WP 1

1.1 Running Python Code

This task consists of two parts to get ourselves familiar with running Python code:

1. Start the Python interpreter in interactive mode and print a message.
2. Write a Python script that when executed prints a message. Execute the script.

1.2 Input & Output

Write a Python script that asks the user to input one line of text which you save in a variable. Afterwards output the user provided text prefixed by some text of yours on the screen in one line.

WP 2

2.1 Variables

This task consists of two parts to get ourselves familiar with creating variables:

1. Create a variable `z` of type `complex` that stores the value i. Verify that Python correctly evaluates i^2 to -1. What is the meaning of `z.real` and `z.imag`? Test with other complex variables to verify your assumption.

2. Create a variable of type `str` that stores the following text as its value:

   "That's a pity", she said.

2.2 Reading numbers & String formatting

Write a Python script that asks the user to specify a fraction by entering a numerator and a denominator, which are both integers. Store both values in variables of type `int`. Afterwards output the fraction in decimal notation limited to 3 digits after the decimal separator.

WP 3

3.1 Conditionals – Leap years

Create a program that asks for a year to test whether it is a leap year or not and output the result.

Reminder: Every year that is exactly divisible by four is a leap year, except for years that are exactly divisible by 100, but these centurial years are leap years if they are exactly divisible by 400.

Hint: You can either solve this using nested conditional statements or logical operators.

Hint: To faster test your logic omit the user input for testing purposes and instead set the variable storing the year to a fixed value.

3.2 Conditionals – Solving an equation

Create a program that asks for three real values, `a`, `b` and `c`, to be input. Print all solutions of

$$ax^2 + bx + c = 0$$

using the provided values for the coefficients a, b, and c. Make sure that your program handles special cases, e.g., division by zero, correctly without raising an error.

Hint: The required function to calculate the square root of a real value is named `sqrt` and is part of the module `math`. To use this function add `import math` at the beginning of your program. To use the function you have to specify from which module it is, i.e., you have to write `math.sqrt(...)`.

Reminder: The solution(s) of a quadratic equation can be found by $x = \dfrac{-b \pm \sqrt{b^2 - 4ac}}{2a}$.

WP 4

4.1 Lists – Vector & Matrix

This task consists of two parts to get ourselves familiar with creating list variables.

1. Create a variable `vector` of type `list` that stores a mathematical vector with all of its elements set to a constant, e.g.:

$$\begin{pmatrix} \pi \\ \pi \\ \pi \end{pmatrix}$$

   Do not explicitly write the constant n-times in your code, but use list operations.

2. Create a variable `matrix` of type `list` that stores a mathematical matrix, e.g.:

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$$

   Invoke `print` exactly three times to output the matrix with each row printed in its own line. Afterwards change a single element of the matrix.

4.2 Lists – Tables

Create a single variable of type `list` that holds name, age and height of different people by storing information of each person in a separate *row*, e.g.:

Jacob	24	1.84
Lisa	23	1.75
Alex	20	1.70

Apply the `sort` method on this list. Afterwards output this list once completely and once only the information about every other person, i.e., every other row.

WP 5

5.1 Substrings

Create a program that asks for a sentence, i.e., one line of text, to be entered and store it in a string. Create a list where each element is one word of the entered string.

Hint: Look up `help(str.split)`.

5.2 Counting characters

Create a program that asks for a sentence, i.e., one line of text, to be entered and store it in a string. Determine how often each character appears in the string.

Note: For this task knowledge from WP 6, in particular `for` loop, is required.

Hint: Use a `set` and a `dict`.

WP 6

6.1 Scalar Product

Create a program that computes the scalar product of two vectors $\vec{a}$ and $\vec{b}$ which are stored as lists.

Note: Do not use a pre-built solution, use a loop.

Reminder: The scalar product of two vectors is defined as:

$$\vec{a} \cdot \vec{b} = \sum_i a_i \, b_i$$

That is, the vectors are multiplied component-wise and the products are summed up.

6.2 Digit Sum

Create a program that asks for an integer and determines the sum of its digits.

Note: Do not use a pre-built solution, use a loop.

WP 7

7.1 Sort a list of vectors

Create a program that sorts a list of vectors, e.g.

```
vectors = [(9, 1), (8, 2), (7, 3), (6, 4), (5, 5)],
```

by their length.

Note: Use a lambda expression.

Reminder: The length of a vector `(x, y)` is given by $\sqrt{x^2 + y^2}$.

7.2 Compute π using the Monte Carlo method

Write a function that returns an approximation of the value of π. To approximate the value of π using the Monte Carlo method follow these instructions:

- Generate N pairs of positive random numbers, e.g., x and y, in the range $[0, 1)$ representing random points within the first quadrant of the unit square.
- For each pair of numbers check if these are inside the unit circle, i.e., check if the condition $x^2 + y^2 < 1$ is true.
- The ratio of the inside-count and the total-sample-count N is an estimate of the ratio of the two areas, i.e., $\frac{\pi}{4}$.

Write your function such that you can pass N as an optional argument controlling the accuracy of your approximation of π.

Hint: The function `random.uniform(low, high)` returns a sample from an uniform distribution over the half-open interval [`low`, `high`).

Note: Large values of N are necessary to achieve a good approximation.

WP 8

8.1 Fibonacci Series

Write a function that returns the n-th Fibonacci number.

Reminder: The Fibonacci number F_n is given by $F_n = F_{n-1} + F_{n-2}$ for $n > 1$ with $F_1 = 1$ and $F_0 = 0$.

Remark: While this can be implemented using an iterative approach, you shall implement it using recursive function calls. Feel free to additionally implement it using an iterative approach.

8.2 Integral

This task is divided into two steps.

1. Write a function `integrate` that approximates the area underneath the graph of a function f using the rectangle approximation, i.e., compute the value

   $$\int_a^b f(x)\,\mathrm{d}x.$$

   The integrand f shall be a parameter of your function `integrate`, i.e., your function shall have the following signature:

   ```
   integrate(f, a, b, N)
   ```

   where N is the number of rectangles used in your approximation.

   Test your code by verifying that `integrate(math.sin, 0, math.pi, 10000)` returns approximately 2 and `integrate(math.sin, 0, 2*math.pi, 10000)` approximately 0.

   Reminder: To approximate an integral using the rectangle approximation follow these steps:

   - Divide the integration interval into N blocks of equal width.
   - Calculate the area of each rectangle by evaluating the function f at the middle point of your rectangle and multiplying it by its width.
   - Sum over the area of all N rectangles to get an approximation of the integral.

2. Write a function that takes a function f as an argument and returns a function itself. The returned function shall compute an approximation of the antiderivative with lower bound set to 0, i.e., it shall compute an approximation to:

   $$\int_0^x f(x)\,\mathrm{d}x$$

   In addition your function shall have an additional parameter with a default value to set the accuracy of your approximation, i.e., your function shall have the following signature:

   ```
   antiderivative(f, N = 10000)
   ```

   Your function should be usable in the following way:

   ```
   F = antiderivative(math.sin)

   print(F(math.pi)) # Should output approximately 2.0
   ```

   Hint: Re-use your `integrate` function from the first part of this task.

WP 9

9.1 Sieve of Eratosthenes

Write a function that returns a list of all prime numbers up to a given upper boundary N using the sieve of Eratosthenes. This algorithm works as following:

1. Create a list of consecutive integers from 2 through N.
2. Set p to the smallest prime number.
3. In the list, mark all multiples of p, except p itself.
4. Find the smallest number in the list greater than p that is not marked. If there is no such number, stop. Otherwise set p to this number, which is the next prime, and repeat from step 3.

Once the algorithm terminates, the numbers not marked in the list are all prime numbers up to N.

9.2 File I/O

Imagine we wrote a small arcade game and want to save all scores of all players in a table, i.e., each row contains a player's name and score. For this purpose:

1. Write a function that appends a new score to `scores.txt`.
2. Write a function that reads `scores.txt` and returns a list of the 10 highest scores, sorted by score in descending order.

WP 10

10.1 Exception handling – Solving an equation

Do task 3.2 again, but this time using `try` blocks while still restricting the coefficients to real values. Only use complex arithmetic if required, i.e., if solutions are complex values.

10.2 Exception handling – File I/O

In task 9.2 the `scores.txt` file may contain malicious data, e.g., a row might contain a score that is not an integer. Expand your code to handle these cases.

Hint: A common approach is to go through your code and check each statement whether it might raise an exception.

Optional: Explain why the following code can not be aborted using `CTRL + C`:

```
1 import time
2
3 print("Press CTRL + C to prevent waiting very long:")
4
5 for x in range(6000):
6     try:
7         time.sleep(0.01)
8     except:
9         print("\nNope, I fooled you!")
```

WP 11

The tasks of WP 11 and 12 represent a contiguous problem. We will create basic functionality required to create a classical adventure game.

11.1 Class `Item`

We start by writing code that describes an item which well later be carried by a player. An item has a name and an effect. The effect is a boost (an integer value) of a specific characteristics of the player. You are free to choose the characteristics, e.g., health, strength, intelligence, speed and charisma. The effect might either be permanent, i.e., it is applied solely by the fact that the player is carrying the item, or is a consumable object, i.e., the boost is only applied (temporarily) after consuming the object.

We create a class `Item` that can store these information. Create an `__init__` method such that the data describing an item can be entered on its initialization. In addition add a `__str__` method allowing you to `print` instances of your class describing the item.

Hint: Once this task is finished the following code shall work:

```
items = []
items.append(Item("Holy Hand Grenade of Antioch", "strength", 10, True))
items.append(Item("Berret", "charisma", 3, False))

for item in items:
    print(item)
```

11.2 Class `Player`

In this task we start creating a class representing players. This class will be further expanded later. A player has a name and a value for each of the characteristics we have decided upon in the previous task. In addition the player has a backpack to carry items. The backpack shall be an empty list for now.

We create a class `Player` that can store these information. Again, we create an `__init__` and a `__str__` method. However, instead of specifying a value for each characteristics of the player on initialization we set a base value for each and a fluctuation which will be the same for all instances of the class. When initializing a new instance of the class `Player` the actual value for each characteristics is set by picking a random number in the range from its base value minus its fluctuation to its base value plus its fluctuation. The name of the player shall still be set on initialization as an argument. The string returned by `__str__` shall contain all information about the players current state.

Hint: Base and fluctuation of a characteristics ought to be class variables.

Hint: The actual values of the characteristics are best to be stored in a dictionary.

Hint: Once this task is finished the following code shall work:

```
players = []
players.append(Player("Dusky Joe"))
players.append(Player("Petra van Chameleon"))

for player in players:
    print(player)
```

WP 12

In this work package we continue the work started in the previous work package.

12.1 The Backpack

So far your `Player` included an empty backpack. We now add the functionality to add items to the backpack by adding an `add_iten` method to your `Player` class. This method shall expect one argument of type `Item` and add it to the player's backpack if the following conditions are satisified:

- The backpack does not have unlimited space, but only holds a certrain amount of items. It should only be allowed to add items to the backpack if that limit is not exceeded yet.

  Note: You may want to introduce a new class variable `backpack_limit` to your class `Player`.

- While a player may have arbitrary many items of the same kind if the item is consumable, they can not hold two identical permanent items. That is, a player may have two *Holy Hand Grenade of Antioch*, but only at most one *Berret*.

If the backpack limit is exceeded, the player should be asked whether they want to discard one of the items already in their backpack to pick up the new item or not pick up the new item.

Note: Don't forget to update the method `__str__` of class `Player` to reflect the fact that a backpack is not always empty.

12.2 Current Characteristics

Add a `get` method to your `Player` class. This method should take a string as a parameter and return the current value of the characteristic identified by this argument. It shall take the effect of permanent items into consideration, but ignore consumable items.

Hint: Helper methods `get_consumable_items` and `get_permanent_items` might be useful.

Hint: Once this task is finished the following code shall work:

```
items = []
items.append(Item("Holy Hand Grenade of Antioch", "strength", 10, True))
items.append(Item("Berret", "charisma", 3, False))
items.append(Item("Bullwhip", "strength", 1, False))
items.append(Item("Dune - The Desert Planet", "intelligence", 3, False))

player = Player("Dusky Joe")
print("Base strength:", player.get("strength"))

player.add_item(items[0])
player.add_item(items[1])
player.add_item(items[2])

print("With items:", player.get("strength"))
```

WP 13

13.1 Our own exception class

A typical exception one has to handle is malicious user input. However, there is no predefined exception class for these. Define your own `UserInputError` exception class and use it to implement a `inputInteger` function that takes an optional argument `prompt`, asks the user to input an integer, returns the entered integer as type `int`, and raises an `UserInputError` on malicious user input.

13.2 Class Matrix

Implement your own `Matrix` class representing a mathematical matrix. Your class shall have at least the functionality to:

- Initialize a matrix of given size.
- Convert a matrix to a string and allow it to be printed.
- Access a single element using the subscript operator, e.g., `[i, j]`.
- Add two matrices using the `+` operator.
- …

Note: Of course, various implementations of matrices are available, hence using any of these ready-made solutions – inparticular NumPy – is forbidden in this task.

WP 14

14.1 Generators – π

π can be approximated using the infinite series

$$\lim_{N\to\infty} \sum_{n=1}^{N} \frac{1}{n^2} = \frac{\pi^2}{6}$$

We want to observe the convergence of this series to approximate π by printing the intermediate results for each N. Obviously we do not want to store all values in a list, but instead use a generator.

Hint: You may use the following code as a starting point and only fill in the missing parts (...):

```
import math

def pi(N):
    s = ...
    ...
        ...
        ... math.sqrt(s * 6)

for n, s in enumerate(pi(1000)):
    print(f"Pi approximated by {n:3} summands: {s:.10e}")
```

14.2 Probability

Create a program that finds an *approximate* solution to the following question:

A stick is randomly broken into three parts.
What is the probability that the three parts can be arranged to form a triangle?

Note: If you do an experiment N times, and a certain outcome occurs n times, then $\frac{n}{N}$ is a reasonable approximation for the probability of the outcome for big N.

WP 15

15.1 Primality Test – Multiple Files

Write a function that determines whether a given number is prime or not. This function shall be implemented in `isprime.py` and be called from another `.py` file. `isprime.py`, only when run in script mode, shall inform the user about the function defined in it.

Note: Do not use your code from task 9.1 as this code is very likely very inefficient for this problem.

15.2 Bubblesort

We previously used the `sort` method or `sorted` function to sort the elements of a list. In this task we now implement our own sorting function using the Bubble sort algorithm.

Optionally: Add an additional parameter with a default value to your function to allow specifying by which criteria the list shall be sorted.

WP 16

16.1 Installing a package

In this task we install our first package. To not alter any system wide setup we do this using a virtual environment. Hence, we first create a virtual environment and then install a package within it.

Note: In case you do not know which package to install, you may install `matplotlib` which we'll need soon anyway. You may use this following code to check if the installation was successful:

```
import matplotlib

print(matplotlib.__version__)
```

Note: It is up to you on which system you perform this exercise.

16.2 Caesar Cipher

Cryptography is the study of how to make messages secret or how to read secret messages. A very simple encryption technique is called the Caesar cipher. The basic idea is that each letter is replaced by a letter that is a certain number of letters away, so for example if the shift was 2, then 'a' would become 'c', 'b' would become 'd', etc. (and 'z' will become 'b').
Write a function that given a string and a shift, will return the encrypted string for that shift.

Note: The same function can be used to decrypt a message, by passing it a negative shift.

Note: Restrict your function to only accept and return lowercase letters, spaces and dots. Spaces and dots should not be changed.

Note: The function `ord` may be used to convert a character to return its ASCII code.

Note: To test your function decrypt the following message, which was encrypted with a shift of 13:

```
"pbatenghyngvbaf lbh unir fhpprrqrq va qrpelcgvat gur fgevat."
```

WP 17

17.1 Plot a function

In this task we create two plots.

1. Create a single plot that contains two graphs in the range from 0 to $3\,\pi$:

$$\begin{aligned} f_1(x) &= \exp(-0.25\,x) \cdot \sin(2\,x) \\ f_2(x) &= \sin(-x) \cdot \cos(2\,x) \end{aligned}$$

   Label both graphs and add a legend to your plot such that one can distinguish both graphs.

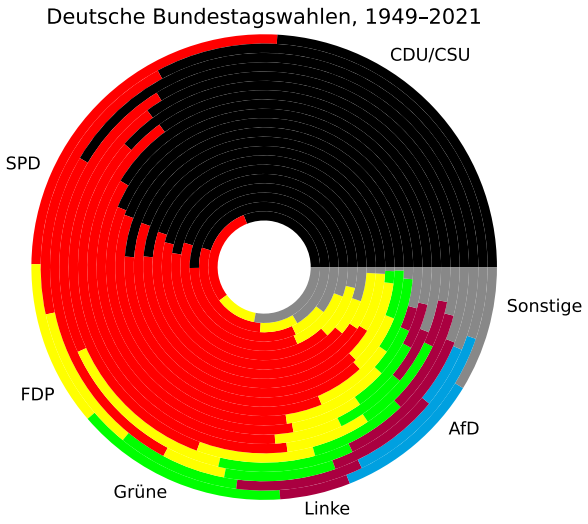
2. Create a plot of the function

$$e(x) = \sin(x) \cdot \ln(x + 0.1)$$

   in the range from 0 to $3\,\pi$ with 40 points. Each of the poijnts shall have a random errorbar between 5 to 10 % of the actual value.

17.2 German federal election results

Download the file elections.csv. It contains the results of the federal elections in (West) Germany since 1949. From these, generate the following plot:



Hint: To create nested pie plots, you can use the optional parameter radius. It can take a float value that represents the radius of the pie chart to plot. Further, the optional parameter wedgeprops controls further details about how the slices of the pie chart are rendered on screen. It takes a dict that maps certain strings to numeric values. One of these strings is 'width', which tells the matplotlib how wide a ring in the pie chart should be. I used a width of 0.05.

WP 18

18.1 Lissajous figures

Plot some Lissajous figures, i.e., plot $\sin(m \cdot \omega)$ against $\sin(n \cdot \omega + \psi)$ where m and n are integers and phase shift Ψ. Create a figure containing six Lissajous figures with different frequency ratios, e.g., 1:2, 3:2, 3:4, 5:4, 5:6, 9:8, each plotted in its own subplot.

Hint: These figures look best when an aspect ratio 1:1 is ensured for the axes.

18.2 Germany as a scatter plot

Download the file cities.csv. It contains a list of German cities, their latitude and longitude, and population. Plot these values in a scatter plot such that it allows you to see the outlines of Germany and in which part of Germany the population density is highest.

Note: To properly recognize the outline of Germany the aspect ratio of the figure is crucial.

WP 19

19.1 System of Linear Equations

Solve the following system of four equations:

$$\begin{aligned} +6x_1 - 2x_2 - 3x_3 + 4x_4 &= +7 \\ -3x_1 + 2x_2 + 5x_3 + 8x_4 &= -4 \\ +5x_1 + 8x_2 + 9x_3 + 3x_4 &= +1 \\ -8x_1 - 9x_2 + 7x_3 + 1x_4 &= -2 \end{aligned}$$

Verify the solution!

19.2 Decompositions

Compute the singular value decomposition of the matrix

$$A = \begin{pmatrix} 2.5 & 8.4 & 4.5 & 2.2 \\ 7.8 & 9.4 & 5.4 & 2.8 \\ 2.9 & 8.4 & 5.5 & 6.7 \\ 3.4 & 4.3 & 7.7 & 9.9 \end{pmatrix}.$$

Verify that the decomposition is correct!

WP 20

20.1 Brute Force Linear Optimization – Part 1

You are craving sweets but can not spend more than 10 € on them. Your trusted local dealer gave you a price list of different offered products containing sugar. You added scores to these products by taking package size and taste into account.

Product	Price per unit	Score per unit
Dark chocolate	1,00 €	5
Milk chocolate	0,90 €	4
Caramel nut bars	2,30 €	12
Coco cubes	1,50 €	7
Mint drops	0,30 €	1

Write a program that tells you how you should invest your money to best satisfy your sugar addiction, i.e., find out which configuration of sweets gives the highest score while remaining within your budget.

A possible approach is to compute the score of all possible combinations of sweets. Convince yourself that, with the given prices and budget, one can limit itself to $10 \times 11 \times 4 \times 6 \times 33 = 157080$ combinations.

Start by writing code that represents the givens (the price- and score list). From these data compute the numbers 10, 11, 4, 6, and 33, i.e., the maximum possible amount of each sweet you can buy given your budget.

Next we need all possible combinations of the numbers 0…10, 0…11, 0…6 and 0…33. Afterwards we compute the total price and score for each for these combinations.

Hint: Use NumPy's meshgrid to generate all possible combinations. To ease debugging you may want to reduce the number of different sweets, and hence the dimensionality of the issue, at first.

20.2 Brute Force Linear Optimization – Part 2

Now that we have the total prices and scores of all these combinations we have to sort out all combinations that exceed your budget.

Hint: Instead of deleting these combination, set their total score to -1 to mark them as too expensive.

Having calculated the total score of all possible combinations the only remaining task is now to find out which of these has the highest score.

Hint: While the function `np.max` finds the biggest value in an `np.ndarray`, the function `np.argmax` does not find the biggest value itself, but where that value can be found. Unfortunately, the answer will always be the flattened index, i.e., a single integer no matter how many dimensions `A` has. Fortunately, this flattened index can be translated back into a multi-dimensional index using the `np.unravel_index` function. Using these two functions, find out which of the configurations scored highest.

WP 21

21.1 Differentiation and Integration

First, calculate the derivative of

$$f(x) = a \cdot \sin(b\,x + c) + d$$

and then set $a = 1$, $b = 1$, and $c = 0$.

Second, calculate the antiderivative, i.e., the indefinite integral, of

$$g(x) = \frac{-1}{2i}\left(\exp(i\,x) - \exp(-i\,x)\right)$$

Note: To verify your results you can simply compare them as both should be equal.

21.2 Differential Equation

Solve the following differential equation:

$$y''(t) + \omega^2\, y(t) = 0$$

Note: Solutions are $y(t) = c_1 \cos(\omega\, t) + c_2 \sin(\omega\, t)$.

Optional: Solve the differential equation with initial conditions:

$$\begin{aligned} y(0) &= 1 \\ y\left(\frac{\pi}{2\omega}\right) &= 1 \end{aligned}$$

Note: The solution is $y(t) = \cos(\omega\, t) + \sin(\omega\, t)$.

WP 22

22.1 German cities

Download the file cities.csv. It contains a list of German cities, their latitude and longitude, and population. Write a program that finds the biggest and smallest city of Germany and prints the average population of a city in Germany.

22.2 German federal election results

Download the file elections.csv. It contains the results of the federal elections in (West) Germany since 1949. From these calculate the number of seats each party won. Instead of computing these using the actual rules to calculate the number of seats we calculate the number of seats according to:

1. Parties with less than 5% of the vote receive 0 seats.
2. There are a maximum of 598 seats.
3. A party's share of seats corresponds to the share of votes it received out of the total of all votes cast minus the votes for parties that did not surpass the 5%-threshold.

Save the result as a CSV or Microsoft Excel spreadsheet.

Note: The resulting file shall have a column labeled *Jahr* just like the input file.

WP 23

23.1 One-dimensional Integral – SciPy

In this exercise we consider the exponential function:

$$f(x_1) = \exp(x_1)$$

We want to integrate this function setting a specific lower bound, but leaving the upper bound open, i.e., we define a second function depending on the upper bound:

$$g(x_2) = \int_0^{x_2} \exp(x_1)\,\mathrm{d}x_1$$

Use numerical integration to calculate $g(x_2)$. Plot both functions in the range from 0 to 1.

23.2 Two-dimensional Integral

Numerically compute the integral of a two-dimensional function given lower and upper bounds:

$$\int_0^1 \int_0^1 \sqrt{x\,y}\ \,\mathrm{d}x\,\mathrm{d}y$$

WP 24

24.1 Curve fitting

In this exercise we want to fit a function of the form

$$a \exp\left(-b\,(x-c)^2\right)$$

where a, b, and c are positive real values to a dataset.
Instead of using a given dataset we create an artificial dataset for x from 0 to 1 using specific values for all parameters, e.g., by setting:

$$\begin{aligned} a &= 3 \\ b &= 20 \\ c &= 0.3 \end{aligned}$$

To make it a little harder for the curve fitting algorithm add some noise to your dataset by adding or subtracting a random number to respectively from each data point. Plot the artificial dataset and the fit in a single plot. Change the size of the noise to see the effect on the fit.

Hint: In case the fitting algorithm fails to find a sane fit, provide an initial guess close to the actual parameters to the fitting function.

24.2 Curve fitting – Polynomials

Fit a polynomial function of degree d to the artificial dataset generated in the previous task. Manually vary the degree of the polynomial function to get a best possible fit.

Note: While this task can be achieved using the curve fitting function from SciPy explicitly defining the polynomial for each degree d, it's less complicated when using functions from NumPy. In particular, it is recommended to have a look at the NumPy functions `polyfit` and `polyval`.

WP 25

25.1+2 Lotka-Volterra predator-prey model – Simulation and Visualization

We want to simulate the extent of a population of two species. One of them will be the prey of the other species, e.g., we might consider rabbits and wolves.

We assume that, for the prey species, the resources are unlimited and hence they multiply exponentially: In each time step, e.g., in each year, they grow by a given factor $1 + \alpha$. Here, α is the percentage of growth, e.g., $\alpha = 1$ means 100% growth rate or doubling of the population each year.

Example: In the first year, there are 100 rabbits and $\alpha = 1$. Then, in the second year there are 200 rabbits, in the third year there are 400 rabbits, and so on.

However, there are also predators which hunt and kill the prey species. The more predators there are, the more difficult it gets for the prey, and the more prey animals there are the easier the predators find meal. The number of killed prey animals is given by $\beta \cdot [$ number of prey animals $] \cdot [$ number of predator animals $]$, where β is some constant.

Example: In the first year, there are $x = 100$ rabbits $y = 20$ wolves and we assume $\alpha = 1$ and $\beta = 0.03$. Then the change Δx in rabbit population is:

$$\Delta x = \alpha\, x - \beta\, x\, y = 1 \cdot 100 - 0.03 \cdot 100 \cdot 20 = 40$$

So, one year later, there are 40 more rabbits and hence a total of 140 rabbits.

The predators grow proportionally to the number of prey animals they kill. Also, each year, a fixed portion of them die of old age. Hence, there are proportionality constants γ and δ that decribe how many predators are born and die each year:

Example: In the first year, there are $x = 100$ rabbits $y = 20$ wolves and we assume $\gamma = 0.01$ and $\delta = 0.5$. Then the change Δy in wolf population is:

$$\Delta y = \gamma\, x\, y - \delta\, y = 0.01 \cdot 100 \cdot 20 - 0.5 \cdot 20 = 10$$

So, one year later, there are 10 more wolves and hence a total of 30 wolves.

As a last refinement, we should not forget that not all prey and predators are born at the same time nor do they die synchronously. To reflect that fact, we evaluate the model several times per year and re-scale the effects on the predator- and prey-populations by the number of times we evaluate.

Example We compute the number of rabbits and wolves each month, i.e. $T = 12$ times per year. With this, we get for the first month:

$$\Delta x = \frac{1}{T}\,(\alpha\, x - \beta\, x\, y) = \frac{40}{12}$$
$$\Delta y = \frac{1}{T}\,(\gamma\, x\, y - \delta\, y) = \frac{10}{12}$$

This means that after one month there are $103.\overline{3}$ rabbits and $20.8\overline{3}$ wolves in total. Ignore the fact that a floating point number for the number of animals does not make biological sense. For the second month, substitute $x = 103.\overline{3}$ and $y = 20.8\overline{3}$ back into the equation, and so forth.

Implement this model to simulate the evolution of the population for several, e.g., 100, years and visualize the results in a plot.

WP 26

26.1+2 Fourier Transform of an Image

In this exercise we want to have a look at how one might compress images using Fourier transforms. For this we need an image sample to test our compression on. The function `scipy.datasets.ascent` returns a 2-dimensional Numpy array of integer values which can be interpreted as an greyscale image. Use Matplotlib's `imshow` function to display the image in a plot.
Now compute the 2-D discrete Fourier transform of that image using the `fft` module of SciPy. The transform is a 2-dimensional array of complex numbers and may also be plotted as an array by computing the logarithm of the absolute value of each point, i.e., `np.log(np.abs(transform))`.

Implement a high-pass filter, i.e., set the pixels in the transform corresponding to low frequencies to zero. Afterwards compute the inverse Fourier transform of the filtered transform and plot it. Use different cutoffs to see the effect on the reconstructed image.

Note: Due to the symmetry involved, we need to block out both ends of the spectrum.

Repeat, but this time using a low-pass filter.

Repeat, but this time use the inverse of the combination of a low- and high-pass filter, i.e., set all pixels in the transform to zero except those corresponding to high or low frequencies.

Info: This task illustrates the basic idea of JPEG compression.

WP 27

27.1 Multiprocessing – Queue

Create a program that first stores a list of work items in a queue, e.g., tuples describing which mathematical function shall be integrated including lower and upper bound. Next, create several processes that process the work items listed in the queue one at a time.

Note: Ensure that each work item is only processed once!

27.2 Multiprocessing – Pipes

Create a program which starts two processes. Both processes essentially do the same: First, they acquire a random number and send this number to the other process. Next, they wait for the number from the other worker and add this number to their own random number. Finally, both processes output the result of the addition.

Note: Ensure that there are no race conditions!

WP 28

28.1+2 Graphical User Interface – Calculator

Create a very basic calculator with a Graphical User Interface using tkinter. This means, in addition to an element to display the current input and the result, the calculator should only have buttons for digits (0-9), decimal separator, arithmetic operators (+, -, *, /, %, =) and a button to clear previous input.

Note: The logic of the calculator does not have to be perfect. You get one point for the proper arrangement of the individual widgets within the main window and a second point for reacting to the user input, i.e., reacting to the user clicking on buttons.

Project

Instead of writing a final exam, each student must write a project to demonstrate its active participation in the course. The prerequisite for being able to submit a project is the active processing of at least 75% of all exercises. Further regulations:

- A project should be approximately 150 lines of code, but mostly it is about the level of complexity.
- You may work in groups of up to three. When working in groups, a larger scope is expected.
- The topic of the project can be freely chosen, but must be approved by the course instructor before processing. This is mainly to ensure that the scope of your project is neither too small nor too big.
- Accepted file formats for source code are `.py` or `.ipynb` files. If the source code is submitted in any other format, the project is immediately deemed to have failed.
- By default, you will not be given a grade on your project.
- If you want or need a grade, you **must** let me know **in advance**. In case you are not sure whether you need one please consult your examination regulations.
- The source code of your project and all required resources to run your project, e.g., images and sample data, have to be sent in via e-mail using your mail adress of your university to peter.georg@physik.uni-regensburg.de. In case of working in a group it is sufficient to send a single e-mail with all remaining participants listed as recipients or in CC.
- Third-party modules used that were not presented in the lecture must be listed as requirements in a format that pip can recognize, i.e., in `requirements.txt`.
- Registration in **FlexNow before** submitting a project is **manadatory**.
- In some cases a registration in FlexNow is not possible, in this case you'll get a paper certificate. If this is the case you **must** let me know when submitting your project at latest.
- The **deadline** for submitting the project ends on **April 28, 2024 at 23:59**.

Grading Guideline

- If your code is executable and does the announced job for correct user input, you'll get a 4.0.
- Putting everything in a reasonable structure, e.g., by using functions and classes, will improve your grade by one level.
- Introducing safety features, e.g., to catch invalid user input or checking required files exist, will improve your grade by one level.
- Using predefined methods and modules of Python will improve your grade by one level.
- Using third-party modules, whether shown in class or not, will improve your grade by one level.
- Properly commenting your code as required will improve your grade by one level.
- Obviously, the best possible grade is 1.0, hence one does not need to fulfill all points above to get the best possible grade.
- Particular elegant solutions will give you an extra point.
- Particular redundant solutions lead to point deduction.

Project Ideas

- Vocabulary Trainer
- Accounting Software: Manage the account balance of multiple clients in file(s), add and keep track of transactions, graphical representation of the cash flow, …
- Mastermind: A board game.
- Set: A card game.
- Mensch ärgere dich nicht: A board game.
- 2048: A video game.
- Watten: A card game.
- Sudoku-Solver: Solves an unsolved Sudoku riddle provided as input.
- Battleship: You might want to have a look at how to communicate over network.
- Nine Men's Morris: A board game
- Conway's Game of Life
- Data Visualization.
- Scientific Simulations (with visualization).