Question 1:

The value of “this” **does not** depend on

A. How a function is being called
B. Where the function is being defined (correct answer)
C. If its an arrow function or a regular function

Question 2:

What is the output of the following question? (Assume running in a browser, non-strict mode)

```
function f2() {
  'use strict';
  return this;
}

f2();
```

A. window
B. undefined (correct answer)
C. f2()

Question 3:

True or false - Calling a bind method returns a function that can be called again at some point of time in the future

a. True (correct answer)
b. False

Question 4:

What is the output of the following output?

```
function C2() {
  this.a = 37;
  return {a: 38};
}

var o = new C2();
console.log(o.a);
```

a. 37
b. 38 (correct answer)
c. undefined

Question 5:

True or false - If “this” arg is passed to call, bind, or apply on invocation of an arrow function it will be ignored.

a. True (correct answer)
b. False

Question 6:

Call, apply and bind are found on the prototype of?

a. Object
b. Array
c. Function (correct answer)

Question 7:

If a piece of Javascript code is being run in “strict mode”, which of the following statements is **false**?

a. Using eval() and arguments object is not allowed (correct answer)
b. Using with() is not allowed
c. List of words like implements, private, protected (reserved for future ECMAScript versions) is turned into keywords

Question 8:

What changes should be made to the following function so that we can call it a pure function?

```
function doubleValues (array) {
    // the input is an array
    return array.map(no => no * 2);
}
```

a. We should change the parameter in the function to use the spread operator (...array)
b. We should use forEach instead of map
c. No changes needed (correct answer)

Question 9:

Given the following HTML code, how do we target “app”?

```
<div class=”app”>
</div>
```

a. document.getElementById(“app)
b. document.getElementByClassName(“app”)[0]
c. document.getElementsByClassName(“app”)[0] (correct answer)

Question 10:

Which of the following statements about classes is true?

a. Code inside classes is run in strict mode (correct answer)
b. Referencing classes before they are defined gives no errors
c. We can define more than one constructors inside a class to have constructor overloading

Question 11:

How would you change the font color of “hi1” to green? (Assume we have used the appropriate code to target it already and have it set to a variable called box1)

** hi1 **

a. box1.color = ‘green’;
b. box1.style.color = ‘green’; (correct answer)
c. The font color cannot be changed this way, we need to use CSS

Question 12:

Why shouldn't arrow functions be used as constructors and methods in a class?

a. Javascript does not allow arrow functions to be used inside a class
b. This binding is not set properly (correct answer)
c. There is no problem with using arrow functions as constructors and methods in the class.

Question 13:

Which of the following statements is false about static methods in javascript?

a. They are called using the following syntax - ClassName.staticMethodName
b. Static members are not directly accessible using this keyword from non-static methods
c. Static methods automatically pick up this binding from instances created out of the class (correct answer)

Question 14:

Which of the following statements is false about the inheritance and the prototype chain in Javascript?

A. Each object has a private property that holds a link to another object called its prototype
B. the [[Prototype]] can be accessed directly in your code using the dot notation (correct answer)
C. null has no prototype, and acts as the final link in this prototype chain

Question 15:

What is logged to the console?

```
const message = 'Hello, Planet!'
const object = {
 message: 'Hello, World!',
 getMessage() {
   const message = 'Hello, Earth!';
   return this.message;
 }
};
console.log(object.getMessage());
```

a. Hello, Earth!
b. Hello, World! (correct answer)
c. Hello, Planet!

Question 16:

What is the output logged to the console?

```
function Pet(name) {
 this.name = name;
 this.getName = () => this.name;
}
const cat = new Pet('Fluffy');
console.log(cat.getName());
const { getName } = cat;
console.log(getName());
```

a. Fluffy, Fluffy (correct answer)
b. Undefined, Fluffy
c. Fluffy, undefined

Question 17:

What is the output logged to the console?

```
const object = {
 who: 'World',
 greet() {
   return `Hello, ${this.who}`;
 },
 farewell: () => {
   return `Goodbye, ${this.who}`;
 }
```

```
};
console.log(object.greet());
console.log(object.farewell());
```

a. Hello, World ; Goodbye, World
b. Hello, undefined ; Goodbye, undefined
c. Hello, World ; Goodbye, undefined (correct answer)

Question 18:

In the bubbling phase of event propagation, which of the following statements is true?

a. Every parent that has an event handler defined, will have the associated callbacks triggered (correct answer)
b. Event bubbling cannot be stopped at any parent element, once it has been triggered by a child element
c. Every event in the browser bubbles up

Question 19:

What will be the output of the following piece of code, when we have clicked “p”?

```
<form>FORM
  <div>DIV
    <p>P</p>
  </div>
</form>
<script>
const form = // has form targetted
const div = // has div targetted
const p = // has p targetted

p.addEventListener(‘click’, function() {
    console.log(‘p’);
});
div.addEventListener(‘click’, function(event) {
    event.stopPropagation();
    console.log(‘div’);
});
form.addEventListener(‘click’, function() {
    console.log(form);
});
</script>
```

a. p, div (correct answer)
b. p, div, form
c. p

Question 20:

Which of the following statements is false about eventListeners?

a. Event listeners can be unattached from elements
b. We have access to the event object inside the callback function
c. Event listeners are capable of recognizing what type of event is being handled by it (correct answer)