Exercises related to HTML, CSS, and JavaScript

- This document provides exercises to learn some basics of web programming. Mostly these exercises deal with JavaScript programming.
- To do these exercises, you need an editor and a web browser. The editor should be such that it can highlight the syntax of the HTML, CSS, and JavaScript languages. For example, Notepad++ is a suitable editor.
- You should test your HTML pages with several browsers to ensure that they work as required.

Kari Laitinen
http://www.naturalprogramming.com
2014-01-06 File created.
2018-11-04 Last modification.

EXERCISES RELATED TO BASICS OF HTML

HTML is an acronym of Hyper-Text Mark-up Language. With this language all the pages that are available on the Internet are described. These pages are commonly called HTML pages although they may contain also other languages than plain HTML. The term Hyper-Text is used in the name of the language just to tell that the texts in HTML pages differ in many ways from conventional texts, such as the pages of a book.

With HTML it is possible to describe the content of a web page. In addition it is possible to use another language called CSS to specify how the content of a web page looks like on the screen. In these exercises, however, we'll just concentrate on HTML.

Exercise 1:

Create an HTML file (e.g. **first_page.html**) that specifies a page that contains a heading and two paragraphs of text. Use the HTML tags `<h1>`, `</h1>`, `<p>`, and `</p>` in this exercise.

As the texts in the heading and paragraphs you can use any texts you like.

Exercise 2:

Add an unordered list to your first web page. An unordered list should look like the following when it is shown by a browser:

- An unordered list can be specified with the tags `<ul>` and `</ul>`.
- An unordered list typically contains a number of list items that can be specified with tags `<li>` and `</li>`.
- After you have created your unordered list, check out what happens when you convert it to an ordered list by replacing the tags `<ul>` and `</ul>` with `<ol>` and `</ol>`, respectively.

Exercise 3:

Add an image to your web page. In this exercise you must use the `<img>` tag. As an image, you can use any **.jpg** or **.png** file you find on the Internet.

Exercise 4:

Create another **.html** file that contains a heading and a couple of paragraphs. You could name this new file **another_page.html**, and you should place it into the same folder where your first **.html** is.

After you have created the new **.html** page, add a link to the first page so that the browser will load **another_page.html** when you click the text *Go to the other page*. in the first page.

You need to use the `<a>` and `</a>` tags in this exercise. Inside the tag `<a>` you need to use a

`href` attribute that specifies which page will be loaded when the link is clicked.

Exercise 5:

HTML tags like `<a>` can have certain attributes. The `href` attribute is mandatory in the `<a>` tag. Additionally it is possible to use the `title` attribute which specifies a text that emerges when the mouse cursor is moved above a link. This kind of text is called a *tool tip*.

Modify the link that you created in the previous exercise so that a tool tip says "This leads you to another page." when the mouse cursor is over the link.

Exercise 6:

It is possible to use a picture (image) as a link. Modify your page so that the picture that is on your page will also serve as a link that leads to another page.

Exercise 7:

Upload your two **.html** files to a server and test that they work as real internet pages.

EXERCISES WITH ButtonDemo.html

The web page specified by **ButtonDemo.html** contains a button with which it is possible to change the text that is shown on the screen. The file **ButtonDemo.html** has a JavaScript function named `change_text()` which is called after the button is pressed. When the button is pressed repeatedly the text changes Hello! ... Well done! ... Hello! ... Well done! ... Hello! ...

Exercise 1:

Modify the program so that the initial text shown on the screen is "Monday", and it will change in the following way when the button is pressed repeatedly: Monday ... Tuesday ... Wednesday ... Thursday ... Friday ... Saturday ... Sunday ... Monday ... Tuesday ... etc.

You should also change the button text so that there is written "Change day" on the button.

One way to solve this problem is to write a long `if` ... `else if` ... `else if` ... construct inside the JavaScript function. This is probably an easy way to solve the problem.

Another possibility is to use an array of strings and an index variable. You can add the following definitions right before the `change_text()` function

```
var day_index = 0 ;

var days_of_week = [ "Monday", "Tuesday", "Wednesday", "Thursday",
                     "Friday", "Saturday", "Sunday" ] ;
```

When you have an array like the one above, you can, after the button is pressed, increment the index variable and take one string from the array. When the index variable has a correct value, the text can be modified with the statement

```
text_element.innerHTML = days_of_week[ day_index ] ;
```

Exercise 2:

Improve the program by adding another centered text and another button to the page. When the new button is pressed it should change the new text in the following way: January ... February ... March ... April ... etc.

You should use a new JavaScript function that will be called after the new button is pressed. You could also use the following definitions in your program:

```
var month_index = 0 ;

var names_of_months  =  [ "January", "February", "March", "April",
                          "May", "June", "July", "August",
                          "September", "October", "November", "December" ] ;
```

Exercise 3:

Use CSS definitions to make the background of the entire page light yellow. Studying the page **Game.html** should be helpful when doing this and the following exercise.

Exercise 4:

Use CSS definitions to make the buttons bigger and better-looking. The page could look like the following:

Try also `button:hover` and `button:active` definitions for your buttons. For example, with the following definition the button background color changes when the mouse cursor 'hovers' over the button:

```
button:hover
{
   background-color : GoldenRod ;
}
```

EXERCISES WITH PictureViewing.html

When you download the file **PictureViewing.html** you must also download the images that are located in the subfolder named **images**. You should have the picture files in this subfolder when you test and develop the page locally on your computer.

Exercise 1:

Add a new image to the list of pictures that is being displayed by the page.

Exercise 2:

Add a new button with text "Previous" to the page. By pressing this button it must be possible to view the previous image. You can add the new button into the same `<div>` element where the Next button is defined. You'll need a new JavaScript function for the new button. The name of the new function could be `show_previous_picture()`.

Exercise 3:

Add two new buttons with which it is possible to shrink or enlarge the shown picture. You need two new JavaScript functions for these buttons. The functions should modify the "width" attribute of the `<img>` element to change the picture sizes. (The height of the picture should adjust automatically when the width is changed.)

Exercise 4:

Improve the page so that a line of text is shown with each picture. The texts should tell what image is in question. You should have the texts in an array of strings. To make this task easier you can use the file names that are in an array of strings.

Exercise 5:

Try to find out how to advance to the next picture by clicking the current picture. This means that it should be possible to see the next picture either by clicking the Next button or by clicking the currently shown picture.

EXERCISES WITH MovingRectangle.html

Exercise 1:

Modify the CSS style definition so that the initial width of the rectangle border is 6 pixels.

Add a new selectable color to the list of selectable colors. The name of the new color must be "lawn green". The Web Programming page at http://www.naturalprogramming.com/ has a link through which you can find ’color names’, i.e., strings that can be used for colors.

Exercise 2:

Add a button with text "Reset" to the page. This button should bring the rectangle to its original position. Additionally the button should set the color of the rectangle to red.

Exercise 3:

Add two new buttons with which it is possible to adjust the size of the rectangle. The other button should make the rectangle larger and the other button should make it smaller.

The style properties which you should modify with these buttons are "width" and "height". You should declare JavaScript variables like

```
var rectangle_width  = 160 ;
var rectangle_height = 120 ;
```

These variables should contain the values that will be assigned to style properties.

Exercise 4:

Find out how to use so-called radio buttons in a web page. Add a number of radio buttons to the **MovingRectangle.html** page so that it is possible to adjust the border of the rectangle with the radio buttons.

EXERCISES WITH GuessAWord.html

GuessAWord.html is a rudimentary computer game in which the player has to try to guess the characters of a word that is 'known' by the game. Study the page source code and play the game in order to find out how the game has been programmed.

Exercise 1:

Improve the Guess-A-Word game so that the word to be guessed is randomly taken from an array of string objects. Such an array can be created with a statement such as

```
var words_to_be_guessed =
                [ "VIENNA", "HELSINKI", "COPENHAGEN",
                  "LONDON", "BERLIN", "AMSTERDAM" ] ;
```

A random index for an array such as the one above can be created with the `Math.random()` and `Math.floor()` methods with a statement like

```
var random_word_index =
                Math.floor( Math.random() * words_to_be_guessed.length ) ;
```

The `Math.random()` method returns a `double` value in the range 0.0 .... 1.0 so that the value 1.0 is never returned. The above expression thus calculates a suitable random index as the method `Math.floor()` ensures that `double` values are always rounded 'downwards' to the smaller integer value.

Exercise 2:

Improve the program so that it counts how many guesses the player makes during the game. The page should show below the buttons a line of text that tells how many guesses have been made. The number of guesses should increase by one each time after one of the buttons is pressed. The following variable could be useful in this task

```
var number_of_guesses = 0 ;
```

Exercise 3:

Improve the page so that it displays the text "Congratulations!" after all characters of the secret word have been guessed. This can be done by checking if the array of guessed characters contains the string `'-'`. An array method named `indexOf()` can be used to find out whether or not `'-'` belongs to an array. Study the documentation of JavaScript array methods.

Exercise 4:

After a game has finished, the user must be offered a possibility to play another game. Modify the program so that a button with the text "New Game" appears on the screen after all characters of the word are guessed. The button must disappear and variables must be initialized for a new game if the user presses the button.

In this exercise you should have a new function (e.g. `initialize_game()`) that does all the initializations that are necessary at the beginning. This function should be called when the page is fully loaded, and each time the user wants to start a new game. The function can be called in the `<body>` tag in the following way

```
<body onload="initialize_game()">
```

A button can be made to appear or disappear by modifying its CSS property named `display`. When the `display` property has value `none`, the button (or some other element) is not shown. When the `display` property is changed to value `inline`, the button will become visible on the screen. In this exercise you just specify a button in the normal way and make it invisible in the beginning.

Exercise 5:

Improve the game so that, when the user is playing the game by pressing the buttons, those buttons that have been pressed will be made inactive (disabled). It does not make any sense to push the same button twice during a single game. A button can be disabled with a statement like

```
pressed_button.disabled = true ;
```

Probably it is also necessary to make a CSS definition for the disabled buttons:

```
button:disabled
{
   color: White ; /* text color on the button. */
   background-color: LightGray ;
}
```

All buttons should be re-enabled when a new game starts. This can be done in the `initialize_game()` function with statements like

```
var letter_buttons = document.getElementsByClassName( "letter_button" ) ;

for ( var button_index in letter_buttons )
{
   letter_buttons[ button_index ].disabled = false ;
}
```

To make the above statements work, each button must be given the class definition `letter_button` when the buttons are created.

Possible other improvements for the page:

There are many possibilities to further improve the Guess-A-Word game:

- There could be two possible ways to show the disabled buttons. Those buttons containing ’correct’ letters would be shown with a different color than those buttons whose letters did not belong to the secret word.
- The buttons on the screen could be rearranged to three rows so that their order would resemble the usual QWERTY keyboard.
- It should be possible to use the physical computer keyboard to make guesses.
- There could be several categories of words that could be guessed, and the words could be imported from a database from a server. One random letter of the secret word could be shown when a new game starts.

EXERCISES WITH Olympics.html

The web page **Olympics.html** is an example in which object-oriented programming is done with JavaScript. With the new version of JavaScript it is possible to define classes with keyword `class` just like in many other programming languages. **Olympics.html** has a JavaScript class named `olympics`, and there is an array that contains `olympics` objects for most known Summer Olympics.

Exercise 1:

Add new `olympics` objects to the end of the list of olympics. Use Google to find out information about missing or future Olympic Games. When you add the objects correctly, the program can automatically generate buttons for them.

Exercise 2:

The 4-year periods in which Olympic Games are held are called Olympiads. The 1896 Olympic Games, that were the first modern Olympic Games, are the games of the first Olympiad, the next games in 1900 are the games of second Olympiad, and so on. During some Olympiads no games were arranged because of world wars.

The formula for calculating the Olympiad number for Olympic Games is:

```
var olympiad =  Math.floor( ( olympic_year - 1896 ) / 4 ) + 1 ;
```

The `Math.floor()` function rounds numbers downwards to an integer.

Improve the page so that, for example, instead of the line

```
Olympic Games 1992
```

the page shows the line

```
Olympic Games 1992, XXV Olympiad
```

in which the Olympiad is shown in Roman numerals.

To implement and test this feature you can first show the Olympiad in normal numbers, and later convert the number to Roman numerals. The conversion can be made with the initialized array that can be found in comments at the end of file **Olympics.html**. You can use the Olympiad as an index when accessing the array.

You can do this exercise by modifying only the function `olympic_year_was_selected()`. You can, however, place the initialized array outside the function.

Exercise 3:

Wikipedia has pages with addresses (URLs) like

```
https://en.wikipedia.org/wiki/1992_Summer_Olympics
https://en.wikipedia.org/wiki/1996_Summer_Olympics
https://en.wikipedia.org/wiki/2000_Summer_Olympics
https://en.wikipedia.org/wiki/2004_Summer_Olympics
```

You can see that only the olympic year is different in these addresses, so it will be easy to make links to these pages when the olympic year is known.

Improve the page so that the second text line like

Barcelona, Spain

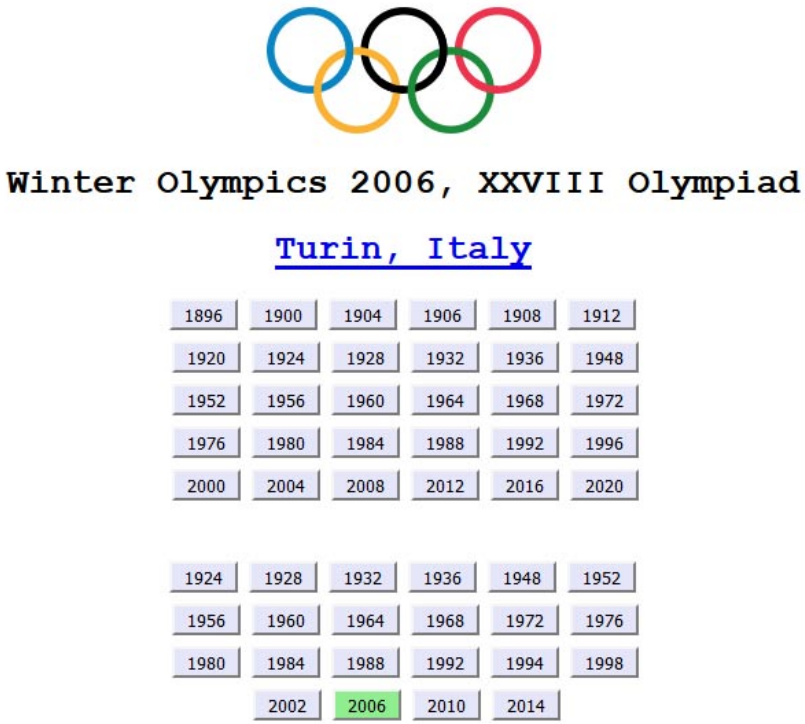
will be a link to a corresponding Wikipedia page. Again you can do this by just modifying the function `olympic_year_was_selected()`.

You can construct the link (the `<a>` element) by joining strings and the numerical olympic year with the `+` operator. Remember that when you need to include a double quotation mark `"` into a string, you need to precede it with a backslash `\`. Inside the `<a>` tag you should have the attributes

```
target="_blank"
title="More info in Wikipedia"
```

Exercise 4:

Improve the page further so that it will have buttons to select Winter Olympics. In the end, the page should look like the following:



You should have a new `<div>` with `id=area_for_buttons` to create the buttons for Winter Olympics. This new `<div>` can come after the `<div>` for existing buttons.

At the end of the file **Olympics.html** you can find, inside comments, a suitable array to be

used to create the buttons for Winter Olympics.

You can use the same method to react to both buttons for Summer Olympics and buttons for Winter Olympics. To distinguish these two types of buttons, you can use the `name` attribute. For example, you can define `name=summer_olympics_button` when you create a button for Summer Olympics.

Then, when you react to the pressings of the buttons, you can make tests such as:

```
if ( pressed_button.name == "summer_olympics_button" )
{
    ...
```

Instead of using the text "Olympic Games 1988", you should use texts like "Summer Olympics 1988" or "Winter Olympics 1988", depending on what type of button was pressed.

Also the links to Wikipedia must be different in case of Winter Olympics. The Wikipedia addresses for Winter Olympics look like

```
https://en.wikipedia.org/wiki/1968_Winter_Olympics
https://en.wikipedia.org/wiki/1972_Winter_Olympics
https://en.wikipedia.org/wiki/1976_Winter_Olympics
```

EXERCISES WITH MonthCalendar.html

When you download the file **MonthCalendar.html**, you must also download the file **importables/DateMethodsMore.js** and put it into a subfolder named **importables**.

The file **DateMehodsMore.js** contains self-made methods to be used with `Date` objects. Those methods are used in `EnglishCalendar` class to convert the calendar to a `<table>` element. You do not need to use or study those methods while doing these exercises.

Exercise 1:

The calendar shown in the page is a `<table>` element that contains `<tr>`, `<th>`, and `<td>` elements and a `<caption>` element. You can't see the actual HTML code of the table as it is generated with a JavaScript method. You can, however, make CSS definitions that dictate what the table looks like.

Currently the `<th>` and `<td>` elements of the table are colored with yellowish warm colors. Select different colors for these elements, so that they look good together. You can explore color combinations if you go to https://color.adobe.com/

The Web Programming page has a link to acceptable color names. Alternatively, you can use numerical colors definitions such as #43ACF3 and #6ED7F8.

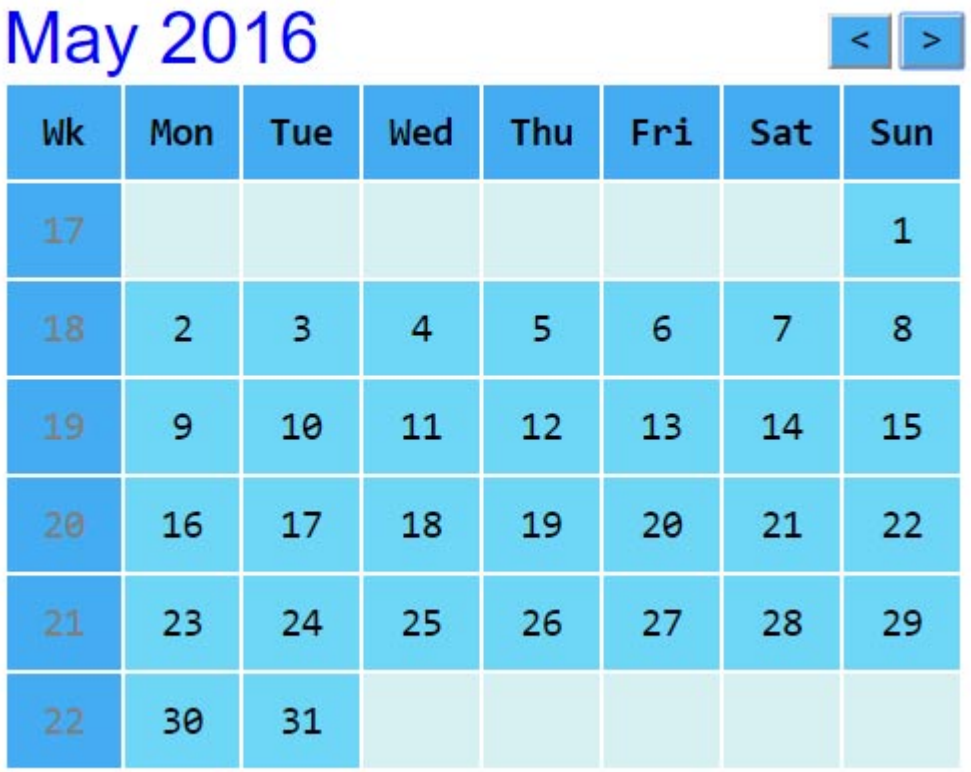
Exercise 2:

Add more CSS definitions so that those `<td>` elements that represent days of the previous month or the days of the next month will have a different color. In addition you should also use a new color for those `<td>` elements that contain week numbers.

You can do this when you note that the JavaScript method that generates the table sets the following `id`s for the mentioned table elements:

- `<td id=week_number>` for table cells with week numbers
- `<td id=day_of_previous_month>` for days of previous month
- `<td id=day_of_next_month>` for days of next month

After these improvements, the calendar could look as the one in the following picture. (In this picture `<td>` elements that belong to the previous or the next months have a lighter color, and the `<td>` elements containing week numbers have the same color as the `<th>` elements.)



Exercise 3:

Improve the page so that it will have buttons to decrement and increment the year of the calendar.

The buttons can be specified in the following way.

```
<button onclick="minus_one_year()">-</button>
<button onclick="plus_one_year()">+</button>
```

You can put these buttons before the existing buttons on the same line. When you use just one-character texts on the buttons, the buttons do not need to be wider.

The calendar year can be decremented with the following function.

```
function minus_one_year()
{
   calendar_to_show.decrement_year() ;

   var div_for_calendar = document.getElementById( "calendar_div_id" ) ;

   div_for_calendar.innerHTML = calendar_to_show.to_table_element() ;
}
```

You can increment the year with a similar function. The `EnglishCalendar` 'class' has a method named `increment_year()`.

Exercise 4:

At the end of the HTML file you can find, in comments, two 'subclasses' for the `EnglishCalendar`. When you start using these classes, you can easily change the language of the calendar.

You should first specify a new button like

```
<button onclick="change_language()">L</button>
```

This button could first work so that when it is pressed, a `SpanishCalendar` object is taken into use and the program starts showing that calendar. This can be done with the following function:

```
function change_language()
{
   calendar_to_show = new SpanishCalendar( calendar_to_show.get_year(),
                                           calendar_to_show.get_month() ) ;

   var div_for_calendar = document.getElementById( "calendar_div_id" ) ;

   div_for_calendar.innerHTML = calendar_to_show.to_table_element() ;
}
```

In the end, you should make the L(anguage) button work so that, when it is pressed many times the calendar language changes in the following way

English -> Spanish -> German -> English -> Spanish -> etc.

You need to create a new kind of calendar object depending on which object is on display. You may need an additional variable, such as

```
var calendar_language = "English" ;
```

to make the page 'remember' which type of object is being shown. In the end, the calendar could look like the following.

Sem	Lun	Mar	Mie	Jue	Vie	Sab	Dom
35				1	2	3	4
36	5	6	7	8	9	10	11
37	12	13	14	15	16	17	18
38	19	20	21	22	23	24	25
39	26	27	28	29	30		

Possible further improvements:

By clicking a day in the calendar, it could be possible to calculate a distance from the current date to the clicked day. There already is a function that is called when a day is clicked. **Apollo11.html** is an example about calculating temporal distances.

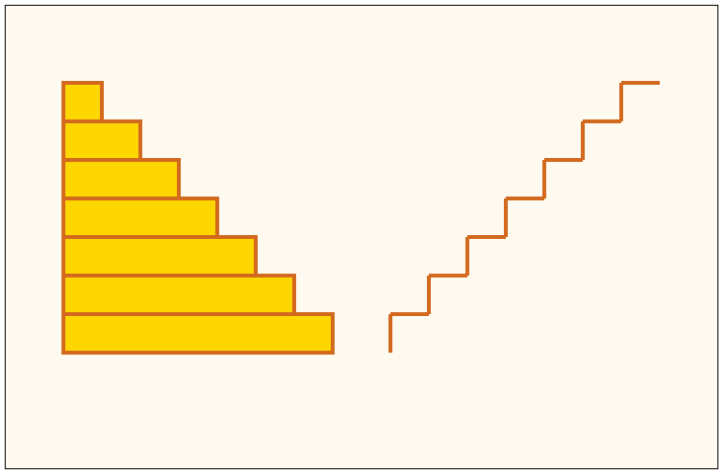
EXERCISES WITH StepsCanvas.html

The page **StepsCanvas.html** is an example that contains a `<canvas>` element. This element represents a page area to which we can draw with certain JavaScript methods.

When we use a `<canvas>` element, we can control the contents of the area of the `<canvas>` just by using JavaScript, without modifying CSS definitions or the HTML code of the page. The size of the `<canvas>` element must be defined in HTML and/or CSS.

Exercise 1:

The current version of the page has ascending and descending steps. Modify the program so that descending and ascending steps will be drawn in the following way:



Exercise 2:

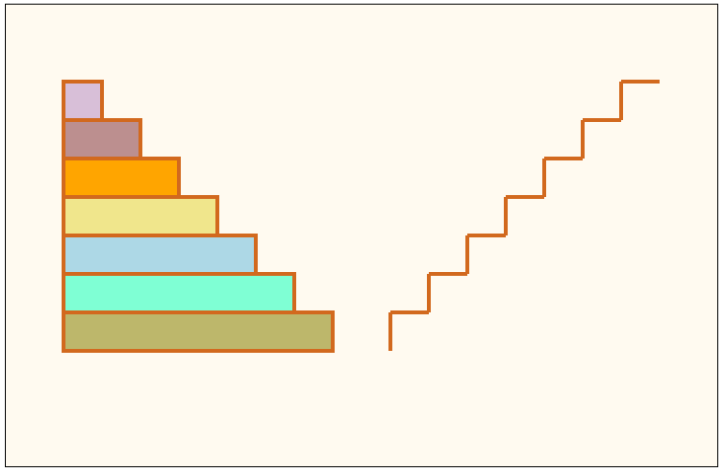
Modify the program so that each descending step is filled with a different color. To do this you have to use a new `fillStyle` before you fill the rectangle that represents a step. You can use the following array that contains acceptable color names.

```
var step_colors = [ "DarkKhaki", "Aquamarine", "LightBlue", "Khaki",
                    "Orange", "RosyBrown", "Thistle", "Tomato" ] ;
```

You can start using one color from this array with a statement like:

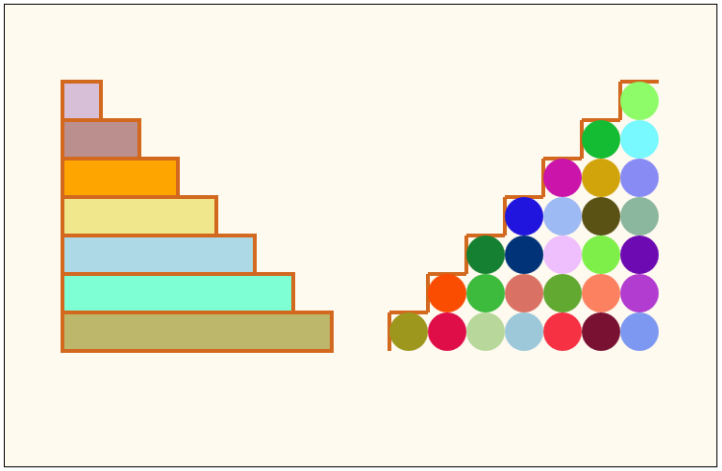
```
context.fillStyle = step_colors[ step_counter ] ;
```

After you have completed this exercise, the page should look like



Exercise 3:

Improve the program so that you draw balls with random colors under the ascending steps, in the following way.



To do this, you probably need a loop inside a loop. You can first draw all the balls with the same color, and later start using a random color.

A ball with radius `step_size / 2` can be drawn with the statements:

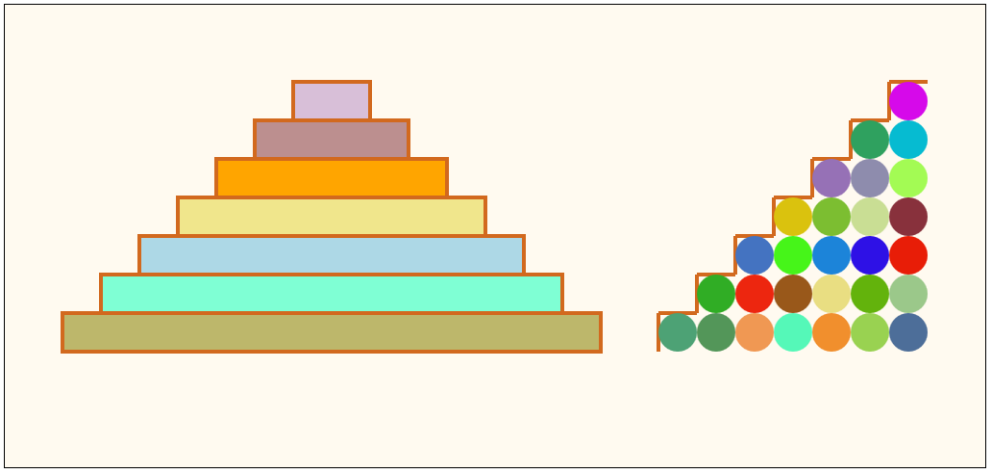
```
context.beginPath() ;
context.arc( ball_center_x, ball_center_y,
             step_size / 2, 0, 2 * Math.PI, true ) ;
context.closePath() ;
context.fill() ;  // Fill the ball
```

A random color can be set with the following statements

```
var random_color_as_int = Math.floor( Math.random() * 0xFFFFFF ) ;

// We'll convert the int value to a hexadecimal string, ensuring that
// there will be enough leading zeroes so that the string length is 6.

var random_color_as_string = "000000" + random_color_as_int.toString( 16 ) ;
random_color_as_string = random_color_as_string.slice(
                                    random_color_as_string.length - 6 ) ;

context.fillStyle = "#" + random_color_as_string ;
```

Exercise 4:

Modify the descending steps on the left so that they will look like a pyramid, in the following way.



In order to make the pyramid fit into the canvas area, you should make the area, say, 280px wider than it is in the original program. You also need to change the drawing position of the ascending steps.

After you have completed these exercises, you can say that you've learned to build a pyramid if somebody asks what you have learned at school.

EXERCISES WITH SinglePictureCanvas.html

Download the page **SinglePictureCanvas.html**, and study its source code to find out how you'll find the picture file that you need to download as well.

Exercise 1:

Modify the page so that the picture will be drawn only once, so that it is drawn exactly in the middle of the canvas area. The picture should be drawn in its natural size.

You can position the picture in the middle of the canvas when you know the width and the height of the canvas as well as the natural width and height of the picture. If you use the original picture, it fits well on the canvas.

The coordinates that are given to the `drawImage()` method specify the point where the upper left corner of the picture is going to be. You must adjust these coordinates so that the picture will be in the middle of the canvas.

(Sometimes there are problems when we try to draw picture files on a canvas. If you encounter strange problems when you try to draw an image, try some other picture file.)

Exercise 2:

Improve the page so that you draw a frame around the picture that is exactly in the middle of the canvas. One possibility is that you draw, before drawing the picture, a filled rectangle that is slightly larger than the picture. When you draw the picture over the rectangle, the edges of the regtangle will look like a frame.

Exercise 3:

Improve the program so that you draw a brick wall behind the picture that is ’hanging’ in the middle of the canvas area.

You can first fill the entire canvas area with a color that is the color of mortar, the material that is between bricks in a brick wall. After you have drawn the ’mortar’, you can draw the bricks so that there is a gap between the bricks.

Among the canvas-related pages you can find a page named **BrickWallCanvas.html**. You can copy suitable source code lines from that **.html** file to do this exercise.

Note that it is not ’dangerous’ to make drawings outside the canvas area. Such drawings will simply not be visible. Therefore it is possible that you draw the first brick of a row with a negative x-coordinate.

After you have completed this exercise, the page should look like the following.

Exercise 4:

Improve the program so that it will be possible to enlarge or shrink the shown picture and its frame with the Arrow Up and Arrow Down keys.

To do this, you have to make the program react to keyboard input. See the page **KeyboardInputCanvas.html** for help. It will be sufficient if you write only one function that reacts to the pressings of the keys.

You can define a global variable like

```
var picture_enlargement = 1.0 ;
```

whose value you can alter in the method that reacts to key pressings. Then you can use the value of this variable in the `draw_on_canvas()` method, for example, in calculations like

```
var desired_picture_width = picture_enlargement * picture_width ;
var desired_picture_height = picture_enlargement * picture_height ;
```

Exercise 4:

If you still have time and enthusiasm left, improve the program so that it is possibe to change the picture that is hanging on the brick wall. Changing the picture should be done with the Arrow Right and Arrow Left keys.

You can use an array of Image objects, and from that array you can take one image to use with a statement like

```
picture_to_show = pictures_to_be_shown[ index_of_current_picture ] ;
```

You should also have a global index variable whose value is changed when the Arrow Right or Arrow Left keys are pressed.

You can copy the needed array, the statements for initializing the array, and the needed index variable from the page **PictureShowCanvas.html**.

EXERCISES WITH ClickingsCanvas.html

ClickingsCanvas.html shows how we can react to the pressings of the mouse buttons with JavaScript, and how we can find out the coordinates of the clicked point.

Exercise 1:

Modify the `draw_on_canvas()` function so that it no longer draws the points that have been clicked. Instead, the program must print a text right in the center of the canvas area. It should also print the text by using quite a large font. For the text and font you should define the following global variables into the program (global variables are those that are defined outside JavaScript functions and they are visible to all functions):

```
var text_to_show = "Click!" ;
var large_font   =  "bold 24px serif" ;
```

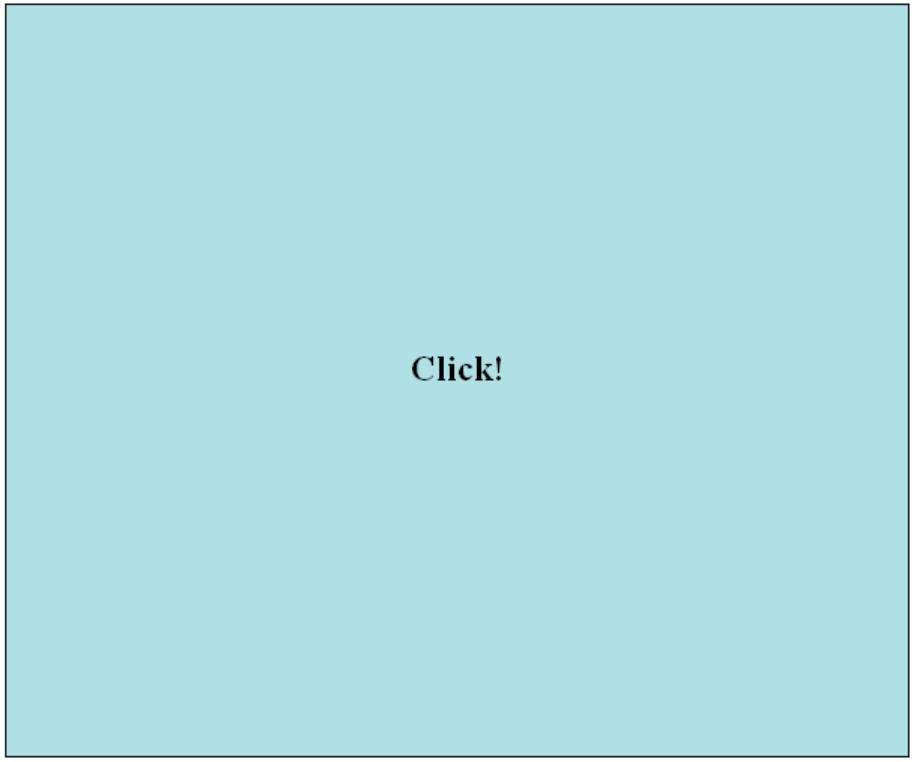
By studying the example pages **DrawingDemoCanvas.html** and **EarthAndMoonCanvas.html** you'll find out how you can draw the above text string by using the defined font. You can print the text roughly in the center area of the canvas if you use coordinates that you get when you divide `canvas.width` and `canvas.height` by two.

To print the text accurately in the center of the canvas area, you have to adjust the x-coordinate of the text depending on how long or wide the text is. To check the width of a text, there exists a method named `measureText()`. When you write

```
context.measureText( text_to_show ).width
```

you can refer to the width of the text (in pixels) in current graphics context. Note that the font must be set before using `measureText()`.

After you have completed this exercise, the canvas should look like the following:



Exercise 2:

Modify the program so that when the canvas area is clicked with the mouse, the text that is shown on the screen will change. The text must be "North-East" if the clicked point is located in North-East in relation to the center point of the canvas, i.e., the x-coordinate of the clicked point is greater than the x-coordinate of the center point and the y-coordinate of the clicked point is smaller than the y-coordinate of the center point. Similarly, the text must change to "South-East", "North-West", or "South-West" depending on how the clicked point relates to the center point of the canvas.

You can solve this problem so that you move the variables

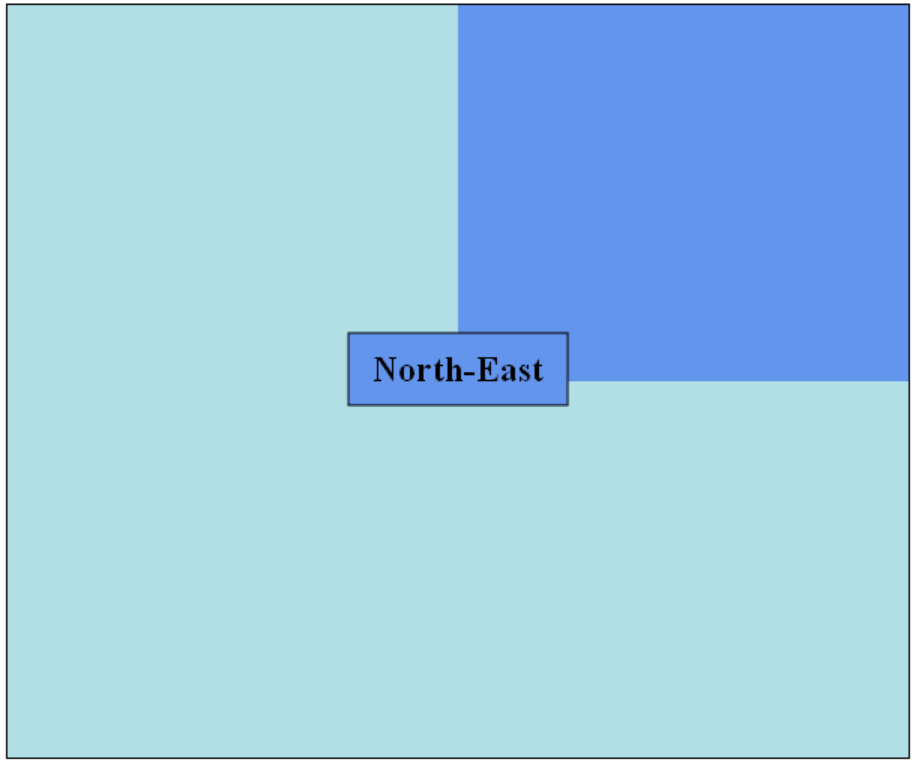
```
var clicked_point_x = -1 ;
var clicked_point_y = -1 ;
```

out of method `on_mouse_down()` and make them global variables. If you initialize them with value -1, you can use this value to check whether any clickings have taken place.

When you set the values of the above variables in method `on_mouse_down()`, you can then use these values in method `draw_on_canvas()` and change the text depending on how the clicked point relates to the center point of the canvas.

Exercise 3:

Improve the User Experience of the program so that the text that is shown in the center of the canvas is inside a colored and framed rectangle, and that quarter of the canvas that was clicked last is shown with a different highlighting color, in the following way



This exercise can be done by modifying the `draw_on_canvas()` method.

Exercise 4:

If you have time and enthusiasm left, you can further improve the program so that you use two methods that react to mouse pressings (see **MouseDemoCanvas.html**).

The method (function) `on_mouse_down()` could remove the highlighting color of the previously clicked quarter of the canvas, and only after the mouse button is released and `on_mouse_up()` is called a new quarter of the canvas is ’painted’ with the highlighting color.

You will most likely need a boolean variable that tells the `draw_on_canvas()` method when a mouse button is pressed down.

EXERCISES WITH DrawingRectanglesCanvas.html

The page **DrawingRectanglesCanvas.html** displays a canvas on which you can draw rectangles by using the mouse.

Exercise 1:

The `draw_on_canvas()` function of the page always draws all the rectangles that the user has drawn so far. In addition it draws the rectangle that is not yet finished if the user is in the 'process of drawing'.

The function uses an array of colors to give each rectangle an own color. It always takes the last color from the array, and puts it to the beginning of the array after it has been used. This way the same color pattern will be repeated if the number of rectangles exceeds the number of available colors.

Add a new color to the array and study how it affects the behavior of the program.

Exercise 2:

Make all the rectangles somewhat transparent. This means that if a rectangle is drawn over another rectangle, the older rectangle is visible through the rectangle that is written later.

This modification can be done with statements in which you give a suitable value to the graphics context property named `globalAlpha`. In the world of colors, the alpha value usually

specifies how transparent a color is. By setting the value of `globalAlpha` you can specify that all colors have a certain transparency or opacity.

You should modify the value of `globalAlpha` so that the colors are not transparent at all when the canvas background is filled. Then, when the rectangles are drawn the colors should be somewhat transparent.

Exercise 3:

By studying program **MouseDemoCanvas.html** you can find out how to react to the `onmouseover` and `onmouseout` events. Make the page **DrawingRectanglesCanvas.html** such that its background color will change when the mouse cursor is brought over the canvas. The background color must change again when the mouse cursor leaves the canvas area.

Exercise 4:

Modify the `on_mouse_down()` method so that it checks whether the Ctrl key is pressed simultaneously with a mouse button. If that is true, the coordinates of the last drawn rectangle should be removed from the arrays, and a new drawing operation should not begin.

The user should thus be able to delete the last rectangle if he or she presses the Ctrl key together with a mouse button.

To do this exercise, you must study the documentation of JavaScript Array Object to find out how the last element of an array can be removed.

Exercise 5:

Modify the `on_mouse_down()` method further so that it checks whether both the Ctrl key and the Shift key is pressed simultaneously with a mouse button. If that is true all the rectangles must be removed, i.e., the used arrays must be emptied.

It is important that a new drawing operation does not begin if the mentioned keys are used.

EXERCISES WITH MovingBallsWithPointerCanvas.html

In these exercises you'll learn to do some object-oriented programming with JavaScript. The page **MovingBallsWithPointerCanvas.html** uses a file named **Ball.js** that contains the definition of a class named `Ball`. Objects of type `Ball` represent balls that are shown on the screen. The files **MovingBallsWithPointerCanvas.html** and **Ball.js** must be in the same folder when you do these exercises.

In the new version of JavaScript there is the keyword `class` with which it is possible to define classes in the same way as in some other programming languages. Inside JavaScript classes, constructors are specified with the word `constructor`.

Exercise 1:

Now the page shows three balls on the screen. Modify the program so that you add a fourth ball into it. You can create the new `Ball` object with a statement like

```
var fourth_ball = new Ball( 400, 500, "Chocolate" ) ;
```

You need to modify the methods `on_mouse_down()` and `draw_on_canvas()` in order to make the program move and draw the new ball. Other methods do not need to be modified.

Exercise 2:

In the current version of the program there are some problems. For example, if you move the balls so that the red ball is under some other balls, and you try to move the ball that is on top of the other balls, it is likely to happen that the red ball will be moved. It is also possible to move a ball under another ball, which is not very logical.

To solve these problems, the program should be made such that the ball that is the last one moved, should be the last one that is drawn to the screen. Also, the topmost ball should be the one that will be moved if a ball is located on top of another ball.

You should solve these problems by first defining an array into which you store all the `Ball` objects. A suitable empty array can be defined in the following way

```
var balls_on_screen = [] ;
```

and the `Ball` objects can be put to the array with statements like

```
balls_on_screen.push( first_ball ) ;
balls_on_screen.push( second_ball ) ;
balls_on_screen.push( third_ball ) ;
balls_on_screen.push( fourth_ball ) ;
```

The `push()` method puts an object to the end of an array. After you have made the above modifications you can change the `draw_on_canvas()` method so that you use the following loop to draw all the `Ball` objects.

```
for ( var ball_index = 0 ;
          ball_index < balls_on_screen.length ;
          ball_index ++ )
{
   balls_on_screen[ ball_index ].draw( context ) ;
}
```

You should first make the above modifications and ensure that your program works well.

Then, when you want that the last ball that is moved will be the last to be drawn, you must modify the program so that the last moved ball will be put to the end of the array. This can be done by deleting a `Ball` object from the array and then pushing it to the end of the array.

You can do this modification by adding the following lines to the `on_mouse_down()` method.

```
if ( ball_being_moved != null ) // If a ball to be moved is found.
{
   var index_of_ball_being_moved =
                       balls_on_screen.indexOf( ball_being_moved ) ;

   // The following line removes the ball from the array.
   balls_on_screen.splice( index_of_ball_being_moved, 1 ) ;

   // Here you have to write a statement to push the object
   // to the end of the array.
}
```

After you have done the above modifications, one problem is solved. There is, however, still the problem that the topmost ball is not always the ball that will be moved. To correct this problem, you should modify the mechanism how a ball is searched in the `on_mouse_down()` method. Because the `Ball` objects that are the last ones moved and drawn are at the end of the array, you should start searching for the clicked ball by starting from the end of the array. Therefore, you should replace the `if ... else if ... else if ...` construct with the following kind of loop.

```
var ball_index = balls_on_screen.length - 1 ; // index of the last ball
var search_for_clicked_ball_ready = false ;

while ( search_for_clicked_ball_ready == false )
{
   if ( /* You should call contains_point() here. */ )
   {
      //  You should add one statement here
      search_for_clicked_ball_ready = true ;
   }

   if ( ball_index > 0 )
   {
      ball_index -- ;
   }
   else
   {
      search_for_clicked_ball_ready = true ;
   }
}
```

Exercise 3:

Now, as you have all the `Ball` objects stored in a dynamic JavaScript array, it will be easy to add more `Ball` objects that will be shown on the screen.

Improve the program so that if the Ctrl key of the keyboard is pressed simultaneously when the mouse button is pressed, you add a new `Ball` object to the end of the array. The center point of the new ball should be the current mouse cursor position. When the Ctrl key is down no moving of a ball will begin.

If you have done the previous exercises correctly, the new ball will be moved and drawn automatically when it is put to the array.

See the page **MouseDemoCanvas.html** to find out how to check whether the Ctrl key is pressed simultaneously with a mouse button.

Exercise 4:

Inside the file **InheritanceDemoCanvas.html** there is a class named `MulticoloredBall` that is a subclass of the `Ball` class. Copy the definition of the `MulticoloredBall` class to the file with which you are currently working, and modify your program so that if both the Ctrl key and the Shift key are pressed together with a mouse button, a `MulticoloredBall` object is added to the end of the array containing the ball objects.

Note that three colors must be specified when a `MulticoloredBall` object is created.

EXERCISES WITH PlayingCardsCanvas.html

When you start working with **PlayingCardsCanvas.html**, you must download the **.html** file as well as a file named **playing_cards_images.zip**. You have to unzip the **.zip** file so that you get the image files needed by the application.

The application contained in **PlayingCardsCanvas.html** assumes that the image files are located in a subfolder named `playing_cards_images`.

Do not modify the `Card` and `CardDeck` classes when you do these exercises.

Exercise 1:

Modify the program so that all cards will be face-up, i.e., suit and rank visible, when new cards are dealt with the DEAL button. Two new lines of statements are needed to do this exercise.

Exercise 2:

Modify the program so that the card deck is shuffled at the beginning when the page is loaded, i.e., it will not be necessary to press the SHUFFLE button to put the cards into a random order. This problem can be solved by inserting a single line of code to the program.

Exercise 3:

Modify the program so that the cards are dealt when the page is loaded, i.e., it is not necessary to press the DEAL button to get cards on the ’table’.

You could do this exercise so that you make a copy of the method `button_to_deal_cards_clicked()` and rename it to, say, `initialize_game()`. You should take out the call to `draw_on_canvas()` from the new method. Then you just call the `initialize_game()` from a suitable location in the program.

In the next exercise you will then rewrite the `button_to_deal_cards_clicked()` method.

Exercise 4:

Modify the method that is executed when the DEAL button is pressed so that it deals cards only to replace those cards that are turned face-down, i.e., those cards whose rank and suit are visible to the user will ’stay’ on the ’table’. (You can forget the lonesome card while dealing new cards.)

You can set the required position for the new cards so that you use the position of the cards that will be replaced. You can thus use a loop that begins in the following way.

```
for ( var card_index  =  0 ;
          card_index  <  5 ;
          card_index  ++ )
{
   if ( row_of_cards[ card_index ].card_is_face_down() )
   {
      var new_card = card_deck.get_card() ;

      var card_position = row_of_cards[ card_index ].get_card_position() ;

      // set the position for the new card and turn the new card
      // put the new card to the array so that it replaces the turned card
```

Exercise 5:

After having done the previous exercises, the page is now a game in which the player can take new cards to replace ’bad’ cards on the table.

The five cards in the row can be considered a poker hand (see http://en.wikipedia.org/wiki/Poker_hand) and the player can try to make the poker hand better by replacing some cards with new cards from the deck.

Improve the game so that it informs the user if the five cards in the row form a poker hand called Flush, i.e., all five cards belong to the same suit.

The background color of the canvas should change if there is a Flush on the ’table’.

The `Card` class provides a method named `belongs_to_suit_of()` that can be used in this exercise.

Exercise 6:

Improve the game further so that it checks also if the five cards in the row form a poker hand called ’Four of a kind’. The background color could turn "Gold" when there is a ’Four of a kind’.

Because the SHUFFLE button is no longer needed, change it to a NEW GAME button with which it will be possible to make a new CardDeck object, shuffle the deck, and deal new cards to the ’table’.

EXERCISES WITH AnimationDemoCanvas.html

Exercise 1:

Now the program is such that the blinking ball stays approximately at the center of the canvas. Modify the program so that the blinking ball will move gradually downwards. Always when the ball becomes visible after it had disappeared, it should be located a few pixels below its previous position. You can do this by modifying the value of the y-coordinate of the ball. Remember that in the graphical coordinate system the y-coordinate grows when we move downwars on the canvas.

You should control that the y-coordinate of the ball will not grow too large, so that the ball disappears entirely. After the ball has reached the bottom of the canvas, it should remain there and continue blinking.

Exercise 2:

After you have done the previous exercise, the ball should be blinking on the ’bottom line’ of the canvas.

Modify the program so that when the ball has reached the bottom line, it starts moving to the left, staying visible on the bottom line. After the ball has reached the bottom line, you should stop modifying its y-coordinate and start modifying its x-coordinate. The movement of the ball should continue upwards when it has reached the lower left corner (the South-West

corner) of the canvas.

The ball should behave so that always when it reaches a corner or a ’wall’ it should turn right and continue moving. After this exercise is completed, the ball should be circulating clockwise the edges of the canvas.

You should use `if` ... `else if` ... `else if` ... constructs to control the movement of the ball. One way to control the ball is to use a variable like

```
var current_direction = "DOWN" ;
```

and give this variable other values (e.g. "`LEFT`", "`UP`", and "`RIGHT`") depending on which is the direction the ball should go.

Exercise 3:

Make your program more object-oriented by taking into use the `Ball` class that is defined in file **Ball.js**. You should replace the ball of your program with a `Ball` object.

Exercise 4:

Modify the program so that the ball can circulate either clockwise or counter-clockwise. The ball should change its direction when any key of the keyboard is pressed. The following kind of variable can be used to store the information regarding which way the ball should move:

```
var going_clockwise = true ;
```

EXERCISES WITH GunCanvas.html

The page **GunCanvas.html** is a kind of 'military' application that displays a gun that can be fired by pressing a key of the keyboard or a mouse button. In the JavaScript program there is a `Gun` class and a `Shell` class.

Exercise 1:

Improve the program so that it will be possible to use the Arrow Up and Arrow Down keys to adjust the angle of the barrel of the gun. You can do this by first adding the following methods to the `Gun` class.

```
raise_barrel()
{
   this.barrel_angle_in_degrees += 5 ;
}

lower_barrel()
{
   this.barrel_angle_in_degrees -= 5 ;
}
```

Then you should modify the `on_key_down()` function so that you test which keyboard key is pressed. If we decide that the Space key will be used to fire the gun, the following construct can be used in the `on_key_down()` function:

```
if ( event.which  ==  0x20 ) // Is it the Space key?
{
   // The following statement will create a new Shell object.
   // The old Shell object is destroyed by the carbage collector.

   flying_shell = gun.fire() ;
}
else if ( event.which  ==  38 ) // is it the Arrow Up key?
{
   gun.raise_barrel() ;
}
else if ( event.which  ==  40 ) // is it the Arrow Down key?
{
   gun.lower_barrel() ;
}
```

The barrel angle of the `Gun` class is used when a `Shell` object is created. Therefore, the correct angle is automatically copied to fired `Shell` objects.

Exercise 2:

Improve the program so that several `Shell` objects can be flying at the same time. Instead of a single variable (object pointer) you should have an array in which you store the created `Shell` objects. You can use an array such as

```
var flying_shells = [] ;
```

and you can put a `Shell` object to the end of the array in the following way

```
flying_shells.push( gun.fire() ) ;
```

By studying program **RandomExplosionsCanvas.html** you will find out what kind of a loop can be used to draw objects that are stored in an array. With a similar loop you can also move the `Shell` objects stored in an array.

Exercise 3:

Improve the program so that there will be an explosion when a shell hits the ground.

You can copy and use the existing `Sprite` class that is available in **RandomExplosionsCanvas.html**.

Your program can work so that when a shell hits its target, i.e., the ground, it will be removed from the list of flying shells, and in place of it a `Sprite` representing an explosion will be created.

You need to know when a shell has hit the ground and and you need to know the shell

coordinates to create a corresponding `Sprite`. Therefore, you can insert the following methods to the `Shell` class:

```
has_reached_target()
{
   return ( this.shell_position_y + this.radius >= this.ground_level_y ) ;
}

get_position_x()
{
   return this.shell_position_x ;
}

get_position_y()
{
   return this.shell_position_y ;
}
```

You can use a new array in which you store the `Sprite` objects which a created to represent exploding shells. The needed array can be defined in the following way

```
var shell_explosions = [] ;
```

EXERCISES WITH AtomCanvas.html

The page **AtomCanvas.html** is an example of animation. The canvas area is redrawn with the `draw_on_canvas()` function almost one hundred times in each second.

Exercise 1:

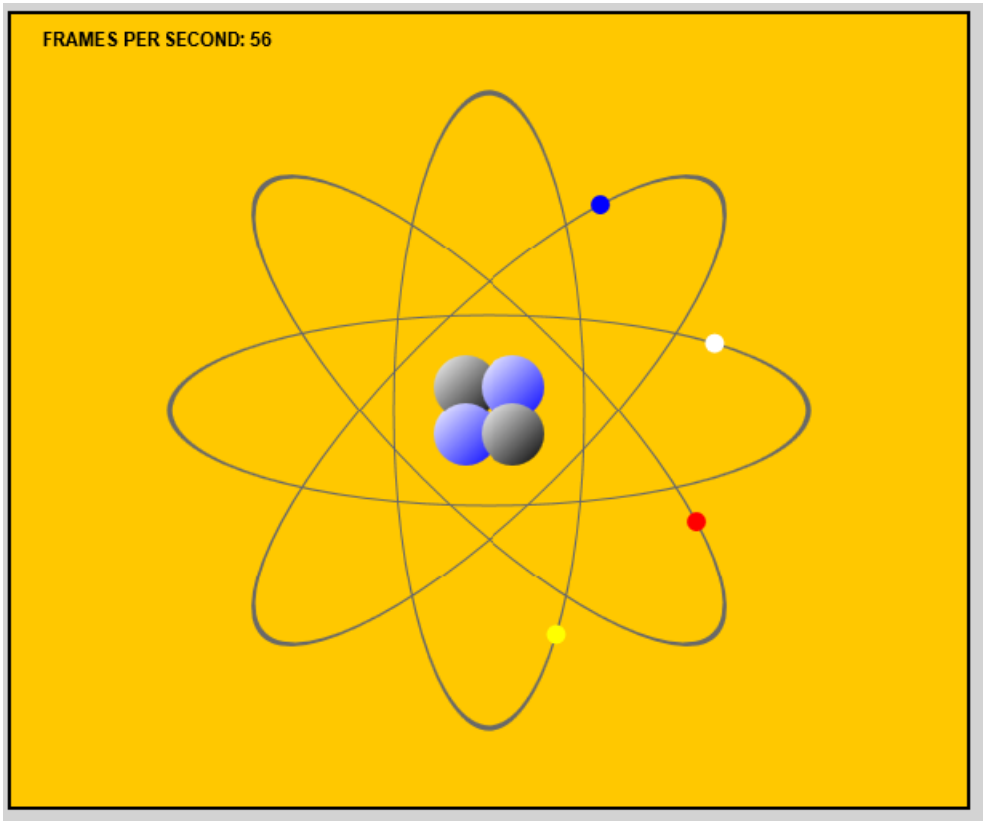
Make the orbits of the electrons somewhat narrower. This can be done by modifying the parameters that are given to the `stroke_oval()` function, and by modifying the statements that calculate electron positions. Currently the elliptical orbits of the electrons are 400 pixels wide and 200 pixels high. If you drop the height to 120 pixels, the orbits become narrower. (Note that this is a very small modification.)

Exercise 2:

Add two new electrons to orbit the nucleus of the atom. One of the new electrons should circulate on a vertical ellipse, and the other new electron should circulate on a horizontal ellipse.

This exercise can be done by modifying only the `draw_on_canvas()` function.

After this exercise, the atom should look like the following.



Exercise 3:

So-called gradient colors are in use in the drawing of the nucleus particles, which are objects of ’class’ `NucleusParticle`. The colors of the particles change gradually from white to the specified particle color.

Modify the particle colors so that ’light’ will seem to come from North-East. Now the colors are such that ’light’ seems to come from North-West.

Exercise 4:

In the program there is a mechanism to count how many times in a second the canvas is redrawn, i.e., how many times the function `draw_on_canvas()` is executed. There is already an `if` construct that finds out when a second has elapsed.

By using the existing `if` construct, modify the program so that the nucleus particles get new particle colors each time a second has elapsed.

You should write a `set_color()` method for the class `NucleusParticle`. Note that you do not need to modify the "White" that is the starting gradient color.

To get random colors, you could use an array of colors like

```
var particle_colors = [ "Teal", "Tomato", "Turquoise", "Violet",
                        "YellowGreen", "Gold", "Green", "blue" ] ;
```

and you can copy the `shuffle()` function from file **PlayingCardsCanvas.html** and use that function to shuffle the colors in the array. Then you can set the colors of the nucleus particles in the following way

```
upper_left_proton.set_color( particle_colors[ 0 ] ) ;
lower_left_neutron.set_color( particle_colors[ 1 ] ) ;
...
```

(There are other ways of getting random colors, but with the above mechanism you can select random colors from a known set of colors.)

EXERCISES WITH BouncingBallCanvas.html

The page **BouncingBallCanvas.html** is an example that contains a hierarchy of classes. The lowest in the hierarchy is `ExplodingBouncer` whose objects are balls that move, bounce, and rotate on the screen. An object of type `ExplodingBouncer` can also explode and disappear from the screen.

Unless otherwise specified, in these exercises you do not have to modify the classes of this program. The modifications should be made to the 'main' program.

Exercise 1:

Add another object of type `ExplodingBouncer` to the program. You can do this with a statement like

```
var another_ball = new ExplodingBouncer( 400, 300, "cyan",
                                         bouncing_area ) ;
```

You can make this object move when you add the following line to the `run_this_application()` method

```
another_ball.move() ;
```

The object will be drawn to the screen when you add the following statement to the `draw_on_canvas()` method

```
another_ball.draw( context ) ;
```

Exercise 2:

Modify the program so that when the page is loaded 10 balls starts moving and bouncing on the screen. This can be done so that you first create an array for the balls in the following way.

```
var game_balls = [] ;
```

After you have this empty array, you can create balls and put them into the array with a loop like

```
for ( var ball_counter = 0 ;
          ball_counter < 10 ;
          ball_counter ++ )
{
   var new_ball = new ExplodingBouncer( 400, 300, "Gold",
                                        bouncing_area ) ;

   game_balls.push( new_ball ) ;
}
```

You can use the above loop, but it is recommened that you copy the function named `prepare_a_new_game()` which is given in comments at the end of the **BouncingBallCanvas.html** file. By calling this function you can create the balls so that each ball has a different position initially.

(We use names like `game_balls` and `prepare_a_new_game()` here because in the following exercises we will develop this program towards a computer game.)

To make the 10 balls move on the screen, you should put the following loop to the `run_this_application()` method:

```
for ( var ball_index = 0 ;
          ball_index < game_balls.length ;
          ball_index ++ )
{
   game_balls[ ball_index ].move() ;
}
```

To draw the balls, you should add the following loop to the `draw_on_canvas()` method:

```
for ( var ball_index = 0 ;
          ball_index < game_balls.length ;
          ball_index ++ )
{
   game_balls[ ball_index ].draw( context ) ;
}
```

After you have successfully made the 10 balls in the array move on the screen, you can remove the original ball and the ball of the first exercise from the program.

Exercise 3:

If you did the previous exercise according to the given instructions, all the balls have the same color. To make this ’game’ more interesting, the balls should have different colors. You should give different colors to the balls by first defining an array like

```
var ball_colors = [ "Gold", "FireBrick", "DarkViolet", "DeepSkyBlue",
                    "Olive", "Orchid", "OrangeRed", "PeachPuff",
                    "Snow", "Thistle" ] ;
```

Then you can refer, with an index, to the elements of this array when you create the balls, i.e., the `ExplodingBouncer` objects.

If you decide to use more than 10 balls on the screen, you have to put more color names to the above array.

Exercise 4:

The class `ExplodingBall` has inherited a method named `contains_point()` with which it is possible to check whether or not a certain point belongs to the area of the ball object. In addition, the class provides a method named `explode_ball()` with which it is possible to explode a ball so that it eventually and automatically disappears from the screen.

In this exercise your task is to modify the program so that when a ball is clicked with the mouse, the ball will be exploded and destroyed. The page will thus become a game in which the player can destroy the balls by clicking them with the mouse.

By studying the page **MovingBallsWithPointerCanvas.html** you will find out how to react to the pressings of the mouse buttons. The program can work so that when a mouse button is pressed down, each ball will be 'asked' with the `contains_point()` method that does the clicked point belong to the ball area. If yes, the ball will be exploded.

In the original version of the program, the `explode_ball()` method is used when the Esc key is pressed. After the `explode_ball()` method is called, the drawing methods of the classes will automatically take care that the ball explodes and disappears slowly.

Exercise 5:

Improve the game so that the balls start moving only after the Space key of the keyboard has been pressed. (The key code of the space key is 32.) This can be done quite easily when you first define a global variable such as

```
var game_is_being_played = false ;
```

and you give this variable the value `true` after the space key has been pressed.

You should test the value of the above variable in the `run_this_application()` method, and move the balls only if the game is being played. It is not harmful to draw the balls when they are not moving. The balls should start moving in random directions after the pressing of the Space key.

As the pressing of the Space key is a kind of start for the game, you should also ensure that the balls will not be exploded before the game starts.

Exercise 6:

Improve the game so that when all balls have been exploded and destroyed, the program recognizes this, and it deletes the destroyed balls from the array of game balls and creates new balls to the array. After this the program can start again to wait for the pressing of the Space key, i.e., the start of a new game.

To do this exercise you must use a method named `is_exploded()` that is available in the `ExplodingBouncer` class. This method returns `true` or `false` depending on if the ball state is `BALL_EXPLODED`. By using this method you can check, before moving a ball, whether the ball has exploded.

To find out if all balls have exploded, one possibility is to count the number of exploded balls and compare that to the length of the array that contains the balls.

You should note that a ball does not become exploded immediately after the `explode_ball()` method has been called. It will take some time for an explosion to proceed.

Exercise 7:

To learn something about so-called singleton classes, add one more feature to this game. The game should count how many times the balls hit the ’walls’ of the bouncing area. You should copy the class named `CollisionCounter` that is available in the program

`http://www.naturalprogramming.com/jsprograms/nodejsfilesextra/SingletonClassDemo.js`

When you add the to the `Bouncer` class the following data field

```
this.collision_counter  =  CollisionCounter.get_instance() ;
```

each ball can refer to a single `CollisionCounter` object and increment the counter when a collision with a wall occurs.

Then, when you specify the following object in the ’main’ program

```
var collision_counter = CollisionCounter.get_instance() ;
```

you can refer to the single collision counter, and display the collision counter in the `draw_on_canvas()` function in the following way

```
context.fillStyle = "Black" ;

context.fillText( "Collisions: " + collision_counter.get_count(), 20, 20 ) ;
```

The collision counter must be reset when a new game starts.