Programming in JAVASCRIPT

LAB 5

Environmental configuration

1. We use a previous model project named *NameLabJS.zip*
2. Similarly, change the name of the catalogue according to the formula.
3. The works operating on the view are performed in the section where id=*"Project 5".*

JQuery: Animations

In this laboratory we will use the *animate()* function to move the *<div>* tag, which is initially hidden outside the left edge of the page. This marker is positioned in an **absolute** manner. Usually, when we hide an element of a page, the other elements of the page are moved to fill its previous position. For example, if you hide an image, it will disappear from the page and the text below it will be moved upwards.

It may happen that we do not want the content to jump up or down. In this case we can use **CSS style** sheets and **absolute positioning possibilities.**

We can make the *<div>, <p> or <img>* marker appear above the rest of the page content as if they were placed on a separate layer.

We can achieve this effect by using the *position* property and assign an *absolute* value to it.

For example.

```
#login {
    position: absolute;
    left: 536px;
    top: 0;
    width: 400px;
}
```

Task 1: Preparation of an animated navigation bar.

Copy the photos attached to the laboratory to the image catalogue of your project, attached files with plug-ins to the script catalogue.

We will add two plugins to the project.

The first of the repositories:

```
<script src="http://code.jquery.com/color/jquery.color-2.1.2.min.js"></script>
```

The second of the attached materials:

```
<script src="js/jquery.easing.1.3.min.js"></script>
<script src="js/jquery.easing.compatibility.js"></script>
```

Place in the project somewhere under *<body>*

index.html

```
<div id="dashboard">
  <img src="img/small/blue.jpg" width="70" height="70" alt="blue">
  <img src="img/small/green.jpg" width="70" height="70" alt="green">
  <img src="img/small/orange.jpg" width="70" height="70" alt="orange">
  <img src="img/small/purple.jpg" width="70" height="70" alt="purple">
  <img src="img/small/red.jpg" width="70" height="70" alt="red">
  <img src="img/small/yellow.jpg" width="70" height="70" alt="yellow">
</div>
```

Check that the links to the images work by launching the project in your browser and then insert them into the style file:

syles.css

```
#dashboard {
      width: 112px;
      background-color: rgb(110,138,195);
      padding: 20px 20px 0 20px;
      position: absolute;
      left: -92px;
      z-index: 100;
}
#dashboard img {
      margin-bottom: 20px;
      border: 1px solid rgb(0,0,0);
}
```

Place the function `hover()` inside the `$(document).ready()`function:

index.html

```
$(document).ready(function() {
      $('#dashboard').hover(); // koniec funkcji hover
}); // koniec funkcji ready
```

Calling the *$('#dashboard')* will download the *<div>* tag *(with* dashboard ID*).*

The *hover()* function requires two arguments - two anonymous functions - describing what to do when placing the mouse pointer in the item area and when removing it.

Between the brackets in the call of *hover()* add two empty anonymous functions:

index.html

```
$(document).ready(function() {
  $('#dashboard').hover(
    function() {
    }, // koniec pierwszej funkcji anonimowej
    function() {
    } // koniec drugiej funkcji anonimowej
  ); // koniec funkcji hover
}); // koniec funkcji ready
```

Inside the first anonymous function, type:

```
$(this).animate(); // end of animate function.
```

And add an object letter specifying the CSS properties you want to animate:

index.html

```
$(document).ready(function() {
  $('#dashboard').hover(
    function() {
      $(this).animate({
        left: '0',
        backgroundColor: 'rgb(255,255,255)'
      }); // koniec funkcji animate
    }, // koniec pierwszej funkcji anonimowej
    function() {
    } // koniec drugiej funkcji anonimowej
  ); // koniec funkcji hover
}); // koniec funkcji ready
```

The first argument to call the *animate()* function is an object letter specifying the animated CSS properties. In our case, the current value of the *Left-hand property of* the *<div>* tag is -92px, *which* means that a significant part of it is hidden outside the left-hand edge of the browser window.

If we change the value of this property to *0 as part of the* animation, we will move the item to the right and display it in its entirety on the page. Similarly, by using the *colour* plug-in, we can smoothly change the background colour of this marker from blue to white.

The next step will be to determine the duration of the animation.

Behind the closing curly bracket } enter a comma, press the Enter key and enter *500*.

A comma indicates the end of the first argument passed in an *animate()* function call, while the number *500* indicates the duration of the animation (expressed in milliseconds).

Now you can give the method for determining the pace of animation.

Behind the number *500*, enter a comma, press the Enter key and type *'easeInSine'* so that the code looks the same as shown below:

index.html

```
$(document).ready(function() {
  $('#dashboard').hover(
    function() {
      $(this).animate({
        left: '0',
        backgroundColor: 'rgb(255,255,255)'
      },
      500,
      'easeInSine'
      ); // koniec funkcji animate
    }, // koniec pierwszej funkcji anonimowej
    function() {
    } // koniec drugiej funkcji anonimowej
  ); // koniec funkcji hover
}); // koniec funkcji ready
```

The last argument for calling the *animate()* function - in our case it is in the form of *'easeInSine'* - dictates the method of determining the speed of animation, which makes the animation start quite slowly and then accelerate.

Save the file, display the page in your browser and point the *<div>* tag *with your mouse*. The marker should move to the right and appear in full on the page.

Now we will add the hiding effect of the navigation bar.

Place the following piece of code in the second anonymous function:

index.html

```
$(this).animate(
  {
    left: '-92px',
    backgroundColor: 'rgb(110,138,195)'
  },
  1500,
  'easeOutBounce'
); // koniec funkcji animate
```

The above code reverses the changes made by the first animation - it moves the item to the left, beyond the edge of the browser window and again changes its background to blue. In this case, the animation takes a little longer - one and a half seconds rather than half - and we use a different method of determining the tempo.

Save the file. Display it in the browser, point the `<div>` marker *with the mouse*, and then remove its indicator from the item area. As you see, the *`<div>` marker is* moved to and from the page.

However, if you try to place the mouse pointer several times and quickly in its area and then remove it, you will notice some strange behaviour: the marker will be slid into and out of the page long after you have stopped moving the mouse pointer.

The reason for this problem is the way jQuery queues up the animations performed. All animations operating on a certain element go to a special, related queue. If, for example, we want to display an element gradually and then fade it out, jQuery will perform each of these effects in order, one by one.

The solution to this problem is to stop all marker animations before the next one begins. In other words, if you place the mouse pointer in the area of the marker that is currently being animated, the animation should be interrupted and then the next one started, according to how the `mouseover` event is handled. The jQuery library provides a feature that allows you to interrupt the current animation in this way; this is `. stop()`.

Add a `.stop()` call between `$(this)` and . `animate`.

Calling the stop() function causes all animations performed on the `<div>` element to stop *before the* next one begins and does not allow more animations to be queued.

```
$(this).stop().animate(...
```

TASK 2.

Now we will add a *hover()* function to each drawing, which is to swap drawings after placing the mouse cursor over the picture and then restore the original version of the picture after moving the cursor to another place on the page.

index.html

```
$('#dashboard img').each(function() {
    $(this).hover(
      function() {

      },
      function () {

      }
    ); // Koniec funkcji hover.
  }); // Koniec funkcji each.
```

The jQuery library's *each()* function allows you to easily move in a loop through a collection of elements and perform a series of operations on them. This function takes an anonymous function as an argument.

Add a line before hover():

```
let imgFile = $(this).attr('src');
```

The construction *$(this)* indicates the element currently being processed in the loop. This means that it corresponds to each *<img>* tag in turn. The *attr()* function of the jQuery library takes the specified HTML attribute (in this script it is the *src* attribute of the drawing) and saves it in the *imgFile* variable. The *src* property of the first picture has the value *img/small/blue.jpg* and is the path to the image file visible on the page.

Then add the lines:

```
let preloadImage = new Image();
let imgExt = /($3.4)/;
```

```
preloadImage.src = imgFile.replace(imgExt,'_h$1');
```

The script creates a *preloadImage* variable of *Image* type and stores such an object in it. The code then preloads the drawing by assigning a value to the *src* property of the Image object.

In this task, each drawing on a page is replaced by an image which has a suffix *"_h"* in its name. For example, the *blue.jpg* image is replaced by the *blue_h.jpg* image. Both files are in the same directory, so the same path leads to them.

This solution: instead of manually entering the value of the *src* property to preload the drawing (for example, *preloadImage.src='img/small/blue_h.jpg'*), it sets the required value based on the *src* property of the original drawing.

In other words, if we know the path to the image displayed on the page, it is sufficient to add an underscore character and the letter h to the page immediately before the extension. So:

img/small/blue.jpg will be converted to *img/small/blue_h.jpg,*

img/small/oranges.jpg on *img/small/oranges_h.jpg.*

And it is this operation that is performed by two consecutive lines of code. The first of them:

let imgExt = /(\w{3,4}$/; - creates a regular expression.

Regular expressions are patterns of characters that can be searched for in other chains; an example of such a pattern can be three digits after one another. Regular expressions can be difficult and complex, but the one used in this example corresponds to a dot sign preceding any three letters at the very end of the string. For example, this pattern will correspond to the string *. jpg* in the */img/small/blue.jpg* chain.

In the next line:

preloadImage.src = imgFile.replace(imgExt,'_h$1');

the *replace()* method was used to replace the text found with another passage. For example, the script changes ".jpg" in the path to "_h.jpg", i.e. replaces the name *img/small/blue.jpg* with *img/small/blue_h.jpg.*

After the initial loading of new images, the event *hover()* can be assigned to the drawings.

In the first anonymous function above, add the code:

```
$(this).attr('src', preloadImage.src);
```

Its task is to change the src properties of the original drawing to the src properties of the new picture.

In the second anonymous function, add a code:

```
$(this).attr('src', imgFile);
```

Its task is to restore the *src* attribute from the original drawing.

Task 3. FancyBox Gallery.

Finally, we will use a great gallery display plugin - *FancyBox*

Add to plug-ins:

```
<link rel="stylesheet"
href="_COPY0@3.5.7/dist/jquery.fancybox.min.css" />
<script src="_COPY0@3.5.7/dist/jquery.fancybox.min.js"></script>
```

And **all** images are surrounded by a link to their larger counterpart according to the pattern:

```
<a data-fancybox="gallery" href="img/large/blue.jpg" rel="gallery"
title="BLUE">
<img src="img/small/blue.jpg" width="70" height="70" alt="blue">
</a>
```

And that's it. ☺

ZIP THE PROJECT AND PLACE IT ON MSTeams Platform