05. 함수

함수는, 특정 코드를 하나의 명령으로 실행 할 수 있게 해주는 기능입니다.

예를 들어서, 우리가 특정 값들의 합을 구하고 싶을 때는 다음과 같이 코드를 작성합니다.

```
const a = 1;
const b = 2;
const sum = a + b;
```

한번, 이 작업을 함수로 만들어보겠습니다.

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

const sum = add(1, 2);
console.log(sum);
```

결과는 3이 됩니다.

함수를 만들 때는 `function` 키워드를 사용하며, 함수에서 어떤 값을 받아올지 정해주는데 이를 파라미터(매개변수)라고 부릅니다.

함수 내부에서 `return` 키워드를 사용하여 함수의 결과물을 지정 할 수 있습니다.

추가적으로, `return` 을 하게 되면 함수가 끝납니다. 만약 다음과 같이 코드가 작성된다면, return 아래의 코드는 호출이 안됩니다.

```
function add(a, b) {
  return a + b;
  console.log('호출이 되지 않는 코드!');
}

const sum = add(1, 2);
console.log(sum);
```

연습

함수 사용을 연습해봅시다.

Hello, name!

name 이라는 파라미터를 넣으면 콘솔에 'Hello name!' 이라는 결과를 출력하는 코드를 작성해봅시다.

```
function hello(name) {
  console.log('Hello, ' + name + '!');
}
hello('velopert');
```

결과물은 다음과 같습니다.

```
"Hello, velopert!"
```

console.log 를 하게 될 때 우리는 문자열을 조합하기 위해서 + 연산자를 사용했습니다. 이렇게 문자열을 조합 할 때 + 를 사용해도 큰 문제는 없지만, 더욱 편하게 조합을 하는 방법이 있습니다. 바로, ES6 의 템플릿 리터럴 (Template Literal)이라는 문법을 사용하는 것 입니다.

> **ES6 가 뭔가요?**
>
> ES6 는 ECMAScript6 를 의미하며, 자바스크립트의 버전을 가르킵니다. ES6는 2015년에 도입이 되었습니다. ES6 는 ES2015 라고 불리기도 합니다. 그리고 2015년 이후에 계속해서 새로운 자바스크립트 버전이 나오고 있습니다. ES7(ES2016) ES8(ES2017) ES9(ES2018) ES10(ES2019).. 새로운 자바스크립트 버전이 나올때마다 새로운 문법이 소개됩니다.
>
> 브라우저 버전에 따라 지원되는 자바스크립트 버전이 다릅니다. 하지만, 보통 웹 개발을 할 때에는 Babel 이라는 도구를 사용하여 최신 버전의 자바스크립트가 구버전의 브라우저에서도 실행되도록 할 수 있습니다. (정확히는, 최신버전 자바스크립트를 구버전 형태로 변환하는 작업을 거칩니다.)

템플릿 리터럴을 사용하여 구현을 해봅시다.

```
function hello(name) {
  console.log(`Hello, ${name}!`);
}
hello('velopert');
```

점수를 성적등급으로 변환하기

이번에는 점수가 주어졌을 때 A, B, C, D, F 등급을 반환하는 함수를 만들어봅시다.

```
function getGrade(score) {
  if (score === 100) {
    return 'A+';
  } else if (score >= 90) {
    return 'A';
  } else if (score === 89) {
    return 'B+';
  } else if (score >= 80) {
    return 'B';
  } else if (score === 79) {
    return 'C+';
  } else if (score >= 70) {
    return 'C';
  } else if (score === 69) {
    return 'D+';
  } else if (score >= 60) {
    return 'D';
  } else {
    return 'F';
  }
}

const grade = getGrade(90);
console.log(grade);
```

grade 가 90 일 때에는 결과가 A 가 됩니다.

화살표 함수

함수를 선언하는 방식 중 또 다른 방법은 화살표 함수 문법을 사용 하는 것 입니다.

```
const add = (a, b) => {
  return a + b;
};

console.log(add(1, 2));
```

`function` 키워드 대신에 `=>` 문자를 사용해서 함수를 구현했는데요, 화살표의 좌측에는 함수의 파라미터, 화살표의 우측에는 코드 블록이 들어옵니다.

그런데, 만약에 위와 같이 코드 블록 내부에서 바로 return 을 하는 경우는 다음과 같이 줄여서 쓸 수도 있습니다.

```
const add = (a, b) => a + b;
console.log(add(1, 2));
```

아까 만들었던 getGrade 함수처럼 여러 줄로 구성되어있는 경우에는 코드 블록을 만들어야합니다.

```
const getGrade = score => {
  if (score === 100) {
    return 'A+';
  } else if (score >= 90) {
    return 'A';
  } else if (score === 89) {
    return 'B+';
  } else if (score >= 80) {
    return 'B';
  } else if (score === 79) {
    return 'C+';
  } else if (score >= 70) {
    return 'C';
  } else if (score === 69) {
    return 'D+';
  } else if (score >= 60) {
    return 'D';
  } else {
    return 'F';
  }
};
const grade = getGrade(90);
console.log(grade);
```

화살표 함수와 일반 function 으로 만든 함수와의 주요 차이점은 화살표 함수에서 가르키는 this 와 function 에서 가르키는 this 가 서로 다르다는 건데요, 이에 대한 자세한 내용은 나중에 알아보도록 하겠습니다