# 자바스크립트 기본

## 함수

- 유사한 동작을 하는 코드가 여러 곳에 있을 경우 묶어 주기
- 함수는 프로그램을 구성하는 **주요 구성 요소(building block)**로 함수를 이용하면 중복 없이 **유사한 동작을 하는 코드를 여러 번 호출 가능**
- 언어 자체에서 제공하는 빌트인(built-in) 함수(parseInt, eval 등등)와 브라우저에서 제공하는 빌트인 함수(alert, prompt, confirm 등등)가 존재함
- 언어에서 제공하는 빌트인 함수외에도 직접 **사용자 함수를 정의**하여 호출 가능

### 함수 선언

```
function 함수이름(파라미터) {
  // ...함수 본문...
}
```

- 함수 정의 및 호출

```
function showMessage() {
  alert( '안녕하세요!' );
}

// 함수 호출
showMessage();
```

- 함수 내에서 선언한 변수인 **지역 변수(local variable)는 함수 안에서만 접근 가능**

```
function showMessage() {
  let message = "안녕하세요!"; // 지역 변수

  // 함수 내부에서만 message 접근 가능
  alert( message );
}
```

- 함수 내부에서 외부 영역에 정의된 외부 변수(outer variable)에 접근 가능 (값 수정도 가능)

```
let userName = 'John';

function showMessage() {
  let message = 'Hello, ' + userName;
  alert(message);

  // 외부 변수값 수정
  userName = "Bob";
}
```

- 매개변수(parameter)를 이용하면 임의의 데이터를 함수 안에 전달 가능
    - 함수에 전달된 매개변수는 복사된 후 함수의 지역변수가 됨

```
// 두 개의 파라미터 존재(from, text)
function showMessage(from, text) {
  alert(from + ': ' + text);
}

showMessage('Ann', 'Hello!');
```

- 매개변수에 값을 전달하지 않으면 그 값은 undefined가 됨
    - 다른 언어와 달리 함수에 정의된 **파라미터 값을 모두 전달하지 않아도 정상동작**함을 유의!

```
// 이 경우 text에는 undefined가 할당됨
showMessage("Ann");
```

- 기본값 할당 가능
    - 매개변수 오른쪽에 대입 기호(=)을 붙이고 undefined 대신 설정하고자 하는 기본값을 써주기

```
// 만약 text 값이 전달되지 않을 경우 "no text given" 값을 기본값으로 사용
function showMessage(from, text = "no text given") {
  alert( from + ": " + text );
}
```

## 매개변수 기본값 설정할 수 있는 또 다른 방법

- **ES6 이전 문법으로 코드 작성** 시 활용할 수 있는 방법
- 매개변수를 undefined와 비교

```
function showMessage(text) {
    if (text === undefined) {
        text = '빈 문자열';
    }

    alert(text);
}
```

- OR 연산자 사용

```
function showMessage(text) {
    text = text || '빈 문자열';
    // ...
}
```

- null 병합 연산자(nullish coalescing operator) 사용

```
function showCount(count) {
    alert(count ?? "unknown");
}

// 0은 falsy 값이지만 값은 엄연히 존재하므로 0 출력
showCount(0);
// null => unknown 출력
showCount(null);
// undefined => unknown 출력
showCount();
```

### 반환 값

- **return 지시자를 이용**하여 함수 반환값 설정 가능

```
function sum(a, b) {
    return a + b;
}
```

- return문이 없거나 return 지시자만 있는 함수는 **undefined를 반환**함을 유의! (함수는 모두 값을 반환한다고 봐도 무방)

```
function doNothing() { /* empty */ }

// true
alert( doNothing() === undefined );
```

- return문을 만나면 함수는 즉시 종료됨

## 함수 표현식

- **함수 선언문(Function Declaration)** 방식으로 함수 만들기

```
function sayHi() {
  alert( "Hello" );
}
```

- **함수 표현식(Function Expression)** 을 사용해서 함수 만들기
    - 함수를 생성하고 **변수에 값을 할당하는 것처럼 함수가 변수에 할당**됨
- 자바스크립트에서 함수는 값이므로 **함수를 값처럼 취급**할 수 있음

```
// 함수 표현식을 사용해서 함수 만들기
// 함수를 만들고 그 함수를 변수 sayHi에 할당
let sayHi = function() {
    alert( "Hello" );
};

// 함수도 값처럼 취급되므로 대입하여 함수 복사 가능 (sayHi 함수에 괄호가 없음(=함수 호출 아님)을 유의!)
let func = sayHi;

// "Hello" 경고문 출력
func();
```

- 함수는 값이므로 **함수의 인자값으로 함수 전달 가능**

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

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

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

// 사용법: 함수 showOk와 showCancel가 ask 함수의 인수로 전달됨
ask("동의하십니까?", showOk, showCancel);
```

- 함수 ask의 인수, showOk와 showCancel은 **콜백 함수(callback function) 또는 콜백**이라고 불리움
    - 왜냐하면 함수를 직접 호출하는 것이 아니고 **파라미터로 전달한 함수가 나중에 호출(called back)**되기 때문

```
// 다음과 같이 사용 가능
ask(
    "동의하십니까?",
    function() { alert("동의하셨습니다."); },
    function() { alert("취소 버튼을 누르셨습니다."); }
);
```

- 위와 같이 이름 없이 선언한 함수는 **익명 함수(anonymous function)**라고 불리움

## 함수 표현식 vs 함수 선언문

- **차이1** : 문법
    - 함수를 선언하는 생김새가 다름

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

// 함수 표현식
let sum = function(a, b) {
    return a + b;
};
```

- **차이2** : 함수 생성 시점
    - 함수 표현식은 실제 **실행 흐름이 해당 함수에 도달했을 때 함수를 생성**
    - 함수 선언문은 **함수 선언문이 정의되기 전에도 호출** 가능

```
// 이 시점에 함수 호출 가능
sayHi("John");

// 함수 호출 구문 뒤에 함수 정의해도 사용 가능
function sayHi(name) {
    alert( `Hello, ${name}` );
}
```

함수 표현식은 함수 표현식 구문을 해석한 시점 이후부터 호출 가능

```
// 호출 불가 (Uncaught ReferenceError: sayHi is not defined)
sayHi("John");

// 함수 표현식을 이용한 값 할당 이후 호출 가능
let sayHi = function(name) {
    alert( `Hello, ${name}` );
}

// 이 시점에서는 호출 가능
sayHi("John");
```

- 특별히 함수 표현식을 써야 할 이유가 있지 않다면 **함수 선언문 방식으로 함수를 정의하는 것이 권장**됨

# 화살표 함수 기본 (*)

- 함수 표현식보다 **단순하고 간결한 문법으로 함수를 선언**하는 방식 => **화살표 함수(arrow function)** 사용
- 화살표 함수 선언 문법
    - 화살표 오른쪽의 **표현식이 평가되고 반환**됨
        - 구문(statement)과 표현식(expression)의 차이 생각해보기

```
let func = (arg1, arg2, ...argN) => expression
```

- 활용 사례

```
let sum = (a, b) => a + b;
```

- **전달할 파라미터가 하나**밖에 없다면 파라미터를 감싸는 괄호를 생략 가능

```
// 전달할 파라미터가 한 개이므로 괄호 생략 가능
let double = n => n * 2;

// 물론 써줘도 무방함
let double = (n) => n * 2;
```

- 전달할 파라미터가 하나도 없을 경우 괄호만 쓰기 (이때는 괄호 생략 불가)

```
let sayHi = () => alert("안녕하세요!");
```

- 화살표 함수를 이용해서 더 간결한 코드 작성 가능
    - **함수 본문이 한 줄이거나, 표현식으로 대체 가능한 간단한 함수**는 화살표 함수를 사용해서 만드는 것이 편리

```
let age = prompt("나이를 알려주세요.", 18);

let welcome = (age < 18) ?
    () => alert('안녕') :
    () => alert("안녕하세요!");
```

- 평가해야 할 표현식이나 구문이 여러 개인 복잡한 함수를 화살표 함수로 정의할 경우 **중괄호 블록과 명시적인 return 문을 이용**

```
// 중괄호는 본문 여러 줄로 구성되어 있음을 알려줌
let sum = (a, b) => {
  let result = a + b;
  // 중괄호를 사용했다면, return문을 이용하여 결괏값을 반환해야 함
  return result;
};
```