Apuntes TypeScript para pasar a Obsidian ->

Primeros pasos:

1 Instalar node.js -> https://nodejs.org/en

2 comprobar todo okey con comando: **node -v** -> ver version

```
xtalo@DESKTOP-7S66PIN MINGW64 ~/Desktop/DAM_GIT/Acceso_Datos/TypeScriptPRacticando (main)
$ node -v
v20.10.0
```

3 Instalar por la terminal TypeScript:

comando: npm i -g typescript

npm -> sistema de paquetes de node.js
i -> install
-g -> argumento para que sea global

```
xtalo@DESKTOP-7S66PIN MINGW64 ~/Desktop/DAM_GIT/Acceso_Datos/TypeScriptPRacticando (main)
$ npm i -g typescript

added 1 package in 6s
```

4 comprobar todo okey con: **tsc -v**

```
xtalo@DESKTOP-7S66PIN MINGW64 ~/Desktop/DAM_GIT/Acceso_Datos/TypeScriptPRacticando (main)
$ tsc -v
Version 5.3.3
```

Si sale error , abrir terminal como admin

```
Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Windows\system32> get-ExecutionPolicy
Restricted
PS C:\Windows\system32> set-ExecutionPolicy remotesigned

Execution Policy Change
The execution policy helps protect you from scripts that you do not trust. Changing the execution policy might
you to the security risks described in the about_Execution_Policies help topic at
https:/go.microsoft.com/fwlink/?LinkID=135170. Do you want to change the execution policy?
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help (default is "N"): a
PS C:\Windows\system32>
```

Ya estaría todo listo e instalado para practicar typeScript.

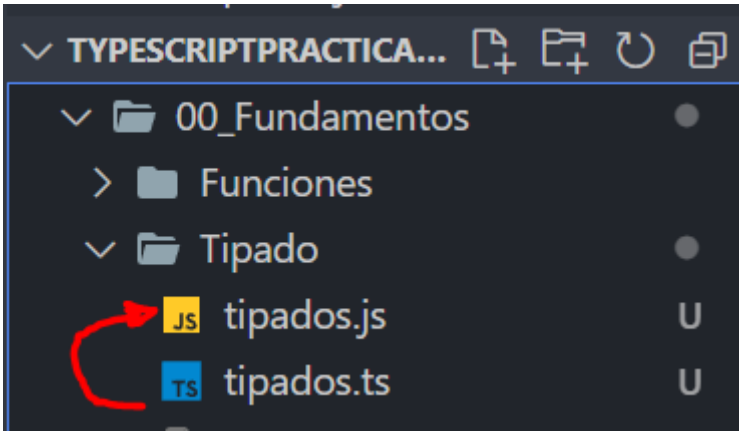
¿De que rollo va TypeScript?

TypeScript tiene la misión de tipar los datos, cosa que no se hace en javaScript.

Cuando escribimos en ts (typescript), a la hora de la verdad, lo que acaba de ejecutandose es un archivo js (javaScript). Cuando acabamos un archivo tsc, lo convertimos a js mediante el comando **tsc**:

```
xtalo@DESKTOP-7S66PIN MINGW64 ~/Desktop/DAM_GIT/Acceso_Datos/TypeScriptPRacticando/00_Fundamentos/Tipado (main)
$ tsc tipados.ts
```

en este ejemplo pasamos de un archivo ts a js:



Configuración de TypeScript

TypeScript tiene una configuración customizable, esto quiere decir que podemos cambiar la forma de escribir en él para ser más permisivos o menos permisivos.

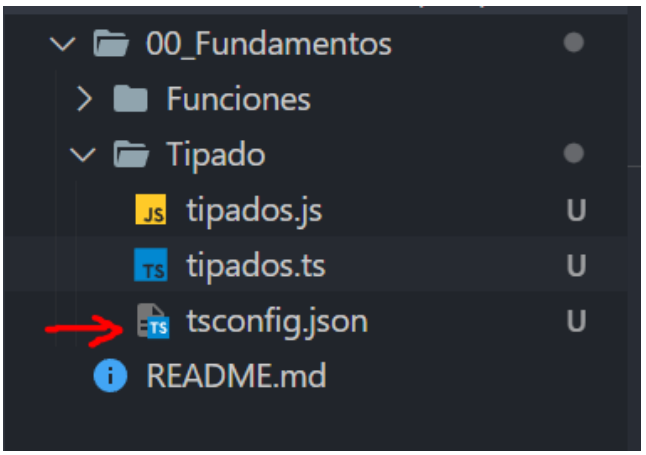
Para ello, se usa un archivo de configuración llamado **tsconfig.json**, este archivo se genera con el comando **tsc -init**.

```
xtalo@DESKTOP-7S66PIN MINGW64 ~/Desktop/DAM_GIT/Acceso_Datos/TypeScriptPRacticando/00_Fundamentos/Tipado (main)
$ tsc -init

Created a new tsconfig.json with:

  target: es2016
  module: commonjs
  strict: true
  esModuleInterop: true
  skipLibCheck: true
  forceConsistentCasingInFileNames: true


You can learn more at https://aka.ms/tsconfig
```




```jsonc
{
  "compilerOptions": {
    /* Visit https://aka.ms/tsconfig to read more about this file */

    /* Projects */
    // "incremental": true,                              /* Save .tsbuildi
    // "composite": true,                                /* Enable constra
    // "tsBuildInfoFile": "./.tsbuildinfo",              /* Specify the pa
    // "disableSourceOfProjectReferenceRedirect": true,  /* Disable prefer
    // "disableSolutionSearching": true,                 /* Opt a project
    // "disableReferencedProjectLoad": true,             /* Reduce the num

    /* Language and Environment */
    "target": "es2016",                                  /* Set the JavaSc
    // "lib": [],                                        /* Specify a set
    // "jsx": "preserve",                                /* Specify what J
    // "experimentalDecorators": true,                   /* Enable experim
    // "emitDecoratorMetadata": true,                    /* Emit design-ty
    // "jsxFactory": "",                                 /* Specify the JS
    // "jsxFragmentFactory": "",                         /* Specify the JS
    // "jsxImportSource": "",                            /* Specify module
    // "reactNamespace": "",                             /* Specify the ob
    // "noLib": true,                                    /* Disable inclu
    // "useDefineForClassFields": true,                  /* Emit ECMAScrip
    // "moduleDetection": "auto",                        /* Control what m
```

En este archivo nos encontraremos tantas opciones que ni los de TypeScript se las saben… Cuando tengamos que cambiar alguna la cambiaremos.

La línea 14 de la imagen tiene el parámetro “target”, sirve para indicar a que versión de js se está transcribiendo.

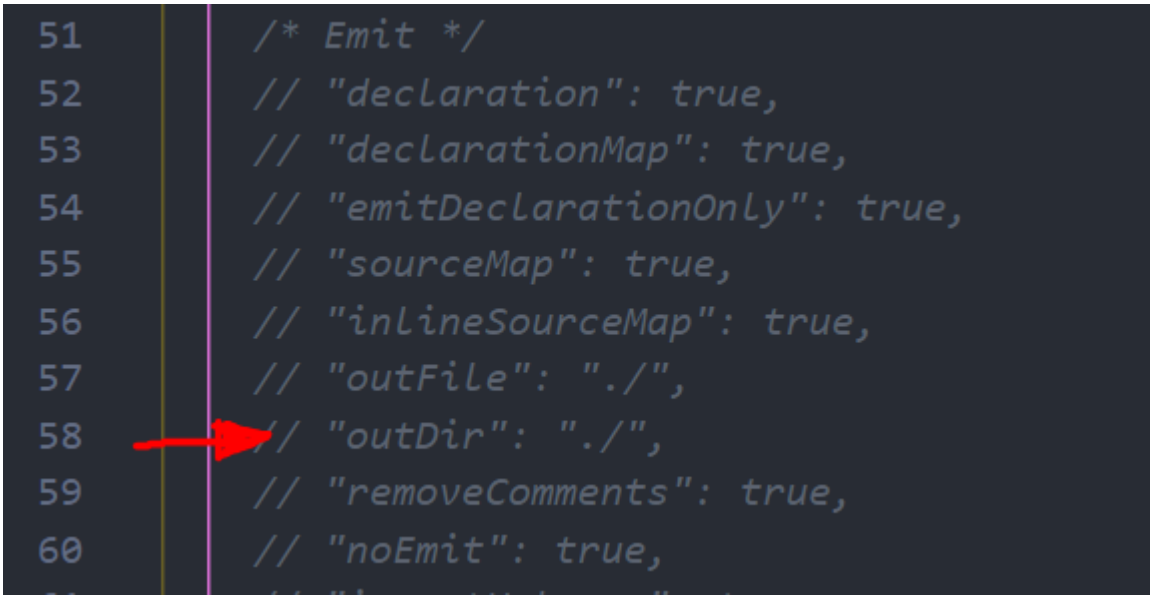
Parámetros de configuración Importantes

rootDir

En la línea 29 de la imagen tenemos el parámetro “rootDir” comentado, este parámetro, indica al proyecto, donde se encuentran los archivos de typeScript.

Podemos descomentarla e indicar dónde estarán los archivos de ts para que el proyecto lo sepa a la hora de la transcripción (procura que la ruta exita).

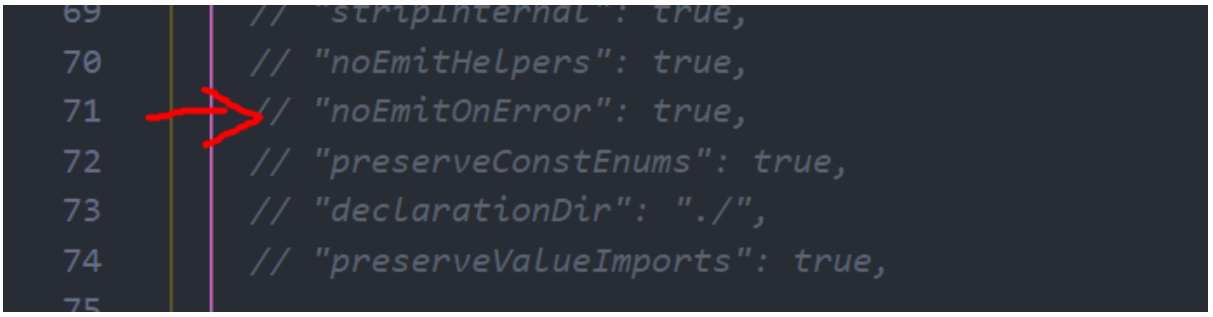
outDir



outDir indica dónde irán los archivos js transcritos, por convención se usa la carpeta dist.

justo debajo de outDir esta **removeComments**, sirve para eliminar los comentarios en el archivo js que tengamos en el archivo ts.

noEmitOnError



Si el código fuente de ts tiene algún error, este parámetro indicará si se transcribe o no a js.

```
70    // "noEmitHelpers": true,
71    "noEmitOnError": true,
72    // "preserveConstEnums": true,
73    // "declarationDir": "./",
```

Importante

Una vez configurado el tsconfig.json, a partir de ahora se usara solo el comando tsc, y este hará los cambios automáticos como está configurado.

```
xtalo@DESKTOP-7S66PIN MINGW64 ~/Desktop
$ tsc
```

Nota: Si ahora tuvieramos un ts fuera de src (menos en dist), en el mismo nivel de config.json, nos saltara error de que no todos los archivos estan en src.