| *Course:* | **Maschinelles Lernen und künstliche Intelligenz/**  **Machine Learning and Artificial Intelligence** | | |
| --- | --- | --- | --- |
| *(Group) Member:* | *Nr.* | *Name* | *Matrikel* |
| *1* | **Miguel Angel Lopez Mejia** | **5197617** |
| *2* |  |  |
| *3* |  |  |

**Task Reporting Template**

Please read the given Task carefully. You can find all necessary information in Moodle.

## Definition

### Introduction and Explanations

This task aims to familiarize you as students with the basics of Python and VS-Code to fulfill following tasks:

1. Program a first "Hello World" program with Python. Feel free to create an extended version also which combines some strings to a major string that is printed.
2. Program a "dice-algorithm" program with Python. Using required Python packages and confirming the randomness.

The expected result is two working Python applications including documentation and source code. The "Hello World" program is intended to show the functionality of the development environment. The “dice program” is your first program. Use standard packages and functions of Python. The dice-algorithm must prove through mathematical algorithms that the random number generator used works correctly. For example, by a diagram of the statistical distribution.

### Allowed Tools

* The script is going to be executed using ***Python version 3.13.2***
* ***Visual Studio Code*** is going to be used as IDE (Integrated Development Environment)
* Libraries: random, sys, matplotlib.pyplot, logging, numpy, collections, statistics

## Your Preparation/ Concept

### Structure your Problem

##### Task 1.1 - I have to work on:

* My task is to print ‘Hello World from *<my\_name>*’ in the Command line/ Terminal
  + In order to solve the problem, I created a script that works in a virtual environment, once executed, asks the user to introduce a name and by using a class called ***‘HelloWorld‘,*** it concatenates the string ‘Hello World from’ + <user\_name> and displays it in the console.

##### Task 1.2 - I will work on following points for first part:

* + The problem to solve in this task is to create an algorithm that simulates dice by using python.
  + In order to solve the problem, I created a script that works in a virtual environment, once executed, asks the user to introduce the number of dice by using a class called ***DiceGenerator,*** depending on the number of dice, a for loop is used, and with help of the library ‘random’ the script simulates the behavior of a die by generating numbers between 1 and 6. Then the result of each dice is displayed in the console.

##### Task 1.3 - I will work on following points for second part:

* + The problem to solve in this task is to create an algorithm that simulates dice by using python and prove randomness, which, in statistics, implies that the occurrence of one event does not influence the occurrence of another.
  + In order to solve the problem, I created a script that works in a virtual environment, once executed, the script will follow these steps:
* Asks the user to introduce the number of times that the die is going to be rolled by using a class called ***DiceGenerator.***
* Depending on the number of rolls, a for loop is used, and with help of the library ‘random’ the script simulates the behavior of a die by generating numbers between 1 and 6, each number is stored in a list.
* The next step is to count how many times each face of the die appears in the list.
* ***Randomness***, I used the ***Chi-Square*** method to determine if there is a significant difference between the expected frequencies and the observed frequencies. The formula of Chi-square is represented by where:
  + is the observed frequency, in this case, the frequency of each face of the die.
  + is the expected frequency of each face of the die, in this case ⅙.
  + In order to use the Chi-square method, we need to consider:
    - ***Null hypothesis (*** = The die is fair and each face has the same probability of ⅙.
    - ***Critic Value*** = (Considering a significance level , It is the threshold defined to decide when a difference between what is observed and what is expected is *"too large"* to be attributed to chance, 0.05 is the standard - Check Images/3\_Dice\_Chi\_square\_probability\_table.png for reference)
    - To reject , we expect that
* Once randomness is proved, a graphic is display, showing the values and the result of each roll number
* Finally, an histogram is displayed, comparing the die values vs the frequency.

#### Flows:





#### Libraries:

* **random** is a python library that provides functions to generate random numbers, commonly used for simulations, games and applications that require unpredictable behavior.
* **matplotlib** is a comprehensive library for creating static, animated, and interactive visualizations.
* **logging**: this module defines functions and classes which implement a flexible event logging system for applications and libraries.
* **numpy** is a Python library providing a multidimensional array object, derived objects like masked arrays and matrices, and tools for fast array operations, including math, logic, shape manipulation, sorting, I/O, Fourier transforms, linear algebra, statistics, and random simulation.
  + **sys** is a library that allows to interact with the run-time environment of python
  + **collections**: This module implements specialized container data types providing alternatives to Python’s general purpose built-in containers, [dict](https://docs.python.org/3/library/stdtypes.html#dict), [list](https://docs.python.org/3/library/stdtypes.html#list), [set](https://docs.python.org/3/library/stdtypes.html#set), and [tuple](https://docs.python.org/3/library/stdtypes.html#tuple).
  + **statistics**: this module provides functions for calculating mathematical statistics of numeric ([Real](https://docs.python.org/3/library/numbers.html#numbers.Real)-valued) data.

| *# region Libraries*  **import** random **import** matplotlib.pyplot **as** plt **import** logging **import** numpy **as** np  **import** sys **from** collections **import** Counter **from** statistics **import** mean  *# endregion* |
| --- |

### Your Implementation

#### Virtual Environment

* + In order to run the script, a virtual environment must be created, open a new terminal and create it as it is shown:



* + Execute the following command in the console in order to create a new virtual environment: ***python -m venv venv*** once the virtual environment is created, it will be displayed in the explorer section:



* + In order to activate the virtual environment, execute this script in the console: ***venv/scripts/activate*** .



#### Task 1.1 - Hello World

* In order to execute the script, run this command in the console: ***python .\1\_HelloWorld.py.py***, the name of the user will be requested:
* The script will show the name of the user concatenated with ‘Hello World from’:

#### Task 1.2 - Dice

* In order to execute the script, run this command in the console: ***python .\2\_Dice.py*** , the number of dice will be requested:
* The script contains a class called ***DiceGenerator*** within 3 methods:
  + ***get\_dice\_number***: Ask the user to introduce a number of dice
  + ***generate\_random:*** Generates random values (1-6) for a specified number of dice.
  + ***count\_values***: Prints the value of each dice from a dictionary of dice results.
* Once executed, the script will show the results:

#### 

#### 

#### 

#### 

#### Task 1.3 - Dice Ext

* In order to execute the script, run this command in the console: ***python .\3\_Dice\_Ext.py*** . The number of rolls is requested:



* The script contains a class called ***DiceGenerator*** within 6 methods, each method contains the description of they work and examples:
  + ***get\_roll\_number***: Ask the user to introduce a number of rolls
  + ***roll\_die:*** Generates random values (1-6) for a specified number of rolls.
  + ***count\_ocurrences\_per\_face***: Counts the occurrences of each die face in the results and logs the statistics.
  + **test\_randomness**: Uses the ***Chi-Square*** method to determine if there is a significant difference between the expected frequencies and the observed frequencies.
  + **plot\_results**: Generates a bar chart of dice roll results with a mean line.
  + **histogram\_die**: Generates a histogram showing the frequency distribution of dice roll results.
* The output of the script is going to show the following ***(in this case the number of rolls is 10)***:
* The number of time that each face of the die appeared and the percentage
* The conclusion of the test of randomness, by using the Chi-square method, as well as the expected mean vs the observed mean .
* A graphic that shows the dice number and the value that was associated to it, as well as the mean that was obtained.:



* An histogram that shows the face/die value vs the frequency:

#### Analysis

I created 3 more tests for this exercise in order to analyze the data and create a conclusion of the behavior of the script/die.

##### Test - 20 rolls

* Test with 20 rolls:





##### Test - 10,000 rolls:



##### Test - 100,000 rolls:

### 

### 

### 





### 

### Conclusion

I created a table that contains the results obtained for distinct number of rolls:

| **Face** | **Number of rolls** | **Frequency** | **Percentage** | **Mean** | **Chi-square** |
| --- | --- | --- | --- | --- | --- |
| 1 | 10 | 2 | 20 | 3.3 | 3.199 |
| 2 | 10 | 1 | 10 | 3.3 | 3.199 |
| 3 | 10 | 3 | 30 | 3.3 | 3.199 |
| 4 | 10 | 2 | 20 | 3.3 | 3.199 |
| 5 | 10 | 0 | 0 | 3.3 | 3.199 |
| 6 | 10 | 2 | 20 | 3.3 | 3.199 |
| 1 | 20 | 5 | 25 | 3.35 | 3.40 |
| 2 | 20 | 4 | 20 | 3.35 | 3.40 |
| 3 | 20 | 2 | 20 | 3.35 | 3.40 |
| 4 | 20 | 1 | 5 | 3.35 | 3.40 |
| 5 | 20 | 4 | 20 | 3.35 | 3.40 |
| 6 | 20 | 4 | 20 | 3.35 | 3.40 |
| 1 | 10000 | 1721 | 17.21 | 3.4958 | 6.3224 |
| 2 | 10000 | 1585 | 15.85 | 3.4958 | 6.3224 |
| 3 | 10000 | 1685 | 16.85 | 3.4958 | 6.3224 |
| 4 | 10000 | 1685 | 16.85 | 3.4958 | 6.3224 |
| 5 | 10000 | 1672 | 16.72 | 3.4958 | 6.3224 |
| 6 | 10000 | 1652 | 16.52 | 3.4958 | 6.3224 |
| 1 | 100000 | 16795 | 16.79 | 3.49823 | 3.8627 |
| 2 | 100000 | 16655 | 16.66 | 3.49823 | 3.8627 |
| 3 | 100000 | 16470 | 16.47 | 3.49823 | 3.8627 |
| 4 | 100000 | 16718 | 16.72 | 3.49823 | 3.8627 |
| 5 | 100000 | 16737 | 16.74 | 3.49823 | 3.8627 |
| 6 | 100000 | 16626 | 16.63 | 3.49823 | 3.8627 |

Table 1: Results Die algorithm

According to the ***Table 1: Results Die algorithm***, I created this graphic, which allows me to conclude that the *obtained mean* tends to be closer to the *expected mean* *(3.5)* in relation with the number of rolls:

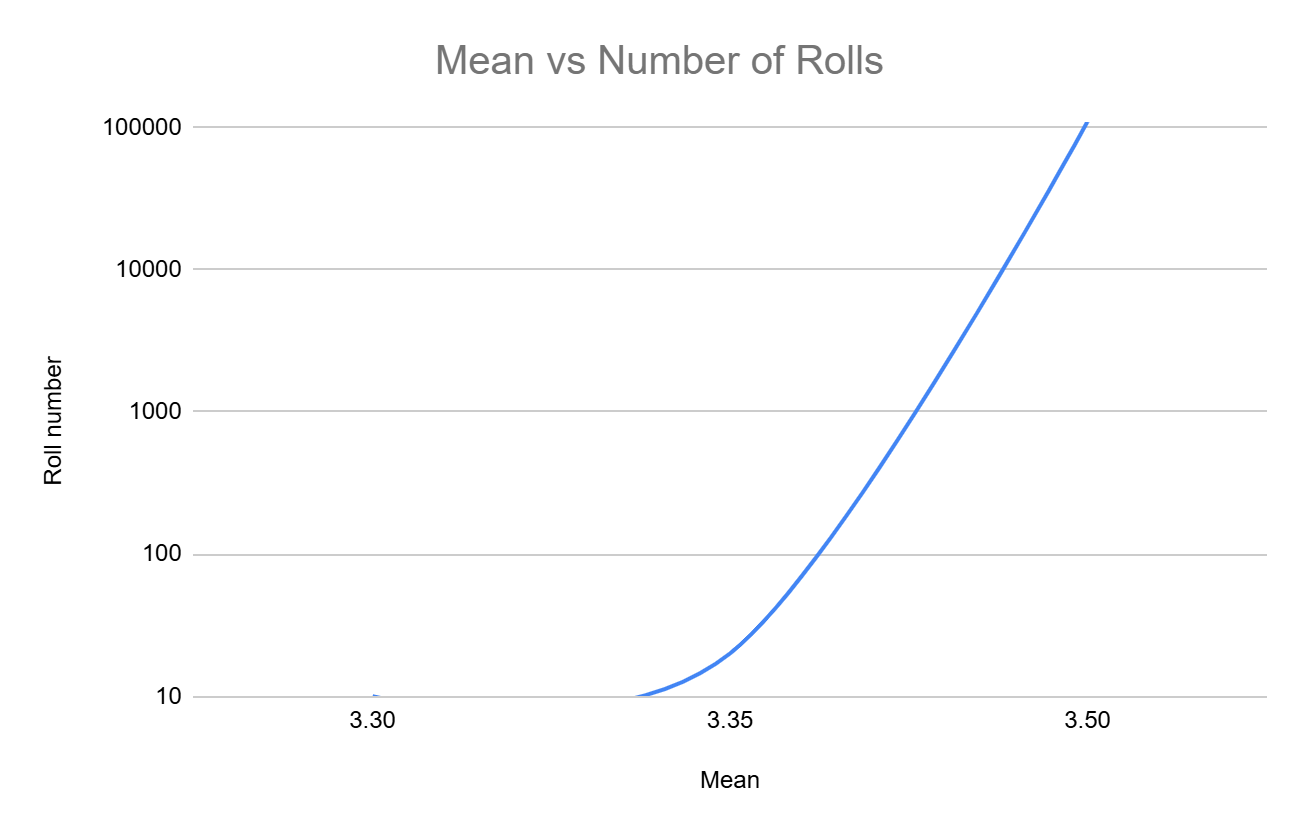


Figure 26: Mean vs Number of rolls

In the ***Table 2: Ranges of results per Number of rolls***, I can appreciate that the difference between the *minimum percentage* and the *maximum percentage* in each roll, tends to decrease when the number of rolls increase. I expect that the percentage of each face of the dice to be and the column ‘**Range (Max - Min)’ shows how the difference between the percentages of each face decreases when the number of rolls increase.**

| **Number of rolls** | **Minimum Percentage** | **Maximum Percentage** | **Range (Max - Min)** |
| --- | --- | --- | --- |
| 10 | 0.0 | 30.0 | 30.0 |
| 20 | 5.0 | 25.0 | 20.0 |
| 10000 | 15.85 | 17.21 | 1.36 |
| 100000 | 16.47 | 16.80 | 0.33 |

Table 2: Ranges of results per Number of rolls

Finally, regarding the Chi-square values obtained, from ***Table 1: Results Die algorithm,*** the results don’t follow the same pattern as the mean and percentage/frequency, this is due to the formula sums square deviations, and even small proportional deviations (e.g., 1721 vs. 1666.67) produce larger contributions when scaled by a larger sample size. I can take the test with 10,000 rolls as an example.

#### For 10,000 rolls:

* **Frequencies**: [1721, 1585, 1685, 1685, 1672, 1652].
* **Expected frequency**:
* **Chi-square calculation**:
  + Face 1:
  + Face 2:
  + Face 3: 0.2017
  + Face 4: 0.2017
  + Face 5: 0.0171
  + Face 6: 0.1291

That is why we can see that in the test with 10,000 rolls, the Chi-square is bigger than the other tests.

## References, extra reading links, and resources

* Corder, G. W., & Foreman, D. I. (2009). *Nonparametric statistics for non-statisticians: A step-by-step approach*. Wiley.
* Python Software Foundation. (n.d.). *random — Generate pseudo-random numbers*. Python 3 Documentation.<https://docs.python.org/3/library/random.html>
* Python Software Foundation. (n.d.). *sys — System-specific parameters and functions*. Python 3 Documentation.<https://docs.python.org/3/library/sys.html>
* PythonBasics.org. (n.d.). *Try Except in Python*.<https://pythonbasics.org/try-except/>
* W3Schools. (n.d.). *Python Classes and Objects*.<https://www.w3schools.com/python/python_classes.asp>