# Homework: PHP Associative Arrays and Functions

This document defines the homework assignments from the ["PHP Web Dev Basics" Course @ Software University](https://softuni.bg/trainings/1746/php-web-developmentbasics-september-2017).

## Problem 1. Letter Repetition

You will be given a **single string**, containing **random ASCII character**. Your task is to **print every character**, and how **many times** it has been used in the **string**.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| aaabbaaabbbccc | a -> 6  b -> 5  c -> 3 |
| The quick brown fox jumps over the lazy dog | T -> 1  h -> 2  e -> 3  -> 8  q -> 1  u -> 2  i -> 1  c -> 1  k -> 1  b -> 1  r -> 2  o -> 4  w -> 1  n -> 1  f -> 1  x -> 1  j -> 1  m -> 1  p -> 1  s -> 1  v -> 1  t -> 1  l -> 1  a -> 1  z -> 1  y -> 1  d -> 1  g -> 1 |

## Word Mapping

Write a PHP program **WordMapper.php** that takes a text from a textarea **with attribute name=”input”** and prints each word and the number of times it occurs in the text. The search should be **case-insensitive**. The result should be printed as an **HTML table with border attribute set to 2**. If you get 0/100 in judge check in details the zero test.

|  |  |
| --- | --- |
| **Input** | **Output** |
| The quick brows fox.!!! jumped over..// the lazy dog.! |  |
|  |

## Count Real Numbers

Read a **list of real numbers with fgets(STDIN)** and **print them in ascending order** along with their **number of occurrences**.

### Examples

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Input** | **Output** |  | **Input** | **Output** |  | **Input** | **Output** |
| 8 2.5 2.5 8 2.5 | 2.5 -> 3 times  8 -> 2 times | 1.5 5 1.5 3 | 1.5 -> 2 times  3 -> 1 times  5 -> 1 times | -2 0.33 0.33 2 | -2 -> 1 times  0.33 -> 2 times  2 -> 1 times |

### Hints

* Pass through each input number num and increase counts[num] (when num exists in the array) or assign counts[num] = 1 (when num does not exist in the dictionary).
* Pass through all numbers num in the array and print the number num and its count of occurrences counts[num].
* Use ksort() to sort associative array by keys.

## Coloring Texts

Write a PHP program **TextColorer.php** that takes a text from a textfield **with attribute name=”input”** , colors each character according to its ASCII value and prints the result. If the ASCII value of a character is odd, the **character** should be printed in **blue**. If it’s even, it should be printed in **red**.For the colors **use the <font> tag.**

|  |  |
| --- | --- |
| **Input** | **Output** |
| What a wonderful world! | W h a t a w o n d e r f u l w o r l d ! |
|  |

## Navigation Builder

Write a PHP program **Navigation.php** that takes data from several input fields and builds 3 navigation bars. The input fields are **categories with attribute name=”categories”**, **tags with attribute name=”tags”** and **months with attribute name=”tags”**. The first navigation bar should contain a list of the **categories**, the second navigation bar – a list of the **tags** and the third should contain the **months**. The entries in the input strings will be separated by a **comma** and **space** "**,** ". When you print your result don’t forget to **use** **<h2> tag for “Categories”, “Tags”, and “Months”.** Semantic HTML is required.

|  |  |  |
| --- | --- | --- |
| **Input** | | **Output** |
| categories | Knitting, Cycling, Swimming, Dancing |  |
| tags | person, free time, love, peace, protest |
| months | April, May, June, July |
|  | |

## Problem 6. 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 7. 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 8. 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 |

### Hints

* **Parse the commands** by splitting by space. The first element is the username, while the third element is the password.
* Store the **user entries** in **Array(string => string)** with key **username** and value **password**.

## Problem 9. 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  ==================== |

## \*Template format

Write a program that receives data about a quiz and prints it formatted as an XML document. The data comes as pairs of question-answer entries. The format of the document should be as follows:

|  |
| --- |
| **XML** |
| <?xml version="1.0" encoding="UTF-8"?>  <quiz>  <question>  {question text}  </question>  <answer>  {answer text}  </answer>  </quiz> |

The **input** comes as a string in which the questions and answers will be separated by “, “.

The **output** should be printed on the console.

### Examples

|  |
| --- |
| **Input** |
| Who was the forty-second president of the U.S.A.?, William Jefferson Clinton |
| **Output** |
| <?xml version="1.0" encoding="UTF-8"?>  <quiz>  <question>  Who was the forty-second president of the U.S.A.?  </question>  <answer>  William Jefferson Clinton  </answer>  </quiz> |

|  |
| --- |
| **Input** |
| Dry ice is a frozen form of which gas?, Carbon Dioxide, What is the brightest star in the night sky?, Sirius |
| **Output** |
| <?xml version="1.0" encoding="UTF-8"?>  <quiz>  <question>  Dry ice is a frozen form of which gas?  </question>  <answer>  Carbon Dioxide  </answer>  <question>  What is the brightest star in the night sky?  </question>  <answer>  Sirius  </answer>  </quiz> |

## Cooking by Numbers

Write a program that receives a number and a list of five operations. Perform the operations in sequence by starting with the input number and using the result of every operation as starting point for the next. Print the result of every operation in order. The operations can be one of the following:

* chop – divide the number by two
* dice – square root of number
* spice – add 1 to number
* bake – multiply number by 3
* fillet – subtract 20% from number

The **input** comes in 2 lines. On the first line you will receive your starting point and must be parsed to a number. On the second line you will receive 5 commands separated by “, “ each one will be the name of the operation to be performed.

The **output** should be printed on the console. Do not **round** the result.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 32  chop, chop, chop, chop, chop | 16 8 4 2 1 |

|  |  |
| --- | --- |
| **Input** | **Output** |
| 9  dice, spice, chop, bake, fillet | 3  4  2  6  4.8 |

## Modify Average

Write a program that modifies a number until the average value of all of its digits is **higher than 5**. In order to modify the number, your program should append a **9** to the end of the number, when the average value of all of its digits is **higher than 5** the program should stop appending. If the number’s average value of all of its digits is already **higher than 5**, no appending should be done.

The **input** is a single number received as a string.

The **output** should consist of a single number - the final modified number which has an average value of all of its digits **higher than 5**. The **output** should be printed on the console.

### Constraints

* **The input number will consist of no more than 6 digits.**
* **The input will be a valid number (there will be no leading zeroes).**

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 101 | 1019999 |
| 5835 | 5835 |

## Validity Checker

Write a program that receives two points in the format **x1, y1, x2, y2** and checks if the distances between each point and the start of the cartesian coordinate system (0, 0) and between the points themselves is **valid**. A distance between two points is considered **valid**, if it is an **integer value**. In case a distance is valid write "**{x1, y1} to {x2, y2} is valid"**, in case the distance is invalid write **"{x1, y1} to {x2, y2} is invalid"**.

The order of comparisons should always be first **{x1, y1}** to **{0, 0}**, then **{x2, y2}** to **{0, 0}** and finally **{x1, y1}** to **{x2, y2}**.

The **input** consists of one string in which the coordinates are separated by “, “(look at the examples).

For each comparison print on the **output** either "**{x1, y1} to {x2, y2} is valid"** if the distance between them is valid, or **"{x1, y1} to {x2, y2} is invalid"**- if it’s invalid.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 3, 0, 0, 4 | {3, 0} to {0, 0} is valid  {0, 4} to {0, 0} is valid  {3, 0} to {0, 4} is valid |
| 2, 1, 1, 1 | {2, 1} to {0, 0} is invalid  {1, 1} to {0, 0} is invalid  {2, 1} to {1, 1} is valid |

## Treasure Locator

You will be given a series of coordinates, leading to a buried treasure. Use the map to the right to write a program that locates on which island it is. After you find where all the treasure chests are, compose a list and print it on the console. If a chest is not on any of the islands, print “On the bottom of the ocean” to inform your treasure-hunting team to bring diving gear. If the location is on the shore (border) of the island, it’s still considered to lie inside.

The **input** comes as a string of variable number of elements separated by “, “ that must be parsed to numbers. Each pair is the coordinates to a buried treasure chest.

The **output** is a list of the locations of every treasure chest, either the name of an island or “On the bottom of the ocean”, printed on the console.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 4, 2, 1.5, 6.5, 1, 3 | On the bottom of the ocean  Tonga  Tuvalu |
| 6, 4 | Samoa |

## Trip Length

You will be given the coordinates of 3 points on a 2D plane. Write a program that finds the two shortest segments that connect them (without going back to the starting point). When determining the listing order, use the order with the lowest numerical value (see the figure in the hints for more information).

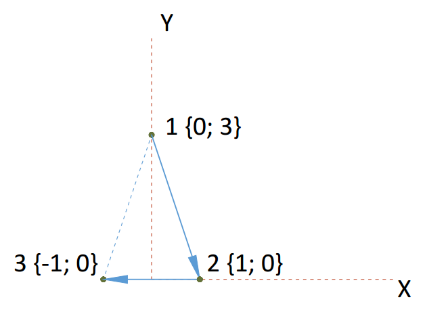
The **input** comes as a string with 6 elements separated by “, “ that must be parsed to numbers. The order is **x1, y1, x2, y2, x3, y3**.

The **output** is the return value of your program as a string, representing the order in which the three points must be visited and the final distance between them. See the examples for more info.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 0, 0, 2, 0, 4, 0 | 1->2->3: 4 |

|  |  |
| --- | --- |
| **Input** | **Output** |
| 5, 1, 1, 1, 5, 4 | 2->1->3: 7 |



|  |  |
| --- | --- |
| **Input** | **Output** |
| -1, -2, 3.5, 0, 0, 2 | 1->3->2: 8.154234499766936 |

### Hints

You can find the horizontal and vertical offset between two points by calculating the difference between their coordinates. Use Pythagoras’ theorem to find the distance.

If more than one shortest paths exist, choose the one with lowest numerical value. For instance, in the figure on the right, 1🡪2🡪3 is the same distance as 3🡪2🡪1, but we chose to start at 1, since it’s lower than 3. When choosing the second point, we encounter the same issue – 1🡪3🡪2 would be the same as 1🡪2🡪3, but we chose to visit 2 first, because it’s lower than 3.

## \*\*Radio Crystals

It’s time to put your skills to work for the war effort – creating management software for a radio transmitter factory. Radios require a finely tuned quartz crystal in order to operate at the correct frequency. The resource used to produce them is quartz ore that comes in big chunks and needs to undergo several processes, before it reaches the desired properties.

You need to write a program that monitors the current thickness of the crystal and recommends the next procedure that will bring it closer to the desired frequency. To reduce waste and the time it takes to make each crystal your program needs to complete the process with the least number of operations. Each operation takes the same amount of time, but since they are done at different parts of the factory, the crystals have to be transported and thoroughly washed every time an operation different from the previous must be performed, so this must also be taken into account. When determining the order, always attempt to start from the operation that removes the largest amount of material.

The different operations you can perform are the following:

* Cut – cuts the crystal in 4
* Lap – removes 20% of the crystal’s thickness
* Grind – removes 20 microns of thickness
* Etch – removes 2 microns of thickness
* X-ray – increases the thickness of the crystal by 1 micron; this operation can only be done once!
* Transporting and washing – removes any imperfections smaller than 1 micron (round down the number); do this after every batch of operations that remove material

At the beginning of your program, you will receive a number representing the desired final thickness and a series of numbers, representing the thickness of crystal ore in microns. Process each chunk and print to the console the order of operations and number of times they need to be repeated to bring them to the desired thickness.

The **input** comes as a string with a variable number of elements separated by “, “ that must be parsed to numbers. The first number is the target thickness and all following numbers are the thickness of different chunks of quartz ore.

The **output** is the order of operation and how many times they are repeated, every operation on a new line. See the examples for more information.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 1375, 50000 | Processing chunk 50000 microns  Cut x2  Transporting and washing  Lap x3  Transporting and washing  Grind x11  Transporting and washing  Etch x3  Transporting and washing  X-ray x1  Finished crystal 1375 microns |

#### Explanation

The operation that would remove the most material is always cutting – it removes three quarters of the chunk. Starting from 50000, if we perform it twice, we bring the chunk down to 3125. If we cut again, the chunk will be 781.25, which is less than the desired thickness, so we move to the next operation, but we first round down the number (transporting & washing). Next, we lap the chunk – after three operations, the crystal reaches 1600 microns. One more lapping would take it to 1280, so we move on to the next operation instead. We do the same check with grinding, and finally by etching 2 times, the crystal has reached 1376 microns, which is one more than desired. We don’t have an operation which only takes away 1 micron, so instead we etch once more to get to 1374, wash and then x-ray to add 1 micron, which brings us to the desired thickness.

|  |  |
| --- | --- |
| **Input** | **Output** |
| 1000, 4000, 8100 | Processing chunk 4000 microns  Cut x1  Transporting and washing  Finished crystal 1000 microns  Processing chunk 8100 microns  Cut x1  Transporting and washing  Lap x3  Transporting and washing  Grind x1  Transporting and washing  Etch x8  Transporting and washing  Finished crystal 1000 microns |

## \*\*DNA Helix

Write a program that prints a DNA helix with length, specified by the user. The helix has a repeating structure, but the symbol in the chain follows the sequence ATCGTTAGGG. See the examples for more information.

The **input** comes as a single string element that must be parsed to a number. It represents the length of the required helix.

The **output** is the completed structure, printed on the console.

### Examples

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Input** | **Output** |  | **Input** | **Output** |
| 4 | \*\*AT\*\* \*C--G\* T----T \*A--G\* | 10 | \*\*AT\*\*  \*C--G\*  T----T  \*A--G\*  \*\*GG\*\*  \*A--T\*  C----G  \*T--T\*  \*\*AG\*\*  \*G--G\* |

## 11.\* Super Calculator

Write a program that **reads a command** from the console and **executes it**. Your program should read commands until **finally** is given as a command. After that you will be given one command again but this time **instead of receiving numbers you should use the results of all the previous command** if they are enough for the command. (if you have to multiply and have only 1 number you should do nothing). If you have **more numbers than required** you should **repeat the command as many times as possible** and after each time save **the result as the last number of the sequence of results** before it and remove from the sequence the numbers used.(if given multiply and have 4 numbers you should multiply the first 2, remove them and save the result as last number and then repeat) .If the command **requires one number** you should **repeat it with all the numbers once for each** and save the result. (if given “root” you should take the root of all numbers once each and save the results). If after **“finally”** you are given command **and while executing it throws error,** after the error you should **return the original sequence of results, then you will be given another command and you should try to execute it** with the original sequence of results. The commands are as follows:

* **sum**– after you read this command you will receive 2 more lines of strings representing two numbers. You should sum them and save the resulting number.
* **multiply** – after you read this command you will receive 2 more lines of strings representing two numbers. You should multiply them and save the resulting number.
* **divide** – after you read this command you will receive 2 more lines of strings representing two numbers. You should divide them and save the resulting number. If the second number in the equation is 0 you should throw Exception **“Division by zero exception”.**
* **subtract** – after you read this command you will receive 2 more lines of strings representing two numbers. You should sum them and save the resulting number.
* **factorial** – after you read this command you will receive 1 line of string representing one number. You should return the factorial of that number.
* **root** – after you read this command you will receive 1 line of string representing one number. You should return the square root of that number. Note that **if the number is negative you should throw Exception** "Can't take the root of negative number".
* **power -** – after you read this command you will receive 2 more lines of strings representing two numbers. You should sum them and save the resulting number.
* **absolute -** after you read this command you will receive 1 line of string representing one number. You should return the modulus of that number (it’s absolute value).
* **pythagorean -** after you read this command you will receive 2 more lines of strings representing two numbers. You should use the Pythagorean theorem and return and save the resulting number.
* **triangleArea -** after you read this command you will receive 3 more lines of strings representing three numbers. You should use the Heron’s formula to calculate the area of a triangle with the three numbers as its sides and return and save the resulting number. If you come to result “NAN” you should throw new Exception "Can't take the root of negative number".
* **quadratic -** after you read this command you will receive 3 more lines of strings representing three numbers. You should calculate a quadratic equation (ax^2 – bx – c) where the first number is “a”, the second “b” and the third “c” and return and save the resulting number. **If “a” is 0** you should **throw new Exception “Division by zero exception”.**

The **input** comes one line at a time with a command or string representing a number.

The **output** is the final result. If it has more than 1 number the numbers should be separated by “, “.

### Examples

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Comments** |
| multiply  2  2  sum  1  1  finally  multiply | 8 | multiply= 2 \* 2 = 4  sum =1 + 1 = 2  result = [4,2]  multiply 4 \* 2 = 8 |
| divide  100  0  root  -10  quadratic  1  10  20  finally  absolute | Caught exception: Division by zero.  Caught exception: Can't take the root of a negative number  10 | We cant divide 100/0 so we throw new exception and result is empty  We can’t take the root ofa negative number so we throw new exception and result is still empty  The result of the quadratic equasion is -10  result = [-10]  The absolute value of -10 is 10. |
| subtract  10  5  power  10  2  finally  triangleArea | 5, 100 | We subtract 5 from 10 and the result is 5.  We take 10 to the power of 2 and the result is 100.  result = [5, 100]  Since we need 3 numbers and have only 2 we do nothing. |
| factorial  5  divide  0  10  pythagorean  4  3  finally  sum | 125 | The factorial of 5 is 120.  Division of 0 with 10 is possible and the result is 0  Pythagorean with 4 and 3 is 5.  result = [120, 0, 5]  The sum of all numbers in result is 125. |
| multiply  10  10  subtract  10  50  divide  10  -1  finally  multiply | 40000 | 10 \* 10 = 100  10-50 = -40  10/-1 = -10  result = [100, -40, -10]  first 100 \*-40 = -4000  result = [-10, -4000]  since we have the required count of numbers we repeat  -10 \* -4000 = 40000  result = [40000] |