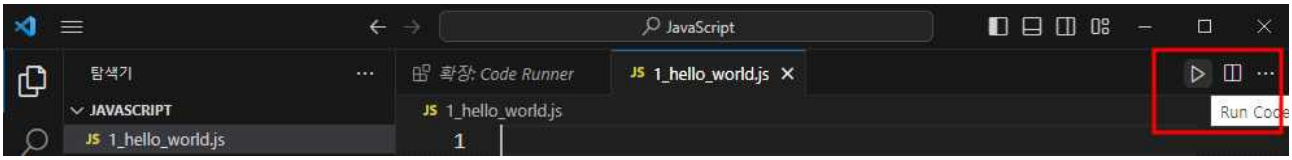
JavaScript

1장. 자바스크립트에 필요한 IDE 설치와 JavaScript 기초

1. Visual Studio Code 설치

① plug-in 설치

- code runner(Control +Alt + L)
- live server



2. 프로젝트 폴더 생성 후 깃허브와 연결

3. 기본 출력과 주석 처리하기

▶ 1_hello_world.js

```
// 코멘트를 작성하는 첫번째 방법입니다.
console.log('Hello World')

/**
 * 이렇게 하면 여러줄에 걸쳐서
 * 코멘트를 작성할 수
 * 있습니다.
 * */

console.log('Hello', 'World')
```

4. 변수의 사용 및 선언과 할당

① 변수선언

- let, const

② 선언과 할당

- 선언 : 변수를 선언하는 것
- 할당 : 선언된 변수에 값을 넣는 것

③ 변수 naming 규칙

- 일반적으로 영어와 숫자를 사용
- 특수기호는 달러($)와 언더스코어(_) 만 사용 가능

- 숫자로 변수명을 시작할 수 없다.
- 키워드를 변수명으로 사용할 수 없다.

ex> var const = '';

④ Naming Convention
- camel case : 첫문자 소문자 뒤는 단어 시작마다 첫글자를 대문자로(namingConvention)
- snake case : 전체 소문자를 사용하며 단어별로 _ 로 연결(naming_convention)
- pascal case : 매 단어 첫자를 대문자로 사용(Naming Convention)

※ 자바스크립트를 비롯한 대부분 개발툴에서 camel case를 사용함.

▶ 2_varialbe.js

```
/**
 * 변수의 사용
 * 1. var : 더 이상 쓰지 않는다.
 * 2. let
 * 3. const
 */

var name = '김형민'
console.log(name)
var age = 22
console.log(age)

let ive = '아이브'
console.log(ive)

ive = '안유진'
console.log(ive)

let girlGroup
// undefined 출력(선언되었으나 할당되지 않았다.)
console.log(girlGroup)

const newJeans = '뉴진스'
console.log(newJeans)
newJeans = '뉴 진스'
```

```
김형민
22
아이브
안유진
undefined
뉴진스
k:₩lect₩$_Front_End₩JavaScript₩1_basics₩2_variable.js:25
newJeans = '뉴 진스'
           ^

TypeError: Assignment to constant variable.
k:₩lect₩$_Front_End₩JavaScript₩1_basics₩2_variable.js:21
newJeans = '뉴 진스'
           ^

TypeError: Assignment to constant variable.
```

5. Data Type

여섯개의 Primitive Type과 한 개의 Object 타입이 있다.

1) Number : 숫자
2) String : 문자열
3) Boolean : 논리값(true, false)
4) undefined
5) null
6) Symbol
7) Object : 객체(Function, Array, Object)

※ 각 변수의 type을 확인하는 법 : typeof

① Number Type

▶ 3_1_number_test.js

```
const age = 32
const tempature = -10
const pi = 3.14
console.log(typeof age)
console.log(typeof tempature)
console.log(typeof pi)
```

```
console.log('--------------------')

const infinity = Infinity  // 양의 무한대 값
const nInfinity = -Infinity // 음의 무한대 값
console.log(typeof infinity)
console.log(typeof nInfinity)
```

② String Type

```
※※ Template Literal

1) Escape Character
 ⓐ new Line : ₩n
 ⓑ tab : ₩t
 ⓒ ₩(Back_Slash)는 두번 입력해서 사용

2) 백틱(`) 사용
 ⓐ 여러 줄에 걸쳐 출력할 때 매우 유용하다.

3) Template Literal 내에서의 변수 사용방법
 ⓐ 백틱 내에 변수명을 ${변수명}으로 감싸 처리한다.
```

▶ 3_2_string_test.js

```
const codeTest = '코드테스트'
console.log(codeTest)

// 문자열 안에 작은 따옴표와 큰 따옴표 처리하기
const ive = "'아이브' 장원영"
console.log(ive)

// Escape Character
const iveYuJin = '아이브 ₩n안유진'
console.log(iveYuJin)

const iveWonYoung = '아이브 ₩t장원영'
console.log(iveWonYoung)

const iveGaEul = '아이브 ₩' 가을'
console.log(iveGaEul)
```

```
console.log('--------------------')

// Template Literal에서 백틱(`)으로 변수 사용
const groupName = '아이브'
console.log(groupName + '안유진')
console.log(`${groupName} 안유진`)
```

③ Boolean Type

▶ 3_3_boolean_test.js

```
const isTrue = true
const isFalse = false

console.log('True : ' + isTrue)
console.log('False : ' + isFalse)
```

④ undefined Type : 사용자가 직접 값을 초기화하지 않았을 때 지정되는 값.
(직접 undefined로 값을 초기화 하는 것은 ○○다)

▶ 3_4_undefined_test.js

```
let noInit = undefined
let noInit2

console.log(noInit)
console.log(noInit2)

console.log(typeof noInit2)
```

⑤ null Type : undefined와 마찬가지로 값이 없다는 뜻이나, 자바스크립트에서는 개발자가 명시적으로 없는 값으로 초기화할 때 사용된다.

☞ 영국의 컴퓨터 과학자 Tony Hoare가 만든 개념
- 프로그래밍 언어인 C언어에서는 생성된 메모리의 주소를 '포인터'라는 것이 가리키게 되어 포인터를 통해 데이터를 가져올 수 있게 된다. 그러나, 이때 포인터가 아무것도 가리키고 있지 않다는 것을 나타내기 위해 NULL이라는 개념을 사용하게 되는 것
- null 참조를 만든 것은 10억 달러의 실수 → 2009년 소프트웨어 컨퍼런스(null 만든 본인 이야기)

▶ 3_5_null_test.js

```
let init = null
console.log(init)
console.log(typeof init)

// 약간의 버그개념으로 이해해야 함.

null
object
```

⑥ Symbol Type
- 유일무이한 값을 선언할 때 사용
- 다른 Primitive Type과는 다르게 Symbol 함수를 호출해서 만든다.

▶ 3_6_Symbol_test.js

```
const test1 = '1'
const test2 = '1'
console.log(1 == '1')
console.log(test1 == test2)
console.log(test1 === test2)

console.log('-------------------------')

const symbol1 = Symbol('1')
const symbol2 = Symbol('1')
console.log(symbol1 === symbol2)
console.log(symbol1 === test1)
```

⑦ Object Type
- Map 자료구조 처럼 키 : 값의 쌍으로 이루어져 있다.

▶ 3_7_object_test.js

```
const dictionary = {
  red : '빨강색',
  orange : '주황색',
  yellow : '노랑색',
}

console.log(dictionary)
```

```
console.log(dictionary.red)
console.log(dictionary['yellow'])
console.log(typeof dictionary)

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

{ red: '빨강색', orange: '주황색', yellow: '노랑색' }
빨강색
노랑색
object
```

⑧ Array Type

- 값을 리스트로 나열 할 수 있는 타입

▶ 3_7_array_test.js

```
const iveMemberArray = [
  '안유진',
  '장원영',
  '레이',
  '이서',
  '가을',
  '리즈',
]

console.log(iveMemberArray)

/**
 * Array 안에는 index(0부터 시작) 가 들어있어서
 * index 값을 이용해 다양한 처리를 할 수 있다.
 *  */

console.log(iveMemberArray[5])
console.log(iveMemberArray[3])

// 배열에 값을 할당하는 방법
iveMemberArray[0] = '아이브'
console.log(iveMemberArray)
console.log(typeof iveMemberArray)
```

▷ typing 종류

ⓐ static typing : 변수 선언 시 변수에 타입을 명시한다
ⓑ dynamic typing : 변수 선언시 type을 명시하지 않아도 변수에 값이 할당될 때 값에 의해 타입을 추출한다.

6. 호이스팅(Hoisting)

- 모든 선언된 변수가 최상단으로 이동되는 것처럼 느껴지는 현상
- var 키워드는 변수 호이스팅을 막지 못한다.
- 변수 선언 후 할당이라는 안정적 개발환경을 만족시키기 위해서는 const 및 let 키워드 사용을 권장.

▶ 4_hoisting.js

```
console.log(name)
// ReferenceError 오류가 발생하지 않음!!
var name = '김형민'
console.log(name)

// let과 const도 hoisting 처리 된다.

console.log(yuJin)
//ReferenceError: Cannot access 'yuJin' before initialization
let yuJin = '안유진'
```

7. 연산자(Operators)

1) 산술연산자

▶ 5_operators.js

```
// 1) 산술연산자
console.log(10 + 10)
console.log(10 - 10)
console.log(10 / 10)
console.log(10 * 10)
console.log(10 % 10)  // 나머지를 구하는 연산자
console.log(10 % 3)
console.log(10 * (10+10))

console.log('---------------------------')

// 2) 증감연산자
let number = 1
number++
console.log(number)
```

```
number--
console.log(number)

console.log('--------------------------')

let result = 1
console.log(result)
result = number++
// ① result = number
// ② number = number + 1
console.log(result, number) // 1 2 출력

result = number --
// ① result = number
// ② number = number - 1
console.log(result, number) // 2 1 출력

result = ++ number
// ① number = number + 1
// ② result = number
// 현대에서 거의 사용하지 않음.(뒤통수 맞을 수 있음.)
console.log(result, number)

console.log('--------------------------')
// 숫자가 아닌 타입에 +, - 사용하기
let sample = '99'
console.log(+sample)
console.log(typeof +sample)

console.log(sample)
console.log(typeof sample)

sample = true
console.log(+sample)
console.log(typeof +sample)
// 실제 true = 1로, false = 0으로 표현되기도 한다.

sample = '안유진'
console.log(+sample)
```

```
// NaN : Not a Number

console.log('---------------------------')

// 3) 할당연산자(assignment operator)
number = 100
console.log(number)
number += 10
console.log(number)
number -= 10
console.log(number)
number *= 10
console.log(number)
number /= 10
console.log(number)
number %= 10
console.log(number)

console.log('---------------------------')

// 4) 비교연산자
// ① 값의 비교
console.log(5 == 5)
console.log(5 == '5')
console.log(0 == '')
console.log(true == 1)
console.log(false == 0)
console.log(true == '1')

console.log('---------------------------')

// ② 값과 타입을 같이 한꺼번에 비교
console.log(5 === 5)
console.log(5 === '5')

console.log('---------------------------')

/**
 * ③ 관계연산자
 * - 값의 비교 : > , < , >= , <=
```

```
 * - 값과 타입을 같이 한꺼번에 비교 : !==
 */

// ④ 삼항조건연산자
console.log(10 > 0 ? '10이 0보다 크다' : '10이 0보다 작다.')

console.log('--------------------------')

// ⑤ 논리연산자(&& , ||)
console.log(true && true)
console.log(true && false)

console.log(true || true)
console.log(true || false)

console.log(10>1 && 20>2);
console.log(10>1 || 20>2);

/**
 * ⑥ 단축평가(short circuit evaluation) - 실무서 많이 씀
 *  - && 사용 : 좌측 true → 우측 반환
 *  - && 사용 : 좌측 false → 우측 반환
 *  - || 사용 : 좌측 true → 좌측 반환
 *  - || 사용 : 좌측 false → 우측 반환
 */

console.log('--------------------------')
console.log(true || '아이브')
console.log(false || '아이브')

console.log(false && '아이브')
console.log(true && '아이브')

console.log(true && true && '아이브')
console.log(true && false && '아이브')

// ⑦ 지수연산자
console.log(2 ** 2)
console.log(10 ** 3)
```

```
// ⑧ null 연산자 : 좌측값이 null 혹은 undefined 이면 우측값 반환
console.log('--------------------------')
let name
console.log(name)

name = name ?? '김형민'
console.log(name)

name = name && '아이브'
console.log(name)
```

2장 JS기본 문법

1. If and swith

▶ if and switch : 6_if_and_switch.js

```
// ① if 구문
```

```
let number = 5;
if((number % 2) ===0){
   console.log('number 변수는 짝수 입니다.');
} else {
   console.log('number 변수는 홀수 입니다.');
}

if((number % 2) === 0){
   console.log('2의 배수');
} else if((number % 3) === 0){
   console.log('3의 배수');
} else if((number % 4) === 0){
   console.log('4의 배수');
} else if((number % 5) === 0){
   console.log('5의 배수');
} else {
   console.log('2, 3, 4, 5의 배수가 아닙니다.');
}

// ② switch 구문
const englishDay = 'monday';
let koreanDay;
switch(englishDay){
   case 'monday' :
      koreanDay = "월요일";
      break;
   case 'tuesday' :
      koreanDay = "화요일";
      break;
   case 'wendnesday' :
      koreanDay = "수요일";
      break;
   case 'thursday' :
      koreanDay = "목요일";
      break;
   case 'friday' :
      koreanDay = "금요일";
      break;
   case 'saturday' :
```

```
    koreanDay = "금요일";
    break;
  default :
    koreanDay = '주말'
    break;
}

console.log(koreanDay);
```

2. Loop

▶ loop : 7_loops.js

```
// ① for loop
// 0에서 9까지 출력+
for(let i=0 ; i < 10 ; i ++){
  console.log(i);
}

// ② 중첩 for Loop
for (let i=0; i < 3; i++){
  for(let j=3; j > 0; j--){
    console.log(i, j);
  }
}

console.log('')

// ③ 퀴즈> * 을 이용해서 5×5의 정사각형, 우측으로
// 기울어진 삼각형을
// 각각 출력하시오.

console.log('--- 5×5의 정사각형 ---');
let square = '';
let side = 5;
for(let i=0; i<side; i++){
  for(let j=0; j<side; j++){
    square = square + '*';
  }
  square = square + '₩n';
}
console.log(square);
```

```
console.log('--- 우측 기울어진 삼각형 ---');
for(let i=0; i<side; i++){
  for(let j=0; j<i; j++){
    square = square + '*';
  }
  square = square + '₩n';
}
console.log(square);

console.log('------------------')

// ④ for ~ in Loop
const yuJin = {
  name : '안유진',
  year : 2003,
  group : 'ive'
}

for(let key in yuJin){
  console.log(key);
}

console.log('------------------')
const iveMemberArray = ['안유진', '가을', '레이', '장원영', '리즈', '이서'];
for(let key in iveMemberArray){
  //console.log(key);
  console.log(`${key} : ${iveMemberArray[key]}`);
}

console.log('------------------')
// ⑤ for ~ of loof : iterator 한 자료구조에서 사용가능(list, array)
for(let value of iveMemberArray){
  console.log(value);
}

console.log('------------------')
// ⑥ while 구문
let number = 0;
while(number < 10){
```

```
    number++;
    console.log(number);
}
console.log(number);

console.log('------------------')
// ⑦ do ~ while 구문
number = 0;
do {
    number++;
    console.log(number);
} while(number < 10);
console.log(number);

// ⑧ break/continue 구문
console.log('break------------------')
for(let i=0; i<10; i++){
    if(i===5){
        break;
    }
    console.log(i);
}

console.log('continue------------------')
for(let i=0; i<10; i++){
    if(i===5){
        continue;
    }
    console.log(i);
}
```

3. 타입변환

▶ loop : 8_type_conversion.js

1) 명시적 형변환

```
// 1) 명시적 형변환
let age = 32;
let stringAge = age.toString();
console.log(typeof stringAge, stringAge);
```

2) 암묵적 형변환

```
// 2) 암묵적 형변환
let test = age + '';
console.log(typeof test, test);

// 기능적으로는 가능하나 명시적 코드작성을 위해 거의 사용되지 않음
```

3) 명시적 변환 추가 학습

① 문자로 변환 ; toString()

② 숫자로 변환 : parseInt, parseFloat

③ Boolean type으로 변환

- 무조건 true 인 경우 : Object, Array, 비어있지 않은 스트링, null
- 무조건 false 인 경우 : 아무 글자도 없는 빈 스트링, 값이 없는 경우, 0(zero), undefined

```
console.log(typeof (99).toString(), (99).toString());
console.log(typeof parseInt('0'), parseInt('0'));
console.log(typeof parseFloat('0.99'), parseFloat('0.99'));

// boolean 타입으로 변환
console.log('--boolean 타입으로 변환--');
console.log(' !₩'x₩' : ' + ! 'x');
console.log(' !!₩'x₩' : ' + !! 'x');
console.log(' !!₩'₩' : ' + !! '');
console.log(' !! 0 : ' + !! 0);
console.log(' !! ₩'0₩' : ' + !! '0');
console.log(' !! ₩'false₩' : ' + !! 'false');
console.log(' !! false : ' + !! false);
console.log(' !! undefind : ' + !! undefined);
console.log(' !! null : ' + !! null);
console.log(' !! {} : ' + !! {});
console.log(' !! [] : ' + !! []);
```

4. function

▶ 함수의 필요성

→ 만약에 2라는 숫자에 *10 / 2 % 3 을 계산한 후 스트링으로 변환해서 출력하고 싶다면 어떻게 해야 할까? 또, 3을 바꾸고 싶다면? → 중복 코드가 많이 발생 → 함수 필요

※ 좋은 코드는 DRY원칙을 따르도록 구성하는 것이 중요
D : Don't
R : Repeat
Y : Youself

▶ function : 9_function.js

```
// 1) 단일 함수의 생성
console.log('1) 단일함수의 생성----------')
function calculate(number){
   console.log((number*10/2%3).toString());
}

calculate(4);

// 2) 여러개 파라미터 사용
console.log('2) 여러개 파라미터 사용----------')
function multiply(x, y){
   console.log(x * y);
}
multiply(2, 4);

// 3) default 파라미터 사용
console.log('3) default 파라미터 사용----------')
function minus(x, y = 1){
   console.log(x - y);
}
minus(5);
minus(5, 2);

// 4) 결과 반환받기
console.log('4) 결과 반환받기 ----------')
function divide(x, y){
   return x / y;
}
```

```
console.log(divide(7, 3));

// 5) Arrow 함수
console.log('5) Arrow 함수 ----------')
// ① 명시적 함수 선언 방법
const multi01 = function(x,y){
  return x * y;
}
// arrow 함수 표현법
const multi02 = (x, y) => {
  return x * y;
}
console.log('multi01(7, 3) : ' + multi01(7, 3));
console.log('multi02(7, 3) : ' + multi02(7, 3));

// 간편하게 표현하는 법
const multiply3 = (x, y) => x*y;
console.log(multiply3(2, 3));

// 인자가 1개 일때
const multiply4 = x => x*2;
console.log(multiply4(2));

// 6) agruments 키워드
console.log('6) agruments 키워드----------')
const multiplyThree = function(x, y, z){
  console.log(arguments);
  return x*y*z;
}
// arguments 키워드는 Arrow Function에서는 사용 불가

console.log(multiplyThree(4,5,6));

console.log('무한 agruments 받기----------')
const multiplyAll = function(...agruments){
  return Object.values(arguments).reduce((a,b) => a*b, 1);
}
console.log(multiplyAll(2,3,4,5,6,7,8,9));
```

```
// Immediately invoked function
console.log('즉시 실행함수----------');
(function (x, y){
   console.log(x*y);
})(4, 5);

// 타입확인
console.log('타입확인----------');
console.log(typeof multiply);
console.log(multiply instanceof Object);
```

5. 기본적으로 제공하는 function

▶ 함수의 필요성

→ 만약에 2라는 숫자에 *10 / 2 % 3 을 계산한 후 스트링으로 변환해서 출력하고 싶다면 어떻게 해야 할까? 또, 3을 바꾸고 싶다면? → 중복 코드가 많이 발생 → 함수 필요

▶ 10_etc_basic_function.js

```
// Array Function
let iveMembers = [
   '안유진',
   '가을',
   '레이',
   '장원영',
   '리즈',
   '이서'
];

console.log(iveMembers);

// ① push() : Array 맨 끝에 새로운 item추가(array 직접 변경)
iveMembers.push('An Yu Jin');
console.log(iveMembers);
// push()는 삽입 후 전체 추가된 것 포함한 배열의 길이를 반환한다.
console.log(iveMembers.push('Jang Won Young'));

// ② pop() : Array 맨 마지막 item 반환 → 삭제
console.log(iveMembers.pop());
console.log(iveMembers);
```

```
// ③ shift() : Array 맨 첫번째 값을 반환받고 삭제
console.log(iveMembers.shift());
console.log(iveMembers);

// ④ unshift()는 맨 앞에 추가한 후 push()와 같은 값을 리턴한다.
console.log(iveMembers.unshift('안유진'));
console.log(iveMembers);

// ⑤ splice(시작위치, 삭제할 item 수)
iveMembers.splice(0 ,3);
console.log(iveMembers);



/**
 * 기존 정의한 원본 데이터가 직접 변경되는 것은 자료처리상 상당히 위험한 부분이다.
 * 가급적 원본을 건드리지 않고 함수를 통해 변형 한 후
 * 원하는 결과를 얻어 사용하는 것이 바람직하다.
 */

// 원본을 변형하지 않는 함수.

// 배열 리셋

iveMembers = [
  '안유진',
  '가을',
  '레이',
  '장원영',
  '리즈',
  '이서'
];

// ⑥ concat() :  새로운 item을 배열 맨 뒤에 삽입하나, 원본은 수정되지 않는다.
console.log(iveMembers.concat('An Yu Jin'));
console.log(iveMembers);

// ⑦ slice(삭제할 시작인덱스, 몇 인덱스 전까지) : 원본은 수정되지 않는다.
console.log(iveMembers.slice(0, 3));
console.log(iveMembers);
```

```
// ⑧ spread operator (...) : 기존 배열을 펼쳐서 처리함.
let iveMembers2 = [...iveMembers];
console.log('iveMembers2 : ' + iveMembers2); // 기존 배열을 펼쳐서 복사 함.

// 배열안에 배열 넣기
let iveMembers3 = [iveMembers];
console.log(iveMembers3);

let iveMembers4 = iveMembers;
console.log(iveMembers4 === iveMembers);

// ⑨ join() : 배열 내 자료를 ,(comma)로 구분한 후 문자열 형태로 반환
console.log(iveMembers.join());
console.log(typeof iveMembers.join());

// 구분자 주어서 출력하기
console.log(iveMembers.join('/'));

// ⑩ sort() : 오름차순 정렬하며 반환값 없고, 원본수정함.
console.log('sort ----------')
iveMembers.sort();
console.log(iveMembers);

// ⑩ sort() : 역순 정렬하며 반환값 없고, 원본수정함.
console.log('reverse ----------')
iveMembers.reverse();
console.log(iveMembers);

// sort() 함수로 오름차순, 내림차순 정렬하기
console.log('sort Asc/Desc----------')
let numbers = [
  1,
  9,
  7,
  5,
  3,
];

console.log(numbers);
```

```
// Ascending Sort
numbers.sort((a,b)=>{
  return a>b ? 1 : -1;
});
console.log(numbers);

// Descending Sort
numbers.sort((a,b)=>{
  return a>b ? -1 : 1;
});
console.log(numbers);

// ⑫ map() : 원래 배열을 변형시켜 새로운 배열을 돌려 줌(원본 유지)
console.log('map ----------')
console.log(iveMembers.map((x)=> x));
console.log(iveMembers.map((x)=> `아이브 : ${x}`));
console.log(iveMembers);

// 안유진에게만 아이브 : 안유진으로 출력하기
console.log(iveMembers.map((x)=>{
  if(x === '안유진'){
    return `아이브 : ${x}`
  } else {
    return x;
  }
}));

// ⑬ filter()
console.log('filter ----------')

numbers = [
  1,
  9,
  8,
  5,
  3,
];
console.log('전체 : ' + numbers.filter((x)=> true));
// 짝수만 출력하기
console.log('짝수 : ' + numbers.filter((x)=> x % 2 === 0));
```

```
// 3보다 큰 수 출력하기
console.log('3보다 큰수 : ' + numbers.filter((x)=> x > 3));

// 3보다 큰 수 찾아서 오름차순 정렬하기
console.log('3보다 큰 수 찾아서 오름차순 정렬하기 : ' +
        numbers.filter((x) => x>3).sort((a,b) => a>b ? 1 : -1));

// ⑭ find() : 해당하는 첫번째 값 반환
//    findIndex() : 해당하는 인덱스 반환
console.log('find --------------------');
console.log(numbers.find((x) => x % 2 === 0));
console.log(numbers.findIndex((x) => x % 2 === 0));

// ⑮ reduce() : 배열을 순회하면서 특정 기능을 처리
// reduce((p, n) => p+n, 0));
// p : previous, n : next, 0 : 첫 로딩 때 privious 값
console.log('reduce --------------------');

// reduce() 함수를 이용해서 배열의 합 구하기
console.log(numbers.reduce((p, n)=> p + n, 0));
```

6. Object

▶ 11_object.js

① object의 생성 – key : value 쌍으로 이루어지면 함수를 통해 method를 구현할 수 있다.

```
let yuJin = {
   name : '안유진',
   group : '아이브',
   dance : function(){
      return '안유진이 춤을 춥니다.';
   }
};

console.log(yuJin);
console.log(yuJin.name);
console.log(yuJin['group']);

const key = 'name';
console.log(yuJin[key]);
```

```
console.log(yuJin.dance());
```

② 변수로 선언된 값으로 객체 생성하기

```
// ② 변수로 선언된 값으로 객체 생성하기
console.log('② 변수로 선언된 값으로 객체 생성하기---------')
const nameKey = 'name';
const nameValue = '안유진';
const groupKey = 'group';
const groupValue = 'IVE';
const yuJin2 = {
  [nameKey] : nameValue,
  [groupKey] : groupValue,
}
console.log(yuJin2);
```

③ 변수 수정 및 새로운 key : value 추가하기와 삭제하기

```
// ③ 변수 수정 및 새로운 key : value 추가하기
console.log('③ 변수 수정 및 새로운 key : value 추가하기---------')
yuJin2['name'] = '민지'
yuJin2['group'] = '뉴진스';
console.log(yuJin2);
yuJin2.englishName = 'Min Ji';
console.log(yuJin2);
console.log('③ 삭제하기---------')
delete yuJin2.englishName;
console.log(yuJin2);
```

④ 객체의 특징

※ const로 선언했는데 삭제와 추가가 가능하다?

- const로 선언할 경우 객체 자체를 변경 할 수는 없다.
- 객체 안의 프로퍼티나 메서드는 변경할 수 있다.

```
// ④ 객체의 특징
console.log('④ 객체의 특징--------------')
const danielle = {
  name : '다니엘',
  group : '뉴진스',
}
```

```
console.log(danielle);
//danielle = {};
// TypeError: Assignment to constant variable.

danielle.group = 'NewJeans'
console.log(danielle);
```

⑤ 객체의 키/밸류 값을 배열로 얻어오기

```
// ⑤ 객체의 키/밸류 값을 배열로 얻어오기
console.log('⑤ 객체의 키/밸류 값을 배열로 얻어오기----------')
console.log(Object.keys(danielle));
console.log(Object.values(danielle));

// 좀 더 빠르게 객체 선언하기
const name = '민지';
const minJi = {
  name,
  // or
  name : name,
  // 내용이 같아서 한개만 생성 됨.
}
console.log(minJi);
```

7. copy by value vs copy by references

▶ 12_copy_by_value_and_references.js

① 참조의 종류

- call by value : 값에 의한 전달
- call by references : 참조에 의한 전달

1) 기본적으로 object를 제외한 primitive type은 값에 의한 전달을 하는 call by value 방식이다.
2) 객체는 call by references 방식을 취한다. 이는 객체가 인스턴스가 되면서 힙 영역에 올라가게 되고 이의 주소를 스택에 저장하기 때문에 그렇다.

② call by value

```
// ② call by value
console.log('② call by value -------')
let original = '안녕하세요';
let clone = original;
console.log('original : ' + original);
```

```
console.log('clone : ' + clone);

clone = clone + '뉴진스입니다.';
console.log('original : ' + original);
console.log('clone : ' + clone);

console.log(original === clone);
```

③ call by references

```
// ③ call by references
console.log('② call by references -------')
let originalObject = {
  name : '장원영',
  group : '아이브',
}

let cloneObject = originalObject;
console.log(originalObject);
console.log(cloneObject);
console.log('----------------------------')
cloneObject.name = '민지';
cloneObject.group = '뉴진스';
console.log(originalObject);
console.log(cloneObject);

console.log(originalObject === cloneObject);
```

④ object 에서 spread operator 사용하기

```
// ④ object 에서 spread operator 사용하기
console.log('④ object 에서 spread operator 사용하기-------')
let wonYoung1 = {
  name : '장원영',
  group : '아이브',
}

const wonYoung2 = wonYoung1;

let wonYoung3 = {
  name : '장원영',
```

```
  group : '아이브',
}

console.log(wonYoung1 === wonYoung2);
console.log(wonYoung1 === wonYoung3);
console.log(wonYoung2 === wonYoung3);
console.log('spread operator로 복사하기 -----')
const wonYoung4 = {... wonYoung3};
console.log(wonYoung4 === wonYoung3);
```

8. try ~ catch

▶ 13_try_catch.js

① error 객체의 생성

```
// ① error 객체의 생성
console.log('① error 객체의 생성 -----');
function runner1(){
  console.log('Hello');
  throw new Error('큰 문제가 생겼습니다.');
  console.log('Reach here?')
}

runner1();
-----------------------------------------------------------------------------------------------------------
  throw new Error('큰 문제가 생겼습니다.');
  ^

Error: 큰 문제가 생겼습니다.
```

② try ~ catch 구문으로 수정하기

```
// ② try ~ catch 구문으로 수정하기
console.log('② try ~ catch 구문으로 수정하기 -----');
function runner2(){
  try{
    console.log('Hello');
    throw new Error('큰 문제가 생겼습니다.');
    console.log('Reach here?')
  } catch(e) {
    console.log('---------catch---------');
    console.log('무슨 문제가 있나요?');
    console.log(e);
```

```
  } finally {
    // finally의 사용은 선택적이나 사용했을 때는 무조건 해당 block이 실행 됨
    console.log('---------finally---------');
  }
}

runner2();
```

2장. 클래스와 OOP

1. 클래스의 정의

▶ 2_class_and_oop → 1_class.js

> - 클래스(class, 어원: classification)는 객체 지향 프로그래밍(OOP)에서 특정 객체를 생성하기 위해 변수와 메소드를 정의하는 일종의 틀(template)이다. 객체를 정의하기 위한 메소드와 변수로 구성된다.
> ----- Wikipedia

1) 클래스? - 정보를 일반화 해서 정의하는 방법
즉, 공통의 정보를 추출해서 변수로 선언하고, 객체가 가질 수 있는 운동성을 메서드로 정의한다.

▷ 그룹명 : 아이브

name	year	englishName
안유진	2003	AnYuJin
가을	2002	Gaeul
레이	2004	Ray
장원영	2004	JangWonYoung
리즈	2004	LIZ
이서	2007	LeeSeo

▷ 그룹명 : 뉴진스

name	year	englishName
민지	2004	MinJi
하니	2004	Hanni
해린	2006	
혜인	2008	
다니엘	2005	Danielle

2) 아이돌그룹 클래스 정의(클래스의 이름은 대문자로 시작) → 멤버 입력

```
// ① 아이돌멤버 클래스 정의
class IdolMember{
   name;
   year;
```

```
  // 생성자
  constructor(name, year){
    this.name = name;
    this.year = year;
  }
}

const yuJin = new IdolMember('안유진', 2003);
console.log(yuJin);
const gaeul = new IdolMember('가을', 2002);
console.log(gaeul);
const ray = new IdolMember('레이', 2004);
console.log(ray);
const jangWonYoung = new IdolMember('장원영', 2004);
console.log(jangWonYoung);
const liz = new IdolMember('리즈', 2004);
console.log(liz);
const leeseo = new IdolMember('리즈', 2007);
console.log(leeseo);

// ② 인스턴스에서 값 가져오기
console.log('② 인스턴스에서 값 가져오기 ---------')
console.log(leeseo.name);
console.log(leeseo.year);

// ③ method 정의
console.log('③ method 정의 ---------')
class IdolMember2{
  name;
  year;

  // 생성자
  constructor(name, year){
    this.name = name;
    this.year = year;
  }
  sayName(){
    return `안녕하세요. 저는 ${this.name} 입니다.`;
  }
}
```

```
const minJi = new IdolMember2('민지', 2004);
console.log(minJi.sayName());
console.log(typeof IdolMember2);
console.log(typeof minJi);
```

2. Getter and Setter

▶ 2_getter_and_setter.js

```
class IdolMember{
  name;
  year;

  // 생성자
  constructor(name, year){
    this.name = name;
    this.year = year;
  }

  /**
   * ① getter and setter 생성
   * 1) 데이터를 가공해서 새로운 데이터를 반환할 때
   * 2) private 한 값을 반환할 때
   * */

  get getNameAndYear(){
    return `${this.name} - ${this.year}`;
  }

  set setName(name){
    this.name = name;
  }

  playDance(){
    return `${this.name}이 춤춰요`;
  }
}

const wonYoung = new IdolMember('장원영', 2003);
// getter 호출 시 함수에 붙이는 괄호()를 붙이지 않는다.
console.log(wonYoung.getNameAndYear);
```

```
// setter 는 변수에 값을 직접 할당하는 것과 같은 방법을 사용한다.
//wonYoung.setName('이무기');  ----- 오류 발생함.
wonYoung.setName = 'JangWonYoung';

console.log(wonYoung);
console.log(wonYoung.playDance());

  /**
   * ② private member 변수의 사용
   * 변수를 # 으로 선언하고 getter를 통해 접근한다.
   * */

console.log('② private member 변수의 사용 --------')
class IdolMember2{
  #name;
  year;

  // 생성자
  constructor(name, year){
    this.#name = name;
    this.year = year;
  }
  // getter
  get getName(){
    return this.#name;
  }

  //setter
  set setName(name){
    this.#name = name;
  }
}

const yuJin = new IdolMember2('안유진', 2003);
// private로 선언하면 console.log(yuJin); 결과에서도 확인이 불가하다.
console.log(yuJin);
console.log(yuJin.name);  // getter가 없으면 undefined
console.log(yuJin.getName);
```

```
yuJin.setName = 'AnYuJin';
console.log(yuJin);
console.log(yuJin.getName);
console.log(yuJin.year);
```

3. static 키워드

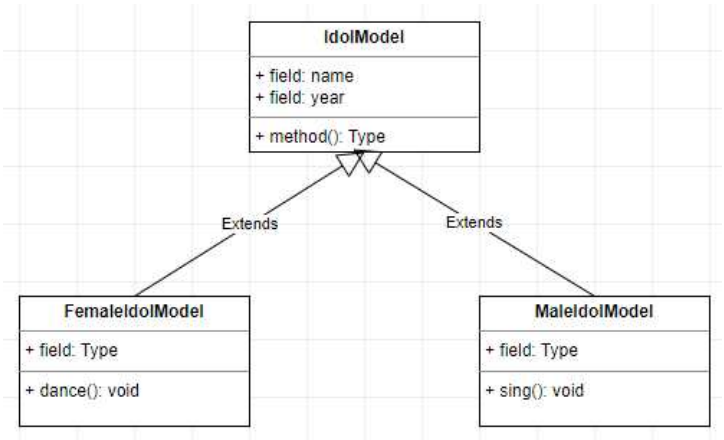
▶ 3_static_keyword.js

```
// ① static 변수의 선언과 사용
class IdolMember{
  static groupName = '뉴진스';
  name;
  year;

  // factory constructor : 자주 사용함.
  constructor(name, year){
    this.name = name;
    this.year = year;
  }
  static getGroupName(){
    return this.groupName;
  }
}

const minJi = new IdolMember('민지', 2004);
// static 으로 선언한 groupName은 보이지 않는다.
console.log(minJi);
console.log('minJi.groupName = ' + minJi.groupName); // undefined
console.log('IdolMember.groupName = ' + IdolMember.groupName);
console.log(IdolMember.getGroupName());


// ② factory constructor
console.log('② factory constructor-----')
// Object type 으로부터 값을 입력받아 클래스 인스턴스 생성
class IdolMember2{
  name;
  year;

  // factory constructor : constructor
  constructor(name, year){
```

```
    this.name = name;
    this.year = year;
  }
  // factory constructor : object
  static fromObject(Object){
    return new IdolMember2(Object.name, Object.year);
  }

  // factory constructor : list
  static fromList(list){
    return new IdolMember2(
      list[0],
      list[1],
    )
  }
}

const gaEul = IdolMember2.fromObject(
  {
    name : '가을',
    year : 2002
  }
);

console.log(gaEul);

const yuJin2 = IdolMember2.fromList(['안유진', 2002]);
console.log(yuJin2);

// factory constructor를 사용하면 다양한 형태의 data를 입력받아
// 객체 인스턴스를 생성하여 활용할 수 있다.
```

4. inheritance

※ 상속이란 : 상속은 객체들 간의 관계를 구축하는 방법이다. super class 또는 부모클래스 등, 기존의 클래스로부터 속성과 동작을 상속받을 수 있다.



위의 모델을 적용하기 위한 코딩하기

▶ 4_inheritance.js

```
class IdolModel{
  name;
  year;
  constructor(name, year){
    this.name = name;
    this.year = year;
  }
}

class FemaleIdolModel extends IdolModel{
  dance(){
    return `${this.name}이 춤을 춥니다.`;
  }
}

class MaleIdolModel extends IdolModel{
  sing(){
    return `${this.name}이 노래를 합니다.`;
  }
}

const yuJin = new FemaleIdolModel('안유진', 2003);
console.log(yuJin);
console.log(yuJin.dance());
console.log(yuJin.name);
```

```
const jiMin = new MaleIdolModel('지민', 1995);
console.log(jiMin);
console.log(jiMin.sing());
console.log(jiMin.year);

const jyp = new IdolModel('박진영', 1971);
console.log(jyp);
console.log(jyp.name);
//console.log(jyp.dance());

console.log('instance 확인 --------------------')
let result = yuJin instanceof IdolModel;
console.log('yuJin instanceof IdolModel = ' + result);
result = yuJin instanceof FemaleIdolModel;
console.log('yuJin instanceof FemaleIdolModel = ' + result);
result = yuJin instanceof MaleIdolModel
console.log('yuJin instanceof MaleIdolModel = ' + result);

console.log('instance 확인 --------------------')
let result = yuJin instanceof IdolModel;
console.log('yuJin instanceof IdolModel = ' + result);
result = yuJin instanceof FemaleIdolModel;
console.log('yuJin instanceof FemaleIdolModel = ' + result);
result = yuJin instanceof MaleIdolModel
console.log('yuJin instanceof MaleIdolModel = ' + result);
-----------------------------------------------------------------------------------------------------------
console.log(jyp.dance());
                    ^


TypeError: jyp.dance is not a function
```

5. super 키워드와 override

▶ 5_super_and_override.js

① super key & 생성자 오버라이딩

```
class IdolModel{
  name;
  year;
  constructor(name, year){
    this.name = name;
    this.year = year;
    }
  sayHello(){
    return `안녕하세요 ${this.name}입니다.`;
  }
}

class FemaleIdolModel extends IdolModel{
  // 자식클래스에만 존재하는 파트(노래? 춤?)
  part;

  // construct를 재정의 한다.
  constructor(name, year, part){
    super(name, year);  // super()는 부모의 생성자를 호출
    this.part = part;
  }

  // 부모클래스의 sayHello() 메서드를 재정의
  sayHello(){
    return `안녕하세요. ${this.name}입니다. 저는 ${this.part}를 맡고 있습니다.`
    // javascrip는 전통적인 OOP가 아니기 때문에 this.name이라고 표현해야 함.
  }

  dance(){
    return `${this.name}이 춤을 춥니다.`;
  }
}

// 테스트
const yuJin = new FemaleIdolModel('안유진', 2003, '보컬');
console.log(yuJin);
console.log(yuJin.sayHello());
```

6. 문제

▶ 6_oop_quiz.js

1) Country 클래스는 나라이름과 나라에 해당하는 아이돌 그룹 정보를 리스트로 갖고 있다.
   (name, idolGroup)
2) IdolGroup 클래스는 아이돌 그룹 이름 정보와 멤버 정보를 리스트로 들고 있다.
   (name, members)
3) Idol 클래스는 아이돌 이름과 출생년도 정보를 갖고 있다.
   (name, year)
4) MaleIdol 클래스는 Idol 클래스와 동일하게 name, year 프로퍼티가 존재한다.
   추가로 sing() 함수를 실행하면 `${이름}이 노래를 합니다.` 라는 스트링을 반환해 준다.
5) FemaleIdol 클래스는 Idol 클래스와 동일하게 name, year 프로퍼티가 존재한다. 추가로 dance() 함수를 실행하면 `${이름}이 춤을 춥니다.`라는 스트링을 반환한다.

map()을 이용하여 위의 내용을 구축하시오.

```
const iveMembers = [
  {name : '안유진', year : 2003},
  {name : '가을', year : 2002},
  {name : '레이', year : 2004},
  {name : '장원영', year : 2004},
  {name : '리즈', year : 2004},
  {name : '이서', year : 2007},
];

const btsMembers = [
  {name : '진', year : 1992},
  {name : '슈가', year : 1993},
  {name : '제이홉', year : 1994},
  {name : 'RM', year : 1994},
];

class Country{
  name;
  idolGroups;
  constructor(name, idolGroups){
    this.name = name;
    this.idolGroups = idolGroups;
  }
}
```

```
class IdolGroup{
  name;
  members;
  constructor(name, members){
    this.name = name;
    this.members = members;
  }
}

class Idol{
  name;
  year;
  constructor(name, year){
    this.name = name;
    this.year = year;
  }
}

class FemaleIdol extends Idol{
  sing(){
    return `${this.name}이 노래를 합니다.`;
  }
}

class MaleIdol extends Idol{
  dance(){
    return `${this.name}이 춤을 춥니다.`
  }
}

// 주어진 두개의 아이돌 배열로 만들기 - map() 이용
const cIveMembers = iveMembers.map((x)=>new FemaleIdol(x.name, x.year));
console.log('map()으로 ive 멤버(배열)를 하나씩 읽어 생성자로 그룹 만들기')
console.log(cIveMembers);

console.log('-------------------------------------');

console.log('map()으로 bts 멤버(배열)를 하나씩 읽어 생성자로 그룹 만들기')
const cBtsMembers = btsMembers.map((x)=>new MaleIdol(x.name, x.year));
console.log(cBtsMembers);
```

```
// 그룹클래스 만들기
console.log('그룹 클래스 만들기');
const iveGroup = new IdolGroup('아이브', cIveMembers);
console.log(iveGroup);

console.log('-------------------------------------');

const btsGroup = new IdolGroup('BTS', cBtsMembers);
console.log(btsGroup);

// 나라 지정해서 아이브와 BTS 넣기
console.log('Country 클래스 만들기');
const korea = new Country('대한민국', [iveGroup, btsGroup]);

console.log('출력 ---------------------------')
console.log(korea);
console.log(korea.idolGroups[0].members[0]);
console.log(cIveMembers[0].sing());

console.log(korea.idolGroups[1].members[1].dance());

// 한번에 작업하기
console.log('----------한번에 작업하기-----------')
const koreanGroup = new Country(
  '대한민국', [
    new IdolGroup(
      '아이브',
      iveMembers.map((x)=> new FemaleIdol(x.name, x.year))
    ),
    new IdolGroup(
      'BTS',
      btsMembers.map((x)=> new MaleIdol(x.name, x.year))
    )
  ]
)

console.log(koreanGroup);
console.log(koreanGroup.idolGroups[0].members[3].sing());
```

3장. 객체의 모든 것

1. 여러가지 객체 선언 방법

▶ 3_all_object → 1_make_a_class.js

1) object를 생성해서 객체 생성 → { key : value }

2) class를 instance화 해서 생성 → OOP

3) function을 사용해서 객체 생성하는 방법

```
// 1) object를 생성해서 객체 생성 → { key : value }
console.log('1) object를 생성해서 객체 생성 → { key : value }');
const yuJin = {
  name : '안유진',
  year : 2003,
};
console.log(yuJin);
console.log('----------------------------------');

// 2) class를 instance화 해서 생성 → OOP
console.log('2) class를 instance화 해서 생성 → OOP');
class Idol{
  name;
  year;
  constructor(name, year){
    this.name = name;
    this.year = year
  }
}
const wonyoung = new Idol('장원영', 2004);
console.log(wonyoung);
console.log('----------------------------------');

// 3) function을 사용해서 객체 생성하는 방법
console.log('3) function을 사용해서 객체 생성하는 방법')
console.log('① this 키워드 사용하기')
// 함수는 생성자가 존재하지 않는다.
// this 키워드를 사용할 때만 new 사용가능

function IdolFunction(name, year){
  this.name = name;
  this.year = year;
}
```

```
const minJi = new IdolFunction('민지', 2004);
console.log(minJi);
console.log('----------------------------------------');
console.log('② new로 인스턴스가 만들어 지도록 조치하기')

function girlGroup(name, memberName){
  if(! new.target){
    return new girlGroup(name, memberName);
  }
  this.name = name;
  this.memberName = memberName;
};

console.log(girlGroup('NewJeans', '다니엘'));
console.log('----------------------------------------');

console.log('③ Arrow 함수로 생성하기')
const girlGroupNewJeans = (name, year) => {
  this.name = name;
  this.year = year;
};

/**
 * ※ 화살표 함수를 사용하면 안되는 곳
 * 1) 화살표 함수로 메소드를 정의하는 것은 피해야 한다.
 * 2) 화살표 함수는 생성자 함수로 사용할 수 없다.
 * 3) addEventListener 함수의 콜백 함수를 화살표 함수로 정의하면
 *    this가 상위 컨택스트인 전역 객체 window를 가리킨다.
 * 4) 클래스의 정의도 피한다.
 */
```

2. Property attribute

▶ 2_property_attribute.js

※ attribute : 변수 생성 시 자동으로 생성되는 것
1) 데이터 프로퍼티 : 키와 값으로 형성된 실직적 값을 갖고 있는 프로퍼티
2) 액세서 프로퍼티 : 자체적으로 값을 갖고 있지는 않지만 다른 값을 가져오거나 설정할 때 호출되는 함수로 구성된 프로퍼티. 예를 들면, getter와 setter 두가지가 있다.

```
const wonyoung = {
  name : '장원영',
```

```
  year : 2004,
};

console.log(Object.getOwnPropertyDescriptor(wonyoung, 'name'));

------------------------------------------------------------------------------------------------------------
{ value: '장원영', writable: true, enumerable: true, configurable: true }
```

▷ property attribute

① value : 실제 프로퍼티 값

② writable : 값을 수정할 수 있는지 여부(false : 수정불가)

③ enumerable : 열거 가능 여부. for~in Loop를 사용할 수 있는 지 여부

④ configurable : property attribute(value, writable, enumerable)가 재정의가 가능한지 여부를 판단.

- false : property 상태나 attribute 변경이 금지
- 단, writable이 true 인 경우 값 변경과 writable을 변경하는 것은 가능.

```
const wonyoung = {
  name : '장원영',
  year : 2004,
};

// 모든 property attribute 출력
console.log('1) 모든 property attribute 출력--------------------------');
console.log(Object.getOwnPropertyDescriptor(wonyoung, 'name'));
console.log('------------------------------------------------------');

// data property and access property
console.log('2) Data Property and Access Property-------------------');
const yuJin = {
  name : '안유진',
  year : 2003,
  age : 0,
  get getAge(){
    return new Date().getFullYear() - this.year;
  },
  set setAge(age){
    this.age = age;
  }
}
console.log('yuJin.age = ' + yuJin.age);
```

```
console.log(Object.getOwnPropertyDescriptor(yuJin, 'age'));

console.log('▷ 나이 계산해서 얻어오기 → yuJin.setAge = yuJin.getAge;')
yuJin.setAge = yuJin.getAge;
console.log(Object.getOwnPropertyDescriptor(yuJin, 'age'));
console.log('-------------------------------------------------');

// data property의 재정의
console.log('3) Data Property 재정의 → ① 간단한 추가 방법--------------');
yuJin.height = 172;
console.log(yuJin);



console.log('3) Data Property 재정의 → ② 직접 추가 방법--------------');
console.log(wonyoung);
Object.defineProperty(wonyoung, 'weight', {
  value : 45, writable : true, enumerable : true, configurable : false
});
console.log(wonyoung);

console.log('3) Data Property 재정의 → ③ writable → false ----------');
Object.defineProperty(yuJin, 'height', {writable : false});
yuJin.height = 180;
// 값이 변경되지 않지만 오류도 발생하지 않는다.
console.log(yuJin);

console.log('4) Data Property 재정의 → ④ enumerable → false ---------');
console.log(Object.keys(wonyoung));
for(let key in wonyoung){
  console.log(key);
};
console.log('enumerable → false로 변경 후---------------------------');
Object.defineProperty(wonyoung, 'name', {enumerable : false});
for(let key in wonyoung){
  console.log(key);
};
```

3. Immutable Object(불변 객체)

▶ 3_immutable.js

```
const yuJin = {
  name : '안유진',
```

```
  year : 2003,
  age : 0,
  get getAge(){
    return new Date().getFullYear() - this.year;
  },
  set setAge(age){
    this.age = age;
  }
}
yuJin.setAge = yuJin.getAge;
console.log(yuJin);
console.log('----------------------------------------------------');
console.log('1) Extensible : 객체 확장 가능 여부 확인')
console.log('▷ 확장가능? ' + Object.isExtensible(yuJin));
console.log('▷ 확장 못하게 하기');
Object.preventExtensions(yuJin);
console.log('▷ 확장테스트 ------------------------------------');
yuJin.height = 172;
console.log(yuJin);
console.log('▷ 확장가능? ' + Object.isExtensible(yuJin));

console.log('----------------------------------------------------');
console.log('2) Seal : 봉인하기  ---- 가장 많이 사용 함.(default : false')
console.log('▷ 봉인되었음?(추가/삭제 불가) ' + Object.isSealed(yuJin));
console.log('▷ 봉인하기');
Object.seal(yuJin);
console.log('▷ 봉인테스트 ------------------------------------');
yuJin.height = 172;
delete yuJin.name;
console.log(yuJin);
console.log('▷ 봉인되었음? ' + Object.isSealed(yuJin));

console.log('----------------------------------------------------');
console.log('3) Freeze : 얼려버기기(읽기전용) - 많이 사용 함.(default : false')
console.log('▷ 읽기 전용(Freeze)? ' + Object.isFrozen(yuJin));
console.log('▷ 얼려버리기');
Object.freeze(yuJin);
console.log('▷ 다 얼었나? ------------------------------------');
yuJin.height = 172;
yuJin.name = 'An Yu Jin';
```

```
console.log(yuJin);
console.log('▷ Frozon? ' + Object.isFrozen(yuJin));
```

4. Prototype

▶ 4_1_basic_prototype.js

1) __proto === prototype

```
console.log('-----------------------------------------------------');
console.log('1) 빈 클래스 생성하고 살펴보기')

const testObj = {};
console.log(testObj.__proto__);
// [Object: null prototype] {}


// __proto__ : 모든 객체에 존재하는 프로퍼티이다.
// 클래스의 상속에서 부모클래스에 해당되는 값이다.

function IdolModel(name, year){
  this.name = name;
  this.year = year;
}

console.log(IdolModel.prototype);    // 결과 :  {}

console.log('-----------------------------------------------------');
console.log('2) 내부의 모든 숨겨진 prototype 살펴보기')
console.dir(IdolModel.prototype, {showHidden:true,});
/* <ref *1> {
  [constructor]: [Function: IdolModel]
   {
    [length]: 2,
    [name]: 'IdolModel',
    [arguments]: null,
    [caller]: null,
    [prototype]: [Circular *1]
  }
} */

console.log('-----------------------------------------------------');
console.log('3) prototype과 IdolModel → circular reference')
```

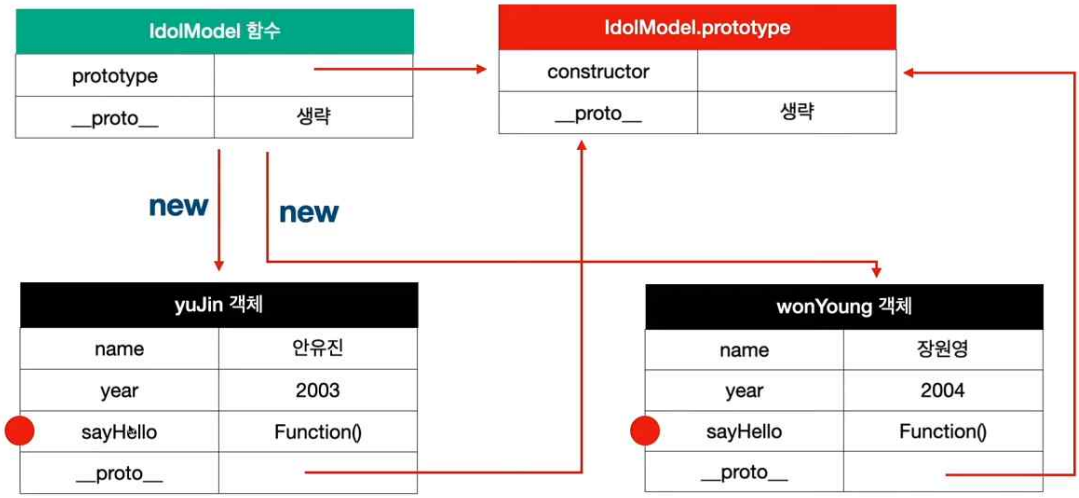
```
console.log(IdolModel.prototype.constructor === IdolModel);
console.log(IdolModel.prototype.constructor.prototype === IdolModel.prototype);

console.log('-----------------------------------------------------');
console.log('4) 부모객체 여부확인해 보기')
const yuJin = new IdolModel('안유진', 2003);
console.log(yuJin.__proto__);
let result = yuJin.__proto__ === IdolModel.prototype
console.log('yuJin.__proto__ === IdolModel.prototype : ' + result);

result = IdolModel.__proto__ === Function.prototype
console.log('IdolModel.__proto__ === Function.prototype : ' + result);

result = Function.prototype.__proto__ === Object.prototype
console.log('Function.prototype.__proto__ === Object.prototype : ' + result);
console.log('-----------------------------------------------------');

console.log(yuJin.toString());
result = yuJin.toString() === Object.prototype.toString()
console.log('yuJin.toString() === Object.prototype.toString() : ' + result);

console.log('-----------------------------------------------------');
```

2) 프로토타입의 활용

▶ 4_2_app_prototype.js

```
console.log('-----------------------------------------------------');
console.log('1) 일반적 메서드 생성 및 활용')

function IdolModel2(name, year){
  this.name = name;
  this.year = year;
  this.sayHello = function(){
    return `${this.name}이 인사를 합니다.`
  }
}

const yuJin = new IdolModel2('안유진', 2003);
console.log(yuJin.sayHello());
```

```
const wonYoung = new IdolModel2('장원영', 2004);
console.log(wonYoung.sayHello());

console.log(yuJin.sayHello() === wonYoung.sayHello());

// 특정 메서드가 어느 인스턴스 클래스에서 나왔는지 확인하는 법
console.log(yuJin.hasOwnProperty('sayHello'));  //true

//  기능이 동일하나 클래스마다 고유한 영역에 생성되므로 메모리가 낭비됨.
```

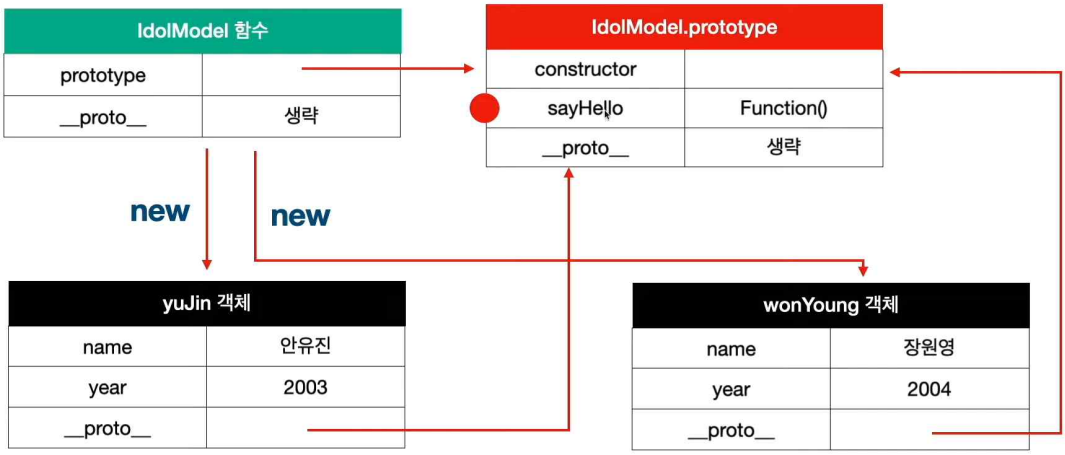


▶ 공유프로퍼티 만들기

```
console.log('-----------------------------------------------------');
console.log('1) 일반적 메서드 생성 및 활용')

function IdolModel2(name, year){
   this.name = name;
   this.year = year;
   this.sayHello = function(){
      return `${this.name}이 인사를 합니다.`
   }
}

const yuJin = new IdolModel2('안유진', 2003);
console.log(yuJin.sayHello());

const wonYoung = new IdolModel2('장원영', 2004);
console.log(wonYoung.sayHello());

console.log(yuJin.sayHello === wonYoung.sayHello);
```

```
// 특정 메서드가 어느 인스턴스 클래스에서 나왔는지 확인하는 법
console.log(yuJin.hasOwnProperty('sayHello'));   //true

//  기능이 동일하나 클래스마다 고유한 영역에 생성되므로 메모리가 낭비됨.

console.log('-----------------------------------------------------');
console.log('2) 공유 프로퍼티 만들기')

function IdolModel3(name, year){
  this.name = name;
  this.year = year;
};

IdolModel3.prototype.sayHi = function(){
  return `${this.name}이 인사를 합니다.`;
};

const yuJin3 = new IdolModel3('안유진', 2003);
console.log(yuJin3.sayHi());

const wonYoung3 = new IdolModel3('장원영', 2004);
console.log(wonYoung3.sayHi());

console.log(yuJin3.sayHi === wonYoung3.sayHi);   //true

// 특정 메서드가 어느 인스턴스 클래스에서 나왔는지 확인하는 법
console.log(yuJin3.hasOwnProperty('sayHi'));   //true

console.log('-----------------------------------------------------');
console.log('3) 공유 프로퍼티 Override')

function IdolModel4(name, year){
  this.name = name;
  this.year = year;

  this.sayHi = function(){
    return `안녕하세요 인스턴스 메서드입니다.`;
  };
};
```

```
IdolModel4.prototype.sayHi = function(){
  return `안녕하세요 프로토타입 메서드입니다.`;
};

const yuJin4 = new IdolModel4('안유진', 2003);
console.log(yuJin4.sayHi());
```



5. Closure

▶ 5_closure.js

```
console.log('-----------------------------------------------------');
console.log('1) Lexical Scope')

let numberThree = 3

function functionOne(){
  var numberThree = 100;
  functionTwo();
}

function functionTwo(){
  console.log(numberThree);
}

functionOne();

// 자바스크립는 선언된 위치가 상위 스코프를 정하게 된다.
// 이를 Lexical Scope 라고 한다.

console.log('-----------------------------------------------------');
console.log('2) Closure')
// 하위함수가 상위함수 보다 더 오래 살아있는 경우를 말한다.

function getNum(){
  var num = 5;

  function innerGetNum(){
    console.log('innerGetNum()');
    return num;
  }
  console.log('getNum()');
  return innerGetNum();
}

// console.log(num);  // error : ReferenceError: num is not defined

console.log(getNum());
```

```
// 위 상황에서는 innerGetNum 함수는 getNum 함수보다 오래
// 살아남지 않는다.

function GetNumber(){
  let num = 10;
  function InnerGetNumber(){
    console.log('InnerGetNumber()');
    return num;
  }
  console.log('GetNumber()');
  return InnerGetNumber;
}

const runner = GetNumber();
console.log(runner);
console.log(runner());

console.log('-----------------------------------------------------');
console.log('3) Closure 사용 예')

function cacheFunction(){
  let count = 10 * 10;
  function innerCacheFunction(inputNum){
    return count * inputNum;
  }
  return innerCacheFunction;
}

const calc = cacheFunction();
console.log(calc(3));
console.log(calc(20));
```