10주차 과제_일반적인 function과 arrow function의 차이

경제학과 2020110210 공소연

1. 문법 구조
   - 일반적인 함수: `function` 키워드를 사용하여 함수를 선언한다.
   - 화살표 함수: 화살표(`=>`)를 사용하여 함수를 선언한다.

```
// 일반적인 함수
function twice(a) {
  return a*2;
}

// 화살표 함수
const TT = (a) => a*2;
```

2. this 바인딩
   - 일반적인 함수: 일반적인 함수는 자체적으로 `this` 값을 바인딩한다. `this`는 함수가 호출된 방법에 따라 동적으로 결정된다.
   - 화살표 함수: 화살표 함수는 자신의 `this`를 가지지 않고, 주변 범위(lexical scope)의 `this` 값을 상속받는다. 즉, 화살표 함수의 `this`는 화살표 함수가 정의된 곳의 `this`와 같다.

```
// 일반적인 함수
function greet() {
  console.log('안녕하세요, ' + this.name);
}
const person = {
  name: 'John',
  greet: greet
};
person.greet(); // 안녕하세요, John

// 화살표 함수 #1
const greet = () => {
  console.log('안녕하세요, ' + this.name);
};
const person = {
  name: 'John',
  greet: greet
};
person.greet(); // 안녕하세요, undefined

// 화살표 함수 #2
const obj = {
  name: 'John',
  sayHello: function() {
    setTimeout(() => {
      console.log('Hello, ' + this.name);
    }, 1000);
  }
};

obj.sayHello(); // Hello, John (1초 후에 출력)
```

```
/* 화살표 함수인 () => { console.log('Hello, ' + this.name); }는
자체적인 this 값을 가지지 않고, 주변 범위인 sayHello 메서드의 this 값을 상속받는다.
따라서 화살표 함수의 this는 sayHello 메서드의 this와 동일한 객체인 obj를 가리킨다. */
```

3. arguments 객체

- 일반적인 함수: 일반적인 함수 내에서 `arguments` 객체를 사용하여 인자에 접근할 수 있습니다. `arguments` 객체는 함수 내에서 사용할 수 있는 인수들의 컬렉션입니다.
- 화살표 함수: 화살표 함수는 자체적인 `arguments` 객체를 가지지 않습니다. 따라서 화살표 함수 내에서는 `arguments` 객체를 사용할 수 없습니다. 대신, 외부 범위에서 전달된 인자를 사용해야 합니다.

```
// 일반적인 함수
function myFunction(a, b) {
  console.log(arguments);
}
myFunction(1, 2, 3); // Arguments(3) [1, 2, 3, callee: ƒ, Symbol(Symbol.iterator): ƒ]

// 화살표 함수 : 함수의 매개변수를 직접 사용하여 인수에 접근해야 한다.
const myArrowFunction = (a, b) => {
  console.log(a, b);
};
myArrowFunction(1, 2); // 1 2
```

4. 생성자 함수로 사용할 수 없음

- 일반적인 함수: 일반적인 함수는 생성자로 사용될 수 있습니다. `new` 키워드와 함께 호출하면 새로운 객체를 생성하여 반환할 수 있습니다.
- 화살표 함수: 화살표 함수는 생성자로 사용될 수 없습니다. `new` 키워드와 함께 호출하면 에러가 발생합니다.

```
// 일반적인 함수
function Person(name) {
  this.name = name;
}
const john = new Person('John');
console.log(john.name); // John

// 화살표 함수
const Person = (name) => {
  this.name = name;
};
const john = new Person('John'); // TypeError: Person is not a constructor
```