**Object Oriented Programming**

**Objects:**Object is the basic unit of object-oriented programming. An object represents a particular instance of a class. There can be more than one instance of an object.

An Object is a collection of **data members and associated member functions** also known as methods.  
  
**Classes:**  Objects with similar properties and methods are grouped together to form a Class. Thus a Class represent a set of individual objects. **Characteristics** of an object are represented in a class as **Properties**. The **actions** that can be performed by objects becomes functions of the class and is referred to as **Methods**.

**The blueprint is the class...the house is the object.  The people living in the house are data stored in the object's properties.**

**Abstraction:** Abstraction means showing essential features and hiding non-essential features to the user.

 There are lots of things you could know about a person (their address, height, shoe size, DNA profile, passport number, significant personality traits ...) , but in this case we are only interested in showing their name, age, gender, and interests .This is known as **abstraction** — **creating a simple model of a more complex thing**, which represents its **most important aspects in a way that is easy to work** with for our program's purposes.

**Encapsulation:** Encapsulation means which binds the data and code (or) writing operations and methods in single unit (class).

**Encapsulation = Abstraction + Data Hiding. (**Protecting our Data**)**

**Inheritance:** Deriving a new class from the existing class,is called Inheritance.

**OOJS**

**Object in JS:**

**primitive data types**: Number, String, Boolean, Undefined, and Null are immutable

**complex data type** :objects are mutable (can be changed).Anything that doesn’t belong to any of these five primitive types is considered an object.

An object is an unordered list of primitive data types (and sometimes reference data types) that is stored as a series of name-value pairs.

var ageGroup = {30: "Children", 100:"Very Old"};

console.log(ageGroup["30"]); // Children​

console.log(ageGroup.30) // This will throw an error​

primitive data saved-as-value

objects: save-as-reference

**Object Data Properties Have Attributes**

Each data property (object property that store data) has not only the name-value pair, but also 3 attributes (the three attributes are set to true by default):

* **Enumerable**: I can access to all of them using a for..in loop. Also, enumerable property keys of an object are returned using [Object.keys](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/keys" \t "_blank) method.
* **Writable**: I can modify their values, I can update a property just assigning a new value to it: ob.a = 1000;
* **Configurable**: I can modify the behavior of the property, so I can make them non-enumerable, non-writable or even non-cofigurable if I feel like doing so. Configurable properties are the only ones that can be removed using the delete operator.

**Configurable** prevents any attempts to **'redefine' the three properties** of a key with **Object.defineProperty**, chrome will throw an error sign

*Uncaught TypeError: Cannot redefine property: foo*

The **writable** attribute simply avoids this value from being edited

**Creating Objects**

1. **Object Literals**
2. **Object Constructor**

**Constructor**

JS uses special functions called **constructor functions** to define objects and their features.

A constructor function name usually starts with a capital letter — this convention is used to make constructor functions easier to recognize in code.

### Using the create() method

JavaScript has a built-in method called [create()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/create), which allows you to create a new object instance based on an existing object.

var person2 = Object.create(person1);

[**JavaScript Prototype in Plain, Detailed Language**](http://javascriptissexy.com/javascript-prototype-in-plain-detailed-language/)

Every JavaScript function has a prototype property (this property is empty by default), and you **attach properties and methods on this prototype property when you want to implement inheritance**

This **prototype property** is **not enumerable;** that is, it isn’t accessible in a for/in loop. But Firefox and most versions of Safari and Chrome have a **\_\_proto\_\_ “pseudo” property** (an alternative syntax **depricated**) that allows you to access an object’s prototype property.

Every JavaScript function has **a prototype property** (**this property is an empty object by default**), and you **attach properties and methods on this prototype property when you want to implement inheritance**

var person2 = Object.create(person1);

1. What create() actually does is to **create a new object from a specified prototype object.**

2. Here, person2 is being created using person1 as a prototype object. You can check this by entering the following in the console:

**person2.\_\_proto\_\_**

This will return the **person1** object.

NOTE :

If you try to look up a key on an object and it is not found, JavaScript will look for it in the prototype. It will follow the "**prototype chain**" until it sees a null value. In that case, it returns undefined.

**the properties** that JavaScript includes on  **Object.prototype are not enumerable**

function Tree(name) {  
 this.myvar = name;

var localvar=”local”;

}

1) methods inside constructor **can use both  local variable** in the constructor and **this.myVar**

**var** localVar = {}

methods that does not require access to the local variable can be defined on the prototype.

prototypes can access **this.myvariable** = "value";

but prototypes  cannot access **var localVar** = "localValue";