12강

Q. 아래의 빈칸과 서로 차이점을 서술하시오

```
function add(x, y) { // x,y 을 뭐라하는가 _인자__
return x + y;
}
add(2,5); // 들어가는 값에 대한 단어 : _인수_
```

인수(Argument): 함수를 호출할 때 건네주는 변수

인자(Parameter): 매개변수로도 불리며 함수에서 정의되어 사용되는 변수

**Q. 본인이 생각하기에 이상적인 개발자는 어떤 형태인가?
해당 질문에 대해서 진지하게 한번쯤은 고민할 만 합니다. 보통 프로그래머로 취업을 한다면 인터뷰에서 무조건 물어보는 내용이기도 합니다.**

피드백이 정확하고 빠른 사람.

함께 일하는 사람인 만큼 서로가 서로의 수준을 잘 파악해야 하고 그에 맞는 업무를 분배하는 것이 효율적인 일처리라 생각합니다.

또한 모르는 부분이 어디고 어디가 안되는지 무엇이 부족한지 빠르게 피드백 해주며 그 부분에 대해 자신이 해결할 수 있는지 아니면 다른 사람에게 부탁을 해야하는지 파악이 빠른 개발자가 이상적인 개발자라 생각합니다.

Q. 선언문에서는 함수 이름을 생략할 수 없다. 만약 함수 이름을 생략하면 나오는 에러는 어떤건지 확인해보세요.

```
//input
function(x, y) {
    return x+y;
}
coonsole.log(add(2, 5))

//output
Uncaught SyntaxError: Function statements require a function name
```

Q. { } 는 블록문일까 객체 리터럴일까? 본인의 생각을 쓰고 그 이유에 대해서 서술하시오.

블록문

복합문이라 불리기도 하며 0개 이상의 구문을 묶을 때 사용한다. 한 쌍의 중괄호로 구성되며 선택적으로 레이블을 붙일 수 있으며 하나의 문을 기대하는 곳에서 다수의 문을 실행할 수 있다. if else, for문에 함께 사용된다.

```
//블록문
{
  StatementList;
}
```

```
//레이블된 블록문
LabelIdentifier: {
  StatementList;
}
```

객체 리터럴(object literal)

사람이 이해할 수 있는 문자나 약속된 기호를 사용해 값을 생성하는 표기법을 말함.

객체 리테럴의 중괄호는 코드 블록을 의미하는 것이 아닌 값으로 평가되기에 닫는 괄호 뒤에는 세미콜론을 붙인다. 중괄호 내에 0개 이상의 프로퍼티를 정의

```
//object literal
var hello = {
  name : 'NANA', //property
  intro : function() {
    console.log(`My name is ${this.name}`); //object literal
  }
};
```

위의 두 개념을 정의하여 보았을 때 블록문은 객체 리터럴의 부분집합으로 보이므로 {}는 블록문이라 생각한다.

Q. 하단의 에러는 왜 날까?

```
var add1 = (function() {
  var a = 10;
  return function (x, y){
    return x + y + a;
  };
}()); //-> bold처리한거 이거 뭐임?

console.log(add1(1,2)); // 13

var add2 = (function() {
  var a = 10;
  return new Function('x', 'y', 'return x + y + a;')
}());

console.log(add2(1,2)); // ReferenceError: a is not defined
```

new연산자와 함께 호출되면 함수 객체를 생성해서 반환을 하는데 x, y는 생성했지만 a는 따로 생성을 하지 않았기 때문이다.

Q. 아래 함수를 실행해보고 결과 값을 적으시오.

```
function add(x, y){
console.log(x,y);
return x+y;
}
add(2, 5);
console.log(x, y); //2 5

function add(x, y) {
return x + y;
}
console.log(add(2)); //NaN

function add(x, y) {
console.log(arguments);
return x + y;
}
console.log(add(2,5,10)); //7
```

Q. 해당 단원은 여러분을 위해 비어드렸습니다. 반드시 공부해와 주세요. 문서 형태는 마음대로 지만 다만 본인이 공부했다는 티는 나셔야 합니다.

<Call by Reference, Call by Value에 대해 공부하자>

Call by Value : 값에 의한 호출, 값의 복사

복사된 값을 인자로 넘겨 매개변수로 전달한다.

장: 복사하여 처리하기에 안전하며 원래의 값이 보존된다.

단: 복사를 하기에 메모리 사용량이 늘어난다.

Call by reference : 참조에 의한 호출, 주소의 복사

실제 데이터가 존재하는 주소를 가리키는 주소값을 인자로 넘겨 매개변수로 전달.

장: 복사하지 않고 직접 참조를 하기에 빠르다

단: 직접 참조를 하여 원해 값이 영향을 받는다

js에선 기본형과 참조형(객체) 구분없이 모두 call by value를 사용한다.

→ 참조형이여도 참조형의 주소값을 복사하여 매개변수로 전달함.

Q. 재귀함수로 팩토리얼을 구현해보시오. 그리고 해당 코드에 대한 리뷰를 해 보세요.

```
function factorial(n) {
    console.log(n) //확인용

    if (n == 1) {
        return 1;
    } return n * factorial(n - 1);

}
//factorial(3);
console.log(factorial(3)); //팩토리얼 출력

//변하지 않는 'n'과 1까지 계속되는 'n - 1'이 필요하여 인자를 2개로 늘려해보았지만
//1개로 충분히 할 수 있다 판단하여 1개로 제작.
```

Q. callback 지옥이라는 말이 유명하다. 직접 지옥을 만들어보자.
그리고 calllback 지옥이 왜 위험한지 서술하시오.

callback function : 파라미터로 함수를 전달받아, 함수의 내부에서 실행하는 함수.

callback Hall : 비동기 호출이 자주 일어나는 프로그램. 함수의 매개변수로 넘겨지는 콜백 함수가 반복되어 코드의 들여쓰기 수준이 감당하기 힘들어질 정도로 깊어지는 현상.

```
console.log("2의 배수 출력")

function add(x, callback) {
    let sum = x + x;
    console.log(sum);
    callback(sum);
}

add(2, function (result) {
    add(result, function (result) {
        add(result, function (result) {
            add(result, function (result) {
                add(result, function (result) {
                    add(result, function (result) {
                        add(result, function (result) {
                            add(result, function (result) {
                                add(result, function (result) {
                                    add(result, function (result) {
                                        add(result, function (result) {
                                            add(result, function (result) {
                                                add(result, function (result) {
                                                    add(result, function (result) {
                                                        add(result, function (result) {
                                                            add(result, function (result) {
                                                                add(result, function (result) {
                                                                    add(result, function (result) {
                                                                        add(result, function (result) {
                                                                            add(result, function (result) {
                                                                                add(result, function (result) {
                                                                                    add(result, function (result) {
```

```
                                                                                        add(result, function (result) {
                                                                                            add(result, function (result) {
                                                                                                add(result, function (result) {
                                                                                                    add(result, function (result) {
                                                                                                        add(result, function (result)
                                                                                                            add(result, function (resul
                                                                                                                add(result, function (r
                                                                                                                    add(result, functio
                                                                                                                        add(result, fu
                                                                                                                            add(result,
                                                                                                                            })
                                                                                                                        })
                                                                                                                    })
                                                                                                                })
                                                                                                            })
                                                                                                        })
                                                                                                    })
                                                                                                })
                                                                                            })
                                                                                        })
                                                                                    })
                                                                                })
                                                                            })
                                                                        })
                                                                    })
                                                                })
                                                            })
                                                        })
                                                    })
                                                })
                                            })
                                        })
                                    })
                                })
                            })
                        })
                    })
                })
            })
        })
    })
})
```

Q. 아래의 코드 중 어떤 것이 순수 함수이며 어떤 것이 비순수 함수인지 서술하시오

```
var count = 0;
function increase(n) { //순수함수
return ++n;
}
count = increase(count);
console.log(count);
count = increase(count);
console.log(count);

var count = 0;
function increase() { //비순수함수
return ++count;
}
count = increase(count);
console.log(count);
count = increase(count);
console.log(count);
```

추가과제1

콜백 지옥을 해결하기 위한 예방법, 대처법 찾아오기

promise 사용 : 비동기 방식에서 resolve(해결), reject(실패)를 분리하여 메소드를 수행.

resolve: then으로 돌아간다.

reject: catch로 돌아간다.

async & await 적용 : promise 기반 코드를 더욱 쓰기 쉽고 읽기 쉽게 하기 위하여 나옴.

async: 함수 앞에 붙여 Promise를 반환. promise가 아니어도 promise로 감싸서 반환

awaite: promise앞에 붙어서 promise가 다 처리될 때까지 기다리는 역할, 결과는 그 후에 반환

추가과제2

배운 함수들 사용해서 120줄 이상 코드를 짜오기. (몇 개는 리턴 값을 주고 몇 개는 리턴 값을 안주는 형식. 재귀랑 callback들어가는 함수, 화살표 생성방식도 사용하셈)

```
//리턴 값이 있는 예제
//callback이 있는 예제
//재귀함수가 들어가는 예제
// 화살표 함수가 들어가는 예제


console.log("리턴 값 예제1")

let ex_1 = 10;
function example() {
    console.log("first ex_1 : %d", ex_1);
    ex_1 += 1
    return ex_1;
}
console.log("final ex_1 : %d\n", example(ex_1))


console.log("리턴 값 예제2")

let ex_2 = 5
function example2() {
    console.log("first ex_2 : %d", ex_2);
    ex_2 *= ex_2
    console.log("second ex_2 : %d", ex_2)
    return ex_2;
}
console.log("final ex_2 : %d\n", example2(ex_2))


console.log("재귀함수 예제1")

function ex3(n) {
    console.log(n);
    if (n >= 5) {
        return 1
    }
    return n + ex3(n + 1)
}
console.log(ex3(3))


console.log("재귀함수 예제2")

function ex4(n, i) {
    console.log("%d * %d = %d", n, i, n * i);
    if (i >= 9) {
        return 1
    }
    return ex4(n, i + 1)
}
ex4(4, 1)


console.log("재귀함수 예제3")

function ex5(n, i) {
    console.log("%d * %d = %d", n, i, n * i);
    if (i < 9) {
        return ex5(n, i + 1)
    }
    else if (n < 9) {
        return ex5(n + 1, 1)
    }
}
ex5(2, 1);


console.log("call back이 들어가있는 예제1")
```

```
function ex6(x, y, result) {
    result(x + y)
}

function result(result) {
    console.log(result)
}
ex6(3, 2, result)


console.log("\ncall back이 들어가있는 예제2")

function ex7_add(a, b) {
    return a + b
}

function ex7_sub(a, b) {
    return a - b;
}

function ex7_mul(a, b) {
    return a * b;
}

function ex7_divi(a, b) {
    return a / b;
}

function ex7(a, b) {
    console.log("add : %d", ex7_add(a, b))
    console.log("sub : %d", ex7_sub(a, b))
    console.log("mul : %d", ex7_mul(a, b))
    console.log("divi : %d", ex7_divi(a, b))
}
ex7(10, 2)

console.log("\n화살표 함수가 들어간 예제1")

var add = (a, b) => a + b;
console.log(add(3, 2));

var sub = (a, b) => a - b;
console.log(sub(3, 2));

var mul = (a, b) => a * b;
console.log(mul(3, 2));

var divi = (a, b) => a / b;
console.log(divi(3, 2));


console.log("\n화살표 함수가 들어간 예제2")

console.log("원의 방정식 구하기")
function circle(a, b) {
    console.log("원의 중심 : %d %d", a, b)
    return theCircle(a, b)
}

var theCircle = (a, b) => {
    let result = a * a + b * b;
    return result
}

console.log("원의 방정식 : %d", circle(3, 4))
```