About var, let and const

var

- var declares a variable.
- This process of creating a variable in JavaScript is called "declaring" a variable:
- Variables are containers that store information in them.
- Once a variable is declared it does not have any value, one needs to store a value using "="

Syntax

var name;

name = value

or

var name =value

Variable names -

- Must begin with a letter, or $, or _
- are case sensitive (y and Y are different)
- Reserved JavaScript words cannot be used as names

Examples

var age=25;

var name="ngit"

var company;

company="ngit"

let

- let , like var, is also used to store values.
- It was introduced in 2015.
- Variables declared using let cannot be redeclared.
- And they also must be declared before use.

Syntax

let name;

name = value

or

let name =value

Variable names -

- Must begin with a letter, or $, or _
- are case sensitive (y and Y are different)
- Reserved JavaScript words cannot be used as names

Examples

let age=25;

let name="ngit"

let company;

company="ngit"

Important Note: Variables defined with 'let' have block scope. Will learn about scope soon.

const

- Variables declared with const cannot be redeclared.
- They also cannot be reassigned.
- Variables defined with const have Block Scope. Will learn about scope soon.
- We declare a variable with const when we are sure that the value will not be changed

Syntax

const name=value;

Examples

const PI = 3.1415;
PI = 3.14; // error
PI = PI + 10; // error

Also
const PI ;
PI = 3.14; // error . Should be declared and initialized at the same time.

Note: a common convention is to use all-uppercase letters for const

About Operators

Types

- Arithmetic
- Assignment
- Relational or Comparison
- Logical
- Conditional

Arithmetic

Addition – works as expected with numbers but when used with a string results in string concatenation.
let a=10; let b =20; let c = a+b; //30

let p="hello"; var q=" world"; var r = p +q; //hello

world let x = 5;

let y = 3;

// addition

console.log('x + y = ', x + y); // 8

// subtraction

console.log('x - y = ', x - y); // 2

// multiplication

console.log('x * y = ', x * y); // 15

// division

console.log('x / y = ', x / y); // 1.6666666666666667

// remainder

console.log('x % y = ', x % y);   // 2

// increment

```
console.log('++x = ', ++x); // x is now 6

console.log('x++ = ', x++); // prints 6 and then

increased to 7 console.log('x = ', x);   // 7
```

```
// decrement
console.log('--x = ', --x); // x is now 6
console.log('x-- = ', x--); // prints 6 and then
decreased to 5 console.log('x = ', x);  // 5
//exponentiation
console.log('x ** y =', x ** y);
```

Assignment

Operator	Name	Example
=	Assignment operator	a = 7; // 7
+=	Addition assignment	a += 5; // a = a + 5
-=	Subtraction Assignment	a -= 2; // a = a - 2
*=	Multiplication Assignment	a *= 3; // a = a * 3
/=	Division Assignment	a /= 2; // a = a / 2
%=	Remainder Assignment	a %= 2; // a = a % 2
**=	Exponentiation Assignment	a **= 2; // a = a**2

Relational or Comparison

Operator	Description	Example
==	Equal to: returns true if the operands are equal	x == y
!=	Not equal to: returns true if the operands are not equal	x != y

===	Strict equal to: true if the operands are equal and of the same type	x === y

!==	Strict not equal to: true if the operands are equal but of different type or not equal at all	x !== y
>	Greater than: true if left operand is greater than the right operand	x > y
>=	Greater than or equal to: true if left operand is greater than or equal to the right operand	x >= y
<	Less than: true if the left operand is less than the right operand	x < y
<=	Less than or equal to: true if the left operand is less than or equal to the right operand	x <= y

```
// equal operator
    console.log(2 == 2); // true
    console.log(2 == '2'); // true

    // not equal operator
    console.log(3 != 2); // true
    console.log('hello' != 'Hello'); // true

    // strict equal operator
    console.log(2 === 2); // true
    console.log(2 === '2'); // false

    // strict not equal operator
    console.log(2 !== '2'); // true
    console.log(2 !== 2); // false
```

Logical Operator

Logical operators perform logical operations and return a boolean value, either true or false.

```
// logical AND
console.log(true && true); // true
console.log(true && false); // false
// logical OR
console.log(true || false); // true
// logical NOT
console.log(!true); // false
```

Operator	Description	Example
&&	Logical AND	(10==20 && 20==33) = false
||	Logical OR	(10==20 || 20==33) = false
!	Logical Not	!(10==20) = true

About Loops

Loops offer a quick and easy way to do something repeatedly.

For loop

Syntax

```
for (initialExpression; condition; updateExpression) {

  // for loop body

}
```

- The initialExpression initializes and/or declares variables and executes only once.
- The condition is evaluated.
- If the condition is false, the for loop is terminated.
- If the condition is true, the block of code inside of the for loop is executed.
- The updateExpression updates the value of initialExpression when the condition is true.
- The condition is evaluated again. This process continues until the condition is false.

Example

Example 1 -

```
// program to display text 5 times

const n = 5;

// looping from i = 1 to 5
for (let i = 1; i <= n; i++) {
   console.log(`I love my country.`);

}
```

Output –

```
I love my country.
I love my country.
 I love my country.
 I love my country.
 I love my country.
```

Example 2 –

```
// program to display the sum of natural
numbers let sum = 0;
const n = 100

// looping from i = 1 to n
// in each iteration, i is increased by
1 for (let i = 1; i <= n; i++) {
   sum += i;  // sum = sum + i
}

console.log('sum:',
sum); Output : sum:
5050
```

Example 3

Infinite loop –

```
for(let i = 1; i > 0; i++) {
   // block of code
}
```

While loop

- A while loop evaluates the condition inside the parenthesis ().
- If the condition evaluates to true, the code inside the while loop is executed.
- The condition is evaluated again.
- This process continues until the condition is false.
- When the condition evaluates to false, the loop stops.

Syntax

```
while (condition) {
   // body of loop
}
```

Examples

Example 1

```
// program to display numbers from 1 to 5

// initialize the variable

let i = 1, n = 5;


// while loop from i = 1 to 5

while (i <= n) {

console.log(i);
 i += 1;
}

// infinite while loop

while(true){

   // body of loop

}
```

Do ..while

- The body of the loop is executed at first. Then the condition is evaluated.
- If the condition evaluates to true, the body of the loop inside the do statement is executed again.
- The condition is evaluated once again.
- If the condition evaluates to true, the body of the loop inside the do statement is executed again.
- This process continues until the condition evaluates to false. Then the loop stops.

Syntax

```
do {

    // body of loop

} while(condition)
```

Examples

```
    // program to display numbers

    let i = 1;
const n = 5;

// do...while loop from 1 to 5
do {
    console.log(i);
    i++;
} while(i <= n);
```

Output –

1

2

3

4

5

// infinite do...while

loop const count = 1;

do {

  // body of loop
} while(count == 1)