What will happen when the following code is executed?

```
if (true) {
var x = 6;
let y=7;
const z=9;
}
console.log(x);
console.log(y);
console.log(z);
//The Output will be -
6
y is not defined
z is not defined
```

What will happen when the following code is executed?

```
const a = 10;
const b = 20;
b = a || 10;
b = a || 12;
console.log(b);
//The Output will be -
Uncaught TypeError : Assignment to constant variable.
```

What will happen when the following code is executed?

```
var x = 5;
console.log(x);
if (true) {
var x = 6;
console.log(x);
}
console.log(x);
//The Output will be -
5
6
6
```

What will happen when the following code is executed?

```
var x = 5;
console.log(x);
if (false) {
var x = 6;
console.log(x);
}
console.log(x);
//The Output will be -
5
5
```

What will happen when the following code is executed?

```
var x = 5;
function a() {
var x = 6;
return x;
}
console.log(x);
console.log(a());
//The Output will be -
5
6
```

What will happen when the following code is executed?

```
var x = 5;
function a() {
x = 6;
return x;
}
console.log(x);
console.log(a());
//The Output will be -
5
6
```

What will happen when the following code is executed?

```
var x = 5;
```

```
function a() {
let x = 6;
return x;
}
console.log(x);
console.log(a());
//The Output will be -
5
6
```

What will happen when the following code is executed?

```
let x = 5;
function a() {
let x = 6;
return x;
}
console.log(x);
console.log(a());
//The Output will be -
Uncaught SyntaxError: Iden_fier 'x' has already been declared
at<anonymous>: 1: 1
```

What will happen when the following code is executed?

```
const x = 5;
function a() {
let x = 6;
return x;
}
console.log(x);
console.log(a());
//The Output will be -
Uncaught SyntaxError: Iden_fier 'x' has already been declared
at<anonymous>: 1: 1
```

What will happen when the following code is executed?

```
const x = 5;
function a() {
```

```
x = 6;
return x;
}
console.log(x);
console.log(a());
//The Output will be -
Uncaught SyntaxError: Iden_fier 'x' has already been declared
at<anonymous>: 1: 1
```

What will happen when the following code is executed?

```
const x = 5;
function a() {
const x = 6;
return x;
}
console.log(x);
console.log(a());
//The Output will be -
Uncaught SyntaxError: Iden_fier 'x' has already been declared
at<anonymous>: 1: 1
```

What will happen when the following code is executed?

```
const x = 5;
function a() {
var x = 6;
return x;
}
console.log(x);
console.log(a());
//The Output will be -
Uncaught SyntaxError: Iden_fier 'x' has already been declared
at<anonymous>: 1: 1
```

What will happen when the following code is executed?

```
var x = 5;
function a() {
const x = 6;
```

```
return x;
}
console.log(x);
console.log(a());
//The Output will be -
5
6
```

What will happen when the following code is executed?

```
let x = 5;
function a() {
var x = 6;
return x;
}
console.log(x);
console.log(a());
//The Output will be -
Uncaught SyntaxError: Iden_fier 'x' has already been declared
at<anonymous>: 1: 1
```

What will happen when the following code is executed?

```
var x = 5;
function a() {
var x = 6;
return x;
}
x;
a();
//The Output will be -
6
```

What will happen when the following code is executed?

```
var x = 5;
function a() {
var x = 6;
return x;
}
```

```
x = a();
//The Output will be -
6
```

What will happen when the following code is executed?

```
var x = 5;
function a() {s
var x = 6;
return x;
}
x/
/The Output will be -
5
```

What will happen when the following code is executed?

```
var x = 5;
function a() {
var x = 6;
return x;
}
console.log(x);
console.log(a());
//The Output will be -
5
6
```

What will happen when the following code is executed?

```
var x = 5;
function a() {
var x = 6;
}
console.log(x);
console.log(a());
//The Output will be -
5
```

What will happen when the following code is executed?

```
var x = 5;
function a() {
let x = 6; return x;
}
console.log(x);
console.log(a());
//The Output will be -
5
```

What will happen when the following code is executed?

```
var x = { num: 5 };
function a() {
  delete x.num;
  return x.num;
}
console.log(a());
//The Output will be -
Undefined
```

What will happen when the following code is executed?

```
var x = 10;
function a() {
  delete x;
  return x;
}
console.log(a());
//The Output will be -
10 – Delete won't work because it is not an object
```

What will happen when the following code is executed?

```
(function() {
  var a = b = 3;
}
console.log(a);
console.log(b);
//The Output will be -
Uncaught ReferenceError: a is not defined
```

3
var a = b = 3; is actually shorthand for: b = 3;var a = b;

What will happen when the following code is executed?

```
var myObject = {
  foo: "bar",
  func: function () {
    var self = this;
    console.log(this.foo);
    console.log(self.foo);
    (function () {
      console.log(this.foo);
      console.log(self.foo);
    })();
  }
};
myObject.func();
//The Output will be –
bar
bar
undefined
bar
```

What will happen when the following code is executed?

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

function foo2() {
  return
  {
    bar: "hello";
  }
}
console.log(foo1());
```

```
console.log(foo2());
//The Output will be –
{ bar: "hello" }
Undefined – because return statement should be in same line
```

What will happen when the following code is executed?

```
console.log(0.1 + 0.2);
console.log(0.1 + 0.2 == 0.3);
//The Output will be –
0.30000000000000004
False
```

What will happen when the following code is executed?

```
(function () {
   console.log(1);
   setTimeout(function (){console.log(2)}, 1000);
   setTimeout(function (){console.log(3)}, 0);
   console.log(4);
})();
//The Output will be –
1
4
3
4
```

What will happen when the following code is executed?

```
console.log(1 + "2" + "2");  //122
console.log(1 + +"2" + "2");  // 32
console.log(1 + -"1" + "2");  // 02
console.log(+"1" + "1" + "2");  //112
console.log("A" - "B" + "2");  //NAN2
console.log("A" - "B" + 2);  // NAN
console.log(1 + undefined) );  // NAN
console.log(1 + null); // 1
console.log(1 + true); //2
console.log(1 + false);  //1
console.log(true + false); //1
```

```
console.log(true + 'false'); //truefalse
console.log(false + false); //0
console.log(3>2>1); //false
console.log(1<2<3); //true
console.log((0 || 1)); //1
console.log((1 || 2));//1
console.log((0 && 1)); //0
console.log((1 && 2));//2
console.log(false == '0') // true
console.log(false === '0') // false
```

What will happen when the following code is executed?

```
console.log(
 (function f(n){
  return ((n > 1) ? n * f(n-1) : n)
 })(10)
);
//The Output will be –
3628800 – factorial of 10
```

What will happen when the following code is executed?

```
for (let i = 0; i < 5; i++) {
setTimeout(
  function () {
    console.log(i);
  }, i * 1000 );
}
//The Output will be –
0
1
2
3
4
```

What will happen when the following code is executed?

```
for (var i = 0; i < 5; i++) {
setTimeout(
```

```
    function () {
      console.log(i);
    }, 1000 );
}
//The Output will be –
5 will print 5 times
```

What will happen when the following code is executed?

```
var b = 1;
function outer() {
  var b = 2;
  function inner() {
    b++;
    var b = 8;
    console.log(b);
  }
  inner();
}
outer();
//The Output will be –
8
```

What will happen when the following code is executed?

```
const a = [{ type: "a" }, { type: "b" }];
const b = [...a];
a[0].type = "c";
console.log(a[0].type);
console.log(b[0].type);
//The Output will be – Shallow Copy
C
C
```

What will happen when the following code is executed?

```
function fun() {
  var abc = "test";
  function abc() {
    return "hello";
```

```
  }
  return abc();
}
console.log(fun());
//The Output will be
Uncaught TypeError: abc is not a function
```

What will happen when the following code is executed?

```
function dump() {
  console.log(a);
  console.log(b);
  var a = 10;
  let b = 6;
}
dump();
//The Output will be
Undefined
Uncaught ReferenceError: Cannot access 'b' before initialization
```

What will happen when the following code is executed?

```
function dump() {
 a = 10;
console.log(a);
}
dump();
//The Output will be
10
```

What will happen when the following code is executed?

```
function dump() {
  var a = 10;
  console.log(a);
}
var a = 22;
console.log(a);
//The Output will be
22
```

What will happen when the following code is executed?

```
var x = 100;
var obj = {
  x: 43,
  y: "test",
  z: function () {
    return this.x;
  }
};
console.log(obj.z());
//The Output will be
43
```

What will happen when the following code is executed?

```
var x = 0;
var y = 23;
if (x) {
  console.log(x);
}
if (y) {
  console.log(y);
}
//The Output will be
23
```

What will happen when the following code is executed?

```
console.log(sum(10, 20));
console.log(diff(10, 20));
function sum(a, b) {
  return a + b;
}
let diff = function (x, y) {
  return x - y;
};
//The Output will be
30
```

Uncaught ReferenceError: Cannot access 'diff' before initialization

What will happen when the following code is executed?

```
var person = {
  age: 43,
  grow:  ()=> {
    this.age++;
  }
};
person.grow();
console.log(person.age);
//The Output will be
43
```

What will happen when the following code is executed?

```
var person = {
  age: 43,
  grow: function () {
    this.age++;
  }
};
person.grow();
console.log(person.age);
//The Output will be
44
```

What will happen when the following code is executed?

```
let a = 10;
var a = 20;
console.log(a);
//The Output will be
```

Uncaught SyntaxError: Identifier 'a' has already been declared

What will happen when the following code is executed?

```
var p = 3;
var b = p++;
var c = ++p;
```

```
console.log(p, b, c);
//The Output will be
5, 3, 5
```

What will happen when the following code is executed?

```
for (let i = 0; i < 100; i++) {
  if (i === 6) {
    break;
  }
  console.log(i);
}
console.log(i);
//The Output will be
0 1 2 3 4 5
Uncaught ReferenceError: i is not defined
```

Guess the outputs of the following codes:

// Code 1:

```
function func1(){
  setTimeout(()=>{
    console.log(x);
    console.log(y);
  },3000);
  var x = 2;
  let y = 12;
}
func1();
```

Output: Outputs 2 and 12 . Since, even though let variables are not hoisted, due to async nature of javascript, the complete function code runs before the setTimeout function. Therefore, it has access to both x and y.

// Code 2:

```
function func2(){
  for(var i = 0; i < 3; i++){
    setTimeout(()=> console.log(i),2000);
}
}
```

func2();

Output: Outputs 3 , three times since variable declared with var keyword does not have block scope. Also, inside the for loop, the variable i is incremented first and then checked.

// Code 3:

```
(function(){
  setTimeout(()=> console.log(1),2000);
  console.log(2);
  setTimeout(()=> console.log(3),0);
  console.log(4);
})();
```

Output:

2
4
3
1 // After two seconds

Even though the second timeout function has a waiting time of zero seconds, the javascript engine always evaluates the setTimeout function using the Web API and therefore, the complete function executes before the setTimeout function can execute.

Guess the outputs of the following code:

// Code 1:

```
let x= {}, y = {name:"Ronny"},z = {name:"John"};
x[y] = {name:"Vivek"};
x[z] = {name:"Akki"};
console.log(x[y]);
```

Output: Output will be {name: “Akki”}.

Adding objects as properties of another object should be done carefully.

Writing x[y] = {name:”Vivek”} , is same as writing x[‘object Object’] = {name:”Vivek”} ,

While setting a property of an object, javascript coerces the parameter into a string.

Therefore, since y is an object, it will be converted to ‘object Object’.

Both x[y] and x[z] are referencing the same property.

// Code 2:

```
function runFunc(){
  console.log("1" + 1);
  console.log("A" - 1);
  console.log(2 + "-2" + "2");
  console.log("Hello" - "World" + 78);
  console.log("Hello"+ "78");
}
runFunc();
```

Output: Outputs in the following order:

11
Nan
2-22
NaN
Hello78

// Code 3:

```
let a = 0;
let b = false;
console.log((a == b));
console.log((a === b));
```

Output: Output in the following order due to equality coercion:

true
false

Guess the output of the following code:

```
var x = 23;
(function(){
  var x = 43;
  (function random(){
    x++;
    console.log(x);
    var x = 21;
  })();
})();
```

Output is NaN .

random() function has functional scope, since x is declared and hoisted in the functional scope.

Rewriting the random function will give a better idea about the output:

```
function random(){
  var x; // x is hoisted
  x++; // x is not a number since it is not initialized yet
  console.log(x); // Outputs NaN
  x = 21; // Initialization of x
}
```

Guess the outputs of the following code:

// Code 1

```
let hero = {
  powerLevel: 99,
  getPower(){
    return this.powerLevel;
  }
}
let getPower = hero.getPower;
let hero2 = {powerLevel:42};
console.log(getPower());
console.log(getPower.apply(hero2));
```

Output in the following order:

undefined

42

Reason - The first output is undefined since when the function is invoked, it is invoked referencing the global object:

window.getPower() = getPower();

// Code 2

```
const a = function(){
  console.log(this);
  const b = {
    func1: function(){
      console.log(this);
    }
```

```
  }
  const c = {
   func2: ()=>{
     console.log(this);
   }
  }
  b.func1();
  c.func2();
 }
 a();
```

Outputs in the following order:
global/window object
object "b"
global/window object
Since we are using arrow function inside func2, this keyword refers to the global object.

```
 // Code 3
  const b = {
  name:"Vivek",
  f: function(){
    var self = this;
    console.log(this.name);
    (function(){
      console.log(this.name);
      console.log(self.name);
    })();
  }
 }
 b.f();
```

Outputs in the following order:
"Vivek"
undefined
"Vivek"
Only in the IIFE inside the function f , the this keyword refers to the global/window object.

Guess the outputs of the following code:

****Note** - Code 2 and Code 3 require you to modify the code, instead of guessing the output.

// Code 1

```
(function(a){
  return (function(){
    console.log(a);
    a = 23;
  })()
})(45);
```

Output: 45 .

Even though a is defined in the outer function, due to closure the inner functions have access to it.

// Code 2

```
// Each time bigFunc is called, an array of size 700 is being created,
// Modify the code so that we don't create the same array again and again
function bigFunc(element){
  let newArray = new Array(700).fill('♥');
  return newArray[element];
}
console.log(bigFunc(599)); // Array is created
console.log(bigFunc(670)); // Array is created again
```

Output : This code can be modified by using closures,

```
function bigFunc(){
  let newArray = new Array(700).fill('♥');
  return (element) => newArray[element];
}
let getElement = bigFunc(); // Array is created only once
getElement(599);
getElement(670);
```

// Code 3

```
// The following code outputs 2 and 2 after waiting for one second
// Modify the code to output 0 and 1 after one second.
function randomFunc(){
  for(var i = 0; i < 2; i++){
```

```
   setTimeout(()=> console.log(i),1000);
  }
}
randomFunc();
```

Output: Can be modified in two ways:

Using let keyword:

```
function randomFunc(){
  for(let i = 0; i < 2; i++){
    setTimeout(()=> console.log(i),1000);
  }
}
randomFunc();
```

Using closure:

```
function randomFunc(){
  for(var i = 0; i < 2; i++){
  (function(i){
     setTimeout(()=>console.log(i),1000);
    })(i);
  }
}
randomFunc();
```