Javascript -2

함수와 객체

함수

함수

- 호출하여 반복적으로 수행할 수 있는 statement의 집합
- 함수를 선언하고 이후 함수이름( ) 의 방법으로 호출하여 사용할 수 있다.
- 함수를 실행하는데 필요한 매개 변수(parameters, arguments)를 전달할 수 있고
- 함수의 실행 결과 값을 반환 받을 수 있다.

함수

함수 선언

- 함수 선언문(Statement)
  - function 이름(매개변수,…) { statements }

```
function square(n) {
    return n*n
}
```

  - return이 생략될 경우 undefined가 반환된다.
  - 함수 이름 뒤에 (매개변수,…) 를 붙여 실행한다.

```
console.log(square(3))
```

함수

함수 선언

- 함수 선언문(Statement)
  - Hoisting이 적용된다.

```
1  hello()
2
3  function hello(){
4      console.log('Hello')
5  }
```

Hoisting

- var, function 키워드로 선언된 변수와 함수의 '선언' 을 스코프의 제일 앞으로 끌어 올리는 동작. 변수의 할당은 하지 않는다. (undefined 처리)

```
1   console.log(cat)
2   var cat='cat'
3   var cat='new cat'
4   console.log(cat)
```

```
1   console.log(hello())
2
3   function hello(){
4       return 'Hello'
5   }
```

함수

함수 표현식

- 함수 표현식(Expression): Hoisting 되지 않는다.
  - const/let 변수이름 = function [이름] (매개변수, ···) { statements }

```
1   const hello = function sayHello(name) {
2       console.log('Hello '+ name)
3   }
4
5   const bye = function (name) {
6       console.log('Bye ' + name)
7   }
8
9   // sayHello(); // 표현식에서의 이름으로는 호출할 수 없다.
10  hello('user');
11  bye('user');
```

에러 스택 메시지에서는 정의한 이름을 볼 수 있다.

함수

함수 표현식

- 함수 표현식(Expression): 람다식으로 정의
  - const/let 변수이름 = (매개변수,…) => { statements }

```
1  const hello = (name) => {
2      console.log('Hello '+ name)
3  }
4
5  hello('user');
```

함수

- 한 번만 사용하는 함수 선언이 가능하다.

```
1  (function(){
2      console.log('one time')
3  })();
4
5  (()=> {
6      console.log('one time')
7  })();
8
9  // 여기서는 위의 두 함수를 호출할 수 없다.
```

객체

객체

- 0개 이상의 속성과 함수를 가지는 데이터의 모음
- 원시 데이터 타입이 아닌 대부분의 데이터 모음
- Javascript에서는 함수도 객체

객체를 생성하는 방법 -1

- 중괄호{ } 로 묶인 0개 이상의 속성 이름과 속성 값
  - 속성 이름은 문자열 또는 Symbol을 사용할 수 있음
  - 속성 이름이 식별자 기준을 만족하면 속성이름:값 으로 선언
    - $, _, 유니코드 글자, 숫자(0-9, 첫글자 불가능)
  - 그 외의 속성 이름은 [속성이름]:값 으로 선언

객체를 생성하는 방법 -1

- 객체의 속성을 사용할 때 최종 생성된 속성 이름이
- 식별자 기준을 만족하면
  - 객체.이름
- 아니면 (심볼이나 ‘’ 등)
  - 객체[이름]

```
1  const value='value'
2  const symbol = Symbol('A')
3
4  const myObject = {
5      id:1,
6      _desc_:'my object',
7      value, // value:value
8      '?':'?',
9      [symbol]:'abc',
10     [value + (()=>1)()]: 1 // 동적 이름
11 }
12
13 console.log(myObject.id)
14 console.log(myObject._desc_)
15 console.log(myObject.value)
16 console.log(myObject['?'])
17 // console.log(myObject.'?') // error
18 console.log(myObject[symbol])
19 //console.log(myObject.symbol) // undefined
20 console.log(myObject.value1)
```

객체를 생성하는 방법 - 2

- Javascript에서 함수는 하나의 객체. 함수와 new 연산자를 이용해 객체 생성.

```
1  // new 연산자와 함께 호출할 경우
2  // Student.protype을 상속받은 객체가 생성되고
3  // 만들어진 객체를 this로 사용하여 아래 함수가 호출된다.
4  function Student(sid, sname){
5    this.studentId=sid;
6    this.studentName=sname;
7  }
8
9  const student1 = new Student(1, 'std1');
10 const student2 = new Student(2, 'std2');
11
12 console.log(student1);
13 console.log(student2);
```

객체를 생성하는 방법 - 3

- Object의 create 함수 사용

객체의 속성

데이터 속성

- 생성 이후 만들어진 객체에 속성을 추가하거나 삭제 하는 것이 가능

```
1  let obj = {
2      id:1
3  }
4  console.log(obj)
5
6  obj.value='value'
7  console.log(obj)
8
9  delete obj.value
10 console.log(obj)
```

객체의 속성

데이터 속성

- 각 데이터들은 다음 속성을 지님
  - value: 값. default-undefined
  - writable: 값 수정이 가능한가? default-false
  - enumerable: 열거형인가? default-false
  - configurable: 삭제하거나 접근성을 변경할 수 있는가? default-false
  - get/set 접근자

객체의 속성

데이터 속성

- Object.defineProperty()
- 속성 추가하기

```
1  const obj = {
2      id:1,
3      arr:[1, 2, 3]
4  }
5  console.log(obj)
6
7  Object.defineProperty(obj, 'value', {
8      configurable:false,
9      enumerable:true, // false 로 할 경우 console.log 에서 확인 불가능
10     value:'aa',
11     writable:false
12 });
13 console.log(obj)
14
15 console.log(delete obj.value) // false - not configurable
16 console.log(obj)
17 obj.value='123'
18 console.log(obj)
```

객체의 속성

데이터 속성

- Object.defineProperty()
- get()/set()을 이용한 가상의 속성
- value와 get/set을 동시에 쓸 수는 없음

```
1   const obj = {
2       id:1,
3       arr:[1, 2, 3]
4   }
5   console.log(obj)
6
7   Object.defineProperty(obj, 'value', {
8       configurable:false,
9       enumerable:true,
10      get(){ return this.id },
11      set(v) {this.id=v}
12  });
13  console.log(obj)
14
15  console.log(delete obj.value) // false - not configurable
16  console.log(obj)
17  obj.value=123
18  console.log(obj)
```

Prototype

- Javascript의 객체가 상속 받는 대상
- null > object

```
1  const obj={}
2  const str = 'a'
3  function f() {}
4
5  console.log(typeof obj)
6  console.log(typeof str)
7  console.log(typeof Object.getPrototypeOf(str))
8  console.log(typeof f)
```

Prototype

- Property 상속: 객체의 데이터 속성과 달리 prototype에 Property를 추가할 경우 모든 객체에 해당 속성이 추가된다.

함수와 객체

함수 객체

- Javascript의 함수는 일급객체 (First-class object): type 이 Function인 객체임
- 일급 객체
  - 함수의 매개변수/반환값 이 될 수 있다.
  - 변수에 할당할 수 있다.
  - 비교 연산을 적용할 수 있다.

함수

객체로 사용하기

- 함수의 매개 변수 또는 반환값이 될 수 있다.

```
function add(n1, n2){
    return n1+n2;
}

function sub(n1, n2) {
    return n1-n2;
}

function executor(f, a, b){
    return f(a, b)
}

const result = executor(add, 1, 1)
console.log(result)
```

```
function add(n1, n2){
    return n1+n2;
}

function sub(n1, n2) {
    return n1-n2;
}

function executor(type){
    if (type == 'add')
        return add
    else
        return sub
}

const f = executor('add')
console.log(f(1, 1))
```

함수

객체로 사용하기

- 변수에 저장하거나 내부 객체로 선언하기

```
1  function outer(){
2      function inner(){
3          console.log('inner')
4      }
5
6      console.log('outer');
7      inner();
8      inner();
9  }
10
11 outer();
```

```
1  function outer(){
2      const inner = function (){
3          console.log('inner')
4      }
5
6      console.log('outer');
7      inner();
8      inner();
9  }
10
11 outer();
```

함수

Generator

- function* 선언으로 Generator 객체를 생성할 수 있다.
- Generator: return 이 아닌 yield를 하며 함수의 실행을 종료하지 않고 값을 외부로 전달. iteration이 가능하게 해준다.

```
1   function* gen(){
2       yield 0;
3       yield 1;
4       yield 2;
5   }
6
7   const g = gen()
8   console.log(g.next().value)
9   console.log(g.next().value)
10  console.log(g.next()) // done:false
11  console.log(g.next()) // done:true
```

함수의 매개변수

원시타입

- 원시 타입 변수는 함수의 local 변수로 복사본이 제공된다.
- 함수 안에서 값을 수정해도 원본에는 영향이 없다.

```
1  function op(number){
2      number += 1
3      console.log(number) // 11
4  }
5
6  let number = 10
7  op(number)
8  console.log(number) // 10
```

```
1  function op(value){
2      value += '!!'
3      console.log(value) // user!!
4  }
5
6  let value = 'user'
7  op(value)
8  console.log(value) // user
```

함수의 매개변수

기본 값

- 매개 변수의 기본 값을 지정할 수 있다.

```
1  function prim(data=1) {
2      console.log(data)  1
3  }
4
5  function obj(data={id:1}) {
6      console.log(data)  { id: 1 }
7  }
8
9  prim()
10 obj()
```

함수의 매개변수

가변 매개변수

- 개수가 정해지지 않은 매개변수. Array로 전달된다.

```
1  function test(...params) {
2    console.log(params.length)
3    if(params.length > 0)
4      console.log(params[0])
5  }
6
7  test()
8  test(1)
9  test(1, 2, 3)
```

함수의 매개변수

객체타입

- 객체 타입 변수는 함수의 참조(reference)가 전달된다.
- 함수 안에서 값을 수정하면 객체의 속성이 변경된다.

```
1   function op(obj){
2       obj.prop='porp' // add property
3       console.log(obj) //
4   }
5
6   let obj = {
7       id:1
8   }
9   console.log(obj)
10  op(obj)
11  console.log(obj) // property prop added
```

```
1   function op(array){
2       array.push(4)
3       console.log(array) // 1, 2, 3, 4
4   }
5
6   let array = [1, 2, 3]
7   console.log(array)
8   op(array)
9   console.log(array) // item added
```

객체의 함수

- 특정 객체에 소속된 함수 정의하기
- ES6 이전 문법

```
1  const obj = {
2      id:1,
3      func:function(){
4          console.log(this.id)
5      }
6  }
7
8  obj.func()
```

ES6 문법

```
1  const obj = {
2      id:1,
3      func (){
4          console.log(this.id)
5      }
6  }
7
8  obj.func()
```

구조 분해 할당

구조 분해 할당

Restructuring assignment

- 객체나 배열을 변수로 분해하여 할당
- 배열

```
const arr=['a','b']

let [value1, value2] = arr
console.log(value1, value2);
```

```
const arr=['a','b', 'c', 'd']

let [value1, value2, ...restValues] = arr
console.log(value1, value2);
console.log(restValues);
```

```
function getResult(){
  return [1, 2, 3];
}

let [value1, value2, ...restValues] = getResult();
console.log(value1, value2);
console.log(restValues);
```

구조 분해 할당

Restructuring assignment

- 객체

```
function getInfo(){
  return {
    name:'ABC',
    price:1000
  };
}

let {name, price} = getInfo();
console.log(name, price);
```

함수 파라미터

```
const user={
  id:123,
  name:'user',
  address:'korea'
};

function printId({id}){
  console.log(id);
}

printId(user);
```

Collections

Collections

Array, Set, Map

- Array (배열):
  - 대괄호로 정의
  - 순서를 가지는 데이터의 집합.
  - 정수 0 이상의 index를 이용하여 항목에 접근.
  - 가변의 길이를 가진다.

```
1  const arr = [1, 2, 3, 4];
2  console.log(arr);
3
4  console.log(arr[0]); // index를 이용한 접근
5
6  arr.push(5); // 마지막에 항목 추가
7  console.log(arr);
8
9  arr.pop(); // 마지막 상목 삭제
10 console.log(arr);
11
12 const value = arr.splice(1, 2); // 1번 index부터 2개 삭제
13 console.log('removed:', value);
14 console.log(arr);
```

Collections

Array, Set, Map

- Array 의 함수들
  - map: 배열의 각 항목을 이용해 새로운 배열을 만든다.

```
1  const arr = [1, 2, 3];
2  const arr2 = arr.map((item)=>{ return `${item}th` });
3  console.log(arr2);
```

  - filter: 배열의 각 항목 중 true를 반환한 아이템만을 가지는 새로운 배열을 만든다.

```
1  const arr = [1, 2, 3, 4, 5, 6];
2  const arr2 = arr.filter((i)=>{
3      if (i%2==0) return true;
4      else return false;
5  });
6  console.log(arr2);
```

Collections

Array, Set, Map

- Array 의 함수들
  - find: 최초로 true를 리턴한 아이템을 반환하고 검색을 멈춘다.

```
const arr = [1, 2, 3, 4, 5, 6];
const value = arr.find((i)=>{
    if (i==3) return true;
    else return false;
});
console.log(value);
```

  - findIndex: 최초로 true를 반환한 index를 반환하고 검색을 멈춘다.

```
const arr = [1, 2, 3, 4, 5, 6];
const value = arr.findIndex((i)=>{
    if (i==3) return true;
    else return false;
});
console.log(value);
```

Collections

Array, Set, Map

- Array 의 함수들
  - includes: 주어진 값을 포함하면 true, 아니면 false를 반환한다.
  - forEach: 배열의 모든 항목을 iteration한다.

```
1  const arr=[1, 2, 3, 4, 5];
2
3  console.log(arr.includes(0));
4  arr.forEach((v, i)=>{
5      console.log(`value:${v}, index:${i}`);
6  });
```

Collections

Array, Set, Map

- Array 의 함수들
  - pop: 가장 마지막 요소를 삭제하고 반환
  - shift: 첫 요소를 삭제하고 반환

```
1   const arr=[1, 2, 3, 4, 5];
2
3   console.log(arr.pop());
4   console.log(arr.shift());
5   console.log(arr);
```

Spread

- Array, String 같이 iterable 한 데이터의 각 항목을 각각 나눈다.

```
1   const arr=[1, 2]
2
3   // 인자가 2개 필요한 함수.
4   function add(num1, num2){
5       return num1+num2;
6   }
7
8   // Spread
9   const result = add(...arr);
10  console.log(result);
```

Collections

Array, Set, Map

- Array (배열)의 복사
  - Shallow copy: 객체의 참조만 복사. 사실상 하나의 객체

```
1  const arr = [1, 2, 3, 4];
2  const arr2 = arr; // 참조만 복사
3
4  console.log(arr2);
5
6  arr2.pop(); // arr2에 변화는 arr에 변화와 동일
7  console.log(arr);
```

Collections

Array, Set, Map

- Array (배열)의 복사
  - Deep copy: 같은 값을 가지는 새로운 Array를 생성
    - Array에 값만 있을 경우 slice() 로 가능
    - Array의 항목으로 object가 있을 경우 별도의 함수를 구현해서 모두 복사해야 진짜 Deep copy로 동작

```
1   const arr = [1, 2, 3, 4];
2   const arr2 = arr.slice();
3
4   console.log(arr);
5   console.log(arr2);
6
7   arr2.pop();
8
9   console.log(arr);
10  console.log(arr2);
```

Collections

Array, Set, Map

- Set(집합): 중복을 허용하지 않는 데이터의 모음

```
1  const set = new Set([1, 2, 3]);// 배열로 초기화 가능
2  set.add(1); // 중복되는 값
3  set.add(4);
4  console.log(set);
5  const obj = {value:1}; // 객체도 넣을 수 있음.
6  set.add(obj);
7  console.log(set);
```

Collections

Array, Set, Map

- Map:
  - Key, Value의 쌍으로 가지는 순서를 가지는 데이터의 모음
  - Key는 유일함.

```
1   const map = new Map();
2
3   map.set('k1', 'v1');
4   map.set('k2', 2);
5   map.set('k3', 3.0);
6
7   console.log(map);
```

Collections

Array, Set, Map

- **Map 과 Object의 차이: Object가 아닌 Map을 써야 할 때**

	Map	**Object**
Key	어떤 타입이라도 가능	string 또는 Symbol만 가능
Size	size 속성으로 확인 가능	별도의 기능을 구현해야 함
Iteration	for 문 등으로 iteration 가능	iterable 하도록 별도의 함수(keys, values) 등을 구현해야 함.

Iteration

- for in VS for of
- for in 은 세부 항목을 string으로 변환한 뒤 iteration 한다.
- for of 는 세부 항목을 iteration 한다.

```
//ES6 for in - 주의
console.log('for in');
for(let i in arr){
  console.log(i+0);
}

//ES6 for of
console.log('for of');
for(let i of arr){
  console.log(i+0);
}
```

```
const arr=[1, 3, 5, 7, 9]

// 전통적인 방법
console.log('traditional');
for(let i=0; i<arr.length; i++){
  console.log(arr[i]);
}

//ES6 Array.forEach
console.log('for each');
arr.forEach((item)=>{
  console.log(item);
});

//ES6 for of
console.log('for of');
for(let i of arr){
  console.log(i);
}
```