

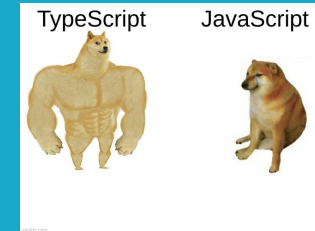


TYPESCRIPT



Los Mavericks Tecnológicos – Valentina Adelsflugel, Brandon Schiffer, Tobías Serpa

- TypeScript fue creado por Microsoft y fue anunciado por primera vez el 1 de octubre de 2012
- Motivación principal: Mejorar el desarrollo de aplicaciones con JavaScript a gran escala
- Superset de JavaScript



ORIGEN DE TYPESCRIPT

PARA QUÉ USAR TS?

- Desarrollo Web
- Desarrollo “Server-side” (backend) con Node.js
- Aplicaciones Móviles con React

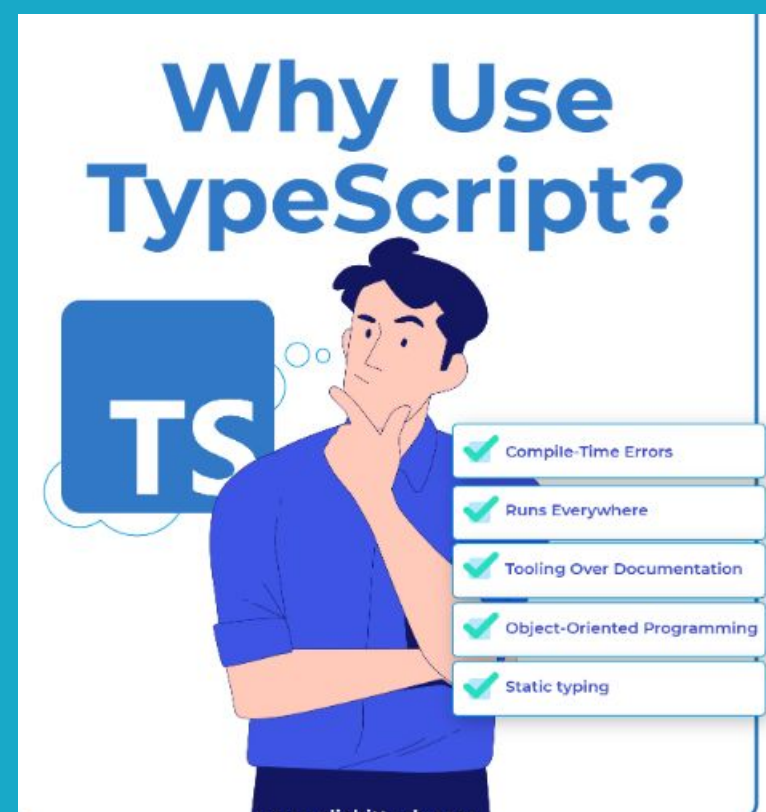
PARA QUÉ NO USAR TS?

- Proyectos de pequeña escala
- Data science

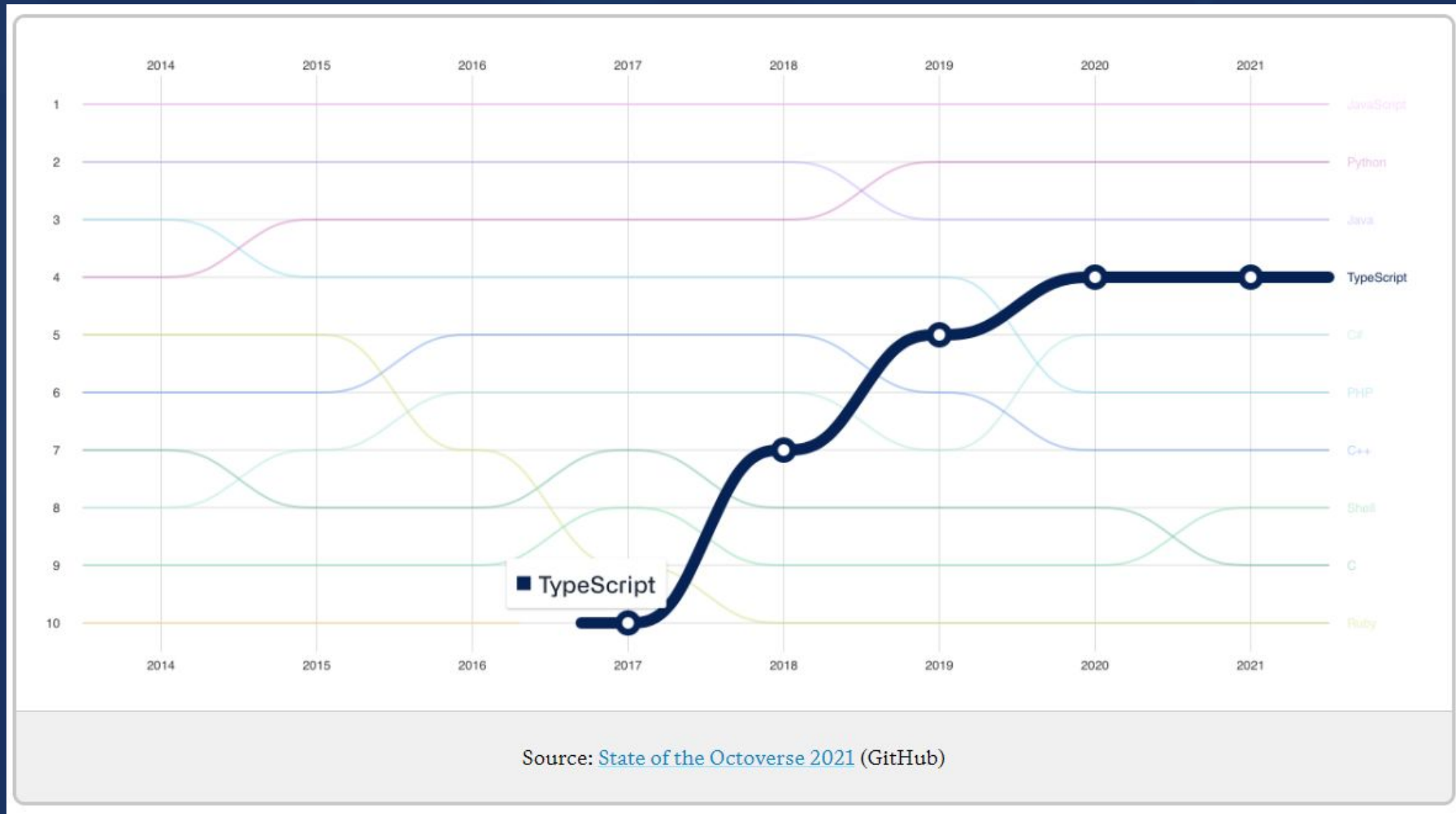


CASOS REALES DE USO DE TS

- Microsoft Visual Studio Code
- Slack
- Asana

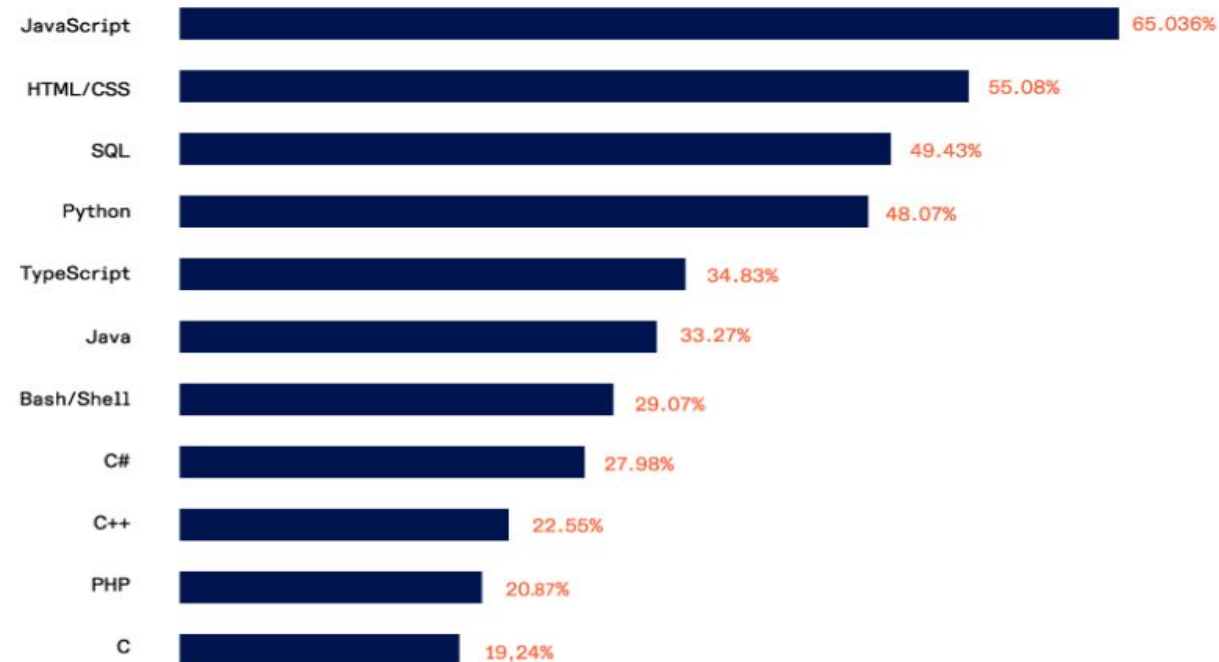


EVOLUCIÓN DEL USO DEL LENGUAJE



ESTADÍSTICAS DEL USO DEL LENGUAJE EN LA ACTUALIDAD

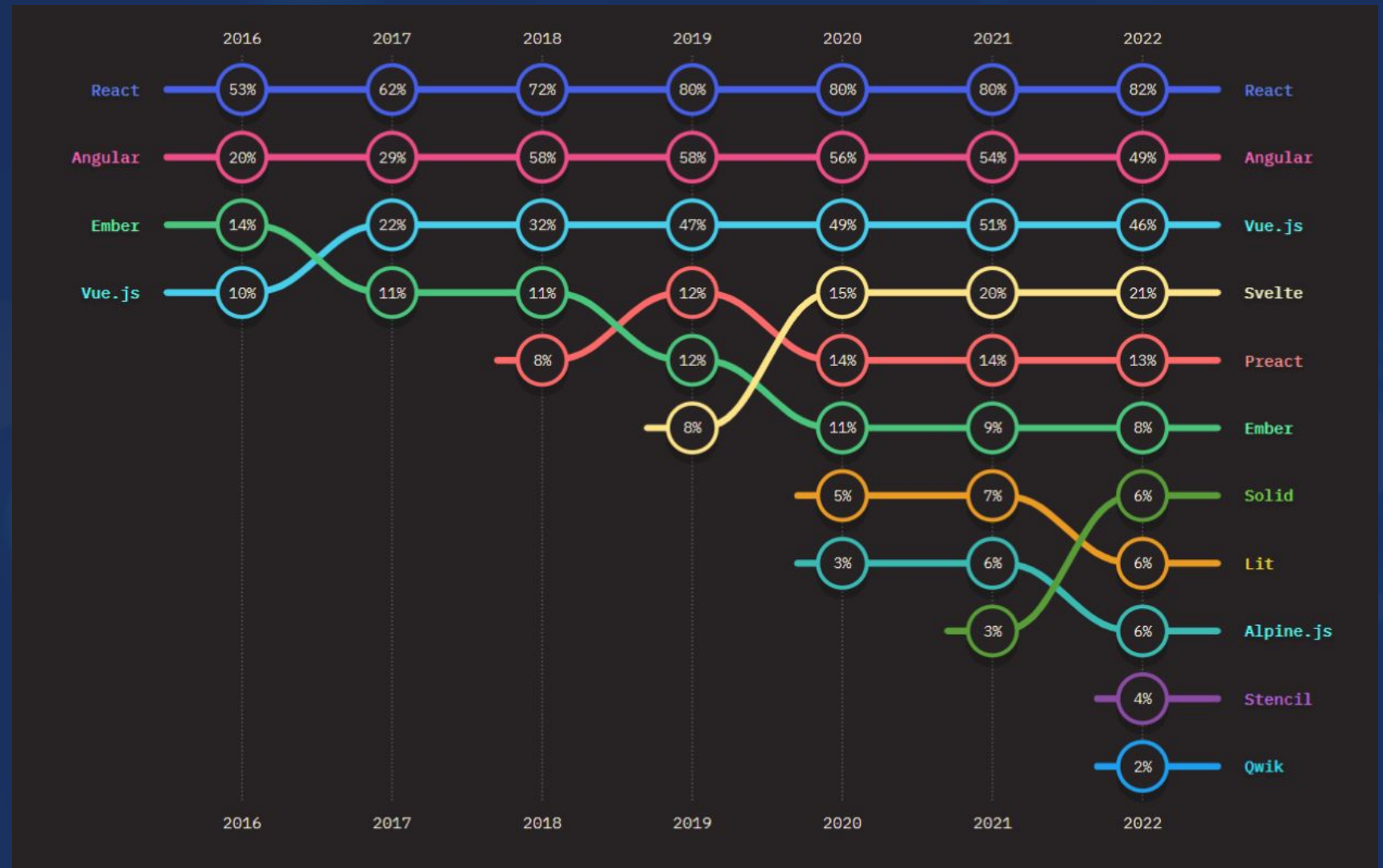
Most used programming languages among developers



Source: Statista

Most used programming languages

EVOLUCIÓN DEL USO DE FRAMEWORKS JS/TS



ESTADÍSTICAS DEL USO DE FRAMEWORKS EN LA ACTUALIDAD

Most popular web frameworks



Node.js

- Node.js no es un framework.
- Node.js es un entorno de ejecución de JavaScript que te permite ejecutarlo fuera del navegador. Node.js mejora los tiempos de procesamiento y es fácil de configurar.
- NPM

Más info: <https://builtin.com/software-engineering-perspectives/nodejs>

INFLUENCIA DE TYPESCRIPT

- Flow: sistema de tipos estático para JavaScript desarrollado por Facebook
- Babel: herramienta utilizada para transpilar el código JavaScript moderno a versiones compatibles con navegadores más antiguos
- Angular: framework de JavaScript desarrollado por Google para construir aplicaciones web

vs Flow

- Compatibilidad con Frameworks
- Decoradores
- Sintaxis

TypeScript vs Flow	
TypeScript	Flow
<ol style="list-style-type: none">1. Uses static typing with type annotations2. Can be easily integrated with existing JavaScript projects3. Has a larger community and is more widely used4. Has a steeper learning curve due to the complex type system and additional language features <p>www.freshersnow.com</p>	<ol style="list-style-type: none">1. Uses static typing with type inference2. Flow requires more effort to integrate with existing JavaScript projects3. Has a smaller community compared to TypeScript4. Has a relatively lower learning curve compared to TypeScript due to its simplicity <p>www.freshersnow.com</p>

```
// Flow Example
// @flow
function sum(a, b) {
  return a + b;
}

const result = sum(5, 10);
console.log(result);

// TypeScript Example
function sum(a: number, b: number): number {
  return a + b;
}

const result: number = sum(5, 10);
console.log(result);
```

Más info: <https://medium.com/geekculture/typescript-vs-flow-b4eb3778cf6b>
<https://www.freshersnow.com/typescript-vs-flow/>



INFLUENCIAS HACIA TYPESCRIPT APARTE DE JS

- C#: Sintaxis de declaración de tipos, la inferencia de tipos, las clases y la herencia
- Java: Conceptos de clases, interfaces y herencia
- EcmaScript: define la especificación de JavaScript

vs JavaScript

- Tipado
- Compilado
- Conceptos POO
- Genéricos, Decoradores, Módulos

Más info: <https://www.crehana.com/blog/transformacion-digital/que-es-typescript/>
<https://cynoteck.com/blog-post/typescript-vs-javascript-vs-ecmascript-know-the-difference/>

 JAVASCRIPT	VS	 TYPESCRIPT
1 It does not support optional parameters.		1 It supports optional parameters.
2 It is interpreted language that is why it highlights the errors at runtime.		2 It compiles the code and highlights errors during the development time.
3 JavaScript does not support modules.		3 TypeScript gives support for modules.
4 In this, number, string are the objects.		4 In this, number, string are the interfaces.
5 JavaScript does not support generics.		5 TypeScript supports generics.

Ts

```
export const Ayuda: Command = {
  name: "ayuda",
  description: "Muestra todos los comandos disponibles",
  type: ApplicationCommandType.ChatInput,
  run: async (client: Client, interaction: CommandInteraction) => {
    const Commands = await loadCommands();
    const commandsList = Commands.map((command) => command.name).join(", ");
    await interaction.followUp({
      ephemeral: true,
      content: `Estos son los comandos disponibles: ${commandsList}`
    });
  },
};
```

Js

```
const Ayuda = {
  name: "ayuda",
  description: "Muestra todos los comandos disponibles",
  type: ApplicationCommandType.ChatInput,
  run: async (client, interaction) => {
    const Commands = await loadCommands();
    const commandsList = Commands.map((command) => command.name).join(", ");
    await interaction.followUp({
      ephemeral: true,
      content: `Estos son los comandos disponibles: ${commandsList}`
    });
  },
};

export { Ayuda };
```

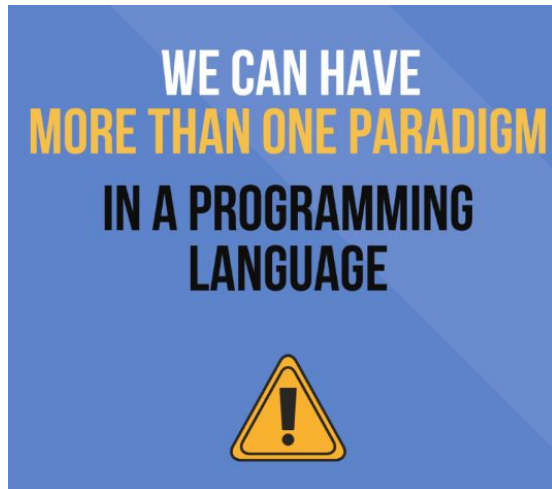


CARACTERÍSTICAS
BÁSICAS

EJEMPLIFICACIÓN
CON TP

PARADIGMAS DE PROGRAMACIÓN

- Múltiples paradigmas de programación
 - POO
 - Programación Funcional
 - Programación estructurada
 - Programación imperativa
 - Programación de script
 - Programación genérica



POO - CLASES E INTERFACES

- Clases
 - permiten definir objetos con propiedades y comportamientos
- Interfaces
 - Define la estructura y el contrato de un objeto
 - Establece un conjunto de propiedades y métodos que un objeto debe implementar para cumplir con dicha interfaz

EJEMPLOS INTEGRADORES

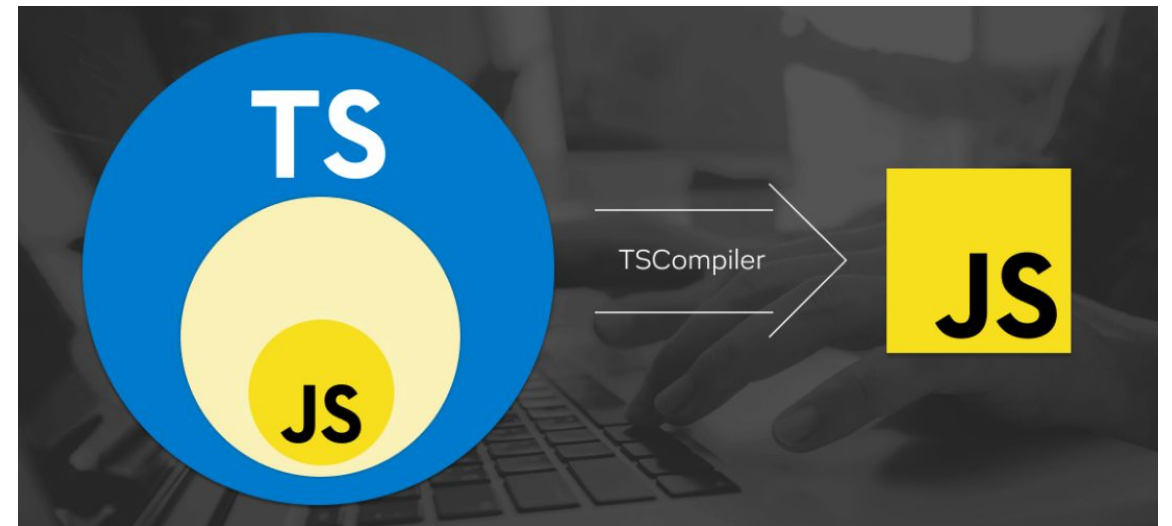
```
export interface Command extends ChatInputApplicationCommandData {  
  run: (client: Client, interaction: CommandInteraction) => void;  
}
```

```
export const Ayuda: Command = {  
  name: "ayuda",  
  description: "Muestra todos los comandos disponibles",  
  type: ApplicationCommandType.ChatInput,  
  run: async (client: Client, interaction: CommandInteraction) => {  
    const Commands = await loadCommands();  
    const commandsList = Commands.map((command) => command.name).join(", ");  
    await interaction.followUp({  
      ephemeral: true,  
      content: `Estos son los comandos disponibles: ${commandsList}`  
    });  
  },  
};
```

```
export class DatabaseConnection {  
  private static instance: DatabaseConnection;  
  private dataSource: DataSource;  
  private static dataSrcPromise: Promise<DataSource>;  
  private isConnected: boolean;  
  
  private constructor() {  
    this.dataSource = new DataSource(require("../config.json").database);  
    this.isConnected = false;  
    DatabaseConnection.dataSrcPromise = this.connect();  
  }  
  
  public static initializeDB() {  
    if (!DatabaseConnection.instance) {  
      DatabaseConnection.instance = new DatabaseConnection();  
    }  
  }  
  
  private connect(): Promise<DataSource> {  
    if (!this.isConnected) {  
      this.isConnected = true;  
      return this.dataSource.initialize();  
    }  
    return Promise.resolve(this.dataSource);  
  }  
}
```

COMPILADO

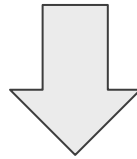
- Transformación de código fuente en código JS
- Proceso:
 - Archivo de configuración: “tsconfig.json”
 - Ejecución del compilador TSC
 - Análisis estático, verificación de coherencia de tipos y detección de errores
 - Transpilación a JS
 - Generación de archivo de salida



TIPADO ESTÁTICO E INFERENCIA DE TIPOS

- Tipado estático: Permite declarar y verificar tipos de variables, parámetros de función y valores de retorno en tiempo de compilación.
- Inferencia: Deducción automática de tipos en función de su contexto

```
const materiaOption = interaction.options.get("nombre");
```



```
const materiaOption: CommandInteractionOption<CacheType> | null = interaction.options.get("nombre");
```

CALLBACK, HIGH ORDER Y ARROW FUNCTION

- TypeScript y Oz: funciones son ciudadanas de primera clase
- High-Order: función que puede tomar una o más funciones como argumentos y/o devolver una función como resultado
- Callback: función que se pasa como argumento a otra función y se invoca dentro de esa función en un momento posterior, logrando la ejecución asíncrona
- Arrow Function: forma concisa de escribir funciones. Sintaxis para definir funciones anónimas

```
const correlativasFaltantes = correlativas.filter((correlativa) => !alumnoMaterias.includes(correlativa));
```

DECORADORES

- Permite agregar metadatos o modificar el comportamiento de clases, métodos, propiedades y otros elementos en tiempo de compilación
- Tipos: Clase, métodos y de propiedades y parámetros

Más info: <https://diegoboscan.com/decoradores-en-typescript/>

```
@Entity()
@Unique(['nombre'])
export class Carreras {
  @PrimaryColumn({ type: 'int', generated: true })
  id!: number;

  @Column({ type: 'varchar', length: 100 })
  nombre!: string;

  @Column({ type: 'float', nullable: true })
  duracion!: number;
}

@Entity()
export class Materia {
  @PrimaryColumn({ type: 'varchar', length: 10 })
  codigo!: string;

  @Column({ type: 'varchar', length: 100 })
  nombre!: string;

  @Column({ type: 'int', nullable: true })
  creditos!: number;

  @Column({ type: 'varchar', length: 200, nullable: true })
  correlativas!: string;

  @ManyToOne(() => Carreras, carreras => carreras.id)
  carrera!: Carreras;
}
```

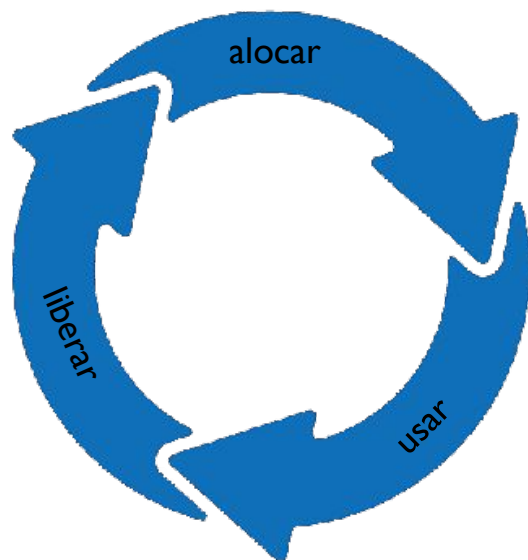



CARACTERÍSTICAS
AVANZADAS

EJEMPLIFICACIÓN
CON TP

MANEJO DE MEMORIA

- TS vs C: manejo implícito
- Garbage Collector

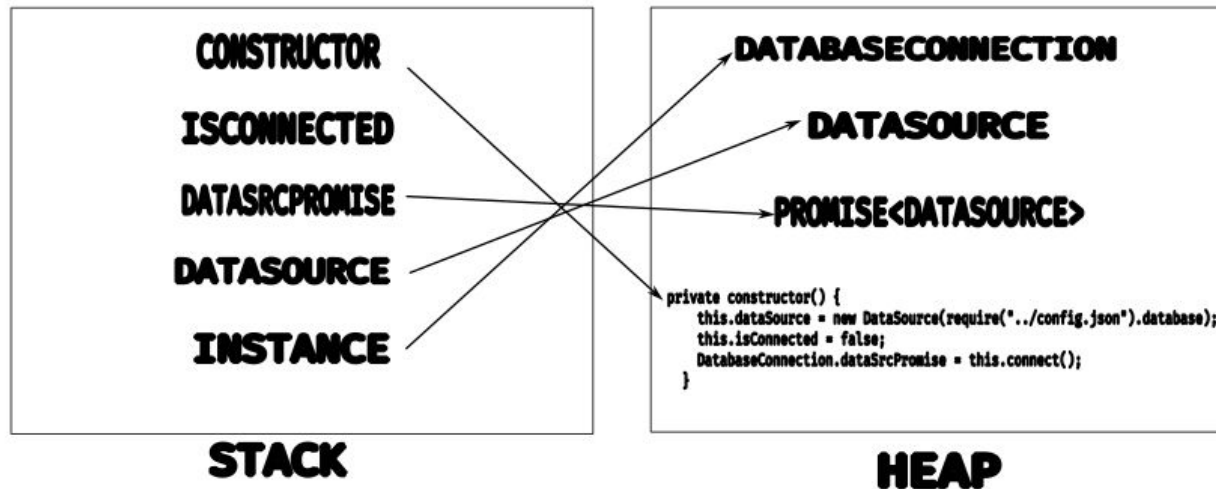


Stack	Heap
Primitive data types and references	Objects and functions
Size is known at compile time	Size is known at run time
Fixed memory allocated	No limit for object memory

Más info: <https://baldur.gitbook.io/js/js-principles/memory-management>

Un ejemplo...

```
private static instance: DatabaseConnection;  
private dataSource: DataSource;  
private static dataSrcPromise: Promise<DataSource>;  
private isConnected: boolean;  
  
private constructor() {  
  this.dataSource = new DataSource(require("../config.json").database);  
  this.isConnected = false;  
  DatabaseConnection.dataSrcPromise = this.connect();  
}
```



Concurrencia

- Un Proceso/Hilo.
- Asincronismo.
- Promesas.
- Async/Await.
- Callback.



```
public static loadDBData() {  
    let newPromise: Promise<DataSource> | undefined;  
    this.dataSrcPromise  
        .then(async (ds) => {  
            await loadCarreras(ds);  
            const newLocal = './src/data/INFORMATICA.csv';  
            await loadData(ds, newLocal);  
            await loadData(ds, './src/data/ELECTRONICA.csv');  
            await loadData(ds, './src/data/SISTEMAS.csv');  
            await loadMateriasPorCarrera();  
            newPromise = Promise.resolve(ds);  
        })  
        .finally(() => {  
            this.dataSrcPromise = newPromise!;  
        });  
}
```

```
export async function loadMateriasPorCarrera() {  
    let materias = (await DatabaseConnection.getAllMaterias());  
    materiasPorCarrerasDelAlumno = materias.filter((value, index, self) => self.indexOf(value) === index);  
}
```

Errores

Excepciones

```
export class DBError implements Error {  
  name: string;  
  message: string;  
  
  constructor(message: string) {  
    this.name = "DBError";  
    this.message = message;  
  }  
}
```

```
public static async getNombreMateriasPorCodigo(codigos: string[]): Promise<string[]> {  
  try {  
  } catch (error) {  
    console.error("Se produjo un error al obtener el código de la materia:", error);  
    throw new DBError("Se produjo un error al obtener el código de la materia: " + error);  
  }  
}
```

Custom Exceptions

A comparación de Java, no permite throws clauses, pero se puede lograr así, usando union types:

```
public static async getNombreMateriasPorCodigo(codigos: string[]): Promise<string[] | DBError> {
```

Entonces obligamos a que al llamar a esta función, se maneje el error

```
try {
  const ds = await this.dataSrcPromise;
  const materia = await ds.manager.findOne(Materia, { where: { codigo: codigoMateria } });

  if (materia && materia.correlativas) {
    return materia.correlativas.split("-");
  } else {
    console.log("No se encontraron correlativas para la materia con ese código.");
    return null;
  }
} catch (error) {
  console.error("Se produjo un error al obtener las correlativas de la materia:", error);
  return null;
}
```

Handling de Excepciones

REFERENCIAS

- **Página Oficial TS:** <https://www.typescriptlang.org/>
- **Programming Typescript:** <https://books-library.net/files/books-library.net-10132058Ts3U9.pdf>
- **Uso de TS:** <https://css-tricks.com/the-relevance-of-typescript-in-2022/>
- **Estadísticas**
 - <https://ordergroup.co/en-us/blog/best-web-frameworks-and-languages-to-use-in-2022/>
 - <https://www.lambdatest.com/blog/best-javascript-frameworks/7>
- **Información general:**
 - <https://www.computercareers.org/5-reasons-why-you-should-learn-typescript/>
 - <https://basarat.gitbook.io/typescript/getting-started/why-typescript>
 - <https://www.peerigon.com/en/why-use-typescript/>
 - <https://www.unir.net/ingenieria/revista/que-es-typescript/#:~:text=Tras%20dos%20a%C3%B1os%20de%20desarrollo,tanto%20en%20un%20nivel%20superior.>
 - <https://keepcoding.io/blog/typescript/>
 - <https://programamos.es/unidad-6-el-lenguaje-de-programacion-typescript/>
 - <https://www.freecodecamp.org/news/what-is-typescript/>