JavaScript

참조 자료형 01

함수

Function

- 참조 자료형에 속하며 모든 함수는 Fuction object

함수 정의

함수 구조

```
function name ([param[, param, [..., param]]]) {
    statements
    return value
}
```

- function 키워드
- 함수의 이름
- 함수의 매개변수
- 함수의 body를 구성하는 statements
- return 값이 없다면 undefined를 반환

함수 정의

선언식(function declaration)

```
function funcName () {
    statements
}
```

- 호이스팅 됨
- 코드의 구조와 가독성 면에서는 표현식에 비해 장점이 있음

```
add(1,2) // 3

function add (num1, num2) {
    return num1 + num2
}
```

표현식(function expression)

```
const funcName = function () {
    statements
}
```

- 호이스팅 되지 않음
  - 변수 선언만 호이스팅되고 함수 할당은 실행 시점에 이루어짐

```
sub(2, 1) // ReferenceError: Cannot access 'sub' before initialization

const sub = function (num1, num2) {
    return num1 - num2
}
```

- 함수 이름이 없는 '익명 함수'를 사용할 수 있음

함수 표현식 사용을 권장하는 이유

- 예측 가능성
  - 호이스팅의 영향을 받지 않아 코드의 실행 흐름을 더 명확하게 예측할 수 있음
- 유연성
  - 변수에 할당되므로 함수를 값으로 다루기 쉬움
- 스코프 관리
  - 블록 스코프를 가지는 let이나 const와 함께 사용하여 더 엄격한 스코프 관리가 가능

매개변수

1. 기본 함수 매개변수 (Default function parameter)

- 전달하는 인자가 없거나 undefined가 전달될 경우 이름 붙은 매개변수를 기본값으로 초기화

```
const greeting = function (name = 'Anonymous') {
    return `Hi ${name}`
}

greeting() // Hi Anonymous
```

2. 나머지 매개변수 (Rest parameters)

- 임의의 수의 인자를 '배열'로 허용하여 가변 인자를 나타내는 방법
- 작성 규칙
  - 함수 정의 시 나머지 매개 변수는 하나만 작성할 수 있음
  - 나머지 매개변수는 함수 정의에서 매개변수 마지막에 위치해야 함

```
const myFunc = function (param1, param2, ...restPrams) {
    return [param1, param2, restPrams]
}

myFunc(1, 2, 3, 4, 5) // [1, 2, [3, 4, 5]]
myFunc(1, 2) // [1, 2, []]
```

매개 변수와 인자 개수가 불일치 할 때

- 매개 변수 개수 > 인자 개수

▶ 누락된 인자는 undefined로 할당

```
const threeArgs = function (param1, param2, param3) {
    return [param1, param2, param3]
}

threeArgs() // [undefined, undefined, undefined]
threeArgs(1) // [1, undefined, undefined]
threeArgs(2, 3) // [2, 3, undefined]
```

- 매개 변수 개수 < 인자 개수

▶ 초과 입력한 인자는 사용하지 않음

```
const noArgs = function () {
    return 0
}
noArgs(1, 2, 3) // 0

const two Args = function (param1, param2) {
    return [param1, param2]
}
twoArgs(1, 2, 3) // [1, 2]
```

Spread Syntax

'...' (전개 구문)

- 배열이나 문자열과 같이 반복 가능한 항목을 펼치는 것 (확장, 전개)
- 전개 대상에 따라 역할이 다름

  ▶ 배열이나 객체의 요소를 개별적인 값으로 분리하거나 다른 배열이나 객체의 요소를 현재 배열이나 객체에 추가하는 등

전개 구문 활용처

1. 함수와의 사용
    1. 함수 호출시 인자 확장
    2. 나머지 매개변수 (압축)
2. 객체와의 사용
3. 배열과의 사용

전개 구문 활용

- 함수와의 사용
    1. 인자 확장 (함수 호출 시)

```
<!-- spread-syntax.html -->

function myFunc(x, y, z) {
    return x + y + z
}

let numbers = [1, 2, 3]

console.log(myFunc(...numbers)) // 6
```

    2. 나머지 매개 변수 (함수 선언 시)

```
<!-- spread-syntax.html -->

function myFunc2(x, y, ...restArgs) {
    return [x + y + restArgs]
}

console.log(myFunc2(1, 2, 3, 4, 5)) // [1, 2, [3, 4, 5]]
console.log(myFunc2(1, 2)) // [1, 2, []]
```

화살표 함수 표현식

Arrow function expressions

- 함수 표현식의 간결한 표현법

화살표 함수로 변경 결과

```
const arrow = function (name) {
    return `hellow, ${name}`
}
```

⇓

```
const arrow = name => `hello, ${name}`
```

화살표 함수 작성 과정

1. function 키워드 제거 후 매개변수와 중괄호 사이에 화살표(=>) 작성
2. 함수의 매개변수가 하나 뿐이라면, 매개변수의 '()' 제거 가능 (단, 생략하지 않는 것을 권장)
3. 함수 본문의 표현식이 한 줄이라면, '{}'와 'return' 제거 가능

```
const arrow1 = function (name) {
    return `hello ${name}`
}

// 1. function 키워드 삭제 후 화살표 작성
const arrow2 = (name) => { return `hellow, ${name}` }

// 2. 인자의 소괄호 삭제 (인자가 1개일 경우에만 가능)
const arrow3 = name => { return `hellow, ${name}` }

// 3. 중괄호와 return 삭제 (함수 본문이 return을 포함한 표현식 1개일 경우에만 가능)
const arrow4 = name => `hellow, ${name}`
```

참고

화살표 함수 심화

```
// 1. 인자가 없다면 () or _ 로 표시 가능
const noArgs1 = () => 'No args'
const noArgs2 = _ => 'No args'


// 2-1. object를 return 한다면 return 을 명시적으로 작성해야 함
const returnObject1 = () => { return { key: 'value' } }

// 2-2. return을 작성하지 않으려면 객체를 소괄호로 감싸야 함
const returnObject2 = () => ({ key: 'value' })
```