CYDEO

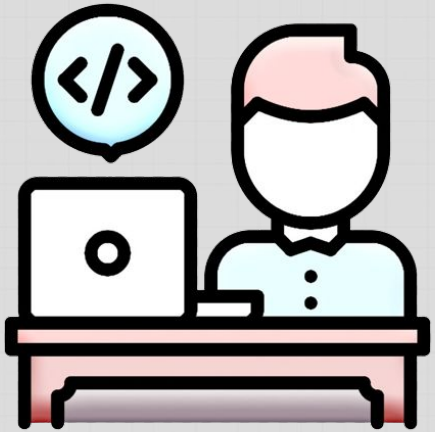
JavaScript Programming Day01

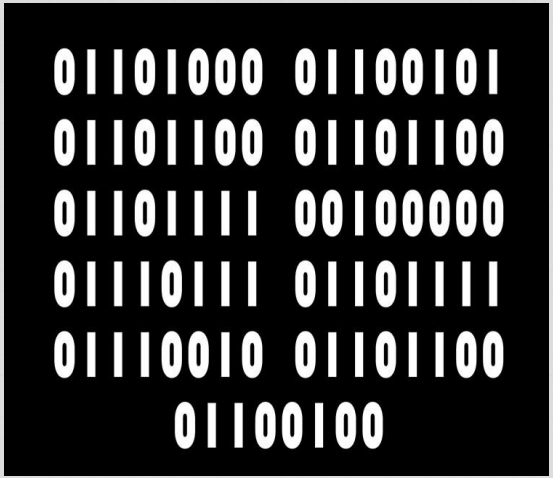
Content

- Introduction to JavaScript Programming
- Variables and Data Types
- Operators
- Decision Makings
- Loops
- Arrays
- String

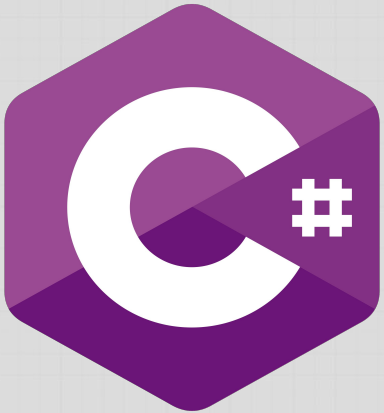
Programming Language

- A computer language
- Used by programmers to Communicate with computers





Most Popular Programming Languages

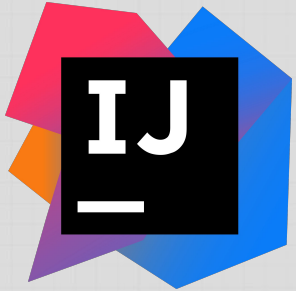


Why Learn JavaScript?

- The language of the web
- Beginner-friendly syntax
- High demand in the job market
- Versatile for both frontend and backend development

Different IDEs for JavaScript

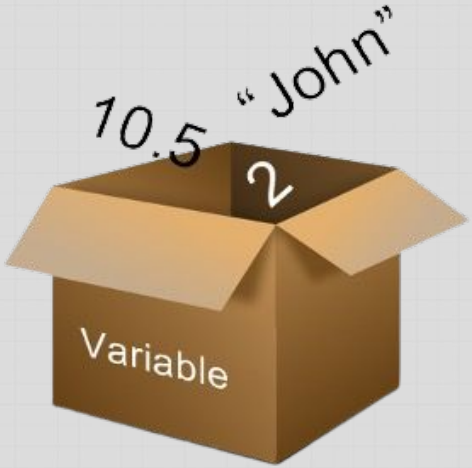
- An integrated development environment (IDE) is a software application that provides comprehensive facilities to computer programmers for software development

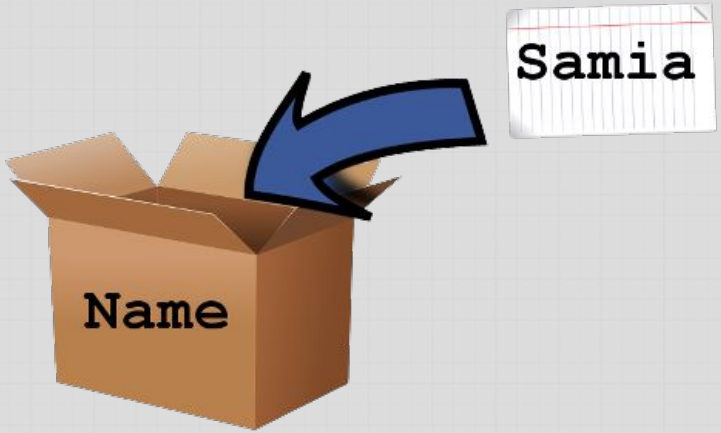


Variables and Data Types

What Is A Variable?

- A variable is a container for storing a data value





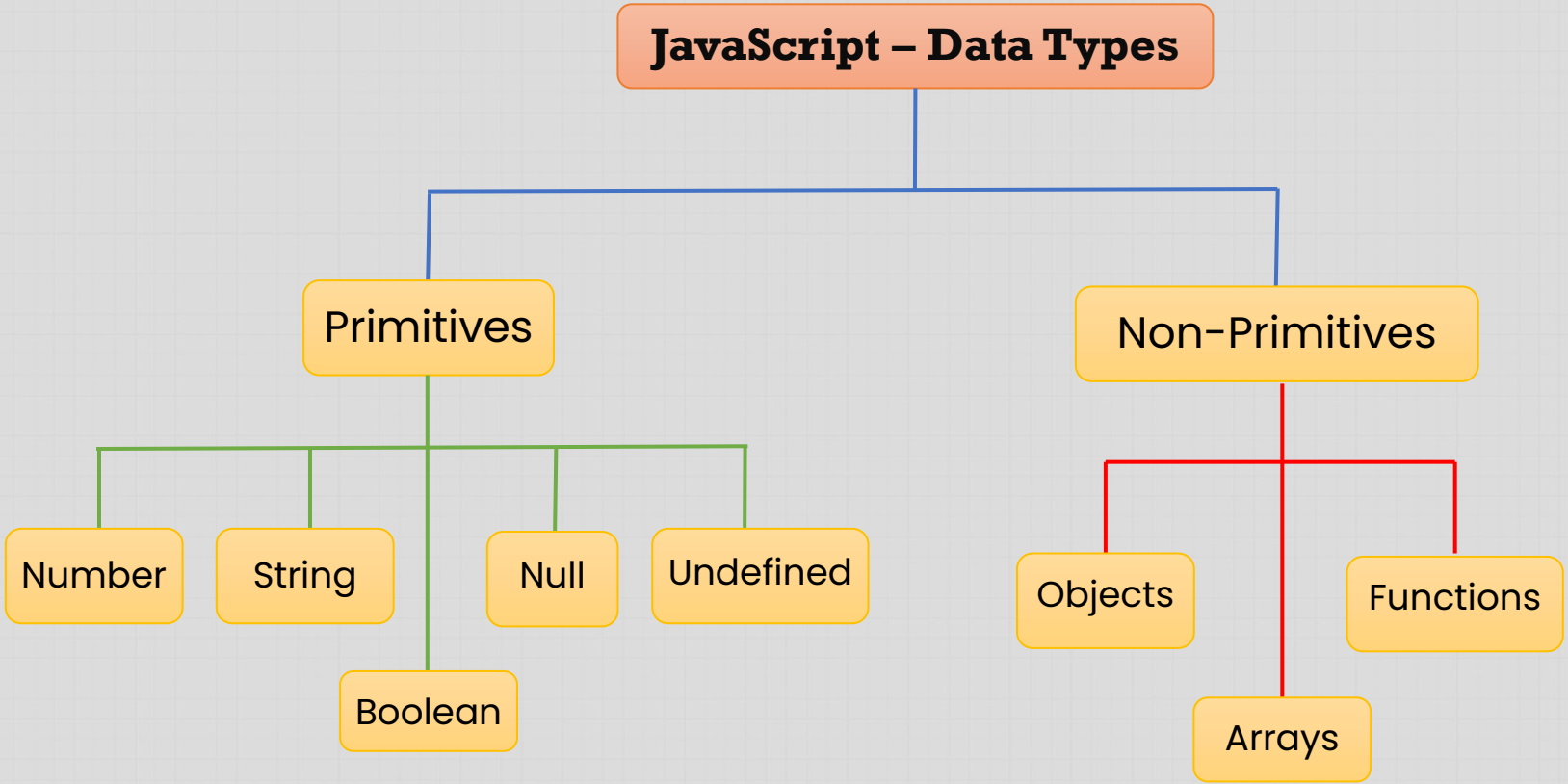
Variable

- Improves the reusability of the data
- Variables must be declared before use

```
var variableName = Data;

let variableName = Data;

const VARIABLE_NAME = Data;
```

```
var magicWord = "Wooden Spoon";

let number = 300;

const MAX_USER = 6;

var isEmployed = true;

let isMarried = false;
```

Data Types In JavaScript



Operators

Arithmetic Operators

NAME	OPERATOR	PURPOSE & NOTES	EXAMPLE	RESULT
ADDITION	+	Adds one value to another	10+5	15
SUBTRACTION	-	Subtracts one value from another	10-5	5
DIVISION	/	Divides two values	10/5	2
MULTIPLICATION	*	Multiplies two values	10*5	50
MODULUS	%	Divides two values and returns the remainder	10%3	1

Shorthand Operators

NAME	SHORTHAND OPERATOR	MEANING
Assignment	x = y	x = y
Addition Assignment	x += y	x = x + y
Subtraction Assignment	x -= y	x = x – y
Multiplication Assignment	x *= y	x = x * y
Division Assignment	x /= y	x = x / y
Remainder Assignment	x %= y	x = x % y

Relational Operators

Operator	Description
>	Greater than
>=	Greater than or equal
<	Less than
<=	Less than or equal
==	Equal
===	Strict Equal
!=	Not equal

All the relational operators will return Boolean (True or False)

Logical Operators

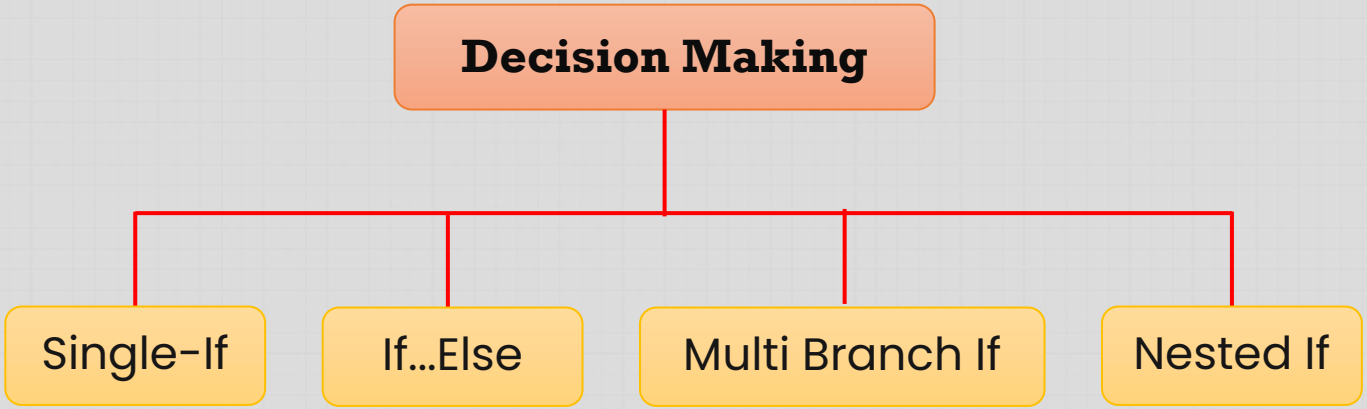
OPERATOR	DESCRIPTION
&&	Logical AND
||	Logical OR
!	Logical NOT

All the logical operators will return Boolean (True or False)

Decision Makings

If Statements

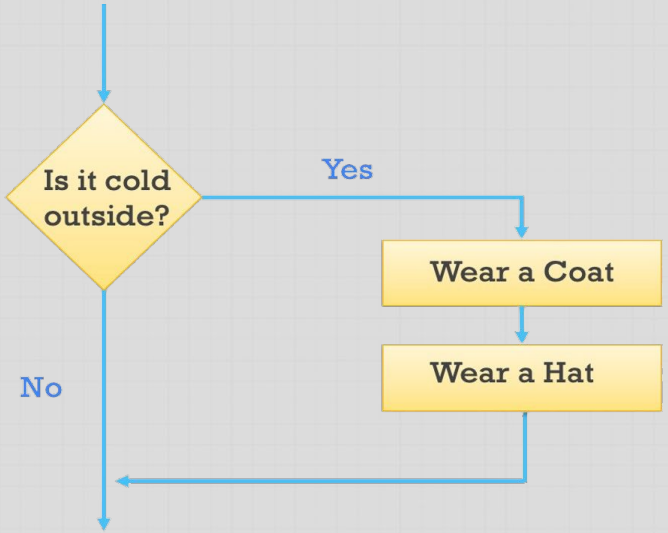
- Used for making decisions based on specified criteria



Single If

- The if statement evaluates a condition
- If the condition evaluates to true, any statements in the subsequent code block are executed

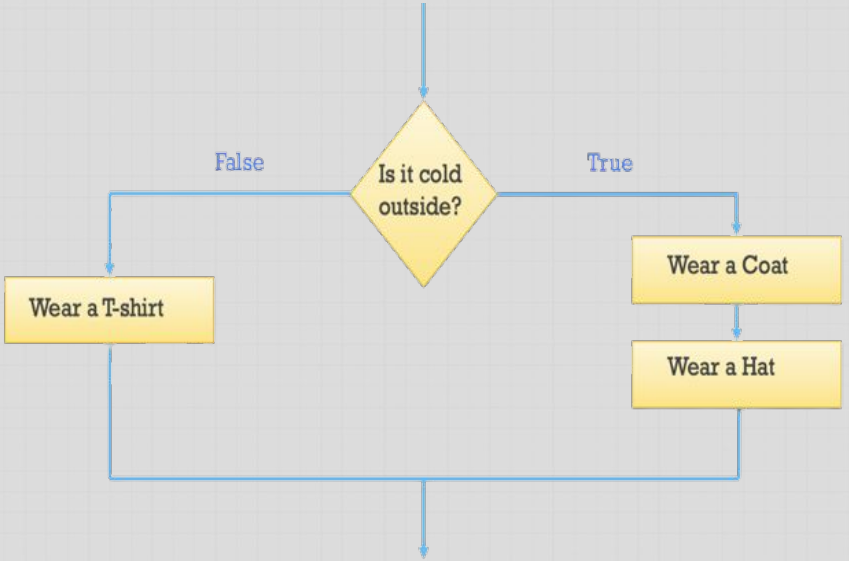
```
if(Condition){
    Statements
}
```



If...Else

- The if...else statement checks a condition
- If it resolves to true the first code block is executed
- If the condition resolves to false, the second code block is run instead

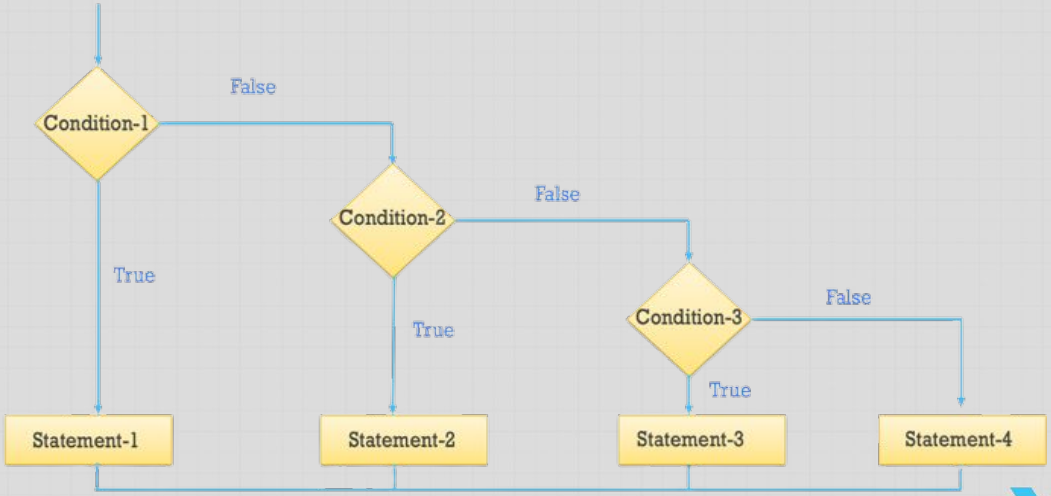
```
if(Condition){
    Statements
}else{
    Statements
}
```



Multi-branch If

- Multi-branch if statement can be used to create an else if clause
- It is used to make decision among several alternatives

```
if(Condition1){
    Statements
}else if(Condition2){
    Statements
}else{
    Statements
}
```



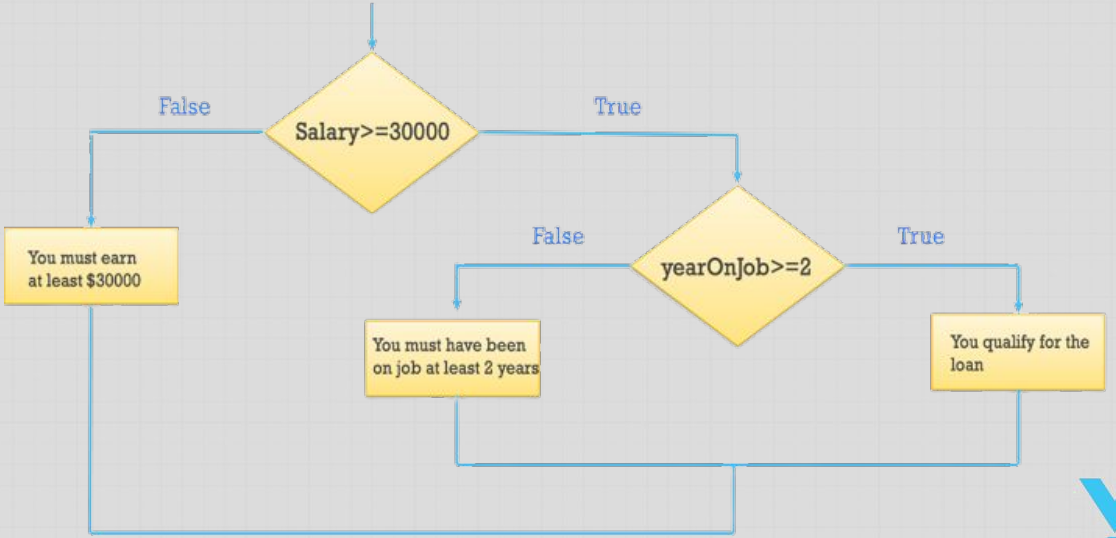
Multiple else if blocks can be given

Nested If

- Nested if statements can be used for creating a pre-condition
- It's used if one condition can be evaluated to several alternatives

```
if(Condition){
    if(Condition){
        Statements
    }
}
```

Outer and Inner If statements can be any type of if statement



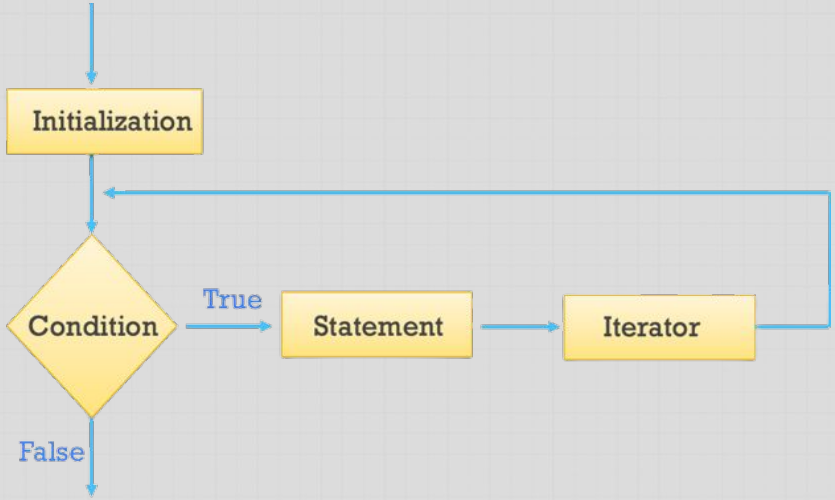
Loops

Loops

- Used for repeating a set of statements
- There are two types of loops:
  - For Loop
  - While Loop
  - Do-While Loop

For Loop

- Runs the given code a specific number of times
- Initialization is the starting point of the loop
- Condition is the ending point of the loop
- Iterator is responsible for making the condition false



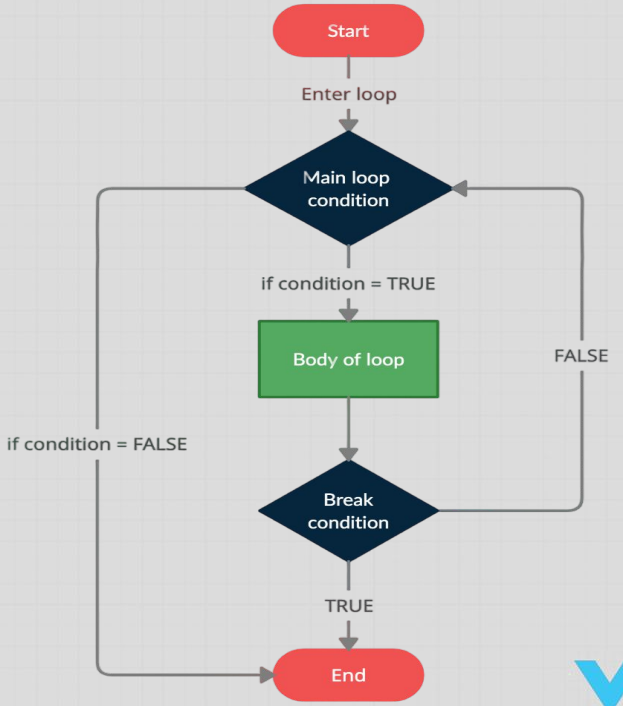
For Loop Syntax

```
for(initialization; condition; iterator){
        Statements
}
```

- The initialization expression initializes the loop
- When the condition expression evaluates to false, loop stops running
- The Iteration gets executed after each iteration through the loop

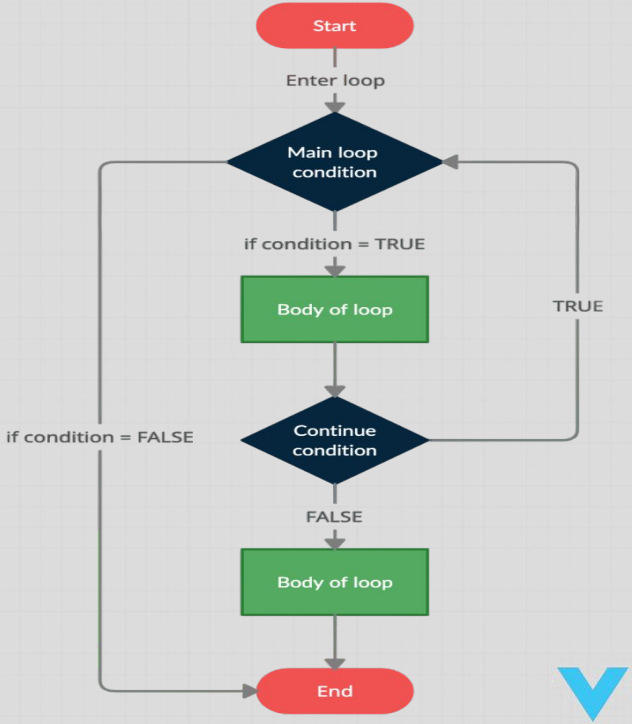
Break Statement

- Causes the termination of the loop
- Tells the interpreter to go on to the next statement of code outside of the loop



Continue Statement

- Skips the current iteration of the loop
- Tells the interpreter to jump to the next iteration



String

String

- String is a sequence of characters, surrounded by double quotes (“”) or single quotes (‘’)
- A string object is immutable; once it is created, it can’t be altered

```
let greeting = "Hello World";

let name = "Wooden Spoon";
```

```
let greeting = 'Hello World';

let name = 'Wooden Spoon';
```

String: Sequence of Characters

- Strings are ordered sequences of characters, and each character has a unique index number

```
let s = 'CYDEO !';
//index: 0123456
```

String Methods

Method Name	Method Name	Method Name
charAt()	length	toLowerCase()
toUpperCase()	trim()	indexOf()
lastIndexOf()	replace()	replaceFirst()
substring()	repeat()	isEmpty()
isBlank()	equals ()	equalsIgnoreCase()
contains()	startsWith()	endsWith()

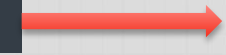
String Concatenation

- The action of linking things together by using “+” or “${}” operator

```
let age = 20;
console.log("I am " + age + " years old");
```

I am 20 years old

```
let age = 20;
console.log(`I am ${age} years old`);
```



I am 20 years old

Arrays

Arrays

- Special type of variable that's used to store multiple values of any types
- The values in the array are ordered, changeable, can be duplicated, and can be of any data type
- Each element has a unique index number

```
let days = ["MON", "TUE", "WED", "THU", "FRI"];

let fruits = ["Cherry", "Lemon", "Cherry"];

let scores = [75, 78, 85, 90, 93, 95, 85];

let myArray = ["A", "B", 1, 2, true, false];
```

Accessing Array Elements

- Elements of an array can be accessed by using the square brackets [ ]
- Index number needs to be provided within the square brackets

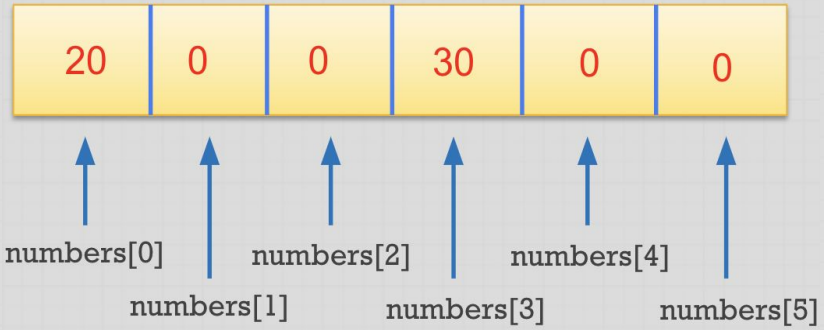
```
numbers[0] = 20;
```

arrayRefVar

index

value

```
numbers[3] = 30;
```



Assigning Values to Array Indexes

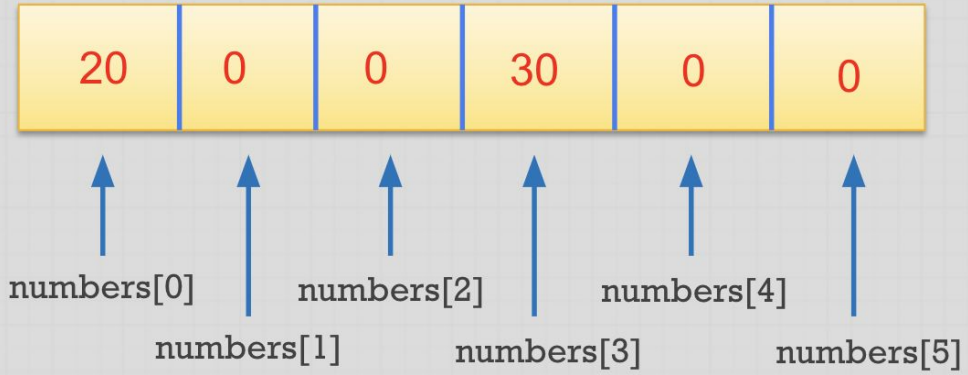
```
numbers[0] = 20;
```

arrayRefVar

index

value

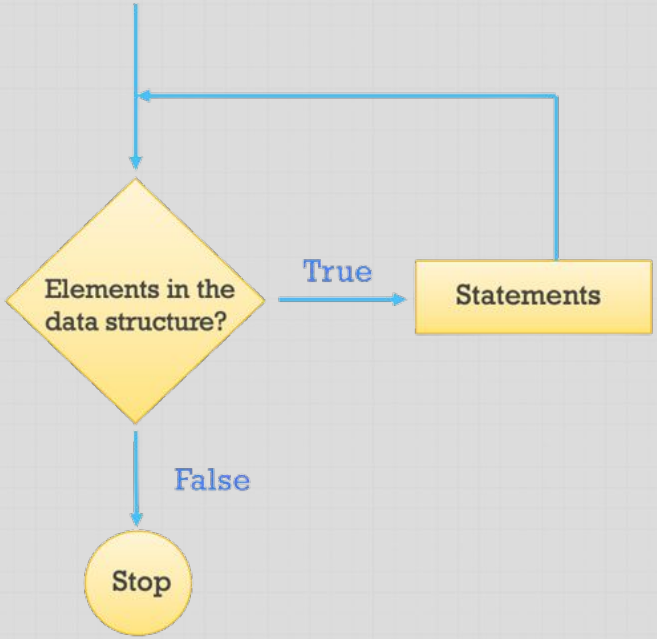
```
numbers[3] = 30;
```



For...of Loop

- Used to access each successive value of a sequence
- Iteration order and number of iterations are fixed
- Does not have index numbers

```
for (let element of sequence) {

    // Statements

}
```



Array Methods

Method Name	Description
push(element)	Adds the specified element to the end of the array
unshift(element)	Adds the specified to the beginning of the array
splice(index, # of elements)	Remove the specified number of element starting from index
shift()	Removes the first element from the array
pop()	Removes the last element from the array