Exam set 2 – May 2017

Andrea Valente

Index:

Exercise 1

1.1 2 points

1.2 3 points

1.3 2 points

1.4 3 points

=> 10 points

Exercise 2

2.1 2 points

2.2 2 points

2.3 3 points

=> 7 points

Exercise 3

3.1 5 points

3.2 5 points

3.3 3 points

=> 13 points

Exercise 4

4.1 12 points

4.2 8 points

=> 20 points

total = 50 points

Time 2 hours, use any helping material to solve them.

Exercise 1  
==========

Look in the file pizza.html

There is a constructor function for the Pizza data-type.

In this program:

- A pizza has a name, a price and it can be organic or not.

- A pizza can provide its own description, with a toString method.

What you have to do:

1) Finish implementing the Pizza constructor, so that each Pizza object

actually has the right attributes.

To be sure that your job is done, when you are done the assertions in the file, in section:

// 1) testing the Pizza constructor

should pass correctly, without errors.

2) Finish implementing the toString function, so that it

returns a string describing the current pizza.

For example, a pizza with name 'Abc' and price 123, and organic, will be described as:

'Pizza \*Abc\*, price 123kr, organic'

for non-organic pizza, like name 'Xyz' and price 666, and not organic, the description is:

'Pizza \*Xyz\*, price 666kr'

To be sure that your job is done, when you are done the assertions in the file, in section:

// 2) testing the toString method

should pass correctly, without errors.

3) I created an array with 4 pizza, then I saved it in the localstorage.

You should load the array aCloneOfTheTable from localstorage, so it becomes a clone of

the tableOrder array.

To be sure that your job is done, when you are done the assertions in the file, in section:

// 3) testing an array of pizzas

should pass correctly, without errors.

4) Print the price of each pizza in the array. Use a for-loop.

To be sure that your job is done, when you are done the

assertions in the file, in section:

// 4) print the price of each pizza in the array

should pass correctly, without errors.

The image finalScreenshot.png shows how the page should look like, when your have solved all

the above points (see below).

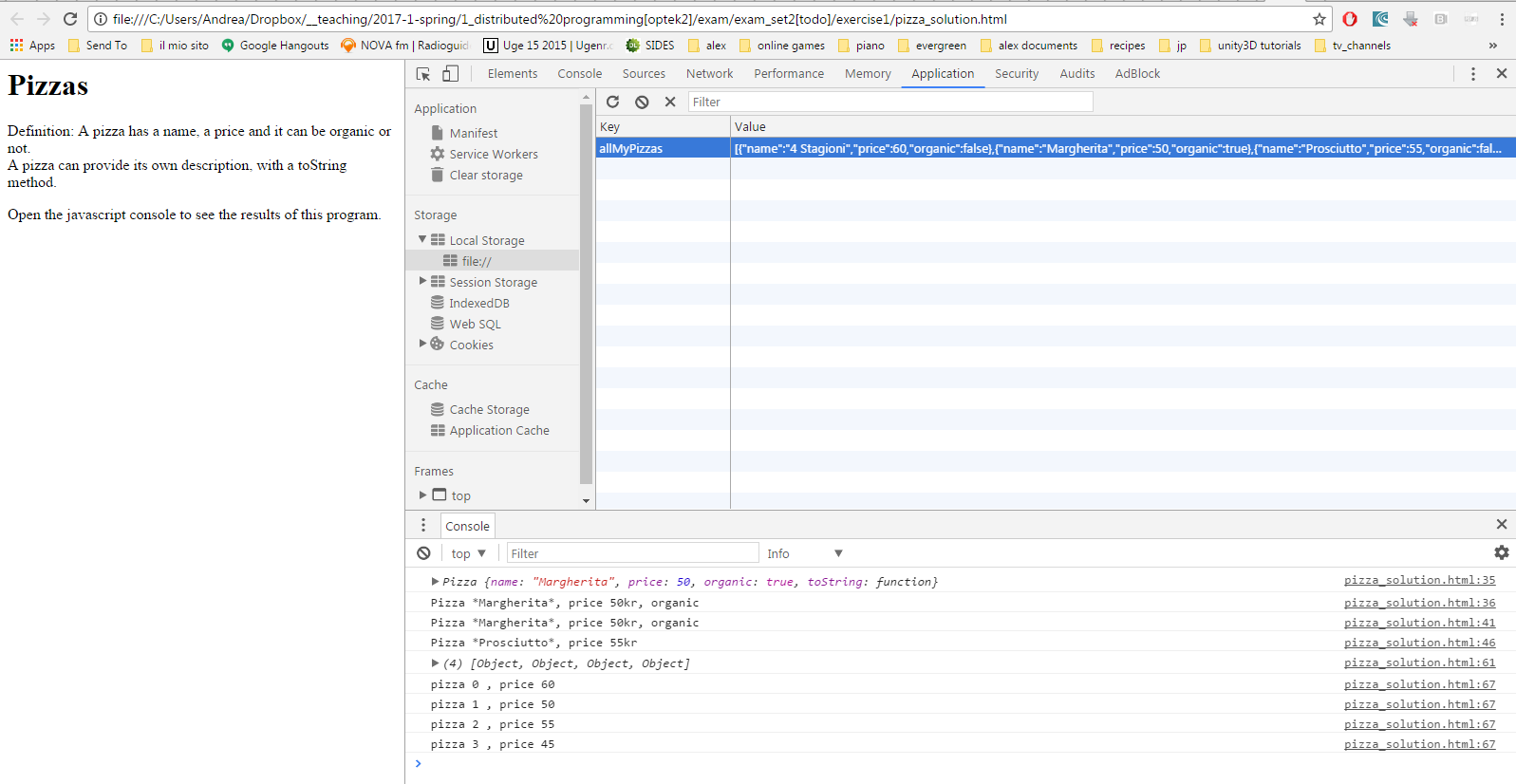


Figure 1 Image finalScreen.png

Exercise 2

==========

Work with a responsive page. In the code I'm using a Foundation grid.

The text in the example is taken from the book Postsingular by R. Rucker.

1) Change the HTML page so that the paragraph with title 'From Chpt 2: "Nant Day"'

only occupies 10 columns on a large screen, and 11 on a small screen.

2) Change again the HTML page. Add a new DIV with a picture of the cover of the book.

Give it a title like 'Cover' using headline h3.

I provided the cover image in file: images/cover.jpg

Place the cover image to the left of the section with title 'Original text', in the same row as

'Original text' and 'Author' sections.

On large screen the 3 sections in that row should occupy:

-> 5 colums - 'Cover'

-> 2 colums - 'Original text'

-> 5 colums - 'Author'

On a small screen instead:

-> 3 colums - 'Cover'

-> 3 colums - 'Original text'

-> 6 colums - 'Author'

3) Add a div on top of the page, with 3 DIVs inside. Place the following text inside

the 3 DIVs (left to right): 'sideImageLeft', 'TITLE' and 'sideImageRight'

(see image screenshot07.png and screenshot08.png)

On large screens the 3 DIVs should occupy 2, 8 and 2 columns respectively;

on small screens the 3 DIVs should be the same size.

Images:

- screenshot01.png => shows the page on a small screen

- screenshot02.png => shows the page on a large screen

- screenshot03.png => shows the page after point 1 is solved (on a small screen)

- screenshot04.png => shows the page after point 1 is solved (on a large screen)

- screenshot05.png => shows the page after point 2 is solved (on a small screen)

- screenshot06.png => shows the page after point 2 is solved (on a large screen)

- screenshot07.png => shows the page after point 3 is solved (on a small screen)

- screenshot07.png => shows the page after point 3 is solved (on a large screen)

|  |  |
| --- | --- |
| C:\Users\Andrea\Dropbox\__teaching\2017-1-spring\1_distributed programming[optek2]\exam\exam_set2[todo]\exercise2\screenshot01.png01 | C:\Users\Andrea\Dropbox\__teaching\2017-1-spring\1_distributed programming[optek2]\exam\exam_set2[todo]\exercise2\screenshot02.png02 |
| C:\Users\Andrea\Dropbox\__teaching\2017-1-spring\1_distributed programming[optek2]\exam\exam_set2[todo]\exercise2\screenshot03.png03 | C:\Users\Andrea\Dropbox\__teaching\2017-1-spring\1_distributed programming[optek2]\exam\exam_set2[todo]\exercise2\screenshot04.png04 |
| C:\Users\Andrea\Dropbox\__teaching\2017-1-spring\1_distributed programming[optek2]\exam\exam_set2[todo]\exercise2\screenshot05.png05 | C:\Users\Andrea\Dropbox\__teaching\2017-1-spring\1_distributed programming[optek2]\exam\exam_set2[todo]\exercise2\screenshot06.png06 |
| C:\Users\Andrea\Dropbox\__teaching\2017-1-spring\1_distributed programming[optek2]\exam\exam_set2[todo]\exercise2\screenshot08.png07 | C:\Users\Andrea\Dropbox\__teaching\2017-1-spring\1_distributed programming[optek2]\exam\exam_set2[todo]\exercise2\screenshot07.png08 |

Exercise 3

==========

Look in the files:

- exercise03.html

- js/DieRoller.js

The DieRoller.js file define a DieRoller data-type, that represents a machine that can

roll many times the same die.

In the HTLM file you can see that an object of DieRoller type is created, using

the constructor function.

In the HTML page the die roller rolls a die with 6 faces.

In the HMTL page there is also a form, with text inputs and buttons.

1) Using jQuery, write an event handler for the click event of the 'rollBtn' button

so that when the button is clicked, the text in 'itemTextInput' is used to determine how

many times the die should rolled. The results generated by the latest roll of

DieRoller object should be visible in the 'mainArea' DIV.

Do not clear the text in 'itemTextInput', so the user can roll repeatedly.

NOTE: call the roll() method of the DieRoller object, each time the 'rollBtn' button is clicked.

Also: the roll() method takes a number as parameter, so be sure that the value you read from

the input text is converted to a number before you call roll().

2) Add another click handler for the 'averageBtn' button, so that it

writes the average of all rolls in the DIV with class 'average'

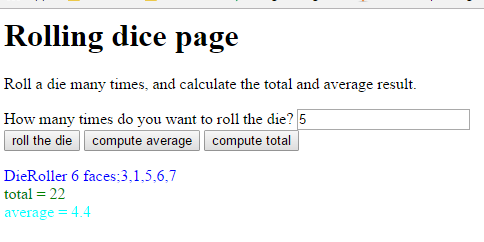
3) Add another click handler for the 'totalBtn' button, so that it

writes the total of all rolls in the DIV with class 'total'

Note: when you roll the die again, the total and the average should disappear from the

respective DIVs.

The result should look like in image: finalScreenshot.png (below)



Exercise 4

==========

Look at the file: solarSystem.html

For a game, you need to generate a description of a solar system that has from 5 to 10 planets

(use a random value to decide how many).

Each planet can have:

- and 0 to 5 continents (use a random value to decide how many)

- an ocean (10% probability)

Each ocean can have:

- aquatic life (10% probability)

Using jQuery:

1) Write a program that generates a random solar system, using nested DIVs to represent the planets, continents, and oceans and aquatic life.

2) When the user clicks the button, create a new DIV representing a space ship (give it the class 'spaceship').

If there is at least a planet in your solar system that has aquatic life, put the spacesphip inside the DIV of that planet. If in your current solar system there are no planets with aquatic life, then place your spaceship on the 3rd planet.

Image generation.png shows the result after you have implemented point 1 above,

and image spaceshipLanded.png shows the result after pressing the button to land the spaceship.

|  |  |
| --- | --- |
| C:\Users\Andrea\Dropbox\__teaching\2017-1-spring\1_distributed programming[optek2]\exam\exam_set2[todo]\exercise4\generation.pnggeneration.png | C:\Users\Andrea\Dropbox\__teaching\2017-1-spring\1_distributed programming[optek2]\exam\exam_set2[todo]\exercise4\spaceshipLanded.pngspaceshipLanded.png |