# Computer Science Principles

## Web Programming

## JavaScript Programming Essentials

CHAPTER 4: OBJECTS

DR. ERIC CHOU

IEEE SENIOR MEMBER

# Objects

- Objects in JavaScript are very similar to arrays, but objects use strings instead of numbers to access the different elements. The strings are called **keys** or ***properties***, and the elements they point to are called **values**.
- Together these pieces of information are called **key-value pairs**. While arrays are mostly used to represent lists of multiple things, objects are often used to represent single things with multiple characteristics, or **attributes**.
- For example, in Chapter 3 we made several arrays that listed different animal names. But what if we wanted to store different pieces of information about one animal?

# Overview

LECTURE 1

# Object Introduction

- There are **no classes** in JavaScript
- Instead, properties can be created and deleted dynamically

```
var o1 = new Object();
o1.testing = "This is a test";
delete o1.testing;
```

**Create an object `o1`**
**Create property `testing`**
**Delete `testing` property**

# Object Creation

- Objects are created using new expression

```
new Object()
```

*Constructor* and argument list

- A constructor is a function
  - When called via new expression, a new empty Object is created and passed to the constructor along with the argument values
  - Constructor performs initialization on object
    - Can add **properties** and **methods** to object
    - Can add object to an **inheritance hierarchy**

# Object Creation

- The **Object()** built-in constructor
  - Does not add any properties or methods directly to the object
  - Adds object to hierarchy that defines default toString() and valueOf() methods (used for conversions to String and Number, resp.)

# Property Creation

- **Assignment** to a non-existent (even if inherited) property name creates the property:

```
o1.testing = "This is a test";
```

- **Object initializer** notation can be used to create an object (using Object() constructor) and one or more properties in a single statement:

```
var o2 = { p1:5+9, p2:null, testing:"This is a test" };
```

# Enumerating Properties

## key:value pair

- Special form of `for` statement used to **iterate through all properties** of an object:

```
var hash = new Object();
hash.kim = "85";
hash.sam = "92";
hash.lynn = "78";
for (var aName in hash) {
  window.alert(aName + " is a property of hash.");
}
```

Produces three alert boxes; order of names is implementation-dependent.

# Accessing Property Values

- The JavaScript object dot notation is actually shorthand for a more general **associative array** notation in which Strings are array indices:

```
hash.kim ⟶ hash["kim"]
```

- Expressions can supply property names:

```
window.alert(aName + " scored " + hash[aName]);
```

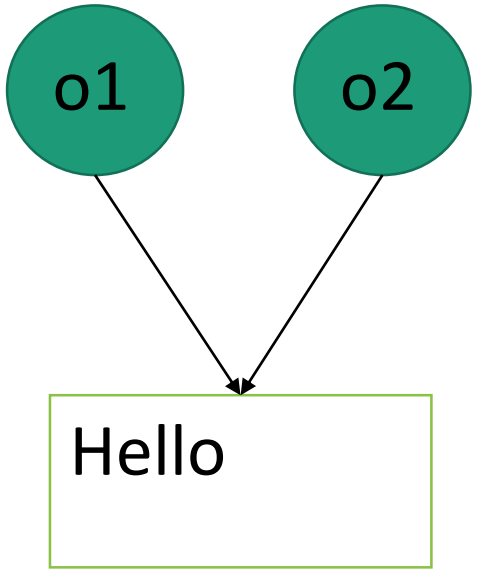
Converted to String
if necessary

# Object Values

- Value of Object is reference to object:

o2 is another name for o1

```
var o1 = new Object();
o1.data = "Hello";
var o2 = o1;
o2.data += " World!";
window.alert(o1.data);
```

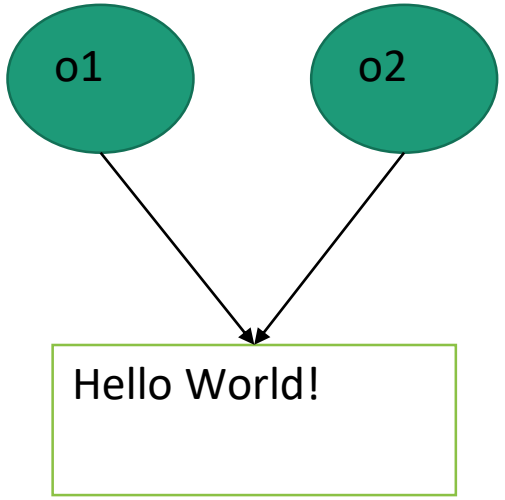


# Object Values

- Value of Object is reference to object:

o1 is changed

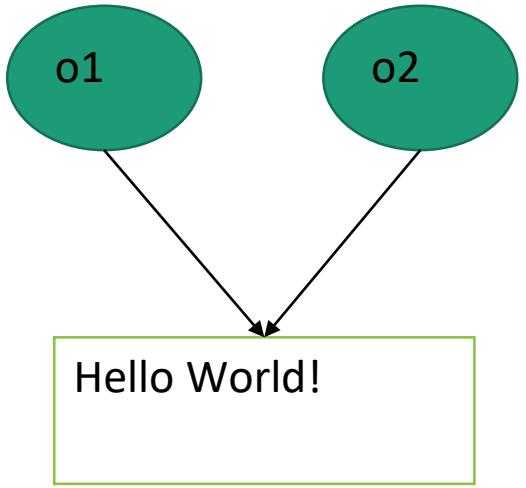
```
var o1 = new Object();
o1.data = "Hello";
var o2 = o1;
o2.data += " World!";
window.alert(o1.data);
```



# Object Values

- Value of Object is reference to object:

```
var o1 = new Object();
o1.data = "Hello";
var o2 = o1;
o2.data += " World!";
window.alert(o1.data);
```




Output is `Hello World!`

# Object Methods

- JavaScript **functions are stored as values** of type Object
- A function declaration creates a function value and stores it in a variable (property of `window`) having the same name as the function
- A **method** is an object property for which the value is a function

# Object Methods

Creates global variable named `leaf` with function value

```
function leaf() {
  return this.left == null && this.right == null;
}

function makeBTNode(value) {
  var node = new Object();
  node.left = node.right = null;
  node.value = value;
  node.isLeaf = leaf;
  return node;
}
```

# Object Methods

```
function leaf() {
  return this.left == null && this.right == null;
}

function makeBTNode(value) {
  var node = new Object();
  node.left = node.right = null;
  node.value = value;
  node.isLeaf = leaf;
  return node;
}
```

# Object Literals

LECTURE 2

# JavaScript: Object literals

## Description

- An object literal is zero or more pairs of comma-separated list of property names and associated values, enclosed by a pair of curly braces.
- In JavaScript an object literal is declared as follows:

1. **An object literal without properties:**

```
var userObject = {}
```

2. **An object literal with a few properties :**

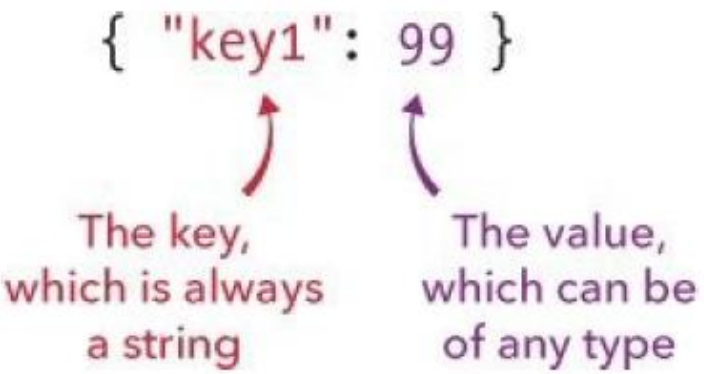
```
var student = {
   First-name : "Suresy",
   Last-name : "Rayy",
   Roll-No : 12
};
```

# JavaScript: Object literals

## Description

3. **An object literal with properties without double quote:**

```
var cat = {
    "legs": 3,
    "name": "Harmony",
    "color": "Tortoiseshell"
};
```



# JavaScript: Object literals

## Syntax

- Object literals maintain the following syntax rules:
  1. There is a colon (:) between property name and value.
  2. A comma separates each property name/value from the next.
  3. There will be no comma after the last property name/value pair.

# Constructor

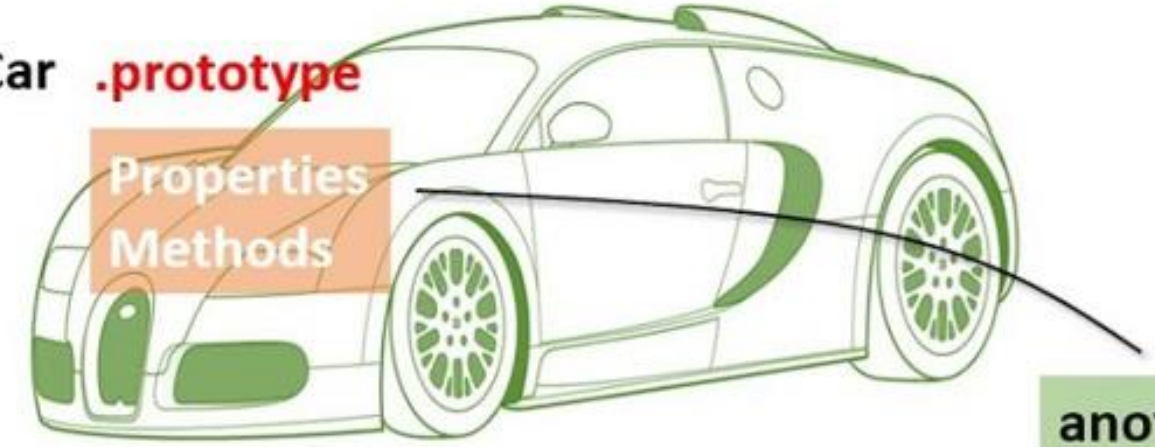
LECTURE 3

# Class

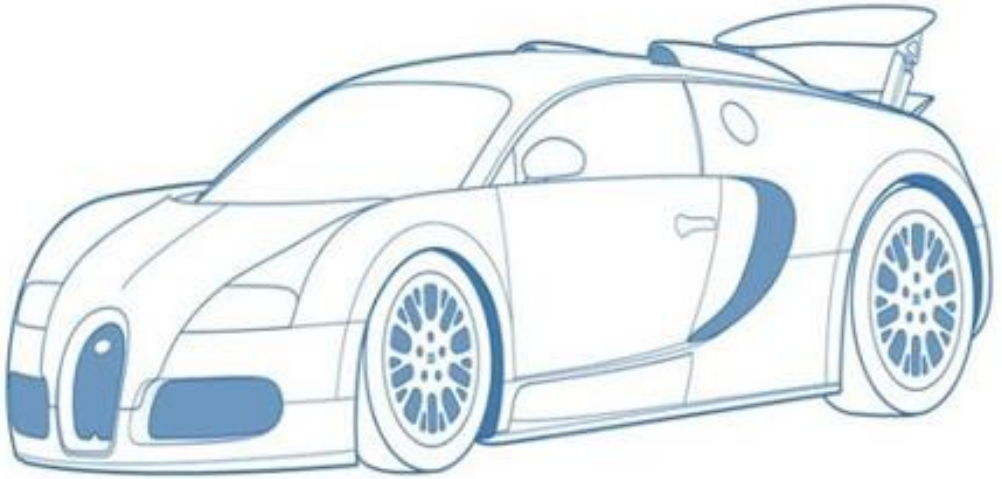
## JavaScript has no Class

- JavaScript has no class.
- JavaScript function has a property called **prototype**. A JavaScript function can be used as a Class. And those data fields added to prototype will become data fields for objects created with **new** operator and the function. The function works like a **constructor**.

# OBJECT Prototypes




anotherCar = new Car()



# Prototype in JavaScript

- JavaScript is a dynamic language. You can attach new properties to an object at any time as shown below.

Example: Attach property to object

```
function Student() {
                this.name = 'John';
                this.gender = 'Male';
}

var studObj1 = new Student();
studObj1.age = 15;
alert(studObj1.age); // 15

var studObj2 = new Student();
alert(studObj2.age); // undefined
```

# Prototype

## Every function includes prototype object by default.

- As you can see in the above example, age property is attached to studObj1 instance. However, studObj2 instance will not have age property because it is defined only on studObj1 instance.
- So what to do if we want to add new properties at later stage to a function which will be shared across all the instances?
- The answer is **Prototype**.
- The prototype is an object that is associated with every functions and objects by default in JavaScript, where function's prototype property is accessible and modifiable and object's prototype property (aka attribute) is not visible.

# Data Prototype

## Demo Program: Student.html

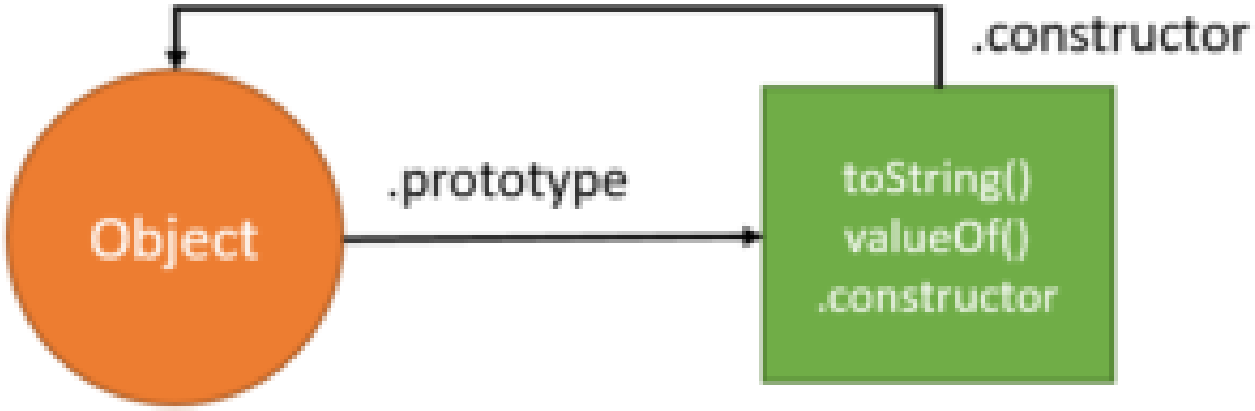
```
<html>
    <body onload="main()">
        <script>
            function Student() {   // constructor for Student Class
                this.name = 'John';
                this.gender = 'Male';
            }
            function main(){
             var studObj1 = new Student();
             studObj1.age = 15;
             alert(studObj1.age); // 15
             var studObj2 = new Student();
             alert(studObj2.age); // undefined
            }
        </script>
    </body>
</html>
```

- Function can be used to create objects.
- **this** is used as the reference to the object to be created.
- **new** operator is used to call a function as constructor.

# Data Prototype

## Demo Program: Student2.html

# Go Brackets!!!!



```html
<html>
    <body onload="main()">
        <script>
            function Student() {   // constructor for Student Class
            }

            function main(){
               Student.prototype.name = "John";
               Student.prototype.gender ="male";
               var studObj1 = new Student();
               studObj1.age = 15;
               alert(studObj1.age); // 15
               alert(studObj1.name); // John

               var studObj2 = new Student();
               alert(studObj2.age); // undefined
               alert(studObj2.name); //John
               Student.prototype.name = "Tom";
               alert(studObj1.name);
               alert(studObj2.name);
               studObj2.name = "Mary";
               alert(studObj1.name);
               alert(studObj2.name);
            }
        </script>
    </body>
</html>
```

Student Objects has no properties at beginning.

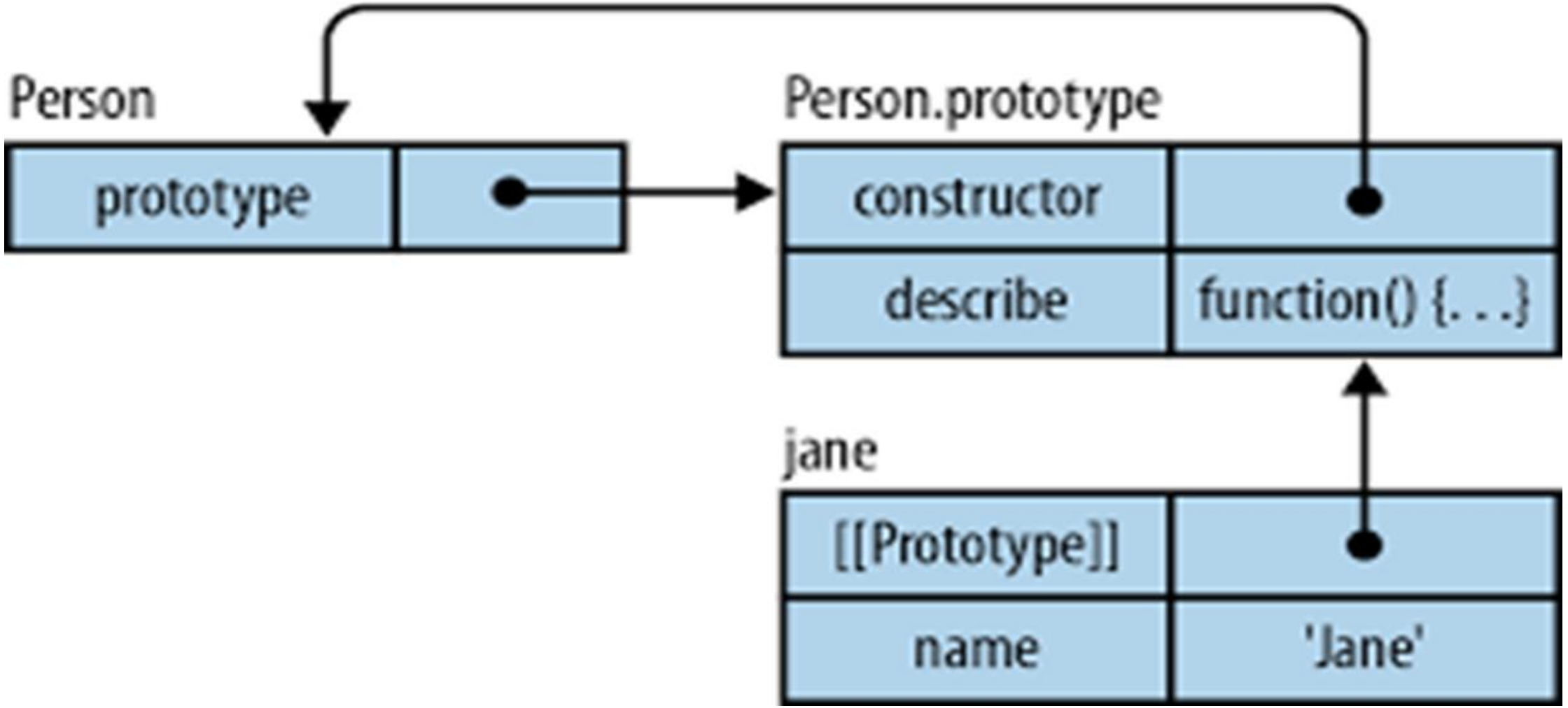
Two properties created.

Student Object studObj1 created
And, added with an age property.

Student Object studObj2 created
And, added **without** an age property.

Default value for name property
is updated (like class variable).

Instance variable for studObj2 updated.
(studObj1.name  remain as Tom).


Person
prototype
Person.prototype
constructor
describe
function() {. . .}
jane
[[Prototype]]
name
'Jane'

# Student Prototype

LECTURE 4

# Student Prototype (Constructor using this)

## Demo Program: Student4.html

# Go Brackets!!!

```html
<html>
<body onload="main()">
<script>
function studentToString(){
    var str = "";
    str += "name: " + this.name +"<br>";
    str += "age: " + this.age +"<br>";
    str += "address: " + this.address + "<br>";
    return str;
}

// Student Prototype
function Student(n, ag, add){
    this.name = n;
    this.age  = ag;
    this.address  = add;
    this.toString = studentToString;
}
//Student.prototype.toString = studentToString(); // method definition


function main(){
   var s1 = new Student("Eric", 20, "1 A Street, Los Angeles, CS 90007");
   var s2 = new Student("Tom", 18, "2 A Street, Los Angeles, CS 90007");
   document.write("<h3>Student Records: </h3>");
   //alert(s1.name);
   //alert(s1.toString());
   document.write(s1.toString());
   document.write(s2.toString());
}
</script>
</body>
</html>
```

**Dynamic Binding of Objects for this reference variable. This means this prototype**

**Function Reference added as instance method**

**Student Records:**

name: Eric
age: 20
address: 1 A Street, Los Angeles, CS 90007
name: Tom
age: 18
address: 2 A Street, Los Angeles, CS 90007

# Student Prototype (Object as base object)

## Demo Program: Student5.html

Go Brackets!!!

```html
<html>
<body onload="main()">
<script>
    // Student Prototype
function Student(n, ag, add){
    var obj = new Object();
    obj.name = n;
    obj.age  = ag;
    obj.address  = add;
    obj.toString = function (){
        var str = "";
        str += "name: " + this.name +"<br>";
        str += "age: " + this.age +"<br>";
        str += "address: " + this.address + "<br>";
        return str;
    }
    return obj;
}

function main(){
   var s1 = new Student("Eric", 20, "1 A Street, Los Angeles, CS 90007");
   var s2 = new Student("Tom", 18, "2 A Street, Los Angeles, CS 90007");
   document.write("<h3>Student Records: </h3>");
   //alert(s1.name);
   //alert(s1.toString());
   document.write(s1.toString());
   document.write(s2.toString());
}
</script>
</body>
</html>
```

Use Object() as base Object and add all different properties to it.

Inline anonymous function as a method to the object

A object reference is returned.

**Student Records:**

name: Eric
age: 20
address: 1 A Street, Los Angeles, CS 90007
name: Tom
age: 18
address: 2 A Street, Los Angeles, CS 90007

# Keys Without Quotes

- In our first object, we put each key in quotation marks, but you don't necessarily need quotes around the keys — this is a valid cat object literal as well:

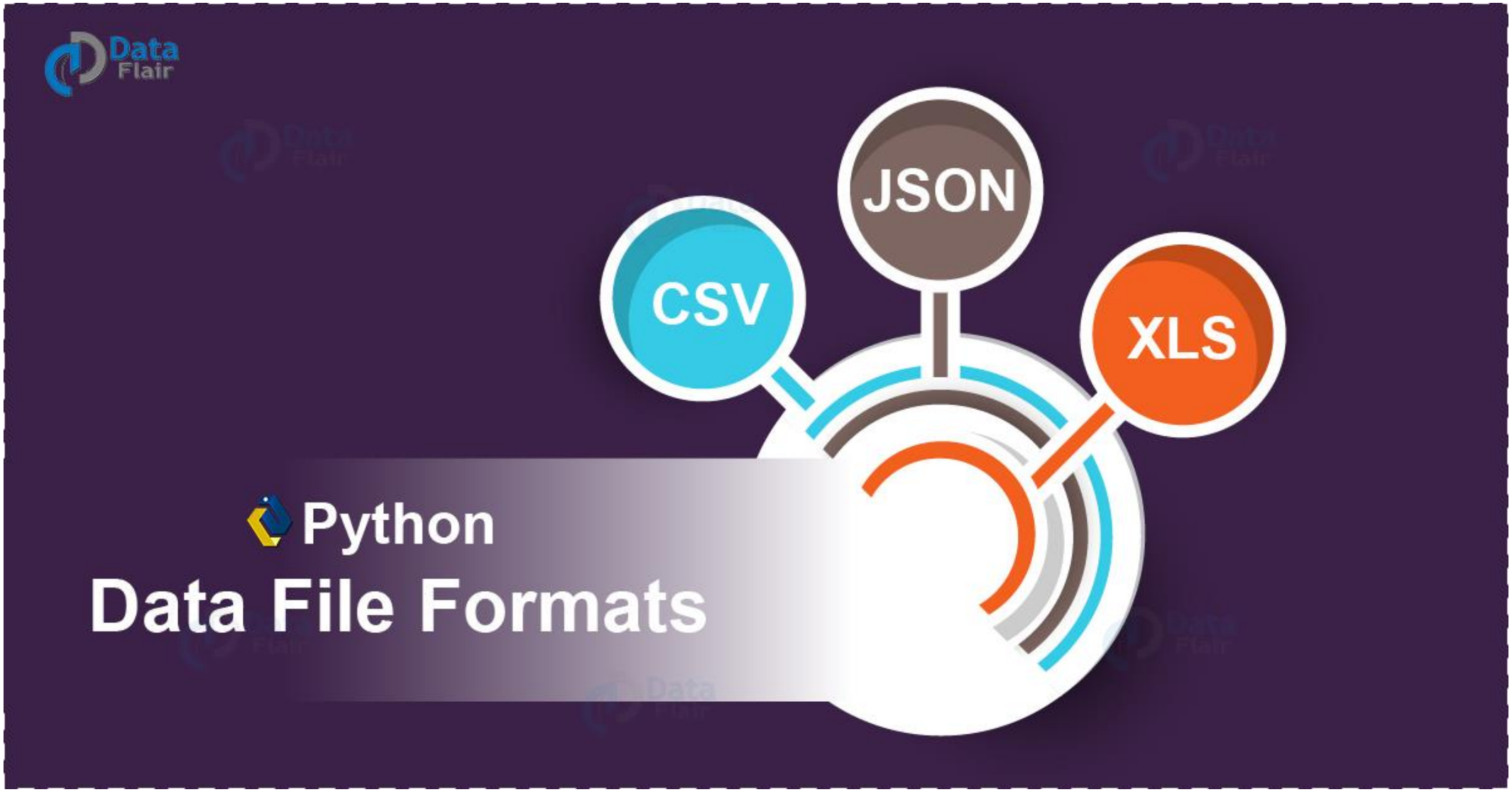
```
var cat = {
  legs: 3,
  name: "Harmony",
  color: "Tortoiseshell"
};
```

- JavaScript knows that the keys will always be strings, which is why you can leave out the quotes. If you don't put quotes around the keys, the **unquoted** keys have to follow the same rules as variable names: spaces aren't allowed in an unquoted key, for example. If you put the key in quotes, then spaces are allowed:

```
var cat = {
  legs: 3,
  "full name": "Harmony Philomena Snuggly-Pants Morgan",
  color: "Tortoiseshell"
};
```

# JavaScript Array and Object

LECTURE 5


Data Flair
JSON
CSV
XLS
Python
Data File Formats

| JavaScript Object Notation (JSON) | Python |
| --- | --- |
| Object | dict |
| Array | list |
| String | unicode |
| Number (int) | int, long |
| Number (float) | float |
| true | True |
| false | False |
| null | None |

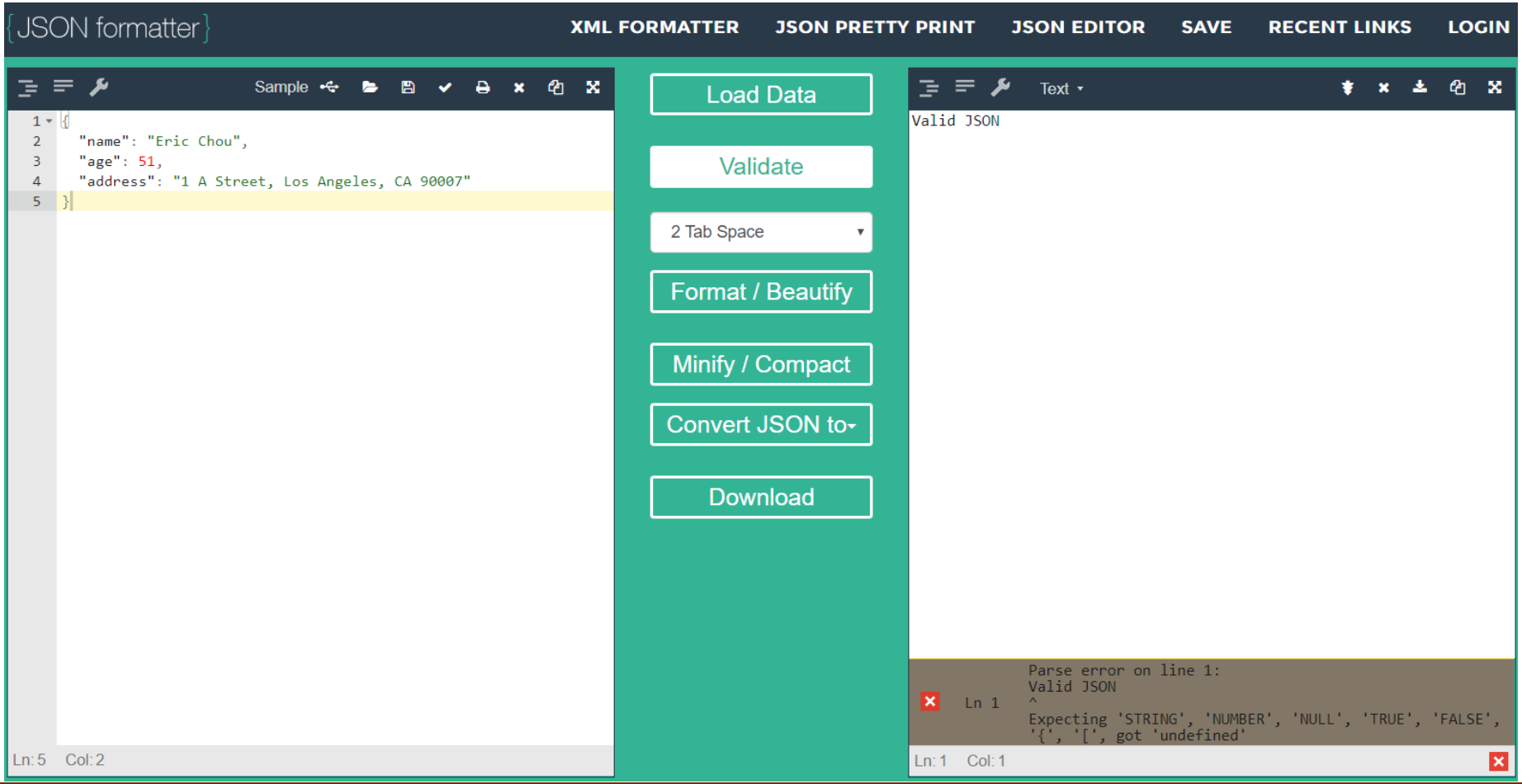
# JSON Online Editor

https://jsonformatter.org

- JavaScript array and object can all be created using on-line editor (JSON editor).
- We use this JSON Formatter as our exemplary software to create JSON objects and arrays.

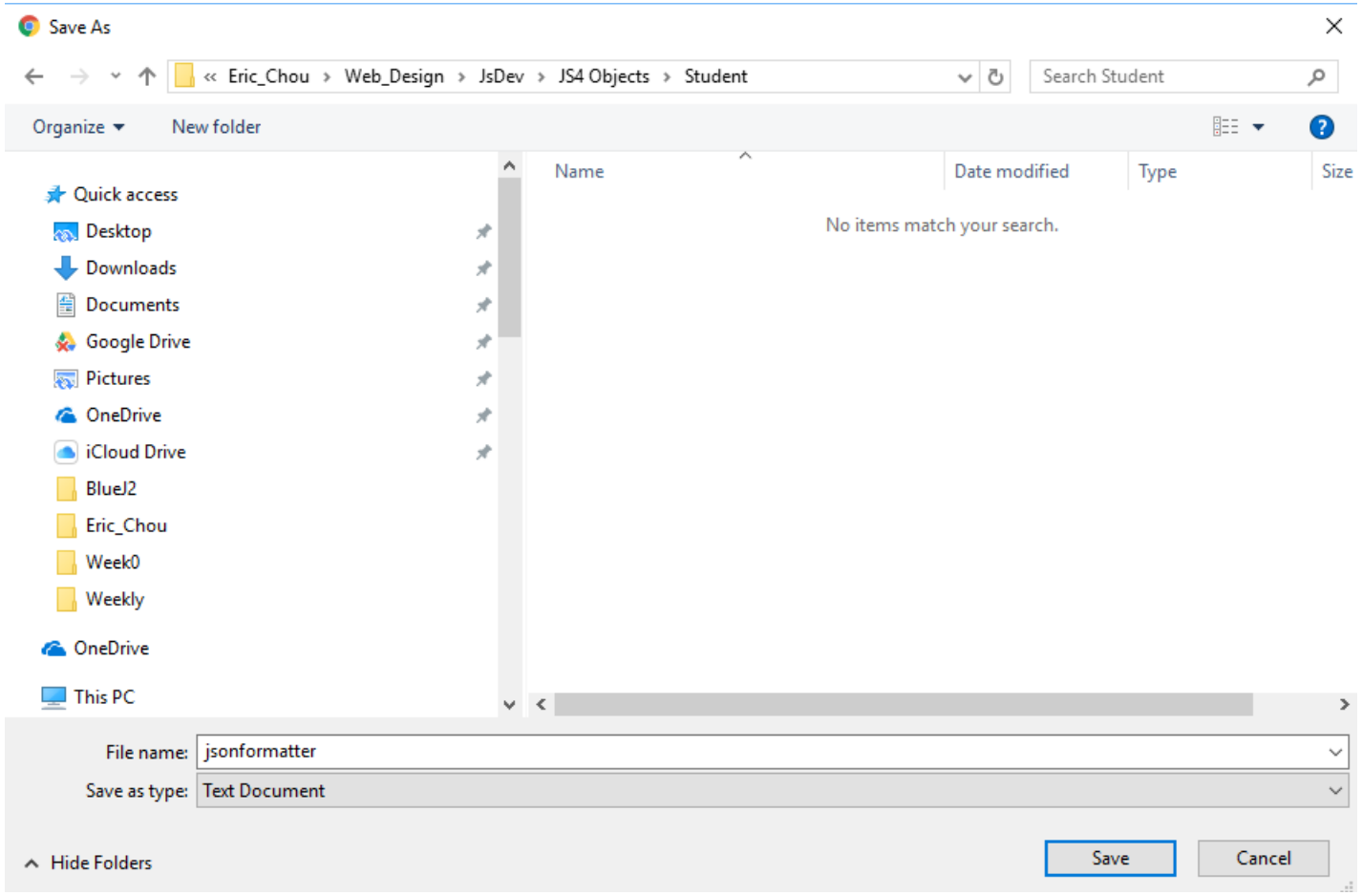
# Use JSON Editor to Create a Student Record

## Step 1: Create an Object



# Use JSON Editor to Create a Student Record

## Step 2: save to a file.



Student6.html

```
<html>
<body onload="main()">
<script>
function studentToString(){
        var str = "";
        str += "name: " + this.name +"<br>";
        str += "age: " + this.age +"<br>";
        str += "address: " + this.address + "<br>";
        return str;
}
function main(){
    var s1 = {
                "name": "John",
                "age": 51,
                "address": "1 A Street, Los Angeles, CA 90007",
                "toString" : studentToString
            }
    var s2 = {
                "name": "Tom White",
                "age": 18,
                "address": "2 A Street, Los Angeles, CA 90007",
                "toString" : studentToString
            }
    document.write("<h3>Student Records: </h3>");
    document.write(s1.toString());
    document.write(s2.toString());
}
</script>
</body>
</html>
```

# Step 3: use it in html file

JSON Objects properties with double quote. This will be compatible with Python dictionary as well.

JSON Objects are object only. No prototype.

Methods added later.

**Student Records:**

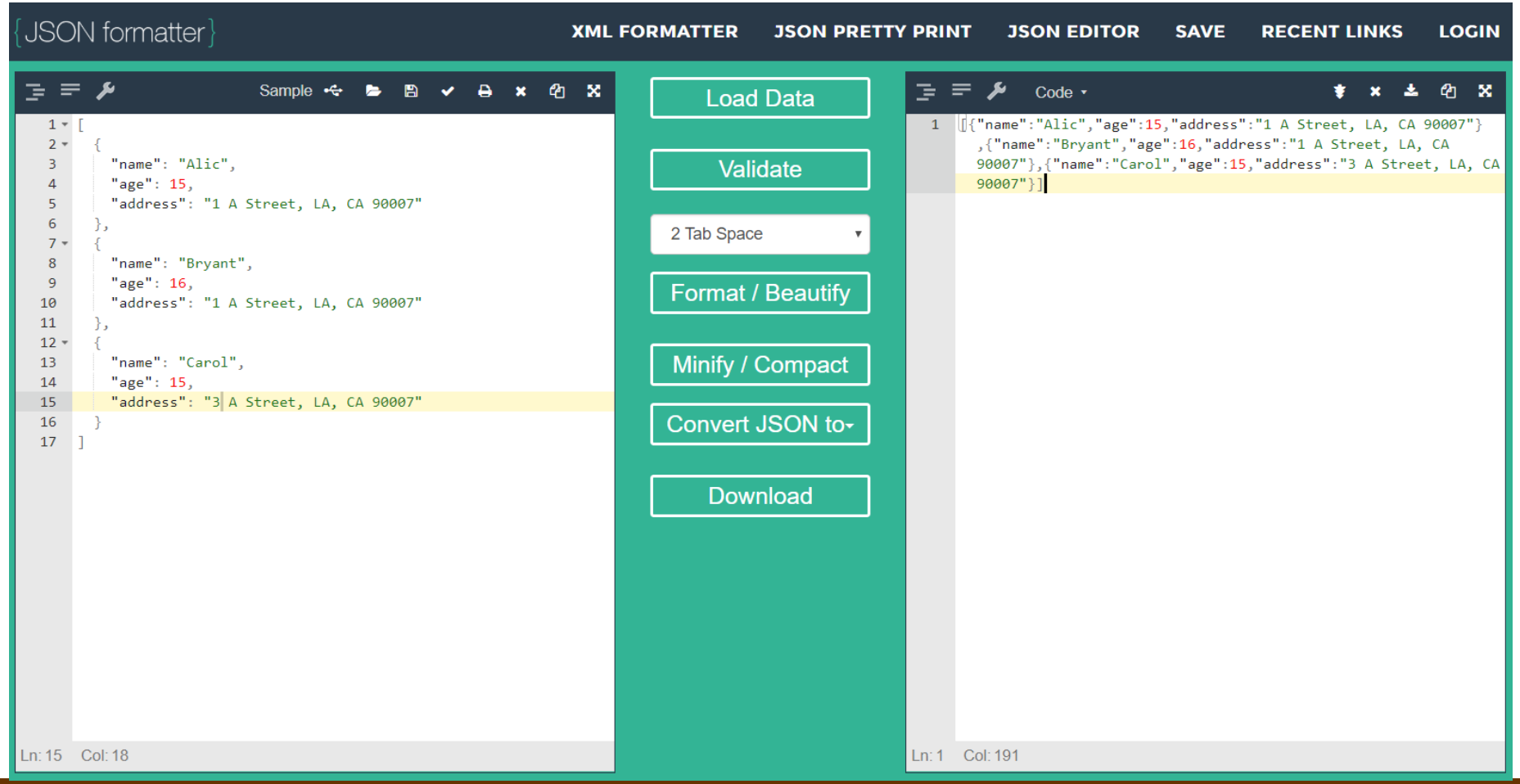
name: John
age: 51
address: 1 A Street, Los Angeles, CA 90007
name: Tom White
age: 18
address: 2 A Street, Los Angeles, CA 90007

# Arrays

- Array objects are represented by comma-separated value list enclosed with brackets:  [1, 2 ,3, 4, 5]
- Array is also valid in JSON editor.
- JSON format is always recommended because it is the new de facto standard in internet industry now.
- JSON editor provides minify function to make the JSON data smaller but less readable.  You may restore it by beautifying it.

# Array of Objects

## Minified Array of Objects



# Array of Objects

## method added

```
[
  {
    "name": "Alic",
    "age": 15,
    "address": "1 A Street, LA, CA 90007",
    "toString": studentToString
  },
  {
    "name": "Bryant",
    "age": 16,
    "address": "2 A Street, LA, CA 90007",
    "toString": studentToString
  },
  {
    "name": "Carol",
    "age": 15,
    "address": "3 A Street, LA, CA 90007",
    "toString": studentToString
  }
]
```

```
<html>
<body onload="main()">
<script>
function studentToString(){
        var str = "";
        str += "name: " + this.name +"<br>";
        str += "age: " + this.age +"<br>";
        str += "address: " + this.address + "<br>";
        return str;
}
function main(){
   var student_list = [
       {
        "name": "Alic",
        "age": 15,
        "address": "1 A Street, LA, CA 90007",
        "toString": studentToString
       },
       {
        "name": "Bryant",
        "age": 16,
        "address": "2 A Street, LA, CA 90007",
        "toString": studentToString
       },
       {
        "name": "Carol",
        "age": 15,
        "address": "3 A Street, LA, CA 90007",
        "toString": studentToString
       }
    ]
   document.write("<h3>Student Records: </h3>");
   for (var i in student_list) document.write(student_list[i].toString());
}
</script>
</body>
</html>
```

For each key value, array's key is its index

**Student Records:**

name: Alic
age: 15
address: 1 A Street, LA, CA 90007
name: Bryant
age: 16
address: 2 A Street, LA, CA 90007
name: Carol
age: 15
address: 3 A Street, LA, CA 90007

# Accessing An Object and Its Data Fields

LECTURE 6

# Accessing Values in Objects

## Indexing []

- You can access values in objects using square brackets, just like with arrays. The only difference is that instead of the index (a number), you use the key (a string).

```
> cat["name"];
```

"**Harmony**“

# Accessing Values in Objects

## dot notation (domain specifier/data fields accessor)

- Just as the quotes around keys are optional when you create an object literal, the quotes are also optional when you are accessing keys in objects. If you're not going to use quotes, however, the code looks a bit different:

```
> cat.name;

"Harmony"
```

- This style is called *dot notation*. Instead of typing the key name in quotes inside square brackets after the object name, we just use a period, followed by the key, without any quotes. As with unquoted keys in object literals, this will work only if the key doesn't contain any special characters, such as spaces.

# Get Key List

## Object.keys()

- Instead of looking up a value by typing its key, say you wanted to get a list of all the keys in an object.
- JavaScript gives you an easy way to do that, using Object.keys():

```
var dog = { name: "Pancake", age: 6, color: "white", bark: "Yip yap yip!" };
var cat = { name: "Harmony", age: 8, color: "tortoiseshell" };
> Object.keys(dog);
["name", "age", "color", "bark"]
> Object.keys(cat);
["name", "age", "color"]
```

- **Object.keys(*anyObject*)** returns an array containing all the keys of *anyObject*. .keys() is a prototype function.

# Adding Values to Objects

- An empty object is just like an empty array, but it uses curly brackets, { }, instead of square brackets:

```
var object = {};
```

- You can add items to an object just as you'd add items to an array, but you use strings instead of numbers:

```
var cat = {};
cat["legs"] = 3;
cat["name"] = "Harmony";
cat["color"] = "Tortoiseshell";
> cat;
{ color: "Tortoiseshell", legs: 3, name: "Harmony" }
```

# Adding Keys with Dot Notation

- You can also use dot notation when adding new keys. Let's try the previous example, where we started with an empty object and added keys to it, but this time we'll use dot notation:

```
var cat = {};
cat.legs = 3;
cat.name = "Harmony";
cat.color = "Tortoiseshell";
```

- If you ask for a property that JavaScript doesn't know about, it returns the special value undefined. undefined just means "There's nothing here!" For example:

```
var dog = {
   name: "Pancake",
   legs: 4,
   isAwesome: true
};
> dog.isBrown;
undefined
```

# Combining Arrays and Objects

## Array of Objects

- For example, an array of dinosaur objects might look like this:

```
var dinosaurs = [
{ name: "Tyrannosaurus Rex", period: "Late Cretaceous" },
{ name: "Stegosaurus", period: "Late Jurassic" },
{ name: "Plateosaurus", period: "Triassic" }
];
```

# Combining Arrays and Objects

## Array of Objects

- To get all the information about the first dinosaur, you can use the same technique we used before, entering the index in square brackets:

```
> dinosaurs[0];
{ name: "Tyrannosaurus Rex", period: "Late Cretaceous" }
```

# Combining Arrays and Objects

## Array of Objects

- If you want to get only the name of the first dinosaur, you can just add the object key in square brackets after the array index:

  > dinosaurs[0]["name"];

  "**Tyrannosaurus Rex**"

- Or, you can use dot notation, like this:

  > dinosaurs[1].period;

  "**Late Jurassic**"

# Example for Array of Objects

LECTURE 7

# An Array of Friends

- Let's look at a more complex example now. We'll create an array of friend objects, where each object also contains an array. First, we'll make the objects, and then we can put them all into an array.

```
var anna = { name: "Anna", age: 11, luckyNumbers: [2, 4, 8, 16] };
var dave = { name: "Dave", age: 5, luckyNumbers: [3, 9, 40] };
var kate = { name: "Kate", age: 9, luckyNumbers: [1, 2, 3] };
```

- First, we make three objects and save them into variables called anna, dave, and kate. Each object has three keys: name, age, and luckyNumbers.
- Each name key has a string value assigned to it, each age key has a single number value assigned to it, and each luckyNumbers key has an array assigned to it, containing a few different numbers.

# An Array of Friends

- Next we'll make an array of our friends:

```
var friends = [anna, dave, kate];
```

- Now we have an array saved to the variable friends with three elements: anna, dave, and kate (which each refer to objects). You can retrieve one of these objects using its index in the array:

```
> friends[1];
{name: "Dave", age: 5, luckyNumbers: Array[3]}
```

# An Array of Friends

- This retrieves the second object in the array, dave (at index 1). Chrome prints out Array[3] for the luckyNumbers array, which is just its way of saying, “This is a three-element array.” (You can use Chrome to see what’s in that array; see Exploring Objects in the Console.) We can also retrieve a value within an object by entering the index of the object in square brackets followed by the key we want:

```
> friends[2].name
"Kate"
```

- This code asks for the element at index 2, which is the variable named kate, and then asks for the property in that object under the key "name", which is "Kate". We could even retrieve a value from an array that’s inside one of the objects inside the friends array, like so:

```
> friends[0].luckyNumbers[1];
4
```

# An Array of Friends

- Figure 4-2 shows each index. friends[0] is the element at index 0 in the friends array, which is the object anna. `friends[0].luckyNumbers` is the array [2, 4, 8, 16] from the object called anna.
- Finally, `friends[0].luckyNumbers[1]` is index 1 in that array, which is the number value 4

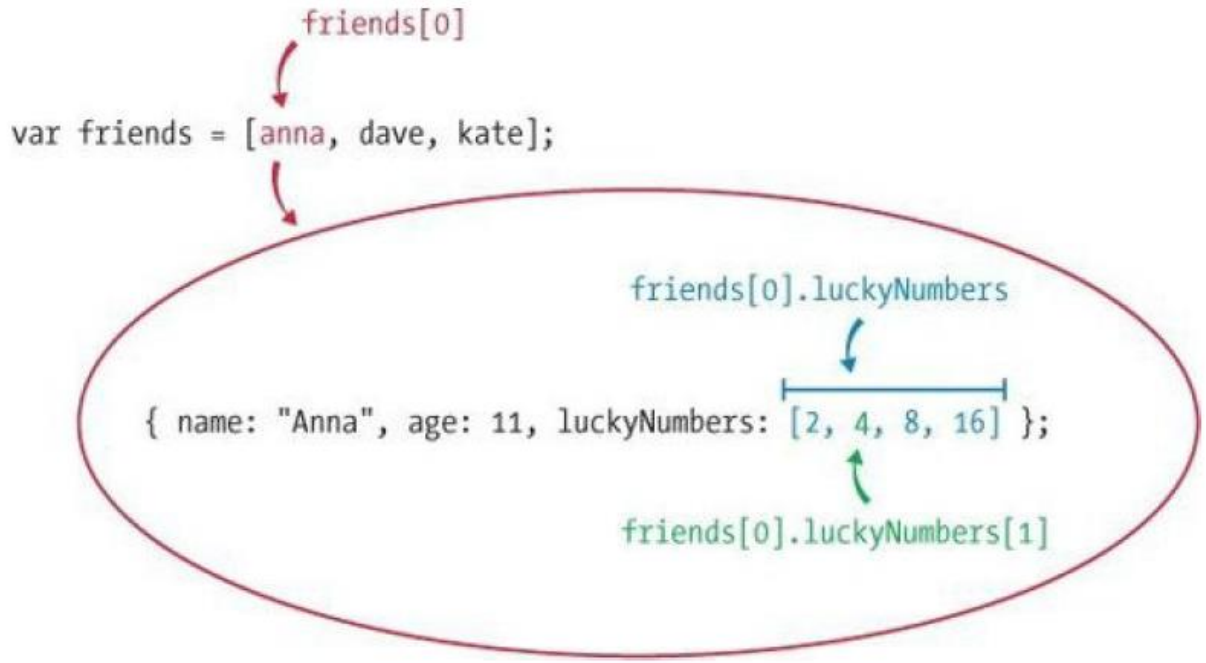



*Figure 4-2. Accessing nested values*

# Exploring Objects in the Console

LECTURE 8

# Exploring Objects in the Console

Chrome will let you dig into objects that you print out in the console. For example, if you type . . .

```
friends[1];
```

Chrome will display the output shown in Figure 4-3.

```
friends[1];
▶ Object {name: "Dave", age: 5, luckyNumbers: Array[3]}
```

*Figure 4-3. How an object is displayed in the Chrome interpreter*

# Exploring Objects in the Console

The triangle on the left means that this object can be expanded. Click the object to expand it, and you'll see what's shown in Figure 4-4.

```
friends[1];
▼ Object {name: "Dave", age: 5, luckyNumbers: Array[3]}
    age: 5
  ▶ luckyNumbers: Array[3]
    name: "Dave"
  ▶ __proto__: Object
```

*Figure 4-4. Expanding the object*

# Exploring Objects in the Console

You can expand `luckyNumbers`, too, by clicking it (see Figure 4-5).

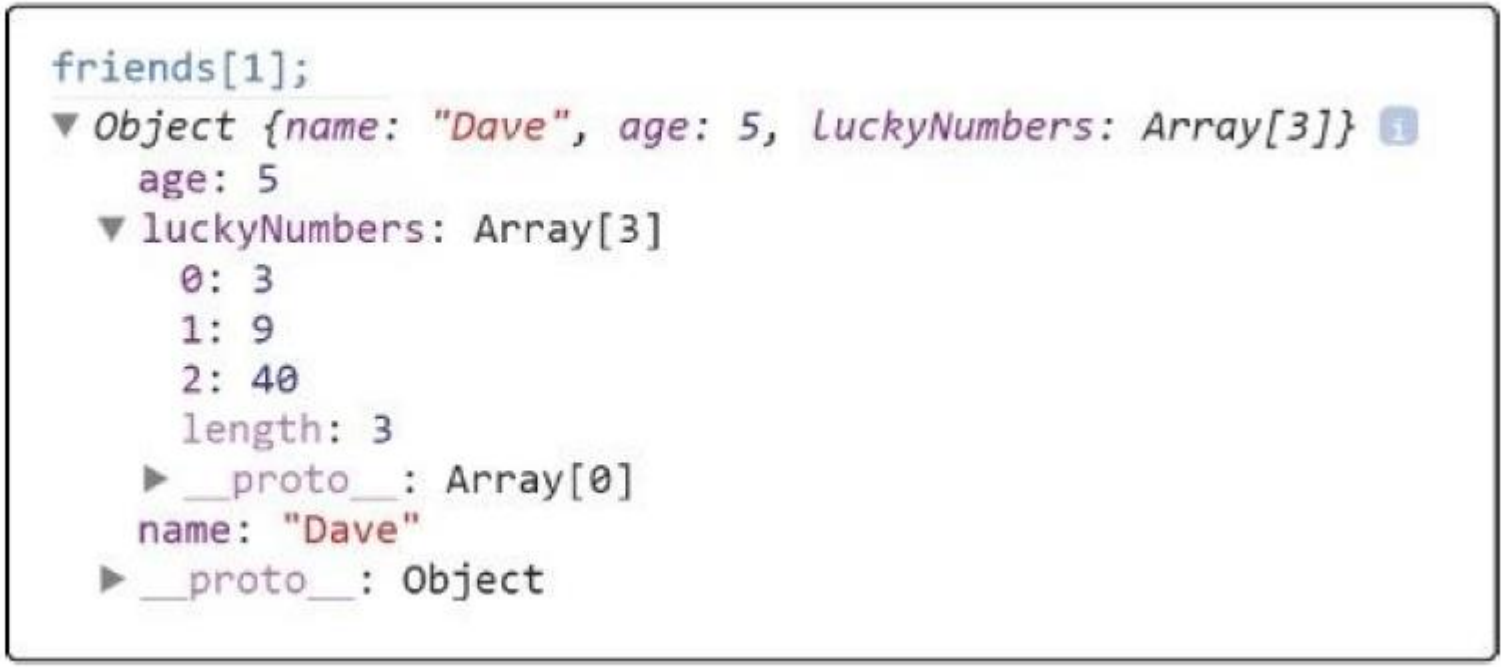



*Figure 4-5. Expanding an array within the object*

# Exploring Objects in the Console

Don't worry about those `__proto__` properties — they have to do with the object's *prototype*. We'll look at prototypes later, in Chapter 12. Also, you'll notice that the interpreter shows the value of the array's `length` property.

You can also view the entire `friends` array and expand each element in the array, as shown in Figure 4-6.

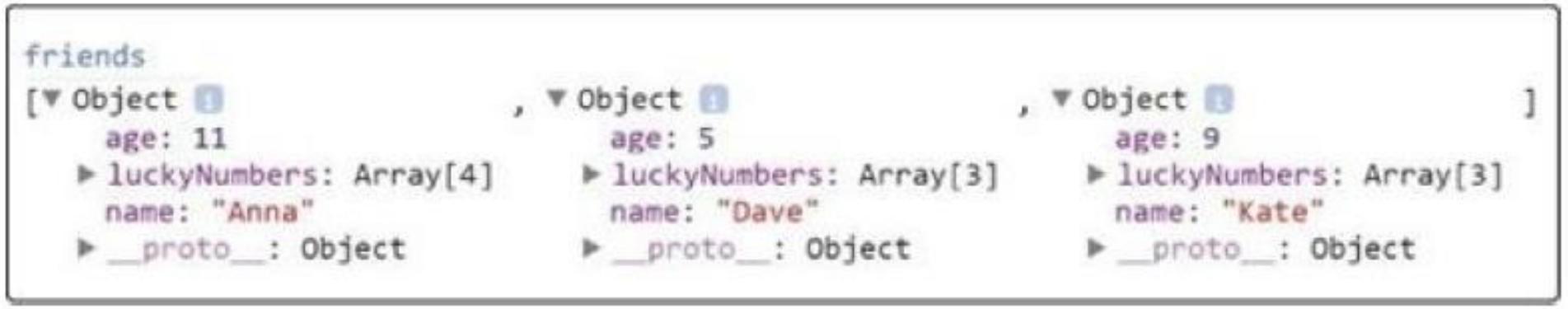



*Figure 4-6. All three objects from the `friends` array, as shown in the Chrome interpreter*

# Project 1: Keeping Track of Owed Money

LECTURE 9

# Objects with Key-Value Pairs

- Let's say you've decided to start a bank. You lend your friends money, and you want to have a way to keep track of how much money each of them owes you.
- You can use an object as a way of linking a string and a value together. In this case, the string would be your friend's name, and the value would be the amount of money he or she owes you. Let's have a look.

```
❶ var owedMoney = {};
❷ owedMoney["Jimmy"] = 5;
❸ owedMoney["Anna"] = 7;
❹ > owedMoney["Jimmy"];
  5
❺ > owedMoney["Jinen"];
  undefined
```

# Objects with Key-Value Pairs

At ❶, we create a new empty object called owedMoney. At ❷, we assign the value 5 to the key "Jimmy". We do the same thing at ❸, assigning the value 7 to the key "Anna". At ❹, we ask for the value associated with the key "Jimmy", which is 5. Then at ❺, we ask for the value associated with the key "Jinen", which is undefined because we didn’t set it.

# Objects with Key-Value Pairs

- Now let's imagine that Jimmy borrows some more money (say, $3). We can update our object and add 3 to the amount Jimmy owes with the plus-equals operator (+=) that you saw in Chapter 2.

```
owedMoney["Jimmy"] += 3;
> owedMoney["Jimmy"];
8
```

- This is like saying owedMoney["Jimmy"] = owedMoney["Jimmy"] + 3. We can also look at the entire object to see how much money each friend owes us:

```
> owedMoney;
{ Jimmy: 8, Anna: 7 }
```

# Key-Value Pair (Set and Get Value)

## Demo Program: keyvalue.html

# Go Brackets!!!

```
<html>
<body>
<script>
document.write("<h1>Keeping Track of Owed Money</h1>");
var owedMoney ={};
owedMoney["Jimmy"] = parseInt(prompt("Enter Jimmy's Money"));
owedMoney["Anna"] = parseInt(prompt("Enter Anna's Money"));
owedMoney["Linen"] = parseInt(prompt("Enter Linen's Money"));
document.write("<h2>Jimmy, Anna, and Linen's Owed Money</h2>");
document.write("<font size=\"+2\">");
var newline = "<br>";
for (var person in owedMoney){
    document.write(person+"'s owed money is "+owedMoney[person]+
                   " dollars. "+newline);
}
document.write("</font>");

document.write("<h2>Inquiry for Owed Money</h2>");
var person = prompt("Check whose owed money? (Jimmy, Anna, or Linen): ").trim();
var money = owedMoney[person];
document.write("<font size=\"+2\">");
document.write(person+"'s owed money is: "+money+" dollars. ");
document.write("</font>");
</script>
</body>
</html>
```

Prompt to get an integer.

For each key (person)

# Keeping Track of Owed Money

## Jimmy, Anna, and Linen's Owed Money

Jimmy's owed money is 5 dollars.
Anna's owed money is 4 dollars.
Linen's owed money is 3 dollars.

## Inquiry for Owed Money

Jimmy's owed money is: 5 dollars.

# Project 2: Storing Information About Your Movies

LECTURE 1

# Heterogenous Objects

```
var movies = {
  "Finding Nemo": {
        releaseDate: 2003,
        duration: 100,
        actors: ["Albert Brooks", "Ellen DeGeneres", "Alexander Gould"],
        format: "DVD"
  },
 "Star Wars: Episode VI - Return of the Jedi": {
        releaseDate: 1983,
        duration: 134,
        actors: ["Mark Hamill", "Harrison Ford", "Carrie Fisher"],
        format: "DVD"
  },
 "Harry Potter and the Goblet of Fire": {
        releaseDate: 2005,
        duration: 157,
        actors: ["Daniel Radcliffe", "Emma Watson", "Rupert Grint"],
        format: "Blu-ray"
  }
};
```

# Double Quotes

- You might have noticed that I used **quotes** for the movie titles (the keys in the outer object) but not for the keys in the inner objects. That's because the movie titles need to have spaces — otherwise, I'd have to type each title like **StarWarsEpisodeVIReturnOfTheJedi**, and that's just silly!

- I didn't need quotes for the keys in the inner objects, so I left them off. It can make code look a bit cleaner when there aren't unnecessary punctuation marks in it.

# Inquiry on Console

Now, when you want information about a movie, it’s easy to find:

```
var findingNemo = movies["Finding Nemo"];
> findingNemo.duration;
100
> findingNemo.format;
"DVD"
```

# Inquiry on Console

- Here we save the movie information about *Finding Nemo* into a variable called findingNemo. We can then look at the properties of this object (like duration and format) to find out about the movie.
- You can also easily add new movies to your collection:

```
var cars = {
    releaseDate: 2006,
    duration: 117,
    actors: ["Owen Wilson", "Bonnie Hunt", "Paul
    Newman"],
    format: "Blu-ray"
};
movies["Cars"] = cars;
```

# Inquiry on Console

- Here we create a new object of movie information about *Cars*. We then insert this into the movies object, under the key "Cars".
- Now that you're building up your collection, you might want to find an easy way to list the names of all your movies. That's where Object.keys comes in:

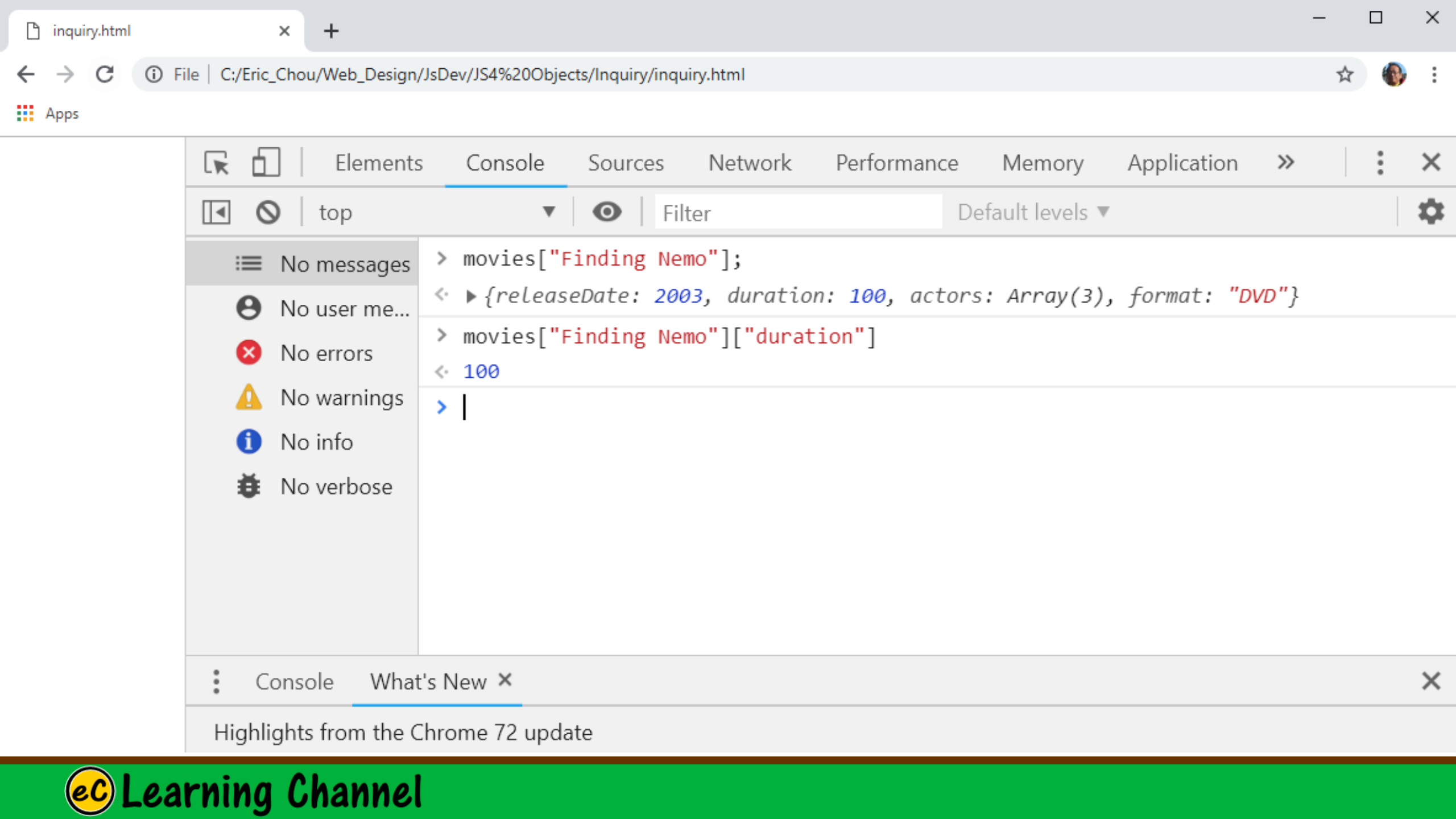
> Object.keys(movies);

["**Finding Nemo**", "**Star Wars: Episode VI - Return of the Jedi**", "**Harry Potter and the Goblet of Fire**", "**Cars**"]

# Inquiry

## Demo Program: inquiry.html

# Go Brackets!!!


inquiry.html
File | C:/Eric_Chou/Web_Design/JsDev/JS4%20Objects/Inquiry/inquiry.html
Apps
Elements
Console
Sources
Network
Performance
Memory
Application
top
Filter
Default levels
No messages
No user me...
No errors
No warnings
No info
No verbose
movies["Finding Nemo"];
▶{releaseDate: 2003, duration: 100, actors: Array(3), format: "DVD"}
movies["Finding Nemo"]["duration"]
100
Console
What's New
Highlights from the Chrome 72 update
eC Learning Channel

# Built-in Objects

LECTURE 10

# Built-in Objects

- The **global object**
  - Named `window` in browsers
  - Has properties representing all global variables
  - Other built-in objects are also properties of the global object
    - Ex: initial value of `window.Array` is `Array` object
  - Has some other useful properties
    - Ex: `window.Infinity` represents Number value

# Built-in Objects

- The global object and variable resolution:

```
i = 42;
```

What does `i` refer to?
1.Search for local variable or formal parameter named `i`
2.If none found, see if global object (`window`) has property named `i`

- This is why we can refer to built-in objects (`Object`, `Array`, etc.) without prefixing with `window.`

# Built-in Objects

- `String()`, `Boolean()`, and `Number()` built-in functions can be called as constructors, created “wrapped” Objects:

```
var wrappedNumber = new Number(5.625);
```

- Instances inherit `valueOf()` method that returns wrapped value of specified type:

```
window.alert(typeof  wrappedNumber.valueOf());
//   Output is “number”
```

# Built-in Objects

**TABLE 4.8** Some of the Methods Inherited by `String` Instances

| Method | Description |
|---|---|
| `charAt(Number)` | Return string consisting of single character at position (0-based) Number within this string. |
| `concat(String)` | Return concatenation of this string to String argument. |
| `indexOf(String, Number)` | Return location of leftmost occurrence of String within this string at or after character Number, or −1 if no occurrence exists. |
| `replace(String, String)` | Return string obtained by replacing first occurrence of first String in this string with second String. |
| `slice(Number, Number)` | Return substring of this string starting at location given by first Number and ending one character before location given by second Number. |
| `toLowerCase()` | Return this string with each character having a Unicode Standard lowercase equivalent replaced by that character. |
| `toUpperCase()` | Return this string with each character having a Unicode Standard uppercase equivalent replaced by that character. |

# Built-in Objects

- The `Date()` built-in constructor can be used to create `Date` instances that represent the current date and time

```
var now = new Date();
```

- Often used to display local date and/or time in Web pages

```
window.alert("Current date and time: "
                    + now.toLocaleString());
```

- Other methods: `toLocaleDateString()`, `toLocaleTimeString()`, *etc.*

# Built-in Objects

- `Math` object has methods for performing standard mathematical calculations:

```
Math.sqrt(15, 3);
```

- Also has properties with approximate values for standard mathematical quantities, *e.g.*, $e$ ( `Math.E` ) and $\pi$ (`Math.PI`)