Programming in JAVASCRIPT

LAB 3

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 whereid=*"Project 3".*

4. jQuery selectors

4.1. Identifiers

Each page element that has a specific identifier (id value) can be downloaded using the identifier selector. Let's assume that the website contains HTML code in the following form:

```
<p id="message">Special message</p>
```

To download such an item using the old method using only methods available in the DOM, the following code should be used:

```
varmessagePara = document.getElementById("message");
```

A similar call using the jQuery library has the following form:

```
varmessagePara = $('#message');
```

Unlike the way the DOM method is used, for the jQuery selector it is not enough to provide the item identifier itself *('message')* - use the full CSS selector *('#message')*.

4.2. Elements

The jQuery library also provides its own replacement for the DOM method: *getElementsByTagName()*

For example, to retrieve all *<a>* tags placed on a page using the old DOM method, you would need to use the call below:

```
varlinksList = document.getElementsByTagName('a');
```

After using jQuery, the corresponding call is as follows:

```
varlinksList = $('a');
```

4.3. Classes

jQuery provides a very simple way of downloading all elements belonging to the class with the given name. To do this, just use the CSS class selector, which has the following form:

```
$('.submenu')
```

Also in this case it is worth noting that the CSS class selector used in the jQuery call looks like a CSS class selector - that is, it consists of a class name preceded by a dot. After downloading the tags you can operate them using other jQuery capabilities. For example, to hide all tags belonging to a .submenu class, you can use the following call:

```
$('.submenu').hide();
```

TASK 1. jQuery foundation

Add the *pq* class and *pullquote* class to your CSS file. Example:

styles.css

```
.pullquote {
	float: right;
	clear: right;
	width: 200px;
	padding: 10px;
	font-size: 20px;
	background-color: rgba(255, 211, 224, 0.5);
	border-radius: 10px;
	margin: 20px 0 10px 10px;
	font-style: italic;
}
```

Add the class and configure it yourself. It should be a plain text. The *pullquote* class will be a quote. Modify it according to your preferences.

Then add a <script> tag and start jQuery.

index.html

```
<script>
	$document.ready(function(){

	}); // koniec funkcji ready
</script>
```

The comment *// end of theready function* will prove particularly useful in the future, when our program will become much larger and more complex. Larger programs will often have character sequences*});* each of which will mean the end of an anonymous function and calling some other function. The comments behind them make it possible to identify each one and make it much easier for us to understand what it is all about when we return to such code in the future.

In the *id="Project X"* section place a few paragraphs *<p> and* texts - any *content*. Let it be some quotes, sentences, proverbs etc. Assign a class *"pq"* to some.

We download all tags containing the texts we want to display as highlighted quotes. Add bold text from the example below:

index.html

```
<script>
	$(document).ready(function() {
		$('span.pq')
	}); // koniec funkcji ready
</script>
```

The expression *$('span.pq')* is the jQuery selector that allows you to download all tags belonging to pq class. Now we will add the code needed to view tags *and* perform operations on them.

Add bold text from the example below:

index.html

```
<script>
	$(document).ready(function() {
		$('span.pq').each(function() {

		}); // koniec funkcji each
	}); // koniec funkcji ready
</script>
```

each() is a jQueryperformance function that allows you to view a collection of selected elements of the site. It requires a single argument - an anonymous function (a function that has no name).

We will now process the next tags we have collected. The first task will be to create a copy of the processed ** element.

index.html

```
<script>
	$(document).ready(function() {
		$('span.pq').each(function() {
			var quote=$(this).clone();
		}); // koniec funkcji each
	}); // koniec funkcji ready
</script>
```

The *.clone()* function of the jQuery library replicates the current element, including the entire HTML code inside it. In this case we create a copy of the ** tag, *including the* text inside it, which we want to display as a highlighted quote.

Cloning of elements shall cause them to be copied in their entirety, including any attributes. In our case the copied ** tag belongs to a class named pq. In the next step we remove this class from the copied tag.

index.html

```
<script>
	$(document).ready(function() {
		$('span.pq').each(function() {
			var quote=$(this).clone();
			quote.removeClass('pq');
			quote.addClass('pullquote');
		}); // koniec funkcji each
	}); // koniec funkcji ready
</script>
```

removeClass() removes the specified class name from the tag, while *addClass()* adds the specified class name to the tag. In this case, the class name change operation is performed on a copy of the ** tag, so we can use a CSS class called pullquote *to* format the copied tag and give it the appearance of a highlighted quote.

After each change, try to keep track of the page.

The next step will be to add a tag to the website code.

index.html

```
<script>
	$(document).ready(function() {
		$('span.pq').each(function() {
			var quote=$(this).clone();
			quote.removeClass('pq');
			quote.addClass('pullquote');
			$(this).before(quote);
		}); // koniec funkcji each
	}); // koniec funkcji ready
</script>
```

The bold line of code is the last element of the function - so far we have operated on a copy of the tag stored in the memory of the browser. The user wouldn't see the function until the page document objects are added to the model. In this step we add a copy of the tag; place it in the HTML code of the page immediately before the original element. The resultant

The website will contain HTML code in the following form:

```
<span class="pullquote">... nie ma problemów nie do rozwiązania. Są
tylko problemy, którym poświęciliśmy za mało czasu.</span>
<span class="pq">... nie ma problemów nie do rozwiązania. Są tylko
problemy, którym poświęciliśmy za mało czasu.</span>
```

Although the code may suggest that the two text fragments will be placed right next to each other on the page presented in your browser, the CSS styles used will make the quote stand out and display at the right edge of the page.

5. Events

An event represents the moment when a particular phenomenon occurs. For example, when you click an item with the mouse, when you release the button, the browser will signal the occurrence of the click event. Programmers call the moment when the browser informs you that an event has occurred a report.

When the browser clicks, they report several events. First, immediately after pressing the button, they report the *mousedown* event. Then, when you release the button, they report the *mouseup* event and then the *click* event.

The *click* event is also triggered when you select a link using the keyboard. If you access the link using the *Tab* key and then press the *Enter* key, the browser will report the *click* event.

1.1. Events related to the document and the window

load - reported after the browser has loaded all page files: the HTML file as well as attached drawings, videos and external CSS and JavaScript files.

resize - when you change the size of the browser window by clicking on the maximize button or dragging the window edge

scroll - reported after dragging the slider about- using the keyboard (up and down arrows, Home or End keys, and so on) or the scrolling mouse wheel.

unload - when you click a link to go to a new page, you will close the bookmark or browser window

1.2. Events related to forms

submit-to-event is reported when submitting the form.

reset - allows you to cancel all changes made to the form and restore it to its original state

change - many fields in the form report the *change* event when the state changes, for example by clicking an option button or selecting a link from the drop-down menu

Focus - when you click on a text field or navigate to it using the *Tab* key, you activate it. This means that your browser will "focus" on this field.

Blur - that is the opposite of a *focus* event. The browser reports the *blur* event when the user leaves the active field with the *Tab* key or a mouse click.

1.3. Events related to the keyboard

keypress - the event is reported when the key is pressed.

keydown - the event works similarly to a *keypress* event - it is reported when a key is pressed. It occurs just before the *keypress* event.

keyup is an event that the browser reports when you release the key.

1.4. Event assignment.

In jQuery most of the events of the DOM model correspond to library functions of the same name. Therefore, to assign an event to an item, you just need to add a dot, an event name and a pair of brackets. For example, if you want to add a *mouseover* event to every link on a page, you can do it as follows:

```
$('a').mouseover();
```

The following code adds the *click* event to the item with the *menu* identifier:

```
$('#menu').click();
```

In this way, all events can be used. However, adding an event is not the end of the job. For a page to react to an event report, you need to assign a function to it.

1.5. Forwarding functions to an event.

The name of the previously defined function can be passed on to the event:

```
$('#start').click(startSlideShow);
```

When assigning functions to events, parentheses should be omitted, as standard placed after a function name to call it. This means that the following entry is **incorrect:**

~~`$('#start').click(startSlideShow());`~~

However, the most common way to handle events is to assign anonymous functions to them:

```
$('#menu').mouseover(function() {
      $('#submenu').show();
}); // endmouseover
```

Task 2. Events.

1. Display a dialog each time you double-click anywhere on the page. Using the *dbclick()* event

2. Add some links in *id="Project 3"* and make a new paragraph appear when you hover your mouse over them (not click). Use the code below:

```
let message = "<p>You have made the link with the mouse!</p>";
$('.main').append(message);
```

3. In section id=*"Project 4"* add a form and in it the button *"click me"*, which after clicking will change the text on it. Use the code:

```
$(this).val("I'm clicked!");
```

4. In the above form, add an additional 10 numbered (in the name) buttons, which will change the style (colour, font, font size + any attribute) of any elements on the page after clicking, and after clicking, they will add a paragraph in the *'.main'* section with information about what exactly they changed and what is the TAG of the element being changed. After clicking, the button changes its subtitle to: *"NR. clicked"*. *NR* is the button number. Use the new classes from the CSS file.

ZIP THE PROJECT AND PLACE IT ON MSTeams Platform