Asynchronous & Hoisting

phuong.vu@htklabs.com

Agenda

- Asynchronous Nature of Javascript
- Callbacks
- Promises
- Hoisting

Asynchronous? What’s Asynchronous?

- General concepts of programming languages introduce two distinct mechanisms of concurrency, namely synchronous and asynchronous control flow.
- **Synchronous** control flow means that a process runs only as a result of some other process being completed or handing off operation.
- **Asynchronous** control flow means that a process operates independently of other processes.
- JavaScript runs in the browser which is itself a single process on the operation system.
- JavaScript runs in a single thread within the browser process.

Synchronous Code

- We could explain synchronous control flow by explaining how to read a book:
  - take the book
  - only when you took the book: read page 1
  - only when you read page 1: read page 2
  - only when you read page 2: read page 3
  - …
  - only when you read the last page: burn the book
- In synchronous, each step is executed only when the previous step has completed.

Asynchronous Code

- If these actions are asynchronous, we can’t ensure that the first step is finished firstly, the second step secondly etc:
- However, since JavaScript is single-threaded, it’s a little different and asynchronous control flow doesn’t exactly mean that - a process operates independently of other processes.
- Instead, the script is always run completely first, while all asynchronous actions are queued up by the browser, and only when the script is run completely (the single script thread), the asynchronous actions are executed. ***Always remember:*** the whole script is executed before any asynchronous action is performed!

Asynchronous Code

- The difference can be confusing since determining if a function is asynchronous or not depends a lot on context
- Ex:

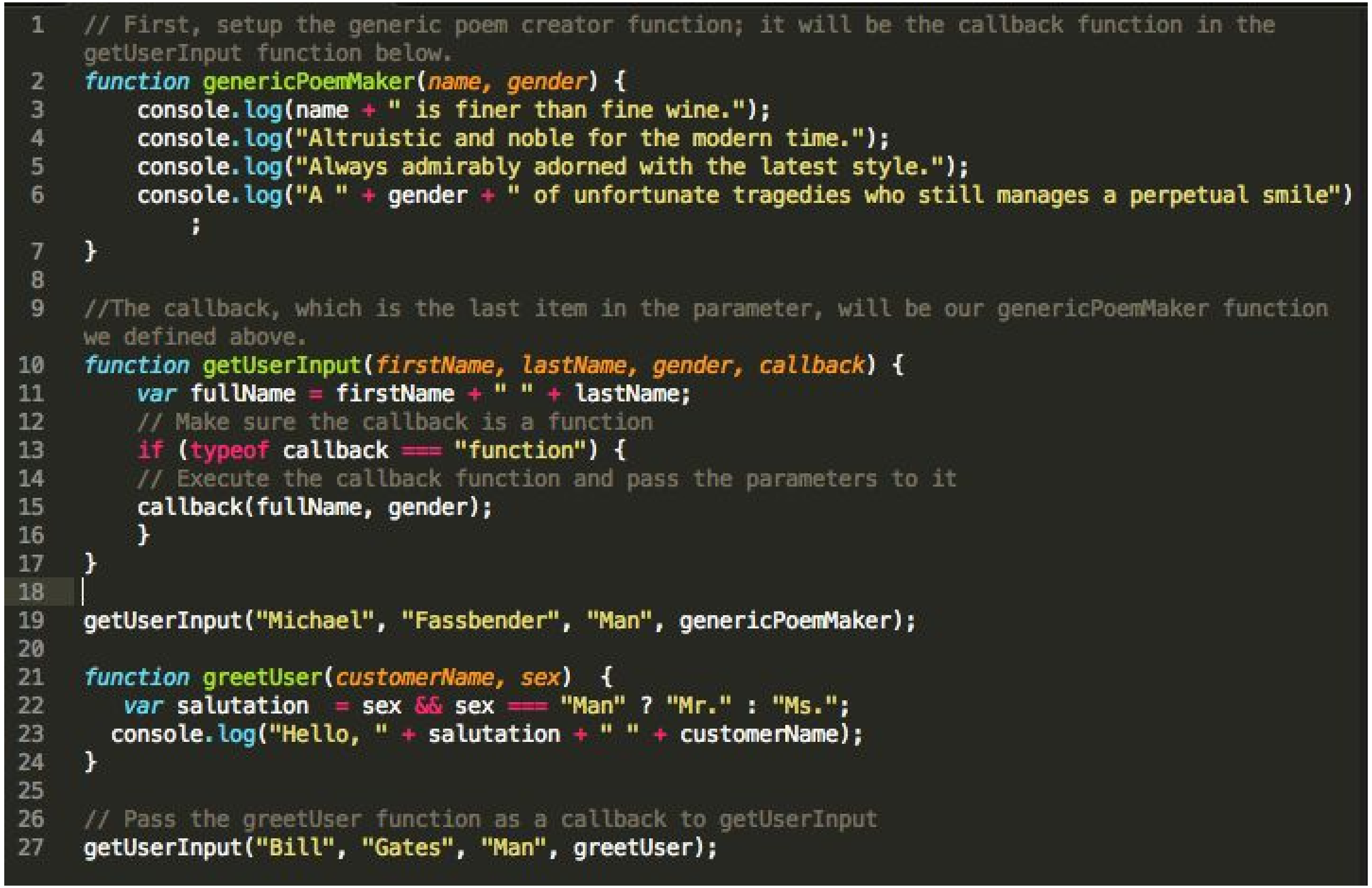
```
var myNumber = 1;
function addOne() { myNumber++ } // define the function
addOne(); // run the function
console.log(myNumber); // logs out 2
```

- Ex:

```
var myNumber = 1;
setTimeout(function() {
      myNumber++;
}, 500 );
console.log(myNumber); // logs out 1
```

Callbacks

- Callbacks are functions that are executed asynchronously, or at a later time.
- Is a function that is passed to another function as a parameter, and is called (or executed) inside the other function.



```javascript
// First, setup the generic poem creator function; it will be the callback function in the getUserInput function below.
function genericPoemMaker(name, gender) {
    console.log(name + " is finer than fine wine.");
    console.log("Altruistic and noble for the modern time.");
    console.log("Always admirably adorned with the latest style.");
    console.log("A " + gender + " of unfortunate tragedies who still manages a perpetual smile");
}

//The callback, which is the last item in the parameter, will be our genericPoemMaker function we defined above.
function getUserInput(firstName, lastName, gender, callback) {
    var fullName = firstName + " " + lastName;
    // Make sure the callback is a function
    if (typeof callback === "function") {
    // Execute the callback function and pass the parameters to it
    callback(fullName, gender);
    }
}

getUserInput("Michael", "Fassbender", "Man", genericPoemMaker);

function greetUser(customerName, sex)  {
   var salutation  = sex && sex === "Man" ? "Mr." : "Ms.";
  console.log("Hello, " + salutation + " " + customerName);
}

// Pass the greetUser function as a callback to getUserInput
getUserInput("Bill", "Gates", "Man", greetUser);
```

How Callback Functions Work?

- When we pass a callback function as an argument to another function, we are only passing the function definition.
- We are not executing the function in the parameter.
- And since the containing function has the callback function in its parameter as a function definition, it can execute the callback anytime.

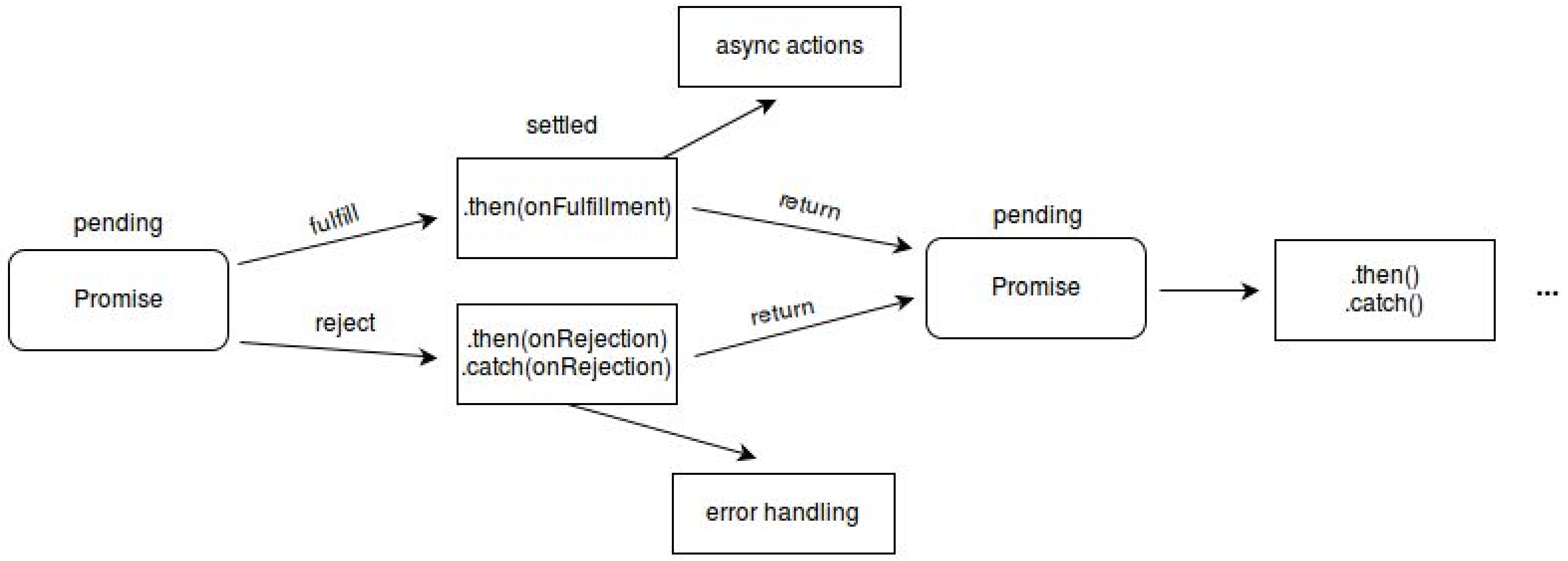
What is problem of callback?

```
getData(function(a){
    getMoreData(a, function(b){
        getMoreData(b, function(c){
            getMoreData(c, function(d){
                getMoreData(d, function(e){
                    ...
                });
            });
        });
    });
});
```

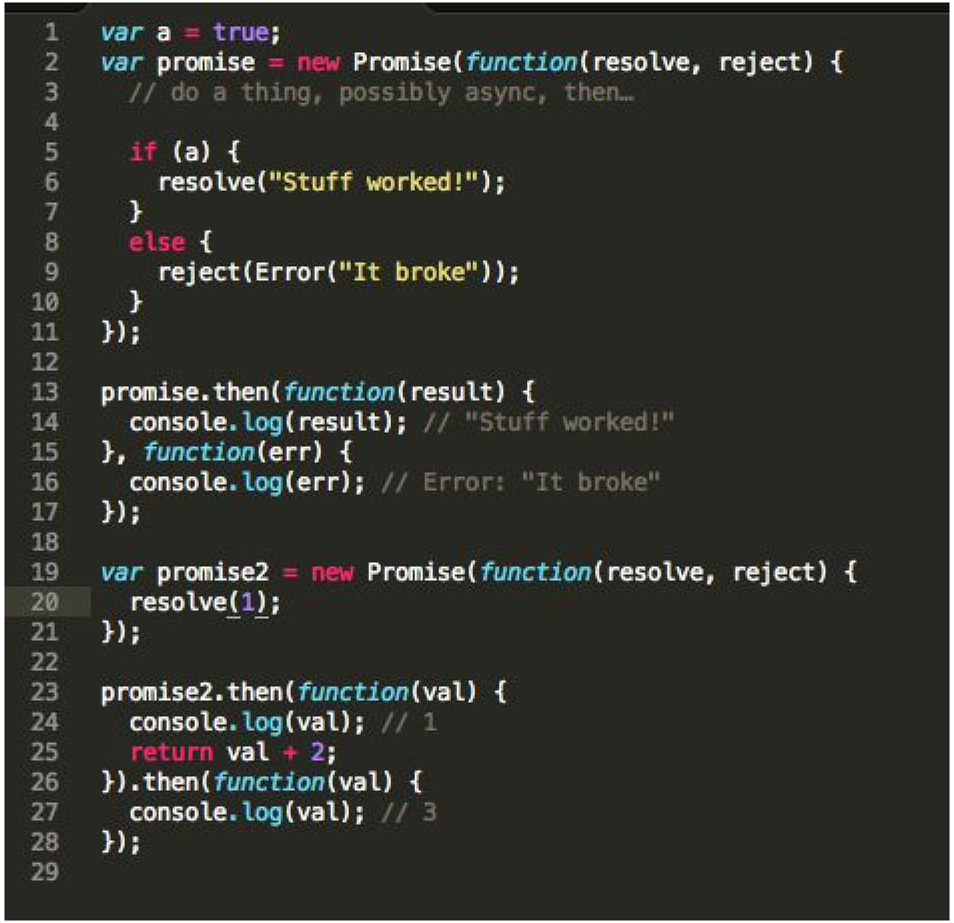
Promises

- A promise represents the eventual value returned from the single completion of an operation.
- A Promise represents an operation that hasn't completed yet, but is expected in the future.
- A promise is in one of three different states:
  - pending - The initial state of a promise.
  - fulfilled - The state of a promise representing a successful operation.
  - rejected - The state of a promise representing a failed operation.
- We use **new Promise** to construct the promise.

Promises



pending
Promise
fulfill
reject
settled
.then(onFulfillment)
.then(onRejection)
.catch(onRejection)
async actions
error handling
return
return
pending
Promise
.then()
.catch()
...



```
var a = true;
var promise = new Promise(function(resolve, reject) {
  // do a thing, possibly async, then…

  if (a) {
    resolve("Stuff worked!");
  }
  else {
    reject(Error("It broke"));
  }
});

promise.then(function(result) {
  console.log(result); // "Stuff worked!"
}, function(err) {
  console.log(err); // Error: "It broke"
});

var promise2 = new Promise(function(resolve, reject) {
  resolve(1);
});

promise2.then(function(val) {
  console.log(val); // 1
  return val + 2;
}).then(function(val) {
  console.log(val); // 3
});
```

Understanding JavaScript hoisting

- Is JavaScript's default behavior of moving declarations to the top.
- Is when JavaScript moves variable and function declarations to the top of their scope before any code is executed.

Always declare variables at the top of scope

Declarations vs Expressions

Function Declarations

```
function hello() {
    alert('hello');
}
```

Function Expressions

```
var hello = function() {
    alert('hello');
};
```

Work as expected

```
x();
function x() {
    alert('I am x');
}
```

Error

```
x();
var x = function() {
    alert('I am x');
}
```

Declarations vs Initializations

- `var x = 7;` : both a declaration and an initialization
- `var x;` : is a declaration
- `x = 7;` : is initialization

Assignment

1. Write a function that add 2 numbers a and b, b=0 if undefined and return after (a+b) ms as callback

Write other function which perform:

- Generate a random number N
- Call above function to add numbers (N, undefined)
- In callback, get result T and repeat
    - Generate a random number N
    - Call above function to add this random number and result of previous (N, T)

Repeat 3 times

Assignment (con’t)

2. Implement assignment 1 using promise
3. Find other solution
4. Self-check code with jslint

Q&A

Thanks for your listening!