Assignment 01

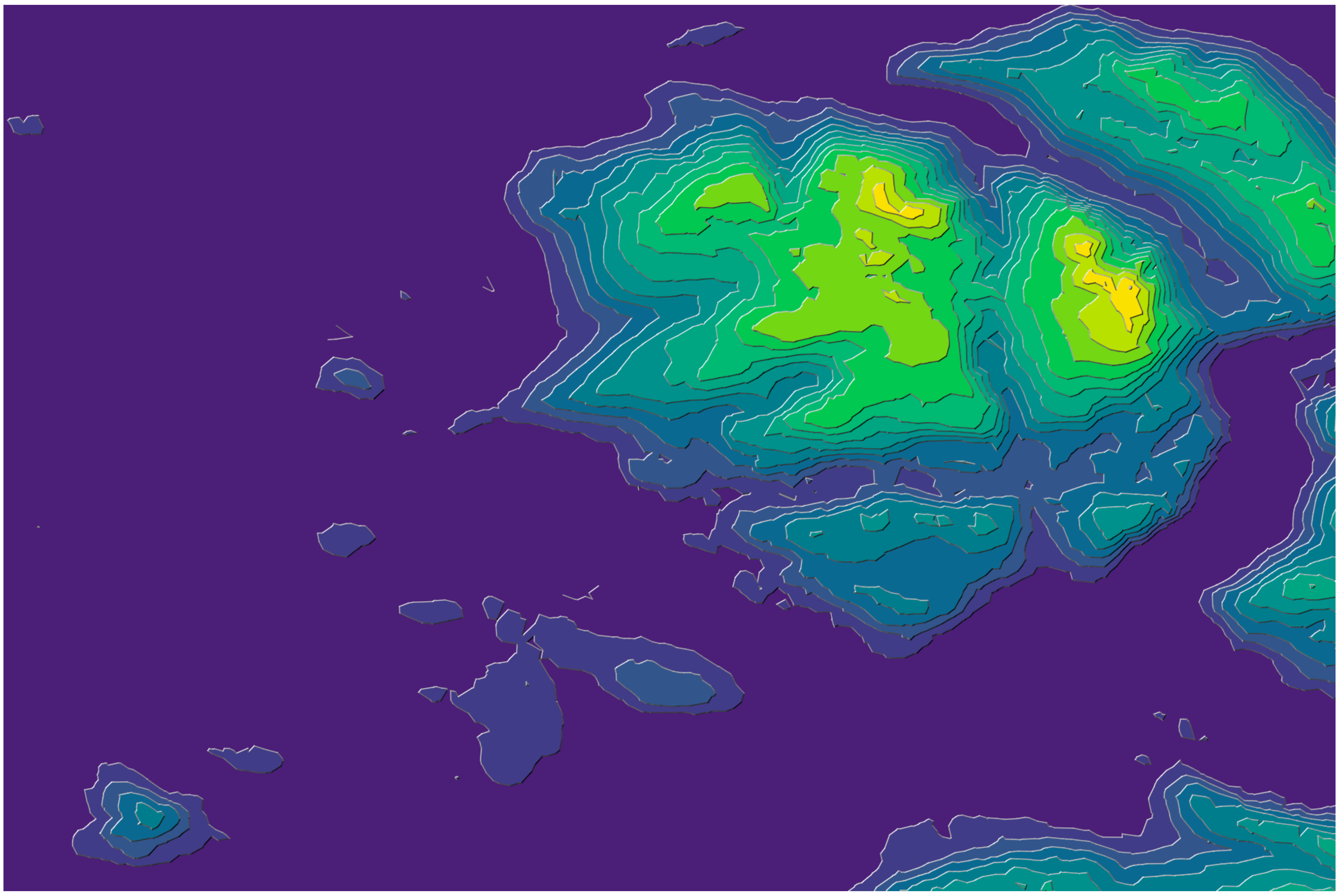
# Tanaka contours

Deadline is **2024-11-28@18:00**

Late submission? 10% will be removed for each day that you are late (3 days max)

It's worth 10% of the final mark

This is an individual assignment



The aim of the assignment is to extract the isolines from a triangulated TIN, and structure them so that we can easily create Tanaka contours with QGIS.

The Tanaka contours method, named after the cartographer Kitiro Tanaka who formalised it in 1950, is a visualisation method to shade isolines to give a better impression of the relief of a terrain (more information on Wikipedia). The lines are illuminated/white when facing the source of light, and shaded/black when not. As is usually the case with the visualisation of terrains, we use the North-West direction as the source of light (why that is).

You have to develop a Python program that:

1. reads a gridded DTM (in GeoTIFF format);

2. randomly samples some of its cells;
3. creates the Delaunay triangulation (DT) of the samples;
4. outputs a GeoJSON of the contours and the DT (in PLY format).

## The input/output of your Python program

Help/starting code is in the `/hw/01/` folder of the TUDelft GitLab repository of the course (you need to login with your NetID).

I give you an example of a Python program that accepts 3 input parameters, outputs a bogus DT, and gives you some examples and tips for the assignment. You can reuse/steal any part of that help code.

To run the code you first need to install the following libraries:

```
pip install -U startinpy
pip install -U rasterio
pip install -U numpy
```

and then:

```
python geo1015_hw01.py myinput.tiff 0.01 '(100, 500, 50)'
```

The arguments are as follows:

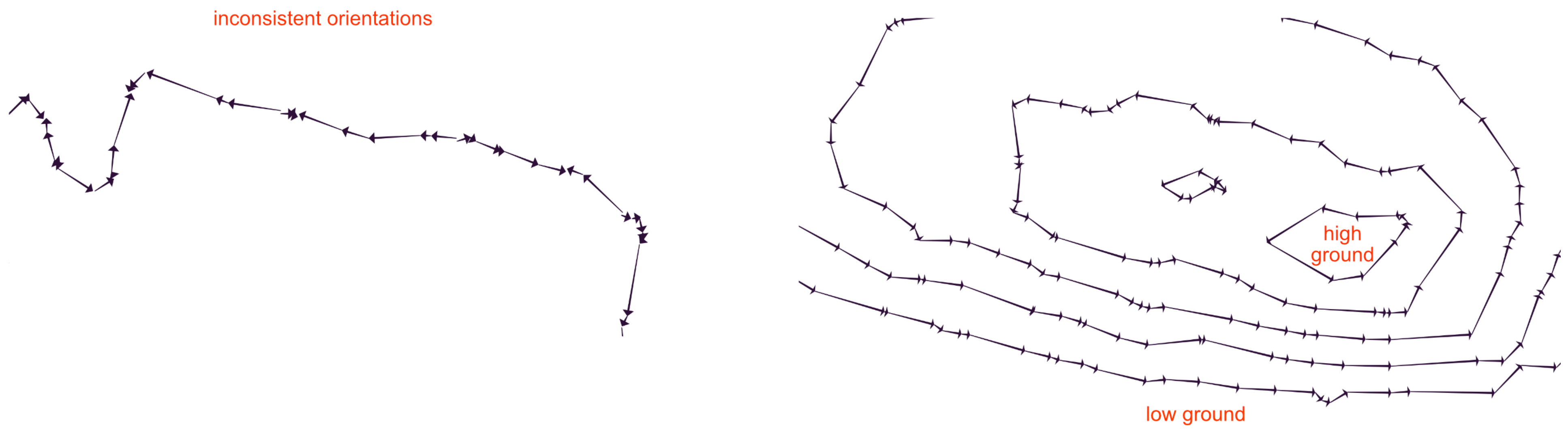
1. the input GeoTIFF file
2. the percentage of input cells (including the `no-data` values) that should be inserted, it's a number [0, 1]: 0.1 would means that a 300x200 input file would have a DT with 6000 vertices
3. a Python range that defines what contours need to be extracted. It is given as a string: `(100, 500, 100)` means `range(100, 500, 100)` which means 4 contour heights would be extract: 100, 200, 300, and 400.

The program must then write **in the same folder**:

1. the contours structured as explained below: `mycontours.geojson`
2. the DT in PLY format: `mydt.ply` using startinpy builtin function `write_ply()`. A PLY file can be read natively in QGIS.

## Structuring the isolines counter-clockwise

To be able to create Tanaka contours, we need to orient the isolines consistently. In our case, this means that closed isolines are oriented counter-clockwise, and that higher ground is on the left of a line.



Notice that the algorithm in the book to extract isolines from a DT will not give you any consistent orientation, as shown in the figure above. You have to either modify it, or post-process its output.

## Output format: GeoJSON

Your contours need to be output in the format GeoJSON. Each straight-line segment will be a feature and its geometry is a `LineString`, and it has 3 attributes (called `"properties"` in GeoJSON), as you can see here:

```
{
  "type": "FeatureCollection",
  "features": [
    {
      "type": "Feature",
      "geometry": {
        "type": "LineString",
        "coordinates": [ [0.1, 0.2], [10.0, 10.0] ]
      },
      "properties": {
        "height": 100,
        "azimuth": 45,
        "lightness": 0.75
      }
    },
    {
      "type": "Feature",
      "geometry": {
        "type": "LineString",
        "coordinates": [ [10.0, 10.0], [15.2, 21.8] ]
      },
      "properties": {
        "height": 100,
        "azimuth": 67.2,
        "lightness": 0.95
      }
    }
  ]
}
```

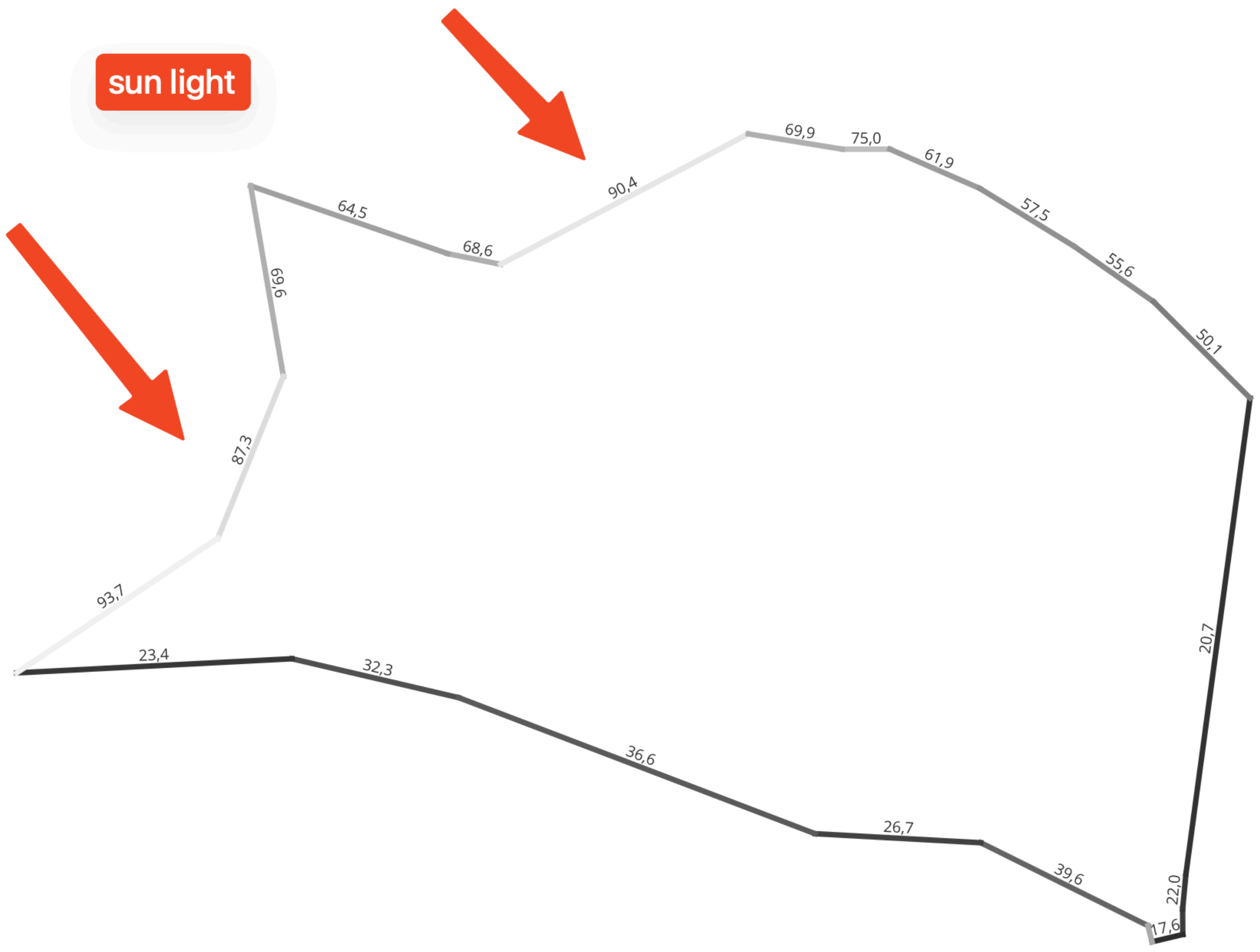
There exist Python libraries to read and write GeoJSON, but I don't recommend using them. Python can natively read and write JSON (`import json`); I give an example in the starting code.

The 3 attributes you have to store in your output GeoJSON file are:

**1. height**: the height of the isoline

**2. azimuth**: the azimuth of the line segment. It is the angle (in degrees) between the North and the line segment, measured clockwise in the horizontal plane (North=0; East=90; South=180; West=270).

**3. lightness**: the value (in the range [0,100]) for the colours of the Tanaka line segment, when a HSL colour model is used (HSL stands for hue, saturation, and lightness). The idea is that a line directly facing the sun light will be white ("HSL(0, 0, 100)") and one in the opposite direction will be black ("HSL(0, 0, 0)"). We linearly interpolate in between those values. See the values for the lightness below for the `contour_example.geojson` given in the demo code.



It's rather complex to obtain the lightness, see the explanation of Anita Graser (watch out she orients her line segments clockwise, so I had modified the formula). Based on the azimut (in degrees), just use this:

```
#-- get a value [0, 180]
l = abs((azimuth - 225) % 360) - 180)
#-- get the value in the range [0, 100] as QGIS expects
lightness = l / 180 * 100
```

> Observe that technically GeoJSON can only have geometries stored in WGS84, but in practice we can ignore this. The CRS can be ignored for this assignment, just store the coordinates in the same CRS as that of the input. The file can be opened in QGIS, just make sure there is no CRS for the view.

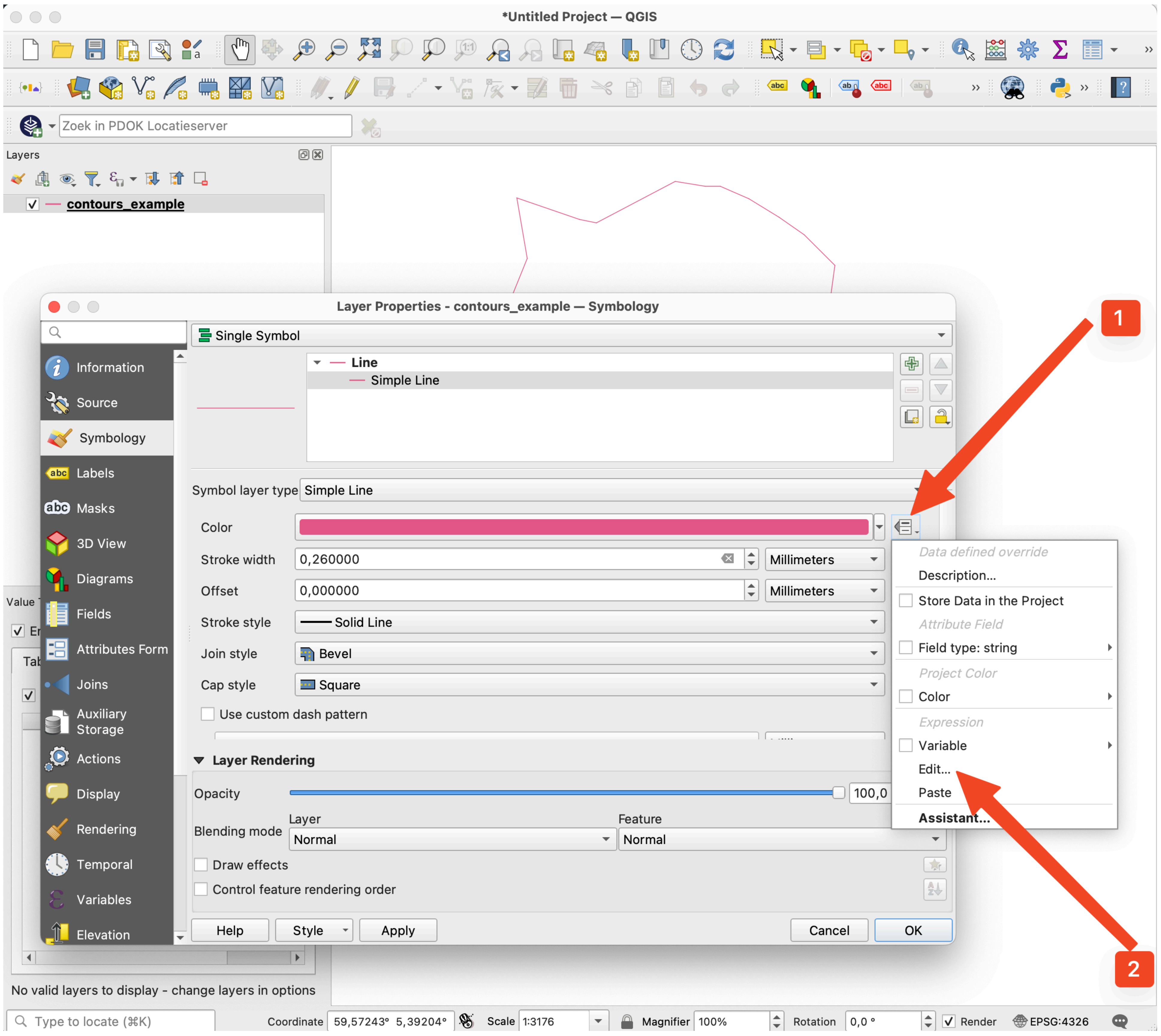
## How to shade the isolines

The idea for this homework comes from the following two blog posts explaining how to create Tanaka contours with QGIS:

1. https://anitagraser.com/2015/05/24/how-to-create-illuminated-contours-tanaka-style/
2. https://landscapearchaeology.org/2018/tanaka-contour-lines/

Notice that it's quite complex and requires many steps; I ask you to automate this as much as possible so that only the last step related to the visualisation is carried out in QGIS.

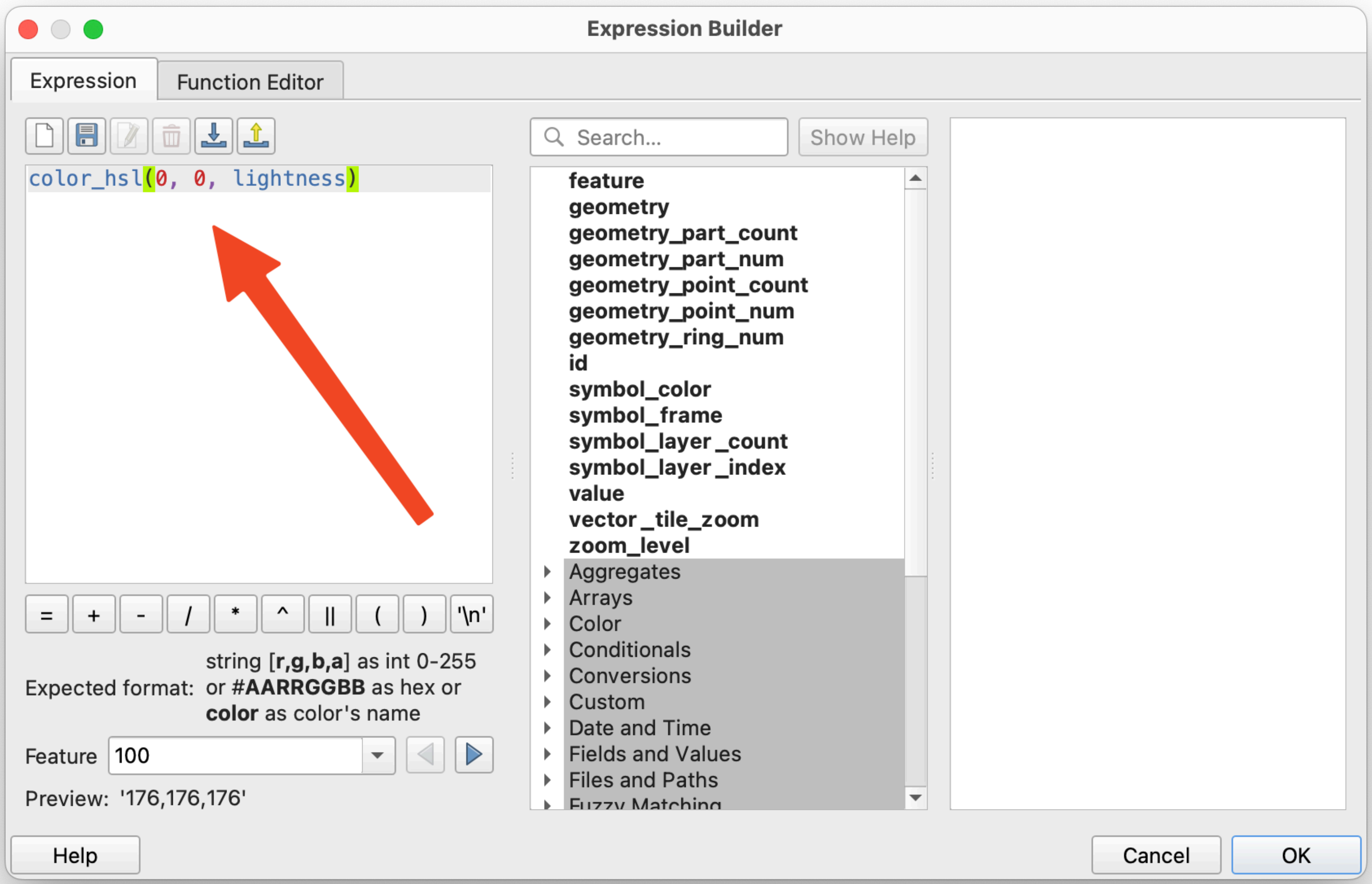
Based on the "lightness" attribute that you generate and store, simply change the styling of the lines in QGIS like below to obtain Tanaka contours (the expression is `color_hsl(0, 0, lightness)`, which will use the value of the attribute in your dataset).



Single Symbol
Symbol layer type: Simple Line
Color
Stroke width: 0,260000 Millimeters
Offset: 0,000000 Millimeters
Stroke style: Solid Line
Join style: Bevel
Cap style: Square
Use custom dash pattern
Layer Rendering
Opacity: 100,0
Blending mode: Layer Normal, Feature Normal
Draw effects
Control feature rendering order

No valid layers to display - change layers in options

Type to locate (⌘K) Coordinate 59,57243° 5,39204° Scale 1:3176 Magnifier 100% Rotation 0,0 ° ✓ Render EPSG:4326



```
color_hsl(0, 0, lightness)
```

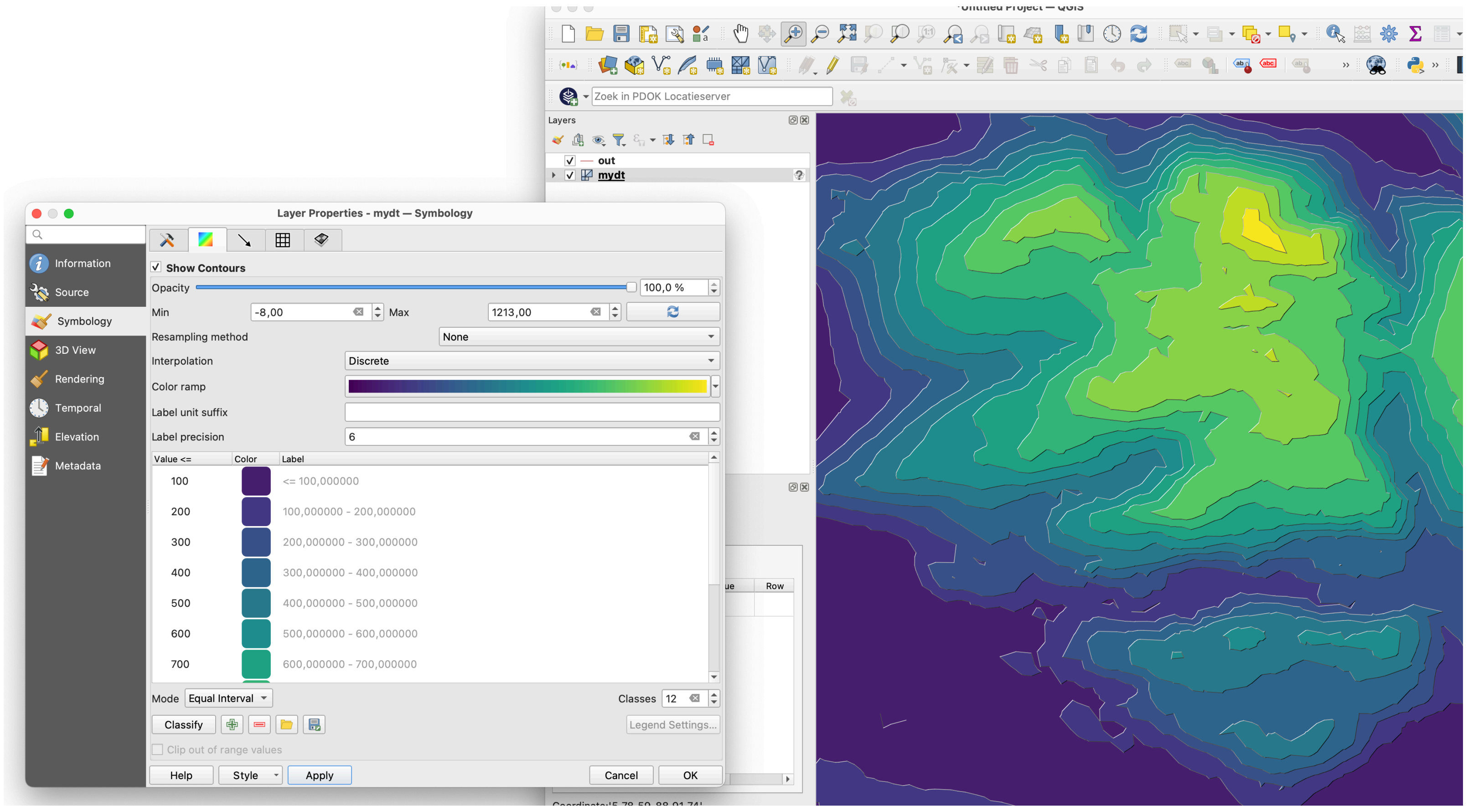
feature, geometry, geometry_part_count, geometry_part_num, geometry_point_count, geometry_point_num, geometry_ring_num, id, symbol_color, symbol_frame, symbol_layer_count, symbol_layer_index, value, vector_tile_zoom, zoom_level
Aggregates, Arrays, Color, Conditionals, Conversions, Custom, Date and Time, Fields and Values, Files and Paths, Fuzzy Matching

Expected format: string [r,g,b,a] as int 0-255 or #AARRGGBB as hex or color as color's name

Feature: 100

Preview: '176,176,176'

Finally, load your `mydt.ply` (that your code must export) and change the colours to 'Discrete' and put the same range as your isolines:



Show Contours: ✓
Opacity: 100,0 %
Min: -8,00  Max: 1213,00
Resampling method: None
Interpolation: Discrete
Color ramp
Label unit suffix:
Label precision: 6

| Value <= | Color | Label |
|---|---|---|
| 100 | 🟪 | <= 100,000000 |
| 200 | 🟪 | 100,000000 - 200,000000 |
| 300 | 🟦 | 200,000000 - 300,000000 |
| 400 | 🟦 | 300,000000 - 400,000000 |
| 500 | 🟦 | 400,000000 - 500,000000 |
| 600 | 🟩 | 500,000000 - 600,000000 |
| 700 | 🟩 | 600,000000 - 700,000000 |

Mode: Equal Interval  Classes: 12
Clip out of range values

And, voilà 🚀 ❤️

## Good to know

- the GeoTIFF input file can contain `no_data` values, be warned!
- for "real" Tanaka contours, the width of the contours should also be modified based on their orientation, but we can ignore this for this assignment
- you can reuse the given code, modify it as you wish, add functions, etc. Please put all the code in the same file (`geo1015_hw01.py`)
- no external libraries can be used, except startinpy, numpy, and rasterio. Everything from the Python standard library is fine (math, json, sys, etc.)
- the time it takes for your program to terminate has no influence on the marks you'll get

## Marking

| Criterion | Points |
|---|---|
| followed all rules and compiles/runs without modifications | 2.0 |
| isoline extracted (no consistent orientation) | 2.0 |
| complex cases handled (eg horizontal triangles) | 1.0 |
| orientation of isoline (ccw) | 2.0 |
| azimuth | 1.0 |
| lightness | 1.0 |
| valid output GeoJSON file | 1.0 |

## What to submit and how to submit it

You have to submit **one** file: `geo1015_hw01.py`.

Don’t forget to add your name and student-number at the top of the tile, as the comments tell you.

Oh, and **no** report is necessary.

Upload the file to this Dropbox page.

*[last updated: 2024-11-06 11:06]*