ES6 EXERCISE

1) Difference between var and let keywords

OLD CODE	NEW CODE
`var catName;`	`let catName;`
`var quote;`	`let quote;`
`function catTalk() {`	`function catTalk() {`
`  "use strict";`	`  "use strict";`
`  catName = "Oliver";`	`  catName = "Oliver";`
`  quote = catName + " says Meow!";`	`  quote = catName + " says Meow!";`
`}`	`}`
`catTalk();`	`catTalk();`

2) Scopes of var and let keywords

OLD CODE	NEW CODE
`function checkScope() {`	`function checkScope() {`
`  'use strict';`	`  'use strict';`
`  var i = 'function scope';`	`  let i = 'function scope';`
`  if (true) {`	`  if (true) {`
`    i = 'block scope';`	`    let i = 'block scope';`
`    console.log('Block scope i is: ', i);`	`    console.log('Block scope i is: ', i);`
`  }`	`  }`
`  console.log('Function scope i is: ', i);`	`  console.log('Function scope i is: ', i);`
`  return i;`	`  return i;`
`}`	`}`

3) Declaring read-only variable using const keyword

OLD CODE	NEW CODE
`function printManyTimes(str) {`	`function printManyTimes(str) {`
`  "use strict";`	`  "use strict";`
`  // Only change code below this line`	`  // Only change code below this line`
`  var sentence = str + " is cool!";`	`  const SENTENCE = str + " is cool!";`
`  for (var i = 0; i < str.lengt`	`  for (let i = 0; i < str.length`

h; i+=2) {
    console.log(sentence);
  }
  // Only change code above this line
}
printManyTimes("freeCodeCamp");
	; i+=2) {
	    console.log(SENTENCE);
	  }
	  // Only change code above this line
	}
	printManyTimes("freeCodeCamp");

4) Mutate an array declared with const

OLD CODE	NEW CODE

const s = [5, 7, 2];
function editInPlace() {
  'use strict';
  // Only change code below this line
  // Using s = [2, 5, 7] would be invalid
  // Only change code above this line
}
editInPlace();
	const s = [5, 7, 2];
	function editInPlace() {
	  'use strict';
	  // Only change code below this line
	    s[0]=2;
	    s[1]=5;
	    s[2]=7;
	  // Only change code above this line
	}
	editInPlace();

5) Prevent object mutation

OLD CODE	NEW CODE

function freezeObj() {
  'use strict';
  const MATH_CONSTANTS = {
    PI: 3.14
  };
  // Only change code below this line
  // Only change code above this line
  try {
    MATH_CONSTANTS.PI = 99;
  } catch(ex) {
    console.log(ex);
  }
  return MATH_CONSTANTS.PI;
	function freezeObj() {
	  'use strict';
	  const MATH_CONSTANTS = {
	    PI: 3.14
	  };
	  // Only change code below this line
	    Object.freeze(MATH_CONSTANTS);
	  // Only change code above this line
	  try {
	    MATH_CONSTANTS.PI = 99;
	  } catch(ex) {
	    console.log(ex);

}
const PI = freezeObj();
	}
	return MATH_CONSTANTS.PI;
	}
	const PI = freezeObj();

6) Using arrow functions to write concise anonymous functions

OLD CODE	NEW CODE

var magic = function() {
"use strict";
return new Date();
};
	const magic = () => new Date();

7) Arrow functions with parameters

OLD CODE	NEW CODE

var myConcat = function(arr1, arr2) {
"use strict";
return arr1.concat(arr2);
};
console.log(myConcat([1, 2], [3, 4, 5]));
	const myConcat = (arr1, arr2) => arr1.concat(arr2);
	console.log(myConcat([1, 2], [3, 4, 5]));

8) Setting default parameters for functions

OLD CODE	NEW CODE
const increment = (number, value) => number + value;	const increment = (number, value = 1) => number + value;

9) Using rest parameter with function parameters

OLD CODE	NEW CODE

const sum = (x, y, z) => {
const args = [x, y, z];
return args.reduce((a, b) => a + b, 0);
}
	const sum = (...args) => {
	return args.reduce((a, b) => a + b, 0);
	}

10) Using spread operator to evaluate arrays in-place

OLD CODE	NEW CODE

const arr1 = ['JAN', 'FEB', 'MAR', 'APR', 'MAY'];
let arr2;
arr2 = []; // Change this line
 	
console.log(arr2);
	const arr1 = ['JAN', 'FEB', 'MAR', 'APR', 'MAY'];
	let arr2;
	arr2 = [...arr1]; // Change this line
	console.log(arr2);

11) Setting destructuring assignment to extract values from objects

OLD CODE	NEW CODE

const increment = (number, value) => number + value;
const HIGH_TEMPERATURES = {
  yesterday: 75,
  today: 77,
  tomorrow: 80
};
 	
// Only change code below this line
const today = HIGH_TEMPERATURES.today;
const tomorrow = HIGH_TEMPERATURES.tomorrow;
// Only change code above this line
	const HIGH_TEMPERATURES = {
	  yesterday: 75,
	  today: 77,
	  tomorrow: 80
	};
	
	// Only change code below this line
	const {today,tomorrow} = HIGH_TEMPERATURES;
	// Only change code above this line

12) Using destructuring assignment to assign variables from objects

OLD CODE	NEW CODE

const HIGH_TEMPERATURES = {
  yesterday: 75,
  today: 77,
  tomorrow: 80
};
	const HIGH_TEMPERATURES = {
	  yesterday: 75,
	  today: 77,
	  tomorrow: 80
	};

// Only change code below this line
const highToday = HIGH_TEMPERATURES.today;
const highTomorrow = HIGH_TEMPERATURES.tomorrow;
// Only change code above this line
	// Only change code below this line
	const {today: highToday, tomorrow: highTomorrow} = HIGH_TEMPERATURES;
	// Only change code above this line

13) Using destructuring assignment to assign variables from nested objects

OLD CODE	NEW CODE

const LOCAL_FORECAST = {
  yesterday: { low: 61, high: 75 },
  today: { low: 64, high: 77 },
  tomorrow: { low: 68, high: 80 }
 }
};
 	
// Only change code below this line
const lowToday = LOCAL_FORECAST.today.low;
const highToday = LOCAL_FORECAST.today.high;
// Only change code above this line
	const LOCAL_FORECAST = {
	  yesterday: { low: 61, high: 75 },
	  today: { low: 64, high: 77 },
	  tomorrow: { low: 68, high: 80 }
	}
	};
	
	// Only change code below this line
	const {today: {low: lowToday, high: highToday}} = LOCAL_FORECAST;
	// Only change code above this line

14) Using destructuring assignment to assign variables from arrays

OLD CODE	NEW CODE

let a = 8, b = 6;
// Only change code below this line
	let a = 8, b = 6;
	// Only change code below this line
	[a, b] = [b, a];

15) Using destructuring assignment with the rest parameter to reassign array elements

OLD CODE	NEW CODE

const source = [1,2,3,4,5,6,7,8,9,10];
function removeFirstTwo(list) {
  "use strict";
const arr = list; // Change this line
  return arr;
}
const arr = removeFirstTwo(source);
	const source = [1,2,3,4,5,6,7,8,9,10];
	function removeFirstTwo(list) {
	  "use strict";
	  let [a,b,...arr] = list; // Change this line
	  return arr;
	}
	const arr = removeFirstTwo(source);

16) Using destructuring assignment to pass an object as function parameters

OLD CODE	NEW CODE

const stats = {
  max: 56.78,
  standard_deviation: 4.34,
  median: 34.54,
  mode: 23.87,
  min: -0.75,
  average: 35.85
};
 	
// Only change code below this line
const half = (stats) => (stats.max + stats.min) / 2.0;
	const stats = {
	  max: 56.78,
	  standard_deviation: 4.34,
	  median: 34.54,
	  mode: 23.87,
	  min: -0.75,
	  average: 35.85
	};
	
	// Only change code below this line
	const half = ({max, min}) => (max + min) / 2.0;

17) Create strings using template literals

OLD CODE	NEW CODE

const result = {
  success: ["max-length", "no-amd", "prefer-arrow-functions"],
	const result = {
	  success: ["max-length", "no-amd", "prefer-arrow-functions"],
	  failure: ["no-var", "var-on-

failure: ["no-var", "var-on-top", "linebreak"],
  skipped: ["no-extra-semi", "no-dup-keys"]
};
function makeList(arr) {
  // Only change code below this line
  const failureItems = [];
  // Only change code above this line
 	
  return failureItems;
}
 	
const failuresList = makeList(result.failure);
	top", "linebreak"],
	  skipped: ["no-extra-semi", "no-dup-keys"]
	};
	function makeList(arr) {
	  "use strict";
	  // change code below this line
	  const failureItems = [];
	  for (let i = 0; i < arr.length; i++) {
	    failureItems.push(`<li class="text-warning">${arr[i]}</li>`);
	  }
	  // change code above this line
	  return failureItems;
	}
	
	const failuresList = makeList(result.failure);

18) Writing concise object literal declarations using object property shorthand

OLD CODE	NEW CODE

const createPerson = (name, age, gender) => {
  "use strict";
  // Only change code below this line
  return {
    name: name,
    age: age,
    gender: gender
  };
  // Only change code above this line
};
	const createPerson = (name, age, gender) => {
	  "use strict";
	
	  // Only change code below this line
	  return {name, age, gender};
	
	  // Only change code above this line
	};

19) Writing concise declarative functions with ES6

OLD CODE	NEW CODE

// Only change code below this line
const bicycle = {
  gear: 2,
  setGear: function(newGear) {
    this.gear = newGear;
  }
};
// Only change code above this line
bicycle.setGear(3);
console.log(bicycle.gear);
	// Only change code below this line
	const bicycle = {
	  gear: 2,
	  setGear(newGear) {
	    this.gear = newGear;
	  }
	};
	// Only change code above this line
	bicycle.setGear(3);
	console.log(bicycle.gear);

20) Using class syntax to define a constructor function

OLD CODE	NEW CODE

// Only change code below this line
 	
// Only change code above this line
 	
const carrot = new Vegetable('carrot');
console.log(carrot.name); // Should display 'carrot'
	// Only change code below this line
	class Vegetable {
	  constructor(name) {
	    this.name = name;
	  }
	}
	// Only change code above this line
	
	const carrot = new Vegetable('carrot');
	console.log(carrot.name); // Should display 'carrot'

21) Using getters and setters to control access to an object

OLD CODE	NEW CODE

// Only change code below this line
 	
// Only change code above this line
	// Only change code below this line
	class Thermostat {
	  constructor(farenheit) {
	    this._farenheit = farenheit;
	  }

const thermos = new Thermostat(76); // Setting in Fahrenheit scale
let temp = thermos.temperature;
 // 24.44 in Celsius
thermos.temperature = 26;
temp = thermos.temperature; // 26 in Celsius
	get temperature() {
	    return 5/9 * (this._farenheit - 32);
	  }
	  set temperature(celsius) {
	    this._farenheit = celsius * 9.0/5 + 32;
	  }
	}
	// Only change code above this line
	
	const thermos = new Thermostat(76); // Setting in Fahrenheit scale
	let temp = thermos.temperature;
	// 24.44 in Celsius
	thermos.temperature = 26;
	temp = thermos.temperature; // 26 in Celsius

22) Creating a module script

OLD CODE	NEW CODE

<html>
  <body>
    <!--
 Only change code below this line -->
 	
    <!--
 Only change code above this line -->
  </body>
</html>
	<html>
	  <body>
	    <!--
	 Only change code below this line -->
	<script type="module" src="index.js"></script>
	    <!--
	 Only change code above this line -->
	  </body>
	</html>

23) Using export to share a code block

OLD CODE	NEW CODE
const uppercaseString = (string	export const uppercaseString = (

) => {
  return string.toUpperCase();
}
 	
const lowercaseString = (string) => {
  return string.toLowerCase()
}
	string) => {
	  return string.toUpperCase();
	}
	
	export const lowercaseString = (string) => {
	  return string.toLowerCase()
	}

24) Reusing javascript code using import

OLD CODE	NEW CODE

// Only change code above this line
 	
uppercaseString("hello");
lowercaseString("WORLD!");
	import {uppercaseString, lowercaseString} from './string_functions.js';
	// Only change code above this line
	
	uppercaseString("hello");
	lowercaseString("WORLD!");

25) Using '*' to import everything from a file

OLD CODE	NEW CODE

// Only change code above this line
 	
stringFunctions.uppercaseString("hello");
stringFunctions.lowercaseString("WORLD!");
	import * as stringFunctions from "./string_functions.js";
	// Only change code above this line
	stringFunctions.uppercaseString("hello");
	stringFunctions.lowercaseString("WORLD!");

26) Creating an export fallback using export default

OLD CODE	NEW CODE

function subtract(x, y) {
  return x - y;
}
	export default function subtract(x, y) {
	  return x - y;
	}

27) Importing a default export

OLD CODE	NEW CODE

// Only change code above this line
subtract(7,4);
	import subtract from "./math_functions.js";
	// Only change code above this line
	subtract(7,4);

28) Creating a javascript promise

OLD CODE	NEW CODE
-	const makeServerRequest = new Promise((resolve,reject) => {});

29) Completing a promise with resolve and reject

OLD CODE	NEW CODE

const makeServerRequest = new Promise((resolve, reject) => {
  // responseFromServer represents a response from a server
  let responseFromServer;
 	
  if(responseFromServer) {
    // Change this line
  } else {
    // Change this line
  }
});
	const makeServerRequest = new Promise((resolve, reject) => {
	  // responseFromServer represents a response from a server
	  let responseFromServer;
	  if(responseFromServer) {
	    // Change this line
	    resolve("We got the data");
	  } else {
	    // Change this line
	    reject("Data not received");
	  } });

30) Handling a fulfilled promise with then keyword

OLD CODE	NEW CODE
const makeServerRequest = new Promise((resolve, reject) => { // responseFromServer is set to true to represent a successf	const makeServerRequest = new Promise((resolve, reject) => { // responseFromServer is set to true to represent a successful

```
ul response from a server
  let responseFromServer = true;

  if(responseFromServer) {
    resolve("We got the data");
  } else {
    reject("Data not received");
  }
});
```

```
 response from a server
  let responseFromServer = true;

  if(responseFromServer) {
    resolve("We got the data");
  } else {
    reject("Data not received");
  }
});
makeServerRequest.then(result => {
  console.log(result);
});
```

31) Handling a rejected promise with catch keyword

OLD CODE

```
const makeServerRequest = new Promise((resolve, reject) => {
  // responseFromServer is set to false to represent an unsuccessful response from a server
  let responseFromServer = false;

  if(responseFromServer) {
    resolve("We got the data");
  } else {
    reject("Data not received");
  }
});

makeServerRequest.then(result => {
  console.log(result);
});
```

NEW CODE

```
const makeServerRequest = new Promise((resolve, reject) => {
  // responseFromServer is set to false to represent an unsuccessful response from a server
  let responseFromServer = false;

  if(responseFromServer) {
    resolve("We got the data");
  } else {
    reject("Data not received");
  }
});

makeServerRequest.then(result => {
  console.log(result);
});
makeServerRequest.catch(error => {
  console.log(error);
});
```