# Exercises: Functions, Objects and Classes

This document defines a set of tasks to be done as a part of the [PHP Web Development Basics - 2018](https://softuni.bg/trainings/2163/php-web-development-basics-september2018#lesson-9663).

You can check your solutions here: <https://judge.softuni.bg/Contests/467/Functions-Objects-and-Classes-Exercises>.

# Part I: Functions

## 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.

### 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 |

## \*\* 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.”.**
* **subtract** – after you read this command you will receive 2 more lines of strings representing two numbers. You should subtract the second from the first and then 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 a negative number**".
* **power -** – after you read this command you will receive 2 more lines of strings representing two numbers. You should return the first to the power of the second 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 a 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. ” .** If you have **no real answers** to the equation you should **return 0**. If you have 2 you should **return theirs sum**.

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 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] |

## \*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\* |

# Part II: Objects and Classes

## Last Digit Name

Write a class **Number** that will hold an integer number. Write a **method** in the class that returns the **English name** of the last digit of the given number. Write a program that reads an integer and prints the returned value from this method.

### Examples

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Input** | **Output** |  | **Input** | **Output** |
| 1024 | four |  | 512 | two |

## Number in Reversed Order

Write a class **DecimalNumber** that has a method that **prints all its digits** in **reversed order**.

### Examples

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Input** | **Output** |  | **Input** | **Output** |
| 256 | 652 |  | 1.12 | 21.1 |

## Opinion Poll

Using the Person class, write a program that reads from the console **N** lines of personal information and then prints all people whose **age** is **more than 30** years, **sorted in alphabetical order**.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 3  Pesho 12  Stamat 31  Ivan 48 | Ivan - 48  Stamat - 31 |
| 5  Nikolai 33  Yordan 88  Tosho 22  Lyubo 44  Stanislav 11 | Lyubo - 44  Nikolai - 33  Yordan - 88 |

## Company Roster

Define a class **Employee** that holds the following information: **name, salary, position, department, email** and **age**. The **name, salary**, **position** and **department** are **mandatory** while the rest are **optional**.

Your task is to write a program which takes **N** lines of employees from the console and calculates the department with the highest average salary and prints for each employee in that department his **name, salary, email and age** – **sorted by salary in descending order**. If an employee **doesn’t have** an **email** – in place of that field you should print “**n/a**” instead, if he doesn’t have an **age** – print “**-1**” instead. The **salary** should be printed to **two decimal places** after the seperator.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 4  Pesho 120.00 Dev Development [pesho@abv.bg](mailto:pesho@abv.bg) 28  Toncho 333.33 Manager Marketing 33  Ivan 840.20 ProjectLeader Development [ivan@ivan.com](mailto:ivan@ivan.com)  Gosho 0.20 Freeloader Nowhere 18 | Highest Average Salary: Development  Ivan 840.20 [ivan@ivan.com](mailto:ivan@ivan.com) -1  Pesho 120.00 [pesho@abv.bg](mailto:pesho@abv.bg) 28 |
| 6  Stanimir 496.37 Temp Coding [stancho@yahoo.com](mailto:stancho@yahoo.com)  Yovcho 610.13 Manager Sales  Toshko 609.99 Manager Sales [toshko@abv.bg](mailto:toshko@abv.bg) 44  Venci 0.02 Director BeerDrinking [beer@beer.br](mailto:beer@beer.br) 23  Andrei 700.00 Director Coding  Popeye 13.3333 Sailor SpinachGroup [popeye@pop.ey](mailto:popeye@pop.ey) | Highest Average Salary: Sales  Yovcho 610.13 n/a -1  Toshko 609.99 [toshko@abv.bg](mailto:toshko@abv.bg) 44 |

## \*Speed Racing

Your task is to implement a program that keeps track of cars and their fuel and supports methods for moving the cars. Define a class **Car** that keeps track of a car’s **Model, fuel amount, fuel cost for 1 kilometer** and **distance traveled**. A Car’s Model is **unique** - there will never be 2 cars with the same model.

On the first line of the input you will receive a number **N** – the number of cars you need to track, on each of the next **N** lines you will receive information for a car in the following format “<**Model> <FuelAmount> <FuelCostFor1km>**”, all **cars start at 0 kilometers traveled**.

After the **N** lines until the command “**End**” is received, you will receive a commands in the following format “**Drive <CarModel> <amountOfKm>**”, implement a method in the **Car** class to calculate whether or not a car can move that distance, if it can the car’s **fuel amount** should be **reduced** by the amount of used fuel and its **distance traveled** should be increased by the amount of kilometers traveled, otherwise the car should not move (Its fuel amount and distance traveled should stay the same) and you should print on the console “**Insufficient fuel for the drive**”. After the “**End**” command is received, print each car and its current fuel amount and distance traveled in the format “**<Model> <fuelAmount> <distanceTraveled>**”, where the fuel amount should be printed to **two decimal places** after the separator.

### Note

Use double precision of two decimal digits after the floating point for calculations. Output two decimal digits after the floating point. Round up for this task.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 2  AudiA4 23 0.3  BMW-M2 45 0.42  Drive BMW-M2 56  Drive AudiA4 5  Drive AudiA4 13  End | AudiA4 17.60 18  BMW-M2 21.48 56 |
| 3  AudiA4 18 0.34  BMW-M2 33 0.41  Ferrari-488Spider 50 0.47  Drive Ferrari-488Spider 97  Drive Ferrari-488Spider 35  Drive AudiA4 85  Drive AudiA4 50  End | Insufficient fuel for the drive  Insufficient fuel for the drive  AudiA4 1.00 50  BMW-M2 33.00 0  Ferrari-488Spider 4.41 97 |

## \*Raw data

You are the owner of a courier company and want to make a system for tracking your cars and their cargo. Define a class **Car** that holds information about **model, engine, cargo** and a **collection of exactly 4 tires**. The engine, cargo and tire **should be separate classes**, create a constructor that receives all information about the Car and creates and initializes its inner components (engine, cargo and tires).

On the first line of input you will receive a number **N** - the amount of cars you have, on each of the next **N** lines you will receive information about a car in the format “**<Model> <EngineSpeed> <EnginePower> <CargoWeight> <CargoType> <Tire1Pressure> <Tire1Age> <Tire2Pressure> <Tire2Age> <Tire3Pressure> <Tire3Age> <Tire4Pressure> <Tire4Age>”** where the speed, power, weight and tire age are **integers**, tire pressure is a **double.**

After the **N** lines you will receive a single line with one of 2 commands “**fragile**” or “**flamable**” , if the command is “**fragile**” print all cars whose **Cargo Type is “fragile”** with a tire whose **pressure is** **< 1**, if the command is “**flamable**” print all cars whose **Cargo Type is “flamable”** and have **Engine Power > 250**. The cars should be printed in order of appearing in the input.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 2  ChevroletAstro 200 180 1000 fragile 1.3 1 1.5 2 1.4 2 1.7 4  Citroen2CV 190 165 1200 fragile 0.9 3 0.85 2 0.95 2 1.1 1  fragile | Citroen2CV |
| 4  ChevroletExpress 215 255 1200 flamable 2.5 1 2.4 2 2.7 1 2.8 1  ChevroletAstro 210 230 1000 flamable 2 1 1.9 2 1.7 3 2.1 1  DaciaDokker 230 275 1400 flamable 2.2 1 2.3 1 2.4 1 2 1  Citroen2CV 190 165 1200 fragile 0.8 3 0.85 2 0.7 5 0.95 2  flamable | ChevroletExpress  DaciaDokker |

## Car Salesman

Define two classes **Car** and **Engine.** A **Car** has a **model, engine, weight** and **color**. An Engine has **model**, **power, displacement** and **efficiency**. A Car’s **weight** and **color** and its Engine’s **displacements** and **efficiency** are **optional**.

On the first line you will read a number **N** which will specify how many lines of engines you will receive, on each of the next **N** lines you will receive information about an **Engine** in the following format “<**Model> <Power> <Displacement> <Efficiency>**”. After the lines with engines, on the next line you will receive a number **M** – specifying the number of Cars that will follow, on each of the next **M** lines information about a **Car** will follow in the following format “<**Model> <Engine> <Weight> <Color>**”, where the engine in the format will be the **model of an existing** **Engine**. When creating the object for a **Car**, you should keep a **reference to the real engine** in it, instead of just the engine’s model, note that the optional properties **might be missing** from the formats.

Your task is to print each car (in the order you received them) and its information in the format defined bellow, if any of the optional fields has not been given print “**n/a**” in its place instead:

**<CarModel>:  
 <EngineModel>:  
 Power: <EnginePower>  
 Displacement: <EngineDisplacement>  
 Efficiency: <EngineEfficiency>  
 Weight: <CarWeight>  
 Color: <CarColor>**

### Bonus\*

Define the magic method \_\_toString() to have a reusable way of displaying the objects.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 2  V8-101 220 50  V4-33 140 28 B  3  FordFocus V4-33 1300 Silver  FordMustang V8-101  VolkswagenGolf V4-33 Orange | FordFocus:  V4-33:  Power: 140  Displacement: 28  Efficiency: B  Weight: 1300  Color: Silver  FordMustang:  V8-101:  Power: 220  Displacement: 50  Efficiency: n/a  Weight: n/a  Color: n/a  VolkswagenGolf:  V4-33:  Power: 140  Displacement: 28  Efficiency: B  Weight: n/a  Color: Orange |
| 4  DSL-10 280 B  V7-55 200 35  DSL-13 305 55 A+  V7-54 190 30 D  4  FordMondeo DSL-13 Purple  VolkswagenPolo V7-54 1200 Yellow  VolkswagenPassat DSL-10 1375 Blue  FordFusion DSL-13 | FordMondeo:  DSL-13:  Power: 305  Displacement: 55  Efficiency: A+  Weight: n/a  Color: Purple  VolkswagenPolo:  V7-54:  Power: 190  Displacement: 30  Efficiency: D  Weight: 1200  Color: Yellow  VolkswagenPassat:  DSL-10:  Power: 280  Displacement: n/a  Efficiency: B  Weight: 1375  Color: Blue  FordFusion:  DSL-13:  Power: 305  Displacement: 55  Efficiency: A+  Weight: n/a  Color: n/a |

## Pokemon Trainer

You wanna be the very best pokemon trainer, like no one ever was, so you set out to catch pokemon. Define a class **Trainer** and a class **Pokemon**. Trainers have a **name**, **number of badges** and a **collection of pokemon**, **Pokemon** have a **name**, an **element** and **health**, all values are **mandatory**. Every Trainer **starts with 0 badges**.

From the console you will receive an unknown number of lines until you receive the command “**Tournament**”, each line will carry information about a pokemon and the trainer who caught it in the format “<**TrainerName> <PokemonName> <PokemonElement> <PokemonHealth>”** where **TrainerName** is the name of the Trainer who caught the pokemon, names are **unique** there cannot be 2 trainers with the same name. After receiving the command “**Tournament**” an unknown number of lines containing one of three elements “**Fire**”, “**Water**”, “**Electricity**” will follow until the command “**End**” is received. For every command you must check if a trainer has atleast 1 pokemon with the given element, if he does he receives 1 badge, otherwise all his pokemon **lose 10 health**, if a pokemon falls **to 0 or less health he dies** and must be deleted from the trainer’s collection. After the command “**End**” is received you should print all trainers **sorted by the amount of badges they have in descending order** (if two trainers have the same amount of badges they should be sorted by order of appearance in the input)in the format “<**TrainerName> <Badges> <NumberOfPokemon>**”.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| Pesho Charizard Fire 100  Gosho Squirtle Water 38  Pesho Pikachu Electricity 10  Tournament  Fire  Electricity  End | Pesho 2 2  Gosho 0 1 |
| Stamat Blastoise Water 18  Nasko Pikachu Electricity 22  Jicata Kadabra Psychic 90  Tournament  Fire  Electricity  Fire  End | Nasko 1 1  Stamat 0 0  Jicata 0 1 |

## Google

Google is always watching you, so it should come as no surprise that they know everything about you (even your pokemon collection), since you’re really good at writing classes Google asked you to design a Class that can hold all the information they need for people.

From the console you will receive an unkown amount of lines until the command “**End**” is read, on each of those lines there will be information about a person in one of the following formats:

* “**<Name> company <companyName> <department> <salary>**”
* “**<Name> pokemon <pokemonName> <pokemonType>”**
* “**<Name> parents <parentName> <parentBirthday>**”
* “**<Name> children <childName> <childBirthday>**”
* “**<Name> car <carModel> <carSpeed>**”

You should structure all information about a person in a class with nested subclasses. People’s names are **unique** - there won’t be 2 people with the same name, a person can also have **only 1** **company** and **car**, but can have **multiple** **parents, chidlren** and **pokemon**. After the command “**End**” is received on the next line you will receive a single name, you should print all information about that person. Note that information can change during the input, for instance if we receive multiple lines which specify a person’s company, only the **last one** should be the one remembered. The salary must be formated to **two decimal places** after the seperator.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| PeshoPeshev company PeshInc Management 1000.00  TonchoTonchev car Trabant 30  PeshoPeshev pokemon Pikachu Electricity  PeshoPeshev parents PoshoPeshev 22/02/1920  TonchoTonchev pokemon Electrode Electricity  End  TonchoTonchev | TonchoTonchev  Company:  Car:  Trabant 30  Pokemon:  Electrode Electricity  Parents:  Children: |
| JelioJelev pokemon Onyx Rock  JelioJelev parents JeleJelev 13/03/1933  GoshoGoshev pokemon Moltres Fire  JelioJelev company JeleInc Jelior 777.77  JelioJelev children PudingJelev 01/01/2001  StamatStamatov pokemon Blastoise Water  JelioJelev car AudiA4 180  JelioJelev pokemon Charizard Fire  End  JelioJelev | JelioJelev  Company:  JeleInc Jelior 777.77  Car:  AudiA4 180  Pokemon:  Onyx Rock  Charizard Fire  Parents:  JeleJelev 13/03/1933  Children:  PudingJelev 01/01/2001 |

### Bonus\*

Define the magic method \_\_toString() to have a reusable way of displaying the objects.

## Oldest Family Member

Create class **Person** with fields **name** and **age**. Create a class **Family**. The class should have **list of people**, method for adding members (**void addMember(Person member)**) and a method returning the oldest family member(**Person getOldestMember())**. Write a program that reads name and age for **N** people and **adds them to the family**. Then **print** the **name** and **age** of the oldest member.

If you’ve defined the class correctly, the test should pass.

### Examples

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Input** | **Output** |  | **Input** | **Output** |
| 3  Pesho 3  Gosho 4  Annie 5 | Annie 5 |  | 5  Steve 10  Christopher 15  Annie 4  Ivan 35  Maria 34 | Ivan 35 |

## Car

Create a class **Car**. Every car has a **speed (km/h), fuel (liters)** and **fuel economy (L/100km)** (given in the same order on the first line). They should be stored in the class. Your task is to create a program which executes one of the commands:

* **Travel <distance>** – makes the car travel the specified <distance>

If you are given a distance which you don't have enough fuel to travel, just go as far as you can.

* **Refuel <liters>** – refuels the car with the specified <fuel>
* **Distance** – gets the total travel distance
* **Time** – get the total travel time
* **Fuel** – gets the remaining fuel
* **END** – stops the program

### Output

Print the total distance traveled, total travel time and fuel left at the end of the trip as in the Example below.

### Note

Round values up to one decimal digit after the decimal point, applies for **kilometers** and **liters**.

Show only minutes, discarding the seconds. For Example 2 minutes 40 seconds and 2 minutes 20 seconds all become 2 minutes.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 100 20 20  Travel 100  Distance  Time  Fuel  END | Total Distance: 100.0  Total Time: 1 hours and 0 minutes  Fuel left: 0.0 liters |

## Date Modifier

Create a class **DateModifier** which stores the difference of the days between two Dates. It should have a method which takes two String parameters representing a date as Strings and **calculates the difference in the days between them.**

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 1992 05 31  2016 06 17 | 8783 |
| 2016 05 31  2016 04 19 | 42 |

### Hint

Use the **DateTime** class.