JavaScript (ES5 and earlier)

LAB GUIDE

DESTRUCTURING

1. Use array and object destructuring feature to create variable that hold values as shown below.

```
const iPhone11 = {
   name: 'iPhone 11',
   manufacturer: 'Apple',
   price: 699,
   specs: {
      color: 'White',
      memory: {
         value: 128,
         unit: 'MB'
      },
      cameras: {
         front: '12 MP Wide',
         rear: '12 MP Ultra Wide'
      },
      availableColors: [ 'Black', 'Green', 'Yellow', 'Purple', 'Red', 'White' ]
   },
   calculateDiscoutedPrice: function( percentage ) {
      return this.price * ( 100 - percentage ) / 100;
   }
}

// create the variables using destructuring - the variables should have values as shown below.
// destructure here...

// below line logs - iPhone 11   Apple   128   12 MP Ultra Wide   Green
console.log( name, brand, ram, rearCamera, secondColor );
```

2. Write a standalone function ```printPhoneDetails``` that prints a description of the phone with discounted price as in an earlier exercise. The function is passed the iPhone11 object but it destructures to create only the required variables when accepting the object as argument.

```
// iPhone11.calculateDiscoutedPrice = iPhone11.calculateDiscoutedPrice.bind( iPhone11 )
printPhoneDetails.call( iPhone11 ); // Logs "Apple iPhone11 is available at a 10% discounted rate of $629.1"
```

NOTE: You must use destructuring when accepting the argument.

REST OPERATOR

3. Write a function *_sum_* that calculates and returns the sum of arguments (assume all arguments are numbers) passed to it, and call it like so. Use a rest parameter to implement this.

```
var result = sum( 1, 2, 3, 4 );
console.log( result ); // prints 10

var result = sum( 1, 2, 3, 4, 5  );
console.log( result ); // prints 15
```

4. Write function ```max``` that can accept any number of arguments (assume all numbers) and returns the maximum of the numbers. Make use of the rest operator to group the arguments into an array before finding and returning the maximum.

```
max( 25, 65, 35, 45 ); // 65
max( 25, 65, 35, 75, 45 ); // 75
```

SPREAD OPERATOR

5. Use the spread operator, along with the max function, to find the maximum of values in this array.

```
const numbers = [ 25, 65, 35, 75, 45 ];
```

6. Use object spread to make a shallow copy of the following object.
* Again, use array and object spread (as required) to create a deep copy of the object.
* Test out if making a change to name and front camera details on the iPhone11 object affects the shallow copy.
* What about the deep copy? Is it affected?

```
const iPhone11 = {
   name: 'iPhone 11',
   manufacturer: 'Apple',
   price: 699,
   specs: {
      color: 'White',
      memory: {
         value: 128,
         unit: 'MB'
      },
      cameras: {
         front: '12 MP Wide',
         rear: '12 MP Ultra Wide'
      },
      availableColors: [ 'Black', 'Green', 'Yellow', 'Purple', 'Red', 'White' ]
   }
}
```

ARROW FUNCTIONS

7. When will you use an arrow function? When will you NOT use an arrow function?

8. The following snippet tries to set new salaries for employees. However it does not work as intended. Why? Debug the code by setting breakpoints and stepping through function calls. Correct it using an appropriate arrow function definition.

```
const payroll = {
   employees: [
      { name: 'John', dept: 'IT', salary: 1000 },
      { name: 'Maria', dept: 'Finance', salary: 2000 },
      { name: 'David', dept: 'Finance', salary: 3000 }
   ],
   hikePercentage: {
      IT : 10,
      Finance: 20
   },
   raise: function() {
      this.employees.forEach(function ( employee ) {
         const dept = employee.dept;

         const salary = employee.salary;
         const hikePercentage = this.hikePercentage[dept];

         employee.salary = ( ( 100 + hikePercentage ) / 100 ) * salary;
      });
   }
}

payroll.raise();
console.log( payroll.employees );
```

CLASSES AND INHERITANCE

9. Create a Movie class that represents details of a movie. Suggested information to have in an object of the class - name, cast (an array of strings with cast member's names), yearOfRelease, boxOfficeCollection, addToCast(newMember) that accepts a new cast member's name and adds to the cast array, addToCollection(amount) that accepts box office collections for a week and adds it to the current boxOfficeCollection. Create 2 objects of this class that represent any 2 movies. Call the methods addToCast() and addToCollection() and verify they work according to expectations.

10. Create a SequelMovie class that inherits from Movie class. SequelMovie has an additional property called earlierMovies - an array of Movie objects. It has an additional method called getLifetimeEarnings() that returns the sum of boxOfficeCollection of all earlier movies along with the SequelMovie object's boxOfficeCollection.

11. Define a Project class with id (number), name (string), client (string). Define some Project objects (suggest using sample data below).

```
const dbsPayroll = new Project( 1001, 'DBS payroll', 'DBS' );
const intranetDeployment = new Project( 2001, 'Intranet v2 deployment', 'Internal' );
```

12. Define an Employee class with id (number), name (string), role (string), department (string), projects (array of Projects). Define some Employee objects (suggest using sample data below).

```
const john = new Employee( 1, 'John', 'Web Developer', 'IT', [ dbsPayroll ] );
const jane = new Employee( 2, 'Jane', 'Project Manager', 'IT', [ dbsPayroll, intranetDeployment ] );
const mark = new Employee( 3, 'Mark', 'System Administrator', 'Operations', [ intranetDeployment ] );
```

MODULES

13. Define Project and Employee classes in separate modules (as default exports in those modules). Create another module that creates an array of projects, and an array of employee objects, and exports these (as named exports). Create an HTML page that imports projects and employees and shows the list of projects and employees in 2 separate tables.

FETCH API, PROMISES AND ASYNC..AWAIT

14. Given the following APIs

* Retrieve users with a particular username (in the example below, username is Bret)

```
https://jsonplaceholder.typicode.com/users?username=Bret
```

* Retrieve posts by a user with given id (in the example below, user id is 1)

```
https://jsonplaceholder.typicode.com/users/1/posts'
```

* Retrieve comments for post with given id (in the example below, post id is 1)

```
https://jsonplaceholder.typicode.com/comments?postId=1
```

Write a function that accepts the username and returns a promise that resolves with the email ids of people who have commented on the first post of the first user matching the given username.

```
function getCommentersEmailIds( username ) {
}
```

For example, if username = Bret, then first user matching username Bret has id = 1 (user whose name is "Leanne Graham" and username is "Bret"). The first post by this user has id = 1. The list of people who commented on this post are ["Eliseo@gardner.biz", "Jayne_Kuhic@sydney.com", "Nikita@garfield.biz", "Lew@alysha.tv", "Hayden@althea.biz"]

Test your function out, for example by passing username as 'Bret'

15. Rewrite the exercise in the previous function using async-await.