자바스크립트

함수 고급

목차

- Callback 함수
- Arrow 함수
- Method Chaining
- 즉시 호출
- 엄격 모드

Callback 함수

- 콜백함수
  - 함수의 레퍼런스 값을 파라미터로 전달하여 호출되는 함수
  - 자바스크립트의 콜백함수는 엄밀한 콜백 함수는 아님.

```
const callback_proc = function()
{
    console.log('callback_proc()');

}

function myFunction(proc)
{
    proc();
}

myFunction(callback_proc);
```

Callback 함수

- 콜백 함수의 응용
  - forEach() : 배열의 요소를 순회

```
const callbackProc = function(value, index, array)
{
    console.log(`${index} 번째 요소 : ${value}`);
}

const numbers = [11, 22, 33, 44, 55];
numbers.forEach(callbackProc);
```

```
const numbers = [11, 22, 33, 44, 55];
numbers.forEach(function(value, index, array) {
    console.log(`${index} 번째 요소 : ${value}`);});
```

  - forEach()를 MDN에서 찾고 callback 함수의 파라미터를 살펴봅시다.
  - 2단을 출력하는 코드를 배열, forEach(), callback함수로 만들어봅시다.

Callback 함수

- 콜백 함수의 응용
  - map() : 호출한 결과를 모아 새로운 배열을 반환
  - 만약 callback함수에서 리턴을 하지 않는다면 undefined 배열 생성

```
const numbers = [10, 20, 30, 40, 50];
const newNumbers = numbers.map(function(value, index, array) {
    return value + 5;});

console.log(newNumbers);
```

  - ['홍길동, '세종대왕', '김구', '안중근'] 배열의 각 요소에 "훌륭한 " 이라는 말머리를 붙이는 코드를 map()을 이용하여 작성하시오

Callback 함수

- 콜백 함수의 응용
  - filter() : 주어진 함수의 테스트를 통과하는 모든 요소를 모아 새로운 배열로 반환

```
const numbers = [1, 2, 3, 4, 5];
const newNumbers = numbers.filter(function(value, index, array) {
    if (value < 4)
    {
        return true;

    }  });

console.log(newNumbers);
```

  - 임의의 숫자 10개가 있는 배열을 만들고 홀수만으로 새로운 배열을 만드는 코드를 filter()을 이용하여 작성하시오

Arrow 함수

- 화살표 함수 표현 (Arrow Function Expression)
  - 전통적인 함수 표현을 간단하게 만든 sugar-code
  - 화살표함수는 메소드가 아님
  - 형식

    (param1, param2, …, paramN) => { statements }

    (param1, param2, …, paramN) => expression // 동일함: => { return expression; }

Arrow 함수

- 이해과정

```
const numbers = [10, 20, 30, 40, 50];
const newNumbers = numbers.map(function(value, index, array) {
    return value + 5

    });

console.log(newNumbers);
```

  - 1번 박스 생략

```
const newNumbers = numbers.map((value, index, array) => {
    return value + 5;});
```

  - 1번 , 2번 박스 생략

```
const newNumbers = numbers.map((value, index, array) => value + 5);
```

Method Chaining

- 메소드 체이닝(Method Chaining)
  - 임시객체를 이용하여 메소드를 연속적으로 호출하는 코딩기법

```
const numbers = [11, 22, 33, 44, 55];

const newNumbers = numbers.filter((value, index) => true)
                          .map((value, index) => value - (index+1));

console.log(newNumbers);
```

  - 아래 배열에서 짝수만 도출한 뒤 이를 두배하시오

```
const numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];
```

즉시 호출

- 즉시 호출
  - 임시 객체를 이용하여 함수의 선언과 동시에 호출하는 기법
  - 형식 : (function (){ })()
  - 복수개의 <script>에 대한 독립성 보장(이름충돌등)

```
(function dummy(x)
{
    console.log("DUMMY!!" + x);

})(10);
```

엄격 모드

- 엄격모드
  - 문법 사용에 대한 규칙 강화
  - 형식 : use strict
  - 일반적으로 즉시호출에 use strict사용, 다른 코드 블럭간의 간섭방지

정상: 변수선언을 안해도 됨

```
(function dummy()
{
    x = 10;
    console.log("DUMMY!!" + x);

})();
```

ERROR : 변수선언을 안하고 x 사용

```
(function dummy()
{
    'use strict'
    x = 10;
    console.log("DUMMY!!" + x);

})();
```