# Exercises: Methods

You can check your solutions in **Judge system**: <https://judge.softuni.bg/Contests/3160/Methods>

## Grades

Write a method that **receives a grade** between **2.00** and **6.00** and **prints the corresponding grade in words**

* 2.00 – 2.99 - "Fail"
* 3.00 – 3.49 - "Poor"
* 3.50 – 4.49 - "Good"
* 4.50 – 5.49 - "Very good"
* 5.50 – 6.00 - "Excellent"

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 3.33 | Poor |
| 4.50 | Very good |
| 2.99 | Fail |

### Hints

1. Read the grade from the console and pass it to a method

Graphical user interface, text, application, chat or text message

Description automatically generated

1. Then create the method and make the if statements for each case

Text

Description automatically generated

## Sign of Integer Numbers

Create a method that prints the **sign** of an integer number **n**:

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 2 | The number 2 is positive. |
| -5 | The number -5 is negative. |
| 0 | The number 0 is zero. |

## Calculations

Write a program that receives a **string** on the first line (“add”, “multiply”, “subtract” or “divide”) and on the next **two lines** receives **two numbers**. Create **four methods** (for each calculation) and **invoke the right one** depending on the command. The method should also **print the result** (needs to be void).

### Example

|  |  |
| --- | --- |
| **Input** | **Output** |
| subtract  5  4 | 1 |
| divide  8  4 | 2 |

### Hints

1. **Read the command** on the first line and the **two numbers**, and then make an **if/switch statement** for each type of calculation:

Graphical user interface, text, application

Description automatically generated

1. Then create the **four methods** and **print** the result:

Text

Description automatically generated

## Printing Triangle

Create a method for **printing triangles** as shown below:

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 3 | 1  1 2  1 2 3  1 2  1 |
| 4 | 1  1 2  1 2 3  1 2 3 4  1 2 3  1 2  1 |

### Hints

1. After you read the input
2. Start by creating a method **for printing a single line** from a **given start** to a **given end**. Choose a **meaningful name** for it, describing its purpose:

Text, letter

Description automatically generated

1. Create another method for printing the whole triangle. Again choose a **meaningful name** for it, describing its purpose.
2. Think how you can use the **PrintLine()** method to solve the problem
3. After you spent some time thinking, you should have come to the conclusion that you will need **two loops**
4. In the first loop you can print the **first half** of the triangle:

A picture containing diagram

Description automatically generated

1. In the second loop you can **print the second half** of the triangle:

A picture containing chart

Description automatically generated

## Calculate Rectangle Area

Create a method that calculates and **returns** the [area](http://www.mathopenref.com/trianglearea.html) of a rectangle by given width and height:

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 3  4 | 12 |
| 6  2 | 12 |

### Hints

1. Read the input
2. Create a method, but this time **instead** of typing **"static void"** before its name, type **"static double"** as this will make it to **return a value of type double**:

A picture containing text

Description automatically generated

1. **Invoke** the method in the main and **save the return value in a new variable**:

Text

Description automatically generated

## Repeat String

Write a method that **receives a string** and a **repeat count** n (integer). The method should **return a new string** (the old one repeated n times).

### Example

|  |  |
| --- | --- |
| **Input** | **Output** |
| abc  3 | abcabcabc |
| String  2 | StringString |

### Hints

1. Firstly read the **string** and the repeat count **n**
2. Then create the **method** and **pass it the variables**

Graphical user interface, text, application

Description automatically generated

1. In the Main() **method**, print the result

## Math Power

Create a method that **calculates** and **returns** the value of a **number raised to a given power**:

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 2  8 | 256 |
| 3  4 | 81 |

### Hints

1. As usual, read the input
2. Create a **method** which will have **two parameters** - the **number** and the **power**, and will return a result of type **double**:

Graphical user interface, text, application

Description automatically generated

1. **Print** the result

## Greater of Two Values

Create a method **GetMax()** that **returns the greater** of two values (the values can be of type **int**, **char** or **string**)

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| int  2  16 | 16 |
| char  a  z | z |
| string  aaa  bbb | bbb |

## Multiply Evens by Odds

Create a program that **multiplies the sum** of **all even digits** of a number **by the sum of all odd digits** of the same number:

* Create a method called **GetMultipleOfEvenAndOdds()**
* Create a method **GetSumOfEvenDigits()**
* Create **GetSumOfOddDigits()**
* You may need to use **Math.Abs()** for negative numbers

### Examples

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Comment** |
| -12345 | 54 | Evens: 2 4  Odds: 1 3 5  Even sum: 6  Odd sum: 9  6 \* 9 = 54 |

## Math Operations

Write a method that receives **two number** and an **operator**, **calculates** the result and **returns** it. You will be given **three lines of input**. The first will be the **first number**, the second one will be the **operator** and the last one will be the **second number**. The possible operators are: ‘/’, ‘\*’, ‘+’, ‘-‘.

Print the result **rounded up to the second decimal point**.

### Example

|  |  |
| --- | --- |
| **Input** | **Output** |
| 5  \*  5 | 25 |
| 4  +  8 | 12 |

### Hint

1. **Read** the input and create a method that returns a **double** (the result of the operation)

Graphical user interface, text, application, email

Description automatically generated

## Smallest of Three Numbers

Write a method to **print the smallest of three integer numbers**. Use an **appropriate name** for the method.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 2  5  3 | 2 |
| 600  342  123 | 123 |
| 25  21  4 | 4 |

## Vowels Count

Write a method that receives a **single string** and prints the **count of the vowels**. Use an appropriate name for the method.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| SoftUni | 3 |
| Cats | 1 |
| JS | 0 |

## Characters in Range

Write a method that receives **two characters** and prints on a **single line** **all the characters between them** according to **ASCII** table.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| a  d | b c |
| #  : | $ % & ' ( ) \* + , - . / 0 1 2 3 4 5 6 7 8 9 |
| C  # | $ % & ' ( ) \* + , - . / 0 1 2 3 4 5 6 7 8 9 : ; < = > ? @ A B |

## Password Validator

Write a program that **checks** if a given **password is valid**. Password **rules** are:

* **6 – 10 characters (inclusive)**
* **Consists only of letters and digits**
* **Have at least 2 digits**

If a **password is valid** print "Password is valid". If it is **not valid**, for every unfulfilled rule print a message:

* **"Password must be between 6 and 10 characters"**
* **"Password must consist only of letters and digits"**
* **"Password must have at least 2 digits"**

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| logIn | Password must be between 6 and 10 characters  Password must have at least 2 digits |
| MyPass123 | Password is valid |
| Pa$s$s | Password must consist only of letters and digits  Password must have at least 2 digits |

### Hints

Write a **method** for each rule.

## Middle Characters

You will receive a **single string**. Write a method that prints the **middle character**. If the length of the string is **even** there are **two middle characters**.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| aString | r |
| someText | eT |
| 3245 | 24 |

## Factorial Division

Read **two integer numbers**. Calculate [factorial](https://en.wikipedia.org/wiki/Factorial) of each number. **Divide** the **first result** by the **second** and print the **division** formatted **to the second decimal point**.

### Examples

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Input** | **Output** |  | **Input** | **Output** |
| 5  2 | 60.00 |  | 6  2 | 360.00 |

## Palindrome Integers

A **palindrome** is a number which reads the same backward as forward, such as 323 or 1001. Write a program which **reads a positive integer numbers** until you receive "**End**". For each number print **whether the number is palindrome or not**.

### Examples

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Input** | **Output** |  | **Input** | **Output** |
| 123  323  421  121  END | false  true  false  true |  | 32  2  232  1010  END | false  true  true  false |

## Top Number

A **top number** is an integer that holds the following properties:

* Its **sum of digits is divisible by 8**, e.g. 8, 16, 88.
* Holds at least **one odd digit**, e.g. 232, 707, 87578.

Write a program to print **all top numbers** in the range [1…n].

### Examples

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Input** | **Output** |  | **Input** | **Output** |
| 50 | 17  35 |  | 100 | 17  35  53  71  79  97 |

## \* Array Manipulator

Trifon has finally become a junior developer and has received his first task. It’s about **manipulating an array of integers**. He is not quite happy about it, since he hates manipulating arrays. They are going to pay him a lot of money, though, and he is willing to give somebody half of it if to help him do his job. You, on the other hand, love arrays (and money) so you decide to try your luck.

The array may be manipulated by one of the following commands

* **exchange {index}** – splits the array **after** the given index, and exchanges the places of the two resulting sub-arrays. E.g. [1, 2, 3, 4, 5] -> exchange 2 -> result: [4, 5, 1, 2, 3]
  + If the index is outside the boundaries of the array, print “**Invalid index**”
* **max** **even/odd**– returns the **INDEX** of the max even/odd element -> [1, 4, 8, 2, 3] -> **max odd** -> print **4**
* **min** **even/odd** – returns the **INDEX** of the min even/odd element -> [1, 4, 8, 2, 3] -> **min even** > print **3**
  + If there are two or more equal **min/max** elements, return the index of the **rightmost** one
  + If a **min/max even/odd** element **cannot** be found, print **“No matches”**
* **first {count}** **even/odd**– returns the first {count} elements -> [1, 8, 2, 3] -> **first 2 even** -> print [**8, 2]**
* **last {count}** **even/odd** – returns the last {count} elements -> [1, 8, 2, 3] -> **last 2 odd** -> print [**1, 3]**
  + If the count is greater than the array length, print “**Invalid count**”
  + If there are **not** **enough** elements to satisfy the count, print as many as you can. If there are **zero** **even/odd** elements, print an empty array “[]”
* **end** – stop taking input and print the final state of the array

### Input

* The input data should be read from the console.
* On the first line, the initial array is received as a line of integers, separated by a single space
* On the next lines, until the command “**end**” is received, you will receive the array manipulation commands
* The input data will always be valid and in the format described. There is no need to check it explicitly.

### Output

* The output should be printed on the console.
* On a separate line, print the output of the corresponding command
* On the last line, print the final array in **square brackets** with its elements separated by a comma and a space
* See the examples below to get a better understanding of your task

### Constraints

* The **number of input lines** will be in the range [2 … 50].
* The **array elements** will be integers in the range [0 … 1000].
* The **number of elements** will be in the range [1 .. 50]
* The **split index** will be an integer in the range [-231 … 231 – 1]
* **first/last count** will be an integer in the range [1… 231 – 1]
* There will **not** be redundant whitespace anywhere in the input
* Allowed working time for your program: 0.1 seconds. Allowed memory: 16 MB.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 1 3 5 7 9  exchange 1  max odd  min even  first 2 odd  last 2 even  exchange 3  end | 2  No matches  [5, 7]  []  [3, 5, 7, 9, 1] |
| **Input** | **Output** |
| 1 10 100 1000  max even  first 5 even  exchange 10  min odd  exchange 0  max even  min even  end | 3  Invalid count  Invalid index  0  2  0  [10, 100, 1000, 1] |
| **Input** | **Output** |
| 1 10 100 1000  exchange 3  first 2 odd  last 4 odd  end | [1]  [1]  [1, 10, 100, 1000] |