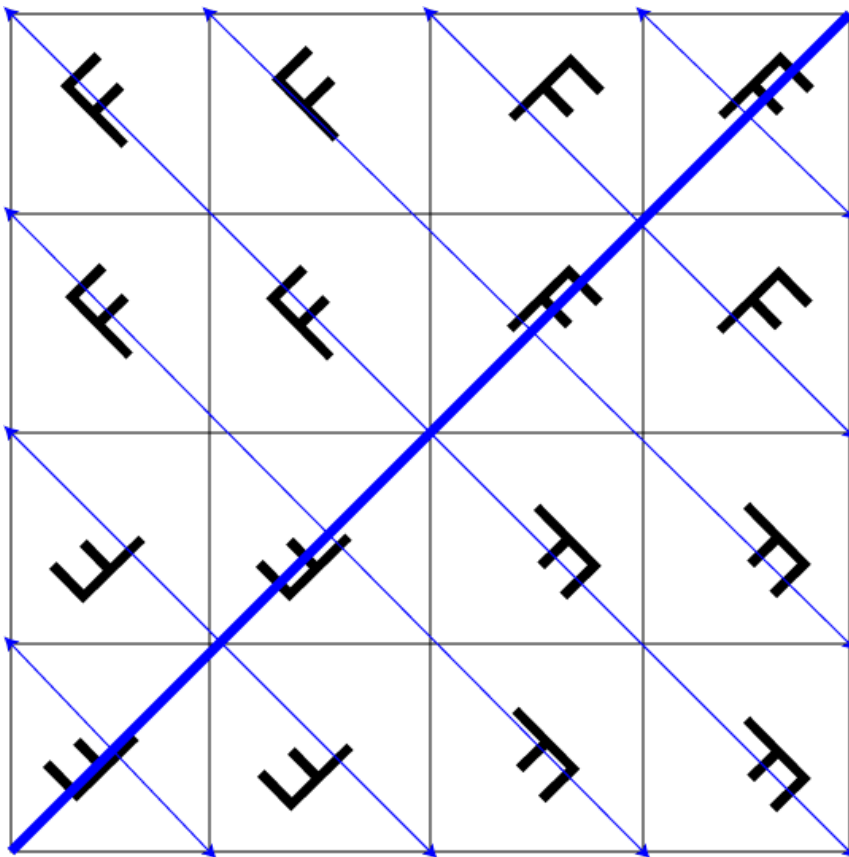
Lines starting with the first row and first column oriented towards the lower right corner must be a palindrome.

Mirroring:

- 0 to 180

Remains the same:

- 90,270



## 2nd Diagonal (3)

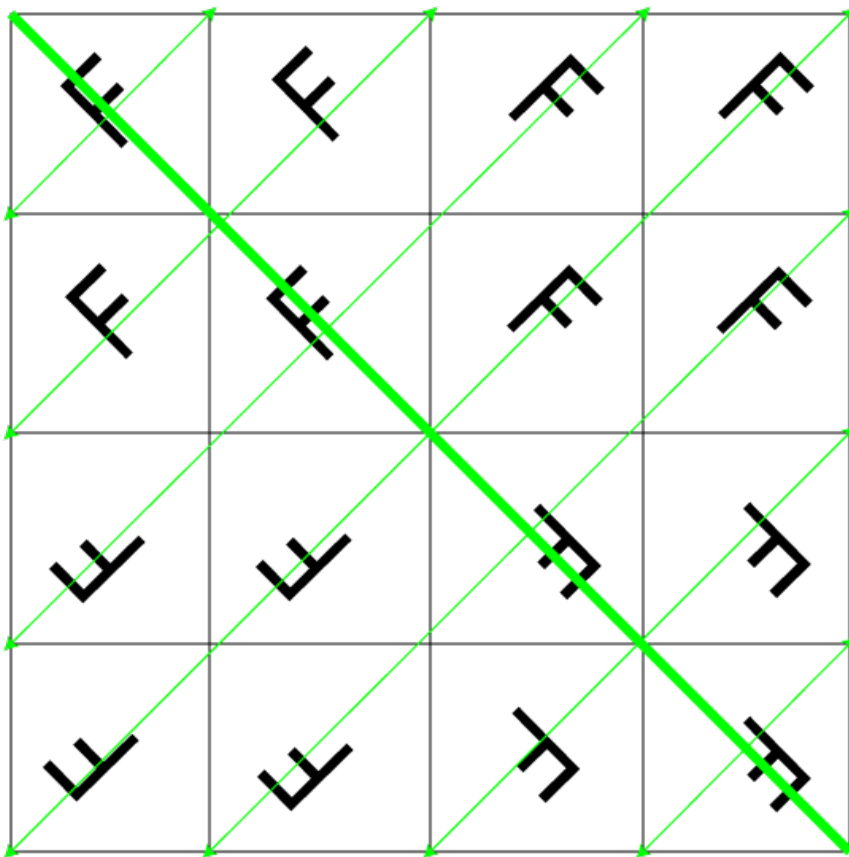
Lines starting with the first column and last row oriented towards the upper right corner must be a palindrome.

Mirroring:

- 90 to 270

Remains the same:

- 0,180



[FlexTileAnalyzer](#) - [FlexBlock](#)

The [TileMakers](#) Detection Algorithm

## Tile Makers Symmetry

This is a guide to the `getTileMakersSymmetry` function from the [FlexBlock](#) file.

### Details

Currently the only structure that is searched for, together with its rotations, is the:

0 90  
270 180

- Possible squares are also located *across the border*.
- The maximal amount of matches in a  $N \times N$  sized matrix is  $(N/2 \times N/2)$ .
  - a matrix of  $4 \times 4$  will have 4 possible matches
  - a matrix of  $5 \times 5$  will have  $2.5 \times 2.5 = 6.25$  matches.
- A full match including all 4 squares can only be found inside the matrix. It gives 1 match point.
- A half match at the borders of the flextile award  $1/2$  match point if found.
- A quarter match is found in the edges of the flextile.
- Only a perfect match is awarded a point.

### Example

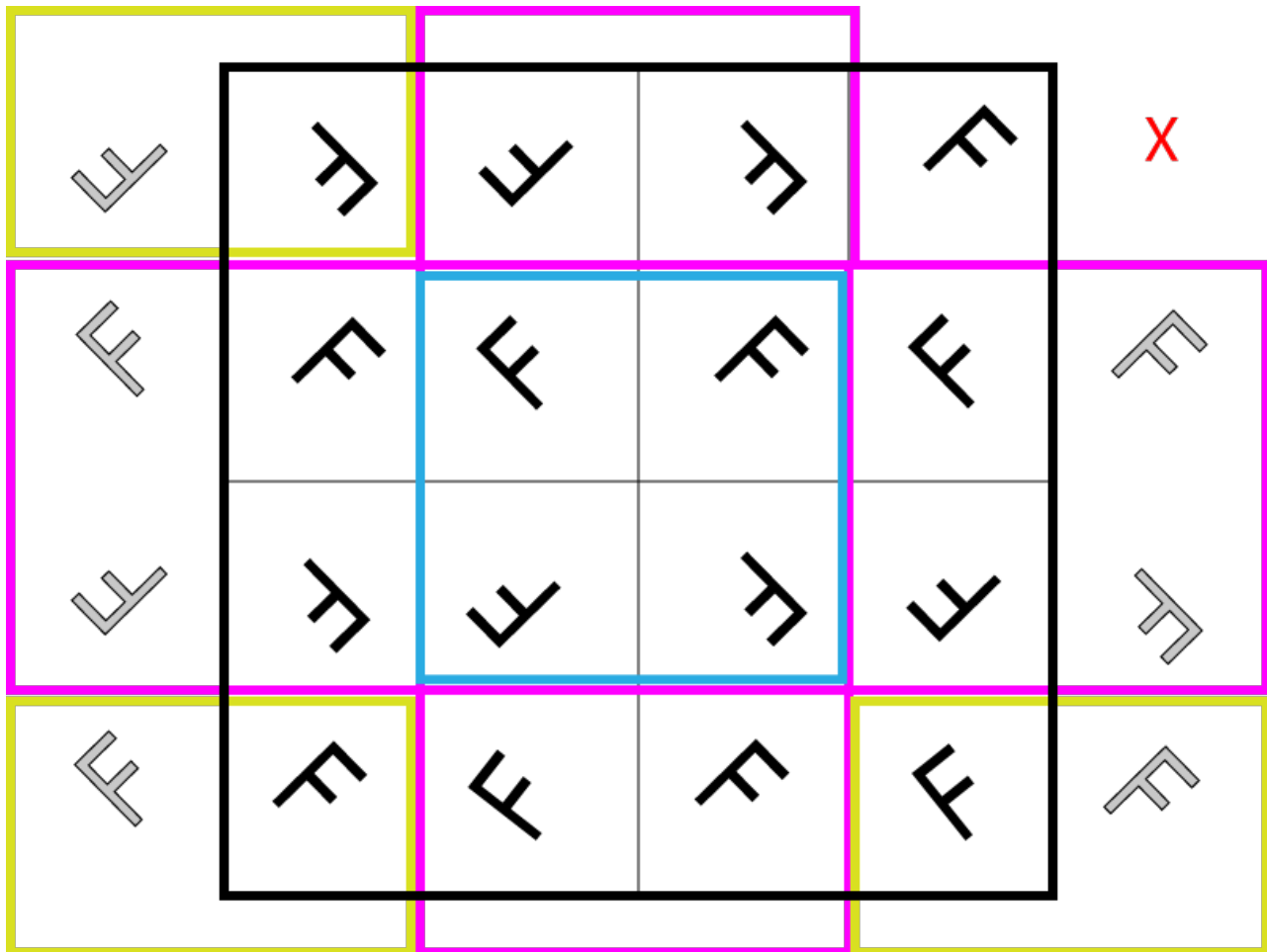
Explanation of the image below:

- Black F's denote orientations of a block. These are only found inside the flex tile.
- The gray F's denote possible extensions to the flex tile, that would be classified as tile makers tiles. They are not found



in the data but could be there if the tile would be bigger.

- The black area is the space of the data.
  - The blue area is classified as a full match. It gives the tile 1 full point.
  - The purple areas are only classified as a match if they are on the correct border of the tile. The gray F's indicate possible extensions and therefore such as a purple match gives only 1/2 match point.
  - At the corners of the tile there are possible yellow areas. An yellow area is exactly specified. The left upper corner has to match the lower right corner of the selected prespecified tile makers block. In this case (0,90,180,270; numbered clockwise). The upper left corner has to be a 180 orientated tile. Etc. Such as an match awards 1/4 match point.
  - The red X indicates a block that matches no possible orientation. Awards 0.
- in summary a 4x4 tile has a maximum of 4 full matches (2x2).
  - 1 full match = 1 point; 4 half matches = 2 points, 3 quarter matches = 3/4 points. Therefore the tile has a Tile Makers sym. value of  $3.45/4 = 0.8425$ .



[FlexTileAnalyzer](#) - [FlexBlock](#)

2x1 and 2x2 rotational symmetry

## Rotational Symmetry

This is a guide to the getRotationalSymmetries function in the [FlexBlock](#) file.

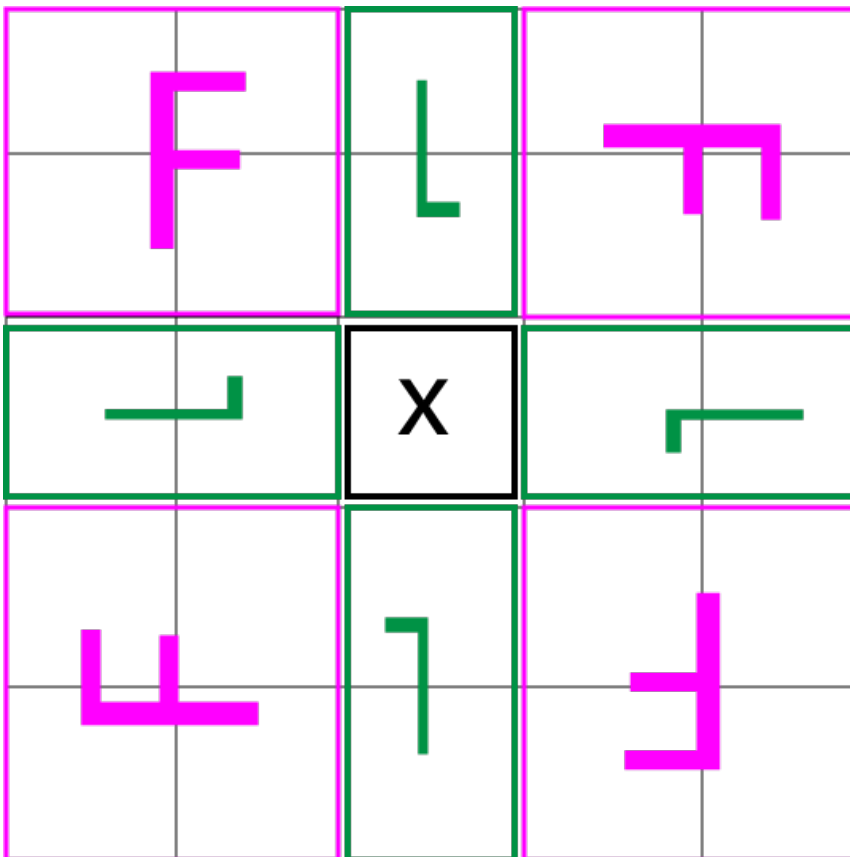
### Details

- This function only accepts a set of four equally sized square matrices (NxN).
- The input is a list of tile orientations.
- The output is a number between 0 and 1.

1. 2x1 block division
  2. 2x2 block division
  3. 0 denotes maximal error and therefore no symmetry.
  4. 1 denotes minimal error and therefore a perfectly symmetrical matrix.
- Is based on the [getRatioOfEquivalenceByRotation](#)

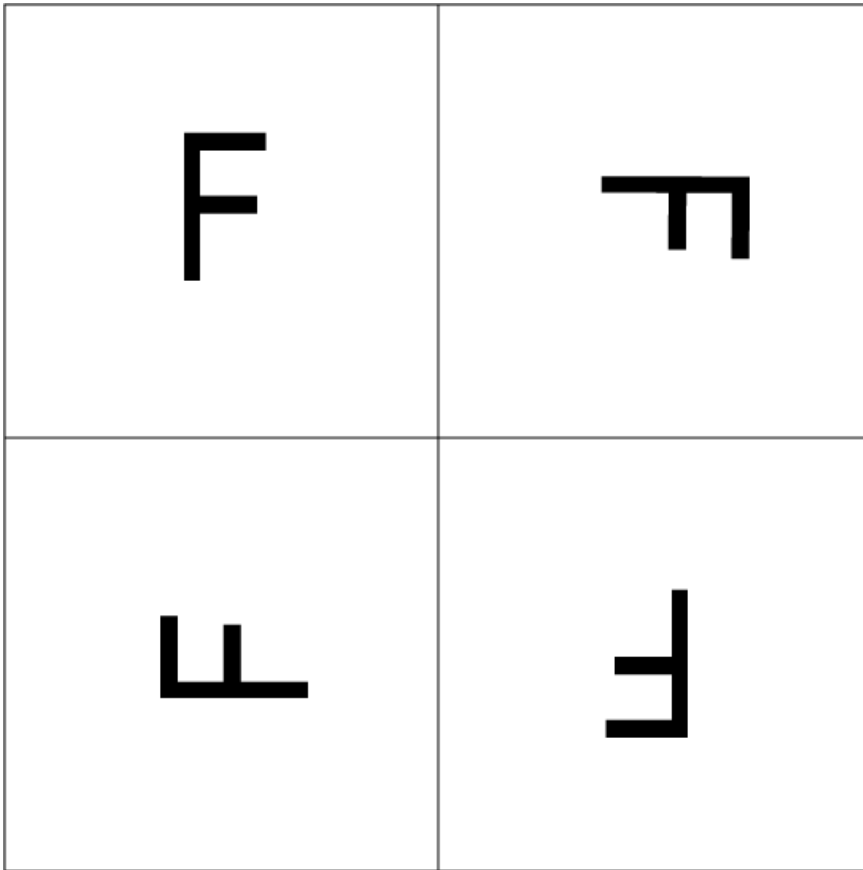
Logic:

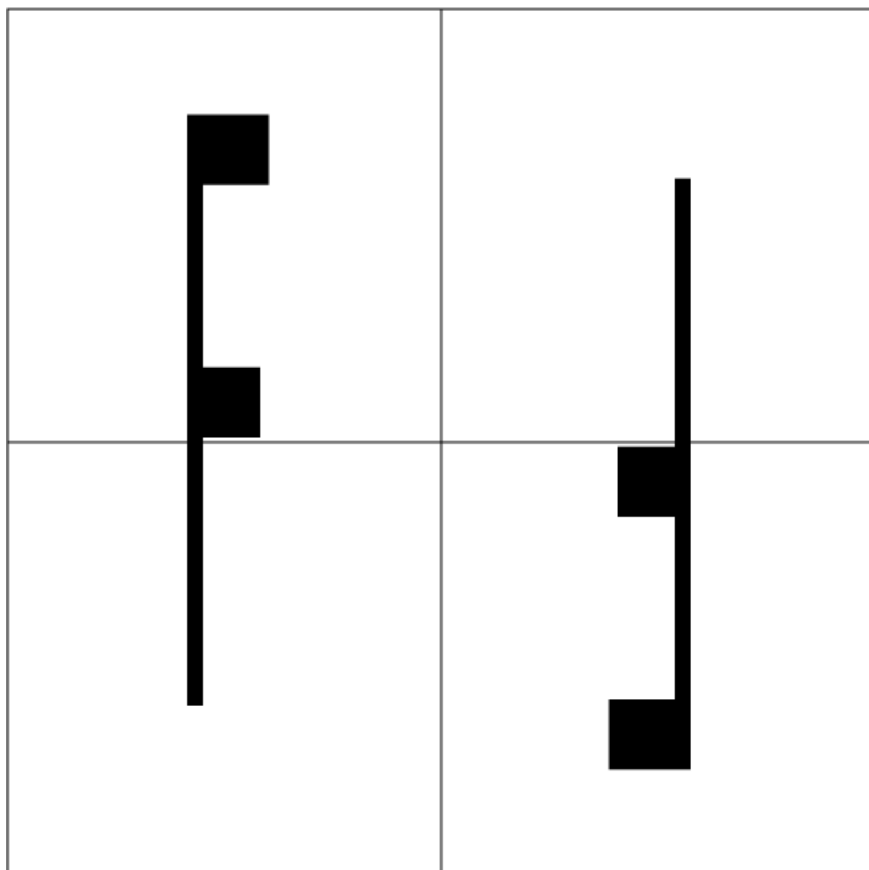
1. The matrix is first divided into four equally sized blocks. In the program they are denoted *top left*, *top right*, *bottom left*, *bottom right*. This is done by the *divideMatrixIntoTwoTimesTwo* function.
2. Each block is rotated to a zero orientation. The angle of block rotation is dependent on its position in the full matrix. See [angleAware](#)
3. Based on the [getRatioOfEquivalenceByRotation](#) function, the similarity of the block is evaluated.
  - To calculate the 2x1 rotational symmetry we only need to compare the blocks along the diagonals. We know that a symmetrical image that is divided 2x1 will have an equal output value to one that is divided 1x2.
  - To calculate the 2x2 rotational symmetry we have to compared all blocks with all other blocks. Therefore a total amount of  $(n-1)!$  comparisons will be required. Because  $n=4$ , we have 6 comparisons.
4. The sum of equivalence ratios for all comparisons is computed and returned as output.
5. Matrices with an odd number of lines use an extended procedure. The main blocks (denoted purple) are calculated equally to the evenly sized matrix procedure. Middle blocks (green) are compared to each other in an extra loop. Their contribution to the overall ratio of symmetry is one divided by the number of lines or columns in the main (purple) block. A 5x5 matrix has 2 main block lines and therefore the contribution of the middle lines is 1/2 of the contribution of the main block. The true middle is ignored.
6. The middle lines are ignored in the 2x1 rotational symmetry!!!



## Examples

### 2x2 block rotational

**2x1 block rotational**




---

[FlexTileAnalyzer](#)

Explanation of the translational symmetry

## Translational Symmetry

Translational symmetry is a measure based on the [ShannonEntropy](#) found in the [FlexBlock](#) and [FlexTileAnalyzer](#) files.

### Details

Entropy facts for the flex tiles:

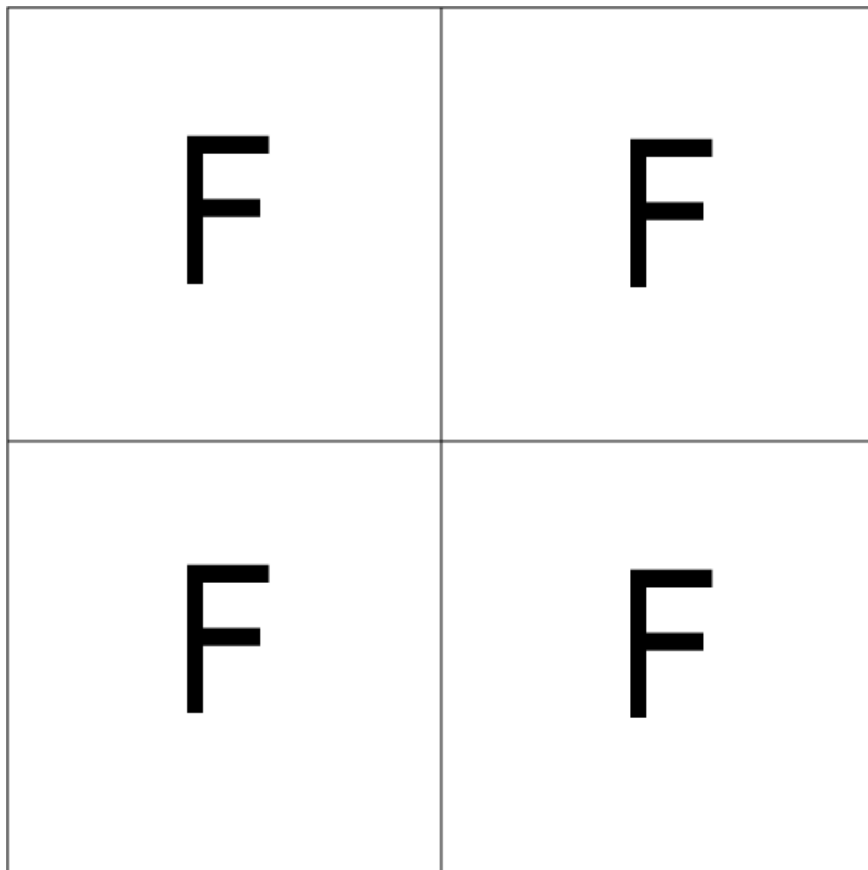
- there are four possible orientations in a flex tile block.
  - the more equally the probabilities of occurrence for each separate orientation are distributed, the higher the entropy.
- Examples:
- (0,0,0,1) yields an Entropy measure of 0 bits (no information is required to encode the block, because there is no variation).
  - (0.5, 0.5, 0, 0) yields an Entropy measure of 1 bits (you have to choose between two equally probable options)
  - (0.25, 0.25, 0.25, 0.25) has the highest possible Entropy of 2 bits.
- therefore the Entropy of the flex tile is in the range 0 to 2.

The function:

- has to be normalized to the 0 to 1 range.
- is inverse to the Entropy. 0 denotes no symmetry and 1 denotes uniform tile orientation.

- therefore the function is,  $TS = 1 - (\text{Entropy}/2)$

## Example




---

[FlexBlock](#) - [FlexTileAnalyzer](#)

Calculates the Entropy for a list of probabilities

## Shannon Entropy $S$

The Shannon Entropy  $S$  specifies the number of bits that are required to represent a given string of values. The more probable one of the options is, the lower the entropy measure will be. If all of the options are equally distributed, the entropy is maximal.

## Implementation

Takes a list of probabilities  $p$  of  $X_i$  as input. Their sum must not be 1.0, as we are dealing with approximated float values. For further information on machine number approximation refer to [Wiki](#)

Returns the Shannon Entropy in bits.

The Entropy is used as a measure in the [Translational Symmetry](#)

---

<http://upload.wikimedia.org/math/8/7/e/87efdf0d38947240683250d3a24466e0.png> (wiki)

---

[PyStats](#) - [FlexTileAnalyzer](#)

Counts and blocks the orientation of the flex tiles.

# getOrientationCountsBy90Increment

A part of the [FlexBlock](#) used for visualization of the features of a flextile.

## Details

- takes a set of orientations as input
  - these values must be numbers. Otherwise an error will be thrown and the value will be converted to 0.
- returns a set of four [orientation](#) ratios.
  1. The 0 degrees orientation
  2. The 90 degrees orientation
  3. The 180 degrees orientation
  4. The 270 degrees orientation
- the ratio is also a probability of occurrence of the given orientation symbol.
- values of +/- 45 degrees from the segment midpoint are grouped together.

---

[FlexBlock](#) - [FlexTileAnalyzer](#)

Rotates and compares two blocks.

# getRatioOfEquivalenceByRotation

A function of the [FlexBlock](#) that rotates and compares sub-parts of a matrix in order to detect [Rotational Symmetry](#)

## Details

- Input:
  - two square matrices x and y of equal size
  - expected *rotation* offset between the two matrices
  - the value of the rotation is the offset of y compared x. X will be rotated *rotation* degrees to the *right*.
- Returns a ratio of equivalence between 0 and 1
  - 0 denotes maximal error and no equivalence
  - 1 denotes maximal equivalence and therefore rotational symmetry for the given *rotation* offset.

Logic:

1. The positions of the values in the matrix X are rotated *rotation* degrees to the right.
2. The values are adapted to the novel position. [angleAware](#)
3. The two matrices are compared and an error sum is computed. The final ratio is computed as 1 - (sum of differences between matrices) / (size of matrix)

---

[FlexBlock](#) - [FlexTileAnalyzer](#)

Axis aware test of symmetry

# getDifferenceInPalindromeString

This function takes as input a list of data.

- The list represents a row, column or diagonal of the matrix.
- Palindrome detection is not limited to an even number of places.
- Data type is not limited to integer. Float, decimal and string is acceptable
- preprocess the data, delete whitespace. The 2 parts of the string must be EQUAL in order to get a zero difference.

You can specify an axis of symmetry you want to detect.

- [Axis](#) are according to the convention.

- [NaN; Not A Number](#) values will be ignored

Returns the difference of the two parts of the array. Zero denotes a perfectly symmetrical string. The max error is equal to the half of the length of the array.

In case of an axis specification, the inverted second part is adapted. This only works if all values are numbers. NaN values will remain the same. For more information read [Invert By Axis Procedure](#)

---

[SymAxis](#) - [FlexBlock](#) - [FlexTileAnalyzer](#)

Adapts the values in an array based on the axis of symmetry

## invert by axis procedure

Inverts / rotates the values based on the symmetry [axis](#) specified.

This function assumes a diagonal orientation of the tiles.

Example: a flex tile oriented towards the upper left corner is identified by a orientation ID of 0. Its horizontal inversion is 270, therefore the lower left corner. Should the tile be oriented vertically the horizontal inversion remains the same.

There is currently no functionality for vertically or horizontally oriented tiles.

% refers to the [Modulo Operation](#)

### Vertical (0)

$\text{angle} = (90 - \text{other}) \% 360$

- 0 becomes 90
- 90 becomes 0
- 180 becomes 270
- 270 becomes 180

### Horizontal(2)

$\text{angle} = (270 - \text{other}) \% 360$

- 0 becomes 270
- 90 becomes 180
- 180 becomes 90
- 270 becomes 0

### 1st Diagonal(1)

$\text{angle} = (180 - \text{other}) \% 360$

- 0 becomes 180
- 90 remains 90
- 180 becomes 0
- 270 remains 270

### 2nd Diagonal (3)

$\text{angle} = (360 - \text{other}) \% 360$

- 0 remains 0
- 90 becomes 270
- 180 remains 180
- 270 becomes 90

---

[SymAxis](#) - [getDifferenceInPalindromeString](#) - [FlexBlock](#) - [FlexTileAnalyzer](#)