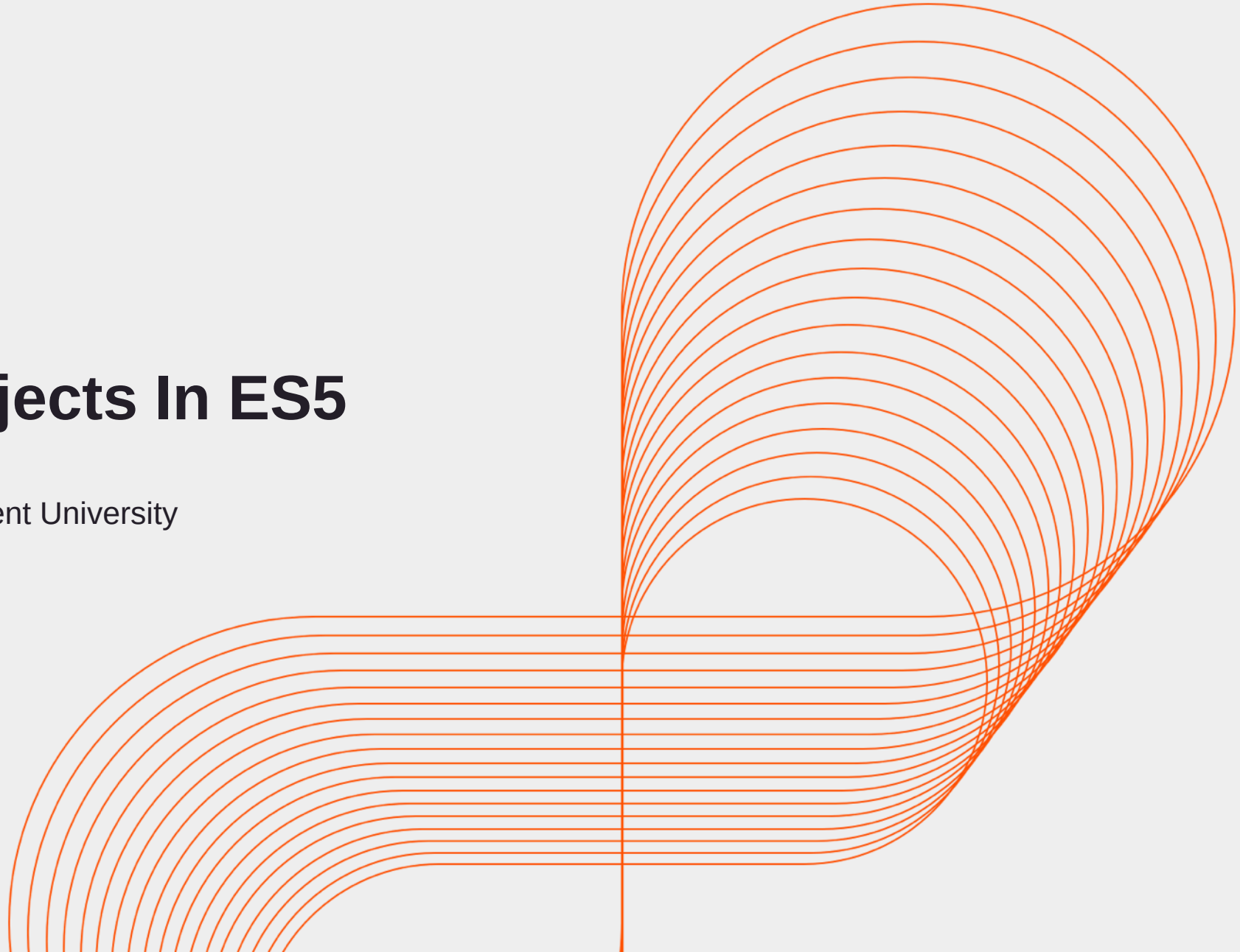
Persistent

JavaScript Objects In ES5

Persistent Interactive | Persistent University



Key learning points :

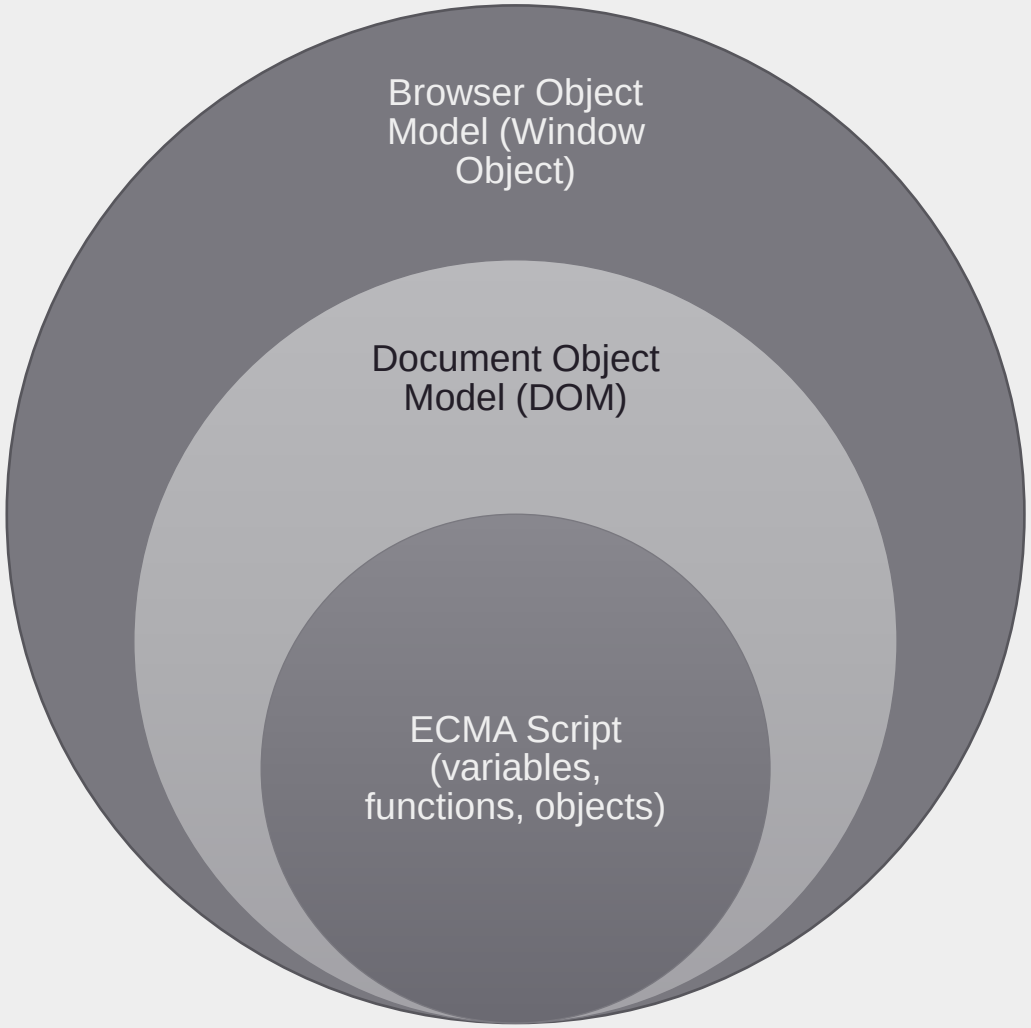
- Overview & significance of Objects

- Different ways to create user defined objects in detail
  - direct instances as a blank object
  - Object Literals
  - Object Constructors

- User defined object properties
  - operations like create, read , update and delete

- Singleton design pattern

JavaScript Object Hierarchy



Objects

- Unordered collection of properties each of which contains :-
  - a primitive value,
  - object or
  - function

- A grouping of name-value pairs where the value may be data or a function

- Created based on a reference type and are kept in hash tables

JavaScript Object

- A JavaScript object has properties and methods
  - Example: String JavaScript object has ‘length’ property and ‘toUpperCase()’ method

```
<script type="text/javascript">

        var txt="Hello World!“

        document.write(txt.length);

        document.write(txt.toUpperCase());

</script>
```

How to create Objects ?

- User Defined Objects
  - create direct instance of object as an empty object
  - object literal
  - using template, known as object constructors

- Built-in Objects
  - String
  - Date
  - Array
  - Boolean
  - Math

User defined or custom Objects

- Create direct instance of a JavaScript object
  - Using curly braces ‘{}’
  - Through built in ‘Object’ Constructor using new operator

- In both the cases above, blank or empty object is created
  - object without any properties

```
var personObj1 = {};

var personObj2 = new Object();
```

User defined properties of an object

- Add and update properties
  - object name followed by
  - dot ‘.’ operator followed by
  - property name

- Using dot operator, property will be
  - added if does not exist
  - updated if already existing

- Use ‘delete’ operator to delete properties
  - returns true if property is deleted successfully
  - returns false if property can’t be deleted

```
personObj.id = 1;

personObj.name = 'Anand’;


delete personObj.id;

delete personObj.name;
```

User defined properties continued..

- Fetching properties
  - using dot '.' operator
  - using loops like for in

```
console.log(personObj.id); // 1

console.log(personObj.name); // Anand

for(var prop in personObj){

        console.log(personObj[prop]);
}
 // 1
// Anand
```

Functions as object properties

- Adding method to object
  - object name followed by
  - dot ‘.’ operator followed by
  - method name

- Invoking object’s method

```
personObj.getId = function(){

        return this.id;
};

console.log(personObj.getId()); // 1
```

Object Literals

- List of name value pairs wrapped in curly braces
  - name and value are separated by colon ‘:’
  - name value pair are separated by comma ‘,’
  - name represents property
  - value is simply the value of the property which could be either plain data or some function expression

- Used to enclose properties in a tidy package
  - helps to encapsulate data
  - minimize usage of global variables

Example of an Object Literal

```
var personObj = {
                    id : 1,
                    getId : function(){
                            return this.id;
                    }
};

console.log(personObj.getId()); // 1
```

Limitations of Object Literals

- Object creation is not easy if multiple objects of same types are required
  - need to type each object instance along with their properties even if they follow almost same design
  - lot of typing and redundancy
  - more error prone
  - any addition\update\deletion of property requires changes in a lot of places
  - difficult to read, understand and debug code, overall, not maintainable

- Lot of object literals means a lot of code
  - may slow download times for browser

Solution to previous issues : Object Constructors

- Creating objects out of object constructor is a 2-step process :
  - define object constructor
  - create objects using ‘new’ operator

- Object constructors are like factory that can create objects of same type

- Mostly, function declarations are used to write object constructors

- As a best practice :-
  - constructor’s name should start from capital to differentiate
  - it from a normal function

Defining Object Constructors

- Write a function declaration whose name will be the name of the object constructor and used to create objects

- Properties inside object constructor
  - add properties using ‘this’ keyword followed by ‘.’ operator followed by property name
  - property name and property value are separated by assignment operator ‘=’
  - property value (optional) can be specified after ‘=’

```
function Employee(fName, loc){

        this.firstName = fName;
        this.location = loc;
}
```

Create objects using object constructors

- Objects are created using ‘new’ keyword
- Object properties can also be added after object’s creation

```
var e1 = new Employee('Sachin','Mumbai');

var e2 = new Employee('Virat','Delhi’);

e1.getFirstName = function(){
        return this.firstName;
};

console.log(e1.getFirstName()); // Sachin
```

Let us understand behavior of ‘this’ keyword

- ‘this’ :-
  - always refers to the current object
  - dynamic, takes reference of object at run time

- Inside the object constructor
  - properties are added using ‘this’ keyword
  - without using ‘this’, one can’t refer the object properties i.e. add, access, update, delete operations won’t work

- Properties are associated with objects and ‘this’ must be used if object name can’t be used explicitly and should be referred at run time

Object Constructors & Literals - Usage

- Object Constructors
  - is preferred if some pre-processing is required before creation of the objects
  - is required if multiple instances of the object needs to be created where each instance can be changed during the lifetime of the script

- Object Literals
  - is required if using object as a singleton and not requiring
    more than one instance of the object
  - is a preferred option for name spacing so that your JavaScript
    code doesn’t interfere (or vice versa) with other scripts running on the page

Singleton design pattern

- Only one instance of an object of its kind

- Singleton pattern :-
  - ensure an object only has one instance
  - provides a global point of access to it
  - used for name spacing and reducing the global variables

- Creating an object using object literal is an example of singleton.

Singleton as an object literal

```
var emp1 = {
            name : "John“
};

var emp2 = {
 name : "John“
};

console.log(emp1 === emp2); // false
console.log(emp1 == emp2);  // false
```

Summary : Session

With this we have come to an end of our session, where we discussed :

- Objects, its features and characteristics
- Different ways to write objects like literals, constructors etc
- Custom object properties along with all CRUD operations
- Singleton design pattern

At the end of this session, we expect you to :

- Understand all discussed concepts related to objects
- Appropriately use objects whenever required

Appendix

- References
- Key Contacts

Reference Material : Books

Head First JavaScript Programming

- By: Eric T. Freeman; Elisabeth Robson
- Publisher: O'Reilly Media, Inc.

Professional: JavaScript® for Web Developers

- By: Nicholas C. Zakas
- Publisher: Wrox

Object-Oriented JavaScript

- By: Stoyan Stefanov; Kumar Chetan Sharma
- Publisher: Packt Publishing

Persistent

Thank You !!!

Persistent Interactive | Persistent University