Exercise Sheet 4

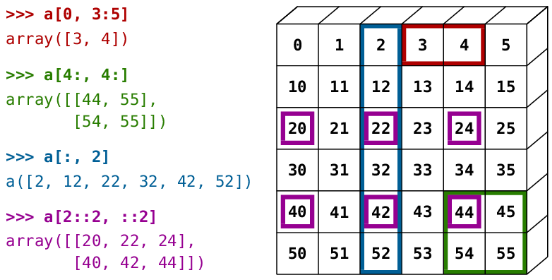
Exercise 1 - Creating arrays

a) Create a simple two dimensional array. Use the functions len(), numpy.shape() on these arrays. How do they relate to each other? And to the ndim attribute of the arrays?

b) Experiment creating arrays with arange, linspace, ones, zeros, eye and diag. Create different kinds of arrays with random numbers. Try setting the seed before creating an array with random values. Look at the function np.empty. What does it do? When might this be useful?

Exercise 2 - Indexing and slicing

a) Try the different flavours of slicing, using start, end and step: starting from a linspace, try to obtain odd numbers counting backwards, and even numbers counting forwards.



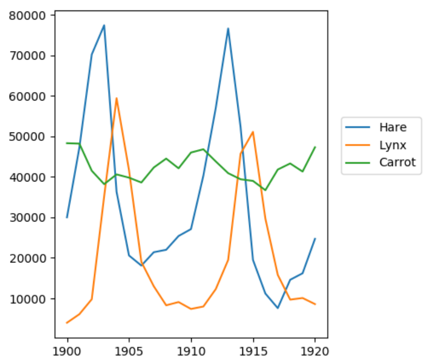
b) Reproduce the slices in the diagram above. You may use the following expression to create the array:

```
np.arange(6) + np.arange(0, 51, 10)[:, np.newaxis]
```

Exercise 3 - Data statistics

The data in populations.txt describes the populations of hares and lynxes (and carrots) in northern Canada during 20 years:

```
data = np.loadtxt('data/populations.txt')
year, hares, lynxes, carrots = data.T  # trick: columns to variables

import matplotlib.pyplot as plt
plt.axes([0.2, 0.1, 0.5, 0.8])

plt.plot(year, hares, year, lynxes, year, carrots)

plt.legend(('Hare', 'Lynx', 'Carrot'), loc=(1.05, 0.5))
```



This exercise uses the data in populations.txt. You can load this into a numpy array using

```
data = np.loadtxt('populations.txt')
```

Compute (all without for-loops) and print:

a) The mean and std of the populations of each species for the years in the period.

b) Which year each species had the largest population.

c) Which species has the largest population for each year. (Hint: argsort & fancy indexing of np.array(['H', 'L', 'C']))

d) Which years any of the populations is above 50000. (Hint: comparisons and np.any)

e) The top 2 years for each species when they had the lowest populations. (Hint: argsort, fancy indexing)

f) Compare (plot) the change in hare population (see help(np.gradient)) and the number of lynxes. Check correlation (see help(np.corrcoef)).

```
import numpy as np

data = np.loadtxt('../../../data/populations.txt')
year, hares, lynxes, carrots = data.T
populations = data[:,1:]

print("       Hares, Lynxes, Carrots")
print("Mean:", populations.mean(axis=0))
print("Std:", populations.std(axis=0))

j_max_years = np.argmax(populations, axis=0)
print("Max. year:", year[j_max_years])

max_species = np.argmax(populations, axis=1)
species = np.array(['Hare', 'Lynx', 'Carrot'])
print("Max species:")
print(year)
print(species[max_species])

above_50000 = np.any(populations > 50000, axis=1)
print("Any above 50000:", year[above_50000])

j_top_2 = np.argsort(populations, axis=0)[:2]
print("Top 2 years with lowest populations for each:")
print(year[j_top_2])

hare_grad = np.gradient(hares, 1.0)
print("diff(Hares) vs. Lynxes correlation", np.corrcoef(hare_grad, lynxes)[0,1])

import matplotlib.pyplot as plt
plt.plot(year, hare_grad, year, -lynxes)
plt.savefig('plot.png')
```

Exercise 4 - Crude integral approximations

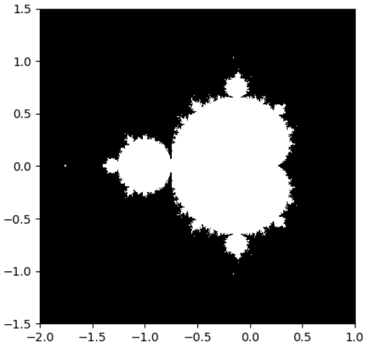
Write a function f(a, b, c) that returns $a^b - c$. Form a 24x12x6 array containing its values in parameter ranges $[0, 1] \times [0, 1] \times [0, 1]$.
Approximate the 3-d integral

$$\int_0^1 \int_0^1 \int_0^1 (a^b - c)da\, db\, dc$$

over this volume with the mean of the array. The exact result is: $\ln 2 - \frac{1}{2} \approx 0.1931\ldots$ - what is your relative error? (Hints: use elementwise operations and broadcasting. You can make np.ogrid give a number of points in given range with np.ogrid[0:1:20j].)

```
import numpy as np
from numpy import newaxis

def f(a, b, c):
    return a**b - c

a = np.linspace(0, 1, 24)
b = np.linspace(0, 1, 12)
c = np.linspace(0, 1, 6)

samples = f(a[:,newaxis,newaxis],
            b[newaxis,:,newaxis],
            c[newaxis,newaxis,:])

# or,
#
# a, b, c = np.ogrid[0:1:24j, 0:1:12j, 0:1:6j]
# samples = f(a, b, c)

integral = samples.mean()

print("Approximation:", integral)
print("Exact:", np.log(2) - 0.5)
```

Exercise 5 - Mandelbrot fractal



Write a script that computes the Mandelbrot fractal. The Mandelbrot iteration:

```
N_max = 50
some_threshold = 50

c = x + 1j*y

z = 0
for j in range(N_max):
    z = z**2 + c
Point (x, y) belongs to the Mandelbrot set if |z| < some_threshold.
```

Do this computation by:

a) Construct a grid of $c = x + 1j * y$ values in range $[-2, 1]x[-1.5, 1.5]$

b) Do the iteration

c) Form the 2-d boolean mask indicating which points are in the set

d) Save the result to an image with:

```
import matplotlib.pyplot as plt
plt.imshow(mask.T, extent=[-2, 1, -1.5, 1.5])

plt.gray()
plt.savefig('mandelbrot.png')

"""
Compute the Mandelbrot fractal
"""
import numpy as np
import matplotlib.pyplot as plt
from numpy import newaxis

def compute_mandelbrot(N_max, some_threshold, nx, ny):
    # A grid of c-values
    x = np.linspace(-2, 1, nx)
    y = np.linspace(-1.5, 1.5, ny)

    c = x[:,newaxis] + 1j*y[newaxis,:]

    # Mandelbrot iteration

    z = c
    for j in range(N_max):
        z = z**2 + c

    mandelbrot_set = (abs(z) < some_threshold)

    return mandelbrot_set

# Save

mandelbrot_set = compute_mandelbrot(50, 50., 601, 401)

plt.imshow(mandelbrot_set.T, extent=[-2, 1, -1.5, 1.5])
plt.gray()
plt.savefig('mandelbrot.png')
```