

JS-function

| | |
|------|--------------|
| ☰ 태그 | |
| 📅 날짜 | @2023년 5월 2일 |

일반 함수 : 함수 선언식과 함수 표현식

화살표 함수 문법 총정리

arguments 와 가변인자

JS 에서 함수를 정의하는 방법은 크게 두가지로 나눌 수 있다.

1. function 키워드를 사용하는 방법
2. 화살표 함수를 사용하는 방법

일반함수 - function 키워드

1. 함수 선언식
2. 함수 표현식

함수 선언식

```
function main() {  
    console.log("hello")  
};  
  
main();
```

함수 선언식으로 작성된 함수의 특징은 호이스팅이 된다.

호이스팅이란 함수의 선언을 코드의 최상단으로 끌어올린다는 것이다.

위 코드처럼 작성된 main함수는 호이스팅이 되기 때문에 코드상에서 호출하기 전에 선언문 위에 main을 호출해도 정상 작동한다.

함수 선언식으로 함수를 정의할 때는 함수의 이름을 필수적으로 넣어줘야한다.

이렇게 함수의 이름을 필수적으로 넣어줘야하기 때문에 함수 선언식 방법으로는 이름이 없는 함수인 익명 함수를 만들수가 없다.

익명 함수를 만들때는 함수 선언식이 아니라 함수 표현식을 사용하면 된다.

함수 표현식

```
const main = function() {  
  console.log("hello")  
}  
  
main()
```

함수 표현식은 함수 선언식이란 문법적으로 굉장히 비슷하지만 큰 차이점으로 함수 표현식 방법으로 함수를 정의하면 이름이 없는 함수인 익명함수를 만들 수 있다는 것이다.

함수 표현식 또한 위처럼 function키워드를 사용해서 함수를 정의한다.

하지만 함수 선언식과는 다르게 함수의 이름을 생략해 줄 수 있다.

```
이름이 있는 네임드 function  
const main = function main() {  
  console.log("hello")  
}  
main()  
  
이름이 없는 익명 함수  
// 익명 함수는 main이라는 변수에 할당을 했기 때문에 변수의 이름을 함수로 사용하여 호출할 수 있다.  
const main = function() {  
  console.log("hello")  
}  
main()
```

함수 표현식은 호이스팅이 되지 않는다.

화살표 함수는 es6버전에서 등장했다.

함수를 정의할 때 사용하는 또 다른 문법이다.

이름 그대로 화살표를 사용한다.

화살표 함수는 기존의 함수 표현식을 훨씬 더 간결한 문법으로 작성할 수 있게 해주는 장점이 있다.

화살표 함수

```
const main = () => {  
  console.log("hello")  
}  
main()
```

화살표 함수는 항상 이름이 없는 익명 함수 이기 때문에 함수 표현식에서 했던 방법처럼 함수를 변수에 할당하여 원할때 마다 함수를 호출할 수 있다.

화살표 함수는 함수의 생김새에 따라 몇가지 기호를 생략해서 더욱 간결하게 만들어 줄 수 있다.

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

위 처럼 코드가 단 한줄 밖에 없는 경우 함수를 감싸는 중괄호와 리턴 키워드를 생략해 줄 수 있다.

```
const add = (a, b) => a + b  
add()
```

중괄호가 생략된 화살표 함수는 따로 리턴 키워드를 작성하지 않아도 자동으로 이 표현식으로부터 평가된 결과를 리턴해 준다.

```
const print = text(매개변수) => console.log(text)  
print("hi")
```

화살표함수의 매개변수가 단 하나라면 매개변수의 소괄호도 생략해 줄수 있다.

```
const print = (text1, text2(매개변수)) => console.log(text1)
```

```
print("hi")
```

매개변수가 두개라면 소괄호를 생략할 수 없다.

매개변수가 하나라도 없을 경우 또한 소괄호를 생략할 수 없다.

```
const getObject = () => {  
  return { name : "별코딩" }  
}  
getObject()
```

객체 하나를 리턴하는 함수..

이 함수도 코드가 한 줄이기 때문에 마찬가지로 함수를 감싸는 중괄호와 리턴 키워드를 생략할 수 있다.

하지만 리턴 키워드도 중괄호로 감싸져 있기 때문에 리턴 키워드를 생략해버리면 문법에 혼란이 온다.

그래서 객체를 리턴할 때는 객체를 소괄호로 감싸주면 된다.

```
const getObject = () => ({ name : "별코딩" })  
getObject()
```

화살표 함수의 가장 큰 장점은 간결함이다.

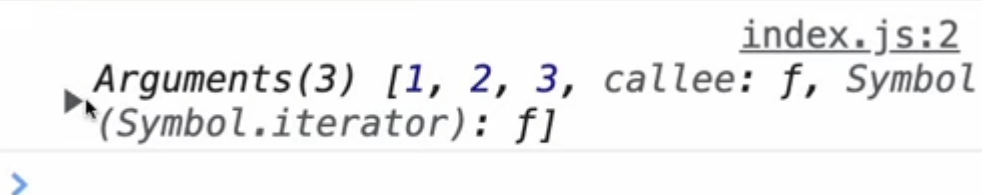
하지만 화살표 함수와 일반 함수의 차이점은 간결함에서만 끝나지 않는다.

그 다음으로 볼 차이점은 함수의 arguments이다.

```
function main() {  
  console.log(arguments)  
}
```

```
main(1, 2, 3)
```

function 키워드로 정의한 함수의 경우에는 매개변수로 따로 명시해주지 않아도 arguments라는 변수를 암묵적으로 전달 받는다.



우리가 전달해준 1, 2, 3이라는 인자가 배열 형태로 들어가 있는걸 볼 수 있다.

이 arguments 변수는 함수가 전달 받은 인자를 담고 있는 배열 형태 객체이다.

정확히 배열은 아니지만 배열 형태의 객체이다.

우리가 전달해준 인자를 배열 형태로 접근 할 수 있다.

```
console.log( arguments[0]);
```

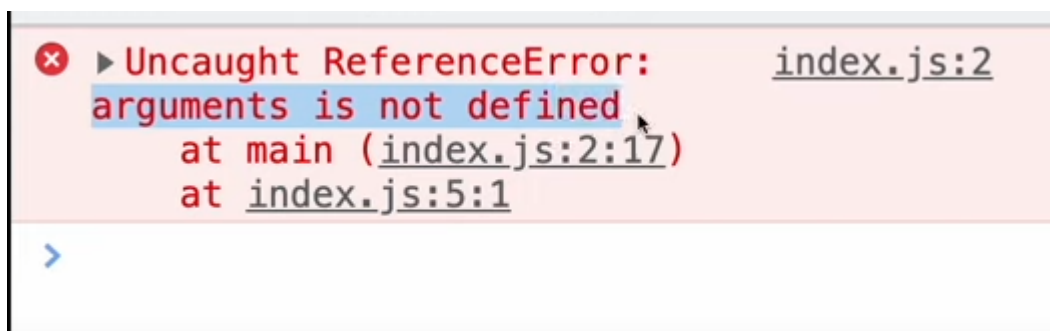
```
console.log( arguments[1]);
```

```
console.log( arguments[2]);
```

arguments 변수는 인자의 개수가 정해지지 않는 가변 인자가 전달되는 함수를 쓸 때 유용하게 사용될 수 있다.

하지만 화살표 함수는 일반 함수랑은 다르게 이 arguments라는 변수를 전달 받지 않는다.

```
const main = () => {  
  console.log(arguments)  
}  
main(1,2,3)
```



위 처럼 arguments에 접근하면 js는 arguments를 찾을 수 없다고 에러나 나오게 된다.

화살표 함수는 일반 함수 처럼 가변 인자를 처리 할수 없는건 아니다.

화살표 함수에서 가변 인자를 처리하고 싶다면 매개변수로 아무거나 넣은 다음에 앞에 ...을 붙여서 나머지 매개변수 구문이라는 사용해 주면된다.

```
const main = (...args) => {  
  console.log(args)  
}  
main (1,2,3)
```

► (3) [1, 2, 3]

index.js:2

