Google | طويق

Advanced JavaScript

أكاديمية طويق
TUWAIQ ACADEMY

وزارة الاتصالات وتقنية المعلومات
MINISTRY OF COMMUNICATIONS AND INFORMATION TECHNOLOGY

الاتحاد السعودي للأمن السيبراني والبرمجة والدرونز
SAUDI FEDERATION FOR CYBERSECURITY, PROGRAMMING & DRONES

مهارات المستقبل

stc

Objectives

- Arrow Function
- Basic Data Structures (Queue - Stack - Tree)
- Asynchronous programming
- Event Queue

Some Basic Computer Science Data Structures

1- Queue

data enters the queue at the back and leaves from the front; a first in, first out (FIFO) data structure.

Queue terminology:

- **Enqueue** - an operation that adds an item to the back of a queue.
- **Dequeue** - an operation that removes an item from the front of a queue.

Source: https://tutsmaster.org/how-to-represent-stack-in-c-data-structure/

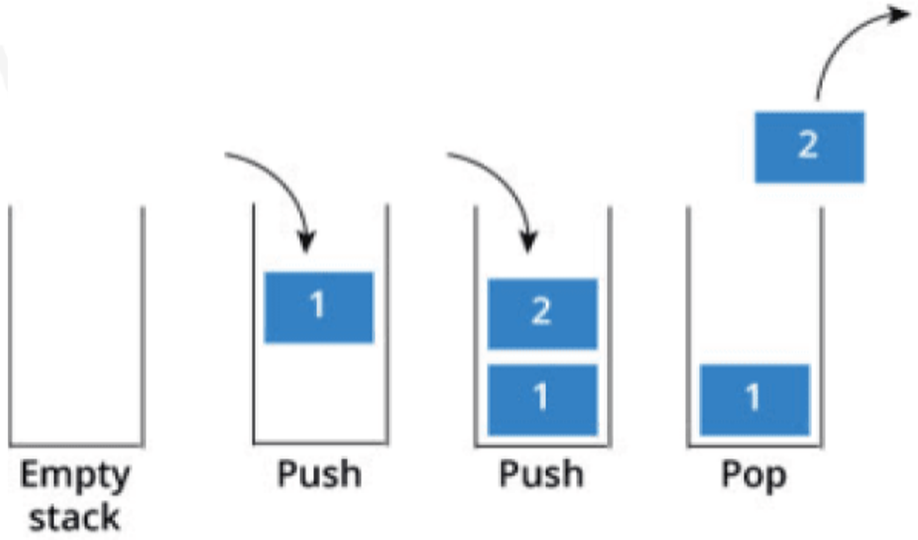
Some Basic Computer Science Data Structures

2- Stack

- items are added to the top of the stack and removed from the top of the stack;
- a last in, first out (LIFO) data structure.

Stack terminology:

- **Push** - an operation that adds an item to the top of a stack.
- **Pop** - an operation that removes an item from the top of a stack.



Source: https://tutsmaster.org/how-to-represent-stack-in-c-data-structure/

Some Basic Computer Science Data Structures

3- Tree

stores data in a hierarchy made up of a collection of nodes connected by edges.

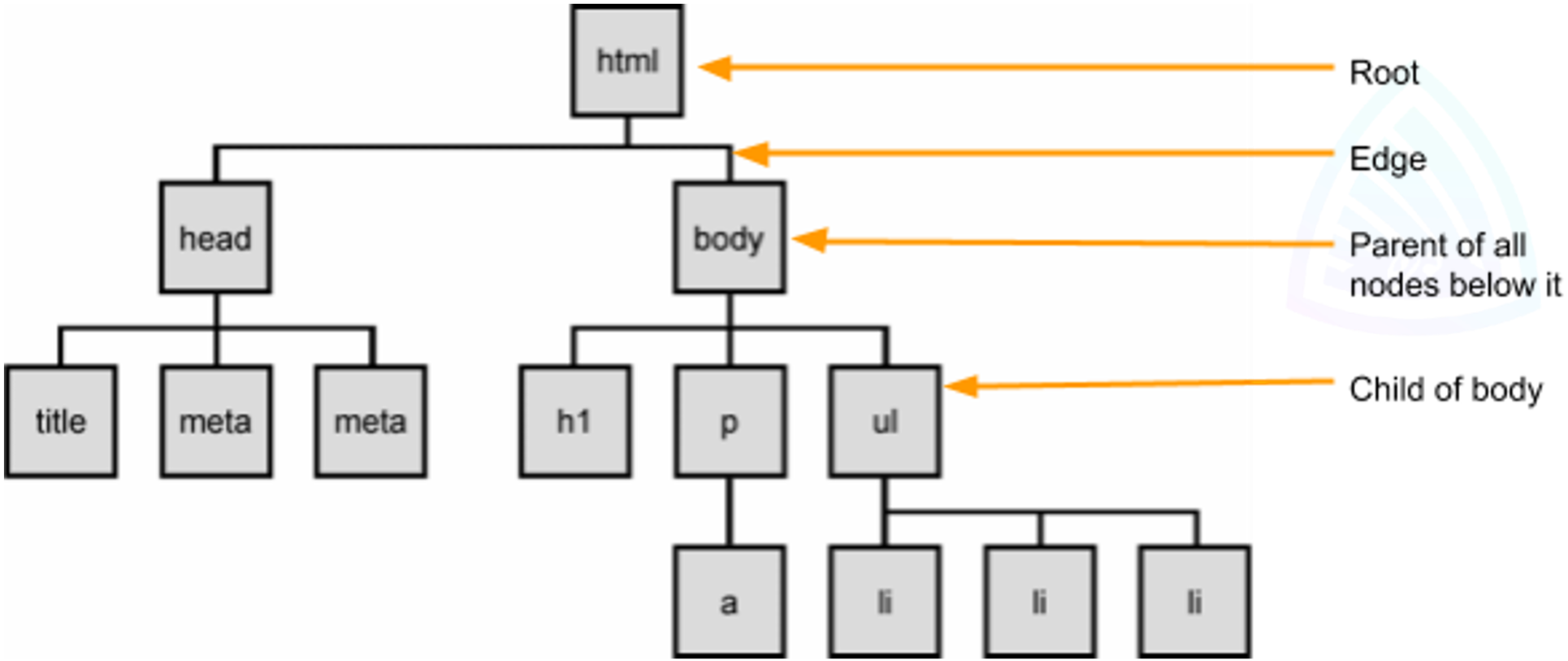


Image modified from https://programmingfordigitalart.wordpress.com/2015/03/05/w6-html-and-js/

Arrow Function

An alternative way to write a traditional function in JavaScript is to use the arrow function expression

```
function(a, b){
        return a + b
}

(a,b) => return a + b
```

```
function sum(a, b){
        return a + b
}

let sum = (a,b) => a + b
```

Asynchronous Programming

Synchronous code: when lines of code are executed in order

Asynchronous code: The order to execute each line of code is given as it is encountered BUT your program doesn't stop and wait for an instruction to be completely executed before moving on to the next instruction.

Everything can move **quicker** because other tasks can execute in the background, but we need to make sure they are **executed in the correct order.**

Asynchronous Programming

Callback functions: a function that is passed as an argument to another function. The callback isn't executed right away. It is executed later, somewhere within the calling function.

```
function a(s, callback) {
    callback(s);
}

function b(s) {
    console.log(s + "! It's me!");
}

//call function 'a' with a predefined callback
'b'
a("Hello world", b);

//call function 'a' with a custom callback.
a("Hello world", function(s) {
    console.log(s + ", I can use callbacks");
});
```

Asynchronous Programming

A useful method that requires a callback function to work is the `map()` function, which **creates a new array** with the results of calling a provided function on **every element** in this array:

```
// create an array
let numbers = [1, 4, 9, 16];

// call map(), passing a function
let mapped = numbers.map(function(x) { return x * 2 });

// log the result
console.log(mapped);

// Output would be: 2,8,18,32
```

Asynchronous Programming

Timing events: with JavaScript you can make certain functions of code wait for a certain amount of time before being executed.

- setTimeout: `setTimeout(aFunction, 3000);`
- setInterval: `setInterval(aFunction, 3000);`

Asynchronous Programming

Promises: allows the interpreter to carry on with other tasks while a function is executed without waiting for that function to finish.

- `doSomethingThatReturnsAPromise().then(successCallback, failureCallback);`

```
require('isomorphic-fetch');

fetch("https://www.example_API.com/")
    .then(res => res.json())
    .then((result) => {
          console.log(result);
       }),     (error) => {
       console.log(error);
      }
```

Asynchronous Programming

You could also **create** your own promise object:

```
let promise = new Promise(function(resolve, reject) {
  // do something, possibly async, then...

  if (/* everything turned out fine */) {
    resolve("Stuff worked!");
  }
  else {
    reject(Error("It broke"));
  }
});


//Code example src:
//https://developers.google.com/web/fundamentals/primers/promises
```

Google | طويق

Asynchronous Programming

Async functions: can contain an `await` expression that pauses the execution of the async function and waits for the passed Promise's resolution, and `then` resumes the async function's execution and returns the resolved value

```
require('isomorphic-fetch');

const request = async () => {
    const response = await    fetch("https://www.example_API.com/");
    const json = await response.json();
    console.log(json);
}
request();
```

How Does JavaScript Work?

- JavaScript is a **hosted language** in that it never runs on its own; rather it runs in a container.
- The **container** can be the **browser** (on the client-side) or in **Node.js** (on the server-side).
- JavaScript uses an **event-driven model** with a **single thread** of execution.

JavaScript's Event Loop

1. When your source code is **compiled**, the functions are **dequeued** onto the call stack and executed by the interpreter.

1. When a function with a callback is encountered, the callback is sent to an API. Once the **API** is finished processing the callback function, it pushes the callback onto the task queue.

1. The event loop sits between the **call stack** and the **task queue**. When the call stack is empty, the event loop will dequeue a function from the task queue onto the call stack where it will be executed.

How Does JavaScript Work?

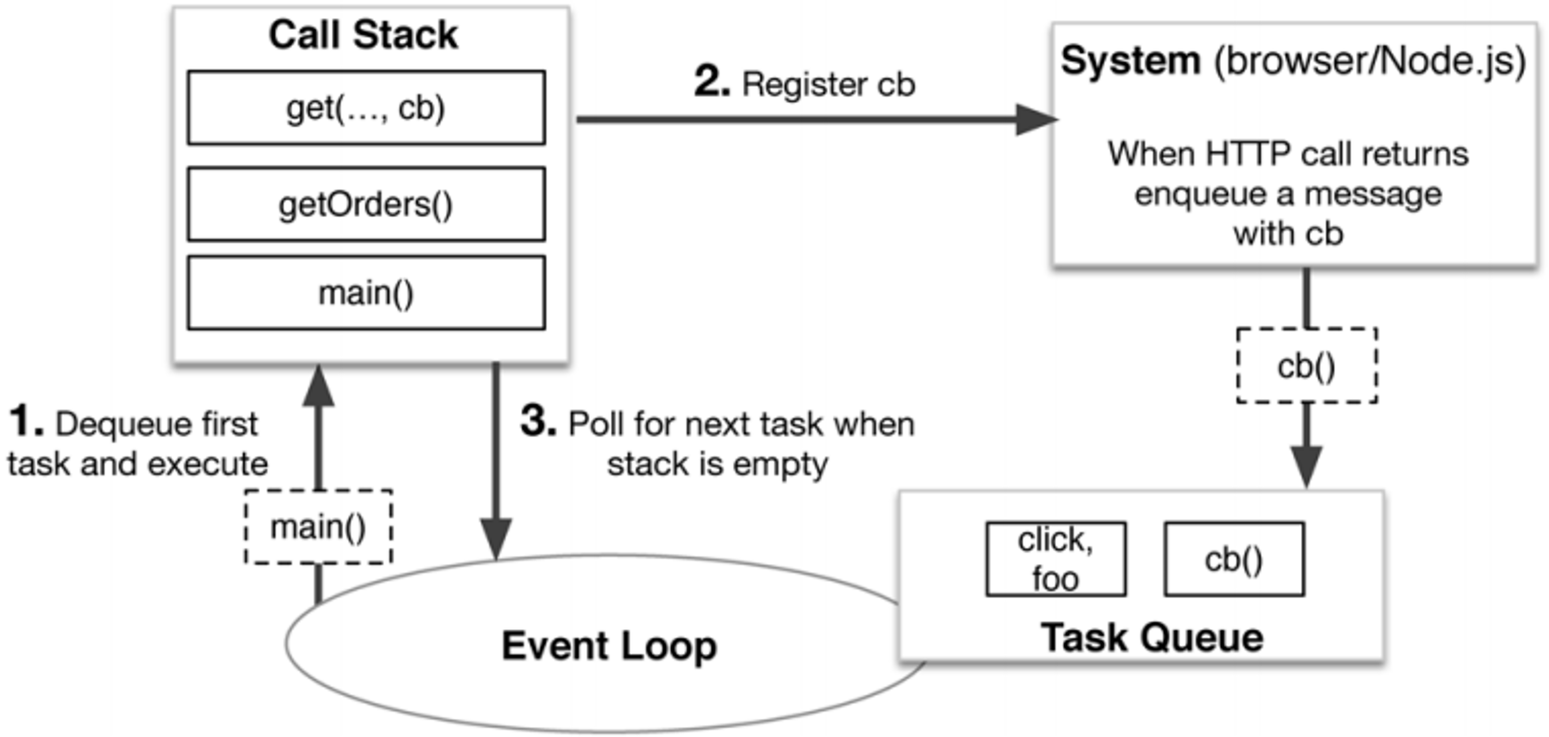


Image from article by Gallaba, Mesbah & Beschastnikh

Resources

- What is the JavaScript event loop really all about - Java Brains

Summary

- Basic Data Structures (Queue - Stack - Tree)
- Arrow Function
- Asynchronous programming
- Event Queue