01. Write a function called "addNumbers" that takes two numbers as arguments and returns their sum. Call the function before it is declared to demonstrate hoisting.

```
console.log(addNumbers(5, 10)); // 15

function addNumbers(num1, num2) {
  return num1 + num2;
}
```

02. Write a function called "multiplyNumbers" that takes two numbers as arguments and returns their product. Use function expressions to define the function and call the function before it is declared to demonstrate hoisting.

```
console.log(multiplyNumbers(3, 4)); // ReferenceError: Cannot access
'multiplyNumbers' before initialization

const multiplyNumbers = function (num1, num2) {
  return num1 * num2;
};
```

03. Write a function that takes two numbers as arguments and returns their sum. Declare a variable inside the function using the var keyword and log its value to the console before it is assigned a value to demonstrate variable hoisting.

```
function addNumbers(num1, num2) {
  console.log(sum); // undefined
  var sum;
  sum = num1 + num2;
  return sum;
}

console.log(addNumbers(5, 10)); // 15
```

03. Write a function that takes two numbers as arguments and returns their sum. Declare a variable inside the function using the var keyword and log its value to the console before it is assigned a value to demonstrate variable hoisting.

```
function addNumbers(num1, num2) {
  console.log(sum); // undefined
  var sum;
  sum = num1 + num2;
  return sum;
}

console.log(addNumbers(5, 10)); // 15
```

04. Declare three variables, one using let, one using var, and one using const, all inside a block scope. Assign them values and log their values to the console before and after they are declared to demonstrate variable hoisting.

```
{
  console.log(x); // undefined due to hoisting
  console.log(y); // throws ReferenceError
  console.log(z); // throws ReferenceError

  var x = "PW";
  let y = "Skills";
  const z = "!";

  console.log(x); // "PW"
  console.log(y); // "Skills"
  console.log(z); // "!"
}
```

05. Declare a variable using let inside a block scope and attempt to log its value to the console before it is assigned a value to demonstrate the temporal dead zone.

```
{
  console.log(x); // throws ReferenceError

  let x = "hello";

  console.log(x); // "hello"
}
```