JS Questions LIJIA XU (1)

1. not defined ⇒ neither declared nor defined
undefined ⇒ declared but not defined
var x; // declaration
typeof x === 'undefined'; //return true
console.log(y); // ReferenceError; y is not defined

2. if (x <= 100){...}
if (!(x > 100)){...}
| if value of x is NaN, then the code on left ⇐ not same
Also true for any x convert to x return NaN
ex. [1,2,3], {a:2} // use isNaN() to check

3. drawback of declaring methods directly in JS objects? memory inefficient + easy to read
can also create method in constructor prototype
var test = function(a, b, c){
// property definitions
};
test.prototype.x = function(){...};

4. closure in JS ⇒ a function defined inside another function and has access to the variable which is declared and defined in parent function scope.

console.log(mul(2)(3)(4)); //24
5. mul function which works console.log(mul(4)(3)(4)); //48
function mul(x){
return function(y){
return function(z){
return x*y*z;
};
};
};
function is the first class object, so it can be returned by function
*a function is an instance of the object type
*function can have properties and has a link back to its constructor method
*can be stored as var *can pass as para to another function

6. empty an array in JS var arrayList = ['a', 'b', 'c'];
arrayList = []; // only referenced the array by its original variable.
arrayList.splice(0, arrayList.length);
arrayList.length = 0 // will also update all references
*not often | while (arrayList.length){arrayList.pop();}

7. check if an object is an array
function greet(param){
if (typeof param === 'string'){
}
else{
}
} // for method overloading
jQuery
if($.isArray(arrayList)
{...} else {...}
JQuery use: object.prototype.toString.call;
if (Object.prototype.toString.call(arrayList) === '[object Array]'){ }

8. output of the code
var output = (function(x){
delete x;
return x;
})(0); // outputs 0
delete is used to delete a property from object, x is local variable

9. var x = 1;
var output = (function(){
delete x;
return x;
})(); // will return 1
x is global variable

10.
var x = {foo: 1};
var output = (function(){
delete x.foo;
return x.foo;
})();
// outputs undefined

11. empl got company as prototype property
var Employee = {
company: 'xyz'
}
var emp1 = Object.create(Employee);
delete emp1.company
console.log(emp1.company); //'xyz'
// emp1.hasOwnProperty('company') → false
delete will not delete the prototype property.
ways to delete company:
delete Employee.company
delete emp1.__proto__.company

12. what is undefined x 1 in JS
var trees = ["redwood", "bay", "ok"];
delete trees[1];
trees[1] === undefined // true
uninitialized index ✱✱
different browser display differently

13 output? var trees = ["xyz", "xxx", "test"];
delete trees[1]; // array length not influenced
console.log(trees.length); // outputs 3

14 output?
var bar = true;
console.log(bar + 0); // 1
console.log(bar + "xyz"); // "true xyz"
console.log(bar + true); // 2
console.log(bar + false); // 1
// undefined

15. output? var z = 1, y = z = typeof y; // right to left
var z;
z = 1;
var y;
z = typeof y;
y = z;
console.log(y);

16 Output?
var foo = function bar(){
return 12;};
typeof bar();
// output ReferenceError
fix 1:
var bar = function(){
return 12;};
typeof bar();
definition can only have one reference variable
fix 2: // function
function bar(){return 12;};
typeof bar();
var foo = function bar(){
// foo visible here
// bar visible here
console.log(typeof bar());
};
// foo is visible here
// bar undefined here

17 difference between declaring functions
* defined at runtime and is called function expression: foo(); // error
var foo = function(){
// some code
}
foo; // okay
*defined at parse time and called function statement; declarations
bar(); // okay
function bar(){
// some code
}
bar(); // okay

18. function definition hoisting only occurs for function declarations, not function expression
// outputs: "Definition hoisted!"
definitionHoisted();
// Type error: undefined is not a function
definitionNotHoisted();
function definitionHoisted(){
console.log("Definition Hoisted!");
}
var definitionNotHoisted = function(){
console.log("Definition not hoisted!");
}

Named function expression
funcName(); // ReferenceError: funcName is not defined
varName(); // TypeError: undefined is not a function
var varName = function funcName(){
console.log("Definition not hoisted!");
};

19. Output? var salary = "1000$";
(function(){
console.log("original was " + salary);
var salary = "5000$";
console.log("New salary " + salary);
})(); // "original was undefined"
because of hoisting

20. typeof instanceof
typeof returns a string
typeof (null) // return object
number string boolean object function undefined Symbol
var dog = new Animal();
dog instanceof Animal; // output: true
// true because dog inherits from Animal.prototype
var name = new String("xyz");
name instanceof String; // output: true

21. length of the associative array?
in JS, associative array same as object
var counterArray = {
A: 3, // method 2: IE9+, browsers
B: 4 // Object.getOwnPropertyNames(counterArray).length // 3
}; // method 1: Object.keys(counterArray).length; // 3
counterArray["C"] = 1;
Method 3: This way ignore the properties
function getLength(object){
var count = 0;
for (key in object){
if (object.hasOwnProperty(key)){
count++};
}
return count;
} // method if Underscore and lodash already included:
_.size({one: 1, two: 2, three: 3});
⇒ 3

22 Difference Function, Method, and Constructor
In JS, three different usage patterns of one single construct.
var obj = {
helloWorld: function(){
return "Hello" + this.name;},
name: 'John Carter'}
obj.helloWorld();