Expressions, statements, and expressions evaluation:

Introduction:

When we write a program, it can be considered to be a collection of statements, where the ordering of the statements matter. This is analogous to a paragraph being a collection of sentences and in order of the paragraph to be logical, the order sentences and content of the sentences are important. Within in a single sentence there are rules to the ordering of the words as well.

Eg.

```
There was a house in the mountains.  It was red and blue.
```

The above example is a simple paragraph that has a very specific meaning. It's a statement on the location of the house and the color of the house. If the order of the two sentences were switched the paragraph will no longer have meaning. The pronoun 'it' no longer refers to anything.

```
It was red and blue. There was a house in the mountains.
```

Similarly a program can be considered a collection of statements where the ordering of the statement are important. So we begin our study of JavaScript understanding JavaScript statements and expression evaluations.

Simple expressions and values:

The simplest expressions are literals. They have irreducible value or they evaluate to themselves.

Example 1:

Open a console window in Chrome.
type in `1`
When you do this, web browser will interpret your statement give back a value. In this case it gives back `1`.

Example 2:

Similarly if you type in `"Hello"` you will also get back the term `"Hello"`

These are literal values or literal expressions.

Types:

In the above example we see examples of 2 types. Numbers and Strings. There are 5 types in JavaScript, and will be 6 in the upcoming standard.

- Boolean
- Null
- Undefined
- Number
- String
- Symbol (new in ECMAScript 6)

Boolean Type

A boolean type has the value of either `true` or `false`.

Number Type

A number type represents any numeric value with a 64 bit floating point number.
Example of number values include:
1
1.2
56
3.156

String Type

A string is a sequence of characters. Literal strings are surrounded by either single quotes or double quotes. Examples of strings are:

```
"The train in spain went over the river."
"<h1> This is HTML</h1>"
```

Variables

Variables are placeholders whose value can change. To define a variable use the keyword **`var`** followed by the name of a variable.

```
var  myvariable;
```

The above example creates a variable with `null` type, because there is no value stored in it.

```
var myvariable = 4;
```

The above example creates a variable with `number` type and has an initial value of 4.

A variable can only hold one value at a time. The value of a variable can change! The type of the variable can also change. This is why it's called a variable. When we use variables it evaluates to value that is stored. If there is no stored value it evaluates to null.

In JavaScript values have type and the type of a variable is determined by the value it is storing. If we change the value of the variable, the type of the variable can also change to match the value.

Operators

We can combine values with operators to create something of the same type or something of a different type.

Common operators

The minus sign, ' – ' , is an operator. It is an operator that finds the difference between the number on the left hand side of the operator with the number on the right hand side of the operator.
Eg:

```
4 – 5
```

Finds the difference between 4 and 5 and has the value of -1.

Not all operators work with all types.

The '+' operator:
The '+' is unique as it does something different with different types.

If we use '+' with numbers, it is addition, but if we use '+' with strings it is concatenation.

Arithmetic operators

Operator	Name	Example usage

+	addition
	concatenation
		`4+5`
		`"car"+"dog"`

-	subtraction	`4-5`
*	multiplication	`4*5`
/	division	`4/5`
%	modulus	`4%6`
**	exponentiation	`4**6`

Comparison operators

Operator	Name	Example usage
>	greater than	5 > 6

>=	greater than or equal	5 >= 6
<	less than	5 < 6
<=	less than or equal	5 <= 6
===	Equals without type conversion	5 === 6
==	Equals with type conversion	5 == 6
!=	Not equal with type conversion	5 != "6"
!==	Not equal without type conversion.	5 !== 6

Boolean operators

Operator	Name	Example usage
&&	and	true && false
||	or	true || false

Grouping operators

Operator	Name	Example usage
(…)	brackets	(1+1)*(2+2)

Assignment operator

Operator	Name	Example usage

=	Assignment	temp = "cat"
		temp = 6

+=		temp +=6
		temp += "dog"

-=		temp -=6
*=		temp *= 6
/=		temp /=6
%=		temp %=6

Long expression and operator precedence.

We can construct long expressions consisting of multiple values and multiple operators. Each of these expressions will evaluate to a single value. The question then becomes which operator do we apply first.

Example:

```
3+5*6
```

In the above example, the answer is 33 because the multiplication operations occurs before the addition operation.

In general the order happens as follows:

1. Grouping operators
2. Arithmetic operators
    a. Multiplication, divisions, exponentiation
    b. Addition and subtraction
3. Comparison operators
4. Boolean operators
    a. and
    b. or
5. Assignment operators

A full table for order of operations can be found here:
https://developer.mozilla.org/en/docs/Web/JavaScript/Reference/Operators/Operator_Precedence

Automatic Type conversion:

What happens when you add a number to a string?

```
"Hello" + 10
```

When you add a number to a string, the number is converted to a string and the two are concatenated, so the above expression will evaluate to `"Hello10"`.

What happens when you add a string to a number?

```
10 + "Hello"
```

The above expression will evaluate to `"10Hello"`

The above is an example of automatic type conversion where the numbers are converted to strings before the operation is carried out. When using operators with mixed types JavaScript will convert some of the values for you.

What happens when you compare a number to string?

```
10 == "10"
```

The above expression evaluates to `true`. When you compare numbers with strings, the string is converted to a number. If it cannot be converted, then it takes on the value of NaN.

If we want the above expression to be false, we would need to use the non converting comparison operator '==='.

```
10 === "10"
```

The above expression evaluates to false, because '===' does not convert the string to a number first. So, if the types are different on the left and right hand sides then '====' will always be false.

Showing your work when evaluating complex instructions

Evaluating an expression refers finding the value of an expression. An expression has one value, determined when a program runs. Expression can be evaluated 1 sub-expression at a time. To show your work, then is to evaluate each of the sub expression in the correct order.

Example 0:

```
var hello =40;
```

Evaluate the following expression

```
hello + 40*2
```

Answer:

Step-By-Step Evaluation:
Step 1:
First we replace the variable hello with it's value.
to get
`40+40*2`
Step 2:
Next we perform the multiplication operation first because it has higher precedence.
`40+80`
Step 3:
Finally we perform the addition operation and get
`80`
our final answer.

Example 1:

Evaluate the following expression.

```
“Hello” + “World “ + 20+15;
```

Answer:

Step-By-Step Evaluation:

```
“Hello” + “World “ + 20+15
“Hello World” + 20+15
“Hello World20” + 15
```

```
“Hello World2015”
```

Example 2: expressions with variables

Assuming the following variable declarations

```
var a= 5;
var b = 6;
var c = a+b;
```

Evaluate the following expression:

```
0+1+“:The answer is:” + (a+b+20)*c + a;
```

Answer:

Memory Values

Variable	Value
a	5
b	6
c	11

Step-by-step evaluation:

```
0+1+“:The answer is:” + (a+b+20)*c + a
0+1+“:The answer is:” +(5+6+20)*11 + 5
1+“:The answer is:” +(5+6+20)*11 + 5
“1:The answer is:” +(5+6+20)*11 + 5
“1:The answer is:” +(5+6+20)*11 + 5
“1:The answer is:” +(11+20)*11 + 5
“1:The answer is:” +(31)*11 + 5
“1:The answer is:” +314 +5
“1:The answer is:314”+5
“1:The answeris:3145”
```

Creating your own expressions

At the heart of computer programming is creating your own expressions often involving variables. Since variables can change value as a program runs, expressions involving variables makes your program flexible.

If we want to write an expression that multiplies a number by 2 we could write.

```
var num;

//Some code goes here

var answer = num*2;
```

The above code will multiply whatever is in `num` by 2 and store it in answer. The value of `num` will be determined statements between

```
var num;
```

and

```
var answer = num*2;
```