10월 12일 (화)

1. 프로그래머스 JavaScript Study Session 1.

- 사전 연습 문제에 대한 해설

1번 문제

강의 노트 질문 0

1번 문제

(1번 문제 코드)

```
function Cat(name, age) {
  this.name = name;
  this.age = age;
}
const tabby1 = Cat('nana', 5)
console.log(tabby1.name)
```

1. 위의 코드를 실행하면 무엇이 출력되나요? *

- 'nana' 가 출력된다.
- 빈 문자열이 출력된다.
- undefined 가 출력된다.
- 오류가 발생한다.

위 1번 문제에서 왜 그 답을 선택했는지 (최대한 자세히)서술해주세요. *

장문형 텍스트

- 정답은 '오류가 발생한다'
- `apply` , `call` , `bind` 등으로 `this` 에 대해 주입한 상황이 아니고 `new` 키워드없이 실행한 함수 내 this는 전역 객체(window)를 바라본다.
- 즉 `this.name = name` 의 결과는 `window.name = name` 이라는 이야기.

스코프를 쓸 때는 지역, 글로벌 중 어떤 걸로 쓰일지 생각하고 써야한다.

사전 퀴즈의 내용이 기술면접에서 나오는 내용과 비슷하다.

자바스크립트는 다른 언어와 다르게 New 키워드를 사용해서 함수를 생성한다.

- 인스턴스를 만드는 것은 자바스크립트의 의도된 기능이다.

2번 문제

2번 문제

(2번 문제 코드)

```
(function(name){
  console.log(`hello ${name}`)
})('roto')
```

2. 위 코드를 실행하면 무엇이 출력되나요? *

- ◯ 함수가 선언되기만 하고 아무것도 출력되지 않는다.
- ◯ 'hello' 만 출력된다.
- ◯ 'hello roto' 가 출력된다.
- ◯ 오류가 발생한다.

위 2번 문제에서 왜 그 답을 선택했는지 (최대한 자세히)서술해주세요. *

장문형 텍스트

- `hello roto` 가 출력된다.
- `와 `로 감싼 문자열은 ES6에 있는 `template strings` 라는 문법으로,
- 즉시 실행 함수 표현(IIFE, Immediately Invoked Function Expression)이라고 하며, 함수를 정의함과 동시에 실행한다.
- JavaScript 특성상 변수의 scope는 해당 변수를 감싸고 있는 function에 한정되는데, 이걸 이용해 변수나 함수의 전역화를 최소화 시킬 수 있다.
- 다음은 대표적인 IIFE의 응용이다.

```
var logger = (function(){
  // logCount는 밖에서 접근할 수 없다. 일종의 private 효과
  var logCount = 0;
  function log(message) {
    console.log(message);
    logCount = logCount + 1;
  }
  function getLogCount() {
    return logCount;
  }
  return {
    log: log,
    getLogCount: getLogCount
  }
})()
```

- 즉시 실행 함수 표현 (IIFE) ⇒ (function( ){ }) : 스코프를 제한시키기 위한 의도. 즉 모듈화를 위해 예전에 많이 썼다. 요즘엔 import 문법때문에 잘 안씀

3번문제

3번 문제

(3번 문제 코드)

```
var idiots = {
  name: 'idiots',
  genre: 'punk rock',
  members: {
    roto: {
       memberName: 'roto',
       play: function() {
        console.log(`band ${this.name} ${this.memberName} play start.`)
      }
    }
  }
}

idiots.members.roto.play()
```

3. 위 코드를 실행하면 무엇이 출력되나요? 그리고, 그것이 출력되는 이유는 무엇인지 최대한 자세히 서술해주세요. (주관식) *

장문형 텍스트

- function scope 관련 문제
- 정답은 `band undefined roto play start.` 출력
- play 함수의 this 내에는 name이 없기 때문에 undefined가 출력 되는 것
- 해결법

```
var idiots = {
  name: 'idiots',
  genre: 'punk rock',
  members: {
    roto: {
      memberName: 'roto',
      play: function() {
        console.log(`band ${idiots.name} ${this.memberName} play start.`
      }
    }
  }
}
```

- 객체 리터럴 문제 객체 안에 객체안의 this 가 무엇을 가르키는지?

this 가 무엇을 가르키는지 항상 생각해야한다. 항상 다른 것을 가르키기 때문에

this를 일관적이게 작성하는 것이 코드를 잘 짜는 것이다.

4번 문제

4번 문제

(4번 문제 코드)

```
function RockBand(members) {
  this.members = members;
  this.perform = function() {
    setTimeout(function(){
      this.members.forEach(function(member){ member.perform() })
    }, 1000)
  }
}


var theOralCigarettes = new RockBand([
  {
    name: 'takuya',
    perform: function() { console.log('a e u i a e u i')}
  }
])
```

4. 위 코드를 실행하면 에러가 발생하는데, 해당 코드가 본래의 의도대로 실행되게 하는 법에 대해 아는 방법을 모두 나열하고 설명해주세요. *

장문형 텍스트

- 맨 아래 `theOralCigarettees.perform()` 을 실행하는 게 빠졌네요..
- `perform` 함수 아래 `setTimeout` 으로 인해 실행되는 함수의 `this` 는 `RockBand` 의 `this` 가 아니기 때문에, 참조 오류가 발생
- 클로저를 이용한 해결법

```
function RockBand(members) {
  var that = this;
  this.members = members;
  this.perform = function() {
    setTimeout(function(){
      that.members.forEach(function(member){ member.perform() })
    }, 1000)
  }
}

var theOralCigarettes = new RockBand([
  {
    name: 'takuya',
    perform: function() { console.log('a e u i a e u i')}
  }
])

theOralCigarettes.perform()
```

- bind를 이용한 해결법

```
function RockBand(members) {
  var that = this;
  this.members = members;
  this.perform = function() {
    setTimeout(function(){
      this.members.forEach(function(member){ member.perform() })
    }.bind(this), 1000)
  }
}

var theOralCigarettes = new RockBand([
  {
    name: 'takuya',
    perform: function() { console.log('a e u i a e u i')}
  }
])

theOralCigarettes.perform()
```

- Arrow 펑션안에 있는 this 는 상위의 this 를 찾는다.

bind() 함수와 apply() 콜함수 의 차이에 대해서 배워보자.

이 모든게 this와 관련되어 있다.

5번 문제

5번 문제

(마지막 문제 코드)

```
const numbers = [1, 2, 3, 4, 5]
for(var i = 0; i < numbers.length; i++){
  setTimeout(function() {
    console.log(`number index ${i}`)
  }, 3000)
}
```

5. 위 코드를 실행하면, 숫자가 순차적으로 0부터 4까지 출력되지 않고 모두 5만 출력된다. 왜 그런 현상이 발생하는지, 또 어떻게 수정해야 본래의 의도대로 동작할지 설명해주세요. *

장문형 텍스트

- 전형적인 클로져 문제
- setTimeout이 실행되는 시점에는 루프가 이미 끝나있어서 i는 5가 들어가있어서 생기는 문제
- i를 var 대신 `let` 을 쓰는 걸로 해결할 수 있음
- 혹은 setTimeout을 IIFE로 감싸고, 파라메터로 i를 넘기는 것으로 해결 가능

```
const numbers = [1, 2, 3, 4, 5];
for(var i = 0; i < numbers.length; i++){
  (function(count){
    setTimeout(function(){
      console.log(`number index ${count}`);
    }, 1000);
  })(i)
}
```

즉시 실행함수(IIFE) 를 사용해서 코드를 고친다.

var 를 let 으로 변경하면 똑같은 효과를 본다.

⇒ **var는 호이스팅**과 같은 효과로 for 문의 밖에 선언된다고 보면되고 let 은 for 문안에서만 적용되는 블록스코프이다.

forEach 를 쓰면 똑같은 효과를 본다.

AJAX 를 루프로 한번에 여러번 실행할 때 이러한 문제를 자주 갖는다.

call() 과 apply() 는 배열로 파라미터를 전해주냐 마냐의 차이이다. 둘 다 모두 객체를 생성할 때 사용하는 것이다.

이게 기술이 왜 나왔는지 알아야하고, 어떻게 쓰이고, 기술의 철학을 알아한다.

새로운 기술이 나오면 예전에 비해 어떤 점이 개선되었는지 알아야지 그 기술을 제대로 이해하고 사용할 수 있다.

6번 문제

6번 문제

for문을 이용한 방법

```
function printCats() {
  const userNames = []
  for(let i = 0; i < users.length; i++) {
    if (users[i].type === 'cat') {
      userNames.push(users[i].name);
    }
  }
  console.log(userNames.join(''));
}
```

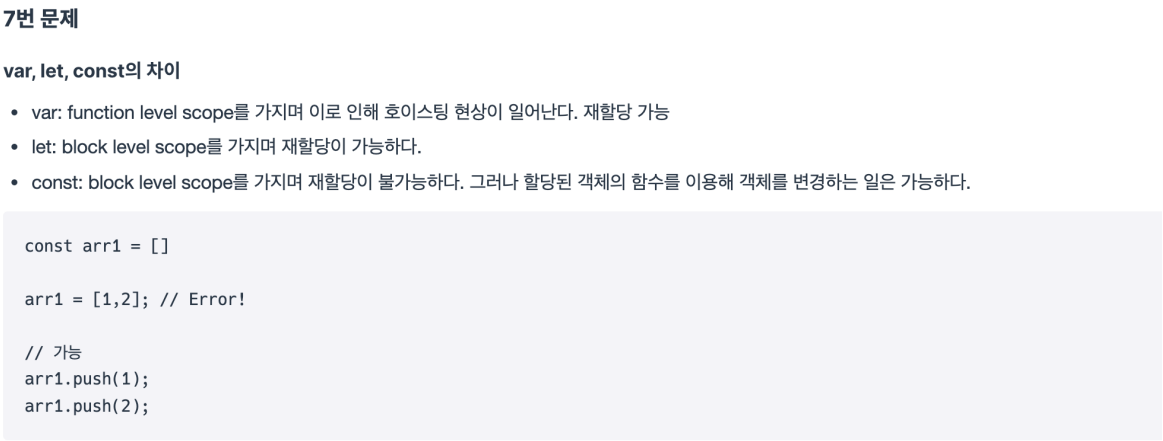
es5를 이용한 방법

```
function printCats() {
  console.log(users.filter(user => user.type === 'cat').map(user => user.name).join(''));
}
```

es5 이용한 것이 에러가 덜 난다.

아래 방식을 권고

7번 문제



7번 문제

var, let, const의 차이

- var: function level scope를 가지며 이로 인해 호이스팅 현상이 일어난다. 재할당 가능
- let: block level scope를 가지며 재할당이 가능하다.
- const: block level scope를 가지며 재할당이 불가능하다. 그러나 할당된 객체의 함수를 이용해 객체를 변경하는 일은 가능하다.

```
const arr1 = []

arr1 = [1,2]; // Error!

// 가능
arr1.push(1);
arr1.push(2);
```

var, let, const 의 차이에 대해

아직까지 var를 쓰는 회사가 있기때문에 알아둬야한다.

불변성에 대한 이슈가 있었음

웬만하면 const를 쓰는 것이 버그가 덜 남

const 배열 은 완전 재할당이 불가능한 것이 아니라 push 같은 것은 가능하다.

const 위주로 짜는 것을 권고한다.

8번 문제

8번 문제

클로저란?

자신의 scope 외부에 있는 것을 가져와 쓰는 것.

```
var a = 1;

function hello() {
  console.log(a); // a가 hello function scope에 없는데에도 접근 가능함.
}
```

클로저에 대한 좀 더 자세한 설명은 아래 문서를 참고하자.

- https://developer.mozilla.org/ko/docs/Web/JavaScript/Guide/Closures
- https://hyunseob.github.io/2016/08/30/javascript-closure/

클로저에 대한 내용