**append VS appendChild**

append and appendChild are two popular methods used to add elements into the Document Object Model(DOM). They are often used interchangeably without much troubles, but if they are the same, then why not scrape one....Well they are only similar, but different. Here's how:

## .append()

This method is used to add an element in form of a Node object or a DOMString (basically means text). Here's how that would work.

// Inserting a Node object

const parent = document.createElement('div');

const child = document.createElement('p');

parent.append(child);

// This appends the child element to the div element

// The div would then look like this <div><p></p></div>

// Inserting a DOMString

const parent = document.createElement('div');

parent.append('Appending Text');

// The div would then look like this <div>Appending Text</div>

## .appendChild()

Similar to the **.append** method, this method is used to elements in the DOM, but in this case, only accepts a Node object.

// Inserting a Node object

const parent = document.createElement('div');

const child = document.createElement('p');

parent.appendChild(child);

// This appends the child element to the div element

// The div would then look like this <div><p></p></div>

// Inserting a DOMString

const parent = document.createElement('div');

parent.appendChild('Appending Text');

// Uncaught TypeError: Failed to execute 'appendChild' on 'Node': parameter 1 is not of type 'Node'

### Differences

1. .append accepts Node objects and DOMStrings while .appendChild accepts only Node objects
2. const parent = document.createElement('div');
3. const child = document.createElement('p');
4. // Appending Node Objects
5. parent.append(child) // Works fine
6. parent.appendChild(child) // Works fine
7. // Appending DOMStrings
8. parent.append('Hello world') // Works fine
9. parent.appendChild('Hello world') // Throws error
10. .append does not have a return value while .appendChild returns the appended Node object
11. const parent = document.createElement('div');
12. const child = document.createElement('p');
13. const appendValue = parent.append(child);
14. console.log(appendValue) // undefined
15. const appendChildValue = parent.appendChild(child);
16. console.log(appendChildValue) // <p><p>
17. .append allows you to add multiple items while appendChild allows only a single item
18. const parent = document.createElement('div');
19. const child = document.createElement('p');
20. const childTwo = document.createElement('p');
21. parent.append(child, childTwo, 'Hello world'); // Works fine
22. parent.appendChild(child, childTwo, 'Hello world');

// Works fine, but adds the first element and ignores the rest

# HTMLCollection vs NodeList

The main difference between an [HTMLCollection](https://developer.mozilla.org/en-US/docs/Web/API/HTMLCollection" \t "_blank) and a [NodeList](https://developer.mozilla.org/en-US/docs/Web/API/NodeList" \t "_blank) is that one is **live** and one is **static**. This means that when an element is [appended](https://developer.mozilla.org/en-US/docs/Web/API/Node/appendChild) to the DOM, a live node will recognize the new element while a static node will not.

1. HTMLCollection

The element methods *getElementsByClassName()* and *getElementsByTagName()* return a live HTMLCollection. It only includes the matching elements (e.g. class name or tag name) and does not include text nodes, it provides only two methods *item* and *namedItem*.

In the example below, all the elements with the class name of *fruits* is selected. The *item()* method is then used to access the fruit at index 0 and a class name of *fruit\_\_01* is added to that element.

const fruits = document.getElementsByClassName(‘fruits’);  
fruits.item(0).classList.add(‘fruit\_\_01’)

2. NodeList

The element method \*querySelectorAll()\* returns a static NodeList. They look like arrays but are not.

NodeLists have a defined forEach method as well as a few other methods including *item*, *entries*, *keys*, and *values*.

NodeLists behave differently depending on how you access them; if you access elements using *childNodes*, the returned list is live and will update as more elements are added to the node. If it’s accessed using *querySelectorAll()*, the returned list is static and will not update if more elements are added to the node.

// returns static collection  
 const fruits = document.querySelectorAll(‘.fruits’);// returns live collection  
 const fruits = document.querySelector(‘.fruits’);  
 const childFruit = fruits.childNodes;