# Ministry of Education of Republic of Moldova Technical University of Moldova

Report

## on Probability and Statistics

Laboratory Work Nr. 1



**Performed by** b. **Cristian Brinza FAF-212**

**Verified by** pr.univ **Cristifor Fiștic**

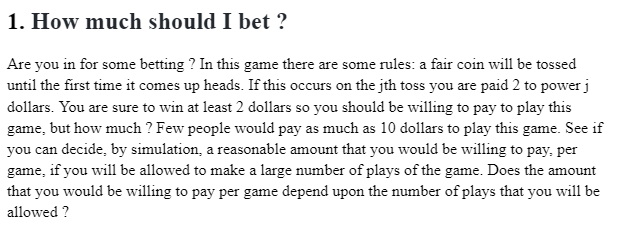
# Chișinău 2021

*Topic:*  Programming discrete math theme in Python programming language.

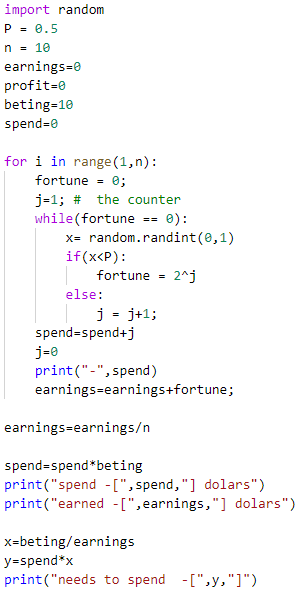
**Purpose of the laboratory work:** Accumulation of practical skills for developing and programming computational processes and program testing skills.

1. **How much shoud i bet?**

* **Condition**



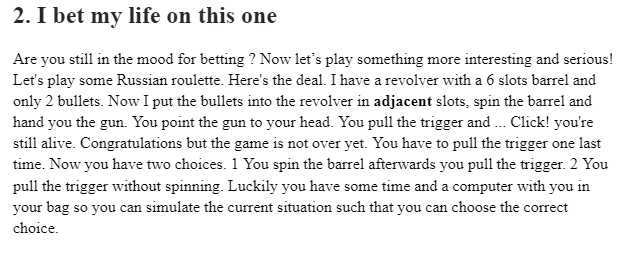
* **Code:**



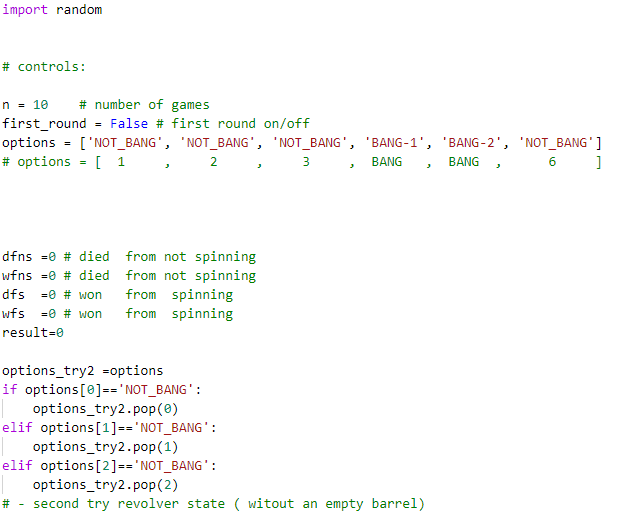
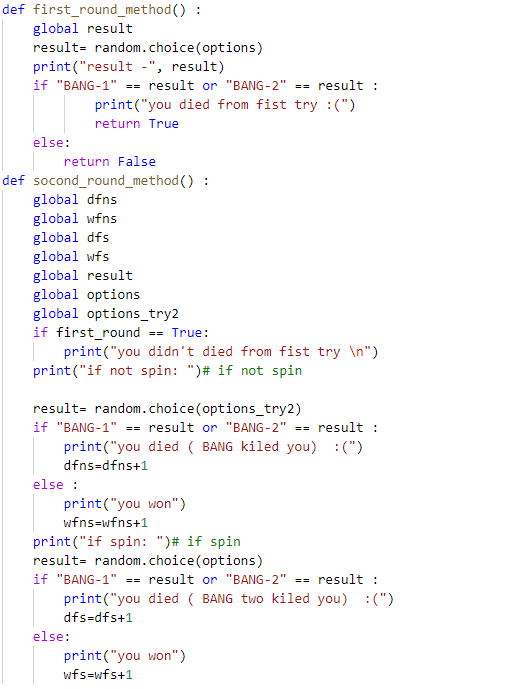
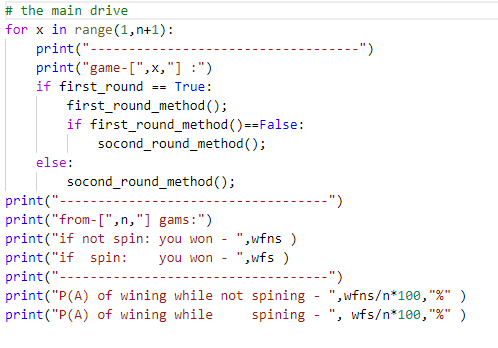
* **Comented code:**
* Running this program several times with n = 100,000,000 – the average return
* varied from $90.1672 to $29.0768.
* A fair price to pay to play the game under the idealized rules will take more
* consideration than just the simulation above.
* It also implies something counter-­‐ intuitive to the problem – that someone should pay
* more than $10 (and potentially any value) to play the game.
* This does not depend upon the number of plays that you will be allowed
* (since a large n can be seen as averaging any lesser number of plays).
* Speaking strictly theoretically (that there can be an infinite number of plays of the
* game, that the person offering the game has infinite wealth and that the bettor
* has infinite wealth) it follows that the expected return of the game is given as:
* (photo)
* This implies that there isn’t necessarily an answer to how much someone might pay.
* This situation, realistically, is immaterial since there is not infinite wealth
* (and not enough time to play the game that many times). Thus, this requires
* constraining the game.
* Assuming that the person offering the game is only willing to pay out a certain amount
* on a single game, the game has a finite Expected Value which depends on that number and
* it could be calculated or simulated. Tweaking the program above, if the max is $1024,
* the fair value is somewhere around $11 and if the max is $2048, then the fair value is
* somewhere around $12 (so a prudent investor would pay less than that).

1. **I bet my life on this one**

* **Condition**



* **Code:**

* **Comented code:**

Assume that the barrel rotates clockwise after the hammer hits and is pulled back. You

are given the choice between an unconditional and a conditional probability of death. Select a chamber C at random from the 6 choices.

If we randomise the gun we have a 2 in 6 (or 1 in 3 ) chance of dying. The possible set ups at the start of the game are as follows:

1 BB----

2 -BB---

3 --BB--

4 ---BB-

5 ----BB

6 B----B

As in the first round we didn't die we know it was not scenarios 1 and 6, eliminating those rows, and the first column as that chamber

is no longer in the game - we're left with this:

2 BB---

3 -BB--

4 --BB-

5 ---BB

Giving us a 1 in 4 chance of dying. By re spinning the gun it would have been 1 in 3. So in the case of 2 consecutive bullets it would

not be in your interests to re-spin the chamber.

Assume that the barrel rotates clockwise after the hammer hits and is pulled back. You

are given the choice between an unconditional and a conditional probability of death. Select a

chamber C at random from the 6 choices.

In the case the barrel is spun again, the probability of death is the (unconditional) probability

that C contains a bullet, which is 2/6 = 1/3.

If the trigger is pulled without the extra spin, the probability is P(A|B), where A is the

event that C contains a bullet and B is the event that the slot next to C in the counterclockwise

direction is empty (as this is the slot that was clicked on the previous time). Now P(B) = 4/6

and P(A ∩ B) = 1/6 (only one chamber contains a bullet and has an empty chamber next to it

in the counterclockwise direction), so P(A|B) = 1/4.

The above is written with absolute mathematical precision. More informally, you may argue,

in the second case, that only one of the four empty slots will lead to your death if the trigger is

pulled again. Note also that if the two bullets are not next to each other, two empty slots will

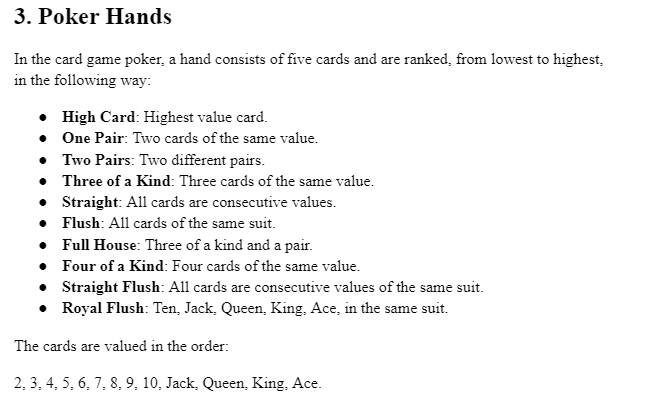
result in your death if the trigger is pulled again.

* **Output:**

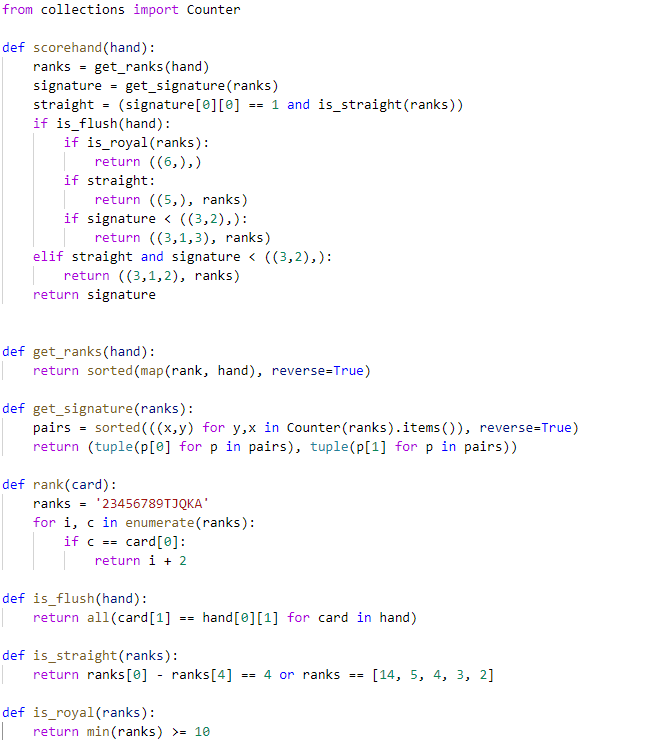
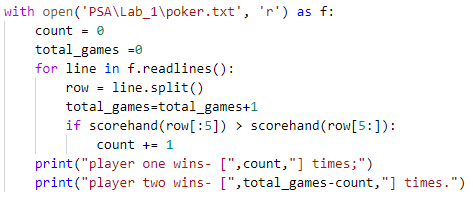
So in the case of 2 consecutive bullets it would not be in your interests to re-spin the chamber.

1. **Poker Hands**

* **Condition**



* **Code:**

* **Comented code:**

# Poker hands are ranked as follows:

# high card < 1 pair < 2 pair < 3 of a kind < straight < flush

#           < full house < 4 of a kind < straight flush < royal flush

# scorehand(hand) Returns the signature of a hand (see below)

# modified to account for flushes and straights.

#

# straight is scored as ((3,1,2), (a,b,c,d,e)) to place it between

# 3 of a kind and full house.

#

# flush is scored as ((3,1,3), (a,b,c,d,e)) to place it between

# straight and full house.

#

# straight flush is scored as ((5,), (a,b,c,d,e)) to place it

# above 4 of a kind ((4,1), (a,b)).

#

# royal flush is scored as ((6,),) to place it above straight flush.

# Get the ranks of the cards and sort them in reverse order.

def get\_ranks(hand):

    return sorted(map(rank, hand), reverse=True)

# The signature is an ordered pair of tuples describing the number of cards

# of each rank. The first tuple contains the frequencies, in descending order.

# The second tuple contains the corresponding ranks.

#

# 3 of a kind has a signature of the form ((3,1,1), (a,b,c))

# full house has a signature of the form ((3,2), (a,b))

# four of a kind has a signature of the form ((4,1), (a,b))

#

# Two poker hands can be compared by comparing their signatures, using the standard

# lexicographic ordering, unless one or both hands is a straight or a flush.

#

def get\_signature(ranks):

    pairs = sorted(((x,y) for y,x in Counter(ranks).items()), reverse=True)

    return (tuple(p[0] for p in pairs), tuple(p[1] for p in pairs))

# Returns the rank of a card as an integer

# (T = 10, J = 11, Q = 12, K = 13, A = 14)

def rank(card):

    ranks = '23456789TJQKA'

    for i, c in enumerate(ranks):

        if c == card[0]:

            return i + 2

# Returns True if the hand is a flush, False otherwise.

def is\_flush(hand):

    return all(card[1] == hand[0][1] for card in hand)

# Returns True if the hand is a straight, False otherwise.

# A precondition is that the hand does not contain a pair.

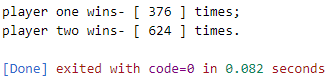
def is\_straight(ranks):

    return ranks[0] - ranks[4] == 4 or ranks == [14, 5, 4, 3, 2]

# Returns True if all cards are Royal (10, J, Q, K, A), False otherwise.

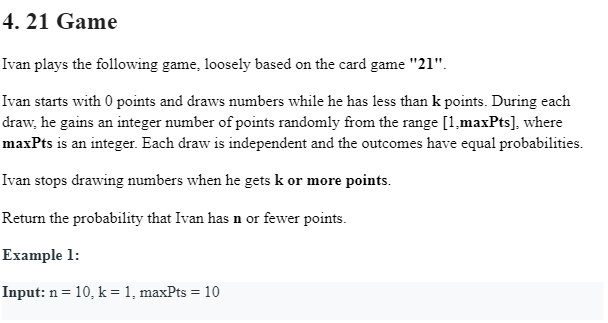
def is\_royal(ranks):

    return min(ranks) >= 10

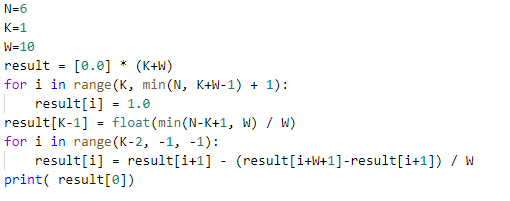
* **Output:** 

1. **21 Game**

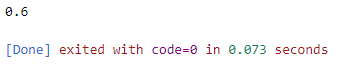
* **Condition**



* **Code:**

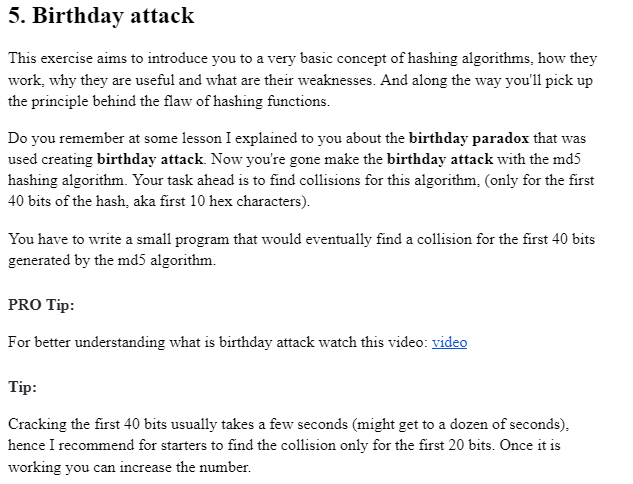


* **Comented code:**
* N: Here is equivalent to a boundary, the requirements are required to finally extract numbers and the comparison with N
* K: Here, a condition that can continue to extract numbers
* W: Number face value, that is, the number extracted, is within 1 to W
* Now look at the example 1:
* N = 10, K = 1, W = 10
* Because Alice is open in 0, 0 <k = 1, then she can now continue to draw a number, and the digital face value is in [1, 10]. Requires the probability of extracting numbers and less than n (n = 10)? It can be seen more obvious here, the probability is 1. Because the face value is in [1, 10], whether the number is extracted, it will be greater than K, which is unattracted, and the end and the result will fall between [1, 10], and the results are certainly less than equal to N. Therefore, the probability is 1.
* Look at Example 2:
* N = 6, K = 1, W = 10
* This is the n value here. The amount of extracted is also as shown in the range of the face value [1, 10], and no matter which number is drawn, it cannot be extracted again, because and is greater than K, the ultimate and the same fall between [1, 10] . However, the N value here has changed to 6, where the final number is extracted and only in the [1, 6], which does not exceed N, which accounts for 60% of the total part, so the probability is 0.6.
* In addition, 3, it is more complicated. It is also a case in which we mainly need to analyze.
* We can see that in the foregoing examples 1, 2, Alice starts extracting numbers in 0, then accumulates with the extracted numbers, then compares to K, determine if it can be extracted again, and whether it does not exceed N?
* In fact, it can be seen here that Alice can win the probability that Alice is actually related to the score before the beginning of the beginning.
* Presentresult[i] So the probability of extracting the number can win with the fraction of I, then the final requirement isresult[0] the result of.
* Now require state transfer equations, first look at some of the conditions given in the topic.
* When the score exceeds K, stop the number of words at this time is set, and when the score does not exceed N, it is determined to be victory, otherwise it will fail.
* Now, first look at the maximum score that can be extracted in the last time. Because more than K, it is not possible to perform it again. So want to take again, the maximum score allowed at this time is K - 1. So re-extracting the maximum number is W, so the maximum score is K - 1 + W.
* That is, when $ k \ leq i \ leq min (n, k + w-1) is $, this timeresult[i]=1, While $ I> MIN (N, K + W-1),result[i]=0。
* So now what is the value of result [i] when $ 0 \ LEQ i <k, how should it be?
* In fact, when $ 0 \ LEQ i <k, when the extraction is performed, the probability at this time is whether the cumulative score after the number is taken over. The previous topic said that the probability of extracting between [1, W] numbers is equal. Then the state transfer equation is as follows:
* $$
* result[i] = \frac{result[i+1]+result[i+2]+...+result[i+W]}{W}
* $$
* Although the state transfer equation has already begun, it will find that the transfer equation will be taken in the code. Optimization below:
* Here, what is the result [i + 1]? According to the front state transfer equation:
* $$
* result[i+1] = \frac{result[i+2]+result[i+3]+...+result[i+W+1]}{W}
* $$
* But this time is $ 0 \ LEQ i <k-1 $.
* It can be seen that in fact, result [i] has a large part in the middle of result [i + 1], then the value of result [i] can be obtained by result [i + 1]:
* $$
* result[i]-result[i+1] =\frac{result[i+1]-result[i+W+1]}{W}
* $$
* at this time:
* $$
* result[i] = result[i+1] - \frac{result[i+W+1]- result[i+1]}{W}
* $$
* Here, $ I = k - 1 $, does not apply to the above formula, then put it in the first transfer equation:
* $$
* result[K-1] = \frac{result[K]+result[K+1]+result[K+W-1]}{W}
* $$
* We have a condition for $ result [i] = 1 $, ie when i's value ranges from $ [k, min (n, k + w-1)].
* At this time, i is a number of possible wins in K-1, and the result is falling on $ min (n, k + w-1) - k + 1 off, then the result is:
* $$
* result[K-1]=\frac{min(N, K+W-1)-K+1}{W}=\frac{min(N-K+1, W)}{W}
* $$
* Other values ​​have a new transfer equation for obtaining.
* **Output:**

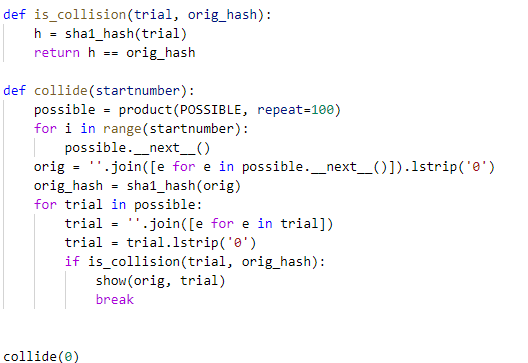


1. **Birthday attack**

* **Condition**



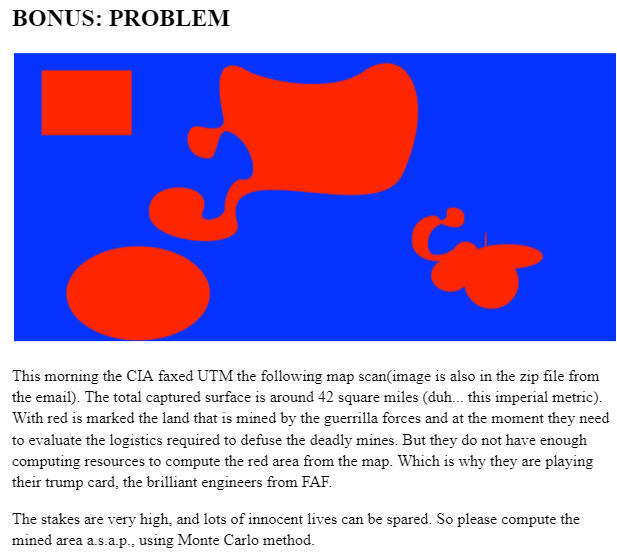
* **Code:**

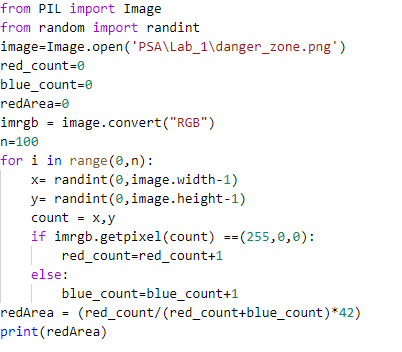
* **Comented code:**
* POSSIBLE = '0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ'
* # Number of characters to consider in the hash function, starting at the
* # beginning. e.g. 6 means we use the first 6 bytes
* HASHCHARS = 6
* def sha1\_hash(s):
* ''' Returns the shortened sha1 hash of s. If the input is bytes, they
* will be hashed directly; otherwise they will be encoded to ascii
* before being hashed.
* '''
* if type(s) is bytes:
* # Already encoded, just hash the bytes.
* return sha1(s).hexdigest()[:HASHCHARS]
* else:
* # Convert it to ascii, then hash.
* return sha1(s.encode('ascii')).hexdigest()[:HASHCHARS]
* def show(orig\_str, collision\_str):
* ''' Print the original string, the collision string, and then recompute
* the hashes of each of them and print those, to prove that we found
* a collision.
* '''
* # Do the encoding to ascii for bytes output
* orig\_ascii = orig\_str.encode('ascii')
* collision\_ascii = collision\_str.encode('ascii')
* # Print stuff.
* print('Collision found!')
* print(orig\_str
* + ' (bytes: ' + str(hexlify(orig\_ascii)) + ')'
* + ' hashes to ' + str(sha1\_hash(orig\_ascii))
* + ', but ' + collision\_str
* + ' (bytes: ' + str(hexlify(collision\_ascii)) + ')'
* + ' also hashes to ' + str(sha1\_hash(collision\_ascii)))
* def is\_collision(trial, orig\_hash):
* ''' Returns true if the hash of trial is the same as orig\_hash.
* '''
* h = sha1\_hash(trial)
* return h == orig\_hash
* def collide(startnumber):
* ''' Search for collisions in the hash. Start with the possible match
* at index startnumber and look for collisions by searching upward
* from there.
* Note that this means if you choose a large value (e.g. 400000) this
* will not look for collisions on possibilities 0 <= x <= 400001, so
* choose a low number unless you want this to run for quite a while.
* '''
* # Iterator that yields possible characters.
* possible = product(POSSIBLE, repeat=100)
* # Iterate over the product until we reach the specified startnumber
* for i in range(startnumber):
* possible.\_\_next\_\_()
* # This is our collision target
* orig = ''.join([e for e in possible.\_\_next\_\_()]).lstrip('0')
* orig\_hash = sha1\_hash(orig)
* # Iterate over the possible options
* for trial in possible:
* # Convert the tuple from itertools.product into a string
* trial = ''.join([e for e in trial])
* # Strip the leading zeros (who cares about zeros!)
* trial = trial.lstrip('0')
* # Exit if we found a collision
* if is\_collision(trial, orig\_hash):
* show(orig, trial)
* break
* if \_\_name\_\_ == '\_\_main\_\_':
* if len(argv) > 1:
* n = int(argv[1])
* if len(argv) > 2:
* HASHCHARS=int(argv[2])
* collide(n)
* else:
* collide(1)

1. **Bonus**

* **Condition**



* **Code:**



* **Output:**



***Conclusion:***

It’s very interesting to see what you are creating by writing code. It’s like art based on code. Skills were developed to compile, run and test a simple program in the Python programming language.

As a result of the elaboration of the given paper, the basis was applied for the practical application of the theoretical knowledge.

The structures/concepts/algorithms used in this problem, after writing, compiling the program ,, several times.

Thus one can judge about the wide possibilities offered by the Python language regarding data manipulation. In this practical work I realized the knowledge accumulated during the theoretical and practical classes, I consolidated the material and in some places I learned new things. It allowed the assessment of knowledge in writing style both for the grade and personally. It allowed us to correct mistakes and possible future misunderstandings. Under the guidance of the teacher, we conducted the first individual study on this subject, this facilitating the adaptation to the knowledge of the use of theoretical material.

The verification of the results confirms that the elaborated program works correctly.

Linear algorithms can be used to calculate mathematical expressions.

Where drawn conclusions about Python programming language. As in the end I can say that the study had a positive impact on my personal education.

## **Bibliography:**

1. <https://www.geeksforgeeks.org/python-classes-and-objects/>