Practice Exercises

Programming and Data Processing

Table of Contents

1 Exercises ★ ☆ ☆ 3

Count Even Numbers 3

Sin(x) (5p) 3
- Version 1 3
- Version 2 3

Combining Arrays (10p) 3
- Version 1 3
- Version 2 3
- Version 3 4

Translation (10p) 4

Elfstedentocht 5

Euler-Mascheroni constant 5

Average Per Item (10p) 5

Squares (10p) 5

Circle 5

Unique Elements 6

Fractions & Factorials Sum 6

Doubles 6

Datatypes 6

2 Exercises ★ ★☆ 7

Insertion Sort 7

Matrix Multiplication 7

Prime Factorisation 7

Sub List Check 8

Geometric Mean 8

Golden Ratio 8

Average Infections 9

Plotting (15p) 9
- Version 1 9
- Version 2 9
- Version 3 10
- Version 4 10

Smallest Unique Numbers 10

Logistic Map 10
Anagrams 11
Longest increasing 11
Angular momentum 11
Perfect numbers 11
Madelung constant 11
3 Exercises ★ ★ ★ 13
Bisection Method (20p) 13
Lucas Numbers (20p) 14
Eleven Test (20p) 14
Merge Sorted (20p) 14
Bubble sort 14
Password Check (20p) 15
Approximating π (20p) 15
Another Approximation for π 15
Approximating Sine 16
Bitcoin Rates (20p) 16
Integral Approximation (20p) 16
Magic Squares 16
Neural Network 17
Rock Paper Scissors 17
Collatz Sequence 18
Finding Sequences 18
Caesar cipher 18
Distance Table 19
Temperature jumps 20
Transfer Function Plot 20
Absorbance 21
Two product problem 21

1 Exercises ★ ☆ ☆

Count Even Numbers

Write a function `counteven()` which, given a list with integer numbers, returns the amount of even numbers.

For example:

```
1. In [6]: x = [2, 4, 5, 5, 1, 2, 6]
2. In [7]: counteven(x)
3. Out[7]: 4
```

Sin(x) (5p)

Version 1

Define a NumPy array of exactly 100 numbers with the values of $\sin(x)$ in the interval $x = [-10, 10]$.

Version 2

Define a NumPy array with the values of $\sin(x)$ in the interval $x = [-10, 10]$ and a step size of 0.25.

Combining Arrays (10p)

Version 1

Given are the following NumPy arrays:

```
A : [ 1 2 3 4 5]
B : [ 2 6 9 12 13]
C : [-23 45 12 35 36]
```

Make a script which does the following using just 1 statement:

a. Combines the arrays `A` and `B` to the following array:

   ```
   D: [ 1 2 3 4 5 2 6 9 12 13]
   ```

b. Combines the arrays A, B and C to the following array (so the three arrays form the columns of the new array):

   ```
   E : [[ 1 2 -23 ]
   [ 2 6 45 ]
   [ 3 9 12 ]
   [ 4 12 35 ]
   [ 5 13 36 ]]
   ```

Version 2

Given are the following NumPy arrays:

```
A : [[  1   2   3   4   5],
     [  2   6   9  12  13]]
B : [[-23  45  12  35  36],
     [ 12 -34  56   1   2],
     [  2  -3   5   2  23]]
```

Make a script which does the following:

a. Uses slicing of B to define the following NumPy array:

```
C: [ 12  56  5]
```

b. Uses just 1 statement to combine arrays A and B to the following NumPy array:

```
D : [[  1   2   3   4   5],
     [  2   6   9  12  13],
     [-23  45  12  35  36],
     [ 12 -34  56   1   2],
     [  2  -3   5   2  23]]
```

Version 3

Given are the following NumPy arrays:

```
A : [[  1   2  3  4  5],
     [  2   6  9 12 13],
     [-23  45 12 35 36],
     [ 12 -34 56  1  2],
     [  2  -3  5  2 23]]
```

Make a script which uses slicing to define the following NumPy arrays:

```
B: [ 2  6  9 12 13]
C: [[12 35 36]
    [56  1  2]
    [ 5  2 23]]
D: [[  1   3   5]
    [-23  12  36]
    [  2   5  23]]
```

Translation (10p)

Given are two lists:

```
english = ['one', 'two', 'three', 'four', 'five']
dutch = ['een', 'twee', 'drie', 'vier', 'vijf']
```

Make a script that uses 1 statement, which uses list comprehension to make a list containing the combined text of the above lists:

```
eng_dutch = ['one=een', 'two=twee', 'three=drie', 'four=vier',
'five=vijf']
```

Elfstedentocht

You are given a list of years in which the Elfstedentocht has been held. The years are given without the century, and are all in the 1900's (so e.g. '41' corresponds to '1941').

```
tocht = [9, 12, 17, 29, 33, 40, 41, 42, 47, 54, 56, 63, 85, 86, 97]
```

a. **Using list comprehensions**, write a script that defines a new list list called `tocht_new` which includes the digits for the century (so 1909, 1912…)
b. **Using list comprehensions**, write code that defines a new list which only contains the years after '45.
c. Write a script that prints information in the following format:

```
Elfstedentocht 1 was in '09
Elfstedentocht 2 was in '12
…
```

Euler-Mascheroni constant

The Euler-Mascheroni constant ($\gamma \approx 0.577$) can be approximated by

$$\gamma = -\ln N + \sum_{n=1}^{N} \frac{1}{n}$$

This approximation gets better if N gets larger.

Write a script that calculates the approximation for $N = 1000$.

Average Per Item (10p)

Given are two lists:

`mark_1 = [7, 9, 4, 5, 7]` and `mark_2 = [6, 5, 5, 8, 9]`

Make a script which uses list comprehension to define a list `average` containing the average per item of the above lists, so `average = [6.5, 7, 4.5, 6.5, 8]`.

Squares (10p)

a. Write a script which uses list comprehension to define a list `k` consisting of the squares i^2 for all real **even** numbers $2 \leq i \leq 100$.
b. Print all the values of the list to the console using the iterator `range()`. For example:

```
The square of 2 is 4
The square of 4 is 16
```

c. Do the same as b but now use the `enumerate()` iterator.

Circle

a. Define two numpy arrays *x* and *y* that each contain 100 uniformly distributed random numbers between -1 and 1.
b. Combine these two arrays into one two-dimensional array *p* such that the first column of *p* contains all the *x* values, and the second column all the *y* values.

Each row in *p* corresponds to a random point in 2d-space.

c. Create an array *circle* that only contains the rows of *p* that correspond to points with a distance from the origin greater than 0.5, but smaller than 1.0.
Hint: the distance from the origin s is given by $s = \sqrt{x^2 + y^2}$.

Unique Elements

Write a function `unique(alist)` which, given a list, returns a new list with all unique elements of the list. It is not allowed to use the set-type or other NumPy functions implementing this in one time.
For example:

```
alist = [1, 2, 2, 4, 'a', 6, 'a', 1]
unique(alist)
>> [1, 2, 4, 'a', 6]
```

Fractions & Factorials Sum

Given is the following sum:

$$s = \sum_{i=1}^{N} \frac{1}{i!}$$

Write a script which calculates the sum for $N = 30$.

Doubles

Write a function `doubles(alist)` which returns a new list with the doubles of the given list. For example:

```
alist = [1, 2, 2, 4, 'a', 6, 'a', 1]
doubles(alist)
>> [2, 'a']
```

Datatypes

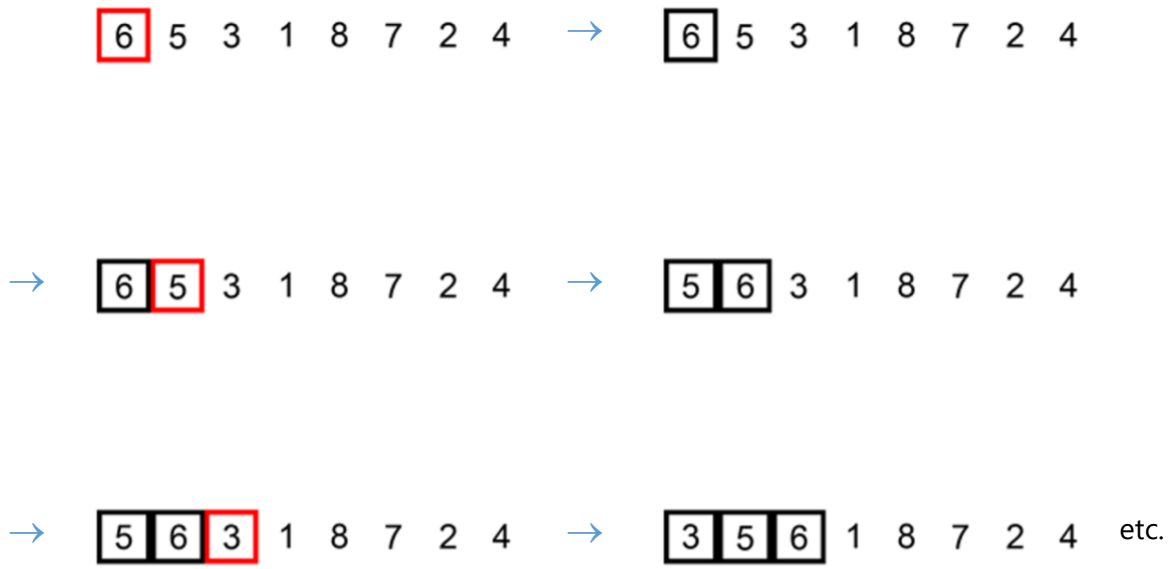
Match each kind of information with the most logical data type to represent the information.

Information		*Data type*	
A	filename	1	boolean
B	year	2	string
C	pressure measurement	3	integer
D	10 measurements in a series	4	float
E	validity of a measurement	5	numpy array

2 Exercises ★ ★☆

Insertion Sort

Implement the *insertion sort* algorithm. This is a sorting algorithm. Each time, it takes the next number in the sequence and places it on the right spot in the preceding sequence of numbers, as is illustrated in the figure below for the first three numbers.



Matrix Multiplication

Write a function `matmul(A, B)` which multiplies two matrices A and B. The matrices A and B are both 2D NumPy arrays with dimension $n \times m$ and $m \times k$ respectively, thus the new NumPy array has dimensions $n \times k$. Implement the multiplication yourself without using existing NumPy functions.

Prime Factorisation

Write a python function `prime_factorisation(n)` that takes a positive integer n as input and returns a list of prime factors of n in ascending order.
The prime factorisation of an integer involves finding the prime numbers that divide the given number exactly, without leaving a remainder. For example, the prime factorisation of 24 is $[2, 2, 2, 3]$, since $2 \cdot 2 \cdot 2 \cdot 3 = 24$.

A simple algorithm of this would look like:

1. Start with $i = 2$.

2. While n is divisible by i, divide n by i and add i to the list of factors.
3. Increase i by one and repeat the above step.

Example:

```
1. print(prime_factorisation(24))  # Output: [2, 2, 2, 3]
2. print(prime_factorisation(37))  # Output: [37]
3. print(prime_factorisation(120)) # Output: [2, 2, 2, 3, 5]
4. print(prime_factorisation(17))  # Output: [17]
```

Sub List Check

Write a function `find_sequence(test_list, sub_list)` which determines if the list `test_list` contains a certain sub list, `sub_list`. If this is the case, the function returns True and False otherwise. For example:

```
In [1]: test_list = [5, 6, 3, 8, 2, 1, 7, 1]
In [2]: sub_list = [8, 2, 3]
In [3]: find_sequence(test_list , sub_list)
Out[3]: False
```

And

```
In [1]: test_list = [5, 6, 3, 8, 2, 1, 7, 1]
In [2]: sub_list = [8, 2, 1]
In [3]: find_sequence(test_list , sub_list)
Out[3]: True
```

Geometric Mean

The geometric mean μ of a sequence of positive numbers $x_1, x_2, x_3, \dots, x_N)$ is defined as follows:

$$\mu = \sqrt[n]{x_1 x_2 x_3 \cdots x_n}$$

E.g. for the number 2 and 8, the geometric mean is equal to $\sqrt[2]{2 \cdot 8} = 4$.

Write a function `geomean(x)` which calculates the geometric mean for a given NumPy array `x`. If one or more elements of the array x is negative, the function has to return -1.

Golden Ratio

The golden ratio is number 1.618033 … . This ratio can be approximated with the following recursive relation:

$$f(n) = \begin{cases} 1 & if\ n = 0 \\ 1 + \dfrac{1}{f(n-1)} & if\ n > 0 \end{cases}$$

The bigger the value of n, the better $f(n)$ approximates the golden ratio.

a. Write a **recursive** function `golden_ratio_rec(n)` which approximates the golden ratio for a given value n.

b. Write a **non-recursive** function `golden_ratio(n)` which approximates the golden ration for a given value n.

Average Infections

We have saved data of measurements in a 1D NumPy array (see testdata.py).

```
data : [52, 52, 42, 81, 101, 71, 53, 106, 99, 103, etc]
```

The data are the amount of positive tests of a certain infection on consecutive days. We want to determine the average of 7 days of this. So, the first average is the average of the first 7 days, the second average is the average of day 2-8, the third of day 3-9 etc. The result of the data given above is

```
run_av3 : [64.57, 72.28, 79.0, 87.71, 96.71, 108.71, etc]
```

Write a script which computes the 7-day average (as a NumPy array) of the data, without using readymade NumPy or SciPy functions.

Plotting (15p)

Version 1

a. Make a line plot (with black colour) of the function:

$$f(x) = \frac{1}{\pi}\sqrt{x+10}$$

in the interval $x = [-5, 5]$. Choose a suitable number of datapoints yourself.

b. Make a plot with a red marker on the line for each integer number of x.
c. Add a legend to the plot (choose suitable labels for both lines).
d. Add suitable axis labels and give the figure a title.

Version 2

a. Make a line plot (with colour black) of the function:

$$f(x) = x\sin(x^2) + 1$$

in the interval $x = [-2\pi, 2\pi]$. Choose a suitable number of datapoints yourself.

b. Make a second line (with colour red and a dotted line) for the function:

$$g(x) = \sin(x^2) + 2x^2\cos(x^2)$$

c. Add a legend to the plot (choose suitable labels for both lines).
d. Add suitable axis labels and give the figure a title.

Version 3

a. Write a function parabola(x, a, b, c) which calculates the values of a polynomial $y = ax^2 + bx + c$ given the values of x, a, b and c. The variables are the arguments of the function and the value of y is returned. Write the function such that x and a are mandotary arguments and b and c are optional with default value 0. For example:

```
print(parabola(1,2,c=3))
```

gives: 7

b. Make a line plot of this function for $a = 1$, $b = 2$ and $c = 3$ in the interval $x = [-10, 10]$. Choose a suitable number of datapoints yourself.
c. Plot in the same figure, using red markers (e.g. *) for the integer numbers of x.
d. Add a legend with suitable labels.
e. Add suitable text for the axes and give the figure a title.

Version 4

a. Generate a line plot (black line) of the function
$f(x) = \frac{1}{1+e^{-x}}$ on the interval $x = [-10,10]$
Choose an appropriate number of sampling points.
b. Add a second line (red dotted line) for the derivative of $f(x)$:

$$f'(x) = \frac{e^{-x}}{(1+e^{-x})^2}$$

c. Add a legend with appropriate labels.
d. Choose appropriate axis labels and give a title to your figure.

Smallest Unique Numbers

Write a function *smallest*(l, n) that returns the *n* smallest *unique* integers from the list. For example:

```
In [1]: z = [4, 2, 9, 678, 45, -45, -23, 34, 34, 0, -1, -1]
In [2]: smallest(z, 4)
```

would give

```
Out [2]: [-45, -23, -1, 0]
```

as these are the four smallest unique numbers in the list.

Logistic Map

The logistic map is a recursive function given by

$$x_n = \begin{cases} \frac{1}{2}, & n = 0 \\ rx_{n-1}(1 - x_{n-1}), & n \geq 1 \end{cases}$$

So, the starting value of the function $(n = 0)$ is $\frac{1}{2}$.

a) Write a function logistic_map(r, n) that calculates x_n for n iterations for a certain r.
b) Write code that returns x_n for $r = 3.4$ and $n = 0,1, \ldots,10$ and print this in a list.

Anagrams

Write a function `anagrams(word1, word2)` that takes in two strings and checks if they are anagrams or not. So, `anagrams('heart', 'earth')` should return True, and `anagrams('thermo', 'python')` should return False. Make sure your function is case insensitive.

Longest increasing

Write a function `length_longest_increasing(x)` that determines the length of the longest streak of consecutive increasing numbers in a given list `x`.

Angular momentum

Given are the positions r, masses m, and velocities v of several point masses. These can be found in `angular_momentum_data.py`.
The total angular momentum is given by:

$$\vec{L} = \sum_i \vec{r}_i \times \vec{p}_i$$

With $\vec{p} = m\vec{v}$ is momentum. The cross product is defined by:

$$\vec{a} \times \vec{b} = \begin{bmatrix} a_y b_z - a_z b_y \\ a_z b_x - a_x b_z \\ a_x b_y - a_y b_x \end{bmatrix}$$

Write a script that calculates the total angular momentum L of the system.

Perfect numbers

A perfect number is a natural number which is exactly equal to the sum of its divisors. The smallest example of a perfect number is 6: it is the sum of its divisors 1, 2, and 3 (note: the number itself is not considered a divisor here). Another example is 28 (divisors 1, 2, 4, 7, 14).

a) Write a function `perfect(n)` that determines if a gives number is a perfect number. The function should return a boolean.
b) Use your function to determine the first three perfect numbers.

Madelung constant

The *Madelung constant* α of NaCl can be calculated using a series:

$$\alpha = 6 - \frac{12}{\sqrt{2}} + \frac{8}{\sqrt{3}} - \cdots + \cdots = 1.7476 \ldots$$

However, this series converges very slowly. A better series is the following:

$$\alpha = 12\pi \sum_{\substack{m\geq 1 \\ m\ odd}} \sum_{\substack{n\geq 1 \\ n\ odd}} \cosh^{-2}\left(\frac{\pi}{2}\sqrt{m^2+n^2}\right)$$

Note that $\cosh^{-2}(x) = 1/\cosh(x)^2$, and that the sums only use odd m and n values.

Calculate the Madelung constant using the series above, for m and n both up to 17.

3 Exercises ★ ★ ★

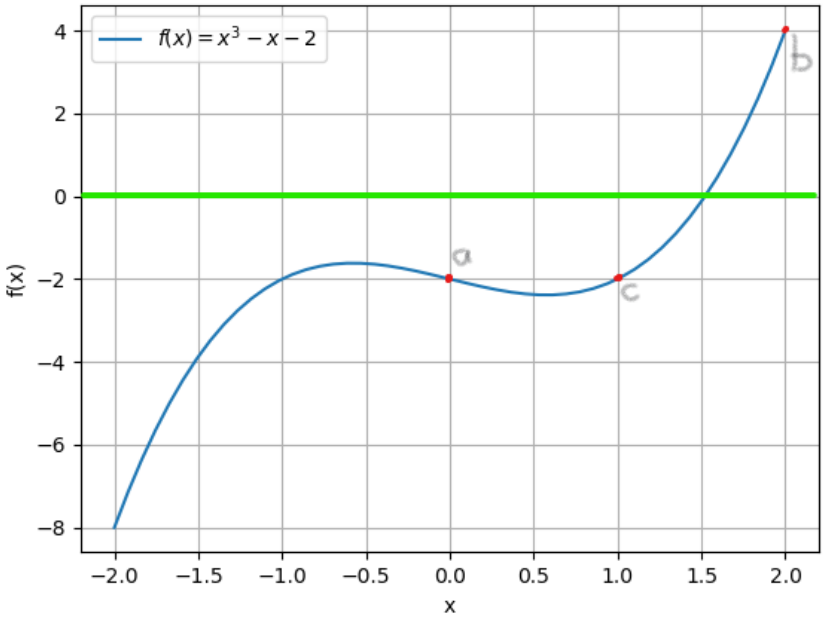
Bisection Method (20p)

Given is the function:

$$f(x) = x^3 - x - 2$$

Write a script which uses the *bisection method* to find the intersection of the function with the x-axis within an accuracy of 1^{-10}.

The function is plotted below and shows the intersection with the x-axis is somewhere between $x = 0$ and $x = 2$.
In the *bisection method*, you start with two limits $x = a$ and $x = b$, in between which lies the intersection (in the figure this is $a = 0$ and $b = 2$). Hence it holds the sign of $f(a)$ and $f(b)$ are opposite provided there is only one intersection between the coordinates. In this case, $f(a)$ is negative and $f(b)$ is positive. Next, you choose a point c in the middle of this interval (so $c = 1$ in this case) and you evaluate the function value $f(c)$. Depending on the sign of $f(c)$, the values of a or b is replaced with c so that the intersection lies again in between a and b. Here, a will be replaced with c because $f(c)$ has the same sign as $f(a)$ resulting in a new interval $a = 1$ and $b = 2$. By repeating this procedure with the new interval, the limits a and b will near the intersection more each time, and therefore an increasingly better estimate of the intersection point is found.



Lucas Numbers (20p)

De Lucas numbers L_n compose a sequence defined as follows:

$$L_n = \begin{cases} 2 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ L_{n-1} + L_{n-2} & \text{if } n > 1 \end{cases}$$

The first Lucas numbers are hence:

$$2, 1, 3, 4, 7, 11, 18, 29, 47, 76, 123, \ldots$$

Write a script that calculates the sum of the first 25 Lucas numbers.

Eleven Test (20p)

Write a function `eleven_test(account)` which determines if a bank account number satifies the 'eleven test'. The function has an input which is an integer with a length of 9 digits and an output which is a boolean (True if number satisfies the condition, and False if it does not).

The eleven test goes as follows. Take a bank account number, for example 736160221. You multiply every digit with its position and calculate the sum:

1 × 1 + 2 × 2 + 3 × 2 + 4 × 0 + 5 × 6 + 6 × 1 + 7 × 6 + 8 × 3 + 9 × 7 = 176.

The sum of 176 is divisible by 11 so this bank account number meets the eleven test.

Merge Sorted (20p)

Write a function `merge_sorted(list1, list2)` which combines two sorted lists so that the new list is still sorted. Implement the function yourself without using existing python or NumPy sort functions.
For example:

```
list1 = [1, 4, 6]
list2 = [2, 3, 5]
merge_sorted(list1, list2)
>> [1, 2, 3, 4, 5, 6]
```

Bubble sort

Write a function `bubblesort(x)` that sorts the list `x` in ascending order using the *bubble sort*, which is an algorithm that works like this:

1. Take the first two element of x
2. Swap them if they are not in the right order
3. Now shift over to the second and third element
4. Swap them if they are not in the right order

5. Repeat this for the third and fourth element, 4th and 5th, etc. until the end of the list.
6. Now check if you made any swaps in steps 1 to 5. If so, start at step one again and do all the steps. If there were no swaps, then the list is sorted, and you are done.

Examples and more details of this algorithm you can find on e.g. Wikipedia if needed.

Password Check (20p)

Write a function `passwordcheck(password)` which checks if the string `password` satisfies to the following requirements:

1. At least 1 letter [a-z];
2. At least 1 number [0, 9];
3. At least 1 symbol [#@&];
4. At least 8 characters long.

The function returns True if the conditions are satisfied, otherwise it returns False.
For example:
`passwordcheck('ght23&ut')` gives `True;`
`passwordcheck('ert1trew')` gives `False`.

Approximating π (20p)

We want to approximate the number π with the following series:

$$\sum_{n=0}^{N} 4\frac{-1^n}{2n+1}$$

a. Write a function `pi_approximation(N)` which calculates the above sum given a value for the number of terms N.
b. Which value of N needs to be chosen if you want π accurate for up to 3 decimals? Use the script you write in a.

Another Approximation for π

We are going to approximate π^2 using the following formula:

$$\pi^2 \approx 8 + 16\sum_{n=1}^{N} \frac{1}{(2n+1)^2(2n-1)^2}$$

a. Write a function `pi2_approx(N)` which evaluates the sum above for a given value of N.
b. Adapt your function so that it will keep adding terms until the difference between π^2 and your approximation is less than 10^{-6}.

Approximating Sine

a. Write a function `taylor_sine(x, N)` which calculates the Nth order Taylor approximation of $\sin(x)$:

$$\sum_{n=0}^{N} \frac{(-1)^{n+2} x^{2n+1}}{(2n+1)!}$$

b. Write a script that uses the function to determine the amount of terms N needed to get an accuracy for the sum of 5 decimals for $x = \pi/4$.

Bitcoin Rates (20p)

Given is a NumPy array with Bitcoin rates in euros as a function of the time (increasing). For example: `btc_euros = np.array([102, 95, 110, 75, 43, 98, 101, 88])`

To make profit, you first need to buy bitcoin at a low rate and sell bitcoin when the prices are high. In the above example, you could buy for € 102 and sell at € 110 to make a profit of € 8. If you can only buy once and sell once, what is the maximum profit? In the example this is € 101 - € 43 = € 58. Write a function `optimize_profit(btc_euro)` which determines the maximum profit.

Integral Approximation (20p)

Write a script that approximates the integral of the function $f(x) = x\sin(x)$ in the interval $x = 0$ to $x = 1$. We assume we sum over a nonnegative function in the interval. We can divide the interval into N equally divided subintervals (of length Δ). For each subinterval i we can use the value x_i in the middle of the subinterval to approximate the integral with

$$\sum_{i=1}^{i=N} f(x_i)\Delta$$

Calculate this sum for $N = 10$, $N = 100$ and $N = 1000$.

Magic Squares

A magic square is square for which the sum of the numbers in each row, column and diagonal has the same result. For example:

8	**3**	**4**
1	**5**	**9**
6	**7**	**2**

In this case, the sum of each row, column and diagonal is equal to 15.

Write a function `magic_square(m)` which checks if a square is a magic square for a given 2D NumPy array `m`. The function returns True if the square is a magic square and False otherwise.

Neural Network

In a neural network, the last layer of neurons is often described with the softmax function:

$$y_i = \frac{e^{x_i}}{\sum_i e^{x_i}}$$

Here x is a vector with N elements $(x_1, x_2, x_3, \ldots, x_N)$ and the sum in the denominator is the sum over all elements in the vector. Given the NumPy array in neural_network_data.py:

```
x : [ 0.99   0.2   0.1   0.4   0.7   etc]
```

Write a function that computes the softmax function for a given vector x and a given index i of an element in that vector.

Rock Paper Scissors

Write a function `rock_paper_scissors(player1, player2)` which determines which players has one the game. The input of the function are the choices of the players as strings with the possibilities:
`'rock'`, `'paper'` or `'scissors'`.

The output of the function is an integer with the following connotations:

-1: one or more of the players' input is invalid
0: draw
1: player 1 won
2: player 2 won

The rules of the game are: rock beats scissors, scissors beats paper and paper beats rock. Example outputs:

```
player1 = 'rock'
player2 = 'paper'
rock_paper_scissors(player1, player2)
>> 2
player1 = 'rock'
player2 = 'rock'
rock_paper_scissors(player1, player2)
>> 0
player1 = 'coconut'
player2 = 'monkey'
```

```
rock_paper_scissors(player1, player2)
>> -1
```

Collatz Sequence

De Collatz sequence starts with a given natural number n. The next number in the sequence is decided as follows:

$$\begin{cases} \frac{n}{2} & (n\ even) \\ 3n+1 & (n\ odd) \end{cases}$$

The process repeats on the obtained number. The sequence terminates when it reaches 1.
Example sequence:

$$12, 6, 3, 10, 5, 16, 8, 4, 2, 1$$

According to the Collatz Conjecture, the sequence will always terminate (i.e., n will eventually reach 1), no matter which n you start with. (Bonus points: prove this, and send the proof to a TA)

Write a function `collatz(n)` that calculates the number of steps needed for the sequence to reach 1, given a starting number n. (In the above example, that would be 9)

Finding Sequences

Write a function `find_sequence(test_list, sub_list)` that looks if the `sub_list` is a part of the `test_list`. If `test_list` contains the `sub_list`, it should return `True`, and `False` otherwise. For example:

```
In [1]: test_list = [5, 6, 3, 8, 2, 1, 7, 1]
In [2]: sub_list = [8, 2, 3]
In [3]: find_sequence(test_list , sub_list)
Out[3]: False
In [4]: sub_list2 = [8, 2, 1]
In [5]: find_sequence(test_list , sub_list2)
Out[5]: True
```

Caesar cipher

The Caesar cipher is an old way to encrypt messages. It works by shifting the letters in the alphabet by a certain number of places. If you use a shift of 3 (this number is called the *key*), the shifted alphabet will look like this:

```
Alphabet | ABCDEFGHIJKLMNOPQRSTUVWXYZ
Shifted  | XYZABCDEFGHIJKLMNOPQRSTUVW
```

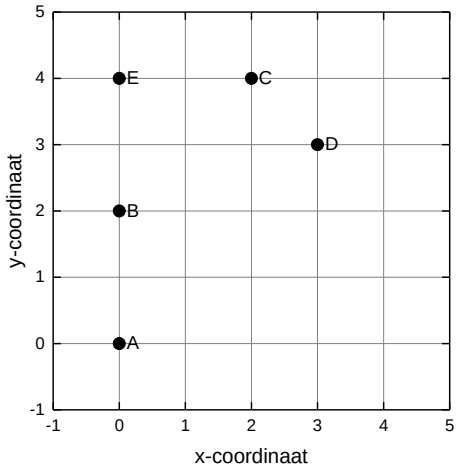
To now encode a secret message, you take your letters in the message and replace them with the letters in the shifted alphabet. So an ‘A’ becomes and ‘X’, a ‘B’ becomes ‘Y’, etc. For decoding an encrypted message, you need to do the shift the other way around. In this case, you could use the bottom alphabet and take the corresponding letter in the top alphabet.

a) Now, write a function `caesar_decode(cipher_text, key)` that *decodes* a cipher text, using a shift of `key`.

HINT: Define the alphabet as a string: `alphabet = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'`. *You can then find the index of a letter by using* `alphabet.index()`. *E.g.* `alphabet.index('E')` *gives 4.*

b) Decode the cipher text `'FTQWQKIMERAGDFQQZ'` using your code, without knowing the key. Use your function from a).

Distance Table



Map of the cities A, B, C, D and E.

We want to make a distance table of locations on a map (see figure above). On the map, each city has a *x* and *y*-coordinate, so that we can write an array with their locations like this: `loc = np.array([[0,0],[0,2],[2,4],[3,3],[0,4]])`

The distance table contains the shortest distance between each pair of two cities. It will look like the table below (where 0 corresponds to A, 1 to B, etc.). A few distances have already been filled in.

Row/column	*0*	*1*	*2*	*3*	*4*
0	0	2	-	-	-
1	2	0	-	-	-
2	-	-	0	-	-
3	-	-	-	0	-
4	-	-	-	-	0

a) Write a script that outputs a filled in distances table, with `loc` as input. For the table, use a 2d-Numpy array.

Now we want to find a route passing through each city. We denote such a route like this: `route = [0, 3, 4, 2, 1]` for a route that start at city A, then goes to D, E, C and ends at B.

b) Write a function `dist_travel(table, route)` that calculates the total length of a given `route` list, using the distance_table `table`.

Temperature jumps

We are given a Numpy array containing temperature measurements. For example, take
`T_data = np.array([10.2, 9.5, 11.0, 7.5, 4.3, 9.8, 10.1, 8.8])`
We are looking for the greatest temperature increase within this data set. This does not have to happen on consecutive days. In the example data above, the largest temperature increase is 10.1 – 4.3 = 5.8.

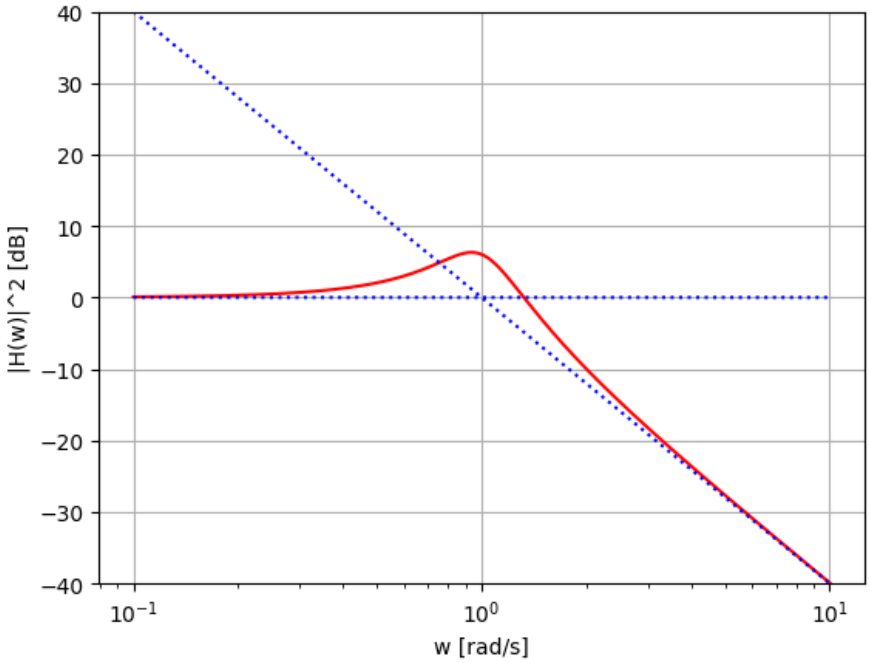
Write a function that takes in an array `T_data` and returns the largest temperature increase.

Transfer Function Plot

For a second order system, the amplitude of the *transfer function* is given by

$$|H(\omega)| = \frac{k}{\sqrt{\left(1 - \left(\frac{\omega}{\omega_n}\right)^2\right)^2 + 4\zeta^2\left(\frac{\omega}{\omega_n}\right)^2}}$$

We use $k = 1, \omega_n = 1, \zeta = 0.25$. See the figure below.



Write a script that reproduces the figure *exactly*.

Note: The *x*-axis is logarithmic, and the *y*-axis is in decibels! So plot $20\log_{10}(H)$ for the correct units.

Absorbance

For this exercise, use the files absorbance_sample.csv and absorbance_blanco.csv.
Both files contain a single column of numbers. The first file contains measurements of light intensity, and the second one contains the corresponding blanco measurement. To process this data, we can calculate the absorbance using

$$A = -\log_{10}\left(\frac{sample}{blanco}\right)$$

Write a script that reads the data from these files and creates a new file absorbance.csv with a column of numbers corresponding to the absorbance of the measurement.

Two product problem

Given is: a numpy array `w` with unique integers, and an integer `product`. The assignment is to find two numbers in the array, such that their product is equal to the given product. For example, if `w = [1, 2, 3, 4, 5]` and `product = 12`, then the solution is the pair of numbers (3, 4), as $3 \times 4 = 12$.

In the case that there are multiple solutions, we choose the pair that has the *lowest sum of its indices*. So, if `w = [1, 2, 3, 4, 5, 6]` and `product = 12`, we have to options: (3, 4) and (2, 6). But the sum of the indices in the first pair is $2 + 3 = 5$ and for the second pair it is $1 + 5 = 6$. So, the function returns the first pair (3, 4).

Write a function `two_product_problem(w, product)` that solves the problem described. It should return the pair (if there is one) as a tuple, and if there is none, it should return False.