1. PREFACE

This file contains exercises for *Python programming*. Our main goal is to give tasks to the students for their own practicing.

These tasks are not only that ones that can be in a typical exam of a subject of this topic, but they also try to cover a bit wider range of problems.

The chapters are organized by data types and the color of the symbol shows the level of the difficulties of the exercises (blue: easy, red: middle, black: difficult, as the color of the ski slopes ☺).

For the easier understanding the tasks (except some, for example in the last two chapters due to their complexity) are given with examples. For the simplicity the examples use mainly integer numbers (for example instead of float numbers and in cases when comparable data are needed) and the float numbers are given with (only) one decimal digit.

We hope that every student can find suitable exercises for her/his knowledge, and she/he will be able to solve them improving her/his problem-solving skills.

We wish you good thinking and a lot of good solutions.

Pusztai Pál

2. SIMPLE DATA TYPES

This chapter contains exercises that can be solved with using simple data types. If we have more (for example *n*) data to process, we do not need to store them in a variable of a compound data type (for example in a *list*). We can process all data with an iteration that works step by step, it reads a data to a variable of a simple data type then it processes that data. Therefore, these exercises ask to read the data as well. Of course, these tasks can be solved with using compound data types too.

2.1. int

- Calculate the time between two times of a day. Both times are given with three integer numbers, namely hours, minutes and seconds. Give the result in the same form.

  Remark: The solution can be done with *datetime* module but solve this task without it.

  Example: 12, 15, 58; 12, 17, 1 → 0, 1, 3
- For given *n* and *k* positive integer numbers determine the value of the binomial coefficient.

  Example: *n*=5, *k*=2 → 10
- For a given *n* positive integer print the first *n* elements of the *Fibonacci numbers*. The way of calculating: the first two elements are 1, and the other elements are the sum of the previous two elements.

  Example: *n*=6 → 1, 1, 2, 3, 5, 8.
- For a given positive integer number decide that it is prime number or not. A prime number can only be divided by 1 and itself.

  Examples: 7 → prime

  12 → not prime
- With the solution of the previous task print the prime numbers in a given [*a*, *b*] interval.

  Example: [1, 10] → 2, 3, 5, 7
- Print the prime factorization of a given integer number that is greater than 1. Use only the multiplication operator between the prime factors.

  Example: 12 → 2*2*3
- Expand the previous exercise with using the power operator if a prime factor divides the number more times.

  Examples: 12 → 2**2*3

  1024 → 2**10
- Determine the greatest common divisor of two positive integer numbers.

  Example: 12, 8 → 4
- For two given positive integer numbers decide that they are relatively prime numbers or not. The relatively prime numbers have only one common divisor, namely 1.

  Examples: 15, 16 → relatively primes

  12, 8 → not relatively primes
- With the solution of the previous task print all relatively prime numbers in a given [*a*, *b*] interval. Print each relatively prime pair only once.

  Example: [8, 11] → (8, 9); (8, 11); (9, 10); (9, 11); (10, 11)

2.2. float

- There are given n numbers as input data where the value of n is also an input data. Read the data, determine and print the followings.
  - The ratio of the positive numbers given in percentage.
    Example: n=4; 1, −1, 2, 0 → 50.0
  - The numbers are in ascending order or not.
    Examples: n=4; 1, 3, 3, 7 → ascending
    n=3; 1, 3, 2 → not ascending
  - The numbers give an *arithmetic sequence* (in the given order) or not. In an arithmetic sequence the difference between the consecutive terms is the same.
    Examples: n=4; 1, 3, 5, 7 → arithmetic sequence
    n=3; 1, 2, 2 → not arithmetic sequence
- There are given $n \geq 3$ numbers. Read the data and determine the mean of the data such that one minimum value and one maximum value are not included in the calculation.
  Examples: n=3; 1, 3, 5 → 3.0
  n=4; 1, 3, 1, 5 → 2.0
- There are given n points on a plane with their (x, y) *Cartesian coordinates* as input data where the value of n is also an input data. Read the data, determine and print the followings.
  - The center of gravity. (Hint: This point has coordinates that are the means of x coordinates and means of y coordinates of the points.)
    Example: n=4; (0, 0), (1, 0), (1, 1), (0, 1) → (0.5, 0.5)
  - Let us consider the points as shots to a target. The center of this target is in the origin, and it contains 10 circles with 1, 2, …, 10 radiuses. The value of a shot can be calculated from the distance between the point and the origin. If this distance is in the interval [0, 1] than the value is 10, if it is in the interval (1, 2] than the value is 9, and so on, if it is in the interval (9, 10] than the value is 1, otherwise the value is 0. Determine the sum of the value of the shots.
    Example: n=4; (0, 0), (1, 0), (10, 0), (10, 10) → 21
  - The farthest point from the origin. If there are more such points let the first one (in the input sequence) be the result.
    Example: n=3; (1, 0), (1, 1), (−1, 1) → (1, 1)
- Solve the $ax+by=c$ and $dx+ey=f$ linear equations where the values of a, b, c, d, e, f are float numbers. The equations may have 0, 1, or infinite such solution.
  Examples: $x-y=1$ and $x=0$ → the solution: (0, −1)
  $x+y=1$ and $x+y=2$ → there is no solution
  $x+y=1$ and $2x+2y=2$ → there are infinite solutions
- There are given some (at least one) input numbers. We do not know the number of the input data. The user will type the numbers as input values and finally she/he will give 0 that is not belonged to the input data, it only signs that there is no more input number. Read the input, determine and print the square mean of the data. The square mean is the square root of the sum of the square of the data divided by the number of the data.
  Example: 3, 5, 0 → $\frac{\sqrt{3^2+5^2}}{2}$
- There are given n input numbers where the value of n is also an input data. Read the data and print the number of the elements of the longest ascending subsequence.
  Example: n=5; 2, 1, 3, 2, 1 → 2

2.3. str

- Construct a string that contains a given character and has a given length.

  Example: '*', 3 → "***"
- Make a new string from a given string such that the result contains the same characters but in reverse order.

  Example: "Abracadabra" → "arbadacarbA"
- There is given a string that contains a name. The name has exactly two parts, the first name and the last name separated with exactly one space. Create a new name with changing the order of the two parts of the name.

  Example: "Little John" → "John Little"
- Two given strings are *similar* if they have the same length, and their characters are the same except exactly one position in that they have different character. Decide for two given strings that they are similar or not.

  Examples: "Little John", "Little john" → similar

  "Little John", "little john" → not similar
- Determine the number of the integer numbers in a given interval that satisfy the following criteria. In the examples the proper integer numbers are also given (in parentheses) for the easier understanding.
  - The digits of the number are all different.

    Example: [115, 125] → 4 (120, 123, 124, 125)
  - The number contains 5 as a digit.

    Example: [45, 55] → 7 (45, 50, 51, 52, 53, 54, 55)
  - The sum of the digits of the number can be divided by 5.

    Example: [123, 140] → 4 (127, 131, 136, 140)
- Classify a given string that is longer than one character with the followings. The string is *ascending* if its characters are in ascending order, it is *descending* if its characters are in descending order, it is *constant* if its characters are the same, and it is *other* otherwise.

  Examples: "abbc" → ascending

  "YXX" → descending

  "aaaaa" → constant

  "Abba" → other
- Construct a new string from a given string such that the result string has no leading and trailing spaces, and it contains only one space next to each other inside the string.

  Example: " Little John and Robin " → "Little John and Robin"
- Construct a new string from a given string such that the result string has no leading and trailing spaces, and it contains only one space next to each other inside the string.

  Example: " Little John and Robin " → "Little John and Robin"
- There is given a string that satisfies the properties of the result string in the previous task. Construct a new string such that the result string contains the same words but in reverse order.

  Remark: The solution can be done with *str.split* method or without it. In the first case the solution is simpler because of using a string list. Our given difficulty level (marked with black color) corresponds to the second case.

  Example: "Little John and Robin" → "Robin and John Little"

- Code a given string with a given code table. If a character is not included in the code table, then it should be the same. Make the inverse of this coding that decodes a string that was previously coded.

  Example: string: ”apple cramble”
  code table: a-b, b-c, c-d
  result: ”bpple drbmcle”

- Two strings are *similar* if their lengths differ from each other exactly by one and the longer string has a character that if it is omitted then the shorter string can be obtained. Decide for two given strings that they are similar or not.

  Examples: ”Little John”, ”Little Johny” → similar
  ”Little John”, ”Little John” → not similar

- A given string contains a positive integer given in the hexadecimal (16) number system. Make a new string that contains the equivalent number given in the binary (2) number system. Solve the conversion from the binary number system to the hexadecimal number system as well.

  Remark: If you use the proper Python functions (for example: int, bin, hex) then it is an easy exercise, otherwise (if you do the conversion by yourself) it is a middle task.

  Examples: ”B18” → ”101100011000”
  ”101100011000” → ”B18”

- Generate a random string with a given length $n \leq 26$ such that it contains only different English uppercase letters.

  Example: n=5 → ”GWEZT”

- Code a given string with the given key with reversing the key length parts of the string. Let the partition start at the beginning of the string and if there is a shorter part at the end of the string than the key, let it remain the same. Make the inverse of this coding that decodes a string that was previously coded.

  Examples: ”Little John”, 3 → ”tiLeltoJ hn”
  ”Little John”, 1 → ”Little John”
  ”Little John”, 20 → ”Little John”

3. COMPOUND DATA TYPES

This chapter contains exercises that can be solved with using compound data types.

3.1. tuple

- There is given a point on a plane with its (x, y) *Cartesian coordinates*. Calculate the *polar coordinates* of this point such that the angle is given in degree (in interval [0, 360]) instead of radian. Make this conversion in the opposite direction as well.

  Examples: (1.0, 1.0) → (1.4, 45.0)

  (1.0, 90.0) → (0.0, 1.0)

- There is a given sequence of elements that are comparable.
  - Count the number of the ascending subsequences. A subsequence begins at the first element and at any element that is smaller than the element before it.

    Example: (1, 2, 3, 2, 1, 2, 2, 3, 4) → 3
  - Determine the longest ascending subsequence. If there are more such subsequences then let the result be the first one.

    Example: (3, 2, 4, 5, 1, 3, 4) → (2, 4, 5)
- There is given a positive integer as money in euro that we want to pay. Let us pay it out with using as few banknotes as possible. The banknotes of euro are: 500, 200, 100, 50, 20, 10, 5, 2, 1.

  Example: 4409 → 500: 8, 200: 2, 5: 1, 2: 2
- There are given $n \geq 2$ points on a plane with their *Cartesian coordinates*.
  - Determine a point that produces the smallest sum distance from the other points and determine this minimum sum too.

    Example: n=5; (0, 0), (−1, 0), (0, −1), (1, 0), (0, 1) → (0, 0); 4.0
  - Determine the distance between the farthest two points.

    Examples: n=3; (0, 0), (1, 0), (0, 1) → 1.4

    n=4; (−1, 0), (0, −1), (1, 0), (0, 1) → 2.0
  - Determine the pair of points whose distance is the greatest. If there are more such pairs then give all different pairs.

    Example: n=4; (−1, 0), (0, −1), (1, 0), (0, 1) → (−1, 0), (1, 0); (0, −1), (0, 1)
- There are given n points on a plane with their *Cartesian coordinates*. The points are the vertices of a convex polygon according to a circumstance.
  - Determine the center of gravity of the polygon.

    Example: n=4; (0, 0), (1, 0), (1, 1), (0, 1) → (0.5, 0.5)
  - Determine the perimeter of the polygon.

    Example: n=4; (0, 0), (2, 0), (2, 2), (0, 2) → 8.0
  - Determine the length of the longest side of the polygon.

    Example: n=3; (0, 0), (1, 0), (0, 1) → 1.4
  - Determine the length of the shortest diagonal of the polygon.

    Example: n=5; (0, 0), (2, 0), (2, 2), (1, 3), (0, 2) → 2.0
  - Determine the area of the polygon. (Hint: Divide the polygon to n−2 triangles, use Heron's formula to calculate the area of one triangle and sum these areas.)

    Example: n=5; (0, 0), (2, 0), (2, 2), (1, 3), (0, 2) → 5.0

- A square matrix (where the number of the rows equals to the number of the columns) is *symmetric* if the matrix equals to its transpose. Formally $A = [a_{ij}]_{nxn}$ is symmetric if $a_{ij} = a_{ji}$ for all indices i and j. Decide that a square matrix is symmetric or not.

  Examples: n=3, $\begin{bmatrix} 2 & 7 & 6 \\ 7 & 5 & 1 \\ 6 & 1 & 8 \end{bmatrix}$ → symmetric

  n=3, $\begin{bmatrix} 2 & 9 & 4 \\ 7 & 5 & 3 \\ 6 & 1 & 8 \end{bmatrix}$ → not symmetric

- An $n \times n$ matrix is a *magic square* if it contains the 1, 2, …, n^2 integer numbers and the sums of the numbers in each row, each column, and both main diagonals are the same. Decide that a square matrix is magic square or not.

  Examples: n=3, $\begin{bmatrix} 2 & 9 & 4 \\ 7 & 5 & 3 \\ 6 & 1 & 8 \end{bmatrix}$ → magic square

  n=3, $\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$ → not magic square

- An element of a matrix is a *saddle point* if it is the minimum value of its row and it is the maximum value of its column. Determine the all saddle points in a given matrix. Solve this problem with unique and not unique minimum and maximum values.

  Examples: n=3, m=4, unique case, $A = \begin{bmatrix} 1 & 1 & 1 & 2 \\ 3 & 2 & 3 & 4 \\ 5 & 0 & 4 & 1 \end{bmatrix}$ → $a_{22} = 2$

  n=3, m=2, unique case, $A = \begin{bmatrix} 1 & 1 \\ 3 & 1 \\ 5 & 0 \end{bmatrix}$ → no saddle point

  n=3, m=3, not unique case, $A = \begin{bmatrix} 3 & 1 & 0 \\ 2 & 2 & 2 \\ 3 & 4 & 3 \end{bmatrix}$ → $a_{31} = 3,\ a_{33} = 3$

3.2. list

- There is given an ascending data sequence and a given data. Insert the data into the sequence for the proper place keeping the sorting.

  Example: [1, 2, 4, 5], 3 → [1, 2, 3, 4, 5]

- There are given two ascending data sequences.
  - Merge the elements to a new sequence that is also ascending.

    Example: [2, 4, 5], [1, 3] → [1, 2, 3, 4, 5]
  - Merge the elements to a new ascending sequence such that the elements of the result are all different.

    Example: [2, 4, 5], [1, 1, 2, 3] → [1, 2, 3, 4, 5]

- Generate the prime numbers that are not greater than a given $n \geq 2$ with using *Sieve of Eratosthenes*. This is an old and nice algorithm that does not use division. It can be given with four steps. Step 1: Write down (into a list) all integer numbers from 2 to n and let them all be unmarked. Step 2: Select the smallest number from this list that is unmarked (that is 2 at first). The selected number is a prime number. Step 3: Mark this number and delete all multiples of it from the list (that are 4, 6, … at first). Step 4: If there is unmarked number in the list then continue with step 2, otherwise the algorithm is finished, the list contain only marked numbers that are the wanted prime numbers. Implement this algorithm such that the list is realized with a Python list.
- A card game uses 52 cards. For identifying the cards we use integer numbers from 1 to 52. Make a random hand for n players such that each player gets m cards (where $n*m \leq 52$). Let the result be a matrix sized $n \times m$. Of course a card can be given to at most one player.

  Example: n=3, m=2 → $\begin{bmatrix} 39 & 51 \\ 1 & 14 \\ 48 & 8 \end{bmatrix}$

- There is given an $n \times m$ matrix. Solve the following exercises.
  - Change the elements of two columns that are given with their indices.

    Example: n=3, m=4, i=1, j=4, $\begin{bmatrix} 2 & 2 & 5 & 3 \\ 1 & 2 & 1 & 1 \\ 0 & 0 & 2 & 4 \end{bmatrix} \rightarrow \begin{bmatrix} 3 & 2 & 5 & 2 \\ 1 & 2 & 1 & 1 \\ 4 & 0 & 2 & 0 \end{bmatrix}$
  - Subtract the minimum values of the rows from the elements of the proper rows.

    Example: n=3, m=4, $\begin{bmatrix} 2 & 2 & 5 & 3 \\ 1 & 2 & 1 & 1 \\ 0 & 0 & 2 & 4 \end{bmatrix} \rightarrow \begin{bmatrix} 0 & 0 & 3 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 2 & 4 \end{bmatrix}$
  - Delete the columns where the sum of the elements is 0.

    Example: n=3, m=4, $\begin{bmatrix} 3 & 2 & 0 & 1 \\ 1 & -1 & 0 & 3 \\ 0 & -1 & 0 & 0 \end{bmatrix} \rightarrow n$=3, m=2, $\begin{bmatrix} 3 & 1 \\ 1 & 3 \\ 0 & 0 \end{bmatrix}$
  - Delete the rows that contain only 0 elements.

    Example: n=3, m=2, $\begin{bmatrix} 3 & 0 \\ 1 & 2 \\ 0 & 0 \end{bmatrix} \rightarrow n$=2, m=2, $\begin{bmatrix} 3 & 0 \\ 1 & 2 \end{bmatrix}$
  - Insert a new row as the first row that contains the sums of each columns of the original matrix.

    Example: n=2, m=3, $\begin{bmatrix} 1 & 2 & 3 \\ 4 & 2 & 0 \end{bmatrix} \rightarrow n$=3, m=3, $\begin{bmatrix} 5 & 4 & 3 \\ 1 & 2 & 3 \\ 4 & 2 & 0 \end{bmatrix}$
  - Insert a new column as the last column that contains the minimum values of each rows of the original matrix.

    Example: n=3, m=3, $\begin{bmatrix} 3 & 2 & 5 \\ 1 & 2 & 1 \\ 4 & 0 & 2 \end{bmatrix} \rightarrow n$=3, m=4, $\begin{bmatrix} 3 & 2 & 5 & 2 \\ 1 & 2 & 1 & 1 \\ 4 & 0 & 2 & 0 \end{bmatrix}$

- Sort the rows of the matrix by ascending order of the sums of the rows.

  Example: n=3, m=4, $\begin{bmatrix}3&2&5&2\\1&2&1&1\\4&0&2&0\end{bmatrix} \rightarrow \begin{bmatrix}1&2&1&1\\4&0&2&0\\3&2&5&2\end{bmatrix}$

- There is given an $n \times m$ character matrix that contains English lowercase letters.
  - Sort the rows of the matrix such that the strings that can be obtained by concatenating the characters of the rows are in ascending order of alphabet.

    Example: n=4, m=5, $\begin{bmatrix}p&e&a&c&h\\a&p&p&l&e\\m&a&n&g&o\\l&e&m&o&n\end{bmatrix} \rightarrow \begin{bmatrix}a&p&p&l&e\\l&e&m&o&n\\m&a&n&g&o\\p&e&a&c&h\end{bmatrix}$

  - Sort the columns of the matrix such that the characters of a given (ith) row are in ascending order of alphabet.

    Example: n=4, m=5, i=2, $\begin{bmatrix}p&e&a&c&h\\a&p&p&l&e\\m&a&n&g&o\\l&e&m&o&n\end{bmatrix} \rightarrow \begin{bmatrix}p&h&c&e&a\\a&e&l&p&p\\m&o&g&a&n\\l&n&o&e&m\end{bmatrix}$

- An $n \times n$ matrix is a *magic square* if it contains the 1, 2, …, n^2 integers and the sums of the numbers in each row, each column, and both main diagonals are the same. Generate a random magic square with given n.

  Example: n=3 $\rightarrow \begin{bmatrix}4&3&8\\9&5&1\\2&7&6\end{bmatrix}$

- There is given a square character matrix. Build the string that can be obtained with concatenating the elements of the matrix in the following order.

  Example: n=5, $\begin{bmatrix}T&h&t&a&o\\i&-&s&n&t\\s&k&-&-&a\\-&s&s&h&r\\i&o&-&d&!\end{bmatrix} \rightarrow$ This-task-is-not-so-hard!

3.3. set

- Generate a random lottery ticket that contains 5 different integer numbers from interval [1, 90]. Let the result be a set.

  Example: {39, 51, 14, 18, 48}

- There are given n closed intervals in the number line. The intervals are given with their left and right endpoints that are integer numbers. Two intervals *overlap* if they have common part (i.e. when there exists an integer value x such that x is included in both intervals), otherwise they are *disjunct*.
  - Decide that if the intersection of the intervals is empty or not, if not, give this intersection as an interval.

    Examples: n=4; [0, 3], [1, 2], [−3, −2], [−1, 2] → the intersection is empty

    n=3; [0, 3], [1, 2], [−3, 2] → [1, 2]
  - An integer number is *covered* if it is included in at least one interval. Determine the number of the covered integer numbers.

    Example: n=3; [0, 3], [1, 2], [−3, −2] → 6
  - Decide that if all pairs of intervals are disjunct or not. If not, determine an interval that overlaps the most other intervals.

    Examples: n=3; [0, 0], [1, 3], [−2, −1] → the intervals are disjunct

    n=4; [0, 0], [0, 3], [1, 2], [−2, −1] → [0, 3]
  - Minimize the number of intervals with uniting the intervals that overlap.

    Example: n=4; [0, 0], [0, 3], [2, 4], [−2, −1] → n=2; [0, 4], [−2, −1]

- Generate the prime numbers that are not greater than a given $n \geq 2$ with using *Sieve of Eratosthenes*. This algorithm is described in the previous chapter. Implement this algorithm such that the list used by the algorithm is realized with a Python set.

3.4. dict

- There are given *n* integers that are years in the 21st century. Make the statistics of the data.

  Example: *n*=3; 2022, 2001, 2022 → {2001: 1, 2022: 2}

- There are given some characters and a string. Make the statistics of the characters included in the string.

  Example: ”a”, ”b”; ”Abracadabra” → {”a”: 4, ”b”: 2}

- As the special case of the previous exercise make the statistics of the characters included in a given string that are digits or English uppercase letters. If an examined character is not included in the string (i.e. the number of this character is 0) then it should not be in the result.

  Example: ”Bond, James Bond 007” → {”B”: 2, ”J”: 1, ”0”: 2, ”7”: 1}

- There is given a string that contains only English lowercase letters and spaces. It can be assumed that there are no leading and trailing spaces and there are no two spaces next to each other in the string. A *word* starts at the beginning of the string and after each space. Make a statistics of the words included in the string.

  Example: ”one plus one is two” → {”one”: 2, ”plus”: 1, ”is”: 1, ”two”: 1}

- There are given data pairs in a year. Each pair contains a time given in a string using format of *mm.dd.hh* (*mm*: month, *dd*: day, *hh*: hour) and a temperature that is an integer number. Determine the average temperatures for the months. Handle the case too, when a month has no data.

  Example: Input: ”01.01.01”, −1; ”02.28.00”, 4; ”01.31.02”, 0

  Output: 1: −0.5; 2: 4.0; 3: no data; …; 12: no data

- Expand the previous exercise such that in printing the result the names of the months should be used instead of their ordinal numbers.

  Example: Input: ”01.01.01”, −1; ”02.28.00”, 4; ”01.31.02”, 0

  Output: January: −0.5; February: 4.0; March: no data; …; December: no data

- There are given ages and weights of *n* people. Both data are nonnegative integer numbers. Determine the mean of the weights of each age.

  Example: *n*=4; (25, 50), (50, 60), (25, 90), (50, 50) → {25: 70.0, 50: 55.0}

- There are given *n* float numbers and a positive integer *m*. Calculate the minimum value (*a*) and the maximum value (*b*) of the data, separate the interval [*a*, *b*] to *m* equal intervals, and finally determine the number of the data that are included in each intervals. Let the results be given in integer percentage. The intervals are open at the left side and closed at the right side except the first one that is closed at the left side too.

  Example: *n*=4, *m*=2; 1.0, 3.0, 1.8, 5.0 → {[1.0, 3.0]: 75, (3.0, 5.0]: 25}

- There are given *n* students and *m* subjects. We know the names of the students and their marks of the subjects (each student has got exactly one mark of one subject). We know the names and the credits of the subjects as well. Determine the sum of the credits of the performed (the mark is greater than 1) subjects of each students. Determine the ratios of the marks of the subjects (i.e. how many students have got mark 1, 2, …, 5) given in integer percentage.

  Example:

  Input: *n*=4, *m*=2; Subjects: ”Mathematics”, 5; ”Python programming”, 3
  Students: ”Jim”, 1, 2; ”Lee”, 2, 3; ”Ben”, 2, 1; ”Ned”, 1, 1

  Output: ”Jim”, 3; ”Lee”, 8; ”Ben”, 5; ”Ned”, 0
  ”Mathematics”, 50, 50, 0, 0, 0
  ”Python programming”, 50, 25, 25, 0, 0

3.5. Exercises for using different data types

- There are given two sets. Make a list that contains the common elements of the sets.

  Example: {8, 1, 5, 2}, {1, 2, 3} → [1, 2]

- There are given two lists. Each list contains 5 different integer numbers from interval [1, 90]. One of them contains our numbers in a lottery ticket and the other contains the winning numbers of the week. Determine the number of the winning numbers in our ticket (i.e. the numbers of the common numbers of the lists). Hint: This task can be solved using only lists, but using sets in the calculation a shorter (and nicer) solution can be given.

  Example: [12, 21, 34, 42, 56], [21, 56, 77, 84, 90] → 2

- Determine the number of the different characters in a given string.

  Example: ”abba” → 2

- There is given a list that contains comparable data. Make a new list that contains only the different elements of the original list with keeping the original order of the elements.

  Examples: [8, 1, 5, 1, 5, 5, 2] → [8, 1, 5, 2]

  ['a', 'b', 'b', 'a'] → ['a', 'b']

- There is given a list that contains different data. The number of data (i.e. the length of the list) is an even number. Generate randomly a *complete pairing* (i.e. pairs that contain all data and each data is included in exactly one pair). For example the list contains names of students, in the classroom there are benches with two seats, and we want to generate a random seating arrangement. Solve the problem such that the result is a list that contains pairs given in tuple (or set or list).

  Examples: [8, 1, 5, 2] → [(2, 1), (8, 5)]

  ['a', 'b', 'c', 'd', 'e', 'f'] → [{'c', 'b'}, {'a', 'e'}, {'f', 'd'}]

  [”Bob”, ”Ben”, ”Joe”, ”Rod”] → [[”Joe”, ”Ben”], [”Rod”, ”Bob”]]

- There are n students in a course. We use integer numbers (from 1 to n) to identify the students. Generate random groups with a given size m. If the number of students cannot be divided by the desired group size, there should be fewer students in the last group.

  Examples: n=4, m=2 → {4, 1}, {2, 3}

  n=7, m=3 → [3, 6, 1], [2, 7, 5], [4]

- A card game uses 52 French-suited cards. This cards use four suits (club ♣, diamond ♦, heart ♥, spade ♠) and each suit has got 13 cards (2, 3, 4, 5, 6, 7, 8, 9, 10, Jack, Queen, King, Ace). Make a random hand for n players such that each player gets m cards (where $n*m \leq 52$). Of course a card can be given to at most one player. Let the result be a matrix sized $n \times m$ that contains strings. Each string corresponds to a card where the first character gives the suit of the card.

  Example: n=3, m=2 → [[”♥Ace”, ”♠King”], [”♣2”, ”♦2”], [”♠10”, ”♣9”]]

- Solve the previous task such that the result list contains sets that contain the proper strings.

  Example: n=3, m=2 → [{”♥Ace”, ”♠King”}, {”♣2”, ”♦2”}, {”♠10”, ”♣9”}]

- We plan a meeting for n people whose spare times are given with two integer numbers that give the beginning and ending hours (for example 14, 17 means that this person is free from 14.00 to 17.00). Everyone should attend the meeting. Determine the beginning and finishing time of the longest possible meeting.

  Examples: n=3; (”Eve”, 10, 18), (”Ben”, 14, 17), (”Ted”, 16, 19) → 16, 17

  n=3; (”Eve”, 10, 18), (”Ben”, 14, 17), (”Ted”, 18, 19) → there is no solution

- There are given n strings. Let the *weight* of a string be the number of the different characters of the string (for example: ”apple” → 4). Determine the descending order of the strings by their weights.

  Example: n=3; (”apple”, ”banana”, ”peach”) → [”peach”, ”apple”, ”banana”]

- There are given $n \geq 2$ points on a plane with their *Cartesian coordinates*.
  - Determine the ascending order of the points by the distance from the origin.

    Example: n=3; (0, 1), (1, 1), (0, 0.5) → (0, 0.5), (0, 1), (1, 1)
  - Determine the ascending order of the points by the distance from the center of gravity of the points.

    Example: n=4; (0, 0), (3, 0), (0, 3), (0, 1) → (0, 1), (0, 0), (0, 3), (3, 0)
  - Determine the order such that the first point is the closest point to the origin, the second point is the closest point to the first point that is not included in the sorted list, and so on.

    Example: n=4; (0, 0), (3, 0), (0, 3), (0, 1) → (0, 0), (0, 1), (3, 0), (0, 3)
- There are given n people with their names and birthdays. Determine the pair that has the minimum difference of their birthdays. Let the result be a list of pairs that contain the names of the proper people.

  Example:

  Input: n=3; (”Ian”, 2001.01.01), (”Bob”, 2002.01.01), (”Rod”, 2004.01.01)

  Output: [(”Ian”, ”Bob”)]

  Example:

  Input: n=3; (”Ian”, 2001.01.01), (”Bob”, 2001.01.01), (”Rod”, 2001.01.01)

  Output: [(”Ian”, ”Bob”), (”Ian”, ”Rod”), (”Bob”, ”Rod”)]
- There are given n nonnegative at most 4 digits long integer numbers in the decimal number system. Calculate the forms of the numbers given in the binary and in the hexadecimal number system. Determine an ascending order of the numbers such that the lengths of the numbers (i.e. the number of the digits of the numbers) are the same. This length is determined by the length of the greatest number. Fill the shorter numbers with leading 0 to reach the proper length.

  Remark: If you use the proper Python functions (for example: bin, hex) then it is an easy exercise, otherwise (if you do the conversion between the given number systems by yourself) it is a difficult task (marked with black color). We used black color to suggest you to solve this task without the mentioned functions (because in this case it is more interesting and challenging ☺).

  Example:

  Input: n=3; 3, 255, 0

  Output: [(0, ”00000000”, ”00”), (3, ”00000011”, ”03”), (255, ”11111111”, ”FF”)]

4. FILE

Each exercise in the previous chapters can be solved such that the input data are given in a given text file from where the solution reads them. In this case they are good (and easy) exercises of file handling too.

- There is given a text file (source) and a (searched) string. Make a new text file (target) with a given name (without modifying the source) that contains the lines of the source that contain the searched string with keeping the format and the order of these lines.
- A *post address* is a string with three parts: *zip code* (contains only digits), *place name* (not contains space), *other data*. There is exactly one space between the parts. The addresses are given in a text file such that one line contains one address. The data are not sorted. Solve the following exercises.
  - Make a list (to the display) of the addresses by ascending order of the zip code.
  - Realize the searching by zip code (i.e. make a list of addresses that have a given zip code).
  - Make a sorted list of the addresses to a given text file by ascending order of the place name.
  - Realize the approximate searching by place name (i.e. select the addresses whose place name begin with a given string).
- We make *mixtures* from *raw materials*. Each mixture has got a unique name (that starts with an English letter). Each raw material has got a unique identifier that contains exactly three digits. A mixture can contain arbitrary number (at least one) of raw materials that are given with their identifiers and the amounts. The data are stored in a text file. In the first line of a mixture contains the name of the mixture and the following lines contain the data (identifier and amount) of the raw materials belonged to that mixture. The mixtures are sorted (by ascending order of their names) in the text file and the row materials of one mixture are also sorted (by ascending order of their identifiers). Solve the following exercises.
  - Load the data from a given text file to a proper data structure.
  - Save the current data (from the memory) to a given text file (using the format described above).
  - Realize the following operations of mixtures: searching, inserting a new mixture, deleting.
  - Realize the operations of a given raw material of a given mixture: inserting, deleting, and modifying its amount.

5. FUNCTION

Each exercise in the previous chapters can be solved with a function as well. In this case they are good exercises of using functions, so this chapter contains only one exercise (for using a function as a parameter of another function ☺).

- There is given a function *f* that is continuous in a given interval [*a*, *b*]. Compute the definite integral of *f* in this interval.

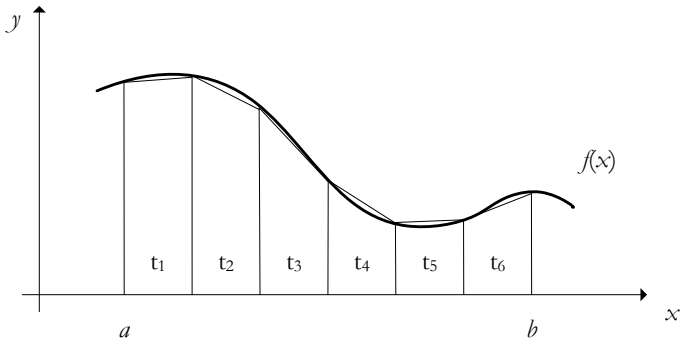

Figure 1. The trapezoidal rule with uniform grid

Hint: This computing can be done by using approximate calculating with trapezoidal rule as follow. Divide the interval [*a*, *b*] into *n* equal parts then draw vertical lines at these division points. Connect the points of intersection of the adjacent parallels and the function (see Figure 1.). We obtain a trapezoidal system whose sum of areas approximates the definite integral of the function. By refining the scale (for example halving each subinterval), the sum of the areas of the trapezoids gives an even better approximation of the definite integral.

The solution can therefore be to make a series consisting of the sum of the areas of the trapezoids belonging to the refinement scale, this series is close to the integral value sought, and if two consecutive sum of areas hardly differ from each other (the absolute value of the difference is less than a given ε accuracy), then the sum produced by the last integral is accepted as an approximation of the definite integral.

Formalized, if the interval is divided into *n* equal parts (i.e. using uniform grid) and the value of the function taken at the *i*th division point is denoted by y_i, the sum of the trapezoids can be written as:

$$T = h(\frac{y_0 + y_n}{2} + \sum_{i=1}^{n-1} y_i)$$

where y_0 is the function value taken in *a*, y_n is the function value taken in *b* and $h=(b-a)/n$.

Remark: A well written function focuses to only its task. It means that the function gets their input data (in its parameters) that are necessary to its operation, does the operation and gives back the result as the value of the function. Thus the communication with the user (for example reading input data, printing the result) has to be done in the caller (i.e. the part of the program that calls the function), not in the function itself.