# Table of Contents

# Welcome to *Front-end Interview Questions*

This gitbook is a **collection** of the interview questions I have either came across in an real interview or seen during research.

Join this project: https://github.com/arthur-zheng/FEIQ

Email: arthur.z.me 'at' gmail.com.

Thanks again for stopping by.

## Content:

1. Javascript Questions `in progress`
2. HTML\/CSS Questions `future`
3. Algorithms `future`

## Special Thanks:

1. www.Stackoverflow.com
2. www.javascriptissexy.com
3. ...

Long live Javascript!

# Declaration

```
function test() {
    var a = b = 100;
}
test();
console.log(b);        // ?
```

The answer is 100 instead of `undefined`. Following is what actually happened:

```
function test() {
    var a = undefined;
    b = 10;
    a = b;
}
...
```

### How about using 'use strict'?

```
(function() {
    'use strict';
    var a = window.b = 5;
})();
console.log(b);
```

# NaN

### 1. When is NaN be produced?

```
// Only '+' sign will try to do some concat.
// Other operations ( / * - ) will produce NaN when number and s
tring are mixed.
console.log('abc'/4);
console.log(4*'a');

// How to check NaN:
Number.isNaN(NaN);    // true, only if is NaN

// Be careful of pitfalls from isNaN() (not Number.isNaN()):
isNaN({});          // true
isNaN('a');         // true
typeof(NaN);        // "number"
NaN === NaN         // false
```

## + - * /

### 1. What will be logged?

```
let a = 10/3;
a === 3;

console.log(2 + '1');            // 21
console.log('2' + 1);            // 21

console.log(2 / '1');            // 2
console.log(2 - '1');            // 1
console.log('2' - 1);            // 1

console.log('2' - 'a');          // NaN

console.log(-'1');               // -1
console.log(+'-1');              // -1
```

### 2. Does it equal? Why?

```
const n = 0.1 + 0.2;
console.log(n === 0.3);
```

## Check Types

### 1. How to check if is `object` ?

```
toString.call({});                      // "[object Object]"
Object.prototype.toString.call(obj);    // "[object Object]"
if (obj instanceof Object) {...}        // true
```

### 2. How to check `array` ?

```
const arr = [];

Array.isArray([]);                      // true, IE 9+
toString.call([]);                      // "[object Array]"
Object.prototype.toString.call(arr);    // "[object Array]"
if (arr instanceof Array) {...}         // true
```

### 3. How to check `null` , `undefined` , `NaN` ?

```
val === null;             // true (if it is null)
val === undefined         // true (if it is undefined)
Number.isNaN(NaN)         // true
```

## `forEach()` , `for...in` and `for...of`

What's the difference?

1. `forEach` is only for `Arrays` .
2. `for...of` is only for `iterable objects` , means cannt use it on `objects` .
3. `for...in` is used for loop properties of `objects` . But it only loops `enumerable properties` . For example it doesn't touch an array's

`length` property.

4. `for...of` only work with `collections`.
5. loops cannot be stopped for all of the 3.

---

## References


1. *Check if object is array?* http://stackoverflow.com/questions/4775722/check-if-object-is-array
2. *“foreach” vs “for of” vs “for in” in JavaScript* http://qnimate.com/foreach-vs-for-of-vs-for-in-in-javascript/

# The *this* keyword

The this keyword is always **confusing**. Especially when functions are invoked in different ways:

1. As a method, like `obj.foo()`
2. As purely function, like `foo()`
3. As a constructor, like `new Student()`
4. Indirectly using `apply()` , `call()` , `bind()` and `()=>{}`

## 1. As a Method, as in Java

```
const jon = {
    firstName: 'Jon',
    lastName: 'Snow',
    fullName() {
        console.log(`${this.firstName} ${this.lastName}`)

    }
    weapon: {
        name: 'Longclaw',
        use() {
            console.log('Pew, ${this.name} used.');
         }
     }
}
// jon [dot] fullName()
jon.fullName();                // "Jon Snow"
// jon.weapon [dot] use()
jon.weapon.use();              // "Pew, Longclaw used."
```

Invoked as method works the same as Java.

`fullName` method was invoked as a method of `jon` . Because there is a `jon.` right before it. So `jon` will passed into `fullName()` as its `this` . Samely, `jon.weapon` was passed into `use()` as `this` .

**Takeaway**: *Anything* before the last **[dot]** will be passed into the method as the method's `this` keyword.

But there's nothing or even [dot] before the function? Read on.

## 2. As Purely Function (Global)

```
function foo() {
    console.log(this);    // this means, whatever passed into fo
o as this
}
foo();                    // ?
```

Keep in mind that everything in Browser happens within the `window` scope:

```
var a = 100;              // a dangerous global variable
console.log(window.a)     // 100
console.log(this.a)       // 100
console.log(this === window); // true
console.log(this.document === document); // true
```

So, the `foo` ( or `window.foo` ) was invoked and thus the **default** `this` ( or `window` ) was logged.

**Takeaway**: In browser, `window` is the default `this` .

A little bit more tricky interview question in here, by combining tip-1 and tip-2:

```
const jon = {
    firstName: 'Jon',
    lastName: 'Snow',
    fullName() {
        console.log(`${this.firstName} ${this.lastName}`)
    }
}
// pull the funtion out
const fullNameOutside = jon.fullName;
// invoke it as pure function
fullNameOutsite();            // "undefined undefined", as this
is window
```

Since there's nothing or [dot] before the `fullNameOutside`, it is invoked as a **pure function**, thus `window` will be the `this`.

**Takeaway**: Where/how a functions was **declared** is much less important than **how** the function was **invoked**.

## 3. As Constructor

```
// a constructor
function Student(id) {
    this.id = id;
    console.log(this.id);      // ?
}
// new instance
const Tom = new Student(10092);
```

**Takeaway**: What happends when a constructor is invoked (in the above code for example):

1. A new empty object `{}` was created.
2. Tom was pointed to the new empty object.
3. `Tom` was passed into to the constructor `Student()` as `this`.
4. Execute the constructor.

So, in here, the `this` will be the new instance object `Tom`.

## 4. As with `apply()`, `call()` and `bind()`

Since this is so flexible, what if sometime we want to mannally control the `this`? `apply()`, `call()` and `bind()` was created to do solve this problem.

```
function showFullName() {
    return `${this.firstName} ${this.lastName}`;
}
showFullName();                // "undefined undefined"

const jon = {
    firstName: 'Jon',
    lastName:  'Snow'
}
showFullName.apply(jon);       // 'Jon Snow'
showFullName.call(jon);        // 'Jon Snow'
showFullName.bind(jon)();      // 'Jon Snow'
```

## 5. [dot]-Rule Exception: Arrow Functions

Sorry the above [dot] rule doesn't work in here.

In ECMAScript 6, arrow functions was introduced. One of arrow function's features is that it **automatically** bind `this` for you when arrow function was declared. This feature will definitly confuse a lot of beginners.

```
const jon = {
    firstName: 'Jon',
    lastName: 'Snow',

    fullName: () => `${this.firstName} ${this.lastName}`,
    // Same as:
    fullName: function() {
        return `${this.firstName} ${this.lastName}`
    }.bind(this);

    getThis: () => this,
    // Same as
    getThis: function() {return this}.bind(this)
}
jon.fullName();                // "undefined undefined"
jon.getThis();                 // window
```

What Babel (https://babeljs.io/repl) compiles proves our conclusion:

Before:

```
function test() {
    const jon = {
        firstName: 'Jon',
        lastName: 'Snow',
        fullName: () => `${this.firstName} ${this.lastName}`,
    }
    jon.fullName();             // "undefined undefined"
}
test();
```

After:

```
function test() {
    var _this = this;
    var jon = {
        firstName: 'Jon',
        lastName: 'Snow',
        fullName: function fullName() {
            return _this.firstName + ' ' + _this.lastName;
        }
    };
    jon.fullName();             // "undefined undefined"
}
test();
```

We can tell is that, the arrow function binds *the scope which wraps the outside object* with *this*.

**Takeaway**: The [dot] rule **doesn't** work with arrow function since `this` was already **invisibly** binded/specified as soon as arrow function was declared.

## Conclusions

1. The default `this` is window in browser.
2. Whatever was before the [dot] will be passed into the method as `this`. This rule **doesn't work** with arrow functions, since their `this` was already binded when being declared.
3. How one function was **invoked** is way more important than how/where it was **declared**.

---

## References:


1. *Understanding Javascript's this With Clarity, and Master It*: http:\/\/javascriptissexy.com\/understand-javascripts-this-with-clarity-and-master-it\/
2. stackoverflow: http://stackoverflow.com/questions/2130241/pass-correct-this-context-to-settimeout-callback

3. *this* (MDN) https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/this

# In the Real World

Make sure you have read last chapter *this - basics* before you read following.

One reason Javascript looks complicated is: Javascript are usually used to deal with DOM, Async calls and libs written by random people... Things get complicated, in real world, in real interviews.

## 1. Callbacks Without Auto Binding

`setTimeout()` is a popular one in interviews:

```
const tom = {
    examAnswer: 'x = 100',
    startExam() {
        setTimeout(function() {
            console.log(`My answer is: ${this.examAnswer}`);
        }, 1000);
    }
}
tom.startExam();
```

Answer is code will not work as expected since `this` is `window` inside the callback. `forEach()` has the same issue:

```
const daenerys = {
    home: "King's landing",
    dragons: [
        {name: 'Drogon', location: 'The Wall'},
        {name: 'Rhaegal', location: 'Castel Black'},
        {name: 'Viserion', location: 'Winterfell'}
    ],
    // trying to set dragon's location to daenerys.home
    dragonsGoHome() {
        // attention: callback in forEach()
        this.dragons.forEach(function(dragon) {
            dragon.location = this.home;        // this === wind
ow
        });
    }
}
daenerys.dragonAttack();
daenerys.dragons[0].home;    // undefined
```

The this will become `window` again in the callbacks. Why? because in the above `setTimeout()` or `forEach()`, callback is invoked in purely function form. Kind of like:

```
// a function supports callback
function deleteDomNodesWithCallback(parentNode, callback) {
    // do the deleting stuff
    ...
    // callback is invoked as a purely function
    callback();
}
deleteDomNodesWithCallback(iAmTheNode, iAmCallback);
```

Since we know what just happened, how to **fix** `this` by specifying the right `this`?

```
// Way 1:
// using 'that', or closure
const tom = {
    examAnswer: 'x = 100',
    startExam() {
        var that = this;
        setTimeout(function() {
            console.log(`My answer is: ${that.examAnswer}`);
        }, 1000);
    }
}
tom.startExam();

// Way 2:
// use bind(), a cleaner & better way
const tom = {
    examAnswer: 'x = 100',
    startExam() {
        setTimeout(function() {
            console.log(`My answer is: ${that.examAnswer}`);
        }.bind(this), 1000);
    }
}
tom.startExam();
```

And at last a fix for Daenerys:

```
var daenerys = {
...
 dragonsGoHome() {
    var _that = this;         // created only for demo purpose
    this.dragons.forEach(function(dragon) {
        // inside the callback
        dragon.location = this.home;
    }.bind(this));           // `this` equals to _that
 }
...
// daenerys [dot] dragonsGoHome
daenerys.dragonsGoHome();
```

Since this part once confused me for a long time, please allow me to assume that you need a final explanation:

1. When we call `daenerys.dragonsGoHome()`, we pass `daenerys` into method `dragonsGoHome()` as `this`.
2. So inside `dragonsGoHome`, `this` equals to `daenerys` object.
3. The `bind(this)` is inside the same context/this-scope as `dragonsGoHome()`. So `this` equals `daenerys`.

## 2. Callbacks With Auto Binding

Some libary such as jQuery, or DOM API binds `this` for you in the background. Unlike above.

```
// jQuery.js
$('button').click(function(event) {
    console.log(this);
});
// the button object will be logged

// Pure DOM API
document.querySelector('html').addEventListener('click', function
() {
    console.log(this);
});
// the html DOM object will be logged
```

Keep in mind those functions' behavior.

## 3. HTML Inline (Auto Binding)

Not a recommended practise since we always want to seperate Representing (HTML) and Logic (Javascript).

```
<button onclick="alert(this.tagName.toLowerCase());">
    Show this
</button>

// 'button'
```

## 4. Closure

Looks something new at first glance but exactly what we have seen before:

```
var rhaegal = {
    name: 'Rhaegal',
    layEgg() {
        // hidden private variable in closure
        let eggs = 0;
        const lay = function() {
            eggs++;
            console.log(`${this.name}'s eggs are: ${eggs}.`)
        }
        return lay();
    }
}
rhaegal.layEgg();
```

Answer is: `'s eggs are 1.`

**Remember**: How a function is invoked is the most important thing. The function `lay()` is invoked as `purely function` thus the `this` inside will be `window`. Simply like that.

How to fix? Similarly to the above example `tom`. We can either use `that` or `bind`. Or arrow function `=>`:

```
var rhaegal = {
    name: 'Rhaegal',
    layEgg() {
        // hidden private variable in closure
        let eggs = 0;
        const lay = () => {
            eggs++;
            console.log(`${this.name}'s eggs are: ${eggs}.`)
        }
        return lay();
    }
}
rhaegal.layEgg();                    // Rhaegal's eggs are: 1.
```

## References:

1. *Understanding Javascript's this With Clarity, and Master It*: http:\/\/javascriptissexy.com\/understand-javascripts-this-with-clarity-and-master-it\/
2. *this* (MDN) https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/this
3. *The Final Steps to Mastering JavaScript's "this" Keyword* https://www.sitepoint.com/mastering-javascripts-this-keyword/

# Hoisting

Hoisting means lifting.

It was actually created to *simplify*(wait, what?) Javascript originally, by avoiding referencing error. After hoisting, we can use a variable before it's declared (unlike Java).

## 1. Definition and Example

```
function test() {
    console.log(val);        // ?
    console.log(foo());     // ?
    var val = 1;
    function foo() {
        return 2;
    }
}
test();
```

Here's what actually happened:

```
function test() {
    // a variable declaration will only be hoisted to it's own s
cope's top
    var val = undefined;
    function foo() {
        return 2;
    }
    console.log(val);        // val was lifted but is init as un
defined
    console.log(foo());     // foo() was lifted
    val = 1;
}
test();
```

The behavior that Javascript Interpreter (such as Google V8) hoists/lifts the declaration part up to the scope's top is called `hoisting`.

# 2. Types of Hoisting

## 1. Declare + Assign:

1. variables: `var val = 10`.
2. functions: `var foo = function() {...}`

For these forms, only the left/declaration part will be hoisted. As `var val = undefined`, `var foo = undefined` will be lifted to top.

```
test() {
    ...
    var val = 100;
    var foo = function() {...};
}
// Becomes:
test() {
    var val = undefined;
    var foo = undefined;
    ...
    val = 100;
    foo = function() {...};
}
```

## 2. Declare-only:

For functions declared as `function foo() {...};` will be fully lifted to the top of the scope.

```
test() {
    var foo = 2;
    ...
    function foo() {...};
}
// Becomes:
test() {
    var foo;
    function foo() {...};
    // all declarations are on top
    foo = 2;
    ...
}
```

## 2. With ES6

```
function test() {
    console.log(val);
    let val = 1;
}
test();
```

We got an error says: `Uncaught ReferenceError: val is not defined`. This is because `let` and `const` will not be hoisted.

## 3. A Little Bit More Complex (Ref.1):

```
var num = 1;
function() {
    if (!num) {
        var num = 100;
    }
    console.log(num);
}
```

Answer is 100. Why? Here's what actually happens:

```
var num = 1;
function() {
    var num = undefined;
    if (!num) {             // true
        var num = 100;
    }
    console.log(num);
}
```

Remember declaration will be hoisted to top of it's scope. And `if` doesn't create new scope for `var`.

What about we use const?

```
const num = 1;
function() {
    if (!num) {
        const num = 100;
    }
    console.log(num);
}
```

Will get error: `Uncaught TypeError: Identifier 'num' has already been declared`. Because:

1. `const` will not be hoisted, thus we will get into `if`.
2. Then we are trying to declare `num` again with `const`, which is not allowed.

---

## References:


1. *JS: Explain "hoisting"*: http://lucybain.com/blog/2014/hoisting/
2.

# Closure

## What is closure?

```
function outer() {
    const val = 1;        // val is inside inner()'s closure
    return function inner() {
        console.log(val);
    }
}
```

Closure is created when the following happens:

1. A function `inner()` is created.
2. Inside `inner()`, some variable references to something **out of** `inner()`. Like `val`;
3. The referenced outer variable `val` becomes a closure variable and will always exist until no more referenced.

The funciton `inner()` is called closure.

## 1. Most Classical Question

A common interview question starts like this:

```
const array = [];
for (var i=0; i < 10; i++) {
    array[i] = function() {
        console.log(i);
    }
}
a[5]();          // ?
```

The answer is `10`.

Because the function `a[5]` references a variable `i` which is outside. And the `i` will be always there. Since all the `i` referenced by a[0], a[1], a[2]... are the same **outer one**, all of them will be the same value: 10 (what the i became at the end).

```
const arr = [];
for (var i=0; i < 10; i++) {
    arr[i] = function() {
        i++;
        console.log(i);
    }
}
arr[5]();        // ?
arr[6]();        // ?
```

Yes, `11` and `12` will be logged. This proves same `i` was **shared** and **modified**.

## 2. Follow up: How to fix

Then follow up will be: how do you fix it?

```
// Way 1: Use function:
const arr = [];
for (var i=0; i < 10; i++) {
    arr[i] = (function(val) {    // line-4
        return function() {
            console.log(val);
        }
    })(i);                       // pass i as innner function's
val
}
arr[5]();         // 5
arr[8]();         // 8
```

As i is a primitive variable. We pass it by value. Means we are creating a copy and pass it into `line-4`'s function in every loop. No i is `shared` now.

```
// Way 2: Use let
const arr = [];
for (let i=0; i < 10; i++) {
    arr[i] = function() {
        console.log(i);
    }
}
arr[5]();        // 5
arr[8]();        // 8
```

This is one of the situation `let` was created for. For each loop, a new `i` was created. And none of them is shared.

## 3. What will be logged?

```
// Question 1
(function(x) {
    return (function(y) {
        console.log(x);        // what will be logged?
    })(2);
})(1);

// Question 2
(function(y){
    return (function(y){
        console.log(y);        // what will be logged?
    })(2);
})(1);
```

## 4. When do you need it? [none coding]

http://javascriptissexy.com/oop-in-javascript-what-you-need-to-know/ Create private variables:

```
// private in the regular functions

// private in constructors
function Student(id = 0, age = 20) {
    var studentId = id;
    this.age = age;
}
const stu1 = new Student(10192, 20);
stu1.studentId;        // undefined
```

## 5. What are the good and bad part of closure? [none coding]

```
//
```

---

## References:

1. 你应该知道的25道 Javascript 面试题

# Event Loop Questions

To help understand Javascript's Event Loop System. I recommend you to read this book: *Secrets of Javascript Ninja* https://www.manning.com/books/secrets-of-the-javascript-ninja

## 1. `setTimeout()`

```
function test() {
    console.log(1);
    setTimeout(function() {
        console.log(2);
    }, 1000);
    setTimeout(function() {
        console.log(3);
    }, 0);
    console.log(4);
}
test();
```

Answer is: `1, 4, 3, 2`. Whatever inside setTimeout will be asigned into the end of event loop. So after `1` was logged, `4` will be the next. Then event loop moves to the `0` and `3`.

```
function test() {
    for (let i = 0; i < 5; i++) {    // we used let in here
        setTimeout(function() {
            console.log(i);
        }, 1000);
    }
}
test();
```

As we know from chapter closure, a new i was created each loop. Thus the answer is `0, 1, 2, 3, 4`. This is exactly what we expect from event loop.

## 2. `setInterval()`

What's the difference between the `setTimeout` and `setInternal`?

```
// Taobao Interview Question
// Author: oklai@zhihu
setTimeout(function(){
    /* ... */
    setTimeout(arguments.callee, 10);
}, 10);
setInterval(function(){
    /*... */
}, 10);
```

---

## References:


1. *5 More JavaScript Interview Exercises* https://www.sitepoint.com/5-javascript-interview-exercises/
2. 有哪些经典的 Web 前端或者 JavaScript 面试笔试题？https://www.zhihu.com/question/19841848

## Keys are strings

What is the output out of the following code? Explain your answer.

```
vara={},
b={key:'b'},
c={key:'c'};
a[b]=123;
a[c]=456;
console.log(a[b]);
```

一道有趣的题目，答案是 456。 我们知道，Javascript 中对象的 key 值，一定会是一个 string 值，如果不是，则会隐式地进行转换。当执行到 a[b]=123] 时，b 并不是一个 string 值，将 b 执行 toString() 方法转换（得到 “[object Object]“），a[c] 也是相同道理。所以代码其实可以看做这样执行：

```
vara={},
b={key:'b'},
c={key:'c'};
// a[b]=123;
a["[object Object]"]=123;
// a[c]=456;
a["[object Object]"]=456;
console.log(a["[object Object]"]);
这样就一目了然了。
```

## delete

JavaScript also has a delete operator that "undefines" an object a property (thanks zproxy), it can be handy in certain situations, you can apply it to object properties and array members, variables declared with var cannot be deleted, but implicitly declared variables can be:

```
var obj = {
    val: 'Some string'
};
alert(obj.val); // displays 'Some string'
delete obj.val;
alert(obj.val); // displays 'undefined'
```

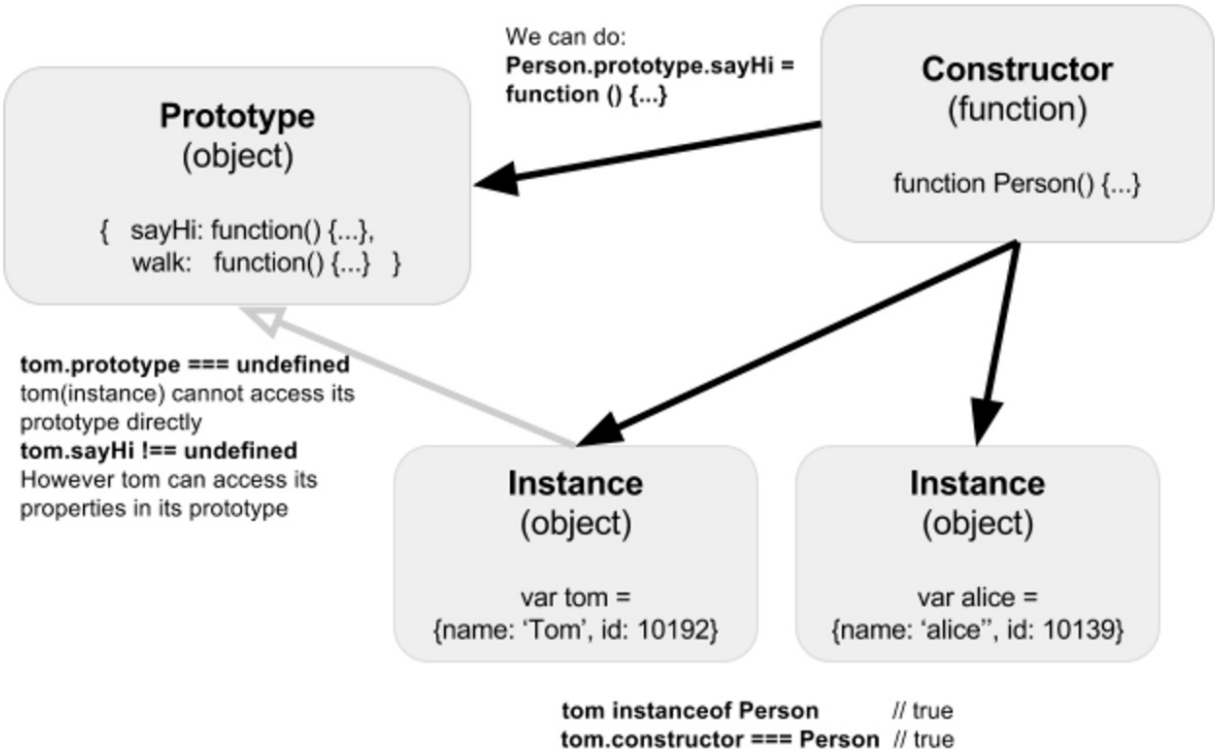
---

## References:

1. *9 JavaScript Tips You May Not Know* http://codetunnel.io/9-javascript-tips-you-may-not-know/

# Prototype

## What is prototype and what is constructor?




Ref: http://tobyho.com/2010/11/22/javascript-constructors-and/

## Some prototype and inheritance coding:

Write a `Person` class that we can specify `name` :

```
function Person (name) {
    this.name = name;
}
```

Ok, now add a function to all the people instances so that they can tell you who they are by saying `hi, I am <name>` .

```
Person.prototype.sayHi = function () {
    console.log(`hi, I am ${this.name}.`);
}
// Test it
const alice = new Person('Alice');
const tom = new Person('Tom');

alice.sayHi();        // 'hi, I am Alice.'
tom.sayHi();          // 'hi, I am Tom.'
```

Next, inheritance part. Create another class called `Student` class. that takes a `name` and `id` as argument, inherit from `Person` class (means using `Student` constructor).

```
// constructor part
function Student (name, id) {
    Person.call(this, name);
    this.id = id;
}
// prototype part
Student.prototype = Object.create(Person.prototype);
Student.prototype.constructor = Student;

// Replace the "sayHello" method
Student.prototype.sayHi = function(){
    console.log("Hi, I'm " + this.firstName + ". My id is " + th
is.id + ".");
};
// Example usage:
var student1 = new Student("Tom", "10192");
student1.sayHi();
// "Hello, I'm Janet. My id is 10192."

// Check that instanceof works correctly
console.log(student1 instanceof Person); // true
console.log(student1 instanceof Student); // true

// More notes please check the ref link bellow
```

Source: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Introduction_to_Object-Oriented_JavaScript

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects

# Array (Queue, Stack, Tree)

## 1. `forEach()` and `map()`

What's the difference between `forEach()` and `map()` ?

1. Return value
2. ?
3. ?

What does

```
const arr = [0, 0, 0, 0, 1, 0, 0, 0];
arr.forEach(function(val, index) {
    if (val === 1) break;
    else arr[index] = 3;
})
console.log(arr);        // ?
```

Answer is `Uncaught SyntaxError: Illegal break statement` . There's no built-in ability to `break` in `forEach` .

## 2. `every()` and `some()`

What's the difference between `every()` and `some()` ?

## 3. `slice()` and `splice()`

**There's no `splice()` for strings.**

```
Array.prototype.slice;            // exist
String.prototype.slice;           // exist

Array.prototype.splice;           // exist
String.prototype.splice;          // undefined, use subString() o
r slice()
```

**Use splice to insert items into array:**

```
//
```

## 3. Implement Stack using Array

How to use array as stack?

```
const stack = [];     // stack: []
stack.pop()           // undefined
stack.push(1)         // stack: [1]
stack.push(3)         // stack: [1, 3]
stack.pop()           // stack: [1], 3 is returned
stack[stack.length-1];     // peak()
```

## 4. Implement Queue using Array

How to use array as queue?

```
const queue = [];     // queue: []
queue.push(2);        // queue: [2]
queue.push(4);        // queue: [2, 4]
const i = queue.shift()     // queue: [4], 2 is returned
```

## 5. Sort Array

What will the following return?

```
[1,2,11,3].sort();
```

Answer is: `[1, 11, 2, 3]` .

A little surprising right? Reason is `sort(comparator)` 's default comparator treats array items as `string` .

Fix:

```
[1, 2, 11, 3].sort((a, b) => a - b);
// [1, 2, 3, 11]
```

More about `Array.prototype.sort()` : https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/sort

## References:


1. *9 JavaScript Tips You May Not Know* http://codetunnel.io/9-javascript-tips-you-may-not-know/
2. *Interviewing a front-end developer* http://blog.sourcing.io/interview-questions?utm_source=ourjs.com
3. *How to short circuit Array.forEach like calling break* http://stackoverflow.com/questions/2641347/how-to-short-circuit-array-foreach-like-calling-break

# String

## 1. Remove Spaces

1.1 How do you change " abc " to "abc"? (remove beginning and ending spaces)

```
" abc  ".trim();    // "abc"
```

1.2 How do you change " a b c " to "abc"? (remove all spaces)

```
" a b c ".replace(' ', '');
```

1.3 How do you change "a  b c" to "a b c" ? (Single space to multi-space)

```
// failed: replace doesnt work recursively
" a    b c".replace(' ', '').split('').join(' ');

// Using Regex
" a    b c".replace( /\s+/g, ' ' );
// Ref: http://stackoverflow.com/questions/1981349/regex-to-repl
ace-multiple-spaces-with-a-single-space
```

## 2. Implement `repeatify()`

Implement `repeatify()` :

```
String.prototype.repeatify = String.prototype.repeatify ||
function(times) {
    /* write your own here */
}
// how it works
'Aoo'.repeatify(3); // 'AooAooAoo'
```

One of the answers could be:

```
String.prototype.repeatify = String.prototype.repeatify || funct
ion(times) {
    // avoid concat for better performance:
    // like: str = ''; str += this;
    const str = [];
    for (var i = 0; i < times; i++) {
        str.push(this);
    }
    return str.join('');
};
```

Another faster approach using binary search

```
//
String.prototype.repeat = function (times) {
    if(times === 1) return this;
    var halfString = String.prototype.repeat.call(this, Math.flo
or(times/2));
    return half + half + (times & 1 ? this : '');
}
```

A tricky way:

```
// Ref: http://stackoverflow.com/questions/202605/repeat-string-
javascript
String.prototype.repeat = function(num) {
    return new Array(num + 1).join(this);
}
```

## 3. Add space to String

Write a function to add a space to string.

```
String.prototype.addSpace = function() {
 /* your code here */
}
"abc".addSpace(); // "a b c"
```

One of the solution is using `Array.prototype.split()` and `String.prototype.join()` :

```
String.prototype.addSpace = function() {
 return this.split('').join(' ');
}
```

## 4. Check Palindrome

How to check if a string is palindrome?

```
function isPalindrome(str) {
    return (str == str.trim().toLowerCase().split('').reverse().
join(''));
}
```

## 5. All .toString()

Difference between all the `toString()` , like:

```
const arr = [1,2,3],
      obj = {},
      num = 1,
      nul = null,
      undef;

// for arrays
// Array.prototype.toString is more useful
Array.prototype.toString.call(arr);    // '1,2,3'
Object.prototype.toString.call(arr);   // '[object Array]'
Number.prototype.toString.call(arr);   // error

// Array.prototype.toString works with objects
Array.prototype.toString.call(obj);    // '[object Object]'
Object.prototype.toString.call(obj);   // '[object Object]'
Number.prototype.toString.call(obj);   // error

// Number.prototype.toString seems only work for numbers
Array.prototype.toString.call(num);    // '[object Number]'
Object.prototype.toString.call(num);   // '[object Number]'
Number.prototype.toString.call(num);   // '12'

// Array.prototype.toString seems only work with array/object
Array.prototype.toString.call(nul);    // error
Object.prototype.toString.call(nul);   // '[object Null]'
Number.prototype.toString.call(nul);   // error

// Object.prototype.toString works with anything
Array.prototype.toString.call(undef);  // error
Object.prototype.toString.call(undef); // '[object Undefined]'
Number.prototype.toString.call(undef); // error
```

In Progress

In Progress

## Promise Basics

Basic promise coding: https://davidwalsh.name/promises

## Promise.all()

## Promise Wrapper

Write a function that returns a promise and takes `link as params`

```
function get(url) {
    /* Your code here */
}

/* Usage */

// get zoo #123 's info
get('/zoo/123').then(data => displayZoo);

// takes multiple links and do something when all get returns
get(['/zoo/1', '/zoo/2', 'zoo/3']).then(data => compareZoos);

// internal code you dont need to worry about
function compareZoos() { /* ... */ }
```

## Traffic Lights

By using promise, write a function to simulate a dummy traffic light system:

1. Red light for 5s
2. Yellow light for 2s
3. Green light for 4s

```
// red() yellow(), green() to use
function red() { console.log('red'); }
function yellow() { console.log('yellow'); }
function green() { console.log('green'); }

// your code here:
// Use Promise.resolve() etc
```

## References

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects

# All Others

In a lot of interviews, you will be asked to implement some features.

## 1. Math.random()

What is it for and how to use it?

`Math.random` returns a number `[0,1)`

```
// Generate a number between 1-100
// [1,100)
```

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Math/random

## 1. Implement `bind()`

Implement your own `bind()` :

```
String.prototype.bind = String.prototype.bind ||
function(context) {
    // write your own here
};
const obj = {
    val: 100
};
const test = function() {
    console.log(this.val);
};
const test2 = test.bind(obj);
test2();        // 100
```

One of the answers could be:

```
String.prototype.bind = String.prototype.bind ||
function(context) {
    const _func = this;
    return function() {
        _func.apply(context, arguments);
    };
}
```

# Number

## 1. Round() and toFixed()

1.1 How to round a number to integer?

```
// http://www.w3schools.com/jsref/jsref_round.asp
// The round() method rounds a number to the nearest integer.
// Note: 2.49 will be rounded down, 2.5 will be rounded up.

Math.round(10.4);       // 10
Math.round(12.5);       // 13
```

1.2 How to round a decimal

```
// http://www.w3schools.com/jsref/jsref_tofixed.asp
// The toFixed() method converts a number into a string, keeping
 a specified number of decimals.

// Note:

// Since javascript is using float to represent a int,
// sometimes the result is not predictible, like:

(1.5).toFixed()          // 2
(1.05).toFixed(1)        // 1.1        rounded up
(1.005).toFixed(2)       // 1.00       rounded down
(1.04).toFixed(1);       // 1.0        works as expected
```

## 2. Number.prototype.toString()

```
(100).toString();        // '100'
(8).toString(2);         // '1000'
(16).toString(16);       // '10'
```

More info please check chapter - String.

# HTML and Browser Event

## 1. What is sementric html? Why do some people think it's useful?

https://www.wikiwand.com/en/Semantic_HTML

## 2. What is event bubbling and capturing?

Ref: http://www.quirksmode.org/js/events_order.html

## 3 What event doesn't bubble?

Any events specific to one element do not bubble: submit, focus, blur, load, unload, change, reset, scroll, most of the DOM events (DOMFocusIn, DOMFocusOut, DOMNodeRemoved, etc), mouseenter, mouseleave, etc

Ref: http://stackoverflow.com/questions/5574207/html-dom-which-events-do-not-bubble

## 4. How to cancel a `<a>` event using coding?

```
event.preventDefault();
```

Ref: http://stackoverflow.com/questions/10276133/how-to-disable-html-links

# 1. Difference between Block & Inline?

```
 1. inline(only accepts left/right padding/margin, no top/bottom
 will take effect), block accepts all
 2. inline accepts vertical-align, blocks doesn't
 3. block's default width is 100%
 4. inline doesn't accept width/height, block accepts all
 5. when a inline element is floated, it accepts width and heigh
t
```

Credit: http://www.impressivewebs.com/difference-block-inline-css/

# 2. Different Ways to Hide an Element?

```
// More common
display: none;
visibility: hidden;
opacity: 0;

// Tricky
position: absolute;
z-index: 1000;
margin: -100px; ???
left:-9999px;
position:relative ???
left:-9999px;
position:absolute; ???
transform: translateX(-9999px);
```

Credit: https://kitt.hodsden.org/blog/2013/07/5_3_ways_hide_element_css

**Follow-up: What's the difference between `visibility: none` and `opacity: 0`?**

Click event. Element with `visibility: none` **doesn't** respond to click-event. But `opacity: 0` element does.

Source: Personal experience

## 3. Selectors

What are those selectors??

```
s
```

Source: https://code.tutsplus.com/tutorials/the-30-css-selectors-you-must-memorize--net-16048

## 4. Centering

for centering, the ultimate way is to use flex. However, to support some old browsers, we need some classical approaches.

#### 1. Vertically align elements?

1. for inline elements

   ```
   vertical-align: center;
   ```

2. for block elements:
   i. use flex if you can

   ```
   .parent   { display: flex }
   .children { align-self: center; }
   /* or */
   .parent   {
        display: flex;
        flex-direction: column;
        justify-content: center;
   }
   ```

   ii. use negative margin if know the height

```
.parent { position: relative; }    /* to work with childr
en's absolute position */
.children {
     position: absolute;
     height: 200px;
     margin-top: -100px;
     top: 50%;
}
```

iii. use transform is dont know the height

```
.parent { position: relative; }
.children {
     position: relative;
     top: 50%;
     transform: translateY(-50%);
}
```

**2. Horizontally align Elements?**

1. for inline elements:

```
.item-parent {
     text-align: center;
}
```

2. for block-level elements
   i. use flex, supports multiple items as well:

```
.parent {
     display: flex;
     justify-content: center;
}
```

   ii. otherwise use margin: 0

```
.children { margin: 0 auto; }
```

**3. Follow-up: How about both vertical and horizontal align?**

1. Use flex:

```
.parent {
 display: flex;
 justify-content: center;    /* this will center all items o
n the main axis */
}
.children {
 align-self: center;
}
```

2. Old fashion ways, the trick of 0

```
/* https://www.smashingmagazine.com/2013/08/absolute-horizon
tal-vertical-centering-css/ */
.children {
     margin: auto;
     position: absolute;
     top: 0; left: 0; bottom: 0; right: 0;
}
```

3. use translate

```
.parent { position: relative; }
.children {
     position: absolute;
}
```

Source: https://css-tricks.com/centering-css-complete-guide/

More about vertical using flex: https://philipwalton.github.io/solved-by-flexbox/demos/vertical-centering/

## 5. What is responsive design? How to use it?

Media Query:

https://developer.mozilla.org/en-US/docs/Web/CSS/Media_Queries/Using_media_queries

## 6. Text Space

**How to display space in HTML text?**

` ` Note: there is a ; at the bottom

**How to display the text ` ` ?**

```
// Text we want to display exactly:
...
The text 'nbsp;' means 'space'
...
```

Answer:

```
The text 'a&amp;nbsp;' means 'space'    // use &amp; to escape ;
```

Source: Presonal experience

---

## Reference:


1. *How to write out HTML entity name ( , <, >, etc)?* http://stackoverflow.com/questions/17427713/how-to-write-out-html-entity-name-nbsp-lt-gt-etc
2. *Does opacity:0 have exactly the same effect as visibility:hidden?* http://stackoverflow.com/questions/272360/does-opacity0-have-exactly-the-same-effect-as-visibilityhidden?answertab=votes#tab-top
3.

## 1. Sticky Footer

Please read `sticky footer: five ways`:

https://css-tricks.com/couple-takes-sticky-footer/

## 2. Centering a unknown img vertically and horizontally?

Please check the `centering - both horizontally and vertically` section in previous chapter.

## 3. 50 100 150

## 33.333

# DOM

A lof of interviewer like to ask you how to manipulate DOM nodes. Sometimes using 3rd party lib such as jQuery is allowed, sometimes it is not.

This chapter focuses on native DOM APIs.

## 1. Basic manipulation to make sure you are familar with DOM API

1.1 Create add some text into the `<h1 id='target'></h1>`

```
//
```

1.2 Create a span with text content `hello` and append it to the `<div id='target'></div>`

```
//
```

1.3 As shown bellow, a `<ul>` has multiple `<li>`, add event to all of the `<li>` so that when it's clicked, alerts the index of it.

```
//
```

1.4 Count number of a given node's children

```
//
```

Ref: http://harttle.com/2015/10/01/javascript-dom-api.html

You dont need jQuery: http://ourjs.com/detail/573a9cec88feaf2d031d24fc

## Create a function...

That, given a DOM Element on the page, will visit the element itself and all of its descendents (not just its immediate children). For each element visited, the function should pass that element to a provided callback function. The arguments to the function should be:

```
a DOM element
a callback function (that takes a DOM element as its argument)
```

DFS:

```
function Traverse(p_element,p_callback) {
    p_callback(p_element);
    var list = p_element.children;
    for (var i = 0; i < list.length; i++) {
        Traverse(list[i],p_callback); // recursive call
    }
}
```

## Create a function...

Give all elements inside html DOM a different color

```
// Need verify
[].forEach.call($('*'), function(ele) {
    ele.style.outline = "1px solid #" + (~~(Math.random() * (1 <
< 24))).toString(16);
});
```

## `Node Object` vs `Element Object` ?

http://stackoverflow.com/questions/9979172/difference-between-node-object-and-element-object

## `children` vs `childNodes` ?

http://stackoverflow.com/questions/7935689/what-is-the-difference-between-children-and-childnodes-in-javascript

## `textContent` vs `createTextNode()`?

They are the same except: for security perspective, `createTextNode()` will escape string and show them as they are.
http://stackoverflow.com/questions/31643204/textnode-or-textcontent

---

## References:


1. 你应该知道的25道 *Javascript* 面试题 http://web.jobbole.com/84723/
2. *You Don't Need jQuery* http://ourjs.com/detail/573a9cec88feaf2d031d24fc

// array flatten

```
var a = [1,2,[3],[4,[5,6]],7,8];
// way 1
function flattern(a) {
    return a.reduce((result,i)=>{
        return result.concat(Array.isArray(i)? flattern(i) : i);
    }, []);
}
// way 2
function flattern(a) {
    var stack = [], result = [];
    stack = stack.concat(a);
    while (stack.length) {
        let i = stack.pop();
        if (Array.isArray(i)) stack = stack.concat(i);
        else result.unshift(i);
    }
    return result;
}
flattern(a);
```

# System Design

Some companies might ask you system design questions (related or) unrelated to front-end developing.

## 1. Design a Restful API for a email web app

## 2. Design a Restful API for a email web app

## 3. Design a Restful API for a email web app

# Random

Random questions related to web dev.

## 1. What is same-origin policy?

**For security purpose, two website which are NOT from the same origin cannot share:**

1. Cookies, localStorage, IndexDB
2. DOM
3. Ajax

Same-origin must have the same:

1. domain name (google.com vs apple.com)
2. protocol (http vs https)
3. port number (4080 vs 80)

PS: A page may change its own origin with some limitations. A script can set the value of document.domain to its current domain or a superdomain of its current domain. If it sets it to a superdomain of its current domain, the shorter domain is used for subsequent origin checks.

For example, assume a script in the document at http://store.company.com/dir/other.html executes the following statement: Like:

```
document.domain = "company.com";
```

More info: Ref: https://developer.mozilla.org/en-US/docs/Web/Security/Same-origin_policy

## 2. How to get around of same-origin policy?

**We can use multiple approaches to get around same-origin policy, such as:**

1. JSONP (Only support `GET`, more info:

http://stackoverflow.com/questions/3839966/can-anyone-explain-what-jsonp-is-in-layman-terms, and (Chinese): http://www.cnblogs.com/dowinning/archive/2012/04/19/json-jsonp-jquery.html)
2. Server delegation (server does the request to different origin)
3. WebSocket (WebSocket have no same-origin policy)
4. CORS (Cross-Origin Resource Sharing)

## What will happen when you type in a URL and hit `enter` ?

http://stackoverflow.com/questions/2092527/what-happens-when-you-type-in-a-url-in-browser

## Get vs Post?

http://stackoverflow.com/questions/3477333/what-is-the-difference-between-post-and-get

## Put vs Post in Restful API?

http://stackoverflow.com/questions/630453/put-vs-post-in-rest

## Enable a element to be tab-able?

Use tabIndex. That are:

1. some elements are born to be tab-able. Such as `<a>` , `<input>` etc.
2. The sequence of tab-able elements when pressed tab is how they appear in `.html` file.
3. to enable an element to be tab-able, add `tabIndex='0'` to it.
4. to change tab-able elements' sequence. change tabIndex to `1,2,3...`
5. to disable tab-able, use `tabIndex = '-1'`

ref: https://www.paciellogroup.com/blog/2014/08/using-the-tabindex-attribute/

## Removing The Dotted Outline

https://css-tricks.com/removing-the-dotted-outline/

# 1. Get Integer Part

Firstly, the classic ways:

```
// Math.floor() works for positive only
Math.floor(-1.2);     // -1

// Math.ceil() works for negative only
Math.ceil(1.3);       //  2

// Math.round() works fine
Math.round(1.2);      //  1
Math.round(-1.2);     // -1

// ParseInt()
parseInt(1.2, 10);    // 1
parseInt(-1.2, 10);   //-1
```

Then, tricks probably won't help you pass the code review:

```
~~1.1                 //  1
~~1.8                 //  1
~~(-1.1)              // -1
~~(-1.8)              // -1
~~-1.1                // -1
const three = ~~(10/3);

// another trick
console.log(1.2 | 0);   //  2
console.log(-1.2 | 0);  // -1

// another way using bit
1.2>>0                // 1
-1.2>>0               // -2
```

The reason bit works is that internally, bit operation is like:

1. convert the number from double-float to int
2. then do bit operations, |(or), ~(revert) etc
3. then convert back to float

## 2. Get Boolean

```
// using !!
!!5         // true
!!true      // true
!!''        // false
!!null      // false
![]         // true
```

## 3. Odd, Even

```
function isOdd(n) {
    return !!(n & 1);
}
isOdd(12);              // false
isOdd(1);               // true

// Pay attention that following returns 0
12 & 1 === 0            // 0, instead of true
(12 & 1) === 0          // true
```

## 4.How to detect a variable is declared or not?

We know that if we reference an none-exists key inside a object, it returns `undefined` .

```
const obj = {};
obj.noneExist;    // undefined
```

And we also know that if we are trying to access an variable that is not defined, it returns an error:

```
let newValue = oldValNotExist;
// Uncaught ReferenceError: oldValNotExist is not defined...
```

So, question is, how to tell if `oldValNotExist` exist or not without throwing any error?

```
// use typeof
if (typeof oldValNotExist !== 'undefined') { ... }
// no error will be thrown

// btw, dont forget this:
typeof null === 'object'        // true
```

Ref.: http://bonsaiden.github.io/JavaScript-Garden/#function.arguments

---

## References:


1. 装逼指南
2. useless es5: https://rainsoft.io/make-your-javascript-code-shide-knockout-old-es5-hack/