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

▶ 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();
```