JavaScript Class July 10 2024

1. Asynchronous Programming
   1. Synchronous programming – things happen one at a time
      1. Problematic if task being executed takes time, like a network request
      2. Program can’t make progress
   2. Asynchronous programming – allows multiple things to happen at the same time
      1. When you start an action, the program continues to run
      2. When the action finishes, the program is informed and gets access to the result
   3. Example – program making two requests and combining results
      1. Synchronous – make the requests one after the other
         1. Drawback – second request will be started only when the first one has finished
            1. Total time taken will be at least the sum of the two response times
         2. Solution – start additional threads of control
            1. Thread – another running program whose execution may be inserted along with other programs by the operating system

Threads could run at the same time on different processors

* + - * 1. A second thread could start the second request, then both threads wait for results to come back, then they resynchronize to combine results
        2. JavaScript does not rely on threads but on asynchronous
  1. Callbacks
     1. An approach to asynchronous programming
        1. Make functions that need to wait for something take a function as an extra argument
        2. The extra argument is the callback function
        3. The asynchronous function starts a process, sets things up so that the callback function is called when the process finishes, and then returns
     2. Example – setTimeout function
        1. Waits a given number of milliseconds and then calls a function  
             
           setTimeout(() => console.log(“Tick”), 500);
     3. Example – reading a file from a device’s storage
        1. Function that reads a file’s content as a string and passes it to a callback function  
             
           readTextFie(“shopping\_list.txt”, content => {  
            console.log(`Shopping List:\n${content}`);  
           });  
           // Shopping List:  
           // Peanut butter  
           // Bananas
* Asynchronous programming
  + Definition
  + Callbacks
    - Allow us to wait on certain code to finish execution before running the next bit of code
  + Promises
    - Like callbacks, allow us to wait on certain code to finish execution before running the next bit of code
    - Represents the eventual result of an asynchronous operation
    - A placeholder - an object on which we can attach callbacks
    - Promise states:
      * Pending - async operation hasn’t completed yet
      * Fulfilled - Operation has completed and the promise has a value
      * Rejected - Operation has completed with an error or has failed
      * A promise is settled if it is not pending
      * Once a promise has settled, it is settled for good and cannot transition
    - Promise syntax:
      * runFunction().then(successFunc, failureFunc);
      * First, invoke runFunction()
      * runFunction() returns a Promise
      * Once Promise is settled:
        + If Promise is fulfilled, successFunc is invoked
        + If Promise fails, failureFunc is. Invoked

**JavaScript Promises**

A Promise is a guarantee that something will happen in the future. When that time comes, the Promise will be kept or not kept, just like with people. A JavaScript Promise is a placeholder for something that is supposed to be returned from the server. (Why do we need a placeholder?)

A Promise takes a callback function as a parameter. That callback function takes two functions as parameters - resolve and reject. In order to use those functions, you need to include them as parameters (is this true?). You can name them whatever you want - the first parameter will always be resolve and the second parameter will always be reject. Even though they are functions, you don’t include the parentheses after them because if you did they would execute immediately.

The callback function passed as a parameter for the Promise can be called promise\_executor. The resolve function is a callback provided by the promise instance. So is the reject function. Each take one parameter that is of type Object. If the Promise is resolved, the resolve method passes the parameter to the .then callback function. If the Promise is rejected, the reject method passes the parameter to the .catch callback function.