# Exercises: Arrays and Methods

Problems for exercises and homework for the [“Programming Fundamentals Extended” course @ SoftUni](https://softuni.bg/courses/programming-fundamentals).

You can check your solutions here: <https://judge.softuni.bg/Contests/Practice/Index/418>.

## Hello, Name!

Write a **method** that receives a name as **parameter** and prints on the console. “Hello, <name>!”

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| Peter | Hello, Peter! |

## Min Method

Create a GetMin(int a, int b) method, that returns the **smaller** of two numbers. Write a program that reads three numbers from the console and prints the **smallest** of them. Use the GetMin(…) method you just created.

### Examples

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Input** | **Output** |  | **Input** | **Output** |
| 1  2  3 | 1 |  | -100  -101  -102 | -102 |

## String Repeater

Write a **RepeatString(str, count)** method that receives a **string** and an **integer** **N** as **parameters** and returns the string, **repeated** **N** times. After that, print the result on the console.

|  |
| --- |
| static string RepeatString(string str, int count)  {  string repeatedString = string.Empty;    for (int i = 0; i < count; i++)  {  // TODO  }  return repeatedString;  } |

### Examples

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Input** | **Output** |  | **Input** | **Output** |
| str  2 | strstr | roki  6 | rokirokirokirokirokiroki |

## Nth Digit

Write a **method** FindNthDigit(number, index) that receives a **number** and an **index N** as **parameters** and returns the **Nth digit** of the number (counted from **right to left**, starting from **1**). After that, **print** the result on the console.

|  |
| --- |
| static int FindNthDigit(long number, int index)  {  …  } |

### Examples

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Input** | **Output** |  | **Input** | **Output** |  | **Input** | **Output** |
| 83746  2 | 4 | 93847837  6 | 8 | 2435  4 | 2 |

### Hints

* Use a while loop to perform the algorithm until the given number becomes **0**.
* On every iteration of the while loop, check if the **current index** of the number equals the **index** you are looking for. If it is, return the number at that index (number % 10). If it’s not the correct index, **remove** the last digit from the number (number / 10).
* Keep track of which **digit** you are checking by **index** (**right to left**, starting from **1**). When you **find** the digit, **return** that **index**.

## Integer to Base

Write a **method** **IntegerToBase(number, toBase)** that receives a **number** and a **base** as **parameters** and returns the number, **converted** to that **number base.** After that, **print** the result on the console.

The **number** will **always** be in **base 10** and the **base** parameter will **always** be **between 2 and 10**.

|  |
| --- |
| static string IntegerToBase(long number, int toBase)  {  …  } |

### Examples

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Input** | **Output** |  | **Input** | **Output** |  | **Input** | **Output** |
| 3  2 | 11 | 4  4 | 10 | 9  7 | 12 |

### Hints

* Declare a variable where the **result** will be stored.
* Perform the calculations, needed to convert the number:
  + Take the **remainder** of the **number**, divided by the **base.**
  + Add the **remainder** to the **front** of the result string.
  + Divide the **number** by the **base**
  + The above algorithm should run **until** the number is **0**

## Notifications

Write 2 methods:

|  |
| --- |
| static string ShowSuccess(string operation, string message)  {  …  } |

which prints a message in the following format:

|  |
| --- |
| Successfully executed {operation}.  ==============================  Message: {message}. |

and

|  |
| --- |
| static string ShowError(string operation, int code)  {  …  } |

which prints a message in the following format:

|  |
| --- |
| Error: Failed to execute {operation}.  ==============================  Error Code: {code}.  Reason: {reason}. |

The **reason** is determined by the **code**:

* If the code is **positive**, the reason is "Invalid Client Data"
* If the code is **negative**, the reason is "Internal System Failure"

Then, create a program which accepts an integer **N**.

On the next **N** lines, the user receives a **result** – "success" or "error". If the current operation is success, it should prompt the user for an **operation** and a **message**. If the operation is error, it should prompt the user for an operation and a code.

If neither "success", nor "error" is given, **nothing** should be printed and the program should **continue**.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 2  success  User registration  User registered successfully  error  connecting to server  -403 | Successfully executed User registration.  ==============================  Message: User registered successfully.  Error: Failed to execute connecting to server.  ==============================  Error Code: -403.  Reason: Internal System Failure. |

|  |  |
| --- | --- |
| **Input** | **Output** |
| 3  success  Database query  Updated 2 rows  warning  error  credit card purchase  2 | Successfully executed Database query.  ==============================  Message: Updated 2 rows.  Error: Failed to execute credit card purchase.  ==============================  Error Code: 2.  Reason: Invalid Client Data. |

## \* Numbers to Words

Write a method Letterize(number) which accepts a number and prints it as **words**, according to these conditions:

* If the number is more than 999 you should print – "too large"
* If the number is less than -999 you should print – "too small"
* If the number is negative, you should print "minus" before it.
* If the number does **not** have 3 digits, **do not** print it

The program should accept an integer **N.** On the next **N lines,** you should accept **numbers** and print them as **words**.

### Examples

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Input** | **Output** |  | **Input** | **Output** |
| 3  999  -420  1020 | nine-hundred and ninety nine  minus four-hundred and twenty  too large | 2  15  350 | three-hundred and fifty |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Input** | **Output** |  | **Input** | **Output** |
| 4  311  418  519  -9945 | three-hundred and eleven  four-hundred and eighteen  five-hundred and nineteen  too small | 2  500  123 | five-hundred  one-hundred and twenty three |

### Hints

* The first set of special cases comes when a number’s right on the **hundreds** (i.e. 100, 200, 300, etc.). Print them like this: “one-hundred”, “two-hundred”, “three-hundred”.
* The second set of special cases comes when a number’s last 2 digits are **less than 10** (i.e. 101, 305, 609, etc.). Print them like this: “one-hundred and one”, “three-hundred and five”, “six-hundred and nine”
* The third set of special cases comes when a number is in the **teens** (i.e. **111, 814, 919**). Print them like this: “one-hundred and eleven”, “eight-hundred and fourteen”, “nine-hundred and nineteen”

## \* String Encryption

Write a method Encrypt(char letter) which encrypts the letter in the following way:

* Take the first and the last digit of its **ASCII** code and append them together in a string.
* Append at the **start** of the resulting string the **character corresponding to**:
  + the **ASCII code** of the letter **+** the **last digit** of the **ASCII code** of the letter
* Then append at the **end** of the resulting string the **character corresponding to**:
  + the **ASCII code** of the letter **-** the **first digit** of the **ASCII code** of the letter
* The method should return the encrypted string.

Example

* j -> p16i
  + j’s ASCII code is **106 🡺** First digit – **1**, last digit – **6**
  + Append the first and last digit 🡺 **16**
  + Append the character, resulting from the ASCII code + last digit to the **start** 🡺 106 + 6 🡺 112 🡺 p
  + Append the character, resulting from the ASCII code - first digit to the **end** 🡺 106 - 1 🡺 105 🡺 i
  + End result 🡺 p16i

Assume that the input for this method will **always** be valid.

The main program should read an input from the user - an integer **N**.

On the next **N lines,** you will receive **characters**. **Encrypt** them and **append** them to an **encrypted string**.

In the end, what you should have is a long **encrypted string** of characters like this:

* S, o, f, t, U, n, i 🡺 V83Kp11nh12ez16sZ85Mn10mn15h

Your task is to **print** the **encrypted string.**

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 7  S  o  f  t  U  n  i | V83Kp11nh12ez16sZ85Mn10mn15h |

|  |  |
| --- | --- |
| **Input** | **Output** |
| 7  B  i  r  a  H  a  x | J72Ah97Xl99Zr17jH66<n15hv14qh97X |

### Hints

* Start with an empty **result** string – use string.Empty.
* Loop **n** times and each time **append** the **encrypted character** to the result string.
* To get the **first** and **last** digits of the **ASCII** code, use the same algorithm as the one you’d use to solve the **Integer to Base** problem.

## Largest Element in Array

Read an integer **N** and on the next **N** lines read an **array** of **integers.** Then, find its **largest** element.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 4  5  6  8  7 | 8 |
| 3  -2  -18  -5348 | -2 |
| 6  2  2  2  2  2  2 | 2 |

## Count of Negative Elements in Array

Read an integer **N** and on the next **N** lines read an **array** of **integers.** Then, find the **count** of **negative** elements in the array.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 5  1  -2  3  -4  5 | 2 |
| 3  1  3  2 | 0 |
| 6  -1  -2  -3  -4  -4  4 | 5 |

## Count of Given Element in Array

Read an array of integers and print how many times a given element exists in it.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 5 2 3 1 5  2 | 1 |
| 1 4 4 4 1  4 | 3 |
| 8 4 9 0 0  0 | 2 |

### Hints

* Use the .Split() function to read the array on a single line.

## Count Occurrences of Larger Numbers in Array

Read an **array** of **real numbers** and a number **p**. Find how many elements are **bigger** than **p** in the array and **print** the count.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 3 2 3.5 3 2 4 3 4 -2 -7 3  2.9 | 7 |
| 5 6 105 3 2 849  100 | 2 |
| 1.5 4.1 9.3 10.5 0.85  4.9 | 2 |

## Increasing Sequence of Elements

Read an **array of integers** and find out if it is an **increasing sequence**. Print Yes if it does and No if it doesn’t.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 1 5 10 12 | Yes |
| 1 5 2 12 | No |

## Equal Sequence of Elements in Array

Read an **array of integers** and find out if all the elements in the array are **the same**.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 5 5 5 | Yes |
| 3 4 4 | No |

## Count of Capital Letters in Array

Read an **array of strings** and find out how many of them are **capital English letters** (such as A, B, C etc.). Print the count on the console.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| Hello SoftUni I am A student | 2 |
| I Think A B and C are the first three letters of the alphabet | 4 |

## Array Symmetry

Read an **array of strings** and find out if it’s **symmetric**. If it is, print “Yes”, otherwise print “No”.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| hi you hi | Yes |
| ho hi hi ho | Yes |
| hi ho ho ho | No |

### Hints

* Iterate over the array as follows:
  + Check whether the **first** and **last** elements are equal
  + Then check whether the **second** and the **next to last** elements are equal
  + Continue this pattern until you either **find an inequality** or **reach the middle** of the array.
* Alternate solution: **reverse** the array and check if it is the **same** as the **original** array.

## \* Altitude

You are an airplane pilot, trying to maneuver your airplane to safety from an unknown danger.

An array holds a sequence of up / down commands and **numbers**. Its first element **always** holds the **initial altitude**. Thecommand up **increases** the altitude by the next number, while the command **down decreases** the altitude by the **next number**.

If at any point the altitude becomes either **zero** or **negative**, print “crashed” and **end** **the program**. If by the end, the altitude is **positive**, however, print “got through safely. current altitude: {altitude}m”.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 300 up 3 down 7 up 5 | got through safely. current altitude: 301m |
| 50 up 2 down 20 up 3 down 36 up 1 down 2 up 2 | crashed |
| 11 up 12 down 20 down 3 down 36 down 1 down 2 down 2 | crashed |

## \* Ballistics Training

You are the anti-aircraft operator, trying to shoot down the airplane from the previous problem. You’ll be given instructions to get to the current coordinates of the plane. Shoot it down.

You will be given an array which holds 2 numbers – the **target** **X** and **Y** **coordinates** of the plane.

Afterwards, you will be given a second array, which holds a sequence of **left** / **right** / **up** / **down** commands and **numbers**. We start at position **{x=0, y=0}**. Manipulate the firing position in the following way:

* Up **increases** **y** by the next number.
* Down **decreases** **y** by the next number.
* Left **decreases x** by the next number.
* Right **increases x** by the next number.

After you process all the commands, print “firing at [{x}, {y}]”. After that, check if the **firing position** **coordinates** line up with the **target coordinates**. If they do, print “got 'em!”. If not – print “better luck next time...”

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 25 -3  right 18 down 6 left -7 up 3 | firing at [25, -3]  got 'em! |
| 150 33  right 108 down 4 left -11 up 3 right 30 up 33 right 2 | firing at [151, 32]  better luck next time... |

1. **Debugging Exercise: Tricky Strings**

The goal of this exercise is to practice **debugging techniques** in scenarios where a piece of code does not work correctly. Your task is to **pinpoint the 4 bugs** and **fix them** (without rewriting the entire code).

You can download the broken solution from [here](https://softuni.bg/downloads/svn/soft-tech/Jan-2017/Programming-Fundamentals-Extended-Jan-2017/05.%20Programming-Fundamentals-Debugging-and-Troubleshooting-Code/05.%20Programming-Fundamentals-Debugging-and-Troubleshooting-Code-Exercises-Broken-Solutions.zip).

**Problem Description**

You are given a **delimiter**. On the next line, you will receive a number **N.** On the next **N lines**, you will receive **strings** on each line. Your task is to **print** the strings, **separated** by the **delimiter**.

Note: the delimiter and strings could be ***anything***: whitespace and empty stringsare **acceptable input**!

**Examples**

|  |  |  |
| --- | --- | --- |
| **Input** | **Program Output (Wrong)** | **Expected Output (Correct)** |
| -  5  I  Am  Five  Strings  Long | I-IAm-IAmFive-IAmFiveStrings-IAmFiveStringsLong-IAmFiveStringsLong- | I-Am-Five-Strings-Long |
| \_  5  you  cannot  trick  me | you\_youcannot\_youcannot\_youcannottrick\_youcannottrickme\_youcannottrickme\_ | you\_cannot\_\_trick\_me |
| '  7  S  o  f  t  U  n  i | S'So'Sof'Soft'SoftU'SoftUn'SoftUni'SoftUni' | S'o'f't'U'n'i |

**Hints**

* Download the source code and get familiar with it
* Deal with poor code formatting - Remove unnecessary blank lines, indent the code properly
* Fix method parameters naming
* Give methods a proper name

1. **Debugging Exercise: Triangle Formations**

The goal of this exercise is to practice **debugging techniques** in scenarios where a piece of code does not work correctly. Your task is to **pinpoint the bugs** and **fix them** (without rewriting the entire code).

You can download the broken solution from [here](https://softuni.bg/downloads/svn/soft-tech/Jan-2017/Programming-Fundamentals-Extended-Jan-2017/05.%20Programming-Fundamentals-Debugging-and-Troubleshooting-Code/05.%20Programming-Fundamentals-Debugging-and-Troubleshooting-Code-Exercises-Broken-Solutions.zip).

**Problem Description**

You are given **3 integer numbers: a**, **b and c**, which will represent the **3 sides of a triangle**. Your task is to check whether the triangle is **valid**.  
  
If it is, print "**Triangle is valid.**".  
Otherwise print "**Invalid Triangle.**" and **end the program**.

If it is valid, you have to check if it is a **right triangle** (a2 + b2 == c2).  
If it is a right triangle, print "**Triangle has a right angle between sides a and b**", depending on which side forms a **right angle**. If the sides **b** and **c** form a right angle, print "**Triangle has a right angle between sides b and c**", and so on.

**Examples**

|  |  |  |
| --- | --- | --- |
| **Input** | **Program Output (Wrong)** | **Expected Output (Correct)** |
| 3  4  5 | Invalid Triangle.  Triangle has a right angle between sides a and b | Triangle is valid.  Triangle has a right angle between sides a and b |
| 5  5  5 | Invalid Triangle.  Triangle has no right angles. | Triangle is valid.  Triangle has no right angles |
| 3  1  1 | Invalid Triangle.  Triangle has no right angles. | Invalid Triangle. |

**Hints**

* Download the source code and get familiar with it
* Deal with poor code formatting - Remove unnecessary blank lines, indent the code properly
* Fix method parameters naming
* Give methods a proper name

1. **Debugging Exercise: Tetris**

The goal of this exercise is to practice **debugging techniques** in scenarios where a piece of code does not work correctly. Your task is to **pinpoint the bugs** and **fix them** (without rewriting the entire code).

You can download the broken solution from [here](https://softuni.bg/downloads/svn/soft-tech/Jan-2017/Programming-Fundamentals-Extended-Jan-2017/05.%20Programming-Fundamentals-Debugging-and-Troubleshooting-Code/05.%20Programming-Fundamentals-Debugging-and-Troubleshooting-Code-Exercises-Broken-Solutions.zip).

**Problem Description**

You will receive an integer **N**. On the next several lines, you will receive commands in the form of **directions** – **left**, **right**, **up** and **down**

Your task is to print the famous **Tetris** block – the **T-shape**, facing the given **direction**, with a **size** of **N**.

When you receive the command “**exit**”, you should stop the program execution.

**Examples**

|  |  |  |
| --- | --- | --- |
| **Input** | **Program Output (Wrong)** | **Expected Output (Correct)** |
| 1  left  right  exit | *(no output)* | **.\***  **\*\***  **.\***  **\*.**  **\*\***  **\*.** |
| 2  up  down  exit | \*\*  \*\*  \*\*  \*\*  \*\*  \*\*  \*\*  *(infinite loop)* | **..\*\*..**  **..\*\*..**  **\*\*\*\*\*\***  **\*\*\*\*\*\***  **\*\*\*\*\*\***  **\*\*\*\*\*\***  **..\*\*..**  **..\*\*..** |
| 3  right  left  exit | \*\*\*\*\*\*  \*\*\*\*\*\*  .\*.  ......  ......  \*\*\*\*\*\*  \*\*\*\*\*\* | **\*\*\*...**  **\*\*\*...**  **\*\*\*...**  **\*\*\*\*\*\***  **\*\*\*\*\*\***  **\*\*\*\*\*\***  **\*\*\*...**  **\*\*\*...**  **\*\*\*...**  **...\*\*\***  **...\*\*\***  **...\*\*\***  **\*\*\*\*\*\***  **\*\*\*\*\*\***  **\*\*\*\*\*\***  **...\*\*\***  **...\*\*\***  **...\*\*\*** |

**Hints**

* Download the source code and get familiar with it
* Deal with poor code formatting - Remove unnecessary blank lines, indent the code properly
* Fix method parameters naming
* Give methods a proper name

1. **Debugging Exercise: Mining Coins**

The goal of this exercise is to practice **debugging techniques** in scenarios where a piece of code does not work correctly. Your task is to **pinpoint the bugs** and **fix them** (without rewriting the entire code).

You can download the broken solution from [here](https://softuni.bg/downloads/svn/soft-tech/Jan-2017/Programming-Fundamentals-Extended-Jan-2017/05.%20Programming-Fundamentals-Debugging-and-Troubleshooting-Code/05.%20Programming-Fundamentals-Debugging-and-Troubleshooting-Code-Exercises-Broken-Solutions.zip).

**Problem Description**

In this problem you have to decrypt **encrypted data** and **calculate** its **value**. You will receive an **integer** N. On the next **N** lines, you will receive 3-digit integers.

The input integers will be **indexed** - starting from **1 to N**. The integers represent **characters** of an **encrypted message**.

You must take the **first** and the **last** digit and form a **2-digit number** from them, depending on the **current index** of the input number:

* if the index is an **odd** number – **subtract** the **middle digit** from the **2-digit number**.
* If the index is **even** number – **add** the **middle digit** to the **2-digit** number.

If the resulting number is an **English** **alphabet** **letter** (Lower or Uppercase) **append** it to a string. If it is not, **ignore** it.

After that, calculate the **sum** of the **3 digits** and **divide** it by **N**.

The result from this operation should be a **floating-point number**, **rounded** to the **7-th digit** after the decimal point.

Do the same for **all input integers** and **sum** the results. The sum is the **value** of the **decrypted data**.

The final result should be the **decrypted string** (the letters, appended together), and the **value**.

**Examples**

|  |  |  |
| --- | --- | --- |
| **Input** | **Program Output (Wrong)** | **Expected Output (Correct)** |
| 8  836  736  733  795  650  778  886  694 | Message:  Value: 15.25000000 | **Message: SOFTUNI**  **Value: 17.6250000** |
| 7  618  811  918  918  716  716  710 | Message: aa  Value: 12.71429000 | **Message:** **CRacKME**  **Value: 13.8571400** |

**Hints**

* Download the source code and get familiar with it
* Deal with poor code formatting - Remove unnecessary blank lines, indent the code properly
* Fix method parameters naming
* Give methods a proper name