|  |  |
| --- | --- |
| **Exercise** | JS.1 |
| **App** | **Part1Exercise** |
| **Purpose** | Try to keep track of variable environment and execution contexts. |
| **Description** | The **script.js** file contains a bit of code to perform some slightly convoluted calculations. |
| **Steps** | Before actually running the code, see if you can figure out what the expected output should be, i.e. what the values of **res1**, **res** 2 and **res3** are when they are printed out to the console. |

|  |  |
| --- | --- |
| **Exercise** | JS.2 |
| **App** | **Part2Exercise** |
| **Purpose** | Try to work with object building |
| **Description** | Imagine that we are taking the first small steps into building a game. In the game, there will be “participants”. A participant has a name, and a number of “hit points”. There will be two types of participants:   * **Hero**: also has a “role” (like e.g. Wizard or Hunter), a “level” (from 1 and up), and a number of gold coins (a numeric value) * **Beast**: also has a “value”, which is a numeric value |
| **Steps** | Write JS code to enable creation of **Hero** and **Beast** objects, given the above spe­ci­fications. You can use any style you prefer, also including elements from ES6. Remember to test your code a bit as well ☺. |

|  |  |
| --- | --- |
| **Exercise** | JS.3 |
| **App** | **Part3Exercise** |
| **Purpose** | Work with array processing methods like **map**, **filter** and **reduce** |
| **Description** | The **script.js** file contains a number of small exercises, which should be solved using **map**, **filter** and **reduce** |
| **Steps** | Open **script.js**, and try to solve exercise #1 to #7 |

|  |  |
| --- | --- |
| **Exercise** | JS.RolePlay |
| **App** | **RolePlay** |
| **Purpose** | Try to implement a very small role-play game in JavaScript |
| **Description** | In JS.2, we took some small steps towards building a small role-play game in JavaScript. In this exercise, we aim at implementing a very simple logic for two game participants fighting against each other.  More specifically, try to extend the **Hero** class with the below pro­perties and functions:   * A “health points” property * A method for “dealing damage”. This method should return a numeric value. * A method for “receiving damage”, which will cause the health points of the Hero to be reduced. * A method for deciding if the Hero is dead or not, dead being defined as having zero or less health points   A match between two Heroes should follow this simple logic:  While both Heroes are alive:   1. First Hero deals damage; the dealt damage is received by the second Hero. 2. Second Hero (if still alive) deals damage; the dealt damage is received by the first Hero. |
| **Steps** | 1. Extend the Hero class – in whatever way your prefer to define it – to include the functionalities described above. 2. Define and use a function that can execute a match between two Heroes. You can test this by doing simple **console.log** calls in the JS code. 3. Once the game logic works, try to create a simple GUI for the game, by adding HTML to the **index.html** file. The GUI should contain four elements:    * A button for letting the first Hero attack the second Hero.    * A button for letting the second Hero attack the first Hero.    * A button for resetting the game to an initial state.    * A status text describing the current state of the game (how many health points do the Heroes have left, etc..) |