Clases en JavaScript

Las clases de JavaScript son fábricas de objetos.

1. Declarar una clase: Se utiliza la palabra clave `class` seguida del nombre de la clase.

```
class NombreClase {

}
```

2. Constructor: Dentro de la clase se define una función constructora usando la palabra `constructor()`. Se ejecuta cuando se crea una nueva instancia de la clase.

```
class NombreClase {
  constructor() {
    // código a ejecutar
  }
}
```

3. Propiedades y métodos: Dentro de la clase se definen propiedades y métodos como funciones.

```
class NombreClase {
  propiedad1 = 'valor';

  metodo1() {
    // código del método
  }
}
```

4. Extends: Se puede heredar de otra clase utilizando la palabra `extends`.

```
class NombreClase extends OtraClase {

}
```

5. Instanciar objeto: Se crea una nueva instancia de la clase con la palabra `new`.

```
const objeto = new NombreClase();
```

6. Acceder a propiedades y métodos: Se acceden utilizando la sintaxis de punto.

```
objeto.propiedad1;
objeto.metodo1();
```

Nombre de la clase

Propiedades o atributos: privados o publicos. Los privados siempre comienzan con símbolos hash (#)

constructor() es un método especial que se llama después de la creación de una nueva instancia

método o función

instancias - "creación de objetos"

```
class Persona {
    #PrimerNombre; // (A)
    constructor(PrimerNombre) {
        this.#PrimerNombre = PrimerNombre; // (B)
    }
    describe() {
        return `persona llamada ${this.#PrimerNombre}`;
    }
    static extractNames(persons) {
        return persons.map(person => person.#PrimerNombre);
    }
}
const ana = new Persona('Ana');
const juan = new Persona('Juan');

console.log(ana.describe());
```

Ejemplo 2

Puedo cargar datos por constructor o con el método set

métodos get - "traer"

método set - "cargar"

instancia

```
class Empleado {
    #nombre;
    constructor(nombre) {
        this.#nombre = nombre;
    }
    get nombre() {
        return this.#nombre;
    }

    set nombre(value) {
        this.#nombre = value;
    }
}

const empleado = new Empleado("Arle");
console.log(empleado.nombre);
empleado.nombre = "Manuel";
console.log(`el nuevo nombre de este objeto es  ${empleado.nombre}`)
```

Ejemplo 3

```
class Subasta {
    constructor(producto, precioBase) {
        this.producto = producto;
        this.precioBase = precioBase;
        this.pujas = [];
        this.finalizada = false;
    }
    agregarPuja(puja) {
        if (!this.finalizada && puja > this.getPrecioActual()) {
            this.pujas.push(puja);
            console.log(`Se agregó una puja por $${puja}`);
        } else {
            console.log(`La puja por $${puja} no es válida`);
        }
    }
    finalizar() {
        this.finalizada = true;
        console.log(`La subasta para ${this.producto} ha finalizado`);
    }
    getPrecioActual() {
        if (this.pujas.length > 0) {
            return Math.max(...this.pujas);
        } else {
            return this.precioBase;
        }
    }
}

const subas1 = new Subasta("Gafas", 100);
subas1.agregarPuja(102);
subas1.agregarPuja(105);
subas1.agregarPuja(110);
subas1.finalizar();
console.log("la subasta termno en " + subas1.getPrecioActual())
```

Se definen las propiedades con el constructor.

this hace referencia a la clase

instancia

Ejemplo 4

```
class Animal {
    constructor(nombre, especie) {
        this.nombre = nombre;
        this.especie = especie;
    }
    comer() {
        console.log(`${this.nombre} está comiendo.`);     }
    mover() {
        console.log(`${this.nombre} se está moviendo.`);   }
}
class Perro extends Animal {
    constructor(nombre, raza) {
        super(nombre, "perro");
        this.raza = raza;
    }

    ladrar() {
        console.log(`${this.nombre} está ladrando.`);   }
}
let miPerro = new Perro("Fido", "Labrador");
miPerro.comer(); // muestra "Fido está comiendo."
miPerro.mover(); // muestra "Fido se está moviendo."
miPerro.ladrar(); // muestra "Fido está ladrando."
```

herencia

Ejemplo 5

```
class Calculadora {
    static sumar(num1, num2) {
        return num1 + num2;
    }

    static restar(num1, num2) {
        return num1 - num2;
    }
}
console.log(Calculadora.sumar(2, 3)); // muestra 5
console.log(Calculadora.restar(5, 2)); // muestra 3
```

Los métodos estáticos se pueden utilizar sin crear una instancia de la clase.

guía práctica de Recursividad.

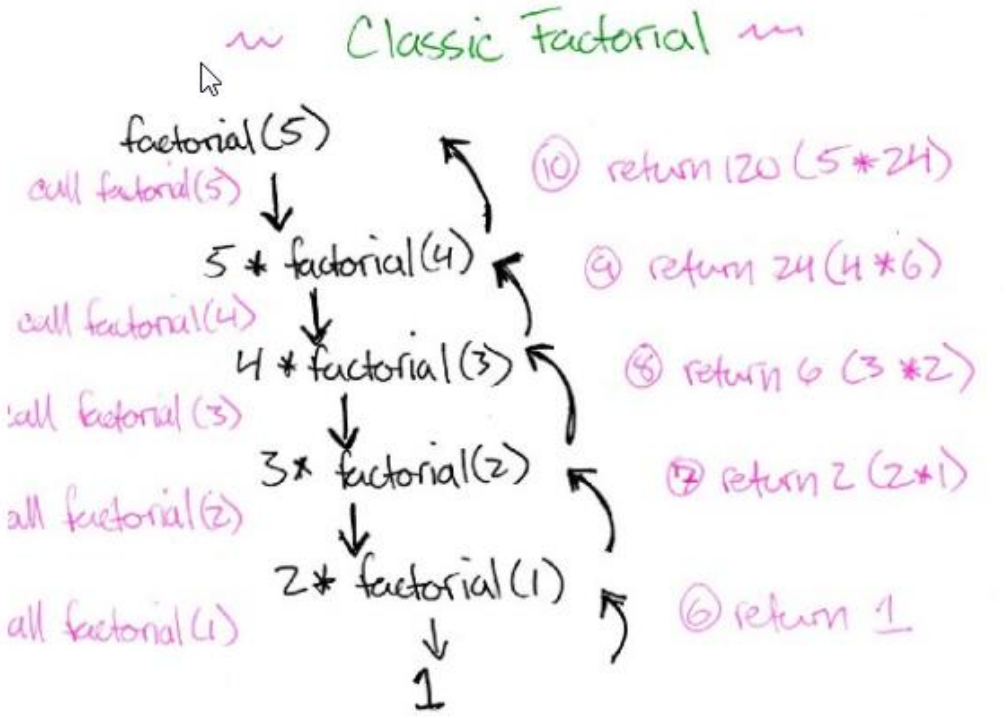
Una función recursiva es una función que se llama a sí misma hasta que una “condición base” sea verdad, y la ejecución se detiene.

Factorial:

condición base que detiene la ejecución

Se repite la función y se va almacenando en la pila

```
1  let factorial = x => {
2      if(x == 0) return 1;
3      return x * factorial( x: x-1);
4  }
5
6  let numero = 5;
7  console.log(`factorial de ${numero} es ${factorial(numero)}`);
```



La sucesión de Fibonacci:

Se trata de una secuencia infinita de números naturales; a partir del 0 y el 1, se van sumando a pares, de manera que cada número es igual a la suma de sus dos anteriores, de manera que: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55...

```
let x = numero =>{
    if (numero <= 1) return 1;
    return x( numero: numero-1)+ x( numero: numero-2);
}


let numero = 5;
for (let i=0; i <= numero;i++){
    console.log(x(i))
}
```

Función invertir una cadena de texto:

```
let  reverse= str=> {
    if (str.length === 0) return ''
    return str[str.length - 1] +
        reverse(str.substring(0, str.length - 1))}


console.log(reverse( str: "Maria"));
```

Función imprimir números ascendente y descendente.

```
let printFun = test=> {
    if (test < 1)
        return;
    else {
        console.log(test + " ");
        printFun( test: test - 1); // statement 2
        console.log(test + " ");
        return;
    }
}

printFun( test: 3);
```

Ejercicios propuestos

Ejercicio 1:

Construya un algoritmo con JavaScript" para las estadísticas de atención de una universidad teniendo en cuenta los siguientes requisitos:

1. Hay dos módulos de atención: terminal para llamada telefónica y oficina.
2. El sistema brinda las estadísticas de todo el proceso de atención:
    - Cantidad de usuarios atendidos.
    - Atendidos por día y especificación por segmento (Estudiante – docente) en cada uno de los módulos de atención.
    - Se permite trasferir de módulo de atención y se debe generar estadística de esta trasferencia.

Ejercicio 2:

El software que se desarrollará controlará un cajero automático (ATM) a través de una simulación usando el lenguaje de programación JavaScript.

- El cajero automático atenderá a un cliente a la vez. Se le pedirá al cliente que inserte su documento de identidad y su pin de 4 dígitos, los cuales se enviarán al banco para su validación como parte de cada transacción. El cliente podrá entonces realizar una o más transacciones. El menú se mostrará en la consola hasta que el cliente indique que no desea realizar más transacciones.
- El cajero automático debe ser capaz de proporcionar los siguientes servicios al cliente:

- Un cliente debe poder realizar un retiro de efectivo de cualquier cuenta adecuada vinculada al documento de identidad, en múltiplos de $50000. Se debe obtener la aprobación del banco antes de entregar efectivo.
- Un cliente debe poder realizar un depósito en cualquier cuenta vinculada al documento de identidad, consistente en efectivo y/o cheques. El cliente ingresará el monto del depósito en el cajero automático e indicar si es efectivo o cheque.
- Un cliente debe poder realizar una transferencia de dinero entre dos cuentas cualesquiera vinculadas a al documento de identidad.
- Un cliente debe poder realizar una consulta de saldo de cualquier cuenta vinculada al documento de identidad.

- El cajero automático comunicará al cliente los resultados de cada transacción dependiendo de su tipo. Ejemplo “retiro exitoso, puede tomar x dinero de la bandeja principal”
- Si el banco determina que el PIN del cliente no es válido, se le pedirá al cliente que vuelva a ingresar el PIN antes de que se pueda continuar con la transacción. Si el cliente no puede ingresar correctamente el PIN después de tres intentos saldrá de la aplicación.
- El cajero automático tendrá un panel de operador con un interruptor que permitirá apagar o encender el cajero.

Ejercicio 3:

- Desarrollar en JavaScript un programa para la gestión reservas de un hotel, el cual, debe tener las siguientes características y consideraciones:
- Un cliente puede reservar cualquier tipo de habitación: individual, doble y familiar.
- Las habitaciones pueden ser para fumadores o no fumadores.
- Las mascotas solo se aceptan en habitaciones familiares.
- El hotel cuenta con 3 habitaciones de cada tipo.
- No se puede exceder el número de personas por habitación: individual 2 personas, 4 personas para doble y 6 personas para familiar.
- El hotel necesita una estadística de las reservas: nombre de quien reserva, país de origen, número de personas, el periodo de la estadía, número de personas que están ocupando el hotel y si el huésped trajo mascota.

Ejercicio 4

Se necesita simular en JavaScript la atención de clientes a través de la asignación de turnos en un banco. Se debe usar arreglos o objetos dependiendo del algoritmo que diseñe. Y tener en cuenta las siguientes restricciones y requisitos.

- Hay tres tipos de clientes: cliente preferencial, cliente general y cliente que no tiene cuenta en el banco
- Hay dos tipos de atención: caja o asesoría.
- Los de atención de caja se clasifican en depósitos y retiros.
- El banco cuenta con 5 cajas, de las cuales la 1 y 2 están reservadas para retiros.

- Aquellos clientes presenciales se atienden primero de los demás tipos.
- La caja 5 es solo asesoría.
- A medida que se atienden clientes se va liberando las cajas y distribuyendo entre los usuarios de las colas.

Ejercicio 5

Desarrollar en JavaScript los siguientes algoritmos que den solución a la problemática planteada.

Implementar una clase en JavaScript, la cual tenga los siguientes atributos y métodos.

Atributos:

- Código.
- Descripción.
- Precio de compra.
- Precio de venta.
- Cantidad en bodega.
- Cantidad mínima requerida en bodega.
- Cantidad máxima de inventario permitida.
- Porcentaje de Descuento.

Métodos:

- Solicitar pedido: devuelva true si debe solicitar el producto al proveedor y false en caso contrario.
- Calcular total a pagar: devuelva la cantidad total a pagar al proveedor dado una cantidad de unidades de compra.

Adicionalmente se desea dos subclases para los siguientes tipos de productos:

- Prendas de vestir (como lo son blusas, jeans, camisas, etc.) el cual debe tener los siguientes parámetros adicionales:

- Talla: S, M, L, etc.
- Permite planchado: verdadero o falso.

- Calzado (como lo son tenis, calzado formal, sandalias, etc.) el cual debe tener el siguiente parámetro adicional:
  - Talla: 35, 36, 37, etc.

Diseñar un programa que:

- Consulte el número de productos de tipo de prendas de vestir a manejar.
- Consulte el número de productos de tipo calzado a manejar.
- Cree en una estructura de datos (arrays, map, set), los productos de prendas de vestir en el cual se guardarán las instancias de cada uno de ellos.
- Cree una estructura de datos (arrays, map, set) de productos de calzado en el cual se guardarán las instancias de cada uno de ellos.

Ejercicio 6:

Una subasta o remate es una venta organizada basado en la competencia directa, y generalmente pública, es decir, a aquel comprador que pague la mayor cantidad de dinero o de bienes a cambio del producto.

Hacer en JavaScript una simulación de subasta que cumpla con las siguientes características:

1. Se podrá registrar los productos a subastar almacenados (id del producto, nombre del producto, fecha y precio inicial de subasta).
2. Cada persona puede pujar por el producto que desea, indicando la fecha, el producto y el valor ofrecido.
3. Se puede ver la lista de productos registrados.
4. La lista de ofertas por producto.
5. Seleccionar una oferta ganadora.