question01 - d. frameset

question02 - a. aside

question03 - b. buffered

question04 - b. It defers script execution until the page has been rendered

question05 - c. pseudo-class

question06 - b. flex-flow

question07-b. parent to child

question08 - c. Lexical

question09- a. setTimeout()

question10 - d. navigator

#######Question11 : these two functions give different outputs
output of foo1 is Object {bar: "hello"}
and foo2 returns undefined
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.

######Question12:
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.

#####Question13:
It will print
1
2
4
As when i === 3 the function does not reach to console and got continue.

#######Question14:
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.

#####Question15:

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.

######Question16

######Question17:

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