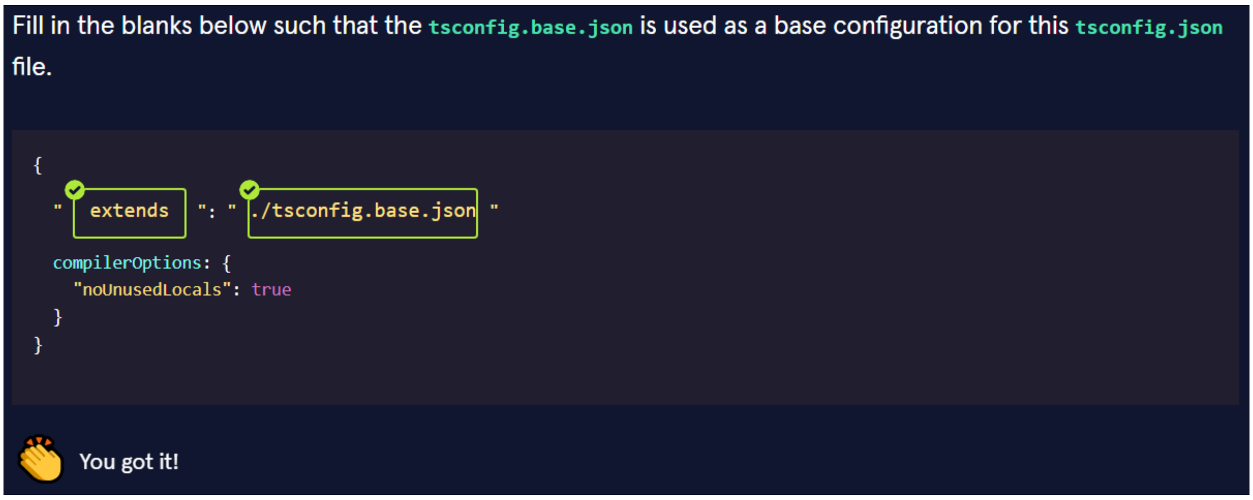
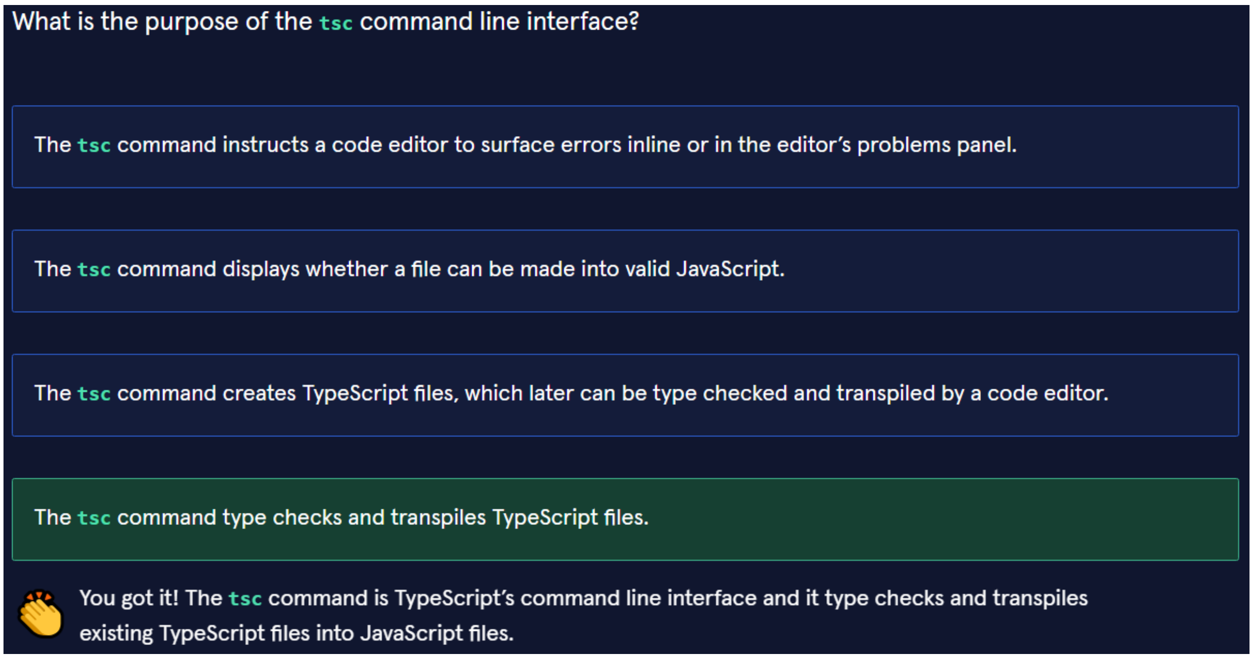
QUIZ



Fill in the blanks below such that the `tsconfig.base.json` is used as a base configuration for this `tsconfig.json` file.

```
{
  " extends ": " ./tsconfig.base.json "
  compilerOptions: {
    "noUnusedLocals": true
  }
}
```

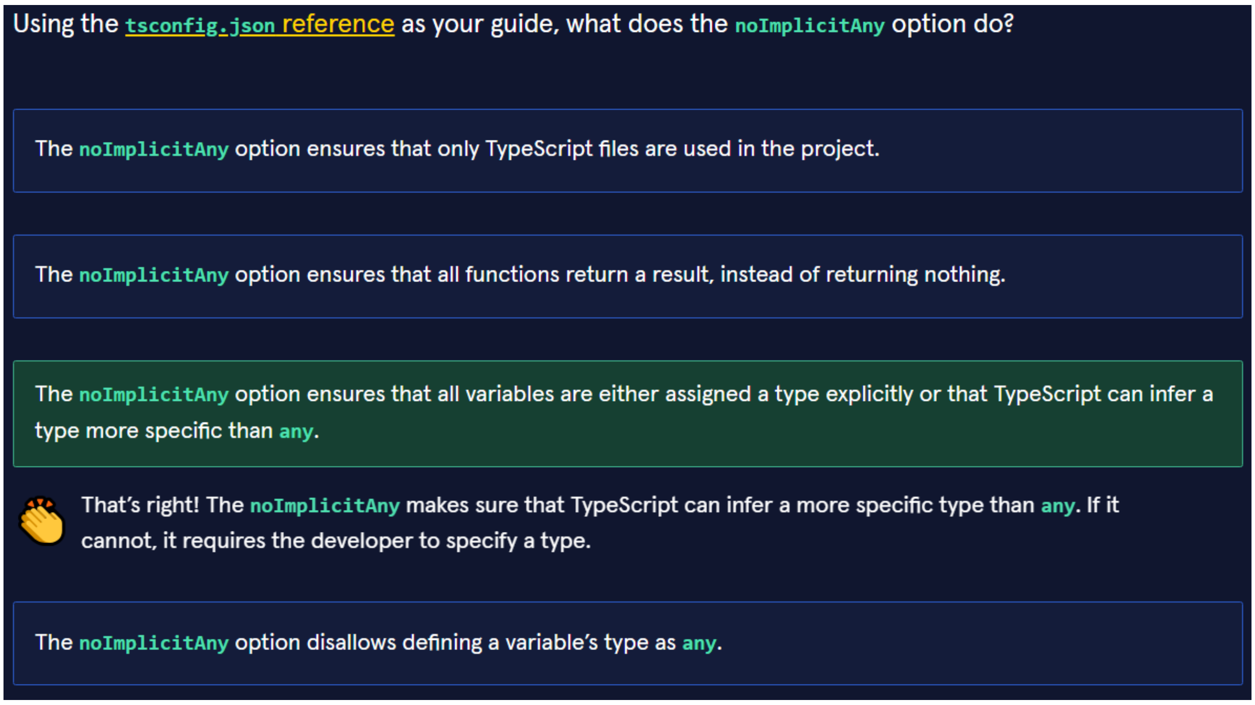
You got it!



What is the purpose of the `tsc` command line interface?

- The `tsc` command instructs a code editor to surface errors inline or in the editor's problems panel.
- The `tsc` command displays whether a file can be made into valid JavaScript.
- The `tsc` command creates TypeScript files, which later can be type checked and transpiled by a code editor.
- [✓ The `tsc` command type checks and transpiles TypeScript files.]

You got it! The `tsc` command is TypeScript's command line interface and it type checks and transpiles existing TypeScript files into JavaScript files.



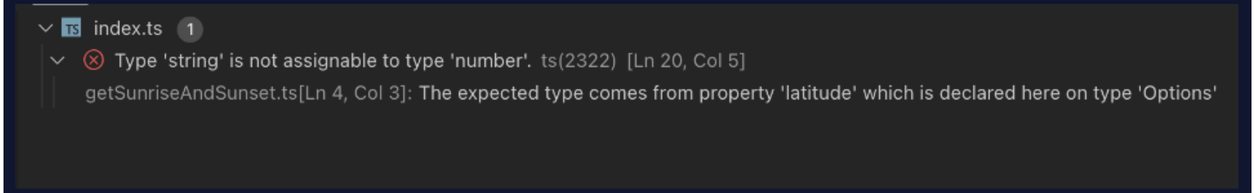
Using the `tsconfig.json` reference as your guide, what does the `noImplicitAny` option do?

- The `noImplicitAny` option ensures that only TypeScript files are used in the project.
- The `noImplicitAny` option ensures that all functions return a result, instead of returning nothing.
- [✓ The `noImplicitAny` option ensures that all variables are either assigned a type explicitly or that TypeScript can infer a type more specific than `any`.]

That's right! The `noImplicitAny` makes sure that TypeScript can infer a more specific type than `any`. If it cannot, it requires the developer to specify a type.

- The `noImplicitAny` option disallows defining a variable's type as `any`.

The image shows a TypeScript error inside of VSCode's problems panel. Identify (1) which file contains the TypeScript error and (2) which type definition the error is erroring against.



```
index.ts 1
  Type 'string' is not assignable to type 'number'. ts(2322) [Ln 20, Col 5]
  getSunriseAndSunset.ts[Ln 4, Col 3]: The expected type comes from property 'latitude' which is declared here on type 'Options'
```

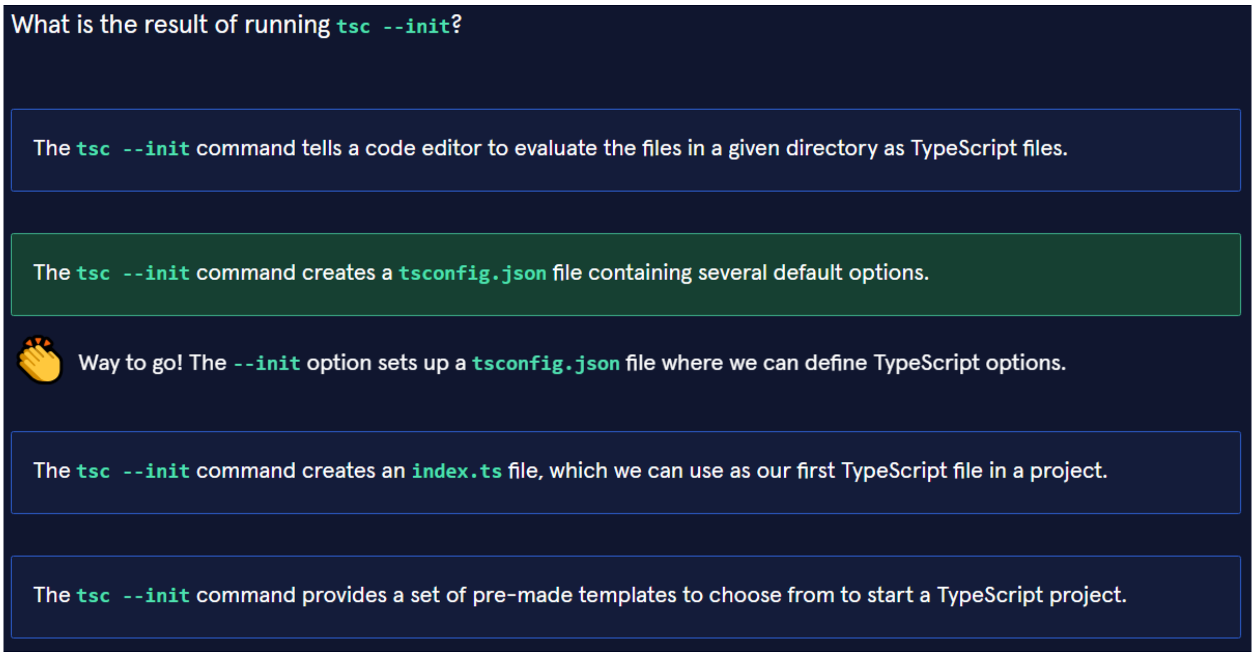
- (1) `index.ts`, (2) `string`
- (1) `getSunriseAndSunset.ts`, (2) `Options`
- (1) `index.ts`, (2) `latitude`
- [✓ (1) `index.ts`, (2) `Options`]

👏 Very good! The `index.ts` file has a variable that is typed as a `string` while the `Options` type expects a `number`.

We'd like TypeScript to check for and warn us if any unused parameters exist in a TypeScript file. Fill in the blanks below to complete the code such that `tsc` will check for unused function parameters. Check the documentation to find the proper configuration option.

```
tsc --noUnusedParameters
```

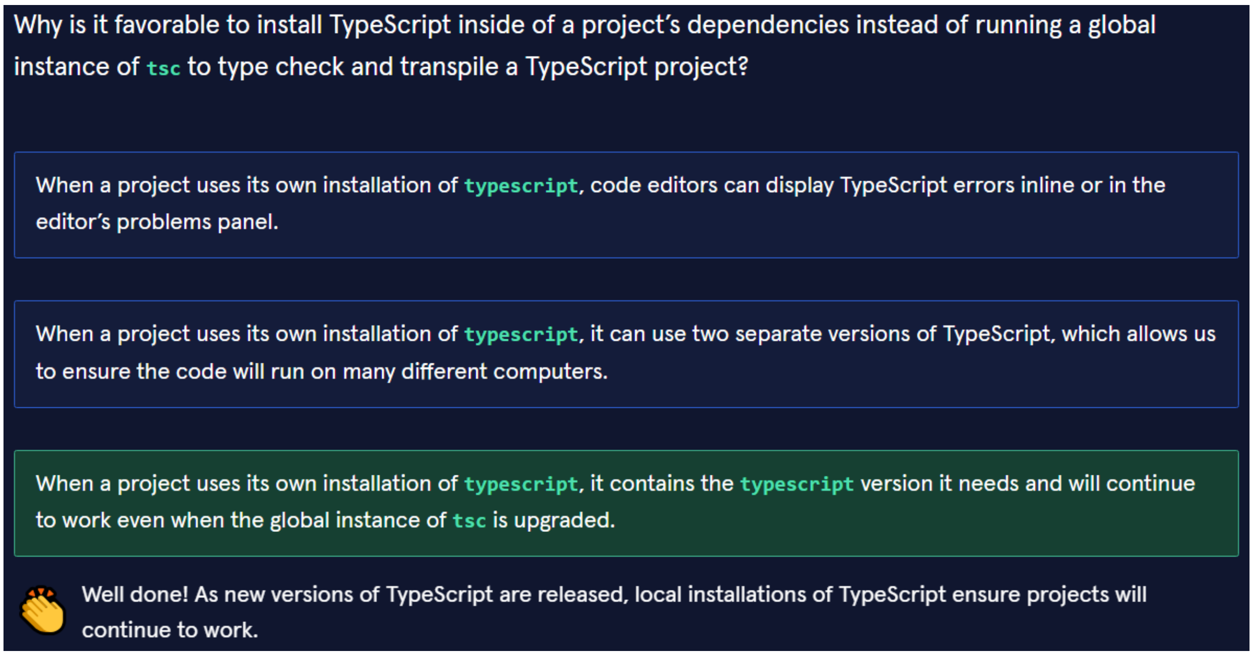
👏 You got it!


What is the result of running `tsc --init`?

- The `tsc --init` command tells a code editor to evaluate the files in a given directory as TypeScript files.
- [✓ The `tsc --init` command creates a `tsconfig.json` file containing several default options.]

👏 Way to go! The `--init` option sets up a `tsconfig.json` file where we can define TypeScript options.

- The `tsc --init` command creates an `index.ts` file, which we can use as our first TypeScript file in a project.
- The `tsc --init` command provides a set of pre-made templates to choose from to start a TypeScript project.


Why is it favorable to install TypeScript inside of a project's dependencies instead of running a global instance of `tsc` to type check and transpile a TypeScript project?

- When a project uses its own installation of `typescript`, code editors can display TypeScript errors inline or in the editor's problems panel.
- When a project uses its own installation of `typescript`, it can use two separate versions of TypeScript, which allows us to ensure the code will run on many different computers.
- [✓ When a project uses its own installation of `typescript`, it contains the `typescript` version it needs and will continue to work even when the global instance of `tsc` is upgraded.]

👏 Well done! As new versions of TypeScript are released, local installations of TypeScript ensure projects will continue to work.