**JavaScript Paradigma POO**

Poo es hace referencia a un paradigma de programación que se centra en objetos.

**Objetos:** {},[] para JS los array también son objetos, el propósito de los objetos es que tienen métodos.

**Crear objetos:** agrupar datos

Const user ={

Nam:’Aleja’,

Lastname:’Orrego’,

Age:30,

Hobbies:[ ‘read’, ’programming’, ’run],

Address:{

Street: ‘some street’,

City: ‘Medellín’

}

}

**Propiedades**: Datos que describen a un objeto ya sea real o virtual.

Con esto estamos modelando un objeto.

Ejemplo Auto

* Modelo, color, fecha de creación, velocidad, precio

Ejemplo cuentas de banco

* Número, tipo, dueño, monto, banco

**Métodos**: Los objetos también pueden hacer cosas

Ejemplo del auto

* Avanzar, retroceder, prender, apagar

**Uso del this**: obtener los valores de un objeto

* This.name retorna el valor de un objeto

**Constructor**: crear objetos de manera automática, es equivalente a una class, esto nos permited definit las propiedades y métodos que tiene un objeto.

Function Person(){

This.name =’’

This.lasname =’’

This.showFullName =function (){

Return `${this.name}`

}

}

**Crear un objeto** a partir de una plantilla.

Const user =new Persona()

Pasar valores

User.name=’Alejandra’

**New object** es lo mismo que escribir {}, esto se utilizaría para generar apartir de expresiones

Ejemplo

Const num= new Object(12)//para crear un objeto tipo number o de tipo primitivo.

Ver las claves de un objecto

* Object.keys(user) = retorna arreglo con las claves
* Object.values(user)= retorna arreglo con los valores del objecto

**Palabra clave new:** sin la palabra new se ejecuta una función mientras que con la palabra new crea un nuevo objeto.

**Use strict**: evitar problema de asignarle a un objeto por encima un constructor.

**Prototype**: Alterar el constructor para que un método sea creado de forma general a la plantilla, al mostrar el objeto no se muestra por que estos se crean el contructor.

Con esto también se pueden alterar los objetos de JavaScript ejemplo crear un nuevo método para un objeto tipo String.

En JS a la programación orientada a objetos se le conoce como programación orientada prototipos.

Person.prototype.age=30;

Person.prototype.greet = function(){

Return ‘hello’

}

**Class**: Conjunto de métodos que permiten definir un objeto, forma de escribir más bonito las funciones que eran tipo constructor

Ejemplo:

Modelo de objetos

Class Person {

//constructor devuelve un objeto

Constrctor(name, lasname){

This.name= name

This.lasname= lastname

}

//crear metodos

Greet(){

Return ‘hello’

}

}

Crear instacioas

Const user= new Person(‘alejandra’,’Orrego’)

**Principios de la POO = Abstracción**

Propósito crear código modular y reutilizable, dividir el código en partes.

Características:

* ***Asociación***: poder unir dos objetos.
* ***Agregación***: un objeto grande creado con pequeños siendo independientes
* ***Composición***: un objeto grande conformado por pequeños en el cual el grande ya no serviría si se quita uno pequeño.

***Encapsulación***: concentrar datos en una sola entidad, la lógica no se ve.

***Herencia***: un objete hereda característica de una o más objetos

***Polimorfismo***: permite dependiendo de lo que se pase se procesa y al final da una respuesta dependiendo de lo enviado.

**Ejemplo de asociación:**

Dos objetos relacionados, pero si quita la relación siguen funcionando

Objeto1.padre =objeto2

Sirve por ejemplo para amigos en una red social.

**Ejemplo de la agregación:**

Es una forma de asociación donde se relacionen objetos y hay roles mayores (aggregate) y su hijo se les llama (component).

Objetos personas que hacen parte de u nobjeto company

Company.trabajador.push(persona)

Un componente no tiene vida independiente cuando depende de otro

Como por ejemplo la dirección de un usuario que está definida como objeto.

**Encapsulación**: simplifica el uso de un objeto, no dar acceso directo al usuario solo da métodos.

Usar getters and setters

**Herencia**: crear objetos especializados a partir de uno más genérico

Un programador es un usuario

Ejemplo

Function Person(){

This.name=’’

This.lasname=’’

}

Los objetos de progrmador heredan de persona

Function programmer(){

This.languaje=’’

}

Con esto le emos dicho que a heredado de person

Programmer.prototype = new Person()

Const programmer = new Programmer()

Utilizando clases es mucho mejor

Class Person(){

Constructor(name, lasname){

This.name=name

This.lastname=lastname

}

}

Class Programmer extends Person {

Contructor(languaje){

Super()//trae los inicializadores del contructor padre

This.languaje=languaje

}

}

Polimorfismo: una de sus características es la sobrecarga, polimorfismo paramétrico, es decir con datos genreicos, polimorfismo enclisivo.

*Sobrecarga*: overloading creo métodos con el mismo nombre y estos procesan distinto tipos de datos lo que cambia es el paso de parámetros

Function suma(x=0,y=0,z=0)

Con esto podemos simular la sobrecarga

*Polimorfismo paramétrico:* capacidad de un método que funciona con parámetros de cualquier tipo (generics)

En javascript se puede pasar cualquier tipo de dato sin especificar

Function Stack(){

This.items=[]

}

*Polimorfismo subtipo:* pasar un tipo de dato que vienen de una clase que hereda

class Person {

constrctor(name, lastname){

this.name=name

this.lastname=lasname

}

}

Class Programmer extends Person{

Contructor(name , lastname, languaje){

Super(name, lastname)

This.languaje =lenguaje

}

}

Donde p es cualquier tipo de dato

Function writeFullName(p){

p.name, p.lastname p.languaje

}

Typeof = muestra el tipo de dato

RegExp() = expresiones regulares