The `querySelector()` is a method of the `Element` interface. The `querySelector()` method allows you to select the first element that matches one or more CSS selectors.

If the `selector` is not valid CSS syntax, the method will raise a `SyntaxError` exception.

If no element matches the CSS selectors, the `querySelector()` returns `null`.

The `querySelectorAll()` method returns a static `NodeList` of elements that match the CSS selector. If no element matches, it returns an empty `NodeList`.

To convert the `NodeList` to an array, you use the `Array.from()` method like this:

```
let nodeList = document.querySelectorAll(selector);
let elements = Array.from(nodeList);
```

1) Universal Selector

The universal selector is denoted by `*` that matches all elements of any type:

```
*
```

The following example uses the `querySelector()` selects the first element in the document:

```
let element = document.querySelector('*');
```
Code language: JavaScript (javascript)

And this selects all elements in the document:

```
let elements = document.querySelectorAll('*');
```

2) Type selector

To select elements by node name, you use the type selector e.g., `a` selects all `<a>` elements:

```
elementName
```

The following example finds the first `h1` element in the document:

```
let firstHeading = document.querySelector('h1');
```
Code language: JavaScript (javascript)

The following example finds all `h2` elements:

```
let heading2 = document.querySelectorAll('h2');
```

3) Class selector

To find the element with a given CSS class, you use the class selector syntax:

```
.className
```
Code language: CSS (css)

The following example finds the first element with the `menu-item` class:

```
let note = document.querySelector('.menu-item');
```
Code language: JavaScript (javascript)

The following example finds all elements with the `menu` class:

```
let notes = document.querySelectorAll('.menu-item');
```

4) ID Selector

To select an element based on the value of its id, you use the id selector syntax:

```
#id
```
Code language: CSS (css)

The following example finds the first element with the id `#logo`:

```
let logo = document.querySelector('#logo');
```
Code language: JavaScript (javascript)

Since the `id` should be unique in the document, the `querySelectorAll()` is not relevant.

5) Attribute selector

To select all elements that have a given attribute, you use one of the following attribute selector syntaxes:

[attribute] : — Has the attribute (any value)

```
<input type="text" name="username">
<input name="email">
document.querySelectorAll('input[name]');
```

[attribute=value] : — Exact match

```
<input type="text" name="username">
<input name="email">
<input type="password">
document.querySelector('input[name="email"]')
document.querySelector('input[type="text"]')
```
Note : Matches <input name="email"> exactly.

[attribute~=value] — Contains word (space-separated)

```
<input class="btn primary">
<input class="btn secondary">
document.querySelector('[class~="primary"]');
```
Note: Matches elements whose attribute contains the word primary in a space-separated list.

[attribute|=value] — Starts with value or value followed by -

document.querySelector('[lang|="en"]');
Note : Matches lang="en" and lang="en-US"

[attribute^=value] — Starts with value

<input name="userName">
<input name="userEmail">
document.querySelector('[name^="user"]');
Note : Matches any input whose name starts with "user".

[attribute$=value] — Ends with value

<input name="firstName">
<input name="lastName">
document.querySelector('[name$="Name"]')
Note : Matches any input whose name ends with `"Name"`.

[attribute*=value] — Contains value

<input name="userName">
<input name="userEmail">
document.querySelector('[name*="Email"]');
Note : Matches any input whose name contains `"Email"` anywhere.

6. Grouping selectors

To group multiple selectors, you use the following syntax:

```
selector, selector, ...
```

The selector list will match any element with one of the selectors in the group.

The following example finds all `<div>` and `<p>` elements:

```
let elements = document.querySelectorAll('div, p');
```

Combinators

What are Combinators?

In CSS, combinators define the relationship between two selectors.
They describe how elements are related in the DOM hierarchy.

There are 4 main combinators:
1.Descendant (space)
2.Child (>)
3.Adjacent sibling (+)
4.General sibling (~)

1.Descendant combinator (space) :

```
<div>
      <p>Direct child p </p>
      <section>
         <p>Nested p </p>
      </section>
      </div>
<div> <p> my name is omar </p></div>
   <p>Outside div </p>
```

document.querySelectorAll('div p');

Note : Select all `<p>` inside any `<div>` (no matter how deep)

2. Child combinator (>)

Selects elements that are direct children only

```
<div>
 <p>Direct child p </p>
 <section>
  <p>Nested p </p>
 </section>
</div>
 document.querySelector('div > p');
  document.querySelectorAll('div > p');
```

Note : Select <p> that is a direct child of

3. Adjacent sibling combinator (+)

```
Selects the first element immediately after another element.
  <h1>Title</h1>
  <p>Paragraph right after h1 </p>
  <p>Another paragraph </p>
  <h1> another h1 tag </h1>
  <p> This my name </p>

document.querySelector('h1 + p');
```

Note : Select all <p> elements that are siblings of <h1> and come after it.

4. General sibling combinator (~)

```
<h1>Title</h1>
    <p>First sibling paragraph ✅</p>
    <p>Second sibling paragraph ✅</p>
    <div>Not a p ❌</div>
    <p> after div p tag </p>
document.querySelector('h1~p');
```

document.querySelectorAll('h1~p');
Note : Select all <p> elements that are siblings of <h1> and come after it

1. Pseudo-classes in JavaScript

Pseudo-classes target states of elements (hover, first-child, nth-child, etc.).

```
document.querySelectorAll("selector:pseudo-class")

// :first-child
document.querySelectorAll("li:first-child"); // <li>First</li>

// :last-child
document.querySelectorAll("li:last-child"); // <li>Fourth</li>

// :nth-child(2)
document.querySelectorAll("li:nth-child(2)"); // <li>Second</li>

// :nth-of-type(3)
document.querySelectorAll("li:nth-of-type(3)"); // Third <li>

// :not(.active)
document.querySelectorAll("li:not(.active)"); // all <li> except class="active"

// :required
document.querySelectorAll("input:required"); // first input

// :checked
document.querySelectorAll("input:checked"); // checked checkbox

// :enabled
```

```
document.querySelectorAll("input:enabled");

// :disabled
document.querySelectorAll("input:disabled");

// :empty
document.querySelectorAll("li:empty");  //  selects  empty  li  if
present

// :hover ❌
// JS cannot directly "query" hover state (dynamic). Instead use
events.
```

Full List of Useful Pseudo-classes

Here’s a categorized set:

Structural

:first-child

:last-child

:nth-child(n)

:nth-last-child(n)

:only-child

:first-of-type

:last-of-type

:nth-of-type(n)

`:nth-last-of-type(n)`

`:only-of-type`

`:empty`

`:root`