[11/08] Javascript : function, return, arguments, callback

- 함수(function) : 미리 약속된 계산식
  - 정의 : 작업을 수행하거나 값을 계산하는 명령문의 집합인 프로시저(procedure)와 유사, 프로시저가 함수로 쓰이려면 입력을 반드시 받아야 하고 입력과 명확한 관계가 있는 출력을 반환해야 한다.
  - 형식 : 함수명, 매개변수들, 괄호
    - 함수명
    - 함수의 매개변수들, 괄호로 묶고( ) 쉼표로 구분 ,
    - 함수를 정의하는 JavaScript 문으로 중괄호로 묶음 { }

  - 기본 함수 및 설정

```
function hello(name) {
    console.log("hello"+name);
}

let a = hello("kmh")        hellokmh
```

    - function : 함수 선언
    - hello : 함수명
    - ( ) : 매개변수(parameter) // 함수명( ) 를 의미
    - { } : 중괄호안의 내용으로 해당 함수의 기능을 정의

    - 이제 function의 선언과 함수명, 매개변수, 중괄호로 인하여 hello( ) 의 ( ) 안에 값을 입력시 자동으로 “hello”+( ) 가 실행
    - 즉, 매개변수()의 값을 { } 의 조건으로 변경
      - hello(name) -> {console.log(“hello”+name)}
      - hello(“kmh”) -> hellokmh 출력

    - 함수 및 변수의 정의후, 함수의 호출을 해야 작동

```
function add(a,b){
    console.log(a+b);
}

let a = 1;
let b = 2;
add(a,b)                    3
```

```
function add(a,b){
    console.log(a+b);
}
add(1,2)                    3
```

- 함수정의의 조건문

```
function add(a,b){
    // if(b==undefined){b=5}
    return a + (b||1);
}
console.log(add(3,));       4
```

```
function add(a,b){
    if(b==undefined){b=5}
    return(a+b);
}                           8
console.log(add(3,));
```

  - 특정 매개변수가 있으면 시행, 아니면(or) 1을 시행
    - or(||) : True 면 b, False 면 1
  - 특정 매개변수가 undefined이라면 해당 조건문(b=5) 시행
    - !b : b==undefined 와 동일

- return : 지정된 값을 반환

```
function add(a,b){
    return a+b;
}
console.log(add(1,2));
```
3

  - function add( ) 선언으로 인하여 a,b 는 a+b 로 변환

- 자료형 미구분 : 자바스크립트는 자료형 지정이 없음

```
function add(a,b){
    return(a+b);
}
console.log(add("papa",4));
```
papa4

  - string 과 num 의 + 형태가 출력됨
  - 추후, 자료형 지정이 가능한 타입스크립트를 학습 예정

- 매개변수의 부족 : NaN이 발생

```
function add(a,b){
    // if(b==undefined){b=5}
    return(a+b);
}
console.log(add(3,));
```
NaN

  - 예시

```
function power(a,b,c){
    if(!b){b=5;}
    if(!c){c=6;}
    return a*b*c;
}
console.log(power(1,"", ));
```
30

```
function power(a,b,c){
    return a*(b||5)*(c||6);
}
console.log(power(1,"", ));
```
30

    - 좌측이 선호적
    - (b||5) : b 가 True(존재하면) b 를 시행, b가 False(없으면) 5를 시행

- 가변 매개변수 : 매개변수를 미지정

```
function power(...a){
    let result = 0;
    for(let i=0; i<a.length; ++i)
    result += a[i];
    return result
}
console.log(power(1,2,4));
```
7

```
function power(){
    let result = 0;
    for(let i=0; i<arguments.length; ++i)
    result += arguments[i]
    return result
}
console.log(power(1,2,4));
```
7

  - …a : 모든 a를 의미
  - a.length : 자료를 배열의 길이만큼 받음
  - arguments : 가변 매개변수, 배열로 자료값을 받음

- callback : 이전에 완료한 과정을 언제든 호출 가능

```
function add(a,b){
    return a+b;
}
let a = add(3,4); //call
console.log(a);   //back
```

```
function add(a,b){
    return a+b;
}

let f = add;
console.log(typeof f);

let b = f(3,4);
console.log(b);
```
A : 처리에 1분 소요

B : 처리에 5분 소요

C : 처리에 1분 소요

  - A,B,C의 처리속도가 상이하므로, 빠른 애들은 빨리 처리하기 위함
  - 콜백은 SAS 호출에서 DAS 호출이 가능, 처리 순서에 지연시간 부여 가능
    - 순차 접근 저장 매체(SASD, Sequential Access Storage Device) -tape
    - 직접 접근 저장 매체(DASD, Direct Access Storage Device) - cd
  - 전면(foreground)는 짧은 과정을(우선순위, callback)
    후면(background)는 늦은 과정을 나누어 추후 늦은 과정을 전면으로
  - 콜백은 함수 자체가 자료형이자, 함수 자체의 전달이기도 함

  - [1]자바스크립트는 코드를 위에서 아래로 순차적으로 실행합니다.(SASD) 그러나, 코드가 다른 행위가 일어난 뒤에 실행되는 경우도 있고 순차적으로 실행되지 않을 때도 있습니다. 이런 걸 비동기 프로그래밍이라고 합니다.
  - 콜백은 태스크가 끝나기 전에 함수가 실행되지 않는 것을 보장합니다. 콜백은 그 태스크가 끝난 직후에 실행될 것입니다. 콜백은 비동기 자바스크립트 코드를 작성할 수 있게 해주고 여러 문제와 에러들로부터 안전히 지켜줍니다.
  - 자바스크립트에서 콜백 함수를 만드는 방법은 어떤 함수의 파라미터로써 함수를 넘기고 어떤 행위나 태스크가 완료된 직후에 콜백 함수를 호출하는 것입니다.

[1] 자바스크립트의 콜백 함수 / 콜백 함수(Callback) 개념 & 응용 - 완벽 정리

- callback 사용

```
function test1(f){
    let result = f(3,4);
    console.log(result);
}

function add(a,b){
    return a+b;
}

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

test1(add);          7
test1(multiply);     12
```

A

B

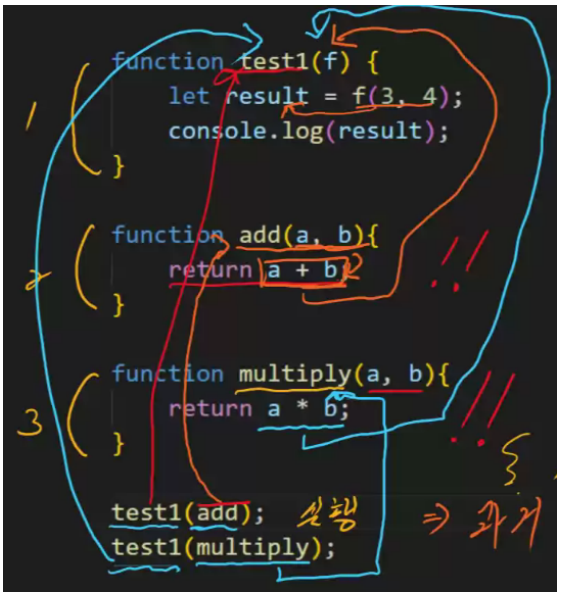
C

D

```
function test1(f){
    let result = f(3,4);
    console.log(result);
}

let add = (a,b) => {
    return a+b
}

let multiply = (a,b) => {
    return a*b
}

test1(add);          7
test1(multiply);     12
```

function / redirection

  - function 는 함수 지정이 많아지는 단점, redirection 대형플랫폼 수준에서 지연가능성




```
function test2(f){
    let result = f(5,8);
    console.log(result);
}

test2((a,b) => {return a+b});   13
test2((a,b) => {return a*b});   40
```

  - D는 callback 함수를 의미
    - 과거에 정의된 함수를 호출하여 사용
    - 다른 함수의 파라미터의 data로 전달되게 호출되는 함수
  - test1(f) 와 f(3,4) 의 f는 동일하게 작성
    - D의 test1(add) 가 실행
    - D의 test1이 A의 test1으로
    - D의 add 는 B의 a,b에서 a+b 로
    - 이것은 다시 A의 (f) 로
    - D의 test1(multiply) 가 실행
    - D의 test1이 A의 test1으로
    - D의 multiply 는 C의 a,b에서 a*b로
    - 이것은 다시 A의 (f)로

- setTimeout[2] : 시간 관련 callback 함수
  - 기본 코드

```
function sayHi() {
    console.log("안녕하세요.");
}
setTimeout(sayHi, 1000);
```
```
안녕하세요.
```

    - 구조 : setTimeout(func|code, [delay], [arg1], [arg2], ...)
      - func|code : 위 이미지에서는 sayHi 에 해당
      - delay : 1000 이면 1초, 2000이면 2초후 출력

  - 응용 코드

```
function printTime(msg){
    console.log(msg, new Date());
}

setTimeout(printTime, 1000, "1초 후");
setTimeout(printTime, 2000, "2초 후");
setTimeout(printTime, 3000, "3초 후");
```
```
1초 후 2023-11-08T05:15:46.753Z
2초 후 2023-11-08T05:15:47.754Z
3초 후 2023-11-08T05:15:48.757Z
```

    - printTime : 명사와 명사의 조합시 뒷 명사의 첫 글자는 대문자로
    - 짧게 걸리는 애들 부터 출력됨
    - new Date() : 내 컴퓨터의 시간과 동기화
      - new : 생성자라서 새롭게 붙여줘여 함
      - Date : 내 컴퓨터 기준의 날짜와 시간
    - setTimeout(func|code, [delay], [arg1], [arg2], ...) 의 구조로 인하여
      - [arg1] : 1초 후

- 자바스크립트의 타이머 함수

함수 이름	설명
setTimeout(함수, 시간)	일정 시간 후 함수 실행
setInterval(함수, 시간)	일정 시간 간격으로 함수 반복 실행
clearTimeout(id)	실행되고 있는 timeout 을 중지
clearInterval(id)	실행되고 있는 interval 을 중지

[2] 자바스크립트의 **setTimeout()**과 **setInterval()** 함수

- object : { } , 자료의 주소값을 기억하고 있는 집합

```
let person = {name:"김민현", age:25};
console.log(person);
console.log(person.name);
console.log(person.age);
```

```
{ name: '김민현', age: 25 }
김민현
25
```

오브젝트에서 작성

```
let person1 = {}

person1.name = "김민현";
person1.age = 17;

console.log(person1);
console.log(person1.name);
console.log(person1.age);
```

```
{ name: '김민현', age: 17 }
김민현
17
```

빈 오브젝트 내 삽입

  - object : person ex) 상품 목록, 학생의 이름과 번호
  - elements : name:"김민현", age:25

- object 의 callback

```
function createPerson(s,i){
    return {name:s, age:i};
}
let a = createPerson("김민현", 15);
console.log(a);
```

```
{ name: '김민현', age: 15 }
```

- 자바스크립트의 object 의 name 은 유일하게 작성

```
function createPerson(s,i){
    return {name:s, age:i};
}
let person1 = createPerson("김민현", 17);
let person2 = createPerson("박지영", 17);
let p = person1;

console.log(person1);
console.log(person2);

console.log(person1 == person2);
console.log(person1 == p);

function equals(person1, person2){
    return person1.name == person2.name || person1.age == person2.age
}

console.log(equals(person1, person2));
```

```
{ name: '김민현', age: 17 }
{ name: '박지영', age: 17 }

false
true

true
```

  - 자바스크립트는 동일한 이름 있으면 못찾으므로 유일하게 작성해야 함(?)

- array 와 object 의 출력

```
let ps1 = {name:"박지영", age:48};
let ps2 = {name:"박지영", age:17};
let ps3 = {name:"박태민", age:20};
let ps4 = {name:"박택석", age:26};

let person = [ps1, ps2, ps3, ps4];
console.log(person);

for(let i=0; i<person.length; i++){
    console.log(person[i])
}
```

```
[
  { name: '박지영', age: 48 },
  { name: '박지영', age: 17 },
  { name: '박태민', age: 20 },
  { name: '박택석', age: 26 }
]
{ name: '박지영', age: 48 }
{ name: '박지영', age: 17 }
{ name: '박태민', age: 20 }
{ name: '박택석', age: 26 }
```

    - console.log(person[i].name) : 이름만 순서대로 출력

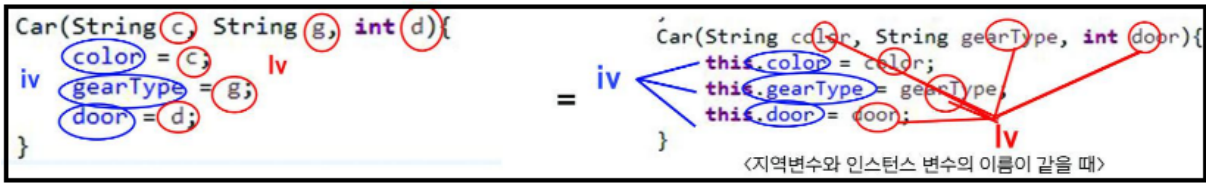
- 참조변수 this
    -

```
let rectangle = {
    width:5, height:7, area:function(){
        return this.width * this.height;
    }
};
console.log(rectangle.area());
```

```
35
```

    - function의 함수명, 파라미터가 없는 상태
    - method : width:5, height:7
    - 자바스크립트에서 메소드 생성시 변수명 앞에 this 는 반드시 붙임
        - ex) this.width , this.height

- 참조변수 this[3] : 인스턴스가 자기 자신을 참조하는데 사용하는 변수
  - 특징
    - 인스턴스 주소가 저장
    - 자신을 포함한 가장 가까운 객체를 출력
    - 인스턴스 메서드에서 사용 가능, 클래스 메서드에서는 사용 불가
    - 모든 인스턴스 메서드에서는 this 참조 변수가 숨겨진 지역 변수로 존재
  - 용도 : 함수의 지역변수(lv)와 인스턴스 변수(iv)의 이름이 같은 경우 구별할 때 사용



  - 좌측은 원래 this.color, this.gearType, this.door 로 가능하나 생략한 것
    - 같은 클래스 안이므로 생략이 가능
  - 우측 코드에서는 lv 와 iv 의 이름은 같음
  - 만약 iv가 this 없이 사용한다면 우측에 있는 모든 변수는 lv가 되는 것
  - 참조 변수 this 는 lv 와 iv 를 구별하기 위해 사용

- 생성자 this() : 같은 클래스에서 다른 생성자 호출할 때 사용
  - 특징
    - 다른 생성자의 호출은 생성자의 첫 줄에서만 사용에 주의
    - 생성자 코드내에서만 호출 가능
  - 용도 : 생성자 호출, 코드의 중복 제거

```
class Car {
    String color;
    String gearType;
    int door;

    Car() {
        color = "white";
        gearType = "auto";
        door = 4;
    }

    Car(String c, String g, int d) {
        color = c;
        gearType = g;
        door = d;
    }
}
```

* 코드의 재사용성을 높인 코드

```
Car() {
    //Card("white","auto",4);
    this("white","auto",4);
}
```

```
Car() {
    door = 5;
    this("white","auto",4);
}
```

[3] 객체지향 : 참조변수 this, 생성자 this()