**Week 4**

**HTML**

* HTML stands for Hyper Text Markup Language
* HTML is the standard markup language for Web pages
* HTML **elements** are the building blocks of HTML pages
* HTML elements are represented by **<> tags**

**CSS**

* CSS stands for Cascading Style Sheets
* CSS describes how HTML elements are to be displayed on screen, paper, or in other media
* CSS saves a lot of work. It can control the layout of multiple web pages all at once
* External style sheets are stored in CSS files

# CSS Syntax

A CSS rule consists of a selector and a declaration block.



The selector points to the HTML element you want to style.

The declaration block contains one or more declarations separated by semicolons.

Each declaration includes a CSS property name and a value, separated by a colon.

Multiple CSS declarations are separated with semicolons, and declaration blocks are surrounded by curly braces.

<html><head>

<style>

body {

background-color: lightblue;

}

h1 {

color: white;

text-align: center;

}

p {

font-family: verdana;

font-size: 20px;

}

</style>

</head>

<body>

<h1>My First CSS Example</h1>

<p>This is a paragraph.</p>

</body>

</html>

# CSS Selectors

A CSS selector selects the HTML element(s) you want to style.

We can divide CSS selectors into five categories:

* Simple selectors (select elements based on name, id, class)
* [Combinator selectors](https://www.w3schools.com/css/css_combinators.asp) (select elements based on a specific relationship between them)
* [Pseudo-class selectors](https://www.w3schools.com/css/css_pseudo_classes.asp) (select elements based on a certain state)
* [Pseudo-elements selectors](https://www.w3schools.com/css/css_pseudo_elements.asp) (select and style a part of an element)
* [Attribute selectors](https://www.w3schools.com/css/css_attribute_selectors.asp) (select elements based on an attribute or attribute value)

## The CSS element Selector

The element selector selects HTML elements based on the element name.

<html><head>

<style>

p {

text-align: center;

color: red;

}

</style></head>

<body>

<p>Every paragraph will be affected by the style.</p>

<p id="para1">Me too!</p>

<p>And me!</p>

</body></html>

## The CSS id Selector

The id selector uses the id attribute of an HTML element to select a specific element.

To select an element with a specific id, write a hash (#) character, followed by the id of the element.

<html><head>

<style>

#para1 {

text-align: center;

color: red;

}

</style>

</head>

<body>

<p id="para1">Hello World!</p>

<p>This paragraph is not affected by the style.</p>

</body></html>

## The CSS class Selector

The class selector selects HTML elements with a specific class attribute.

To select elements with a specific class, write a period (.) character, followed by the class name.

<html><head><style>

.center {

text-align: center;

color: red;

}

</style>

</head>

<body>

<h1 class="center">Red and center-aligned heading</h1>

<p class="center">Red and center-aligned paragraph.</p>

</body></html>

## The CSS Universal Selector

The universal selector (\*) selects all HTML elements on the page.

<!DOCTYPE html>

<html><head>

<style>

\* {

text-align: center;

color: blue;

}

</style></head>

<body>

<h1>Hello world!</h1>

<p>Every element on the page will be affected by the style.</p>

<p id="para1">Me too!</p>

<p>And me!</p>

</body></html>

## The CSS Grouping Selector

The grouping selector selects all the HTML elements with the same style definitions.

Look at the following CSS code (the h1, h2, and p elements have the same style definitions):

<html>

<head>

<style>

h1, h2, p {

text-align: center;

color: red;

}

</style>

</head>

<body>

<h1>Hello World!</h1>

<h2>Smaller heading!</h2>

<p>This is a paragraph.</p>

</body>

</html>

**Three Ways to Insert CSS**

There are three ways of inserting a style sheet:

* External CSS
* Internal CSS
* Inline CSS

**External CSS**

With an external style sheet, you can change the look of an entire website by changing just one file

Each HTML page must include a reference to the external style sheet file inside the <link> element, inside the head section.

<html>

<head>

<link rel="stylesheet" href="mystyle.css">

</head>

<body>

<h1>This is a heading</h1>

<p>This is a paragraph.</p>

</body>

</html>

**mystyle.css**

body {  
  background-color: lightblue;  
}  
h1 {  
  color: navy;  
  margin-left: 20px;  
}

## Internal CSS

An internal style sheet may be used if one single HTML page has a unique style.

The internal style is defined inside the <style> element, inside the head section.

<html>

<head>

<style>

body {

background-color: linen;

}

h1 {

color: maroon;

margin-left: 40px;

}

</style>

</head>

<body>

<h1>This is a heading</h1>

<p>This is a paragraph.</p>

</body>

</html>

## Inline CSS

An inline style may be used to apply a unique style for a single element.

To use inline styles, add the style attribute to the relevant element. The style attribute can contain any CSS property.

<html><body>

<h1 style="color:blue;text-align:center;">This is a heading</h1>

<p style="color:red;">This is a paragraph.</p>

</body>

</html>

**What is JavaScript?**

* An object-oriented computer programming language commonly used to create interactive effects within web browsers.
* JavaScript is **a text-based programming language used both on the client-side and server-side**.
* Where HTML and CSS are languages that give structure and style to web pages, JavaScript gives web pages interactive elements that engage a user
* JavaScript programs can be inserted almost anywhere into an HTML document using the <script> tag.

<html>

<body>

<p>Before the script...</p>

<script>

alert( 'Hello, world!' );

</script>

<p>...After the script.</p>

</body>

</html>

**JavaScript Can Change HTML Content**

One of many JavaScript HTML methods is getElementById().

<html>

<body>

<h2>What Can JavaScript Do?</h2>

<p id="demo">JavaScript can change HTML content.</p>

<button type="button" onclick='document.getElementById("demo").innerHTML = "Hello JavaScript!"'>Click Me!</button>

</body>

</html>

**JavaScript Statements**

* In a programming language, the list of programming instructions are called **statements**.
* **A JavaScript program** is a list of programming **statements.**
* JavaScript statements are composed of: Values, Operators, Expressions, Keywords, and Comments.

**Comments**

Single line comments start with //

Multi-line comments start with /\* and end with \*/

**Variables**

Variables are containers for storing data (storing data values).

In this example, x, y, and z, are variables, declared with the var keyword:

var x = 5;

var y = 6;

### Declaring a JavaScript Variable

* Using var
* Using let
* Using const

<html>

<body>

<h1>JavaScript Variables</h1>

<p>In this example, x, y, and z are variables.</p>

<p id="demo"></p>

<script>

var x=5;

var y = 6;

var z = x + y;

document.getElementById("demo").innerHTML ="The value of z is: " + z;

</script>

</body>

</html>

**Practice:** <script>

var x = 5;

var y = 6;

var z = x + y;

document.getElementById("demo").innerHTML ="The value of z is: " + z;

</script>

**Practice:** <script>

const price1 = 5;

const price2 = 6;

let total = price1 + price2;

document.getElementById("demo").innerHTML ="The total is: " + total;

</script>

**Constants:** Variables defined with const cannot be Redeclared and cannot be Reassigned.

const PI = 3.14;

**Data types**

|  |  |  |
| --- | --- | --- |
| DataTypes | Description | Example |
| String | represents textual data | 'hello', "hello world!" etc |
| Number | an integer or a floating-point | 3, 3.234, 3e-2 etc. |
| BigInt | an integer with arbitrary precision | 900719925124740999n , 1n etc. |
| Boolean | Any of two values: true or false | true and false |
| undefined | a data type whose variable is not initialized | let a; |
| null | denotes a null value | let a = null; |
| Symbol | unique and immutable | let value = Symbol('hello'); |
| Object | key-value pairs of collection of data | let student = { }; |

# Interaction

## [alert](https://javascript.info/alert-prompt-confirm" \l "alert)

## This one we’ve seen already. It shows a message and waits for the user to press “OK”

<script>

alert('HI there'); // with specified content

alert(); // without any specified content

</script>

## [prompt](https://javascript.info/alert-prompt-confirm#prompt)

The function prompt accepts two arguments:

<script>

// prompt example

let age = prompt('How old are you?');

alert(`You are ${age} years old!`);

</script>

## [confirm](https://javascript.info/alert-prompt-confirm" \l "confirm)

The syntax:

result = confirm(question);

The function confirm shows a modal window with a question and two buttons: OK and Cancel.

<script>

// confirm example

let isHappy = confirm('Are you Happy?');

alert(`You are ${isHappy}`);

</script>

**Types of JavaScript Operators**

There are different types of JavaScript operators:

* Arithmetic Operators
* Assignment Operators
* Comparison Operators
* Logical Operators
* Conditional Operators
* Type Operators

Comparisons

| **Operator** | **Meaning** |
| --- | --- |
| < | less than |
| > | greater than |
| <= | less than or equal to |
| >= | greater than or equal to |
| == | equal to |
| != | not equal to |

**Control flow**

Control flow in JavaScript is how your computer runs code from top to bottom. It starts from the first line and ends at the last line, unless it hits any statement that changes the control flow of the program such as loops, conditionals, or functions.

**Functions**

A JavaScript function is a block of code designed to perform a particular task.

A JavaScript function is executed when "something" invokes it (calls it).

<html>

<body>

<script>

function msg(){

alert("hello! this is message");

}

</script>

<input type="button" onclick="msg()" value="call function"/>

</body>

</html>

<html>

<body>

<script>

function getcube(number){

alert(number\*number\*number);

}

</script>

<form>

<input type="button" value="click" onclick="getcube(4)"/>

</form>

</body>

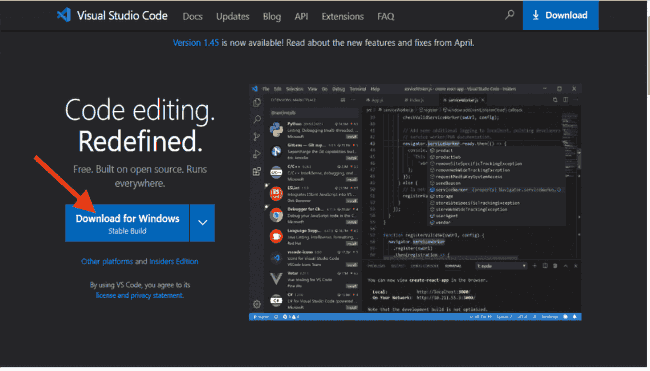
</html>

**Setting Up the Environment and Tools for front end development**

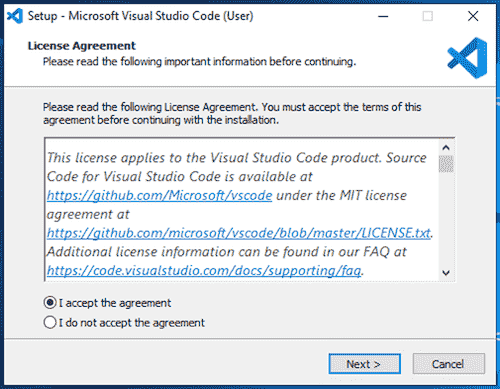
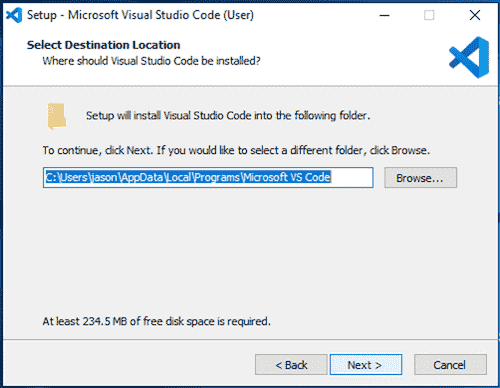
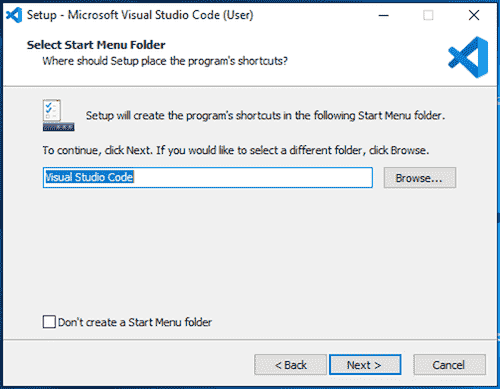
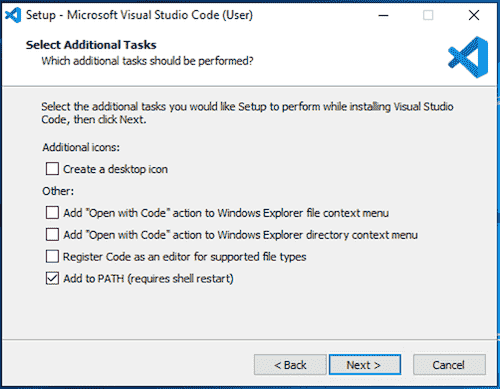
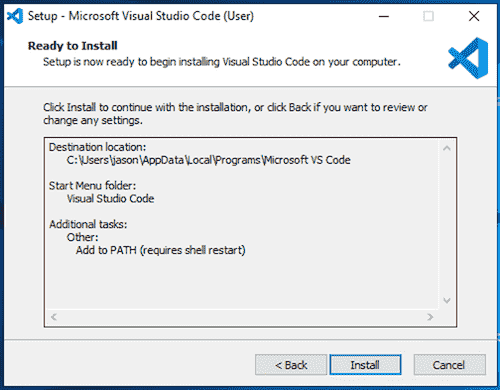
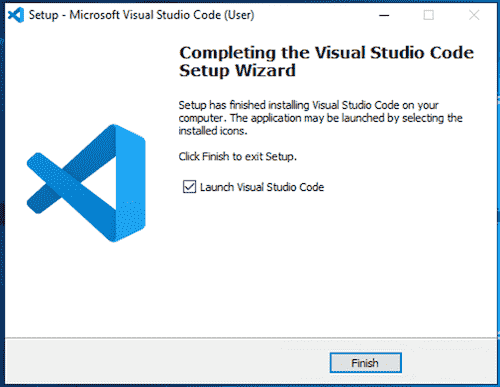
**Installing VS Code**

VS Code is a free code editor that runs on Windows, Mac and Linux.

Download VS Code from <https://code.visualstudio.com/>.



Install Visual Studio Code by opening the downloaded setup file and following the prompts.

**VS Code extensions**

VS Code extensions let you add languages, debuggers, and tools to your installation to support your development workflow. VS Code's rich extensibility model lets extension authors plug directly into the VS Code UI and contribute functionality through the same APIs used by VS Code.

Bring up the Extensions view by clicking on the Extensions icon in the **Activity Bar** on the side of VS Code or the **View: Extensions** command (Ctrl+Shift+X)



**JSON**

* JSON stands for JavaScript Object Notation
* JSON is a text format for storing and transporting data
* JSON is "self-describing" and easy to understand
* JSON is a lightweight data-interchange format
* JSON is plain text written in JavaScript object notation
* JSON is used to send data between computers
* JSON is language independent

**Why Use JSON?**

The JSON format is syntactically similar to the code for creating JavaScript objects.

JavaScript program can easily convert JSON data into JavaScript objects.

JSON data can easily be sent between computers, and used by any programming language.

JavaScript has a built in function for converting JSON strings into JavaScript objects:

JSON.parse()

JavaScript also has a built in function for converting an object into a JSON string:

JSON.stringify()

**JSON Syntax Rules**

JSON syntax is derived from JavaScript object notation syntax:

* Data is in name/value (key/value) pairs: A name/value pair consists of a field name (in double quotes), followed by a colon, followed by a value

"name":"John"

* Data is separated by commas
* Curly braces hold objects
* Square brackets hold arrays

**JSON Values**

In JSON, *values* must be one of the following data types:

* a string
* a number
* an object
* an array
* a boolean
* null

**Practice:** <html>

<body>

<h2>Access a JavaScript object</h2>

<p id="demo"></p>

<script>

const myObj = {name:"John", age:30, city:"New York"};

document.getElementById("demo").innerHTML = myObj.name;

or

document.getElementById("demo").innerHTML = myObj["name"];

</script>

</body>

</html>

# JSON.parse()

A common use of JSON is to exchange data to/from a web server.

When receiving data from a web server, the data is always a string.

Parse the data with JSON.parse(), and the data becomes a JavaScript object.

**Practice:** <html>

<body>

<h2>Creating an Object from a JSON String</h2>

<p id="demo"></p>

<script>

const txt = '{"name":"John", "age":30, "city":"New York"}'

const obj = JSON.parse(txt);

document.getElementById("demo").innerHTML = obj.name + ", " + obj.age;

</script>

</body>

</html>

**Array as JSON**

When using the JSON.parse() on a JSON derived from an array, the method will return a JavaScript array, instead of a JavaScript object.

**Practice:** <html>

<body>

<h2>Parsing a JSON Array.</h2>

<p>Data written as an JSON array will be parsed into a JavaScript array.</p>

<p id="demo"></p>

<script>

const text = '[ "Ford", "BMW", "Audi", "Fiat" ]';

const myArr = JSON.parse(text);

document.getElementById("demo").innerHTML = myArr[0];

</script>

</body>

</html>

# JSON.stringify()

A common use of JSON is to exchange data to/from a web server.

When sending data to a web server, the data has to be a string.

Convert a JavaScript object into a string with JSON.stringify().

**Practice:** <html><body>

<h2>Create a JSON string from a JavaScript object.</h2>

<p id="demo"></p>

<script>

const obj = {name: "John", age: 30, city: "New York"};

const myJSON = JSON.stringify(obj);

document.getElementById("demo").innerHTML = myJSON;

</script>

</body></html>

**JavaScript Objects**

Objects are variables too. But objects can contain many values.

The values are written as **name:value** pairs (name and value separated by a colon).

**Practice:** <script>

// Create an object:

const person = {

firstName: "John",

lastName: "Doe",

age: 50,

eyeColor: "blue"

};

// Display some data from the object:

document.getElementById("demo").innerHTML =person.firstName + " is " + person.age + " years old.";

or

document.getElementById("demo").innerHTML =person["firstName"] + " " + person["age"];

</script>

**Object Properties**

The **name:values** pairs in JavaScript objects are called **properties**

Accessing Object Properties

*objectName.propertyName*

*or*

*objectName["propertyName"]*

**Object Methods**

Objects can also have **methods**.

Methods are **actions** that can be performed on objects.

Methods are stored in properties as **function definitions**.

|  |  |
| --- | --- |
| **Property** | **Property Value** |
| firstName | John |
| lastName | Doe |
| age | 50 |
| fullName | function() {return this.firstName + " " + this.lastName;} |

const person = {  
  firstName: "John",  
  lastName : "Doe",  
  id     : 5566,  
  fullName : function() {  
    return this.firstName + " " + this.lastName;  
  }  
};

Note: this refers to the **person object**.

**Accessing Object Methods**

*objectName.methodName()*

**Practice:** <script>

// Create an object:

const person = {

firstName: "John",

lastName: "Doe",

id: 5566,

fullName: function() {

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

}

};

// Display data from the object:

document.getElementById("demo").innerHTML = person.fullName();

</script>

**Constructors**: A **constructor** in Java is a special method that is used to initialize objects

**Practice:** <script>

// Constructor function for Person objects

function Person(first, last, age, eye) {

this.firstName = first;

this.lastName = last;

this.age = age;

}

// Create a Person object

const myFather = new Person("John", "Doe", 50, "blue");

// Display age

document.getElementById("demo").innerHTML =

"My father is " + myFather.age + ".";

</script>

# JavaScript Object Accessors

Getters and setters allow you to get and set object properties via methods.

This example uses a lang property to get the value of the language property:

**Practice:** <script>

// Create an object:

const person = {

firstName: "John",

lastName: "Doe",

language: "en",

get lang() {

return this.language;

}

};

// Display data from the object using a getter:

document.getElementById("demo").innerHTML = person.lang;

</script>

**Practice:** <script>

// Create an object:

const person = {

firstName: "John",

lastName: "Doe",

language: "NO",

set lang(value) {

this.language = value;

}

};

// Set a property using set:

person.lang = "en";

// Display data from the object:

document.getElementById("demo").innerHTML = person.language;

</script>

**Prototype**

All JavaScript objects inherit properties and methods from a prototype.

The prototype property allows you to add new methods to objects constructors:

**Practice:** <script>

function Person(first, last, age, eye) {

this.firstName = first;

this.lastName = last;

}

Person.prototype.nationality = "English";

const myFather = new Person("John", "Doe", 50, "blue");

document.getElementById("demo").innerHTML =

"The nationality of my father is " + myFather.nationality;

</script>

**ES6**

ECMAScript (**European Computer Manufacturers Association Script**) is a scripting language based on JavaScript

ECMAScript 2015 was the second major revision to JavaScript.

ECMAScript 2015 is also known as ES6 and ECMAScript 6.

**Arrow functions**

Arrow functions allows a short syntax for writing function expressions.

You don't need the function keyword, the return keyword, and the **curly brackets**.

**Practice:**

<script>

const x = (x, y) => x \* y;

document.getElementById("demo").innerHTML = x(5, 5);

</script>

**Template strings**

**Template Literals** use back-ticks (``) rather than the quotes ("") to define a string:

**Practice:** <script>

let text = `Hello world!`;

document.getElementById("demo").innerHTML = text;

</script>

**Interpolation**

**Template literals** provide an easy way to interpolate variables and expressions into strings.

The method is called string interpolation.

The syntax is:

${...}

**Variable Substitutions**

**Template literals** allow variables in strings:

**Practice:** <script>

let firstName = "John";

let lastName = "Doe";

let text = `Welcome ${firstName}, ${lastName}!`;

document.getElementById("demo").innerHTML = text;

</script>

# JavaScript Object Prototypes

All JavaScript objects inherit properties and methods from a prototype.

**Prototype Inheritance**

All JavaScript objects inherit properties and methods from a prototype:

* Date objects inherit from Date.prototype
* Array objects inherit from Array.prototype
* Person objects inherit from Person.prototype

The Object.prototype is on the top of the prototype inheritance chain:

Date objects, Array objects, and Person objects inherit from Object.prototype.

**Adding Properties and Methods to Objects**

Sometimes you want to add new properties (or methods) to all existing objects of a given type.

Sometimes you want to add new properties (or methods) to an object constructor.

Using the **prototype** Property

The JavaScript prototype property allows you to add new properties to object constructors:

**Practice:** <script>

function Person(first, last) {

this.firstName = first;

this.lastName = last;

}

Person.prototype.nationality = "English";

const myFather = new Person("John", "Doe");

document.getElementById("demo").innerHTML =

"The nationality of my father is " + myFather.nationality;

</script>

The JavaScript prototype property also allows you to add new methods to objects constructors:

**Practice:** <script>

function Person(first, last, age, eye) {

this.firstName = first;

this.lastName = last;

this.age = age;

this.eyeColor = eye;

}

Person.prototype.name = function() {

return this.firstName + " " + this.lastName

};

const myFather = new Person("John", "Doe", 50, "blue");

document.getElementById("demo").innerHTML ="My father is " + myFather.name();

</script>

**The Spread (...) Operator**

The ... operator expands an iterable (like an array) into more elements:

**Practice:** <script>

const cars1 = ["Saab", "Volvo", ..."BMW"];

const cars2 = ["Fiat", "Toyota"];

const combined = [cars1, ...cars2];

document.getElementById("demo").innerHTML = combined;

</script>

**Map**

Creating a Map from an Array

A Map holds key-value pairs where the keys can be any datatype.

A Map remembers the original insertion order of the keys.

A Map has a property that represents the size of the map

**How to Create a Map**

You can create a JavaScript Map by:

* Passing an Array to new Map()
* Create a Map and use Map.set()

**new Map()**

You can create a Map by passing an Array to the new Map() constructor:

**Practice:** <script>

// Create a Map

const fruits = new Map([

["apples", 500],

["bananas", 300],

["oranges", 200]

]);

document.getElementById("demo").innerHTML = fruits.get("apples");

</script>

**Map.set()**

You can add elements to a Map with the set() method:

**Practice:**

<script>

// Create a Map

const fruits = new Map();

// Set Map Values

fruits.set("apples", 500);

fruits.set("bananas", 300);

fruits.set("oranges", 200);

document.getElementById("demo").innerHTML = fruits.get("apples");

</script>

**Map Methods**

|  |  |
| --- | --- |
| **Method** | **Description** |
| new Map() | Creates a new Map object |
| set() | Sets the value for a key in a Map |
| get() | Gets the value for a key in a Map |
| clear() | Removes all the elements from a Map |
| delete() | Removes a Map element specified by a key |
| has() | Returns true if a key exists in a Map |
| forEach() | Invokes a callback for each key/value pair in a Map |
| entries() | Returns an iterator object with the [key, value] pairs in a Map |
| keys() | Returns an iterator object with the keys in a Map |
| values() | Returns an iterator object of the values in a Map |

|  |  |
| --- | --- |
| **Property** | **Description** |
| size | Returns the number of Map elements |

**JavaScript Set**

A JavaScript Set is a collection of unique values.

Each value can only occur once in a Set.

A Set can hold any value of any data type.

**How to Create a Set**

You can create a JavaScript Set by:

* Passing an Array to new Set()
* Create a new Set and use add() to add values
* Create a new Set and use add() to add variables

**The new Set() Method**

Pass an Array to the new Set() constructor:

<script>

// Create a Set

const letters = new Set(["a","b","c"]);

// Display set.size

document.getElementById("demo").innerHTML = letters.size;

</script>

**Create a Set and add literal values:**

<script>

// Create a Set

const letters = new Set();

// Add Values to the Set

letters.add("a");

letters.add("b");

letters.add("c");

// Display set.size

document.getElementById("demo").innerHTML = letters.size;

</script>

**Create a Set and add variables:**

<script>

const letters = new Set(); // Create a Set

// Create Variables

const a = "a";

const b = "b";

const c = "c";

// Add the Variables to the Set

letters.add(a);

letters.add(b);

letters.add(c);

// Display set.size

document.getElementById("demo").innerHTML = letters.size;

</script>

**Practice:**<script>

// Create a new Set

const letters = new Set(["a","b","c"]);

// Add a new Element

letters.add("d");

letters.add("e");

// Display set.size

document.getElementById("demo").innerHTML = letters.size;

</script>

**Set Methods**

|  |  |
| --- | --- |
| **Method** | **Description** |
| new Set() | Creates a new Set |
| add() | Adds a new element to the Set |
| delete() | Removes an element from a Set |
| has() | Returns true if a value exists |
| clear() | Removes all elements from a Set |
| forEach() | Invokes a callback for each element |
| values() | Returns an Iterator with all the values in a Set |
| keys() | Same as values() |
| entries() | Returns an Iterator with the [value,value] pairs from a Set |

|  |  |
| --- | --- |
| **Property** | **Description** |
| size | Returns the number elements in a Set |

**What is meant by TypeScript?**

* TypeScript is a syntactic superset of JavaScript which adds static typing.
* This basically means that TypeScript adds syntax on top of JavaScript, allowing developers to add types.
* TypeScript being a "Syntactic Superset" means that it shares the same base syntax as JavaScript, but adds something to it.

**Why TypeScript?**

* TypeScript uses compile time type checking. Which means it checks if the specified types match **before** running the code, not **while** running the code
* JavaScript is better suited for small-scale applications, while TypeScript is better for larger applications.
* TypeScript is better than JavaScript in terms of language features, reference validation, project scalability, collaboration within and between teams, developer experience, and code maintainability.

### Setting up development environment for TypeScript

### Pre-requisite to install TypeScript

1. Text Editor or IDE
2. Node.js Package Manager (npm)
3. The TypeScript compiler

### Ways to install TypeScript

There are two ways to install TypeScript:

1. Install TypeScript using Node.js Package Manager (npm).
2. Install the TypeScript plug-in in your IDE (Integrated Development Environment).

### Install TypeScript using Node.js Package Manager (npm)

**Step-1** Install Node.js. It is used to setup TypeScript on our local computer.

To install Node.js on Windows, go to the following link: [**https://www.javatpoint.com/install-nodejs**](https://www.javatpoint.com/install-nodejs)

To verify the installation was successful, enter the following command in the Terminal Window.

node -v

npm -v

**Step-2** Install TypeScript. To install TypeScript, enter the following command in the Terminal Window.

* npm install typescript --save-dev         //As dev dependency
* npm install typescript -g                      //Install as a global module

or

* npm install -g typescript
* npm install typescript@latest -g          //Install latest if you have an older version

**Step-3** To verify the installation was successful, enter the command **$ tsc -v** in the Terminal Window.

**Install Live server**

npm install -g live-server

**Create and run first program in TypeScript**

* open command prompt
* go to d: drive(any drive)
* d:\>mkdir typescript
* d:\>cd typescript
* d:\typescript> npm install typescript --save-dev
* open visual studio code
* file-open folder-choose typescript folder from d:
* create new file- save it as types.ts(any name.ts)
* Write the below code and save it
* console.log("Hello World");
* go to command prompt and compile the program
* tsc types.ts
* run the program
* node types.js
* Observe the output

**Basic Types**

| Built-in Data Type | keyword | Description |
| --- | --- | --- |
| Number | number | It is used to represent both Integer as well as Floating-Point numbers |
| Boolean | boolean | Represents true and false |
| String | string | It is used to represent a sequence of characters |
| Void | void | Generally used on function return-types |
| Null | null | It is used when an object does not have any value |
| Undefined | undefined | Denotes value given to uninitialized variable |
| Any | any | If variable is declared with any data-type then any type of value can be assigned to that variable |

**Example**

**let a: null = null;**

**let b: number = 123;**

**let c: number = 123.456;**

**let d: string = ‘Geeks’;**

**let e: undefined = undefined;**

**let f: boolean = true;**

console.log(a);

console.log(b);

console.log(c);

console.log(d);

console.log(e);

console.log(f);

**let a: any = null;**

**let b: any =123;**

**let c: any = 123.456;**

**let d: any = ‘Geeks’;**

**let e: any = undefined;**

**let f: any = true;**

**Control flow statement**

TypeScript control statements:

1. If Statement  
2. If else statement  
3. if else if statement

**TypeScript If Statement:**

If statement is used to execute a block of statements if specified condition is true.

if(condition){

*//Block of TypeScript statements.*

}

**TypeScript If Else Statement:**

If else statement is used to execute either of two block of statements depends upon the condition. If condition is true then if block will execute otherwise else block will execute.

if(condition){

*//Block of TypeScript statements1.*

}else{

*//Block of TypeScript statements2.*

}

**TypeScript If Else If Statement:**

If else statement is used to execute one block of statements from many depends upon the condition. If condition1 is true then block of statements1 will be executed, else if condition2 is true block of statements2 is executed and so on. If no condition is true, then else block of statements will be executed.

if(condition1){

*//Block of TypeScript statements1.*

}else if(condition2){

*//Block of TypeScript statements2.*

} . . . else if(conditionn){

*//Block of TypeScript statementsn.*

}else{

*//Block of TypeScript statements.*

}

**Example**

var num:number = 2;

if(num==1){

console.log("TypeScript Statement 1");

}

else if(num==2){

console.log("TypeScript Statement 2");

}

else if(num==3){

console.log("TypeScript Statement 3");

}

else{

console.log("TypeScript Statement n");

}

# TypeScript - for Loops

1. for loop
2. for..of loop
3. for..in loop

## for Loop

The for loop is used to execute a block of code a given number of times, which is specified by a condition.

for (first expression; second expression; third expression ) {

// statements to be executed repeatedly

}

for (let i = 0; i < 3; i++) {

console.log ("Block statement execution no." + i);

}

**for...of Loop**

TypeScript includes the for...of loop to iterate and access elements of an array, list, or tuple collection.

let arr = [10, 20, 30, 40];

for (var val of arr) {

console.log(val); // prints values: 10, 20, 30, 40

}

**for...in Loop**

Another form of the for loop is for...in. This can be used with an array, list, or tuple.

let arr = [10, 20, 30, 40];

for (var index in arr) {

console.log(index); // prints indexes: 0, 1, 2, 3

console.log(arr[index]); // prints elements: 10, 20, 30, 40

}

# TypeScript - while Loop

The while loop is another type of loop that checks for a specified condition before beginning to execute the block of statements. The loop runs until the condition value is met.

while (condition expression) {

// code block to be executed

}

let i: number = 2;

while (i < 4) {

console.log( "Block statement execution no." + i )

i++;

}

**do..while loop**

The do..while loop is similar to the while loop, except that the condition is given at the end of the loop. The do..while loop runs the block of code at least once before checking for the specified condition.

do {

// code block to be executed

}

while (condition expression);

let i: number = 2;

do {

console.log("Block statement execution no." + i )

i++;

} while ( i < 4)

**Functions**

In TypeScript, functions can be of two types: named and anonymous.

**Named Functions**

A named function is one where you declare and call a function by its given name.

function display() {

console.log("Hello TypeScript!");

}

display(); //Output: Hello TypeScript

**Functions can also include parameter types and return type.**

function Sum(x: number, y: number) : number {

return x + y;

}

Sum(2,3); // returns 5

**Anonymous Function**

* An anonymous function is one which is defined as an expression.
* This expression is stored in a variable.
* So, the function itself does not have a name.
* These functions are invoked using the variable name that the function is stored in.

let display = function() {

console.log("Hello TypeScript!");

};

display (); //Output: Hello TypeScript!

**An anonymous function can also include parameter types and return type.**

let Sum = function(x: number, y: number) : number

{

return x + y;

}

Sum(2,3); // returns 5

**Function Parameters**

Parameters are values or arguments passed to a function. In TypeScript, the compiler expects a function to receive the exact number and type of arguments as defined in the function signature.

function Greet(greeting: string, name: string ) : string {

return greeting + ' ' + name + '!';

}

Greet('Hello','Steve');

### Default Parameters

TypeScript provides the option to add default values to parameters.

function Greet(name: string, greeting: string = "Hello") : string {

return greeting + ' ' + name + '!';

}

Greet('Steve');//OK, returns "Hello Steve!"

**Modern UI technologies**

1. React

### 2. Angular

### 3. Flutter

### 4. Vue.js

### 5. JQuery

### 6. Emberjs

### 7. Semantic UI

### 8 **Backbonejs**

### **9.   Foundation**

### **10.  Svelte**