Objectives

- Javascript - A New Language

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 `lab07` and change to this directory with these comamnds:

```
$ mkdir lab07
$ cd lab07
```

Exercise: Say Hello to Javascript

Running JS

In this course we focus on running JS in the browser, but for this first lab we just want to get some feeling for the syntax of JS. As such we will be using node to run all of your code, which is a command line version of the JavaScript engine run in Google Chrome.

For these activities you may need to get command line arguments or print to the console. Some sample code is provided below on how you can do this through node, do note that reading command line arguments is a feature unavailable when running JS in the browser.

```
// node script.js one two
const one = process.argv[2]; // will be "one"
const two = process.argv[3]; // will be "two"
console.log("Hello") // prints "Hello" to the terminal
```

Completing Your First JavaScript program

Create a program in a file called **hello_world.js** to prepend a command line argument provided string with "Hello " and print the result. For example

```
$ node hello_world.js world
Hello world
$ node hello_world.js Andrew!
Hello Andrew!
```

Working from Home

You can complete all the Javascript exercises in your own computer.

You'll first need to install **node** on your computer. You can find installation instructions here: https://nodejs.org/en/download/

If you have problem installing **node** ask in the course forum.

You run the same commands on your own computer:

```
$ node hello_world.js world
Hello world
$ node hello_world.js Andrew!
```

```
Hello Andrew!
```

When you think your program is working you can use `autotest` to run some simple automated tests:

```
$ 2041 autotest js_hello_world
```

Autotest Results

98% of **479** students who have autotested **hello_world.js** so far, passed all autotest tests.

- **98%** passed test *0*
- **98%** passed test *1*
- **98%** passed test *2*
- **98%** passed test *3*

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

```
$ give cs2041 lab07_js_hello_world hello_world.js
```

Sample solution for `hello_world.js`

```
// Say Hello to Javascript

// get the first command-line argument

// process.argv[0] will be "node"
// process,argv[1] will be "hello_world.js"
// process,argv[2] will be the first command line argument
//                 or Undefined if there is none

name = process.argv[2];

message = 'Hello ' + name;

// or message = `Hello ${name}`;
// or message = "Hello ".concat(name);
// or ...

console.log(process.argv);
```

Exercise: Say Hello to HTML

First Site

Create a basic HTML file called `index.html` to give the world a bit of information about you. Your site should:

1. Display your Name
2. Display your degree (if you don't wish to share this info, make a degree up, such as Bachelor of napping)
3. Display your favourite programming langauge
4. Display a picture (can be of anything)
5. Have a title, you can give it whatever title you like

Once you have all the elements add some css in a file called `styles.css`, you can go crazy here if you like but at a minimum:

1. Give the image some sort of border
2. Change the font family of the page
3. Change the background of the page, either give it a color or a background image

Once you are done link your styles.css to your index.html, you can preview your site as you build it by visiting the location of the file in the browser of your choice. For example if you saved your index.html and styles.css at `/home/andrew/files/site` you can visit `file:///home/andrew/files/site/index.html` to see the page.

The last step is to get this website live to the world! CSE gives you the ability to have a personal website, simply copy yout index.html and styles.css into your home directory under `public_html`, update the permissions and then visit `http://cgi.cse.unsw.edu.au/~z9999999/` where z9999999 is your zid.

```
$ mkdir -p ~/public_html
$ chmod 755 ~/public_html
$ cp index.html styles.css ~/public_html
$ chmod 644 ~/public_html/styles.css ~/public_html/index.html
```

See Zain's site for a simple example. 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 23 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 lab07_js_first_site index.html styles.css
```

Exercise: Debugging Javascript

Download the following file, variables.js

```
$ wget -q https://cgi.cse.unsw.edu.au/~cs2041/19T2//activities/js_variables/variables.js
```

It contains a function **doubleIfEven** which takes a single number **n** as argument.

If **n** is even, **doubleIfEven** should return double **n**

If **n** is odd, **doubleIfEven** should return **n**

However the code in **variables.js** has bugs:

```
$ node variables.js 3
false // should be 3
$ node variables.js 2
2 // should be 4
```

Your task is to fix the bugs.
When you think your program is working you can use `autotest` to run some simple automated tests:

```
$ 2041 autotest js_variables
```

Autotest Results

99% of **472** students who have autotested **variables.js** so far, passed all autotest tests.

- **99%** passed test *0*
- **99%** passed test *1*
- **99%** passed test *2*
- **99%** passed test *3*

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

```
$ give cs2041 lab07_js_variables variables.js
```

Sample solution for `variables.js`

```
function doubleIfEven(n) {
  let x = n;
  if (even(x)) return double(x);
  return x;
}

function even(a) {
  let x = a;
  if (x%2==0) x = true;
  else x = false;
  return x;
}

function double(a) {
    return a*2;
}

// Get command line argument and run function

const input = process.argv.slice(2).map(x => parseInt(x));
if (input === undefined) {
    throw new Error(`input not supplied`);
}

for (let num of input) {
    console.log(doubleIfEven(num));
}
```

Exercise: Javascript Sports

This zip file contains some starter code and sample sport statistic data you can work with for this activity. Your task is to write the JavaScript function **countStats** in count_sport_stats.js which given a list of career statistics for a team of rugby players returns the total number of matches they have played and the total number of tries they have scored.
countStats will be given a list in this format:

```
[
    {
        "id": 112814,
        "matches": "123",
        "tries": "11"
    }
]
```

countStats should return an object of the form:

```
{
    "matches": m,
    "tries": t
}
```

Where *m* is the sum of all matches for all games and *t* is the sum of all tries for all games.
For example, given this input

```
[
    {"id": 1,"matches": "123", "tries": "11"},
    {"id": 2,"matches": "1",   "tries": "1"},
    {"id": 3,"matches": "2",   "tries": "5"}
]
```

countStats should return

```
{
    matches: 126,
    tries: 17
}
```

To get started extract the files

```
$ wget -q https://cgi.cse.unsw.edu.au/~cs2041/19T2//activities/js_count_sport_stats/count_sport_stats.zip
$ unzip count_sport_stats.zip
$ cd count_sport_stats
```

You can then test your code as such

```
$ node count_sport_stats.js stats.json
{ matches: 1725, tries: 165 }
```

When you think your program is working you can use `autotest` to run some simple automated tests:

```
$ 2041 autotest js_count_sport_stats
```

Autotest Results

97% of **462** students who have autotested **count_sport_stats.js** so far, passed all autotest tests.

- **98%** passed test *0*
- **98%** passed test *1*

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

```
$ give cs2041 lab07_js_count_sport_stats count_sport_stats.js
```

Sample solution for `count_sport_stats.js`

```
// a functional programmming style solution using reduce

function countStats(list) {
   return list.reduce((acc, curr) => ({
      matches: +curr.matches + acc.matches,
      tries: +curr.tries + acc.tries
   }), {
       matches: 0,
       tries: 0
   });
};

const json = process.argv[2];
if (json === undefined) {
  throw new Error(`input not supplied`);
}
// include the json file via node's module system
const stats = require(`./${json}`);
console.log(countStats(stats.results));
```

Alternative solution for `count_sport_stats.js`

```
// an iterative solution

function countStats(list) {
    var total_matches = 0;
  var total_tries = 0;

    for (var i = 0; i < list.length; i++) {
    // The + is needed to convert the string to an intger
    total_tries += +list[i].tries;
    total_matches += +list[i].matches;
    }

    return {
    matches: total_matches,
    tries: total_tries
  };
}

const json = process.argv[2];
if (json === undefined) {
  throw new Error(`input not supplied`);
}
// include the json file via node's module system
const stats = require(`./${json}`);
console.log(countStats(stats.results));
```

Exercise: Javascript Sports 2

This zip file contains some starter code and sample sport statistic data you can work with for this activity. Write a JavaScript function **makeTeamList** which takes 3 arguments: a list of career statistics for a team of rugby players, a list of player names, and a list of team names.
makeTeamList's first argument will describe the team with an object in this format:

```
{
    "players": [
        {
            "id": 112814,
            "matches": "123",
            "tries": "11"
        }
    ],
    "team": {
        "id": 10,
        "coach": "John Simmons"
    }
}
```

makeTeamList's second argument will be a list of player names in this format:

```
[
    {
        "id": 112814,
        "name": "Greg Growden"
    }
]
```

makeTeamList's thirds argument will be a list of team names in this format:

```
[
    {
        "id": 10,
        "team": "NSW Waratahs"
    }
]
```

makeTeamList should returns a 'team sheet' that lists the team, coach, players in that order in the following format:

```
[
    "Team Name, coached by CoachName",
    "1. PlayerName",
    "2. PlayerName"
    ....
]
```

Each element should be a string.

The players should be ordered by the number of matches they have played, from most to least.

For example, given these 3 arguments as input

```
{
    "players": [
        {"id": 1,"matches": "123", "tries": "11"},
        {"id": 2,"matches": "1",   "tries": "1"},
        {"id": 3,"matches": "2",   "tries": "5"}
    ],
    "team": {
        "id": 10,
        "coach": "John Simmons"
    }
}
```

```
[
    {"id": 1, "name": "John Fake"},
    {"id": 2, "name": "Jimmy Alsofake"},
    {"id": 3, "name": "Jason Fakest"}
]
```

```
[
    {"id": 10, "team": "Greenbay Packers"}
]
```

makeTeamList should return a list containing exactly these strings:

```
[
    "Greenbay Packers, coached by John Simmons",
    "1. John Fake",
    "2. Jason Fakest",
    "3. Jimmy Alsofake"
]
```

To get started extract the files

```
$ wget -q https://cgi.cse.unsw.edu.au/~cs2041/19T2//activities/js_join_sport_stats/join_sport_stats.zip
$ unzip join_sport_stats.zip
```

You can then test your code as such

```
$ cd join_sport_stats
$ node join_sport_stats.js team.json names.json teams.json
[ 'NSW Warratahs, coached by Barry Cassidy',
  '1. Ronaldo',
  '2. Buffalo Bill',
  '3. Jesse James',
  '4. Cleopatra I',
  '5. Marc Antony' ]
```

When you think your program is working you can use `autotest` to run some simple automated tests:

```
$ 2041 autotest js_join_sport_stats
```

Autotest Results

98% of **418** students who have autotested **join_sport_stats.js** so far, passed the autotest test.

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

```
$ give cs2041 lab07_js_join_sport_stats join_sport_stats.js
```

Sample solution for `join_sport_stats.js`

```
function makeTeamList(teamData, namesData, teamsData) {
    // Take it step by step.
    let { players, team } = teamData;
    const names = namesData;
    const teams = teamsData;

    // do the first join
    const playerMap = [...players, ...names].reduce((map, current) => {
        const { id, ...rest } = current;
        if (map[id]) {
            map[id] = { ...map[id], ...rest}
        } else {
            map[id] = { ...rest };
        }

        return map;
    }, {});

    const teamName = teams.find(t => t.id === team.id).team

    players = Object.values(playerMap)
        .map(({ name, matches }) => ({
            name,
            matches: parseInt(matches)
        }))
        .sort((a, b) => b.matches - a.matches)
        .map((player, index) => `${index + 1}. ${player.name}`);

    return [`${teamName}, coached by ${team.coach}`, ...players];
}

const teamJson = process.argv[2];
const namesJson = process.argv[3];
const teamsJson = process.argv[4];
if (teamJson === undefined || namesJson === undefined || teamsJson === undefined) {
  throw new Error(`input not supplied`);
}

// some sample data
const team  = require(`./${teamJson}`);
const names  = require(`./${namesJson}`);
const teams  = require(`./${teamsJson}`);
console.log(makeTeamList(team, names.names, teams.teams));
```

Alternative solution for `join_sport_stats.js`

```
// an iterative solution

function makeTeamList(teamData, namesData, teamsData) {

    const team_name = get_team_name(teamData.team.id, teamsData)

    const team_details = team_name + ", coached by " + teamData.team.coach;
    let team_sheet = [team_details];

    const players = teamData.players.sort((a, b) => b.matches - a.matches);

    for (var i = 0; i < players.length; i++) {
        const player_name = get_player_name(players[i].id, namesData);
        team_sheet.push((i + 1) + ". " + player_name)
    }

    return team_sheet
}

function get_team_name(id, teamsData) {
    for (var i = 0; i < teamsData.length; i++) {
        if (id == teamsData[i].id) {
            return teamsData[i].team;
        }
    }
}

function get_player_name(id, namesData) {
    for (var i = 0; i < namesData.length; i++) {
        if (id == namesData[i].id) {
            return player_name = namesData[i].name;
        }
    }
}

const teamJson = process.argv[2];
const namesJson = process.argv[3];
const teamsJson = process.argv[4];
if (teamJson === undefined || namesJson === undefined || teamsJson === undefined) {
  throw new Error(`input not supplied`);
}

// some sample data
const team  = require(`./${teamJson}`);
const names  = require(`./${namesJson}`);
const teams  = require(`./${teamsJson}`);
console.log(makeTeamList(team, names.names, teams.teams));
```

Challenge Exercise: What Have You Drunk?

Download objects.zip and extract it.

```
$ wget -q https://cgi.cse.unsw.edu.au/~cs2041/19T2//activities/js_objects/objects.zip
$ unzip objects.zip
$ cd objects
```

In **objects/objects.js** there is part of the code for a **Person** object,
This object tracks how the person spending on drinks in UNIbar.

Finish up the missing functions to keep track of drink orders.

Firstly you need to write the function **buyDrink** which is given as argument a drink object, for example:

```
{
     name: "beer",
     cost: 8.50,
```

```
    alcohol: true
}
```

buyDrink should add the cost of the drink to the person's tab (total drinks bill) if the person is

1. old enough to drink (over 18 if the drink is alcohol)
2. buying the drink will not push their tab over $1000

If these conditions don't hold the function should do nothing.
You also need to write write a function **getReceipt** which returns a summary of all drinks a person bought, grouped by name, as a object. For example:

```
[
    {
        name: "beer",
        count: 3,
        total: 25.50
    },
    {
        name: "cola",
        count: 1,
        total: 2.50
    }
]
```

You can test your code as such, note that the script simply takes in a persons name, their age and a file of the drinks they want to order.

```
$ node objects.js Andrew 21 drinks.json
[ { name: 'cola', count: 2, total: 7 },
  { name: 'beer', count: 1, total: 5.5 },
  { name: 'fanta', count: 1, total: 3.5 } ]
$ node objects.js Lisa 21 drinks.json
[ { name: 'cola', count: 2, total: 7 },
  { name: 'fanta', count: 1, total: 3.5 } ]
```

When you think your program is working you can use `autotest` to run some simple automated tests:

```
$ 2041 autotest js_objects
```

Autotest Results

91% of **126** students who have autotested **objects.js** so far, passed all autotest tests.

- **93%** passed test *0*
- **91%** passed test *1*

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

```
$ give cs2041 lab07_js_objects objects.js
```

Sample solution for `objects.js`

```
/*
 * Fill out the Person prototype
 * function "buyDrink" which given a drink object which looks like:
 * {
 *     name: "beer",
 *     cost: 8.50,
 *     alcohol: true
 * }
 * will add the cost to the person expences if the person
 * is
 *    1. old enough to drink (if the drink is alcohol)
 *    2. buying the drink will not push their tab over $1000
 *
 * in addition write a function "getReceipt" which returns a list as such
 * [
 *    {
 *        name: beer,
 *        count: 3,
 *        total: 25.50
 *    }
 * ]
 *
 * which summaries all drinks a person bought by name in order
 * of when they were bought (duplicate buys are stacked)
 */

function Person(name, age) {
    this.name = name;
    this.age = age;
    this.tab = 0;
    this.history = {};
    this.historyLen = 0;
    this.canDrink = function() {
      return this.age >= 18;
    };
    this.canSpend = function(cost) {
      return this.tab + cost <= 1000;
    }
}

Person.prototype.buyDrink = function(drink) {
  if (!this.canSpend(drink.cost)) return;
  if (drink.alcohol && !this.canDrink()) return;

  if(this.history[drink.name]){
    this.history[drink.name].count++;
    this.history[drink.name].total+=drink.cost;
  } else {
    this.history[drink.name] = {
      name: drink.name,
      count: 1,
      total: drink.cost,
      order: this.historyLen++
    }
  }
  this.tab += drink.cost;
}

Person.prototype.getReceipt = function() {
  let all = Object.keys(this.history);
  all = all.sort((a,b)=>this.history[a].order - this.history[b].order);
  let r = [];
  for(let drink of all) {
    let d = this.history[drink];
    delete d.order;
    r.push(d);
  }
  return r;
}
```

```
const name = process.argv[2];
const age = parseInt(process.argv[3]);
const drinksJson = process.argv[4]
if (name === undefined || age === undefined || drinksJson === undefined) {
  throw new Error(`input not supplied`);
}
const drinks = require(`./${drinksJson}`);
const p = new Person(name, age);
for (let drink of drinks) {
  p.buyDrink(drink)
}
console.log(p.getReceipt());
```

Challenge Exercise: Javascript Piping

Write a function called buildPipe that returns a function which runs all of it's input functions in a pipeline

Consider the following code

```
let timesTwo = (a) => a*2;
let timesThree = (a) => a*3;
let minusTwo = (a) => a - 2;
let pipeline = buildPipe(timesTwo, timesThree, minusTwo);
```

In this case pipeline(x) is the same as minusTwo(timesThree(timesTwo(x)))

```
pipeline(6) == 34
```

To begin download and unzip the starter code

```
$ wget -q https://cgi.cse.unsw.edu.au/~cs2041/19T2//activities/js_piping/piping.zip
$ unzip piping.zip
```

You test your code with the basic supplied functions by running

```
$ cd piping
$ node test.js
34
32
```

When you think your program is working you can use `autotest` to run some simple automated tests:

```
$ 2041 autotest js_piping
```

Autotest Results

95% of **81** students who have autotested **piping.js** so far, passed the autotest test.

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

```
$ give cs2041 lab07_js_piping piping.js
```

Sample solution for `piping.js`

```
/*
 * Write a function called buildPipe that returns a function
 * which runs all of it's input functions in a pipeline
 * let timesTwo = (a) => a*2;
 * let timesThree = (a) => a*3;
 * let minusTwo = (a) => a - 2;
 * let pipeline = buildPipe(timesTwo, timesThree, minusTwo);
 *
 * pipeline(6) == 34
 *
 * pipeline(x) in this case is the same as minusTwo(timesThree(timesTwo(x)))
 *
 * test with `node test.js`
 */

function buildPipe(...ops) {
  const _pipe = (a, b) => (arg) => b(a(arg));
  return ops.reduce(_pipe);
}

/* Alternative recursive solution:

function buildPipe(f, ...fs) {
    if (fs.length > 0) return x => buildPipe(...fs)(f(x));
    return x => f(x);
}
*/
module.exports = buildPipe;
```

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 23 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.