Objectives

- Javascript Strikes Back

Preparation

Before the lab you should re-read the relevant lecture slides and their accompanying examples.

Getting Started

Create a new directory for this lab called `lab08` and change to this directory with these comamnds:

```
$ mkdir lab08
$ cd lab08
```

Exercise: Counter

If you are working at CSE, explode the zip file containing the files for this activity:

```
$ unzip /web/cs2041/19T2/activities/js_counter/counter.zip
```

If you are working on your computer, download counter.zip and unzip it .

Then in one window use Python to start a HTTP server for the exercises

```
$ cd counter
$ python3 -m http.server
Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...
```

If you access the URL printed by Python you will see instructions for the exercise.
Remember to refresh the web page in your browser (`control-R` inchrome and firefox) when you make a change to the Javascript so it gets loaded. You can stop the server by pressing `control-c`

You do not change **index.html** but you should read its source:

```
$ cd counter
$ more index.html
```

Notice **index.html** loads **counter.js**. You need to edit **counter.js**

```
$ vi counter.js
```

Your task is to add JavaScript to **counter.js** so that the current time is shown on the web page.
Your Javascript should put the current time in the **output** div.

The time should be updated every second. There is no autotest and no automarking of this question.

When you have completed demonstrate your work to another student in your lab and ask them to enter a peer assessment here

It is prefered you do this during your lab, but if this is not possible you may demonstrate your work to any other COMP(2041|9044) student before Tuesday 30 July 17:59:59. Note, you must also submit the work with give.

When you are finished working on this exercise you must submit your work by running **give**:

```
$ give cs2041 lab08_js_counter counter.js
```

Sample solution for `counter.js`

```
(function() {
   'use strict';
   const output = document.getElementById('output');

   void setInterval(() => {
      const d = new Date();
      output.innerText = `${d.getHours()}:${d.getMinutes()}:${d.getSeconds()}`;
   }, 1000);

}());
```

Exercise: Toggle

If you are working at CSE, explode the zip file containing the files for this activity:

```
$ unzip /web/cs2041/19T2/activities/js_toggle/toggle.zip
```

If you are working on your computer, download toggle.zip and unzip it .

Then in one window use Python to start a HTTP server for the exercises

```
$ cd toggle
$ python3 -m http.server
Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...
```

If you access the URL printed by Python you will see instructions for the exercise.
Remember to refresh the web page in your browser (`control-R` inchrome and firefox) when you make a change to the Javascript so it gets loaded. You can stop the server by pressing `control-c`

You do not change **index.html** but you should read its source:

```
$ cd toggle
$ more index.html
```

Notice **index.html** loads **toggle.js**. You need to edit **toggle.js**

```
$ vi toggle.js
```

Your task is to modify the code in **toggle.js** so that it adds and removes the class 'hide' from the `output` element every 2 seconds.

There is no autotest and no automarking of this question.
When you have completed demonstrate your work to another student in your lab and ask them to enter a peer assessment here

It is prefered you do this during your lab, but if this is not possible you may demonstrate your work to any other COMP(2041|9044) student before Tuesday 30 July 17:59:59. Note, you must also submit the work with give.

When you are finished working on this exercise you must submit your work by running **give**:

```
$ give cs2041 lab08_js_toggle toggle.js
```

Sample solution for `toggle.js`

```
(function() {
   'use strict';
   // write your js here.
   const d = document;
   const output = d.getElementById('output');

   setInterval(() => {
      output.classList.toggle('hide');
   }, 2000);
}());
```

Exercise: Collapse

If you are working at CSE, explode the zip file containing the files for this activity:

```
$ unzip /web/cs2041/19T2/activities/js_collapse/collapse.zip
```

If you are working on your computer, download collapse.zip and unzip it .

Then in one window use Python to start a HTTP server for the exercises

```
$ cd collapse
$ python3 -m http.server
Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...
```

If you access the URL printed by Python you will see instructions for the exercise.
Remember to refresh the web page in your browser (`control-R` inchrome and firefox) when you make a change to the Javascript so it gets loaded. You can stop the server by pressing `control-c`

You do not change **index.html** but you should read its source:

```
$ cd collapse
$ more index.html
```

Notice **index.html** loads **collapse.js**. You need to edit **collapse.js**

```
$ vi collapse.js
```

Your task is to modify the code in **collapse.js**

I frankly don't like UNSW handbook so I've made my own to outline all the COMP courses there are for me. But the page is a bit cluttered.

Write some JS that makes the "extra info" section of each information element disappear on the click of the "up" arrow button on the top right of each element.

Once collapsed the item doesn't need to reexpand just yet.

Optional Challange: Finish this exercise using only a single click event listener and 0 changes to the HTML.

There is no autotest and no automarking of this question.
When you have completed demonstrate your work to another student in your lab and ask them to enter a peer assessment here

It is prefered you do this during your lab, but if this is not possible you may demonstrate your work to any other COMP(2041|9044) student before Tuesday 30 July 17:59:59. Note, you must also submit the work with give.

When you are finished working on this exercise you must submit your work by running **give**:

```
$ give cs2041 lab08_js_collapse collapse.js
```

Sample solution for `collapse.js`

```
// Challange Solution
//     This is pretty unmaintable code, if we were to add anything
//     to the info cards the code would break but
//     this is one of the cases where you can leverage
//     event bubbling to utilise Js and avoid HTML id tags
//     Take with a grain of salt though
(function() {
  'use strict';
  const main = document.getElementById('main');
  main.addEventListener('click', e => {
    if (e.target.tagName == 'I')
      e.target.parentNode.parentNode.parentNode.children[1].children[2].style.display = 'none';
  });
}());

// This is a standard Solution - relies on editing html
//    Note this code doesn't run, the function is just delcared
//    The code above is a Immediately Invoked Function Expression
//    Which declares a function and then runs it immediately
//    To run it remove the brackets and function invoke from above and
//    shift it down here
function standard() { /* eslint-disable-line no-unused-vars */
  'use strict';
  const NUM_CARDS = 2;
  const cards = Array(NUM_CARDS).fill(0).map((_,i) =>`item-${i+1}`);
  cards.map(card =>
    document.getElementById(card).addEventListener('click', toggle)
  );

  function toggle(e) {
    const content = document.getElementById(`${e.target.id}-content`);
    content.style.display = 'none';
  }
}
```

Exercise: Expand

If you are working at CSE, explode the zip file containing the files for this activity:

```
$ unzip /web/cs2041/19T2/activities/js_expand/expand.zip
```

If you are working on your computer, download expand.zip and unzip it .

Then in one window use Python to start a HTTP server for the exercises

```
$ cd expand
$ python3 -m http.server
Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...
```

If you access the URL printed by Python you will see instructions for the exercise.
Remember to refresh the web page in your browser (`control-R` inchrome and firefox) when you make a change to the Javascript so it gets loaded. You can stop the server by pressing `control-c`

You do not change **index.html** but you should read its source:

```
$ cd expand
$ more index.html
```

Notice **index.html** loads **expand.js**. You need to edit **expand.js**

```
$ vi expand.js
```

Your task is to modify the code in **expand.js**

Extend the collapse code to change the icon from an "up" arrow to a down arrow once the `div` is collapsed. The button should

Extend the collapse code to change the icon from an "up" arrow to a down arrow once the `div` is collapsed. The button should then expand the `div` when clicked on and revert the icon.

Optional Challenge: Finish this exercise using only a single click event listener and 0 changes to the HTML.

There is no autotest and no automarking of this question.
When you have completed demonstrate your work to another student in your lab and ask them to enter a peer assessment here

It is prefered you do this during your lab, but if this is not possible you may demonstrate your work to any other COMP(2041|9044) student before Tuesday 30 July 17:59:59. Note, you must also submit the work with give.

When you are finished working on this exercise you must submit your work by running **give**:

```
$ give cs2041 lab08_js_expand expand.js
```

Sample solution for `expand.js`

```
// Standard Solution
//    this relies on adding id's into the html

function standard() { /* eslint-disable-line */
  'use strict';
  const NUM_CARDS = 2;
  const cards = Array(NUM_CARDS).fill(0).map((x,i)=>`item-${i+1}`);

  cards.map((x)=>document.getElementById(x).addEventListener('click',toggle));

  function toggle(e) {
    const content = document.getElementById(`${e.target.id}-content`);
    content.style.display = (content.style.display == 'none') ? 'block' : 'none';
    e.target.innerHTML = (e.target.innerHTML == 'expand_more') ? 'expand_less' : 'expand_more';
  }
}

// Challenge Solution
//     Again , take with a grain of salt though
function challenge() { /* eslint-disable-line */
  'use strict';
  const main = document.getElementById('main');
  main.addEventListener('click',(e)=>{
    // see if the event originated from the button
    if (e.target.tagName == 'I') {
      let state = null;
      if(e.target.innerHTML == 'expand_more'){
        e.target.innerHTML = 'expand_less';
        state = 'block';
      } else {
        e.target.innerHTML = 'expand_more';
        state = 'none';
      }
      // traverse the dom tree to find the content section and toggle it's display
      e.target.parentNode.parentNode.parentNode.children[1].children[2].style.display = state;
    }
  });
}

// Since we didn't wrap these functions and execute them immedietly i.e
//        (function(){console.log('hello workd')})();
// we need to ask the browser to run one of them once the page is loaded.
// the bottom acheives the same effect as window.addEventListener('load', standard)

document.addEventListener('DOMContentLoaded', challenge);
// swap 'standard' to 'challange' above to run the challange Solution
```

Challenge Exercise: js_adventure_script

If you are working at CSE, explode the zip file containing the files for this activity:

```
$ unzip /web/cs2041/19T2/activities/js_adventure_script/adventure_script.zip
```

If you are working on your computer, download adventure_script.zip and unzip it .
Then in one window use Python to start a HTTP server for the exercises

```
$ cd adventure_script
$ python3 -m http.server
Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...
```

If you access the URL printed by Python you will see instructions for the exercise.
Remember to refresh the web page in your browser (`control-R` inchrome and firefox) when you make a change to the Javascript so it gets loaded. You can stop the server by pressing `control-c`

You do not change **index.html** but you should read its source:

```
$ cd adventure_script
$ more index.html
```

Notice **index.html** loads **adventure_script.js**. You need to edit **adventure_script.js**

```
$ vi adventure_script.js
```

Your task is to modify the code in **adventure.js**

There is the basic framework for a simple game, fill out the following functions:

1. Make the player (the ghost) move left and right via the left and right arrow keys.
2. Have the 'z' key toggle the player between walk and sprint mode (slow and fast movement).
3. Have the 'x' key shoot a fireball towards the right of the screen. There is a provided image in `imgs/`. To make things simple the player can only have 1 fireball on the screen at any one time.

There is no autotest and no automarking of this question.
When you have completed demonstrate your work to another student in your lab and ask them to enter a peer assessment here

It is prefered you do this during your lab, but if this is not possible you may demonstrate your work to any other COMP(2041|9044) student before Tuesday 30 July 17:59:59. Note, you must also submit the work with give.

When you are finished working on this exercise you must submit your work by running **give**:

```
$ give cs2041 lab08_js_adventure_script adventure.js
```

Sample solution for `adventure.js`

```
(function() {
   'use strict';
   const player = {
     dom: document.getElementById("player"),
     fb: document.getElementById("fb"),
     fbp: -1,
     x: 0,
     y: 0,
     speed: 10,
     paint: function() {
       this.dom.style.transform = `translateX(${this.x}px) translateY(${this.y}px)`;
       if(this.fbp === -1)
        this.fb.style.transform = `translateX(${this.x+64}px) translateY(${this.y}px)`;
     },
     shoot: function() {
       console.log(this.fbp);
       if(this.fbp !== -1) return;
       this.fb.style.display = "block";
       this.fbp = this.x+64;
       let i = setInterval(function(){
         this.fb.style.transform = `translateX(${this.fbp}px) translateY(${this.y}px)`;
         this.fbp+=5;
       }.bind(this), 5);
       setTimeout(function(){
         clearInterval(i);
         this.fbp = -1;
         this.fb.style.display = "none";
       }.bind(this),1000);
     }
   }
   window.addEventListener('keydown',(e)=>{
     if(e.key === 'ArrowLeft') player.x-=player.speed;
     if(e.key === 'ArrowRight') player.x+=player.speed;
     if(e.key === 'z') player.speed = player.speed === 20 ? 10 : 20;
     if(e.key === 'x') player.shoot();
     player.paint();
   })

}());
```

Submission

When you are finished each exercises make sure you submit your work by running **give**.
You can run **give** multiple times. Only your last submission will be marked.

Don't submit any exercises you haven't attempted.

If you are working at home, you may find it more convenient to upload your work via give's web interface.

Remember you have until **Tuesday 30 July 17:59:59** to submit your work.

You cannot obtain marks by e-mailing lab work to tutors or lecturers.

You check the files you have submitted here

Automarking will be run by the lecturer several days after the submission deadline for the test, using test cases that you haven't seen: different to the test cases `autotest` runs for you.

(Hint: do your own testing as well as running `autotest`)

After automarking is run by the lecturer you can view it here the resulting mark will also be available via via give's web interface

Lab Marks

When all components of a lab are automarked you should be able to view the the marks via give's web interface or by running this command on a CSE machine:

```
$ 2041 classrun -sturec
```

The lab exercises for each week are worth in total 1.2 marks.

Usually each lab exercise will be worth the same - for example if there are 5 lab exercises each will be worth 0.4 marks.

Except challenge exercises (see below) will never total more than 20% of each week's lab mark.

All of your lab marks for weeks 1-10, will be summed to give you a mark out of 12.

If their sum exceeds 9 - your total mark will be capped at 9.

Running Autotests On your Own Computer

An experimental version of autotest exists which may allow you to run autotest on your own computer.
If you are running Linux, Windows Subsystem for Linux or OSX. These commands might let you run autotests at home.

```
$ sudo wget https://cgi.cse.unsw.edu.au/~cs2041/19T2/resources/home_autotest -O/usr/local/bin/2041_autotest
$ sudo chmod 755 /usr/local/bin/2041_autotest
$ 2041_autotest shell_snapshot
```

Autotest itself needs Python 3.6 (or later) installed.
Particular autotests may require other software install, e.g. autotests of perl programs require Perl installed (of course).

The legit autotests need python3.7, git & binfmt-support installed.

The program embeds the autotests themselves, so you'll need to re-download if autotests are changed, added, fixed, ...

If it breaks on your computer post on the class forum and we'll fix if we can, but this is very definitely experimental.

COMP(2041|9044) 19T2: Software Construction is brought to you by
the School of Computer Science and Engineering at the University of New South Wales, Sydney.
For all enquiries, please email the class account at cs2041@cse.unsw.edu.au
CRICOS Provider 00098G