1. null vs undefined

Question: What are the differences between `null` and `undefined`?

Answer: JavaScript has two distinct values for nothing, null and undefined.

undefined

`undefined` means, value of the variable is not defined. JavaScript has a global variable `undefined` whose value is "undefined" and `typeof undefined` is also "undefined". Remember, undefined is not a constant or a keyword. undefined is a type with exactly one value: undefined. Assigning a new value to it does not change the value of the type undefined.

8 Ways to get Undefined:

- A declared variable without assigning any value to it.
- Implicit returns of functions due to missing return statements.
- return statements that do not explicitly return anything.
- Lookups of non-existent properties in an object.
- Function parameters that have not passed.
- Anything that has been set to the value of undefined.
- Any expression in the form of void(expression)
- The value of the global variable `undefined`

null

`null` means empty or non-existent value which is used by programmers to indicate “no value”. null is a primitive value and you can assign null to any variable. null is not an object, it is a primitive value. For example, you cannot add properties to it. Sometimes people wrongly assume that it is an object, because typeof null returns "object".

Btw, `null == undefined` ref: history of typeof null

2. == Vs ===

Question: What are the differences between `==` and `===`?

Answer: The simplest way of saying that, == will not check types and === will check whether both sides are of same type. So, == is tolerant. But under the hood it converts to its convenient type to have both in same type and then do the comparison.

=== compares the types and values. Hence, if both sides are not same type, answer is always false. For example, if you are comparing two strings, they must have identical character sets. For other primitives (number, boolean) must share the same value.

Rule for implicit coercion: Comparison by using == does implicit type conversion under the hood. And rules for implicit coercion are as follows-

- If both operands are same type use ===
- undefined == null
- If one operands is string another is number, convert string to number

- If one is boolean and another is non-boolean, convert boolean to number and then perform comparison
- While comparing a string or number to an object, try to convert the object to a primitive type and then try to compare

Be careful while comparing objects, identifiers must reference the same objects or same array.

```
var a = {a: 1};

var b = {a: 1};

a == b //false

a === b //false


var c = a;

a == c//true

a === c //true
```

Special note: NaN, null and undefined will never === another type. NaN does not even === itself.

Homework: Read confusing special cases of NaN

3. Object Equality

Question: How would you compare two objects in JavaScript?

Basics: JavaScript has two different approaches for testing equality. Primitives like strings and numbers are compared by their value, while objects like arrays, dates, and user defined objects are compared by their reference. This means it compares whether two objects are referring to the same location in memory.

Answer: Equality check will check whether two objects have same value for same property. To check that, you can get the keys for both the objects. If the number of properties doesn't match, these two objects are not equal. Secondly, you will check each property whether they have the same value. If all the properties have same value, they are equal.

```
function isEqual(a, b) {

    var aProps = Object.getOwnPropertyNames(a),

        bProps = Object.getOwnPropertyNames(b);


    if (aProps.length != bProps.length) {

        return false;
        }



        for (var i = 0; i < aProps.length; i++) {

             var propName = aProps[i];



             if (a[propName] !== b[propName]) {

                    return false;

             }

        }

        return true;

    }
```

Limitation:

- If one of the property values is itself an object
- If one of the property values is NaN (the only value in JavaScript that is not equal to itself?)
- If one object has a property with value undefined, while another object doesn't have the property (which thus evaluates as undefined). Btw, you can solve this problem by using hasOwnProperty

ref: object equality in JS, Underscore.js isEqual function

4. True Lies

Question: 11+ true false related questions that will trick you.

falsy: In javascript 6 things are falsy and they are- false, null, undefined, '', 0, NaN

truthy: There are only two truthy things- true and everything that is not false

True False Rapid Fire

Question: Is `'false'` is false?

Answer: No. Because, it's a string with length greater than 0. Only empty string is false.

Question: Is `' '` is false?

Answer: No. Because, it's not an empty string. There is a white space in it.

Question: What about `{}`?

Answer: true. It's an object. An object without any property is an object can't be falsy.

Question: Tell me about `[]`?

Answer: This is also truthy. It's an array object (array is child of object) is truthy.

Question: You talked bout `''` to be falsy. What about `new String('')`?

Answer: Though you are passing empty string to the string constructor, it is creating an String object. More precisely a instance of String object. It becomes an object. Hence, it is not false. so, it is truthy.

Question: Tell me about `new Boolean(false)`

Answer: truthy. As it creates an instance of the Boolean object which is an object. Object is truthy.

Question: `Boolean(function(){})`

Answer: `true` if you pass a truthy value to Boolean, it will be true.

Question: `Boolean(/foo/)`

Answer: `true`

Question: `true%1`

Answer: 0. When you are trying to find reminder of true, true becomes 1 and reminder of 1 while dividing by 1 is 0. you will get same result if you doe `false%1`

Question: `''%1`

Answer: 0

5. Truthy isn't Equal to true

Question: As `[]` is true, `[]==true` should also be true. right?

Answer: You are right about first part, `[]`, empty array is an object and object is always truthy. Hence, if you use `if([]){console.log('its true')}` you will see the log.

However, special case about `==` (double equal) is that it will do some implicit coercion.

Since left and right side of the equality are two different types, JavaScript can't compare them directly . Hence, under the hood, JavaScript will convert them to compare. first right side of the equality will be cooereced to a number and number of `true` would be 1.

After that, JavaScript implementation will try to convert `[]` by using`toPrimitive` (of JavaScript implementation). since `[].valueOf` is not primitive will use `toString` and will get `""`

Now you are comparing "" == 1 and still left and right is not same type. Hence left side will be converted again to a number and empty string will be 0.

Finally, they are of same type, you are comparing `0 === 1` which will be false.

ref: angus croll: truth and eqality in JS, ref: truthy and falsy

6. Extend Core Object

Question: How could you write a method on instance of a date which will give you next day?

Answer: I have to declare a method on the prototype of Date object. To get access to the current value of the instance of the date, i will use `this`

```
Date.prototype.nextDay = function(){
      var currentDate = this.getDate();

       return new Date(this.setDate(currentDate +1));

    }



    var date = new Date();

     date; //Fri May 16 2014 20:47:14 GMT-0500 (Central Daylight Time)

    date.nextDay();//Sat May 17 2014 20:47:14 GMT-0500 (Central Daylight Time)
```

7. bind

Question: If you want to use an arbitrary object as value of `this`, how will you do that?

Answer: There are at least three different ways to doing this by using bind, call and apply. For example, I have a method named deductMontlyFee in the object monica and by default value of this would be monica inside the method.

```
    var monica = {

       name: 'Monica Geller',

      total: 400,

       deductMontlyFee: function(fee){

           this.total = this.total - fee;

           return this.name + ' remaining balance is '+ this.total;

      }

    }
```

If I bind the deductMontlyFee of monica with another object `rachel` and pass rachel as first parameter of the bind function, rachel would be the value of `this`.

```
    var rachel = {name: 'Rachel Green', total: 1500};

    var rachelFeeDeductor = monica.deductMonthlyFee.bind(rachel, 200);
```

```
rachelFeeDeductor(); //"Rachel Green remaining balance is 1300"

rachelFeeDeductor(); //"Rachel Green remaining balance is 1100"
```

`bind` allows you to borrow a method and set the value of `this` without calling the function. It simply returns an exact copy of the function with new value of `this`. You can reuse the same function with new value of this without harming the old one.

```
var ross = {name:'Ross Geller', total:250};

var rossFeeDeductor = monica.deductMonthlyFee.bind(ross, 25);

rossFeeDeductor(); //"Ross Geller remaining balance is 225"

rossFeeDeductor(); //"Ross Geller remaining balance is 200"


rachelFeeDeductor(); //"Rachel Green remaining balance is 900"
```

Question: If an older browser dont have bind function, how will you shim it

Answer: Look at the code below and use your brain.

```
Function.prototype.bind = Function.prototype.bind || function(context){

    var self = this;

    return function(){

        return self.apply(context, arguments);

    };

}
```

8. arguments and call

Question: Write a simple function to tell whether 2 is passed as parameter or not?

Basics: arguments is a local variable, available inside all functions that provides a collection of all the arguments passed to the function. `arguments` is not an array rather an array like object. It has length but doesn't have the methods like forEach, indexOf, etc.

Answer: First convert arguments to an array by calling slice method on an array and pass arguments. After that simply use indexOf.

```
function isTwoPassed(){

   var args = Array.prototype.slice.call(arguments);

    return args.indexOf(2) != -1;

}



isTwoPassed(1,4) //false

 isTowPassed(5,3,1,2) //true
```

Danger: Don't name any argument as "arguments" or dont create any local variable named as "arguments", this will override build in arguments object.

9. apply

Question: How could you use Math.max to find the max value in an array?

Answer: Use apply on Math.max and pass the array as apply takes an array of arguments. Since we are not reading anything from this or using it at all. We can simply pass null as first parameter.

```
function getMax(arr){

   return Math.max.apply(null, arr);

}
```

Extra: call and apply, both takes the value of this as first parameter. However, call takes a collection of arguments after first parameter whereas apply use an array of arguments as second parameter.

Tip: If you have weaker memory like me, you can remember apply starts with "a" and array starts with "a"

10. this

Question: What the heck is `this` in JavaScript?

Answer: At the time of execution of every function, JavaScript engine sets a property to the function called `this` which refer to the current execution context. `this` is always refer to an object and depends on how function is called. There are 7 different cases where the value of `this` varies.

1. In the global context or inside a function this refers to the window object.

2. Inside IIFE (immediate invoking function) if you use "use strict", value of this is undefined. To pass access window inside IIFE with "use strict", you have to pass this.
3. While executing a function in the context of an object, the object becomes the value of this
4. Inside a setTimeout function, the value of this is the window object.
5. If you use a constructor (by using new keyword) to create an object, the value of this will refer to the newly created object.
6. You can set the value of this to any arbitrary object by passing the object as the first parameter of bind, call or apply
7. For dom event handler, value of this would be the element that fired the event

ref: Understand JavaScript this in a crystal clear way

11. Rapid Fire

Question: What is `typeof []`

Answer: Object. Actually Array is derived from Object. If you want to check array use `Array.isArray(arr)`

Question: What is `typeof arguments`

Answer: Object. arguments are array like but not array. it has length, can access by index but can't push pop, etc.

Question: What is `2+true`

Answer: 3. The plus operator between a number and a boolean or two boolean will convert boolean to number. Hence, true converts to 1 and you get result of 2+1

Question: What is `'6'+9`

Answer: 69. If one of the operands of the plus (+) operator is string it will convert other number or boolean to string and perform a concatenation. For the same reason, `"2"+true` will return "2true"

Question: What is the value of `4+3+2+"1"`

Answer: 91 . The addition starts from the left, 4+3 results 7 and 7+2 is 9. So far, the plus operator is performing addition as both the operands are number. After that 9 + "1" where one of the operands is string and plus operator will perform concatenation.

Question: What is the value of `"1"+2+4`

Answer: "124". For this one "1" + 2 will produce "12" and "12"+4 will generates "124".

Question: What is the value of `-'34'+10`

Answer: -24. minus(-) in front of a string is an unary operator that will convert the string to a number and will make it negative. Hence, -'34' becomes, -34 and then plus (+) will perform simple addition as both the operands are number.

Question: What is the value of `+'dude'`

Answer: NaN. The plus (+) operator in front of a string is an unary operator that will try to convert the string to number. Here, JavaScript will fail to convert the "dude" to a number and will produce NaN.

Question: If you have `var y = 1, x = y = typeof x;` What is the value of x?

Answer: "undefined"

Question: for `var a = (2, 3, 5);` what is the value of a?

Answer: 5. The comma operator evaluates each of its operands (from left to right) and returns the value of the last operand. ref: MDN

Question: for `var a = (1, 5 - 1) * 2` what is the value of a?

Answer: 8

Question: What is the value of `!'bang'`

Answer: false. `!` is NOT. If you put `!` in front of truthy values, it will return false. Using !! (double bang) is a tricky way to check anything truthy or falsy by avoiding implicit type conversion of == comparison.

Question: What is the value of `parseFloat('12.3.4')`

Answer: 12.3

Question: What is the value of `Math.max([2,3,4,5]);`

Answer: NaN

Question: `3 instanceof Number`

Answer: false

Question:`null == undefined`

Answer: true

Question: What is the value of `!!function(){};`

Answer: true

Question: What is the value of `typeof bar`

Answer: "undefined"

Question: What is the value of `typeof null`

Answer: "object"

Question: If `var a = 2, b =3` what would be value of `a && b`

Answer: 3

Question: What would be consoled `var foo = 'outside'; function logIt(){console.log(foo); var foo = 'inside';} logIt();`

Answer: undefined

Question: What is `-5%2`

Answer:-1. the result of remainder always get the symbol of first operand

Question: Why `.1+.2 != .3`

Answer:

Question: **`42..toString()`**

Anwser: `"42"`

Question: `4.2..toString`

Anwser: //SyntaxError: Unexpected token .

Question:`42 . toString()`

Anwser: `"42"`

Question: `typeof(NaN)`

Anwser:"number"

Question: `2 in [1,2]`

Anwser: false. Because "in" returns whether a particular property/index available in the Object. In this case object has index 0 and 1 but don't have 2. Hence you get false.

12. log prefix

Question: How could you set a prefix before everything you log? for example, if you `log('my message')` it will log: "(app) my message"

Answer: Just get the arguments, convert it to an array and unshift whatever prefix you want to set. Finally, use apply to pass all the arguments to console.

```
function log(){
  var args = Array.prototype.slice.call(arguments);
  args.unshift('(app)');
  console.log.apply(console, args);
}

log('my message'); //(app) my message
log('my message', 'your message'); //(app) my message your message
```

ref: This is a really good materials, walks you through an interview process.

13. Scope and hoisting

Question: What will you see in the console for the following example?

```
var a = 1;
function b() {
    a = 10;
    return;
    function a() {}
}
b();
console.log(a);
```

Answer: 1

Explanation:

- function declaration `function a(){}` is hoisted first and it behaves like `var a = function () {};`. Hence in local scope variable `a` is created.
- If you have two variables with same name (one in global another in local), local variable always get precedence over global variable.
- When you set `a = 10;`, you are setting the local variable `a` , not the global one. Hence, the value of global variable remain same and you get, 1 in the log. ref: js hoisting/scope
- **Extra:** If you didnt have a function named as "a", you will see 10 in the log.

ref: watch this video or learn more about scope and hoisting

14. Closures Inside Loops

Question: Look at the code below, you have a for loop if you have setTimeout inside it. If log the loop counter inside setTimeout, what will be logged?

```
    for(var i = 0; i < 10; i++) {
      setTimeout(function() {
          console.log(i);
       }, 10);
 }
```

Answer: The above will not output the numbers 0 through 9, but will simply print the number 10 ten times.

Explanation: The console log is inside the anonymous function of setTimeout and setTimeout is executed when current call stack is over. So, the loop finishes and before setTimeout get the chance to execute. However, anonymous functions keep a reference to i by creating a closure. Since, the loop is already finished, the value i has been set to 10. When setTimeout use the value of i by reference, it gets the value of i as 10. Hence, you see 10 ten times.

Solution: You can fix it by avoiding closure. Just create a IIFE (Immediately Invoked Function Expression), it will create its own scope and you can pass i to the function. In that case i will be a local variable (will not refer to i in the closure) and value of the i in every loop will be preserved.

```
for(var i = 0; i < 10; i++) {
     setTimeout((function(i) {
        console.log(i);
     })(i), 10)
```

```
    }
```

Alternative Solution: Look at the code below and use your brain (if any).

```
     for(var i = 0; i < 10; i++) {

       setTimeout(console.log.bind(console, i), 10);

    }
```

15. delete can delete but

Question: Look at the code below, I have a property in a object and I am creating a new object where I am setting it to a new value. If I delete that property what will i get if I try to access that property?

```
    var myObject = {

          price: 20.99,

           get_price : function() {

                 return this.price;

          }

    };

    var customObject = Object.create(myObject);

     customObject.price = 19.99;



    delete customObject.price;

    console.log(customObject.get_price());
```

Answer: You will see 20.99

Explanation: This is very interesting question. When you create `object.create(myObject)` you are creating new object where the `myObject` will be the parent of the newly created object. Hence the price property will be at the parent.

When you are assigning some value to `customObject.price`, you are creating a new property on the child. This means, when you `delete customObject.price` it deletes the `price` property in the customObject (in the child). However, when you call the method getprice, first it looks for `this.price` in the child since the customObject doesn't have price property, JavaScript executor walks through the prototype chain towards the parent. Since customObject was inherited from myObject and myObject has a `price` property, the get_price method returns the price from parent. Hence, you are getting 20.99

16. Pass by value or by reference

Question: Does JavaScript pass parameter by value or by reference?

Answer: Primitive type (string, number, etc.) are passed by value and objects are passed by reference. If you change a property of the passed object, the change will be affected. However, you assign a new object to the passed object, the changes will not be reflected.

```
var num = 10,

      name = "Addy Osmani",

     obj1 = {

        value: "first value"

      },

     obj2 = {

       value: "second value"

      },

      obj3 = obj2;


function change(num, name, obj1, obj2) {

     num = num * 10;

     name = "Paul Irish";

      obj1 = obj2;

     obj2.value = "new value";

}


change(num, name, obj1, obj2);


console.log(num); // 10

 console.log(name);// "Addy Osmani"

 console.log(obj1.value);//"first value"

console.log(obj2.valuee);//"new value"
```

```
    console.log(obj3.valuee);//"new value"
```

ref: Snook: passing by value or reference

17. memoization

Question: How could you implement cache to save calculation time for a recursive fibonacci function?

Answer: You could use poor man's memoization with a global variable. If fibonacci is already calculated it is served from the global memo array otherwise it is calculated.

```
var memo = [];


function _fibonacci(n) {
    if(memo[n]){
     return memo[n];
    }
    else if (n < 2){
       return 1;
    }else{
       fibonacci(n-2) + fibonacci(n-1);
    }
}
```

Better Implementation: implement memoization in JavaScript

18. Cache function execution

Question: How could you cache execution of any function?

Answer: You could have a method where you will pass a function and it will internally maintain a cache object where calculated value will be cached. When you will call the function with same argument, the cached value will be served.

```
    function cacheFn(fn) {

         var cache={};



          return function(arg){

               if (cache[arg]){

                   return cache[arg];

              }

              else{

                   cache[arg] = fn(arg);

                    return cache[arg];

              }

         }

    }
```

Question: What if you are passing more than one argument?

Answer: First we have to use arguments to get all the parameters passed to the function and then we can generate key for the cache object. Generating key for the cache object could be tricky and one solution could be just get the all the parameters and concatenate those. Look at the code below.

```


    return function(){

      var args = arguments;

       var key = [].slice.call(args).join('');

       if(cache[key]){

            return cache[key];

       }

        else{

             cache[key] = fn.apply(thi, args);

            return cache[key];
      }

    }
```

19. JQuery style chaining

Question: If you need to implement the following chaining with call back, how will you implement it?

```
    function slow(callback) {

         setTimeout(function(){

              if (Math.random() > 0.5) {

                   return callback("Error 417",null)

             }

             callback(null, {id:123})

         },500);

    }


    function exec(fn){

     //write your code here

    }


   exec(slow).done(function(data){

        console.log(data);

    }).fail(function(err){

         console.log("Error: " + err);

   })
```

Too much sleepy now. will try to put it up tomorrow.

```
var obj = {   // every method returns obj---------v

     first: function() { console.log('first');   return obj; },

     second: function() { console.log('second'); return obj; },

    third: function() { console.log('third');   return obj; }

}



obj.first().second().third();
```

ref: jquery like chaining or jquery like chaining

20. Animation

Question: How could you implement moveLeft animation?

Answer: Use setInterval that will place the element to the left position by some pixels in every 10ms. Hence, you will see the element moving towards the desired position. When you call setInterval, it returns a timeId. After reaching the desired location, you have to clear the time interval so that function will not be called again and again in every 10ms.

```
function moveLeft(elem, distance) {

   var left = 0;



   function frame() {

      left++;

     elem.style.left = left + 'px';



      if (left == distance)

         clearInterval(timeId)

   }



    var timeId = setInterval(frame, 10); // draw every 10ms

}
```

21. Currying

Question: How would you implement currying for any functions?

What is curring: Curring is partial invocation of a function. Currying means first few arguments of a function is pre-processed and a function is returned. The returning function can add more arguments to the curried function. It's like if you have given one or two spice to the curry and cooked little bit, still you can add further spice to it. A simple example will look like-

```
function addBase(base){
  return function(num){
     return base + num;
  }
}


var addTen = addBase(10);
addTen(5); //15
addTen(80); //90
addTen(-5); //5
```

Explanation: You are creating a closure that return a function. When you are curring with a new number, new number is added to the base you have provided.

Answer: You can add a curry method to the prototype of Function. If now parameters is passed to curry, you simply return the current function. If you have provided arguments to curry there are two steps

- **Step-1:** Concatenate old arguments (provided while creating curry), with new arguments (added after cooking little bit) by using `args.concat(toArray(arguments))`
- **Step-2:** Pass all the arguments to the function by using apply.
- **Extra:** Just be careful to retain the value of this.

```
Function.prototype.curry = function() {
    if (arguments.length<1) {
          return this; //nothing to curry. return function
```

```
      }

      var self = this;

      var args = toArray(arguments);

       return function() {

             return self.apply(this, args.concat(toArray(arguments)));

      }

}



function toArray(args) {

      return Array.prototype.slice.call(args);

}
```

To use it: Just pass the argument to the function.curry method and a function will be returned. Use returned function for further currying

```
 function converter = function(factor, symbol, input){

    return input * factor + symbol;

}



 var milesToKm = converter.curry(1.62, 'km');

mileToKm(3); //result here



var kgToLb = converter.curry(2.2, 'lb');

kgToLb(3); //result here
```