Unit4 Primitive Data Types, Arrays, Loops, and Conditions

At the end of this unit, the learner should be able to:

- correctly use conditional statement and loops in javascript languages
- common data structures in javascript language

I. Primitive data types

Any value that you use is of a certain type. In JavaScript, the following are just a few primitive data types:

- **Number**: This includes floating point numbers as well as integers. For example, these values are all numbers–1, 100, 3.14.
- **String**: These consist of any number of characters, for example, a, one, and one 2 three.
- **Boolean**: This can be either true or false.
- **Undefined**: When you try to access a variable that doesn't exist, you get the special value undefined. The same happens when you declare a variable without assigning a value to it yet. JavaScript initializes the variable behind the scenes with the value undefined. The undefined data type can only have one value–the special value undefined.
- **Null**: This is another special data type that can have only one value–the null value. It means no value, an empty value, or nothing. The difference with undefined is that if a variable has a null value, it's still defined; it just so happens that its value is nothing.

Any value that doesn't belong to one of the five primitive types listed here is an object. Even null is considered an object. In javascripts are as follows:

- Primitive (the five types listed previously)
- Non-primitive (objects).

II. Finding out the value type – the typeof operator

If you want to know the type of a variable or a value, you can use the special typeofoperator. This operator returns a string that represents the data type. The return values ofusing typeof are one of the following:

- number
- string
- boolean
- undefined
- object
- function

1. Numbers

The simplest number is an integer. If you assign 1 to a variable, and then use the typeofoperator, it returns the string number, as follows:

var n = 1235;
alert(typeof n);

The browser isdisplaying: number

Numbers can also be floating point (decimals), for example:

var n2 = 1.23;
alert(typeof n2);

2. Octal and hexadecimalnumbers

When a number starts with a 0, it's considered an octal number. For example, the octal 0377
is the decimal 255:

var a2=0377;

alert(typeof a2); // number
alert(a2); // 255
The last line in the preceding example prints the decimal representation of the octal value.
In JavaScript, you can put 0x before a hexadecimal value, also called hex for short, for example:
var a2=0xff;
alert(typeof a2); //number
alert(a2); // 255

3. **BinaryLiterals**

To display a binary representation of an integer, you had to pass them to the parseInt() function as a string with a radix of 2, as follows:
alert(parseInt(`111`,2)); // 7

4. **Exponentliterals**

1e1 (also written as 1e+1 or 1E1 or 1E+1) represents the number 1 with a 0 after it, or in other words, 10. Similarly, 2e+3 represents the number 2 with three 0s after it, or 2000, for example:
alert(1e1); // 10
alert(1e+1); // 10
alert(2e+3); // 2000
2e+3 means moving the decimal point three digits to the right of the number **2**. There's also 2e-3, meaning you move the decimal point three digits to the left of the number **2**.
alert(2e-3); // 0.002

5. **Infinity**

There is a special value in JavaScript called Infinity. It represents a number too big forJavaScript to handle. Infinity is indeed a number, as typing typeof Infinity in the console will confirm. You can also quickly check that a number with 308 zeros is ok, but 309 zeros is too much. To be precise, the biggest number JavaScript can handle is 1.7976931348623157e+308, while the smallest is 5e-324.
alert(Infinity); // Infinity
alert(1e309); // Infinity

6. **NaN**

You get NaN when you try to perform an operation that assumes numbers, but the operation fails. For example, if you try to multiply 10 by the character "f", the result is NaN, because "f" is obviously not a valid operand for a multiplication:
var a=10*"f"
alert(a); // NaN
NaN is contagious, so if you have even one NaN in your arithmetic operation, the whole result goes down the drain.

7. **Strings**

A string is a sequence of characters used to represent text. In JavaScript, any value placed between single or double quotes is considered a string. This means that 1 is a number, but "1" is a string. When used with strings, typeof returns the string "string".

8. **Booleans**

There are only two values that belong to the Boolean data type: the true and false values used without quotes:

var a=true;
alert(typeof a); // Boolean

III. Logical operators

There are three operators, called logical operators, that work with Boolean values. These are as follows:

! : logical NOT (negation)
&& :logical AND
|| :logical OR

You know that when something is not true, it must be false. Here's how thisisexpressed using JavaScript and the logical ! operator:

var a=!true;
alert(a); // false

Let's see some examples of the other two operators–the logical AND (&&) and the logical OR (||). When you use &&, the result is true only if all of the operands are true. When you use ||, the result is true if at least one of the operands is true:

var b1 = true, b2 = false;
alert(b1||b2); // true

Here's a list of the possible operations and their results:

Operation	**Result**
true&&true	True
true&& false	False
false &&true	False
false && false	False
true || true	True
true || false	True
false || true	True
false || false	False

Table 7.1 boolean table

IV. Comparison

There's another set of operators that all return a Boolean value as a result of the operation. These are the comparison operators. The following table lists them together with example uses:

Operator symbol	**Description**	**Example**

==	**Equality comparison**: This returns true when both operands are equal. The operands are converted to the same type before being compared. They're also called loose comparison.	> 1 == 1;
		true
		> 1 == 2;
		false
		> 1 =='1';
		true

===	**Non-equality comparison**: This returns true if the operands are not equal to each other (after a type conversion).	>1 != 1;
		false
		>1 != '1';

		false >1 != '2'; true

!=	**Non-equality comparison**: This returns true if the operands
	are not equal to each other (after a type conversion).
		>1 != 1;
		false
		>1 != '1';
		false
		>1 != '2';
		true

!==	**Non-equality comparison without type conversion**:
	Returns true if the operands are not equal or if they are of different
	types.
		> 1 !== 1;
		false
		> 1 !== '1';
		true

>	This returns true if the left operand is greater than the right one.	> 1 >1;
		false
		> 33 >22;
		true

>=	This returns true if the left operand is greater than or equal to the right one.	> 1 >= 1; true

<	This returns true if the left operand is less than the right one.	> 1 <1;
		false
		> 1 <2;
		true

<=	This returns true if the left operand is less than or equal to the right one.	> 1 <= 1;
		true
		> 1 <= 2;
		true

Table 7.2 comparison operation

Incrementation and decrementation operators can be used as on the table below

++	Increment a value by 1	Post increment is when the input value is incremented after it's returned, for example:
		`> var a = 123;`
		`> var b = a++;`
		`> b;`
		`123`
		`> a;`
		`124`
		The opposite is pre-increment. The input value is incremented by 1 first and then returned, for example:
		`> var a = 123;`
		`> var b = ++a;`
		`> b;`
		`124`
		`> a;`
		`124`

--	Decrement a value by 1	Post-decrement:
		`> var a = 123;`
		`> var b = a--;`
		`> b;`
		`123`
		`> a;`
		`122`
		Pre-decrement:
		`> var a = 123;`
		`> var b = --a;`
		`> b;`
		`122`
		`> a;`
		`122`

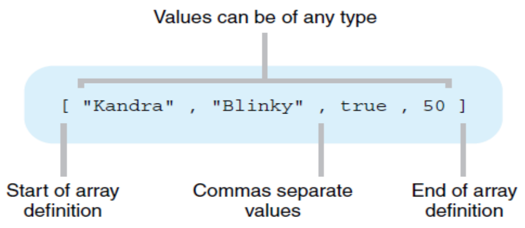
Table7.3 incrementation and decrementation

V. Arrays

Working with lists is an essential part of programming.An ordered list is called an *array* in JavaScript, as it is in many programming languages.The items in an array are called its *elements* and you usually want to work on the elements in some way.

1. Creating an array

To create an array, use square brackets. Once it’s created, you can assign the array to a variable so you can refer to it in your code. Figure 7.1 illustrates the process.



```
[ "Kandra" , "Blinky" , true , 50 ]
```

Figure 7.1 Use square brackets to create an array

The following listing creates two arrays and displays them on the console to give this output:

```
var scores =[10,13,4];
var names =["Kandra","Dax","Blinky"];
alert(scores);
alert(names);
```

Commas separate the elements, which can be numbers, strings, objects, functions, or any data type or mix of types—you can even have arrays of arrays.

2. Accessingarrayelements

We have created an array and assigned it to a variable, so now we can pass that variable to functions, like alert(). At some point we will want to access the elements that make up the array. The square brackets do double duty; they enclose the list when we define the array, and we use them to access individual elements. As shown in figure 7.2, we specify each element with an index, a whole number marking where in the list the element occurs. The first element in an array has anindex of 0, the second an index of 1, and so on. We can think of the index as an offset from the start of the array; the first element is zero away from the start, the second is one away from the start, and so on.

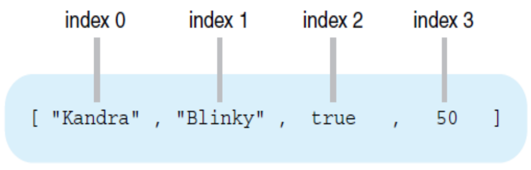


Figure 7.2 Each element of an array has an index, starting at 0.

To retrieve the value of an element at a given index, put the index inside square brackets after the name of a variable to which the array has been assigned, as shown in figure 7.3

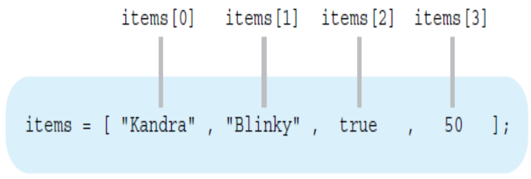


Figure 7.3 Use square brackets and an index to access elements of an array assigned to a variable

Here, we create an array and assign it to a variable:

var scores =[10,13,4,45];

To get the value of the third item in the array, 4, place the index, 2 (because you start with 0), in square brackets after the name of the variable:

alert(scores[2]);

You can assign scores[2] to another variable, set it as a property on an object, use it in an expression, or pass it as an argument to a function. We also make use of the array's length property that simply gives the number of elements in the array.

VI. Conditional statements

Conditional statements enable us to run code only when specified conditions are met.

1. The if statement

To execute a block of code only when a specified condition is met, you use an if statement.

```
if (condition) {
// Code to execute
}
```

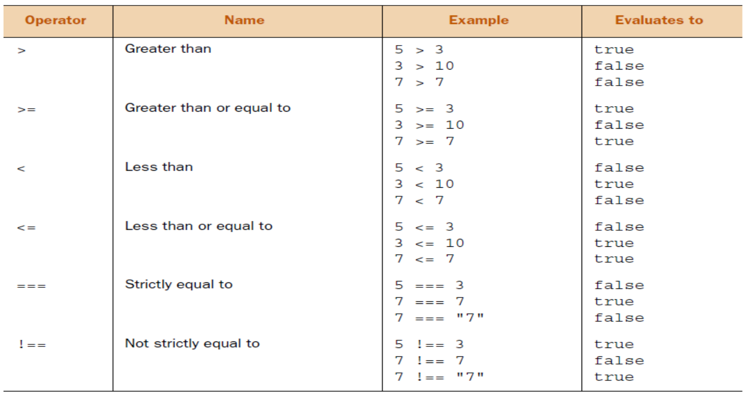
If the condition in parentheses evaluates to true, then JavaScript executes the statements in the code block between the curly braces. If the condition evaluates to false, then JavaScript skips the code block. Notice there's no semicolon after the curly braces at the end of an if statement.

2. The else clause

Sometimes we want different code to be executed if the condition in an if statement evaluates to false. We can make that happen by appending an else clause to the ifstatement.

```
if (condition) {
// Code to execute
}else{
// Code to execute
}
```

The table below presents comparison operators and their uses



Operator	Name	Example	Evaluates to
>	Greater than	5 > 3 3 > 10 7 > 7	true false false
>=	Greater than or equal to	5 >= 3 3 >= 10 7 >= 7	true false true
<	Less than	5 < 3 3 < 10 7 < 7	false true false
<=	Less than or equal to	5 <= 3 3 <= 10 7 <= 7	false true true
===	Strictly equal to	5 === 3 7 === 7 7 === "7"	false true false
!==	Not strictly equal to	5 !== 3 7 !== 7 7 !== "7"	true false true

Table 7.4 Comparison operators

3. Switch

If you find yourself using an if condition and having too many else...if parts, you can consider changing the if to a switch, as follows:

```
var a = '1',
result = '';
switch (a) {
case 1:
result = 'Number 1';
break;
case '1':
result = 'String 1';
break;
default:
result = 'I don't know';
break;
}
```

The result after executing this is "String 1". Let's see what the parts of a switch are:

- The switch statement.
- An expression in parentheses. The expression most often contains a variable, butcan be anything that returns a value.
- A number of case blocks enclosed in curly brackets.
- Each case statement is followed by an expression. The result of the expression iscompared to the expression found after the switch statement. If the result of thecomparison is true, the code that follows the colon after the case is executed.
- There is an optional break statement to signal the end of the case block. If thisbreak statement is reached, the switch statement is all done. Otherwise, if thebreak is missing, the program execution enters the next case block.
- There's an optional default case marked with the default statement andfollowed by a block of code. The default case is executed if none of the previouscases evaluated to true.

VII. Loops

The if...else and switch statements allow your code to take different paths, as if you'reat a crossroad, and decide which way to go depending on a condition. Loops, on the other hand, allow your code to take a few roundabouts before merging back into the main road. How many repetitions? That depends on the result of evaluating a condition before (or after) each iteration.

In JavaScript, the following are the four types of loops:

- whileloops
- do-whileloops

- forloops
- for-inloops

1. **Whileloops**

The while loops are the simplest type of iteration. They look like the following:

vari = 0;
while (i< 10) {
i++;
}

The while statement is followed by a condition in parentheses and a code block in curly brackets. As long as the condition evaluates to true, the code block is executed over and over again.

2. **Do-whileloops**

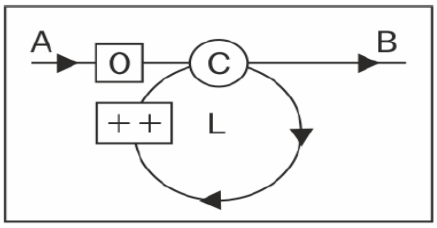
The do...while loops are a slight variation of while loops. An example is shown as follows:

var i = 0;
do {
i++;
} while (i < 10);

Here, the do statement is followed by a code block and a condition after the block. This means that the code block is always executed, at least once, before the condition is evaluated.

3. **For loops**

The for loop is the most widely used type of loop, and you should make sure you're comfortable with this one. It requires just a little bit more in terms of syntax:



In addition to the **C** condition and the **L** code block, you have the following:

- **Initialization**: This is the code that is executed before you even enter the loop(marked with **0** in the diagram)
- **Increment**: This is the code that is executed after every iteration (marked with **++**in the diagram)

The following is the most widely used for loop pattern:

- In the initialization part, you can define a variable (or set the initial value of anexisting variable), most often called i
- In the condition part, you can compare i to a boundary value, such as i< 100
- In the increment part, you can increase i by 1, such as i++

Here's an example:

var punishment = '';
for (vari = 0; i< 100; i++) {
punishment += 'I will never do this again, ';
}

All three parts (initialization, condition, and increment) can contain multiple expressions separated by commas. Say you want to rewrite the example and define the variable punishment inside the initialization part of the loop:

for (vari = 0, punishment = ''; i< 100; i++) {

```
punishment += 'I will never do this again, ';
}
```

The for loops can be nested within each other. Here's an example of a loop that is nested inside another loop and assembles a string containing ten rows and ten columns of asterisks. Think of i being the row and j being the column of an image:

```
var res = '\n';
for (vari = 0; i< 10; i++) {
for (var j = 0; j < 10; j++) {
res += '* ';
}
res += '\n';
}
```

4. **For…in loops**

The for...in loop is used to iterate over the elements of an array, or an object, as you'll see later. This is its only use; it cannot be used as a general-purpose repetition mechanism to replace for or while. Let's see an example of using a for-in to loop through the elements of an array. But, bear in mind that this is for informational purposes only, as for...in is mostly suitable for objects and the regular for loop should be used for arrays. In this example, you can iterate over all of the elements of an array and print out the index (the key) and the value of each element, for example:

```
// example for information only
// for-in loops are used for objects
// regular for is better suited for arrays
var a = ['a', 'b', 'c', 'x', 'y', 'z'];
var result = '\n';
for (vari in a) {
result += 'index: ' + i + ', value: ' + a[i] + '\n';
}
```

Assignment

Exercises1

Evaluate the following expressions and give the right answer. All the steps should be clearly presented.

a. (11<=3*2+5) || (6*2===4*3) && !(2+2===4)
b. (11>=3*2+5) || (6*2<=4*3) && (2+2===4)
c. (11>=3*2+5) || (6*2<=4*3) &&!(true)

Exercises2

Write a program in JavaScript language to read students scores and assign letter grades according to the following scale:

Scores	90-100	80-89	70-79	60-69	0-59
Grade	A	B	C	D	E

- The program should keep track of the number of As, Bs, Cs, Ds and Es.
- When all the grades have been printed the program should print a grade distribution giving the number of students who received each grade.
- The scores are collected from the keyboard and if the score is outside the range 0-100, the computer should print the score and the message **‘INVALID SCORE’.**

Exercises3

Write a program in JavaScript that reads the length and the width of an arbitrary number (N) of plots and calculate their area and perimeter of each. The program should print the length, width, area, perimeter of each field as well as the total area and total perimeter of all the fields.

Exercises4

Write a Program in JavaScript to compute the sales tax on a purchase and the total amount that must be paid after the tax has been added to the purchase amount.

Exercises5

Write a program in JavaScript that reads the radius of 10 circles and calculate the area of each. The program should print the radius as well as the area of each circle.

Exercises6

Write a program in JavaScript that solves the following equations:

a. Ax+B=0, where A and B are real numbers and x the unknown value.
b. $Ax^2+Bx+C=0$, where A, B and C are real numbers and x the unknown value.
c. $\begin{cases} ax+by=c \\ dx+ey=f \end{cases}$ Where a, b, c, d ,e and f are real numbers, but x and y the unknown values.