

JAVASCRIPT QUIRKS

QUIZ #1

#1

1) Is JavaScript case sensitive language?

- Yes
- No



YES

Javascript is case sensitive

- The variables `x` and `X`, are two different variables.

#2

```
....  
(function() {  
    return typeof arguments;  
})();
```

2) What will be the output of the above code snippet?

- "object"
- "array"
- "arguments"
- "undefined"

TYPEOF ARGUMENTS === 'OBJECT'

- The **arguments** object is an Array-like **object**
- You can access the individual argument values through an array index, and it has a **length** property like other arrays, but it doesn't have the standard **Array** methods like **push** and **pop**.

3

```
(function(x){  
    delete x;  
    return x;  
})(1);
```

3) What will be the output of the above code snippet?

- 1
- null
- undefined
- Error

ANSWER IS 1

```
(function(x){  
    delete x;  
    return x;  
})(1);
```

- x is regarded as being created with a variable declaration
- Thus x is marked with the DontDelete flag and delete x would fail by yielding false (but not by throwing an error) .
- <http://perfectionkills.com/understanding-delete/>

POP QUIZ

```
var o = { x: 1 };
delete o.x;
o.x;
```

```
function x(){}
delete x;
typeof x;
```

```
var x = 1;
delete x;
x;
```

```
var z = function(){}
delete z;
typeof z;
```

POP QUIZ

```
var o = { x: 1 };
delete o.x;
o.x;
```

"undefined"

```
var x = 1;
delete x;
x;
```

```
function x(){}
delete x;
typeof x;
```

```
var z = function(){}
delete z;
typeof z;
```

POP QUIZ

```
var o = { x: 1 };
delete o.x;
o.x;
```

"undefined"

```
function x(){}
delete x;
typeof x;
```

"function"

```
var x = 1;
delete x;
x;
```

```
var z = function(){}
delete z;
typeof z;
```

POP QUIZ

```
var o = { x: 1 };
delete o.x;
o.x;
```

"undefined"

```
function x(){}
delete x;
typeof x;
```

"function"

```
var x = 1;
delete x;
x;
```

```
var z = function(){}
delete z;
typeof z;
```

POP QUIZ

```
var o = { x: 1 };
delete o.x;
o.x;
```

"undefined"

```
function x(){}
delete x;
typeof x;
```

"function"

```
var x = 1;
delete x;
x;
```

1

```
var z = function(){}
delete z;
typeof z;
```

"function"

#4

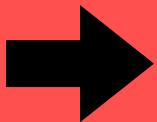
```
var y = 1, x = y = typeof x;  
x;
```

4) What will be the output of the above code snippet?

- 1
- "number"
- undefined
- "undefined"

X === 'UNDEFINED'

```
var y = 1, x = y = typeof x;  
x;
```



```
var y = 1;  
y = typeof x;  
var x = y;  
x;
```

- When `y = typeof x` is executed, `x` has not been defined yet, so `y` becomes the string "undefined", which is then finally assigned to `x` when it is formally declared.

#5

```
var foo = {  
    bar: function() { return this.baz; },  
    baz: 1  
};  
(function() {  
    return typeof arguments[0]();  
})(foo.bar);
```

5) What will be the output of the above code snippet?

- "undefined"
- "object"
- "number"
- "function"

ANSWER IS 'UNDEFINED'

```
var foo = {
    bar: function() { return this.baz; },
    baz: 1
};
(function(){
    return typeof arguments[0];
})(foo.bar);
```

- Because we're calling `foo.bar` with the name `arguments[0]`, `this` in `foo.bar` is not bound to `foo`, but bound to whatever `this` refers to within the anonymous function! Here, that happens to be the global object.
- There is no property called `baz` in the global object, so `typeof` operator yields "undefined".

#6

```
var x = 1;  
if (function() {}) {  
    x += typeof f;  
}  
x;
```

6) What will be the output of the above code snippet?

- 1
- "1function"
- "1undefined"
- NaN

ANSWER IS '1UNDEFINED'

```
var x = 1;
if (function(){}) {
    x += typeof f;
}
x;
```

- Functions are always truthy values in JavaScript
- f is not defined here, so typeof f yields the string "undefined".
- Adding a string to a number results in a string

#7

```
var x = [typeof x, typeof y][1];  
typeof typeof x  
....
```

7) What will be the output of the above code snippet?

- "number"
- "string"
- "undefined"
- "object"

ANSWER IS 'STRING'

```
var x = [typeof x, typeof y][1];
typeof typeof x;
```

- `typeof y` === "undefined"
- `typeof x` === "string"
- `typeof typeof x` === "string"

POP QUIZ

- `typeof typeof undefined`
- `typeof typeof null`
- `typeof typeof 647`
- `typeof typeof []`
- `typeof typeof true`

#8

```
(function f() {  
    function f() { return 1; }  
    return f();  
    function f() { return 2; }  
})();
```

8) What will be the output of the above code snippet?

- 1
- 2
- Error
- undefined

ANSWER IS 2

```
(function f(){  
    function f(){ return 1; }  
    return f();  
    function f(){ return 2; }  
};)
```

- Function declarations and variable declarations are always moved (“hoisted”) invisibly to the top of their containing scope by the JavaScript interpreter.

#9

```
typeof null
```

9) What will be the output of the above code snippet?

- "undefined"
- "object"
- undefined
- null

TYPEOF NULL === 'OBJECT'

- ECMAScript specification defines null as the primitive value that represents the intentional absence of any object value (ECMA-262, 11.4.11)

Difference between null and undefined

```
1 | typeof null          // object (bug in ECMAScript, should be null)
2 | typeof undefined    // undefined
3 | null === undefined // false
4 | null == undefined // true
```

#10

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

```
console.log("a defined? " + (typeof a !== 'undefined'));  
console.log("b defined? " + (typeof b !== 'undefined'));
```

10) What will be the output of the above code snippet?

- "a defined? true" / "b defined? true"
- "a defined? false" / "b defined? true"
- "a defined? true" / "b defined? false"
- "a defined? false" "b defined? false"

A DEFINED? FALSE' / 'B DEFINED? TRUE



The issue here is that most developers understand the statement `var a = b = 3;` to be shorthand for:

27

```
var b = 3;  
var a = b;
```



But in fact, `var a = b = 3;` is actually shorthand for:



```
b = 3;  
var a = b;
```

Therefore, `b` ends up being a global variable (since it is not preceded by the `var` keyword) and is still in scope even outside of the enclosing function.

The reason `a` is undefined is that `a` is a local variable to that self-executing anonymous function

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

If a variable is initialized (assigned a value) without first being declared with the `var` keyword, it is automatically added to the global context and it is thus a global variable.