Question1: **Explore and explain the various methods in console function**

Method	Description
clear()	Clears the console
count()	Logs the number of times that this particular call to count() has been called
error()	Outputs an error message to the console
info()	Outputs an informational message to the console
log()	Outputs a message to the console
table()	Displays tabular data as a table
time()	Starts a timer (can track how long an operation takes)
timeEnd()	Stops a timer that was previously started by console.time()
trace()	Outputs a stack trace to the console
warn()	Outputs a warning message to the console

1. The **console.clear()** method clears the console.

The console.clear() method will also write a message in the console: "Console was cleared".

```
console.clear()
```

2. Writes to the console the number of times that particular console.count() is called.

```
console.count("myLabel");
console.count("myLabel");
```

3. The console.error() method writes an error message to the console. The console is useful for testing purposes.

```
var myArr = ["Orange", "Banana", "Mango", "Kiwi" ];
console.error(myArr)
```

4. The console.info() method writes a message to the console.

```
var myArr = ["Orange", "Banana", "Mango", "Kiwi" ];
console.info(myObj);
```

5. The console.log() method writes a message to the console.

```
var myArr = ["Orange", "Banana", "Mango", "Kiwi" ];
console.log(myArr);
```

Question2: Write the difference between var, let and const with code examples.

The Differences Between let and var

A var variable can be redeclared *and* updated.

A let variable be updated but *not* redeclared.
An example of trying to redeclare a let variable:

// In editor:<script>
let points = 50;
let points = 60;
</script>// In the console I get an error:

Uncaught Syntax Error: Identifier 'points' has already been declared

The Differences Between let and const

`const` variables cannot be updated. `let` variables are *made* to be updated.
// If I define the const variable:
const key = 'xyz123';// Then try to redeclare it:
key = 'xyz1234'// I get the following error:
Uncaught TypeError: Assignment to constant variable.

There is an interesting caveat to this, though. If I create a `const` variable that is an object, the attributes of that object can be updated.

// Creating my person object:
const person = {
 name: 'Joseph',
 age: 33
}// Calling person in the console:
person// It returns:
{name: "Joseph", age: 33}// If I then redeclare the age attribute:
person.age = 34// When I call it:
person// It returns:
{name: "Joseph", **age: 34**}

Question3: Write a brief intro on available data types in Javascript

There are eight basic data types in JavaScript

1. Number :

The *number* type represents both integer and floating point numbers.

There are many operations for numbers, e.g. multiplication `*`, division `/`, addition `+`, subtraction `-`, and so on.

Besides regular numbers, there are so-called "special numeric values" which also belong to this data type: `Infinity`, `-Infinity` and `NaN`.

- `Infinity` represents the mathematical Infinity ∞. It is a special value that's greater than any number.

We can get it as a result of division by zero:

```
n = 123;
n = 12.345;
```

2. BigInt :

`BigInt` type was recently added to the language to represent integers of arbitrary length.

A `BigInt` value is created by appending `n` to the end of an integer:

```
const bigInt = 1234567890123456789012345678901234567890n;
```

3. String:

A string in JavaScript must be surrounded by quotes.

In JavaScript, there are 3 types of quotes.

1. Double quotes: `"Hello"`.
2. Single quotes: `'Hello'`.
3. Backticks: `` `Hello` ``.

Double and single quotes are “simple” quotes. There’s practically no difference between them in JavaScript.

```
str = "Hello";
str2 = 'Single quotes are ok too';
phrase = `can embed another ${str}`;
```

4. Boolean (logical type) :

The boolean type has only two values: `true` and `false`.

This type is commonly used to store yes/no values: `true` means “yes, correct”, and `false` means “no, incorrect”.

```
let nameFieldChecked = true; // yes, name field is checked
let ageFieldChecked = false; // no, age field is not checked
```

5. The “null” value:

The special `null` value does not belong to any of the types described above.

It forms a separate type of its own which contains only the `null` value:

```
let age = null;
```

In JavaScript, `null` is not a “reference to a non-existing object” or a “null pointer” like in some other languages.

It’s just a special value which represents “nothing”, “empty” or “value unknown”.

The code above states that `age` is unknown.

6. The “undefined” value :

The special value `undefined` also stands apart. It makes a type of its own, just like `null`.

The meaning of `undefined` is “value is not assigned”.

If a variable is declared, but not assigned, then its value is `undefined`:

```
let age;
alert(age); // shows "undefined"
```

7. Objects:

The `object` type is special.

All other types are called “primitive” because their values can contain only a single thing (be it a string or a number or whatever). In contrast, objects are used to store collections of data and more complex entities.

Being that important, objects deserve a special treatment. We’ll deal with them later in the chapter Objects, after we learn more about primitives.

8. Symbols

The `symbol` type is used to create unique identifiers for objects. We have to mention it here for the sake of completeness, but also postpone the details till we know objects.