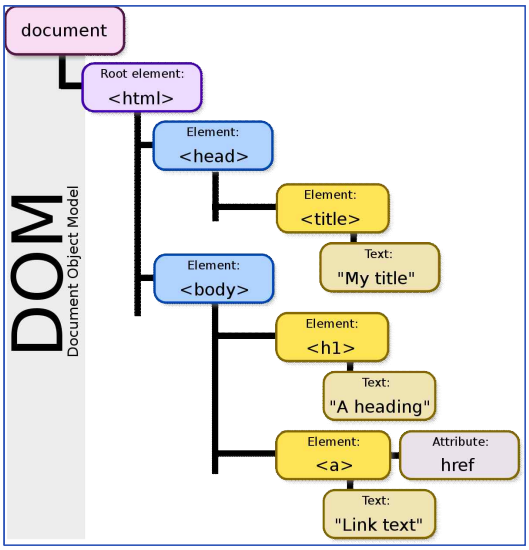
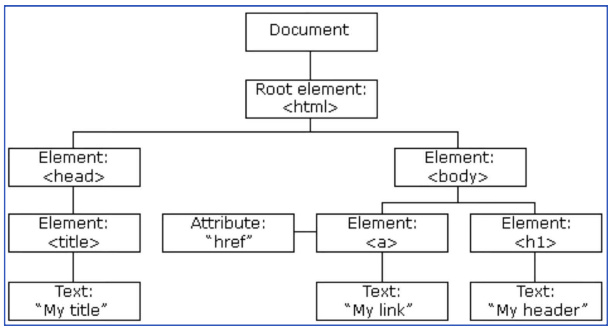
JavaScript에서 DOM, Event 활용

☑ DOM(Document Object Mdel)

DOM은 HTML 문서의 각 element(요소)들을 Tree 형식으로 표현해 준다. 개발자는 JavaScript를 이용해서 생성하거나 수정하거나 삭제할 수 있다.

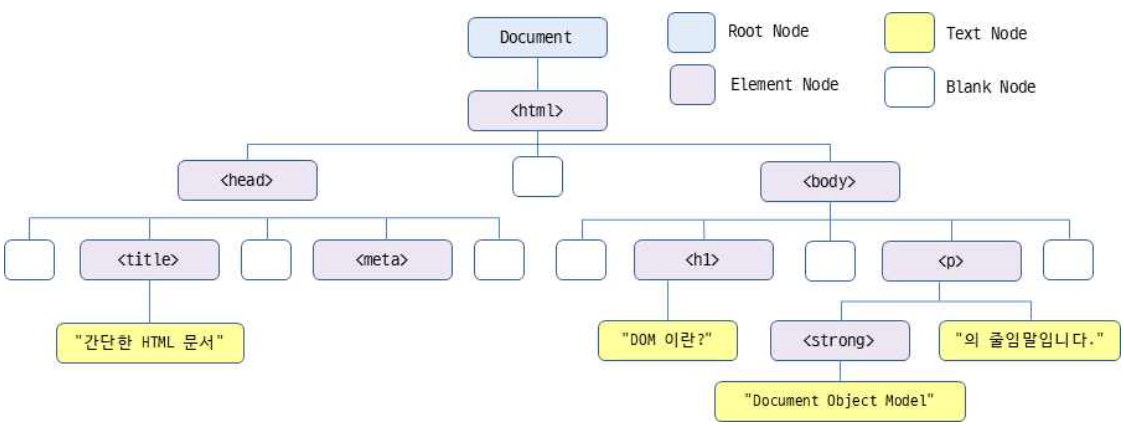




[DOM Tree 구조 예]

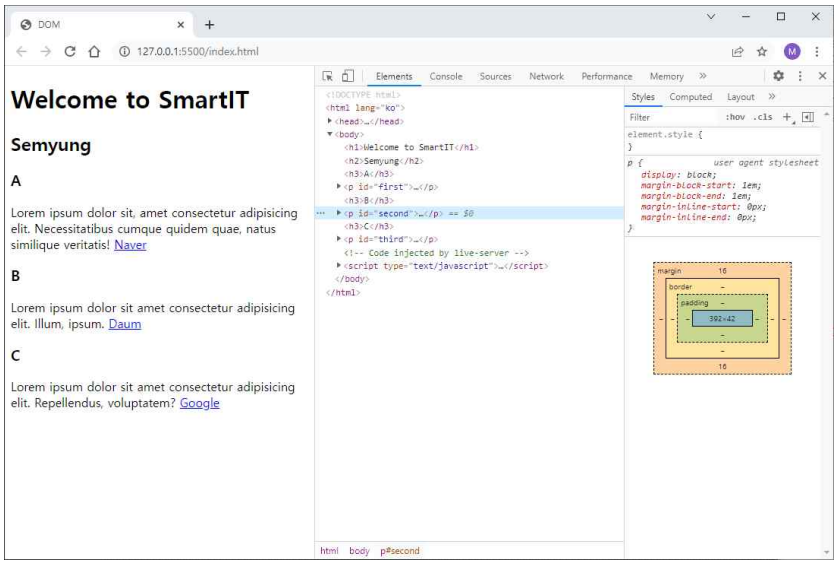
DOM Tree를 구성하는 Object 하나를 Node(노드)라고 한다. Tree에서 위쪽은 parent node(부모 노드), 아래쪽을 child node(자식 노드) 라고 한다. Document를 제외하고 최상단에 있는 html은 root node가 된다.

- ☞ root node : 전체 document를 가리키는 Document Object 이며 document로 참조할 수 있다.
- ☞ element node : HTML element를 가리키는 Object(=> element Object) 이다.
- ☞ text node : text를 가리키는 Object(=> text Object) 이다.



DOM은 Tree 구조의 자료 구조 이다. 이 하나의 객체를 Node 라고 부른다. Tree에서 위쪽은 parent Node, 아래쪽을 child Node 라고 한다. Document를 제외하고 최상단에 있는 <html>은 root Node 가 된다.

Chrome 에서 개발자 도구를 열었을 때 보이는 [Elements] 부분이 바로 DOM 이다.

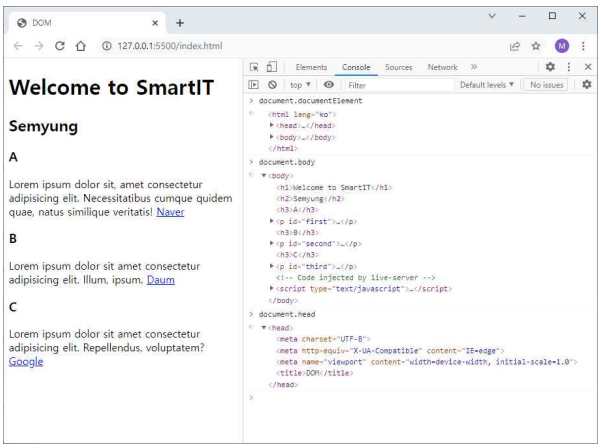


이제부터 console 창을 열고 하나씩 접근해 보도록 한다.

document.documentElement; 하면 <html> tag에 접근할 수가 있다.

document.body; 하면 <body> tag에 접근할 수가 있다.

document.head; 하면 <head> tag에 접근할 수가 있다.



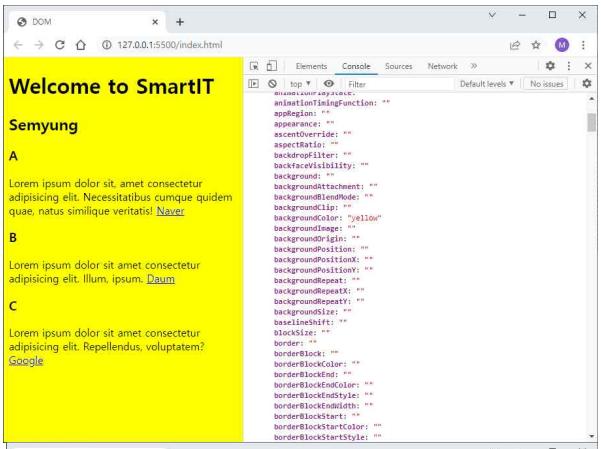
모든 html tag는 객체이다. 이 객체는 JavaScript로 접근하고 제어할 수 있다.

style을 변경하도록 한다.

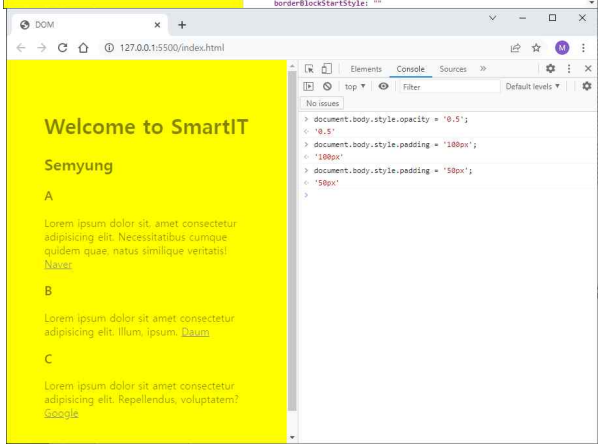
document.body.style; 을 하면 다음과 같이 객체가 출력이 된다. 많은 style들이 보인다.

여기서 backgroundColor="yellow" 로 변경하면 적용되는 것을 볼 수가 있다.
[Elements] 에서 보면 적용된 것을 볼 수 있다.

이렇게 하지 않고 직접 작성하도록 한다.



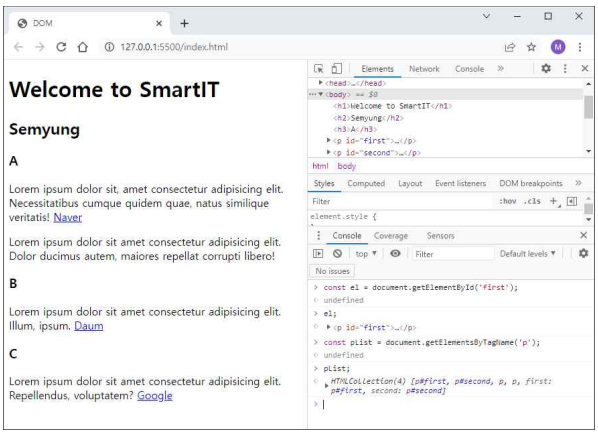
`document.body.style.opacity = '0';` 으로 하면 마무것도 보이지 않는다.
`document.body.style.opacity = '0.5';` 로 하면 반투명으로 보인다.



`document.body.style.padding = '100px';` 로 하면 padding이 적용된 것을 볼 수가 있다.

첫번째 <p> tag 에서 id="first" 로 되어 있는데 id 로 접근을 해 보도록 한다. id 로 접근할 때는 `document.getElementById('first');` 로 할 수가 있다.

```
const el = document.getElementById('first');
el;
```



tag 명으로도 찾을 수 있다. 모든 <p> tag에 접근하도록 해 보자. 4개가 있다고 나온다.

```
const pList = document.getElementsByTagName('p');
pList;
```

id는 한 page에 딱 한번만 사용할 수 있다. 따라서 `getElementById()` 가 된다. s 가 없다. 반면에 <p> tag는 여러번 사용하여도 된다. 따라서 `getElementsByTagName()` 가 되는 것이다. 복수형이다.

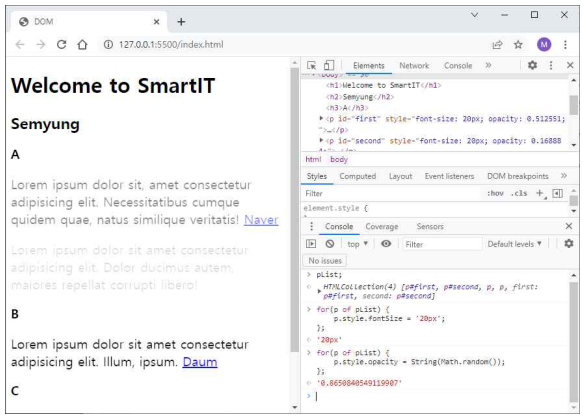
pList는 for...of 로 순회할 수 있다.
fontSize를 '20px' 로 키워보고,

```
for(p of pList) {
    p.style.fontsize = '20px';
}
```

투명도를 random 한 값으로 해 보자.

```
for(p of pList) {
    p.style.opacity = String(Math.random());
```

```
}
```



확인해 보면 fontSize는 20px이 되었고, 투명도는 랜덤한 값이 들어간 것을 확인할 수 있다.

getElementsByClassName(), getElementsByName()를 이용하면 ClassName이나 Name으로도 Node에 접근할 수가 있다. 'link' 라는 clas ㄴ를 가진 요소가 3개가 나온다.

```
document.getElementByClassName('link');
```

```
> document.getElementsByClassName('link');
◂ ▸HTMLCollection(3) [a.link, a.link, a.link]
```

다음 학습할 검색 method는 querySelector() 와 querySelectorAll() method 이다. 자주 사용되는 method 이다. 이 부분을 이해하기 위해서는 CSS의 Selector를 이해하고 있어야 한다.
앞에서 'link' class를 가진 요소를 접근해 보았다. 이것을 다음과 같이 작성할 수가 있다. CSS 문법에서 class는 점(.)으로 접근할 수 있다. 이렇게 하여도 동일한 결과를 가져 온다. id도 # 으로 적어주면 된다.

```
document.querySelectorAll('.link');
document.querySelectorAll('#first');
```

```
> document.getElementsByClassName('link');
◂ ▸HTMLCollection(3) [a.link, a.link, a.link]
> document.querySelectorAll('.link');
◂ ▸NodeList(3) [a.link, a.link, a.link]
> document.querySelectorAll('#first');
◂ ▸NodeList [p#first]
> document.querySelector('#first');
◂ ▸<p id="first">…</p>
```

그러나 id를 선택하면 하나만 출력이 되기 때문에 이럴 땐 document.querySelector('#first'); 로 하면 된다. querySelectorAll()은 모든 Node를 가져오고, querySelector()는 제일 처음 Node 만 가져온다.
CSS Selector를 사용하였을 때 편리한 점은 각 제목들이 <h3> tag로 되어 있는데 다음과 같이 작성해 보자.

```
document.querySelector('h3:nth-of-type(2)');
```

이렇게 하면 두번째 <h3> tag 만 찾는다. 또한 다음과 같이 하면 이렇게 빨간색으로 바꿔줄 수 있다.

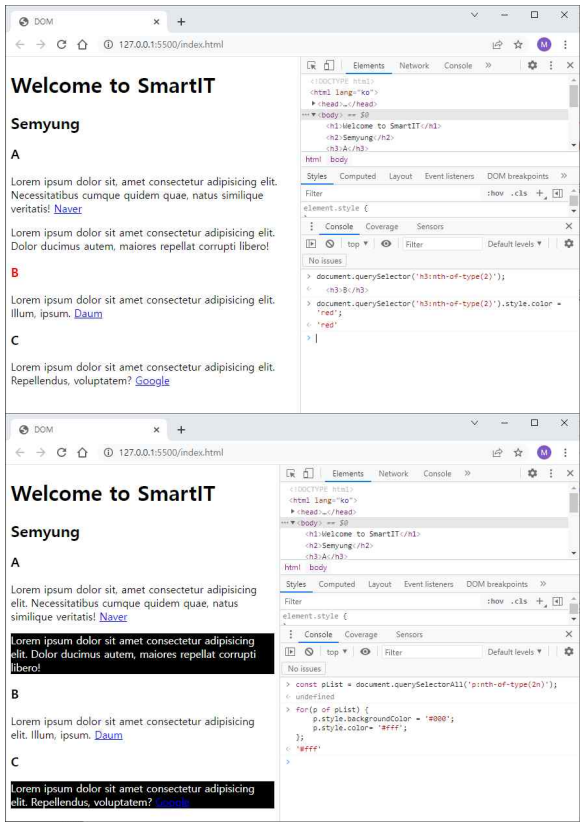
```
document.querySelector('h3:nth-of-type(2)').style.color = 'red';
```

pList에서 다음과 같이 하면 짝수 번째 p 들만 선택이 된다.

```
const pList =
document.querySelectorAll('p:nth-of-type(2n)');
```

for...of로 순회해 보도록 한다. 배경색은 검정색, 글자색은 흰색으로 해 보면 흑백이 번갈아 가면서 변경되는 것을 볼 수 있다.

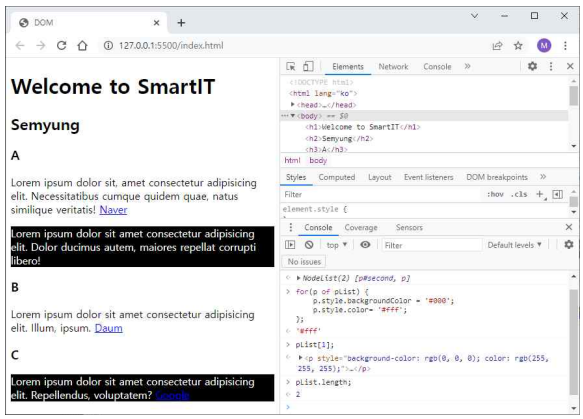
```
for(p of pList) {
    p.style.backgroundColor = '#000';
    p.style.color = '#fff';
}
```



지금까지 사용한 method로 가져온 결과들은 Array 처럼 보이지만 사실 Array는 아니다.
pList; 로 보면 NodeList() 로 출력이 된다.
NodeList는 Array은 아니고 iterable한 collection 이다. 따라서 for...of를 사용하는 것이다. 이렇게 index를 통하여 각 Node에 접근할 수가 있다.
pList[1]; 하면 두번째 요소가 나온다.

pList.length; 를 이용해서 갯수도 알 수도 있다.

그러나 Array가 아니기 때문에 push(), pop(), filter(), join() 같은 method는 사용할 수가 없다.



Node에는 다양한 type 이 있다.
개발할 때는 html element에 주로 접근을 한다. 이후에는 이 type을 구분해 보고, 전체 Node에 접근하는 방법과 element Node에만 접근하는 방법 그리고 parent, child, sibling Node에 접근하는 방법에 대하여 학습하도록 한다.

DOM과 Node에서 getElementXXX()와 querySelector()를 통하여 접근을 해 보았다. 두 method는 반환값이 조금 다르다. document.querySelectorAll('p')와 document.getElementsByTagName('p')는 동일한 동작을 하는 것 처럼 보이지만, 반환값은 NodeList와 HTMLCollection 으로 다르다.

```
> document.querySelectorAll('p');
< ▶ NodeList(4) [p#first, p#second, p, p]
> document.getElementsByTagName('p');
< ▶ HTMLCollection(4) [p#first, p#second, p, p, first: p#first, second: p#second]
```

둘다 Pseudo Array(유사 배열) 객체이면서 iterable 이다. 따라서 for...of 으로 순회를 하였다.

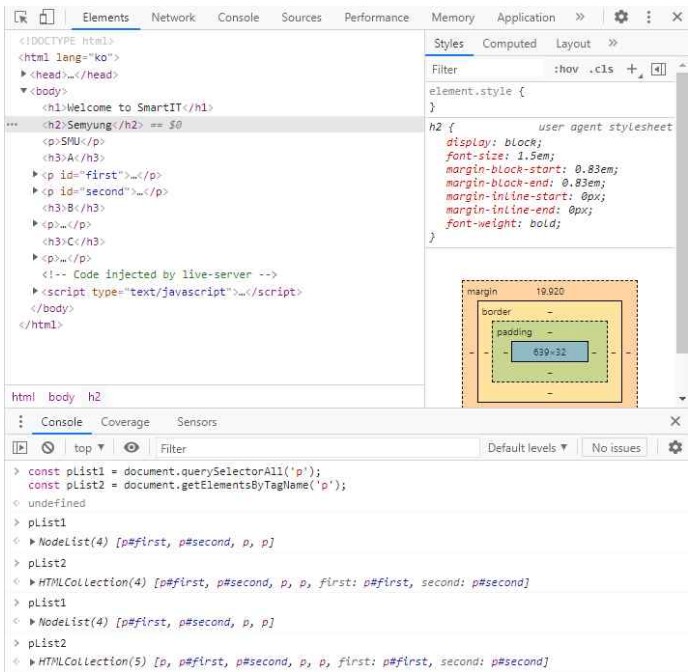
차이점을 알아 보도록 한다. pList1과 pList2를 선언한다.

```
const pList1 = document.querySelectorAll('p');
const pList2 = document.getElementByTagName('p');
```

pList1은 querySelectorAll로 가져온 리스트이고, pList2는 getElementByTageName 으로 가져온 리스트이다. pList1은 NodeList(4)이고 배열 길이는 4개 이다. pList2는 HTMLCollection(4) 이고 같이 배열 길이는 4개 이다.

index.html 파일에 <p> tag를 하나 추가해 보자. 추가한 후 다시 출력해 보면, pList1은 NodeList(4)이고 값이 변하지 않았다. pList2는 HTMLCollection(5) 이고 배열 길이가 5개로 변경되었다.

분명히 선언은 <p> tag 추가하기 전에 하였고, 코드를 다시 작성하지 않았는데 HTMLCollection()은 자동으로 반영이 되어있다.
Node은 변경사항이 실시간으로 반영이 된다.
그에 반에 NodeList()는 한번 저장된 값을 계속 사용한다.



대부분의 경우에는 이렇게 동작을 하지만 아닌 경우도 있다.

이제부터 parent, child, sibling Node에 접근해 보도록 한다.
여기 색상 리스트가 있고, Red, Blue, Green 이 있다.
여기에서 Red르 선택해 보도록 한다.

```
const red = document.getElementById('red');
```

parent Node에 접근해 보도록 한다. red.parentNode; 로 하면 parent Node에 접근할 수 있다.
지금 red에서 ul로 접근을 하였다. red.parentElement; 로도 접근할 수 있다. 동일하게 ul이 나온다.

앞에서 document.documentElement; 는 html 이었다. 여기서에 html의 parent는 무었일까?
document.documentElement.parentNode; 를 하면 #document가 나오고, document.documentElement.parentElement; 를 하면 null이 나온다.

이유는 parentNode는 parent Node를 반환한다. 여기서 html의 parent Node는 document가 되는 것이다.
반면 parentElement는 parent Node 중에 element(요소) Nodea 만 반환한다.
element 란 html tag로 이루어진 element(요소)를 의미한다.

Node는 다양한 type 이 있다. 문서에는 총 12개 있다. 5, 6, 12번은 더 이상 사용되지 않는다.
주요 type 만 살펴보면, 1번이 Node.ELEMENT_NODE 이다. <p>, 같은 element 이다.
2번은 Node.ATTRIBUTE_NODE 이다. 임시 tag의 alt="" 값 같은 것이다.
3번은 Node.TEXT_NODE 이다. 8번이 Node.COMMENT.NODE, 9번이 Node.DOCUMENT_NODE 이다.

parentNode는 parent Node 중 모든 Node를 반환한다. 반면 parenElement는 parent인 Node 중 element Node만 반환한다. document는 element Node가 아니었다. 따라서 html에서 parent element Node를 찾으면 null이 나왔던 것이다.

이제 child Node를 찾아보도록 한다. id="color" 인 ul에서 child 값을 찾아보도록 한다.

```
const ul = document.getElementById('color');
```

2가지 방법이 있다. ul.childNodes; 는 child Node들 전부를 의미한다.
그리고 ul.children; 은 child Node 중에 element Node 만 반환한다. 갯수에서 차이가 난다.

보통 상황이라면 3이 나오는 것이 맞다. ul의 child는 li 3개 이기 때문이다.
그러나 childNodes는 9개가 나왔다.
열어보면 li 요소들 이외에도 text들과 주석을 포함한 모든 type의 요소를 반환한 것이다.
코드를 보면 과 <li> 사이에 text가 없지만 왜 text Node가 함께 출력이 된 것일까? 이것은 html이

parsing 된 후에 DOM이 생성할 때 생기는 공백 문자 이다. space나 줄바꿈도 text Node로 함께 잡힌다. 코드에서 줄바꿈이 되고 가 있고, comment가 있고, 줄바꿈과 li 이런 식으로 하여 9개가 나온 것이다. 공백을 삭제해 보면 4개가 나온다. 권장하지는 않는다.
html을 접근할 때는 element Node 만 선택하는 것이 편리하다. 여기서는 ul.children;을 사용하는 것이다.
ul.childNodes; 를 하면 NodeList 가 출력되고, ul.children; 을 하면 HTMLCollection 이 출력된다. 이제 이 차이점이 보인다.
NodeList는 모든 type의 리스트들이고, HTMLCollection은 element type의 Node들만 모인 것이다.

HTMLCollection은 Node의 변경 사항이 실시간으로 반영되었다. NodeList는 아니었다. 예외적으로 childNodes는 HTMLCollection 같이 실시간 반영이 동작한다. 이 부분을 기억하도록 한다.

ul.firstChild; 하면 첫번째 Node를 가져오고, ul.lastChild; 하면 마지막 Node를 가져온다.

```
> const ul = document.getElementById('color');
< undefined
> ul.firstChild;
< ▶ #text
> ul.lastChild;
< ▶ #text
> ul.firstElementChild;
< ▶ <li id="red">…</li>
> ul.lastElementChild;
< ▶ <li id="green">…</li>
```

이번에는 첫번째와 마지막 element Node를 가져오도록 해 본다.
ul.firstElementChild; ul.lastElementChild; 로 하면 첫번째와 마지막번째 element Node 만 가지고 온다. 이 2가지 method가 많이 사용된다.

이번에는 sibling(형제) 이다. 2번째 'blue;를 기준으로 학습하도록 한다. sibling Node는 이전 Sibling과 다음 Sibling으로 나뉜다.
blue.previousSibling; 과 blue.nextSibling; 이다. 이것도 text가 출력이 된다. 이것은 sibling Node중에 모든 type을 가지고 오는 것이다.
원하는 결과를얻기 위해서는 blue.previousElementSibling; 과 blue.nextElementSibling; 으로 하여야 'red'와 'green'이 출력이 된다.

```
> const blue = document.getElementById('blue');
< undefined
> blue.previousSibling;
< ▶ #text
> blue.nextSibling;
< ▶ #text
> blue.previousElementSibling;
< ▶ <li id="red">…</li>
> blue.nextElementSibling;
< ▶ <li id="green">…</li>
```

표로 정리해 보면, parent의 모든 Node에 접근하기 위해서는 parentNode를 사용한다. element Node에만 접근하기 위해서는 parentElement를 사용한다.

child에 접근할 때 모든 Node에 접근하기 위해서는 childNodes를 사용한다. element Node에만 접근하기 위해서는 children을 사용한다.

firstChild와 lastChild는 첫번째와 마지막 Node를 반한한다. firstElementChild와 lastElementChild는 첫번째와 마지막 자식 중 elementNode 만 반환한다.

형제에 접근하는 법은 모든 Node에 접근할려면 이전 Node는 previousSibling, 다음 Node는 nextSibling으로 접근할 수 있다.
만약 element Node에만 접근하고자 하면 previousElementSibling과 nextElementSibling을 사용하면 된다.

Items	모든 Node	element Node 만
parent	parentNode	parentElement
child	childNodes firstChild lastChild	children firstElementChild lastElementChild
sibling	previousSibling nextSibling	previousElementSibling nextElementSibling