**Homework: Advanced Syntax and Operations**

This document defines the homework assignments from the ["PHP Web Dev Basi HYPERLINK "https://softuni.bg/trainings/2163/php-web-development-basics-september2018"cs" Course @ SoftUni](https://softuni.bg/trainings/2163/php-web-development-basics-september2018).

You can check your solutions here: <https://judge.softuni.bg/Contests/1252/Advanced-Syntax-and-Operations-Exercise>

**Problem 1. Index of Letters**

Write a program that creates an array containing all letters from the alphabet (**a**-**z**). Read a lowercase word from the console and print the **index of each of its letters from the letters array**.

**Examples**

|  |  |
| --- | --- |
| **Input** | **Output** |
| abcz | a -> 0  b -> 1  c -> 2  z -> 25 |
| softuni | s -> 18  o -> 14  f -> 5  t -> 19  u -> 20  n -> 13  i -> 8 |

**Problem 2. 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 **array\_reverse()** extension method from **PHP Build-in methods.**

**Problem 3. Sum Reversed Numbers**

To “**rotate** an array on the right” means to move its last element first: {1, 2, 3} {3, 1, 2}.

Write a program to read an array of **n** **integers** (space separated on a single line) and sum the obtained arrays after each rotation as shown below.

**Examples**

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Comments** |
| 123 234 12 | 774 | 321 + 432 + 21 = 774 |
| 12 12 34 84 66 12 | 220 | 21 + 21 + 43+ 48 + 66 + 21 = 220 |
| 12 12 12 | 63 | 21 + 21 + 21 = 63 |

**Problem 4. Most Frequent Number**

Write a program that finds the **most frequent number** in a given sequence of numbers.

* Numbers will be in the range [0…65535].
* In case of multiple numbers with the same maximal frequency, print the leftmost of them.

**Examples**

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Output** |
| **4** 1 1 **4** 2 3 **4 4** 1 2 **4** 9 3 | 4 | The number **4** is the most frequent (occurs 5 times) |
| **2 2 2 2** 1 **2 2 2** | 2 | The number **2** is the most frequent (occurs 7 times) |
| **7 7 7** 0 2 2 2 0 10 10 10 | 7 | The numbers **2**, **7** and **10** have the same maximal frequence (each occurs 3 times). The leftmost of them is **7**. |

**Problem 5. Max Sequence of Equal Elements**

Write a program that finds the **longest sequence of equal elements** in an array of integers. If several longest sequences exist, print the leftmost one.

**Examples**

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

**Hints**

* Start with the sequence that consists of the first element: **start**=**0**, **len**=**1**.
* Scan the elements from left to right, starting at the second element: **pos**=**1**…**n-1**.
* At each step compare the current element with the element on the left.
* Same value you have found a sequence longer by one **len**++.
* Different value start a new sequence from the current element: **start**=**pos**, **len**=**1**.
* After each step remember the sequence it is found to be longest at the moment: **bestStart**=**start**, **bestLen**=**len**.
* Finally, print the longest sequence by using **bestStart** and **bestLen**.

**Problem 6. Max Sequence of Increasing Elements**

Write a program that finds the **longest increasing subsequence** in an array of integers. The longest increasing subsequence is a **portion of the array** (subsequence) that is strongly **increasing** and has the **longest possible length**. If several such subsequences exist, find the left most of them.

**Examples**

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

**Hints**

* Use the same algorithm like in the previous problem (Max Sequence of Equal Elements).

**Problem 7. 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])**

**Problem 8. Rotate and Sum**

To “**rotate** an array on the right” means to move its last element first: {1, 2, 3} {3, 1, 2}.

Write a program to read an array of **n** **integers** (space separated on a single line) and an integer **k**, rotate the array right **k** **times** and sum the obtained arrays after each rotation as shown below.

**Examples**

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Comments** |
| 3 2 4 -1  2 | 3 2 5 6 | rotated1[] = -1 3 2 4  rotated2[] = 4 -1 3 2  sum[] = 3 2 5 6 |
| 1 2 3  1 | 3 1 2 | rotated1[] = 3 1 2  sum[] = 3 1 2 |
| 1 2 3 4 5  3 | 12 10 8 6 9 | rotated1[] = 5 1 2 3 4  rotated2[] = 4 5 1 2 3  rotated3[] = 3 4 5 1 2  sum[] = 12 10 8 6 9 |

**Hints**

* After **r** rotations the element at position **i** goes to position **(i + r) % n**.
* The **sum[]** array can be calculated by two nested loops: for **r** = **1** … **k**; for **i** = **0** … **n-1**.

**Problem 9. 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:





**Part II: Multidimensional Arrays**

**Problem 10. Sum Matrix Elements**

Write program that **read a matrix** from console and print:

* Count of **rows**
* Count of **columns**
* Sum of all **matrix’s elements**

On first line you will get matrix sizes in format **[rows, columns]**

**Examples**

|  |  |
| --- | --- |
| **Input** | **Output** |
| 3, 6  7, 1, 3, 3, 2, 1 1, 3, 9, 8, 5, 6 4, 6, 7, 9, 1, 0 | 3  6  76 |

**Hints**

* On next **[rows]** lines you will get elements for each column separated with coma and whitespace
* Try to use only **foreach** for printing

**Problem 11. Maximum sum of 2x2 submatrix**

Write a program that **read a matrix** from console. Then find biggest sum of **2x2 submatrix** and print it to console.

On first line you will get matrix sizes in format **rows, columns.**

One next **rows** lines you will get elements for each **column** separated with coma.

Print **biggest top-left** square, which you find and sum of it's elements.

**Examples**

|  |  |
| --- | --- |
| **Input** | **Output** |
| 3, 6  7, 1, 3, 3, 2, 1 1, 3, 9, 8, 5, 6 4, 6, 7, 9, 1, 0 | 9 8  7 9  33 |
| 2, 4  10, 11, 12, 13  14, 15, 16, 17 | 12 13  16 17  58 |

**Hints**

* If you find some max squares, print top-left one

**Problem 12. Maximal Sum**

Write a program, which reads a rectangular matrix of integers of size of **r** rows by **c** columns. Find in the matrix a **platform of size 3 x 3** **with a maximal sum**.

**Input**

* The first line holds the number of rows **r** and the number of columns **c**.
* The next **r** lines hold the elements of the matrix.
* Constraints: 3 ≤ **r**, **c** ≤ 1000.

**Output**

* At the **first line** of the output **print the sum**.
* At the next line **print the platform itself**.
* If **several** platforms of equal sum exist, print the one that is located in the **most upper-left** position.

**Examples**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Input** | **Output** |  | **Input** | **Output** |  | **Input** | **Output** |
| 4 4  5 **6 2 8**  3 **1 9 5**  8 **1 6 9**  1 5 3 4 | Sum = 47  6 2 8  1 9 5  1 6 9 |  | 5 6  1 2 4 8 9 6  2 4 1 3 4 2  2 7 **9 9 9** 7  8 6 **9 9 9** 6  9 5 **9 9 9** 9 | Sum = 81  9 9 9  9 9 9  9 9 9 | 4 6  **1 1 1** 1 1 1  **1 1 1** 1 1 1  **1 1 2** 1 1 1  1 1 1 1 1 1 | Sum = 10  1 1 1  1 1 1  1 1 2 |

**Part III: Associative Arrays**

**Problem 13. Phonebook**

Write a program that receives some info from the console about **people** and their **phone numbers**. Each **entry** should have just **one name** and **one number** (both of them strings).

On each line you will receive some of the following commands:

* **A {name} {phone}** – adds entry to the phonebook. In case of trying to add a name that is already in the phonebook you should change the existing phone number with the new one provided.
* **S {name}** – searches for a contact by given name and prints it in format "**{name} -> {number}**". In case the contact isn't found, print "**Contact {name} does not exist.**".
* **END** – stop receiving more commands.

**Examples**

|  |  |
| --- | --- |
| **Input** | **Output** |
| A Nakov 0888080808  S Mariika  S Nakov  END | Contact Mariika does not exist.  Nakov -> 0888080808 |
| A Nakov +359888001122  A RoYaL(Ivan) 666  A Gero 5559393  A Simo 02/987665544  S Simo  S simo  S RoYaL  S RoYaL(Ivan)  END | Simo -> 02/987665544  Contact simo does not exist.  Contact RoYaL does not exist.  RoYaL(Ivan) -> 666 |
| A Misho +359883123  A Misho 02/3123  S Misho  END | Misho -> 02/3123 |

**Hints**

* **Parse the commands** by splitting by space. Execute the commands until “**END**” is reached.
* Store the **phonebook entries** in **associative array** with key **{name}** and value **{phone number}**.

**Problem 14. Phonebook Upgrade**

**Add functionality to the phonebook** from the previous task to **print all contacts ordered lexicographically** when receive the command “**ListAll**”.

**Examples**

|  |  |
| --- | --- |
| **Input** | **Output** |
| A Nakov +359888001122  A RoYaL(Ivan) 666  A Gero 5559393  A Simo 02/987665544  ListAll  END | Gero -> 5559393  Nakov -> +359888001122  RoYaL(Ivan) -> 666  Simo -> 02/987665544 |

**Hints**

* **Variant I (slower):** Sort all entries in the dictionary by key and print them.
* **Variant II (faster):** Keep the entries in more appropriate data structure that will keep them in sorted order for better performance.

**Problem 15. Mixed Phones**

You will be given several phone entries, in the form of strings in format:

**firstElement : secondElement**

The first element is usually the person’s name, and the second one – his phone number. The phone number consists ONLY of digits, while the person’s name can consist of any ASCII characters.

Sometimes the phone operator gets distracted by the Minesweeper she plays all day, and gives you first the phone, and then the name. **e.g. : 0888888888 : Pesho**. You must store them correctly, even in those cases.

When you receive the command “**Over**”, you are to **print all names** you’ve stored with their phones. The names must be printed in **alphabetical order**.

**Examples**

|  |  |
| --- | --- |
| **Input** | **Output** |
| 14284124 : Alex  Gosho : 088423123  Ivan : 412048192  123123123 : Nanyo  Pesho : 150925812  Over | Alex -> 14284124  Gosho -> 88423123  Ivan -> 412048192  Nanyo -> 123123123  Pesho -> 150925812 |
| Ivan : 13213456  GeorgeThe2nd : 131313  Johnny : 5556322312  Donald : 3212  Over | Donald -> 3212  GeorgeThe2nd -> 131313  Ivan -> 13213456  Johnny -> 5556322312 |

**Problem 16. \*Points Counter**

Write a program, which receives data about a **team**, **player** and **points**.

**Input**

You can have **two** types of input:

* **{TEAM}|{Player}|{points}**
* **{Player}|{TEAM}|{points}**

The **team** **name** will always consist of **uppercase** **Latin** **letters** and the player name will always **start** with **uppercase** **Latin** **letter** and **all** **other** **letters** will be **lowercase**.

The **team** and **player** names **might** be **polluted** with some **prohibited** symbols (‘**@**’, ‘**%**’, ‘&’, ‘**$**’ and ‘**\***’). You have to **delete** **every** **occurrence** of these symbols in **every** **team** and **player** name.

Then, calculate every team’s total score. Every **team’s** **total** **score** equals to the **total** **sum** of the **points** made by **every** **player** **in** the **team**.

**Output**

When you receive the command “**Result**”, print **all** teams, ordered in **descending order** by the **points** they made and the player with **most** points in the **team** in the following format:

|  |
| --- |
| **{teamName} => {totalSumOfPoints}**  **Most points scored by: {nameOfThePlayer}** |

In case of **repeating** **player** **names** for one team, **save** the **value**, which is **received** **last**.

**Constraints**

* The **team** **names** will be **at least** **2** characters **long**
* The **points** for each **player** will be in the interval **[1…100]**

**Examples**

|  |  |
| --- | --- |
| **Input** | **Output** |
| LA|Bryant|70  L%@A|Odom|67  Bry\*%ant|LA|71  James|%CAVA@@LIE$$$RS|54  C@art%er|GR%%IZZ%%LIE@S@@@|49  Anthony|KNICKS|11  UTAH|Jo%%%%hn$$so@@n|24  S@@PU\*R\*S$|Ga\*\*\*so\*\*l|32  Jone@@@@s|KNICKS|5  Result | LA => 138  Most points scored by Bryant  CAVALIERS => 54  Most points scored by James  GRIZZLIES => 49  Most points scored by Carter  SPURS => 32  Most points scored by Gasol  UTAH => 24  Most points scored by Johnson  KNICKS => 16  Most points scored by Anthony |
| SO@@@FTU%\*NI|P\*&@esho|30  SO$$FT\*UNI|Gos%@ho|42  PAARTHURNAX|Maria|35  S\*OFT$$$UNI|Iv\*\*\*\*@an|3  L@u\*b%o@|HE\*\*\*RO@@ES|11  Result | SOFTUNI => 75  Most points scored by Gosho  PAARTHURNAX => 35  Most points scored by Maria  HEROES => 11  Most points scored by Lubo |

**Problem 17. User Logins**

Write a program that receives a **list** of **username-password pairs** in the format “**username -> password**”. If there’s already a user with that username, **replace their password**. After you receive the command “**login**”, **login requests** start coming in, using the **same format**. Your task is to print the status of user login, using different messages as per the conditions below:

* If the password matches with the user’s password, print “**username: logged in successfully**”.
* If the user doesn’t exist, or the password doesn’t match the user, print “**username: login failed**”.

When you receive the command “**end**”, print the **count** of **unsuccessful** login attempts, using the format “**unsuccessful login attempts: count**”.

**Examples**

|  |  |
| --- | --- |
| **Input** | **Output** |
| ivan\_ivanov -> java123  pesh0 -> qwerty  stamat4e -> 111111  princess\_penka -> MyPrince  **login**  pesh0 -> qwertt  ivan\_ivanov -> java123  stamat4e -> 111112  princess\_penka -> MyPrince  end | pesh0: login failed  ivan\_ivanov: logged in successfully  stamat4e: login failed  princess\_penka: logged in successfully  unsuccessful login attempts: 2 |
| johnny\_bravo05 -> woahMama  **login**  johnny\_bravo06 -> woahMama  johnny\_bravo05 -> woahmama  johnny\_bravo05 -> WoahMama  johnny\_bravo05 -> wohMama  johnny\_bravo05 -> woahMama  end | johnny\_bravo06: login failed  johnny\_bravo05: login failed  johnny\_bravo05: login failed  johnny\_bravo05: login failed  johnny\_bravo05: logged in successfully  unsuccessful login attempts: 4 |
| walter\_white58 -> iamthedanger  krazy\_8 -> ese  jesseABQ -> bword  **login**  krazy\_8 -> ese  krazy\_8 -> ese  jesse -> bword  jesseabq -> bword  walter\_white58 -> IAmTheDanger  walter\_white58 -> iamthedanger  end | krazy\_8: logged in successfully  krazy\_8: logged in successfully  jesse: login failed  jesseabq: login failed  walter\_white58: login failed  walter\_white58: logged in successfully  unsuccessful login attempts: 3 |

**Problem 18. Exam Shopping**

A supermarket has **products** which have **quantities**. Your task is to stock the shop before **Exam Sunday**. Until you receive the command “**shopping time**”, **add** the various **products** to the shop’s **inventory**, keeping track of their **quantity** (for inventory purposes). When you receive the aforementioned command, the students start pouring in before the exam and **buy** various **products**.

The format for **stocking** a product is: “**stock $product $quantity**”.

The format for **buying** a product is: “**buy $product $quantity**”.

If a student **tries** to buy a product, which **doesn’t exist**, print “**$product doesn't exist**”. If it does exist, but it’s **out of stock**, print “**$product out of stock**”. If a student tries buying **more** than the quantity of that product, sell them **all** **the** **quantity** of the product (the product is left out of stock, **don’t** print anything).

When you receive the command “**exam time**”, your task is to **print** the **remaining** inventory in the following format: “**product -> quantity**”. If a product is out of stock, **do not** print it.

**Examples**

|  |  |
| --- | --- |
| **Input** | **Output** |
| stock Boca\_Cola 10  stock Boca\_Cola 10  stock Kay's 3  stock Kay's 2  shopping time  buy Boca\_Cola 5  buy Kay's 5  exam time | Boca\_Cola -> 15 |
| stock Lobster\_Energy 20  stock Loreni 30  stock Loreni 30  stock Lobster\_Energy 10  shopping time  exam time | Lobster\_Energy -> 30  Loreni -> 60 |
| stock Boca\_Cola 16  stock Kay's\_Chips 33  stock Lobster\_Energy 60  stock Boca\_Cola 4  stock Loreni 15  stock Loreni 15  stock Loreni 15  stock Loreni 15  shopping time  buy Boca\_Bola 2  buy Lobster\_Energy 20  buy Boca\_Cola 1  buy Boba\_Bola 12  exam time | Boca\_Bola doesn't exist  Boba\_Bola doesn't exist  Boca\_Cola -> 19  Kay's\_Chips -> 33  Lobster\_Energy -> 40  Loreni -> 60 |

**Problem 19. Filter Base**

You have been tasked to sort out a database full of information about employees. You will be given several input lines containing information in one of the following formats:

* **name -> age**
* **name -> salary**
* **name -> position**

As you see you have 2 parameters. There can be only 3 cases of input:  
If the second parameter is an **integer**, you must store it as **name** and **age**.

If the second parameter is a **floating-point number**, you must store it as **name** and **salary**.

If the second parameter is a **string**, you must store it as **name** and **position**.

You must read input lines, then parse and store the information from them, **until** you receive the command   
“**filter base**”. When you receive that command, the **input sequence** of **employee information** should **end**.

On the last line of input, you will receive a string, which can either be “**Age**”, “**Salary**” or “**Position**”. Depending on the case, you must **print all entries** which **have been stored** as **name** and **age**, **name** and **salary**, or **name** and **position**.

In case, the given last line is “**Age**”, you must print every employee, stored with its **age** in the following format:

**Name: name1  
Age: age1  
====================  
Name: name2  
. . .**

In case, the given last line is “**Salary**”, you must print every employee, stored with its **salary** in the following format:

**Name: name1  
Salary: salary1  
====================  
Name: name2  
. . .**

**NOTE:** The **Salary** must be **formatted** to **2 digits** after the decimal point.

In case, the given last line is “**Position**”, you must print every employee, stored with its **position** in the following format:

**Name: name1  
Position: position1  
====================  
Name: name2  
. . .**

**NOTE:** Every entry is **separated** with the **other**, with a **string** of **20 character** **‘=**’.

There is **NO** particular order of printing – the data must be printed in **order** of **input**.

**Examples**

|  |  |
| --- | --- |
| **Input** | **Output** |
| Isacc -> 34  Peter -> CEO  Isacc -> 4500.054321  George -> Cleaner  John -> Security  Nina -> Secretary  filter base  Position | Name: Peter  Position: CEO  ====================  Name: George  Position: Cleaner  ====================  Name: John  Position: Security  ====================  Name: Nina  Position: Secretary  ==================== |
| Ivan -> Chistach  Pesho -> Ohrana  Pesho -> 1323.0456  Ivan -> 732.404  Georgi -> 21  Georgi -> 21.02  filter base  Salary | Name: Pesho  Salary: 1323.05  ====================  Name: Ivan  Salary: 732.40  ====================  Name: Georgi  Salary: 21.02  ==================== |