LIKE LION

2강

자바스크립트 함수

WHATEVER YOU WANT, MAKE IT REAL.

강사 정길용

LIKE LION

{

일급 객체

함수 생성

호이스팅

함수 호출

함수 패턴

}

LIKE LION

▶ 원시 타입(Primitive type)

- number
- string
- boolean
- null
- undefined
- bigint
- symbol

▶ 참조 타입(Refrence type)

- object
    - Array
    - Function
    - Date
    - RegExp
    - .....

LIKE LION

ex02-01.js, 02, 03, 04

- 일급 객체(First-class object)
  - 변수, 배열 엘리먼트, 다른 객체의 프로퍼티에 할당될 수 있다.
  - 함수의 인자로 전달될 수 있다.
  - 함수의 결과 값으로 반환될 수 있다.
  - 리터럴로 생성될 수 있다.
  - 동적으로 생성된 프로퍼티를 가질 수 있다.

- 자바스크립트의 함수(Function)는 일급 객체이다.
  - 함수 == (호출 + 객체)

- 함수가 일급 객체라서 가능한 일
  - 콜백 함수(Callback function)
    - 다른 함수에 인자로 전달되어 어떤 작업의 결과로 호출되는 함수
  - 고차 함수(Higher order function)
    - 함수를 인자로 받거나 반환하는 함수
  - 클로저(Closure)

LIKE LION

ex02-05.js

- function 키워드로 시작하는 함수 정의

```
function add(x, y){
  const result = x + y;
  return result;
}

function(){}
```

- 함수 이름
  - 유효한 식별자이어야 함
  - 생략 가능
- 매개변수 목록
  - 쉼표로 구분된 매개변수 목록과 그 매개변수 목록을 둘러싸고 있는 괄호
  - 매개변수는 생략 가능, 괄호는 필수
- 함수 본문
  - 중괄호로 둘러싸여 있는 자바스크립트 구문
  - 본문은 생략 가능, 중괄호는 필수

LIKE LION

ex02-06.js, 07, 08

- 함수 정의를 변수에 할당
  - 변수명을 함수명처럼 사용

- 변수에 익명함수를 지정

```
const add = function(x, y){
  const result = x + y;
  return result;
};

add(10, 20);
```

- 변수에 기명함수를 지정
  - 변수명을 함수명처럼 사용
  - 함수명을 통한 접근은 해당 함수 내부에서만 사용 가능 (재귀 함수)

```
const f = function factorial(n){
  if(n==1) return 1;
  return n * factorial(n-1);
};

console.log(f(5));
~~console.log(factorial(5));~~
```

LIKE LION

ex02-09.js

▶ Function 생성자 함수 이용

- 함수 객체를 생성해서 반환

```
const add = new Function('x', 'y',
  'let result = x + y; return result;');
```

LIKE LION

ex02-10.js

▶ arrow function 이용

- 함수 표현식의 대안으로 간결하게 함수 정의
- 익명 함수로만 정의 가능
- 실행할 코드가 하나만 있다면 함수 본문의 중괄호 생략 가능
- 함수 본문의 중괄호가 생략될 경우 함수의 코드가 자동으로 리턴값으로 사용됨
- 매개 변수가 하나만 있다면 매개변수의 괄호 생략 가능

```
// 기존 함수
const add = function(x, y) {
  return x + y;
};

// 화살표 함수
const add = (x, y) => {
  return x + y;
}

// 화살표 함수 축약
const add = (x, y) => x + y;
```

```
// 기존 함수
const add10 = function(x) {
  return x + 10;
};

// 화살표 함수
const add = (x) => {
  return x + 10;
}

// 화살표 함수 축약
const add = x => x + 10;
```

LIKE LION

ex02-11.js 

▶ 함수 호이스팅

- 선언문 형태로 정의한 함수의 유효 범위가 코드의 맨 처음부터 시작하게 되는 자바스크립트의 동작 방식
- 특정 블럭의 코드가 자바스크립트 엔진에 의해 실행 되기 전에 호이스팅 단계를 거치는데 이때 선언문 형태의 함수가 생성 되므로 함수 선언 코드보다 먼저 호출하는 코드를 작성하는게 가능

```
console.log(add(2, 3)); // 5
function add(x, y) {
  return x + y;
}
console.log(add(3, 4)); // 7
```

호이스팅 단계 ↓

코드실행 단계 ↓

```
function add(x, y) {
  return x + y;
}
console.log(add(2, 3)); // 5
console.log(add(3, 4)); // 7
```

LIKE LION

ex02-12.js

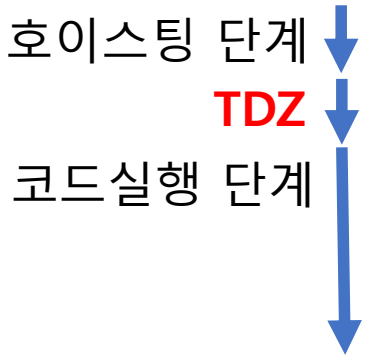
▶ 변수 호이스팅

- 호이스팅 단계에서 var로 선언한 변수의 경우 선언만 되고 그 값은 undefined로 초기화 됨
- let, const로 선언한 변수의 경우 호이스팅 단계에서 선언은 되지만 초기화가 되지 않기 때문에 선언 전에 접근하면 에러 발생 (undefined 값도 할당되지 않은 상태)

▶ TDZ(Temporal Dead Zone)

- 호이스팅 단계에서 변수가 선언된 이후 실제 초기화가 이루어지기 전까지의 구간
- 변수를 초기화 하기 이전에 변수에 접근할 수 없는 일시적인 영역으로, 무분별한 변수 사용을 막고 예측 가능한 코드를 만들기 위한 장치

```
console.log(add(2, 3));
const add = function(x, y) {
  return x + y;
}
console.log(add(3, 4)); // 7
```




```
const add;
console.log(add(2, 3));
add = function(x, y) {
  return x + y;
}
console.log(add(3, 4)); // 7
```

LIKE LION

ex02-13.js 

매개변수와 인자의 수

- 함수에 정의한 매개변수와 함수 호출에 사용되는 인자의 수가 달라도 에러가 발생하지는 않음

매개변수 > 인자

- 부족한 인자에 대한 매개변수에는 undefined가 지정됨

```
function add(x, y, z) {
  console.log(z); // undefined
  return x + y;
}
console.log(add(3, 4)); // 7
```

매개변수 < 인자

- 남는 인자에 대해서는 처리할 매개변수가 없기 때문에 무시됨

```
function add(x, y) {
  return x + y;
}
console.log(add(3, 4, 5)); // 7
```

LIKE LION

ex02-13.js 

암묵적 매개변수

- 모든 함수에서 명시적으로 선언하지 않고 암묵적으로 사용할 수 있는 매개변수
- arguments, this

arguments

- 함수 내에서 arguments 변수로 접근 가능
- 함수에 전달된 모든 인자들을 담고 있는 컬렉션(유사 배열 객체)
- 배열과 비슷하게 length 속성과 index로 각 인자에 접근 가능
- arguments 대신 ES6의 "나머지 매개변수 (...)" 사용을 권장

this

- 함수 내에서 this 키워드로 접근 가능
- "함수 컨텍스트" 객체
- 함수를 호출한 객체에 대한 참조

LIKE LION

ex02-14.js

▶ 함수로 호출

- 일반적인 함수 호출 방법
- 함수명()
- this는 window 객체
  - window 객체는 어디서나 참조 가능하므로 this를 사용할 필요 없음

```
function f1(){
  console.log(this);
  this.alert();
  window.alert();
  alert();
};
f1();

const f2 = function(){
  console.log(this);
};
f2();
```

```
▶ Window {window: Window, se
▶ Window {window: Window, se
```

LIKE LION

ex02-15.js, 02-01

▶ 메서드로 호출

- 객체에 정의된 메서드를 호출할 때
- 객체.메서드명()
- this는 메서드를 정의한 객체
  - this는 생성된 객체를 참조하므로 객체에 종속적인 속성을 부여하는게 가능
  - 함수를 하나만 정의하고 여러 객체에서 메서드로 사용
  - 자바스크립트로 객체지향 프로그래밍을 가능하게 하는 중요한 특징

```
const getPingName = function() {
  console.log(this);
  return this.name;
};
const ping1 = { name: '바로핑', age: 11, getName: getPingName };
const ping2 = { name: '라라핑', age: 9, getName: getPingName };
console.log(baro.age, baro.getName());
console.log(rara.age, rara.getName());
```

```
{ name: '바로핑', age: 11, getName: [Function: getPingName] }
11 바로핑
{ name: '라라핑', age: 9, getName: [Function: getPingName] }
9 라라핑
```

LIKE LION

ex02-16.js 

▶ 화살표 함수 호출

- 일반 함수나 메서드와 동일하게 호출하지만 함수 내부에 arguments나 this가 생성되지 않고 상위 컨텍스트의 arguments, this를 사용하게 됨

```
const getPingName = () => {
  console.log(this);
  return this.name;
};
const ping1 = { name: '바로핑', age: 11, getName: getPingName };
const ping2 = { name: '라라핑', age: 9, getName: getPingName };
console.log(baro.age, baro.getName());
console.log(rara.age, rara.getName());
```

```
{ name: '바로핑', age: 11, info: [Function: getPingInfo] }
undefined은 11 살 입니다.
{ name: '라라핑', age: 9, info: [Function: getPingInfo] }
undefined은 9 살 입니다
```

LIKE LION

ex02-05.js, 15, 17

▶ apply(), call() 메서드로 호출

- 함수에 정의된 메서드
- 함수명.apply(), 함수명.call() 형태로 호출
- this는 apply(), call() 메소드의 첫번째 인자로 전달되는 객체
- this를 명시적으로 지정할 수 있음
- 콜백 함수 호출 시 주로 사용

▶ apply(p1, p2) 메서드

- 두 개의 매개변수를 가짐
- 첫 번째 매개변수(p1)에는 this로 사용할 객체를 전달
- 두 번째 매개변수(p2)에는 함수에 전달할 인자값 배열

▶ call(p1, p2, p3, ...) 메서드

- 여러 개의 매개변수를 가짐
- 첫 번째 매개변수(p1)에는 this로 사용할 객체를 전달
- 두 번째 이후의 매개변수(p2, p3, ...)에는 함수에 전달할 인자값을 차례대로 지정

배열의 push() 메서드 기능

- 배열의 마지막에 지정한 요소를 추가한다.
- this로 지정된 Array 객체의 length 속성값에 해당하는 속성을 만들고 지정한 요소를 저장한 후 length를 하나 증가시킨다.

Array의 push() 메서드를 이용하여 객체를 배열처럼 동작시키기

- length 속성 추가
- Array.prototype.push.call(객체, 추가할 요소)

prototype

- 생성자 함수를 통해 생성되는 객체의 메서드를 정의하는 속성
- 모든 함수에 자동으로 할당됨

LIKE LION

ex02-18.js 

▶ apply() 활용

- 배열 데이터를 각각의 매개변수로 분리하여 전달할 때

- Math.min(n1, n2, n3, ...)
- Math.max(n1, n2, n3, ...)

- `const` a = [e1, e2, e2, ...];
- Math.min(a[0], a[1], a[2], ...???)

- Math.min.apply(Math, a)

▶ 전개 연산자 활용

- Math.min(...a)

생성자 함수(객체지향 언어의 클래스와 비슷) 호출

- 함수를 생성자로 사용할 경우
- new 함수명()
- this는 생성자를 통해 생성된 객체

생성자로 호출될 때의 내부 동작

- 비어있는 객체를 새로 생성
- 새로 생성된 객체는 this 매개변수로 생성자 함수에 전달
- 명시적으로 반환하는 객체가 없다면 생성된 객체를 반환
- 객체지향 프로그램의 new 연산자와 비슷한 동작

생성자를 작성할 때 고려해야 할 것들

- 일반 함수처럼 호출할 수 있지만 이럴 경우 생성자 내부의 this는 window 객체를 가리키므로 객체에 종속적인 값을 지정할 수 없으므로 의미가 없다.
- 명명(naming) 규칙
  - 일반 함수: 작업할 동작을 나타내는 동사로 이름 짓고 소문자로 시작
  - 생성자: 생성할 객체를 나타내는 명사로 이름 짓고 대문자로 시작

LIKE LION

ex02-19, 20

▶ 자바스크립트의 생성자 함수들

- Function

```
let f = new Function('x', 'y', 'return x + y;');
let f = (x, y) => { return x + y; }
```

- Object

```
let obj = new Object();
let obj = {};
```

- String, Number, Boolean

```
const name = new String('김철수');
const age = new Number(30);
const male = new Boolean(true);
```

- Array

```
let arr = new Array();
let arr = [];
```

- Date

```
const date = new Date();
```

익명 함수의 사용처

- 함수의 이름 대신 변수명, 속성명으로 사용할 경우
  - 함수를 변수에 저장
  - 객체의 메서드로 지정
- 함수를 인자값으로 전달할 경우
  - 전달한 인자값은 호출되는 함수 내부에서 적절한 매개변수를 지정해서 사용

```
const f1 = function(){};
const obj = {
  f2: function(){}
};
window.onload = function(){};
setTimeout(function(){}, 1000);
```

```
const f1 = () => {};
const obj = {
  f2: () => {}
};
window.onload = () => {};
setTimeout(() => {}, 1000);
```

LIKE LION

ex02-21.html 

▶ 콜백 함수

- 다른 함수에 인자로 전달되어 어떤 작업의 결과로 호출되는 함수
- 특정한 상황이 되거나(이벤트 발생) 지정한 시간이 흐르면(timeout) 또는 특정 작업의 수행이 끝나면 호출하도록 지정한 함수

```
setTimeout(() => {
  console.log('1초가 흐름');
}, 1000);
```

```
btn.addEventListener('click', () => {
  console.log('버튼 클릭');
});
```

```
function sendData(data, cb) {
  // data를 서버에 전송한다.
  // ......
  cb();
};
```

```
const newPing = { name: '새로핑', age: 7 };
sendData(newPing, () => {
  alert('회원 가입 완료');
});
```

```
const newPost = {
  title: '콜백 함수란', writer: '진지핑'
};
sendData(newPost, () => {
  alert('게시글 작성 완료');
});
```

▶ 순수 함수

- 동일한 입력값에 대해서 항상 동일한 출력을 하는 함수
- Side Effect가 없는 함수
  - 외부값에 영향을 주거나 받지 않는 함수

```
let sum = 0;
function add(a, b) {
  sum = a + b; // 외부값을 변경
}
add(10, 20);
console.log(sum); // 30
add(30, 40);
console.log(sum); // 70
```

```
let sum = 0;
function add(a, b) { // 순수 함수
  return a + b;
}
const sum1 = add(10, 20);
const sum2 = add(30, 40);
console.log(sum1); // 30
console.log(sum2); // 30
```

LIKE LION

ex02-22.js 

고차 함수

- 함수를 인자로 받거나 반환하는 함수
- 함수형 프로그래밍은 순수함수를 통해 부수효과(Side Effect)를 줄이고 오류를 피해 안정성을 높이는게 목적
- 외부 상태 변경이나 가변 데이터에 영향을 받는 조건문, 반복문 대신 함수의 조합으로 로직을 구성하는 함수형 프로그래밍에서 사용

배열 고차 함수(함수를 인자로 받음)

- sort(), forEach(), map(), filter(), reduce(), some(), every(), find(), findIndex()

Currying(함수를 반환)

- Closure 단원

Partial application(함수를 인자로 받고 함수를 반환)

- Closure 단원

LIKE LION

ex02-23.js 

Memoization

- 이전의 계산 결과를 기억하는 기능을 갖춘 함수
- 함수는 객체이기 때문에 함수의 속성값으로 계산 결과 캐시
- 함수에 종속된 속성을 이용하기 때문에 외부에 노출하지 않고 함수 자체적으로 구현 가능

장점

- 이미 수행한 복잡한 연산을 반복하지 않도록 함으로서 성능을 향상
- 사용자가 알 수 없게 내부적으로만 동작

단점

- 캐시에 필요한 메모리 사용량 증가
- 비즈니스 로직과 캐싱 기능의 혼재
- 부하 테스트나 알고리즘의 성능 테스트가 어려워짐