Started on	Wednesday, 6 September 2023, 12:29 PM
State	Finished
Completed on	Thursday, 7 September 2023, 2:11 PM
Time taken	1 day 1 hour
Marks	6.00/6.00
Grade	**10.00** out of 10.00 (**100**%)

QUESTION 1

Correct

Mark 1.00 out of 1.00

Implement the *getPromise(delay, message)* function, which takes an integer number *delay* (between 0 and 2000) and string *message* and returns a Promise that waits for specified amount of time (using *delay* argument) and resolves with the message.

* For correct passing of all tests don't use console.log() method in your code.

Answer: (penalty regime: 0 %)

Reset answer

```
function getPromise(delay, message) {
  return new Promise((resolve, reject) => {
    setTimeout(function () {
      resolve(message);
    }, delay);
  });
}
```

	Test	Expected	Got	

✓	`getPromise(2000, "hello").then(function(data) {`
	`    console.log(data);`
	`});`
	`const end = Date.now() + 3000;`
	`  while (Date.now() < end) {`
	`    const muchCompute = 1 + 2 + 3;`
	`  }`
		hello	hello	✓

✓	`getPromise(2000, "world").then(function(data) {`
	`    console.log(data);`
	`});`
	`const end = Date.now() + 3000;`
	`  while (Date.now() < end) {`
	`    const muchCompute = 1 + 2 + 3;`
	`  }`
		world	world	✓

Passed all tests! ✓

Correct

Marks for this submission: 1.00/1.00.

QUESTION **2**

Correct

Mark 1.00 out of 1.00

Write an *add(x, y)* function that takes two arguments *x* and *y*. The function should return a Promise that resolves with the sum of the two arguments if they are *Numbers*, or rejects with the message "Error!" otherwise.

* For correct passing of all tests don't use console.log() method in your code.

Answer: (penalty regime: 0 %)

Reset answer

```
function add(x, y) {
  return new Promise((resolve, reject) => {
    if (typeof x === 'number' && typeof y === 'number') {
      resolve(x + y);
    }
    reject('Error!');
  });
}
```

	Test	Expected	Got	

✓	add(2,2).then(res => console.log(res));
	const end = Date.now() + 1000;
	while (Date.now() < end) {
	    const muchCompute = 1 + 2 + 3;
	}
		4	4	✓

✓	add(2,"a").catch(err => console.log(err));
	const end = Date.now() + 1000;
	while (Date.now() < end) {
	    const muchCompute = 1 + 2 + 3;
	}
		Error!	Error!	✓

✓	add("b","a").catch(err => console.log(err));
	const end = Date.now() + 1000;
	while (Date.now() < end) {
	    const muchCompute = 1 + 2 + 3;
	}
		Error!	Error!	✓

Passed all tests! ✓

Correct

Marks for this submission: 1.00/1.00.

QUESTION 3

Correct

Mark 1.00 out of 1.00

Implement the *getAge()* function to get user age. To find his age you need to call a getUser() async function that returns a user object in format {role: "somerole", id: 1}.

To get the actual user info you need to call another async function getUserProfile(id), which uses id returned from the previous function and returns user info as an object

{name: "Petro", age: 15}. The *getAge() must return the age of the user.*

* For correct passing of all tests don't use console.log() method in your code.

Answer: (penalty regime: 0 %)

Reset answer

```
const { getUser, getUserProfile } = require('./Helper.js');

async function getAge() {
  const user = await getUser();
  const userProfile = await getUserProfile(user.id);
  return userProfile.age;
}
```

	Test	Expected	Got	

✓	`getAge().then(a => console.log(a));`
	`const end = Date.now() + 1000;`
	`while (Date.now() < end) {`
	`    const muchCompute = 1 + 2 + 3;`
	`}`
		20	20	✓

Passed all tests! ✓

Correct

Marks for this submission: 1.00/1.00.

QUESTION **4**

Correct

Mark 1.00 out of 1.00

Implement the *take()* function that converts a sequence of iterated values into a sequence of length n.

* For correct passing of all tests don't use console.log() method in your code.

For example:

Test	Result

`const arr = ['a', 'b', 'c', 'd'];`
`for (const x of take(2, arr)) {`
`    console.log(x);`
`}`
	a
	b

Answer: (penalty regime: 0 %)

Reset answer

```
function* take(n, iterable) {
  const newArr = iterable.splice(null, n);
  for (const i of newArr) {
    yield i;
  }
}

// function* take(n, iterable) {
//   const newArr = iterable.slice(0, n);
//   for (const i of newArr) {
//     yield i;
//   }
// }
```

	Test	Expected	Got	

✓	`const arr = ['a', 'b', 'c', 'd'];`
	`for (const x of take(2, arr)) {`
	`    console.log(x);`
	`}`
		a
		b
			a
			b
				✓

✓	`const arr = ['a', 'b', 'c', 'd'];`
	`for (const x of take(3, arr)) {`
	`    console.log(x);`
	`}`
		a
		b
		c
			a
			b
			c
				✓

Passed all tests! ✓

Correct

Marks for this submission: 1.00/1.00.

QUESTION 5

Correct

Mark 1.00 out of 1.00

Please, implement a function accountPatients that takes a count of beds in a hospital and returns an array of two functions:

the first one for adding a patient

the second one for discharging a patient

Initially there are no patients in the hospital.

accountPatients should keep track of free beds in a hospital and every time when a patient is admitted or discharged, print the count to the console like in examples:

A patient was admitted, 34 beds are available

A patient was discharged, 54 beds are available

When there are no beds available,

Can not admit a patient, no beds available should be printed

When there is an attempt to discharge a patient when there are no patients,

There are no patients to discharge should be printed

Answer: (penalty regime: 0 %)

Reset answer

```
const accountPatients = (beds) => {
  let patients = 0;

  function admit() {
    if (beds > 0) {
      patients++;
      beds--;
      console.log(`A patient was admitted, ${beds} beds are available`);
    } else {
      console.log('Can not admit a patient, no beds available');
    }
  }

  function discharge() {
    if (patients > 0) {
      patients--;
      beds++;
      console.log(`A patient was discharged, ${beds} beds are available`);
    } else {
      console.log('There are no patients to discharge');
    }
  }

  return [admit, discharge];
};
```

	Test	Expected	Got	

✓	`const [admit, discharge] = accountPatients(100);`
	`admit();`
	`admit();`
	`discharge();`
		`A patient was admitted, 99 beds are available`
		`A patient was admitted, 98 beds are available`
		`A patient was discharged, 99 beds are available`
			`A patient was admitted, 99 beds are available`
			`A patient was admitted, 98 beds are available`
			`A patient was discharged, 99 beds are available`
				✓

	Test	Expected	Got	

✓	const [admit, discharge] = accountPatients(3);
	admit();
	admit();
	admit();
	admit();
	discharge();
	discharge();
	discharge();
	discharge();
		A patient was admitted, 2 beds are available
		A patient was admitted, 1 beds are available
		A patient was admitted, 0 beds are available
		Can not admit a patient, no beds available
		A patient was discharged, 1 beds are available
		A patient was discharged, 2 beds are available
		A patient was discharged, 3 beds are available
		There are no patients to discharge
			A patient was admitted, 2 beds are available
			A patient was admitted, 1 beds are available
			A patient was admitted, 0 beds are available
			Can not admit a patient, no beds available
			A patient was discharged, 1 beds are available
			A patient was discharged, 2 beds are available
			A patient was discharged, 3 beds are available
			There are no patients to discharge
				✓

Passed all tests! ✓

Correct

Marks for this submission: 1.00/1.00.

QUESTION 6

Correct

Mark 1.00 out of 1.00

Write a *checkAdult(age)* function whose input parameter is the *age* of the user age. The function checks whether the set *age* parameter is set correctly, if it is set incorrectly, the corresponding error should be generated and displayed in the console:

- if the *age* value has not been set, you need to create the following error: "Please, enter your age",

- If you set a negative *age* value, you need to create the following error: "Please, enter positive number",

- if a non-numeric value of *age* was specified, you need to create the following error: "Please, enter number",

- if the integer value of *age* was not specified, you need to create the following error: "Please, enter Integer number",

- If the user is under 18, you need to create the following error: "Access denied - you are too young!".

If there is no error, the message “Access allowed” is displayed in the console.

In the function, implement the handling of possible exceptions, providing the output to the console of the name and description of the error.

Regardless of whether the *age* parameter was set correctly or incorrectly, the message “Age verification complete” should be displayed at the end of the test.

For example:

Test	Result

`checkAdult(15);`	`Error Access denied - you are too young!`
	`Age verification complete`

`checkAdult(25);`	`Access allowed`
	`Age verification complete`

Answer: (penalty regime: 0 %)

Reset answer

```
function checkAdult(age) {
  try {
    if (age === undefined || age === null) {
      throw new Error('Please, enter your age');
    }
    if (age < 0) {
      throw new Error('Please, enter positive number');
    }
    if (typeof age !== 'number') {
      throw new Error('Please, enter number');
    }
    if (!Number.isInteger(age)) {
      throw new Error('Please, enter Integer number');
    }
    if (age < 18) {
      throw new Error('Access denied - you are too young!');
    }
    {
      console.log('Access allowed');
    }
  } catch (error) {
    console.log(`${error.name} ${error.message}`);
  } finally {
    console.log('Age verification complete');
  }
}


```

	Test	Expected	Got	

✓	`checkAdult(15);`	`Error Access denied - you are too young!`
		`Age verification complete`
			`Error Access denied - you are too young!`
			`Age verification complete`
				✓

✓	`checkAdult(25);`	`Access allowed`
		`Age verification complete`
			`Access allowed`
			`Age verification complete`
				✓

	Test	Expected	Got	

✓	`checkAdult();`	`Error Please, enter your age`
		`Age verification complete`
			`Error Please, enter your age`
			`Age verification complete`
				✓

✓	`checkAdult(0);`	`Error Access denied - you are too young!`
		`Age verification complete`
			`Error Access denied - you are too young!`
			`Age verification complete`
				✓

✓	`checkAdult(-22);`	`Error Please, enter positive number`
		`Age verification complete`
			`Error Please, enter positive number`
			`Age verification complete`
				✓

✓	`checkAdult("a25");`	`Error Please, enter number`
		`Age verification complete`
			`Error Please, enter number`
			`Age verification complete`
				✓

✓	`checkAdult(33.3);`	`Error Please, enter Integer number`
		`Age verification complete`
			`Error Please, enter Integer number`
			`Age verification complete`
				✓

✓	`checkAdult(16);`	`Error Access denied - you are too young!`
		`Age verification complete`
			`Error Access denied - you are too young!`
			`Age verification complete`
				✓

✓	`checkAdult(20);`	`Access allowed`
		`Age verification complete`
			`Access allowed`
			`Age verification complete`
				✓

Passed all tests! ✓

Correct

Marks for this submission: 1.00/1.00.

◄ Video. Closures [9:51]

Jump to...

Quiz. Asynchronous JS, Closures, Exceptions ►