# JavaScript Advanced - (1) 화살표 함수와 그 특징

화살표 함수(arrow function)란 **함수를 간단하게 표현할 수 있는 ES6 문법**입니다 . 일반 자바스크립트(`function()`)와 비슷한 방식으로 동작하지만 몇 가지 분명한 차이점이 있습니다.

## 1. 화살표 함수의 기본 문법

화살표 함수는 다음과 같은 형태로 작성합니다.

```
let functionName = (parameter1, parameter2, ...) => expression;
```

- `let` 또는 `const` 키워드로 변수를 선언하고 그 변수에 화살표 함수를 할당합니다.
- 괄호(`()`) 안에 매개변수들을 쉼표(`,`)로 구분하여 씁니다.
- 화살표(`=>`) 뒤에 반환할 값을 나타내는 **표현식(expression)**을 씁니다.
- 이렇게 작성하면 인자 arg1…argN 를 받는 함수 func 이 만들어집니다.

화살표 함수도 익명함수입니다. 따라서 이름을 지정하지 않고 익명으로 사용합니다. 따라서 화살표 함수를 변수에 할당하거나 다른 함수의 인자로 전달할 수 있습니다.

예를 들어, 다음과 같이 `sayHello`라는 변수에 화살표 함수를 할당할 수 있습니다.

```
let sayHello = (name) => console.log("Hello, " + name);
```

## 2. 화살표 함수의 특징

화살표 함수는 일반 자바스크립트(function())와 비슷한 방식으로 동작하지만 몇 가지 분명한 차이점이 있습니다.

- 화살표 함수는 **자신의 `this`가 없습니다**. 대신 화살표 함수를 둘러싸는 렉시컬 범위(lexical scope)의 `this`가 사용됩니다. 즉, 화살표 함수 내부에서 this를 참조하면 상위 스코프의 `this` 값을 그대로 가져옵니다.
- 화살표 함수는 `arguments` **객체가 없습니다**. 대신 나머지 매개변수(rest parameters)를 사용할 수 있습니다. 즉, 화살표 함수 내부에서 `arguments` 객체를 참조하면 상위 스코프의 `arguments` 객체 값을 그대로 가져옵니다.
- 화살표 함수는 `new` **연산자로 인스턴스를 생성할 수 없습니다**. 즉, 화살표 함수는 생성자(`constructor`)로 사용할 수 없습니다. 이는 화살표 함수가 `prototype` 프로퍼티가 없기 때문입니다.
- 화살표 함수는 **중복된 매개변수 이름을 허용하지 않습니다**. 즉, `strict mode`와 `non-strict mode` 모두에서 동일한 규칙을 따릅니다.

## 3. 예제 코드

다음은 일반 자바스크립트(`function()`)와 화살표함수(`() => {}`)의 차이점을 보여주는 예제 코드입니다.

### 3-1. `this` 바인딩

다음은 일반 자바스크립트에서 `this` 바인딩을 보여주는 예제 코드입니다.

```
let user = {
  name: "Alice",
  sayHi: function() {
    console.log("Hi, I'm " + this.name);
  }
};

user.sayHi(); // Hi, I'm Alice
```

여기서 `user.sayHi()` 메소드 내부에서 `this` 는 `user` 객체에 바인딩됩니다. 따라서 `this.name`은 `"Alice"`가 됩니다.

그런데 만약 다음과 같이 `setTimeout()`함수 안에 `user.sayHi()` 메소드를 넣으면 어떻게 될까요?

```
let user = {
  name: "Alice",
  sayHi: function() {
    setTimeout(function() {
      console.log("Hi, I'm " + this.name);
    }, 1000);
  }
};

user.sayHi(); // Hi, I'm undefined
```

결과적으로 `"undefined"`가 출력됩니다. 이는 setTimeout()함수 내부에서 this가 전역 객체에 바인딩되기 때문입니다. 따라서 `this.name`은 전역 객체의 `name` 프로퍼티를 참조하게 되는데, 이는 정의되지 않았으로 `"undefined"`가 됩니다.

이러한 문제를 해결하기 위해서는 다음과 같이 `this`를 다른 변수에 저장하거나 `bind()` 메소드를 사용할 수 있습니다.

```
let user = {
  name: "Alice",
  sayHi: function() {
    let self = this; // this를 self에 저장
    setTimeout(function() {
      console.log("Hi, I'm " + self.name); // self.name을 사용
    }, 1000);
  }
};

user.sayHi(); // Hi, I'm Alice

let user2 = {
  name: "Bob",
  sayHi: function() {
    setTimeout(function() {
      console.log("Hi, I'm " + this.name); // this를 bind()로 바인딩
    }.bind(this), 1000); // bind(this)를 사용
  }
};

user2.sayHi(); // Hi, I'm Bob
```

하지만 화살표 함수를 사용하면 이러한 문제가 발생하지 않습니다. 왜냐하면 화살표 함수는 자신의 `this`가 없고 상위 스코프의 `this`를 그대로 가져오기 때문입니다.

```
let user = {
  name: "Alice",
  sayHi: function() {
    setTimeout(() => { // 화살표 함수 사용
      console.log("Hi, I'm " + this.name); // 상위 스코프의 this 사용
    }, 1000);
  }
};

user.sayHi(); // Hi, I'm Alice
```

여기서 화살표 함수 내부에서 this 는 user 객체에 바인딩됩니다. 따라서 `this.name`은 `"Alice"`가 됩니다.

## 3-2. arguments 객체

다음은 일반 자바스크립트에서 arguments 객체를 보여주는 예제 코드입니다.

```
function sum() {
  let result = 0;
  for (let i = 0; i < arguments.length; i++) {
    result += arguments[i];
  }
  return result;
}

console.log(sum(1, 2, 3)); // 6
console.log(sum(4, 5)); // 9
```

여기서 `sum()` 함수는 매개변수를 정의하지 않았지만 `arguments` 객체를 통해 전달된 모든 인자에 접근할 수 있습니다. 따라서 `sum()` 함수는 인자의 개수에 상관없이 모든 인자의 합을 반환할 수 있습니다.

그런데 만약 다음과 같이 화살표 함수로 `sum()` 함수를 정의하면 어떻게 될까요?

```
let sum = () => {
  let result = 0;
  for (let i = 0; i < arguments.length; i++) {
    result += arguments[i];
  }
  return result;
}

console.log(sum(1, 2, 3)); // Uncaught ReferenceError: arguments is not defined
```

결과적으로 에러가 발생합니다. 이는 화살표 함수 내부에서 `arguments` 객체가 정의되지 않기 때문입니다. 따라서 화살표 함수 내부에서 `arguments` 객체를 참조하면 상위 스코프의 `arguments` 객체 값을 그대로 가져옵니다.

이러한 문제를 해결하기 위해서는 다음과 같이 나머지 매개변수(rest parameters)를 사용할 수 있습니다.

```
let sum = (...args) => { // 나머지 매개변수 사용
  let result = 0;
  for (let i = 0; i < args.length; i++) {
    result += args[i];
} return result; }

console.log(sum(1, 2, 3)); // 6 console.log(sum(4, 5)); // 9
```

여기서 `sum()` 함수는 나머지 매개변수를 통해 전달된 모든 인자를 배열로 받을 수 있습니다. 따라서 `sum()` 함수는 인자의 개수에 상관없이 모든 인자의 합을 반환할 수 있습니다.

### 3-3. 생성자

다음은 일반 자바스크립트에서 생성자를 보여주는 예제 코드입니다.

```
function Person(name) {
  this.name = name;
}

let alice = new Person("Alice");
console.log(alice.name); // Alice
```

여기서 `Person()` 함수는 name 매개변수를 받아서 this.name에 할당하는 생성자입니다. 따라서 `new Person("Alice")`로 alice 객체를 생성할 수 있습니다.

그런데 만약 다음과 같이 화살표 함수로 `Person()` 함수를 정의하면 어떻게 될까요?

```
let Person = (name) => {
  this.name = name;
}

let alice = new Person("Alice"); // Uncaught TypeError: Person is not a constructor
```

결과적으로 에러가 발생합니다. 이는 화살표 함수는 `new` 연산자로 인스턴스를 생성할 수 없기 때문입니다. 즉, 화살표 함수는 생성자로 사용할 수 없습니다. 이는 화살표 함수가 `prototype` 프로퍼티가 없기 때문입니다.

### 3-4. 중복된 매개변수 이름

다음은 일반 자바스크립트에서 중복된 매개변수 이름을 보여주는 예제 코드입니다.

```
function add(x, x) { // non-strict mode에서 가능
  return x + x;
}

console.log(add(1, 2)); // 4

"use strict";

function add(x, x) { // strict mode에서 불가능
```