**FUNDAMENTOS DE PROGRAMACIÓN DESDE CERO Y DESARROLLA TU LÓGICA PARA EMPEZAR TU CAMINO COMO DESARROLLADOR**



**ANDRÉS CAMILO CAMACHO LABRADOR**

**Autor**

**CURSO DE UDEMY**

**2024**

TABLA DE CONTENIDO

Pág.

[**1.** **FUNDAMENTOS** 4](#_Toc183785779)

[**1.1** **PALABRAS CLAVE PARA EMPEZAR** 4](#_Toc183785780)

[**1.1** **ALGORITMO** 4](#_Toc183785781)

[**1.2** **VARIABLE** 4](#_Toc183785782)

[**1.3** **PALABRA RESERVADA** 5](#_Toc183785783)

[**1.4** **LENGUAJE DE PROGRAMACIÓN A USAR: JavaScript** 5](#_Toc183785784)

[**1.5** **Console.log(10);** 5](#_Toc183785785)

[**1.6** **CAMELCASE** 5](#_Toc183785786)

[**1.7** **DEPENDENCIA** 6](#_Toc183785787)

[**1.8** **TIPOS DE DATOS Y FLUJO DE CONTROL** 6](#_Toc183785788)

[**1.8.1** **Tipos de variables** 6](#_Toc183785789)

[**1.9** **RECOMENDACIÓN A LA HORA DE NOMBRAR UNA VARIABLE** 7](#_Toc183785790)

[**1.10** **STRINGS – CADENA DE CARACTÉRES** 7](#_Toc183785791)

[**1.11** **BOOLEANS – BOOLEANOS** 7](#_Toc183785792)

[**2.** **ESTRUCTURAS DE CONTROL** 8](#_Toc183785793)

[**2.1** **CONDICIONAL IF (SI) (EXPRESIÓN O CONDICIÓN)** 8](#_Toc183785794)

[**2.2** **CONDICIONALES IF and ELSE** 8](#_Toc183785795)

[**2.3** **CONDICIONAL SWITCH (VARIABLE O EXPRESIÓN A EVALUAR)** 8](#_Toc183785796)

[**3.** **CICLOS O LOOPS** 9](#_Toc183785797)

[**3.1** **CICLO WHILE (MIENTRAS)** 9](#_Toc183785798)

[**3.2** **CICLO FOR (PARA)** 9](#_Toc183785799)

[**3.3** **CICLO DO WHILE (HACER MIENTRAS)** 9](#_Toc183785800)

[**4.** **FUNCIONES Y ARREGLOS** 10](#_Toc183785801)

[**4.1** **CONCEPTO DE FUNCIÓN** 10](#_Toc183785802)

[**4.2** **FUNCIÓN CON PARÁMETROS** 10](#_Toc183785803)

[**4.3** **RETORNO (RETURN)** 10](#_Toc183785804)

[**5.** **ARREGLOS** 11](#_Toc183785805)

[**5.1** **¿Qué son?** 11](#_Toc183785806)

[**6.** **OBJETOS Y CLASES** 12](#_Toc183785807)

[**6.1** **¿Qué son los objetos?** 12](#_Toc183785808)

[**6.2** **¿Qué son las clases?** 13](#_Toc183785809)

[**6.3** **¿Qué son los métodos?** 14](#_Toc183785810)

[**7.** **CREACIÓN DEL JUEGO DEL AHORCADO EN REACT** 17](#_Toc183785811)

# **FUNDAMENTOS**

**CURSO:** Fundamentos de programación desde cero y desarrolla tu lógica para empezar tu camino como desarrollador. Para antes de comenzar, para poder ejecutar en la terminal solamente tener en cuenta 3 cosas:

1. En la terminal ejecutar el siguiente comando: npm install ts-node --save-dev
2. Configurar el script en el package.json del programa: "start": "ts-node app.ts"
3. Selecciona la ruta de tu proyecto cd \ruta\de\ejemplo>
4. Ejecuta tu archivo con el script definido: En lugar de node app.ts, ejecuta: npm start
5. Para ejecutar en simultánea el programa, ingresar el comando: npm run dev:watch

## **PALABRAS CLAVE PARA EMPEZAR**

* Clase
* Lenguaje de alto nivel
* Intérprete
* Compilador
* Objeto

Antes de empezar, instalar:

* NodeJS
* Visual Studio Code

Existen dos tipos de programación:

* **Lenguajes de bajo nivel:** (1,0) encendido y apagado (Binario).
* **Lenguajes de alto nivel:** Es un lenguaje que es humanamente legible y cercano a como los seres humanos hablamos.

## **ALGORITMO**

Es un proceso para resolver un problema. Por lo general es ordenado y secuencial.

## **VARIABLE**

Es el espacio en memoria donde podemos almacenar datos, y estos datos pueden cambiar en un futuro.

## **PALABRA RESERVADA**

Es una palabra que no podemos utilizar fuera de ese contexto.

## **LENGUAJE DE PROGRAMACIÓN A USAR: JavaScript**

Vamos a usar la palabra **let** para declarar variables en JavaScript, también podemos usar **Var**.

El punto y coma dentro de una línea de comando significa que hasta ahí termina la instrucción.

El doble slash **//** en programación se conoce como comentario, y esta no es parte que afecta el código como tal sino más bien como una guía que permita ayudar al codificador en lo que está desarrollando.

## **Console.log(10);**

Este comando permite imprimir el valor en un espacio de la computadora, en este caso 10.

## **CAMELCASE**

El camelcase es una denotación en la programación que indica que toda variable siempre debe comenzar en minúscula seguido de una letra mayúscula. Ejemplo:

**Let** numberOfCars = 10;

Terminal , escribimos node y se nos desplegará una línea parecida a la del navegador:

>

Y dentro de allí podríamos escribir lo mismo lo que escribimos en el navegador:

* **let** numberOfCars = 10;

< Undefinied

* numberOfCars;

<10

* numberOfCars + 1;<11
* numberOfCars = numberOfCars+1;

<11

* numberOfCars;

<11

Visto dentro de la terminal , algo como esto:

Texto

Descripción generada automáticamente

## **DEPENDENCIA**

Una dependencia es algo que su aplicación va a necesitar para funcionar.

## **TIPOS DE DATOS Y FLUJO DE CONTROL**

### **Tipos de variables**

* Números
* Booleanos
* Strings / Cadenas de caracteres
* Estructuras de control de flujo
* if
* if else
* else if
* ciclos o loops
* While
* Do While
* For

## **RECOMENDACIÓN A LA HORA DE NOMBRAR UNA VARIABLE**

1. Para nombrar variables, se recomienda que los nombres de las variables y constantes deben de tener un nombre significativo a lo que almacenaran.
2. Puedes usar letras, números, y guion bajo para definir tus variables y constantes.

**EXCEPCIÓN:** NO PUEDES INICIAR EL NOMBRE CON UN NÚMERO

**let** Nombre1; 🡪 es válido

**let** nombre\_1; 🡪 es válido

**let** \_nombre1; 🡪 es válido

**let** \_; 🡪 es válido

**let** 1nombre; 🡪 No es válido

**let** nombre@; 🡪 No es válido

1. No pueden contener caracteres especiales @ ( ^ % ‘ # ¡ á é í ó ú ñ

## **STRINGS – CADENA DE CARACTÉRES**

Almacenan palabras, frases, cadenas de letras, símbolos, etc.…

Ejemplo:

**let** nombre = “Fernando Herrera”;

**let** nombre = “Claudia Soto”;

## **BOOLEANS – BOOLEANOS**

Son variables que almacenan únicamente 2 estados:

* Verdadero (True)
* Falso (False)

Ejemplo:

**let** activo = true;

**RECOMENDACIÓN:** El nombre de una variable de tipo booleana debe leerse en positivo.

**cd = change directory**

# **ESTRUCTURAS DE CONTROL**

Nos permiten poder decirle a la computadora qué acción tomar y qué camino seguir dependiendo de condiciones y/o decisiones.

## **CONDICIONAL IF (SI) (EXPRESIÓN O CONDICIÓN)**

El condicional IF es una expresión que me permite establecer cuando una acción es verdadera y cuando es falsa. Es decir, si una acción no se cumple, entonces que se cumpla la siguiente.

## **CONDICIONALES IF and ELSE**

El if y el else se pueden anidar, es decir una condición if, podemos realizar otro if else y así sucesivamente sin límite.

## **CONDICIONAL SWITCH (VARIABLE O EXPRESIÓN A EVALUAR)**

El condicional Switch permite realizar múltiples decisiones de forma limpia y eficiente.

Ejemplo:

**Switch** **()** {

**Case 1:** ‘Hoy es Lunes’;

**Break;**

**Case 2:** ‘Hoy es Martes’;

**Break;**

**Case 3:** ‘Hoy es Miércoles’;

**Break;**

}

**Default:** ‘Hoy no es ni Lunes, ni Martes, ni Miércoles’;

# **CICLOS O LOOPS**

Nos permite ejecutar las mismas líneas de código una y otra vez de forma consecutiva.

## **CICLO WHILE (MIENTRAS)**

**While (CONDICIÓN) {**

**//** instrucciones

**}**

Este ciclo es muy útil, cuando no conocemos exactamente la cantidad de veces que necesitamos ejecutar el código.

## **CICLO FOR (PARA)**

Es muy útil cuando sabemos la cantidad de veces que necesitamos ejecutar el código.

Ejemplo:

**for (let** variable = 1; variable <=10; variable++) **{**

// Instrucciones

**}**

## **CICLO DO WHILE (HACER MIENTRAS)**

El ciclo do While o hacer mientras, es un ciclo que primero ejecuta una acción y después la valida si es correcta o no para finalmente imprimirla en pantalla.

Ejemplo:

Aquí se presenta la estructura del do While:

**do {**

**//** Serie de instrucciones o código

**}** While (Condición **===** true) **{**

**//** Valida si la condición es verdadera

**}**

**//** imprime el resultado.

# **FUNCIONES Y ARREGLOS**

En esta sección seguiremos introduciendo conceptos nuevos que son importantes en todo programa, como son:

1. **Arreglos**
2. **Funciones**
3. **Importaciones**
4. **Exportaciones**
5. **Reutilización de código**

Lo que veremos aquí, forma las bases del principio DRY (Don't repeat yourself) (No te repitas a ti mismo).

## **CONCEPTO DE FUNCIÓN**

Una función es un conjunto de líneas de código que realizan una tarea específica y pueden retornar algo.

Ejemplo:

**function** Saludar**() {**

**//** Instrucciones

**}**

## **FUNCIÓN CON PARÁMETROS**

Una función que adquiere o lleva un parámetro, es simplemente un valor que se le envía a la función. Ejemplo:

**function** Saludar**( Nombre ) {**

**//** Instrucciones

**}**

## **RETORNO (RETURN)**

Dentro de la declaración de una función también encontramos otra palabra clave llamada **return**, y lo que esta hace es retornar una función, es decir, es lo que obtendremos cuando la ejecución de la función termine.

# **ARREGLOS**

## **¿Qué son?**

Son un conjunto de cosas similares. Supongamos un canasto de flores, esa variable flores va a representar un arreglo el cual en su interior lo que se van a almacenar son flores de distinto tipo. Ejemplo:

**let** flores = **[ ];**

Este primer arreglo representaría un conjunto vacío ya que no hay nada en su interior. Ahora dentro de ese mismo arreglo vamos a añadirle 3 distintos tipos de flores:

**let** flores = **[“ROSA”, “GIRASOL”, “LIRIO”];**

Ahora vamos a seleccionar la flor rosa, para ello vamos a indicar el elemento en la posición 0 de esta manera:

**let** flores = **[ 0 ]; //** ROSA

# **OBJETOS Y CLASES**

## **¿Qué son los objetos?**

Los objetos en la programación es aquella programación suponiendo que nuestros objetos son abstracciones de cosas que existen en la vida real. Un ejemplo podría ser un auto ya que es un objeto el cual posee marca, color, tamaño. Computador también es un objeto ya que se compone de hardware, software, etc.

Esta es una sección sumamente importante que les ayudará mucho cuando estudien la programación orientada a objetos.

Puntualmente veremos:

* Objetos literales
* Clases
* Niveles de acceso
* Constructores
* Creación de instancias

Para tener en cuenta: **“Los métodos son funciones que se encuentran dentro de un objeto”.**

Ejemplo de la estructura de un objeto en TypeScript:

**let** Carro = **{**

**marca:** BMW,

**placa:** RXJ123,

**año:** 2024

**}**

Los objetos dentro de TypeScript también se pueden almacenar con base en arrays o arreglos y mediante el uso de ciclos, en esta ocasión vamos a usar el ciclo **for** de la siguiente manera:

***let*** fernando: Person = **{**

**name:** "Fernando",

**age:** 36,

**isActive:** true

**}**

export ***let*** melissa: Person = **{**

**name:** "Melissa",

**age:** 30,

    isActive: false

**}**

export ***let*** juan: Person = **{**

**name:** 'Juan',

**age:** 25,

**isActive:** false

**}**

export ***let*** people = **[**fernando, melissa, juan**]**;

for (***let*** i = 0; i < people.length; i++) {

***let*** person = people[i];

    console.log(person.name + '' + person.age);

}

## **¿Qué son las clases?**

Al igual que los objetos, pueden ser representaciones abstractas de cosas. Un ejemplo puede ser el siguiente:

**class** Galleta;

Ahora vamos a verlo en forma de código, similar a un objeto:

**class** Galleta **{**

**constructor (**botones, sabor**) {**

**this.**botones = botones;

**this.**sabor = sabor;

**this.**fechaCreacion = ‘Hoy’;

**}**

**}**

## **¿Qué son los métodos?**

Son funciones relacionadas a la clase. Y cualquier variable que sea una instancia de la clase, tendrá esas funciones. Para este caso veremos varias clases con métodos es decir que dentro de sus paréntesis van a incluirse algo de esa clase.

Ejemplo:

export class Car **{**

***static*** className = 'Carro';

***public*** *readonly* brand: string;

***public*** doors: number;

***public*** fuelTank: number;

***public*** isRunning: boolean;

***public*** type: string;

***private*** *readonly* createdAt: number;

**constructor**( brand: string, type: string ) **{**

**this**.brand = brand;

**this**.doors = 0;

**this**.fuelTank = 0;

**this**.isRunning = false;

**this**.type = type;

**this**.createdAt = 123456789;

**}**

***public*** turnOn() **{**

**if** ( this.isRunning ) **{**

            console.log('El carro ya estaba encendido.... se dañó el motor.');

***return***;

**}**

**if** ( this.fuelTank <= 0 ) **{**

            console.log('El carro no tiene gasolina');

***return***;

**}**

**this**.isRunning = true;

        console.log('El carro está encendido');

    }

*/\**

*gas: number*

*gas > 100, this.fuelTank = 100*

*gas tiene positivo*

*si ya está lleno, no sobre llenar*

*si el tanque es 50 y añado 20, el valor del fuelTank sería 70*

*si fuelTank es 90, y añado 50, fuelTank debería de ser 100*

*\*/*

**fillTank**( gas: number ) **{**

**if** ( gas <= 0 ) **{**

            console.log('El gas tiene que ser positivo');

***return***;

**}**

***let*** newFuelTank = this.fuelTank + gas;

**if** ( newFuelTank >= 100 ) {

            newFuelTank = 100;

**}**

        this.fuelTank = newFuelTank;

**}**

**}**

***let*** myMazda = new Car('Mazda','Sedan');

*// myMazda.brand;*

*// myMazda.type = 'Sedan';*

*// console.log( myMazda.createdAt ); // private*

console.log( myMazda );

myMazda.fillTank(50);

myMazda.fillTank(20);

*// myMazda.fillTank(50);*

myMazda.turnOn();

console.log( myMazda );

console.log( Car.className );

**Para tener en cuenta**

* **Método Público** – **public** : El mundo exterior, cualquier persona lo va a poder leer, modificar
* **Método Privado** – **private**: Se aseguran de que únicamente se van a poder llamar y/o cambiar si estamos dentro de la clase.

# **CREACIÓN DEL JUEGO DEL AHORCADO EN REACT**

Para la creación de este juego, primero abriremos nuestra terminal de comandos, y una vez situado dentro de la carpeta que hemos creado llamada **progra**, de esta manera:

cd "C:\Users\PC\OneDrive\Documentos\Udemy\1. Mi Ruta de Programación\1. Programación Para Principiantes Primeros Pasos\progra"

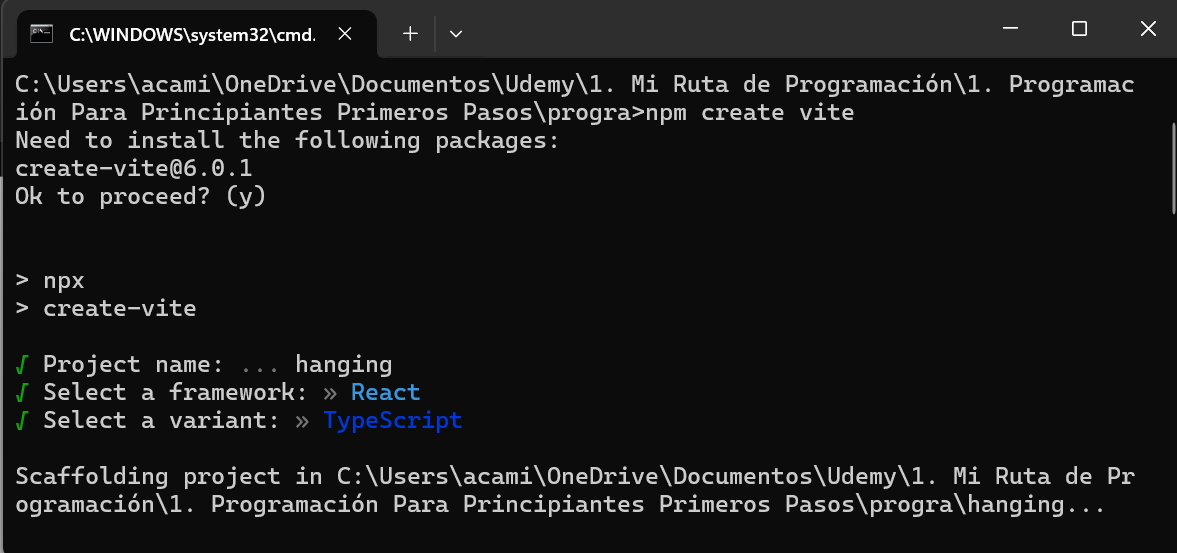
Procederemos a escribir el siguiente comando:

npm create vite

Este comando lo que hace es descargar las herramientas necesarias para poder trabajar con React dentro del Visual Studio Code.

Para esta ocasión nos pide que le asignemos un nombre al proyecto en **Project name**. En este caso el nombre va a ser **hanging**, que al español traduce como ahorcado.

Una vez ejecutado el comando, nos debe aparecer algo como esto, en la cual debemos seleccionar **React** como framework y **TypeScript** como una variante.

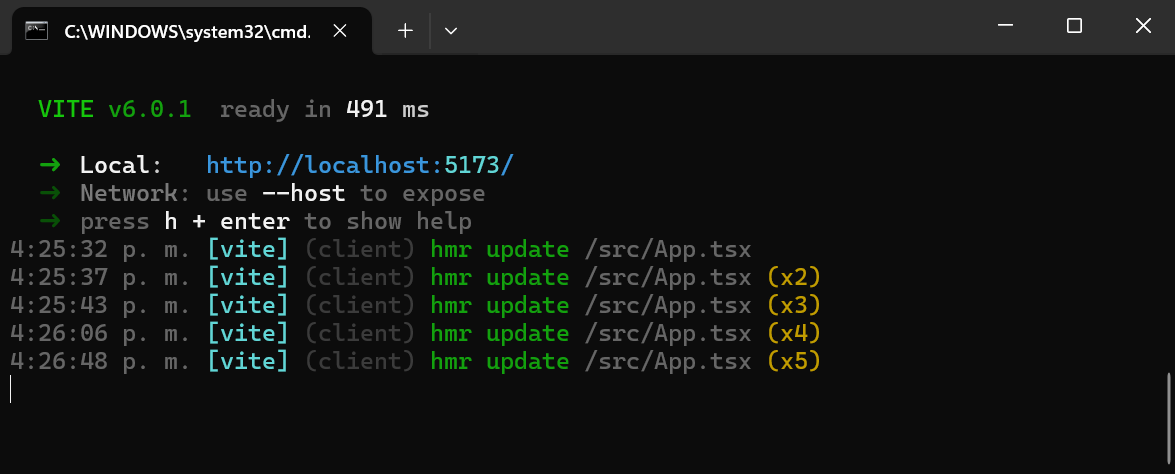


Una vez hecho esto, la consola nos va a pedir que ejecutemos lo siguiente: cd hanging, npm install y npm run dev.

Texto

Descripción generada automáticamente

Una vez ya de haber ejecutado los 3 comandos, nos aparecerá en la terminal la versión de VITE en color verde junto con una ruta de dirección localhost, a lo cual daremos clic en copiar y en nuestro navegador la estableceremos.



Interfaz de usuario gráfica, Aplicación

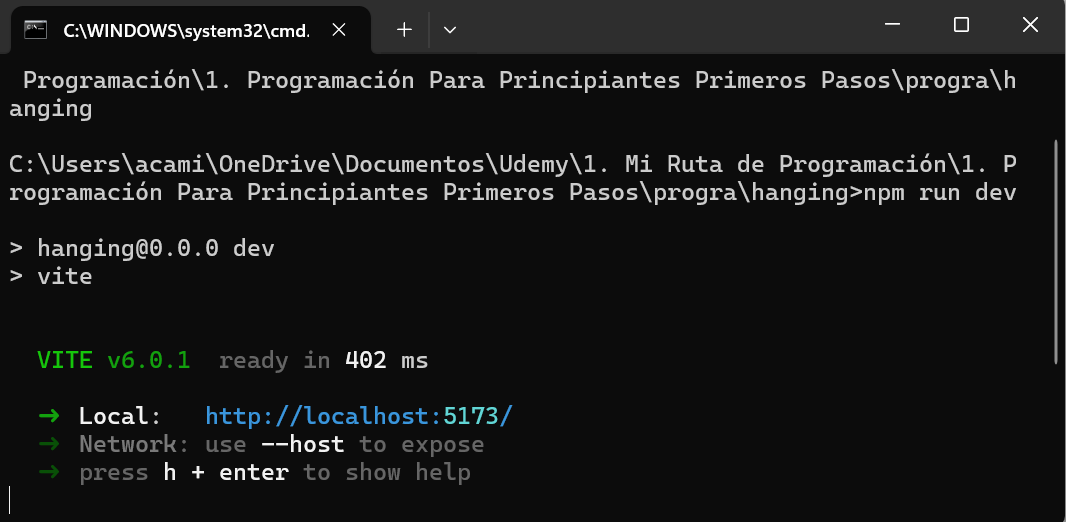
Descripción generada automáticamente

**TIP IMPORTANTE:** Si queremos volver a abrir el servidor una vez lo hayamos cerrado, sigamos los siguientes pasos:

1. Abre terminal y nos ubicaremos en el directorio correcto, en este caso:

C:\Users\acami\OneDrive\Documentos\Udemy\1. Mi Ruta de Programación\1. Programación Para Principiantes Primeros Pasos\progra\hanging

1. Una vez situado en el directorio, ejecutaremos el comando: npm run dev. Con este comando, no será necesario volver a descargar las dependencias y todo lo demás desde cero como se hizo anteriormente.
2. Una vez hecho esto, tendremos nuevamente en nuestra terminal la misma dirección ip asignada y ya con esto podremos seguir trabajando en nuestro proyecto sin perder los cambios.



Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente