Node.JS

Practice – 2

Overview

*In this hands-on session you will continue learning **Node.js**. Upon completion of this hands-on session, you should be able to:*

- *Understand the Call Stack used by Node*
- *Realize the Asynchronous behaviour of JavaScript*
  - *The working of Event Loop in the background*
  - *Identify Callback Hell*
- *Usage of Promise*
  - *PromiseStatus, Invoke response and reject callbacks*
  - *Chaining of Promise methods*
- *Realize what gets stacked in Microtasks and Macrotasks queues*
- *Using ES6+ keywords async / await keywords*
  - *For Defining Asynchronous functions*
  - *Suspend the execution with await keyword*

Hands-on

1. Write a Node.js program to illustrate **Call Stack**.

   - Make sure you have a few functions in the program.
   - It is your choice to have whatever function name you want.

   Record your observation on the sequence of execution of the functions.

2. Write a callback function which will be triggered after **5ms**.
   Use the **setTimeout()** built-in function for achieving this.

   [a] Execute your program and record your observation.
   [b] Include few statements before and after the callback function linked to the **setTimeout()** function is invoked.

   Record your observation of the same with respect to the execution of the statements.

   [c] On setting the timer parameter to 0, will we have a proper execution sequence.
   Check! Record your observation.

   [d] Why does that happen? Write that as a comment in the source code.

3. Build a **Simple Web Application** to understand the Asynchronous behaviour of JavaScript.

   The Web App should have a simple UI with a button widget.
   On the **'click'** event of the button a callback function should be triggered in the JS file to display some text on the console.
   Before and after invocation of the click event you can have some simple console.log statement to check the order of execution of the statement.

   With the help of Browser Developer Tools, **inspect** the behaviour of the application in the 'console'

   Record your observation!

4. Assume a scenario of **eCommerce** web site, where a user is placing order.

   The items are in the cart and now the order needs to be placed. So, in backend the situation could look something like this.

   ```
   amazon.createOrder()
   amazon.proccedToPayment()
   ```

   The **'proceedToPayment'** action is dependent on the **'createOrder'** action.

   Similarly, after **'proceedToPayment'** we have **'showOrderSummary'** and **'updateWallet'** actions to be performed

   Your task is to write a **pseudocode** to illustrate the same with Callback Functions.

5. Callbacks are great! On the other hand, they are difficult to arrange multiple things

   What we want to do is make a DIV and do a few things as follows:
   The initial text for the DIV element is 'Click Me!'

   [a] Change the text to 'Go' when clicked
   [b] Make it a circle after 1 second
   [c] Change background colour of circle to 'red' after 0.5 secs
   [d] Change the shape to square after 1 second
   [e] Change the background colour of square to 'green' after 0.5 secs
   [f] Fade out after 0.5 secs

   Let the initial setting for the 'body' and 'div' with 'go' be as follows:

   ```
   body {
        background: blue;
   }
   ```

```
.go {
     margin: 25px;
     background: white;
     padding: 25px;
     width: 100px;
     height: 100px;
     transition: all 0.2s;
}
```

6. Start Node.js in **REPL** mode and check the following:

   [a] Create a Promise object with an empty callback function
   Observe the **__proto__** of the **Promise** object and find out what does it contain.

   Make a note of the **Promise** object properties/methods

   [b] Create yet another Promise object where the constructor takes two of its traditional default callback function arguments by name **'resolve'** and **'reject'**.

   Now, to the **'resolve'** callback pass a string argument:**"I am getting resolved".**

   To the **'reject'** callback pass a string argument: **"Oops! Rejected."**

   Record your observation for the same.

   [c] Understanding chaining of the **'.then'** methods

   Invoke the Promise method **'resolve'** with the argument 5 and assign the instance to a variable like 'a'

   With 'a' as the **Promise** instance, invoke the **'.then'** method which has a callback function which doubles the value of it argument. Assign the instance to a variable like 'b'

   Repeat the above step with **Promise** instance 'b' and chain it to the **'.then'** method.

   Record your observation of chaining the **'.then'** method.

   This way we can overcome the 'Callback Hell'

7. Write a Node.js program to understand the behaviour of **Microtasks** and **Macrotasks** queue.

   [a] Where will the **WEB API** be placed?
   [b] Where will **Promise** callbacks and process **nextTick** be placed?

8. Once again start a Node.js **REPL** session.

   Define a function which will implicitly return a **Promise** object.

   Invoke the above defined function. Record your observation!

9. Write a Node.js program to illustrate how we can suspend an asynchronous function?

Additionally: If you find time, think of rewriting Q5 with:
[a] Using a **Promise** object,
[b] Using the **async / await** keywords

Write a Node.js program which will display text in different colours.