1. Which of the following was NOT a newly added element in HTML5?
a. article
b. audio
c. nav
d. frameset

Ans - d. frameset

2. Which element may be used within content to represent material that is tangential?
a. aside
b. cite
c. article
d. class

Ans – a. aside

3. In HTML Audio/Video DOM, ____ sets or returns the CORS settings of the audio/video.
a. autoplay
b. buffered
c. preload
d. controller

Ans - b. buffered

4. What is the use of “defer” attribute?
a. It defers rendering of HTML page.
b. It defers script execution until the page has been rendered.
c. It defers rendering of CSS attributes
d. It request for the script asynchronously.

Ans - b. It defers script execution until the page has been rendered.

5. A _____ is used to define a special state of an element.
a. pseudo-tag
b. pseudo-element
c. pseudo-class
d. pseudo-id

Ans - c. pseudo-class

6. Which of the following CSS property can be used to provide the flex-direction and flex-wrap properties?
a. flex
b. flex-flow
c. flex-wrap
d. all of the mentioned

Ans - b. flex-flow

7. As a general rule, properties in CSS inherit from _____ elements
a. child to parent
b. parent to child
c. grandparents to parents d. None of the above

c. Lexical d. Sequential

Ans - b. parent to child

8. What kind of scoping does JavaScript use?
a. Literal
b. Segmentalc.
c. Lexical
d. Sequential

Ans - c. Lexical

9. Which function among the following lets to register a function to be invoked once?
a. setTimeout()
b. setTotalTime()
c. setInterval()
d. setIntervalTime()

Ans - a. setTimeout()

10. Which property is used to obtain browser vendor and version information?
a. modal
b. version
c. browser
d. navigator

Ans -d. navigator

11. Consider the two functions below, Will the return the same thing? Why or why not?

```
function foo1()
{
   return {
         bar: "hello"
   };
}

function foo2()
{
   return
   {
         bar: "hello"
   };
}
```

Surprisingly, these two functions will not return the same thing. Rather:

```
console.log("foo1 returns:");
console.log(foo1());
console.log("foo2 returns:");
console.log(foo2());
```

will yield:

```
foo1 returns:
Object {bar: "hello"}
foo2 returns:
undefined
```

Explanation:- Not only is this surprising, but what makes this particularly gnarly is that foo2() returns undefined without any error being thrown.
The reason for this has to do with the fact that semicolons are technically optional in JavaScript (although omitting them is generally really bad form). As a result, when the line containing the return statement (with nothing else on the line) is encountered in foo2(), a semicolon is automatically inserted immediately after the return statement.
No error is thrown since the remainder of the code is perfectly valid, even though it doesn't ever get invoked or do anything (it is simply an unused code block that defines a property bar which is equal to the string "hello").
This behavior also argues for following the convention of placing an opening curly brace at the end of a line in JavaScript, rather than on the beginning of a new line. As shown here, this becomes more than just a stylistic preference in JavaScript.

12. What is the output of the following code snippet? Also explain the reason behind the result.

```
var num = 0
    console.log(num++)
    console.log(++num)
    console.log(num)
```

Ans = it will print 0
2
2
As num ++ will add 1 after printing
hence num == 0 printed
In second ++num ; 1 will be added before printing hence num == 2 printed
Num's value is 2 so, lastly it printed 3.

13. Find and explain the output of the following:

```
for(let i = 1; i<5; i++){
        if(i === 3) continue;
        console.log(i);}
```

Ans = It will print
1
2
4
As when i === 3 the function does not reach to console and got continue.

14. Find and explain the output of the provided code snippet. console.log(typeof typeof 1)

Ans =

It will return string

Reason: typeof returns a string, of the type of the value you provided, When you check the value returned by typeof, it will be in string form, like:

'object', 'function', 'string' etc.
And you are checking the typeof "object", that's why it returned string.

15. What is a "closure" in JavaScript? Provide an example.

Ans =

Closure means that an inner function always has access to the vars and parameters of its outer function, even after the outer function has returned.

Inner function can access variables and parameters of an outer function (however, cannot access arguments object of outer function). Consider the following example.

```
function OuterFunction() {

    var outerVariable = 1;

    function InnerFunction() {
        alert(outerVariable);
    }

    InnerFunction();
}
```

In the above example, InnerFunction() can access outerVariable.

Now, as per the definition above, InnerFunction() can access outerVariable even if it will be executed separately. Consider the following example.

Example: Closure

```
function OuterFunction() {

    var outerVariable = 100;

    function InnerFunction() {
        alert(outerVariable);
    }

    return InnerFunction;
}
var innerFunc = OuterFunction();

innerFunc(); // 100
```

In the above example, return InnerFunction; returns InnerFunction from OuterFunction when you call OuterFunction(). A variable innerFunc reference the InnerFunction() only, not the OuterFunction(). So now, when you call innerFunc(), it can still access outerVariable which is declared in OuterFunction(). This is called Closure.

16. What is “hoisting” in JavaScript? Write the code transformation after the 1 st pass of JavaScript for the following
code snippet.

Hoisting in javascript - Hoisting is JavaScript's default behavior of moving all declarations to the top of the current scope (to the top of the current script or the current function).

```
Var result = concat(first_name, last_name);
console.log(result);
function concat(x,y){
return x+y;
}
var first_name = “John”;
var last_name = “Doe”;
```

17. Differentiate between callback functions and anonymous function. Give an example of both.

Ans =

Callback and Anonymous functions are used for passing to other functions like Array.prototype.map() for them to use. It's a way to pass logic just as you would pass an object. Consider the example code below;

```
var Arr = [2,3,4,5]
# Callback function
function myCallback(x){
  return x+1;
}
console.log(Arr.map(myCallback));
```

From the code sample above, our callback function is myCallback() , which simply takes an argument x and returns an increment of the argument x+1. We see that our Array.prototype.map() takes as argument our callback function.
We could also write it as:

```
var Arr = [2,3,4,5]

# Anonymous function goes directly
console.log(Arr.map(function(x){ return x+1}));
```

The code above will also do the job, it uses an anonymous function. I.E, a function without a name function(x){return x+1}. Anonymous functions as the name might imply are functions without names. The function is only refered to once, so a variable name doesn't need to be wasted on it.

The difference between anonymous and callback function is that;

1-Anonymous function doensn't need to be named, while calback functions are named

2-Anonymous functions can't be used anywhere outside the .map() function while callback functions are independent and can be used outside of .map() function above.

3-With anonymous functions, it is difficult to identify in call stacks, which makes debugging trickier. While with callback functions, there are identify in call stacks and easy to debug.

4-Callback function doesn't take in an argument when called within another function while anonymous function can take in arguments.

18. What is event bubbling and how to stop an event from bubbling?
It is a way of event propagation in the HTML DOM. It relates to the order in which events are propagated in nested elements.
In bubbling, when an event happens, the handler of the innermost element runs, then the parents, and then the further ancestor elements. In other words, events bubble up or propagate the DOM tree upwards.