

넷째 주

- 변수의 범위
- 클래스 맛보기
- 함수의 사용형태

변수의 범위

변수는 선언되는 위치에 따라 각각의 명칭을 가집니다. 다음 4가지 변수명칭을 알아보겠습니다.

- 전역변수

전역에서 사용하는 데이터를 담는 변수이며 어디서든 접근해서 사용 가능합니다.

- 지역변수

특정 영역에서만 사용할 수 있는 변수입니다. 주로 함수 내부에 만들어지며 함수가 종료되면 변수가 사라집니다.

- 매개변수(파라메터)

함수 외부에서 함수 내부로 데이터를 전달하기 위한 용도로 사용하는 변수입니다.

- 멤버변수(프로퍼티)

클래스 내부에서 만들어지며 주로 객체에서 사용하는 정보를 담는 변수입니다.

클래스 맛보기

javascript는 객체기반의 언어입니다.

프로그램 내부적으로 대부분 객체를 기반으로 만들어진 언어이기 때문에 다른 언어와는 다른 특징들이 나타납니다.

이러한 특징들을 알아보기 전에 간단하게 객체를 다루는 방법에 대해 알아보겠습니다.

클래스란?

앞서 함수가 특정 기능을 하는 구문(로직, 알고리즘)을 묶을 때 사용하는 문법이라면,

클래스는 연관있는 변수와 함수를 하나로 묶을 때 사용하는 문법입니다.

클래스 선언

클래스는 기존에는 function을 통해 선언했으나, es6 이후 class라는 예약어를 통해 선언이 가능합니다.

```
class Person {}
```

위의 코드는 Person이라는 클래스를 선언하는 구문입니다.

보이는 것처럼 클래스 내부에는 아무것도 없습니다.

클래스 사용은 연관있는 변수와 함수를 하나로 묶기 위해 사용한다고 했으니 우선 클래스의 변수를 선언해보겠습니다.

```
class Person {  
    name = 'kim'  
    age = 30  
    height = 190  
    weight = 100  
}
```

클래스 내부의 변수 선언은 그저 변수명을 선언하고 해당 변수에 데이터를 넘겨주면 됩니다.

이런식으로 클래스 내부에 선언된 변수들을 '멤버변수'라고 부릅니다.

클래스 사용

앞에서 선언한 클래스를 사용해보려 합니다.

클래스를 사용하기전에 먼저 알아두어야 할 개념이 있습니다.

바로 클래스를 통해 객체를 생성할 수 있다는것입니다.

여기서 그 유명한 '붕어빵'과 '붕어빵 틀' 이야기가 나옵니다.

'붕어빵 틀' 이 클래스에 해당하고 붕어빵 틀로 찍어낸 '붕어빵' 이 객체가 됩니다.

그리고 객체는 인스턴스 라는 이름으로도 불립니다.

```
const kim = new Person()
```

클래스를 통한 객체 생성은 new 라는 키워드를 사용합니다.

지금은 kim이라는 변수에 Person 클래스를 생성해서 저장한 로직입니다.

각 멤버변수에 접근하려면 '.' 연산자를 통해 접근 가능합니다.

```
console.log('kim.name: ', kim.name);
console.log('kim.age: ', kim.age);
console.log('kim.height: ', kim.height);
console.log('kim.weight: ', kim.weight);
```

Ex - 클래스 만들기

MyCompany 라는 클래스를 만들어주세요.

클래스명: MyCompany

멤버변수:

companyName: string

address: string

workerCount: number

income: number

클래스 함수

클래스는 변수와 함수를 하나로 묶을 때 사용한다고 했습니다.
이번에는 클래스 내부에 함수를 선언하는 방법을 보겠습니다.

```
class Person {  
    name = 'kim'  
    age = 30  
    height = 190  
    weight = 100  
  
    sayHi() {  
        console.log('Hi !!')  
    }  
}
```

sayHi라는 함수를 클래스 내부에 선언했습니다.
기존에 함수를 선언할 땐 function 예약어가 필요했지만 클래스 내부에 선언할 경우 따로 function 예약어 없이 함수 선언이 가능합니다.

클래스 함수

이제 선언한 함수를 사용해보겠습니다.

```
const kim = new Person()  
kim.sayHi()
```

Person 클래스를 사용해 kim이라는 객체를 생성했습니다.

kim이라는 객체에서 '.' 연산자를 통해 sayHi 함수에 접근해서 해당 함수를 실행시키는 로직입니다.

이렇게 클래스 내부에서 선언된 함수를 '멤버함수' 또는 '메서드'라고 부릅니다.

클래스 멤버함수에서 멤버변수 접근

클래스의 멤버함수에서 멤버변수에 접근하기 위해선 `this` 키워드를 사용해야 합니다.

```
class Person {  
    name = 'kim'  
    age = 30  
    height = 190  
    weight = 100  
  
    sayMyName() {  
        console.log('my name is ', this.name);  
    }  
}
```

`sayMyName` 함수는 `name`에 접근하기 위해 `this`를 사용했습니다. 여기서 `this`는 객체 자신을 나타냅니다. `this`가 객체 자신을 말하는것이기 때문에 `this.name`은 객체자체의 `name`을 불러오게 됩니다.

클래스 constructor

지금까지 사용한 Person 클래스는 멤버 변수가 고정되어 있었습니다.

이런 상태로는 다양한 형태의 Person 객체를 만들 수 없습니다.

클래스에서는 각각의 상황에 따라 멤버 변수를 다룰수 있도록 만들어져 있지만

먼저 클래스를 통해 객체를 생성할 때 멤버변수를 설정할 수 있도록 하는방법을 알아보겠습니다.

```
class Person {  
    name = 'kim'  
    age = 30  
    height = 190  
    weight = 100  
  
    constructor(name, age, height, weight) {  
        if (name) this.name = name  
        if (age) this.age = age  
        if (height) this.height = height  
        if (weight) this.weight = weight  
    }  
}
```

객체 생성시 실행되는 함수 중 constructor가 있습니다. constructor는 클래스의 기본적으로 선언되어 있는 함수로 new 키워드를 통해 객체를 생성할 때마다 실행됩니다.

클래스 constructor

객체를 생성하면서 멤버변수들의 초기값들을 넣어주고 싶다면 constructor의 매개변수로 객체 생성시 받을 값을 선언해주고, 해당값들을 this.멤버변수명을 통해 접근해서 값을 넣어줍니다.

이제 constructor의 매개변수에 데이터를 어떻게 전달하는지 보겠습니다.

```
const lee = new Person('lee', 20, 200, 150)
```

객체를 생성할 때 클래스 뒤의 괄호안에 constructor로 전달할 데이터들을 넣어주면 constructor에서 해당 데이터들을 사용할 수 있습니다.

클래스 멤버함수를 이용한 멤버변수 변경

클래스의 멤버함수에서 멤버변수에 접근하는건 this 키워드를 통해 가능하다고 배웠습니다.
이를 이용해서 멤버함수에서 멤버변수를 변경하는것도 가능합니다.

```
class Person {  
    name = 'kim'  
    age = 30  
    height = 190  
    weight = 100  
  
    changeMyName(newName) {  
        this.name = newName  
    }  
}
```

changeMyName은 매개변수로 newName을 받고, 받은 newName을 멤버변수 name에 넣어줍니다.
그러면 기존의 멤버변수 name은 새로운 데이터를 가지게 됩니다.

클래스 멤버함수를 이용한 멤버변수 변경

```
const lee = new Person('lee', 20, 200, 150)
lee.sayMyName() // lee
lee.changeMyName('park')
lee.sayMyName() // park
```

이렇게 어렵게 배운 클래스와 객체는 어디에 사용할까요?

앞의 과정에서 우리가 사용하던 객체들이 있습니다.

document, console 이런것들이죠

document.querySelector(), console.log() 이런 함수들은 모두 객체 안에있는 함수를 사용했던 것 입니다.

Ex - 클래스 함수 만들기

앞의 MyCompany 클래스에 함수를 만들어 주세요.

함수명: increaseWorkerCount, decreaseWorkerCount

요구사항

- increaseWorkerCount 함수를 실행하면 workerCount 가 1씩 증가함
- decreaseWorkerCount 함수를 실행하면 workerCount 가 1씩 감소함

this

this 키워드는 객체 자신을 나타낼 때 사용되는 키워드입니다.

하지만 js언어가 객체 기반으로 만들어진 언어이기 때문에 this가 사용되는 상황에 따라 this가 가져오는 객체가 달라집니다.

이번에는 이런 js의 this에 대해 알아보겠습니다.

this

```
console.log(this) // window
```

전역에서 this를 찍어보면 window 객체가 나옵니다.

window는 객체이고, 객체 안에서 this를 사용하면 this를 포함하는 객체가 나옵니다.

지금 상황에선 window 안에서 this를 사용했기 때문에 window 가 나오는것이죠.

```
function forThis() {  
  console.log('this: ', this);  
}
```

```
forThis() // window
```

함수 안에서 this를 확인해보겠습니다.

함수를 선언했을 때, 이 함수를 포함하고 있는건 window 객체입니다.

따라서 함수 안에서도 window 객체가 반환됩니다.

this

```
class ForThisClass {  
    showThis() {  
        console.log('this: ', this);  
    }  
}  
  
const test = new ForThisClass()  
test.showThis() // ForThisClass
```

클래스 안에서 this 입니다.

this는 자신이 포함된 객체를 반환한다고 그랬습니다.

그래서 showThis를 실행하면 해당 함수를 포함하고 있는 객체가 반환되어서 ForThisClass 객체가 반환됩니다.

this

```
document.body.onclick = function () {  
    console.log('this: ', this); // body  
}
```

이벤트 안에서 this 입니다.

해당 이벤트를 등록하는 DOM이 body 입니다.

DOM 또한 객체이고 객체의 이벤트에 함수를 등록한것이므로 이벤트를 등록한 객체가 나오게 됩니다.

지금같은 경우엔 body가 나오게 됩니다.

this

```
document.body.onclick = function () {  
  function inner() {  
    console.log('this: ', this); // window  
  }  
  
  inner()  
}
```

함수 안에 함수를 만들어서 this를 찍어보겠습니다.

여기선 this가 window 객체를 가져옵니다.

함수 안에 함수(inner())를 만들때, 해당 함수가 등록되는 객체가 지정되지 않으면서 함수의 영역이 전역으로 변경되었습니다.

변수와 함수

javascript에선 함수가 변수처럼 사용되는 경우가 있습니다.
이번엔 다음 세가지 상황에 대해서 알아보겠습니다.

- 변수에 함수를 저장하는 형태
- 함수의 매개변수로 함수를 받는 형태
- 함수의 리턴값으로 함수를 출력하는 형태

변수에 함수를 저장하는 형태

```
function sayHello() {  
    console.log('Hi !!!')  
}  
  
sayHello()  
  
const myFunc = sayHello  
myFunc()
```

위의 코드는 sayHello 함수를 만든 후 sayHello 함수를 myFunc 변수에 저장하는 로직입니다.
자바스크립트는 특이하게도 변수에 함수를 저장하는것이 가능합니다.

매개변수로 함수를 입력받는 형태

```
function sayMyName(name) {  
    console.log('my name is ', name);  
}  
  
function runFunc(someFunction) {  
    someFunction('Lee')  
}  
  
runFunc(sayMyName)
```

runFunc 함수와 sayMyName 함수를 만들었습니다.

sayMyName은 우리가 기존에 알고있던 함수를 만드는 방법이라서 익숙하지만 runFunc 함수는 어딘가 이상하게 보입니다.
runFunc 함수는 매개변수로 입력받은 요소를 실행시키고 있습니다.

이런식으로 매개변수로 입력받는 함수를 콜백함수(callback)라 부릅니다.

콜백함수는 주로 비동기 로직에서 함수가 순서대로 수행되어야 할 때 사용됩니다.

함수를 리턴시키는 함수

```
function genFunction(name) {  
  function sayMyName() {  
    console.log('my name is ', name);  
  }  
  
  return sayMyName  
}  
  
const generatedFunc = genFunction('Kim')  
generatedFunc()
```

genFunction 함수는 sayMyName 함수를 반환하고 있습니다.

반환된 함수를 다시 변수 generatedFunc에 저장시켜서 실행시키는 로직입니다.

익명함수

익명함수란 이름이 없는 함수를 의미합니다.

```
const noNameFunc = function () {  
    console.log('this is no name function')  
}  
  
noNameFunc()
```

이전에 함수를 변수로 저장할 땐 저장할 함수의 이름을 지정해줬었습니다.

하지만 지금의 경우 함수의 이름을 지정하지 않고 익명함수로 함수를 생성해주고 변수에 저장시켰습니다.

기존에 함수를 선언하고 다시 재사용하는 입장에서 봤을 땐 익명함수는 쓸모가 없어 보입니다.

하지만 익명함수 역시 많은곳에서 사용됩니다.

익명함수는 재사용될 필요가 없는 함수를 선언할 때 많이 사용되는데, 주로 이벤트 바인딩 되는 함수나, 함수를 매개변수로 받는 함수에 사용되는 경우가 많습니다.

익명함수

```
setTimeout(function () {  
  console.log('this is set timeout')  
}, 2000);
```

setTimeout은 대표적인 콜백함수를 사용하는 함수입니다.

여기서 setTimeout은 첫번째 매개변수로 함수를 받는데 일일이 함수를 선언해서 다시 넣어주기 귀찮은 경우(함수의 재사용이 필요 없는 경우) 이렇게 익명함수를 통해 매개변수로 전달해줄 수 있습니다.

```
button.onclick = function () {  
  console.log('button click !!!')  
}
```

dom의 click 이벤트에 함수를 바인딩 시키는 경우도 재사용이 필요한 함수가 아닌 경우 익명함수를 통해 이벤트 바인딩을 구현해 줄 수 있습니다.

Ex - setInterval 사용해보기

익명함수를 이용해서 setInterval을 사용해보세요

setInterval, clearInterval 함수의 문법은 다음과 같습니다.

```
setInterval(실행할함수, 주기);  
clearInterval(정지시킬setInterval)
```

setInterval 함수를 이용해서 다음 로직을 구현해보세요

- sum 변수를 생성 초기값 0
- setInterval 함수를 이용해서 sum 변수가 1초 지날때마다 10씩 증가하게 함
- setInterval 함수를 중지시키기 위한 버튼을 만들고
버튼의 click 이벤트에 clearInterval 함수를 바인딩 시켜서 setInterval 정지시킴

숙제 - ImageMovableClass 만들기

멤버변수

- image: image DOM을 저장하는 멤버변수
- left: image의 left 좌표값을 저장하는 멤버변수
- top: image의 top 좌표값을 저장하는 멤버변수
- moveDistance: 방향키를 눌렀을 때 움직이는 거리. 기본값 30

constructor

image, left, top 의 초기값을 설정해줌

window.onkeydown 이벤트에 moveArrow 함수를 바인딩함

멤버함수

- changeDistance: 매개변수로 숫자를 받고, 받은 매개변수로 moveDistance를 변경함
- moveArrow: 키보드 이벤트를 입력받고, 키보드의 code값이 방향키일 경우 해당 방향으로 이미지를 이동시켜주는 로직 구현