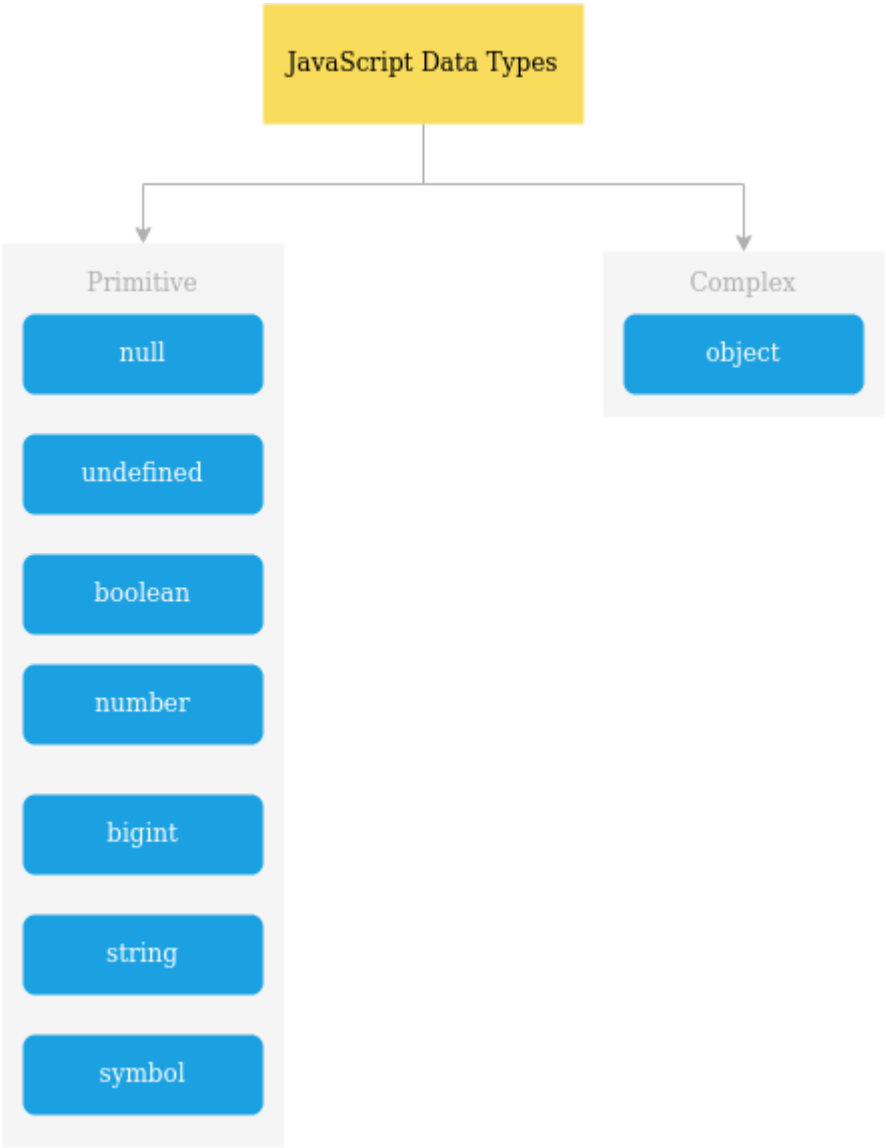
Data Type



JavaScript numeric separator

The numeric separator allows you to create a visual separation between groups of digits by using underscores (_) as separators.

```
const one = 3000000000;
const two = 3_000_000_000;
```

Use underscores (_) as the numeric separators to create a visual separation between groups of digits.

JavaScript exponentiation operator

Syntax : `Math.pow(base, exponent)`

```
let result = Math.pow(2,2);
console.log(result)

result = Math.pow(2,3);
console.log(result);
```

```
(2)

let result = 2 ** 2;
console.log(result);

result = 2 ** 3;
console.log(result);

(3)
```

```
let x = 2;

x **= 4;
console.log(x);
```

The exponentiation operator ** raises a number to the power of an exponent.

- The exponentiation operator `**` accepts values of the type `number` or `bigint`

Conditional Statements

The if Statement

The *if* statement is used to execute a block of code only if the specified condition evaluates to true. This is the simplest JavaScript's conditional statements and can be written like:

```
if(condition) {
        / Code to be executed
}
```

The if...else Statement

```
if(condition) {
    // Code to be executed if condition is true
} else {
    // Code to be executed if condition is false
}
```

```
 <script>
    var now = new Date();
    var dayOfWeek = now.getDay(); // Sunday - Saturday : 0 - 6

    if(dayOfWeek == 5) {
        document.write("Have a nice weekend!");
    } else {
        document.write("Have a nice day!");
    }
    </script>
```

if...else if...else Statement

```
<script>
    var now = new Date();
    var dayOfWeek = now.getDay(); // Sunday - Saturday : 0 - 6

    if(dayOfWeek == 5) {
        document.write("Have a nice weekend!");
    } else if(dayOfWeek == 0) {
        document.write("Have a nice Sunday!");
    } else {
        document.write("Have a nice day!");
    }
    </script>
```

(2)Ternary Operator

The ternary operator provides a shorthand way of writing the *if...else* statements. The ternary operator is represented by the question mark (`?`) symbol and it takes three operands: a condition to check, a result for `true`, and a result for `false`. Its basic syntax is:

var result = (condition) ? value1 : value2

```
<script>
    var userType;
    var age = 21;
    if(age < 18) {
        userType = 'Child';
    } else {
        userType = 'Adult';
    }
    document.write(userType);
    </script>
```

(i)

```
  <script>
    var age = 21;
    var userType = age < 18 ? 'Child' : 'Adult';
    document.write(userType);
    </script>
```

Switch...Case Statements

The *switch..case* statement is an alternative to the *if...else if...else* statement, which does almost the same thing. The *switch...case* statement tests a variable or expression against a series of values until it finds a match, and then executes the block of code corresponding to that match.

```
<script>
   var d = new Date();

   switch(d.getDay()) {
       case 0:
           document.write("Today is Sunday.");
           break;
       case 1:
           document.write("Today is Monday.");
           break;
       case 2:
           document.write("Today is Tuesday.");
           break;
       case 3:
           document.write("Today is Wednesday.");
           break;
       case 4:
           document.write("Today is Thursday.");
           break;
       case 5:
           document.write("Today is Friday.");
           break;
       case 6:
           document.write("Today is Saturday.");
           break;
       default:
```

```
            document.write("No information available for that
day.");
            break;
    }
    </script>
```

The getDay() method returns the weekday as a number from 0 and 6, where 0 represents Sunday.

(2)

```
<script>
    var d = new Date();

     switch(d.getDay()) {
          default:
               document.write("Looking forward to the weekend.");
               break;
          case 2:
               document.write("Today is Saturday.");
               break;
          case 0:
               document.write("Today is Sunday.");
     }
    </script>
```

(3)

```
<script>
    var d = new Date();

     switch(d.getDay()) {
          case 1:
```

```
        case 2:
        case 3:
        case 4:
        case 5:
            document.write("It is a weekday.");
            break;
        case 0:
        case 6:
            document.write("It is a weekend day.");
            break;
        default:
            document.write("Enjoy every day of your life.");
    }
</script>
```

Array

```
<script>
    var fruits = ["Apple", "Banana", "Mango", "Orange", "Papaya"];
    document.write(fruits.length);
    </script>
```

Using Loops

```
<script>
    var fruits = ["Apple", "Banana", "Mango", "Orange", "Papaya"];
    for(var i = 0; i < fruits.length; i++){
        document.write(fruits[i] + "<br>"); // Print array element
    }
    </script>
```

(3)

```
<script>
    var fruits = ["Apple", "Banana", "Mango", "Orange", "Papaya"];
     for(var i in fruits)  {
        document.write(fruits[i] + "<br>");
    }
    </script>
```

Adding New Elements to an Array

To add a new element at the end of an array, simply use the `push()` method,

```
<script>
    var colors = ["Red", "Green", "Blue"];
    colors.push("Yellow");
    document.write(colors + "<br>");
 Red,Green,Blue,Yellow
    document.write(colors.length);
    </script>
```

to add a new element at the beginning of an array use the unshift() method,

```
  <script>
    var colors = ["Red", "Green", "Blue"];
    colors.unshift("Yellow");
    document.write(colors + "<br>");
 Yellow,Red,Green,Blue
    document.write(colors.length);
    </script>
```

Mixed Example :

```
<script>
    var colors = ["Red", "Green", "Blue"];
```

```
    colors.push("Pink", "Voilet");
    colors.unshift("Yellow", "Grey");

    document.write(colors + "<br>");
Yellow,Grey,Red,Green,Blue,Pink,Voilet
    document.write(colors.length);
    </script>
</body>
```

Removing Elements from an Array

To remove the last element from an array you can use the `pop()` method. This method returns the value that was popped out.

```
<script>
    var colors = ["Red", "Green", "Blue"];
    var last = colors.pop();
    document.write(last + "<br>");
    document.write(colors.length);
    </script>
```

```
you can remove the first element from an array using the shift()
method. This method also returns the value that was shifted out.
```

```
<script>
    var colors = ["Red", "Green", "Blue"];
    var first = colors.shift();

    document.write(first + "<br>");
    document.write(colors.length);
    </script>
```

Adding or Removing Elements at Any Position

The `splice()` method is a very versatile array method that allows you to add or remove elements from any index, using the syntax `arr.splice(startIndex, deleteCount)`

```
<script>
    var colors = ["Red", "Green", "Blue"];
    var removed = colors.splice(0,1); // Remove the first element

    document.write(colors + "<br>"); // Prints: Green,Blue
    document.write(removed + "<br>"); // Prints: Red (one item
array)
    document.write(removed.length + "<br>"); // Prints: 1

    removed = colors.splice(1, 0, "Pink", "Yellow"); // Insert two
items at position one
    document.write(colors + "<br>"); // Prints:
Green,Pink,Yellow,Blue
    document.write(removed + "<br>"); // Empty array
    document.write(removed.length + "<br>"); // Prints: 0

    removed = colors.splice(1, 1, "Purple", "Voilet"); // Insert
two values, remove one
    document.write(colors + "<br>"); //Prints:
Green,Purple,Voilet,Yellow,Blue
    document.write(removed + "<br>"); // Prints: Pink (one item
array)
    document.write(removed.length); // Prints: 1
    </script>
```

Creating a String from an Array

There may be situations where you simply want to create a string by joining the elements of an array. To do this you can use the `join()` method. This method takes an optional parameter which is a separator string that is added in between each element. If you omit the separator, then JavaScript will use comma (`,`) by default.

```
<script>
    var colors = ["Red", "Green", "Blue"];
    document.write(colors.join() + "<br>");
    document.write(colors.join("") + "<br>");
    document.write(colors.join("-") + "<br>");
    document.write(colors.join(", "));
    </script>
```

Extracting a Portion of an Array

If you want to extract out a portion of an array (i.e. subarray) but keep the original array intact you can use the `slice()` method. This method takes 2 parameters: start index (index at which to begin extraction), and an optional end index (index before which to end extraction), like `arr.slice(startIndex, endIndex)`.

```
<script>
    var fruits = ["Apple", "Banana", "Mango", "Orange", "Papaya"];
    var subarr = fruits.slice(1, 3);
    document.write(subarr);
    </script>
```

Merging Two or More Arrays

The `concat()` method can be used to merge or combine two or more arrays. This method does not change the existing arrays, instead it returns a new array.

```
 <script>
    var pets = ["Cat", "Dog", "Parrot"];
    var wilds = ["Tiger", "Wolf", "Zebra"];
    var animals = pets.concat(wilds);
    document.write(animals);
    </script>
```

(2)

```
<script>
    var pets = ["Cat", "Dog", "Parrot"];
    var wilds = ["Tiger", "Wolf", "Zebra"];
    var bugs = ["Ant", "Bee"]
    var animals = pets.concat(wilds, bugs);
    document.write(animals);
    </script>
```

Loops in JavaScript

Loops are used to execute the same block of code again and again, as long as a certain condition is met. The basic idea behind a loop is to automate the repetitive tasks within a program to save the time and effort. JavaScript now supports five different types of loops:

- while — loops through a block of code as long as the condition specified evaluates to true.
- do…while — loops through a block of code once; then the condition is evaluated. If the condition is true, the statement is repeated as long as the specified condition is true.
- for — loops through a block of code until the counter reaches a specified number.
- for…in — loops through the properties of an object.

The while Loop

This is the simplest looping statement provided by JavaScript.

The while loop loops through a block of code as long as the specified condition evaluates to true. As soon as the condition fails, the loop is stopped. The generic syntax of the while loop is:

```
while(condition) {
  // Code to be executed
```

```
}
```

```
<script>
   var i = 1;
   while(i <= 5) {
      document.write("<p>The number is " + i + "</p>");
      i++;
   }
   </script>
```

The do...while Loop

The do-while loop is a variant of the while loop, which evaluates the condition at the end of each loop iteration. With a do-while loop the block of code executed once, and then the condition is evaluated, if the condition is true, the statement is repeated as long as the specified condition evaluated to is true.

```
do {
   // Code to be executed
}
while(condition);
```

```
<script>
   var i = 1;
   do {
      document.write("<p>The number is " + i + "</p>");
      i++;
   }
   while(i <= 5);
   </script>
```

Difference Between while and do...while Loop

The while loop differs from the do-while loop in one important way — with a while loop, the condition to be evaluated is tested at the beginning of each loop iteration, so if the conditional expression evaluates to false, the loop will never be executed.

With a do-while loop, on the other hand, the loop will always be executed once even if the conditional expression evaluates to false, because unlike the while loop, the condition is evaluated at the end of the loop iteration rather than the beginning.

The for Loop

The for loop repeats a block of code as long as a certain condition is met. It is typically used to execute a block of code for certain number of times. Its syntax is:

```
for(initialization; condition; increment) {
   // Code to be executed
}
```

The parameters of the for loop statement have following meanings:

• initialization — it is used to initialize the counter variables, and evaluated once unconditionally before the first execution of the body of the loop.

• condition — it is evaluated at the beginning of each iteration. If it evaluates to true, the loop statements execute. If it evaluates to false, the execution of the loop ends.

• increment — it updates the loop counter with a new value each time the loop runs.

The following example defines a loop that starts with i=1. The loop will continued until the value of variable i is less than or equal to 5. The variable i will increase by 1 each time the loop runs:

```
<script>
   for(var i=1; i<=5; i++) {
      document.write("<p>The number is " + i + "</p>");
   }
   </script>
```

```
 <script>
   var fruits = ["Apple", "Banana", "Mango", "Orange", "Papaya"];
    for(var i=0; i<fruits.length; i++) {
      document.write("<p>" + fruits[i] + "</p>");
   }
   </script>
```

The for...in Loop

The for-in loop is a special type of a loop that iterates over the properties of an object, or the elements of an array. The generic syntax of the for-in loop is:

```
for(variable in object) {
   // Code to be executed
}
```

```
<script>
   var person = {"name": "Clark", "surname": "Kent", "age": "36"};
  for(var prop in person) {
      document.write("<p>" + prop + " = " + person[prop] + "</p>");
   }
   </script>
```

(2)

```
<script>
   var fruits = ["Apple", "Banana", "Mango", "Orange", "Papaya"];
    for(var i in fruits) {
      document.write("<p>" + fruits[i] + "</p>");
   }
   </script>
</body>
```

Functions

A function is a group of statements that perform specific tasks and can be kept and maintained separately form main program. Functions provide a way to create reusable code packages which are more portable and easier to debug.

The declaration of a function start with the function keyword, followed by the name of the function you want to create, followed by parentheses i.e. () and finally place your function's code between curly brackets {}. Here's the basic syntax for declaring a function:

```
function functionName() {
   // Code to be executed
}
```

```
<script>
   function sayHello() {
      document.write("Hello, welcome to this website!");
   }
   sayHello();
   </script>
```

Parameters to Functions

```
<script>
   function displaySum(num1, num2) {
      var total = num1 + num2;
      document.write(total);
   }
   displaySum(6, 20);
   document.write("<br>");
   displaySum(-5, 17);
   </script>
```

(2)

```
<script>
   function sayHello(name='World'){
      return `Hello ${name}!`;
```

```
    }

    document.write(sayHello());
    document.write("<br>");
    document.write(sayHello('ram'));
    </script>
```

Returning Values from a Function

```
<script>

    function getSum(num1, num2) {
        var total = num1 + num2;
        return total;
    }


    document.write(getSum(6, 20) + "<br>");
    document.write(getSum(-5, 17));
    </script>
```