USKUDAR UNIVERSITY

INSTITUTE OF SCIENCES, DEPARTMENT OF COMPUTER ENGENEERING, PROGRAM ADVANCED ALGORITHMS

2022-2023 FALL

FINAL PROJECT REPORT

Team members and IDs:

224308902 Nurbek Juraev

214308917 Abdulmajid idris

200201806 Ertuğrul GÜRLER

190200041 FEYZA SARIOĞLU

190201033 Bircan AYGÜN

180201071 Emre DALGIÇ

190201039 Zehra Nur MANGAL

180201011 Ayşe Beyza DOĞAN

PROJECT TITLE: CASHIER ALGORITHM WITH TURKISH CURRENCY IMPLEMENTATION

**Table of contents**

[About our algorithm 2](#_Toc124199740)

[Several versions of our code 2](#_Toc124199741)

[1st and 2nd versions 2](#_Toc124199742)

[3rd and 4th versions 3](#_Toc124199743)

[5th, 6th and 7th versions (GUI) 4](#_Toc124199744)

[Input of our algorithm 6](#_Toc124199745)

[Output of our algorithm 7](#_Toc124199746)

[Pseudocode for the current algorithm 7](#_Toc124199747)

[Running time of the current algorithm 8](#_Toc124199748)

[Comparing running time of the algorithm with various input sizes 8](#_Toc124199749)

[Real-world application of the algorithm 9](#_Toc124199750)

[Importance of our Algorithm 10](#_Toc124199751)

[The programming language used in our implementation 10](#_Toc124199752)

[Conclusion 10](#_Toc124199753)

[APPENDIX 1 11](#_Toc124199754)

[APPENDIX 2 12](#_Toc124199755)

[Different approach Version 1 12](#_Toc124199756)

[Different approach Version 2 13](#_Toc124199757)

[Disadvantages of using different approach: 19](#_Toc124199758)

# About our algorithm

Our algorithm is used to find the minimum number of denominations of Turkish currency needed to make a given amount of money.

Our algorithm is based on greedy algorithm, is one that always takes the best immediate, or local, solution while finding an answer. Greedy algorithms find the overall, or globally, optimal solution for some optimization problems, but may find less-than-optimal solutions for some instances of other problems.

However, we have modified our algorithm to make it an optimal solution provider for our problem.

# Several versions of our code

In our code, a function named findMin is created to find the number of denominations of Turkish currency needed to make a given amount of money. The given amount of money is passed to the findMin function as an argument. The denominations of Turkish currency are stored in an array named deno inside of the findMin function. At the end, the function prints an array named ans which stores the denominations that make the given amount of money. We have two different approaches for our algorithm: top-down and bottom-up. In the top-down approach, the algorithm iterates through the array named deno starting from the largest denomination. In the bottom-up approach, the algorithm iterates through the array named deno starting from the smallest denomination.

Overall, we have 7 versions of our code:

## 1st and 2nd versions

The code works like we have an infinite number of coins or notes for each denomination of Turkish currency. The given amount of money in kurus is passed as an argument to the findMin function. The deno array stores the denominations. The ans array stores the denominations that will be used to make the given amount of money. In each iteration, the inner while loop checks if the current denomination is less than or equal to the remaining amount of money. If the condition is met, the current denomination is added to the ans array. Then, the remaining money is updated. This process continues until all of the denominations are checked. The elements of the ans array are divided by one hundred before they are printed to convert them to lira from kurus. We have two different approaches for this version: bottom-up and top-down.

In the top-down approach, the algorithm iterates through the deno array starting from the largest denomination.

In the bottom-up approach, the algorithm iterates through the deno array in ascending order, starting from the smallest denomination. We have an infinite number of coins or notes. So, the result contains only one kurus, and the number of printed one kurus is equal to the given amount of money.

Ver.1



Ver.2



## 3rd and 4th versions

Number of coins or notes of a denomination of Turkish currency cannot be infinite in real life so we modified our algorithm. There are a limited number of coins or notes for each denomination in this version. An additional array named limits is created to delimit the number of each Turkish currency denomination. The limits array is passed as a second argument to the findMin function. Then, a second check is added to the inner while loop, which is inside the findMin function, that ensures the limit of coins or notes of each denomination does not exceed the current limit. In this way, the inner loop checks two things. First one is if the current denomination is less than or equal to the remaining amount of money. Second one is if the number of the denomination is less than or equal to the current limit. When both of the conditions are met, the current denomination is added to the array named ans. Then, the remaining amount of money and the available limits are updated. This process continues until all of the denominations are checked. We have two different approaches for this new version that limits are included: bottom-up and top-down.

In the bottom-up approach of the algorithm, the program iterates through the deno array starting from the first element which is the smallest denomination.

In the top-down approach of the algorithm, the program iterates through the deno array starting from the biggest denomination that means in reverse order.

Ver. 3 (with limited number of 1 kurus and 5 kurus)



Ver. 4 (with limited number of 100 lira = 0, 50 lira = 1, 20 lira = 1, 10 lira = 5)



## 5th, 6th and 7th versions (GUI)

We've developed frontend so the user can more conveniently and easily access what they want.

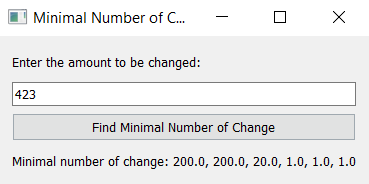
We used python GUI for this frontend and we use the python PyQt5.Qt Widgets library for the graphical user interface. GUI stands for Graphical User Interface, and refers to computer programs that provide a visual means for users to interact with an underlying application or system. A graphical user interface works by waiting for the user to do something. The something is called an event. Events happen when the user types something while your application is in focus or when the user uses their mouse to press a button or other widget. We have 3 version for that. As we mentioned before, in the first case of the greedy algorithm, we change the desired value in the minimum way. Since denomination is adapted to the Turkish currency, we expect the users to enter the desired change in Turkish lira. In 5th version, since there is no limit in our denomination, we can exchange the money you have for at least a coin or note. After the user presses the "find minimal number of change" button, the amount to be changed is stored with the self-variable. It is then passed as a parameter to the self-variable find\_min function. The minimal change in the find\_min function is shown on the screen as a result.

In 6th version, we ask the user to enter the limit values. As an example, we ask how many of the 200 Turkish liras you have. We expect the user first to send the limit values with the "submit limits" button. The variables sent into the submit limit function are saved in the array named self.limit. After the user presses the "find minimal number of change" button, the amount to be changed is stored with the self variable. It is then passed as a parameter to the self variable find\_min function. The minimal change in the find\_min function is shown on the screen as a result.in the second section tracking the number of limits in version 6 was not user-friendly and could be changed accidentally by changing the value in the limits box, hence we have created the 3rd version of our GUI.

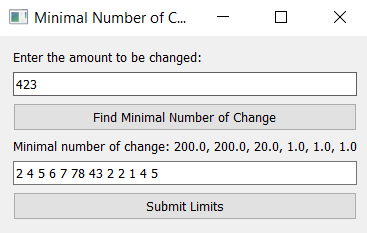
In the latest version, a separate button has been designed for each limit. The purpose of this design is for the user to understand and use the application more easily.

The limits we entered are for 1, 5, 10, 25, 50, 100, 500, 1000, 2000, 5000, 10000, 20000, respectively. After the limits variables are stored, it starts to find the minimum number of denominations of Turkish currency needed to make a given amount of money. Again, the denominations of Turkish currency are stored in the deno array, and the amount of money is passed as an argument to the findMin function. Minimal number of change shows on the screen.

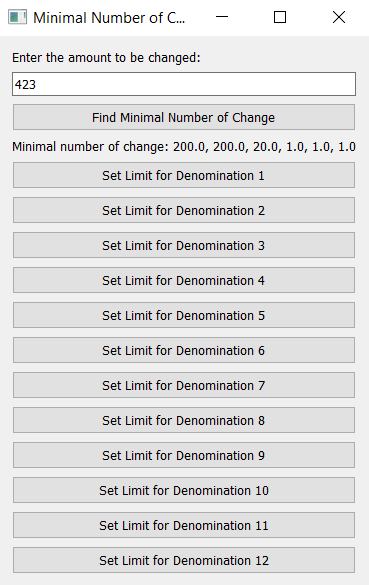
Ver. 5



Ver. 6



Ver. 7



# Input of our algorithm

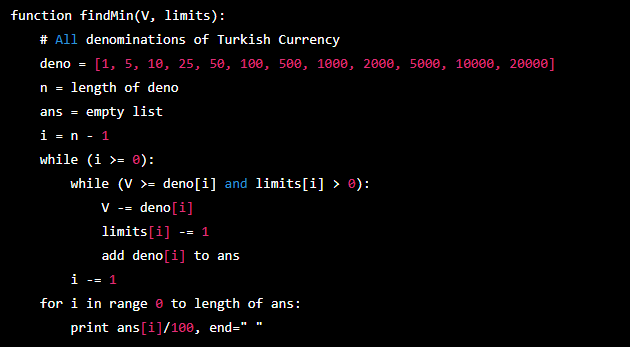
The input for this algorithm includes the variable **V**, which is the value to be expressed in terms of the available denominations of Turkish currency, and the **limits** list, which is a list of the maximum number of each denomination that can be used. **V** is initialized to the value **150.52\*100**, which means the value to be expressed is 150.52 Turkish lira. The **limits** list is initialized to a list of integers representing the maximum number of each denomination that can be used.

# Output of our algorithm

The output of this algorithm will be a list of the denominations of Turkish currency needed to express the value **V** in the most minimal way possible, given the maximum number of each denomination specified in the **limits** list. The output will be printed to the console, with each denomination separated by a space.

For example, if **V** is initialized to **150.52\*100** and **limits** is initialized to **[10, 5, 10, 5, 10, 5, 10, 5, 1, 1, 5, 5]**, the output might be: **"100 50 0.5 0.01 0.01"**. This means that the value of 150.52 Turkish lira can be expressed using 1-100 lira 1-50 lira 1-50 kurus 2-1 kurus denominations, given the specified limits on the number of each denomination that can be used.

# Pseudocode for the current algorithm

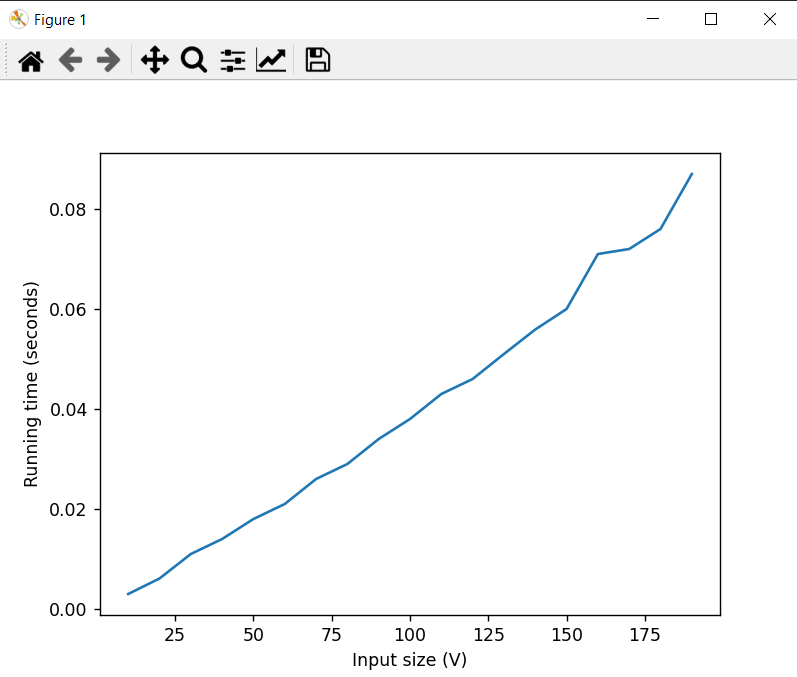


This pseudocode describes the overall structure and logic of the **findMin** function, which takes two inputs: **V**, the value to be expressed in terms of the available denominations of Turkish currency, and **limits**, a list of the maximum number of each denomination that can be used. The function first initializes a list of all the denominations of Turkish currency, and a list **ans** to store the denominations that will be used to express **V**. It then enters a loop that starts at the largest denomination and works its way down to the smallest, trying to use as many of each denomination as possible given the specified limits. The function then prints the list **ans** to the console, with each denomination separated by a space.

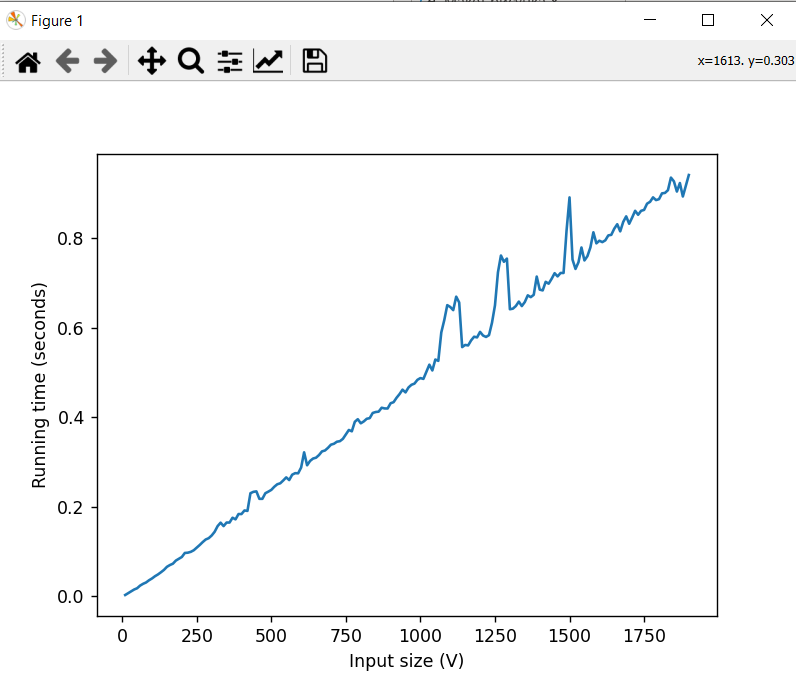
# Running time of the current algorithm

The running time of the current algorithm is O(n), where n is the number of denominations of Turkish currency. This is because the algorithm performs a single pass through the list of denominations, and the number of operations performed within the loop is constant (i.e., the loop does not contain any nested loops or perform any operations that have a variable running time).

## Comparing running time of the algorithm with various input sizes

Graph 1 with input size no more than 200

Graph 2 with input size no more than 2000



# Real-world application of the algorithm

* Vending machine that needs to dispense change to customers. In order to minimize the number of coins or bills it needs to dispense, the vending machine could use our algorithm to find the minimum number of denominations needed to express the amount of change owed.
* Software program used by a bank to count and package coins and bills for transport. This program could use our algorithm to minimize the number of packages needed to transport a given amount of money.

These are just a couple of examples, but our algorithm has many other potential applications in areas such as finance, computer science, and engineering.

# 

# Importance of our Algorithm

The importance of our code is that it helps to find the minimum number of denominations required to make a certain financial transaction in Turkish currency. The code defines a function called "findMin" that takes in two parameters: the value of the transaction and the limits of the available denominations. It then traverses through a list of all denominations of the Turkish currency and finds the minimum number of denominations required to make the transaction. The result is then printed for the user to see.

Our code is important because it helps to ensure the accuracy and efficiency of financial transactions in Turkish currency. By using this code, a cashier or financial institution can quickly and easily determine the minimum number of denominations needed to make a transaction, saving time, and reducing the risk of errors. This is especially important in a country like Turkey where there is a high volume of financial transactions and a need for efficient and accurate exchanges.

# The programming language used in our implementation

We used Python version 3 for coding our Algorithm.

# Conclusion

A greedy algorithm is any algorithm that follows the problem-solving heuristic of making the locally optimal choice at each stage. This algorithm is used to find the minimum number of denominations of any currency needed to make a given amount of money. In our project we modified it for Turkish currency in order to work with Turkish currency as a denominator. We implemented several versions to improve our code. New versions including a bottom-up approach and limits for our denominators. Greedy algorithm can often produce good results, it is not always guaranteed to find the optimal solution. One way to improve a greedy algorithm is to use a different strategy for making the locally optimal choices at each step. This would be more time-consuming, but it would provide a more reliable way of finding the optimal solution. This help reduce the amount of time needed to find the minimum number of denominations, especially for larger amounts of money. For the best optimum result by saving time, we created another version with dynamic programming and an additional version without using any existing algorithms at all. Finally, we tested and selected the best one and we have created GUI for our code so the user can more conveniently and easily access that. We have created 3 versions of GUI starting from the basic one, and then we have included limits in the GUI, and in the final version we have updated to make our GUI more practical and user friendly. In conclusion, we have successfully developed and implemented various algorithms and corresponding GUI designs to solve the problem of currency denomination in Turkish currency. Through a process of coding, testing, and refining, we were able to identify the most effective solution and create a user-friendly interface for it. The end result is a reliable and efficient tool for handling currency denominations in Turkish currency

# APPENDIX 1

Version of program using dynamic programming.

We wanted to use the dynamic programming method in our project. Dynamic programming is mainly an optimization over plain recursion. Wherever we see a recursive solution that has repeated calls for same inputs, we can optimize it using dynamic programming. The idea is to simply store the results of subproblems, so that we do not have to re-compute them when needed later.

The program begins by initializing an empty list ‘ans’ to store selected denominations and a list ‘dp’ to store results of subproblems. The ‘dp’ list is initialized with a value of infinity for each possible amount of money from 0 to the given amount and the value at index 0 is set to 0.

The program then loops through all denominations in ‘deno’ and uses these to build up the ‘dp’ list, starting from the smallest denominations and working up to the largest. For each denominations and each possible amount of money, the program calculates the minimum number of denominations needed to make that amount by taking the minimum of the previously calculated value for that amount and the value for the amount minus the current denomination plus 1. This recursive approach allows the program to build up a solution for the given amount by using solutions for smaller amounts.

Once the ‘dp’ list has been built up, the program loops through the denominations in reverse order and uses the stored minimum number of denominations to select the denominations that should be used to make the given amount. The program keeps track of the remaining number of each denomination that can be used in the ‘limits’ list, and appends the selected denominations to the ‘ans’ list.

Finally, the program prints out selected denominations, which represent the minimal number of changes needed to make the given amount of money.

Overall, the dynamic programming approach used in this code allows for an efficient solution to the problem of finding the minimum number of denominations needed to make a given amount of money, by breaking the problem down into smaller subproblems and using the solutions to these subproblems to build up a solution for the larger problem.

The time complexity of the above code is O(nV), where n is the length of the deno list and V is the amount of money to be converted into denominations. This is because the first loop that calculates the minimum number of denominations needed has a time complexity of O(nV). The time complexity is O(nV) because the loop runs for a maximum of nV iterations.

The second loop that traverses through the denominations in reverse order has a time complexity of O(n). The time complexity of the rest of the code is constant, as it consists of simple operations such as appending to a list and printing.

Therefore, the overall time complexity of the code is O(nV).

We skipped this version of our code as Overall time complexity is worst than what we currently have.

# APPENDIX 2

## Different approach Version 1

Version of program using different approach.

What it actually does is that creating a twelve-element array consists of zeros. The main purpose of the array is storing the value of money to be given to the user. Each element represents a denomination of Turkish currency

It takes the whole number which is taken from the user except for the last four digits and divides it by two and increments the responsible array's element by result. This result is a representation of 200 TL’s in our money. If the number is odd it will increment the specific array element which is responsible for representation of 100 TL's by one.

For the last 4 digits, It is going to take a more complicated approach since there are many denominations in Turkish currency that are responsible for those digits. For the tens digit, the program looks for 50 TL’s, 20 TL’s and 10 TL’s, for ones digit, 5 TL’s and 1TL’s. For the first digit of kurus, the program looks if there are 50 Kurus, 25 Kurus and 10 Kurus, for the second digit of kurus, It searches for 5 Kurus and 1 Kurus. If there are, it is going to increment the responsible array element that we created at the beginning. After a few lines of code we finally have an output ready to show the user.

The time complexity of the above code is O(1), as it consists of examination of some digits and performs a constant number of operations on each digit. The time complexity is constant because the program runs for a fixed number of lines, regardless of the input size.

The time complexity does not depend on the size of the input (the amount of money to be converted into denominations), as the code does not perform any operations that increase in time as the input size increases. Therefore, the time complexity of the code is O(1).

## Different approach Version 2

First version of the program doesn't handle the limitation problem.

At this part we coped with the problem by comparing the "change" array which we got from the first part of the program, by the "limit" array which user gave us at the beginning of the program.

First, if there is a difference between elements, we store the difference. Then we convert the change element which is too much for our limit element to the next type of denomination from the Turkish currency until it satisfies the condition that user allow us at the beginning of the program.

At the end, the second part of the program has a time complexity of O(n), where n is the number of elements in the limit array. This is because the program iterates through the limit once and the rest of the code has a constant time complexity.

Pseudocode of the algorithm:

// This algorithm is implementing a different approach to the cashier algorithm by examining the digits to be given with the limitation problem. Limits are going to be taken from the user. For denomination Turkish Currency is selected.

function(money,limit):

Create an 12 element array which represents money to be given to the user

named "change". Each element represents a denomination in Turkish currency.

Take the digits of the money except for the last four.

AmountOf\_200TL = Add the result from the division of money without four digits

by 2

AmountOf\_100TL = if (money without four digits is odd): --> increment one

Take the rest of the digits.

Take the first digit. //Examine the tens digit.

AmountOf\_50TL = if (digit is bigger than 5):

--> increment one

--> subtract 5 from digit

AmountOf\_20TL = if (digit is bigger than 2):

--> Add the result of division by 2

--> subtract result of division\*2 from digit

AmountOf\_10TL = add remainder to this value

Take the second digit. //Examine the ones digit.

AmountOf\_5TL = if (digit is bigger than 5):

--> increment one

--> subtract 5 from digit

AmountOf\_1TL = add remainder to this value

Take the third and fourth digit. //Examine the rest of the digits.

AmountOf\_50KRS = if (third digit is bigger than 5)

--> increment one

--> subtract 5 from digit

AmountOf\_25KRS = if (third digit is bigger than 2 & fourth digit is bigger than 5):

--> increment one

--> subtract 2 from third digit

-->subtract 5 from fourth digit

=if (third digit is bigger than 2 & fourth digit is smaller than 5)

--> increment one

--> subtract 3 from third digit

-->add 5 to fourth digit

AmountOf\_10KRS = add remainder of third digit to this value

AmountOf\_5KRS = if (fourth digit is bigger than 5):

--> increment one

--> subtract 5 from fourth digit

AmountOf\_1KRS = add remainder to this value.

//For the limitation part

for (every element of "limit" and "change" array):

compare elements and store the difference

if (index is 2 or 8):

increment the next change element by difference\*2

increment the 2 next change element by difference

subtract difference from change element

else if (index is 5 or 10):

increment the next change element by difference\*5

subtract difference from change element

else:

increment the next change element by difference\*2

subtract difference from change element

return change array.

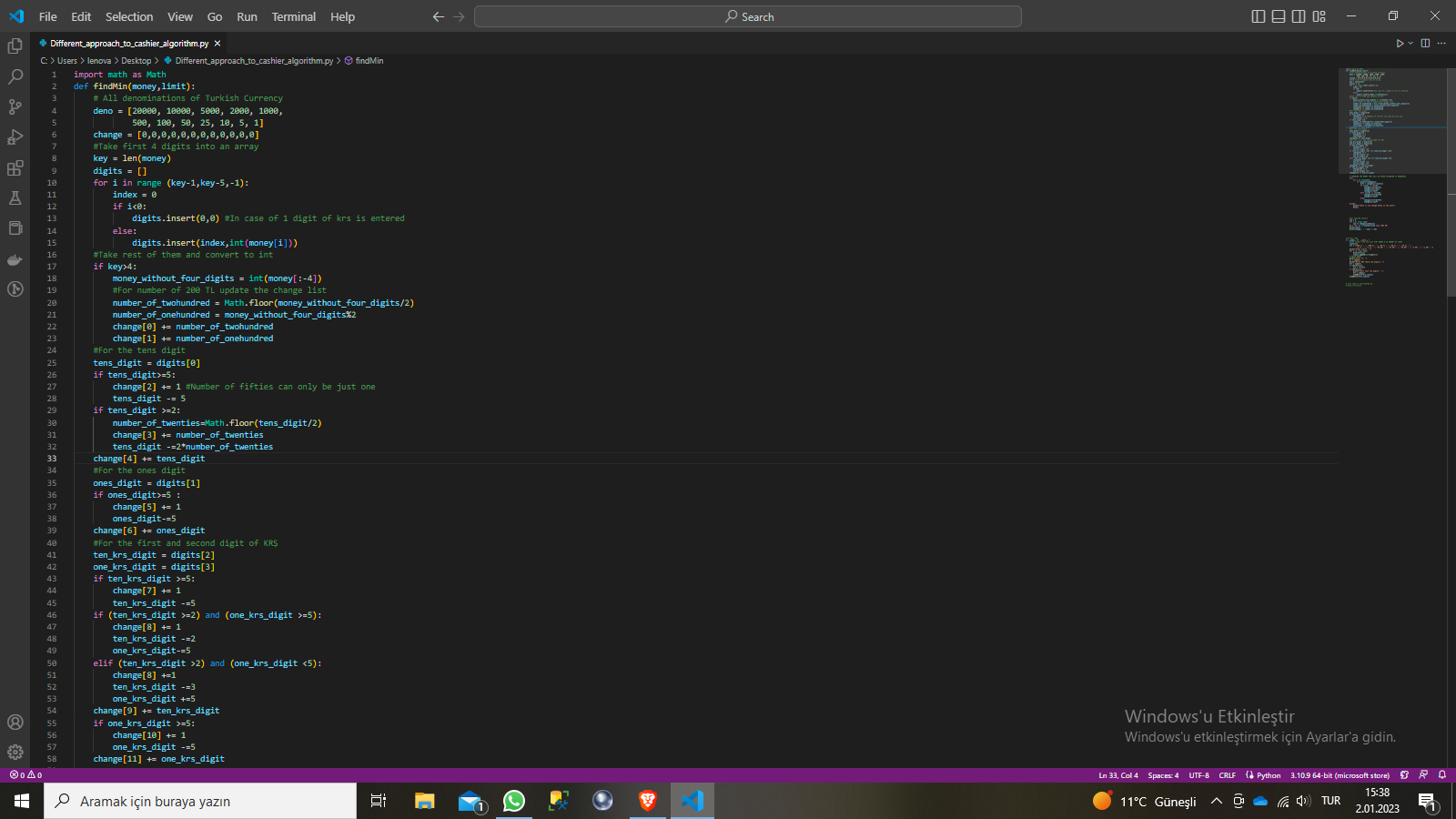
Main function():

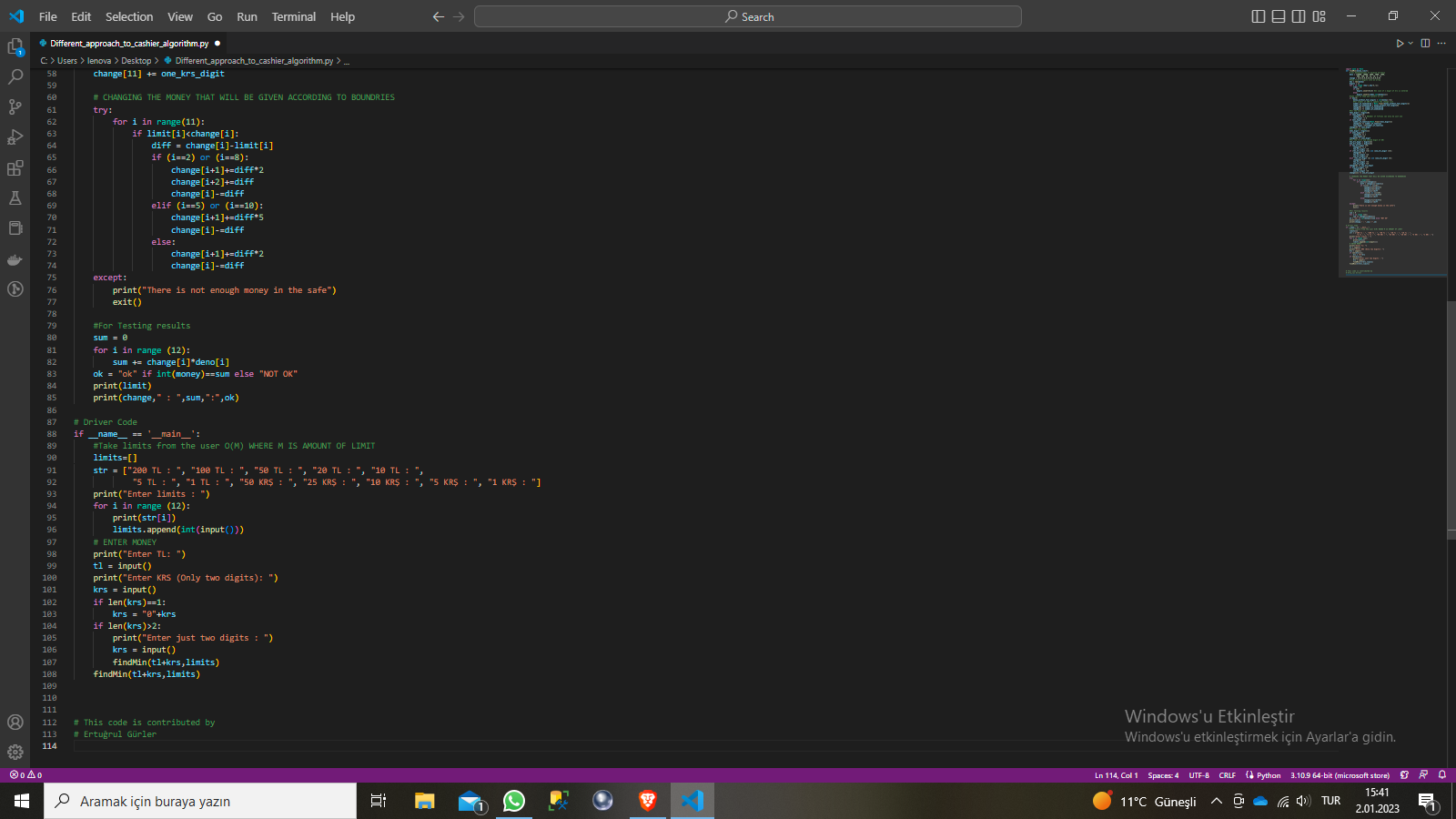
Take money to be given from the user.

Take a limit array from the user.

Start the algorithm(money,limit)

Code itself:





## Disadvantages of using different approach:

One of the biggest disadvantages of using this algorithm is that It is not flexible enough to use other kinds of denominations. Every time we need to implement the other kinds of denominations, we have to make major changes on our source code. That's the biggest disadvantage while using brutal force on Cashier Algorithm

The other disadvantage of using brutal force is that It is not easy to write the source code. Even if we want to change the code for implementing another kind of denomination, programmers will not be able to understand and make a change on source code easily. That's because of the complex structure that calculates every single possibility for digits.

The biggest difficulty to implement other denominations is that the currencies are not always suitable for basic mathematical operations. Sometimes a programmer has to think that while performing an operation the division will not always be an easy task, there will be some fractions to deal with. Depending on the complexity of the denomination it will be harder to examine the responsible digits since some denominations affect multiple digits.