

# Ionic Study

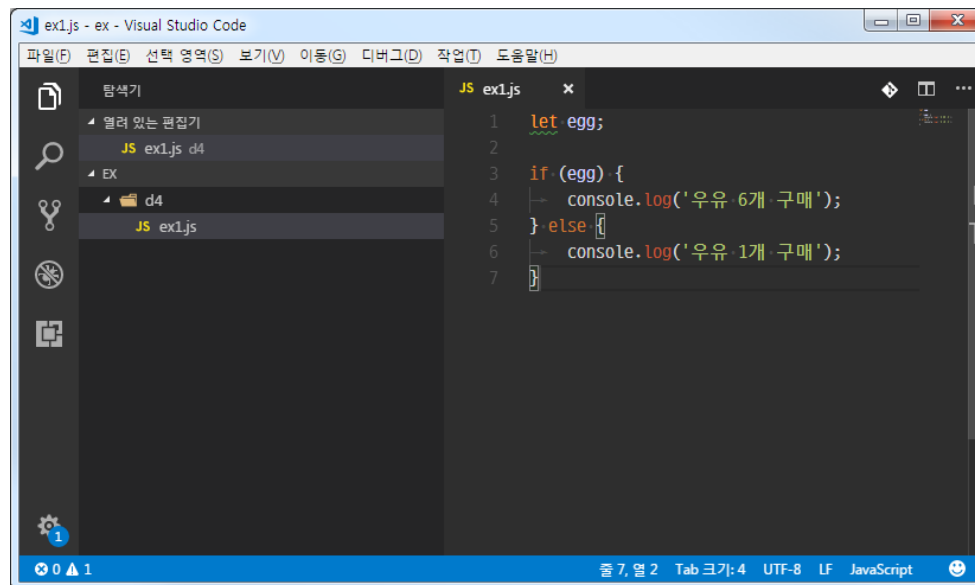
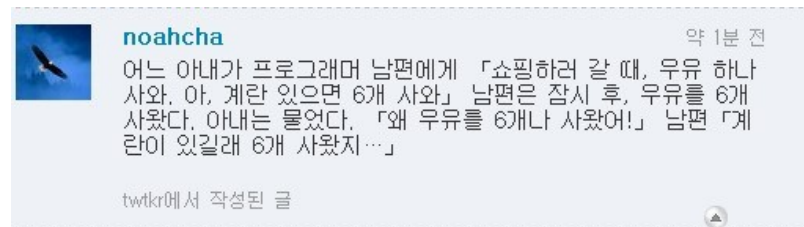
Day 4

# 오늘 할 것들

- 자바스크립트 기본 문법 알아보기
  - 제어구조, 함수

# 제어구조

- if...elseif...else



# 제어구조

- 루프
  - 조건에 따라 블록 안의 내용을 반복하는 구조.
  - for, while, do...while
- 흐름을 제어하는 예약어들
  - 제어 구조 내에서 흐름을 계속 이어갈지, 아니면 흐름을 끊고 제어 구조가 실행되고 있는 블록의 밖으로 탈출할지를 정의할 수 있는 예약어.
  - label, break, continue
- 예외 처리
  - 기본적으로 자바스크립트는 오류가 발생했을 때, 스크립트 실행을 중단함.
  - 이렇게 발생하는 오류를 사용자가 적절하게 재정의 가능.
  - try...catch...finally, throw

# 제어구조

- while 반복문

- 반복 조건이 거짓이 될 때 까지 반복.

- while (조건) { ... }

while (expr) {

- ① expr이 참이라면 블록 내부 진입

...

- ② 블록 내부 문장 실행

}

- ③ 다시 ①로 되돌아감

# 제어구조

- ex1.js



```
JS ex1.js x
1  let i = 0, j = 0;
2
3  while (i < 10) {
4      console.log(i);
5
6      j += (i * 2);
7      i++;
8  }
9  console.log('j = ' + j);
```

# 제어구조

- do...while 반복문
  - while 반복문은 조건이 참일 경우 반복이 실행됨. 따라서 조건에 따라 반복이 됨.
  - do...while 반복문은 조건의 참/거짓 여부를 떠나 반복문이 적어도 한 번 이상은 실행되게 해야 할 경우 사용.

# 제어구조

- do...while 반복문

- do { ... } while (조건);

- do {

- ...

- } while (expr);

- ① 블록 내부 문장 실행

- ② expr이 참이라면 반복문 실행. 거짓이라면 반복 종료



# 제어구조

- ex2.js

A screenshot of a code editor with two tabs: 'JS ex1.js' and 'JS ex2.js'. The 'JS ex2.js' tab is active and shows a JavaScript code snippet. The code consists of six lines: line 1 has 'let i = 0;' with 'i' underlined in green; line 2 is empty; line 3 has 'do {'; line 4 has ' console.log(i);' with a cursor at the start of the line; line 5 has '} while (i > 0);' with 'while' in orange; and line 6 is empty. The editor has a dark background and standard syntax highlighting.

```
JS ex1.js  JS ex2.js  x
1  let i = 0;
2
3  do {
4      console.log(i);
5  } while (i > 0);
6
```

# 제어구조

- for 반복문
  - 반복문에 필요한 제어 정보를 한 줄로 표현이 가능.
  - 초기화 식 지정, 반복 조건식 지정, 증감여부 지정이 가능.
  - 초기식은 단 한번만 수행되므로 주의가 필요.

# 제어구조

- for 반복문

- for (초기식; 조건식; 증감식) { ... }

```
for (init; expr; count) {
```

```
...
```

```
}
```

# 제어구조

- ex3.js

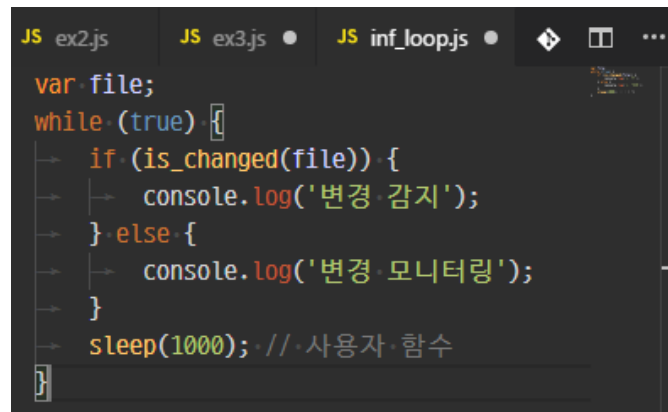
```
JS ex1.js JS ex2.js JS ex3.js
1 console.log('짝수 출력');
2 for (var i=1; i<=10; i++) {
3   if (i%2 === 0) console.log(i);
4 }
5 console.log('\n홀수 출력');
6 for (var i=1; i<=10; i++) {
7   if (i%2 !== 0) console.log(i);
8 }
```

# 제어구조

- 무한 루프
  - 반복 조건이 참인 상태에서, 앞으로도 반복 조건이 변하지 않는 경우 그 반복은 무한히 지속됨.
  - 의도적인 무한 루프, 의도적이지 않은 무한 루프
    - 지속적 모니터링 (의도적)
    - 루프 탈출조건 미충족 (비의도적)
  - node로 실행한 스크립트에서 무한루프 발생시, Ctrl+C를 눌러 스크립트 실행을 강제 종료할 수 있음 (꼭 무한루프뿐만이 아니라 실행중인 스크립트에는 모두 사용 가능)

# 제어구조

- 의도적인 무한루프 (예시)

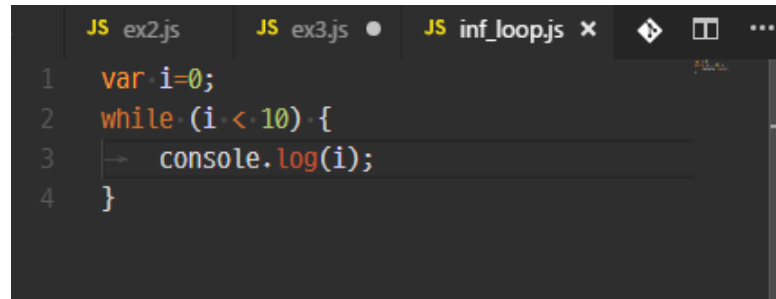


```
JS ex2.js JS ex3.js JS inf_loop.js
var file;
while(true) {
  if (is_changed(file)) {
    console.log('변경 감지');
  } else {
    console.log('변경 모니터링');
  }
  sleep(1000); // 사용자 함수
}
```

- 파일이 변경되면 변경 감지가 표시
- 그 외에는 변경 모니터링 표시

# 제어구조

- 의도적이지 않은 무한루프 (예시)



```
JS ex2.js JS ex3.js JS inf_loop.js x
1 var i=0;
2 while (i < 10) {
3   console.log(i);
4 }
```

The image shows a code editor with three tabs: 'JS ex2.js', 'JS ex3.js', and 'JS inf\_loop.js'. The 'inf\_loop.js' tab is active, displaying a JavaScript code snippet. The code consists of four lines: '1 var i=0;', '2 while (i < 10) {', '3 console.log(i);', and '4 }'. The code is written in a dark-themed editor with syntax highlighting.

- $i < 10$  일 경우 루프를 실행
- $i$ 는 계속 0이므로  $(i < 10)$ 은 true
- true가 반복되므로 무한루프 당침

# 제어구조

- 흐름 제어 - continue
  - 현재 반복을 종료하고 반복문의 처음으로 돌아가서 다음 반복을 실행하는 예약어.
  - 이해가 어렵다면 다음 예시를 참조.



# 제어구조

- ex4.js

```
JS ex3.js JS inf_loop.js JS ex4.js x
1 for(var i = 0; i < 10; i++){
2   if(i === 4){
3     continue;
4   }
5   console.log(i);
6 }
```

i	console.log
0	Y
1	Y
2	Y
3	Y
4	N
5	Y
6	Y
7	Y
8	Y
9	Y

# 제어구조

- 흐름 제어 - break
  - 현재 루프나 switch를 종료.
  - break가 실행되면 루프나 switch문을 마치고 다음 문장을 실행.

# 제어구조

- ex5.js

```
1 for (var i = 0; i < 10; i++) {  
2   if (i > 5) {  
3     break;  
4   }  
5   console.log(i);  
6 }  
7 console.log("I'm groot");
```

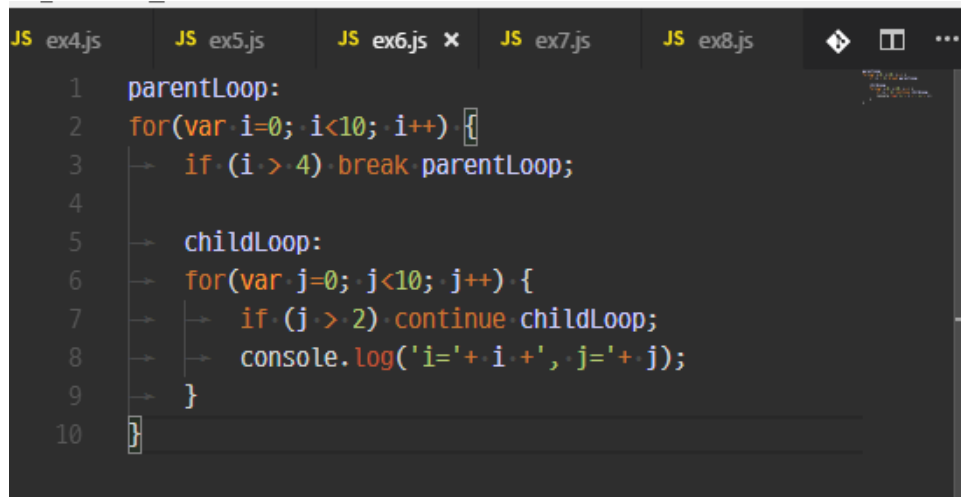
i > 5일 경우 루프를 종료

# 제어구조

- 흐름 제어 - label
  - 반복문과 함께 사용.
  - break나 continue를 사용해 반복문의 어느 위치에서 작업을 멈추고 다시 수행할지를 지정 가능.
  - <레이블명>: 로 사용
    - loop1:  
while(true) ...

# 제어구조

- ex6.js



```
JS ex4.js JS ex5.js JS ex6.js x JS ex7.js JS ex8.js
1 parentLoop:
2 for(var i=0; i<10; i++){
3     if(i>4) break parentLoop;
4
5     childLoop:
6     for(var j=0; j<10; j++){
7         if(j>2) continue childLoop;
8         console.log('i='+i+', j='+j);
9     }
10 }
```

# 제어구조

- 예외 처리 – try...catch...finally
  - 오류가 발생 = 예외(Exception)가 발생(Throw)
  - try 블록 안에서 예외 발생시 catch 블록 안의 내용을 수행
  - finally를 사용하면 예외 발생여부와 상관없이 항상 실행.

# 제어구조

- ex7.js

```
1 try {  
2   const str = 'Hello';  
3   str = 'Ionic';  
4  
5   console.log('str = ' + str);  
6 } catch (e) {  
7   console.log('예외 발생!');  
8   console.log(e);  
9 } finally {  
10  console.log('예외 여부와 상관없이 표시');  
11 }
```

console.log(e)를 사용하면 예외  
객체 e를 확인 가능.

# 제어구조

- 예외 처리 – throw
  - 사용자 정의 예외를 발생시킬 수 있는 키워드
    - 예외를 던지다(Throw)
  - throw 이후의 명령은 실행되지 않고, 곧바로 catch 블록으로 제어 이동 후 예외 처리 수행.



# 제어구조

- ex8.js

```
JS ex4.js JS ex5.js JS ex6.js JS ex7.js JS ex8.js x
1  try-{
2    for(var i=0; i<10; i++){
3      if(i>5) throw '예외 발생! i = '+ i;
4      console.log(i);
5    }
6  } catch(e) {
7    console.log(e);
8  }
```

# 함수

- 독립적으로 분리된 로직
- 프로그램 수준에서 미리 정의되어 있거나 사용자 정의에 의해 만들어진 실행단위.
  - 별도의 값(결과) 반환 없이 일련의 동작을 수행 (서브루틴)
  - 일련의 동작을 수행하고 그 값(결과)을 반환 (함수)
- 자바스크립트의 함수는 1급 객체이다.
  - 함수를 변수나 데이터 구조 안에 담을 수 있으며 인자로 전달할 수 있고 반환 값으로도 사용할 수 있으며, 실행 단계에 생성될 수 도 있음.
- 코어 라이브러리와 사용자 정의 함수로 구분
  - 코어 라이브러리 : 자바스크립트가 기본적으로 제공하는 함수들
  - 사용자 정의 함수 : 사용자가 직접 정의하는 함수
- 선언식과 표현식으로 함수를 나타낼 수 있다.

# 함수

- 함수의 정의
  - function 키워드를 사용
  - function 함수명 (매개변수) { ... }
  - 함수 선언부 내에 return 키워드로 반환할 값을 지정 가능.  
return을 사용하면 해당 키워드 밑의 문장은 모두 무시됨.
    - '해당 함수를 종료하고 어떤 값을 반환한다' 라는 명령이라 생각하면  
Good
- 함수의 호출
  - 함수명(매개변수);

# 함수

## 함수 정의

```
function func(str) {  
    return 'Hello'+ str;  
}
```

## 정의한 함수를 호출

```
var name = 'John',  
    message;  
message = func(name);  
  
console.log(message);
```

# 함수

- 함수의 종류
  - 선언적 함수
  - 익명 함수
  - 중첩 함수
  - 콜백 함수
  - 화살표 함수

# 함수

- 선언적 함수
  - anytime use.
  - 블록 바깥에서 선언된 함수는 전역(global) 함수의 성격을 띄게 됨.
    - 전역 함수로 선언되면 선언한 위치에 관계없이 어디서나 해당 함수를 호출 가능.
    - 범용성(어디서나 사용) OK
    - 은닉성(필요한 경우에만 사용) NG.

# 함수

- ex9.js – 선언적 함수 예제



```
JS ex4.js JS ex5.js JS ex6.js JS ex9.js x JS ex7.js
1 sayHello();
2
3 function sayHello() {
4   console.log('Hello?');
5 }
6
7 sayHello();
```

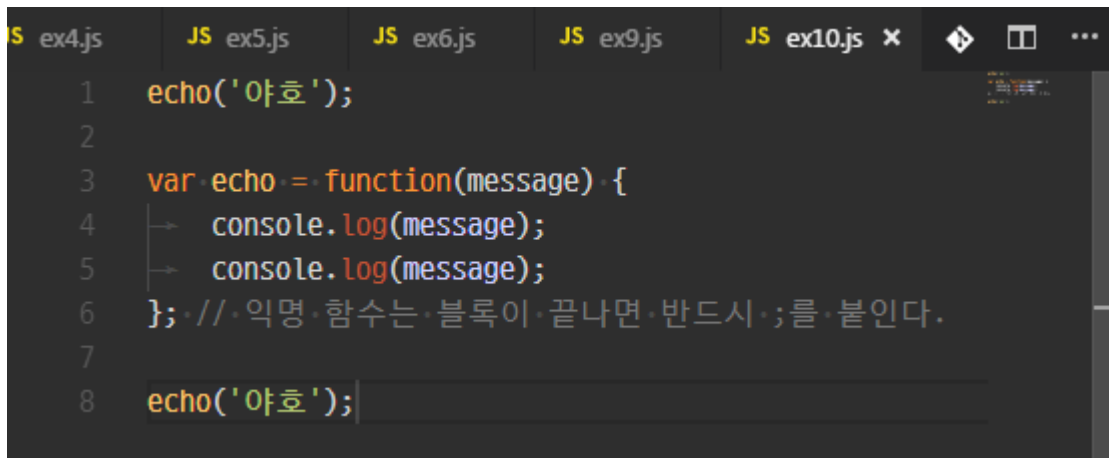
# 함수

- 익명 함수
  - 함수명이 생략된 함수.
  - 이름 대신 변수로 호출
  - 선언이 끝나면 블록을 닫는 기호 옆에 반드시 ;를 붙여줌.
  - 해당 함수가 선언된 이후부터 사용이 가능.



# 함수

- ex10.js – 익명 함수 예제
  - 이 코드가 가진 문제점과 해결 방법은?



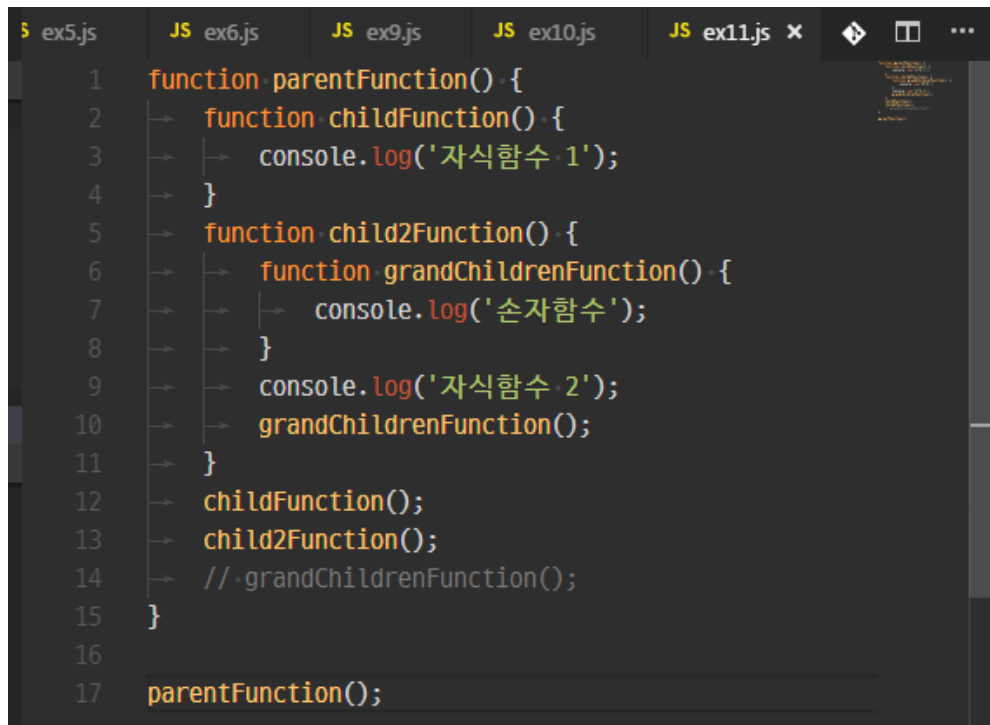
```
JS ex4.js JS ex5.js JS ex6.js JS ex9.js JS ex10.js x
1 echo('야호');
2
3 var echo = function(message) {
4   console.log(message);
5   console.log(message);
6 }; // 익명 함수는 블록이 끝나면 반드시 ;를 붙인다.
7
8 echo('야호');
```

# 함수

- 중첩 함수
  - function in function
  - 함수 내부에 또 다른 함수를 가지는 형태.
  - 특정 함수 내부에서 사용되는 기능을 함수형태로 정의할 때 사용하면 좋음 (은닉성 OK)
  - 부모 함수 내에 있는 변수의 참조 가능.

# 함수

- ex11.js – 중첩 함수 예제
  - 14번째 줄의 주석 (//)을 풀면 무슨일이 생길까요>?



```
5 ex5.js JS ex6.js JS ex9.js JS ex10.js JS ex11.js x
1 function parentFunction() {
2   function childFunction() {
3     console.log('자식함수 1');
4   }
5   function child2Function() {
6     function grandChildrenFunction() {
7       console.log('손자함수');
8     }
9     console.log('자식함수 2');
10    grandChildrenFunction();
11  }
12  childFunction();
13  child2Function();
14  // grandChildrenFunction();
15 }
16
17 parentFunction();
```

# 함수

- 콜백 함수
  - 함수를 매개변수로 사용하는 형태.
  - 함수명 또는 익명함수를 매개변수로 전달.
  - 비동기 프로그래밍시 많이 사용
    - ex) jquery ajax

# 함수

- ex12.js – 콜백 함수 예제

```
JS ex12.js x
1  var rest = function(name) {
2      console.log(name + ' is now resting. ');
3  };
4
5  function doWork(name, work) {
6      console.log('Working hard ' + name + '!');
7      if (typeof work === 'function') work(name);
8  }
9
10 doWork('Jeff', function(name) {
11     console.log(name + ' is now working. ');
12 });
13
14 doWork('Jane', rest);
```

# 함수

- 화살표 함수
  - 익명함수가 축약된 형태.
  - ES2015부터 사용 가능.
  - 다른 함수 표현식에 비해 제한된 형태를 가짐.
  - 함수 표현 블록을 생략하면 return을 쓰지 않아도 반환값을 자동으로 돌려줌

# 함수

- ex13.js – 화살표 함수 예제

```
JS ex12.js JS ex13.js x
1 // 화살표 함수 (기본형)
2 const square = n => {
3   return n*n;
4 };
5
6 // 화살표 함수 (블록 생략)
7 const quad = n => square(n)*n;
8
9 console.log('2의 2승은? ' + square(2));
10 console.log('2의 3승은? ' + quad(2));
11
12 // 화살표 함수 (콜백에서)
13 function doCallback(func) {
14   if (typeof func === 'function') func();
15 } doCallback(() => { console.log('10의 2배는? ' + 10 * 2); });
```

오늘은 여기까지~

See you next day!