# Exercises: Arrays

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

You can check your solutions here: <https://judge.softuni.bg/Contests/172/Arrays-and-Lists-Lab>.

# Arrays

## Day of Week

Enter a **day number** [1…7] and print the **day name** (in English) or “**Invalid Day!**”. Use an **array of strings**.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 1 | Monday |
| 2 | Tuesday |
| 7 | Sunday |
| 0 | Invalid Day! |

### Hints

* Use an **array of strings** holding the day names: {"Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", "Sunday"}.
* Print the element at index (**day-1**) when it is in the range [1…7] or “**Invalid Day!**” otherwise.

## Reverse an Array of Integers

Write a program to read **an array of integers**, **reverse** it and **print** its elements. The input consists of a **number** n (the number of elements) + n integers, each as a separate line. Print the output on a single line (space separated).

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| **3**  10  20  30 | 30 20 10 |
| **4**  -1  20  99  5 | 5 99 20 -1 |

### Hints

* First, read the number n.
* Allocate an array of n integers.
* Read the integers in a for-loop.
* Instead of reversing the array, you can just pass through the elements from the last (**n-1**) to the first (**0**) with a reverse for-loop.

## Last K Numbers Sums Sequence

Enter two integers **n** and **k**. Generate and print the following sequence of n elements:

* The first element is: **1**
* All other elements = sum of the previous **k** elements (if less than **k** are available, sum all of them)
* Example: n = **9**, k = **5** 🡪 **120** = 4 + 8 + 16 + 31 + 61

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 6  3 | 1 1 2 4 7 13 |
| 8  2 | 1 1 2 3 5 8 13 21 |
| 9  5 | 1 1 2 4 8 16 31 61 120 |

### Hints

* Use an **array of integers** to hold the sequence.
* Initially seq[0] = 1
* Use two nested loops:
  + Loop through all elements i = **1 …** n
  + Sum the elements i**-k …** i**-1**: seq[i] = sum(seq[i**-k …** i**-1**])

## Triple Sum

Write a program to read **an array of integers** and find all triples of elements **a**, **b** and **c**, such that **a** + **b** == **c** (where **a** stays left from **b**). Print “**No**” if no such triples exist.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 1 1 1 1 | No |
| 4 2 8 6 | 4 + 2 == 6  2 + 6 == 8 |
| 2 7 5 0 | 2 + 5 == 7  2 + 0 == 2  7 + 0 == 7  5 + 0 == 5 |
| 3 1 5 6 1 2 | 3 + 5 == 5  1 + 5 == 6  1 + 1 == 2  1 + 2 == 3  5 + 1 == 6  1 + 2 == 3 |

### Hints:

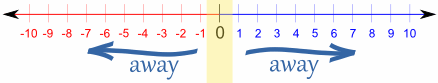
* Read the input numbers in array arr[].
* Use nested loops to generate all pairs {a, b}, such that 0 ≤ a < b < n.
* Check whether arr[] contains the sum arr[a] + arr[b].

## Rounding Numbers Away from Zero

Write a program to read **an array of real numbers** (space separated values), **round** them to the nearest integer in “**away from 0**” style and **print** the output as in the examples below.

Rounding in “[away from zero](https://www.mathsisfun.com/numbers/rounding-methods.html)” style means:

* To round to the nearest integer, e.g. 2.9 🡪 3; -1.75 🡪 -2
* In case the number is exactly in the middle of two integers (midpoint value), round it to the next integer which is away from the 0:



### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 0.9 1.5 2.4 2.5 3.14 | 0.9 => 1  1.5 => 2  2.4 => 2  2.5 => 3  3.14 => 3 |
| -5.01 -1.599 -2.5 -1.50 0 | -5.01 => -5  -1.599 => -2  -2.5 => -3  -1.50 => -2  0 => 0 |

### Hints:

* **Variant I**: Take the **absolute value** of each input number, add **0.5** and **truncate** the integral part. If the original number is negative, make the result also negative.
* **Variant II**: **Search in Internet** for **“rounding away from zero” + C#**. You should find a build-in C# method for rounding in many styles. Choose “away from zero” rounding.

## Reverse an Array of Strings

Write a program to read **an array of strings**, **reverse** it and **print** its elements. The input consists of a sequence of space separated strings. Print the output on a single line (space separated).

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| a b c d e | e d c b a |
| -1 hi ho w | w ho hi -1 |

### Hints

* Read the array of strings.
* **Exchange** the **first** element (at index 0) with the **last** element (at index n-1).
* **Exchange** the **second** element (at index 1) with the element **before the** **last** (at index n-2).
* Continue the same way until the middle of the array is reached.



* Another, shorter approach, is to use the .Reverse() extension method from “System.Linq”.

## Sum Arrays

Write a program that reads two **arrays of integers** and sums them. When the arrays are not of the same size, duplicate the smaller array a few times.

### Examples

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Comments** |
| 1 2 3 4  2 3 4 5 | 3 5 7 9 | 1 2 3 4 +  2 3 4 5 =  3 5 7 9 |
| 1 2 3 4 5  2 3 | 3 5 5 7 7 | 1 2 3 4 5 +  2 3 2 3 2 =  3 5 5 7 7 |
| 5 4 3  2 3 1 4 | 7 7 4 9 | 5 4 3 5 +  2 3 1 4 +  7 7 4 9 |

### Hints

* Assume the first array arr1 has len1 elements and the second arr2 has len2 elements.
* The result array will have max(len1, len2) elements.
* We sum array elements one by one (from the first to the last). To enable **rotating** (take the first element as next after the last), we use the position % length indexing: arr1[i % len1] and arr2[i % len2].

## Condense Array to Number

Write a program to read **an array of integers** and **condense** them by **summing** adjacent couples of elements until a **single integer** is obtained. For example, if we have 3 elements {2, 10, 3}, we sum the first two and the second two elements and obtain {2+10, 10+3} = {12, 13}, then we sum again all adjacent elements and obtain {12+13} = {25}.

### Examples

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Comments** |
| 2 10 3 | 25 | 2 10 3 🡪 2+10 10+3 🡪 12 13 🡪 12 + 13 🡪 25 |
| 5 0 4 1 2 | 35 | 5 0 4 1 2 🡪 5+0 0+4 4+1 1+2 🡪 5 4 5 3 🡪 5+4 4+5 5+3 🡪 9 9 8 🡪 9+9 9+8 🡪 18 17 🡪 18+17 🡪 35 |
| 1 | 1 | 1 is already condensed to number |

### Hints

While we have more than one element in the array nums[], repeat the following:

* Allocate a new array condensed[] of size nums.Length-1.
* Sum the numbers from nums[] to condensed[]:
  + condensed[i] = nums[i] + nums[i+1]
* nums[] = condensed[]

The process is illustrated below:





## Extract Middle 1, 2 or 3 Elements

Write a method to extract the **middle** **1**, **2** or **3** **elements** from array of **n** integers and **print** them.

* **n** = 1 -> **1** element
* even **n** -> **2** elements
* odd **n** -> **3** elements

Create a program that reads an **array of integers** (space separated values) and prints the middle elements in the format shown in the examples.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| **5** | { 5 } |
| 2 3 **8 1** 7 4 | { 8, 1 } |
| 1 2 **3 4 5** 6 7 | { 3, 4, 5} |
| 10 20 30 **40 50** 60 70 80 | { 40, 50 } |

### Hints

* Write different logic for each case (n = 1, even n, odd n)
* n = 1 🡪 take the first element
* odd n 🡪 take elements n/2-1, n/2, n/2+1
* even n 🡪 take elements n/2-1 and n/2

# Lists

## Remove Negatives and Reverse

Read a **list of integers**, **remove all negative numbers** from it and print the remaining elements in **reversed order**. In case of no elements left in the list, print “empty”.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 10 -5 7 9 -33 50 | 50 9 7 10 |
| 7 -2 -10 1 | 1 7 |
| -1 -2 -3 | empty |

### Hints

* Read a text line from the console, split it by space, parse the obtained items as integers and convert them to list of integers.
* Create a new empty list for the results.
* Scan the input list from the end to the beginning. Check each element and append all non-negative elements to the result list.
* Finally, print the results list (at a single line holding space-separated numbers).

## Append Lists

Write a program to **append several lists** of numbers.

* Lists are separated by ‘|’.
* Values are separated by spaces (‘ ’, one or several)
* Order the lists from the **last** to the **first**, and their values from **left** to **right**.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 1 2 3 |4 5 6 | 7 8 | 7 8 4 5 6 1 2 3 |
| 7 | 4 5|1 0| 2 5 |3 | 3 2 5 1 0 4 5 7 |
| 1| 4 5 6 7 | 8 9 | 8 9 4 5 6 7 1 |

### Hints

* Create a new empty list for the results.
* Split the input by ‘|’ into array of tokens.
* Pass through each of the obtained tokens from tight to left.
  + For each token, split it by space and append all non-empty tokens to the results.
* Print the results.

## Sum Adjacent Equal Numbers

Write a program to **sum all adjacent equal numbers** in a list of decimal numbers, starting from **left to right**.

* After two numbers are summed, the obtained result could be equal to some of its neighbors and should be summed as well (see the examples below).
* Always sum the **leftmost** two equal neighbors (if several couples of equal neighbors are available).

### Examples

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Explanation** |
| 3 3 6 1 | 12 1 | **3 3** 6 1 🡪 **6 6** 1 🡪 12 1 |
| 8 2 2 4 8 16 | 16 8 16 | 8 **2 2** 4 8 16 🡪 8 **4 4** 8 16 🡪 **8 8** 8 16 🡪 16 8 16 |
| 5 4 2 1 1 4 | 5 8 4 | 5 4 2 **1 1** 4 🡪 5 4 **2 2** 4 🡪 5 **4 4** 4 🡪 5 8 4 |

### Hints

1. Read the **input** and parse it to **list of numbers**.
2. Find the **leftmost** two **adjacent equal cells**.
3. **Replace** them with their **sum**.
4. **Repeat** (1) and (2) until no two equal adjacent cells survive.
5. **Print** the processed list of numbers.

## Split by Word Casing

Read a **text**, split it into words and distribute them into **3 lists**.

* **Lower-case words** like “programming”, “at” and “databases” – consist of lowercase letters only.
* **Upper-case words** like “PHP”, “JS” and “SQL” – consist of uppercase letters only.
* **Mixed-case words** like “C#”, “SoftUni” and “Java” – all others.

Use the following **separators** between the words: , ; : . ! ( ) " ' \ / [ ] space

Print the 3 lists as shown in the example below.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| Learn programming at SoftUni: Java, PHP, JS, HTML 5, CSS, Web, C#, SQL, databases, AJAX, etc. | Lower-case: programming, at, databases, etc  Mixed-case: Learn, SoftUni, Java, 5, Web, C#  Upper-case: PHP, JS, HTML, CSS, SQL, AJAX |

### Hints

* **Split** the input text using the above described **separators**.
* **Process** the obtained **list of words** one by one.
* Create 3 lists of words (initially empty): lowercase words, mixed-case words and uppercase words.
* Check each word and append it to one of the above 3 lists:
  + Count the **lowercase letters** and **uppercase letters**.
  + If all letters are **lowercase**, append the word to the lowercase list.
  + If all letters are **uppercase**, append the word to the uppercase list.
  + Otherwise the word is considered mixed-case 🡪 append it to the mixed-case list.
* Print the obtained 3 lists as shown in the example above.

## Sort Numbers

Read a **list of decimal numbers** and **sort** them in increasing order. Print the output as shown in the examples below.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 8 2 7 3 | 2 <= 3 <= 7 <= 8 |
| 2 4 -9 | -9 <= 2 <= 4 |

### Hints

* Use the built-in method List<T>.Sort().

## Square Numbers

Read a **list of integers** and **extract all square numbers** from it and print them in **descending order**. A **square number** is an integer which is the square of any integer. For example, 1, 4, 9, 16 are square numbers.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 3 **16 4** 5 6 8 **9** | 16 9 4 |
| 12 **1 9 4 16** 8 **25 49 16** | 49 25 16 16 9 4 1 |

### Hints

* To find out whether an integer is “**square number**”, check whether its square root is integer number (has no fractional part):
  + if (√num == (int)√num) …
* To order the results list in descending order use sorting by lambda function:
  + squareNums.Sort((a, b) => b.CompareTo(a));

## Count Numbers

Read a **list of integers** in range [0…1000] and **print them in ascending order** along with their **number of occurrences**.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 8 2 2 8 2 2 3 7 | 2 -> 4  3 -> 1  7 -> 1  8 -> 2 |
| 10 8 8 10 10 | 8 -> 2  10 -> 3 |

### Hints

Several algorithms can solve this problem:

* Use an **array** count[0…1000] to count in counts[x] the occurrences of each element x.
* **Sort** the numbers and count occurrences of each number.
* Use a dictionary counts<int, int> to count in counts[x] the occurrences of each element x.

#### Counting Occurrences Using Array

1. Read the input elements in array of integers nums[].
2. Assume max holds the largest element in nums[].
   * max = nums.Max()
3. Allocate an array counts[0 … max+1].
   * It will hold for each number x its number of occurrences counts[x].
4. **Scan** the input elements and for each element x increase counts[x].
   * For each value v in [0…max], print v -> count[v].
   * Skip all values which do not occur in nums[], i.e. counts[v] == 0.

This algorithm has a **serious drawback**:

* It depends on **mapping numbers to array indexes**.
* It will work well for input values in the range [0…1000].
* It will **not work** for very large and very small values, e.g. if the input holds -100 000 000 or 100 000 000.
* It will **not work** for real numbers, e.g. 3.14 or 2.5.

#### Counting Occurrences by After Sorting

1. Read the input elements in array of integers nums[]. Example: {8, 2, 2, 8, 2, 2, 3, 7}.
2. Sort nums[] in increasing order: {2, 2, 2, 2, 3, 7, 8, 8}. Now find all subsequences of equal numbers.
3. **Scan** the numbers from left to right. Count how many times each number occurs.
   * Start at count = 1.
   * While the next number on the right is **the same** as the current number, **increase** count and proceed to the next number.
   * When the next number on the right is **different** (or there is no next number), **print** the current number and its count.
   * Continue scanning from the next number on the right.

This algorithm will work correctly for real numbers and very large numbers. It does not depend on mapping numbers to array indexes.

#### Counting Occurrences with Dictionary

Dictionaries map some input **key** (e.g. string, integer or real number) to some **value**. Using a dictionary, we can map an input number x to a dictionary value dict[x] and apply the first occurrence counting algorithm. Learn more about dictionaries at <https://www.dotnetperls.com/dictionary>. In the next lesson you will learn how to use dictionaries.