前端網絡開發人員課程
（二）進階網絡程式設計

# 2. JS DOM II: Elements II

Presented by Krystal Institute

## Learning Objective

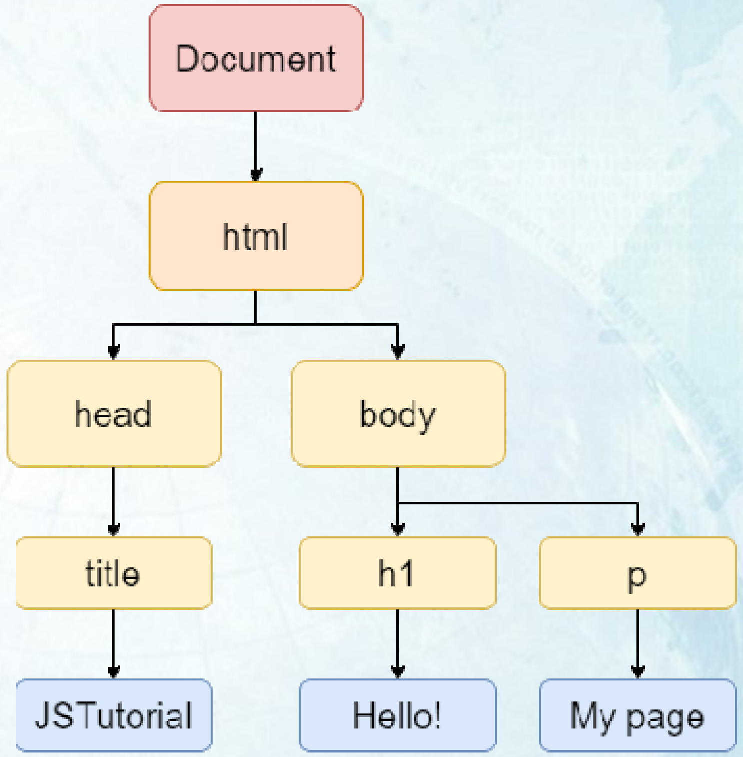
- Understand the relationship between elements and how to traverse between elements
- Learn how to manipulate HTML elements

## Content

## 2.1 Revise on the previous lesson

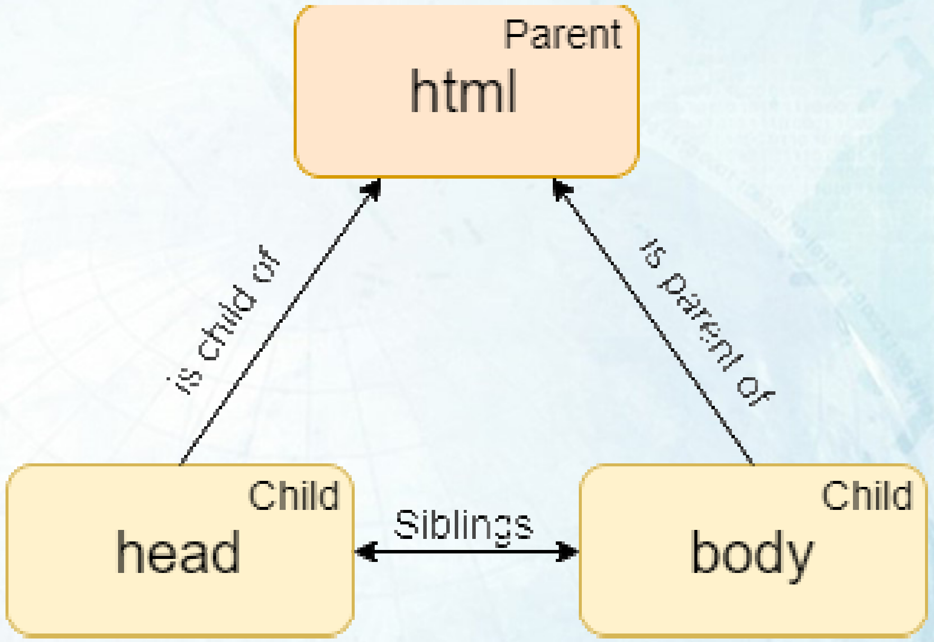
# Document Object Model (DOM)

- A tree of nodes defining the structure of the document
- Allows for addition, removal and modification of nodes



## Node Relationships

- Relationships between each node are same as a traditional family tree
- Child nodes, parent nodes, and siblings exist between nodes



## DOM Selecting Elements

- GetElementById – returns matching element by id
- GetElementByName – returns HTMLCollection Object (array-like) of matching elements by name
- GetElementsByTagName – returns HTMLCollection Object (array-like) of matching elements by tags
- GetElementsByClassName – returns HTMLCollection Object (array-like) of matching elements by classes

## DOM Selecting Elements

- querySelector returns first matching element, querySelectorAll retruns Nodelist of matching elements
- A universal selector ( * ) matches all elements in the document
- A type selector matches elements by tags
- A class selector ( . ) matches elements by classes
- A id selector ( # ) matches elements by id
- A attribute selector ( [ ] ) matches elements by attributes

# DOM Selecting Elements

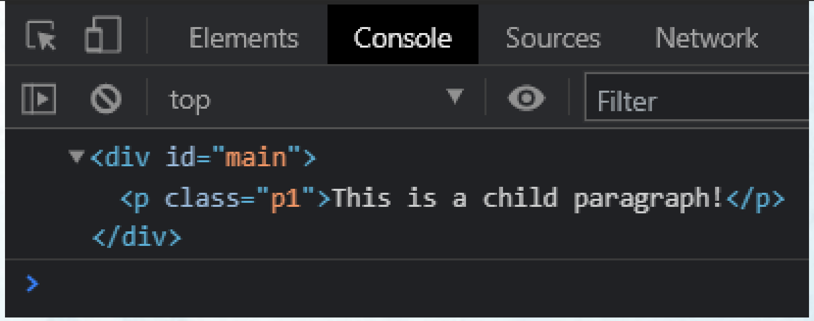
- Using comma ( , ) returns elements matching any one selector
- Using space between selectors ( p a ) matches element inside another
- Using > between selectors ( p > a ) matches elements that are directly inside another
- Using ~ between selectors ( p ~ a ) matches elements that follows one another
- Using + between selectors ( p + a ) matches element that directly follows one another

## 2.2 Traversing Elements

## Parent Node

- parentNode is a property used on a specified node
- Used to get the parent node of the specified node
- It is read-only

```
<body>
    <div id="main">
        <p class="p1">This is a child paragraph!</p>
    </div>

    <script>
        let p1 = document.querySelector('.p1');
        console.log(p1.parentNode);
    </script>
</body>
```



```
▾<div id="main">
   <p class="p1">This is a child paragraph!</p>
 </div>
>
```

## Child Node

- firstChild property returns the first child of a specified element

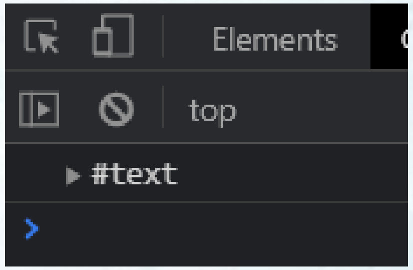
- If the element does not have any child, null will be returned instead

```
<body>
    <div id="main">
        <p id="p1">This is the first child paragraph!</p>
        <p id="p2">This is the second child paragraph!</p>
        <p id="p3">This is the last child paragraph!</p>
    </div>

    <script>
        let main = document.querySelector(‘#main');
        console.log(main.firstChild);
    </script>
</body>
```

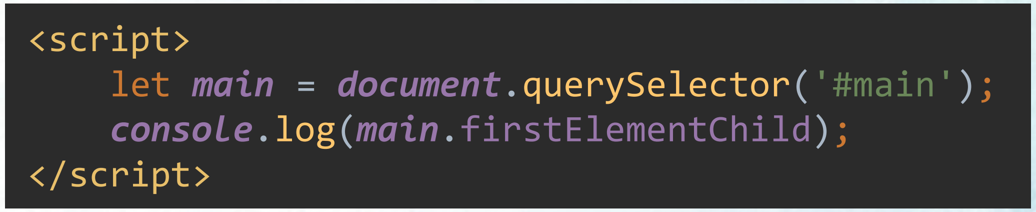
## Child Node

- Whitespace counts as text nodes, it will be returned if firstChild is used

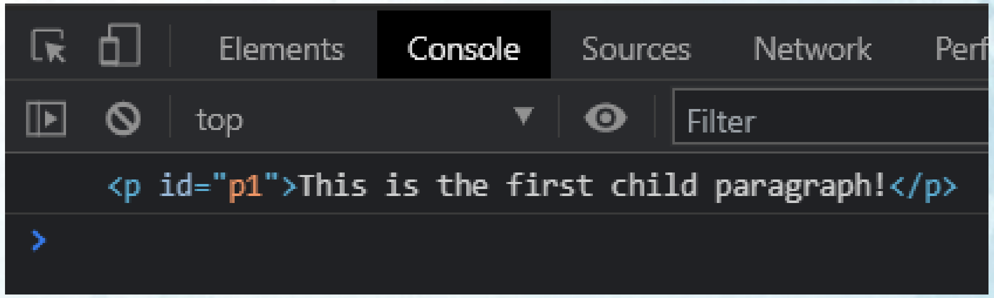
- To return the first element node, use firstElementChild instead



```
▸ #text
```



```
<script>
    let main = document.querySelector('#main');
    console.log(main.firstElementChild);
</script>
```



```
<p id="p1">This is the first child paragraph!</p>
```

## Child Node

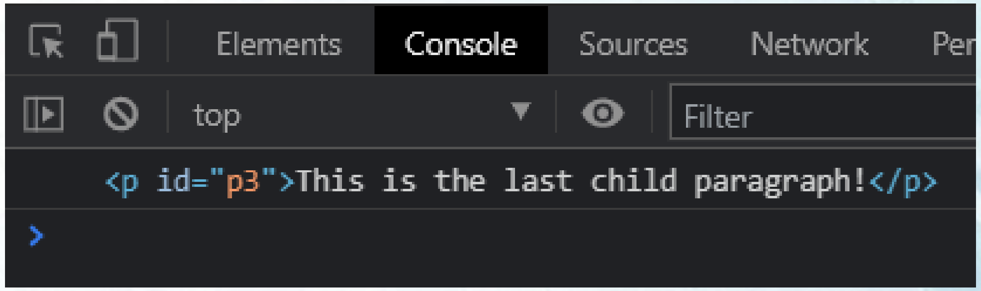
- lastChild works similar to firstChild, returning the last child of the element
- The last child being the white space between </p> and </div>, text node will be returned

```
<body>
    <div id="main">
        <p id="p1">This is the first child paragraph!</p>
        <p id="p2">This is the second child paragraph!</p>
        <p id="p3">This is the last child paragraph!</p>
    </div>

    <script>
        let main = document.querySelector('#main');
        console.log(main.lastChild);
    </script>
</body>
```

## Child Node

- Similarly, using lastElementChild will return the last child element node of the specified element

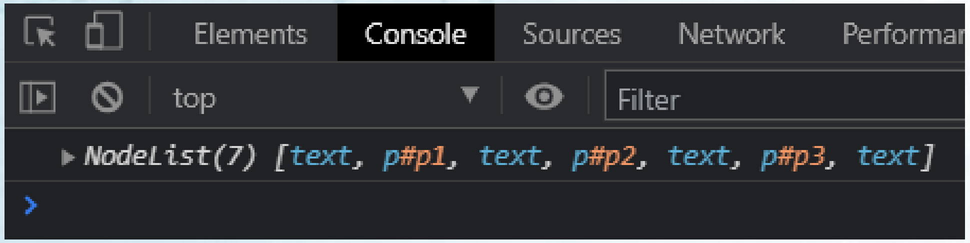
```
<script>
    let main = document.querySelector('#main');
    console.log(main.lastElementChild);
</script>
```



```
<p id="p3">This is the last child paragraph!</p>
>
```

## Child Node

- childNodes property returns a live Nodelist of all child nodes from the specified element
- Again, this includes all types of nodes



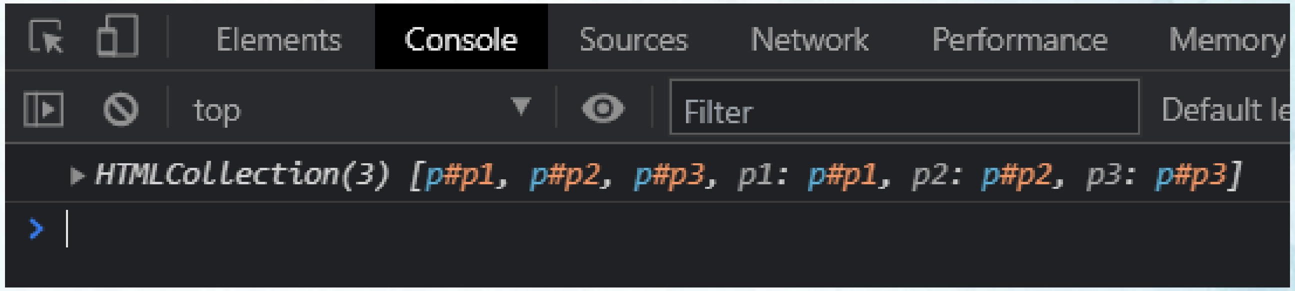
```
▸ NodeList(7) [text, p#p1, text, p#p2, text, p#p3, text]
>
```

```
<body>
    <div id="main">
        <p id="p1">This is the first child paragraph!</p>
        <p id="p2">This is the second child paragraph!</p>
        <p id="p3">This is the last child paragraph!</p>
    </div>

    <script>
        let main = document.querySelector('#main');
        console.log(main.childNodes);
    </script>
</body>
```

## Child Node

- To only get element type nodes, use children instead

```
<script>
    let main = document.querySelector('#main');
    console.log(main.children);
</script>
```



```
▸ HTMLCollection(3) [p#p1, p#p2, p#p3, p1: p#p1, p2: p#p2, p3: p#p3]
> |
```

## Child Node Activity

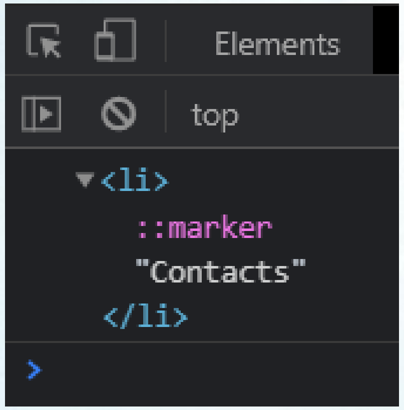
- Activity: display a list of child nodes, as well as the first and last child
- Create the list of elements using <ul>
- Get all children of the <ul>
- Get the first and last child of the <ul>
- Console log all of them

```
<body>
    <div id="main">
        <ul id="list">
            <li>Home</li>
            <li>Products</li>
            <li>About</li>
            <li>Contacts</li>
            <li>Login</li>
        </ul>
    </div>

    <script>
        let children = document.querySelector('#list').children;
        let first =
document.querySelector('#list').firstElementChild;
        let last = document.querySelector('#list').lastElementChild;
        console.log(children);
        console.log(first);
        console.log(last);
    </script>
</body>
```

## Siblings

- nextElementSibling returns the next sibling in a list of element

```
<body>
    <div id="main">
        <ul>
            <li>Home</li>
            <li>Products</li>
            <li class="current">About</li>
            <li>Contacts</li>
            <li>Login</li>
        </ul>
    </div>

    <script>
        let current = document.querySelector('.current');
        console.log(current.nextElementSibling);
    </script>
</body>
```

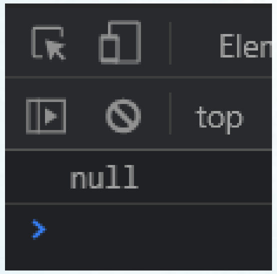


```
▼<li>
    ::marker
    "Contacts"
  </li>
```

## Siblings

- nextElementSibling returns null if it is the last one in the list of elements

```
<body>
    <div id="main">
        <ul>
            <li>Home</li>
            <li>Products</li>
            <li>About</li>
            <li>Contacts</li>
            <li class="current">Login</li>
        </ul>
    </div>

    <script>
        let current = document.querySelector('.current');
        console.log(current.nextElementSibling);
    </script>
</body>
```



```
null
>
```

## Siblings

- previousElementSibling works similar, returning the previous sibling in the list of elements
- Returns null if specified element is the first one in the list

```
<body>
    <div id="main">
        <ul>
            <li>Home</li>
            <li>Products</li>
            <li class="current">About</li>
            <li>Contacts</li>
            <li>Login</li>
        </ul>
    </div>

    <script>
        let current = document.querySelector('.current');
        console.log(current.previousElementSibling);
    </script>
</body>
```

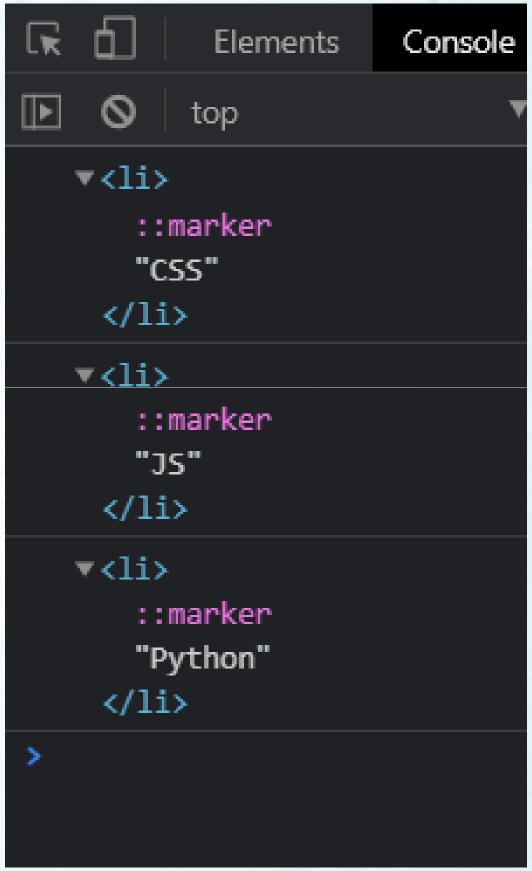
## Siblings Exercise

- Try and create a website with a list of elements <ul> and a button
- The button will display each item of the <ul> on every click
- E.g. on first button click, console log CSS, on second button click, console log JS, etc.
- Does not have to loop back

```
<div id="main">
    <ul id="list">
        <li>CSS</li>
        <li>JS</li>
        <li>Python</li>
        <li>HTML</li>
        <li>ES6</li>
    </ul>
</div>
```

## Siblings Exercise Example

- The first three button clicks look like this



```
▼<li>
   ::marker
   "CSS"
 </li>
▼<li>
   ::marker
   "JS"
 </li>
▼<li>
   ::marker
   "Python"
 </li>
>
```

## Siblings Solution

- Setting up the <li> and <button>

```
<div id="main">
    <ul id="list">
        <li>CSS</li>
        <li>JS</li>
        <li>Python</li>
        <li>HTML</li>
        <li>ES6</li>
    </ul>
    <button type="button" onclick="iteratelist()">Click me to iterate!</button>
</div>
```

# Siblings Solution

- Get the element <ul>
- Get the first child of the <ul> as the current child

- Inside the button function,
- Console log the current child
- Change the current child into the next element sibling

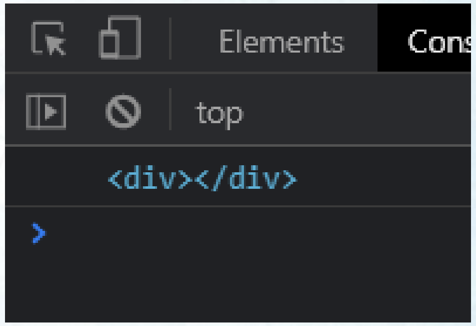
```
<script>
    let mainlist = document.querySelector("#list");
    let current = mainlist.firstElementChild

    function iteratelist() {
        console.log(current)
        current = current.nextElementSibling
    }
</script>
```

## 2.2 Traversing Elements

## Create Element

- To create a new element in the HTML document, use createElement(Tag)
- It will return a new node with element type

```
<script>
    let div = document.createElement('div');
    console.log(div);
</script>
```



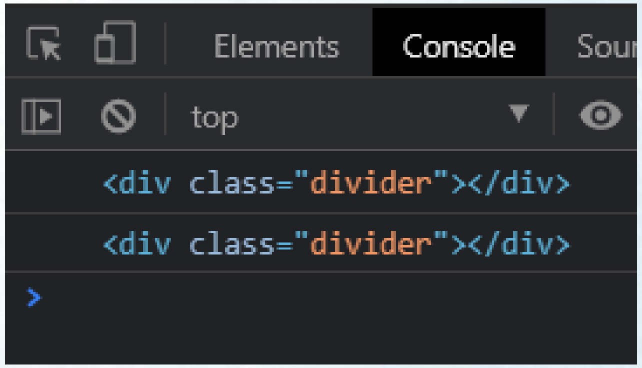
```
<div></div>
>
```

## Create Element

- It is not attached to the DOM Tree
- Its properties can be manipulated
- It is live, changing

  attributes/properties will be reflected

```
<script>
    let div = document.createElement('div');
    console.log(div);
    div.className = "divider";
    console.log(div);
</script>
```

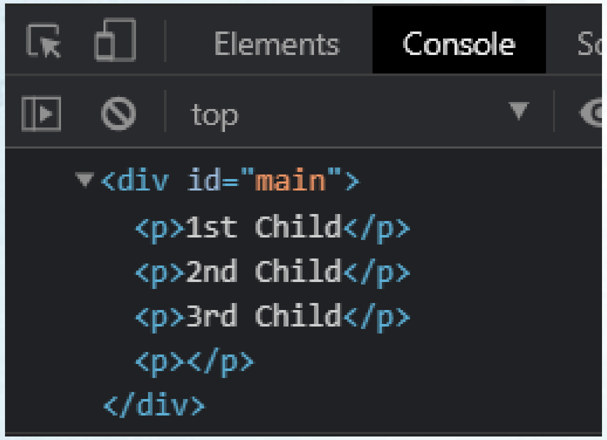


```
<div class="divider"></div>
<div class="divider"></div>
>
```

## Append Child

- appendChild moves an node onto the end of the list of nodes from the specified parent node

```
<body>
    <div id="main">
        <p>1st Child</p>
        <p>2nd Child</p>
        <p>3rd Child</p>
    </div>

    <script>
        let p = document.createElement('p');
        let main = document.querySelector('#main');
        main.appendChild(p);
        console.log(main);
    </script>
</body>
```



```
▼<div id="main">
    <p>1st Child</p>
    <p>2nd Child</p>
    <p>3rd Child</p>
    <p></p>
  </div>
```

## Append Child

- The target node of appendChild will be moved and not copied

- appendChild can be used in most nodes

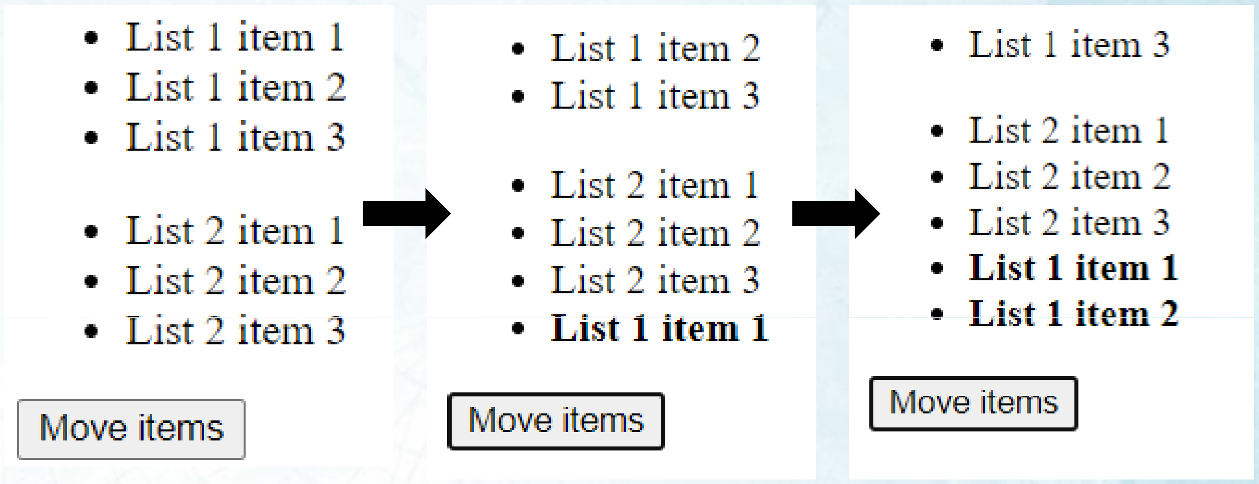
# Append Child Exercise

- Try and create 2 lists <ul> with items inside with a button
- On button click, append the first item in the first <ul> into a created <b> element, and move it to the bottom inside the second <ul>

```html
<ul id="list1">
    <li>List 1 item 1</li>
    <li>List 1 item 2</li>
    <li>List 1 item 3</li>
</ul>
<ul id="list2">
    <li>List 2 item 1</li>
    <li>List 2 item 2</li>
    <li>List 2 item 3</li>
</ul>
<button type="button" onclick="move()">Move items</button>
```

## Append Child Example

- 2 lists of 3 items are shown initially
- On each click, the top item on list 1 is moved to the bottom of list 2
- If <b> is inserted in the moved items, bold letters will occur



- List 1 item 1
- List 1 item 2
- List 1 item 3

- List 2 item 1
- List 2 item 2
- List 2 item 3

Move items

- List 1 item 2
- List 1 item 3

- List 2 item 1
- List 2 item 2
- List 2 item 3
- **List 1 item 1**

Move items

- List 1 item 3

- List 2 item 1
- List 2 item 2
- List 2 item 3
- **List 1 item 1**
- **List 1 item 2**

Move items

## Append Child Solution

- Setting up the lists and button, along with id attribute

```
<ul id="list1">
    <li>List 1 item 1</li>
    <li>List 1 item 2</li>
    <li>List 1 item 3</li>
</ul>
<ul id="list2">
    <li>List 2 item 1</li>
    <li>List 2 item 2</li>
    <li>List 2 item 3</li>
</ul>
<button type="button" onclick="move()">Move items</button>
```

## Append Child Solution

- Inside the function,
- Get the <ul> elements
- Create a <b> element
- Get the first item in list 1
- Add the list item into the <b> element
- Append the <b> element into the second list

```
<script>
    function move() {
        let list1 = document.querySelector("#list1")
        let list2 = document.querySelector("#list2")
        let b = document.createElement("b")
        let item = list1.firstElementChild
        b.appendChild(item)
        list2.appendChild(b)
    }
</script>
```

## Element Text

- textContent can be used to get the text of the element, and the text of ALL its child nodes
- Comments and styles are ignored

```
<body>
    <div id="main">
        <span style="display:none">A hidden text</span>
        A not so hidden text
        <!-- A comment -->
    </div>

    <script>
        let main = document.querySelector("#main")
        console.log(main.textContent)
    </script>
</body>
```

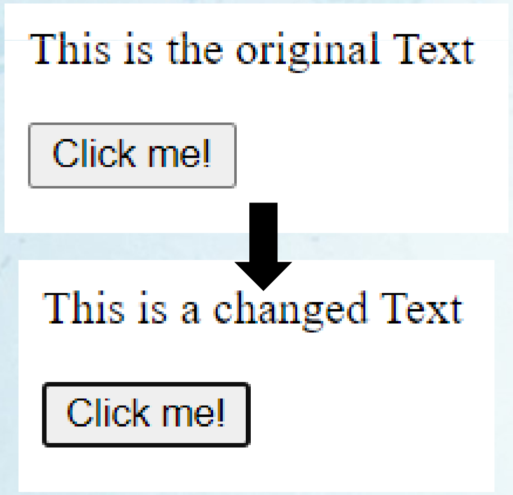
## Element Text

- innerText works similar to textContent, but only returns human-readable text
- Styles with visibility:hidden or display:none will not be returned

```
<body>
    <div id="main">
        <span style="display:none">A hidden text with</span>
        <span style="visibility: hidden">Another hidden text</span>
        A not so hidden text
        <!-- A comment -->
    </div>

    <script>
        let main = document.querySelector("#main")
        console.log(main.innerText)
    </script>
</body>
```

## Element Text

- textContent/innerText can be used to set the text of an element as well!


This is the original Text

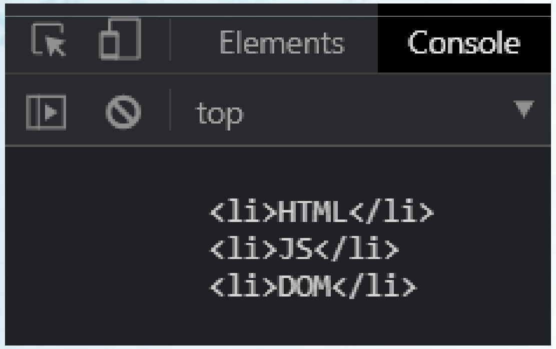
This is a changed Text

```
<body>
    <p>This is the original Text</p>
    <button type="button" onclick="changetext()">Click me!</button>

    <script>
        function changetext() {
            let p = document.querySelector("p");
            p.textContent = "This is a changed Text";
        };
    </script>
</body>
```

## innerHTML

- innerHTML is used to get the HTML markup of a specified element

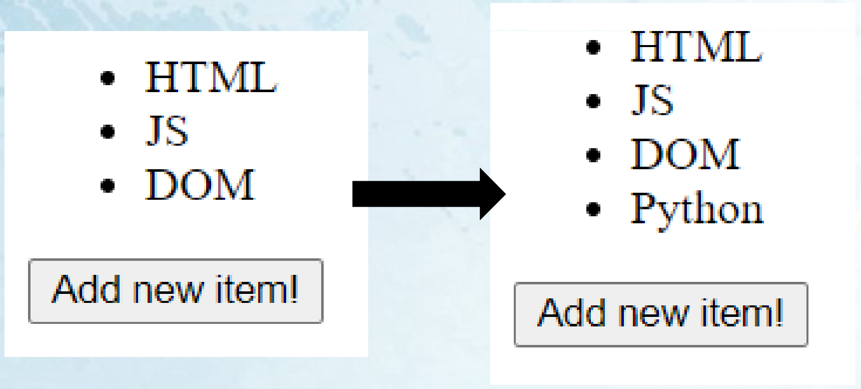
```
<body>
    <ul id="main">
        <li>HTML</li>
        <li>JS</li>
        <li>DOM</li>
    </ul>

    <script>
        let main = document.querySelector("#main");
        console.log(main.innerHTML)
    </script>
</body>
```



```
<li>HTML</li>
<li>JS</li>
<li>DOM</li>
```

## innerHTML

- innerHTML is also used to set the HTML markup of a specified element



- HTML
- JS
- DOM

Add new item!

- HTML
- JS
- DOM
- Python

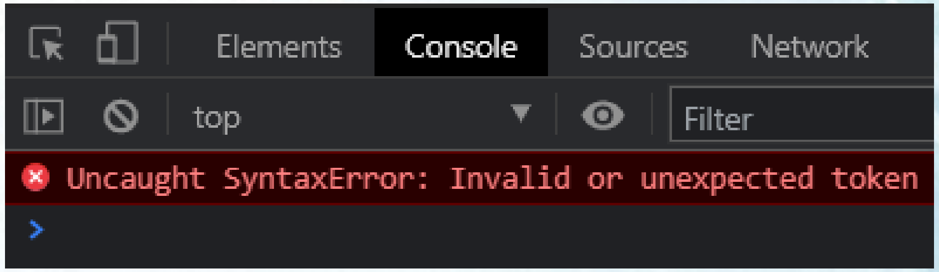
Add new item!

```
<body>
    <ul id="main">
        <li>HTML</li>
        <li>JS</li>
        <li>DOM</li>
    </ul>
    <button type="button" onclick="addhtml()">Add item</button>

    <script>
        function addhtml() {
            let main = document.querySelector("#main");
            main.innerHTML += "<li>Python</li>";
        }
    </script>
</body>
```

## innerHTML

- Do not set innerHTML with user inputs
- HTML5 has a safeguard that disables execution of <script> using innerHTML
- There are other means on executing JS functions

```
<script>
    function addhtml() {
        let main = document.querySelector("#main");
        main.innerHTML +=
        "<script><!– dangerous code --></script>";
    }
</script>
```



```
Uncaught SyntaxError: Invalid or unexpected token
```

Safeguard with HTML5

## innerHTML

- Using error handlers, js functions can be executed, bypassing the safeguard
- Image doesn’t exist, so it will always cause an error, which triggers the error handler
- Some functions will damage your computer!

```
<script>
    function addhtml() {
        let main = document.querySelector("#main");
        main.innerHTML +=
            "<img src='1' onerror='<!-- dangerous code -->'>";
    }
</script>
```

# innerHTML vs. createElement

Efficiency

- Using createElement only creates the Element independently
- Using innerHTML will cause the web browser to recreate all nodes inside the specificed parent element

- CreateElement is more efficient

## innerHTML vs. createElement

Security

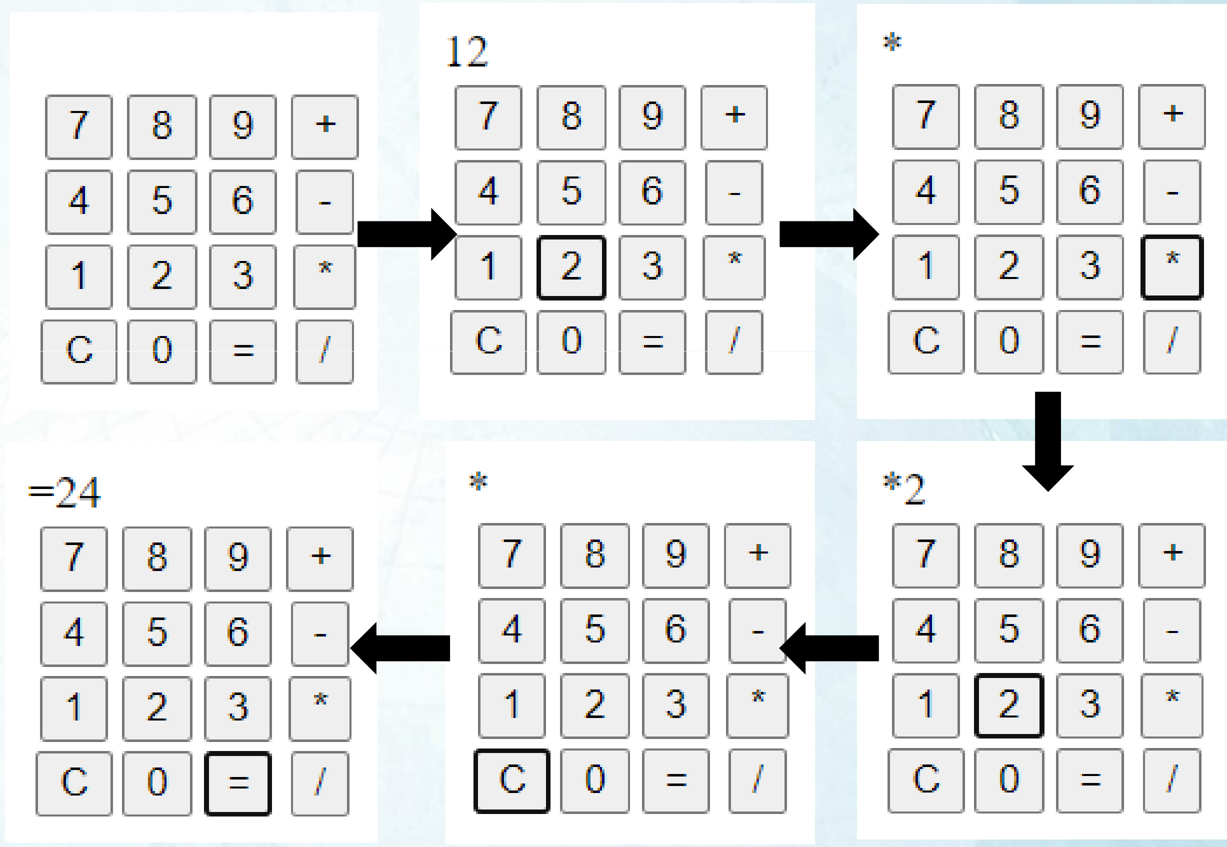
- Using createElement only creates the Element Node
- Using innerHTML with user inputs will have potential danger, and should only be used for a trusted source like a database
- CreateElement is more secure

## Exercise

- Create a website that…
- Resembles a simple calculator, with buttons 1 to 9, 4 basic operator buttons + - * /, a clear button and a enter button
- Has a display that shows the current calculation on top of the buttons
- Displays the calculated number on the display
- Only one calculation is required, no need for multiple steps (e.g. 1 * 2 + 3)
- Please finish it by the end of this lesson

## Exercise Example

- Interface resembles a calculator
- Clicking on the numbers will be shown on the display
- Clicking on the operators will clear the display for the second number input
- Clicking Clear (C) will clear the current number display
- Clicking enter (=) will display the result



## References

- Use these if you need more explanations!
- https://www.javascripttutorial.net/es6/
- https://javascript.info/

- Use this if you need more specific answers!
- https://stackoverflow.com/