Self-Test

Interactive Data Visualization

SS 20

Sabrina Piasecki

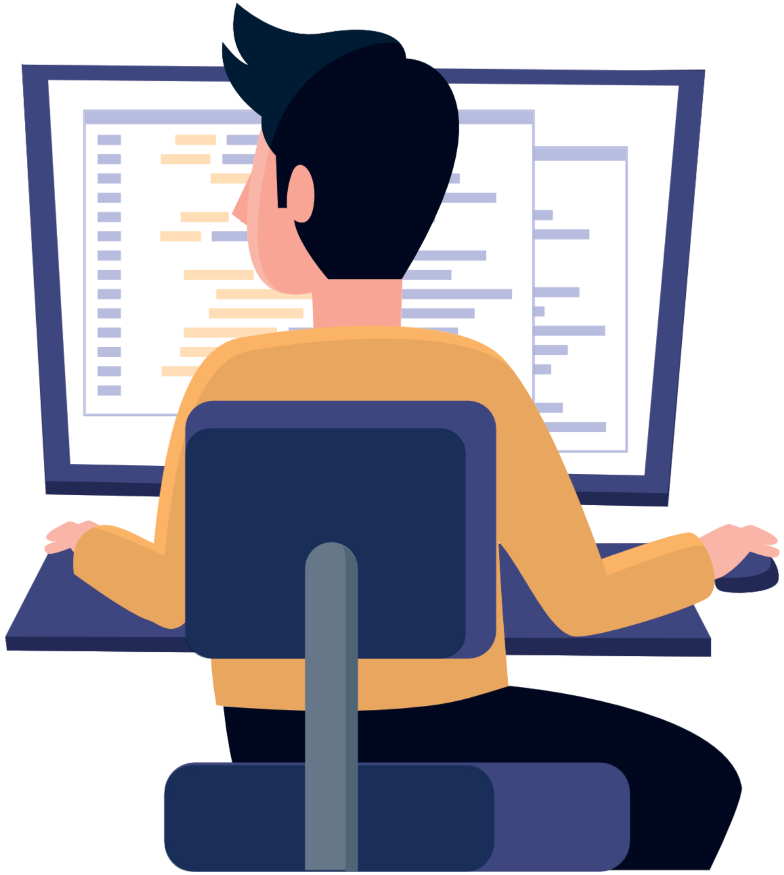
PADERBORN UNIVERSITY

Computergraphics, Visualization & Image Processing

Content

- Self-Test
  - Basic algorithm
  - Helpful code fragments online on PANDA
  - Code for the solution will not be published

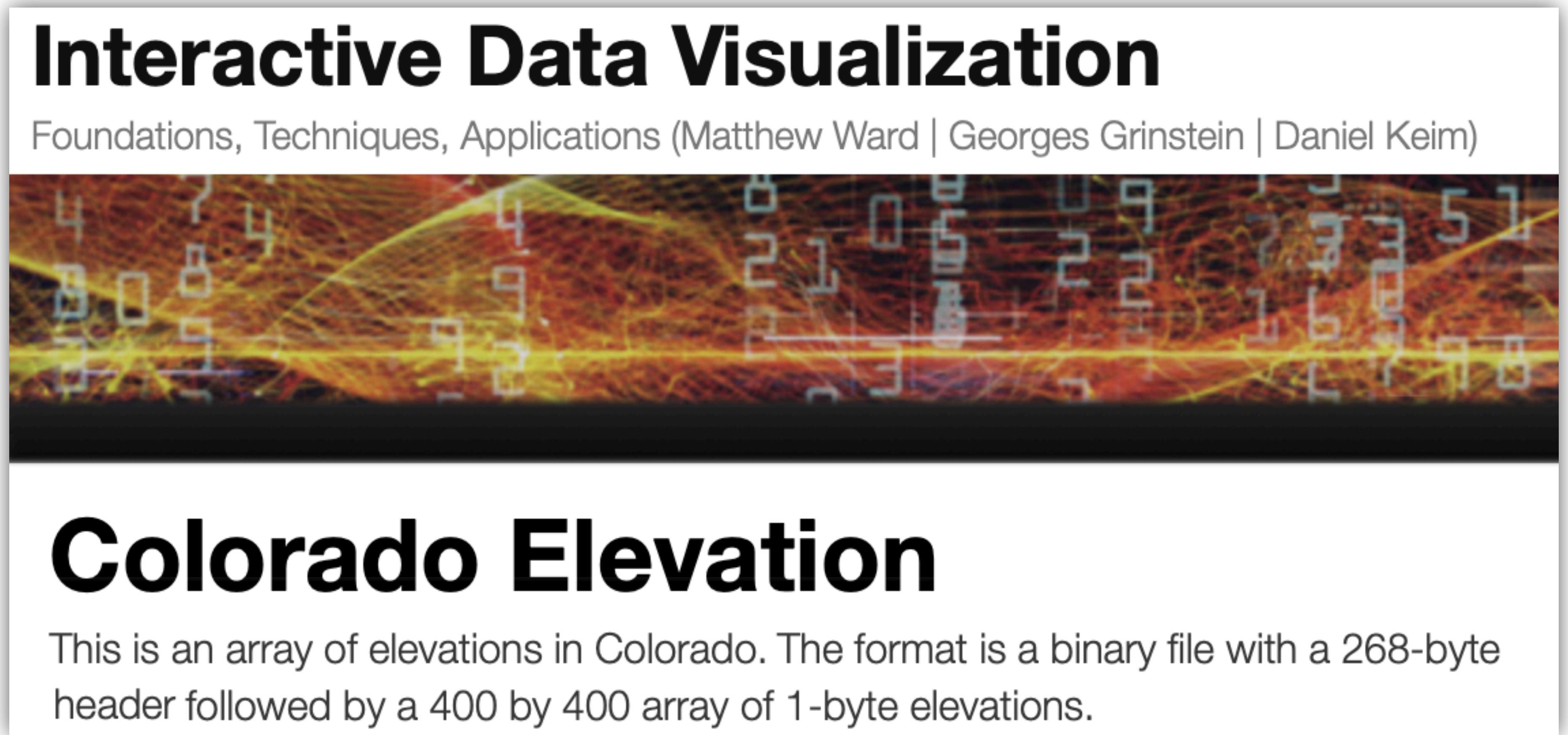
- Additional helpful python basics



Self-Test

Self-Test

Which information are important?



Interactive Data Visualization

Foundations, Techniques, Applications (Matthew Ward | Georges Grinstein | Daniel Keim)

Colorado Elevation

This is an array of elevations in Colorado. The format is a binary file with a 268-byte header followed by a 400 by 400 array of 1-byte elevations.

- 268-byte header
- 400x400 array of byte data

Self-Test

Basic Algorithm:

- Read data and
  save it in an array for further processing
- Drop header
- Check data characteristics
- Create a basic grey scale image
- Crate a color image
- Add a legend
- Adjust labels

Self-Test

Check-List

- → Read data and
  save it in an array for further processing
- Drop header
- Check data characteristics
- Create a basic grey scale image
- Crate a color image
- Add a legend
- Adjust labels

Self-Test

Read binary-files with **open**-function:

Algorithm:

```
• Create an 1D array
• Open file & read
    • Read data and store in array
```

Self-Test

Read binary-files with **open**-function:

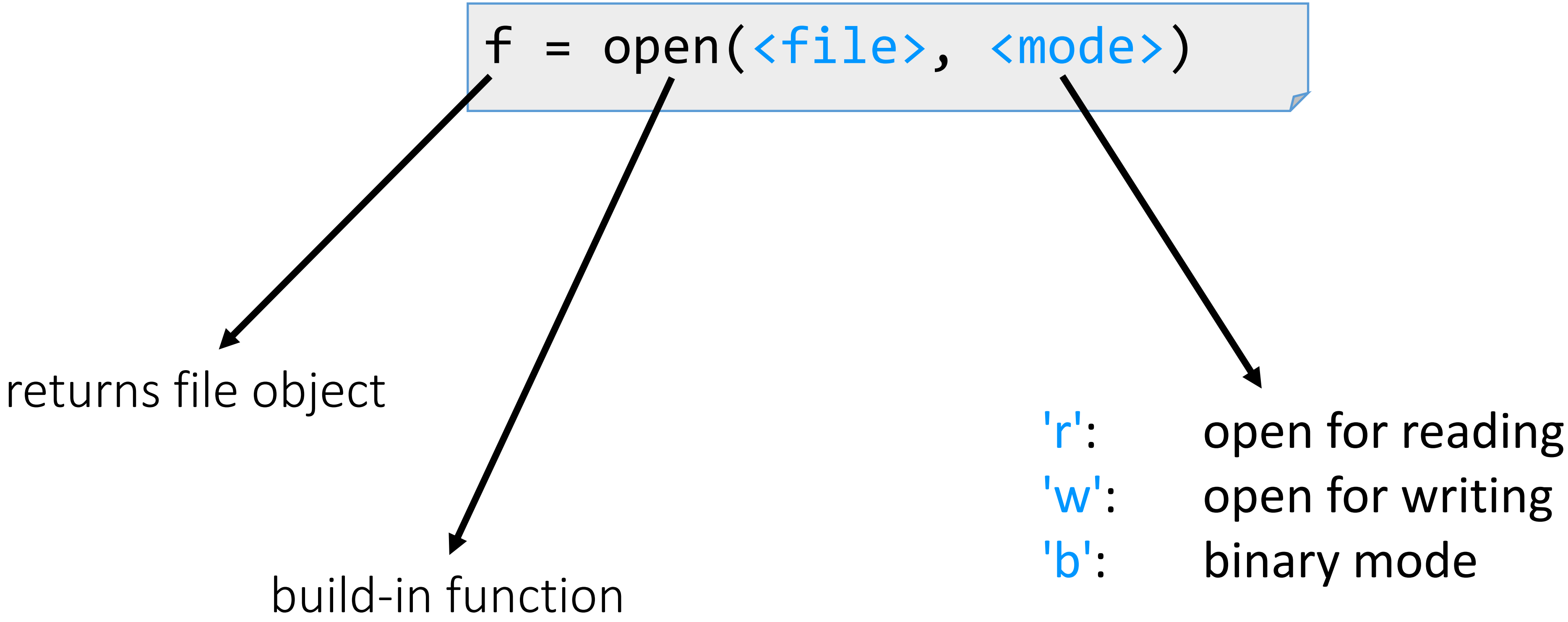
Algorithm:

- Create an 1D array
- Open file & read
  - Read data and store in array

```
dataArr = []
with open(<file>, "rb") as f:
    dataArr = f.read()
```

Python

Read binary-files with **open**-function:



```
f = open(<file>, <mode>)
```

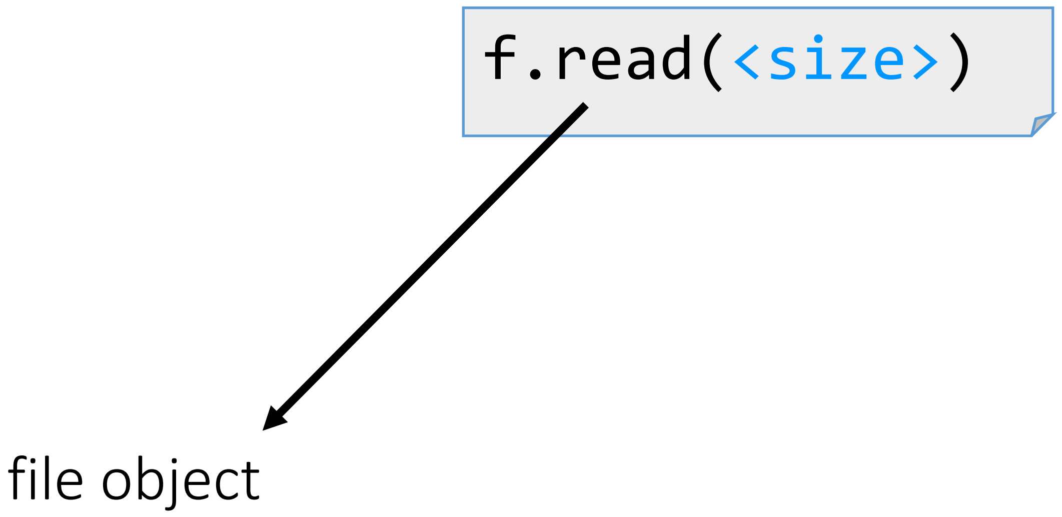
returns file object

build-in function

'r':	open for reading
'w':	open for writing
'b':	binary mode

Python

Read binary-files with **open**-function:

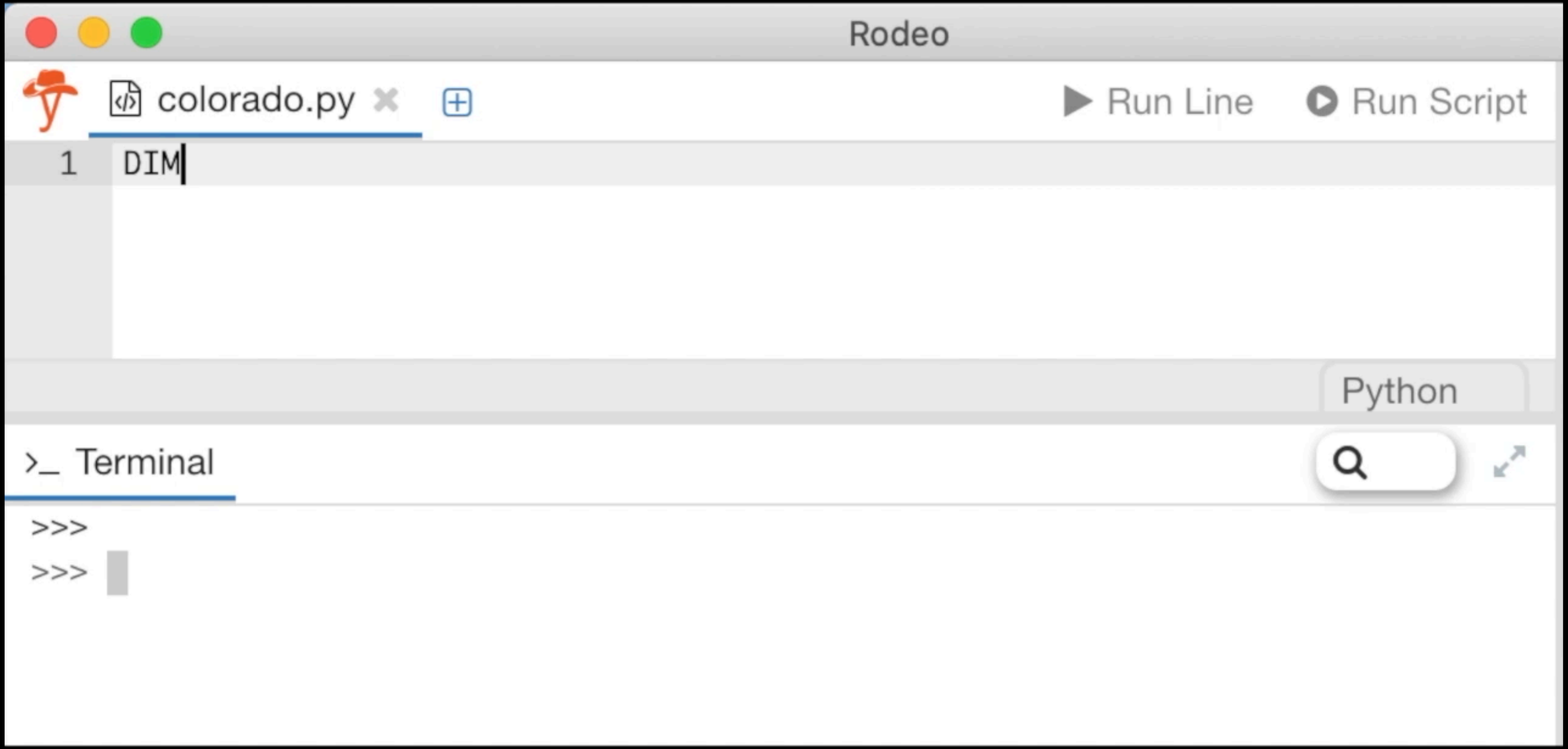


```
f.read(<size>)
```

file object

Self-Test

Read binary-files with **open**-function:



```
DIM
```

```
>>>
>>>
```

Self-Test

Read binary-files with **open**-function – one by one:

Algorithm:

```
• Create an 1D array
• Open file
• Read first byte
• While end of file not reached
    • Store data as integer
    • Do something
    • Add data to array
    • Read next byte
```

Self-Test

Read binary-files with **open**-function – one by one:

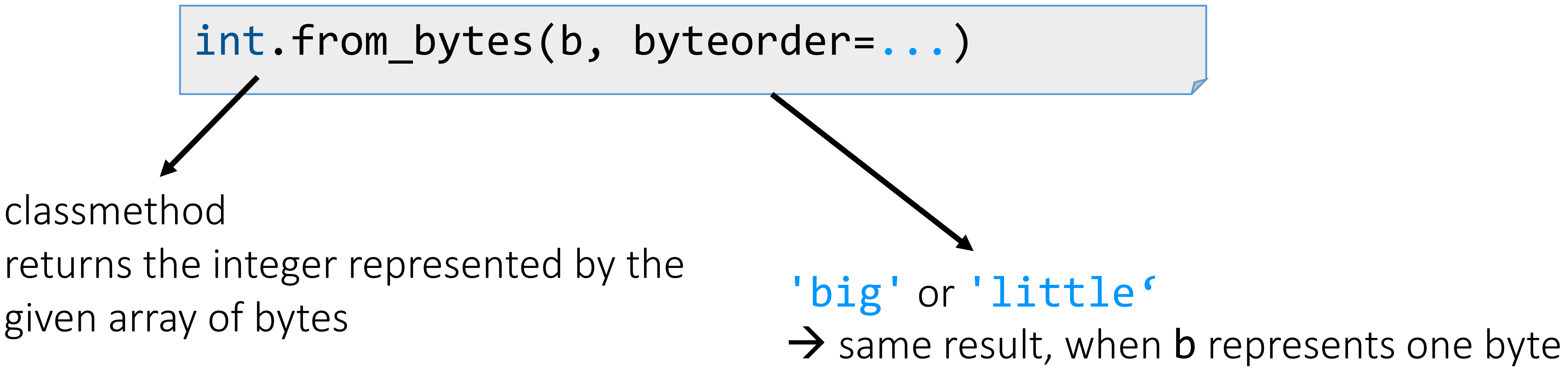
Algorithm:

```
• Create an 1D array
• Open file
• Read first byte
• While end of file not reached
    • Store data as integer
    • Do something
    • Add data to array
    • Read next byte
```

```python
dataArr = []
with open(<file>, "rb") as f:
    byte = f.read(1)
    while byte:
        dataInt = ...
        #do something
        dataArr.append(dataInt)
        byte = f.read(1)
```

Python

How to transform a byte value to an integer value?



```
int.from_bytes(b, byteorder=...)
```

classmethod
returns the integer represented by the given array of bytes

'big' or 'little‘
→ same result, when **b** represents one byte

byte_to_int.py

Self-Test

Check-List

- Read data and
  save it in an array for further processing
- Drop header
- Check data characteristics
- Create a basic grey scale image
- Crate a color image
- Add a legend
- Adjust labels

Phython

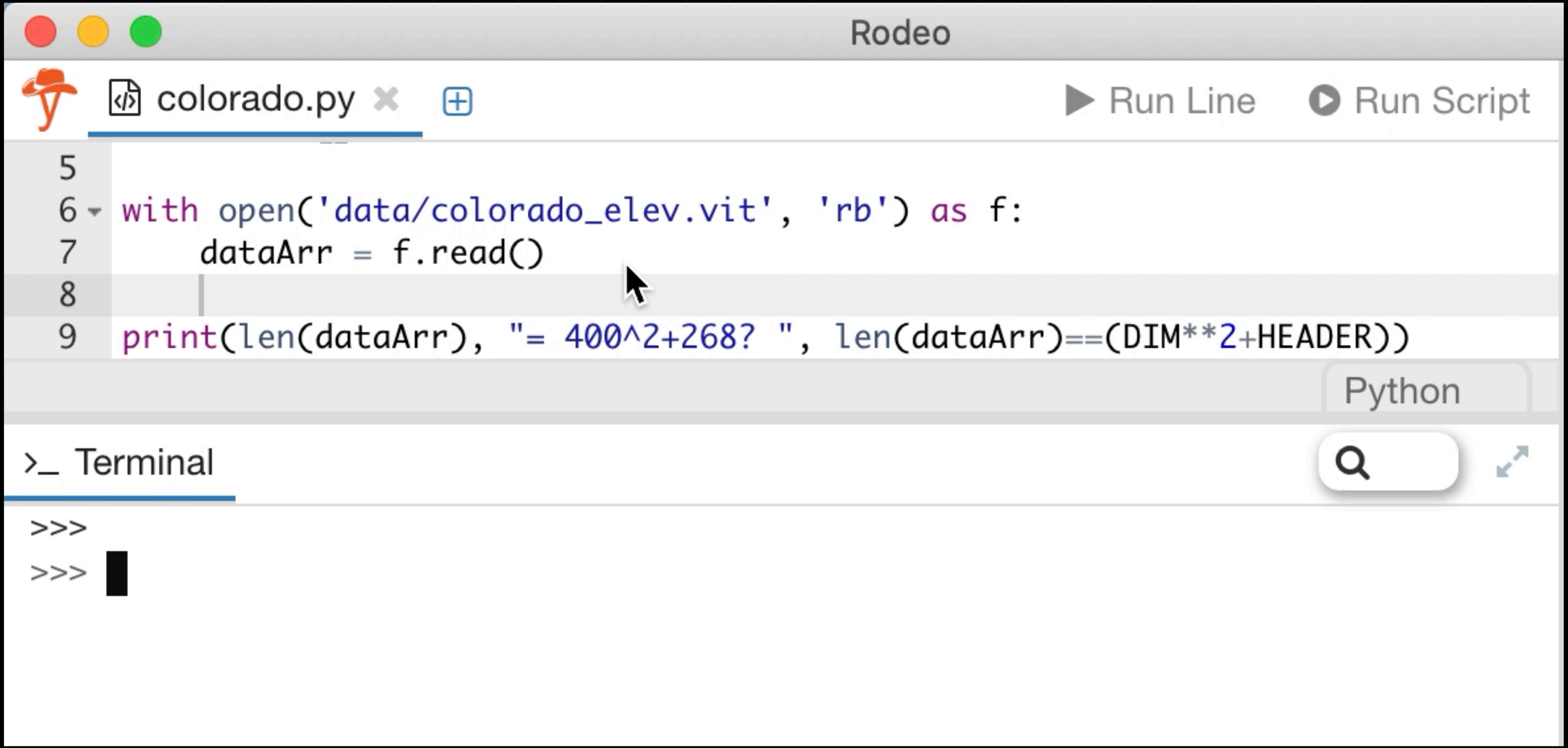
How do drop data from an array?

```
arr = ...
subarr1 = arr[start:end]
subarr2 = arr[start:]
subarr3 = arr[:end]
```

[tryit]drop_data.py

Self-Test

Dropping the header:



```python

with open('data/colorado_elev.vit', 'rb') as f:
    dataArr = f.read()

print(len(dataArr), "= 400^2+268? ", len(dataArr)==(DIM**2+HEADER))
```

```
>>>
>>>
```

Self-Test

Check-List

- Read data and
  save it in an array for further processing
- Drop header
- → Check data characteristics
- Create a basic grey scale image
- Crate a color image
- Add a legend
- Adjust labels

Self-Test

Calculation of minimum and maximum value:

Algorithm:

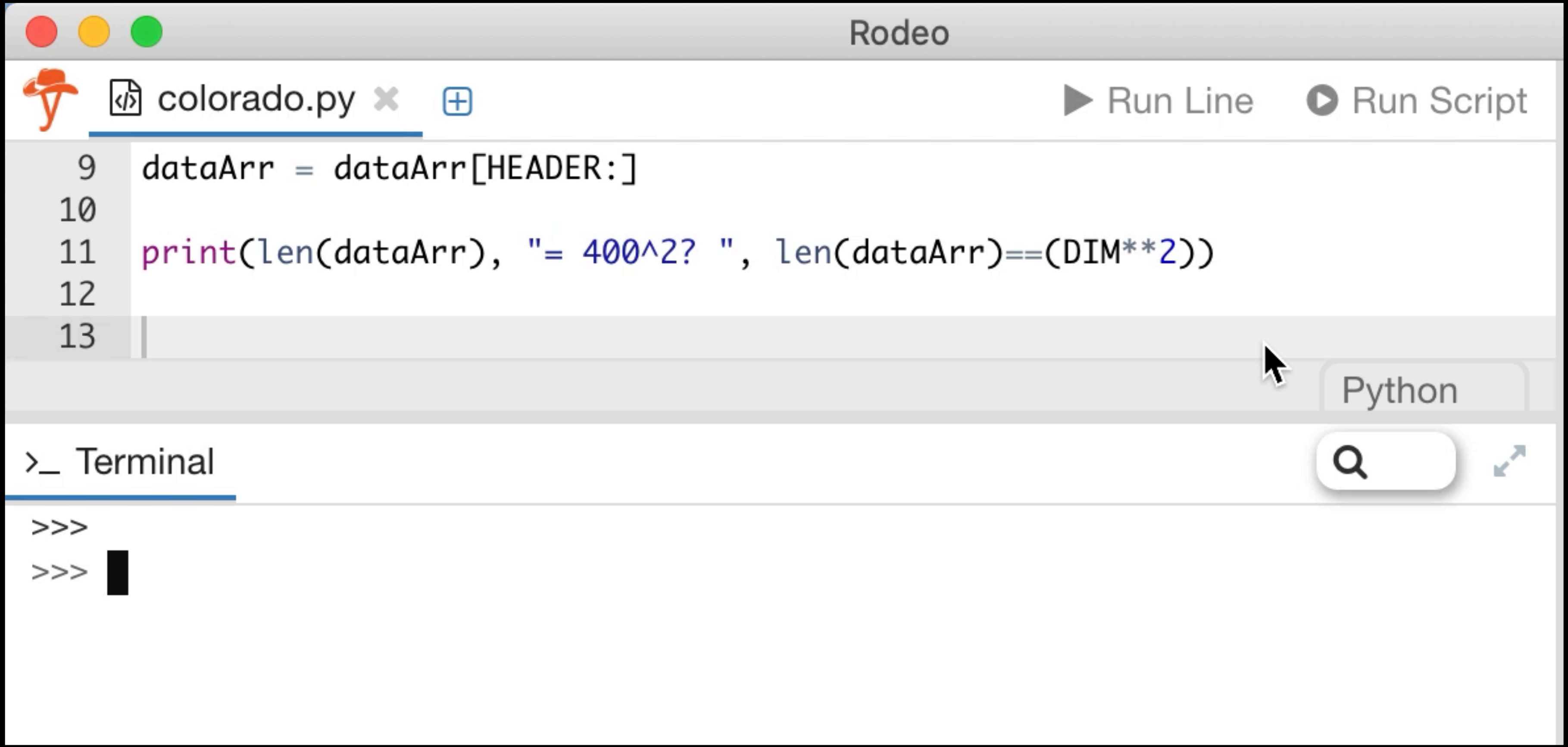
- Calculate minimum value
- Print minimum value

- Calculate maximum
- Print maximum value

```
minVal = min(dataArr)
print("minimum: ", minVal)

maxVal = max(dataArr)
print(”maximum: ”, maxVal)
```

Self-Test

Calculation of minimum and maximum value:



```python
dataArr = dataArr[HEADER:]

print(len(dataArr), "= 400^2? ", len(dataArr)==(DIM**2))

```

```
>>>
>>> 
```

Self-Test

Check-List

- Read data and
  save it in an array for further processing
- Drop header
- Check data characteristics
- → Create a basic grey scale image
- Crate a color image
- Add a legend
- Adjust labels

Self-Test

Create a gray scale image:

Algorithm:

```
• Create empty image

• For each data value and
  corresponding pixel
    • Read data
    • Fill pixel with data

• Show image
• Save image
```

Self-Test

Create a gray scale image:

Algorithm:

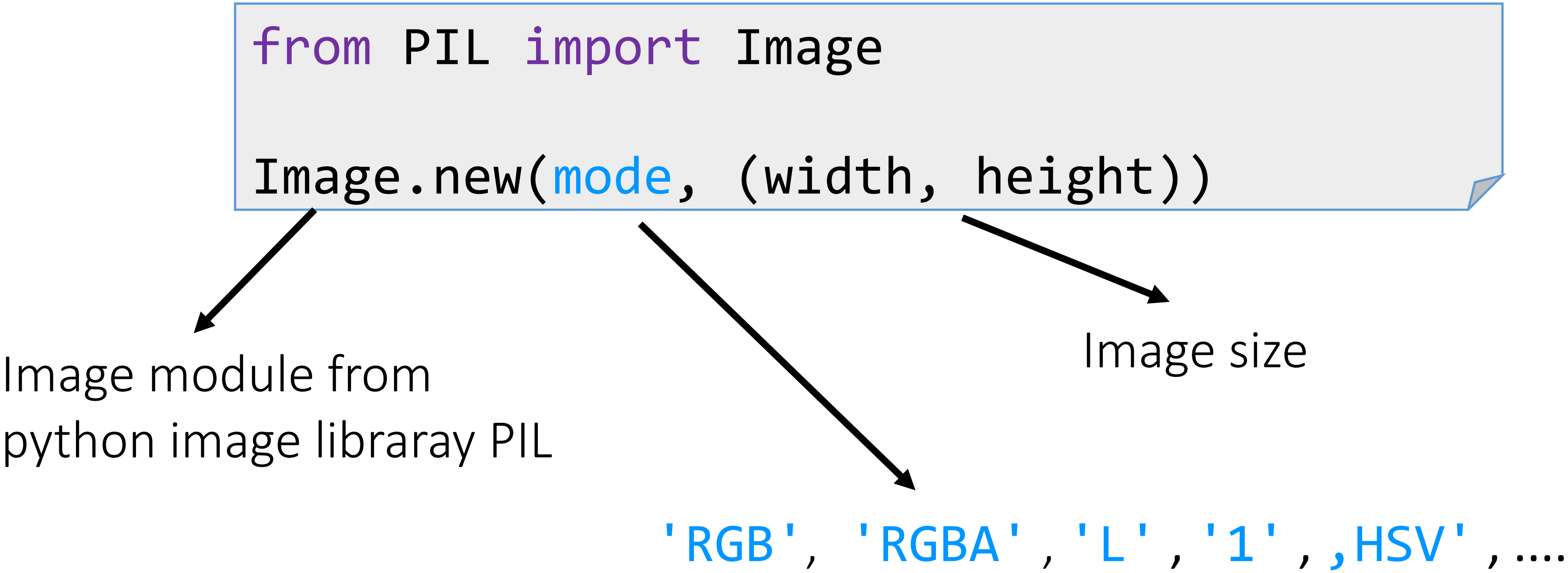
- Create empty image
- For each data value and corresponding pixel
  - Read data
  - Fill pixel with data
- Show image
- Save image

```
image = …
pixelMap = …

for y in range(DIM):
        for x in range(DIM):
                data = dataArr[y*DIM+x]
                pixelMap[x,y] = data

image.show()
image.save(<file>)
```

pil.py

Python

How to work with images?



```
from PIL import Image

Image.new(mode, (width, height))
```

Image module from
python image libraray PIL

Image size

'RGB', 'RGBA','L','1',,HSV',....

Python

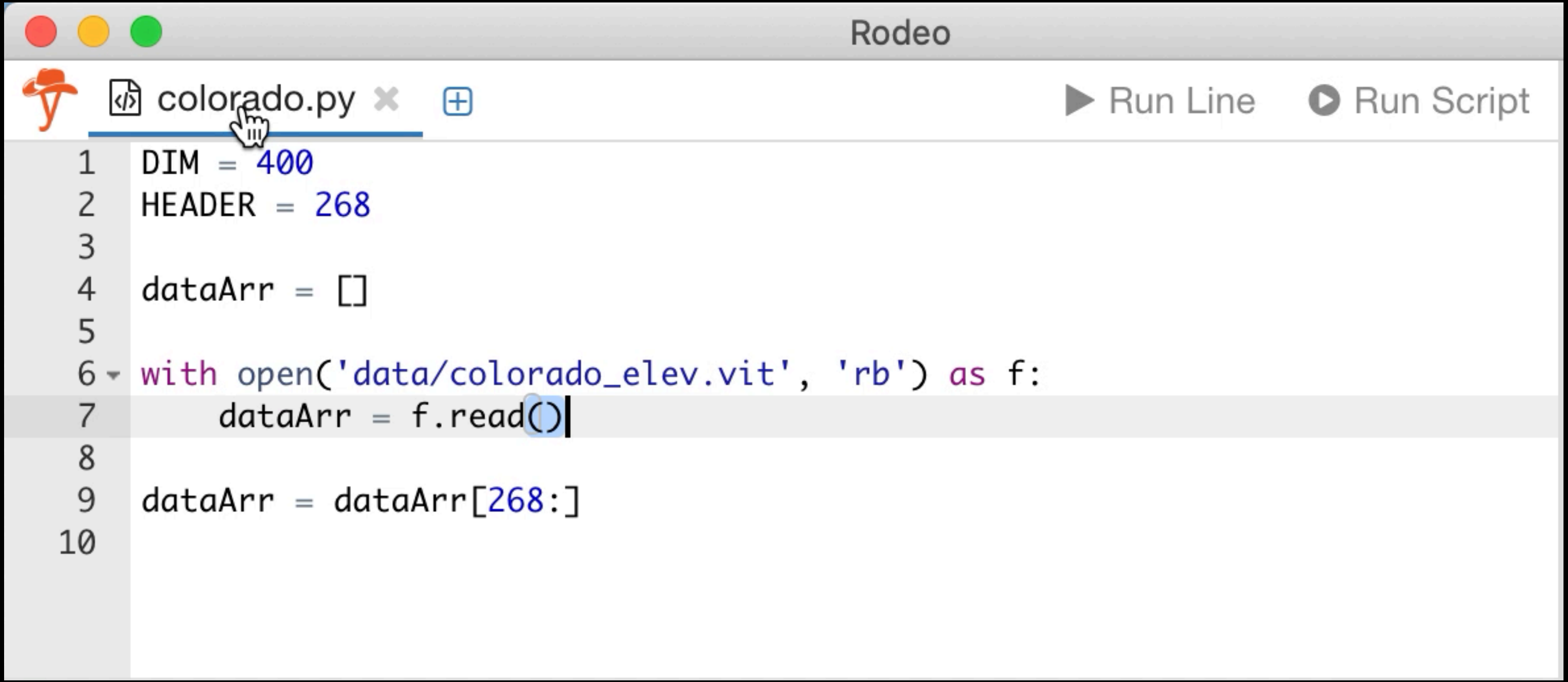
How to work with pixels?

```
from PIL import Image

image = Image.new(mode, (width, height))
pixelMap = image.load()
```

allocate storage for the image and
load the pixel data

Self-Test

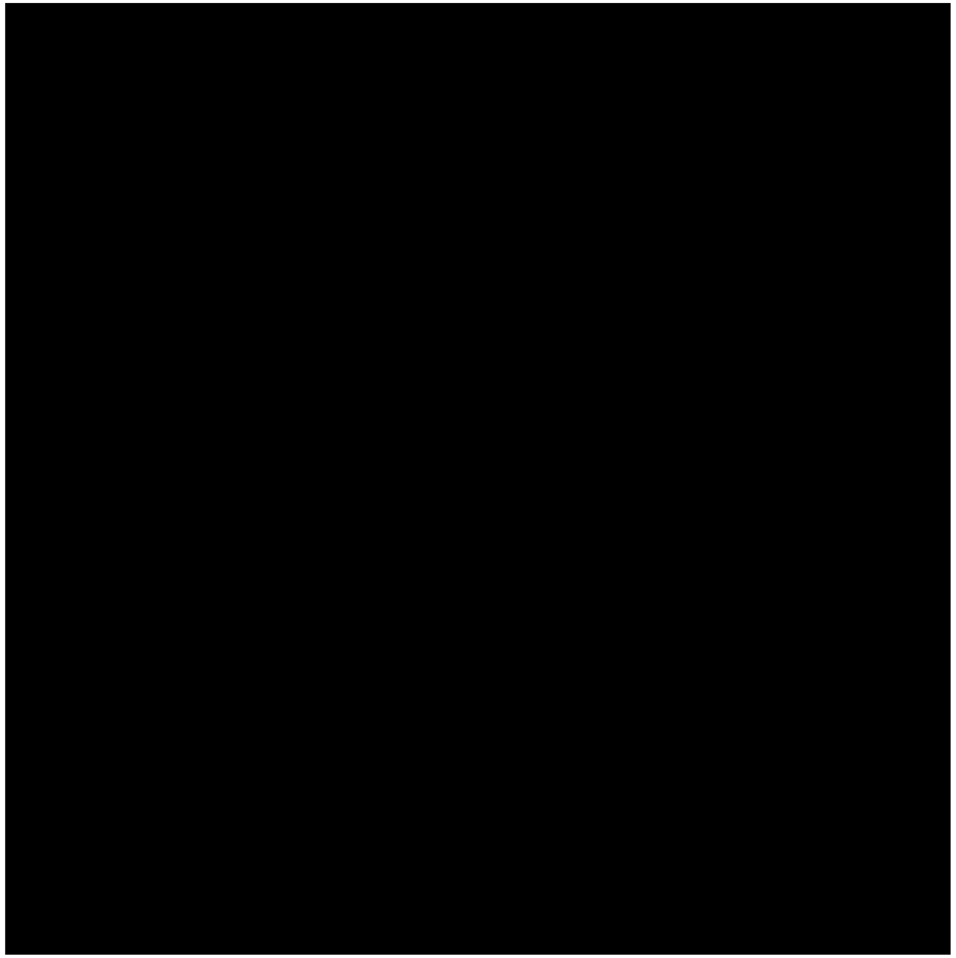
Create a gray scale image:



```python
DIM = 400
HEADER = 268

dataArr = []

with open('data/colorado_elev.vit', 'rb') as f:
    dataArr = f.read()

dataArr = dataArr[268:]

```

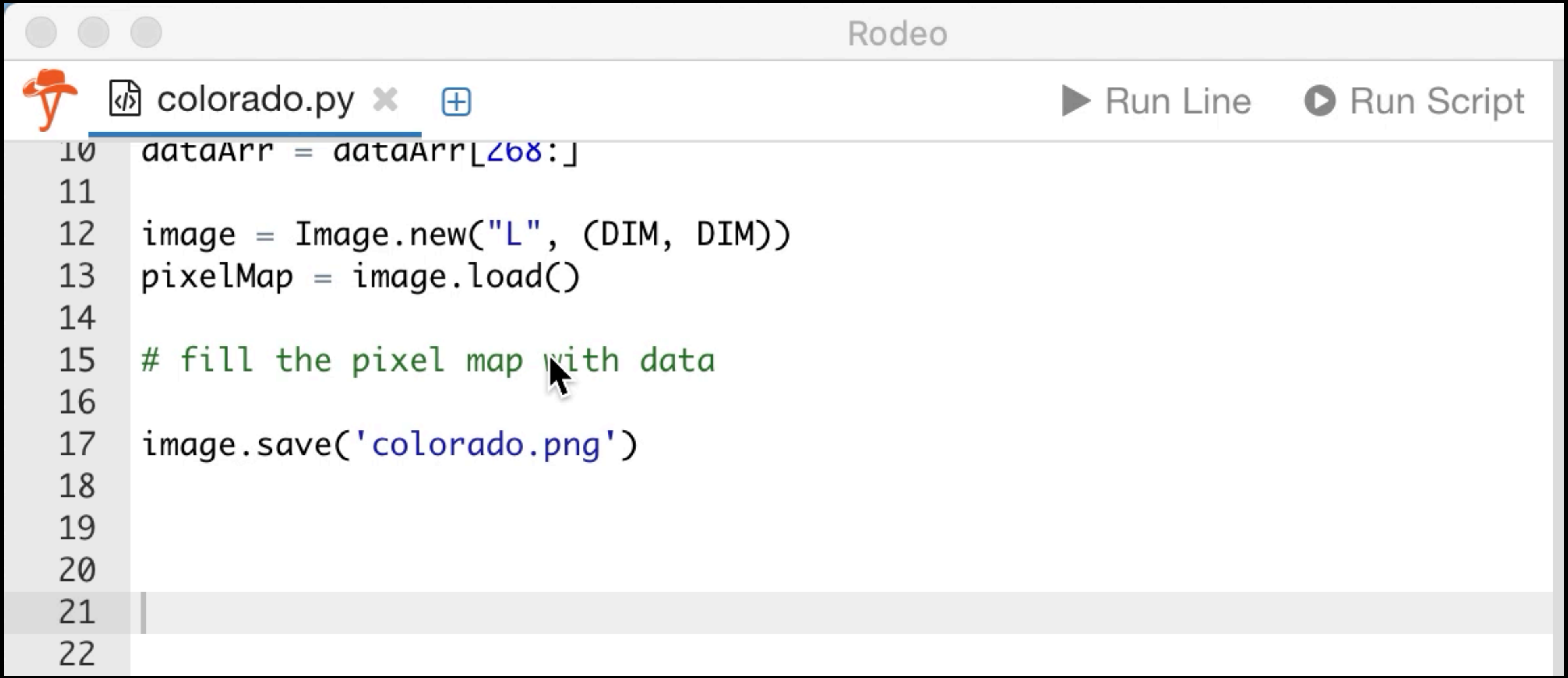
Self-Test

Grey scale image without content



Self-Test

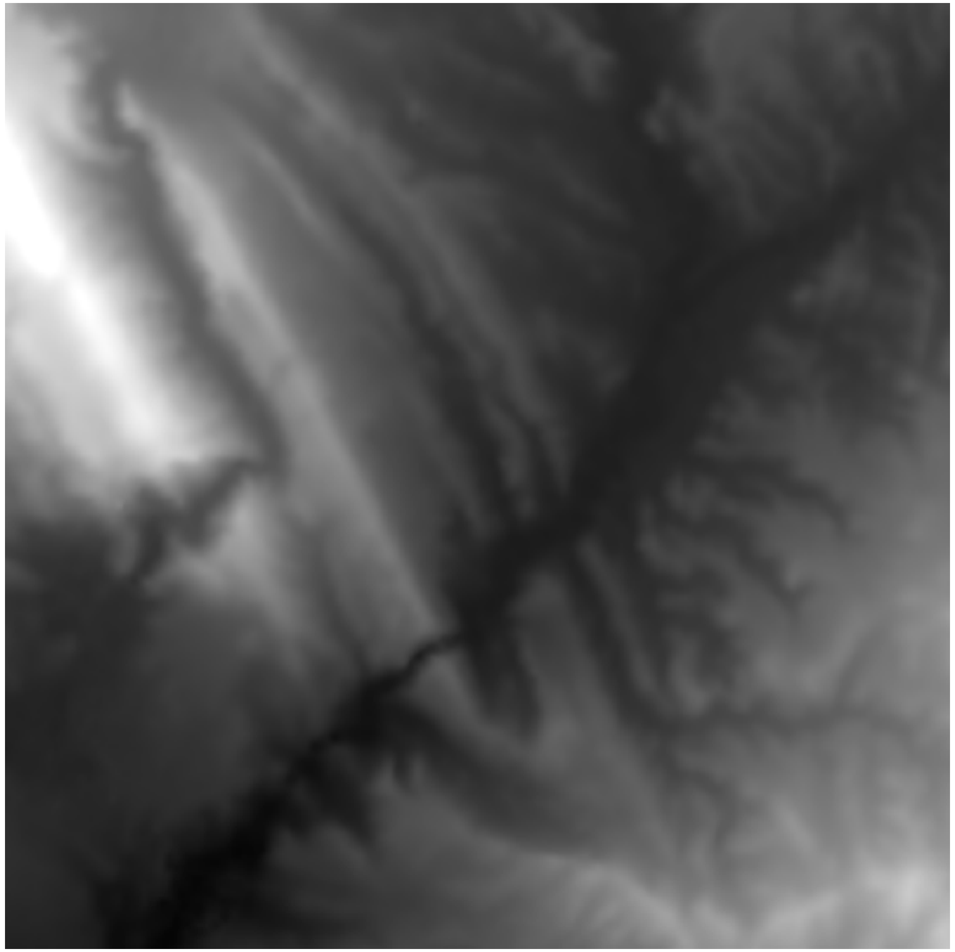
Fill the gray scale image:



```
dataArr = dataArr[268:]

image = Image.new("L", (DIM, DIM))
pixelMap = image.load()

# fill the pixel map with data

image.save('colorado.png')
```

Self-Test

Grey scale image



Self-Test

Check-List

- Read data
- Save data in array for further processing
- Drop header
- Check data characteristics
- Create a basic gray scale image
- → Create a color image
- Add a legend
- Adjust labels

Self-Test

Check-List

- Read data
- Save data in array for further processing
- Drop header
- Check data characteristics
- Create a basic gray scale image
- Create a color image
  - → By hand
  - Using colormaps
- Add a legend
- Adjust labels

Self-Test

- Create visualization:

Algorithm:

```
• Create empty image

• For each data value and
  corresponding pixel
    • Read data
    • Calculate color value

    • Fill pixel with data

• Show image
• Save image
```

Self-Test

- Create visualization:

Algorithm:

- Create empty image
- For each data value and corresponding pixel
  - Read data
  - Calculate color value
  - Fill pixel with data
- Show image
- Save image

```
image = Image.new("RGB", …
pixelMap = …

for y in range(DIM):
        for x in range(DIM):
                data = dataArr[y*DIM+x]
                hue = getHue(data, …)
                hue_rgb = hsv2rgb(hue, …)
                pixelMap[x, y] = hue_rgb
image.show()
image.save(<file>)
```

Self-Test

- Create visualization:

Algorithm:

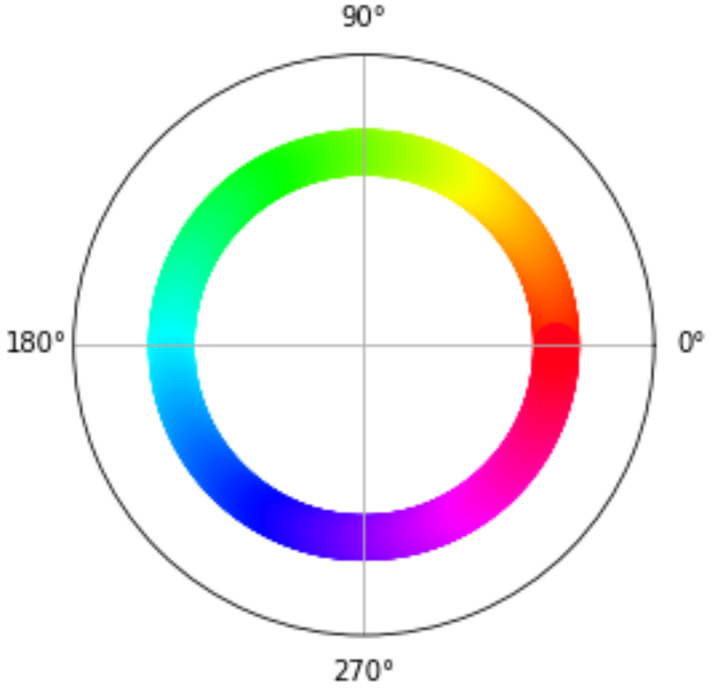
- Create empty image
- For each data value and corresponding pixel
  - Read data
  - Calculate color value
  - Fill pixel with data
- Show image
- Save image

```
image = Image.new("RGB", …
pixelMap = …

for y in range(DIM):
        for x in range(DIM):
                data = dataArr[y*DIM+x]
                hue = getHue(data, …)
                hue_rgb = hsv2rgb(hue, …)
                pixelMap[x, y] = hue_rgb
image.show()
image.save(<file>)
```

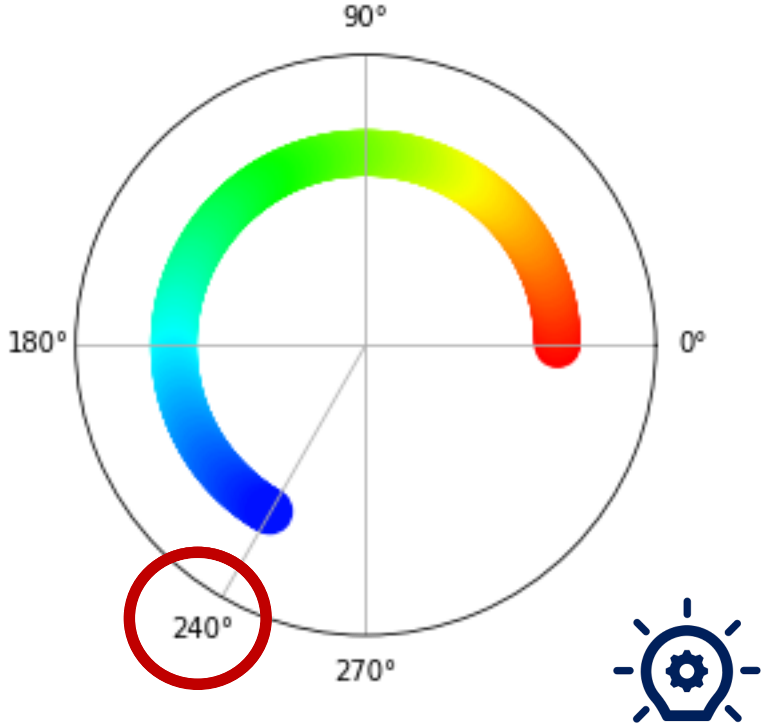
Self-Test

What colors do we want to use?

all hue values in HSV:



hue values in HSV we want to use:

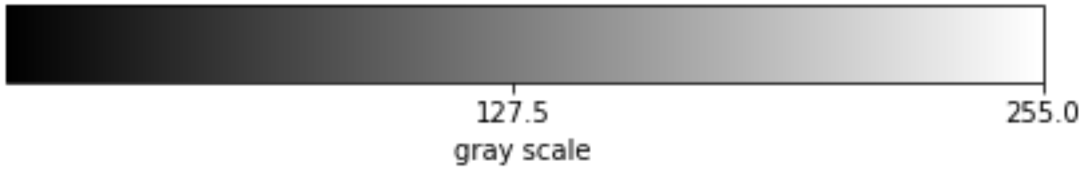


hue.py

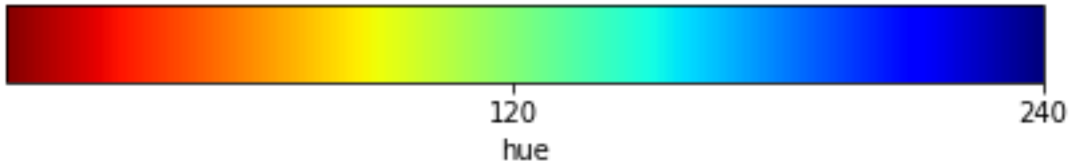
Self-Test

Calculation of hue value - how it works:

what we have:



what we want to have:



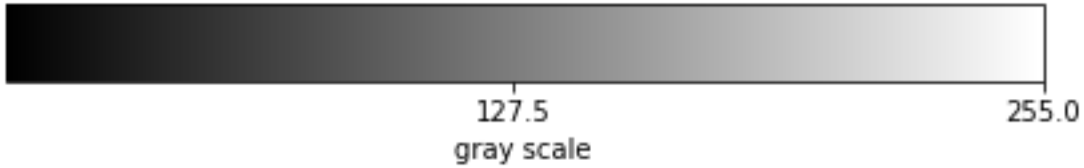
$$\frac{\text{gray scale value}}{255} * 240 = \text{hue value}$$

example: $\frac{127.5}{255} * 240 = 120$

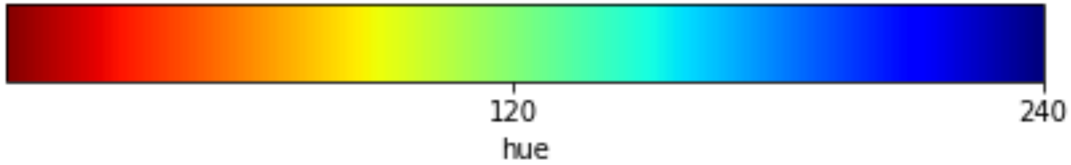
Self-Test

Calculation of hue value - how it works:

what we have:



what we want to have:

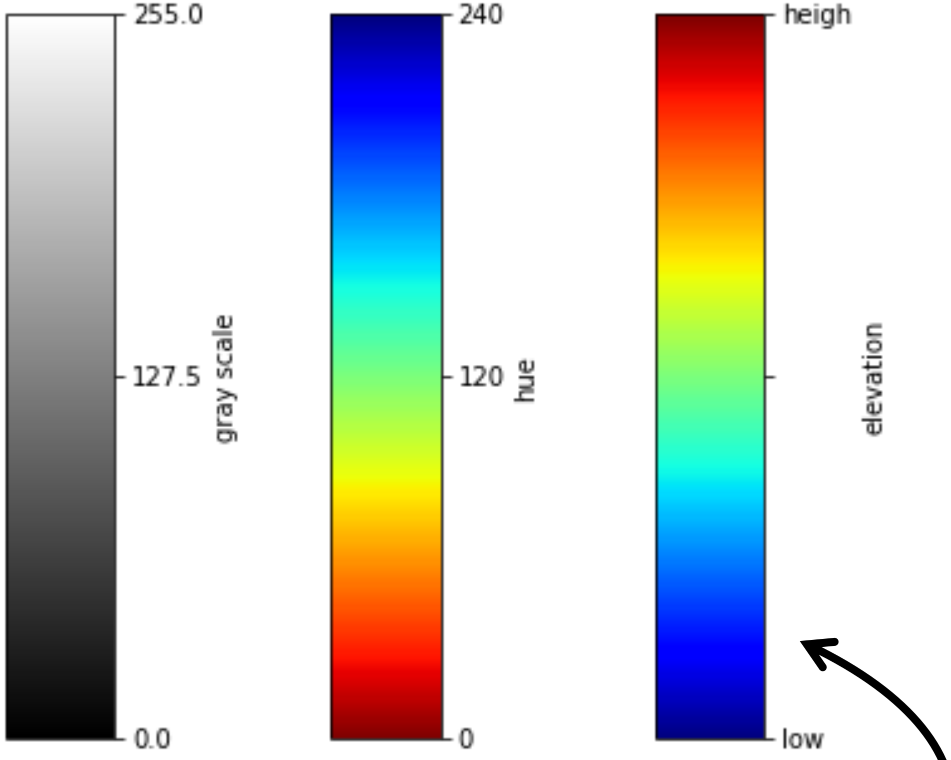


```
def getHue(v, max):
        value = int(240 * (v / max))
        return value
```

Self-Test

Calculation of hue value – how it works:

what we have



what we want to have

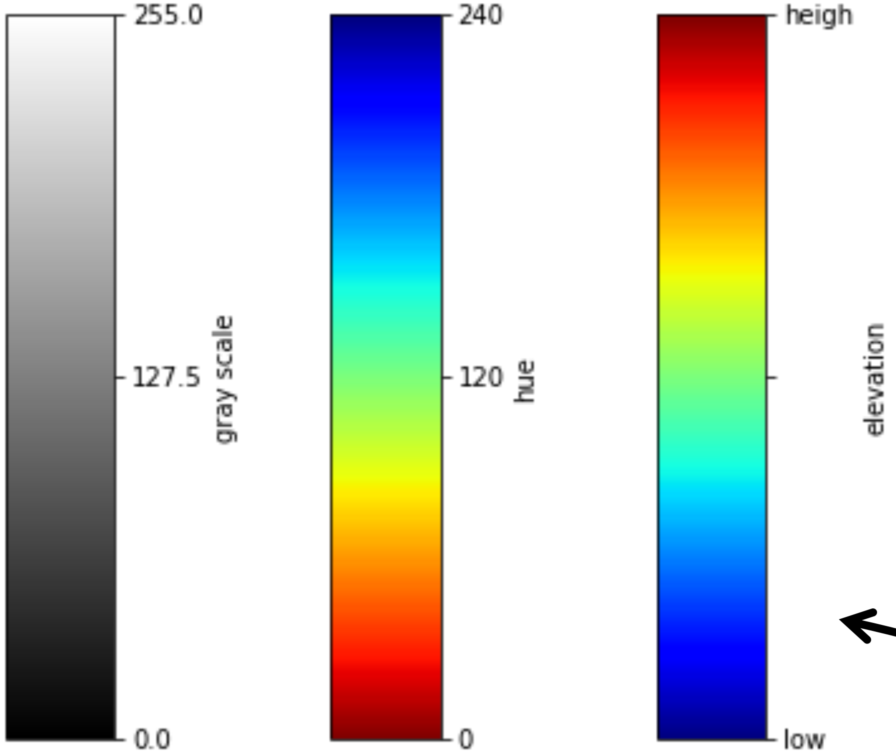
$$240 - \frac{\text{gray scale value}}{255} * 240 = \text{hue value}$$

colorbar.py

Self-Test

Calculation of hue value – how it works:

what we have



what we want to have

```
def getHue(v, max):
        value = int(240 * (v / max))
        return 240 - value
```

Self-Test

- Create visualization:

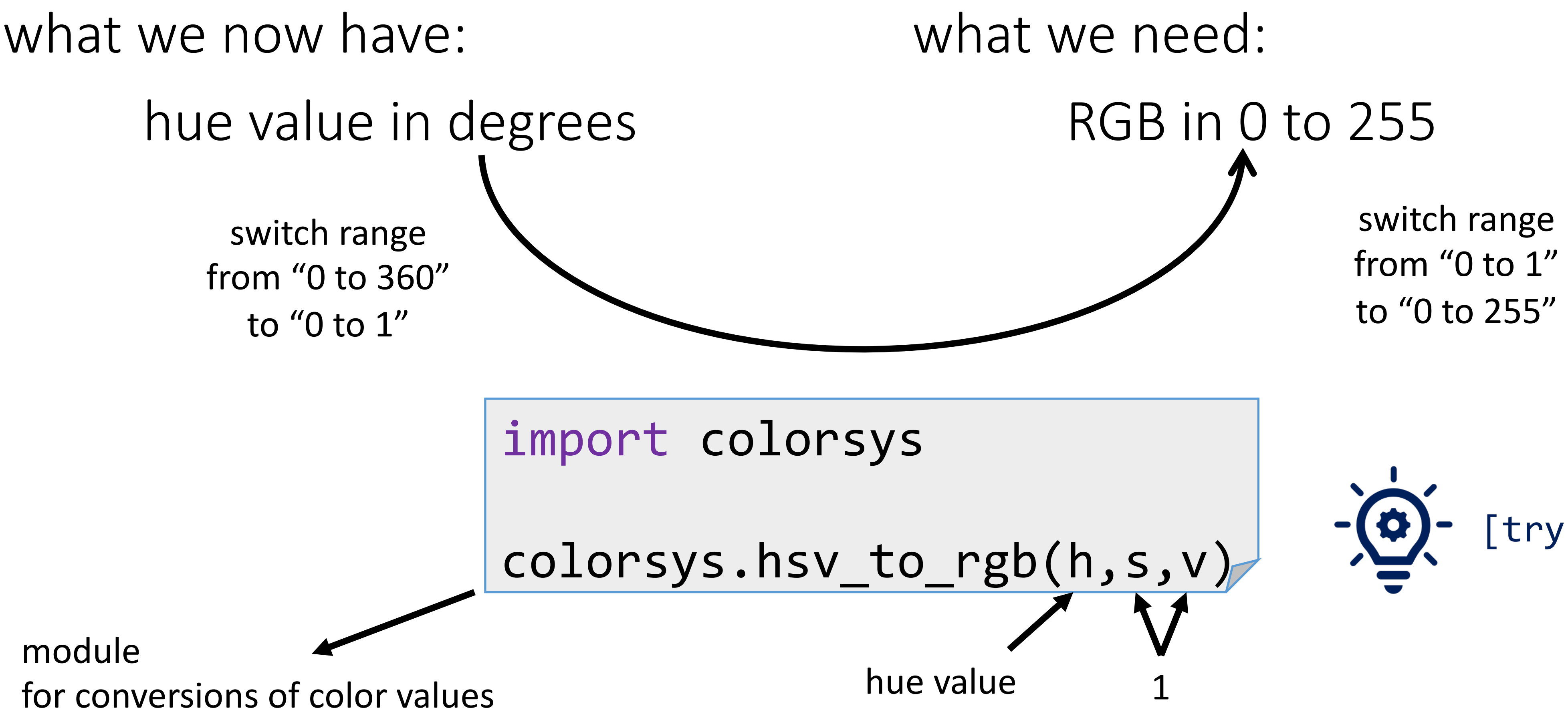
Algorithm:

- Create empty image
- For each data value and corresponding pixel
  - Read data
  - Calculate color value
  - Fill pixel with data
- Show image
- Save image

```
image = Image.new("RGB", …
pixelMap = …

for y in range(DIM):
    for x in range(DIM):
        data = dataArr[y*DIM+x]
        hue = getHue(data, …)
        hue_rgb = hsv2rgb(hue, …)
        pixelMap[x, y] = hue_rgb
image.show()
image.save(<file>)
```

Self-Test

Hue in RGB = transformation from HSV to RGB:



what we now have:

hue value in degrees

what we need:

RGB in 0 to 255

switch range from “0 to 360” to “0 to 1”

switch range from “0 to 1” to “0 to 255”

```
import colorsys

colorsys.hsv_to_rgb(h,s,v)
```

module
for conversions of color values

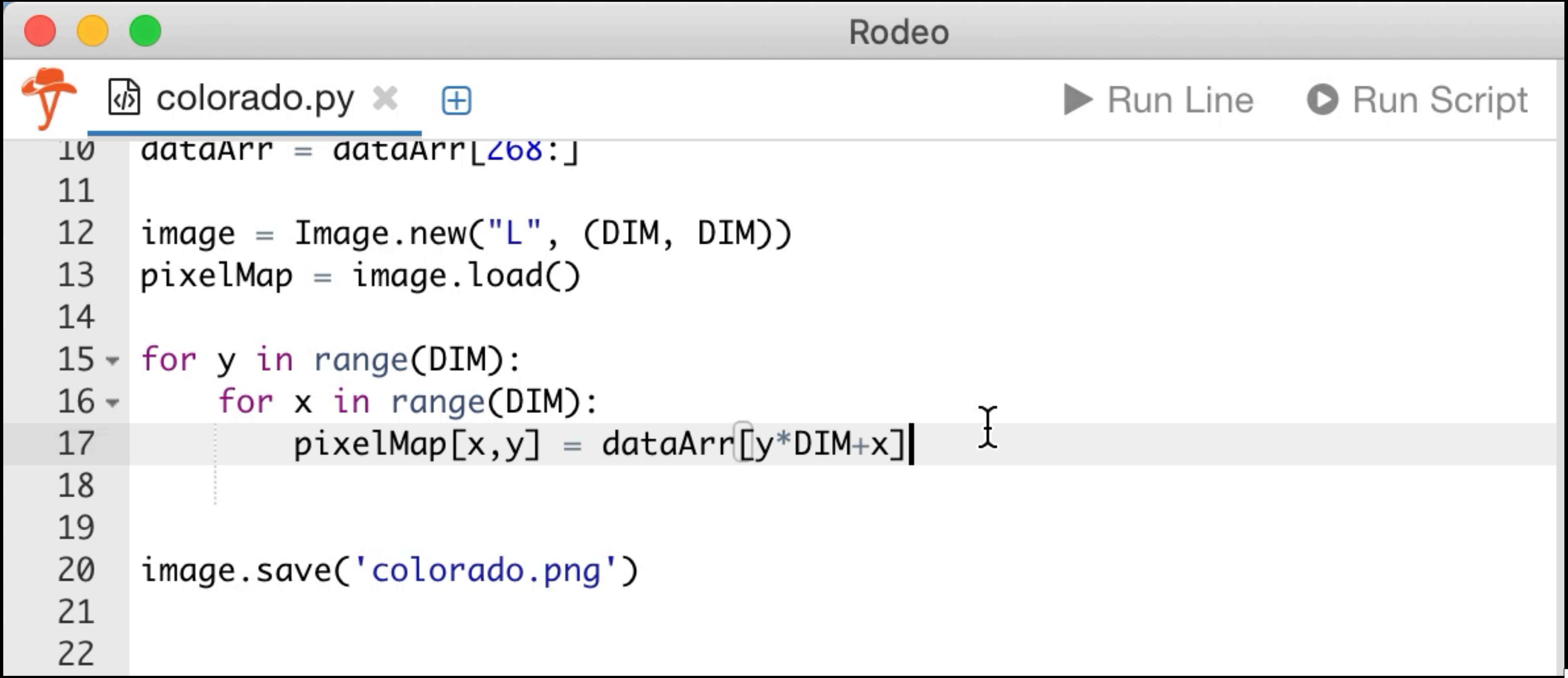
hue value

1

[tryit]hsv_to_rgb.py

Self-Test

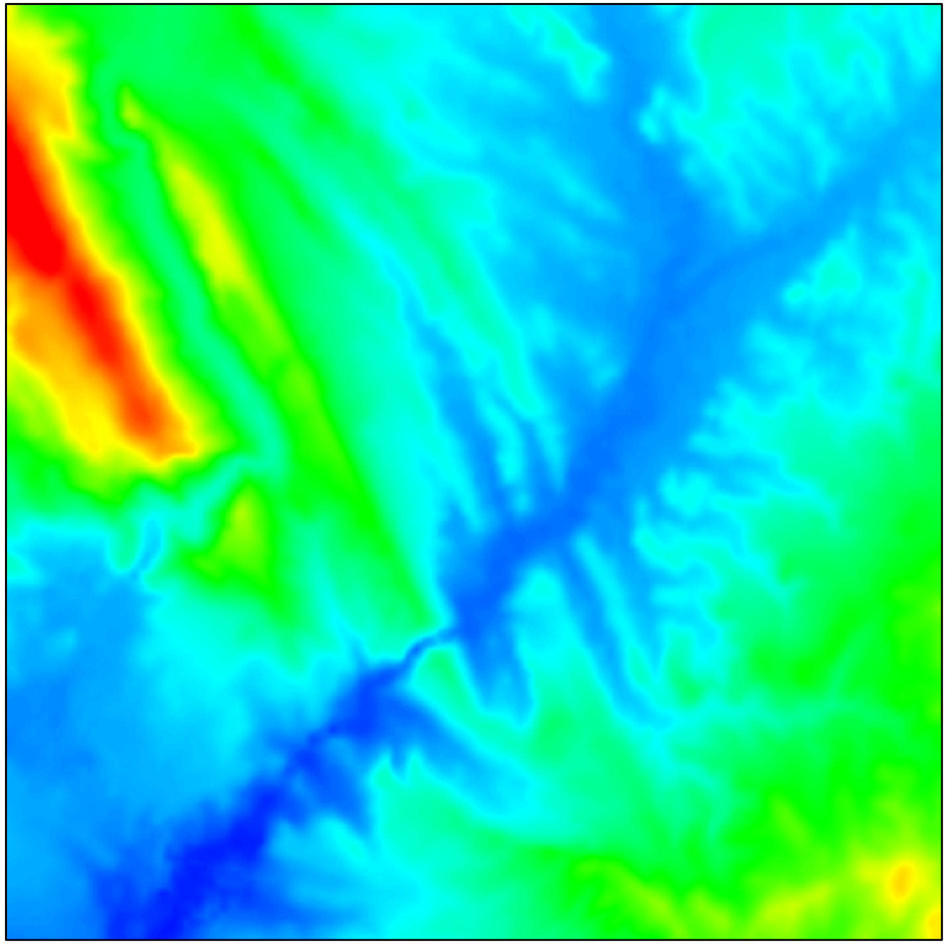
Create a color image:



```
dataArr = dataArr[268:]

image = Image.new("L", (DIM, DIM))
pixelMap = image.load()

for y in range(DIM):
    for x in range(DIM):
        pixelMap[x,y] = dataArr[y*DIM+x]


image.save('colorado.png')
```

Self-Test

Color image



Self-Test

Check-List

- Read data
- Save data in array for further processing
- Drop header
- Check data characteristics
- Create a basic gray scale image
- Create a color image
  - By hand
  - → Using colormaps
- Add a legend
- Adjust labels

Self-Test

Create a color image

Algorithm:

- ...
- Load gray scale image
- Create a plot
- Change colormap to hue values

Self-Test

Create a color image

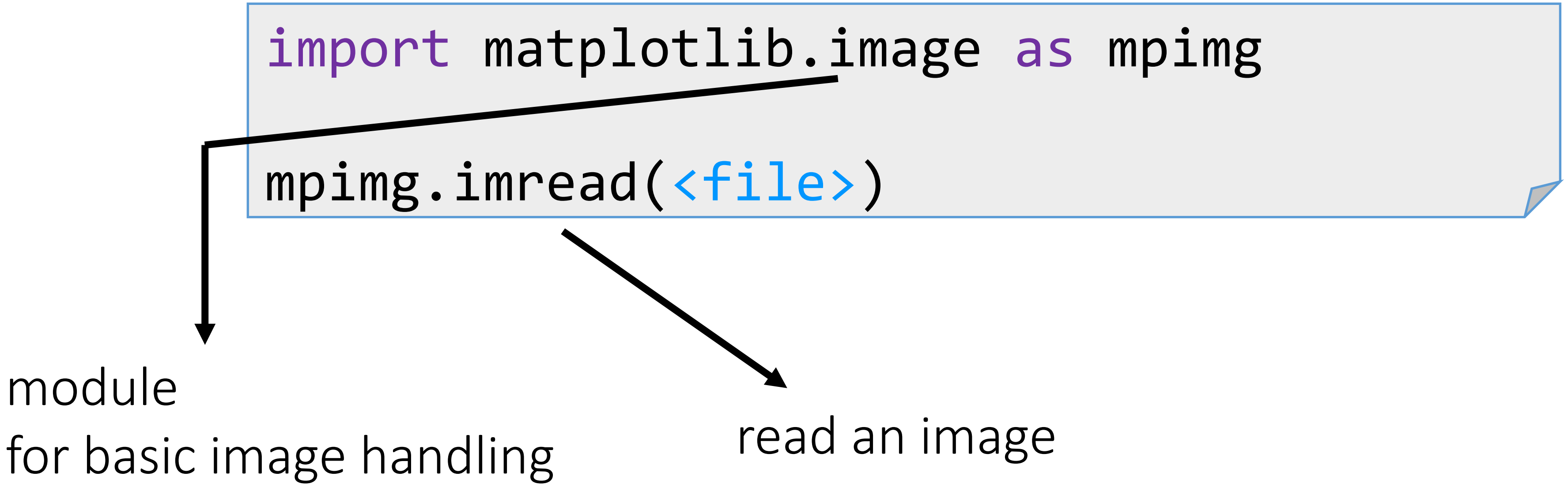
Algorithm:

```
•  ...
•  Load gray scale image
•  Create a plot
•  Change colormap to hue values
```

```
img = mpimg.imread(<file>)
imgplot = plt.imshow(img)
imgplot.set_cmap('jet')
```

Python

How to load an image?



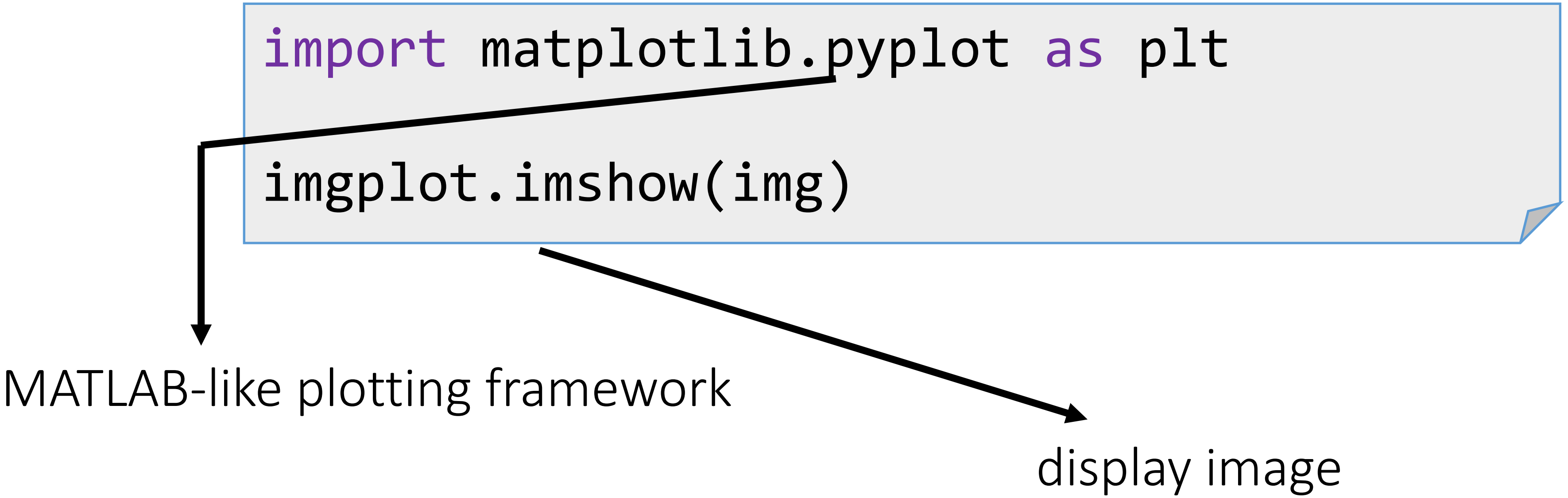
```
import matplotlib.image as mpimg

mpimg.imread(<file>)
```

module
for basic image handling

read an image

Python

How to load an image?



```python
import matplotlib.pyplot as plt

imgplot.imshow(img)
```

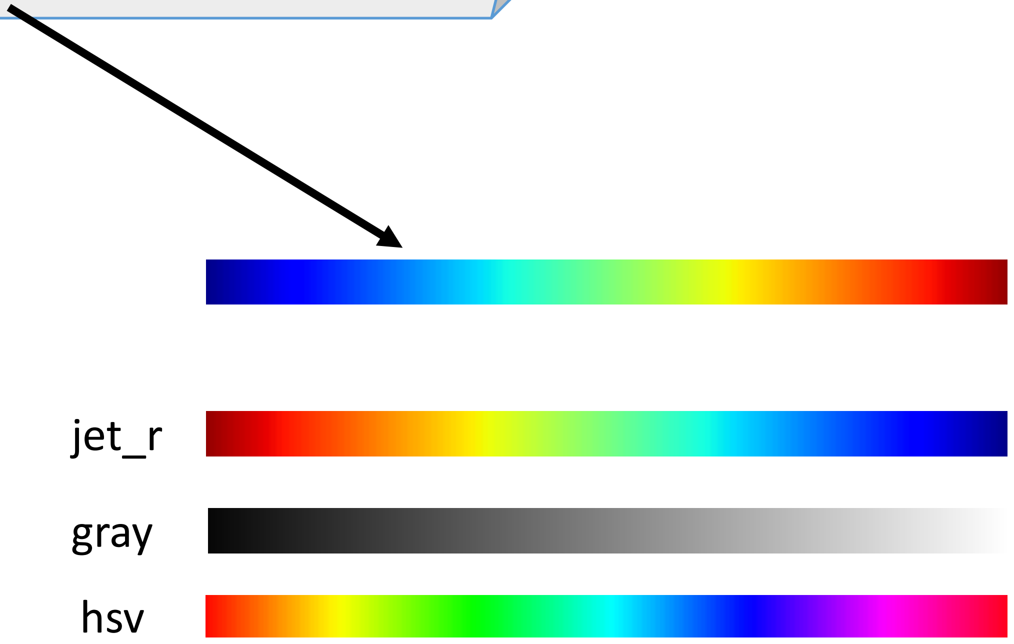
MATLAB-like plotting framework

display image

How to change the colormap?

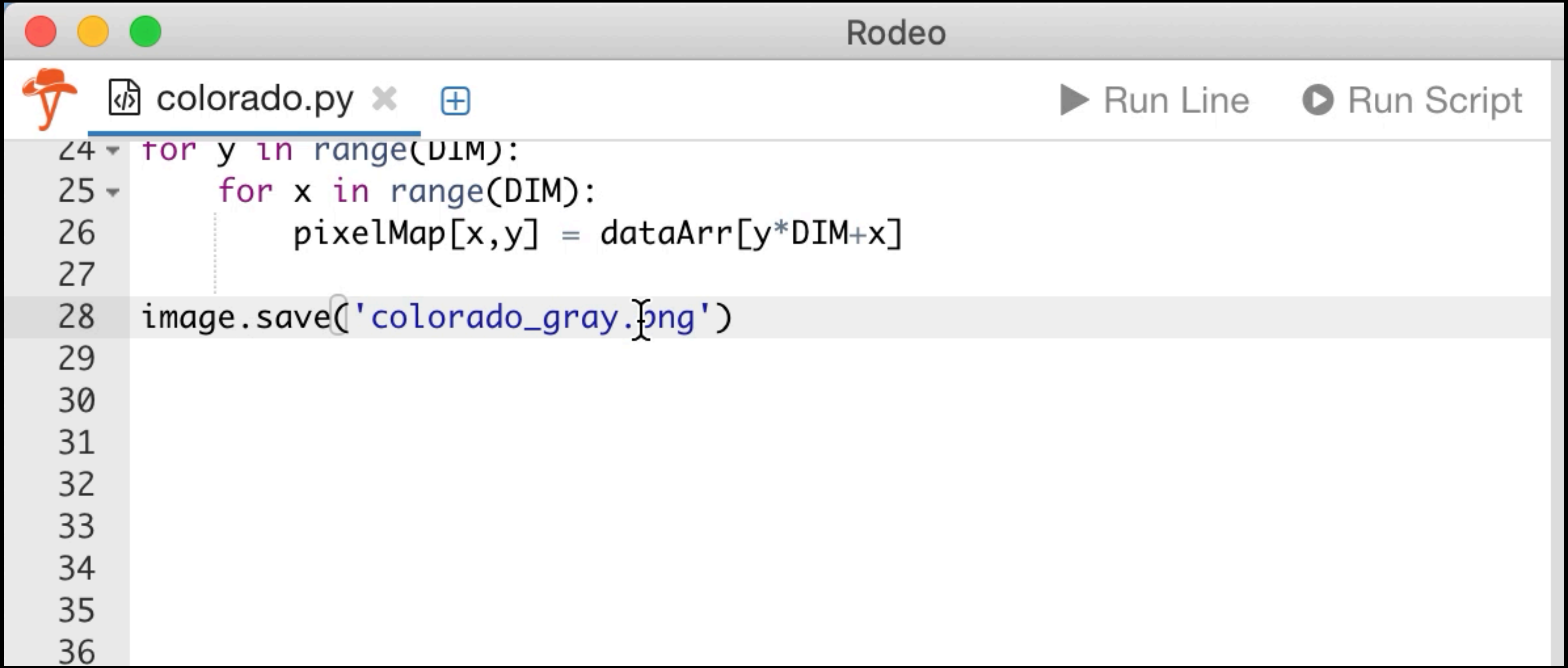
```
import matplotlib.pyplot as plt

imgplot.set_cmap('jet')
```

colormaps define how values are mapped to a color



Self-Test

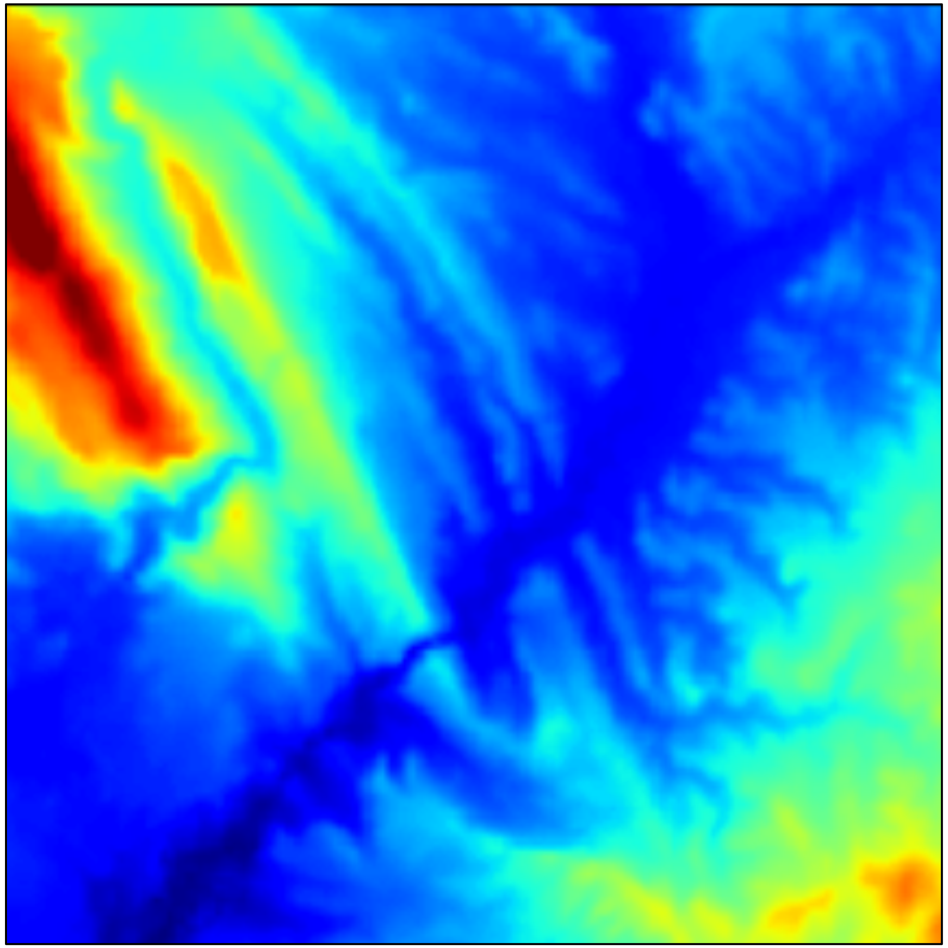
Create a color image:



```
for y in range(DIM):
    for x in range(DIM):
        pixelMap[x,y] = dataArr[y*DIM+x]

image.save('colorado_gray.png')
```

Self-Test

Color image



Self-Test

Check-List

- Read data
- Save data in array for further processing
- Drop header
- Check data characteristics
- Create a basic gray scale image
- Create a color image
- Add a legend
  - By hand
  - Predefined color bars
- Adjust labels

Note: Also for gray scale images a legend is needed.

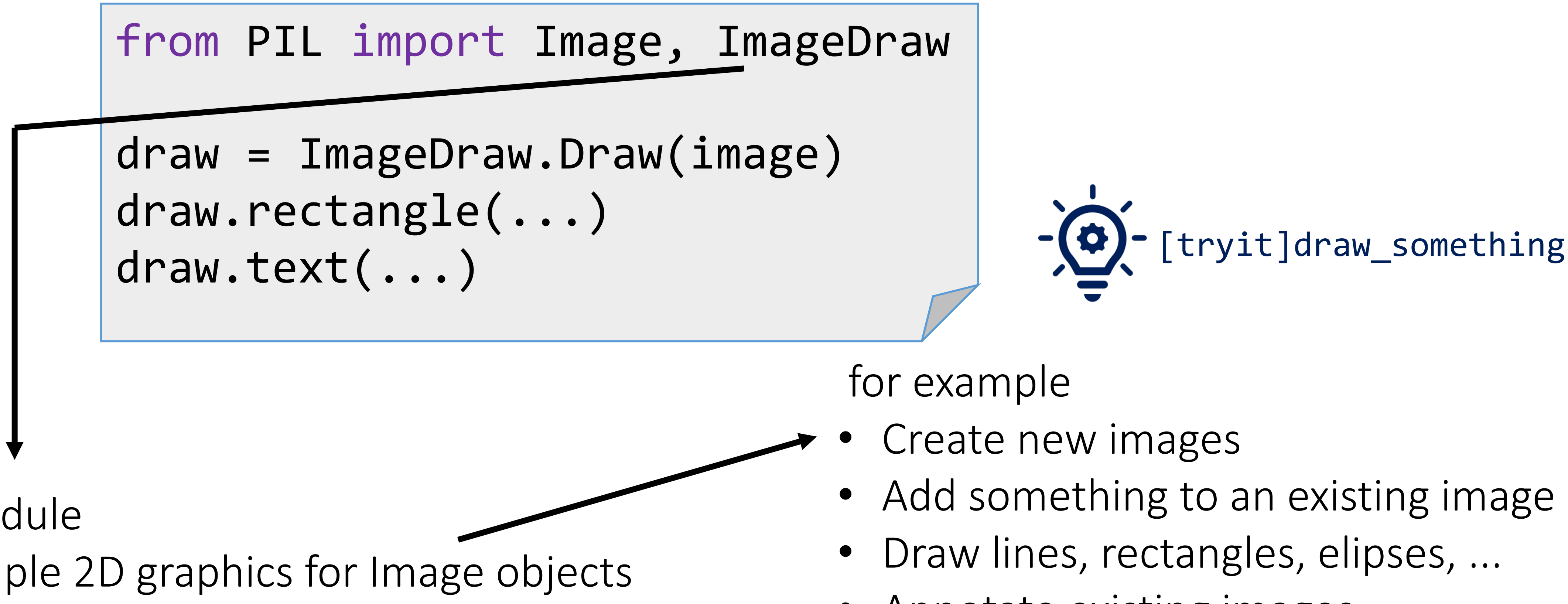
Self-Test

Check-List

- Read data
- Save data in array for further processing
- Drop header
- Check data characteristics
- Create a basic gray scale image
- Create a color image
- Add a legend
  - → By hand
  - Predefined color bars
- Adjust labels

Self-Test

Create a color bar yourself



```
from PIL import Image, ImageDraw

draw = ImageDraw.Draw(image)
draw.rectangle(...)
draw.text(...)
```

module
simple 2D graphics for Image objects

for example

- Create new images
- Add something to an existing image
- Draw lines, rectangles, elipses, ...
- Annotate existing images

[tryit]draw_something.py

Self-Test

Check-List

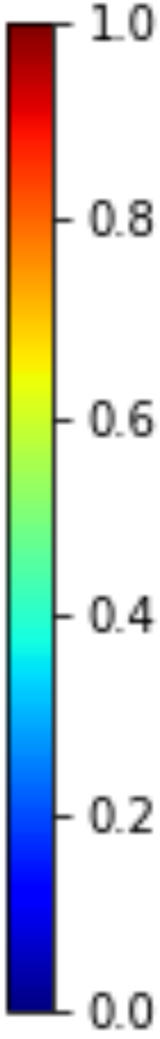
- Read data
- Save data in array for further processing
- Drop header
- Check data characteristics
- Create a basic gray scale image
- Create a color image
- Add a legend
  - By hand
  - → Predefined color bars
- Adjust labels

Python

How to create a legend / colorbar?

```
import matplotlib.pyplot as plt

cbar = plt.colorbar()
```

function for adding a color bar to an plot



Self-Test

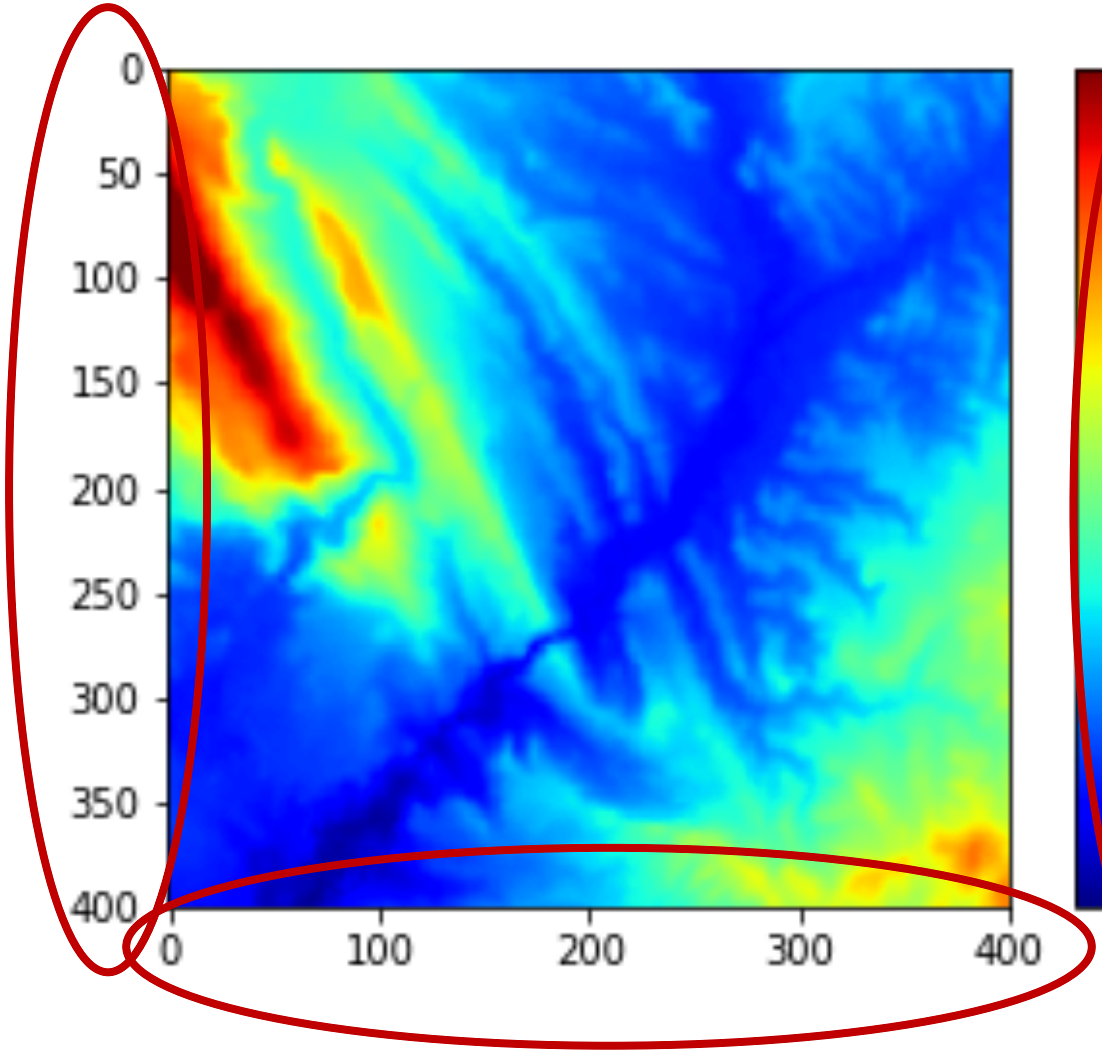
Check-List

- Read data
- Save data in array for further processing
- Drop header
- Check data characteristics
- Create a basic gray scale image
- Create a color image
- Add a legend
- → Adjust labels

Self-Test

What has to be adjust?

- ticks of the axis



- labels of the legend
- title of the legend

- position of the x-axis

Self-Test

PADERBORN UNIVERSITY
Computergraphics, Visualization & Image Processing

Adjustment of labels

Algorithm:

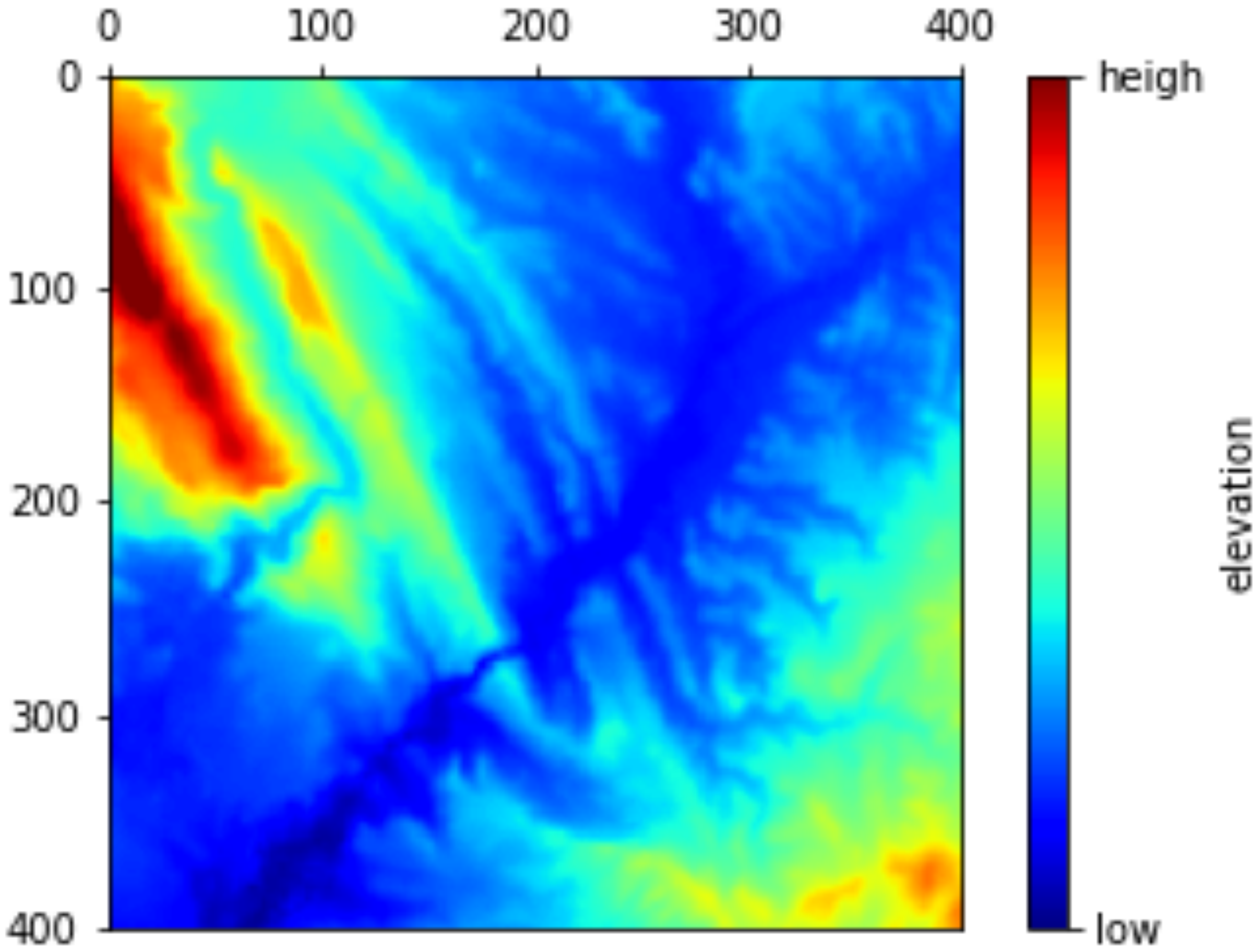
- Add titel to colorbar
- Change labels of colorbar

- Change ticks of y-axis
- Change position of x-axis

```
cbar.set_label('elevation’)
imgplot.set_norm(
        mpl.colors.Normalize(vmin=0, vmax=1))
cbar.set_ticks([0, 1])
cbar.set_ticklabels(["low", "heigh"])

ax = plt.gca()
ax.set_yticks([0, 100, 200, 300, 400])
ax.xaxis.tick_top()
```

Self-Test

What has to be adjust?



More Useful Thinks About Python

PADERBORN UNIVERSITY

Computergraphics, Visualization & Image Processing

Python

More useful libraries

Matplotlib	Visualizations
Mytplotlib.pyplot	- state-based interface to matplotlib - provides a MATLAB-like way of plotting
Numpy	Scientific computing
Pandas	Data manipulation and analysis
Plotly	Graphs (incl. interactive graphs)
Seaborn	Statistical Data visualization (based on matplotlib)
PIL	Python image library: image processing

Python

More useful functions for reading from files

`open(<file>, ...)`
`<file>.read()`
`<file>.close()`
	Opens a file and returns the corresponding file object,
	reads the content of the file and
	closes the file

`numpy.fromfile(<file >, ...)`	Construct an array from data in a text or binary file. Data type has to be defined.

Python

More useful functions matplot.pyplot for create visualizations

`figure(...)`	Create a new figure
`title(...)`	Set the title
`xlabel(...) / ylabel(...)`	Set a title for the x-/y-axis
`xlim(...) / ylim(...)`	Get / Set limits of x-/y-axis
`plot(...)`	plotting (versatile command; result depends on arguments)
`savefig(...)`	Save the current figure

Python

Useful code while working with colors:

`Colorsys`	Module in Python for converting color values of different color systems

Python

Useful functions while working with Image:

`new(...)`	Create a new image
`open(<file>)`	Open an image file
`load()`	Allocates storage for the image and loads the pixel data. Needed when pixal data is required.

Python

Other useful functions:

`round(...)`	Round a number
`range(...)`	Create a sequence of numbers (e.g. for iteration)

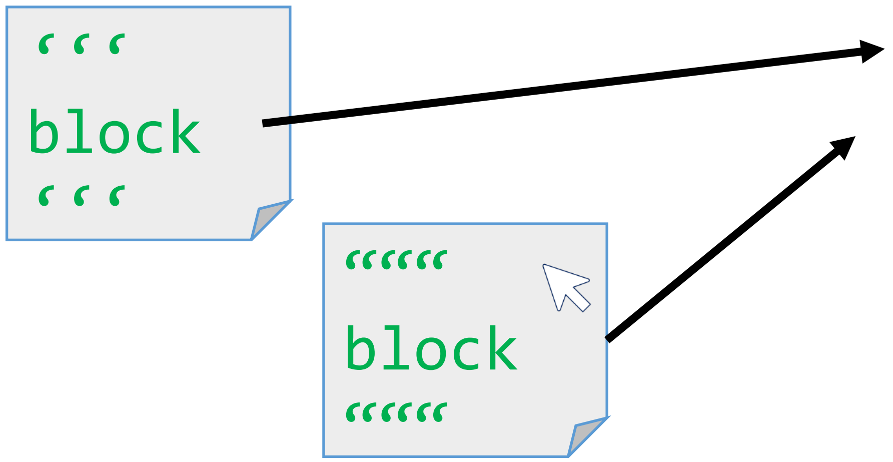
Phython

How do write comments?

```
# seperate line
```

```
x = 3    #at the end of a line
```

```
# multiple
# lines
```



```
‘‘‘
block
‘‘‘
```

```
“““
block
“““
```

Docstring

- used for documentation
- can be printed

Phython

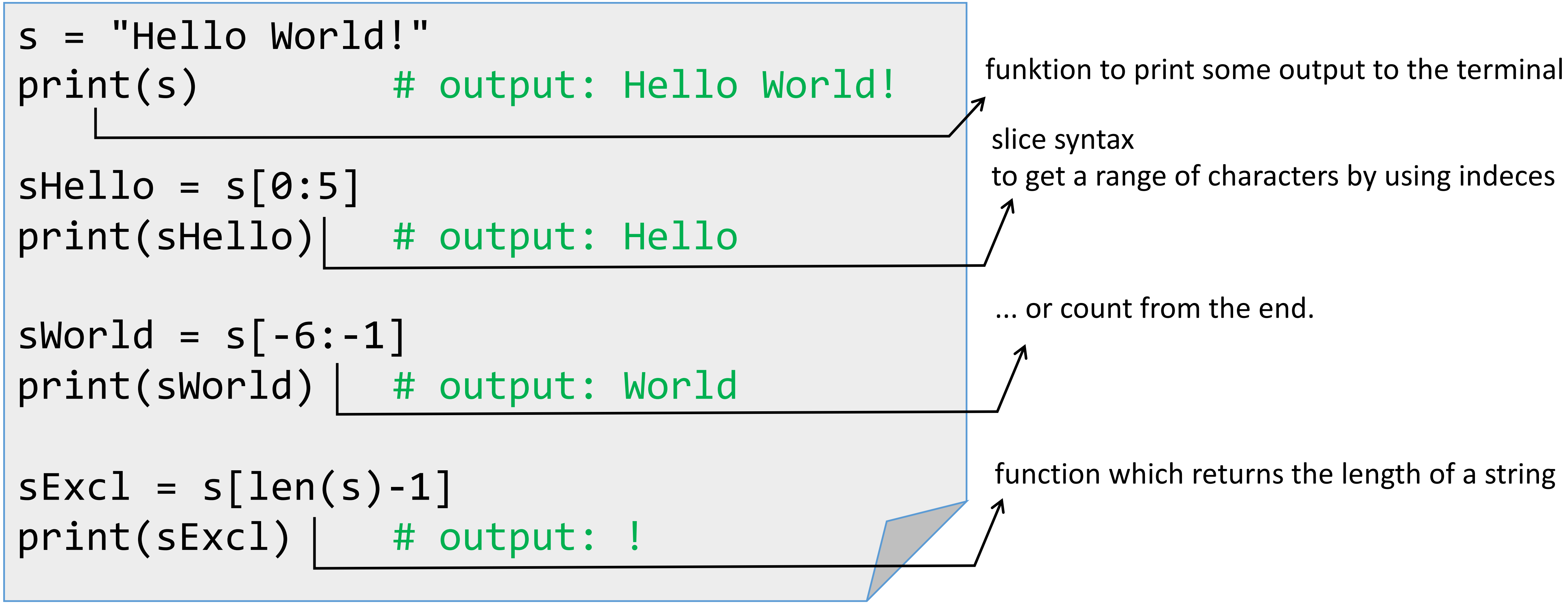
How do define variables?

```
# numbers
DIM = 400
d = 3
```

```
# string
s1 = "hallo"
s2 = 'hi'

# multiline strings
s3 = """hallo
      again"""
s4 = """hey
      you"""
```

Phython

How do work with strings?



```
s = "Hello World!"
print(s)          # output: Hello World!

sHello = s[0:5]
print(sHello)     # output: Hello

sWorld = s[-6:-1]
print(sWorld)     # output: World

sExcl = s[len(s)-1]
print(sExcl)      # output: !
```

funktion to print some output to the terminal

slice syntax
to get a range of characters by using indeces

... or count from the end.

function which returns the length of a string

Phython

How do define a function?

```
def addFunc(a, b):
      c = a + b
      print(a, " + ", b, " = ", c)
      return c

def thirdNum(*a):
    print("The 3rd nr is ", a[2])

x = addFunc(1, 3)     # output: 1 + 3 = 4
thirdNum(1, 3, x, 6)  # output: The 3rd nr is 4
```

Number of arguments is unknown

[tryit]function.py