JS Study

제목 없는 데이터베이스

#1 Basic of JavaScript

Variable

종류 및 선언 방식
const, let, var
const 와 letd의 차이점, var 사용하면 안되는 이유 > **항상 const, 업데이트가 필요한 변수는 let, var은 사용x**
boolean(true, false), null, undefined
let a = 5; 와 a = 7 의 차이 > 왼쪽은 변수를 생성과 동시에 값을 초기화 & 오른쪽은 변수의 값을 업데이트

Arrays

선언 방식, 값을 가져오는 방법, 값을 추가하는 방법(push, unshift),

Objects

array 와 objects의 차이점
array에 item을 넣을 때는 item마다 의미를 넣기 힘들지만 objects는 item마다 의미부여 가능!!
console은 objects
const 는 update 불가
const로 objects 를 선언 후 const를 다른 type으로 변환 불가
const objects = { } 선언 후 objects = bollean (x) but!! constant(objects) 안의 속성을 update하는 것은 가능!!
objects에 속성을 추가 objects.properties = "value";

Function

function 은 어떤 코드를 캡슐화해서 실행을 여러번 가능하게 한다.
( ) 는 function을 실행하는 방법
argument는 function을 실행하는 동안 데이터를 보내는 방법 > ( ) 안에 argument 값을 넣어서 function에 데이터를 전송
argument의 순서 중요
예) 필요한 argument가 하나만 존재할 때 전달되는 argument가 여러개가 전달될 경우, 제일 첫번째 argument를 값으로 받고
그 뒤에오는 argument는 받지 않는다.
argument 명은 지역변수로 존재
argument는 function의 body { } 안에서만 사용 가능 body 외부에서 접근 불가

objects 안에 function 선언

```
const objects = {
  function명: function (argument) {
  내용
```

```
  }
}
```

Returns

console.log를 사용하지 않고, function이 계산의 결과를 나에게 제공

console.log나 alert를 사용하면 결과를 console창이나 페이지 경고창으로 보여주고 끝,

function을 사용하는 궁극적인 목적은 function의 결과를 가지고 운영하는 데 있다.

return을 사용하지 않고, console.log를 사용하면 function의 결과 값은 undefined

하지만 return을 사용하면 그 계산 결과를 계속해서 활용 가능

```
const calculator = {
  plus: function (a, b) {
  alert(a + b);
  }
};

console.log(calculator.plus(2,3))
실행 결과 > 실제 calculator.plus의 값은 undefined
```

```
const calculator = {
  plus: function () {
  alert("hi");
  }
};

console.log(calculator.plus)
실행 결과 > function 명시
```

!!return 사용 시 function은 종료

```
const calculator = {
  plus: function (a, b) {
  console.log("hello")
  return a + b
  console.log("bye bye")
```

```
  }
}
실행 시 console.log("hello") 는 실행되지만,  console.log("bye bye")는 실행안됨
```

Conditionals

```
const age = parseInt(promt("How are you?"));

console.log(isNaN(age));
```

▼ **사용 function**

promt ( ): 사용자에게 입력을 받는 function (아주 오래된 사용 방법)

css 적용 불가, 브라우저에서 기본으로 지원

type of : value의 type을 확인 하기 위한 function

parseInt( ) : String type의 value를 Number type으로 변환

isNaN( ) : 무엇인가 NaN인지 판별하는 function

```
if(condition) {
  // condition === true
} else {
  // condition === false
}
```

condition(조건)은 boolean type 이어야 한다.(true or false)

```
if (isNaN(age) || age < 0) {         //조건문 ||(OR) 은 둘 중 하나만 TRUE면 TRUE
  console.log("Write A Number");
} else if (age < 18) {
  console.log("you are too young.");
} else if (age >= 18 && age <= 50) {  //조건문 &&(AND) 는 둘다 TRUE 일 때 TRUE
  console.log("you can drink");
} else {
  console.log("you can't  drink");
}
```

else 는 선택사항

= & == & === !==

= : 하나의 value를 할당 하는 것

=== : equals 같은지 판단

#2 JavaScript On The Browser

JavaScript는 HTML에 접근하고 읽을 수 있게 설정되어있다.

브라우저가 HTML 정보가 아주 많이 들어 있는 document 라는 object를 전달해 준다.

Document

document 인터페이스는 브라우저가 불러온 웹 페이지를 나타내며, 페이지 컨텐츠(DOM트리)의 진입점 역할을 수행.

console.dir(document)

DOCUMENT

첫번째 document의 항목을 가져오기

getElementById

-ID를 사용하여 HTML의 Element 가져오기

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Document</title>
  </head>
  <body>
    <h1 id="title">Grab Me!!</h1>
    <script src="0314.js"></script>
  </body>
</html>
```

```
document.getElementById("title");
--------------------------------------------------------------------------------
const title = document.getElementById("title");

console.log("title");
--------------------------------------------------------------------------------
실행 시 : <h1 id="title">Grab Me!!</h1> // h1태그를 보여줌
```

getElementsByClassName

class를 사용하여 Element 가져오기

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Document</title>
  </head>
  <body>
    <h1 class="hellos">Grab Me!!</h1>
    <h1 class="hellos">Grab Me!!</h1>
    <h1 class="hellos">Grab Me!!</h1>
    <h1 class="hellos">Grab Me!!</h1>
    <script src="0314.js"></script>
  </body>
</html>
```

```
const hellos = document.getElementByClassName("hello");

console.log("hellos");
--------------------------------------------------------------------------------
실행 시 :
HTMLCollection(4) [h1.hellos, h1.hellos, h1.hellos, h1.hellos]
0: h1.hellos
1: h1.hellos
2: h1.hellos
3: h1.hellos
length: 4

array 형태로 각 h1의 elements를 보여줌
objects형태가 아님
```

getElementsByTagName

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Document</title>
  </head>
  <body>
    <div>
      <h1>Grab Me!!</h1>
    </div>
    <div>
      <h1>Grab Me!!</h1>
    </div>
    <div>
      <h1>Grab Me!!</h1>
    </div>
    <script src="0314.js"></script>
  </body>
</html>
```

```
const title = document.getElementsByTagName("h1");

console.log("title");
--------------------------------------------------------------------------------
실행 시 :
HTMLCollection(4) [h1.hellos, h1.hellos, h1.hellos, h1.hellos]
0: h1.hellos
1: h1.hellos
```

```
2: h1.hellos
모든 h1 tag 를 array형태로 가져옴
objects 형태가 아님
```

querySelector(All)

element를 css selector로 가져올 수 있다.

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Document</title>
  </head>
  <body>
    <div class="hello">
      <h1>Grab Me!!</h1>
    </div>
    <script src="0314.js"></script>
  </body>
</html>
```

```
const hellos = document.querySelector(".hello h1");

console.log(hellos);
--------------------------------------------------------------------------------
실행 시 :
<h1>Grab Me!!</h1>

단 하나만의 element를 return함
.hello h1 이 여러개 일 경우
Returns the first element that is a descendant of node that matches selectors.
여러개를 불러오고 싶을 경우
querySelectorAll(".hello h1") 사용
CSS SELECTOR 를 사용하는 방식으로 가져옴
CLASS > .
ID    > #

getElementsByClassName() 에서 ( )안에는  . 을 안쓰는 이유
>> function 자체가 class를 가져오는 것을 명시했기 때문
```

getElementById(”hello”)

querySelector(”#hello”)

이 두개의 결과는 동일 but! getElementById 는 querySelector(”#hello h1”) 처럼 하위 요소를 선택 불가

Events

Event를 Listen하기 위해서는 HTML Element를 가져와서

addEventListener function을 실행시켜 준다.

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Document</title>
  </head>
  <body>
    <div class="hello">
      <h1>Grab Me!!</h1>
    </div>
    <script src="0314.js"></script>
  </body>
</html>
```

```
const hellos = document.querySelector(".hello h1");

function handleTitleClick() {
  console.log("title was clicked");
}

hellos.addEventListener("click", handleTitleClick);
--------------------------------------------------------------------------------
실행 시 :
Grab Me !! 라는 문구를 클릭 시 console창에 title was clicked

주의!! addEventListener에 사용된 매개변수 handleTitleClick 함수는 ()를 붙이지 않는다.
이유 event가 일어난 뒤 js function을 동작하게 하기 위해서

첫번째 매개변수는 어떤 event를 liste하고 싶은지 명시하고,
두번째 매개변수에는 event를 listen했을 때, 어떤 함수를 실행할지 정한다.
```

console.dir 을 사용하여 element를 확인!!

“on”이 붙은 objects는 event listener이다.

eventlistener 예제)

```
const hellos = document.querySelector(".hello h1");

function handleTitleClick() {
  hellos.style.color = "blue";
}

function handleMouseEnter() {
  hellos.innerText = "mouse is here!!";  //마우스가 위에 있을 때
}

function handleMouseLeave() {
  hellos.innerText = "mouse is gone!!";  //마우스가 떠났을 때
}

hellos.addEventListener("click", handleTitleClick);
//== hellos.onclick = handleTitleClick;
hellos.addEventListener("mouseenter", handleMouseEnter);
hellos.addEventListener("mouseleave", handleMouseLeave);

.addEventListener 를 선호하는 이유는
.removeEventListener 를 이용하여 eventListener를 지우기 때문
```

window

```
function handleWindowResize() {
  document.body.style.backgroundColor = "tomato";
}
function handleWindowCopy() {
  alert("copier!!");
}

window.addEventListener("resize", handleWindowResize);
window.addEventListener("copy", handleWindowCopy);
-------------------------------------------------------------------------------

document.body 중요!! document.div 불가!!
title이나 head같은 중요요소는 접근 가능
div나 h1같은 요소는 querySelector를 이용하여 접근
```

```
const h1 = document.querySelector(".hello h1");
```

```
function handleTitleClick() {
  const currentColor = h1.style.color;
  let newColor;
  if (currentColor === "blue") {
    //h1.style.color === "blue"
    newColor = "tomato";
  } else {
    newColor = "blue";
  }
  h1.style.color = newColor;
}
h1.addEventListener("click", handleTitleClick);
--------------------------------------------------------------------------------
```

CSS를 활용한 JavaScript

```
body {
  background-color: beige;
}

h1 {
  color: blue;

}

.active {
 color: tomato;
}
```

```
const h1 = document.querySelector(".hello h1");

function handleTitleClick() {
  if (h1.className === "active") {
    h1.className = "";
  } else {
    h1.className = "active";
  }
}

h1.addEventListener("click", handleTitleClick);
```

--

```
const h1 = document.querySelector(".hello h1");

function handleTitleClick() {
  const activeClass = "active";
  if (h1.className === activeClass) {
    h1.className = "";
  } else {
    h1.className = activeClass;
  }
}

h1.addEventListener("click", handleTitleClick);
```

—

classList사용

```
const h1 = document.querySelector(".hello h1");

function handleTitleClick() {
  const activeClass = "active";
  if (h1.classList.containss(activeClass)) {
    h1.className = "";
  } else {
    h1.className = activeClass;
  }
}

h1.addEventListener("click", handleTitleClick);
```

.remove 사용

```
const h1 = document.querySelector(".hello h1");

function handleTitleClick() {
  const activeClass = "active";
  if (h1.classList.containss(activeClass)) {
    h1.classList.remove(activeClass);
  } else {
    h1.classList.add(activeClass);
  }
}

h1.addEventListener("click", handleTitleClick);
```

toggle 사용

```
const h1 = document.querySelector(".hello h1");

function handleTitleClick() {
  h1.classList.toggle("active");
}
h1.addEventListener("click", handleTitleClick);

--------------------------------------------------------------------------

toggle은 h1의 classList에 active class가 이미 있는지 확인해서 있다면
제거해주고, 없다면 추가해준다.
```

#2 Login

objects와 property

input 안 value 속성

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Document</title>
  </head>
  <body>
    <div id="loginForm">
      <input type="text" placeholder="What is your name?" />
      <button>Log In</button>
    </div>
  </body>
</html>
```

```
const loginForm = document.getElementById("loginForm");
const loginInput = loginForm.querySelector("input");
const loginButton = loginForm.querySelector("button");
--------------------------------------------------------------------------
짧은 코드로 변경
const loginInput = document.querySelector("#loginForm input");
const loginButton = document.querySelector("#loginForm button");
--------------------------------------------------------------------------

document.getElementById()
loginForm.querySelector()  차이점

>> HTML element안을 직접적으로 검색해서 좀 더 정밀 조사가 가능!
```

```
const loginInput = document.querySelector("#loginForm input");
const loginButton = document.querySelector("#loginForm button");

function btnLoginClick() {
  console.log(loginInput.value);
}

loginButton.addEventListener("click", btnLoginClick);
--------------------------------------------------------------------------

login button을 클릭했을 시 input에 입력된 값 출력 event처리
```

validation 유효성 검사

```
const loginInput = document.querySelector("#loginForm input");
const loginButton = document.querySelector("#loginForm button");

function btnLoginClick() {
  const userName = loginInput.value;
  if (userName === "") {
    alert("Please write your name!!");
  } else if (userName.length > 15) {
    alert("Your name is too long!!");
  }
}

loginButton.addEventListener("click", btnLoginClick);
--------------------------------------------------------------------------
input 입력란이 공백일 경우와 text가 너무 길 경우 처리
```

HTML 에서 제공하는 input의 자체 기능 사용

required, maxlength

input의 유효성 검사를 하기 위해서는 input이 form tag안에 있어야 한다.

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Document</title>
  </head>
  <body>
    <form id="loginForm">
      <input required maxlength="15" type="text" placeholder="What is your name?" />
      <button>Log In</button>
    </form>
  </body>
</html>
```

form은 웹사이트를 재시작 시킴 why? form이 submit되기 때문

form안에서 enter를 누르고, input이 더이상 존재하지 않으면 submit

form안의 button을 누르면 submit

>>click에 신경 쓰지 않아도 됨. form의 submit을 처리

브라우저가 새로고침 하지않고, username을 저장

!!EventListenr의 function

submit의 기본 동작 막기

```
const loginForm = document.querySelector("#loginForm");
const loginInput = document.querySelector("#loginForm input");

function onLoginSubmit(event) {
  event.preventDefault();
  console.log(event);
```

```
}

loginForm.addEventListener("submit", onLoginSubmit);
--------------------------------------------------------------------------
실행 결과:
onLoginSubmit function은 argument로 발생한 일에 대해서 필요로 할만한 정보들을 준다.
모든 EventListener function의 첫번째 argument는 항상 막 벌어진 일들에 대한 정보가 된다.
argument공간만 제공하면 js가 방금 일어난 event에 대한 정보를 지닌 argument를 채워넣는다.
SubmitEvent {isTrusted: true, submitter: button, type: 'submit', target: form#loginForm,
 currentTarget: form#loginForm, …}
```

link의 기본 동작 막기

```
const loginForm = document.querySelector("#loginForm");
const loginInput = document.querySelector("#loginForm input");
const link = document.querySelector("a");

function onLoginSubmit(event) {
  event.preventDefault();
  console.log(loginInput.value);
}

function handleClickLink(event) {
  event.preventDefault();
  console.log(event);
  alert("clicked!!");
}

loginForm.addEventListener("submit", onLoginSubmit);
link.addEventListener("click");
--------------------------------------------------------------------------
실행 결과:
onLoginSubmit function은 argument로 발생한 일에 대해서 필요로 할만한 정보들을 준다.
모든 EventListener function의 첫번째 argument는 항상 막 벌어진 일들에 대한 정보가 된다.
argument공간만 제공하면 js가 방금 일어난 event에 대한 정보를 지닌 argument를 채워넣는다.
SubmitEvent {isTrusted: true, submitter: button, type: 'submit', target: form#loginForm,
 currentTarget: form#loginForm, …}
```

user가 입력 시 form tag를 display: none

```
const loginForm = document.querySelector("#loginForm");
const loginInput = document.querySelector("#loginForm input");
const link = document.querySelector("a");

function onLoginSubmit(event) {
  const userName = loginInput.value;
```

```
  event.preventDefault();
  loginForm.classList.add("hidden");
  console.log(loginInput.value);
}
loginForm.addEventListener("submit", onLoginSubmit);
--------------------------------------------------------------------------
style.css

.hidden {
  display: none;
}
--------------------------------------------------------------------------
실행 결과:
onLoginSubmit function은 argument로 발생한 일에 대해서 필요로 할만한 정보들을 준다.
모든 EventListener function의 첫번째 argument는 항상 막 벌어진 일들에 대한 정보가 된다.
argument공간만 제공하면 js가 방금 일어난 event에 대한 정보를 지닌 argument를 채워넣는다.
SubmitEvent {isTrusted: true, submitter: button, type: 'submit', target: form#loginForm,
 currentTarget: form#loginForm, …}
```

user가 입력 시 form tag를 display: none 후 h1 tag 를 이용하여 user가 입력한 값 보여주기

```
const loginForm = document.querySelector("#loginForm");
const loginInput = document.querySelector("#loginForm input");
const link = document.querySelector("a");
const greeting = document.querySelector("#greeting");
const HIDDEN_CLASSNAME = "hidden";

function onLoginSubmit(event) {
  event.preventDefault();
  loginForm.classList.add(HIDDEN_CLASSNAME);
  const userName = loginInput.value;
  greeting.innerText = `Hello ${userName}`;
  greeting.classList.remove(HIDDEN_CLASSNAME);
}
loginForm.addEventListener("submit", onLoginSubmit);
--------------------------------------------------------------------------
style.css

.hidden {
  display: none;
}
```

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
```

```
    <title>Document</title>
  </head>
  <body>
    <form id="loginForm">
      <input
        required
        maxlength="15"
        type="text"
        placeholder="What is your name?"
      />
      <button>Log In</button>
    </form>
    <h1 id="greeting" class="hidden"></h1>
  </body>
</html>
```

Web Storage

Web Storage API는 브라우저에서 쿠키를 사용하는 것보다 훨씬 직관적으로 key/value 데이터를 안전하게 저장할 수 있는 메커니즘을 제공.

1. ***Window.localStorage***

:우리가 브라우저에 뭔가를 저장할 수 있게 해준다.

Storage 객체는 단순한 key-value 저장소이며, 이는 객체와 비슷. 하지만 이 데이터들은 페이지 로딩에도 온전하게 유지된다.

key와 그 value는 항상 문자열이다. (만약 정수로 키를 사용할 경우 이는 자동으로 string으로 변경)

localStorage.setItem('myCat', 'Tom'); storage 내 값을 저장

Key 값과 그에 대응하는 Value 형태로 저장

localStorage.getItem("myCat"); storage 내 값을 가져오기

localStorage.removeItem("myCat"); storage 내 값을 삭제

localStorage.clear() : storage 내 전체 제거

2. ***Window.sessionStorage***

- 페이지 세션은 브라우저가 열려있는 한 새로고침과 페이지 복구를 거쳐도 남아있습니다.
- **페이지를 새로운 탭이나 창에서 열면, 세션 쿠키의 동작과는 다르게 최상위 브라우징 맥락의 값을 가진 새로운 세션을 생성합니다.**
- 같은 URL을 다수의 탭/창에서 열면 각각의 탭/창에 대해 새로운 `sessionStorage`를 생성합니다.
- 탭/창을 닫으면 세션이 끝나고 `sessionStorage` 안의 객체를 초기화합니다.

sessionStorage 도 만료기간 존재하는지 알아보기!!

sessionStoage.setItem('myCat', 'Tom');

localStorage vs sessionStorage

읽기 전용 속성을 사용하면 `Document`의 `Storage` 객체에 접근할 수 있습니다. 저장한 데이터는 브라우저 세션 간에 공유됩니다. `localStorage`는 `sessionStorage`와 비슷하지만, `localStorage`의 데이터는 만료되지 않고 `sessionStorage`의 데이터는 페이지 세션이 끝날 때, 즉 페이지를 닫을 때 사라지는 점이 다르다.

이 두 개의 매커니즘의 차이점은 데이터가 어떤 범위 내에서 얼마나 오래 보존되느냐에 있다.

세션 스토리지는 웹페이지의 세션이 끝날 때 저장된 데이터가 지워지는 반면에, 로컬 스토리지는 웹페이지의 세션이 끝나더라도 데이터가 지워지지 않는다.

세션 스토리지는 브라우저에서 같은 웹사이트를 여러 탭이나 창에 띄우면, 여러 개의 세션 스토리지에 데이터가 서로 격리되어 저장되며, 각 탭이나 창이 닫힐 때 저장해 둔 데이터도 함께 소멸한다.

로컬 스토리지의 경우 여러 탭이나 창 간에 데이터가 서로 공유되며 탭이나 창을 닫아도 데이터는 브라우저에 그대로 남아 있습니다.

cookie

```
const loginForm = document.querySelector("#loginForm");
const loginInput = document.querySelector("#loginForm input");
const greeting = document.querySelector("#greeting");
const HIDDEN_CLASSNAME = "hidden"; //본인이 생성한 string을 반복 사용시 변수로 사용

function onLoginSubmit(event) {
  event.preventDefault();
  loginForm.classList.add(HIDDEN_CLASSNAME);
  const userName = loginInput.value;
  localStorage.setItem("username", userName);
  greeting.innerText = `Hello ${userName}`;
  greeting.classList.remove(HIDDEN_CLASSNAME);
}

loginForm.addEventListener("submit", onLoginSubmit);
--------------------------------------------------------------------------
style.css

.hidden {
  display: none;
}
```

localStorge 안의 값이 있을 경우 form tag를 숨기고 h1 tag를 보여준다.

localStorge 안의 값이 없을 경우 form tag를 보여주고 h1 tag를 숨긴다.

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Document</title>
  </head>
  <body>
    <form id="loginForm" class="hidden">
      <input
        required
        maxlength="15"
        type="text"
        placeholder="What is your name?"
      />
      <button>Log In</button>
    </form>
    <h1 id="greeting" class="hidden"></h1>
  </body>
</html>
```

```
const loginForm = document.querySelector("#loginForm");
const loginInput = document.querySelector("#loginForm input");
const greeting = document.querySelector("#greeting");
const HIDDEN_CLASSNAME = "hidden";
const USERNAME_KEY = "username";

function onLoginSubmit(event) {
  event.preventDefault(); // 브라우저의 기본 동작을 막음
  loginForm.classList.add(HIDDEN_CLASSNAME); //form tag에 클래스 추가
  const userName = loginInput.value;
  localStorage.setItem(USERNAME_KEY, userName);
  paintGreetings(userName);
}

function paintGreetings(userName) {
  greeting.innerText = `Hello ${userName}`; //백틱 `` 사용
  greeting.classList.remove(HIDDEN_CLASSNAME);
}
const savedUserName = localStorage.getItem(USERNAME_KEY);

if (savedUserName === null) {
  loginForm.classList.remove(HIDDEN_CLASSNAME);
  loginForm.addEventListener("submit", onLoginSubmit);
} else {
  paintGreetings(savedUserName);
}
--------------------------------------------------------------------------------
style.css

.hidden {
  display: none;
}
```

Intervals and Timeout

setIntervals(function, time)

setTimeout(function, time)

```
const clock = document.querySelector("#clock");
```

```
function getClock() {
  const date = new Date(); //js에 내장되어있는 Date objects 사용
  clock.innerText = (`${date.getHour()}:${date.getMinutes()}:${date.getSeconds()}`);
}

getClock(); // 브라우저를 새로고침했을 경우 바로 실행
setInterval(getClock, 1000);
--------------------------------------------------------------------------
매 초마다 date object를 생성
```

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Document</title>
    <link rel="stylesheet" href="css/style.css" />
  </head>
  <body>
    <h2 id="clock">00:00:00</h2>
    <script src="js/clock.js"></script>
  </body>
</html>
```