08.16_실습 강의자료

1교시

강의 이해도 체크,

CSS Selector 문제 체크

flex 문제 체크

핑시트 - 응답 / 디스코드 - 질문(한문제 끝날때마다 확인) / 채팅 - 강의실

2-01 보물지도 (meta, transition)

▼ meta태그에 viewport를 넣는 이유가 무엇인가요?

뷰포트란 현재 보고있는 컴퓨터 그래픽의 영역을 뜻하는데요.

데스트탑에서는 쉽게 생각해 브라우저의 창 크기라고 생각하시면 편합니다.

하지만 모바일의 영역에서는 뷰포트가 보이는 창보다 크거나 작을 수 있습니다.

따라서 데스크탑 에서는 meta 태그가 없어도 정상적으로 작동하지만 모바일 화면에서 볼때는 meta viewport설정을 해줘야 합니다!

```
<meta name="viewport" content="width=device-width, initial-scale=1">
```

width=device-width : 뷰포트의 너비를 장치의 너비로 설정

initial-sacle=1 : 사용자가 페이지를 방문할 때 초기 확대/축소 수준 설정

▼ meta태그에는 또 뭐가 들어가나요?

meta태그는 웹 서버와 웹 브라우저간에 상호 교환되는 정보를 정의하는데 사용합니다.

meta의 속성으로는 charset, http-equiv, name, content 4가지 속성이 있습니다.

charset = “인코딩방법” : 페이지의 문자 인코딩을 선언

http-equiv = “항목값" : 웹 브라우저가 서버에 명령을 내리는 속성

name = “정보 이름" : 몇개의 정보의 이름을 정할 수 있으며 지정하지 않으면 http-equiv와 같은 기능을 합니다.

content = “정보값” : meta 정보의 내용을 지정합니다.

(참고 : https://developer.mozilla.org/ko/docs/Web/HTML/Element/meta)

ex)

3초 후 지정 url로 자동이동

```
<meta http-equiv="refresh" content="3;url=지정한 웹페이지 주소">
```

▼ 미디어쿼리는 무엇이고 어떻게 쓰나요?

미디어 쿼리는 반응형으로 UI를 만들기 위해서 꼭 필요합니다. pc, 모바일, 태블릿은 화면의 크기가 다 다르죠?

우리가 만드는 UI를 유저가 보는 화면 마다 따로따로 작업을 한다면 동일한 작업을 적어도 3번을 해야됩니다. 하지만 미디어 쿼리를 사용하면 width의 크기 별로 CSS를 다르게 줄 수 있답니다.

이렇게 미디어 쿼리를 사용해서 프론트엔드를 구현하는것이 반응형으로 웹페이지를 만드는 것의 시작입니다!

미디어 쿼리의 사용방법은

```
@media (min-width: 360px) and (max-width: 720px) {
    .map {
        background-color: blue;
        transition: background-color 1.5s;
    }
    .map:hover {
        background-color: orange;
    }
}
```

(참고 : https://developer.mozilla.org/ko/docs/Web/CSS/@media)

이런식으로 조건에 따라서 CSS를 변경시켜 줄 수 있습니다.

▼ Transition은 무엇이고 어떻게 사용하나요?

Transition은 특정 요소에 특정한 조건에 해당될 때 효과를 적용할 때 사용하며

CSS 값이 변할 때 (특정 조건) 특정한 효과를 적용하는 것입니다.

트렌지션은 자동으로 발생하지 않고 어떠한 트리거(조건)가 있어야 발동합니다.

```
div:hover{

    /* 효과를 적용 할 대상 지정 */
    transition-property: width, height;
    /* 변경되는 모든 것에 효과를 적용 */
    transition-duration: 1s;
    /* 변경되는 방법 */
    transition-timing-function: linear;
    /* 변경이 시작되는 시간  */
    transition-delay: 1s;

    /* 축약형*/
    transition: width 1s linear 1s

}
```

(참고 : https://developer.mozilla.org/ko/docs/Web/CSS/CSS_Transitions/Using_CSS_transitions)

▼ :hover와 같은게 또 뭐가 있나요?

저번시간에 css selector 문제를 풀어보신 분들이라면 :기호를 몇개 보셨을 텐데요 css selector 에서 첫번째 자식을 선택했던 :firtst-child 도 가상 클래스 선택자 중 하나입니다

이렇게 :가상이벤트 형태로 나타나져있는 것을 가상 클래스 선택자라고 합니다.

이렇게 가상클래스를 사용하게되면 장점은 오직 css를 위해서 새롭게 클래스를 추가할 필요없이 내가 원하는 요소를 선택할 수 있습니다🙂

가상 클래스 선택자의 종류

- `:link` - 방문한 적이 없는 링크
- `:visited` - 방문한 적이 있는 링크

- `:hover` - 마우스를 롤오버 했을 때
- `:active` - 마우스를 클릭했을 때
- `:focus` - 포커스 되었을 때 (input 태그 등)

- `:first` - 첫번째 요소
- `:last` - 마지막 요소
- `:first-child` - 첫번째 자식
- `:last-child` - 마지막 자식
- `:nth-child(2n+1)` - 홀수 번째 자식
- …

▼ : 와 ::의 차이는 뭔가요?

: 는 해당 요소 전체에 적용되는 가상선택자에 적용하는 **가상클래스**라고 하고

ex) :link / :hover / :first-child / :nth-child(n) 등

:: 는 해당 요소의 일부분에 적용되는 가상선택자에 사용하는 **가상요소**라고 합니다

ex) ::before / ::after / ::first-line / ::first-letter 등

가상클래스(:) 같은 경우는 실제 존재하는 요소에 클래스 추가없이 디자인을 입히는 것이라면

가상요소(::) 같은 경우는 실제로 존재하지 않는 요소를 만들어주는 것이에요.

존재하지 않는 요소에 css를 입히기 때문에 html로 특정짓기 어려운 부분에 손 쉽게 스타일을 적용할 수 있어요.

html, js도 버전이 있듯이 CSS도 버전이 있습니다ㅎㅎ

::는 CSS3문법으로 바뀌면서 생겨났어요!

호환성을 위해서 :로 적어도 적용은 되긴 합니다 😊

ex)

```
<style>
    p::after{
        content: "cm"
        }
    p::before{
        content: "~"
        }
</style>
<body>
    <p>10</p>
</body>
```

```
<!-- ~10cm 출력 -->
```

(참고 : https://www.w3schools.com/css/css_pseudo_elements.asp)

(참고 : https://velog.io/@codns1223/WIL-가상클래스-선택자-가상요소-선택자)

2교시

2-02 수박쪼개기(미디어쿼리, border)

▼ 미디어쿼리 보충

미디어쿼리의 width를 지정하는 방법은 and를 사용하는 것도 좋지만

가독성과 편의성을 높힌 (min-width < width < max-width) 를 사용하는것이 더 편리합니다.

▼ border는 순서가 상관없는거 아니였나요?

맞습니다! 순서가 상관없지만 엘리스의 채점코드에는 반영되지 않나봐요ㅜㅜ 정상적으로 코드 작동하는 것 맞습니다!! 걱정하지 마세요~

▼ Transition기능도 추가해볼까요?

3교시

2-03 코더랜드 홈페이지 개편

▼ Transform Transition Animation의 차이는 뭔가요?

정리하자면

Transform은 특정요소에 효과를 적용할 때 사용하고

Transition은 특정 요소에 특정한 조건에 해당될 때 효과를 적용할 때 사용하며

Animation은 특정한 조건 없이 효과를 적용할 때 사용합니다.

하나씩 예시보며 비교해보겠습니다.

Transform(트렌스폼)은 CSS의 요소를 회전, 크기 조정, 기울이기 등을 변화 시킬 수 있는 속성입니다.

```
#test1:hover {
    transform: translateX(50px);
    /* 사이즈 변경 */
    transform: scale(2, 0.5);
    /* 여러개 */
    transform: translateY(50px) rotate(45deg);
    ...
}
```

(참고 : https://developer.mozilla.org/en-US/docs/Web/CSS/transform)

Transition(트랜지션)은 CSS 값이 변할 때 (특정 조건) 특정한 효과를 적용하는 것입니다.

트렌지션은 자동으로 발생하지 않고 어떠한 트리거(조건)가 있어야 발동합니다.

```
div:hover{

    /* 효과를 적용 할 대상 지정 */
    transition-property: width, height;
    /* 변경되는 모든 것에 효과를 적용 */
    transition-duration: 1s;
    /* 변경되는 방법 */
    transition-timing-function: linear;
    /* 변경이 시작되는 시간  */
    transition-delay: 1s;

    /* 축약형*/
    transition: width 1s linear 1s

}
```

(참고 : https://developer.mozilla.org/ko/docs/Web/CSS/CSS_Transitions/Using_CSS_transitions)

Animation(애니메이션)은 트렌지션처럼 변경하지만 트렌지션보다 여러가지의 종류로 다양하게 변화할 수있고 시작, 정지, 반복까지 제어할 수 있습니다.

```
#test4 {
    animation-duration: 1s;
    animation-name: sizeup1;
}

@keyframes sizeup1 {
    from {
        margin-left: 100%;
        width: 100%;
    }
```

```
    to {
        margin-left: 0%;
        width: 20%;
    }
}

#test4:hover {
    /* @keyframes 이름 */
    animation-name: sizeup2;
    /* 변화가 일어나는 기간. 초단위. (기본값 0s) */
    animation-duration: 1s;
    /* 반복횟수 */
    animation-iteration-count: 3;
    /* 무한반복 */
    animation-iteration-count: infinite;

    /* 축약형 */
    /* animation-name: sizeup2; */
}

@keyframes sizeup2 {
    0% {
        width: 100%;
    }

    50% {
        width: 25%;
    }

    100% {
        width: 100%;
    }
}
```

(참고 : https://developer.mozilla.org/ko/docs/Web/CSS/CSS_Animations/Using_CSS_animations)

▼ Animatioin 보충

from / to 로 시작과 끝 뿐만 아니라 %로 분할하여 나타낼 수도 있습니다!

Git

▼ git이란 무엇인가요??

git은 버전 관리 시스템입니다.

지금은 어떤 느낌인지 감이 잘 안 오시겠지만 강의를 듣고 실습을 따라해보시다 보면 조금씩 느낌이 오실껍니다.

먼저 git을 사용하지 않았을 때를 생각해보겠습니다.

1. A1을 개발하고 A2를 개발하고 A3를 개발했는데 A2,3 쓸모없어져서 A1으로 돌아가야 합니다. 어떻게 하면 될까요?
2. A를 개발하고 있다가 A와 다른 방법인 A'를 개발해야 할 때가 생겼습니다. 하지만 A는 건들고 싶지 않습니다. 어떻게 하면 될까요?
3. 여러분과 여러분 옆자리에 계신 레이서분들이 같이 작업을 하고 싶습니다. 어떻게 하면 될까요?

이러한 질문들의 효율적인 답을 찾기 위해 git이라는 버전관리를 사용하는 것입니다.

위의 질문에 대한 답은 차근차근 찾아나가 봐요~ 🙂

지금부터 여러분의 옆에 git이라는 사진기사가 있습니다. 이 친구를 이용해서 개발을 해볼꺼에요!

먼저 git을 사용하기 위해서 처음해야 하는 일이 있습니다.

(다운로드 : **https://git-scm.com/downloads**)

여러분이 작업을하는 공간을 git에게 알려주는 것입니다. **(Working Directory영역)**

```
git init
```

git에게 '여기 봐~ 나 여기서 작업할 꺼야~' 라고 알려주는 겁니다.

그 다음 git에게 알려줄 준비가 된 파일들을 저장해 줍니다.**(Staging Area영역)**

```
git add <파일/디렉토리 경로>
```

git에게 '나 이거이거 작업했어~' 라고 알려주는 겁니다.

그 다음 git에게 사진을 찍으라고 할 겁니다.**(Git Directory영역)**

```
git commit
```

라고 말하는 겁니다.

그러면 git은 사진을 찍고 찍은 사진에 번호표를 붙여서 잘 저장해 놀겁니다.

근데 사실 이 사진기사 git은 마법의 힘이 있습니다. 바로 자기가 가지고 있는 사진으로 시간을 되돌릴 수 있는 마법이에요. 그래서 여러분에 이전에 git한테 '사진찍어~' 라고 했을 때로 돌아갈 수가 있습니다.

그럼 1번 질문이 해결 되었죠?

돌아가는 방법은 어떻게 해야할 까요?

나머지 2, 3번질문은 금요일 git강의를 들으시고 나면 해결방법을 알 수 있게 되실 겁니다ㅎㅎ 화이팅~!

▼ git의 상태는 무엇이있나요?

git이 파일을 보는 관점은 크게 3가지의 상태가 있습니다.

Modified, Staged, Commited 이렇게 3가지 상태가 있는데요 하나씩 알아가 보겠습니다.

Modified는 여러분의 작업한 것을 git이 보았지만 아직 여러분이 git에게 말을 안 한 상태입니다.

예를들어 여러분이 A, B, C를 작업했지만 git에게 B, C만 작업했다고 말한다면 (`git add B C`) git 입장에서는 A가 수정은 되었지만 아직 자기한테 말은 안한 상태겠죠? 그 상태가 바로 **Modified** 한 상태입니다.

Staged는 여러분이 작업한 것을 git에게 알려준 상태입니다. 다른 말로 하면 git add를 한 상태, 다른 말로 하면 git에게 '나 이거 작업했어~'라고 말한 상태입니다. 바로 위의 예시로 치면 B, C 파일이 **Staged**한 상태입니다.

Commited은 여러분이 이제 git에게 사진을 찍으라고 한 뒤의 상태입니다. 이전의 예시를 다시 생각해보면 여러분은 A, B, C파일에서 작업을 했고 git에게 B, C만 알려줬습니다.

그리고 사진을 찍으라고 했으니깐 git은 B, C만 수정된 상태로 사진을 찍겠죠? git 입장에서는 '너가 알려준거는 한번에 다 찍었어' 라고 말하는 상태를 **Commited**한 상태라고 합니다. 쉽게 말하면 git commit 명령어를 친 상태에요. 현재 상태를 저장하겠다는 뜻이죠.

이렇게 commit을 하고나서 다시 파일을 수정한다면 수정된 파일들은 modified 상태로 돌아갑니다.

▼ 파일을 올리는 다양한 방법은?

git add를 하는 방법도 여러가지가 있습니다.

```
git add <파일/디렉토리 경로> // 지정한 파일 저장

git add <파일/디렉토리 경로> <파일/디렉토리 경로> ... //지정한 파일'들' 저장

git add . // 현재 디렉토리 이하의 모든 변경내용 저장

git add -A // working directory내의 모든 변경내용 저장

git add -p // 각 변경사항 터미널에서 직접 확인하고 저장
```

실수로 파일을 staged 상태로 옮겼다면 취소하는 방법도 있습니다.

```
git restore --staged <파일/디렉토리 경로>
```

이때는 반드시 파일을 지정해 줘야 합니다.

▼ git의 기록들 확인하기!

먼저 이전으로 되돌리고 싶으면 이전 기록들을 들여다 봐야겠죠?

이전에 git이 commit을하고 사진들을 모아둔다고 했었죠? 찍은 사진들에 이름을 써놓습니다.

그 이름들을 볼 수 있는게 `git log`라는 명령어에요

```
❯ git log
commit 96ecd66aed7869d06ac2b014ad685ace480525cf (HEAD -> master, origin/master, origin/
HEAD)
Merge: 0f80f28 30119c6
Author: hun2__2 <wognskec@hanyang.ac.kr>
Date:   Wed Nov 30 03:03:37 2022 +0900

    11.30 강의자료 코드 재업로드

commit 0f80f289d4bfd2dab68e83222596d27a759ec247
Author: hun2__2 <wognskec@hanyang.ac.kr>
Date:   Wed Nov 30 02:58:10 2022 +0900

    11.30 강의자료 코드

commit 30119c6903e682819bdedebbf070d6be37ed84a9
Author: hun2__2 <wognskec@hanyang.ac.kr>
Date:   Mon Nov 28 18:13:48 2022 +0900

    11.28 강의 시 사용한 코드 업로드

commit 4f8c70a2fa357850d31af07297574db2eff1ed6a
Author: hun2__2 <wognskec@hanyang.ac.kr>
Date:   Sat Nov 26 19:07:27 2022 +0900

    11.28강의자료 업로드
```

이러한 명령어 에 영어랑 숫자로 만들어진 어떠한 값이 보이시죠? 이 값이 git이 찍은 사진의 사진의 이름(해시주소) 입니다.

tip) 이쁘게 보기 위해서는

`git log --oneline` 를 통해 한줄로 볼 수 도 있습니다.

```
96ecd66 (HEAD -> master, origin/master, origin/HEAD) 11.30 강의자료 코드 재업로드
0f80f28 11.30 강의자료 코드
30119c6 11.28 강의 시 사용한 코드 업로드
4f8c70a 11.28강의자료 업로드
fc3a4da Update README.md
```

사실 git을 사용하기에 유용한 시각화 툴들이 있습니다. 하지만 CLI를 사용하는것에도 익숙해지시기를 추천드려요! 시각화에는 없는 기능들이 CLI에 훨씬 더 많거든요ㅎㅎ

▼ 이전으로 되돌리기

버전관리인 git에서 이전으로 되돌리는 방법은 크게 두가지가 있습니다. 하나씩 알아볼게요!

reset

이전에 작성했던 commit으로 돌아가고 싶으면 commit의 이름을 사용해서 돌아가는 방법이 있습니다.

```
git reset  <commit이름>
git reset --soft <commit이름>
git reset --mixed <commit이름>
git reset <commit이름>
```

여기에 hard, soft, mixed라는 옵션값들이 있는데요

hard : 돌아가려는 커밋 이후의 모든 내용을 지워 버립니다. 되돌린 commit상태와 완전히 동일합니다.

mixed : 돌아가려는 커밋으로 되돌아가고 staging area는 되돌린 commit상태와 동일하며 working directory는 그대로 유지합니다.

soft : 돌아가려는 커밋으로 되돌아가기만 합니다. staging area와 working directory는 아무런 변화도 없다.

어떠한 option도 주지 않으면 soft를 기본으로 실행합니다.

hard는 되돌린 commit 이후의 기록들은 쓰레기통에 버리는 것이기 때문에 조심해서 사용해야 겠죠??

하지만 매번 git log를 쳐서 commit의 이름을 알아내기가 귀찮을 수도 있잖아요? 그럴땐 현재상태를 기준으로 몇번째 전인가를 사용하면 됩니다.

```
git reset HEAD //이전 커밋
git reset HEAD^ // 이전 커밋, 1번째 전 커밋
git reset HEAD~1 // 이전 커밋, 1번째 전 커밋
git reset HEAD~n //n번째 전 커밋
```

하지만…. 여러분이 다른사람과 같이 작업을 하실 때 reset을 사용하게 되시면 세상이 깜깜해지고 손가락에 힘이풀리는 헬게이트가 오픈될 수 있습니다……

그래서 **revert**라는 명령어가 존재하죠 🙂

revert도 reset과 동일하게 커밋을 되돌리지만 reset은 커밋을 삭제하는 반면 revert는 커밋을 추가합니다.

revert는 reset —soft, mixed와 동일한 결과를 가져오지만 `Revert "…"` 라는 commit 메서지가 담긴 commit 이 추가됩니다.