

## 09. 자바스크립트 객체 만들기

---

# 객체 알아보기

---

# 객체란

- 프로그래밍에서 '객체'는 데이터를 저장하고 처리하는 기본 단위
- 자바스크립트에서 객체는 관련된 정보와 동작을 함께 모아 놓은 것
- 내장 객체 : 프로그래밍을 할 때 자주 사용하는 요소들을 자바스크립트에서 미리 정의해 놓은 객체
  - 문서 객체 모델(DOM) : 웹 문서 자체도 객체이고 웹 문서에 포함된 이미지와 링크, 텍스트 필드 등은 모두 이미지 객체, 링크 객체, 폼 객체처럼 각각 별도의 객체.
  - 브라우저 객체 모델 : 웹 브라우저에서 사용하는 정보도 객체로 지정되어 있다.
- 사용자 정의 객체 : 필요할 때마다 사용자가 만들어 사용하는 객체

# 사용자 정의 객체 만들기

객체는 여러 개의 프로퍼티로 구성되어 있는데, 프로퍼티는 '키 : 값' 형태를 가지고 있다.

## 키와 값

객체를 만들 때는 객체 이름 다음에 중괄호({})를 사용하고,  
중괄호 사이에 '키 : 값' 형식으로 필요한 프로퍼티를 나열한다.  
객체의 키는 문자열이나 숫자, 심벌만 사용할 수 있고,  
각 프로퍼티는 쉼표(,)를 넣어 구분한다.

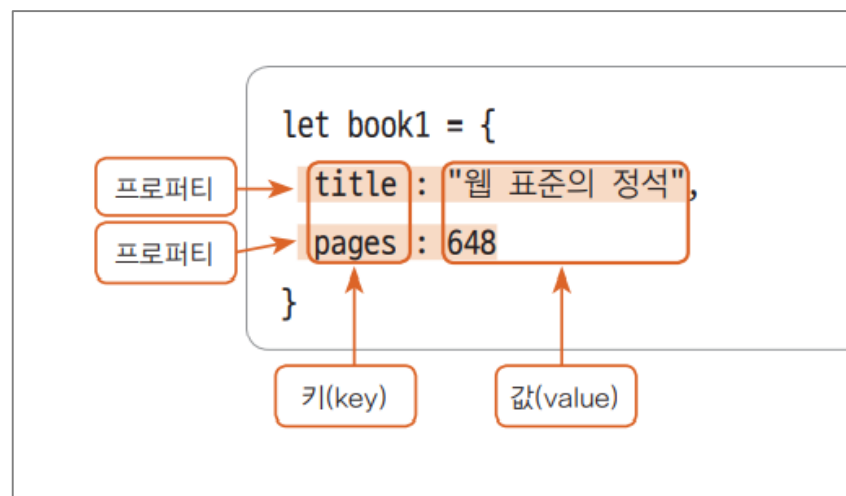
```
객체명 {  
  키1 : 값1,  
  키2 : 값2,  
  .....  
}
```

# 사용자 정의 객체 만들기

## 객체 선언하기

(예) 책 정보를 담고 있는 book1

```
let book1 = {  
  title : "웹 표준의 정석",  
  pages : 648  
}  
book1    // {title : "웹 표준의 정석", pages : 648 }
```



# 사용자 정의 객체 만들기

## 객체 프로퍼티에 접근하기

점 표기법이나 괄호 표기법 사용

괄호 표기법을 사용할 경우 프로퍼티 키의 문자열에는 큰따옴표를 붙여야 한다

```
book1.title      // 점 표기법. 프로퍼티키에 큰따옴표 없음  
book1["title"]   // 괄호 표기법. 프로퍼티키에 큰따옴표 있음
```

## 객체 프로퍼티 수정하기 및 추가하기

```
객체명.키 = 값
```

```
book1.pages = 50  
book1 // {title: "웹 표준의 정석", pages:  
50 }
```

```
book1.author = "고경희"  
book1 // {title: "웹 표준의 정석", pages: 50, author: "고  
경희"}
```

# 사용자 정의 객체 만들기

빈 객체를 만든 후 프로퍼티를 추가하면서 객체를 만들 수 있음

## 빈 객체 만들기

```
let book2 = { }
```

또는

```
let book2 = new Object()
```

## 프로퍼티 추가하기

```
book2.title = "Javascript"  
book2.pages = 500  
book2.author = "고경희"  
book2  
// {title: "Javascript", pages: 500, author:  
"고경희"}
```

## 프로퍼티 삭제하기

```
delete book2.pages  
book2  
// {title: "Javascript", author: "고경희"}
```

# 객체 중첩하기

객체 안에 또다른 객체를 넣을 수 있다. – 둘 이상의 객체 중첩

```
let student = {  
  name : "Doremi",  
  score : {  
    history : 85,  
    science : 94,  
    average : function () {  
      return (this.history + this.science) / 2  
    }  
  }  
}
```

```
student.score.history // 85
```

```
student.score.average() // 89.5
```



# 객체 메서드 정의하기

- 메서드(method) : 객체의 프로퍼티 중 객체의 동작을 지정하는 함수
- 메서드를 선언하는 방법은 일반적인 함수를 선언하는 것과 비슷하다.

```
메서드명 : function() {  
    .....  
}
```

```
let book3 = {  
    title : "점프 투 파이썬",  
    pages : 360,  
    buy : function () {  
        console.log("이 책을 구입했습니다.");  
    }  
}
```

## ES6

```
let book3 = {  
    title : "점프 투 파이썬",  
    pages : 360,  
    buy () {  
        console.log("이 책을 구입했습니다.");  
    }  
}
```

# 메서드와 this

this: 메서드에서 객체 안에 있는 프로퍼티값을 사용할 때, 현재 객체를 가리키는 예약어

```
let book4 = {  
  title : "Javascript",  
  pages : 500,  
  author : "고경희",  
  done : false,  
  finish : function() {  
    this.done === false ? console.log("읽는 중") :  
    console.log("완독");  
  }  
}
```

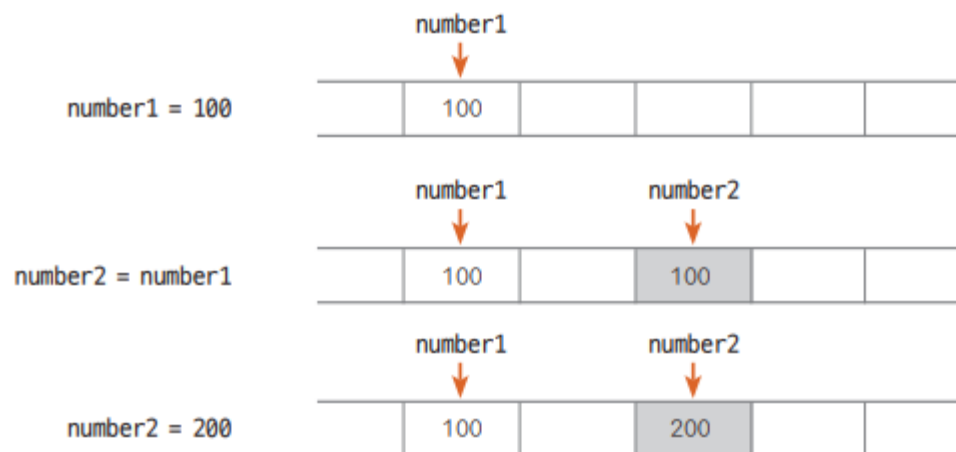
```
book4.finish() // "읽는 중"
```

# 객체 복사하기

## 원시 유형 자료 복사

'값' 을 복사한다

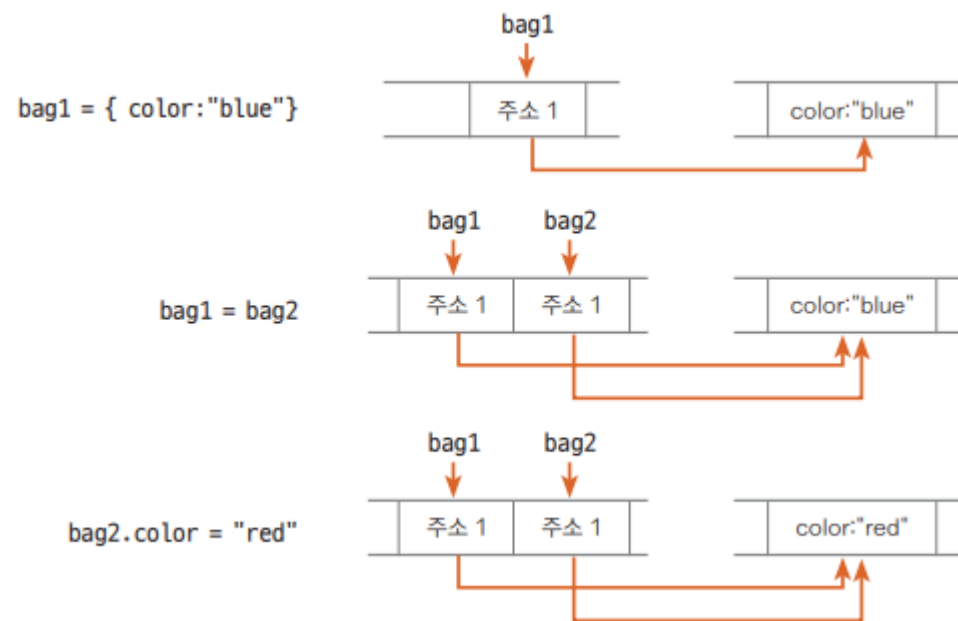
복사한 자료의 값을 변경 → 원래 자료의 값은 그대로



## 객체 복사

'주소' 를 복사한다

복사한 자료의 값 변경 → 원래 자료의 값도 바뀐다



---

# 생성자 함수와 클래스

---

# 생성자 함수

- 객체마다 반복되는 프로퍼티와 메서드가 있다면 객체 틀을 미리 정의해 놓고 필요할 때마다 그 틀을 사용해서 객체를 만들 수 있다.
- 객체의 틀을 만들 때 사용하는 함수를 '생성자 함수'라고 한다.
- 생성자 함수를 사용해서 찍어 내는 객체를 '인스턴스' 또는 '인스턴스 객체'라고 부른다



# 생성자 함수를 사용해 객체 정의하기

- 생성자 함수는 일반적인 함수와 같은 형식
- 함수 이름의 첫 글자는 대문자로 사용한다.
- 함수 내부에서 this를 사용한다

```
기본형 function 함수명(매개변수) {  
    this.키1 : 값1,  
    this.키2 : 값2,  
    :  
    this.메서드1 : function() {...},  
    this.메서드2 : function() {...},  
    :  
}
```

또는

```
기본형 const 함수명(매개변수) = function () {  
    :  
}
```

## 예) 생성자 함수를 사용해 객체 정의하기

```
function Book(title, pages, done = false) {  
  this.title = title;  
  this.pages = pages;  
  this.done = done;  
  this.finish = function () {  
    let str = "";  
    this.done === false ? str = "읽는 중" : str =  
"완독" ;  
    return str;  
  }  
}
```

← 생성자 함수로 객체 정의하기

인스턴스 객체 만들기

```
let book1 = new Book("웹 표준의 정석", 648, false);  
let book2 = new Book("점프 투 파이썬", 360, true);  
console.log(`${book1.title} - ${book1.pages}쪽 -  
${book1.finish()}`);  
console.log(`${book2.title} - ${book2.pages}쪽 -  
${book2.finish()}`);
```

# 클래스를 사용해 객체 정의하기

자바스크립트의 클래스는 정확한 클래스 개념이 아니라 생성자 함수를 좀 더 표현하기 쉽게 바꾼, 선택틱 슈거(syntactic sugar)다.

```
기본형 class 클래스명 {  
  constructor() {  
    프로퍼티1,  
    프로퍼티2,  
    :  
  }  
  메서드1() { ... }  
  메서드2() { ... }  
}
```

또는

```
기본형 const 클래스명 = class {  
  :  
}
```



## 예) 클래스를 사용해 객체 정의하기

```
class Book2 {  
  constructor(title, pages, done) {  
    this.title = title;  
    this.pages = pages;  
    this.done = done;  
  }  
  finish() {  
    let str = "";  
    this.done === false ? str = "읽는 중" : str =  
"완독";  
    return str;  
  }  
}
```

← 클래스로 객체 정의하기

인스턴스 객체 만들기

```
let book1 = new Book2("웹 표준의 정석", 648, false);  
let book2 = new Book2("점프 투 파이썬", 360, true);  
console.log(`${book1.title} - ${book1.pages}쪽 -  
${book1.finish()}`);  
console.log(`${book2.title} - ${book2.pages}쪽 -  
${book2.finish()}`);
```

---

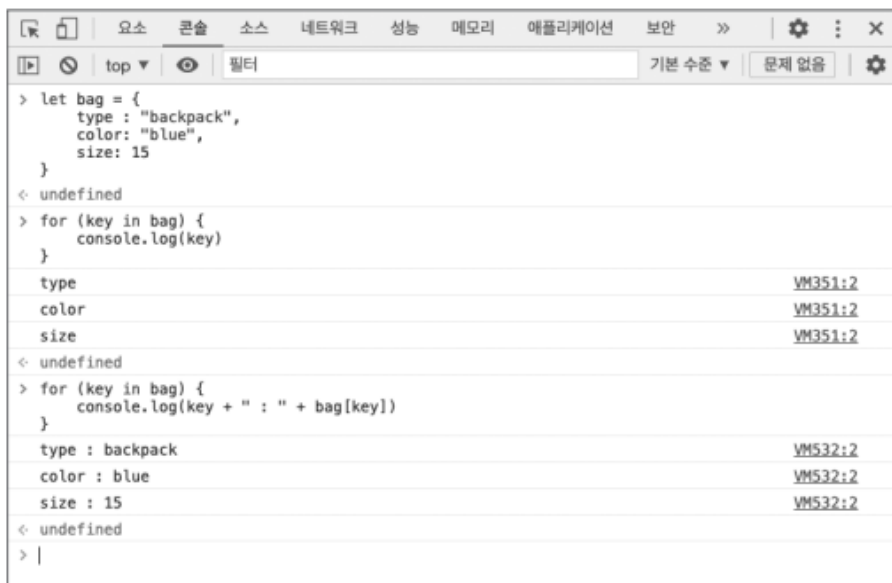
# 객체의 키와 값에 접근하기

---

# for...in 사용하기

for...in 문을 사용하면 객체의 키만 가져올 수 있다.

**기본형** for(변수 in 객체) { ... }



```
> let bag = {
  type : "backpack",
  color : "blue",
  size : 15
}
< undefined
> for (key in bag) {
  console.log(key)
}
type
color
size
< undefined
> for (key in bag) {
  console.log(key + " : " + bag[key])
}
type : backpack
color : blue
size : 15
< undefined
> |
```



예) bag 객체에서 키(key) 가져오기

```
let bag = {
  type : "backpack",
  color : "blue",
  size : 15
}

for(key in bag) {
  console.log(`${key}`);
  // type, color, size
}

for(key in bag) {
  console.log(`${key} : ${bag[key]}`);
  // type : backpack, color : blue, size : 15
}
```

# keys(), values(), entries() 메서드 사용하기

**기본형**    `Object.keys(객체명)`        `// 객체의 키만 배열로 반환합니다.`  
          `Object.values(객체명)`        `// 객체의 값만 배열로 반환합니다.`  
          `Object.entries(객체명)`        `// 객체의 [키, 값] 쌍을 배열로 반환합니다.`

(예) book1 객체의 키와 값 가져오기

```
let book1 = {  
  title : "웹 표준의 정석",  
  pages : 648,  
  buy : function () {  
    console.log("이 책을 구입했습니다.");  
  }  
}  
let keys = Object.keys(book1); // 키만 가져오기  
console.log(keys);  
let values = Object.values(book1); // 값만 가져오기  
console.log(values);  
let entries = Object.entries(book1); // 키와 값 함께 가져오기  
console.log(entries);
```

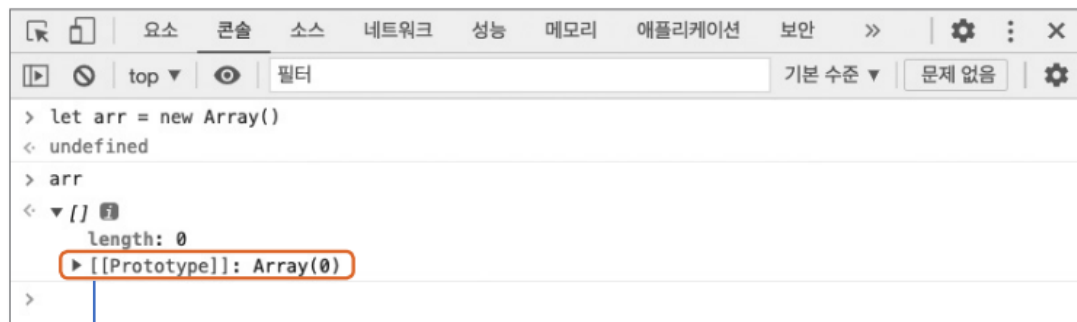
---

# 프로토타입과 클래스에서의 상속

---

# 프로토타입

- 자바스크립트 객체에서 프로토타입은 객체를 만들어 내는 원형
- 모든 객체는 프로토타입을 가지고 있고 프로토타입으로 부터 프로퍼티와 메서드를 상속받는다



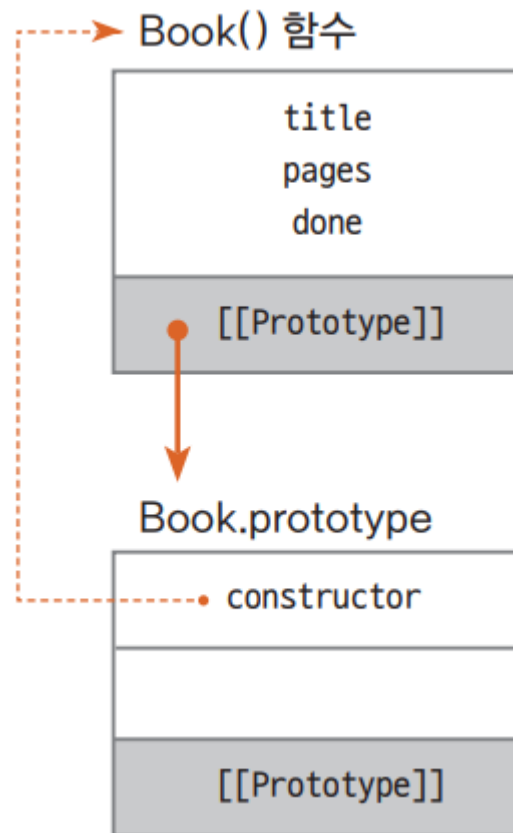
- `[[Prototype]]` 속성: `arr` 객체가 어디에서부터 온 것인지 알려 주는 속성
- `arr` 배열의 프로토타입은 `Array` 객체이고, `arr` 배열은 `Array` 객체의 프로퍼티와 메서드를 상속받는다.
- 이때 `Array` 객체를 `arr` 배열의 프로토타입(prototype)이라고 한다.

# 생성자 함수와 프로토타입 객체

생성자 함수를 선언하는 순간 자동으로 프로토타입 객체가 만들어진다.

```
const Book = function (title, pages, done) {  
  .....  
}  
  
const book1 = new Book("웹 표준의 정석", 648,  
false);  
Book 객체의 프로토타입을 확인해 보자.
```

Book.prototype

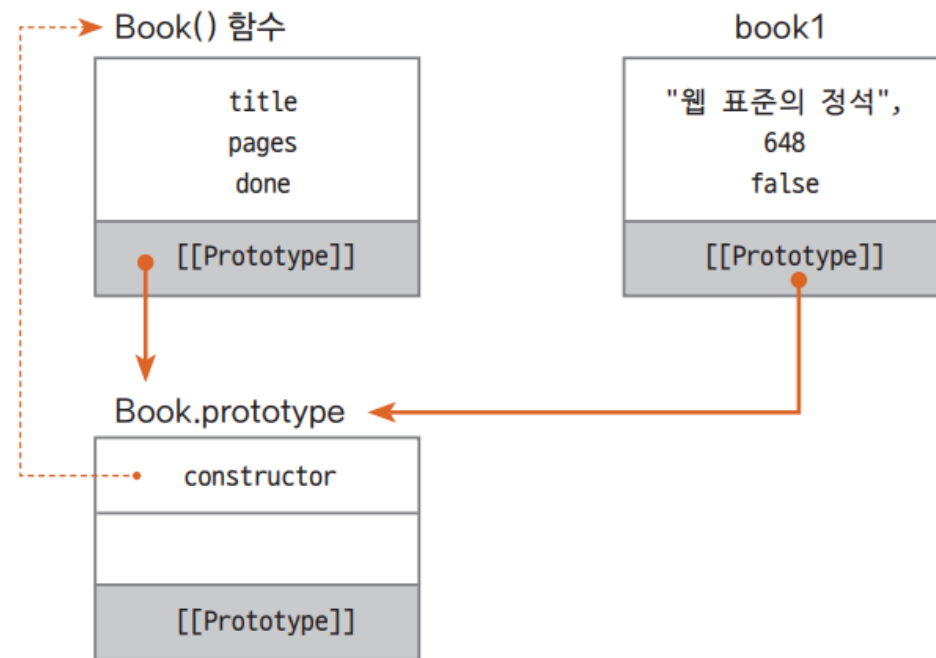


# 생성자 함수와 프로토타입 객체

인스턴스 객체 book1의 프로토타입은 무엇일까?

book1

```
> book1
< Book {title: '웹 표준의 정석', pages: 648, done: false, finish: f}
  done: false
  finish: f ()
  pages: 648
  title: "웹 표준의 정석"
  [[Prototype]]: Object
    constructor: f Book(title, pages, done = false)
    [[Prototype]]: Object
```



- book1 의 프로토타입은 객체인데,
- 이 객체는 생성자 함수 Book()을 통해 만들어진 객체
- book1 객체의 프로토타입은 Book 프로토타입 객체
- book1 객체에서는 Book() 함수에 있는 pages 프로퍼티를 사용할 수 있다.



# \_\_proto\_\_와 prototype

## \_\_proto\_\_ 프로퍼티

- 모든 객체가 가지고 있는 프로퍼티
- 인스턴스 객체에서 자신에게 연결된 프로토타입 객체를 확인할 때 사용한다.

### 예) 인스턴스 객체 book1

book1.\_\_proto\_\_

```
> book1.__proto__
< {constructor: f} ⓘ
  ▶ constructor: f Book(title, pages, done = false)
  ▶ [[Prototype]]: Object
>
```

book1에 연결된 프로퍼티 객체는 Book 객체임을 확인

## prototype 프로퍼티

- 모든 객체가 가지고 있는 프로퍼티
- 프로토타입 객체에서 자기 자신을 확인할 때 사용한다.
- 이 객체가 어떤 생성자 함수를 사용했는지, 어떤 프로퍼티와 메서드를 가지는지 등의 정보를 확인할 수 있다.

Book.prototype

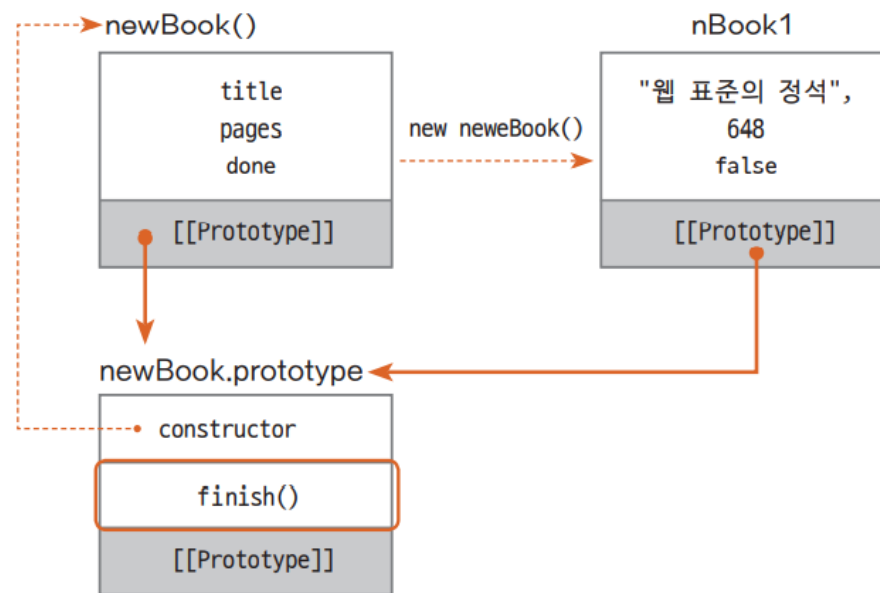
```
> Book.prototype
< {constructor: f} ⓘ
  ▶ constructor: f Book(title, pages, done = false)
  ▶ [[Prototype]]: Object
    ▶ constructor: f Object()
    ▶ hasOwnProperty: f hasOwnProperty()
    ▶ isPrototypeOf: f isPrototypeOf()
    ▶ propertyIsEnumerable: f propertyIsEnumerable()
    ▶ toLocaleString: f toLocaleString()
    ▶ toString: f toString()
    ▶ valueOf: f valueOf()
    ▶ __defineGetter__: f __defineGetter__()
    ▶ __defineSetter__: f __defineSetter__()
    ▶ __lookupGetter__: f __lookupGetter__()
    ▶ __lookupSetter__: f __lookupSetter__()
    ▶ __proto__: (...)
    ▶ get __proto__: f __proto__()
    ▶ set __proto__: f __proto__()
>
```

# 프로토타입 메서드

생성자 함수 밖에서 프로토타입을 사용해 새로운 메서드를 정의할 수 있다

(예) `newBook()` 생성자 함수

```
function newBook (title, pages, done) {  
  this.title = title;  
  this.pages = pages;  
  this.done = done;  
}  
  
newBook.prototype.finish = function() {  
  this.done === false ? str = "읽는 중" : str = "  
  완독";  
  return str;  
}  
  
const nBook1 = new newBook("웹 표준의 정석", 648,  
false):
```



# 프로토타입 상속

*기존 객체명.call(this, 프로퍼티)*

*Object.setPrototypeOf(하위 객체, 상위 객체)*

## Book 객체를 상속하는 Textbook 객체

```
function Book (title, price) {  
  this.title = title;  
  this.price = price;  
}  
Book.prototype.buy = function() {  
  console.log(`${this.title}을(를)  
  ${this.price}원에 구매하였습니다.`);  
}
```

```
function Textbook(title, price, major) {  
  Book.call(this, title, price); // Book 객체 재  
  사용  
  this.major = major; // 새로운 프로퍼티  
}  
Textbook.prototype.buyTextbook = function() {  
  console.log(`${this.major} 전공 서적,  
  ${this.title}을 구매했습니다.`);  
}
```

**Object.setPrototypeOf(Textbook.prototype,  
Book.prototype);**

# 클래스 상속

```
class BookC {  
  constructor(title, price) {  
    this.title = title;  
    this.price = price;  
  }  
  buy() {  
    console.log(`${this.title}을(를)  
    ${this.price}원에 구매하였습니다.`);  
  }  
}  
const book1 = new BookC("자료 구조",  
15000);  
book1.buy();
```

class 새 클래스명 extends 기존 클래스명

super(프로퍼티)

## BookC 클래스를 상속하는 TextbookC 클래스

```
class TextbookC extends BookC {  
  constructor(title, price, major) {  
    super(title, price); // 기존 클래스의 프로퍼티  
    재사용  
    this.major = major; // 새로운 프로퍼티  
  }  
  buyTextbook () { // 새로운 메서드  
    console.log(`${this.major} 전공 서적,  
    ${this.title}을 구매했습니다.`);  
  }  
}
```