apoterenko / **javascript-interview-questions**

Essential JavaScript Interview Questions

#javascript #interview-practice #interview

82 commits | 1 branch | 0 releases | 1 contributor

Branch: master | New pull request | Create new file | Upload files | Find File | Clone or download

apoterenko update #40 + #40.1 — Latest commit c2bb5ac on Mar 18, 2018

README.md — update #40 + #40.1 — a year ago

README.md

Essential JavaScript Interview Questions

1. Explain the result of output:

```
(function test () {
        console.log(
                {}.constructor === arguments.constructor,
                [].constructor === arguments.constructor
        );
})();
```

2. Explain the result of output:

```
(function test (arguments) {
        console.log(arguments[0]);
})(100);


(function test () {
    var arguments;
        console.log(arguments[0]);
})(200);


(function test() {
        function sum() {
                var sum = 0, i;
                for (i in arguments) {
                        sum += i;
                }
                return sum;
        }

        console.log(sum(10, 20, 30, 40, 50));
})();
```

3. Explain the result of output:

```
(function test() {
        console.log(
                function () {} instanceof Object,
                function () {} instanceof Function,
                Function instanceof Object,
                Object instanceof Function
        );
})();
```

4. Explain the result of output:

```
(function test() {
        console.log(
                function () {}.apply.length
        );
})();
```

5. EXPLAIN THE RESULT OF OUTPUT:

5.1 Function & Function.prototype

```
console.log(
        Function instanceof Function,
        Function.prototype instanceof Function,
        Function.prototype.isPrototypeOf(Function),
        Function === Function.prototype,
        Function === Function.prototype.constructor,
        typeof Function.prototype,
        typeof Function
)
```

```
(function test() {
        console.log(
                Function === Object.constructor,
                Function === Number.constructor,
                Function === Function.constructor,
                Function === Window.constructor,
                Object === Object.prototype.constructor,
                Number === Number.prototype.constructor,
                Array === Array.prototype.constructor,
                Window === Window.prototype.constructor
        );
})();
```

6. Explain the result of output:

```
(function test() {
        console.log(
                typeof Object.prototype,
                typeof String.prototype,
                typeof RegExp.prototype
        );
})();
```

```
(function test() {
        console.log(
                typeof undefined,
                typeof typeof undefined,
                typeof null,
```

```
                typeof 1 / null,
                typeof [],
                typeof {},
                typeof document
        );
})();


(function test() {
        console.log(
                typeof Infinity,
                typeof NaN,
                typeof {null: null}.null,
                typeof {NaN: NaN}.NaN,
                typeof {Infinity: Infinity}.Infinity
        );
})();
```

7. Explain the result of output:

```
(function test() {
        var fn = function () {
                return this * 2;
        };

        console.log(fn.apply(undefined));
        console.log(fn.apply(null));
        console.log(fn.apply(1));
})();


(function test() {
        'use strict';
        var fn = function () {
                return this * 2;
        };

        console.log(fn.apply(undefined));
        console.log(fn.apply(null));
        console.log(fn.apply(1));
})();
```

8. Explain the result of output:

```
(function test() {
        console.log(
                Object.prototype.toString.call([]),
                Object.prototype.toString.call({}),
                Object.prototype.toString.call(Window),
                (16).toString(16),
                (true).toString(16),
                (false).toString(16)
        );
})();
```

9. Explain the result of output:

```
(function test() {
        var sum = function (a, b) {
                return a + b;
        };
        console.log(typeof sum.call.apply);
```

```
        console.log(sum.call.apply(null, [1, 2]));
})();
```

10. Explain the result of output:

```
(function test() {
        console.log(void (p = 1 / ""), p);
})();
```

11. Explain the result of output:

```
(function test() {
        (function () {
                a = 1;
                var a = 2;
        })();
        console.log(a);
})();
```

```
(function test() {
        var a = 1;
        function test() {
                if (!a) {
                        var a = 10;
                }
                console.log(a);
        }
        test();
        console.log(a);
})();
```

```
(function test() {
        (function () {
                var a = b = 3;
        })();

        console.log(
                typeof a,
                typeof b
        );
})();
```

12. Which a variant is preferable and why?

```
(function test() {
        console.log(error !== undefined && error.x);
})();
```

```
(function test() {
        console.log(typeof error !== 'undefined' && error.x);
})();
```

13. Explain the result of output:

13.1

```
(function test() {
        var a = [];
        a[100] = undefined;
        console.log(a.length);

        var b = new Array('100');
        console.log(b.length);
})();
```

13.2

```
(function test() {
        var a = [];

        console.log(
                a.length,
                [,].length,
                [, ,].length
        );

        a.length = -1;
        console.log(a.length);
})();
```

13.3

```
(function test() {
        var a = [1, 2, 3, 4, 5];

        a.length = null;
        console.log(a[4]);
})();
```

13.4

```
(function test() {
        var b = [1, 2, 3, 4, 5];

    b.length = undefined;
    console.log(b[4]);
})();
```

13.5

```
(function test() {
    console.log(
        [1,2,[3,4]] + [[5,6], 7, 8]
    );
    console.log([[[1], 2], 3].length);
})();
```

14. Explain the result of output:

```
(function test() {
        console.log(
                9 < 5 < 1,
                2 < 6 < 1,
                1 < 3 < 4
        );
})();
```

15. Explain the result of output:

```
(function test() {
        function fn() {
                return
                {
                        value: "test"
                }
        }

        console.log(
                typeof fn()
        );
})();
```

16. Explain the result of output:

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

        function sum(c) {
                return c;
        }

        console.log(sum(3));
        console.log(sum(2, 4));
})();
```

17. Explain the result of output:

```
(function test() {
        a = 1;
        window.b = 2;
        this.c = 3;
        var d = 4;

        delete a;
        delete b;
        delete c;
        delete d;

        console.log(typeof a, typeof b, typeof c, typeof d);
})();
```

18. Explain the result of output:

```
(function test() {
        var a = 1;

        setTimeout(function () {
                a = 0;
                console.log('Hi!');
        }, 0);

        while (a) {
        }
        console.log('Hello!');
})();
```

19. Explain the result of output:

```
(function test() {
        console.log(
                [] - [],
                [] + [],
                {} - {},
                {} + {}
        );
})();
```

20. Explain the result of output:

```
(function test() {
        var holder = {value: 1},
                holder2 = holder;

        holder.result = holder = {value: 0};

        console.log(
                holder.result,
                holder2
        );
})();
```

21. Explain the result of output:

```
(function test() {
        var test = {
                property: 'Value',

                getPropertyValue: function () {
                        return this.property;
                }
        };

        var getPropertyValue = test.getPropertyValue;

        console.log(
                getPropertyValue(),
                test.getPropertyValue()
        );
})();


(function test () {
        "use strict";

        var holder, fn;
        holder = {
                holderFn: function () {
                        console.log(this);
                }
        };
        fn = holder.holderFn;

        holder.holderFn();
        fn();
})();
```

22. What is the maximum depth of the stack, starting with the "test" function?

```
<script>
    var a = [1, 2, 3, 4, 5];

    (function test() {
        console.log((new Error()).stack);

        var item = a.pop();
        item && setTimeout(arguments.callee, 0);
    })();
</script>
```

```
var a = [1, 2, 3, 4, 5];

(function test() {
    console.log((new Error()).stack);

    var item = a.pop();
    item && arguments.callee();
})();
```

23. Explain the result of output:

```
(function test() {
        function fn() {
                arguments.callee.count = arguments.callee.count || 0;
                return arguments.callee.count++;
        }

        console.log(
                fn(),
                fn(),
                fn()
        );
})();
```

24. Explain the result of output:

```
(function test() {
        var a = '5', b = 2, c = a+++b;
        console.log(c);
})();
```

25. EXPLAIN THE RESULT OF OUTPUT:

25.1 Arithmetic OPERATORS

```
console.log(
        3+-+2+-+1,
        016 * 2,
        0x16 * 2,
        0.1 + 0.2,
        0.1 + 0.2 === 0.3,
        (0.1 + 0.2) + 0.3 === 0.1 + (0.2 + 0.3)
);
```

```
console.log(
        Math.floor(999.99) === ~~999.99,
        Math.floor(-999.99) === ~~-999.99,
        ~function(){}(),
```

```
        ~~function(){}(),
        ~~null,
        ~~undefined,
        ~~[],
        ~~{},
        ~~'Test'
);


console.log(
        1 + "2" + "2",
        1 + +"2" + "2",
        1 + -"1" + "2",
        +"1" + "1" + "2",
        "2" * 3,
        "6" / 2,
        "A" - "B" + "2",
        "A" - "B" + 2
);


console.log(
        Number('Test!'),
        Number(''),
        Number('00010'),
        Number(true),
        Number(NaN),
        parseInt('2', 2),
        parseInt('011', 8),
        parseInt('011', 10),
        parseInt('00C', 16),
        parseInt([10, 9, 8, 7, 6, 5, 4, 3, 2, 1]),
        1 / -0,
        isNaN(1 / -0),
        isFinite(1 / -0),
        0 / -0,
        isNaN(0 / -0),
        isFinite(0 / -0),
        1 / 0,
        isNaN(1 / 0),
        isFinite(1 / 0),
        0 / 0,
        isNaN(0 / 0),
        isFinite(0 / 0),
        NaN === NaN,
        Infinity === Infinity
);
```

26. Explain the result of output:

```
console.log(true == [1] && true == [2]);


console.log(
        new Boolean() == true,
        new Boolean("") == true,
        new Boolean("0") == true,
        new Boolean("1") == true,
        new Boolean("true") == true
);


console.log(
        false == '0',
        false === '0',
        true == '1',
```

```
        true === '1',
        true == '2',
        true === '2'
);
```

27. Explain the result of output:

```
(function () {
    var fn = function () {
        console.log(typeof this);
    };
    fn.call("Hello World!");
})();

(function () {
    "use strict";
    var fn = function () {
        console.log(typeof this);
    };
    fn.call("Hello World!");
})();


console.log(
        (function () {
                return (new this).stack;
        }).apply(Error)
);
```

28. Explain the result of output:

```
(function test() {
        var a = {},
                b = {value: 'test1'},
                c = {value: 'test2'};

        a[b] = 'test3';
        a[c] = 'test4';

        console.log(a[b]);
})();
```

29. Explain the result of output:

```
console.log(
        Date(),
        new Date,
        +Date(),
        +new Date,
        +new Date(),
        +new Date() === +new Date
);
```

30. Explain the result of output:

```
console.log(typeof confirm('Do you like JavaScript?'));
```

31. Eliminate non-existent state of promise.

1. fulfilled

2. awaiting
3. pending
4. refused
5. rejected
6. interrupted

32. Explain the result of output:

```
(function test() {
        console.log(
                [1, 2, 3, 4, 5].map(function (n) {
                        return n === 1 && 1 || arguments.callee(n - 1) * n;
                })
        );
})();


(function test() {
        "use strict";
        console.log(
                [1, 2, 3, 4, 5].map(function (n) {
                        return n === 1 && 1 || arguments.callee(n - 1) * n;
                })
        );
})();
```

33. Explain the result of output:

```
(function test() {
        var s1 = 'test',
                s2 = new String('test'),
                s3 = String('test');

        console.log(
                s1 == s2,
                s1 === s2,
                s1 == s3,
                s1 === s3,
                s1.constructor === s2.constructor,
                s1.constructor === s3.constructor,
                typeof s1,
                typeof s2,
                typeof s3
        );

        console.log(
                s1.slice() == s1,
                s1.slice() == s2,
                s1.slice() == s3,
                s1.slice() === s1,
                s1.slice() === s2,
                s1.slice() === s3
        );

        s1[2] = 'w';
        console.log(s1);
})();
```

34. Explain the result of output:

```
(function test() {
        var a = [1, 2, 3],
```

```
                b = a.reverse(),
                c = [4, 5, 6];

        b.push(c);

        console.log(a.length, b.length);
        console.log(a.slice(-1));
        console.log(b.slice(-1));
})();
```

35. Explain the result of output:

```
(function test() {
        for (var i = 0; i < 5; i++) {
                setTimeout(function () {
                        console.log(i);
                }, 0)
        }
})();
```

36. Explain the result of output:

```
<script>
    Object.defineProperty(this, "value", {
        value: 100,
        writable: false
    });

    value = 200;

    console.log(window.value, this.value, value);
</script>


<script>
    Object.defineProperty(this, "variable", {
        value: 100,
        configurable: false
    });

    console.log(delete variable);
</script>


<script>
    var a = 100;
    console.log(Object.getOwnPropertyDescriptor(this, "a"));

    b = 200;
    console.log(Object.getOwnPropertyDescriptor(this, "b"));
</script>
```

37. Explain the result of output:

```
(function test() {
        var Factory = function () {
                var a = [];
                a[Array.prototype.pop.apply(arguments)] = 1;
                return a;
        };

        console.log(
                Factory(0).length,
```

```
                Factory(100).length,
                Factory(Infinity).length,
                Factory(NaN).length
        );
})();
```

38. Explain the result of output when the page is fully loaded:

```
<html>
    <head>
        <script>
            setTimeout(function () {
                console.log(
                        performance.now() > 5000
                );
            }, 10000);
        </script>
    </head>
    <body>
    </body>
</html>
```

39. Do you see the pitfalls in the code?

```
<html>
    <body>
        <script>
              function nodeHouse(id) {
                var node;
                return {
                  make: function () {
                    node = document.createElement("div");
                    node.setAttribute('id', id);
                    document.body.appendChild(node);
                    return this;
                  },
                  destroy: function () {
                    document.body.removeChild(node);
                    return this;
                  },
                  test: function () {
                    return node.getAttribute('id') == id;
                  }
                };
              }

              var nodesHouse = [],
                      currentNodeHouse;

              for (var i = 0; i < 100000; i++) {
                nodesHouse.push(currentNodeHouse = nodeHouse(i));
                currentNodeHouse.make().destroy();
              }
        </script>
    </body>
</html>
```

40. HOW TO PREVENT A MEMORY LEAK?

40.1 GLOBAL OBJECTS

```
/** This is a modifiable section **/
function CacheKey () {
  this.t = new Array(2000);
```

```
}

var cacheKeys = {},
    index = 0,
    cache = new Map();

/** This is not a modifiable section - START **/
function addToMap() {
  setTimeout(addToMap);

  cacheKeys[index] = new CacheKey();
  cache.set(cacheKeys[index], 'New value');

  delete cacheKeys[index];
  index++;
}

setTimeout(addToMap);
/** This is not a modifiable section - END **/
```

41. Explain the result of output:

```
(function () {
        var dynamicCode = '(function () {return this}())';

        console.log(
                eval(dynamicCode),
                eval.call(null, dynamicCode)
        );
})();

(function () {
        'use strict';
        var dynamicCode = '(function () {return this}())';

        console.log(
                eval(dynamicCode),
                eval.call(null, dynamicCode)
        );
})();
```

42. EXPLAIN THE RESULT OF OUTPUT:

42.1 FUNCTION CONSTRUCTOR SCOPE

```
(function () {
        var a = 1;
        (new Function('a = 2'))();
        console.log('a1:', a)
})();

console.log('a2:', a)
```

42.2 FUNCTION CONSTRUCTOR SCOPE

```
(function () {
        var b = 3;
        (new Function('b = 4')).call(this);
        console.log('b1:', b);
})();

console.log('b2:', b);
```

42.3 FUNCTION CONSTRUCTOR SCOPE, STRICT MODE

```
(function () {
        'use strict';

        var c = 5;
        (new Function('c = 6'))();
        console.log('c1:', c);
})();

console.log('c2:', c);


(function () {
        'use strict';

        var c = 5;
        (new Function('"use strict"; c = 6'))();
        console.log('c3:', c);
})();

console.log('c4:', c);
```

42.4 FUNCTION CONSTRUCTOR SCOPE, STRICT MODE

```
(function () {
        'use strict';

        var d = 7;
        (new Function('d = 8')).call(this);
        console.log('d1:', d);
})();

console.log('d2:', d);


(function () {
        'use strict';

        var d = 7;
        (new Function('"use strict"; d = 8')).call(this);
        console.log('d3:', d);
})();

console.log('d4:', d);
```

43. EXPLAIN THE RESULT OF OUTPUT:

43.1 FUNCTION CONSTRUCTOR, STRICT MODE

```
(function () {
        console.log(
                new Function('return this')()
        );
        console.log(
                new Function('"use strict"; return this')()
        );
})();
```

43.2 FUNCTION CONSTRUCTOR, STRICT MODE

```
(function () {
        'use strict';

        console.log(
                new Function('return this')()
        );
        console.log(
                new Function('"use strict"; return this')()
        );
})();
```

43.3 FUNCTION CONSTRUCTOR, STRICT MODE

```
(function () {
        console.log(
                new Function('return arguments[0]')()
        );
        console.log(
                new Function('"use strict"; return arguments[0]')()
        );
})(123);
```

43.4 FUNCTION CONSTRUCTOR, STRICT MODE

```
(function () {
        'use strict';

        console.log(
                new Function('return arguments[0]')()
        );
        console.log(
                new Function('"use strict"; return arguments[0]')()
        );
})(123);
```

44. EXPLAIN THE RESULT OF OUTPUT:

44.1 FUNCTION DECLARATION, HOISTING

```
(function test() {
    fn();

    function fn () {
        console.log('The function is called!');
    }
})();
```

44.2 FUNCTION EXPRESSION, HOISTING

```
(function test() {
    fn();

    var fn = function () {
        console.log('The function is called!');
    }
})();
```

44.3 FUNCTION EXPRESSION, HOISTING

```
(function test() {
    myFn();

    var fn = function myFn() {
        console.log('The myFn function is called!');
    }
})();
```

44.4 FUNCTION EXPRESSION, IIFE (IMMEDIATELY INVOKABLE FUNCTION EXPRESSIONS)

```
console.log(!function foo() {return 0;});
console.log(!function foo() {return 0;}());
console.log(-function foo() {return 1;});
console.log(-function foo() {return 1;}());
console.log(+function foo() {return 1;});
console.log(+function foo() {return 1;}());
console.log(~function foo() {return 1;});
console.log(~function foo() {return 1;}());
console.log(void function foo() {return 1;});
console.log(void function foo() {return 1;}());
```

45. YOU NEED TO IMPLEMENT:

45.1 "MULTIPLY" FUNCTION. NUMBER INSTANCES

```
console.log((33).multiply(2));  // 66
console.log((33).multiply(3));  // 99
```

45.2 "DUPLICATION" FUNCTION. STRING INSTANCES

```
console.log('HELLO'.duplication());     // HELLOHELLO
```

46. EXPLAIN THE RESULT OF OUTPUT:

46.1 COMMA OPERATOR

```
console.log(
        typeof (true, '1', 1)
);
```

46.2 COMMA OPERATOR

```
(function test() {
        var x, y, z;
        x = (y = 1, z = 2);

        console.log(x);
})();
```

47. EXPLAIN THE RESULT OF OUTPUT:

47.1 new OPERATOR

```
var testClass1 = function () {
        return new Number(1);
};
```

```
var testClass2 = function () {
        return Number(2);
};

console.log(
        new testClass1 instanceof testClass1,
        new testClass2 instanceof testClass2
);
```

48. YOU NEED TO IMPLEMENT:

48.1 instanceof OPERATOR

```
var testClass = function () {}
<YOUR_CODE_HERE>
console.log(new testClass instanceof Array); // true
```

48.2 instanceof OPERATOR

```
var testClass1 = function () {
    <YOUR_CODE_HERE>
}
var testClass2 = function () {
    <YOUR_CODE_HERE>
}

var object1 = new testClass1();
var object2 = testClass1();

console.log(object1 instanceof testClass1 === object2 instanceof testClass1);       // true
console.log(object1 instanceof testClass2 === object2 instanceof testClass2);       // true
```

49. YOU NEED TO IMPLEMENT:

49.1 isPrototypeOf OPERATOR

```
var testObject1 = {x: 1};
var testObject2 = <YOUR_CODE_HERE>

console.log(
        testObject1.isPrototypeOf(testObject2)              // true
);
```

49.2 isPrototypeOf OPERATOR

```
var testClass1 = function () {};
var testClass2 = function () {};

<YOUR_CODE_HERE>

console.log(
        testClass1.prototype.isPrototypeOf(new testClass2)  // true
);
```

50. EXPLAIN THE RESULT OF OUTPUT:

50.1 Object.create METHOD

```
console.log(
        typeof Object.create(null).prototype,
        Object.create(null) instanceof Object,
        Object.create({}) instanceof Object
);
```

50.2 Object.create METHOD & Object.assign METHOD

```
var o = Object.assign(Object.create(null), { a: 1 }, { a: 2 }, { a: 3 });
Object.assign(Object.prototype, { f: function() {} });

for (var i in o) {
        console.log(o[i]);
}
```

51. EXPLAIN THE RESULT OF OUTPUT:

51.1 Event Loop

```
// The first tab is followed to http://localhost
localStorage.setItem('test', 100500);
location.reload(true);
localStorage.setItem('test1', 100501);

setTimeout(function () {
        localStorage.setItem('test2', 100502);
});

setTimeout(function () {
        localStorage.setItem('test3', 100503);
}, 100);

// Then the second tab is followed to http://localhost
console.log(
        localStorage.getItem('test'),
        localStorage.getItem('test1'),
        localStorage.getItem('test2'),
        localStorage.getItem('test3')
);
```