

```
<!--Estudio Shonos-->
```

TypeScript {

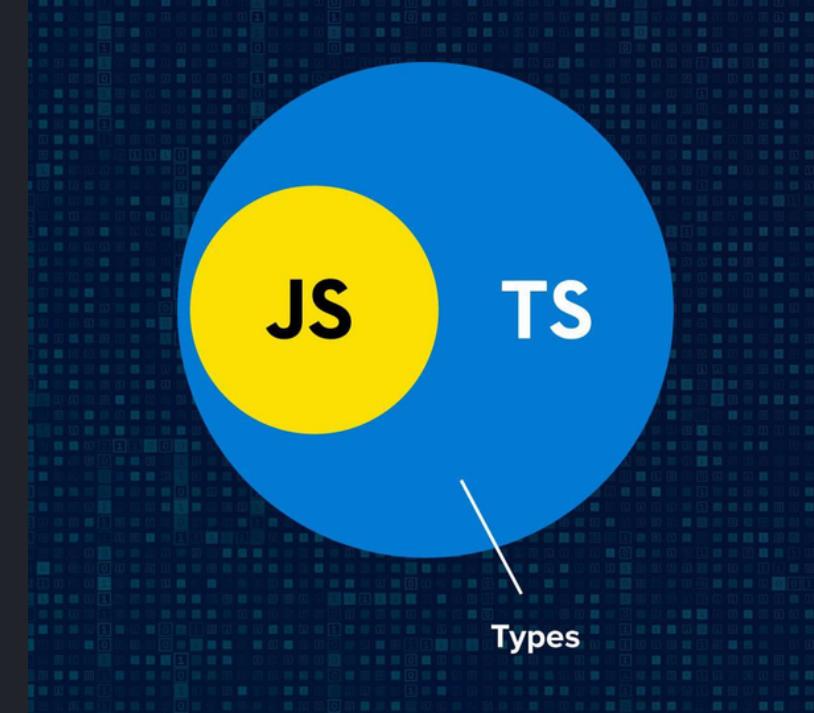
```
<Por="Dexiree Sequera 1152308"/>
```

```
<Por="Angely Pino 1152315"/>
```

```
<Por="Nicole Angarita 1152335"/>
```

```
<Por="Fabio Guerrero 1152319"/>
```

```
}
```



Contenidos

- 00 Introducción
- 01 Origen
- 02 Versión
- 03 Ubicación en Ranking
- 04 Conceptos de clases y
objetos
- 05 Conceptos contenedores
- 06 Herencia y Polimorfismo

Introducción {

TypeScript es un lenguaje de programación de código abierto desarrollado por Microsoft. Es un superconjunto de JavaScript que agrega características estáticas de tipos a la sintaxis de JavaScript. Esto significa que se pueden definir y utilizar tipos para variables, parámetros de funciones, propiedades de objetos, etc., lo que permite detectar errores comunes durante la fase de desarrollo.

// Beneficios

- Detección de errores en tiempo de compilación.
- Autocompletado y sugerencias en el editor.
- Mejor mantenibilidad y escalabilidad del código.
- Reutilización de código existente.

}

Diferencia de Typescript y Javascript

```
1 let ejemplo = 'Hola Compañeros'  
2  
3 console.log(typeof ejemplo)  
4  
5 ejemplo = 2
```

Javascript

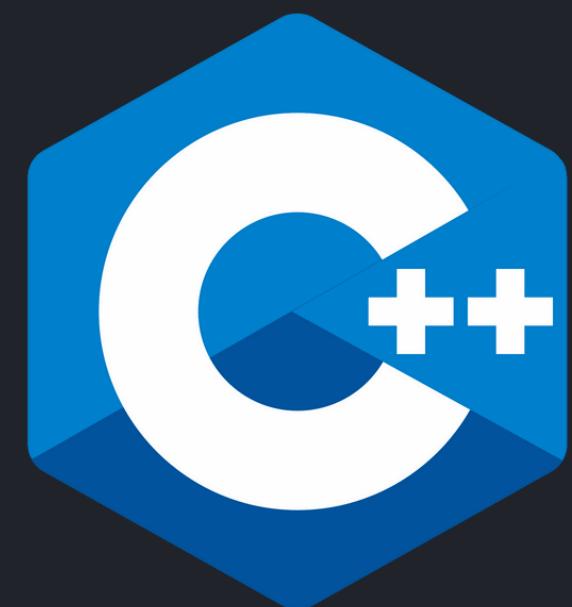
Typescript

```
1 let ejemplo: string = 'Hola Compañeros';  
2 console.log(typeof ejemplo); // Output: string  
3  
4 ejemplo = 2
```

}

Tipado estático y dinámico

Estático: la comprobación de tipificación se realiza durante la compilación, y no durante la ejecución. Ejemplos de lenguajes que usan tipado estático son C, C++, Java y Haskell. Comparado con el tipado dinámico, el estático permite que los errores de tipificación sean detectados antes, y que la ejecución del programa sea más eficiente y segura.



}

Tipado estático y dinámico

Dinámico: la comprobación de tipificación se realiza durante su ejecución en vez de durante la compilación. Ejemplos de lenguajes que usan tipado dinámico son [Perl](#), [Python](#) y [Lisp](#). El tipado dinámico es más flexible (debido a las limitaciones teóricas de la [decidibilidad](#) de ciertos problemas de análisis de programas estáticos, que impiden el mismo nivel de flexibilidad que se consigue con el tipado dinámico), a pesar de ejecutarse más lentamente y ser más propensos a contener errores de programación.

Perl



Origen {

TypeScript surgió en el año 2012 con la finalidad de solventar las deficiencias del lenguaje de programación JavaScript, haciendo de él una herramienta más robusta. Al respecto, podemos decir que la intención de Microsoft fue fortalecer este importante lenguaje.



Anders Hejlsberg

}

Ultima versión {

La última versión de TypeScript, la 4.5, fue lanzada públicamente el 27 de octubre de 2021, y trajo consigo varias mejoras y nuevas características para el lenguaje.

01

Revisión continua del código.

02

Optimizar las Apis de proveedores.

03

Corregir bugs.

04

Optimizar el rendimiento general.

05

Organizar al equipo.

06

Distribuir la faena por bloques.

07

Subir los cambios a la vez.

08

Limpiar la base de datos.

}

Ranking {

#40

Posición

según [TIOBE](#)

#01

Posición
según [NationalDeveloper](#)

#05

Posición
según [StackOverflow](#)

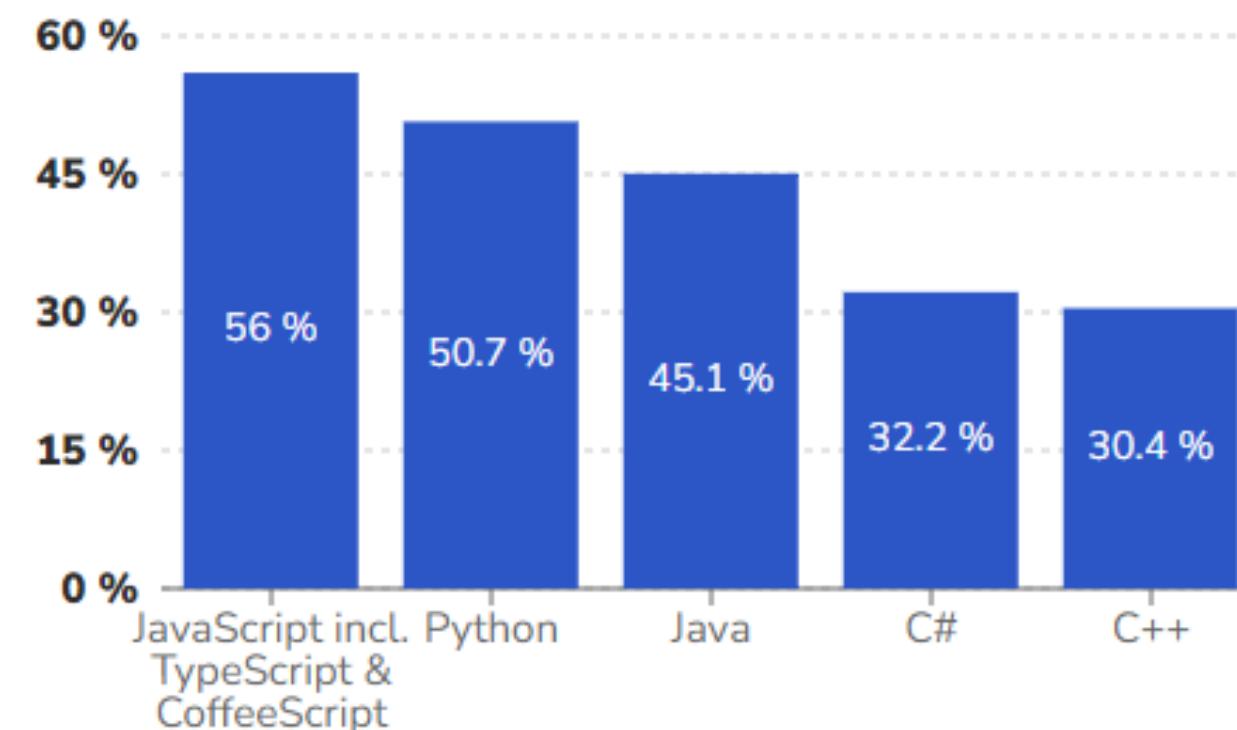
/*TIOBE

Es una empresa holandesa dedicada al análisis de código. Según ellos, cada día analizan más de 1.000 millones de líneas de código de desarrollos públicos y privados de empresas de todo tipo.

*/

NationalDeveloper

Top 5 programming languages used by developers



■ global (n=12.662)

IDATA

}

Stack Over Flow

Ranking {

Posición
según [TIOBE](#)

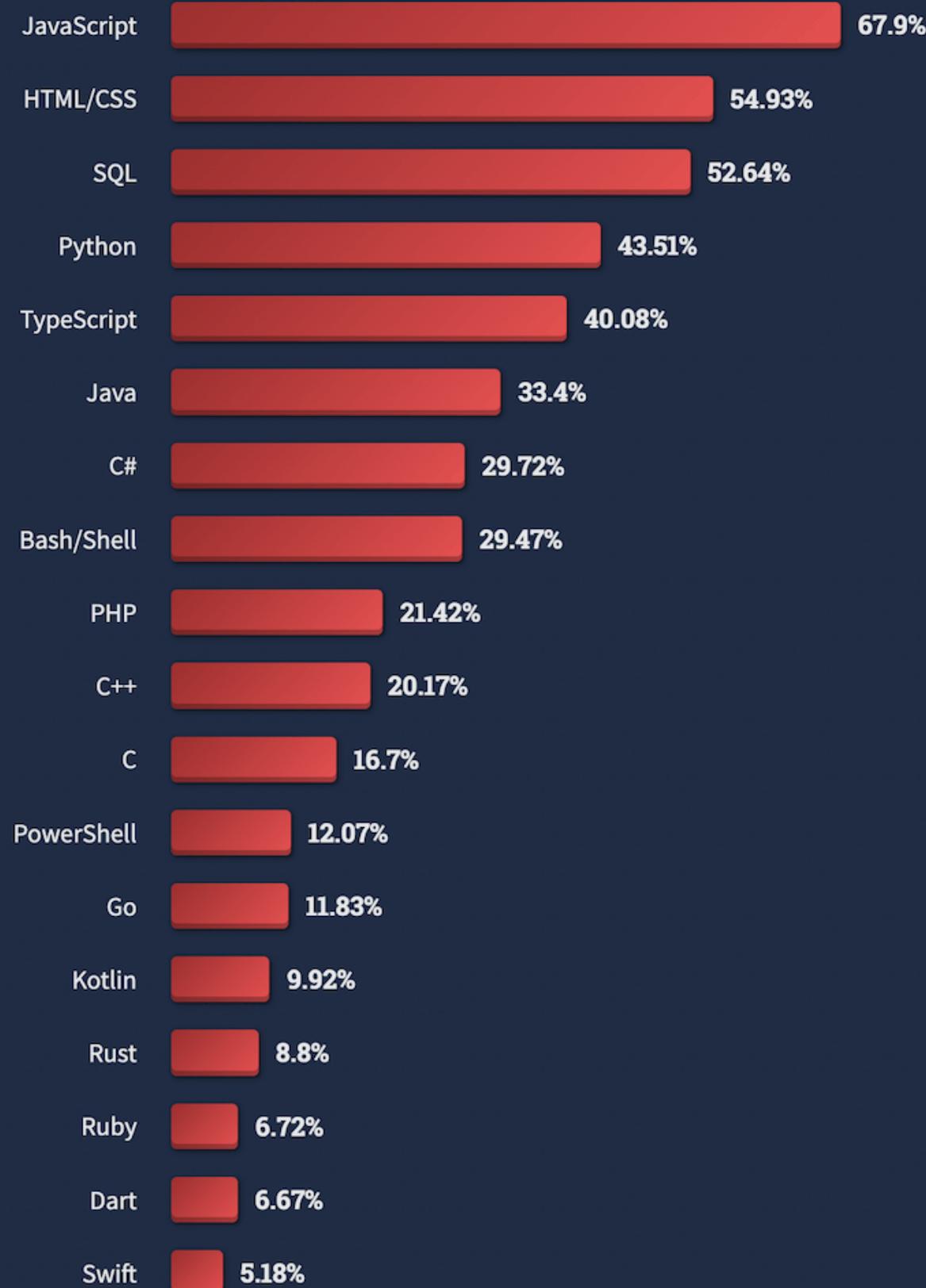
#40

Posición
según [NationalDeveloper](#)

#01

Posición
según [StackOverflow](#)

#05



}

TypeScript sintaxis {

Como TypeScript es un superconjunto de JavaScript, también incluye todas las características de ECMAScript, como variables var, let y const, objetos literales, funciones de flecha, clases, módulos, promesas, async/await, destrucción y más.

/*Tipos de datos
TypeScript incluye tipos de datos básicos como string, number, boolean, array, tuple, enum, any, void, null y undefined.

*/

```
let str: string = 'Hello, World!';  
let num: number = 42;  
let bool: boolean = true;  
let arr: number[] = [1, 2, 3];  
let tup: [string, number] = ['one', 1];  
enum Color { Red, Green, Blue }  
let anyValue: any = 'I can be anything!';  
let nothing: void = undefined;  
let notHere: null = null;  
let missing: undefined = undefined;
```

}

TypeScript sintaxis {

Como TypeScript es un superconjunto de JavaScript, también incluye todas las características de ECMAScript, como variables var, let y const, objetos literales, funciones de flecha, clases, módulos, promesas, async/await, destrucción y más.

/*Funciones

Las funciones en TypeScript pueden tener tipos de datos para sus parámetros y valores de retorno.
*/

```
function greet(name: string): string {
    return `Hello, ${name}!`;
}

greet('Alice'); // Hello, Alice!
```

}

TypeScript sintaxis {

Como TypeScript es un superconjunto de JavaScript, también incluye todas las características de ECMAScript, como variables var, let y const, objetos literales, funciones de flecha, clases, módulos, promesas, async/await, destrucción y más.

/*Interfaces

Las interfaces en TypeScript definen contratos de forma para objetos y clases, lo que permite la verificación de tipos.

*/

```
interface Person {
    firstName: string;
    lastName: string;
}

function fullName(person: Person) {
    return `${person.firstName} ${person.lastName}`;
}

const john: Person = { firstName: 'John', lastName: 'Doe' };
console.log(fullName(john)); // John Doe
```

}

TypeScript sintaxis {

Como TypeScript es un superconjunto de JavaScript, también incluye todas las características de ECMAScript, como variables var, let y const, objetos literales, funciones de flecha, clases, módulos, promesas, async/await, destrucción y más.

/*Clases

Las clases en TypeScript incluyen soporte para propiedades, métodos, modificadores de acceso, herencia y polimorfismo.

*/

```
class Animal {
    private name: string;

    constructor(name: string) {
        this.name = name;
    }

    public makeSound(): void {
        console.log('Some generic sound');
    }
}

class Dog extends Animal {
    constructor(name: string) {
        super(name);
    }

    public makeSound(): void {
        console.log('Woof!');
    }
}

const dog = new Dog('Buddy');
dog.makeSound(); // Woof!
```

}

TypeScript sintaxis {

Como TypeScript es un superconjunto de JavaScript, también incluye todas las características de ECMAScript, como variables var, let y const, objetos literales, funciones de flecha, clases, módulos, promesas, async/await, destrucción y más.

/*Genéricos

Los genéricos en TypeScript permiten crear componentes reutilizables que funcionan con varios tipos de datos.

*/

```
function identity<T>(arg: T): T {
    return arg;
}

console.log(identity<string>('Hello')) // Hello
console.log(identity<number>(42)) // 42
```

}

Conceptos de clases y objetos {

En TypeScript, una clase es una plantilla para crear objetos. Un objeto es una instancia de una clase y tiene propiedades y métodos.

/*2.1 Métodos

Los métodos son funciones que pertenecen a una clase y realizan acciones específicas. Se definen dentro de la clase utilizando la palabra clave `function`.

*/

```
class Persona {  
    saludar() {  
        console.log('¡Hola!');  
    }  
}
```

}

Conceptos de clases y objetos {

En TypeScript, una clase es una plantilla para crear objetos. Un objeto es una instancia de una clase y tiene propiedades y métodos.

/*2.2 Propiedades

Las propiedades son variables que se encuentran dentro de una clase y representan características o atributos de la clase. Se definen usando nombres de variables y tipos de datos.

*/

```
class Persona {  
    nombre: string;  
    edad: number;  
}
```

}

Conceptos de clases y objetos {

En TypeScript, una clase es una plantilla para crear objetos. Un objeto es una instancia de una clase y tiene propiedades y métodos.

/*2.3 Encapsulamiento

El encapsulamiento es un principio de la programación orientada a objetos que consiste en ocultar los detalles internos de una clase y exponer solo lo necesario. En TypeScript, esto se logra usando modificadores de acceso como `public`, `private` y `protected`.

*/

```
class Persona {
    private nombre: string;
    private edad: number;

    constructor(nombre: string, edad: number) {
        this.nombre = nombre;
        this.edad = edad;
    }

    public presentarse() {
        console.log(`Soy ${this.nombre} y tengo ${this.edad} años.`);
    }
}
```

}

Asociación {

La asociación es una relación entre clases en la que una clase utiliza a otra sin depender completamente de ella.

La relación se establece a través de una referencia en una clase a otra.

```
class Motor {  
    // ...  
}  
  
class Auto {  
    private motor: Motor;  
  
    constructor(motor: Motor) {  
        this.motor = motor;  
    }  
}
```

}

Agregación {

La agregación es una forma más fuerte de asociación en la que una clase actúa como un contenedor para otra clase. A diferencia de la asociación, la clase contenedora es responsable de la creación y destrucción de los objetos contenidos.

```
class Rueda {  
    // ...  
}  
  
class Auto {  
    private ruedas: Rueda[];  
  
    constructor() {  
        this.ruedas = [new Rueda(), new Rueda(), new Rueda(), new Ru  
eda()];  
    }  
}
```

}

Composición {

La composición es una relación aún más fuerte que la agregación en la que una clase no puede existir sin la otra. Si la clase contenedora se destruye, también lo hacen las clases contenidas.

```
class Motor {  
    // ...  
}  
  
class Auto {  
    private motor: Motor;  
  
    constructor() {  
        this.motor = new Motor();  
    }  
}
```

}

Herencia {

La herencia es un mecanismo que permite a una clase heredar las propiedades y métodos de otra clase. La clase que hereda se llama subclase y la clase de la que hereda se llama superclase. En TypeScript, la herencia se implementa utilizando la palabra clave `extends`.

```
class Vehiculo {
    protected velocidad: number;

    constructor() {
        this.velocidad = 0;
    }

    acelerar() {
        this.velocidad += 10;
    }
}

class Auto extends Vehiculo {
    encenderRadio() {
        console.log('Radio encendida');
    }
}
```

}

Polimorfismo {

El polimorfismo es un principio que permite a una clase tener múltiples formas. En TypeScript, esto se logra a través de la herencia y la implementación de interfaces.

```
interface Animal {
  hacerSonido(): void;
}

class Perro implements Animal {
  hacerSonido() {
    console.log('Guau');
  }
}

class Gato implements Animal {
  hacerSonido() {
    console.log('Miau');
  }
}

function escucharSonido(animal: Animal) {
  animal.hacerSonido();
}

const perro = new Perro();
const gato = new Gato();

escucharSonido(perro); // Guau
escucharSonido(gato); // Miau
```

{}

