프론트 정리

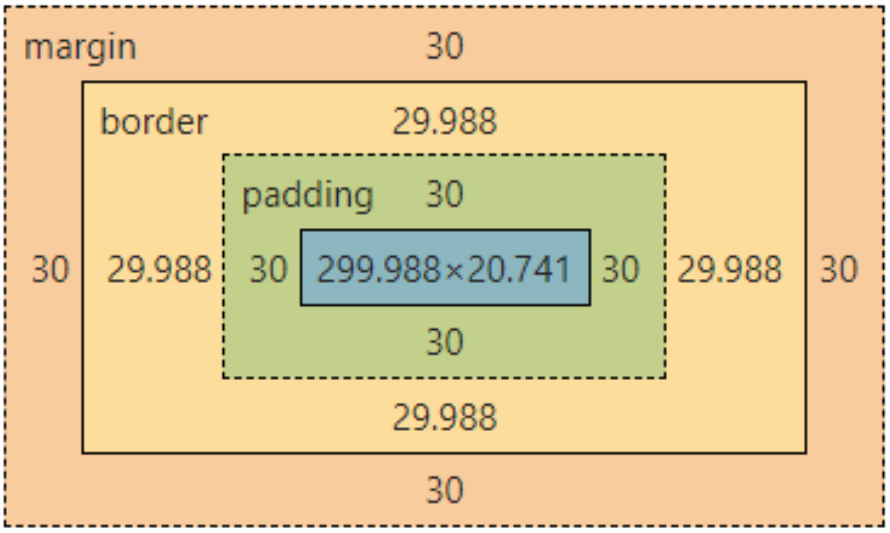
HTML

1. **시맨틱 태그**
    - `<div>` 태그에 의미를 부여하는 것
    - `<header>`, `<footer>`, `<nav>` 등과 같은 시맨틱 태그는 문서의 구조와 의미를 명확하게 함
2. `<a>`
    - `href` : 링크의 목적지를 지정
3. `<form>`
    - `<form>`: 사용자 입력 데이터를 서버로 전송하는 기능
        - `action` : 데이터를 전송할 URL
        - `method` : 데이터를 전송할 방식 *(GET, POST 등..) Back-end 과목에서 배울 예정*
    - `<input>` 태그 안의 내용이 전송됨
4. `<input>`
    - `value`
        - 태그 안의 기본값 설정 (처음부터 작성되어 있는 텍스트)
    - `placeholder`
        - 입력을 시작하면 지워지는 안내 텍스트
    - `type`
        - password
        - number
        - text
    - `readonly`
        - text 쓰기 불가
    - `disabled`
        - 회색 음영으로 표시되며 선택 불가

5. `<script>`
    - 외부 JavaScript 파일을 로드하는 데 사용

CSS

1. **Position**
    - 요소의 배치 방식을 결정하는 속성,
    - `left` , `right` , `top` , `bottom` 으로 상하좌우 얼마나 이동할지 설정 가능
        - `static` : 기본값 `left` , `right` , `top` , `bottom` 가 적용되지 않음
        - `relative` : 원래 배치될 위치(static일 때)를 기준으로 위치를 결정
        - `absolute` : 부모 요소를 기준으로 위치를 결정
2. **Flexbox**
    - `flex-direction` : Flexbox 컨테이너 내에서 항목이 배치되는 방향(main축)을 결정
    - `justify-content` : main축 방향으로 나열 될 때 정렬 기준을 설정
    - `align-items` : Flexbox 컨테이너 내의 요소를 수직으로 중앙에 정렬
    - `flex` : Flexbox의 자식요소들이 차지하는 비율을 결정
3. **CSS 박스 모델**
    - 모든 `HTML element` 는 4가지 영역으로 구성된다
        - `margin` : 요소의 외부 여백 (다른 element와의 간격)
        - `border` : 테두리
        - `padding` : 요소의 내부 여백
        - `content` : 자식 요소들이 포함되는 공간



- `box-sizing`
  - `content-box` : height와 width는 content 영역 에만적용됨
  - `border-box` : height와 width가 border + padding까지 포함하여 적용됨

4. **CSS Display & Visibility**
   - `display : none` : 요소를 없는 것으로 취급함
   - `visibility: hidden` : 요소를 화면에서 보이지 않게 하지만, 해당 요소가 차지하는 공간은 유지
5. **CSS 배경 속성**
   - `background-color` : 배경 색상을 지정해줌
   - `background-image` : 요소의 배경으로 이미지를 설정
6. **CSS 우선순위**
   - 같은 요소에 여러가지 방법으로 CSS가 적용되어 있다면 우선순위는 아래와 같다.
     - `!important`
     - `inline`
     - `id로 지정`
     - `class로 지정`
     - `tag이름으로 지정`
     - `전체(*) 지정`

- `외부 css파일`

JavaScript

1. **데이터 타입**
   - `number`, `string` , `boolean`, `null` , `undefined` , `object`
   - JS는 타입을 엄격하게 관리하지 않기 때문에, 서로 다른 타입을 더하는 것도 가능
     - **기본적으로는 서로 다른 타입을 더하면 문자열의 형태가 됨**
   - `undefined` : 변수가 정의되어 있지 않다, 값이 할당된 적이 없다.
   - `null` : 가리킬 주소가 없다는 의미
   - `NaN` : number타입이지만 유효한 숫자가 아니라는 의미
2. **함수**
   - `Javscript`의 함수는 1급 객체이다.
     - 변수와 같이 취급되며
     - 함수의 매개변수 및 return 값으로 취급될 수 있다.
   - 선언

```
// 함수 선언식
function myFunction() {
}

// 함수 표현식
const myFunction = function() {
}

// 화살표 함수
const myFunction = () => {};
```

3. **객체**
   - `let obj = {};`
   - `Javascript`의 객체는 key와 value의 쌍으로 데이터를 저장하며 `JAVA`의 `Map`과 유사하다.
   - 객체의 속성에 접근할 때

- `obj.a`
- `obj['a']`
- 하이픈이 포함된 경우는 반드시 `object["new-key"] = "value";`와 같은 형태 사용

4. **배열**

- `let arr = [];`
- 관련 메소드
  - `shift`
  - `unshift`
  - `slice`
  - `splice`
  - `pop`
  - `push`
- js의 배열은 크기를 동적으로 바꿀 수 있으며, 임의의 인덱스에 자유롭게 접근 가능

```
let a = [];
// 새로 생성된 빈 배열의 100번 인덱스에 바로 접근가능
a[100] = 1;
```

5. **비교 연산자**

- `==` 연산자는 타입을 자동으로 변환하여 값을 비교
  - 예: `5 == "5"`는 `true`를 반환
- 비교 연산자 `===`는 타입까지 비교하여 엄격한 비교

6. **DOM**

- CSS 선택자를 활용하여 DOM 요소 (HTML element)에 접근가능
- DOM 요소에 접근하는 방법 중 **일부**

querySelector

- `document.querySelector`

querySelectorAll

- `document.querySelectorAll`

getElement

- `document.getElementById`

getElements

- `document.getElementsByClassName`

7. **JavaScript Event**
   - `addEventListener()` : 지정한 유형의 이벤트를 대상이 수신할 때마다 호출할 함수 설정
   - 예시

```
let btn = document.querySelector("#button");
btn.addEventListener('click', () => {confirm("Hi");});
```

8. **var let const**
   - javascript에서 변수는 type으로 선언하지 않고 `var` `let` `const` 를 사용함.
     - `var` : 기존 변수 선언 방식
   - ES6 이후 추가된 방식
     - `let` : 재할당 가능
     - `const` : 재할당 불가능
9. **호이스팅**
   - 변수 선언과 함수 선언이 코드의 최상단으로 끌어올려지는 현상
   - 변수가 선언되기 전 코드 영역에서 해당 변수는 힙 영역에 할당된다
   - `var` `let` `const` 으로 선언한 변수가 모두 호이스팅 되지만 차이가 있음.
     - `var` 는 선언되기 전에 접근하는 경우에는 undefined
     - `let` 과 `const` 는 선언되기 전에 접근하면 TDZ(Temporal Dead Zone)가 적용되어 명시적으로 오류를 발생시킨다. (안전성 높은 프로그램을 위해 let const 사용을 권장함)
10. **this** (디테일하게 몰라도 됨)

- `Java`의 `this`와 유사하게, 특정 객체를 가리키는 키워드이다. (그러나 차이점에 유의 필요)
- this는 일반 함수와 화살표 함수에서 다른 객체를 의미한다.
  - 일반 함수에서 : **호출한 객체**
  - 화살표 함수에서 : **함수를 정의하고 있는 객체**
- 💡**일반함수에서 this를 사용하게 되면 상황에 따라 가리키는 객체가 달라질 수 있으므로 주의, this를 쓸 필요가 있는 경우 화살표 함수를 쓰는 것을 권장.**