FRONT END – PART 2
CONSTRUCTORS AND PROTOTYPES

(Depending Factors)

1. The value of the 'this' keyword inside a function depends on which of the following factors? (multiple options correct)

a) The execution context of the function.
b) Where the function is declared.
c) How the function is called.
d) The function definition.

(Function Call Using 'new')

2. Determine the output and the most appropriate reason for it.

```
function Student(name){
    this.name = name ;
    console.log(this) ;
}
var student1 = new Student("Raj") ;
console.log(student1.name);
```

Note: The window object also has a property 'name' with a value equal to the empty string (' ').

a) An empty string as student1.name is being called in the global context.

b) Undefined as Student() function is not returning anything.

c) Raj as student1 is created using the 'new' keyword, which refers to a newly created object with property 'name' set to Raj.

d) An empty string as student1 is being created in the global context.

(Use of 'new')

3. What is the 'new' keyword primarily used for?

a) Creating a new function prototype.

b) Creating a new instance of an object from a function.

c) Setting the value of 'this' inside a function's execution context.

d) Setting the value of 'this' inside the function definition.

(Output - Explicit Binding)

4. Determine the output and its reason.

```
const CN = {
   name: "Coding Ninjas"
};

function print(){
   console.log(this) ;
}

print.bind(CN)() ;
```

Note: The window object also has a property ‘name’ with a value equal to the empty string (‘ ’).

a) The window object will be printed as the function is called in the global scope.

b) The window object will be printed as the CN object is not hardly bound with the execution context of print().

c) The CN object will be printed as it is hard bound with the CN object using the print() function.

d) The CN object will be printed as it is hard bound with the current execution context of the function.

(Explicit Binding Examples)

5. Look at the code given and find which of the following is an example of an explicit binding rule? Assume all function calls are being made in the global scope. (multiple options correct)

```
var joe = {
   name: "Joe"
};

function print(){
   console.log(this) ;
}
```

a) joe.print();

b) print.apply(joe);

c) window.print.call(joe);

d) print(joe) ;

e) print.bind(joe)() ;

(Compare the Outputs)

6. Look at the following code snippet and select the correct output.

```
var joe = {
    name: "Joe",
    hello: function (){
       console.log("hi, I am " + this.name) ;
    }
}

var globalHello1 = joe.hello.bind(joe) ;
var globalHello2 = joe.hello ;
globalHello1();
globalHello2.bind(joe)();
```

Note: window{--} represents the window object

a) ‘hi, I am Joe’ ‘hi, I am Joe’

b) Syntax Error

c) ‘hi, I am Joe’ window {--}

d) window {--} ‘hi, I am Joe’

(Function Calls)

7. Which of the following function calls will print the joe object? (multiple options correct)

```
var joe = {
    name: "Joe"
};

function print(){
    console.log(this);
}
```

a) print.bind(joe)(); b) joe.print();

c) print.bind(joe); d) print.call(joe);

e) print.apply(joe); f) print(joe);

(The ‘this’ Binding – i)

8. Look at the code snippet below and find the correct statement regarding the ‘this’ binding in the called function.

```
    var juliet = {
        name: "Juliet",
        hello: function(){
            console.log("Hi, I am " + this.name);
        }
    }
    var globalHello = juliet.hello;
    globalHello();
```

a) It’s explicit binding, as you can see explicitly that a function called using an object will set ‘this’ to the object itself.

b) It’s implicit binding as a function called using an object implicitly sets ‘this’ to the object.

c) It’s default binding as function is called in the global scope via no object.

d) It’s implicit binding as the function called has a reference to the juliet object.

(The ‘this’ Binding - ii)

9. Look at the code snippet below and find the correct statement regarding the ‘this’ binding in the called function.

```
    var juliet = {
        name: "Juliet",
        hello: function(){
            console.log("Hi, I am " + this.name);
        }
    }
    juliet.hello();
```

a) It’s explicit binding, as you can see explicitly that a function called using an object will set ‘this’ to the object itself.

b) It’s implicit binding as a function called using an object implicitly sets ‘this’ to the object.

c) It’s default binding as function is called in the global scope.

d) It’s explicit binding as the value of ‘this’ inside the hello() function is set to the juliet object.

(‘this’ in Nested Function)

10. Determine the output and the reason for it.

```
const billy = {
  name: "Billy Jol",
  outer: function() {
       function inner(){
       console.log(this);
     }
     inner();
  }
}
billy.outer();
```

a) The billy object as it gets implicitly binded with the outer() function call, hence with the inner() call as well.

b) The billy object as it gets explicitly binded with the outer() function call, hence with the inner() call as well.

c) The window object as inner() gets called with default binding.

d) The window object as inner() gets called with implicit binding.

(Complex function call)

11. Assume there is an object ‘joe’ with a property hello. This property is a function that console logs the value of this. Look at the code snippet and determine the output in the console.

```
new (joe.hello.bind(joe))();
```

a) hello{}

b) hello{} hello{}

c) joe{}

d) joe{} joe{}

()

12. What will the below statements print on the console?

```
function User(name) {
this.name = name;
this.isAdmin = false;
return;
}
var user = User("Julie");
console.log(user);
```

a) Object { name: "Julie", isAdmin: false }

b) Object { isAdmin: false }

c) undefined

d) Error - name is not defined

(Function Constructor)

13. What will be the output of the code below?

```
function User(name) {
  this.isAdmin = false;
}
```

```
var user = new User("James");
console.log(user);
```

a) User { name: "James", isAdmin: false }

b) User{ isAdmin: false }

c) Error

d) Error - name is not defined

(Function Objects)

14. What should be printed on the console?

```
var obj = {};
function A() { return obj; }
function B() { return obj; }
console.log( new A() == new B() );
```

a) True

b) False

(Comparing the Function Property)

15. Suppose you have a constructor function with name and getThis() properties. The getThis() function simply console logs the value of this. Now you create two objects, obj1 and obj2, using the constructor function by passing the same name as arguments. What will the following line of code print?

```
console.log(obj1.getThis === obj2.getThis);
```

a) True, as both sides have the same function definition.

b) True, as both sides refer to the same function.

c) False, as both sides have the same value of 'this' inside their execution context.

d) False, as both sides refer to two different copies of the same function.

(Function Prototype)

16. Look at the following function definitions. Determine for which of the following a prototype will be made?

```
1. function func1(){
       var a = "I am func1";
       console.log(a);
   }

2. function func2(b) {
       this.b = b;
       return b;
   }

3. function func3(){
         this.c = "I am func3";
         this.print = function(){
            console.log(this);
         }
    }
```

a) Both 1 and 2. b) Only 3.

c) Both 2 and 3. d) All 1, 2 and 3.

(Reference to Function Prototype)

17. Look at the following code snippet and determine if both a and b have an internal reference to the same function prototype or not.

```
function Person(name){
    this.name = name;
    this.print = function(){
        console.log(this);
    }
}
var a = new Person("A");
var b = new Person("B");
```

a) Same, as a constructor function has only one prototype which is created.

b) Same, as a constructor function has two same prototypes which are created for both the objects

c) Different, as a constructor function has two different prototypes created for both the objects.

d) Different, as a and b are two different instances of an object.

(Function Prototype for Every Function)

18. Are prototypes created for every function?

a) True b) False

(prototype.getName)

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

```
function Person(name) {
this.name = name;
}
var p1 = new Person("Joy");
var p2 = new Person("Julie");
Person.prototype.getName = function() { return this.name };
console.log(p1.getName() + " is friend with " + p2.getName());
```

a) undefined is friend with undefined b) Joy is friend with Julie

c) error is shown d) none of the above

(getName())

20. What will be the output for the following code?

```
function Person(name) {
  this.name = name;
}
var p1 = new Person("Joy");
var p2 = new Person("Julie");

Person.prototype.getName = function() { return this.name };
```

```
p1.getName = function() {return ‘John’};
console.log(p1.getName() + " is friend with " + p2.getName());
p2.getName = function() {return ‘Jonson’};
```

a) undefined is friend with undefined

b) John is friend with Jonson

c) Joy is friend with Julie

d) John is friend with Julie

(getPrototypeOf())

21. What will be the output of following code in console?

```
var a = new Boolean();
console.log(Object.getPrototypeOf(a));
```

a) Boolean

b) Boolean {false, constructor: ƒ, toString: ƒ, valueOf: ƒ}

c) undefined

d) None of the Above

(hasOwnProperty())

22. What will be the output of following code snippet?

```
function Person(age) {
this.age = age;
}
var p1 = new Person(9);
console.log(Person.hasOwnProperty("name"));
```

a) True

b) False

(The Prototype Chain – i)

23. Look at the following code and determine the output.

```
function A(){
     this.name =  "A";
}

Object.prototype.color = "red";
var d = new A();

console.log(d.color) ;
```

a) red as d looks up in the prototype chain up to Object.prototype to get the value.

b) red as d directly look at Object.prototype to get the value.

c) undefined as color is not a defined property of A.

d) undefined as color is not a defined property of any object in the prototype chain.

(The Prototype Chain – ii)

24. Look at the following code and determine the output and the most appropriate reason.

```
function A(){
    this.name = "A";
    this.color = "blue";
}
function B(){
    this.name = "B";
}
function C(){
    this.name =  "C";
}

let obj1 = new A() ;
let obj2 = new B() ;
let obj3 = new Object();
let obj4 = new C();

A.prototype.color = "red" ;
B.prototype.color = "red" ;
Object.prototype.color = "pink" ;

console.log(obj1.color, obj2.color, obj4.color, obj3.color);
```

Choose the correct output:
1. pink pink pink pink
2. red red pink pink
3. blue red pink pink

a) 1 as Object.prototype is the highest in the prototype chain; hence all will get pink color assigned.

b) 2 as obj1 and obj2 are not made using the Object() function; hence pink color doesn't get assigned to them.

c) 3 as obj1 and obj2 are not made using the Object() function; hence pink color doesn't get assigned to them, and obj1 has the color blue, so it doesn't look up in the prototype chain.

d) 3 as obj1 has the color blue, so it doesn't look up in the prototype chain, obj2 has no property name 'color', so it looks up in the prototype chain and gets a value red.

(Object())

25. What is the output of following code?

```
const object1 = new Object();
object1.property1 = 42;
console.log(Object());
```

a) Object

b) Object { }

c) Object { property1: 42 }

d) None of the Above

(object)

26. What happens if you run the following code snippet in console?

```
const object1 = new Object();
object1.property1 = 42;
console.log(object1);
```

a) Object b) Object { }

c) Object { property1: 42 } d) None of the Above

(Constructor)

27. What happens if you do not add a constructor to a class?

a) You would not be able to create objects b) It will work as a normal function

c) Default empty constructor will be added automatically d) Error will be thrown

(typeof)

28. What will the following code produce on the console?

```
class Person {
   constructor(name) { this.name = name; }
}
console.log(typeof( Person));
```

a) class b) function

c) constructor d) none of the above

(Object.prototype.constructor)

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

```
class Person {
  constructor(name) { this.name = name; }
}
console.log(Person === Person.prototype.constructor);
```

a) True b) False

(Class Expression)

30. What will the following code print in console?

```
var Person = class {
  constructor() {}
  sayHello() {
    return 'Hello!';
  }
};
var instance = new Person();
console.log(instance.sayHello()+" "+Person.name);
```

a) Hello! Name b) Hello! Undefined

c) instance.sayHello Person.name d) Hello! Person

(Parent Class Function)

31. Which keyword is used to call the base/parent class functions from the child functions?

a) base b) upper c) super d) parent

(Inheritance in JavaScript)

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

```
class Vehicle{
constructor ( doors, wheels){
    this.doors = doors;
    this.wheels = wheels;
}
}
class bus extends Vehicle{
    constructor(windows){
        this.windows = windows;
    }
}
var b = new bus(10);
console.log(bus.windows);
```

a) 10

b) Undefined

c) Uncaught ReferenceError: Must call super constructor in derived class before accessing 'this' or returning from derived constructor

d) None of the Above

ASSIGNMENT

(This keyword)

33. What will the below set of lines print on the browser console?

```
function bike() {
  console.log(this.name);
 }
 var name = "Ninja";
 bike();
```

a) Ninja b) undefined

c) Error - name is not defined d) Error - this is undefined

(Strict mode)

34. What will be printed on the console?

```
"use strict"
function bike() {
    console.log(this.name);
}
var name = "Ninja";
```

```
bike();
```

a) Ninja

b) undefined

c) Error because this is undefined

d) Error because name is undefined

(Object Constructor)

35. What will be the output of the code below?

```
function Abc() {
"use strict"
this.a = 25;
return {a: 35, b: 44};
}
var obj = new Abc();
console.log(obj.a, obj.b);
```

a) 35 44

b) 25 undefined

c) 25 44

d) Error - b is not defined

(What is the Output)

36. What will the below statements print on the console?

```
function User(name) {
this.isAdmin = false;
return name;
}
var user = User("Jack");
console.log(user);
```

a) Object { name: "Jack", isAdmin: false }

b) Object { isAdmin: false }

c) Object { name: "Jack" }

d) "Jack"

(hasOwnProperty())

37. Suppose there is a class 'Person' and an object 'person' is created as follows -

```
class Person = {
constructor(name) { this.name = name; }
…… Other methods …..
}
var person = new Person("James");
```

The below statement returns true or false if a property is present in the person object -

```
person.hasOwnProperty(property)
```

Where does this function come from?

a) From Object’s prototype

b) We must have declared it inside the class

c) None of the above mentioned reason is valid

(Which statement will give error)

38. Which of the following statements will produce an error with the following code?

```
class Person {
constructor(name) {
    this.name = name;
}
get name() {
    return this._name;
}
set name(value) {
    this._name = value;
}
}
var person = new Person("James");
```

a) person.name b) person.name = "Jones"

c) person.name() d) person._name = "Jones"

(Call and Apply)

39. What will the below statements print in the console?

```
var ninja = {
name: 'Ninja',
getName: function() {
    var name = this.name;
    return name;
  }
};
var funcName = function(snack, hobby) {
console.log(this.getName() + ' loves ' + snack + ' and ' + hobby);
};
funcName.call(ninja,'sushi', 'algorithms');
funcName.apply(ninja,['sushi', 'algorithms']);
```

a) Undefined loves undefined and undefined Undefined loves sushi and algorithms

b) Ninja loves sushi and algorithms Ninja loves sushi and algorithms

c) Ninja loves undefined and undefined Undefined loves undefined and undefined

d) Ninja loves undefined and undefined Undefined loves undefined and undefined

e) None of the Above

(Getter Method)

40. Which keyword is used to create a getter method in class?

a) get b) set c) getter d) setter

(Output)

41. What willl be the output of the following code?

```
function MyClass() {}

const obj1 = new MyClass();
const obj2 = MyClass();

console.log(obj1);
console.log(obj2);
```

a) MyClass{} undefined b) MyClass undefined c) undefined MyClass() d) undefined

(Guess the Output)

42. What is the output of the code given below?

```
class MyClass {
   constructor(x) {
       console.log("constructor is called!");
       this.x=x;
   }
}

const obj = new MyClass();
console.log(obj.x);
```

a) undefined undefined b) constructor is called! undefined

c) constructor is called! d) constructor is called! constructor is called!

****