02. JavaScript 기본

자료형
    Js의 모든 자료형
    null & undifined 비교
연산자와 반복문
    논리 연산자
    nullish 병합 연산자 '??'
    반복문 label
함수
    기본
    콜백 함수
    함수 표현식 VS 함수 선언문

자료형

Js의 모든 자료형

- `숫자형`
    - 정수, 부동 소수점 숫자 등의 숫자를 나타낼 때 사용. 한계는 ±2의 53승
- `bigint`
    - 길이 제약 없이 정수 표현 가능, 끝에 `n`을 붙여서 표현
    - ES2020부터 도입
- `문자형`
    - 빈 문자열이나 글자들로 이뤄진 문자열을 나타낼 때 사용
- `boolean형`
    - `true`, `false`를 나타낼 때 사용
- `null`
    - 존재하지 않는(nothing), 비어 있는(empty), 알 수 없는(unknown) 값인 `null` 를 나타내는 자료형
    - 숫자형으로 변환 시, `0`
- `undefined`
    - 할당되지 않은 값인 `undefined` 나타내는 자료형
    - 숫자형으로 변환 시, `NaN`
- `객체형`
    - 복잡한 데이터 구조를 표현할 때 사용
    - 객체 제외 자료형은 `원시 자료형`으로 문자, 숫자, t/f등 하나만 표현 가능
    - 반면 객체는 데이터 컬렉션이나 복잡한 개체(entity)를 표현 가능
- `Symbol형`
    - `객체`의 고유 식별자를 만들 때 사용

```
typeof undefined // "undefined"
typeof 0 // "number"
typeof 10n // "bigint"
typeof true // "boolean"
typeof "foo" // "string"
typeof Symbol("id") // "symbol"

typeof Math // "object", Math는 수학 연산을 제공하는 내장 객체이므로 "object"가 출력
typeof null // "object", null은 객체가 아니지만, Js의 오류로 인해 "object"가 출력
typeof alert // "function", "function"형은 존재하지 않음. 함수는 객체형이지만, Js의 오류로 인해 "fu
```

null & undifined 비교

```
alert( null > 0 );  // false, null은 숫자형 변환 시 0으로 변환
alert( null == 0 ); // false, null은 `==`에서는 숫자형으로 변환되지 않음
alert( null >= 0 ); // true

alert( undefined > 0 ); // false, undefined는 숫자형 변환 시 NaN으로 변환
alert( undefined < 0 ); // false
alert( undefined == 0 ); // false, undefined는 `==`에서는 숫자형으로 변환되지 않음

alert( null == undefined ); // true, null과 undefined는 동등 비교 시 일치
```

- `null`과 `undefined` 는 서로를 비교할 때만 `true` 를 반환
- 이 외의 값과 동등 비교 `==` 를 할 경우 반드시 `false`를 반환
- 고로 `===`를 사용하는 것을 권장
- `undefined`나 `null`이 될 가능성이 있는 변수가 `<`, `>`, `<=`, `>=`의 피연산자가 되지 않도록 주의

연산자와 반복문

논리 연산자

- `||`(OR)
  - 첫 번째 truthy값을 찾는 연산자
  - truthy값의 피연산자 등장 시, 해당 피연산자 반환
  - 모든 피연산자가 falsy라면, 마지막 피연산자 반환
- `&&`(AND)
  - 첫 번째 falsy을 찾는 연산자
  - falsy값의 피연산자 등장 시, 해당 피연산자 반환
  - 모든 피연산자가 truthy라면, 마지막 피연산자 반
- `!`(NOT)
  - 피연산자를 boolean형(`true / false`)으로 변환 후 이 값의 역을 반환
  - `!!`를 사용 시, 피연산자를 boolean형으로 변환 가능

```
// OR 연산자
alert( null || 2 || undefined ); // 2
alert( alert(1) || 2 || alert(3) ); // alert는 값을 반환하지 않아 undefined 반환, alert(1) 실

alert( 1 && null && 2 ); // null
alert( alert(1) && alert(2) ); // 1 출력 후 undefined 출력

alert( null || 2 && 3 || 4 ); // &&연산자의 우선순위가 ||보다 높기에 우선 실행, 2 && 3 = 3, 따라서
```

nullish 병합 연산자 '??'

- ES2020에서 도입
- 첫 번째 정의된 값을 찾는 연산자

```
const x = a ?? b // a가 null도 아니고 undefined도 아니면 a, 그 외에는 b
const y = (a !== null && a !== undefined) ? a : b;
alert(x === y) // true
```

- `||`와의 차이점

```
let height = 0;

alert(height || 100); // 100, ||는 truthy값 반환
alert(height ?? 100); // 0, ??는 정의된(defined)값 반환
```

- 안정성 관련 이슈로 인해 `&&`나 `||`와 함께 사용 불가
  - 함께 사용하고 싶다면 괄호 사용 필요

```
let x = 1 && 2 ?? 3; // SyntaxError: Unexpected token '??'
let x = (1 && 2) ?? 3; // 제대로 동작
```

반복문 label

- *레이블(label)* 은 반복문 앞에 콜론과 함께 쓰이는 식별자
- 반복문 안에서 `break <labelName>`문을 사용하면 레이블에 해당하는 반복문을 빠져나옴

```
labelname: for (let i = 0; i < 3; i++) {
  for (let j = 0; j < 3; j++) {
    let input = prompt(`(${i},${j})의 값`, '');
    // 사용자가 아무것도 입력하지 않거나 Cancel 버튼을 누르면 두 반복문 모두를 빠져나옵니다.
    if (!input) break labelname; // (*)
  }
}
alert('완료!');
```

함수

기본

- JavaScript에서는 함수도 객체로 표현
- 매개변수
  - 매개변수에 인수를 전달하지 않으면, 그 값에 undefined를 할당한 채로 호출된다.

```
function showMessage(from, text) {
  alert( from + ': ' + text );
}

showMessage("name", "Ann"); // name: Ann 출력
showMessage("name"); // name: undefined 출력, 에러 발생하지 않음
```

- 반환 값
  - `return`문이 없거나 `return` 지시자만 있는 함수는 `undefined`를 반환
- 이름 짓기
  - 함수 이름은 함수가 어떤 동작을 하는지 설명
  - 함수가 어떤 동작을 하는지 축약해서 설명해 주는 **동사를 접두어**로 붙여 이름을 만들 것
    - `"get…"` – 값을 반환
    - `"calc…"` – 무언가를 계산
    - `"create…"` – 무언가를 생성
    - `"check…"` – 무언가를 확인하고 boolean값을 반환

콜백 함수

- 함수의 인수로 전달되고, 그 함수의 필요에 따라 "나중에 호출(called back)"되는 함수

```
function ask(question, yes, no) {
  if (confirm(question)) yes()
  else no();
}

function showOk() {
  alert( "동의하셨습니다." );
}

function showCancel() {
  alert( "취소 버튼을 누르셨습니다." );
}

// showOk와 showCancel가 callback함수로 사용됨
ask("동의하십니까?", showOk, showCancel);
```

함수 표현식 VS 함수 선언문

▼ 문(Statement) VS 식(Expression)

```
let y = x + 5; // x + 5는 산술 식 (Expression), 전체는 변수 선언문 (Statement)
```

- 문(Statement)
  - 특정 작업을 수행하는 코드의 실행 단위, 프로그램의 흐름을 제어하거나 동작을 명시
  - 문은 단독으로 실행될 수 있으며, 보통 특정 동작을 지시
  - 조건문, 반복문 등
- 식(Expression)
  - 값을 생성하거나 계산하는 코드의 부분
  - 단독으로 사용 가능하나, 보통 문 내에서 사용
  - 값 식, 변수 참조 식, 산술 식, 함수 호출 식 등

- 함수 표현식
  - 실제 실행 흐름이 해당 함수에 도달했을 때 함수 생성
  - 따라서 실행 흐름이 함수에 도달했을 때부터 해당 함수 사용 가능
- 함수 선언문
  - **함수 선언문이 정의되기 전에도 호출 가능,** 호이스팅이 원인

```
console.log(표현식(2, 3)); // undefined
console.log(선언문(2, 3)); // 5

// 함수 표현식
const 표현식 = function(a, b) { // function 키워드를 사용해 익명 함수 생성 후 값을 변수에 할당
  return a + b;
};

// 함수 선언문
function 선언문(a, b) { // function 키워드로 함수 선언
  return a + b;
}
```