**Coding**

**Life**

A black and white text

Description automatically generated

SH.Mohamed Sihan

WEB DEVELOPER

**Table of Contents**

[Common to all Languages 1](#_Toc176224789)

[1. Operators 1](#_Toc176224790)

[1.1. Arithmetic Operator 1](#_Toc176224791)

[1.2. Assignment Operators 1](#_Toc176224792)

[1.2.1. Assignment Operators 1](#_Toc176224793)

[1.2.2. Shift Assignment Operators 1](#_Toc176224794)

[1.2.3. Bitwise Assignment Operators 2](#_Toc176224795)

[1.2.4. Logical Assignment Operators 2](#_Toc176224796)

[1.3. Comparison Operators 2](#_Toc176224797)

[1.4. String Operators 2](#_Toc176224798)

[1.5. Logical Operators 2](#_Toc176224799)

[1.6. Bitwise Operators 3](#_Toc176224800)

[1.7. Ternary/Conditional Operators 3](#_Toc176224801)

[1.8. Type Operators 3](#_Toc176224802)

[1.8.1. Converting Strings to Numbers 3](#_Toc176224803)

[1.8.2. Converting Numbers to Strings 3](#_Toc176224804)

[1.8.3. Converting Dates to Numbers 4](#_Toc176224805)

[1.8.4. Converting Date to String 4](#_Toc176224806)

[1.8.5. Converting Booleans to Numbers 4](#_Toc176224807)

[1.8.6. Converting Booleans to String 4](#_Toc176224808)

[2. Concatenation 5](#_Toc176224809)

[3. Get Output 5](#_Toc176224810)

[4. Key Words 5](#_Toc176224811)

[5. Variables 6](#_Toc176224812)

[6. Comments 6](#_Toc176224813)

[7. Primitive Data Type 6](#_Toc176224814)

[8. Escape Characters 8](#_Toc176224815)

[9. Arrays 8](#_Toc176224816)

[Languages 9](#_Toc176224817)

[1. JavaScript 9](#_Toc176224818)

[1.1. JS Objects 9](#_Toc176224819)

[1.1.1. JavaScript Object Constructors 10](#_Toc176224820)

[1.2. JavaScript Events 11](#_Toc176224821)

[1.3. Data Types 12](#_Toc176224822)

[1.3.1. JS Strings 12](#_Toc176224823)

[1.3.2. JS Numbers 15](#_Toc176224824)

[1.4. Array 16](#_Toc176224825)

About Language

Types of languages.

1. **High-Level vs. Low-Level Languages**

* High-Level Languages: These languages are abstracted from the hardware and are designed to be easy for humans to read and write. They manage many complexities of the machine, such as memory management, and provide powerful abstractions.

Examples: Python, Java, JavaScript, C#, Ruby, PHP

* Low-Level Languages: These languages provide little abstraction from a computer's hardware and are closer to machine code. They offer more control over hardware but are harder to read and write.

Examples: Assembly Language, C

1. **Procedural vs. Object-Oriented vs. Functional Languages**

* Procedural Languages: These languages follow a step-by-step approach to solve a problem, focusing on the sequence of tasks or procedures to be executed.

Examples: C, Pascal, Fortran

* Object-Oriented Languages: These languages use objects and classes to structure programs. They emphasize data encapsulation, inheritance, and polymorphism.

Examples: Java, C++, Python, Ruby, C#

* Functional Languages: These languages emphasize the use of functions and immutable data. They treat functions as first-class citizens and avoid changing state.

Examples: Haskell, Erlang, Lisp, Scala, F#

1. **Scripting Languages**

* Scripting Languages: These languages are typically used for automating tasks, manipulating data, and controlling other software. They often run within a runtime environment and are interpreted rather than compiled.

Examples: JavaScript, Python, Perl, Ruby, PHP, Shell Scripting (Bash)

1. **Markup and Query Languages**

* Markup Languages: These languages are used to define the structure and presentation of text. They are often used in conjunction with other programming languages.

Examples: HTML, XML, Markdown

* Query Languages: These languages are used to query and manipulate data in databases.

Examples: SQL, XQuery

1. **Domain-Specific Languages (DSLs)**

* Domain-Specific Languages: These languages are tailored for specific application domains, offering specialized syntax and features for types of tasks.

Examples: CSS for styling web pages VHDL/Verilog for hardware description MATLAB for numerical computing

1. **Compiled vs. Interpreted Languages**

* Compiled Languages: These languages are translated into machine code by a compiler before execution, producing an executable file.

Examples: C, C++, Rust

* Interpreted Languages: These languages are executed by an interpreter, translating code line by line or statement by statement during runtime.

Examples: Python, JavaScript, Ruby

1. **Multi-Paradigm Languages**

* Multi-Paradigm Languages: These languages support multiple programming paradigms, allowing developers to choose the most suitable approach for their tasks.

Examples: Python (supports procedural, object-oriented, and functional programming), JavaScript (supports procedural, object-oriented, and functional programming), Scala (supports object-oriented and functional programming)

1. **Systems Programming Languages**

* Systems Programming Languages: These languages are designed for writing system- level software such as operating systems, drivers, and other low-level applications.

Examples: C, C++

1. **Logic Programming Languages**

* Logic Programming Languages: These languages are based on formal logic and are used to express facts and rules about problems within a system.

Examples: Prolog

1. **Concurrent and Parallel Languages**

* Concurrent Languages: These languages provide constructs for concurrent execution, allowing multiple processes or threads to run simultaneously.

Examples: Go, Erlang

* Parallel Languages: These languages are designed for parallel computing, where tasks are executed simultaneously on multiple processors or cores.

Examples: CUDA, OpenCL

|  |  |  |
| --- | --- | --- |
| **Category** | **Description** | **Examples** |
| High-Level | Abstracted from hardware, easy for humans to read and write | Python, Java, JavaScript, C#, Ruby, PHP |
| Low-Level | Closer to hardware, more control over hardware | Assembly Language, C |
| Procedural | Focuses on procedures and steps | C, Pascal, Fortran |
| Object-Oriented | Uses objects and classes | Java, C++, Python, Ruby, C# |
| Functional | Emphasizes functions and immutable data | Haskell, Erlang, Lisp, Scala, F# |
| Scripting | Automates tasks, interpreted languages | JavaScript, Python, Perl, Ruby, PHP |
| Markup | Defines structure and presentation of text | HTML, XML, Markdown |
| Query | Queries and manipulates data in databases | SQL, XQuery |
| DSLs | Tailored for specific domains | CSS, VHDL/Verilog, MATLAB |
| Compiled | Translated into machine code before execution | C, C++, Rust |
| Interpreted | Executed by an interpreter, line by line | Python, JavaScript, Ruby |
| Multi-Paradigm | Supports multiple programming paradigms | Python, JavaScript, Scala |
| Systems Programming | For system-level software | C, C++ |
| Logic Programming | Based on formal logic | Prolog |
| Concurrent | Supports concurrent execution | Go, Erlang |
| Parallel | Designed for parallel computing | CUDA, OpenCL |

Platform

|  |  |  |  |
| --- | --- | --- | --- |
| **Platform** | **Type** | **Description** | **Examples** |
| Windows | Operating System | Widely used OS from Microsoft | Windows 10, Windows Server |
| macOS | Operating System | Unix-based OS from Apple | macOS Ventura |
| Linux | Operating System | Open-source Unix-like OS | Ubuntu, CentOS, Debian |
| LAMP Stack | Web Development Stack | Linux, Apache, MySQL, PHP/Python/Perl | Web server environments |
| MEAN Stack | Web Development Stack | MongoDB, Express.js, Angular, Node.js | Full-stack JavaScript applications |
| MERN Stack | Web Development Stack | MongoDB, Express.js, React, Node.js | Full-stack JavaScript applications |
| JAMstack | Web Development Architecture | JavaScript, APIs, Markup | Static site generators, headless CMS |
| Node.js | Runtime Environment | JavaScript runtime for server-side scripting | Web servers, APIs |
| JVM | Runtime Environment | Executes Java bytecode on various platforms | Java applications |
| .NET Runtime | Runtime Environment | Executes .NET applications | .NET applications |
| React | Development Framework | UI library for building component-based interfaces | React applications |
| Angular | Development Framework | Full-featured framework for web applications | Angular applications |
| Vue.js | Development Framework | Progressive framework for building UIs | Vue.js applications |
| Django | Development Framework | High-level Python Framework for web development | Django applications |
| Flask | Development Framework | Lightweight Python framework for web applications | Flask applications |
| AWS | Cloud Platform | Comprehensive cloud computing services | EC2, S3, Lambda |
| Microsoft Azure | Cloud Platform | Cloud computing service with various capabilities | Azure VMs, Azure SQL Database |
| Google Cloud Platform (GCP) | Cloud Platform | Cloud computing services and tools | Compute Engine, Big Query |
| iOS | Mobile Development Platform | Apple’s mobile OS for iPhones and iPads | iOS apps |
| Android | Mobile Development Platform | Google’s mobile OS for various devices | Android apps |
| React Native | Mobile Development  Framework | Build mobile apps using React | Cross-platform  mobile apps |
| Flutter | Mobile Development  Framework | UI toolkit for building  natively compiled apps | Cross-platform apps |
| MySQL | Database Platform | Open-source relational  Database | MySQL database |
| PostgreSQL | Database Platform | Advanced open-source relational database | PostgreSQL database |
| MongoDB | Database Platform | NoSQL database with JSON-like document Storage | MongoDB database |
| Oracle  Database | Database Platform | Enterprise-level database management system | Oracle Database |

Frameworks

|  |  |  |  |
| --- | --- | --- | --- |
| **Framework** | **Language / Platform** | **Description** | **Example Usage** |
| React | JavaScript | UI library with component-  based architecture | react, react-dom |
| Angular | JavaScript / TypeScript | Full-featured framework for dynamic SPAs | @angular/core, @angular/cli |
| Vue.js | JavaScript | Progressive framework for building user interfaces | vue, vue-cli |
| Express.js | Node.js | Minimalist web framework for server-side applications | express |
| Next.js | JavaScript | Framework for server-side rendering and static site generation | next |
| Django | Python | High-level framework with built-in features | django |
| Flask | Python | Micro-framework for building small to medium  apps | flask |
| FastAPI | Python | Fast API framework with automatic documentation | fastapi |
| Laravel | PHP | PHP framework with elegant syntax and built-in tools | laravel/framework |
| Symfony | PHP | Full-stack framework with reusable components | symfony/symfony |
| CodeIgniter | PHP | Lightweight framework with minimal setup | codeigniter/framework |
| Spring | Java | Enterprise Java framework with extensive capabilities | spring-core, spring-web |
| Hibernate | Java | ORM framework for database operations | hibernate-core |
| ASP.NET  Core | .NET | Cross-platform framework for building web applications | Microsoft.AspNetCore.App |
| Entity | .NET | ORM framework for data | Microsoft.EntityFramework |
| Framework  Core |  | access | Core |
| NestJS | TypeScript /  JavaScript | Server-side framework with  modular architecture | @nestjs/core |

Libraries

|  |  |  |  |
| --- | --- | --- | --- |
| **Library** | **Language / Framework** | **Description** | **Example Usage** |
| React | JavaScript | UI library for building user interfaces | react, react-dom |
| jQuery | JavaScript | Simplifies DOM manipulation and event handling | jquery |
| Lodash | JavaScript | Utility functions for common programming tasks | lodash |
| Moment.js | JavaScript | Date and time manipulation | moment |
| Axios | JavaScript | HTTP client for making requests | axios |
| Bootstrap | CSS | CSS framework with pre- designed components | bootstrap |
| Bulma | CSS | Modern CSS framework based on Flexbox | bulma |
| Tailwind CSS | CSS | Utility-first CSS framework | tailwindcss |
| Django | Python | High-level web framework | django |
| Flask | Python | Lightweight web framework | flask |
| Pandas | Python | Data manipulation and analysis | pandas |
| Requests | Python | HTTP library for making requests | requests |
| Laravel | PHP | PHP framework with elegant syntax | laravel/framework |
| Symfony | PHP | PHP framework with reusable components | symfony/symfony |
| Composer | PHP | Dependency manager for PHP | composer |
| Express | Node.js | Minimalist web framework for Node.js | express |
| Mongoose | Node.js | ODM library for MongoDB | mongoose |
| Socket.IO | Node.js | Real-time communication library | socket.io |
| RxJS | Angular | Reactive programming library | rxjs |
| Angular Material | Angular | UI component library for Angular | @angular/material |
| NgRx | Angular | State management library | @ngrx/store |
| Sequelize | Node.js | ORM for SQL databases | sequelize |
| TypeORM | Node.js/TypeScript | ORM for SQL databases and MongoDB | typeorm |
| Peewee | Python | Small ORM for SQL databases | peewee |

Git & GitHub

1. Config

Git config --global user.name ""

Git config --global user.email ""

1. Create Repositories in GitHub
2. Take Clone
3. Add in Stage Area
4. Commit

git commit -m "any comments"

1. Push

git add commant

1. stage changess in specific files

git add filename

2.

create branch

-----check branch / check the current branch

C:\Users\SIHAN\Desktop\My Systems\MySystems>git branch

\* main

--- pull created branches

C:\Users\SIHAN\Desktop\My Systems\MySystems> git pull

From https://github.com/MSihan23/MySystems

\* [new branch] branch\_for\_david -> origin/branch\_for\_david

Already up to date.

---change to branch and chackout from first

git checkout branch\_for\_david

Switched to a new branch 'branch\_for\_david'

branch 'branch\_for\_david' set up to track 'origin/branch\_for\_david'.

---again go to main branch (the other brancers majed files not show to the main branch)

git checkout main

Switched to branch 'main'

Your branch is up to date with 'origin/main'.

--- if main want to see the othor branchers changed files , the main wan to merge the branches

git merge branch\_for\_david

Updating 65439c9..d8e387f

Fast-forward

fromdavit.html | 1 +

1 file changed, 1 insertion(+)

create mode 100644 fromdavit.html

---tranfer the changes to other branch

git push orgin feature-branch(recent branch):feature-branch(to branch)

-- display the commit history

git log

--- add note to discribe the commitdata

-- attach a notes

git notes add -m "notes" theNumbers

--show notes

git notes show <commit-SHA(number)>

--List notes

git notes 📃

--remove a notes

git notes remove <commit - SHA >

Common to all Languages

1. Operators
   1. Arithmetic Operator

|  |  |
| --- | --- |
| **Operator** | **Description** |
| + | Addition |
| - | Subtraction |
| \* | Multiplication |
| \*\* | Exponentiation |
| / | Division |
| % | Modulus (Remainder) |
| ++ | Increment |
| -- | Decrement |

* 1. Assignment Operators
     1. Assignment Operators

|  |  |
| --- | --- |
| = | Simple Assignment Operator |
| += | Addition Assignment Operator |
| -= | Subtraction Assignment Operator |
| \*= | Multiplication Assignment Operator |
| \*\*= | Exponentiation Assignment Operator |
| %= | Remainder Assignment Operator |
| /= | Division Assignment Operator |

* + 1. Shift Assignment Operators

|  |  |  |
| --- | --- | --- |
| <<= | Left Shift Assignment Operator | x = x << y |
| >>= | Right Shift Assignment Operator | x = x >> y |
| >>>= | Unsigned Right Shift Assignment Operator | x = x >>> y |

* + 1. Bitwise Assignment Operators

|  |  |  |
| --- | --- | --- |
| &= | Bitwise AND Assignment Operator | x = x & y |
| ^= | Bitwise XOR Assignment Operator | x = x ^ y |
| |= | Bitwise OR Assignment Operator | x = x | y |

* + 1. Logical Assignment Operators

|  |  |  |
| --- | --- | --- |
| &&= | Logical AND Assignment Operator | x = x && (x = y) |
| ||= | Logical Or Assignment Operator | x = x || (x = y) |
| ??= | Nullish Coalescing Assignment Operator | x = x ?? (x = y) |

* 1. Comparison Operators

|  |  |
| --- | --- |
| = = | Equal to |
| = = = | Equal value and equal type |
| ! = | Not equal |
| ! = = | Not equal value or not equal type |
| > | Greater than |
| < | Less than |
| > = | Greater than or equal to |
| < = | Less than or equal to |
| ? | Ternary operator |
| name ?? text; | Nullish Coalescing Operator (??) |
| car?.name | Optional Chaining Operator (?.) |

* 1. String Operators
* String Comparison
* String Addition(Concatenate)
* Adding Strings and Numbers
  1. Logical Operators

|  |  |
| --- | --- |
| && | logical and |
| || | logical or |
| ! | logical not |

* 1. Bitwise Operators

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Operator** | **Description** | **Example** | **Same as** | **Result** | **Decimal** |
| & | AND | 5 & 1 | 0101 & 0001 | 0001 | 1 |
| | | OR | 5 | 1 | 0101 | 0001 | 0101 | 5 |
| ~ | NOT | ~ 5 | ~0101 | 1010 | 10 |
| ^ | XOR | 5 ^ 1 | 0101 ^ 0001 | 0100 | 4 |
| << | Left Shift | 5 << 1 | 0101 << 1 | 1010 | 10 |
| >> | Right Shift | 5 >> 1 | 0101 >> 1 | 0010 | 2 |
| >>> | Unsigned Right Shift | 5 >>> 1 | 0101 >>> 1 | 0010 | 2 |

* 1. Ternary/Conditional Operators

(condition ? ifTrue : ifFalse)

* 1. Type Operators
     1. Converting Strings to Numbers

**The Unary + Operator**

The unary + operator can be used to convert a variable to a number:

let y = "5";

let x = + y;

|  |  |  |
| --- | --- | --- |
| **Language** | **Method** | **Example** |
| JavaScript | Number(),  parseInt(),  parseFloat(),  unary + operator | Number('42') → 42  parseInt('42') → 42  parseFloat('42.5') → 42.5  +'42' → 42 |
| Java | Integer.parseInt(), Double.parseDouble() | Integer.parseInt("42") → 42  Double.parseDouble("42.5") → 42.5 |
| Python | int(),  float() | int("42") → 42  float("42.5") → 42.5 |
| PHP | (int),  (float) cast | (int) "42" → 42  (float) "42.5" → 42.5 |
| Node.js | Same as JavaScript | Number('42') → 42  parseFloat('42.5') → 42.5 |
| React (JSX) | Same as JavaScript, often used in JSX expressions | { Number('42') } → 42 |
| Angular | Same as JavaScript | {{ Number('42') }} → 42 |
| MySQL | Implicit conversion,  CAST() | SELECT CAST('42' AS UNSIGNED) → 42  SELECT CAST('42.5' AS DECIMAL(10,2)) → 42.50 |
| MongoDB | $toInt,  $toDouble in aggregation pipelines | {$toInt: "$stringField"} → 42  {$toDouble: "$stringField"} → 42.5 |

* + 1. Converting Numbers to Strings

|  |  |
| --- | --- |
| **JavaScript** | |
| **Method** | **Description** |
| String() | Global Method |
| toString() |  |
| toExponential() | Returns a string, with a number rounded and written using exponential notation. |
| toFixed() | Returns a string, with a number rounded and written with a specified number of decimals. |
| toPrecision() | Returns a string, with a number written with a specified length |

* + 1. Converting Dates to Numbers

|  |  |
| --- | --- |
| **JavaScript** | |
| **Method** | **Description** |
| Number() | Global Method |
| getTime() |  |

* + 1. Converting Date to String

|  |  |
| --- | --- |
| **JavaScript** | |
| **Method** | **Description** |
| String() | Global Method String(Date()) |
| toString() | Date().toString() |
| getDate() | Get the day as a number (1-31) |
| getDay() | Get the weekday a number (0-6) |
| getFullYear() | Get the four-digit year (yyyy) |
| getHours() | Get the hour (0-23) |
| getMilliseconds() | Get the milliseconds (0-999) |
| getMinutes() | Get the minutes (0-59) |
| getMonth() | Get the month (0-11) |
| getSeconds() | Get the seconds (0-59) |
| getTime() | Get the time (milliseconds since January 1, 1970) |

* + 1. Converting Booleans to Numbers

|  |  |  |
| --- | --- | --- |
| **Language** | **Method** | **Example** |
| JavaScript | Use the Number() function or + operator | Number(true) → 1, Number(false) → 0  +true → 1, +false → 0 |
| Java | Implicit conversion via ternary or arithmetic | true ? 1 : 0 → 1, false ? 1 : 0 → 0  true == 1 (boolean comparison) |
| Python | Implicit conversion in arithmetic operations or int() function | int(True) → 1, int(False) → 0 |
| PHP | Implicit conversion in arithmetic operations or (int) cast | (int) true → 1, (int) false → 0 |
| Node.js | Same as JavaScript (Number() or + operator) | Number(true) → 1, +false → 0 |
| React (JSX) | Same as JavaScript (within JSX expressions) | { Number(true) } → 1 |
| Angular | Implicit conversion within expressions | {{ Number(true) }} → 1 |
| MySQL | Use CAST() or implicit conversion in arithmetic | CAST(TRUE AS UNSIGNED) → 1, CAST(FALSE AS UNSIGNED) → 0 |
| MongoDB | Use $cond in aggregation pipelines | {$cond: { if: true, then: 1, else: 0 }} → 1 |

* + 1. Converting Booleans to String

|  |  |  |
| --- | --- | --- |
| **JavaScript** | | |
| **Method** | **Description** | |
| String() | Global Method | String(false)      // returns "false" |
| toString() |  | false.toString()   // returns "false" |

1. Concatenation

|  |  |  |
| --- | --- | --- |
| **Language** | **Concatenation Method** | **Example** |
| JavaScript | + | 'Hello' + ' World' → 'Hello World'  `Hello ${name}` → Hello John |
| Java | +  concat() method | "Hello".concat(" World") → "Hello World" |
| Python | +  formatted strings (f-strings) | 'Hello' + ' World' → 'Hello World'  f"Hello {name}" → Hello John |
| PHP | . (dot) | 'Hello' . ' World' → 'Hello World' |
| Node.js | + | 'Hello' + ' World' → 'Hello World' |
| React (JSX) | {} | { "Hello" + " World" } → Hello World |
| Angular | + | {{ 'Hello' + ' World' }} → Hello World |
| MongoDB | $concat | {$concat: ["Hello", " World"]} → 'Hello World' |
| MySQL | CONCAT() | SELECT CONCAT('Hello', ' World'); → 'Hello World' |

1. Get Output

|  |  |  |
| --- | --- | --- |
| **JavaScript** | | |
| innerHTML | Writing into an HTML element | document.getElementById ("demo").innerHTML = 5; |
| document.write() | Writing into the HTML output | onclick="document.write(5 + 6)" |
| window.alert() | Writing into an alert box | window.alert(5 + 6); |
| window.print() | Print the content of the current window | <button onclick="window.print()">Print this page</button> |
| console.log() | Writing into the browser console |  |
| **Python** | | |
| print() | Writing to the standard output (console) | print(5 + 6) |
| **Java** | | |
| System.out.println() | Writing to the standard  output (console) | System.out.println(5 + 6); |
| System.out.printf() | Writing formatted output to  the console | System.out.printf("Sum: %d%n", 5 +  6); |
| **PHP** | | |
| echo | Writing to the standard  output (web page) | <?php echo 5 + 6; ?> |
| print | Writing to the standard  output (web page) | <?php print 5 + 6; ?> |
| **ReactJS (JSX)** | | |
| return | Writing into the component’s rendered  output | return <div>{5 + 6}</div>; |
| **NodeJS** | | |
| console.log() | Writing into the terminal or  command line | console.log(5 + 6); |
| **Laravel (Blade)** | | |
| {{ }} | Writing into the Blade  template output | {{ 5 + 6 }} |
| **MySQL** | | |
| SELECT | Writing into the query result | SELECT 5 + 6; |
| **TypeScript** | | |
| console.log() | Writing into the browser  console | console.log(5 + 6); |
| **Angular** | | |
| console.log() | Writing into the browser  console | console.log(5 + 6); |
| **Vue.js** | | |
| {{ }} | Writing into the  component’s rendered output | <div>{{ 5 + 6 }}</div> |
| **jQuery** | | |
| console.log() | Writing into the browser  console | console.log(5 + 6); |
| alert() | Writing into an alert box | alert(5 + 6); |
| **Swift** | | |
| print() | Writing to the standard  output (console) | print(5 + 6) |
| **Kotlin** | | |
| println() | Writing to the standard  output (console) | println(5 + 6) |

1. Key Words

|  |  |  |
| --- | --- | --- |
| **Function** | | |
| JavaScript | | Function |
| **Variable** | | |
| JavaScript | | Variable |
| **Object** | | |
| JavaScript | const person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"}; | |
|  | |  |

|  |  |
| --- | --- |
| **Common** | |
| Parentheses | (Parameters) |
| Mutable | They are addressed by reference, not by value. A mutable object is an object whose state can be changed after it is created. |
| Immutable | An immutable object is an object whose state cannot be modified after it is created |
| Operator | An operator is a symbol (or a combination of symbols) that tells the compiler or interpreter to perform a specific mathematical, logical, or other types of operations on one or more values. These operations could range from basic arithmetic to more complex comparisons or manipulations of data. |
| Operant | An operand is a value or variable that an operator acts on. Operands are essentially the data that is manipulated by the operators. |
| Predicate function | Function that returns a Boolean value (true or false) based on some condition. |
| Sparse array | That has some elements missing or empty (i.e., undefined values at certain indexes). |
| Array Iterator | JavaScript is an object that allows you to traverse (or iterate) over the elements of an array in sequence, one by one. It is returned by various array methods like keys(), values(), and entries() that generate these iterators. |

1. Variables

Fixed values are called **Literals**.

Variable values are called **Variables**.

|  |  |  |
| --- | --- | --- |
| **Language** | **Variable Declaration** | **Description** |
| JavaScript | let variableName = value; | Declare a variable with block scope |
|  | const variableName = value; | Declare a constant (immutable) |
| Python | variableName = value | Declare a variable (dynamically typed) |
| Java | type variableName = value; | Declare a variable with explicit type |
| PHP | $variableName = value; | Declare a variable (global scope) |
| ReactJS (JSX) | const variableName = value; | Declare a constant in a React component |
|  | let variableName = value; | Declare a variable with block scope |
| NodeJS | let variableName = value; | Declare a variable with block scope |
|  | const variableName = value; | Declare a constant (immutable) |
| Laravel (Blade) | $variableName = value; | Declare a variable in a Blade template |
| MySQL | SET @variableName = value; | Declare a user-defined variable |
| TypeScript | let variableName: type = value; | Declare a variable with a type of annotation |
|  | const variableName: type = value; | Declare a constant with a type of annotation |
| Angular | variableName: type = value; | Declare a class property or component variable |
| Vue.js | data() { return { variableName: value } } | Declare reactive data properties |
| Swift | var variableName = value | Declare a variable (mutable) |
|  | let variableName = value | Declare a constant (immutable) |
| Kotlin | val variableName = value | Declare a read-only variable |
|  | var variableName = value | Declare a mutable variable |
| Perl | $variableName = value; | Declare a scalar variable (global  scope) |

1. Comments

|  |  |  |
| --- | --- | --- |
| **Language** | **Single Line Comment** | **Multi Line Comment** |
| JavaScript | // | /\* ... \*/ |
| Python | # | ''' ... ''' or """ ... """ |
| Java | // | /\* ... \*/ |
| PHP | // or # | /\* ... \*/ |
| HTML | <!-- ... --> | <!-- ... --> |
| CSS | /\* ... \*/ | /\* ... \*/ |
| ReactJS (JSX) | {/\* ... \*/} | {/\* ... \*/} |
| NodeJS | // | /\* ... \*/ |
| Laravel (Blade) | {{-- ... --}} | {{-- ... --}} |
| MySQL | -- or # | /\* ... \*/ |
| TypeScript | // | /\* ... \*/ |
| Angular | // | /\* ... \*/ |
| Vue.js | // | /\* ... \*/ |
| SASS/SCSS | // | /\* ... \*/ |
| Less | // | /\* ... \*/ |
| jQuery | // | /\* ... \*/ |
| Swift | // | /\* ... \*/ |
| Kotlin | // | /\* ... \*/ |

1. Syntax and Semantics

* **Syntax**: Refers to the rules and structure of code in a programming language. It dictates how code must be written to be understood by the compiler or interpreter.
* **Semantics**: Refers to the meaning behind the code. It is about what the code does and how it behaves when executed.

|  |  |  |  |
| --- | --- | --- | --- |
| **Language** | **Syntax** | **Semantics** | **Example** |
| JavaScript | let x = 5; | Assigns the value 5 to variable x | let x = 5; |
| Python | x = 5 | Assigns the value 5 to variable x | x = 5 |
| Java | int x = 5; | Declares an integer variable x with value 5 | int x = 5; |
| PHP | $x = 5; | Assigns the value 5 to variable $x | $x = 5; |
| ReactJS (JSX) | const x = 5; | Declares a constant x with value 5 | const x = 5; |
| NodeJS | let x = 5; | Assigns the value 5 to variable x | let x = 5; |
| Laravel (Blade) | $x = 5; | Assigns the value 5 to variable $x | $x = 5; |
| MySQL | SET @x = 5; | Sets the user-defined variable @x to 5 | SET @x = 5; |

1. Primitive Data Type

* Primitive values are immutable.

|  |  |  |
| --- | --- | --- |
| **Float vs Double** | | |
| **Feature** | **float** | **double** |
| Precision | Single precision (32 bits) | Double precision (64 bits) |
| Memory Size | 4 bytes (32 bits) | 8 bytes (64 bits) |
| Range | Approximately ±3.4e−38 to ±3.4e38 | Approximately ±1.7e−308 to ±1.7e308 |
| Significant Digits | ~6-7 decimal digits | ~15-16 decimal digits |
| Speed | Faster due to lower precision | Slower due to higher precision |
| Usage | Suitable for less precise calculations where memory is a concern (e.g., graphics, games) | Suitable for more precise calculations (e.g., scientific computations, financial calculations) |
| Default (in some languages) | Some languages default to float for floating-point literals | Many languages, including Java and C++, default to double for floating-point literals |

|  |  |
| --- | --- |
| **JavaScript Primitives** | |
| String | Represents text (strings are treated as primitives). |
| Number | Used for both integers and floating-point numbers. |
| Bigint |  |
| Boolean | Represents true or false. |
| Undefined | Variable did not assign any value. |
| Null | Represents the absence of a value. |
|  | |
| **Java** | |
| byte | 8-bit integer |
| short | 16-bit integer |
| int | 32-bit integer |
| long | 64-bit integer |
| float | 32-bit floating-point number |
| double | 64-bit floating-point number |
| Boolean | true/false value |
| char | 16-bit Unicode character |
|  | |
| **Python** | |
| int | Integer (unlimited size) |
| float | Floating-point number |
| bool | Boolean |
| str | String |
| None | Represents null or no value (special singleton) |
|  | |
| **MySQL** | |
| TINYINT | 8-bit integer |
| SMALLINT | 16-bit integer |
| INT | 32-bit integer |
| BIGINT | 64-bit integer |
| FLOAT | Single-precision floating-point |
| DOUBLE | Double-precision floating-point |
| CHAR | Fixed-length string |
| VARCHAR | Variable-length string |
| BOOLEAN | Represents true or false values |

1. Escape Characters

|  |  |  |
| --- | --- | --- |
| **Code** | **Result** | **Description** |
| \' | ' | Single quote |
| \" | " | Double quote |
| \\ | \ | Backslash |
| \n | Newline | Newline (moves to the next line) |
| \t | (tab) | Tab (inserts a horizontal tab space) |
| \r | Carriage return | Moves the cursor to the beginning of the line |
| \b | Backspace | Deletes the previous character |
| \f | Form feed | Moves to the next page (used in printing) |
| \0 | Null | Null character (end of a string) |
| \v | Vertical tab | Inserts a vertical tab |
| \xhh | Character | Inserts a character based on its hexadecimal code (hh) |
| \uhhhh | Unicode | Inserts a Unicode character with code hhhh |

1. Arrays

Associative Arrays

Arrays with named indexes are called associative arrays (or hashes).

person["firstName"] = "John";

Languages

1. JavaScript
   1. JS Objects

https://www.w3schools.com/jsref/jsref\_obj\_object.asp

* Objects are mutable

const car = {type:"Fiat", model:"500", color:"white"};

Object ==> car

Properties ==> car.type:"Fiat" car.model:"500" car.color:"white"

**key:value**

**Object Methods**

const person = {

firstName: "John",

lastName : "Doe",

id : 5566,

fullName : function() {

return this.firstName + " " + this.lastName;

}

};

**Accessing Object Properties**

* objectName.propertyName
* objectName["propertyName"]

**Adding New Properties**

* person.nationality = "English";

**Deleting Properties**

* delete person.age;
* delete person["age"];

**Access nested objects**

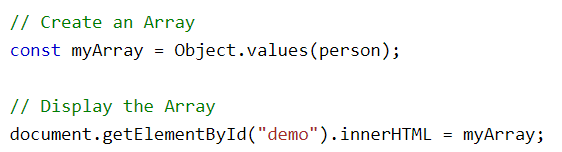
* myObj.myCars.car2;
* myObj.myCars["car2"];
* myObj["myCars"]["car2"];
* let p1 = "myCars"**;** let p2 = "car2"**;** myObj[p1][p2]**;**

**Accessing Object Methods**

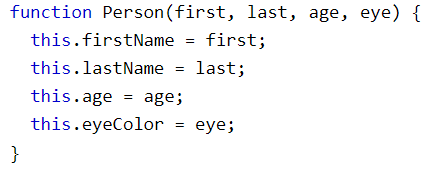
* objectName.methodName()

**Displaying Object Properties**

* A computer code with black text

  Description automatically generatedUsing Object.values()
* Using Object.entries()
  + 1. JavaScript Object Constructors

A group of names on a white background

Description automatically generatedIf we want to create some objects in same type, then we can initiate the object type in one main object and we call the main object as Object Constructor.

**Adding a Property to an Object**

* Person.prototype.nationality = "English";

|  |  |
| --- | --- |
| new Object() | A new Object object |
| new Array() | A new Array object |
| new Map() | A new Map object |
| new Set() | A new Set object |
| new Date() | A new Date object |
| new RegExp() | A new RegExp object |
| new Function() | A new Function object |

* 1. JavaScript Events

**HTML Events**

* An HTML web page has finished loading.
* The HTML input field was changed.
* An HTML button was clicked.

**Event Handler Attributes**

* <element event='some JavaScript'>
* <element event="some JavaScript">

**Common HTML Events**

|  |  |
| --- | --- |
| **Event** | **Description** |
| onchange | An HTML element has been changed |
| onclick | The user clicks an HTML element |
| onmouseover | The user moves the mouse over an HTML element |
| onmouseout | The user moves the mouse away from an HTML element |
| onkeydown | The user pushes a keyboard key |
| onload | The browser has finished loading the page |

* 1. Data Types

|  |  |  |
| --- | --- | --- |
| **JavaScript Primitives** | | |
| String |  | |
| Number |  | |
| Bigint |  | |
| Boolean |  | |
| Undefined |  | |
| Null |  | |
| Symbol |  | |
| **Exponential Notation**  let y = 123e5; // 12300000  let z = 123e-5; // 0.00123 | |

* + 1. JS Strings

**Note:**

* Template Strings

Templates are strings enclosed in backticks (`This is a template string`).

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **String Basic Methods** | | | | |
| **String Methods** | **Definition** | | | **Code** |
| String length |  | | |  |
| **Extracting String Characters** | | | | |
| String charAt() | Returns the **character** at a specified index | | | text.charAt(0); |
| String charCodeAt() | Returns the **code** of the character at a specified index | | | text.charCodeAt(0); |
| String at() | It allows the use of negative indexes while charAt() do not. | | | name.at(-2);  name[2]; |
| String [ ] | read only. str[0] = "A" gives no error (but does not work!) | | | text[0] |
| **Extracting String Parts** | | | | |
| String slice() | Start position, and end position (end not included). | | | text.slice(7, 13);  text.slice(-12, -6); |
| String substring() | Similar to slice().  Start and end values less than 0 are treated as 0 | | | str.substring(7, 13); |
| String substr() | Similar to slice(). | | | str.substr(7, 6); |
|  | | | | |
| String toUpperCase() |  | | | text1.toUpperCase(); |
| String toLowerCase() |  | | | text1.toLowerCase(); |
| String concat() | Joins two or more strings | | text1.concat(" ", text2); | |
| String trim() | Removes whitespace from both sides of a string | | | text1.trim() |
| String trimStart() | Removes whitespace only from the start of a string | | | text1.trimStart(); |
| String trimEnd() | Removes whitespace only from the end of a string. | | | text1.trimEnd(); |
| String padStart() | To pad a number, convert the number to a string first xxxx5 | | | text.padStart(4,"x"); |
| String padEnd() | 5xxxx | | | text.padEnd(4,"x") |
| String repeat() | Returns a string with a number of copies of a string. | | | text.repeat(2); |
| String replace() | Replaces a specified value with another value in a string | text.replace  ("Microsoft", "W3Schools"); | | |
| String replaceAll() |  | text.replaceAll  ("cats","dogs"); | | |
| String split() | A string can be converted to an array with the split() method | const myArray = text.split(","); | | |

**Note:**

* /i -Insensitive

Replace case insensitive, use a **regular expression**

(/MICROSOFT/i, "W3Schools");

* /g flag (global match)

Replace all matches, use a **regular expression**

(/MICROSOFT/i, "W3Schools");

Both

**text.match(/ain/gi)**

|  |  |  |
| --- | --- | --- |
| **JavaScript String Search** | | |
| **String Methods** | **Definition** | **Code** |
| String indexOf() | second parameter is optional | text.indexOf("locate", 15); |
| String lastIndexOf() | second parameter is optional | text.lastIndexOf("locate"); |
| String search() | Can use regular expressions | text.lastIndexOf("locate"); |
| String match() | Returns an **array** containing the results of matching a string against a string | text.match(/ain/gi); |
| String matchAll() | Returns an iterator containing the results of matching a string against a string | text.matchAll("Cats"); |
| String includes() | Returns true if a string contains a specified value. | text.includes("world");  text.includes("world", 12); |
| String startsWith() | Returns true if a string begins with a specified value. | text.startsWith("Hello");  text.startsWith("world", 5) |
| String endsWith() | Returns true if a string ends with a specified value. | text.endsWith("Doe");  text.endsWith("world", 11); |

|  |  |
| --- | --- |
| **String Templates / Template Strings / Template Literals** | |
| Quotes Inside Strings | let text = `He's often called "Johnny"`; |
| Multiline Strings |  |
| String Interpolation | Template String provides an easy way to interpolate variables and expressions into strings.  The method is called string interpolation. |
| Variable Substitutions |  |
| Expression Substitution | let total = `Total: ${(price \* (1 + VAT)).toFixed(2)}`; |
| Html Templates |  |

* + 1. JS Numbers

|  |  |  |
| --- | --- | --- |
| **Value (aka Fraction/Mantissa)** | **Exponent** | **Sign** |
| 52 bits (0 - 51) | 11 bits (52 - 62) | 1 bit (63) |

|  |  |  |
| --- | --- | --- |
| **Numbers Basic Methods** | | |
| **Method** | **Description** | **Code** |
| toString() | Returns a number as a string | x.toString(); |
| toExponential() | Returns a number written in exponential notation. parameter defines the number of characters behind the decimal point | let x = 9.656;  x.toExponential(2)  =>9.66e+0 |
| toFixed() | Returns a number written with a number of decimals | x.toFixed(0) |
| toPrecision() | Returns a number written with a specified length | x.toPrecision(4);  => 9.656 |
| valueOf() | Returns a number as a number |  |
|  | | |
| **Converting Variables to Numbers** | | |
| Number() | Returns a number converted from its argument. |  |
| parseFloat() | Parses its argument and returns a floating point number |  |
| parseInt() | Parses its argument and returns a whole number |  |
|  | | |
| **Number Object Methods** | | |
| Number.isInteger() | Returns true if the argument is an integer |  |
| Number.isSafeInteger() | Returns true if the argument is a safe integer |  |
| Number.parseFloat() | Converts a string to a number |  |
| Number.parseInt() | Converts a string to a whole number |  |
|  | | |

* 1. Array

|  |  |  |
| --- | --- | --- |
| **Method** | **Description** | **Code** |
| Array length | property returns the length (size) of an array | fruits.length |
| Array toString() | converts an array to a string of (comma separated) array values. | fruits.toString(); |
| Array at() | The at() method returns the same as [] | fruits[2] |
| Array join() | It behaves just like toString(), but in addition you can specify the separator | fruits.join(" \* ") |
| Array pop() | removes the last element from an array | fruits.pop(); |
| Array push() | adds a new element to an array | fruits.push("Kiwi"); |
| Array shift() | removes the first array element and "shifts" all other elements to a lower index. | fruits.shift(); |
| Array unshift() | adds a new element to an array (at the beginning), | fruits.unshift("Lemon"); |
| Array delete() | Can delete a certain index after the index in undefined. | delete fruits[0]; |
| Array concat() |  | const myChildren = arr1.concat(arr2, arr3); |
| Array copyWithin() | copies array elements to another position in an array: | fruits.copyWithin(2, 0);  fruits.copyWithin(2, 0, 2); |
| Array flat() | creates a new array with sub-array elements concatenated to a specified depth. | myArr = [[1,2],[3,4],[5,6]]; newArr = myArr.flat(); |
| flatMap() | first maps all elements of an array and then creates a new array by flattening the array. | myArr.flatMap(x => [x, x \* 10]); |
| Array splice() | (2) defines the position **where** new elements should be **added.**  (0) defines **how many** elements should be **removed**. | fruits.splice(2, 0, "1", "2");  fruits.splice(0, 1); |
| Array toSpliced() | new method creates a new array, keeping the original array unchanged | months.toSpliced(0, 1); |
| Array slice() |  | fruits.slice(2);  fruits.slice(1, 3); |
|  | | |
| **Array Find and Search Methods** | | |
| Array indexOf() | searches an array for an element value and returns its position. | *array*.indexOf(*item*,*start*) |
| Array lastIndexOf() | returns the position of the last occurrence of the specified element. | *array*.lastIndexOf(*item*,*start*) |
| Array includes() | This allows us to check if an element is present in an array | *array*.includes(*search-item*) |
| Array find() | returns the value of the first array element that passes a test function. | numbers.find(myFunction);  myFunction(value, index, array) |
| Array findIndex() | returns the index of the first array element that passes a test function. | numbers.findIndex( myFunction);  myFunction(value, index, array) |
| Array findLast() | hat will start from the end of an array and return the value of the first element | temp.findLast(x => x > 40); |
| Array findLastIndex() | finds the index of the last element that satisfies a condition | temp.findLastIndex(x => x > 40); |
|  | | |
| **Array Sort Methods** | | |
| **Alpabetic Sort** | |  |
| Array sort() | method sorts an array alphabetically | fruits.sort(); |
| Array reverse() | method reverses the elements in an array | fruits.reverse(); |
| Array toSorted() | sort an array without altering the original array. | months.toSorted(); |
| Array toReversed() | reverse an array without altering the original array. | months.toReversed(); |
| Sorting Objects |  | |
| **Numeric Sort** | |  |
| Numeric Sort | points.sort(function(a, b){return a - b}); --- Accenting  points.sort(function(a, b){return b - a}); --- Descending | |
| Random Sort | points.sort(function(){return 0.5 - Math.random()}); -- wrong so want to use Fisher Yates shuffle metode | |
| Math.min() |  | |
| Math.max() |  | |
| Home made Min() |  | |
| Home made Max() |  | |
|  | | |
| **Array Iteration Methods** | | |
| Array forEach | callback function | numbers.forEach(function(value) { console.log(value); });  function myFunction(value, index, array) { } |
| Array map() |  | |
| Array flatMap() | const newArr = myArr.flatMap((x) => x \* 2); | |
| Array filter() | function myFunction(value, index, array) {  return value > 18;  } | |
| Array  reduce() | function myFunction(total, value, index, array) {   return total + value; }  **total**: This is the accumulator. It stores the result of the previous iterations. During the first iteration, if no initial value is passed, total will be the first element of the array. | |
| Array reduceRight() |
| Array every() | function myFunction(value, index, array) {   return value > 18; }  check if **every / some** element in an array meets a specific condition (predicate function). | |
| Array some() |
| Array from() | returns an Array object from any object with a length property or any inerrable object | Array.from("ABCDEFG"); |
| Array keys() | returns an Array Iterator object with the keys of an array./index | const keys = fruits.keys(); |
| Array entries() | returns an Array Iterator object with key/value pairs: | fruits.entries();  [2, "Apple"] |
| Array with() | update elements in an array without altering the original array. | months.with(2, "March"); |
| Array Spread (...) |  | |

* 1. Math Function

|  |  |
| --- | --- |
| **Math Properties (Constants)** | |
| Math.E | Returns Euler's number |
| Math.PI | Returns PI |
| Math.SQRT2 | Returns the square root of 2 |
| Math.SQRT1\_2 | Returns the square root of 1/2 |
| Math.LN2 | Returns the natural logarithm of 2 |
| Math.LN10 | Returns the natural logarithm of 10 |
| Math.LOG2E | Returns base 2 logarithm of E |
| Math.LOG10E | Returns base 10 logarithm of E |
|  | |
| **Math Methods** | |
| Math.round(x) | Returns x rounded to its nearest integer |
| Math.ceil(x) | Returns x rounded **up** to its nearest integer |
| Math.floor(x) | Returns x rounded **down** to its nearest integer |
| Math.trunc(x) | Returns the integer **part** of x (new in ES6) |
| Math.sign(x) | Returns if x is negative, null or positive (-1,0,1) |
| Math.pow(x, y) | returns the value of x to the power of y |
| Math.sqrt(x) | returns the square root of x |
| Math.abs(x) | returns the absolute (positive) value of x |
| Math.sin(x)  Math.sin(90 \* Math.PI / 180); | returns the sine (a value between -1 and 1) of the angle x (given in radians) |
| Math.cos(x)  Math.cos(0 \* Math.PI / 180); | returns the cosine (a value between -1 and 1) of the angle x (given in radians) |
| Math.min(0, 150, 30, 20, -8, -200); | Math.min() and Math.max() can be used to find the lowest or highest value in a list of arguments |
| Math.max(0, 150, 30, 20, -8, -200); |
| Math.log(x) | returns the natural logarithm of x |
|  |  |
|  |  |
|  |  |
|  |  |