함수

하나의 키워드에 프로그램 로직을 함축한 형태.

명령어를 만드는 과정으로 이해할 수 있다.

이렇게 만들어진 명령어는 **여러 번 재사용이 가능**하다.

#01. 수학적 접근

일반적인 수학의 함수는 하나의 수식을 대변하는 특정 키워드를 의미한다.

아래의 함수가 있을 때 f라는 키워드로 아래와 같이 x+1이라는 수식을 **재사용**할 수 있으며 이는 아래와 같이 조건값에 따라 각각 다른 결과를 반환하게 된다.

> $f(x) = x + 1$

즉, 아래와 같이 수식이 재활용 된다.

> $f(10) = 10 + 1 = 11$

#02. 자바스크립트에서의 함수 개요

1) 함수 정의하기

function 키워드를 명시하고 함수의 이름을 지정한 후 괄호`()`를 명시한다.

그 뒤에 구문을 그룹화 하기 위한 블록 `{}`을 갖는다.

```
function 함수이름() {
    ... 명령어 ...
}
```

2) 함수의 호출

함수는 정의하는 것 만으로는 아무런 동작을 하지 않는다.

반드시 정의된 함수를 실행시키는 명령을 내려야 하는데 이를 **함수를 호출**한다고 한다.

```
함수이름();
```

#03. 함수가 실행되는데 필요한 조건값

필요에 따라 함수가 실행되는데 필요한 조건값을 정의할 수 있는데, 이를 **파라미터**라고 한다.

1) 수학에서의 의미

> $f(x) = x + 1$

함수 `f(x)`가 자신이 수행되는데 필요한 조건값 **x**를 갖는다. 수학에서는 이를 매개변수 혹은 파라미터라고 부른다.

2) 프로그램에서의 의미

파라미터 정의 하기

프로그램의 함수도 자신이 실행하는데 필요한 조건값을 함수 정의 과정에서 괄호`()`안에 명시할 수 있으며 이를 파라미터라고 부른다.

```
function 함수이름(파라미터) {
    ... 파라미터를 활용한 프로그램 구문 ...
}
```

파라미터를 전달하는 함수 호출

파라미터가 정의된 함수를 호출할 경우 괄호`()`안에 조건에 맞는 값을 전달해야 한다.

```
함수이름(값);
```

3) 다중 파라미터

필요한 조건 값이 여러 개인 경우 콤마(`,`)로 구분하여 나열할 수 있다.

정의된 파라미터들은 원칙적으로 함수 호출시 해당 값들을 전달해야 한다.

다중 파라미터 정의 하기

프로그램의 함수도 자신이 실행하는데 필요한 조건값을 함수 정의 과정에서 괄호`()`안에 명시할 수 있으며 콤마로 구분하여 여러 개를 정의할 수 있다.

```
function 함수이름(파라미터1, 파라미터2, ... 파라미터n) {
    ... 파라미터를 활용한 프로그램 구문 ...
}
```

다중 파라미터를 갖는 함수의 호출

파라미터가 여러개 인 경우 함수를 호출할 때 정의되어 있는 순서대로 값을 전달해야 한다.

```
함수이름(값1, 값2, ... 값n);
```

4) 함수 호출시 파라미터 생략하기

파라미터를 요구하는 함수라 하더라도 호출시에 필요 없는 값은 마지막 파라미터부터 순차적으로 생략 가능함.

값이 전달되지 않은 파라미터는 **undefined**로 식별된다.

5) 파라미터의 기본값 정의

함수 호출시 값이 전달되지 않는 경우를 대비하여 파라미터에 기본값을 정의할 수 있다.

```
function 함수이름(파라미터1=기본값1, 파라미터1=기본값2, ..., 파라미터n=기본값n) {
    ...
}
```

#03. 리턴

함수가 자신이 만들어낸 결과값을 자신을 호출한 위치로 되돌려 주는 것.

1) 수학에서의 리턴

수학에서의 함수도 자신이 호출된 위치로 결과값을 되돌려 준다.

> $f(x) = x + 1$ $y = f(5)$ ← 이 자리에 f(5)의 결과값이 반환된다. $y = 5 + 1$ $y = 6$

2) JS에서의 리턴

결과값을 리턴하는 함수 정의하기

함수를 구성하는 블록{} 안에서 **return** 키워드를 사용하여 값을 전달한다.

```
function 함수이름(파라미터1, 파라미터2, ... 파라미터n) {
    ... 파라미터를 활용한 프로그램 구문 ...
    return 돌려줄_값;
}
```

리턴값을 다른 변수에 할당

리턴값을 갖는 함수는 그 결과를 다른 변수에 대입할 수 있다.

```
const 변수 = 함수이름(파라미터1, 파라미터2, ... 파라미터n);
```

리턴값을 활용한 새로운 수식 구성

함수의 리턴값을 새로운 수식에 포함시킬 수 있다.

```
const 변수 = 100 + 함수이름(파라미터1, 파라미터2, ... 파라미터n);
```

리턴값을 조건식으로 활용하기

함수의 리턴값과 비교식을 구성하여 조건문으로 사용할 수 있다.

```
if (함수이름(파라미터1, 파라미터2, ... 파라미터n) > 0) {
   ...
}
```

논리값을 리턴하는 함수의 조건식 활용

함수의 리턴값이 논리값(`true` / `false`)이라면 그 자체를 조건으로 사용할 수 있다.

```
if (함수이름(파라미터1, 파라미터2, ... 파라미터n)) {
   // 함수의 리턴값이 true인 경우 실행됨
}

if (!함수이름(파라미터1, 파라미터2, ... 파라미터n)) {
   // 함수의 리턴값이 false인 경우 실행됨
}
```

리턴값을 활용한 반복문

함수의 리턴값을 활용하여 반복문의 조건식을 구성할 수 도 있다.

```
for (let i = 0; i<함수이름(파라미터1, 파라미터2, ... 파라미터n); i++) {
   ...
}
```

함수의 실행 중단

함수가 실행되는 도중 **return** 키워드를 만나면 그 즉시 실행을 종료한다.

#04. 함수의 또 다른 형태

1) 함수를 변수에 대입하기

자바스크립트는 함수 자체가 객체 형태이기 때문에 다른 변수에 참조시켜 사용할 수 있다.

```
function 함수이름(...) {
   ...
}

const 변수 = 함수이름;
```

> 여기서는 객체를 특수한 기능을 갖는 변수의 한 종류로 이해합시다.

2) 익명함수

다른 변수에 참조시킬 목적으로 함수를 정의할 때 부터 이름 없이 정의하는 형태

전체적인 정의 형태가 **대입문**이므로 블록을 구성하는 중괄호{}뒤에 세미콜론(;)이 위치해야 한다.

```
const 변수 = function(...) {
   ...
};
```

3) 콜백함수

파라미터로 전달되기 위해 사용되는 함수.

어떤 함수 A가 동작하는 과정 중에서 일부에 대한 처리가 상황에 따라 다르게 구성되어야 할 경우, 그 부분을 함수 형태로 묶어 파라미터로 받도록 할 수 있다.

4) 재귀함수

함수가 처리로직 내부에서 자기 자신을 호출하는 형태.

재귀호출은 마지막에 종료 조건을 명시하지 않는다면 무한루프에 빠지게 된다. 그러므로 재귀호출을 구현할 때 가장 먼저 처리해야 할 것은 종료조건을 명시하는 것이다.

```
function 함수이름(파라미터1, 파라미터2, ... 파라미터n) {
    if (종료조건) {
        return 값;
    } else {
        // 자기 스스로를 호출한다.
        함수이름(파라미터1, 파라미터2, ... 파라미터n);
    }
}
```

팩토리얼 구하기

팩토리얼의 수식을 분석해 본다면 다음과 같이 정의할 수 있다.

> $f(x) = x * f(x-1)$ (단, x가 1 이하인 경우 1)

#05. 화살표 함수 (arrow function)

ES6 버전부터 새롭게 추가된 구문 형식으로 기존의 익명함수 문법을 간단하게 축약하여 사용할 수 있다.

1) 익명함수와의 비교

익명함수 형태로 정의한 함수.

익명함수는 변수에 함수를 대입하는 형태로 정의한다.

```
const 변수 = function(파라미터1, 파라미터2, ... 파라미터n) {
    ... 처리로직 ...
};
```

화살표 함수 형태로 정의한 동일한 함수

function 키워드가 삭제되고 파라미터를 전달하기 위한 소괄호`()`와 블록을 구성하기 위한 중괄호`{}` 사이에 `=>` 기호가 추가된다.

```
const 변수 = (파라미터1, 파라미터2, ... 파라미터n) => {
    ... 처리로직 ...
};
```

2) 화살표 함수 살펴보기

파라미터가 하나만 존재하는 경우

파라미터를 감싸는 소괄호 `()` 를 생략할 수 있다. 파라미터가 없거나 두 개 이상인 경우는 소괄호를 `()` 생략할 수 없다.

```
const 변수 = 파라미터 => {
    ... 처리로직 ...
};
```

처리 로직이 한 줄만 포함되는 경우

리턴을 위한 구문 한 줄만 포함하는 익명함수를 가정해 보자.

```
const 변수 = function(파라미터1, 파라미터2, ... 파라미터n) {
    return 리턴값;
};
```

위와 같은 형태를 화살표 함수로 변경하고자 할 때, 블록을 위한 {}를 생략하고 return 키워드도 생략할 수 있다.

```
const 변수 = (파라미터1, 파라미터2, ... 파라미터n) => 리턴값;
```