

Práctica Typescript



Índice

Introducción:

- Explicación general del proyecto.....2
- Herramientas que utilizaremos.....2
- Elementos a tener en cuenta.....2

Instalaciones:

- Instalación de node.js.....3
- Instalación del compilador Typescript.....7

Primeros pasos:

- Estructura de un proyecto Typescript.....8
- Primer proyecto desde 0.....10
- Instalación proyecto ya existente.....15
- Guía sobre comandos básicos.....17
- Subir el repositorio a Github mediante Git.....18

Práctica:

- Explicación general del proyecto.....21
- Explicación de cada ejercicio.....22
 - Ejercicios de cálculo.
 - Ejercicio con objetos.
 - Ejercicios con fechas.
- Recursos de programación empleados.....30

Introducción:

- **Explicación general del proyecto.**

En esta segunda práctica crearemos un proyecto de Typescript y lo subiremos a Github.

En el transcurso del proyecto observaremos los distintos componentes de un proyecto Typescript, la instalación de las herramientas a emplear, y las distintas funciones que desarrollaremos.

- **Herramientas que utilizaremos.**

Utilizaremos las siguientes herramientas o servicios para llevar a cabo la práctica:

- **Visual Studio Code:** Será el entorno donde hemos realizado el grueso del proyecto, es decir, todas las funciones de nuestro typescript.
- **Git y Github:** Por un lado, Git será la aplicación que utilizaremos para exportar el src de nuestro proyecto a Github, el cual es el sitio web donde tendremos nuestro repositorio de forma online.
- **Typescript:** Descargaremos el paquete de Typescript para utilizarlo como compilador.
- **NodeJS:** Será lo que nos permita llevar a cabo toda la configuración de paquetes y módulos en nuestro proyecto y por así decirlo ocupa un rol más administrativo sobre Typescript.

- **Elementos a tener en cuenta.**

Este proyecto lo considero una continuación del anterior por lo que no volveremos a ver:

- Creación de cuenta y de repositorio de Github (desde web).
- Profundización sobre los comandos de Git.

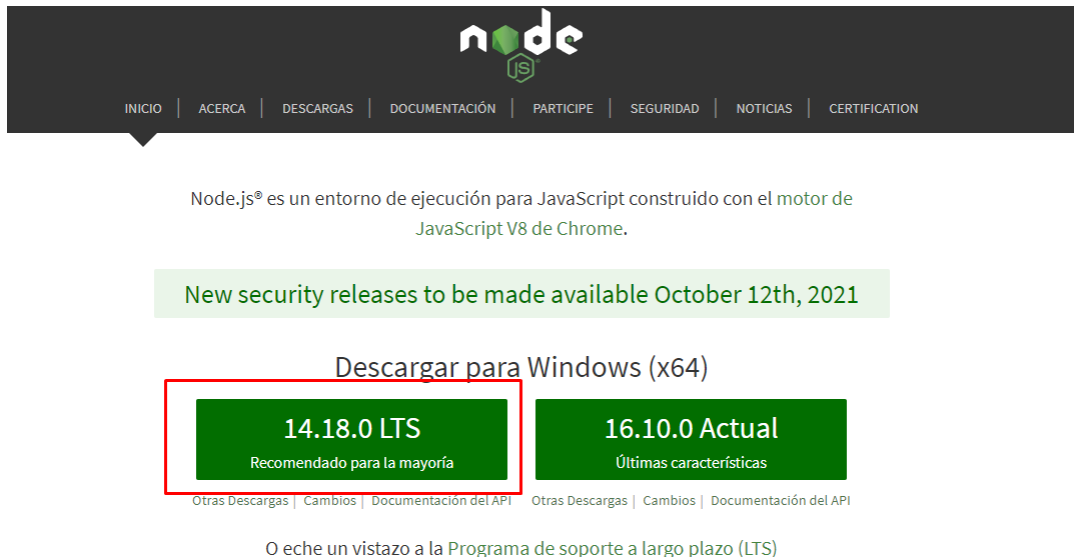
Sin embargo, si se documentará todo lo necesario acerca de la subida del repositorio a Github mediante Git.

Instalaciones:

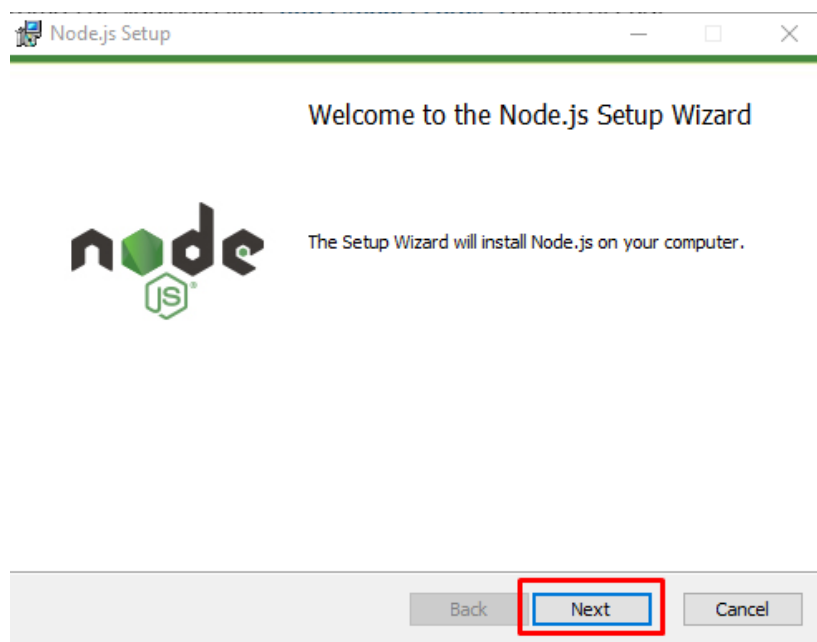
- **Instalación de NodeJS.**

NodeJS lo utilizaremos para tener controlados los distintos módulos que emplearemos en el proyecto, por ejemplo el mismo Typescript para instalarlo.

Nos dirigiremos al siguiente link: <https://nodejs.org/es> desde el cual descargaremos NodeJS. En concreto, la versión remarcada en la captura:



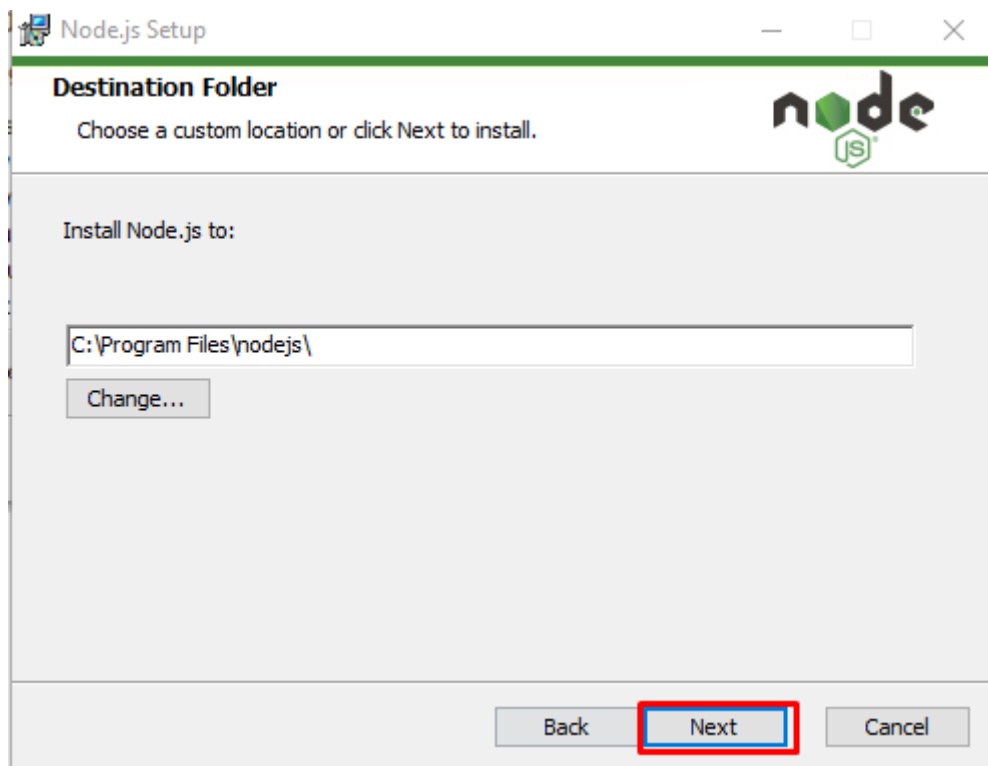
Daremos click a siguiente



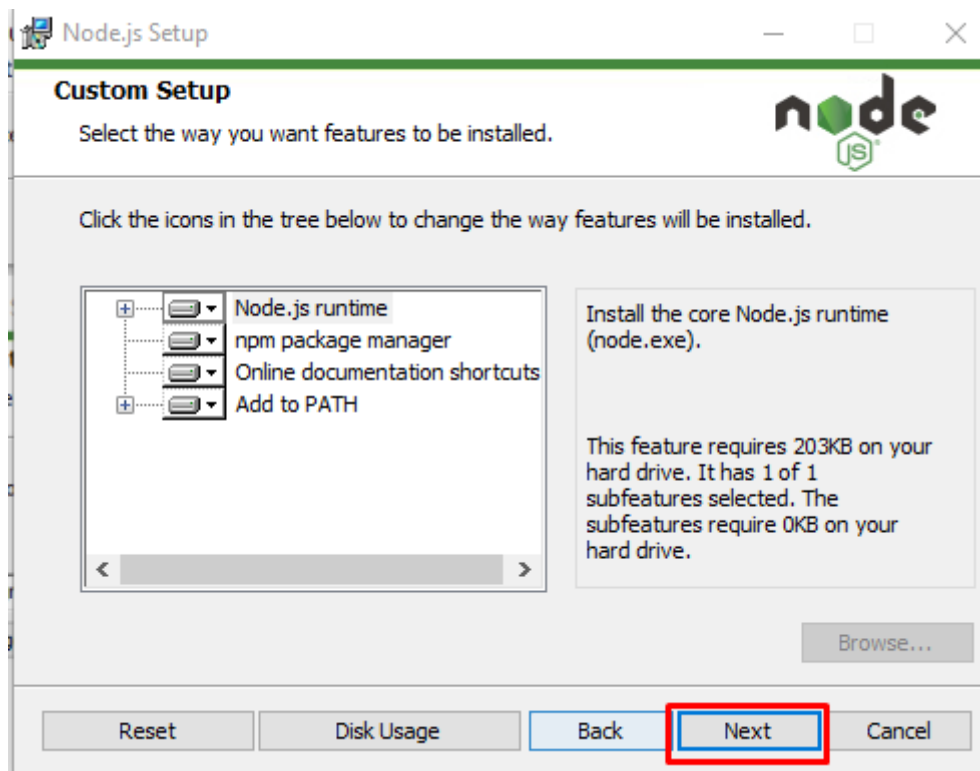
Remarcaremos la casilla de que aceptamos los términos y daremos a siguiente.



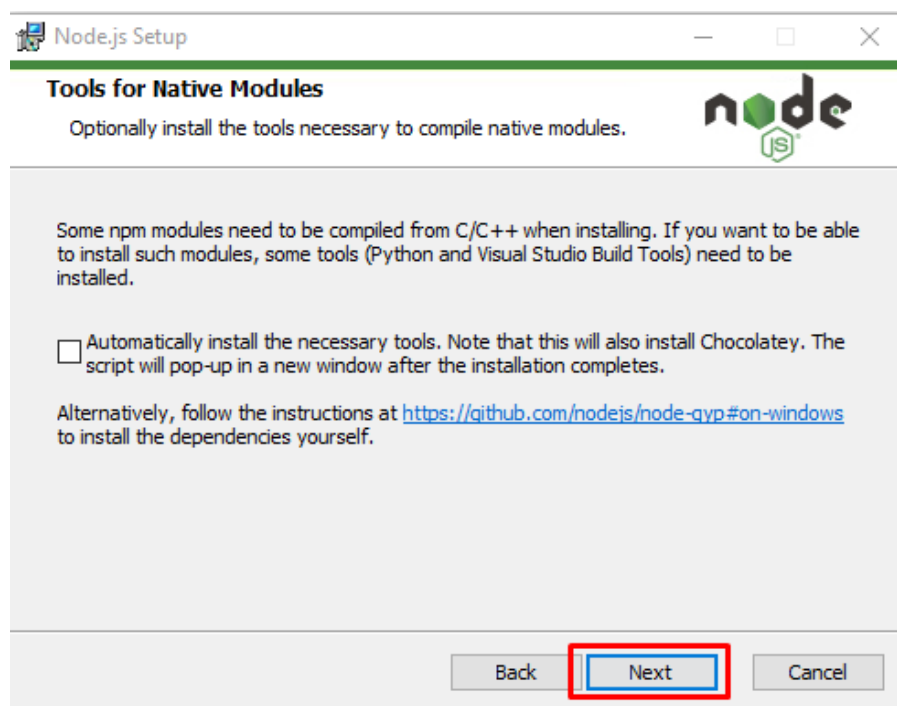
Daremos click en siguiente



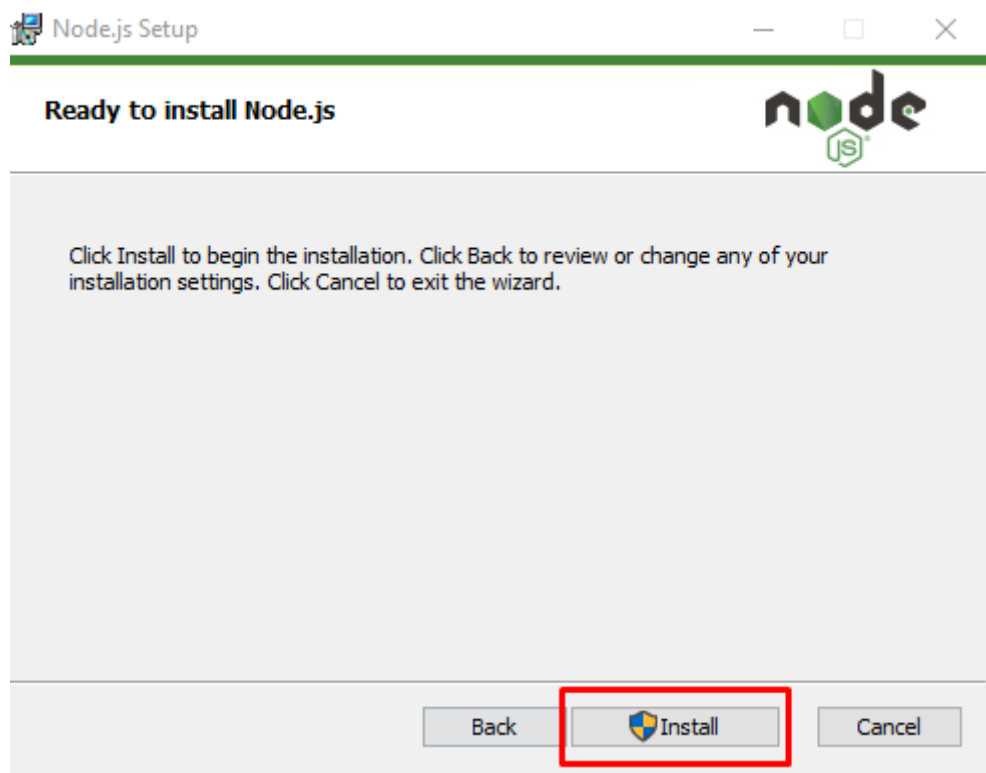
Daremos click en siguiente



Daremos click en siguiente



Daremos click en instalar

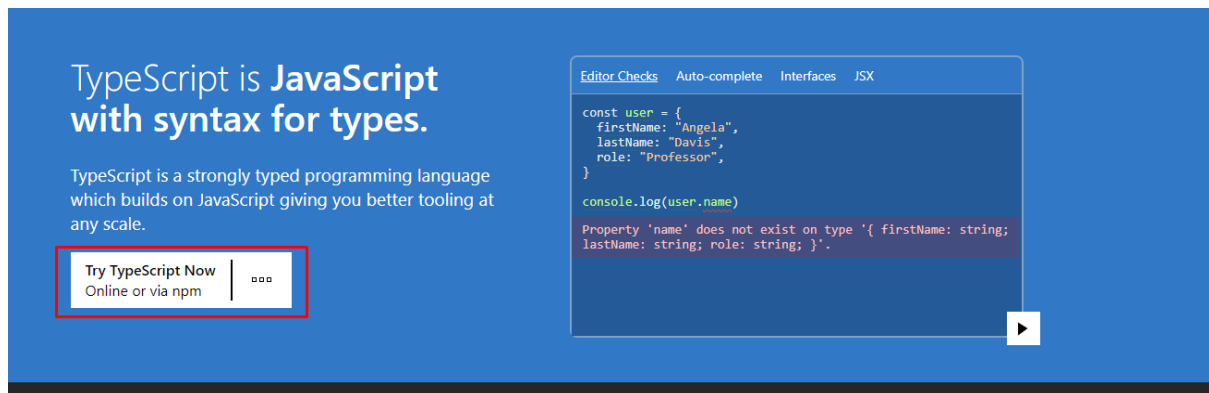


En la consola de Powershell pondremos el comando “node -v” para asegurarnos que se haya instalado correctamente nodejs.

```
PS C:\Users\Adria> node -v  
v14.17.6
```

- **Instalación del compilador Typescript.**

El paquete de Typescript viene por defecto en NodeJS. Para instalar el compilador Typescript mediante comandos los cuales podemos encontrar en la siguiente página: <https://www.typescriptlang.org/>



Primero instalaremos el compilador typescript mediante el comando “npm install -g typescript”

```
PS C:\Users\Adria> npm install -g typescript
C:\Users\Adria\AppData\Roaming\npm\tsc -> C:\Users\Adria\AppData\Roaming\npm\node_modules\typescript\bin\tsc
C:\Users\Adria\AppData\Roaming\npm\tsserver -> C:\Users\Adria\AppData\Roaming\npm\node_modules\typescript\bin\tsserver
+ typescript@4.4.3
updated 1 package in 3.007s
PS C:\Users\Adria>
```

Una vez instalado podremos utilizar el comando “tsc -v” y si está bien instalado el complemento veremos como nos muestra la versión del compilador typescript.

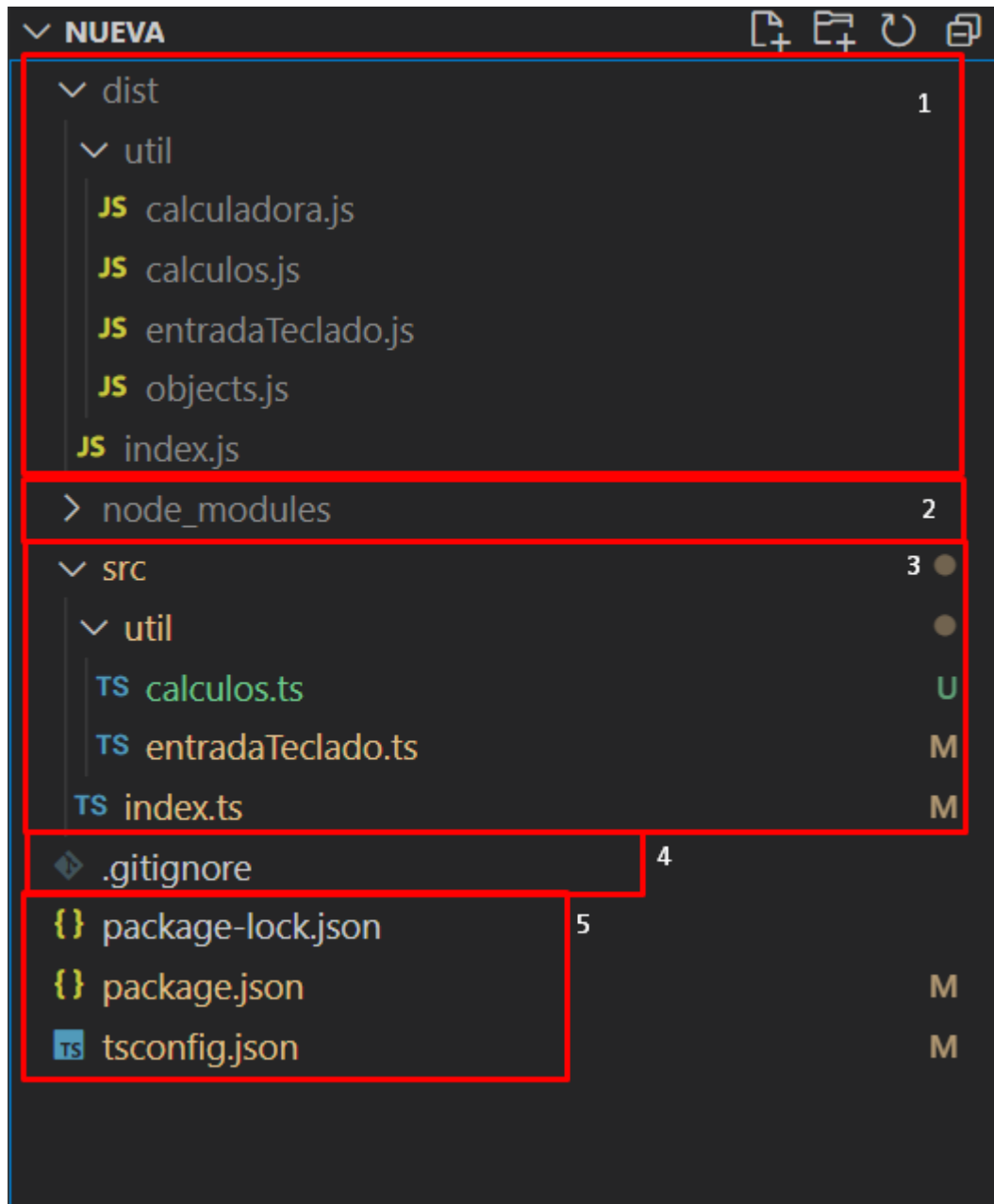
```
PS C:\Users\Adria> tsc -v
Version 4.4.3
```

Tenemos que tener en cuenta que la primera vez que compilemos nos saldrá un error que solucionaremos con el comando “Set-executionPolicy unrestricted”, lo veremos más a fondo en el apartado de “Primeros Pasos”.

Primeros Pasos:

- **Estructura de un proyecto Typescript.**

Un proyecto de Typescript se compone de distintas partes, aquí analizaremos el uso y el por qué de cada una de esas partes, las dividiremos en distintas secciones.



- 1) **Carpeta Dist:** Aquí dispondremos los archivos javascripts, los cuales serán los que verdaderamente se ejecuten. Estos archivos se generarán al compilar los archivos typescripts, que será donde verdaderamente programemos. Seguirá la misma estructura de archivos que la carpeta src.

- 2) **Node_Modules:** Este directorio contendrá cada una de las dependencias que instalemos en el proyecto, es decir, contendrá todos los paquetes instalados mediante NodeJS a través del comando “npm”.
- 3) **src:** Este directorio contendrá los archivos typescript de nuestro proyecto, y por decirlo de alguna manera será el “corazón” de nuestro proyecto y donde dediquemos prácticamente todo el tiempo.

Vemos que la carpeta src contiene el archivo “index.ts” que, su respectiva versión de javascript “index.js”, será el primer archivo que se ejecute al iniciar el proyecto.

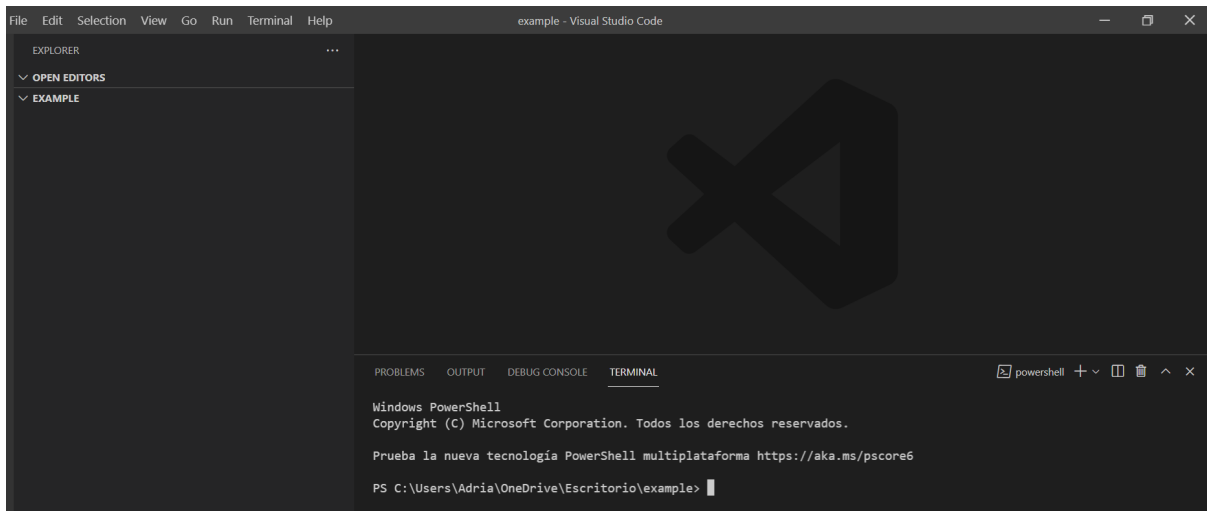
También tenemos el directorio “util” que contendrá en mi caso dos archivos typescript, “calculos.ts” y “entradaTeclado.ts” en los cuales realizaremos la mayor parte de nuestro código, ya que siempre intentaremos dejar el archivo index lo más claro posible.

- 4) **.gitignore:** Este archivo como vimos en la anterior práctica lo emplearemos para ocultar los archivos/directorios que no queremos que se suban a github al realizar una orden push/pull. En la mayoría de casos ocultaremos el directorio node_modules.
- 5) **Archivos de configuración:** El archivo package.json se utiliza únicamente para definir los paquetes que deben depender del proyecto mientras que el archivo package-lock.json es un archivo que se genera para documentar la versión y la fuente de cada uno de los paquetes instalados con npm. El archivo tsconfig.json contiene las configuraciones para el compilador typescript.

- **Primer proyecto**

Para comenzar un nuevo proyecto nos iremos a una nueva carpeta vacía. Partiremos de la base que he dado en “Instalaciones” y “Primeros Pasos” por lo que deberemos tener ya Typescript y NodeJS funcionando.

En una nueva carpeta abriremos visual studio code teniendo una cosa tal que así



Con el comando “npm init -y” crearemos el archivo package.json el cual es

```
PS C:\Users\Adria\OneDrive\Escritorio\example> npm init -y
Wrote to C:\Users\Adria\OneDrive\Escritorio\example\package.json:

{
  "name": "example",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}

PS C:\Users\Adria\OneDrive\Escritorio\example>
```

Lo siguiente será crear el fichero tsconfig.json mediante el comando “tsc --init”

```
PS C:\Users\Adria\OneDrive\Escritorio\example> tsc --init
message TS6071: Successfully created a tsconfig.json file.
PS C:\Users\Adria\OneDrive\Escritorio\example>
```

Ahora dentro de este archivo tsconfig.json deberemos hacer los siguientes cambios:

En la línea 14 cambiaremos el valor de “target” a “es6” quedando de la siguiente manera: (“target”: “es6”,)

```
8      // "tsBuildInfoFile": "./",
9      // "disableSourceOfProjectReferenceRedirect": true,
10     // "disableSolutionSearching": true,
11     // "disableReferencedProjectLoad": true,
12
13     /* Language and Environment */
14     "target": "es5",
15     // "lib": [],
16     // "jsx": "preserve",
17     // "experimentalDecorators": true,
18     // "emitDecoratorMetadata": true,
19     // "jsxFactory": "",
20     // "jsxFragmentFactory": ""
```

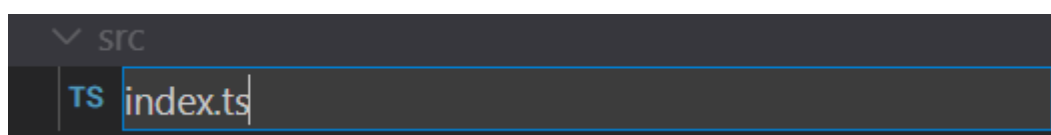
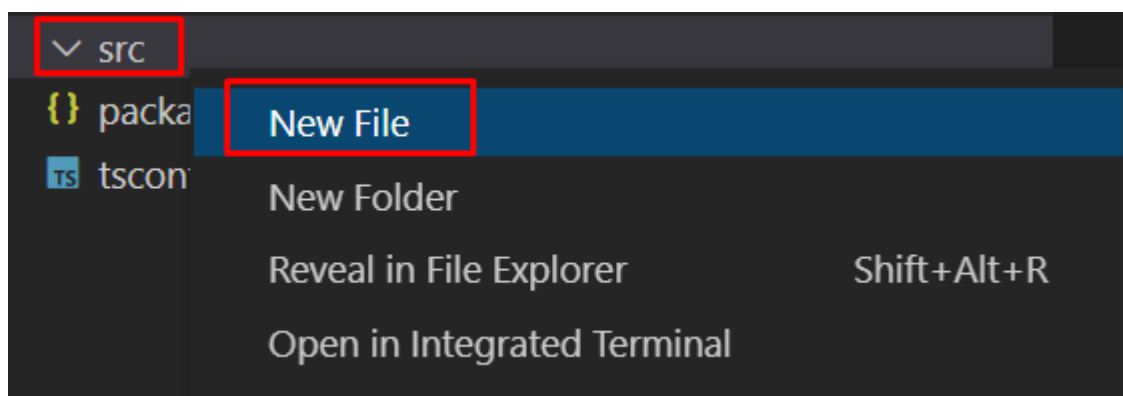
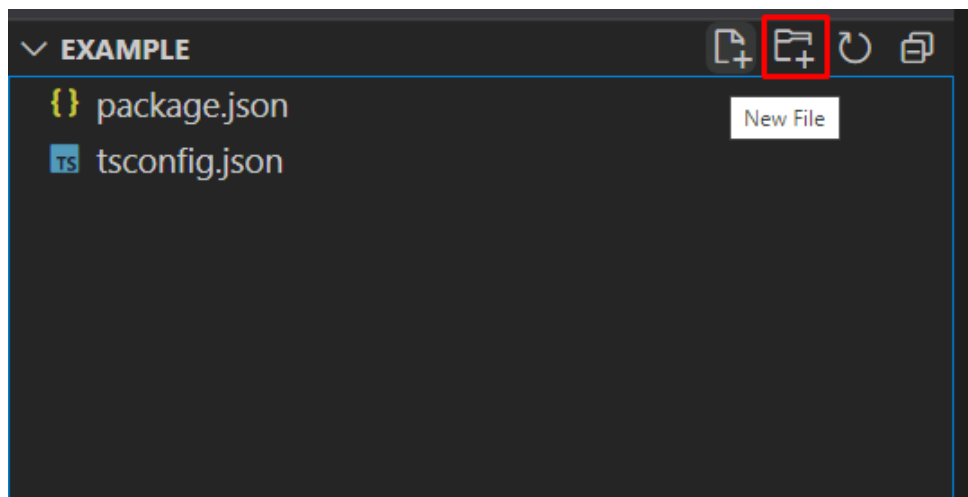
También, en la línea 50 quitaremos el documentado a “outDir” y le daremos el valor “./dist”

```
44     /* Emit */
45     // "declaration": true,
46     // "declarationMap": true,
47     // "emitDeclarationOnly": true,
48     // "sourceMap": true,
49     // "outFile": "./",
50     // "outDir": "./",
51     // "removeComments": true,
52     // "noEmit": true,
53     // "importHelpers": true,
54     // "importsNotUsedAsValues": "remove",
55     // "downlevelIteration": true,
56     // "sourceRoot": ""
```

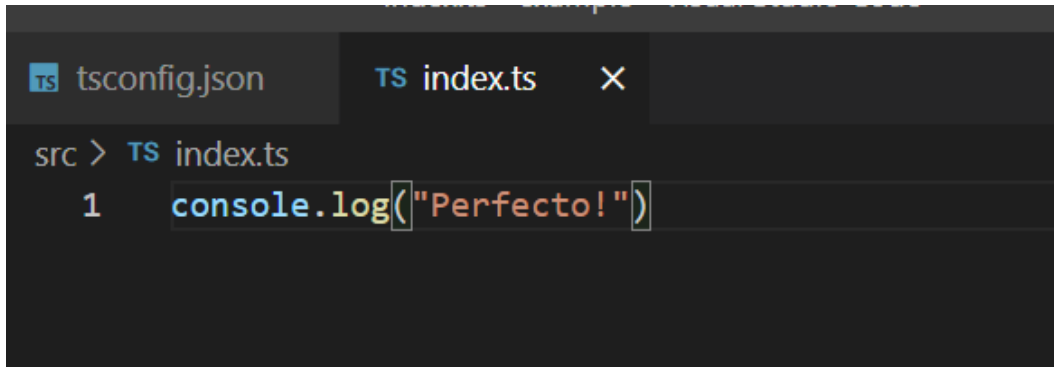
Así debería ser la línea resultante:

```
44      /* Emit */
45      // "declaration": true,
46      // "declarationMap": true,
47      // "emitDeclarationOnly": true,
48      // "sourceMap": true,
49      // "outFile": "./",
50      "outDir": "./dist",
51      // "removeComments": true,
52      // "noEmit": true,
53      // "importHelpers": true,
54      // "importsNotUsedAsValues": "remove",
55      // "downlevelIteration": true,
```

Ahora, crearemos a mano el directorio src y dentro el archivo “index.ts”



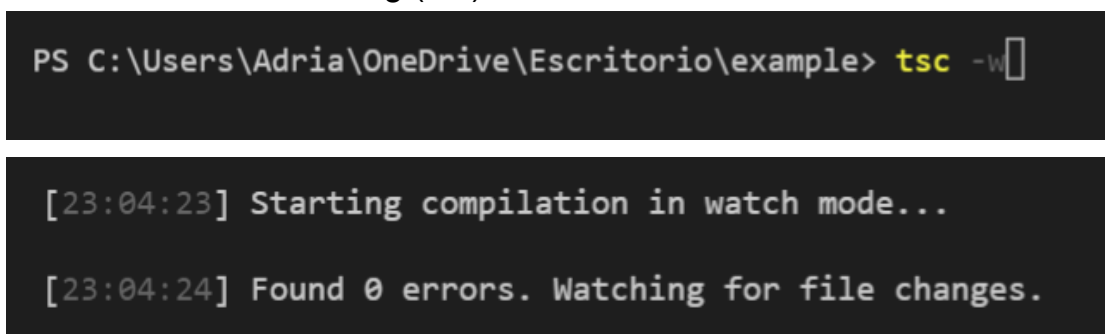
En el archivo escribiremos por ejemplo un `console.log` que diga “Perfecto!”, tras esto, guardaremos el archivo.



```
src > TS index.ts
1  console.log("Perfecto!")
```

Ahora compilaremos, pero ya que es la primera vez que compilamos nos saldrá un error el cual solucionaremos abriendo como administrador la terminal de powershell e introduciendo el comando “**Set-executionPolicy unrestricted**”

Si no es nuestra primera vez compilando, simplemente pondremos el comando “`tsc -w`” y esperaremos a que acabe, tras el segundo mensaje, el de “Found 0 errors. Watching (etc)”. Pulsaremos `Ctrl+C`

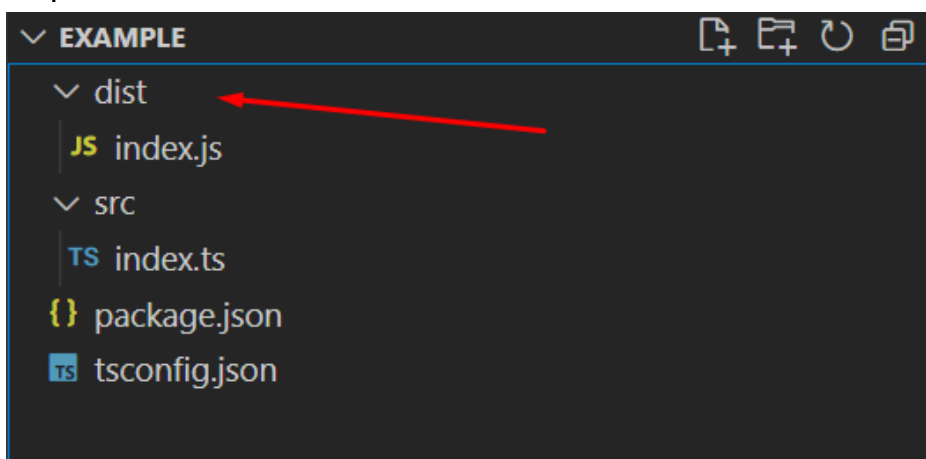


```
PS C:\Users\Adria\OneDrive\Escritorio\example> tsc -w

[23:04:23] Starting compilation in watch mode...

[23:04:24] Found 0 errors. Watching for file changes.
```

En el momento que compilemos podremos observar que se ha creado la carpeta “dist”.



Tras esto, utilizaremos el comando “node dist” para ejecutar nuestro código y veremos que ha funcionado perfecto, tal y como el mismo código nos dice.

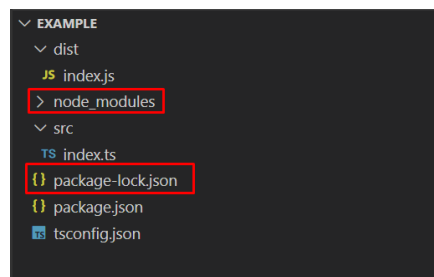
```
PS C:\Users\Adria\OneDrive\Escritorio\example> node dist
Perfecto!
PS C:\Users\Adria\OneDrive\Escritorio\example> 
```

El archivo “package-lock.json” se generaría automáticamente cuando agreguemos mediante el “npm” algún nuevo modulo que modifique lo estipulado en el “package.json” por ejemplo, al utilizar el comando “npm install --save-dev” veremos como nos genera los archivos citados.

```
PS C:\Users\Adria\OneDrive\Escritorio\example> npm install typescript --save-dev
npm notice created a lockfile as package-lock.json. You should commit this file.
npm WARN example@1.0.0 No description
npm WARN example@1.0.0 No repository field.

+ typescript@4.4.3
added 1 package from 1 contributor and audited 1 package in 1.155s
found 0 vulnerabilities

PS C:\Users\Adria\OneDrive\Escritorio\example> 
```

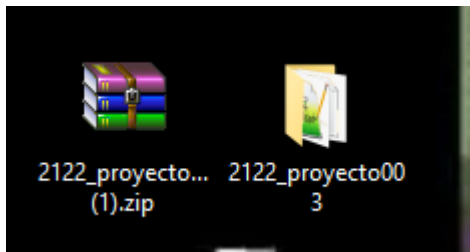


El comando “npm install --save-dev” simplemente nos agregará “DevDependencies” a nuestro package.json

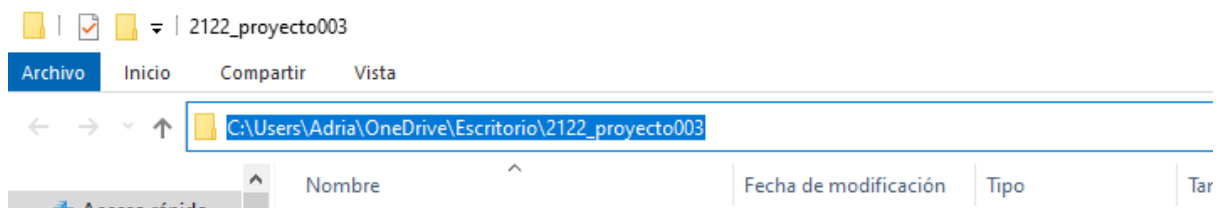
```
1  {
2    "name": "example",
3    "version": "1.0.0",
4    "description": "",
5    "main": "index.js",
6    "scripts": {
7      "test": "echo \"Error: no test specified\" && exit 1
8    },
9    "keywords": [],
10   "author": "",
11   "license": "ISC",
12   "devDependencies": {
13     "typescript": "^4.4.3"
14   }
15 }
16
```

- **Instalación de proyecto ya existente**

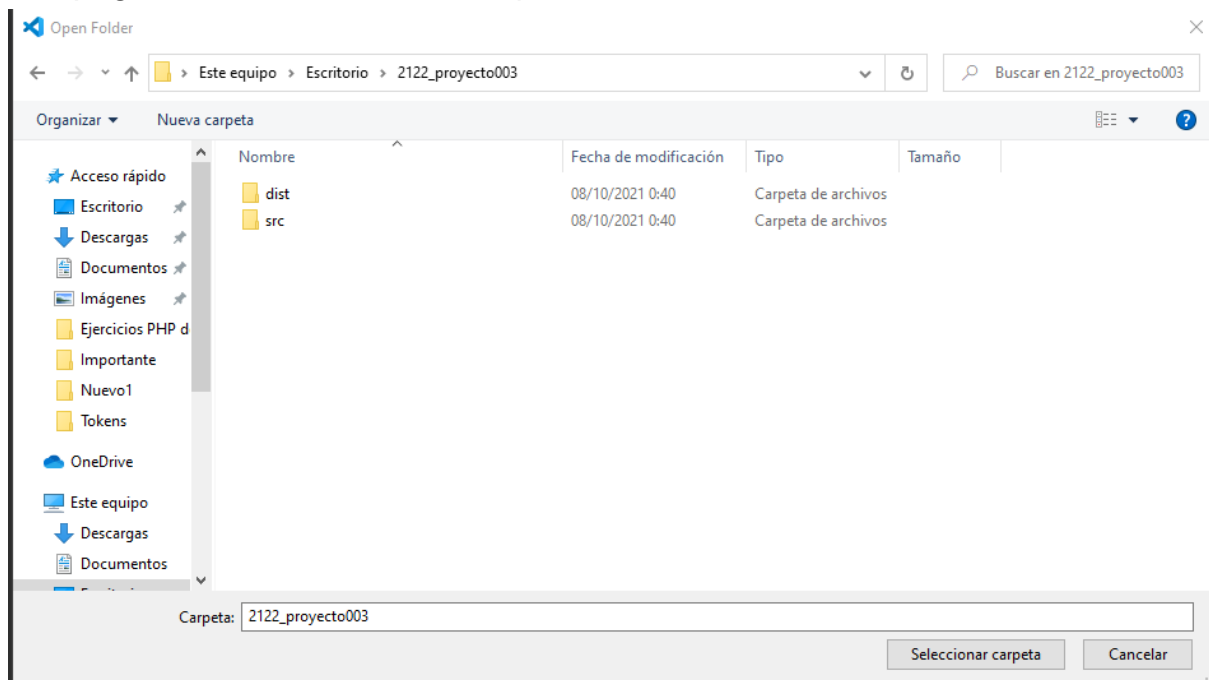
Voy a utilizar para esta prueba el proyecto que está subido en la moodle como “proyecto003” por lo que lo descargamos y lo extraemos.



Copiaremos la URL del proyecto



Y la pegaremos en Visual Studio para abrirlo



Comprobaremos que esté puesto el archivo "index.js" como main dentro del package.json y que dentro de "scripts" esté puesta la ruta del dist, en este caso no está así que la pondremos

```
{
  "name": "1030_nodejs",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  > Debug
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "dependencies": {},
  "devDependencies": {
    "@types/node": "^16.9.6"
  }
}
```

```
{
  "name": "1030_nodejs",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  > Debug
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "dist": "./dist"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "dependencies": {},
  "devDependencies": {
    "@types/node": "^16.9.6"
  }
}
```



Tras esto utilizaremos el comando “npm i --save-dev @types/node”

```
PS C:\Users\Adria\OneDrive\Escritorio\2122_proyecto003> npm i --save-dev @types/node
npm WARN 1030_nodejs@1.0.0 No repository field.

+ @types/node@16.10.3
added 1 package from 42 contributors and audited 1 package in 1.198s
found 0 vulnerabilities
```

Con node dist ejecutaremos el código para ver si inicia y en este caso, vemos que funciona sin problema.

```
PS C:\Users\Adria\OneDrive\Escritorio\2122_proyecto003> node dist

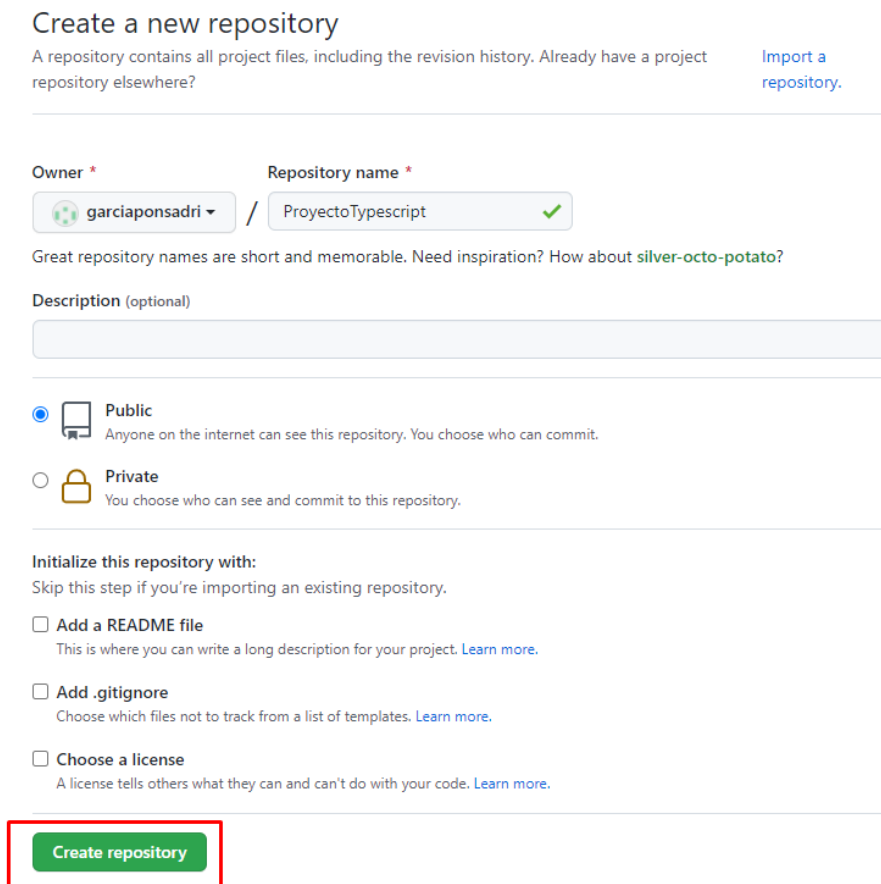
1.- Sumar
2.- Multiplicar
3.- Restar
0.- Salir
opción: : 
```

- Guía de comandos:

COMANDOS	UTILIZACIÓN
npm install -g Typescript	Nos instalará a nivel global el compilador Typescript mediante NodeJS.
tsc -v	Nos mostrará la versión del compilador Typescript (permitiéndonos ver que está instalado correctamente)
npm init -y	Nos generará el archivo package.json
tsc --init	Nos creará el archivo de configuración de typescript “tsconfig.json”
tsc -w	Nos permitirá compilar el código
Set-executionPolicy unrestricted	La primera vez que compilemos deberemos usarlo como administradores para dar permiso a Typescript

- **Subir el repositorio a Github mediante Git**

Lo primero que deberemos hacer habrá tenido que ser instalar Git y crear un repositorio vacío en Github.



Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

Owner * garciaponsadri / Repository name * ProyectoTypescript ✓

Great repository names are short and memorable. Need inspiration? How about [silver-octo-potato](#)?

Description (optional)

☒ **Public**
Anyone on the internet can see this repository. You choose who can commit.

☐ **Private**
You choose who can see and commit to this repository.

Initialize this repository with:
Skip this step if you're importing an existing repository.


☐ **Add a README file**
This is where you can write a long description for your project. [Learn more.](#)

☐ **Add .gitignore**
Choose which files not to track from a list of templates. [Learn more.](#)

☐ **Choose a license**
A license tells others what they can and can't do with your code. [Learn more.](#)

Create repository

Aquí el mismo Github nos dará las instrucciones para realizar la subida del proyecto con git.



Quick setup — if you've done this kind of thing before

[Set up in Desktop](#) or [HTTPS](#) [SSH](#) <https://github.com/garciaponsadri/ProyectoTypescript.git>

Get started by [creating a new file](#) or [uploading an existing file](#). We recommend every repository include a [README](#), [LICENSE](#), and [.gitignore](#).

...or create a new repository on the command line

```
echo "# ProyectoTypescript" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M main
git remote add origin https://github.com/garciaponsadri/ProyectoTypescript.git
git push -u origin main
```

...or push an existing repository from the command line

```
git remote add origin https://github.com/garciaponsadri/ProyectoTypescript.git
git branch -M main
git push -u origin main
```

Nos iremos a la carpeta del proyecto original en lugar del proyecto que hemos creado de prueba.

Aprovechando la terminal de visual studio code, escribiremos “git init”

```
PS C:\Users\Adria\OneDrive\Escritorio\nueva> git init
Reinitialized existing Git repository in C:/Users/Adria/OneDrive/Escritorio/nueva/.git/
PS C:\Users\Adria\OneDrive\Escritorio\nueva> 
```

Realizaremos un “git add .”, en mi caso me ha salido una advertencia así que he vuelto a introducir el comando “git add.” y luego un “git status” para comprobar que se hayan agregado correctamente los archivos.

```
PS C:\Users\Adria\OneDrive\Escritorio\nueva> git add .
warning: LF will be replaced by CRLF in package-lock.json.
The file will have its original line endings in your working directory
warning: LF will be replaced by CRLF in package.json.
The file will have its original line endings in your working directory
PS C:\Users\Adria\OneDrive\Escritorio\nueva> git add .
PS C:\Users\Adria\OneDrive\Escritorio\nueva> git status
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   package.json
        modified:   src/index.ts
        new file:   src/util/calculos.ts
        modified:   src/util/entradaTeclado.ts
        modified:   tsconfig.json
```

Ahora haremos el primer commit mediante “git commit -m “First””.

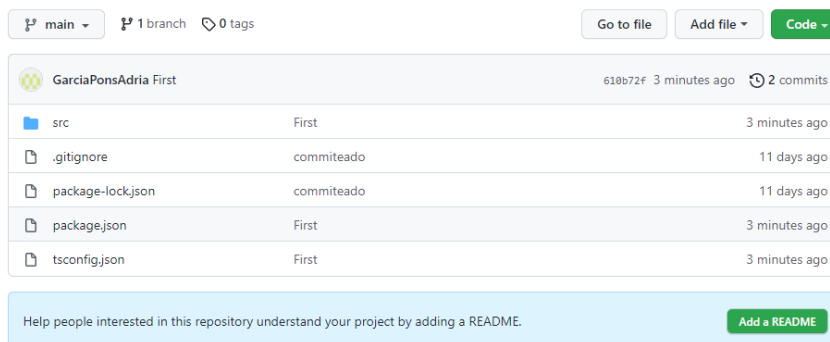
```
PS C:\Users\Adria\OneDrive\Escritorio\nueva> git commit -m "First"
[main 610b72f] First
 5 files changed, 345 insertions(+), 15 deletions(-)
 rewrite src/index.ts (63%)
 create mode 100644 src/util/calculos.ts
PS C:\Users\Adria\OneDrive\Escritorio\nueva> 
```

Cambiamos de rama a main mediante “git branch -M main” y agregaremos el repositorio de github en la variable “origin”, la cual yo ya tenía asignado así que la he eliminado mediante “git remote remove origin” para luego asignarla de nuevo mediante “git remote add origin <link>”

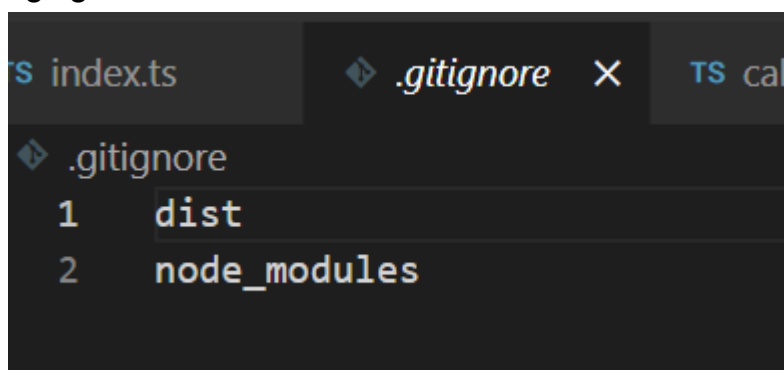
```
PS C:\Users\Adria\OneDrive\Escritorio\nueva> git branch -M main
PS C:\Users\Adria\OneDrive\Escritorio\nueva> git remote add origin https://github.com/garciaponsadri/ProyectoTypescript.git
error: remote origin already exists.
PS C:\Users\Adria\OneDrive\Escritorio\nueva> git remote remove origin
PS C:\Users\Adria\OneDrive\Escritorio\nueva> git remote add origin https://github.com/garciaponsadri/ProyectoTypescript.git
PS C:\Users\Adria\OneDrive\Escritorio\nueva>
```

Y ahora con el comando “git push -u origin main” subiremos el proyecto a Github

```
PS C:\Users\Adria\OneDrive\Escritorio\nueva> git push -u origin main
Enumerating objects: 19, done.
Counting objects: 100% (19/19), done.
Delta compression using up to 4 threads
Compressing objects: 100% (17/17), done.
Writing objects: 100% (19/19), 9.90 KiB | 2.48 MiB/s, done.
Total 19 (delta 3), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (3/3), done.
To https://github.com/garciaponsadri/ProyectoTypescript.git
* [new branch]      main -> main
Branch 'main' set up to track remote branch 'main' from 'origin'.
PS C:\Users\Adria\OneDrive\Escritorio\nueva>
```



Podemos comprobar como los directorios que hemos puesto en el archivo “.gitignore” no se han subido

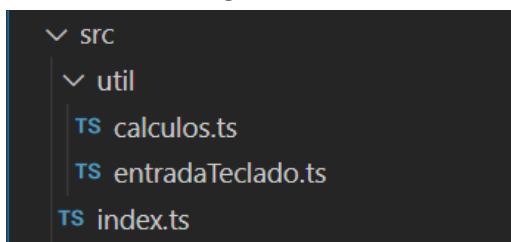


Práctica:

- **Explicación general de la práctica**

La práctica está dividida en 3 archivos, un archivo principal “index.ts” que será su respectiva versión compilada “index.js” la que se ejecute y será este el que llame al resto de archivos. Un segundo archivo “entradaTeclado.js” el cual utilizaremos para emplear su método “leerTeclado()” y por último un tercer archivo “calculos.ts” en el que está el grueso del proyecto.

He decidido tener tan solo 3 archivos Typescript porque me parecen más que suficientes en estos momentos y porque creo que las funciones de “calculo.ts” no tiene la magnitud suficiente como para crearle un archivo propio.



Los archivos “calculos.ts” y “entradaTeclado.ts” estarán dentro de src/util.

En el index simplemente lo he usado como menú donde he hecho un índice de las distintas funciones y luego, al llamar al método “calculos” con la opción elegida imprima el resultado.

```
// Aquí crearemos un bucle eterno en el que pediremos al usuario que elija una opción.
while(1) {
    console.log(`Que opción prefieres!`)
    console.log(`0: Salir`)
    console.log(`1: Numero Primo`)
    console.log(`2: Factorización`)
    console.log(`3: Presupuesto Coche`)
    console.log(`4: Ordenador de números`)
    console.log(`5: Operadora de Fechas`)
    console.log(`6: Cumpleaños calculador`)
    console.log(`7: Calculadora Edad`)
    // Registraremos la opción del usuario y la enviaremos a la función calculos
    // y registraremos el valor que nos retorne en la variable solución
    let entrada = parseInt(await leerTeclado (''))
    let solucion = await calculos(entrada)
```

Bienvenido al menú matemático!

```
Que opción prefieres!
0: Salir
1: Numero Primo
2: Factorización
3: Presupuesto Coche
4: Ordenador de números
5: Operadora de Fechas
6: Cumpleaños calculador
7: Calculadora Edad
:
```

- **Explicación de cada ejercicio**

- Ejercicios de cálculo.

- **Ejercicio 1: Listado de números primos entre dos números**

```
// La primera función será calcular los números primos entre dos números que servirán de delimitadores.
case 1: {
  console.log("| Números primos |")
  console.log(" ")
  console.log("Aquí calcularemos los números primos!")
  console.log("Puede insertar dos números que sirvan de delimitadores")
  console.log("Devolverá el numero de numeros primos (y cada numero primo)")

  // Pediremos los dos numeros que utilizaremos como delimitadores.

  let numero1 = parseInt(await(leerTeclado(' ')))
  let numero2 = parseInt(await(leerTeclado(' ')))

  // Haremos una comprobación para ordenarlos de menor (numero1) a mayor (numero2)
  if(numero1>numero2) {
    var temp=numero1
    numero1=numero2
    numero2=temp
  }
}
```

```
// Declaramos el array primos donde guardaremos los números primos.
// Recorreremos un bucle for utilizando numero1 y numero2 como delimitadores.
// Partiremos de que el número es primo, y tras dividirlo por todos los números
// entre el 2 y el mismo (mediante el segundo bucle for) le asignaremos el valor "false",
// es decir, que no es primo en caso de que haya sido divisible por alguno de estos números.
var primos = []
for (var i=numero1; i<numero2+1; i++) {
  var primo=true
  for (var e=2; e<i-1; e++) {
    if (i%e == 0) {
      primo=false
    }
  }
  // Aquí en caso de que el número no haya sido divisible por
  // ningún numero se agregará al array de números primos.
  if (primo==true) {
    primos.push(i)
  }
}
// Devolveremos el Array de los numeros primos como solución.
return primos
break;
```

| Números primos |

```
Aquí calcularemos los números primos!
Puede insertar dos números que sirvan de delimitadores
Devolverá el numero de numeros primos (y cada numero primo)
: 27
: 56
La solución es: 29,31,37,41,43,47,53
Pulse enter para continuar
: 
```

- Ejercicio 2: Factorización de un número

```
// En esta función factorizaremos un número que aporte el usuario.
case 2: {
    console.log("| FACTORIZACIÓN |")
    console.log(" ")
    // Declaramos los arrays, la variable "pos" y pediremos el número al usuario.
    var divisores = []
    var calculados = []
    var pos=0
    console.log("Inserte el número a factorizar")
    var numero1 = parseInt(await(LeerTeclado(' ')))
    // Mediante dos bucles for listaremos los números primos que utilizaremos como divisores.
    // Parecido a lo que hicimos en la anterior función.
    for (var i=2; i<numero1+1; i++) {
        var primo=true
        for (var e=2; e<i; e++) {
            if (i%e==0) {
                primo=false
            }
        }
        if (primo==true) {
            divisores[pos]=i
            pos++
        }
    }
}

// Finalmente, listaremos la lista de divisores y mediante un bucle dowhile(), iremos dividiendo
// el numero que nos dió el usuario por esta lista de divisores.
console.log("Lista de numeros primos: ")
console.log(divisores)
var o=0;
do {
    if(numero1%divisores[o]==0) {
        console.log(numero1+" | "+divisores[o])
        numero1=numero1/divisores[o]
        calculados.push(divisores[o])
    } else {
        o++
    }
} while(numero1>1);

return(calculados)
break;
}
```

```
Inserte el número a factorizar
: 38
Lista de numeros primos:
[
    2, 3, 5, 7, 11,
    13, 17, 19, 23, 29,
    31, 37
]
38 | 2
19 | 19
La solución es: 2,19
Pulse enter para continuar
: 
```

- Ejercicio 3: Ordenar una lista de números

```
// En esta función ordenaremos todos los números que quiera el usuario de menor a mayor.
case 3: {
    console.log("| ORDENATOR |")
    console.log(" ")
    console.log("Todos los numeros que quieras los ordenare")
    // Primero pediremos al usuario que nos diga cuantos números querrá introducir para ordenarlos.
    console.log("Primero indicame el numero de numeros a introducir")
    let numeroNum = parseInt(await(LeerTeclado(' ')))
    var Numeros = []
    console.log("Perfecto! Ahora a introducir los números")
    // Tras esto, mediante un bucle for iremos pidiendo al usuario que los ingrese de forma ordenada
    // y lo iremos guardando en un array Numeros
    for (i=0; i<numeroNum; i++) {
        console.log("Numero "+(i+1)+"/"+numeroNum+": ")
        Numeros[i]= parseInt(await(LeerTeclado(' ')))
    }
    var Temp
    // Printearemos la lista de números insertada de forma desordenada.
    console.log("Lista de numeros desordenada:")
    console.log(Numeros)
```

```
// Para ordenar el Array utilizaremos el ordenamiento burbuja, el cual se basa en crear
// dos bucles for, el primer bucle for irá recorriendo uno a uno los números del array, el segundo
// partirá desde la posición 0 del array e irá avanzando hasta la casilla donde se haya quedado
// el for principal. Tras esto, realizará la simple comprobación de ver que elemento es mayor,
// el actual en el que se situa el array secundario o el elemento siguiente
for (i=0; i<numeroNum-1; i++) {
    for (e=0; e<(numeroNum-i); e++) {
        if(Numeros[e]>Numeros[e+1]) {
            Temp=Numeros[e]
            Numeros[e]=Numeros[e+1]
            Numeros[e+1]=Temp
        }
    }
}

// Devolveremos el array Numeros que será el array ordenado.
return(Numeros)
```

```
| ORDENATOR |

Todos los numeros que quieras los ordenare
Primero indicame el numero de numeros a introducir
: 8
Perfecto! Ahora a introducir los números
Numero 1/8:
: -2
Numero 2/8:
: 4
Numero 3/8:
: -7
Numero 4/8:
: -12
Numero 5/8:
: -26
Numero 6/8:
: 25
Numero 7/8:
: 27
Numero 8/8:
: 23
Lista de numeros desordenada:
[
    -2,  4, -7, -12,
    -26, 25, 27,  23
]
La solución es: -26,-12,-7,-2,4,23,25,27
Pulse enter para continuar
: 
```


- Ejercicios de objeto.

- Ejercicio 4: Presupuesto Coche

```
// En esta función crearemos un objeto coche al cual le crearemos un método para calcular su precio.
// El usuario irá insertando las características que quiera
// para el coche y el método calculará el respectivo precio.
case 4: {
    console.log("| CostaCoche |")
    console.log(" ")
    class Coche {
        Ruedas: Number;
        Marca: Number;
        Cristales: Number;

        constructor( Ruedas: Number, Marca: Number, Cristales: Number) {
            this.Ruedas=Ruedas;
            this.Marca=Marca;
            this.Cristales=Cristales;
        }

        decirCoche() {
            console.log(this.Ruedas, this.Marca, this.Cristales)
        }
    }

    // Aquí calcularíamos el precio, básicamente el usuario introducirá para cada una de las variables
    // del objeto un número del 1 al 3 que corresponderá a la característica que quiera el usuario para
    // cada una de esas variables. Aquí simplemente iremos sumando los precios de lo elegido por el usuario
    // y devolveremos la solución mediante un return.
    calculadoraPrecio() {
        var Presupuesto=0
        if (this.Ruedas==1) {
            Presupuesto=10
        } else if (this.Ruedas==2) {
            Presupuesto=5
        } else if (this.Ruedas==3) {
            Presupuesto=20
        }

        if (this.Marca==1) {
            Presupuesto=Presupuesto + 15
        } else if (this.Marca==2) {
            Presupuesto=Presupuesto + 25
        } else if (this.Marca==3) {
            Presupuesto=Presupuesto + 20
        }

        if (this.Cristales==1) {
            Presupuesto=Presupuesto + 15
        } else if (this.Cristales==2) {
            Presupuesto=Presupuesto + 25
        } else if (this.Cristales==3) {
            Presupuesto=Presupuesto + 5
        }
    }
}
```

```
        return Presupuesto
    }
}
console.log("Bienvenido! Aquí podrá indicar que tipo de coche quiere")
console.log("En función de lo que elija el precio variará")
console.log("Introduzca el número en función de lo que quiera")
console.log(" ")
// Aquí comenzaremos con la selección de las características.
console.log("Neumáticos:")
console.log("1: Robustos (10$) | 2: Ligeros (5$) | 3: TodoTerrenno (20$)")
let Ruedas = parseInt(await(leerTeclado(' ')))
console.log("1: Mazda (15$) | 2: Tesla (25$) | 3: BMW (20$)")
let Marca = parseInt(await(leerTeclado(' ')))
console.log("1: Tintados (15$) | 2: Antibalas (25$) | 3: Normales (5$)")
let Cristales = parseInt(await(leerTeclado(' ')))

// Aquí instanciaremos la clase coche y tras esto calcularemos el precio
// llamando al método correspondiente.
let MiCoche = new Coche(Ruedas, Marca, Cristales)
solucion=await(MiCoche.calculadoraPrecio())
break;
}
```

| CosteCoche |

```
Bienvenido! Aquí podrá indicar que tipo de coche quiere
En función de lo que elija el precio variará
Introduzca el número en función de lo que quiera

Neumáticos:
1: Robustos (10$) | 2: Ligeros (5$) | 3: TodoTerrenno (20$)
: 2
1: Mazda (15$) | 2: Tesla (25$) | 3: BMW (20$)
: 2
1: Tintados (15$) | 2: Antibalas (25$) | 3: Normales (5$)
: 1
La solución es: 45
Pulse enter para continuar
: █
```

- Ejercicios de fechas.
- **Ejercicio 5: Operadora de fechas**

```
// En este ejercicio pediremos 2 fechas y mediante estas fechas calcularemos cuantos años, meses
// y días hay de diferencia entre ambas.
case 5: {
    console.log("| FECHAS |")
    console.log(" ")
    console.log("Inserte dos fechas para saber los días que hay entre ambas")
    console.log("aaaa-mm-dd")
    // Pediremos las fechas las cuales deberán respetar cierto formato.
    let numero1 = await(leerTeclado(' '))
    let numero2 = await(leerTeclado(' '))
    // Recibiremos las fechas como String así que ahora almacenaremos en dos variables las
    // fechas obtenidas a partir de los strings originales mediante la función new Date()
    let fecha1 = new Date(numero1)
    let fecha2 = new Date(numero2)
    // Ahora nos disponemos a realizar los cálculos, en concreto calcularemos la diferencia entre las
    // fechas, nos devolverá un valor extremadamente grande el cual pasaremos de milisegundos a días totales.
    var diff = Math.abs(fecha1.getTime() - fecha2.getTime());
    solucion = Math.ceil(diff / (1000 * 3600 * 24));
    // Tras esto, a partir de los días calcularemos los años, meses y días restantes
    // y lo mandaremos mediante return.
    var años= Math.trunc(solucion/365)
    var meses= Math.trunc((solucion-años*365)/30)
    var días=Math.trunc(solucion-meses*30-años*365)
    return "Hay entre ambas fechas "+años+" años "+meses+" meses y "+días+" días"
    break;
}
```

| FECHAS |

Inserte dos fechas para saber los días que hay entre ambas

aaaa-mm-dd

: 2002-06-29

: 2005-07-21

La solución es: Hay entre ambas fechas 3 años 0 meses y 23 días

Pulse enter para continuar

:

- Ejercicio 6: Cumpleaños calculador

```
// En esta función pediremos cuantos meses y días quedan para el cumpleaños del usuario.
case 6: {
  console.log(`| CUMPLECALCULADOR |`)
  console.log (" ")
  console.log ("Inserte la fecha de su cumpleaños")
  console.log ("Automáticamente le dirá cuantos días quedan")
  console.log ("mes-día")
  // Aquí pediremos que el usuario inserte su cumpleaños pero tan solo mes y día.
  let numero1 = await(leerTeclado(' '))
  // Declaramos la fecha de hoy mediante "new Date()" sin parámetro y en caso de que hoy sea
  // mayor que la fecha de cumpleaños le asignaremos el valor "2022" al año en lugar de "2021"
  // que hemos dejado por defecto en un origen.
  let numeroedit="2021-"+numero1
  let fechaCumple = new Date(numeroedit)
  let hoy = new Date()
  if (hoy.getTime() > fechaCumple.getTime()) {
    let numeroedit="2022-"+numero1
    fechaCumple = new Date(numeroedit)
  }
  // Ahora como en el ejercicio anterior calcularemos la diferencia, la pasaremos a años, meses y días
  // Y como detalle, en caso de que hoy sea su cumpleaños le felicitaremos :).
  var diff = Math.abs(fechaCumple.getTime() - hoy.getTime());
  var calculo = Math.ceil(diff / (1000 * 3600 * 24));
  if (calculo==1 || calculo==365) {
    console.log("Feliz cumpleaños! :)")
  }
  var meses= Math.trunc(calculo/30)
  var dias=Math.trunc(calculo-(meses*30))
  return "Quedan todavía "+meses+" meses y "+dias+" días"
  break
}
```

| CUMPLECALCULADOR |

Inserte la fecha de su cumpleaños

Automáticamente le dirá cuantos días quedan

mes-día

: 06-29

La solución es: Quedan todavía 8 meses y 25 días

Pulse enter para continuar

:

- Ejercicio 7: Calculadora Edad

```
// En esta última función calcularemos la edad del usuario a partir de su cumpleaños.
case 7: {
  console.log("| CALCULADORA EDAD |")
  console.log (" ")
  console.log ("Inserte la fecha de su cumpleaños")
  console.log ("aaaa-mm-dd")
  // Lo primero será pedirle que inserte su fecha de cumpleaños la cual pasaremos a fecha.
  let numero1 = await(leerTeclado(' '))
  let fechaNacimiento = new Date(numero1)
  let hoy=new Date()
  // Esta vez calcularemos la diferencia del día de hoy con su día de nacimiento y lo pasaremos a
  // años, meses y días para saber cuanto ha vivido exactamente.
  var diff = Math.abs(fechaNacimiento.getTime() - hoy.getTime());
  var calculo = Math.ceil(diff / (1000 * 3600 * 24));
  var años= Math.trunc(calculo/365)
  var meses= Math.trunc((calculo-años*365)/30)
  var dias=Math.trunc(calculo-meses*30-años*365)
  return "Has vivido "+años+" años, "+meses+" meses y "+dias+" días"
}
```

| CALCULADORA EDAD |

Inserte la fecha de su cumpleaños

aaaa-mm-dd

: 2002-06-29

La solución es: Has vivido 19 años, 3 meses y 16 días

Pulse enter para continuar

:

- Recursos de programación empleados.

- Método de leerTeclado de “entradaTeclado.js”

```
1 import { fail } from 'assert'
2 import { parse } from 'path/posix'
3 import readline from 'readline'
4 let readlineI: readline.Interface
5
6 let leelinea = (prompt: string) => {
7     readlineI = readline.createInterface({
8         input: process.stdin,
9         output: process.stdout,
10    })
11    return new Promise<string>((resuelta: any, rechazada: any) => {
12        readlineI.question(`${prompt}: `, (cadenaEntrada: string) => {
13            resuelta(cadenaEntrada)
14        })
15    })
16 }
17
18 export let leerTeclado = async (prompt: string) => {
19     let entrada = await leelinea(prompt)
20     readlineI.close()
21     return entrada
22 }
```

- Comandos:

- import <método> from <url>

```
import {leerTeclado} from "../util/entradaTeclado";
import {calculos} from "../util/calculos";
```

- console.log(<variable>/<"texto">)
- funciones asíncronas

```
let main = async () => {
```

- Llamadas a funciones de forma asíncrona (await())
- Utilización de parseInt(variable)

```
let entrada = parseInt(await leerTeclado (''))
```

- Declaración y utilización de Arrays (Array.push, Array[index])

```
var primos = []
```

```
if (primo==true) {
    primos.push(i)
}
```

```
if(numero1%divisores[o]==0) {
```

- Utilización de operadores de fechas y de funciones de la clase Math (Math.abs, Math.ceil, Math.trunc)

```
let fecha1 = new Date(numero1)
let fecha2 = new Date(numero2)
```

```
let hoy = new Date()
```

```
if (hoy.getTime() > fechaCumple.getTime()) {
```

```
var diff = Math.abs(fecha1.getTime() - fecha2.getTime());
solucion = Math.ceil(diff / (1000 * 3600 * 24));
```

```
var años= Math.trunc(solucion/365)
```

- Bucles y Sentencias:

- Bucles while

```
while(1) {
```

- Bucles for

```
for (i=0; i<numeroNum-1; i++) {
    for (e=0; e<(numeroNum-i); e++) {
```

- Sentencias if

```
if(Numeros[e]>Numeros[e+1]) {
```

```
if (this.Cristales==1) {
    Presupuesto=Presupuesto + 15
} else if (this.Cristales==2) {
    Presupuesto=Presupuesto + 25
} else if (this.Cristales==3) {
    Presupuesto=Presupuesto + 5
}
```

- Sentencias Switch Case

```
switch(entrada) {

    // Establecemos un default para que todo lo que no sea un número del 1-7 provoque que salgamos del proyecto.
    default: {
        console.log("Hasta otra!")
        return "exit"
    }

    // La primera función será calcular los números primos entre dos números que servirán de delimitadores.
    case 1: {
```

- Bucles do while

```
do {
    if(numero1%divisores[o]==0) {
        console.log(numero1+" | "+divisores[o])
        numero1=numero1/divisores[o]
        calculados.push(divisores[o])
    } else {
        o++
    }
} while(numero1>1);
```

- **Objetos y clases:**

- Creación de un objeto (sus variables y método constructor)

```
class Coche {  
    Ruedas: Number;  
    Marca: Number;  
    Cristales: Number;  
  
    constructor( Ruedas: Number, Marca: Number, Cristales: Number) {  
        this.Ruedas=Ruedas;  
        this.Marca=Marca;  
        this.Cristales=Cristales;  
    }  
}
```

- Creación de métodos de objeto

```
decirCoche() {  
    console.log(this.Ruedas, this.Marca, this.Cristales)  
}
```

- Instanciar un objeto y llamar a un método del objeto

```
let MiCoche = new Coche(Ruedas, Marca, Cristales)  
solucion=await(MiCoche.calculadoraPrecio())
```