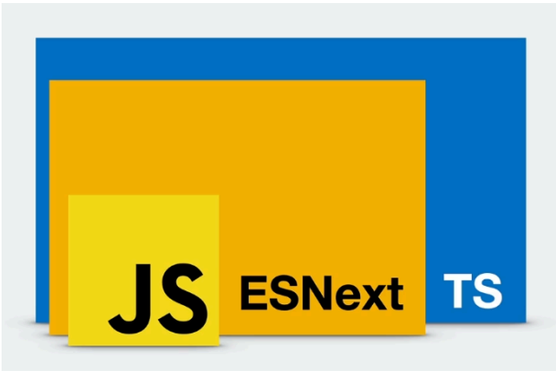
TypeScript: Tu completa guía y manual de mano.

Extensiones.

Extesión JSON as Code: https://marketplace.visualstudio.com/items?itemName=quicktype.quicktype

Sección 1.

1. TypeScript es un super set de JavaScript.



2. TypeScript es transpilado a JavaScript para los navegadores.

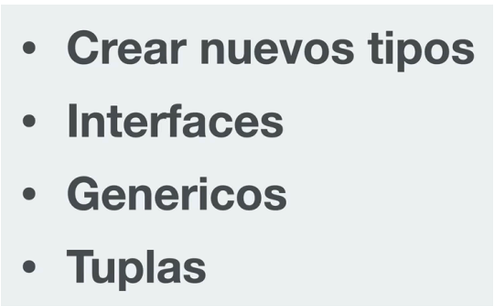
Sección 2.

TSConfig.json

```
tsc --init
```

Sección 3.

1. Hay más tipos de datos en TypeScript que en JavaScript.
    a. undefined es cuando no se le asigna ningún valor.
    b. null está vacío el valor.
    c. Symbol asigna un espacio en memoria único.
    d. De TypeScript.



2. Se recomienda asignar el tipo de dato.

```
const a:number = 10;
let b:number = 10;
```

3. Es buena práctica indicar el tipo de dato en las funciones.

```
function sayHello(msg:string) {
    console.log(msg);
```

```
}
```

4. En este caso, si no se declara el tipo, no genera un error al transformar el dato, pero no es correcto porque genera NaN (Not a Number).

```
(() => {
    let avengers:number = 10;

    console.log(avengers);

    const villians:number = 20;

    if(avengers < villians){
        console.log('Estamos en problemas!');
    } else{
        console.log('Nos salvamos!')
    }

    avengers = Number('55A');

    console.log({avengers}); //NaN.
})()
```

5. No se recomienda usar Any.
6. Casteo:

```
    let avenger: any = 123;
    console.log((avenger as string).charAt(0));
    console.log((<number>avenger).toFixed(0));
```

7. Tuplas son propias de TypeScript.
8. El tipo de dato Void se utiliza comúnmente para indicar que la función no retornará nada.
9. El tipo de dato Error se utiliza comúnmente para indicar que una función terminará con error.

Sección 3.

1. En las funciones existen argumentos obligatorios y opcionales.
    a. Para opcionales, se escriben con “?” y van al final.
    b. O también se pueden ingresar antes de los que se asignan por default.

```
(() => {


    const fullName = (firstName: string, lastName?: string, upper: boolean = false): string => {
        if (upper) {
            return `${firstName} ${lastName || '--'}`.toUpperCase();
        } else {
            return `${firstName} ${lastName || '--'}`;
        }
    }

    // let noName: any;
    // const name = fullName(noName, 'Stark');

    const name = fullName('Tony', 'Stark', true);

    console.log({ name })
})()
```

2. Argumentos REST.
    a. Es un argumento en lista que recibe n cantidad de valores.

```
(() => {
    const fullName = (firstName: string, ...restoDeArgumentos: string[]) => {
        return `${firstName} ${restoDeArgumentos.join(' ')}`;
```

```
    }

    const superman = fullName('Clark', 'Joseph', 'Kent', 'Zavala', 'Sánchez');
    console.log({ superman })
})()
```

3. TypeScript reconoce los tipos de valores en los argumentos cuando se hace referencia.

```
(() => {
    const addNumbers = (a: number, b: number) => a + b;
    const greet = (name: string) => `Hola ${name}`;
    const saveTheWorld = () => `El mundo está salvado`;

    let myFunction;
    myFunction = 10;
    console.log(myFunction);

    myFunction = addNumbers;
    console.log(myFunction(1, 2));

    myFunction = greet;
    console.log(myFunction('Angel'));

    myFunction = saveTheWorld;
    console.log(myFunction());
})();
```

Sección 5.

1. Type en TypeScript está creada para obligar a objetos o arreglos que se crean.
2. Explicación del ts.config: https://cursos.devtalles.com/courses/take/typescript-guia-completa/lessons/35338547-que-es-el-archivo-tsconfig-y-para-que-nos-puede-servir.
    a. No se suele ajustar tan seguido.
    b. Mapeo de JS a TS en el navegador.

    ```
    "sourceMap": true,
    ```

    c. Remover comentarios.

    ```
    "removeComments": true,
    ```

Sección 7.

1. No usar var.
2. El estándar es el ES6 y ESNext.
3. Clases en JavaScript.

```
(() => {
    class Avenger {
        name;
        power;

        constructor(name = 'No name', power = 0) {
            this.name = name;
            this.power = power;
        };
    };

    class FlyingAvenger extends Avenger {
        flying; // Es innecesario declararla aquí, pero es buena práctica.
        constructor(name, power) {
            super(name, power);
            this.flying = true;
        }
```

```
    };

    const hulk = new Avenger('Hulk', 9001);
    const falcon = new FlyingAvenger('Falcon', 50);

    console.log(hulk);
    console.log(falcon);

})();
```

Sección 8.

1. Clases en TypeScript.
    a. Clase básica.

```
(() => {
    class Avenger {
        // Acceso a esta propiedad solo dentro de la clase.
        private name: string = 'Scott Lang';

        // Acceso a esta propiedad dentro de la clase y en clases derivadas.
        public team: string;
        public realName?: string;

        // Acceso a esta propiedad desde cualquier parte.
        static avgAge: number = 35;

        constructor(name: string, team: string, realName?: string){
            this.name = name;
            this.team = team;
            this.realName = realName;
        }
    }

    const antman: Avenger = new Avenger("Antman", "Avengers");
    console.log(antman);

    // console.log(Avenger.avgAge);
})();
```

    b. Herencia.

```
(() => {
    class Avenger {
        constructor(
            public name: string,
            public realName: string,
        ) {
            console.log(`Constructor Avenger llamado!`);

        }
        protected getFullName() {
            return `${this.name} ${this.realName}`;
        }
    };

    class Xmen extends Avenger {
        constructor(
            name: string,
            realName: string,
            public isMutant: boolean
        ) {
            super(name, realName);
            console.log(`Constructor Xmen llamado!`);
        }

        getFullNameDesdeXmen() {
```

```
            console.log(super.getFullName());
        }
    };

    const wolverine = new Xmen('Wolverine', 'Logan', true);
    console.log(wolverine);
    wolverine.getFullNameDesdeXmen();

    const nuevoAvenger = new Avenger('Hola', 'Mundo');
})();
```

c. Se puede usar public, private y protected.

```
// Se puede usar public, private, protected.
// Por defecto es public.
// private: Solo se puede usar dentro de la clase.
// protected: Se puede usar dentro de la clase y en las clases que hereden de esta.
```

d. getters y setters.

```
class Xmen extends Avenger {
    constructor(
        name: string,
        realName: string,
        public isMutant: boolean
    ) {
        super(name, realName);
        console.log(`Constructor Xmen llamado!`);
    }

    get fullName() {
        return `${this.name} - ${this.realName}`;
    }

    // No retorna nada.
    // Solo se recibe un argumento.
    set fullName(name: string){
        this.name = name;

        if(name.length < 3){
            throw new Error('El nombre debe de ser mayor de 3 letras');
        }
    }

    getFullNameDesdeXmen() {
        console.log(super.getFullName());
    }
};
```

e. Clases abstractas.
    a. Las clases abstractas son la base para otras clases, no se pueden instanciar directamente.
    b. Se definen con la palabra reservada abstract y pueden contener métodos abstractos que deben ser implementados en las clases derivadas.

Sección 9.

1. Una interfaz sirve para peticiones http por ejemplo.
2. Lucen muy similares las interfaces y los tipos, pero los tipos no se pueden expandir y las interfaces sí.
3. Lo ideal es crear clases para métodos complejos.
4. Las interfaces no crean instancias.
5. Usos
    a. Tipo cuando sé que no va a extenderse, como patrones, tipo Ridux.
    b. Para lo demás interfaces.

Sección 10.

1. Los namespace sirven de agrupar para utilizarlo en diferentes espacios.
2. Los namespaces se utilizan para agrupar funcionalidades relacionadas y evitar conflictos de nombres en el código.

Sección 11.

1. Funciones genéricas:

```
export const printObject = (argument: any) => {
   console.log(argument);
};

export function genericFunction<T>(argument: T): T {
   return argument;
};

export const genericFunctionArrow = <T>(argument: T) => argument;
```

```
import { printObject, genericFunction, genericFunctionArrow } from './generics/generics';

console.log(genericFunction(123.1416).toFixed(2));
console.log(genericFunction('Hola mundo!!!').toLowerCase());
console.log(genericFunctionArrow(new Date()).getDate());
```

Sección 12.

1. Decoradores.
   a. Son una forma de modificar clases o propiedades en tiempo de ejecución.
   b. Se usan mucho en frameworks como Angular.

Sección 13.

1. index.d.ts sirve para darle definición entre JS y TS.