# Advanced JavaScript Lab: Poppy

The goal of this lab is to practice the creation of a **real-world** JavaScript **notification library**, using the best practices of **quality JavaScript**, without any help of external libraries such as jQuery.

### The Library

**Poppy** is (will be) **your very own library** for displaying notifications (**popups**) whenever something happens in your webpage - you'll see how useful and awesome that is in the **JS Apps** course!



For now, let's stick to developing our library. It should support the creation of **4 main popups**:

|  |  |  |
| --- | --- | --- |
| **Type** | **Effects** | **Visual Output** |
| Success | * fade-in when created * position: bottom left |  |
| Info | * fade-in when created * close button ➔ fade-out after a period of time * position: top left |  |
| Error | * fade-in when created * automatic fade-out after a period of time * position: top right |  |
| Warning | * fade-in when created * on-click function (callback) execution * position: bottom right |  |

So generally, a **popup** has the following properties: **title**, **message**, **type**(*success*, *info*, *error*, *warning*), **autoHide**(holds if the popup should fade out after a period of time), **timeout**(the time the popup should fade out after), **closeButton**(holds if the popup should have a 'x' button), **callback**(a callback function that is executed at a certain event - e.g. when the popup is clicked).

The library interface for the end-user should be as follows:

|  |  |
| --- | --- |
| **Source code** | **Sample Visual Output** |
| poppy.pop('*success*', ***title***, ***message***); poppy.pop('*info'*, ***title***, ***message***); poppy.pop('*error*', ***title***, ***message***); poppy.pop('*warning',****title***, ***message***, ***callback***); |  |

### The Source Code

You are given the following components:

* **index.html** - the HTML file of the test page
* **style.css** - the stylesheet
* **view-factory.js** - the view factorycreates the **HTML representation** of the popups. It receives an object (holding data about the popup), builds the **HTML** and returns it. Study how it works inside.
* **core.js** - the core is responsible for the main logic of the application.
* **data.js** - this script is left empty for future implementation of the popups.

This structure leaves several things in mind - the **view factory** is obviously responsible for **generating the HTML** of the popups. For it to do that, it must be given **data** (which will most likely be stored in **data.js**). And who will **get that data**, **pass it to the view factory** and **process the returned HTML**? - the **controller**, of course! It is responsible for the core logic after all.

### Part 1 - Building the Data Module

Let's start with the **data** script and create the objects we will be working with. Create the necessary **function constructors** for **Success**, **Info**, **Error** and **Warning** popups (use the **table** above for reference of the behavior for the various popups). The objects created should hold **info** for the respective popup.

Avoid repeating code though code reuse techniques. Make sure **data.js** properly reveals itself without needlessly polluting the global scope.

### Part 2 - Test the Popups

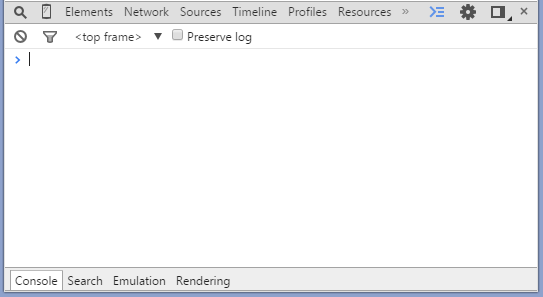
Open **core.js** and create a few objects using the **function constructors** from the **data** script. See if they correctly hold the necessary data (e.g. for a Success popup **autoHide = false**).

### Part 3 - Generating HTML

Now that the created objects seem valid, let's test if the view factory generates the correct HTML. Pass several objects to the view factory. Add the returned HTML to DOM and see if it appears correctly.

**Tip:** Search the internet on how to **add HTML elements to DOM**.

The console should not contain any errors.



### Part 4 - Adding HTML Behavior

So now that we can add popups to the DOM, it's time we attached some effects to them. Effects are part of the logic, so we'll write them in **core.js**.

#### Step 1 - Closing Info Popups

Implement a **click** event on the '**x**' **button**. It should **remove** the popup from DOM.

**Tip**: Search the Internet on how to **get a specific element from DOM**. Use the **addEventListener()** function to attach a click event on that element.

#### Step 2 - Popups Fading in when Added to DOM

Implement **auto fade in effect** whenever a popup is **added** to DOM - the element should gradually increase its opacity until it becomes entirely visible.

#### Step 3 - Popups Fading out when Removed from DOM

Implement **auto fade out effect** whenever a popup **times out** (it auto hides after a period of time) or when it is **removed** - the element should decrease its opacity until it becomes entirely invisible.

Error popups should fade out and be removed after they time out. Info popups should fade out and be removed when clicked.

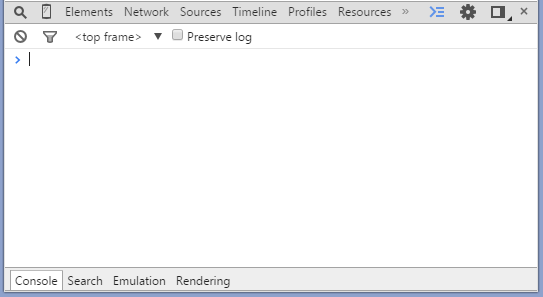
**Tips**: Search the **setInterval()** and **setTimeout()** functions and see how they work. The fade in/out effects can be achieved by manipulating the **opacity property of DOM elements**.

#### Step 4 - Popups with Callback

Implement a **click event** on popups who have a **callback** function. When the popup is clicked, the callback should be executed. In this case, on click the user should be redirected:

|  |  |
| --- | --- |
| poppy.*pop*('warning', 'Attention!', 'You are our 100th visitor.', *redirect*);  function *redirect*() {  window.location = 'https://www.youtube.com/watch?v=HMUDVMiITOU'; } |  |

The console should not contain any errors.



### Part 5 - Improve the Code Quality

See if any **improvements** to the code can be made. Look for the following **code smells**:

* Files revealing too much -> wrap them in **modules**
* Magic values -> declare them as **constants**
* Long functions -> break down in **several** **smaller** **functions**
* Repeating code -> **move it out** in a function and **reuse** it
* Bad variable/function names -> **rename** them

### Part 6 - Publish Your Poppy

Poppy should now be a fully functional JS notification library. You might as well upload it in your **GitHub** account and use it in the future - just make sure you give credits to the [toastr](https://github.com/CodeSeven/toastr) library (the HTML & CSS we used are theirs) and license it under [MIT license](http://opensource.org/licenses/mit-license.php).