섹션 2: 자바스크립트 새로고침

작성일시	@February 20, 2023 2:56 PM
강의날짜	@2023/02/20
편집일시	@February 21, 2023 1:00 AM
분야	React
공부유형	강의
복습	☐
태그	let const var

12. 변수 “let”과 “const” 이해하기

var

- 거의 사용하지 않음

let과 const는 모두 변수들의 범위를 변경하는 것

let

- 값을 수정할 수 있는 변수를 선언할 때 사용

const

- 한 번 지정하면 절대 변하지 않는 값인 상수를 선언할 때 사용
- 새로운 값을 할당할 수 없는 키워드

13. 화살표 함수 (Arrow Functions)

```
// [1]
function myFnc() {
  ...
}
const myFnc = () => {
  ...
}
```

```
// [2]
function printMyName(name) {
  console.log(name);
}

printMyName('Max');

//"Max"


// [3] 인수가 오직 하나일 때,
const printMyName = name -> {
  console.log('Max');
}

printMyName('Max');


// [4]
const printMyName = () => {
  console.log('Max');
}

printMyName();

//"Max"


// [5]
const printMyName = (name, age) => {
  console.log(name, age);
}

printMyName('Max', 28);

//"Max"


//[6]

// 1
const multiply = (number) => {
  return number * 2;
}

console.log(multiply(2));

// 2
const multiply = number => number * 2;

console.log(multiply(2));

// 1과 2는 같은 함수
```

14. Exports와 Imports

```
// [1] person.js
const person = {
  name: 'Max'
}

export default person

// [2] utility.js
export const clean = () => {...}
export const baseData = 10;


// [3] app.js
import person from './person.js'
import prs from './person.js'

// person이든 prs이든 상관 없음
// 언제나 default로 표시한 것을 참조

import { baseData } from './utility.js'
import { clean } from './utility.js'

// 두 개의 다른 상수를 불러오는데 그 파일에서 특정한 어떤 것을 정확하게 가리키기 위해
// export 구문에서 중괄호를 사용함
// 이것을 named export라고 함. 이름으로 어떤 것을 불러오기 때문
// 불러오려는 것의 이름으로 두 상수 clean과 baseData 를 불러오는데, 어떤 것도
// default로 지정하지 않았기 때문
// import { baseData, clean} from './utility.js'도 가능


// [4]
// default export
import person from './person.js'
import prs from './person.js'

// named export
import { smth } from './utility.js'  // 가장 많이 사용
import { smth as Smth } from './utility.js'  // 별칭 사용 가능
import * as bundled from './utility.js'  // 모든 것 import

// 무엇이든 export하는 것에 접근하기 위해서는 bundled.baseData, bundled.clean과 같이 사용
```

15. 클래스 이해하기

클래스 : 객체를 위한 핵심 청사진

- class 키워드로 정의되고, property와 method를 가질 수 있음
- 생성자 함수를 만드는 좀더 편한 방법

- 클래스를 가지고 자바스크립트 객체를 생성할 수 있음

```
// [1]
class Person {
  name = 'Max'  // Property : 클래스에 정의한 변수
  call = () => {...}  // Method : 클래스에 정의한 함수
}


// [2] Usage
const myPerson = new Person()
myPerson.call()
console.log(myPerson.name)


// [3] Inheritance
class Person extends Master


// [4]
class Human {
  constructor() {
    this.gender = 'male';
  }

  printGender() {
    console.log(this.gender);
  }
}

class Person extends Human {
  constructor() {
    super();  // 서브클래스에서는 super 생성자를 먼저 호출해야 함
//  다른 클래스를 확장하고 생성자를 실행한다면, 생성자 함수 안에 super() 메소드를 추가해야함
    this.name = 'Max';
    this.gender = 'female';
  }

  printMyName() {
    console.log(this.name);
  }
}

const person = new Person();
person.printMyName();
person.printGender();
```

16. 클래스, 속성 및 메서드

- property : 클래스와 객체에 추가되는 변수같은 것
- method : 클래스와 객체에 추가되는 함수같은 것

```
// [1] ES6 => 사용하지 않음
constructor() {
  this.myProperty = 'value'
}

myMethod () {...}


// [2] ES7
myProperty = 'value'

myMethod = () => {...}


// [3] ES6/Babel Example
class Human {
  gender = 'male';

  printGender = () => {
    console.log(this.gender);
  }
}

class Person extends Human {
    name = 'Max';
    gender = 'female';

  printMyName = () => {
    console.log(this.name);
  }
}

const person = new Person();
person.printMyName();
person.printGender();
```

17. 스프레드 및 나머지 연산자

```
// [1] Spread : 배열의 원소나 객체의 property를 나누는데 사용
const newArray = [...oldArray, 1, 2]
const newObject = {...oldObject, newProp:5}


// [2] Rest : 함수의 인수 목록을 배열로 합치는데 사용
function sortArgs(...args) {
  return args.sort()
}


// [3]
const numbers = [1, 2, 3];
const newNumbers = [numbers, 4];
```

```
console.log(newNumbers);  // [[1, 2, 3], 4]


// [4]
const numbers = [1, 2, 3];
const newNumbers = [...numbers, 4];

console.log(newNumbers);  // [1, 2, 3, 4]


// [5] Spread
const person = {
  name: 'Max'
};

const newPerson = {
  ...person,
  age: 28
}

console.log(newPerson);

// [object Object] {
//   age: 28,
//   name: "Max"
// }


// [6] Rest
const filter = (...args) => {
  return args.filter(el => el === 1);
}

console.log(filter(1, 2, 3));  // [1]
```

18. 구조분해할당(Destructuring)

Destructuring : 배열의 원소나 객체의 프로퍼티를 추출해서 변수에 저장할 수 있도록 함

- 스프레드 연산자와 같은 것이라고 생각할 수 있는데 아님
- Spread : 모든 원소와 프로퍼티를 가져와서 새 배열이나 객체에 전달
- Destructuring: 원소나 프로퍼티를 하나만 가져와서 변수에 저장

```
// [1] Array Destructuring
[a, b] = ['Hello', 'Max']
console.log(a)  // Hello
console.log(b)  // Max
```

```
// [2] Object Destructuring
{name} = {name: 'Max', age: 28}
console.log(name)  // Max
console.log(age)  // undefined


// [3]
const numbers = [1, 2, 3];
[num1, num2] = numbers;
console.log(num1, num2);

// 1
// 2


// [4] Array Destructuring
const numbers = [1, 2, 3];
[num1, , num3] = numbers;
console.log(num1, num3);

// 1
// 3
```

19. 참조형 및 원시형 데이터 타입

```
// [1]
const number = 1;
const num2 = number;

console.log(num2);  // 1


// [2]
const person = {
  name: 'Max'
};

const secondPerson = person;

person.name = 'Manu';

console.log(secondPerson);

// [object Object] {
//   name: "Manu"
// }

// 단지 포인터를 복사했고 person이 가리키는 메모리에 있는 동일한 객체를 가리키기 때문
// 그래서 person의 name을 변경하면 자동적으로 secondPerson의 이름도 바뀜
// 만약 객체나 배열을 이렇게 복사한다면, 한 앱에 있는 주소에서 객체를 조작할 수도 있음
// 그리고 실수로 앱의 다른 주소에 있는 같은 객체를 다르게 사용하도록 조작할 수도 있음
```

```
// 변경할 수 없는 방법으로 복사하는 법
// 포인터가 아닌 객체를 복사
// Spread 연산자를 사용
// [3]
const person = {
  name: 'Max'
};

const secondPerson = {
  ...person
};

person.name = 'Manu';

console.log(secondPerson);

// [object Object] {
//   name: "Max"
// }

// 객체와 배열이 참조형 자료 타입임
// 만약 재할당한다면, 값이 아닌 포인터를 복사하는 것
// 진짜로 복사하고 싶다면, 새로운 객체를 생성해서 전체 객체를 복사하는 것이 아니라
// property를 복사해야함
```

20. 배열 함수 새로 고침

```
// [1] map() method
const numbers = [1, 2, 3];

const doubleNumArray = numbers.map((num) => {
  return num * 2
});

console.log(numbers);
console.log(doubleNumArray);

// [1, 2, 3]
// [2, 4, 6]
```

JS Array functions

```
// [1] map()
const array1 = [1, 4, 9, 16];
const map1 = array1.map(x => x * 2);

console.log(map1)
```

```
// Array [2, 8, 18, 32]


// [2] find()
const array1 = [5, 12, 8, 130, 44];

const found = array1.find(element => element > 10);

console.log(found)

// 12


// [3] findeIndex()
const array1 = [5, 12, 8, 130, 44];

const isLargeNumber = (element) => element > 13;

console.log(array1.findIndex(isLargeNumber));

// 3


// [4] filter()
const words = ['spray', 'limit', 'elite', 'exuberant', 'destruction', 'present'];

const result = words.filter(word => word.length > 6);

console.log(result)

// Array ["exuberant", "destruction", "present"]


// [5] reduce()
const array1 = [1, 2, 3, 4];

const initialValue = 0;
const sumWithinitial = array1.reduce(
  (accumulator, currentValue) => accumulator + currentValue,
  initialValue
);

console.log(sumWithinitial);

// 10


// [6] concat()
const array1 = ['a', 'b', 'c'];
const array2 = ['d', 'e', 'f'];
const array3 = array1.concat(array2);

console.log(array3);

// Array ["a", "b", "c", "d", "e", "f"]


// [7] slice()
```

```
const animals = ['ant', 'bison', 'camel', 'duck', 'elephant'];

console.log(animals.slice(2));
// Expected output: Array ["camel", "duck", "elephant"]

console.log(animals.slice(2, 4));
// Expected output: Array ["camel", "duck"]

console.log(animals.slice(1, 5));
// Expected output: Array ["bison", "camel", "duck", "elephant"]

console.log(animals.slice(-2));
// Expected output: Array ["duck", "elephant"]

console.log(animals.slice(2, -1));
// Expected output: Array ["camel", "duck"]

console.log(animals.slice());
// Expected output: Array ["ant", "bison", "camel", "duck", "elephant"]


// [8] splice()
const months = ['Jan', 'March', 'April', 'June'];
months.splice(1, 0, 'Feb');
// Inserts at index 1
console.log(months);
// Expected output: Array ["Jan", "Feb", "March", "April", "June"]

months.splice(4, 1, 'May');
// Replaces 1 element at index 4
console.log(months);
// Expected output: Array ["Jan", "Feb", "March", "April", "May"]
```