INFO3139 Lab 3

Rev 1.0

General JavaScript Topic #2 – The spread operator (…)

Hey, wait a minute, wasn't the rest operator using three dots? Yes. Confusing as it is, the three dots is also known as the **spread operator**. The spread operator differs in that it is used for **array construction** and **de-structuring**, as it "spreads out" elements.

Create a new folder called week2**\class2** and create a new file in it called **spread_example1.js** with the following code:

```
// spread operator and arrays

const getOriginalCountries = () => {
  let originalCountries = ['Canada<--original', 'USA<--original'];
  let endCountries = ['England<--End', 'Japan<--End'];
  originalCountries.push(...endCountries); // appends on the end using spread
  return originalCountries;
};

const getAllCountries = () => {
  let original = getOriginalCountries();
  let beginCountries = ['Germany<--begin', 'Mexico<--begin'];
  original.unshift(...beginCountries); // appends to beginning using spread
  return original;
};

getAllCountries().forEach(country => console.log(country));
```

Then execute it:

```
PS E:\winter2023\info3139\programming\nodeexercises\week2\class2> node spread_example1
Germany<--begin
Mexico<--begin
Canada<--original
USA<--original
England<--End
Japan<--End
```

Create another file in the class 2 folder called **spread_example2.js** example that uses the … both as a spread and rest operators in the following code:

```
const foo = (...args) => // example of ... as rest operator
  console.log(`the 4th element of the arr spread out is ${args[3]}`); // example of ... as rest
operator

const arr = [1, 2, 3, 4, 5];
foo(...arr); // example of ... as spread operator, like calling foo(1,2,3,4,5)
// foo(arr) // undefined error because were not passing an args[3] just 1 array arg
```

```
PS E:\winter2023\info3139\programming\nodeexercises\week2\class2> node spread_example2
the 4th element of the arr spread out is 4
```

In summary, when using spread, you are spreading/expanding a single variable into more elements. When using rest arguments, you are gathering all remaining arguments of a function into one array.

Promise.all()

Part of the homework reading on promises discussed the **Promise.all()** function. Use this to **process multiple asynchronous** calls based on array data. To work out the mechanics of using this call do the following:

1. Copy the **non_blocking_routines.js** from the class1 folder over to the new class2 folder.

2. Add another routine to the new **non_blocking_routines.js** file with this content (do not forget to update the exports too):

```
const reverseNameWithAPromise = (name) => {
  return new Promise((resolve, reject) => {
    if (name === "err") {
      // Reject the Promise with an error
      reject({ reverseresults: "some severe error" });
    } else {
      // Resolve (or fulfill) the Promise with data
      resolve({
        reverseresults: `The name ${name}, reversed is ${name
          .split("")
          .reverse()
          .join("")}.`,
      });
    }
  });
};
```

3. Create a new file called nodeexercises\week2\class2**promise_all.js** with this content:

```
import * as rtnLib from "./non_blocking_routines.js";

const promiseAllRtn = (nameArray) => {
  // Promise.all quits on first catch
  Promise.all(
    // note the .map here will create a new array
    nameArray.map((name) => {
      return rtnLib.reverseNameWithAPromise(name);
    })
  )
    .then((resultsArray) => {
      console.log(`\nresults from promises:\n`);
      resultsArray.map((result) => console.log(result.reverseresults));
    })
    .catch((err) => console.log(err.reverseresults));
};

promiseAllRtn(["Bill", "Jane", "Bob"]);
// promiseAllRtn(["Bill", "Jane", "err", "Bob"]);
```

Notice that we're passing an array to the routine and then using the **.map operator (which makes a new array)** to make asynchronous calls against each item in the array.

4. Execute the new code:

```
PS E:\winter2023\info3139\programming\nodeexercises\week2\class2> node promise_all

results from promises:

The name Bill, reversed is lliB.
The name Jane, reversed is enaJ.
The name Bob, reversed is boB.
```

Promise.all is ok, but if any of the promises are rejected, we return control to the .catch clause.and the program finishes. For instance:

5. comment out the 2nd last line and uncomment out the last line which adds the name **err** to the array

```
promiseAllRtn(["Bill", "Jane","err", "Bob"]);
```

6. Execute again and all you will see is:

```
PS E:\winter2023\info3139\programming\nodeexercises\week2\class2> node promise_all
some severe error
```

To get around this limitation you can use a promise function called **Promise.allSettled.**

7. Add another file to the folder called **promise_allsettled.js** with the following:

```
import * as rtnLib from "./non_blocking_routines.js";

const promiseAllSettledRtn = (nameArray) => {
  // Promise.allSettled won't fire the catch if a promise fails
  //  results contain status and value if resolved or reason
  //  if rejected
  Promise.allSettled(
    nameArray.map((name) => {
      return rtnLib.reverseNameWithAPromise(name);
    })
  )
    .then((statusArray) => {
      console.log(`\nstatus from promises\n`);
      statusArray.map((result) => console.log(result.status));
      console.log(`\nresults from promises\n`);
      statusArray.map((result) => {
        result.value // resolve
          ? console.log(result.value.reverseresults)
          : console.log(result.reason.reverseresults);
      });
    })
    .catch((err) => console.log(err));
};

promiseAllSettledRtn(["Bill", "Jane", "err", "Bob"]);
```

8. Execute the new code which should result in:

```
PS E:\winter2023\info3139\programming\nodeexercises\week2\class2> node promise_allsettled

status from promises

fulfilled
fulfilled
rejected
fulfilled

results from promises

The name Bill, reversed is lliB.
The name Jane, reversed is enaJ.
some severe error
The name Bob, reversed is boB.
```

Promse.allSettled also **returns more information** than the promise.all does, it returns a **status** property for each call as well as **a value** if it was resolved or **a reason** if it does not. Study the code from above and see how these three properties are being used.

A Newer Syntax To Deal with Promises (Async/Await)

Another article from the homework reading was on the **async/await** functionality that has been available in node since ver. 7.6. After reading the article you should realize that any promise we have, we can **await**. That's literally all await means: it functions in exactly the same way as calling `.then()` on a promise (but without requiring any callback function).

9. To exercise the async/await technique add a new file in the class2 called **chain_the_async_way.js** with the following code:

```
import * as rtnLib from "./non_blocking_routines.js";

const someAsyncFunction = async (theVar) => {
  try {
    let results = await rtnLib.someRtnWithAPromise(theVar);
    console.log(`The 1st call ${results.val1} ${results.val2}`);
    results = await rtnLib.anotherRtnWithAPromise(theVar);
    console.log(`The 2nd call ${results.val1} ${results.val2} successful`);
    theVar = "err";
    results = await rtnLib.someRtnWithAPromise(theVar); // will fire catch
  } catch (err) {
    console.log(err);
  }
};

let someVar = "no error";
someAsyncFunction(someVar);
```

10. Then test the new code out:

```
PS E:\winter2023\info3139\programming\nodeexercises\week2\class2> node chain_the_async_way
The 1st call was successful
The 2nd call was more successful
some error
```

Promise.all with async/await

11. Create a new file called **promise_all_async.js** with the following code to see how we can incorporate the newer syntax to work with Promise.all:

```
import * as rtnLib from "./non_blocking_routines.js";

// using Promise.all with async/await
const promiseAllAsyncRtn = async (nameArray) => {
  try {
    // Promise.all runs all promises in parallel
    // if any reject the catch is fired
    let resultsArray = await Promise.all(
      nameArray.map((item) => {
        return rtnLib.reverseNameWithAPromise(item);
      })
    );
    console.log(`\nresults from promises using async/await:\n`);
    resultsArray.forEach((result) => console.log(result.reverseresults));
  } catch (err) {
    console.log(err.reverseresults);
  }
};
promiseAllAsyncRtn(["Bill", "Jane", "Bob"]);
// promiseAllAsyncRtn(["Bill", "Jane", "err", "Bob"]);
```

12. Execution results in the same output as before:

```
PS E:\winter2023\info3139\programming\nodeexercises\week2\class2> node promise_all_async

results from promises using async/await:

The name Bill, reversed is lliB.
The name Jane, reversed is enaJ.
The name Bob, reversed is boB.


PS E:\winter2023\info3139\programming\nodeexercises\week2\class2> node promise_all_async
some severe error
```

Promise.allSettled with async/await

13. Now let's redo the promiseallsettled example to use async/await. Create another file called **promise_allsettled_async.js** with the following:

```
import * as rtnLib from "./non_blocking_routines.js";

const promiseAllSettledAsyncRtn = async (nameArray) => {
  // Promise.allSettled won't fire the catch if a promise fails
  //  object returned contains status and value if resolved or reason
  //  if rejected
  try {
    let statusArray = await Promise.allSettled(
      nameArray.map((name) => {
        return rtnLib.reverseNameWithAPromise(name);
      })
    );
    console.log(`\nstatus from promises\n`);
    statusArray.forEach((result) => console.log(result.status));
    console.log(`\nresults from promise.allSettled with async/await\n`);

    statusArray.forEach((result) => {
      result.value // resolve
        ? console.log(result.value.reverseresults)
        : console.log(result.reason.reverseresults);
    });
  } catch (err) {
    // reject
    console.log(err.reverseresults);
  }
};

promiseAllSettledAsyncRtn(["Bill", "Jane", "err", "Bob"]);
```

14. Then execute it and you should again see:

```
PS E:\winter2023\info3139\programming\nodeexercises\week2\class2> node promise_allsettled_async

status from promises

fulfilled
fulfilled
rejected
fulfilled

results from promise.allSettled with async/await

The name Bill, reversed is lliB.
The name Jane, reversed is enaJ.
some severe error
The name Bob, reversed is boB.
```

Lab 3 – 2.5%

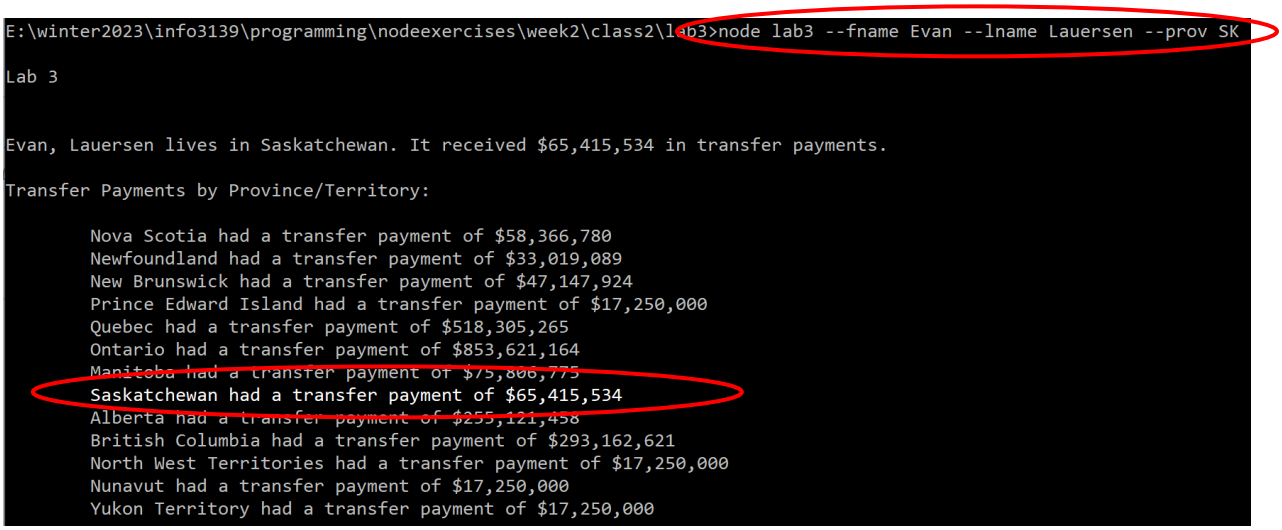
Part A - (2%)

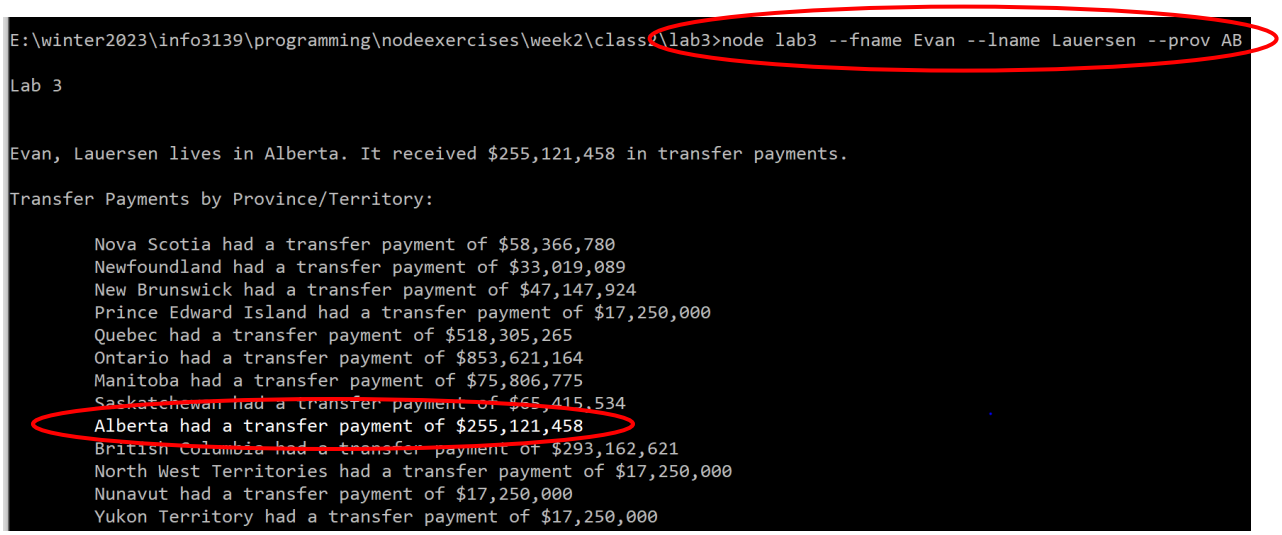
- Copy the week2\class1\lab2\lab2_routines.js to **week2\class2\lab3\lab3_routines**
- Copy the week2\class1\lab2\lab2.js to **week2\class2\lab3\lab3.js**
- Edit the new lab3.js and **convert all** existing .then/.catch syntax to try/catch async/await syntax
- In addition to Lab 2's output, also have lab3.js **dump out all provinces transfer payments** by making a series of calls to the lab3_routines.transferPaymentForProvincePromise **for each province** in lab3_routines.provinces. Use a **Promise.allSettled** for this processing. Note the line that is **bolded** is when the province is the same as the province of residence argument.
- To highlight/un-highlight a single line of console output. you can use this code (the bolding doesn't show that well in the white background terminal, so the screen shots below are from a regular command prompt):

```
console.log(`\x1b[1mI'm a bold line`);
console.log(`\x1b[0mI'm a regular line`);
```

Submit in a single word document:

1. a screen shot for your name and Saskatchewan for the province of residence.
2. a screen shot for your name and Alberta for the province of residence.
3. source code for lab3.js





Part B - (.5%) - Lab 3 Theory Quiz on FOL - 10 questions

Review Questions

1. What term is used when returning an additional asynchronous function from a then?
2. What newer syntax is available to call promises?
3. T/F the syntax from the previous question is just syntactic sugar for promises?
4. What keyword is used to pause when calling a promise until it is resolved or rejected?
5. When do you use a Promise.all?
6. How does Promise.allSettled differ from Promise.all?
7. What properties can you see in the results array from a Promise.allSettled than you won't find in the results from a Promise.all

Homework:

Read over the following:

https://www.twilio.com/blog/working-with-environment-variables-in-node-js-html