FRONT END – PART 2
CLOSURES

(BOM and DOM)

1. Suppose a web page is being loaded inside the Google Chrome browser. Which of the following is/are true regarding the associated object models?

a) The web document being loaded is the root object of the BOM.

b) The BOM may differ if the same web page is loaded in another browser.

c) Within the same browser, BOM and DOM are equivalent.

d) The parent object of the screen object is the DOM.

(Window Object)

2. What will the following code print in the console?

```
var a =10;
function foo(){
var a = 20;
console.log(a);
console.log(window.a);
}
console.log(a);
window.foo();
```

a) 10 20 10 b) 10 10 10 c) 10 20 20

d) 20 20 20 e) None of the Above

(Local Scope)

3. What will the following code print in the console?

```
function foo(a){
a = 20;
console.log(a);
}
foo(10);
console.log(a);
```

a) 10 20 b) 20

c) 10 Undefined **d) 20 Error: a is not defined**

(What is Output)

4. Consider the following code snippet.

```
var a = 10;
console.log(a);
var a =20;
console.log(a);
```

What will be the expected output?

[✓ a) 10 20] b) 10 10

c) 20 20 d) Error: Identifier 'a' has already been declared

(What is the Expected Output)

5. What will the following code print in console?

```
var a =10;
var b;
function outer(){
    var b = 30;
    function inner(a){
        var a = 30;
        console.log(a++ ,b++)
        }
        console.log(a, ++b);
        inner(a);
}
outer();
console.log(a++,b++);
```

```
  i.  20 31
      30 31
      10

 ii.  20 30
      30 30
      10

iii.  30 20
      30 30
      10

iv.   10
      20 31
      30 31

 v.   10 31
      30 31
      10 NaN

vi.  None of the Above
```

a) i b) ii c) iii

d) iv [✓ e) v] f) vi

(What is the Expected Output)

6. What will the following code print in console?

```
var a =10;
function outer(){
    var a = 20;
    var b = 30;
    function inner(a){
        var a = 30;
        console.log(a++ ,b++)
        }
    console.log(a, ++b);
    inner(a);
}
outer();
console.log(a++);
```

```
  i.   20 31
       30 31
       10

 ii.   20 30
       30 30
       10

iii.   30 20
       30 30
       10

 iv.   10
       20 31
       30 31

  v.   10 31
       30 31
       10 NaN

 vi.  None of the Above
```

a) i　　b) ii　　c) iii

d) iv　　e) v　　f) vi

(Console Output)

7. You have index1.html and index2.html files. The index1.html file contains a hyperlink to the index2.html file. Look at the code files given below and guess the output in the console when you open index1.html, and then click on the hyperlink ‘Go to Page 2’.

index1.html

```
<html>
<head>
   <title>Page 1</title>
</head>
<body>
```

```
      <p>This is Page 1</p>
      <a href="index2.html">Go to Page 2</a>
      <script src="script1.js"></script>
    </body>
   </html>
```

```
index2.html
```

```
   <html>
   <head>
       <title>Page 2</title>
   </head>
   <body>
       <p>This is Page 2</p>
       <script src="script2.js"></script>
   </body>
   </html>
```

```
script1.js
```

```
   var c = "script1" ;
   console.log(c) ;
```

```
script2.js
```

```
   var c = "script2" ;
   console.log(c) ;
```

a) script1 script1 as c is a global variable initialised with ‘script1’.

b) script2 script2 as c is a global variable and gets overridden with ‘script2’.

c) script1 script1 as c is not a global variable of the same scope.

d) script1 script2 as c is not a global variable of the same scope.

SOLUTION DESCRIPTION:

Explanation: These two JS files are attached to two different HTML files. Hence these variables don’t share the same scope and are printed in the console accordingly.

(Global Scope)

8. What is the expected output?
Where this is the HTML code

```
   <html>

   <body>

   <script src="script1.js"></script>

   <script src="script2.js"></script>

   </body>

   </html>
```

```
This is script1.

      var a = 10;
      setTimeout(function(){
      console.log(a);
      },1000);

This is script2.

      var a = 20;
      console.log(a);
```

a) 10 20 b) 10 10 **c) 20 20** d) 20 10

(IIFE)

9. What will the following statements print in the console?

```
      var a = (function(){
          return typeof arguments;
      })();
      console.log(a);
```

a) object b) array c) arguments d) undefined

SOLUTION DESCRIPTION:

Explanation: 'arguments' is an inbuilt object in javascript which is accessible only inside functions. It contains information about the arguments passed to the function.

(Identify Scope of Variables)

10. Look at the following code and identify which variables can be accessed in which scope (global, function, block-level)? (multiple options correct)

```
script.js

      {
        var a = 10 ;
        let b = 20 ;
      }

      c = 30 ;

      if(c===30){
          let d = 50 ;
          console.log(d) ;
      }
      else{
           var e = 20;
           console.log(e) ;
      }
```

a) a → block-level **b) b → block-level**

c) a → global d) d → global

e) e → global if the else block ran **f) d → block-level**

g) e → block-level

(Scoping in JavaScript)

11. What kind of scoping does JavaScript use?

a) Literal **b) Lexical** c) Segmental d) Segmental

(What is the Expected Output)

12. What will the following code snippet print?

```
function Adder(x) {
  return function(y) {
    return x + y;
  };
}
var add5 = Adder(5);
var add10 = Adder(10);
console.log(add5(2));
console.log(add10(2));
```

a) 7 12 b) 5 15 c) 5 10 d) 7 15

SOLUTION DESCRIPTION:

Explanation: add5 and add10 are two different references to two different lexical scopes of the same function.

(Closure In JavaScript)

13. What is the expected output?

```
var add = (function ( ) {
  var counter = 0;
  return function ( ) {
       counter += 1;
       return counter;
}
})();
console.log(add());
console.log(add());
console.log(add());
```

a) 0 0 0 b) 0 1 1 c) 0 1 2 **d) 1 2 3**

SOLUTION DESCRIPTION:

Hint : Self Invoking Functions are used

Explanation: The function that IIFE is returning contains the lexical context of the IIFE, and it's stored in ‘add’. It gets invoked 3 times, and each time the counter is updated in the same lexical context.

(var in Loop)

14. What will the following code print in console?

```
for(var a = 1; a < 5; a++){
```

```
setTimeout(function(){
console.log(a)}, 1000);
}
```

a) 1 2 3 4 5   b) 1 2 3 4   c) 1 1 1 1

d) 5 5 5 5   e) None of the Above

(let in Loop)

15. What will the following code print in console?

```
for(let a = 1; a < 5; a++){
    setTimeout(function(){
    console.log(a)}, 1000);
}
```

a) 1 2 3 4   b) 1 1 1 1

c) 5 5 5 5   d) None of the Above

(Arrow Function Declaration)

16. Is the following function declaration allowed in JavaScript?

```
var func = (
  a,
  b,
  c
) => (
  1
);
```

a) Yes   b) No

(Arrow Function)

17. What will the expected output when we run the following code in the console?

```
console.log((function(x, f = () => x) {
  var x;
  var y = x;
  x = 2;
  return [x, y, f()];
})(1));
```

a) 2, 1, 1   b) 1, 1, 2   c) 2, 1   d) 1, 1

(What is the Expected Output)

18. What will the following code print in the console?

```
let user = {
   firstname: 'John',
   lastname: 'Doe',
   getFullName: function(){
        return function(){
            console.log(`The full name of the user is ${this.firstname} $
{this.lastname} `);
        }
    }
}
user.getFullName()();
```

a) The full name of the user is ${this.firstname} ${this.lastname}

b) The full name of the user is undefined undefined

c) The full name of the user is John Doe

d) None of the Above

SOLUTION DESCRIPTION:

In the given example, the function which is getting returned i.e.

```
return function(){
    console.log(`The full name of the user is ${this.firstname} ${this.lastname} `);
}
```

Is a unbounded function (An unbound function is a function that is not bound to an object) So this in the above function refers to the global (window) object. Since unbound functions are implicitly bound to the global scope). And since no firstname and lastname variables were defined in the window environment, ${this.firstname} ${this.lastname} prints undefined undefined.

(What is the Expected Output)

19. What will the following code print in the console?

```
let user = {
   firstname: 'John',
   lastname: 'Doe',
   getFullName: function(){
       return() => {
           console.log(`The full name of the user is ${this.firstname} $
{this.lastname} `);
       }
   }
}
user.getFullName()();
```

a) The full name of the user is ${this.firstname} ${this.lastname}

b) The full name of the user is undefined undefined

c) The full name of the user is John Doe

d) None of the Above

SOLUTION DESCRIPTION:

ES6 arrow functions can't be bound to a this keyword, so it will lexically go up a scope, and use the value of this in the scope in which it was defined.

Hence this refers to the user object in this case and thus ${this.firstname} ${this.lastname} will print John Doe.

ASSIGNMENT

(Block Scope)

20. What is the expected output?

```
{
  let message = "Hello";
  console.log(message);
}
console.log(message);
```

a) Hello Hello

b) Hello undefined

c) Hello Error: message is not defined

d) None of the Above

(let)

21. What is the expected output if we run the following code in console?

```
function makeCounter() {
  let count = 0;
  return function() {
    return count++;
  };
}
let counter1 = makeCounter();
let counter2 = makeCounter();
console.log( counter1() );
console.log( counter1() );
console.log( counter2() );
```

a) 0 1 0

b) 1 2 1

c) 0 1 2

d) 1 2 3

(What is the Expected Output)

22. What is the expected output?

```
function makeArmy() {
  let shooters = [];
  let i = 0;
  while (i < 10) {
    let shooter = function() {
      console.log( i );
    };
    shooters.push(shooter);
    i++;
  }
  return shooters;
}
let army = makeArmy();
army[0]();
army[5]();
```

a) 0 5

b) 0 0

c) 5 5

d) 10 10 e) None of the Above

SOLUTION DESCRIPTION:

Explanation: The 'shooters' array is an array of functions. When you call makeArmy(), the while loop runs, and when the loop ends, the final value of i becomes 10. When you call makeArmy[0](), it runs the function that logs the value of i 10, similarly for makeArmy[5]().

(What is the Expected Output)

23. What is the expected output?

```
function makeArmy() {
  let shooters = [];
  for(let i = 0; i < 10; i++) {
    let shooter = function() {
      console.log( i );
    };
    shooters.push(shooter);
  }
  return shooters;
}
let army = makeArmy();
army[0]();
army[5]();
```

a) 0 5 b) 0 0 c) 5 5

d) 10 10 e) None of the Above

(IIFE)

24. What will the following code snippet print in console?

```
var Sequence = (function sequenceIIFE() {
    var current = 0;
    return {
        getCurrentValue: function() {
            return current;
        },
        getNextValue: function() {
            current = current + 1;
            return current;
        }
   };
}());
console.log(Sequence.getNextValue());
console.log(Sequence.getNextValue());
console.log(Sequence.getCurrentValue());
```

a) 0 0 0 b) 0 1 1 **c) 1 2 2**

d) 1 1 2 e) None of the Above

(What is the Output)

25. What will the following code print in the console?

```
let People = function(person, age) {
        this.person = person;
        this.age = age;
        this.info = function() {
            console.log(this);
           setTimeout(() => {
            console.log(this.person + " is " + this.age + " years old");
           }, 3000);
        }
    }
let person1 = new People('John', 21);
person1.info();
```

a) window {postMessage: ƒ, blur: ƒ, focus: ƒ, close: ƒ, parent: Window, …} person is undefined years old

b) window {postMessage: ƒ, blur: ƒ, focus: ƒ, close: ƒ, parent: Window, …} John is 21 years old

c) People {person: "John", age: 21, info: ƒ} John is 21 years old

d) None of the Above

SOLUTION DESCRIPTION:

The statement 'let person1 = new People('John', 21);' will create a new People object and call it's constructor function. Which assigns value of person and age to the object and define the function info. The statement 'person1.info();' will invoke the function info of the person1 object. In function info 'console.log(this); ' this keyword refers to an object, that object which is executing the current bit of javascript code. After 3000ms setTieout() will print John is 21 years old.

(Arrow function)

26. What will the following function return?

```
var arguments = [1, 2, 3];
var arr = () => arguments[2];
arr();
function foo(n) {
  var f = () => arguments[0] + n;
  return f();
}
foo(3);
```

a) 4

b) 5

c) 6

d) 7

e) None of the Above

SOLUTION DESCRIPTION:

You need to check whether arguments exists in DOM or not.

(Closures)

27. Which of the following is true about closures?

a) Closures are frequently used for currying.

b) Function bundled along with it's lexical scope is closure.

c) Closures are capable of not only reading, but also manipulating the variables of their outer functions.

d) A closure cannot access the variables of the outer function.

SOLUTION DESCRIPTION:

Further Readings:

http://pat-whitrock.github.io/blog/2014/04/29/practical-uses-for-closures/

(Guess the Output)

28. Guess the output of the following code.

```
function greet(user) {
    var user = "Dear " + user;
    return function(greeting="Hello! ") {
        console.log(greeting+user);
    }
}

var sayHello = greet("Saloni");
sayHello();
```

a) Uncaught ReferenceError: greet is not defined

b) Hello! Dear Saloni

c) Hello! Dear undefined

d) undefined

SOLUTION DESCRIPTION:

When variable sayHello is assigned the function greet, it return another function. In outer scope user is updated as `Dear ${user}`. As it is a closure, 'Hello! Dear Saloni' gets printed.

(Guess the Output)

29. What will be the output of the following code?

```
function foo() {
    var a = 8;
    function bar() {
        console.log(a);
    }
    var a = a+2;
    return bar;
}
var fun = foo();
fun();
```

a) 8

b) 10

c) undefined

d) error

(Modify Items in cart)

30. What gets logged to the console?

```
function cart() {
    let items = 0;
    return {
       addItem: function () {
            items++;
        },
        getItem: function () {
            return items;
        }
    };
}

const closure = cart();
closure.addItem();
closure.addItem();
closure.addItem();
console.log(closure.getItem());
```

a) 1   b) 2   **c) 3**   d) undefined

****