"소극적인 노력은 고생이다. 적극적인 노력은 훈련이다."

Lecture 4
JavaScript Functions and Objects: Serious JavaScript

Samkeun Kim <skim@hknu.ac.kr>

http://cyber.hknu.ac.kr/

국립한경대학교
HANKYONG NATIONAL UNIV.

Expanding your vocabulary

You can already do a lot with JavaScript, let's take a look at some of the things you know how to do:

Grab an element from the document object model

```
<script>
  var guessInput = document.getElementById("guess");
  var guess = guessInput.value;
  var answer = null;

  var answers = [ "red",
                  "green",
                  "blue"];

  var index = Math.floor(Math.random() * answers.length);

  if (guess == answers[index]) {
      answer = "You're right! I was thinking of " + answers[index];
  } else {
      answer = "Sorry, I was thinking of " + answers[index];
  }
  alert(answer);
</script>
```

Get the value of a form input text field

Create a new array filled with strings

Use libraries of functions

Get a property of an array, like length

Make decisions based on conditionals

Use the elements of an array

Use browser functions, like alert

```
var guessInput = document.getElementById("guess");
var guess = guessInput.value;

var answer = checkGuess(guess);
alert(answer);
```

We've grabbing the user's guess just like we were on the previous page...

but rather than having all the rest of the code on the previous page as part of the main code, we'd rather just have a nice "checkGuess" function we can call that does the same thing

Expanding your vocabulary

So far, though, a lot of your knowledge is **informal** — sure, you can get an element out of the DOM and assign some new HTML to it, but if we asked you to explain exactly

what **document.getElementById** is technically,

well, that might be a little more **challenging**.

Now to get you there, we're not going to start with a ~~deep, technical analysis~~ of **getElementById**, no no, we're going to do something a little more interesting:

We're going to extend JavaScript's vocabulary and make it do some new things.

How to add your own functions

Create a checkGuess function

1. To create a function, use the function keyword and then follow it with a name, like "checkGuess".

2. Give your function zero or more parameters. Use parameters to pass values to your function. We need just one parameter here: the user's guess.

```
function checkGuess(guess) {
    var answers = [ "red",
                    "green",
                    "blue"];

    var index = Math.floor(Math.random() * answers.length);

    if (guess == answers[index]) {
        answer = "You're right! I was thinking of " + answers[index];
    } else {
        answer = "Sorry, I was thinking of " + answers[index];
    }
    return answer;
}
```

3. Write a body for your function, which goes between the curly braces. The body contains all the code that does the work of the function. For the body here, we'll reuse our code from the previous page.

4. Optionally, return a value as the result of calling the function. Here we're returning a string with a message.

How a function works

새로운 bark() 함수를 정의해 보자:

- 2개의 파라메터를 가진다: **dogName** and **dogWeight**
- Dog의 몸무게에 따라 Dog의 짖는 소리(bark)를 리턴한다

```
function bark(dogName, dogWeight) {
    if (dogWeight <= 10) {
      return dogName + " says Yip";
    } else {
      return dogName + " says Woof";
    }
}
```

Here's our handy bark function.

Now let's invoke it!

How a function works

함수를 호출해 보자!

- 함수 이름(name)을 사용하고 요구되는 아규먼트(arguments)를 제공한다.

Our function name.

Here we're passing two arguments, the name and the weight.

```
bark("Fido", 50);
```

"Fido"

50

When we call bark, the arguments are assigned to the parameter names in the bark function.

```
function bark(dogName, dogWeight) {
    if (dogWeight <= 10) {
      return dogName + " says Yip";
    } else {
      return dogName + " says Woof";
    }
}
```

And any time the parameters appear in the function, the values we passed in are used.

How a function works

함수의 바디(body)를 동작시켜 보자!

- 바디의 문장(statements)은 **탑다운**(from top to bottom) 방식으로 **평가**된다
- 파라메터 **dogName**과 **dogWeight**는 함수에 전달된 아규먼트를 **할당받는다**

"Fido" 50

```
function bark(dogName, dogWeight) {
    if (dogWeight <= 10) {
      return dogName + " says Yip";
    } else {
      return dogName + " says Woof";
    }
}
```

Here we evaluate all the code in the body.

How a function works

함수 바디에 return 문장을 가질 수 있다.

- 호출을 수행한 코드에 값(value)을 리턴해 준다.

Tracing through the function, dogWeight is not less than or equal to 10 so we use the else clause and return "Fido says Woof" as a string value.

The string "Fido says Woof" is returned to the calling code (that's the code that invoked the bark function).

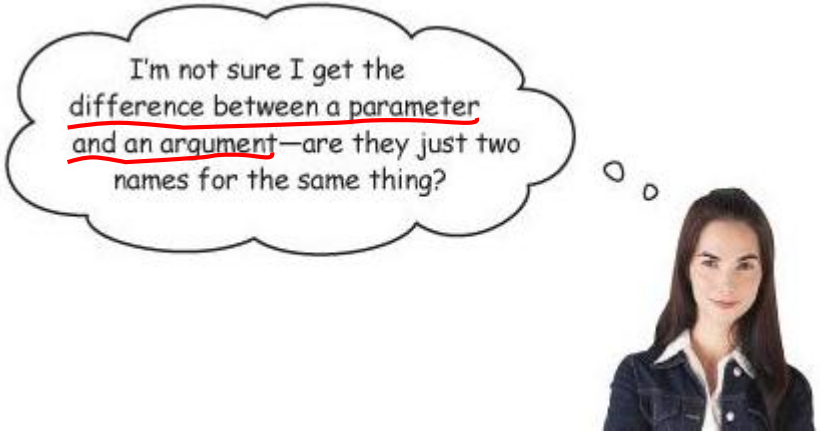
And when the string is returned, it is assigned to the variable sound, which is then passed to alert, resulting in the dialog.

```
function bark(dogName, dogWeight) {
  if (dogWeight <= 10) {
    return dogName + " says Yip";
  } else {
    return dogName + " says Woof";
  }
}
```

"Fido says Woof"

```
var sound = bark("Fido", 50);
alert(sound);
```

Think of the ***HTML5 JavaScript APIs*** as made up of *objects*, *methods* (otherwise known as functions) and *properties*.



No, they're different.

When you define a function you can define it with one or more **parameters**.

When you call a function, you call it with **arguments**:

Here we're defining three parameters: degrees, mode and duration.

```
function cook(degrees, mode, duration) {
   // your code here
}
```

```
cook(425.0, "bake", 45);
```

These are arguments. There are three arguments, a floating point number, a string and an integer.

```
cook(350.0, "broil", 10);
```

순서 중요!!

So you'll only define your parameters once, but you'll probably call your functions with a lot of different arguments.

Anatomy of a Function

Now that you know how to define a function, let's make sure we've got the **syntax** down cold. Here are all the parts of a function's anatomy:

Even if your function has no parameters, you still need an opening and closing set of parens, like ().

Follow the function keyword with the name of your function.

And then zero or more comma separated parameters between parentheses.

Always start with the keyword "function".

```
function addScore ( level , score ) {
    var bonus = level * score * .1;
    return score + bonus;
}
```

The body sits between two curly braces and contains a set of statements (just like the statements you're used to).

Here's the closing brace of the body.

A function can include a statement with the return keyword, but it doesn't have to.

The return statement includes an expression, which is returned as a result of calling the function.

실습과제 4-1: Sharpen Your Pencil (계속)

Use your knowledge of functions and passing arguments to parameters to evaluate the code below.

After you've traced through the code, write the value of each variable below.

```
function dogsAge(age) {
    return age * 7;
}

var myDogsAge = dogsAge(4);

function rectangleArea(width, height) {
    var area = width * height;
    return area;
}

var rectArea = rectangleArea(3, 4);

function addUp(numArray) {
    var total = 0;
    for (var i = 0; i < numArray.length; i++) {
        total += numArray[i];
    }
    return total;
}
```

실습과제 4-1: Sharpen Your Pencil (계속)

```
var theTotal = addUp([1, 5, 3, 9]);

function getAvatar(points) {
    var avatar;
    if (points < 100) {
        avatar = "Mouse";
    } else if (points > 100 && points < 1000) {
        avatar = "Cat";
    } else {
        avatar = "Ape";
    }
    return avatar;
}
var myAvatar = getAvatar(335);
```

Write the value of each variable here...

myDogsAge = ...

rectArea = ...

theTotal = ...

myAvatar = ...

지역변수(local)와 전역변수(global)

Know the difference or risk humiliation

- var 키워드를 이용하여 변수(variable)를 선언할 수 있다

```
var avatar;
var levelThreshold = 1000;
```

These are global variables; they're accessible everywhere in your JavaScript code.

지역변수(local)와 전역변수(global)

함수 내부(inside)에 변수를 선언할 수 있다:

```
function getScore(points) {
    var score;

    for (var i = 0; i < levelThreshold; i++) {
        //code here
    }

    return score;
}
```

The points, score and i variables are all declared within a function.

We call them local variables because they are only known locally within the function itself.

Even if we use levelThreshold inside the function, it's global because it's declared outside the function.

- 변수가 함수 내부(inside)에 선언되면: **LOCAL 변수**
- 변수가 함수 바깥부분(outside)에 선언되면: **GLOBAL 변수**

지역변수와 전역변수의 스코프(Scope)

함수의 변수는 지역적인 스코프 (**locally scoped**)를 갖는다:

변수를 정의하는 위치가 변수의 Scope를 결정한다.

함수 바깥쪽에서 정의한 변수는 전역적인 스코프 (**globally scoped**)를 갖는다:

```
var avatar = "generic";
var skill = 1.0;
var pointsPerLevel = 1000;
var userPoints = 2008;

function getAvatar(points) {
    var level = points / pointsPerLevel;

    if (level == 0) {
        return "Teddy bear";
    } else if (level == 1) {
        return "Cat";
    } else if (level >= 2) {
        return "Gorilla";
    }
}

function updatePoints(bonus, newPoints) {
    for (var i = 0; i < bonus; i++) {
        newPoints += skill * bonus;
    }
    return newPoints + userPoints;
}

userPoints = updatePoints(2, 100);
avatar = getAvatar(2112);
```

These four variables are globally scoped. That means they are defined and visible in all the code below.

Note that if you link to additional scripts in your page, they will see these global variables too!

The level variable here is local and is visible only to the code within the getAvatar function. That means only this function can access the level variable.

And let's not forget the points parameter, which also has local scope in the getAvatar function.

Note that getAvatar makes use of the pointsPerLevel global variable too.

In updatePoints we have a local variable i. i is visible to all of the code in updatePoints.

bonus and newPoints are also local to updatePoints, while userPoints is global.

And here in our code we can use only the global variables, we have no access to any variables inside the functions because they're not visible in the global scope.

변수의 짧은 인생

변수의 고단한 일생은 매우 **짧을** 수 있다.

즉 변수가 **글로벌**(global)이 아니라면, 심지어 글로벌 변수라 해도 일생이 **제한**을 갖는다.

무엇이 변수의 일생을 결정하는가? 다음처럼 한번 생각해 보자:

글로벌 변수는 브라우저에 페이지가 존재하는 한 살아있다.

- 글로벌 변수는 자바스크립트 코드가 페이지에 **로드될 때** 일생을 시작한다. 그러나 페이지가 사라지면 글로벌 변수의 일생도 종료된다.
- 같은 페이지가 다시 로드된다 해도 모든 글로벌 변수는 소멸되었기 때문에 새로 로드된 페이지에서 다시 생성된다.

지역변수는 함수가 종료될 때 사라진다.

- 지역변수는 함수가 최초 호출될 때 생성되어 함수가 값을 **리턴할 때까지** 살아있다.

지역변수와 전역변수의 이름이 같다면?

You "shadow" your global

전역변수 beanCounter와 아래와 같은 함수가 정의되었다고 하자:

```
var beanCounter = 10;

function getNumberOfItems(ordertype) {
    var beanCounter = 0;
    if (ordertype == "order") {
        // do some stuff with beanCounter...
    }
    return beanCounter;
}
```

We've got a global and a local!

- 함수 안에서 beanCounter에 대한 모든 레퍼런스는 지역변수를 참조하는 것으로 간주된다.
- 따라서 전역변수를 지역변수의 그림자 영역(shadow) 안에 있다고 말할 수 있다.

Oh, did we mention functions are also values?

변수를 사용하여 **numbers, boolean values, strings, arrays** 등을 저장(store)할 수 있다.
그러나 변수에 함수를 할당(assign)할 수 있을까?

```
function addOne(num) {
    return num + 1;
}

var plusOne = addOne;

var result = plusOne(1);
```

Let's define a simple function that adds one to its argument.

Now let's do something new. We'll use the name of the function addOne and assign addOne to a new variable, plusOne.

Notice we're not calling the function with addOne(), we're just using the function name.

plusOne is assigned to a function, so we can call it with an integer argument of 1.

After this call result is equal to 2.

Oh, did we mention functions are also values?

함수는 익명(anonymous)이 될 수 있다:

- 왜 이런 함수를 필요로 할까?
- **이름없는 함수**(function without a name)를 정의해 보자:

```
function(num) {
    return num + 1;
}

var f = function(num) {
    return num + 1;
}
var result = f(1);
alert(result);
```

이름없는 함수 생성

변수에 함수 할당

그 변수를 사용하여 함수 호출

결과는?

What you can do with functions as values

왜 이게 유용할까?

여기서 중요한 점은 변수에 함수를 할당할 수 있다는 것이 아니라 **함수가 실제로는 값(value)이라는 사실이다**

Here's a simple init function.

```
function init() {
    alert("you rule!");
}
window.onload = init;
```

Here we're assigning the function we defined to the onload handler.

Hey look, we were already using functions as values!

Or we could get even fancier:

함수의 값을 직접 할당!!

So here we're creating a function, without an explicit name, and then assigning its value to the window.onload property directly.

```
window.onload = function() {
    alert("you rule!");
}
```

Don't worry if window.onload is still a little unclear, we're just about to cover all that.

Wow, isn't that simpler and more readable?


Authors? Hello?
Hello? I'm the girl who bought
the HTML5 book, remember me?
What does all this have to do with
HTML5???


With objects, the future's so bright we really DO have to wear shades...

Did someone say “Objects”?!

객체(Objects)는 자바스크립트 프로그래밍 기술을 한 수준 높여준다.

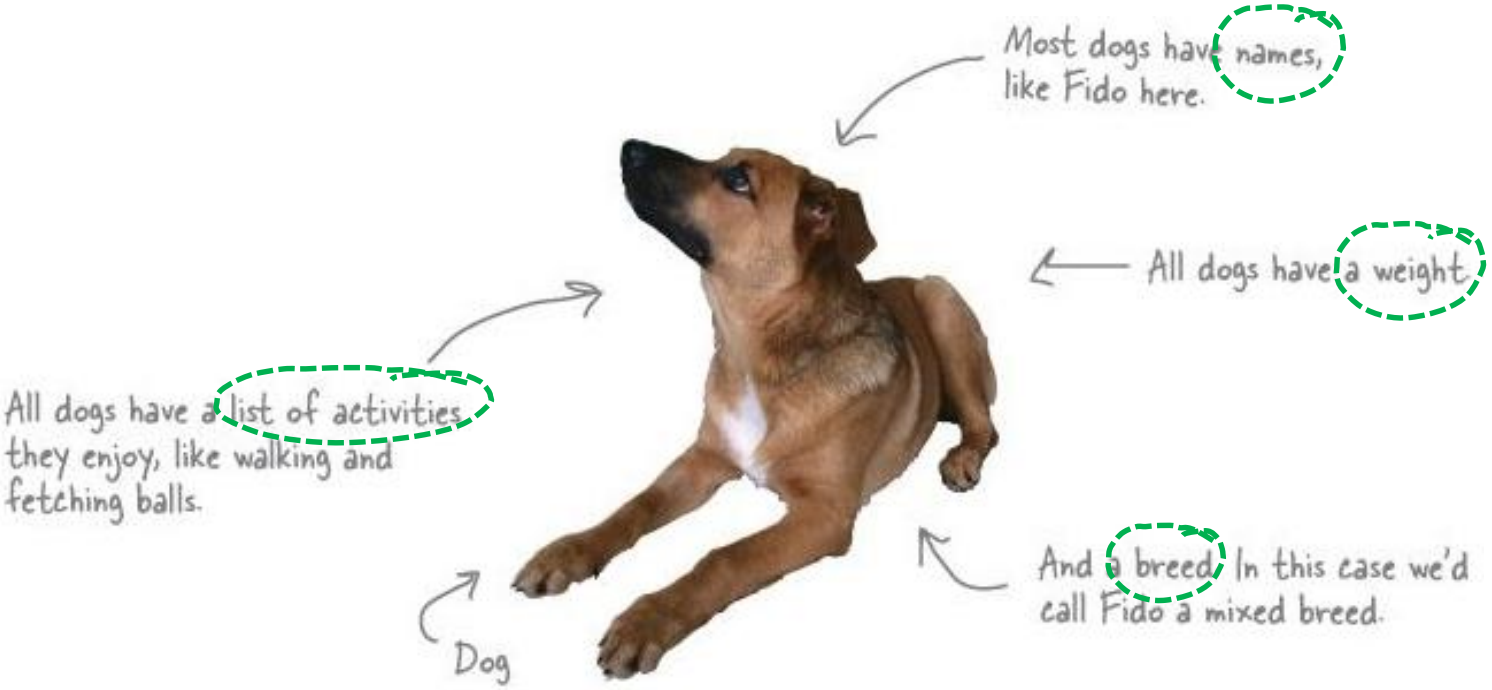
Objects are going to take your JavaScript programming skills to the next level — they’re the key

- to managing complex code,
- to understanding the DOM,
- to organizing your data,
- and they’re even the fundamental way HTML5 JavaScript APIs are packaged up (and that’s just our short list!).

Did someone say "Objects"?!

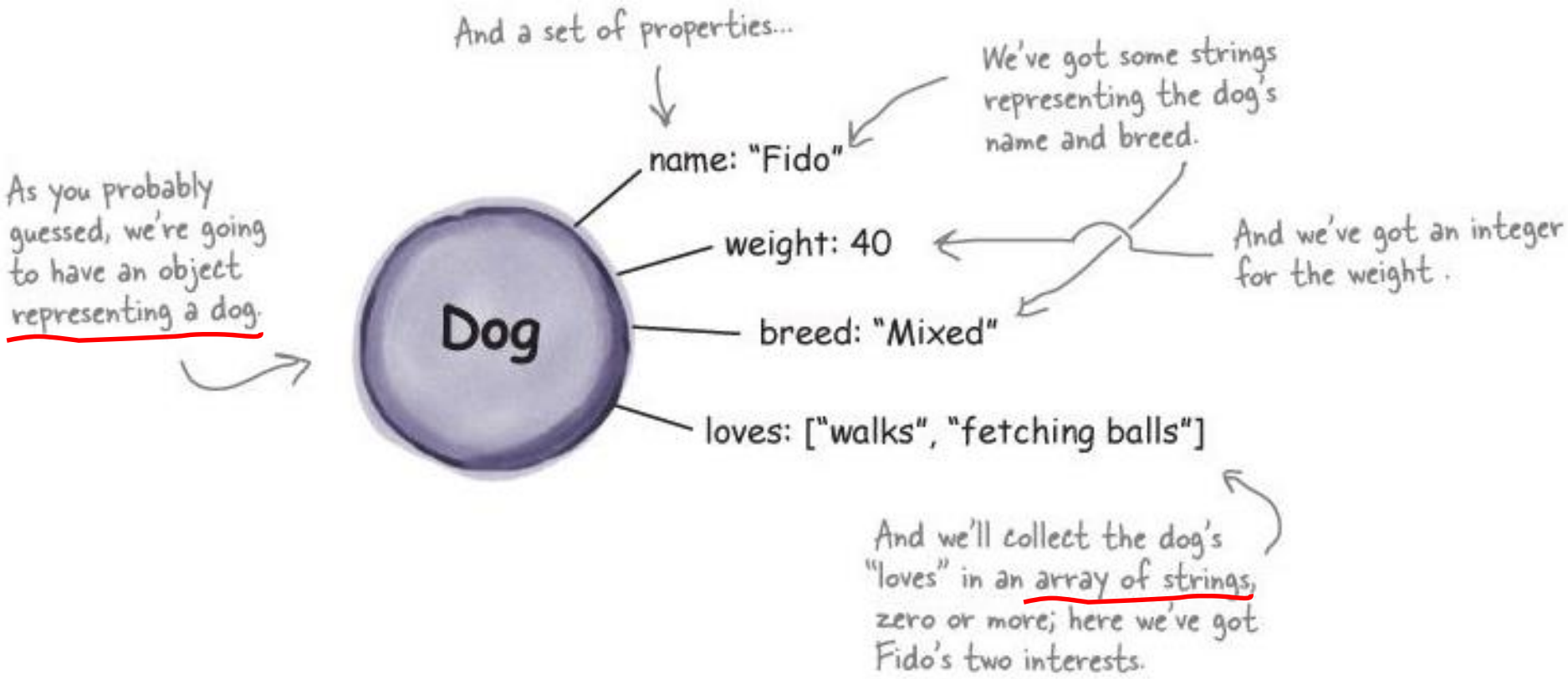
Here's the secret to **JavaScript objects**:
they're just a collection of properties.

Let's take an example, say, a dog. A dog's got properties:



Thinking about properties...

Let's think about those **properties** in terms of **JavaScript data types:**



How to create an object in JavaScript

So we've got an **object** with some **properties**;

How do we create this using JavaScript?

We're going to assign our object to the variable fido.

Start an object with just the left curly brace, then all the properties are going to go inside.

Notice that each property is separated by a comma. NOT a semicolon!

```
var fido = {
    name: "Fido",
    weight: 40,
    breed: "Mixed",
    loves: ["walks", "fetching balls"]
};
```

This object has four properties, name, weight, breed and loves.

Notice that the value of weight is a number, 40, and the values of breed and name are strings.

And of course we have an array to hold the dog's loves.

Some things you can do with objects

1. Access **object properties** with "**dot**" notation:

```
if (fido.weight > 25) {
   alert("WOOF");
} else {
   alert("yip");
}
```

Use the object along with a "." and a property name to access the value of that property.

Use a "."

fido.weight

Here's the object...

... and then the property name.

2. Access **properties** using a string with **[ ]** notation:

```
var breed = fido["breed"];
if (breed == "mixed") {
    alert("Best in show");
}
```

Use the object along with the property name wrapped in quotes and in brackets to access the value of that property.

Now we use [ ] around the property name.

fido["weight"]

Here's the object...

... and the property name in quotes.

We find dot notation the more readable of the two.

Some things you can do with objects

3. Change a **property's value**:

```
fido.weight = 27;
fido.breed = "Chawalla/Great Dane mix";
fido.loves.push("chewing bones");
```

We're changing Fido's weight...

... his breed...

... and adding a new item to his loves array.

push simply adds a new item to the end of an array.

4. Enumerate all an **object's properties**:

To enumerate is to go through all the properties of the object.

```
var prop;
for (prop in fido) {
    alert("Fido has a " + prop + " property ");
    if (prop == "name") {
        alert("This is " + fido[prop]);
    }
}
```

To enumerate the properties we use a for-in loop.

Each time through the loop, the variable prop gets the string value of the next property name.

And we use the [ ] notation to access the value of that property.

Note the order of the properties is arbitrary, so don't count on a particular ordering.

Some things you can do with objects

5. Have fun with an **object's array**:

```
var likes = fido.loves;
var likesString = "Fido likes";

for (var i = 0; i < likes.length; i++) {
    likesString += " " + likes[i];
}
alert(likesString);
```

Here, we're assigning the value of fido's loves array to the variable likes.

We can loop through the likes array and create a likesString of all fido's interests.

And we can alert the string.

6. **Pass an object to a function**:

```
function bark(dog) {
    if (dog.weight > 25) {
        alert("WOOF");
    } else {
        alert("yip");
    }
}

bark(fido);
```

We can pass an object to a function just like any other variable.

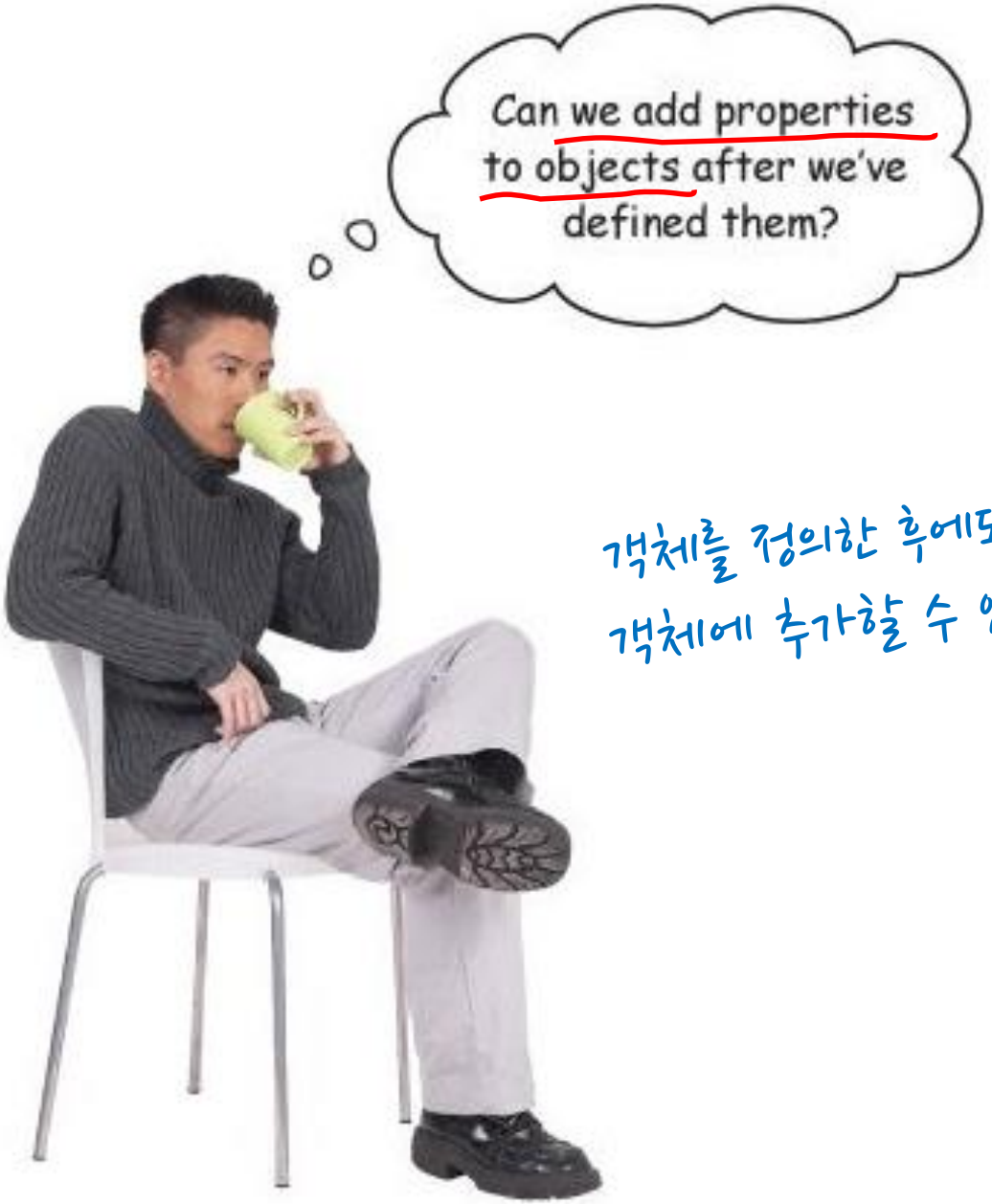
And in the function, we can access the object's properties like normal, using the parameter name for the object, of course.

We're passing fido as our argument to the function bark, which expects a dog object.

Note

The dot operator(.) gives you access to an object's properties:

- **fido.weight** is the size of fido
- **fido.breed** is the breed of fido
- **fido.name** is the name of fido
- **fido.loves** is an array containing fido's interests



객체를 정의한 후에도 프로퍼티를 객체에 추가할 수 있을까?

Yes, you can add or delete properties at any time

객체에 프로퍼티를 추가하는 방법:

단순히 새로운 프로퍼티에 값을 할당한다:

fido.age = 5;

이 시점부터 객체 fido는 새로운 프로퍼티 age를 가지게 된다.

마찬가지로 delete 키워드를 이용하여 프로퍼티를 삭제할 수 있다:

delete fido.age;

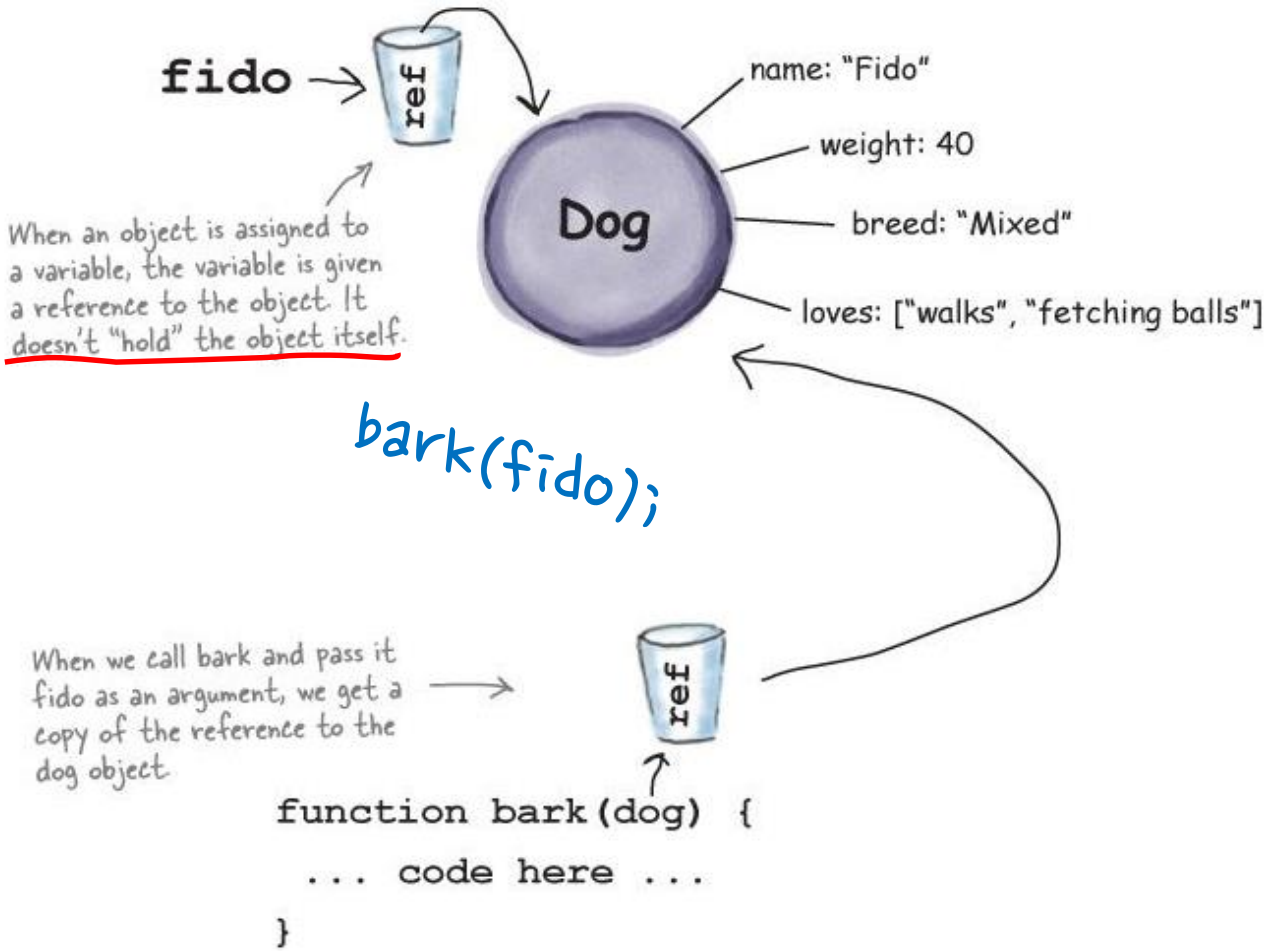
이것은 프로퍼티 값만 삭제하는 것이 아니라 프로퍼티 자체를 삭제하는 것이다.

Let's talk about passing objects to functions:

Let's talk about passing objects to functions

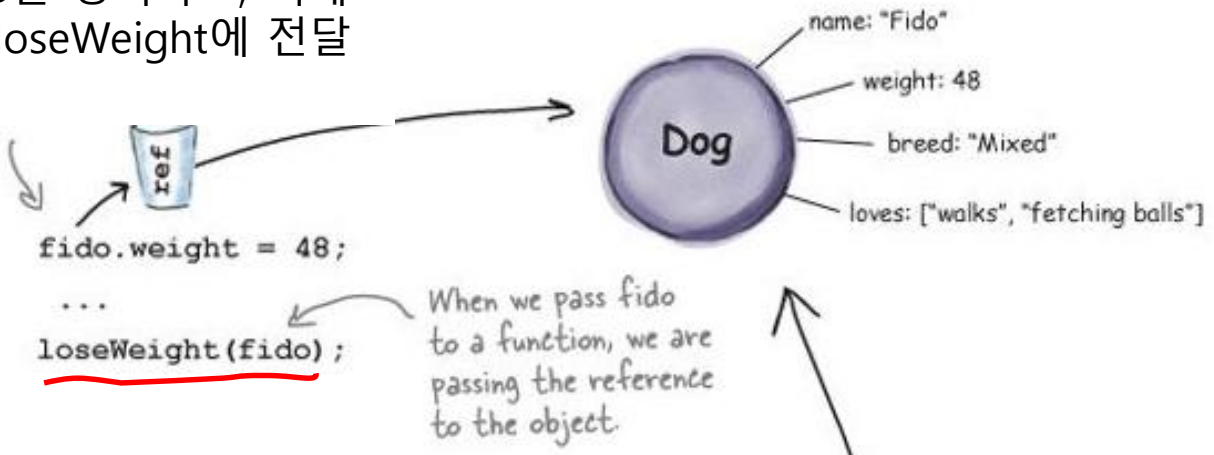
객체가 변수에 할당될 때 그 변수는 객체 자체가 아니라 객체에 대한 레퍼런스를 저장하게 된다.

- ✓ 레퍼런스를 객체에 대한 포인터로 간주하라

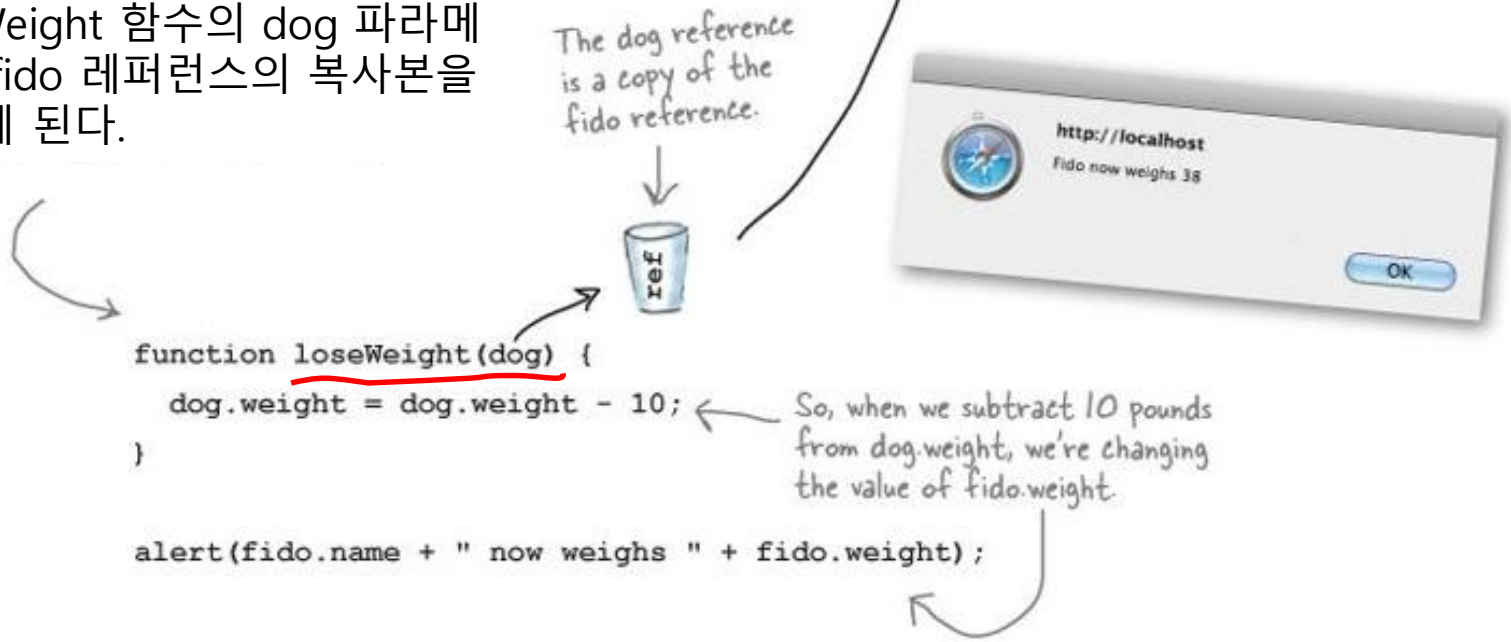


Putting Fido on a diet....

1. 객체 fido를 정의하고, 객체를 함수 loseWeight에 전달한다.



2. loseWeight 함수의 dog 파라메터는 fido 레퍼런스의 복사본을 가지게 된다.



Now Showing at the Webville Cinema

2개의 간단한 movie 객체를 설계해 보자:

각 객체는 a **title**, a **genre**, a **movie rating** (1-5 stars) and a **set of showtimes**를 포함한다

샘플 데이터:

Plan 9 from Outer Space, which shows at 3: 00pm, 7: 00pm and 11: 00pm; it's in the genre "cult classic"; and has a 2-star rating.

Forbidden Planet, which shows at 5: 00pm and 9: 00pm; is in the genre "classic sci-fi"; and has a 5-star rating.

Now Showing at the Webville Cinema

movie 객체를 어떻게 생성할 수 있을까?

We created two objects, movie1 and movie2 for the two movies.

movie1 has four properties, title, genre, rating and showtimes.

```
var movie1 = {
    title: "Plan 9 from Outer Space",
    genre: "Cult Classic",
    rating: 5,
    showtimes: ["3:00pm", "7:00pm", "11:00pm"]
};
```

title and genre are string

rating is a number.

And showtimes is an array containing the show times of the movie as strings.

movie2 also has four properties, title genre, rating and showtimes.

```
var movie2 = {
    title: "Forbidden Planet",
    genre: "Classic Sci-fi",
    rating: 5,
    showtimes: ["5:00pm", "9:00pm"]
};
```

Remember to separate your properties with commas.

We use the same property names but different property values as movie1.

Our next showing is at....

우리는 **객체와 함수를 혼합하여 사용**하는 것을 이미 살펴보았다

이제 한 단계 더 나아가서 영화의 다음 상영시간(showtime)을 알려주는 자바 스크립트 코드를 작성해 보자

작성하게 될 함수는 movie 객체를 아규먼트로 받아 들여 현재시간을 기준으로 다음 상영시간을 알려주는 문자열을 리턴하도록 한다

Here's our new function, which takes a movie object.

We're grabbing the current time using JavaScript's Date object. We're not going to worry about the details of this one yet, but just know that it returns the current time in milliseconds.

1970년 1월 1일 자정 ~

```
function getNextShowing(movie) {
    var now = new Date().getTime();

    for (var i = 0; i < movie.showtimes.length; i++) {
        var showtime = getTimeFromString(movie.showtimes[i]);
        if ((showtime - now) > 0) {
            return "Next showing of " + movie.title + " is " + movie.showtimes[i];
        }
    }
    return null;
}
```

Now use the movie's array, showtimes, and iterate over the showtimes.

For each showtime we get its time in milliseconds and then compare.

If the time hasn't happened yet, then it's the next showing, so return it.

If there are no more shows, we just return null;

"3:00pm"

```
function getTimeFromString(timeString) {
    var theTime = new Date();
    var time = timeString.match(/(\d+)(?::(\d\d))?\s*(p?)/);
    theTime.setHours( parseInt(time[1]) + (time[3] ? 12 : 0) );
    theTime.setMinutes( parseInt(time[2]) || 0 );
    return theTime.getTime();
}
```

Ready Bake Code

Here's some ready bake code that just takes a string with the format like 1am or 3pm and converts it to a time in milliseconds.

Don't worry about this code; it uses regular expressions, which you'll learn later in your JavaScript education. For now, just go with it!

```
var nextShowing = getNextShowing(movie1);
alert(nextShowing);
nextShowing = getNextShowing(movie2);
alert(nextShowing);
```

Now we use the function by calling getNextShowing and use the string it returns in an alert.

And let's do it again with movie2.

How "Chaining" works...

Did you catch this in the previous code?

movie.showtimes.length 축약형

That doesn't look like anything we've seen before. This is really just a shorthand for a series of steps we could have taken to get the length of the showtimes array from the movies object. We could have written this instead:

```
var showtimesArray = movie.showtimes;
var len = showtimesArray.length;
```

First we grab the showtimes array.

Then we use it to access the length property.

But we can do all this in one shot by chaining together the expressions. Let's step through how this works:

```
movie.showtimes.length
```

1. Evaluates to the movie object.
2. Which has a showtimes property, that is an array.
3. Which has a property named length.

실습과제 4-2

Get the code on the previous page typed in and let's give it a test run. You'll see that the **getNextShowing** function takes whatever movie it is handed and figures out the next showing time. Feel free to create some new movie objects of your own and give them a test drive too.
현재시간에 비추어 적어도 하나의 showtime이 디스플레이 되도록 showtimes 배열의 내용을 조정하시오. 예를 들어, 로컬 타임이 12:30pm인 경우:


```
var banzaiMovie = {
    title: "Buckaroo Banzai",
    genre: "Cult classic",
    rating: 5,
    showtimes: ["1:00pm", "5:00pm", "7:00pm"]
};

var nextShowing = getNextShowing(banzaiMovie);
alert(nextShowing);
```

Objects can have behavior too...

객체는 '액티브'하다: 그들은 뭔가를 할 수 있다(can do things)

실제 개들(dogs)은 짖고, 달리고, 공을 주어온다. Dog 객체도 마찬가지로 그렇게 한다

```
var fido = {
    name: "Fido",
    weight: 40,
    breed: "Mixed",
    loves: ["walks", "fetching balls"]
    bark: function() {
        alert("Woof woof!");
    }
};
```

We can add a function directly to our object like this.

Notice we're making use of an anonymous function and assigning it to the bark property of the object.

Rather than saying this is a "function in the object," we just say this is a method. They're the same thing, but everyone refers to object functions as methods.

When an object has a function in it, we say that object has a method (object function)

객체의 메소드를 호출하려면 **dot 표기법**을 사용하면 된다.

아규먼트가 필요한 경우 제공할 수 있다.

Meanwhile back at Webville Cinema...

앞에서 **movie 객체**를 아규먼트로 받아 들이는 getNextShowing 함수를 작성해 보았다.
이제 이 함수를 **메소드**로 정의함으로써 **movie 객체**의 일부분으로 만들 수 있다.

```
var movie1 = {
    title: "Plan 9 from Outer Space",
    genre: "Cult Classic",
    rating: 5,
    showtimes: ["3:00pm", "7:00pm", "11:00pm"],

    getNextShowing: function(movie) {
        var now = new Date().getTime();

        for (var i = 0; i < movie.showtimes.length; i++) {
            var showtime = getTimeFromString(movie.showtimes[i]);
            if ((showtime - now) > 0) {
                return "Next showing of " + movie.title + " is " + movie.showtimes[i];
            }
        }
        return null;
    }
};
```

We've taken our code and placed it in a method of the movie1 object with the property name getNextShowing.

But we know that can't be quite right...

getNextShowing 프로퍼티는 movie 아규먼트를 받는다

var nextShowing = getNextShowing(**movie1**);

우리가 정말로 원하는 것은 다음과 같은 형태의 getNextShowing 이다

```
var nextShowing = movie1.getNextShowing();
```

No argument should be needed here, it's clear which movie we want the next showing of, that is, we want movie1.

fido.bark()처럼

How do we fix this?

이제 getNextShowing 메소드 정의로부터 파라메터를 제거해야 한다.

그러나 **movie.showtimes**에 대한 모든 레퍼런스에 뭔가를 처리해 주어야만 한다.

왜냐하면 파라메터를 제거했기 때문에 movie가 더 이상 변수로 존재하지 않기 때문이다.

Let's get the movie parameter out of there...

다음은 movie 파라미터를 제거하고 그에 대한 모든 레퍼런스를 제거한 것이다:

```
var movie1 = {
    title: "Plan 9 from Outer Space",
    genre: "Cult Classic",
    rating: 5,
    showtimes: ["3:00pm", "7:00pm", "11:00pm"],

    getNextShowing: function() {
        var now = new Date().getTime();

        for (var i = 0; i < showtimes.length; i++) {
            var showtime = getTimeFromString(showtimes[i]);
            if ((showtime - now) > 0) {
                return "Next showing of " + title + " is " + showtimes[i];
            }
        }
        return null;
    }
};
```

We've highlighted the changes below...

제거됨

This all looks pretty reasonable, but we need to think through how the getNextShowing method will use the showtimes property...

...we're used to either local variables (which showtimes isn't) and global variables (which showtimes isn't). Hmmmm....

Oh, and here's another one, the title property.

Now what?

난제(Conundrum):

- 현재 프로퍼티 showtimes와 title에 대한 레퍼런스를 얻었다
- 보통 함수에서는 지역변수, 전역변수, 함수의 파라메터를 참조하게 된다
- 그러나 showtimes와 title은 movie1객체의 프로퍼티이다

이 코드가 작동될 수 있을까?

- Nope. It doesn't work.
- 자바스크립트는 showtimes 와 title 변수가 **undefined**라고 간주한다

How can that be?

Now what?

이들 변수는 **객체의 프로퍼티**이지만 어느 객체의 프로퍼티인 지는 자바스크립트에게 말해주지는 않는다

현재 취급하고 있는 바로 이(**THIS**) 객체라고 말해줄 수단이 필요하다

사실, this라는 자바스크립트 keyword가 있다

- 이 this가 자바스크립트에게 바로 현재 다루고 있는 이 객체를 의미하도록 해주는 방법을 제공한다

Adding the "this" keyword

프로퍼티로 지정한 모든 곳에 **this**를 추가하자.

- 자바스크립트에게 this 객체에 있는 프로퍼티를 원한다는 사실을 알려준다.

```
var movie1 = {
    title: "Plan 9 from Outer Space",
    genre: "Cult Classic",
    rating: 5,
    showtimes: ["3:00pm", "7:00pm", "11:00pm"],

    getNextShowing: function() {
        var now = new Date().getTime();

        for (var i = 0; i < this.showtimes.length; i++) {
            var showtime = getTimeFromString(this.showtimes[i]);
            if ((showtime - now) > 0) {
                return "Next showing of " + this.title + " is " + this.showtimes[i];
            }
        }
        return null;
    }
};
```

Here we've added a this keyword before every property to signify we want the movie1 object reference.

Go ahead and type in the code above and also add the **getNextShowing** function to your movie2 object (just **copy** and **paste** it in).

Then make the changes below to your previous test code. After that give it a spin! Here's what we got:

```
var nextShowing = movie1.getNextShowing();
alert(nextShowing);
nextShowing = movie2.getNextShowing();
alert(nextShowing);
```

Note that we're now calling getNextShowing ON the object. Makes more sense, doesn't it?

```
var nextShowing = movie1.getNextShowing();
alert(nextShowing);
nextShowing = movie2.getNextShowing();
alert(nextShowing);
```

Note that we're now calling getNextShowing ON the object. Makes more sense, doesn't it?

중복된다!!

It seems like we're duplicating code with all the copying and pasting of the getNextShowing method. Isn't there a better way?

Ah, good eye!

getNextShowing을 movie객체에 복사할 때마다 **중복 코드**가 발생한다.

- "객체 지향"프로그래밍의 목적 중의 하나가 바로 '코드 재사용을 극대화'하는 것이다
- 생성자(constructor)를 사용하는 훨씬 더 좋은 방법이 있다

생성자란 무엇인가?

- 객체를 생성할 수 있고 그것들을 모두 똑같이 만들 수 있는 특수한 함수이다
- 객체에 설정하고 싶은 프로퍼티 값을 가져와서 모든 프로퍼티와 메소드를 가진 새로운 객체를 만들어 준다

Let's create a constructor ...

How to create a constructor

Dogs에 대한 생성자(constructor)를 만들어 보자:

Take the property values as parameters

A constructor function looks a lot like a regular function. But by convention, we give the name of the function a capital letter.

대문자로 시작

The parameters of the constructor take values for the properties we want our object to have.

```
function Dog(name, breed, weight) {
    this.name = name;
    this.breed = breed;
    this.weight = weight;
    this.bark = function() {
        if (this.weight > 25) {
            alert(this.name + " says Woof!");
        } else {
            alert(this.name + " says Yip!");
        }
    };
}
```

The property names and parameter names don't have to be the same, but they often are—again, by convention.

Here, we're initializing the properties of the object to the values that were passed to the constructor.

We can include the bark method in the object we're constructing by initializing the bark property to a function value, just like we've been doing.

We need to use "this.weight" and "this.name" in the method to refer to the properties in the object, just as we have before.

Notice how the syntax differs from object syntax. These are statements, so we need to end each one with a ";" just like we normally do in a function.

Now let's use our constructor

Factory built

Now let's use our constructor

- Can use it to create some dogs
- Puts the keyword new before the call

To create a dog, we use the new keyword with the constructor.

And then call it just like any function.

```
var fido = new Dog("Fido", "Mixed", 38);

var tiny = new Dog("Tiny", "Chawalla", 8);

var clifford = new Dog("Clifford", "Bloodhound", 65);
```

```
fido.bark();

tiny.bark();

clifford.bark();
```

Once we've got the objects, we can call their bark methods to make each Dog bark.

How does this really work?

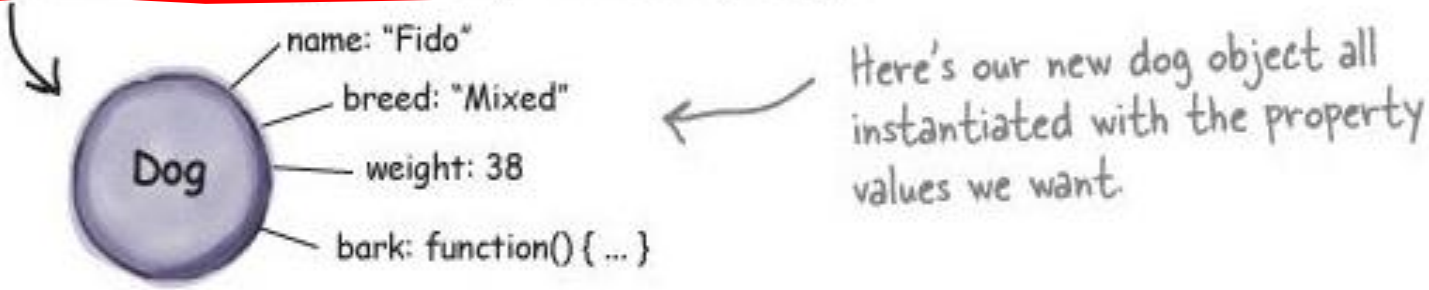
생성자의 코드에 있는 모든 this는 생성자(메소드)를 호출한 객체의 레퍼런스로 해석된다.

따라서 fido.bark를 호출한다면 this는 fido를 참조하게 된다.

How does this know which object it is representing?

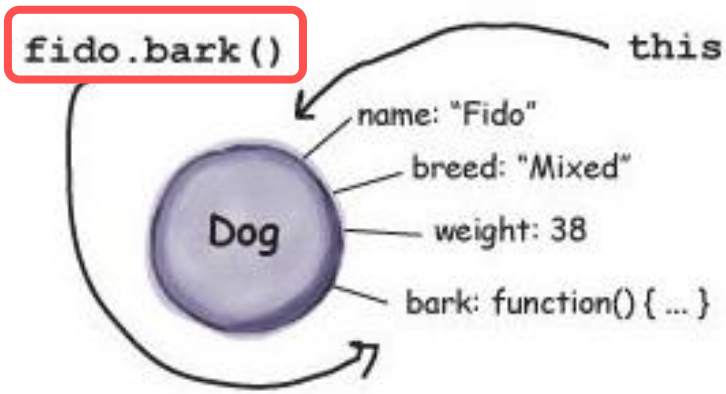
1. **fido에 할당된 dog 객체를 얻었다고 하자**

```
fido = new Dog("Fido", "Mixed", 38);
```



How does this really work?

2. fido 객체의 bark()를 호출한다:

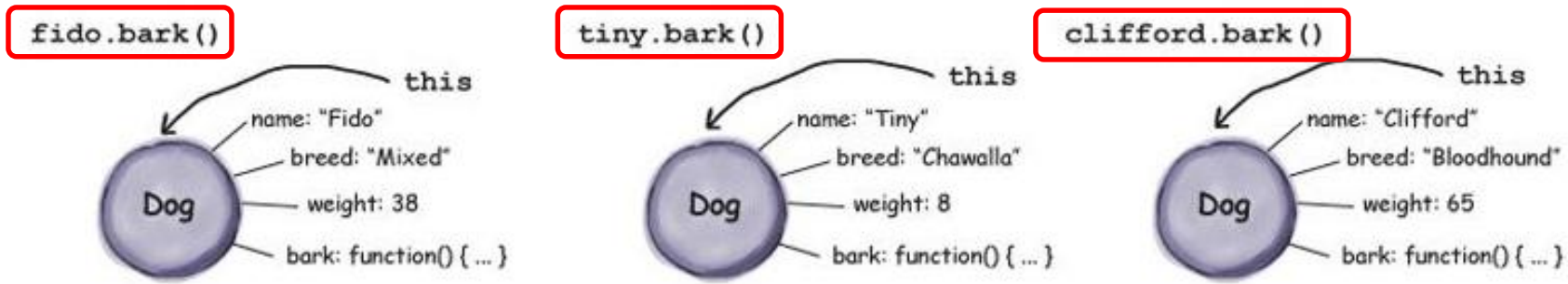


Whenever we invoke a method on an object, JavaScript sets up this to point to the object itself. So here, this points to fido.

And so when we refer to this.name, we know the name is "Fido".

3. 이때 "this"는 항상 메소드가 호출된 객체를 참조한다:

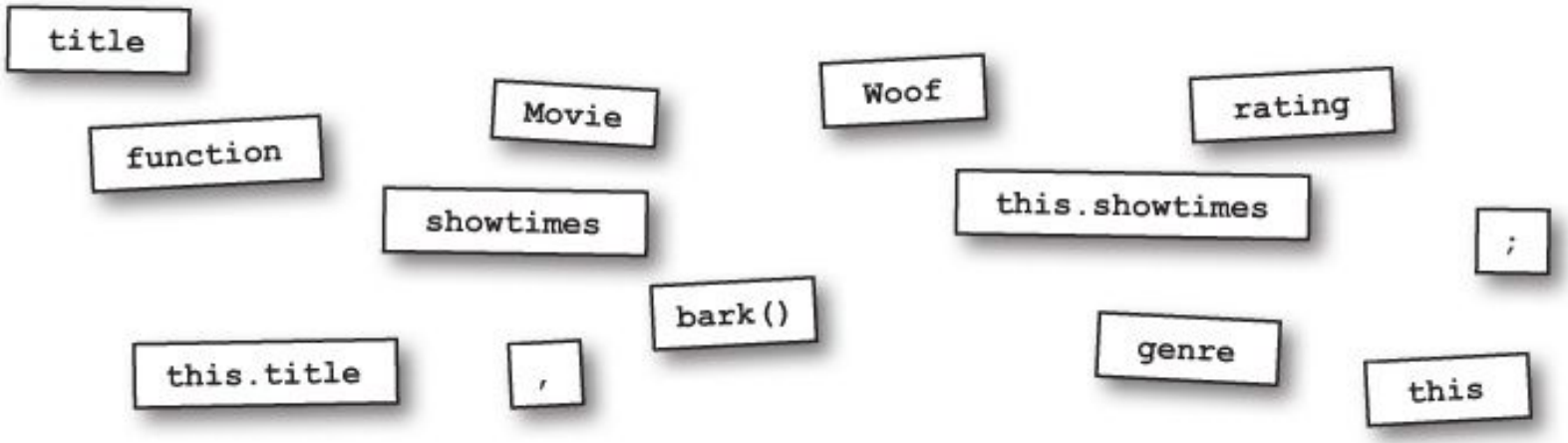
You can call bark on any dog object and this will be assigned to the specific dog before your body code is executed.



Code Magnets

```
function ____________(________, __________, rating, showtimes) {
    this.title = _________;
    this.genre = genre;
    this.________________ = rating;
    this.showtimes = ________________;
    this.getNextShowing = function() {
        var now = new Date().getTime();
        for (var i = 0; i < ____________________.length; i++) {
            var showtime = getTimeFromString(this.__________________[i]);
            if ((showtime - now) > 0) {
                return "Next showing of " + ___________ + " is " + this.showtimes[i];
            }
        }
    } ______
}
```

Use these magnets to complete the code.



실습과제 4-3

Now that you've got a Movie constructor, it's time to make some Movie objects! Go ahead and type in the Movie constructor function and then add the code below and take your constructor for a spin.

```
var banzaiMovie = new Movie("Buckaroo Banzai",
                            "Cult Classic",
                            5,
                            ["1:00pm", "5:00pm", "7:00pm", "11:00pm"]);
```

First we'll create a movie object for the movie Buckaroo Banzai (one of our cult classic favorites). We pass in the values for the parameters.

Notice we can put the array value for showtimes right in the function call.

```
var plan9Movie = new Movie("Plan 9 from Outer Space",
                           "Cult Classic",
                           2,
                           ["3:00pm", "7:00pm", "11:00pm"]);
```

And next, Plan 9 from Outer Space...

```
var forbiddenPlanetMovie = new Movie("Forbidden Planet",
                                     "Classic Sci-fi",
                                     5,
                                     ["5:00pm", "9:00pm"]);
```

And of course, Forbidden Planet.

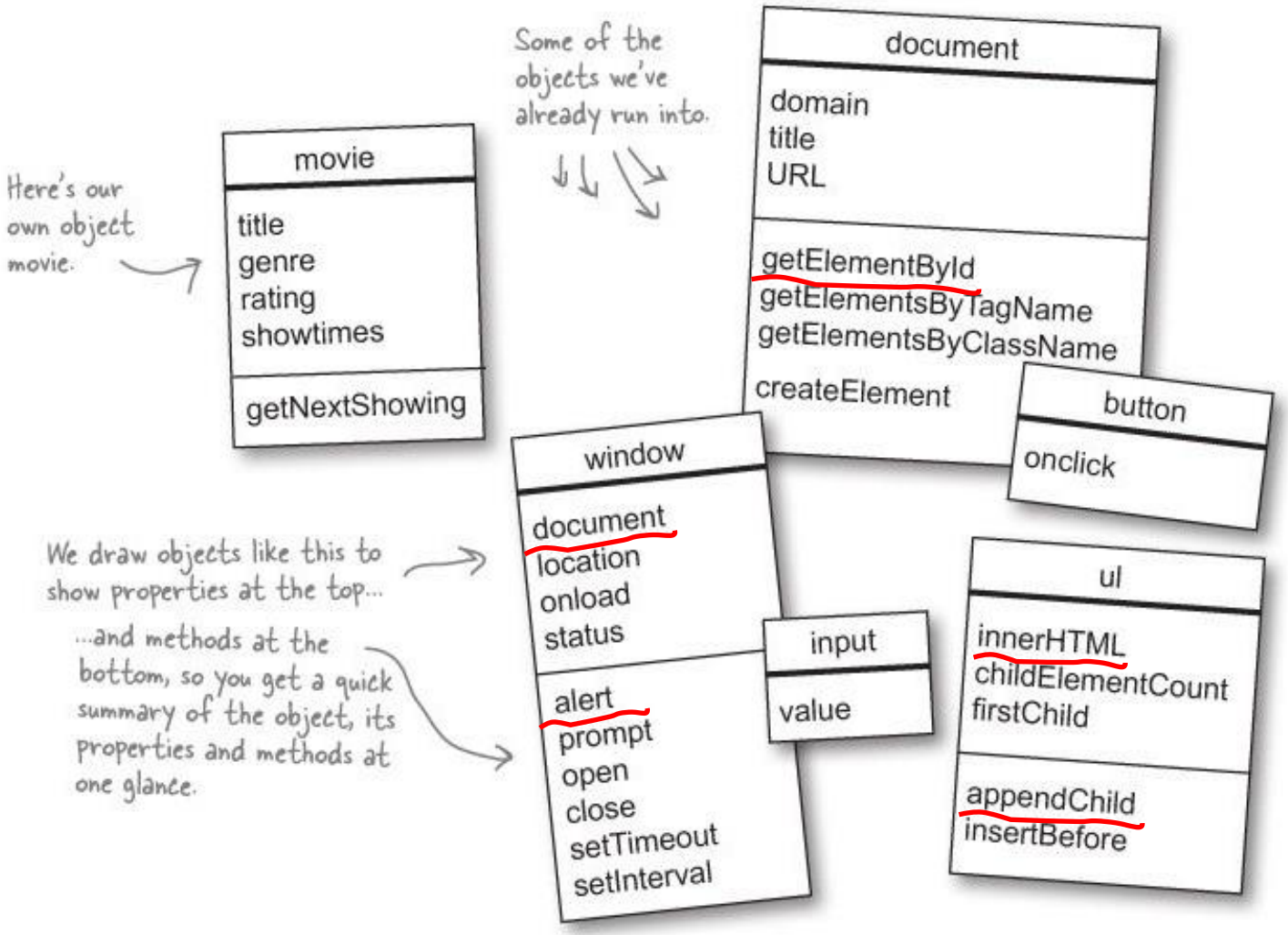
```
alert(banzaiMovie.getNextShowing());
alert(plan9Movie.getNextShowing());
alert(forbiddenPlanetMovie.getNextShowing());
```

Once we've got all our objects created, we can call the getNextShowing method and alert the user for the next showing times.

Congrats, you've made it through functions and objects! Now that you know all about them, and before we end the chapter, let's take a few moments to check out JavaScript objects in the wild; that is, in their native habitat, the browser!

Now, you might have started to notice...

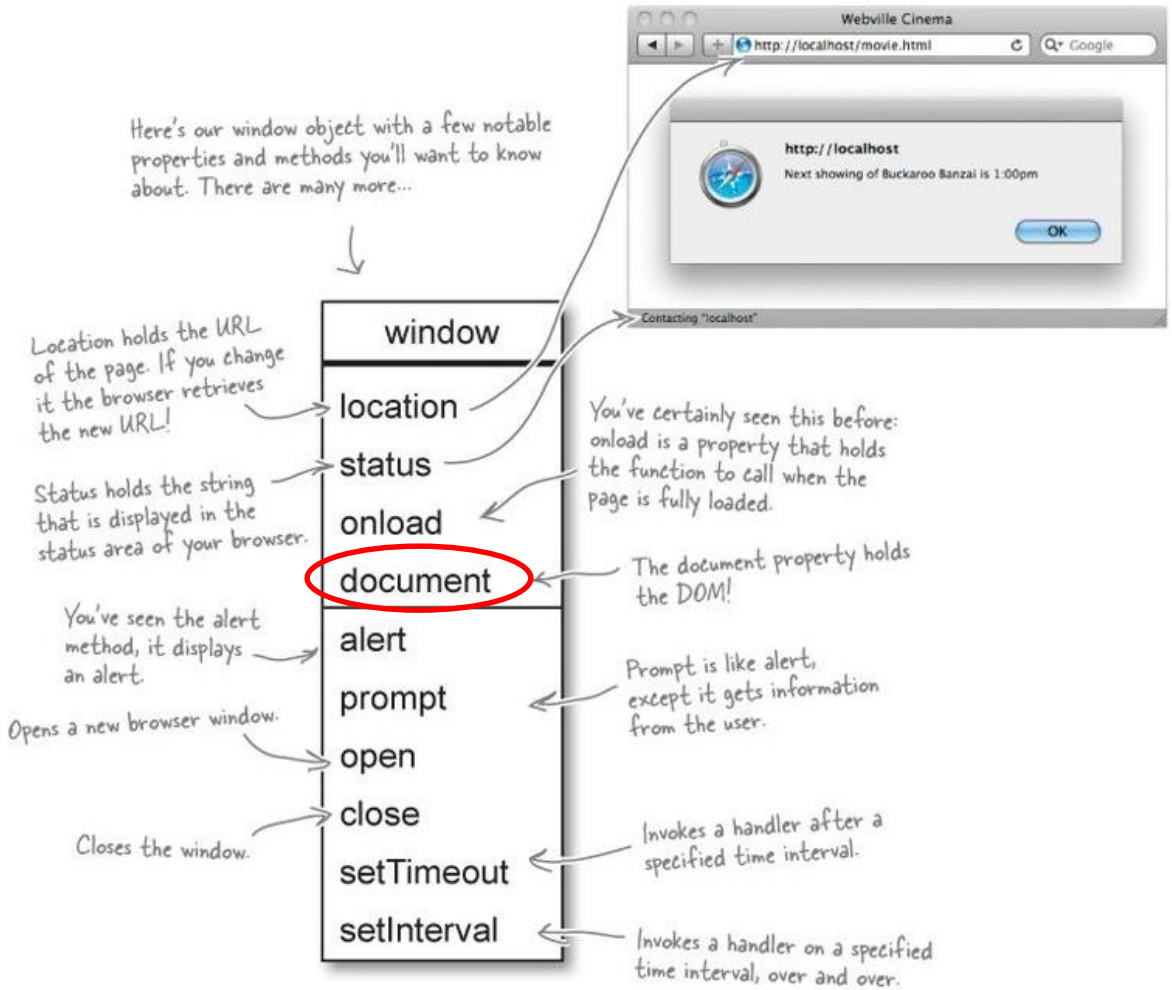
... 객체들이 주변에 널려 있다. 예를 들어, document.getElementById로 부터 돌려 받은 엘리먼트들과 마찬가지로 document와 window도 객체들이다.



What is the window object anyway?

window 객체는 자바스크립트 프로그램을 위한 글로벌 환경(global environment)과 애플리케이션의 메인 윈도우(main window)를 제공해준다

- ✓ 많은 코어 프로퍼티와 메소드를 포함하고 있다



Window is the global object.

- Window 객체는 글로벌 환경으로서 동작한다
- 또한 사용자가 정의하는 모든 글로벌 변수는 결국 window namespace에 놓여지게 되므로 window.myvariable처럼 참조(reference)할 수 있다

A closer look at window.onload

window.onload event handler

By assigning a function to the **window.onload** property, we can ensure our code isn't run until the page is loaded and the DOM is completely set up.

Here's our global window object.

onload is a property of the window object.

This is an anonymous function, which is assigned to the onload property.

```
window.onload = function() {
    // code here
};
```

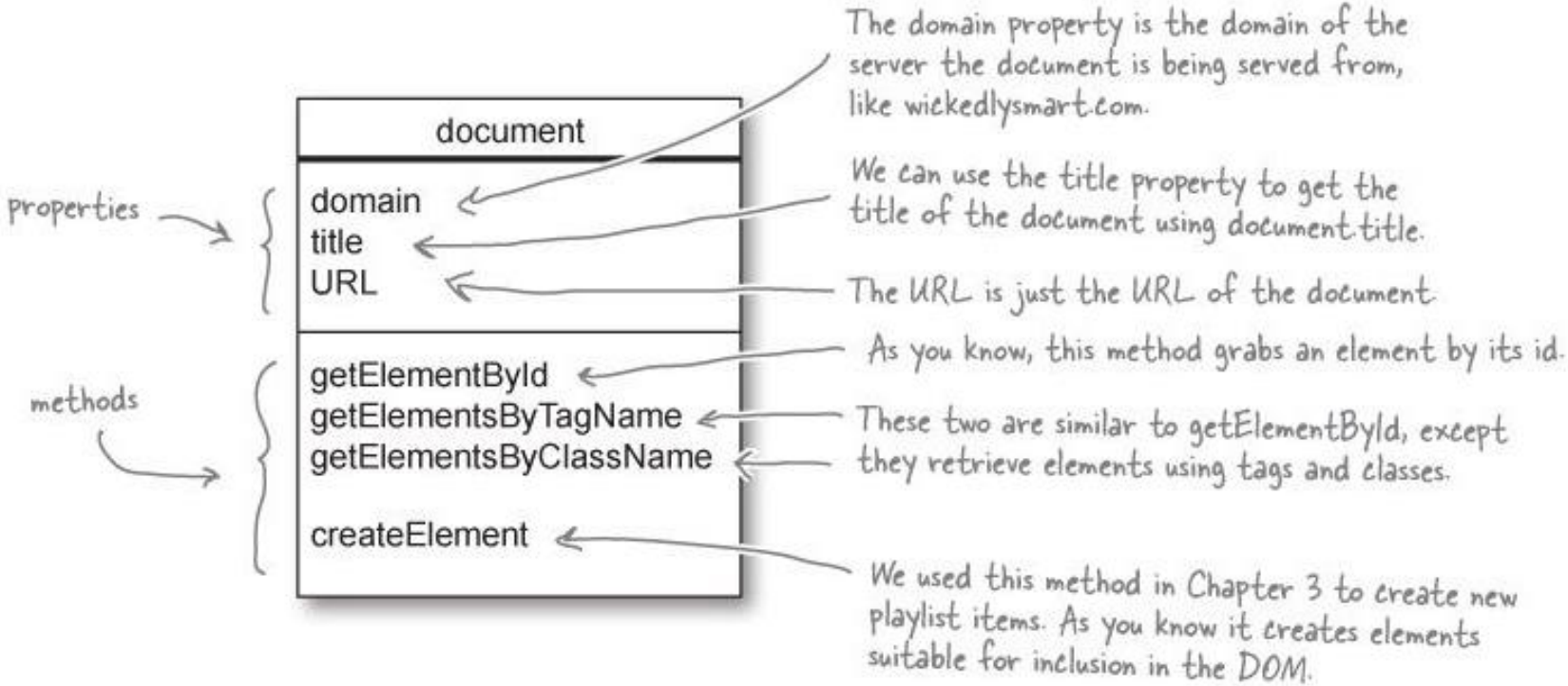
And of course the body of the function is executed once the window fully loads the page and invokes our anonymous function!

Another look at the document object

Document 객체는 DOM을 접근(access)하는데 사용된다.

앞에서 보았듯이 사실 document 객체는 window 객체의 프로퍼티이다.

물론 window.document처럼 사용하지는 않는다. (그럴 필요가 없기 때문에)



A closer look at document.getElementById

We promised in the beginning of this chapter that you'd understand document.getElementById by the end of the chapter.
Well, you made it through functions, objects, and methods, and now you're ready!
Check it out:

document is the document object, a built-in JavaScript object that gives you access to the DOM.

```
var div = document.getElementById("myDiv");
```

getElementById is a method that...

... takes one argument, the id of a element, and returns an element object.

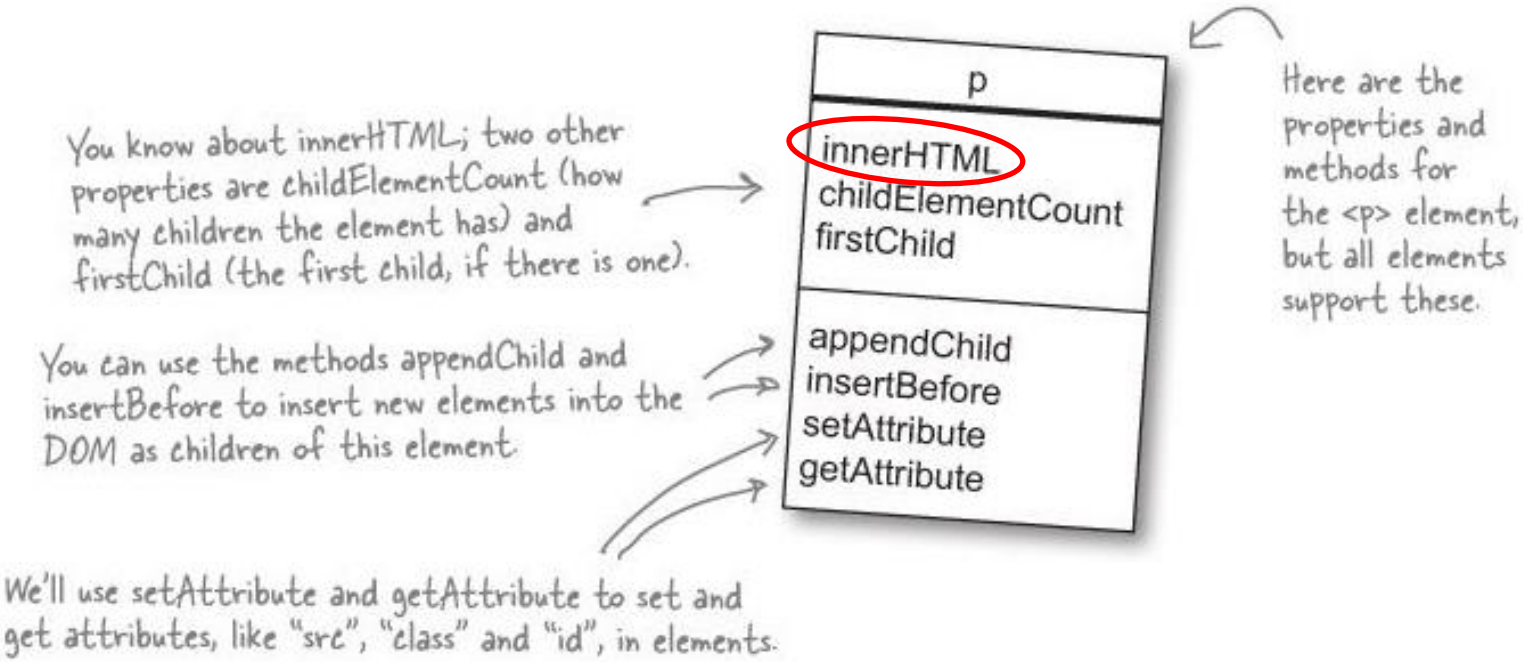
What was a confusing looking string of syntax now has a lot more meaning, right?
Now, that div variable is also an object: an element object.

Let's take a closer look at that too.

Think about: your element objects

getElementById 에 의해 반환되는 엘리먼트 또한 객체이다.

이미 앞에서 innerHTML과 같은 엘리먼트 프로퍼티를 본 적이 있다.



Q & A