

javascript 기초 (함수)

📎 자료	<u>Javascript</u>
☰ 구분	Javascript

function, arrow function

javascript에서 함수를 표현하는 방식은 여러가지가 있다. 크게 2가지

“함수 선언식” and “함수 표현식”

기본적인 형태는 다음과 같다.

```
# 함수 선언식
function kakao(age,name){
    console.log(age,name);
}
kakao(99, 'minho');

# 함수 표현식
const naver=function(age,name){
    console.log(age,name);
}
naver(41, 'minho');
```

함수 선언식은 평소에 우리가 사용하는 함수와 유사하다.

위에 코드를 보면 함수 표현식에서는 변수를 하나 선언하고
그 변수에 함수를 할당 해 주는 형식이다.

함수표현식에서는 “function” 키워드가 생략 가능 함으로 아래와 같이 표현도 가능하다.

```
# 함수 표현식
const naver=(age,name){
    console.log(age,name);
}
naver(41, 'minho');
```

함수를 표현하는 두가지 방식 중에 가장 큰 차이점은
함수 선언식은 ‘호이스팅’이 발생하며
함수 표현식은 ‘호이스팅’이 발생하지 않는다는 것이다.

그리고 Air BnB 코딩 컨벤션 (스타일가이드) 에서 권장하는 것은 “함수 표현식”이다. 그런데
이 함수 표현식을 조금 더 간결하게 사용 할 수 있다.

바로 화살표함수 (Arrow Function) 를 이용하면 가능하다.

Arrow function이 어떻게 생겼는지 먼저 살펴보자.

```
// arrow function

const naver=age=>{console.log(age)}
naver(41);
```

Arrow function 의 예시로 함수 안에 매개변수가 1개 일 때,
화살표 함수가 어떻게 생겼는지 한번 작성해 보았다. 간결해 보이지 않는가?

이번에는 함수의 매개변수가 2개 있을 때를 보자.
그리고 “화살표 함수”를 → “함수 표현식”으로 바꿔보기도 하겠다.

```
// Arrow function
const google=(age,name)=>age+10;

// 함수 표현식 (함수 표현식에서 function은 생략이 가능하다)
const google=function(age,name){
  return age+10
}

console.log(google(40, 'minho'))
```

Arrow function 에서 매개변수가 1개이면 소괄호() 생략이 가능하다. 그러나
매개변수가 1개 이상이면 소괄호를 넣어 주어야 한다.
만약에 매개변수가 아예 없는 경우에는

빈 소괄호 () 또는 언더바 _ 로 대체 가능하다.

함수를 arrow function 으로 바꾸는 방식은 다음과 같다

1. 함수표현식의 function 대신 매개변수 옆에 화살표를 적어 주었다.
2. 매개변수가 1개라면 괄호를 () 생략 하는것이 가능하다.
3. 매개변수가 없다면 빈 괄호 () 또는 언더바 _ 로 표시 하기도 한다.
4. 만약에 함수안의 코드가 1줄 이라면 중괄호 {} 생략 가능하다.
그리고 그 1줄의 코드에 return이 들어간다면 return 키워드도 생략이 가능하다.

arrow function 예를 한번 더 보자.

```
// arrow function

const sayHello = (name, age) => `안녕, ${name}. 나이는 ${age}세`;
```

위의 코드를 이해하기 쉽게 굳이 늘려 보자면 아래 코드와 같을 것이다.

```
const sayHello = (name, age) => {
  return `안녕, ${name}. 나이는 ${age}세`;
}

const result = sayHello("minho", 39);
console.log(result); // 안녕, minho. 나이는 39세
```

정리:

- 변수 선언하듯 `const` 를 썼고, `=` 를 통해 값을 집어넣음
- function 키워드 사라짐
- `=>` 이라는 화살표가 생김

[중요]

JavaScript 에서는 “함수”도 “object” 타입을 갖는 하나의 참조형 “변수”이다.

따라서, 함수는 다른 함수의 “매개변수(parameter)” 또는 “리턴값” 으로도 쓰인다.

뿐만 아니라, 객체의 “속성의 값”으로도 쓰인다.

아래 코드는 객체 안의 함수(메서드)를 사용해서 간단한 계산기가 되는 코드로 예를 들어 보겠다.

```
const myCal = {  
  add: (a, b) => a + b,  
  subtract: (a, b) => a - b,  
  multiply: (a, b) => a * b,  
  devide: (a, b) => a / b,  
};  
  
console.log(myCal.add(5, 2)); // 7
```

추가적으로 한 가지 새로운 함수 형태를 살펴보자.

이름은 “즉시 실행 함수” 이다. 즉시 실행 함수는 딱 한번만 사용하고 사라지는 함수다. 기본 품은 다음과 같다.

```
(function (변수) {  
  console.log("a")  
})(값)
```

한 번만 사용하고 사라질 함수라서 함수 이름이 없다.

예를 한 가지만 더 살펴보자.

```
(function(num) {console.log(num ** 3) })(2)
```

1. 소괄호를 열고
2. function 을 써주고 (num) 매개변수를 적어준다.
3. { 함수 코드 작성 } 후에 맨 마지막에
4. 매개변수에 들어 갈 (인자값)을 적어 두면 된다.

즉시 실행 함수는 딱 한 번만 실행하기 위한 함수로써 초기화 코드 부분에 많이 사용된다.
그 이유는 변수(함수)를 전역으로 너무 많이 선언되는 것을 피하기 위해 서다.
함수를 선언하기 위해서 전역변수를 사용하지 않아도 되기 때문에 많은 변수로 인한
코드충돌을 피할 수 있다.

그러나 주의 사항이 있다. 즉시 실행 함수를 사용 시에는
직전 문장을 마친 후에는 세미콜론(;)을 반드시 적어 줘야 한다. 그렇지 않으면 에러가 난다.

자 이렇게 함수를 살펴 보았다면
반복문과 상당히 비슷 한 forEach문 그리고, 함수관련 메서드도 몇 가지 살펴보자.

forEach문을 사용하는 것은 for문과 같다. 그러나 차이점이 있다면
forEach문에서는 break, continue 을 사용할 수 없다. 이는 다른 언어에서도 마찬가지다.

```
const num=[8,5,2,1]
```

- num 배열의 값을 forEach 문을 사용해서 출력 해 보겠다.

```
num.forEach(ele=>console.log(ele));
```

Arrow Function과 forEach 문을 사용해서 배열값을 출력 해 보았다.

- 함수 함수선언식과 forEach 문을 사용 할 수도 있다. 예를 보자면

```
num.forEach(function test(ele){console.log(ele)});
```

- 사실 우리가 잘 알고있는 for문을 그냥 사용 한다면 아래 코드와 같을 것이다.

```
for(let x=0;x<num.length;x++){
  console.log(num[x]);
}
```

위의 코드들을 모두 살펴보면 `forEach` 문을 사용하면서 `ArrowFunction`을 사용하는 것이 가장 간결하고 간편해 보인다는 것을 확인 할 수 있다.

몇 가지 더 예를 보자.

```
const menus = ["짜장면", "짬뽕", "탕수육"];

menus.forEach((menu) => console.log(`${menu} 좋아!`));

// 짜장면 좋아!
// 짬뽕 좋아!
// 탕수육 좋아!
```

`forEach` 는 단 하나의 파라미터를 가진다. 그 파라미터는 콜백함수가 된다. (콜백 함수에 대한 설명은 뒤에 있으니깐 일단은 넘어가자.)

`menus` 의 각각을 `menu` 로 받고, 배열 각각을 순회하면서 정해진 동작을 한다.

만약, 인덱스가 필요하다면 다음처럼 구현 가능하다.

```
const menus = ["짜장면", "짬뽕", "탕수육"];

menus.forEach((menu, idx) => console.log(`${idx} : ${menu} 좋아!`));

// 0 : 짜장면 좋아!
// 1 : 짬뽕 좋아!
// 2 : 탕수육 좋아!
```

콜백함수의 두번째 파라미터로 `idx` 를 받으면 된다. 이름이 반드시 `idx` 일 필요는 없으며, `index` 로 이름을 지어도 잘 동작한다.

자 이제 `forEach` 문 말고도 다른 `method`에 대해서도 살펴보자.

- `num` 배열의 값에 2를 곱한 값으로 구성되는 새로운 배열을 만들어 보겠다.

```
const num=[8,5,2,1]
```

그냥 for문을 사용 했다면 아래 코드와 같을 것이다.

```
let num2=[]
for(let x=0;x<num.length;x++){
  num2.push(num[x]*2)
}
console.log(num2)
```

그러나 만약에 “map” 이라는 help method를 사용하면 더 간결하게 표현 할 수 있겠다.
코드는 다음과 같다.

```
const t=num.map(ele=>ele*2);
console.log(t)
```

`map` 은 `forEach` 와는 다르게, 리턴값을 모아서 새 배열을 만든다.

```
const menus = ["짜장면", "짬뽕", "탕수육"];

const newMenus = menus.map((menu, idx) => `${idx} : ${menu} 좋아!`);
console.log(newMenus);

// ['0 : 짜장면 좋아!', '1 : 짬뽕 좋아!', '2 : 탕수육 좋아!']
```

둘의 사용은 다음으로 구분 지을 수도 있겠다.

- `forEach` : 단순히 배열을 돌고싶을때
- `map` : 배열의 리턴값을 모아서 새 배열을 만들고 싶을 때

이번에는 filter 라는 method를 살펴보자.

- 이번에는 const num=[8,5,2,1] 배열에서 짝수만 새로운 배열에 담아 보겠다.

그냥 for문을 사용한다면 다음과 같을 것이다.

```
num2=[]
for(let x=0;x<num.length;x++){
  if(num[x]%2===0){
    num2.push(num[x])
  }
}
console.log(num2)
```

만약에 filter 함수를 사용한다면 다음과 같을 것이다.

```
const f=num.filter(ele=>ele%2===0)
console.log(f)
```

filter 라는 help method를 통해서 return 값이 참인 것만! 반환 받았다.

```
const menus = ["짜장면", "짬뽕", "탕수육"];

const newMenus = menus.filter((menu) => menu.length > 2);

console.log(newMenus);

// ['짜장면', '탕수육']
```

[연습문제] 를 풀어보자.

```
// [연습문제] type이 'fruit'인 것들의 name을 출력해보자
const products = [
  { name: 'cucumber', type: 'vegetable' },
  { name: 'banana', type: 'fruit' },
  { name: 'carrot', type: 'vegetable' },
  { name: 'apple', type: 'fruit' },
]
```

정답은 본 PDF 파일을 최하단에 적어 놓겠다. 답을 보지 말고 Map 과 Filter 함수를 써서 풀어보자.

자, 이번에는 Find 함수도 보자.

num 배열 안에 9 라는 값이 있는지 찾아 보겠다.

```
const num = [8,9,5,2,1,9]
const j=9

const findj = num.find(ele=>ele===j)
console.log(findj)
```

Find 함수는 찾는 값이 있다면 가장 먼저 찾은 값을 반환하고

찾는 값이 없다면 “undefined” 를 반환 한다.

자, 이번에는 every 함수도 한번 살펴보자.

배열의 값이 모두 3보다 큰지 확인해 보는 코드다.

```
const num = [8,5,2,1]
const e = num.every(ele=>ele>3)
console.log(e)
```

every 함수는 배열 안의 값이 모두 3보다 커야 true 를 반환한다.

만약에 배열안의 값이 하나라도 5 보다 큰지 확인하려면

some 함수를 사용하면 된다.

```
const num = [8,5,2,1]
const s = num.some(ele=>ele>5)
console.log(s)

// some 함수 사용 true / false 반환
```

map filter find every some 함수를 통해서 많이 사용하는 help method 를 살펴 보았다.

마지막으로 reduce 함수 하나만 더 살펴보자.

const num=[8,5,2,1] 배열의 합을 구해 보겠다. 그렇다면 for문을 돌려 sum을 구하면 되겠다.

```
//sum 을 구해보자
const num=[8,5,2,1]

let sum=0
for(let x=0;x<num.length;x++){
    sum+=num[x];
}
console.log(sum)
```

그런데 reduce 함수를 사용하면 더 간결하게 표현이 가능하다.

```
const num=[8,5,2,1]

const getsum = num.reduce((acc,cur)=>acc+cur,0)
console.log(getsum)
```

num.reduce() 를 통해서 num 배열에 reduce라는 helpmethod를 적용했다.

reduce 함수 안에는 (arrow함수 , 초기값) 이 들어간다.

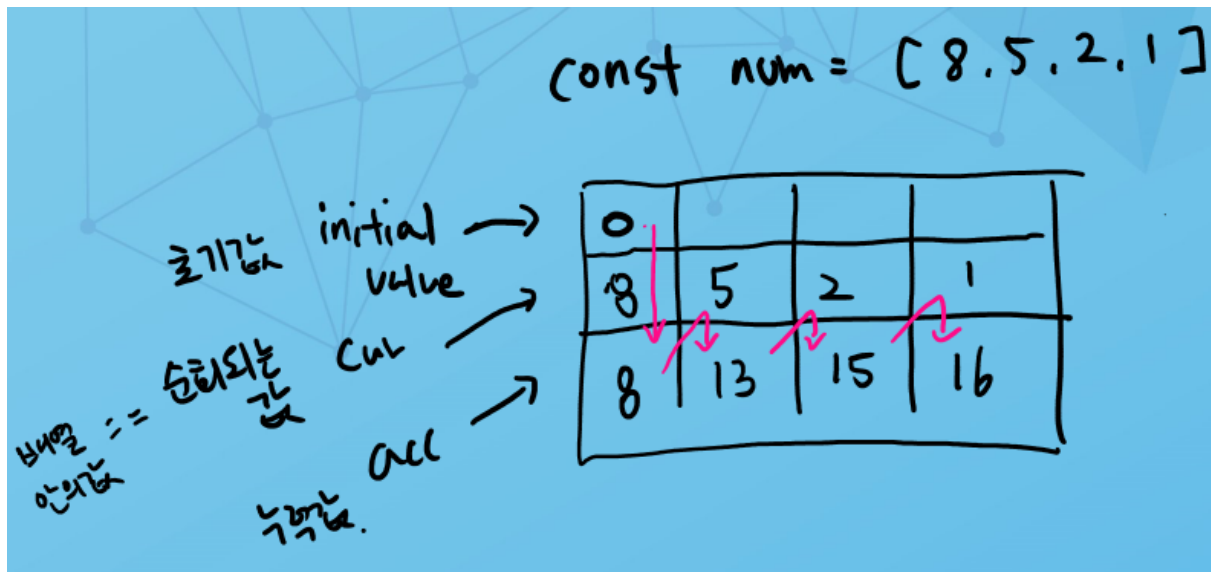
num.reduce((acc,cur)=>acc+cur,0) 를 조금 더 자세하게 보자.

```
// acc = 누적되는 값을 의미 (누적값)
// cur = 순회되는 값을 의미 (배열값)
// ,0 = acc에 들어갈 초기값 (0일경우 생략가능)
```

```
1  const num=[8,5,2,1]
2  ⚡
3  const getsum = num.reduce(( acc , cur )=> acc+cur,0)
4  console.log(getsum)
```

0 + 8

8리턴



프로세스를 하나하나 살펴 보자면,

[8,5,2,1] 숫자 배열이 있다. reduce 함수로 인해서 이 배열에 마치 for문처럼 loop가 작동된다.

그리고 loop의 시작값은 0으로 지정 했다.

첫 loop에서 초기 값(initial) + 배열의 현재 값(current), (0+8)이 되어 누적 값(accumulate)은 8이 된다.

그 다음 loop에서는 누적 값이 13이 되고 그 다음은 15 그 다음은 16이 된다.

배열의 요소가 완료될 때 까지 반복한 다음 결과가 단일 값, 즉 마지막 리턴 값으로 반환이 된다.

자 이렇게 함수에 대해서 계속 살펴보고 있다

잠깐 중간 정리를 하자면 지금까지

- 함수선언식 함수표현식+arrow function 에 대해서 살펴 봤고
- 보너스로 즉시 실행 함수 라는 것도 봤다.
- 그리고 foreach 문과 help method 에 대해서 봤다.

help method 중에 map filter find every some 그리고 reduce 에 대해서 살펴 보았는데

각 method에 인자값으로 함수가 들어 갔었다! 이렇게

다른 함수(method)의 인자로 전달되는 함수를 콜백함수 라고 한다.

아까 num 배열의 값을 forEach 문을 사용해서 출력했었다.

```
const num=[8,5,2,1]
num.forEach(ele=>console.log(ele));
```

이때 forEach 함수(method) 안에 인자로 ArrowFunction이 들어갔는데,
이 ArrowFunction 을 바로 콜백함수라 부른다.

지금까지 살펴본 메서드를 한눈에 살펴 보자면 다음과 같다.

메서드	설명	비고
forEach	배열의 각 요소에 대해 콜백 함수를 한 번씩 실행	반환 값 없음
map	콜백 함수의 반환 값을 요소로 하는 새로운 배열 반환	
filter	콜백 함수의 반환 값이 참인 요소들만 모아서 새로운 배열을 반환	
reduce	콜백 함수의 반환 값들을 하나의 값(acc)에 누적 후 반환	
find	콜백 함수의 반환 값이 참이면 해당 요소를 반환	
some	배열의 요소 중 하나라도 판별 함수를 통과하면 참을 반환	
every	배열의 모든 요소가 판별 함수를 통과하면 참을 반환	

- JavaScript 개발자들은 `for` , `while` 사용을 기피하는 경향이 있다. 불변성 `immutability` 을 지키기 위해서다. 따라서 위에서 연습한 help method `forEach` , `map` , `filter` 등 의 사용법은 반드시 알아두도록 하자.

자, 마지막으로 객체에서 this 키워드를 볼 것인데 this 키워드에 대해서 조금 살펴보자.

- 객체에서의 this !!

내용이 복잡 하다고 생각 할 수도 있어서 결론부터 말하자면,

This는 객체 내에서 어떤 방식으로 호출 되었는가에 따라
의미하는 것이 달라진다!! ((만능)이럴때는 이게 이거고 저럴때는 이게 이거다.)

This 가

일반함수로 호출 되었을때 : 전역객체(window) 브라우저의 최상위 객체를 의미한다.

메서드로 호출 되었을때 : 매서드를 호출한 "그"객체!! 를 의미한다.

생성자 함수로 호출되었을때 : 미래에 생성할 인스턴스!! 를 의미한다.

1. 일반함수로 호출 되었을때 : 전역객체(window) 브라우저의 최상위 객체를 의미한다.

```
// 함수선언식
function fc(){
  return console.log(this)
}
fc()

// 화살표 함수
const fc={()=>{console.log(this)}}
fc();
```

브라우저 최상위 객체 = window		
DOM	BOM(Browser Object Model)	Javascript core (내장객체)
DOM 요소에는 브라우저에 보여지는것등	폼 요소는 브라우저 제어하는것등	객체 문자열 넘버 배열 등등
HTML요소	location (문서 uri관리 새로고침 등)	
CSS 스타일 등이 DOM의 요소들이고	history (문서 열람이력관리 뒤로 앞으로)	
DOM (Document Object Model) 이라 부른다.	screen (모니터 정보 관리 화면크기 색상관리)	
HTML 문서의 요소와 태그들을 모두	등등	
Javascript가 이용할수 있도록 객체화 시키는 것을 말한다		

이번에는 함수선언식의 함수와 화살표 함수를 메서드 안에서 호출 해 보겠다.

2. 매서드로 호출 되었을때 : 매서드를 호출한 "그"객체!! 를 의미한다.

```
const obj={
  a:1,
  b(){
    console.log(this)
  },
}
obj.b();
```

```
// 출력 결과 {a: 1, b: f}
```

메서드 안에 함수를 함수선언식으로 정의해 준 후
obj.b();를 통해서 메서드를 호출해 주었다.

출력 결과는 {a: 1, b: f} 이며, 메서드를 호출한 객체가 출력이 되었다.

하! 지! 만!

그러나 위의 코드를 똑같이 화살표 함수로 표현해 보겠다.

```
const obj={  
  a:1,  
  b:()=>{  
    console.log(this)  
  },  
}  
obj.b();  
  
// 출력 결과: Window {..}
```

출력 결과가 다르다!!

그 이유는 화살표 함수에는 this 라는 것이 존재하지 않기 때문이다.

그래서 화살표 함수에서 this를 사용했다면

화살표 함수는 this 라는 것을 신경쓰지 않고, 자신을 감싸고 있는 외부로부터 this로 할 값을 가져온다.

그래서 화살표 함수에서 this 키워드로 접근을 하려고 했더니,

화살표 함수를 감싸고 있는 최상위 객체인 window가 출력이 된 것이다.

자 이번엔 생성자 함수 안에서 this 를 사용해 보자.

먼저 생성자 함수가 무엇인지 확인해 보자.

생성자 함수란? - 객체를 생성하기 위한 틀, 프레임을 의미한다.

예를 통해서 살펴보자.

만약에 key=name 이고 value가 choi 인 객체를 3개 만든다면?

```
const a1={
  name1:"choi",
}

const b1={
  name1:"choi",
}

const c1={
  name1:"choi",
}
```

이런식으로 표현할 수 있다. 그러나 조금 더 효율적으로 표현 해 보자.

함수를 사용해서 객체를 찍어 내겠다.

생성자 함수를 만드는 방법은 다음과 같다.

```
// 생성자 함수는 함수명을 대문자로 많이 시작한다.
// 1. 함수 생성
function Test(name){
  this.name=name
}

// 2. 객체를 생성자 함수를 통해서 찍어낸다.
const d1=new Test('choi')
const e1=new Test('choi')
```

위 코드가 생성자 함수에서의 this를 사용한 예가 되겠다.

여기서 this 가 의미 하는 것은 새로 생성 될 인스턴스를 의미한다.

지금 위의 코드는 함수 선언식으로 생성자 함수를 만들었다.

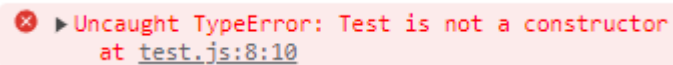
만약에 ArrowFunction으로 생성자 함수를 만들어 보면 어떨까?

```
// 생성자 함수는 함수명을 대문자로 많이 시작한다.
// 1. 함수 생성

const Test=(name)=>{this.name=name}

// 2. 객체를 생성자 함수를 통해서 찍어낸다.
const d1=new Test('choi')
const e1=new Test('choi')
```

바로



Uncaught TypeError: Test is not a constructor
at test.js:8:10

에러가 발생한다.

그 이유는 아까도 말했듯이 화살표 함수는 This 라는것이 존재하지 않기 때문에 new와 함께 실행 할 수 없다.

따라서 화살표 함수는 객체를 생성하는 용도로 사용할 수 없다. 그러한 이유로 화살표 함수는 보통 콜백함수로 많이 사용하지만, 생성자 함수로는 사용할 수 없다는 점을 반드시 유의 하자!

자 이번에는 추가적인 테스트로

4. 이번엔 객체의 method 안에 또 method가 있는 경우를 살펴 보겠다.

```
const myObj2 = {
  numbers: [1, 2, 3],
  myFunc: function () {
    this.numbers.forEach(function (number) {
      console.log(this) // window
    })
  }
}
console.log(myObj2.myFunc())
```

- myObj2 라는 객체 안에는

- myFunc 이라는 함수가 있다.
- this.numbers.forEach() 구문에서 this 는 myObj2 라는 객체를 가르키며, forEach문 안에 콜백함수를 함수 선언식으로 작성을 하였다.
- function (number) {
 console.log(this)
}

가 있다. 여기서의 this는 window를 가르킨다. 그 이유는 콜백 함수는 인자값으로 들어간 (객체와 상관없는) 변수로써의 함수다. 그래서 콜백함수는 일반적인 함수 호출을 한 것과 다름이 없기 때문에, 일반 함수로 호출 되었을때 : 전역객체(window) 브라우저의 최상위 객체를 의미한다.

위의 코드는 콜백 함수를 함수 선언식으로 코드를 작성 했다.

만약에 콜백 함수를 ArrowFunction으로 작성했다면 어떨까?

```
const myObj3 = {
  numbers: [1, 2, 3],
  myFunc: function () {
    this.numbers.forEach((number) => {
      console.log(this) // myObj3
    })
  }
}
console.log(myObj3.myFunc())
```

- myObj3 라는 객체 안에는
- myFunc 이라는 함수가 있다.
- this.numbers.forEach() 구문에서 this 는 myObj3 라는 객체를 가르키며 forEach문 안에 콜백함수로 ArrowFunction으로 코드를 작성 했다.
- (number) => {
 console.log(this)
}

여기서의 this는 window 가 아니라, myObj3를 가르킨다. 그 이유는 다시한번 강조 하지만 콜백 함수는 인자값으로 들어간 (객체와 상관없는) 변수로써의 함수다. 그래서 콜백함수는 일반적인 함수 호출을 한 것과 다름이 없다. 그런데 이번에는 그 일반 함수가 ArrowFunction이다.

그런데 중요한 사실은 ArrowFunction은 아까 This라는 것이 존재하지 않는다고 했다. 따라서 화살표 함수에서 this를 사용 했다면 화살표 함수는 해당 this 라는 것을 신경쓰지 않고, 자신을 감싸고 있는 외부로 부터 this로 할 값을 가져온다.

그렇기 때문에 여기서의 this는 myObj3가 된다.

<끝>

마지막으로 추가자료를 살펴보면서 오늘 마무리 지으면 되겠다.

[추가자료 1]

어제 PDF에서 객체문법을 설명 할 때 ,

객체문법5: 메서드명 축약 (객체 내 메서드에는 function 을 안써도 된다) 는

오늘 method를 공부하고 살펴 본다고 했다. 내용은 간단하다.

아래 코드와 주석을 살펴보자.

```
// 객체

fastfood={
  a:1,
  "bbq":"gold_olive",
  "ham burger":{
    b:1,
    c:2
  },
  abc:function (){
    console.log('hi')
  },
}
// abc(){
//   console.log('hi') // 매서드 선언시 function 생략 가능
// }

console.log(fastfood.a)
console.log(fastfood.bbq)
console.log(fastfood["ham burger"].c)

// 주의
```

```
// 객체이름 (햄버거)에 찍어쓰기가 있다면
// [" "] 사용해야 함!
// . 사용안됨!!
fastfood.abc()
```

객체 안에 속성 값으로 method가 들어간다면

function 키워드를 생략 할 수 있다는 내용이다. 상당히 간단하다.

[추가자료 2]

앞으로 javascript 학습을 위해서 콜백에 대해서 조금 더 알아보자.

```
const sampleFunc = () => {
  console.log("짜잔");
};

setTimeout(sampleFunc, 2000);
```

위 코드에서 `setTimeout` 은 실행을 지연 시킬 때 사용하는 함수다.

즉, 2초 후에 `sampleFunc` 을 실행시키라는 뜻이 된다.

함수의 첫번째 파라미터로 지연시킬 함수, 그리고 두번째 파라미터로 초(ms)가 들어간다.

여기서 눈여겨 볼 점은 `setTimeout` 함수안에 `sampleFunc` 함수 그 자체가 들어간다는 것이다.

이렇게 함수에 인자로 함수 원형이 들어간다면 해당 원형 함수를 콜백 함수라고 했다.

이것은 JavaScript 에서 함수가 하나의 자료형임을 의미하며, 변수처럼 쓰인다는 뜻이다.

방금 적었던 코드를 조금 더 간결하게 표현하자면 다음과 같다.

```
setTimeout(() => console.log("짜잔"), 2000);
```

다음 주에 한번 더 언급 되면서 사용 할 내용이라 이해 차원에서 살펴 보았다.

[연습문제 정답]

```
const answer=products.filter(ele=>ele.type==='fruit')
console.log(answer)

const ans=answer.map(ele=>ele.name)
console.log(ans)
```