웹 앱 개발을 위한 Javascript 기초

5. 자바스크립트 내장 객체

한국기술교육대학교
온라인평생교육원

학습내용

1. 객체
2. 표준 내장 객체

학습목표

1. 객체를 생성하고 사용하는 방법, 객체의 속성과 메소드, 생성자 함수와 프로토타입에 대해 설명할 수 있다.
2. 기본 자료형과 객체 자료형의 차이, 자바스크립트의 표준 내장 객체에 대해 설명할 수 있다.

1. 객체

배열

* 기본 형태

```
// 배열을 선언합니다.
let array = ['사과', '바나나', '망고', '딸기'];
```

* 배열 접근

```
array[0] → '사과'
array[1] → '바나나'
```

배열

* 배열은 요소에 접근할 때 인덱스를 사용하고, 객체는 키를 사용함

인덱스	요소
0	사과
1	바나나
2	망고
3	딸기

1. 객체

객체 기본

객체 선언

```
//객체를 선언합니다.
let product = {
    제품명: '7D 건조 망고',
    유형: '당절임',
    성분: '망고, 설탕, 메타중아황산나트륨, 치자황색소',
    원산지: '필리핀'
};
//출력합니다.
console.log(product);
▶ {제품명: "7D 건조 망고", 유형: "당절임", 성분: "망고, 설탕, 메타중아황산나트륨, 치자황색소", 원산지: "필리핀"}
undefined
```

객체 기본

객체 선언

키	속성
제품명	7D 건조 망고
유형	당절임
성분	망고, 설탕, 메타중아황산나트륨, 치자황색소
원산지	필리핀

1. 객체

객체 기본

* 객체 접근

product['제품명'] → '7D 건조 망고'
product['유형'] → '당절임'
product['성분'] → '망고, 설탕, 메타중아황산나트륨, 치자황색소'
product['원산지'] → '필리핀'

product.제품명 → '7D 건조 망고'
product.유형 → '당절임'
product.성분 → '망고, 설탕, 메타중아황산나트륨, 치자황색소'
product. 원산지 → '필리핀'

객체 기본

* 객체 생성

```
// 객체를 선언합니다.
let object = {
    name: '바나나',
    price: 1200
};
//출력합니다.
console.log(object.name);
console.log(object.price);
```

```
바나나
1200
undefined
```

속성(키)	값
name	'바나나'
price	1200

1. 객체

객체와 반복문

* for in 반복문을 사용해 객체에 반복문을 적용

```
//객체를 선언합니다.
let object = {
    name: `바나나`,
    price: 1200
};

//출력합니다.
for (let key in object) {
    console.log(`${key}: ${object[key]}`);
}
```

```
name: 바나나
price: 1200
undefined
```

속성과 메소드

* 요소

  ⇨ 배열 내부에 있는 값 하나하나

* 속성

  ⇨ 객체 내부에 있는 값 하나하나

1. 객체

속성과 메소드

* 객체의 다양한 자료형

```
var object = {
    number: 273,
    string: 'RintlanTta',
    boolean: true,
    array: [52, 273, 103, 32]
    method: function () {

    }
};
```

속성과 메소드

* 메소드

  ⇨ 객체의 속성 중 자료형이 함수인 속성

* 속성과 메소드

```
let object = {
    name: '바나나',
    price: 1200,
    print: function ( ) {
        console.log(`${this.name}의 가격은 ${this.price}원입니다.`)
    }
};
// 메소드를 호출합니다.
Object.print( );
```

1. 객체

속성과 메소드

* 메소드 내부에서 this 키워드

```
//객체를 선언합니다.
let object = {
    name: `바나나`,
    price: 1200,
    print: function () {
        console.log(`${this.name}의 가격은 ${this.price}원입니다.`)
    }
};
//메소드를 호출합니다.
object.print();
```

```
바나나의 가격은 1200원입니다.
undefined
```

바나나의 가격은 1200원 입니다.

생성자 함수와 프로토타입

* 객체 지향 프로그래밍

  ⇨ 현실의 객체를 모방해서 프로그래밍

* 배열과 객체를 사용하면 여러 개의 데이터를 쉽게 다룰 수 있음

1. 객체

생성자 함수와 프로토타입

* 객체 배열

```
let products = [
    { name: '바나나', price: 1200 },
    { name: '사과', price: 2000 },
    { name: '배', price: 3000 },
    { name: '고구마', price: 700 },
    { name: '감자', price: 600 },
    { name: '수박', price: 5000 },
];
```

생성자 함수와 프로토타입

* 객체에 메소드 추가
* 메소드를 가진 객체의 배열

```
// 객체를 선언합니다.
let products = {
    name: '바나나',
    price: 1200,
    print: function () {
        console.log(`$(this.name)의 가격은 ${this.price}원입니다.`)
    }
};
// 메소드를 호출합니다.
Object.print( );
```

(스크립트 계속)

1. 객체

생성자 함수와 프로토타입

```
    name: '사과',
    price: 2000,
    print: function () {
        console.log(`$(this.name)의 가격은 ${this.price}원입니다.`)
    }
}, {
    name: '배',
    price: 3000,
    print: function () {
        console.log(`$(this.name)의 가격은 ${this.price}원입니다.`)
    }
}, {
```

(스크립트 계속)

생성자 함수와 프로토타입

```
    name: '고구마',
    price: 700,
    print: function () {
        console.log(`$(this.name)의 가격은 ${this.price}원입니다.`)
    }
}, {
    name: '감자',
    price: 600,
    print: function () {
        console.log(`$(this.name)의 가격은 ${this.price}원입니다.`)
    }
}, {
```

(스크립트 계속)

1. 객체

생성자 함수와 프로토타입

```
    name: '수박',
    price: 5000,
    print: function () {
        console.log(`${this.name}의 가격은 ${this.price}원입니다.`)
    }
}];

// 반복해서 출력합니다.
for (let product of products) {
    product.print( );
}
```

생성자 함수와 프로토타입

* 함수를 외부로 내보낸 형태

```
// 상품 목록을 선언합니다.
let products = [
    { name: '바나나', price: 1200 },
    { name: '사과', price: 2000 },
    { name: '배', price: 3000 },
    { name: '고구마', price: 700 },
    { name: '감자', price: 600 },
    { name: '수박', price: 5000 },
];
```

(스크립트 계속)

1. 객체

생성자 함수와 프로토타입

* 함수를 외부로 내보낸 형태

```
// 함수를 선언합니다.
function printProduct(product) {
      console.log(`${product.name}의 가격은
${product.price}원입니다.`);
}

// 반복해서 출력합니다.
for (let product of products) {
    printProduct(product);
}
```

생성자 함수와 프로토타입

* 생성자 함수
  - 객체를 만드는 함수, 대문자로 시작하는 이름 사용

```
function Product(name, price) {
    this.name = name;
    this.price = price;
}
```

```
// 생성자 함수
function Product(name, price) {
    this.name = name;
    this.price = price;
}
// 객체를 생성합니다.
let product = new Product("바나나", 1200);
// 출력합니다.
console.log(product);
▶ Product {name: "바나나", price: 1200}
undefined
```

1. 객체

생성자 함수와 프로토타입

- 프로토타입
  - 생성자 함수로 만든 객체는 프로토타입 공간에 메소드를 지정해서 모든 객체가 공유 하도록 함
  - 객체가 공유 하도록 함, 해당 함수를 생성자 함수로 사용했을 때만 의미가 있음

생성자 함수와 프로토타입

- 프로토타입을 사용한 메소드 생성

```
// 생성자 함수
function Product(name, price) {
      this.name = name;
      this.price = price;
}

// 프로토타입에 메소드를 선언합니다.
Product.prototype.print = function ( ) {
     console.log(`${product, name}의 가격은 ${product.price}원입니다.`);
};
```

(스크립트 계속)

1. 객체

생성자 함수와 프로토타입

* 프로토타입을 사용한 메소드 생성

```
// 객체를 생성합니다.
let product = new product("바나나", 1200);

// 메소드를 호출합니다.
product.print( );
```

생성자 함수와 프로토타입

* 객체 지향적으로 구성한 객체 배열

```
// 생성자 함수
function Product(name, price) {
      this.name = name;
      this.price = price;
}

// 프로토타입에 메소드를 선언합니다.
Product.prototype.print = function ( ) {
     console.log(`${this.name}의 가격은 ${this.price}원입니다.`);
};
```

(스크립트 계속)

1. 객체

생성자 함수와 프로토타입

* 객체 지향적으로 구성한 객체 배열

```
// 상품 목록을 선언합니다.
let products = [
    new Product('바나나', 1200),
    new Product('사과', 2000),
    new Product('사과', 3000),
    new Product('배', 3000),
    new Product('고구마', 700),
    new Product('감자', 600),
    new Product('수박', 600),
];
```

(스크립트 계속)

생성자 함수와 프로토타입

* 객체 지향적으로 구성한 객체 배열

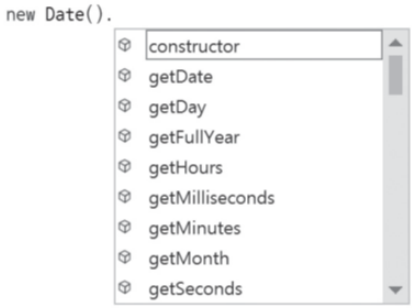
```
// 반복해서 출력합니다.
for (let product of products) {
    product.print( );
}
```

- 바나나의 가격은 1200원입니다.
- 사과의 가격은 2000원입니다.
- 배의 가격은 3000원입니다.
- 고구마의 가격은 700원입니다.
- 감자의 가격은 600원입니다.
- 수박의 가격은 5000원입니다.

2. 표준 내장 객체

기본 자료형과 객체 자료형의 차이

- 자바스크립트는 다양한 객체를 제공
- 통합 개발 환경에서 자동 완성 기능



기본 자료형과 객체 자료형의 차이

- 기본 자료형 : 숫자, 문자열, 불

```
// 기본 자료형
let number = 273;
let string = `안녕하세요`;
let boolean = true;
// 자료형을 출력합니다.
console.log(typeof number);
console.log(typeof string);
console.log(typeof boolean);
```

```
number
string
boolean
undefined
```

2. 표준 내장 객체

기본 자료형과 객체 자료형의 차이

* 객체 숫자, 문자열, 불

```
// 기본 자료형
let number = new Number(273);
let string = new String('안녕하세요');
let boolean = new Boolean(true);
// 자료형을 출력합니다.
console.log(typeof number);
console.log(typeof string);
console.log(typeof boolean);
```

```
object
object
object
undefined
```

기본 자료형과 객체 자료형의 차이

* 기본 자료형의 속성 또는 메소드를 사용할 때 기본 자료형이 자동으로 객체로 변환이 됨

  ⇨ 즉, 기본 자료형과 객체 자료형 모두 속성과 메소드를 사용함

```
let stringA = '과자|1500원';
console.log(stringA.split('|'));

let stringB = new String('과자|1500원');
console.log(stringB.split('|'));
```

```
▶ (2) ["과자", "1500원"]
▶ (2) ["과자", "1500원"]
undefined
```

2. 표준 내장 객체

기본 자료형과 객체 자료형의 차이

* 차이점

  ➪ 기본 자료형은 객체가 아니므로 속성과 메소드를 추가할 수 없음

```
// 변수를 생성합니다.
let primitiveNumber = 273;

// 메소드를 추가합니다.
primitiveNumber.method = function ( ) {
      return 'Primitive Method';
};

//메소드를 실행합니다.
console.log(primitiveNumber.method( ));
```

기본 자료형과 객체 자료형의 차이

```
TypeError: primitiveNumber. method is not a function
  at object.<anonymous>(C:\example\methodToPrimitives.js:2:15)
  at Module._compile (module.js:541:32)
  at Object. Module. extensions..js (module.js:550:10)
  at Module.load(module.js:458:32)
  at tryModuleLoad (module.js:417:12)
  at Function.Module._Load(module.js:409:3)
  at Module.runMain(module.js:575:10)
  at run (node.js:348:7)
  at startup (node.js:140:9)
  at node.js:463:3
```

2. 표준 내장 객체

기본 자료형과 객체 자료형의 차이

* 기본 자료형에 프로토타입으로 메소드 추가

```
// 변수를 생성합니다.
let primitiveNumber = 273;
// 메소드를 추가합니다.
Number.prototype.method = function ( ) {
    return 'Primitive Method';
};
//메소드를 실행합니다.
console.log(primitiveNumber.method( ));
```

```
Primitive Method
undefined
```

Number 객체

* 자바스크립트에서 숫자를 표현할 때 사용
* Number 객체 생성

```
let numberFromLiteral = 273;
let numberFromConstructor = new number(273);
```

메소드	설명
toExponential( )	숫자를 지수 표시로 나타낸 문자열을 리턴합니다.
toFixed( )	숫자를 고정소수점 표시로 나타낸 문자열을 리턴합니다.
toPrecision( )	숫자를 길이에 따라 지수 표시 또는 고정소수점 표시로 나타낸 문자열을 리턴합니다.

2. 표준 내장 객체

Number 객체

- Number 객체의 메소드
  - toFixed() 메소드를 사용해 소수점 자릿수를 자르는 방법

```
// 변수를 선언합니다.
let number = 273.5210332;
// 출력합니다.
console.log(number.toFixed(1));
console.log(number.toFixed(4));
```

```
273.5
273.5210
undefined
```

Number 객체

- 생성자 함수의 속성
- 생성자 함수에 속성과 메소드 추가

```
// 생성자 함수를 생성합니다.
function Constructor( ) { }
constructor.property = 273;
constructor.method = function ( ) { };
// 생성자 함수의 속성과 메소드를 출력합니다.
console.log(constructor.property);
console.log(constructor.method);
```

```
273
ƒ ( ) { }
undefined
```

2. 표준 내장 객체

Number 객체

* Number 생성자 함수의 속성

속성	설명
MAX_VALUE	자바스크립트의 숫자가 나타낼 수 있는 최대 숫자
MIN_VALUE	자바스크립트의 숫자가 나타낼 수 있는 최소 숫자
NaN	자바스크립트의 숫자가 나타낼 수 없는 숫자
POSITIVE_INFINITY	양의 무한대 숫자
NEGATIVE_INFINITY	음의 무한대 숫자

String 객체

* String 객체 생성

```
let stringFromLiteral = '안녕하세요';
let stringFromConstructor = new String('안녕하세요');
```

* 속성과 메소드

속성	설명
length	문자열의 길이를 나타냅니다.

* String 객체의 메소드는 변경된 값을 리턴함

2. 표준 내장 객체

String 객체

String 객체 메소드

메소드	설명
charAt(position)	position에 위치하는 문자를 리턴합니다.
charCodeAt(position)	position에 위치하는 문자의 유니코드 번호를 리턴합니다.
concat(args)	매개 변수로 입력한 문자열을 이어 리턴합니다.
indexOf(searchString, position)	앞에서부터 일치하는 문자열의 위치를 리턴합니다.
lastIndexOf(searchString, position)	뒤에서부터 일치하는 문자열의 위치를 리턴합니다.
match(regExp)	문자열 안에 regExp가 있는지 확인합니다.

String 객체

메소드	설명
replace(regExp, Replacement)	regExp를 replacemen로 바꾼 후 리턴합니다.
search(regExp)	regExp와 일치하는 문자열의 위치를 리턴합니다.
slice(start, end)	특정 위치의 문자열을 추출해 리턴합니다.
split(separator, limit)	separator로 문자열을 잘라 배열을 리턴합니다.
substr(start, count)	start부터 count만큼 문자열을 잘라서 리턴합니다.
substring(start, end)	start부터 end까지 문자열을 잘라서 리턴합니다.
toLowerCase( )	문자열을 소문자로 바꾸어 리턴합니다.
toUpperCase( )	문자열을 대문자로 바꾸어 리턴합니다.

2. 표준 내장 객체

String 객체

* 잘못된 String 객체의 메소드 사용

```
// 변수를 선언합니다.
let string = 'abcdefg';
// 출력합니다.
string.toUpperCase( );
console.log(string);
```

```
abcdefg
undefined
```

* 자기 자신을 변경하지 않고 리턴하는 것뿐이므로 소문자 상태로 출력
* 올바른 String 객체의 메소드 사용

```
// 변수를 선언합니다.
let string = 'abcdefg';
// 출력합니다.
string = string.toUpperCase( );
console.log(string);
```

```
ABCDEFG
undefined
```

* 메소드 활용
* 문자열 포함
  - indexOf( ) 메소드
    - 특정 문자열이 있는지 확인, 위치를 리턴함
    - 문자열이 포함되어 있지 않을 때는 -1을 리턴

```
// 변수를 선언합니다.
let string = '안녕하세요. 좋은 아침입니다.';
// 문자열 내부에 "아침"이라는 문자열이 있는지 확인합니다.
if (string.indexOf('아침') >= 0) {
    console.log('좋은 아침이에요...!');
}
```

```
좋은 아침이에요...!
undefined
```

2. 표준 내장 객체

String 객체

* 문자열 분해

- split ( ) 메소드를 사용하면 특정한 기호를 기반으로 문자열을 분해함

```
//변수를 선언합니다.
let string = '감자,고구마,바난,사과';
//문자열을 쉼표로 자르고 출력합니다.
let array = string.split(',');
console.log(array);
▶ (4) ["감자", "고구마", "바난", "사과"]
undefined
```

Date 객체

* Date 객체 생성 방법

생성자 함수	설명
new Date( )	현재 시간으로 Date 객체를 생성합시다.
new Date(유닉스 타임)	유닉스 타임(1970년 1월 1일 00시 00분 00초부터 경과한 밀리초)으로 Date 객체를 생성합니다.
new Date(〈시간 문자열〉)	문자열로 Date 객체를 생성합니다.
new Date(〈년〉, 〈월-1〉, 〈일〉, 〈시간〉, 〈분〉, 〈초〉, 〈밀리초〉)	시간 요소(년,월-1, 시간, 분, 초, 밀리초)를 기반으로 Date 객체를 생성합니다.

- Month를 나타내는 '월'은 0부터 시작, 0 → 1월, 11 → 12월

2. 표준 내장 객체

Date 객체

* Date 객체 생성

```
// 현재 시간을 기반으로 Date 객체를 생성합니다.
let dateA = new Date();
console.log(dateA);
// 유닉스 타임(1970년 1월 1일 00시 00분 00초부터 경과한 밀리초)
let dateB = new Date(692281800000);
console.log(dateB);
// 문자열을 기반으로 Date 객체를 생성합니다.
let dateC = new Date("December 9, 1991 21:30:00")
console.log(dateC);
// 시간 요소(년, 월 -1, 일, 시간, 분, 초, 밀리초)를 기반으로 Date 객체를 생성합니다.
let dateD = new Date(1991, 12 -1, 9, 21, 30, 0, 0);
console.log(dateD);
```

```
Fri Nov 20 2020 16:16:32 GMT+0900 (대한민국 표준시)
Mon Dec 09 1991 21:30:00 GMT+0900 (대한민국 표준시)
Mon Dec 09 1991 21:30:00 GMT+0900 (대한민국 표준시)
Mon Dec 09 1991 21:30:00 GMT+0900 (대한민국 표준시)
undefined
```

Date 객체

* 메소드 활용
  - Date 객체
  - get○○() 형태 메소드, set○○() 형태 메소드
    ⇨ FullYear, Month, Day, Hours, Minutes, Seconds 등 사용

2. 표준 내장 객체

Date 객체

시간 더하기

- 현재 시간에 1년, 11개월, 7일을 더해 출력. (현재 시간 : 2020년 7월 8일)

```
// 현재 시간을 구합니다.
let date = new Date( );
// 출력1
console. log(date);
// 시간을 더합니다.
date.setFullYear(date.getFullYear( ) + 1);
date.setMonth(date.getMonth( ) + 11);
date.setDate(date. getDate( ) + 3);
// 출력2
console.log(date);
```

```
Fri Nov 20 2020 16:21:22 GMT+0900 (대한민국 표준시)
Sun Oct 23 2022 16:21:22 GMT+0900 (대한민국 표준시)
undefined
```

정리하기

1. 객체

- 객체 기본 : 객체 선언, 객체 접근, 객체 생성
- 객체와 반복문 : for in 반복문을 사용해 객체에 반복문을 적용
- 속성과 메소드 : 요소(배열 내부에 있는 값 하나하나), 속성(객체 내부에 있는 값 하나하나), 객체의 다양한 자료형, 메소드(객체의 속성 중 자료형이 함수인 속성)
- 생성자 함수와 프로토타입 : 객체 지향 프로그래밍, 배열과 객체를 사용하면 여러 개의 데이터를 쉽게 다룰 수 있음, 생성자 함수(객체를 만드는 함수), 프로토타입(생성자 함수로 만든 객체는 프로토타입 공간에 메소드를 지정해서 모든 객체가 공유하도록 함)

2. 표준 내장 객체

- 기본 자료형과 객체 자료형의 차이 : 자바스크립트는 다양한 객체를 제공, 기본 자료형의 속성 또는 메소드를 사용할 때 기본 자료형이 자동으로 객체로 변환이 됨, 기본 자료형은 객체가 아니므로 속성과 메소드를 추가할 수 없음
- Number 객체 : 자바스크립트에서 숫자를 표현할 때 사용
- String 객체 : String 객체의 메소드는 변경된 값을 리턴함
- Date 객체