Javascript

변수와 데이터 타입

- 전역변수
  - 변수의 선언과 할당
    - var 변수이름;
    - 변수이름 = 값;
    - var 변수이름 = 값;
    - 선언과 할당의 종합
  - 변수값 변경하기
    - var num = 500;
    - num =200;
  - 변수 이름 규칙
    - 영어, 숫자, 언더바(_)만 사용가능
    - 첫 글자는 숫자로 시작 불가능
    - 수 개 이상의 단어를 결합한 경우
      - 스네이크 표기법
        - 파이소기 파이써 언어에서 사용
      - 카멜 표기법
        - 파이소기 파이 첫 글자 대문자로 변경 ex) homeWork
- 지역변수
  - 변수의 선언과 할당
    - let 변수이름;
    - 선언
    - 변수이름 = 값;
    - 할당
  - 선언과 할당의 종합
    - let 변수이름 = 값;
  - 중복 선언 금지
    - let 키워드는 중복 선언 불가능
- 상수
  - const 변수이름 = 값;
  - 선언과 동시에 값이 할당 되어야 하며, 최초 값 할당 후 값 변경 불가능
  - number
    - 정수와 실수를 포함한 모든 숫자
  - string
    - 문자열, "", '', `` 감싼 모든 형식 데이터
  - boolean
    - 논리형 true(참) false(거짓)
  - object
    - 객체, 함수(function), 배열(Array), 날짜(date), 정규식(RegExp)등
  - null
    - object형의 한 종류. 나중에 할당하기 위해 비워진 값
  - undefined
    - 선언만 해고 할당되지 않은 상태

연산자

- 산술 연산자
  - 산술 연산자의 종류
    - + 덧셈
    - - 뺄셈
    - * 곱셈
    - / 나눗셈
    - % 나눗셈에서 정수 부분의 몫을 제외한 나머지
  - 연산자의 활용
    - console.log(문장형을 통해 변수간의 연산 결과 출력)
    - 연산 결과를 새로운 변수에 할당
- 대입 연산자
  - 대입 연산자
    - 피연산 우변을 대입하는 = 연산자
    - 연산 결과를 변수 스스로에게 넣어 줄수가
    - 어떠 생성된 변수의 값을 다른 값으로 변경
  - 단일 연산자
    - 어떤 변수에 대한 연산 결과를 다시 변수에 넣어 저장는 연산
    - ++, --, **, /=, %=
  - 증감 연산자
    - 연산자가 나오기 전 정수 만큼이 1씩 증가 및 감소
    - 전에 수식 처리 후, 증감,감소 연산자 수행
    - 후위 증가 및 감소
    - a++ 1, a++ a=a_a
    - ++a 가능
    - 변수 스스로 값에 대해 증가,감소 후 전체 수식 연산
- 비교 연산자
  - 논리값(true 혹은 false)로 결정
    - 3>5, 3<5,4<5
  - 값자리(==, ===)가 자료자리(==, !==)
    - ==, != 는 내용만 비교, ===,!== 는 해당하 타입까지 비교
- 논리 연산자
  - AND (&&)
    - 모든 값이 true인 경우만, 결과가 true 그 외는 false
  - OR (||)
    - 연산값 중 하나라도 true면 결과가 true, 모든 결과가 false일 경우 false
  - NOT (!)
    - 논리형 값 true,false 값을 반대로 전환
- 삼항 연산자
  - let 변수 = 조건식 ? 값1 : 값2;
  - 참일 경우 값1, 거짓인 경우 값2가 변수에 대입
- 타입 표기
  - 캐릭 표기
  - 캐릭 표기

조건문과 조건 연산자

- 조건문
  - if
    - 조건식이 참인 경우 {} 블록 안 실행
    - if (조건식) { } else if (조건식) { } else { }
    - 조건식이 거짓일 경우 if문 다음 실행
    - 여러가지 경우의 수를 나열하는 조건문
  - switch(변수) { case 1: 실행할 명령 break; }
    - switch은 조건식이 참일 때까지 실행
    - 조건에 맞는 case블록부터 break 키워드를 만날 때까지 실행
- 반복문
  - while문
    - 초기식; while(조건식) { 반복할 구문_증감식++;--; }
    - ex) let i=0; while(i<10) { ... i++; }
  - for문
    - for (초기식; 조건식; 증감식;) { 반복할 구문_; }
    - ex) for(let i=0; i<10; i++) {}

함수

- 함수의 표기
  - 함수 정의
    - function 함수이름() { ...명령어... }
  - 함수 호출
    - 함수이름();
  - 함수 이름규칙
    - 영어, 숫자 언어바이 가능 조합을 조합 대문자 "낙타표기"방법표기방법을 사용
- 함수실행을 위한 조건문
  - 매개변수란? 함수 실행에 필요한 조건값 정의
    - 매개변수 정의
      - function 함수이름(매개변수1) { ...명령어... }
    - 매개변수에 전달되는 함수 호출
      - 함수이름(값);
    - 다중매개변수 정의
      - function 함수이름(매개변수1, 매개변수2...,매개변수n){ ...매개변수를 위한 프로그램 구문... }
    - 다중매개변수 함수호출
      - 함수이름(값1, 값2, ...값n);
    - 함수호출시 매개변수 생략시, undefined로 식별
    - 매개변수의 기본값 정의
      - function 함수이름(매개변수1=기본값1, 매개변수2=기본값2...){ ..... }
- 리턴
  - 함수 정의
    - function 함수이름(매개변수1, 매개변수2...){ ...매개변수를 활용한 프로그램 구문... return 출력할 값; }
  - 리턴값을 다른 변수에 할당
    - const 변수 = 함수이름(매개변수1, 매개변수2,...);
  - 조건식 활용
    - if(함수이름(매개변수1, 매개변수2... 매개변수n> 0) { ...실행 될 값... }
  - 논리값 리턴하는 함수 조건식
    - if (함수이름(매개변수1, 매개변수2,... 매개변수n)) { ...함수의 리턴값이 true인 경우 실행 }
    - if (!함수이름(매개변수1, 매개변수2,... 매개변수n)) { ...함수의 리턴값이 false인 경우 실행 }
  - 리턴값 활용한 반복문
    - for (let i=0; i<함수이름(매개변수1, 매개변수2... 매개변수n); i++){ ...반복될 구문... }
- 함수의 또 다른 정체
  - 함수에 변수 대입
    - 함수 자체가 객체 형태이기에 다른 변수에 참조 가능
    - function 함수이름(){ } const 변수 = 함수이름;
  - 익명함수
    - 다른 변수에 참조시킬 목적으로 함수 정의, 정의 형태가 대입문이므로 인하여 끄 뒤에세미콜
    - const 변수 = function (...){ ..... };
  - 화살표 함수
    - 기존 익명함수 문법을 축약
      - 매개변수 여러 개인 함수
        - const 변수 = (매개변수1, 매개변수2,... 매개변수n)=> { ...처리로직... }
      - 매개변수가 한개인 함수 소 생략 가능
        - const 변수 = 매개변수 => { ...처리로직... };
      - 로직이 한 줄인 경우
        - const 변수 =(매개변수), 매개변수2,... 매개변수n)=>리턴값;
  - 콜백함수
    - 직접 함수 정의
      - ex) function 함수이름(매개변수1, 매개변수2){return 매개변수1 +매개변수2;} 콜백함수이름(2,1, 값2, phal);
    - 익명함수 형태 전달
      - 콜백함수이름(값1, 값2, function(매개변수1, 매개변수2){ return 임명형 값; });
    - 화살표함수 형태
      - 콜백함수이름(값1,값2)=>{매개변수1,매개변수2; => { for(let i = 매개변수1; i<매개변수2; i++){ console.log(출력 값); } };
    - 함수 로직이 한 줄인 경우 축약된 형태
      - 콜백함수이름(값1, 값2,)=>console.log(출력값);
    - 최적은 학생퀴지만, 과정이 불충정한 경우 ex)일과제
  - 재귀함수
    - 함수가 자기로직 내부에서 자기 자신을 호출
      - f(x)= x* f(x-1), 단 f(1)=1
    - 피보나치수열
      - f(x)= f(x-1) + f(x-2)

프로토타입

- 개념
  - 메서드 변수 안에 메서 속성을 저장할 수 있는 데이터 타입
- 생성 방법
  - 생성자 함수정의
    - const 변수 = new 함수();
    - 이후 함수 앞에 new를 붙이면 생성자가 됨
    - 화살표 함수는 생서자 형식으로 사용할 수 없음
    - 생성자는 new 키워드에 반응함
  - 생성자 형식변수 조합
    - this.형식변수이름 = 값;
    - 메서드이름 형식변수에 등록
      - const 생성자이름 = function(매개변수1, 매개변수2){ this.형식변수1; this.메서드이름; };
- 메서드
  - prototype 속성
    - 정의형 생성자의 기능 확장
    - 생성자명.prototype.메서드이름 = function(매개변수 1, 2){
- getter, setter
  - 형식변수에 값을 간접적으로 대입하는 setter, 형식변수의 값을 리턴받는 getter
  - Object.defineProperty(생성자.prototype, 함수이름, {getter,setter:정의})
  - getter,setter는 같은 함수 이름을 공유
- JSON 활용
  - 생서자이름.prototype={...}

클래스

- 기본 코드
  - 클래스 이름 (첫 글자는 대문자로 지정)
    - class 클래스이름{ 생성자 ,getter, setter ,메서드... }
  - const 변수이름 = new 클래스이름();
  - 객체 생성
    - 객체.형식변수 = 값; 객체.메서드();
- 인스턴스 정의한 클래스
  - 생성자 함수 안에서 this키워드를 통해 객체 안에 변수 값들을 생성
  - 생성자 함수 constructor로 자동이 정해져 있음
- 메서드만 정의한 클래스
  - 중도나 목적이 같은 메서드들을 별도의 클래스로 묶어줌기
- 메서드와 형식변수를 갖는 클래스
  - 형식변수는 모든 메서드가 공유하는 전역변수 개념
  - 메인이 this.을 통해서만 접근이 가능
  - 형식변수에 값을 간접적으로 대입하는 setter, 형식변수의 값을 리턴받는 getter
- getter,setter
  - 형식변수 직접 접근을 제한하기 위해 형식변수의 이름은 언더바(_)로 시작함 ex) this._형식변수이름
- 상속
  - 기존 기능 확장 클래스
    - 부모 기능 클래스에서 기능을 상속받아 추가로 자신이 구현하는 기능
    - class 부모상속이름 extends 자식클래스이름
  - 상속 공통기능 추출
    - 여러 개의 클래스가 공통되는 부분을 독립적인 클래스로 추출하여, 그 클래스를 상속하여 공유
- 메서드 오버라이드
  - 부모 클래스의 기능을 자식 클래스가 재정의
  - 자식이 정의한 메서드에 의해 부모 메서드는 가려짐
- super 키워드
  - 부모 클래스를 메서드 overridem하고 있는 자식 클래스 안에서 부모의 함수 기능을 호출
- 정적 형식 변수
  - 클래스에 속한 변수나 함수에 static키워드를 접목하면 모든 객체가 공유할 수 있게됨
  - static 변수이름[함수이름]

모듈화 이해

- 직당한 함수를 모듈로 묶목
- module.exports
  - exports의 역할 속성으로 변수,JSON,함수 추가
  - exports :속성으로 객체 추가
  - 클래스에 자체에 모듈 추가
  - 클래스에 대한 개체를 모듈에 추가
- require()
  - module.exports를 통해 등록한 기능 리턴
  - require("./파일경로이름");

비동기처리

- callback hell
  - Javascript 비동기는 콜백함수로 실행됨
  - 콜백함수는 싸여 겨기 한꺼번에 처맞게 수 있음
- simple promise
  - 비동기 처리를 좀 깔끔하게 정돈하는 목적과 실제 콜백의 싸여 표지를 분리하기 위해 쓰는 개념
  - Promise를 리턴하는 함수를 호출하기 위함
- async, await
  - 속시 실행 함수 형태로 정의
  - 일반함수는 언제든지 호출 그 뒤에 호출을 위한 {}를 연달아 넣음

예외처리

- try{}catch{}finally
  - try{...}
    - try{}안의 코드에 에러가 일어나면 바로잡 실행되어 실행
    - 실행 안 코드는 무시되어 에러발
  - catch(err){...}
    - 에러가 있다면, 중단하고 catch(err)로 넘어가서 에러정보 내포내기
  - finally{...}
    - 에러 여부와 상관없이 무조건 실행됨, 생략가능
- 에러 강제
  - throw
    - throw는 그 자체를 에러로 인식
- 에러 강제로 생성 가능 에러
  - 관례적 throw new Error()로 강제 에러 만들기
  - 함수 안에서 error가 생기면 호출 값에 예외를 돌림
  - 에러 정상이 끝난 후 변수 값을 초기한 변수의 선언이 try 블록보다 상위에 있어야함
- 비동기 에러
  - error의 이중을 비동기로 문제의 원인을 알 수 있음
  - try-catch는 동기 방식으로 비동기 방식인 timer와 ajax에는 이중을 못함
  - timer처리의 같은 비동식의 예외처리는 콜백함수 내부에서 처리해야함

Javascript 내장기능

- alert
  - 숫자가 아니면 true, 숫자이거나 변환 가능한 형식은 false
  - console.log(isNaN(처리하고));
- parseFloat
  - 주어진 값에서 변환한 실수(소수점)을 리턴
  - console.log(parseFloat(값));
- parseInt
  - console.log(parseInt(10진 정수값으로 변환, 뒷 진법인지));
- encodeURI
  - 주어진 문자열을 URL에 표현시키기에 적정한 형태로 인코딩(특수문자 제외)
  - const 변수 = encodeURI(상위변수);
- encodeURIComponent
  - 모든 문자열을 URL에 표현시키기 적정한 형태로 인코딩
  - const 변수 = encodeURIComponent(상위변수);
- setTimeout
  - 지정된 함수를 두 번째 인자로 전달된 시간 후에 실행하도록 예약 (일회이기능)
  - setTimeout(() => { ..... }, 시간(1/1000초));
- setInterval
  - 지정된 함수를 두 번째 인자로 전달된 시간마다 한 번씩 호출(라이어기능)
  - setInterval(() => { ..... }, 시간(1/1000초));
- 문자열
  - 싸여있는 인덱싱 위치의 문자를 리턴
    - const 변수이름 = 상위변수이름.charAt();
  - 싸여있는 전달된 내용일 처음 나타나는 위치를 리턴
    - const 변수이름 = 상위변수이름.indexOf("");
  - 싸여있는 전달된 문자가 마지막으로 나타나는 위치를 리턴
    - const 변수이름 = 상위변수이름.lastIndexOf("");
  - 지정된 시작 위치의 문 위치 자르기
    - const 변수이름 = 상위변수이름.substring(시작위치, 종료위치);
  - 모든 문자 대문자로 변환
    - const 변수이름 = 상위변수이름.toUpperCase();
  - 모든 문자 소문자로 변환
    - const 변수이름 = 상위변수이름.toLowerCase();
  - 문자열 앞, 뒤 공백 지우기
    - const 변수이름 = 상위변수이름.trim();
  - 문자열 안에 포함된 구문자를 기준으로 문자열을 잘라 배열로 변환
    - const 변수이름 = 상위변수이름.split("구분자");
  - 첫 번째 싸여있는 내용을 두 번째 싸여있는로 변경한 결과 반환
    - const 변수이름 = 상위변수이름.replace(싸여있어, 싸여있어2);
  - 첫 번째 싸여있는 일치하는 내용일 을 이상일 경우 첫 번째 항목 반환
    - console.log(변수이름.replaceAll(싸여있어, 싸여있어2));
- Math
  - 최대값,최소값
    - var 변수 = Math.max([값, 값,]);
    - var 변수 = Math.min([값, 값,]);
  - 소수점
    - 절올림
      - console.log("소수점 절올림:" + Math.ceil(변수이름));
    - 절삭
      - console.log("소수점 절삭:" + Math.floor(변수이름));
  - 절대값
    - console.log("절대값:" +Math.abs(변수이름));
  - 난수
    - 랜덤 난수 발생
      - console.log("난수:" + Math.random());
    - 두 수 사이의 난수 지정
      - function random(값, 값){ return parseInt(Math.random() *(값2-값1 +1)) + 값1;
- Date
  - 날짜 추론만 나타나는 문자열 반환
    - console.log(date1.toDateString());
  - Date을 나타내는 문자열 ISO 형식
    - console.log(date1.toISOString());
  - 형식 문자열을 사용해 Date 문자열 생성
    - console.log(date1.toLocaleString());
  - 현재 지역형식으로 날짜부분 문자열 생성
    - console.log(date1.toLocaleDateString());
  - 현재 지역형식으로 시간부분 문자열 생성
    - console.log(date1.toLocaleTimeString());
- Number
  - 정개수성
    - console.log(Number.MAX_VALUE);
    - console.log(Number.MIN_VALUE);
  - 정수인지확인
    - console.log(Number.isInteger([값]));
- Array
  - 배열의 원소 추가, 삭제, 변경
    - 맨 앞에 추가
      - data.unshift(0,10);
    - 맨 뒤에 추가
      - const 변수이름 =data.push();
    - 위치와 갯수에 맞게 삭제
      - const 변수이름 = data.splice(추출위치,갯수);
    - 맨 앞에 요소 삭제
      - const 변수이름 = data.shift();
    - 맨 뒤에 요소 삭제
      - const 변수이름 =data.pop();
  - 배열검색
    - 인덱스 from부터 시작해 item(요소)을 찾는다
      - 변수.indexOf(item, from)
    - 인덱스 from부터 시작해 item이 있는지를 검색
      - 변수.includes(item, from
  - 배열검사
    - 전체일치
      - const 변수 = 상위변수이름.filter
    - 배열일치
      - const 변수 = 상위변수이름.find
  - 배열 정렬
    - 변수.sort();
  - 역순배치
    - 변수.reverse();
- Regex
  - 문자열이 특정 조건을 충족하는지 검사하거나 특정 패턴 문자열 검색
    - const 변수명 = /정규표현식/ 변수명.test([검사할 문자열])

Javascript

- 기본 문법 활용
  - 변수의 호출방 명위 변수의 스코프
    - var 키워드
      - 블록은 인덱서 선언된 변수는 블록 밖에서 사용됨
      - 블록인지 상관없이 접근 경우, undefined
    - let 키워드, const 키워드
      - 블록인 밖에서 선언된 변수는 건드로 접후 가능함 싸여있 {} 인덱서 선언된 변수는 블록을 싸여나올 수 없음
    - for문의 초기식 변수의 명위
      - var 키워드 선 할당 후 선언이 가능 싸여진, let, const 키워드는 불가능
  - 조건
    - 특정 조건이 참인 경우, 블록 인덱서 상태조건을 만족
      - if-if
      - if(조건식) { if(조건식) { } else if (조건식) { } else { } }
    - 특정 조건이 참인 경우, 블록 인덱서 반복문 수행
      - if-for
      - if(조건식) { for (초기식; 조건식; 증감식) { 반복할 내용 } }
  - 다중 인덱 포함된 조건식
    - 반복문 조건값과가에 생성된 변수이에 대한 조건 만들
      - for-if
    - 반복문 인덱서 반복 수행이 여다 조건을 만들
      - for-if
    - 이중 반복문으로 배열 반복문 1회 수행할 때 여러, 인덱 반복문은 처음부터 새로 시작
      - for-for
    - 인덱 반복문의 조건식이 배열 반복문의 조건변수를 활용, 인덱 반복문 반복 횟수에 변화를 줌
      - for-for 인덱서 조건제어
    - continue: 증감식으로 이동, break: 현재 반복문 강제 종료
      - for-if
    - continue: 조건식으로으로 강제이동, break: 현재 반복문 강제 종료
      - while-if
      - 주한주도
- 배열
  - 배열의 선언과 할당
    - 선언과 할당이 따로따로 이뤄지는 경우, const 선언 불가
    - ex) let myArr = [1,2,3,4,5];
  - 배열의 이해
    - ex) let newArr = new Array([...])
    - Array클래스
      - new Array()로 배열 생성시, 인덱 숫자 값이 하나일 경우 숫자 값만 줄 뿐 인덱 갯값 -->가 인덱 undefined가 됨
    - 인덱스 0부터 시작하는 일련번호
      - ex) console.log(myArr[0])
    - 배열의 크기
      - 배열의 이름.length
      - 변수 공식: const 변수이름= 배열이름.length -1 =i;
    - for-of문
      - for (const 새로운변수 of 초기값변수이름)
      - 초기값을 새로운 변수에 지 않는 코드
- 객체
  - 기본 객체 정의
    - const 객체이름 = {key:value, key2:value2...};
    - 객체변수이름.속성변수이름 = ;
  - 중첩데이터구조
    - 모든 형태의 데이터 타입이 지정이 지정될 수 있음
    - const 객체이름 = {key:숫자, key2:문자열, ... key:[값1 값2...]};
  - JSON 구조
    - 단일형태의 JSON
      - const 객체이름 = {key:값, key2:값};
    - 배열구조
      - 기본 JSON 포함한 JSON
      - const 객체이름 = { key: 문자열, key: 값 };
    - 배열구조 JSON
      - const 객체이름=[key: { key:값1, key:값2 }, key:값];
  - JSON
    - 중첩구조
      - const 객체이름 ={key : [{key:value, key2:value2}, {key2:value, key4:value}] }
      - JSON의 Value가 배열요 형태, 각 배열의 원소가요 다른 JSON형식인 경우
    - JSON형태
      - 존재하지 않는 key를 사용하거나 다른 변수에 대입시, undefined 처리
      - 존재하지 않는 key에 값을 대입하면 객체 확장
    - JSON 변경은
      - 이후 key도 정의되지 않는 한 객체는 사용에 내용을 덧붙일 수 있음
      - for(let 변수이름 in 객체이름){...}
      - 선언된 변수에 객체에 있는 key값을 전부 다 저장
    - 깊은복사,얕은복사
      - 원본 수정시, 복사본도 수정됨
      - 참조복사(얕은복사)
      - Object.assign(복사할곳, 복사할곳)
      - 원본과 동일한 일대용 갖는 배열 생성, 원소까지 개별 복사
      - 원본 수정시, 복사본은 수정안됨
      - 깊은복사(깊은복사)
    - 구조분해
      - ex) const dummy ={ a1:100, a2: 200, a3: 300, a4: 400}; const {a1, a2, ...rest_o}= dummy;
      - 구조분해 수행 및 나머지 별도 분리 (...변수이름을 이용해 분리)
      - JSON 구조분해는에구조 맞춤
      - {b1, b2, ...rest_o} = {1, 2, 3, 4, 5, 6, 7, 8, 9};
      - 구조분해 수행 및 나머지 별도 분리
      - 배열 구조분해
- 모듈화 이해
  - 직당한 함수를 모듈로 묶목

Javascript

- 기본 문법 활용
  - 변수의 호출방 명위 변수의 스코프
  - 조건
  - 반복문
  - 다중 인덱 포함된 조건식
  - 배열
  - 객체
  - JSON
  - 비동기처리
  - 예외처리
  - 클래스
  - 모듈화 이해
  - Javascript 내장기능

Vanilla5

- HTML 제어하기
  - CSS 제어
    - CSS 속성 접근
      - -내용 적용-
        객체.style.css관련_프로퍼티 = "값";
      - -적용된 내용 조회-
        var 변수 = 객체.style.css관련_프로퍼티
    - CSS 클래스 접근
      - 클래스추가 — 객체.classList.add(name);
      - 클래스 제거 — 객체.classList.remove(name);
      - 클래스 on/off — 객체.classList.toggle(name);
      - 클래스 존재 여부 boolean값으로 반환 — 객체.classList.contains(name);
  - HTML 속성 제어
    - 속성값을 변경 — 객체.setAttribute(name, value);
    - 속성값을 제거 — 객체.removeAttribute(name);
    - 속성의 존재 확인 — 객체.hasAttribute(name);
    - 속성 값을 가져옴 — 객체.getAttribute(name);
- 이벤트 처리
  - Javascript 이벤트
    - 이벤트 리스너 — 이벤트 발생할 때, 연결된 함수 호출 ~할 때 시점을 파악할 수 있음 — 객체.addEventListener('이벤트이름', 콜백함수);
    - 이벤트 핸들러 — 이벤트 발생시, 실행될 코드 영역 — <태그이름 on이벤트이름="...JS코드영역..."></태그이름>
  - HTML 안에 내용 넣기
    - HTML 객체는 InnerHTML 속성 값을 갖는다 — 객체.innerHTML = "내용";
  - HTML태그를 객체로 가져오기
    - ID값으로 가져오기 — const 객체 = document.getElementById("ID이름");
    - 태그 이름으로 가져오기 — const 객체 = document.getElementsByTagName("태그이름")
    - CSS 단일 객체로 가져오기 — const 객체 = document.querySelector("CSS선택자");
    - CSS 복수 객체로 가져오기 — const 객체 = document.querySelectorAll("CSS선택자");
    - CSS Class 값으로 가져오기 — const 객체 = document.getElementsByClassName("CLASS이름");
- 웹 페이지와 Javascript
  - HTML 안에서 Javascript 위치
    - <script></script>태그 안에서 어디에나 가능
      - Html은 한 줄 씩 차례대로 해석
        - <body>태그 맨 마지막에 배치하는 것이 페이지 식별에 더 수월함

- Vanilla5
  - Form요소
    - <form>객체 획득
      - id속성 값으로 접근
        - const 객체 = document.getElementById("id값");
      - name속성 값으로 접근
        - const 객체 = document.name값;
    - 선택 가능한 요소 제어
      - 드롭다운 제어
        - <select>태그 드롭다운 요소는 그 자체가 배열이며, <option> 태그의 배열 요소가 됨.
          - 객체1.seletedIndex = 특정위치
            - 특정 위치 강제 선택
            - 특정위치 값이 0인 경우 선택한 것으로 간주하지 않음
          - const 객체2 = 객체1.seletedIndex
            - 현재 선택된 요소의 위치
          - const 객체3 = 객체1[객체2].value;
            - 현재 선택된 요소의 value값
        - option
          - option 배열은 0부터 시작
            - 0번째 항목은 대부분 항목이 없음.
          - option 태그 value 속성 값이 실제로 작동
        - 드롭다운 객체에 대한 selectedIndex는 선택된 요소의 배열 인덱스를 의미
    - <input>요소의 입력값 처리
      - 값의 설정
        - 객체1.value = "설정값";
      - 값의 조회
        - const 객체2 = 객체1.value;
    - <form>의 rest, submit처리
      - reset
        - document.객체.rest();
      - submit
        - input
          - key up은 실시간으로 내용이 적용됨
          - change는 입력요소가 빠져 나온 후에 실행됨(실시간으로 적용되지 않음)
        - submit 이벤트 처리
          - <form>요소에 onsubmit 이벤트 적용 후, 데이터 전송을 방지하기 위한 처리가 필요함
            이후 입력값을 검사한 후 코드 하단부에서 강제로 submit()처리를 해야 백엔드에 정보전달이 됨

```
document.querySelector("폼요소에 대한 selector").addEventListener("submit", e => {
e.preventDefault();
.
.
.
e.currentTarget.submit();
});
```

        - <form>요소의 action속성에 지정된 페이지로 사용자의 입력 내용 전송하는 기능
        - <input type="submit"/>요소를 클릭하거나 자바스크립트의 구문의 호출을 통해서 구현
          - document.객체.submit();
    - <form>안의 <input>요소 접근
      - id 속성 값으로 접근
        - const 객체 = document.getElementById("id값");
      - name 속성 값으로 접근
        - const 객체 = document.name값.input_name;
  - HTML 생성하기
    - 노드추가
      - 요소.appendChild(자식노드)
        - 자식노드를 요소의 마지막 자식 노드로 추가가됨.
    - 노드 생성
      - 요소노드 생성
        - const 객체 = document.createElement(tagName)
          - 매개변수 tagName에는 태그 이름을 나타내는 문자열을 인수로 전달
      - 텍스트 노드 생성
        - const 객체 = document.createTextNode(text)
          - 매개변수 text에는 텍스트 노드 값으로 사용할 문자열 인수로 전달
  - HTML 탐색하기
    - Node.prototype
      - 노드를 모두 탐색하여 NodeList에 담아 반환한다.
        (노드뿐만 아니라 텍스트 노드도 포함되어 있음)
        - parentNode(부모노드), previousSibling(이전 형제노드),
          nextSibling(다음 형제노드), childNodes(자식노드)
    - Element.prototype
      - 요소노드만 탐색하여 HTMLCollection에 담아 반환.
        (텍스트 노트가 포함되지 않는다.)
        - previousElementSibling(이전 형제노드), nextElementSibling(다음 형제노드),
          children(자식 노드)

Vanilla5

- 데이터 읽기,쓰기
  - 웹 스토리지
    - 세션스토리지
      - 세션 종료 시 (브라우저를 닫을 경우) 클라이언트에 대한 정보 삭제
        *세션은 프론트 엔드와 백엔드가 공유하는 것
        - 입력 된 정보 저장, 비로그인 장바구니
    - 로컬스토리지
      - 클라이언트(프론트엔드)에 대한 정보를 영구적으로 저장
        - 자동 로그인 저장
    - 장점
      - 서버에 불필요하게 데이터를 저장하지 않는다(백엔드에 전송되지 않음)
      - 저장 가능한 데이터 용량이 큼(약 5Mb, 브라우저마다 차이 존재)
    - 특징
      - 사용자 컴퓨터에만 정보를 저장하여 보안에 유리하지만, 백엔드와 공유가 힘듦
      - 클라이언트에 대한 정보 저장
      - key와 value형태로 이루어짐
  - 쿠키
    - 사용자 웹 브라우저에 저장되는 텍스트 데이터
      - 사용 예
        - 오늘 본 상품
        - 팝업창
      - 단점
        - 페이지 이동, 새로고침이 될 때마다 백엔드로 전송됨
        - 암호화 존재 X -> 사용자 정도 도난 위험
        - 쿠키 용량이 작음(약 4kb)
      - 장점
        - 대부분의 브라우저가 지원
    - 정보 저장 방법
      - 영구저장이 가능하지 않음(설정된 초단위 만큼만 유지됨)
        - 0으로 설정 시 즉시 삭제
      - 이름과 값은 모두 URLEncoding처리 되어야함(영어, 숫자 제외)
      - "이름=값; 유효시간; 유효경로; 유효도메인"으로 설정
- CRUD(Create, Read, Update, Delete의 약자)
  - 정보를 추가할 수 있다
  - 정보를 검색할 수 있다
  - 정보를 편집할 수 있다
  - 정보를 삭제할 수 있다
  - Restful API
    - path방식
      - QueryString에서 변수 명은 미리 정해놓고 값만 전달하는 방식
        - QueryString이 가진 단점을 다 가지고 있다.
    - Post, Put, Delete 방식
      - 요청 변수를 HTTP Header에 포함하여 전송하는 방식
        - 파일 업로드 가능
        - 웹 브라우저를 통한 조작 불가능
        - 백엔드 설정에 따라 전송 가능 용량 설정이 가능(일반적으로 20M내외)
        - 요청정보를 열람 할 수 있지만, 사용자에게는 노출이 되지 않음으로 보안상 유리함
    - QueryString이란
      URL 뒤에 ?를 기준으로 '변수명=값&변수명=값&' 형식으로 요청변수 나열
      - 변수를 이용하여 웹 브라우저 조작이 가능하다
      - URL 총 길이는 1024byte로 용량의 한계가 있다.
      - 변수가 URL에 노출되기 때문에 보안용으로 사용하면 안됨
  - ajax방식에서 CRUD는
    Post, Get, Put, Delete
  - SQL 방식에서 CRUD는
    Insert, Select, Update, Delete
    - Backend에서 사용됨
- XMLHttp Request(Ajax)
  - Json
    - 외부에서 Json을 받아드리기 위해 Json 서버 가동
      - Json은 리스트 형식으로 이루어짐
  - Ajax(통신기술)는 데이터를 교환함
  - xml => 데이터 구조적 표현, 태그 이름은 사용자가 지정할 수 있음
    - Json으로 경량화 됨(경량의 데이터 표현)
  - 통신이란,
    2개의 프로그램 안에서 요청과 응답을 위한 메서드와 프로퍼티를 제공한다.
    - 요청=접속Request(req)
      - user -> frontend -> backend에게 요청을 함
    - 응답=결과가 돌어오는 것Response(res)
      - backend->frontend->user에게 응답을 함
  - async/await란,
    프로미스를 기반으로 동작하며, 프로미스 후속 처리 메서드 없이 동기화처럼 결과를 반환
    - async함수
      - async 키워드를 사용해 언제나 프로미스를 반환함.
    - awiat 키워드
      - 언제나 async함수 내부에서 사용해야 하며,비동기 처리가 수행된 상태가 될 때 까지 대기.
        반드시 프로미스 앞에서 사용해야 함.
    - axios 라이브러리
      - 최근에는 구현 과정을 axios를 통해 구현함.

- Vanilla5
  - 라이브러리 활용하기
    - chart
      - type, data, options 값이 제일 중요
        - polar Area
        - 선 그래프
        - 도넛 그래프
        - 파이 그래프
        - 막대 그래프
          - 다중 막대 그래프
          - 기본 막대 그래프
        - 산점도 그래프
          - x축에 따른 y축의 변화
    - lightbox
    - aos
      - data-aos-offset: 대상 요소가 화면 하단으로부터 떨어진 거리.(이 거리에 도달하면 애니메이션이 작동됨)
      - data-aos-easing: 애니메이션 재생 옵션(css의 transition 속성 값을 따름)
      - data-aos-duration: 애니메이션 속도 (1/1000초)
      - data-aos : 애니메이션 종류 문자열
    - animate
      - delay: 시간 => 시작 전 딜레이 시간
      - iterationStart: 반복값 => 반복시작
      - endDelay:시간 => 재생 종료 후 딜레이 시간
      - duration:시간 => 재생에 걸리는 시간(1/1000초)
      - easing: '재생옵션' => (linear, ease-in, ease-out, ease-in-out)
      - iterations = 반복값 => 설정한 횟수만큼 반복, infinity은 무한반복
      - direction:"재생방향(nomal,reverse, alternate, alternate-reverse)"
    - mediabox
  - 기타기능
    - 인터넷 주소
      - 변수가 여러 개일 경우 &로 구분
      - ? 뒤에는 변수이름=변수값
      - #은 현재위치로 이동
    - Location
      - 디렉토리 이하 경로 정보 — 경로 정보 — 아파트 호수
      - 호스트 컴퓨터 이름 — 네트워크 주소 — 아파트 단지
      - 라이브 서버 — 각 폴더와 파일을 열람하겠다는 의미
      - 포트 번호 — 포트번호 1개에 1개의 프로그램과 연결됨(1~65000개 번호가 있음) — 아파트 동
    - HTTP 통신방식의 종류 =>프로토콘 종류
      - https: => 웹 보안(정보들이 다 보안으로 되어있음)
      - http: => 웹 방식(일반 형식으로 정보가 공개됨)

miro