

1 - Instalación del editor Visual Studio Code(VSCode)

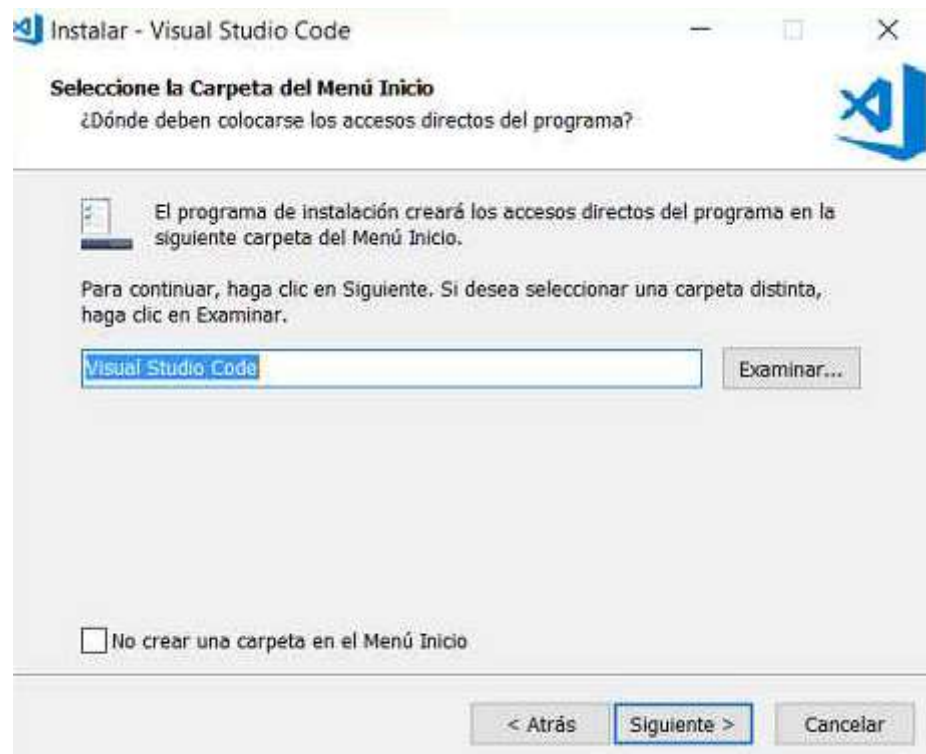
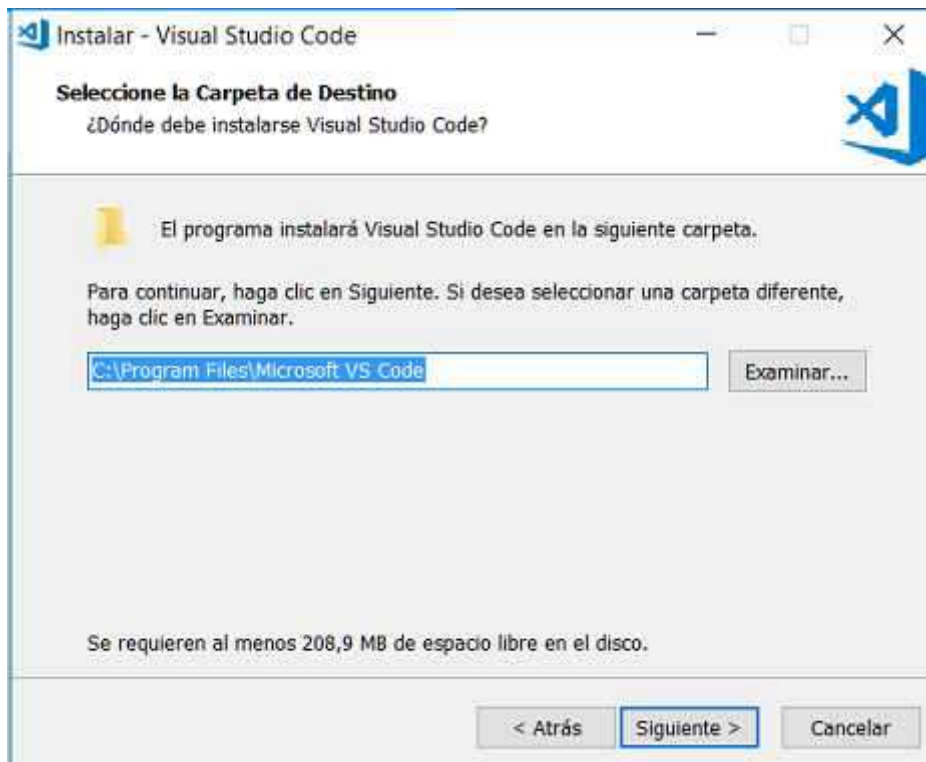
El editor de texto VSCode ha sido creado y es mantenido por Microsoft. Lo distribuye con licencia Open Source y por lo tanto en forma gratuita. Lo debemos descargar del sitio code.visualstudio.com , como podemos comprobar está disponible tanto para Windows, Mac y Linux(<https://code.visualstudio.com/>)

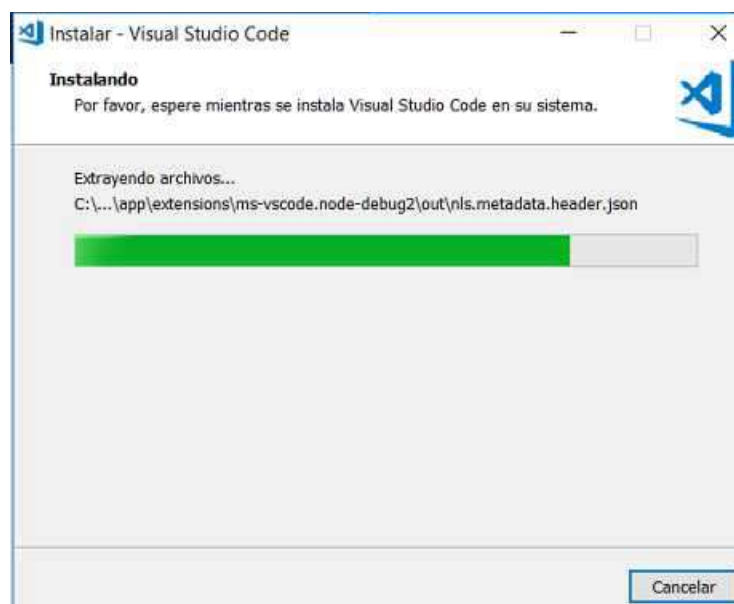
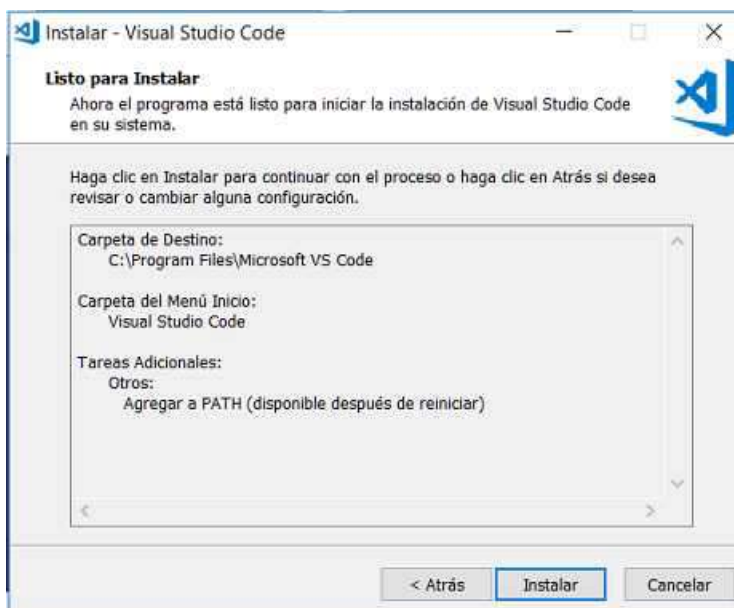
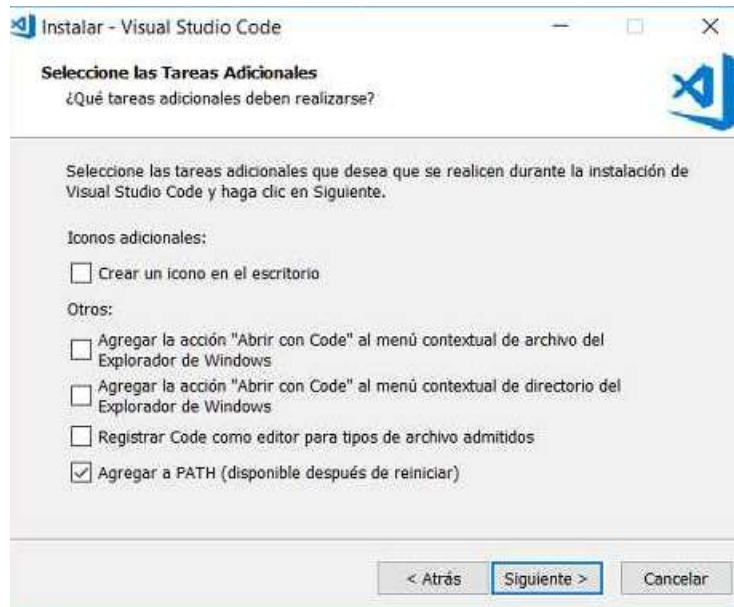


Estaremos trabajando en ambiente Windows (pero su empleo en otros sistemas operativos es similar)

Una vez descargado procedemos a instalarlo siguiendo los pasos del asistente:

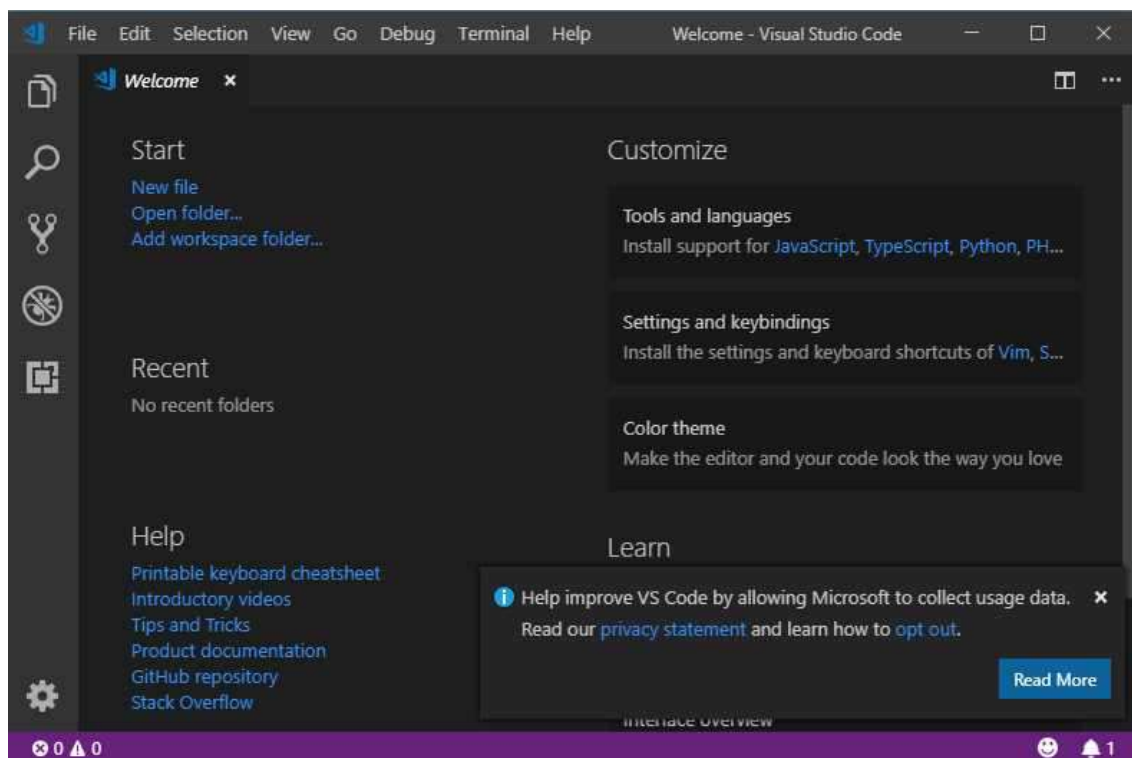








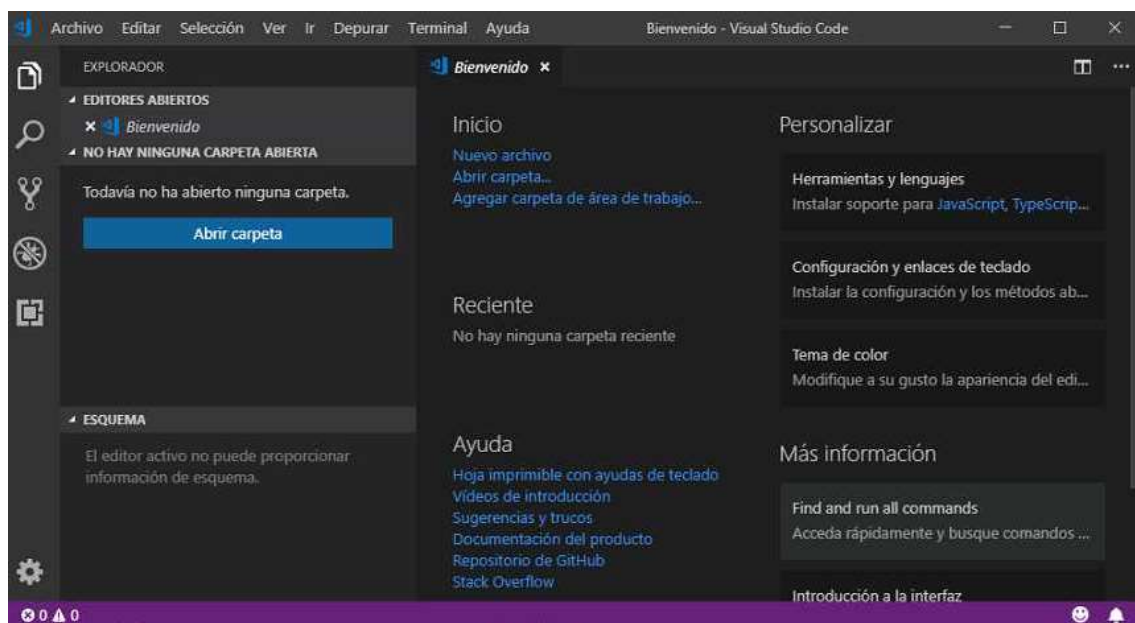
Ya tenemos instalado y funcionando VSCode:



El lenguaje es en Inglés, lo primero que haremos es cambiar a castellano, instalando la extensión de VS Code para tal fin. Presionamos el ícono de la izquierda para administrar extensiones y buscamos 'spanish':

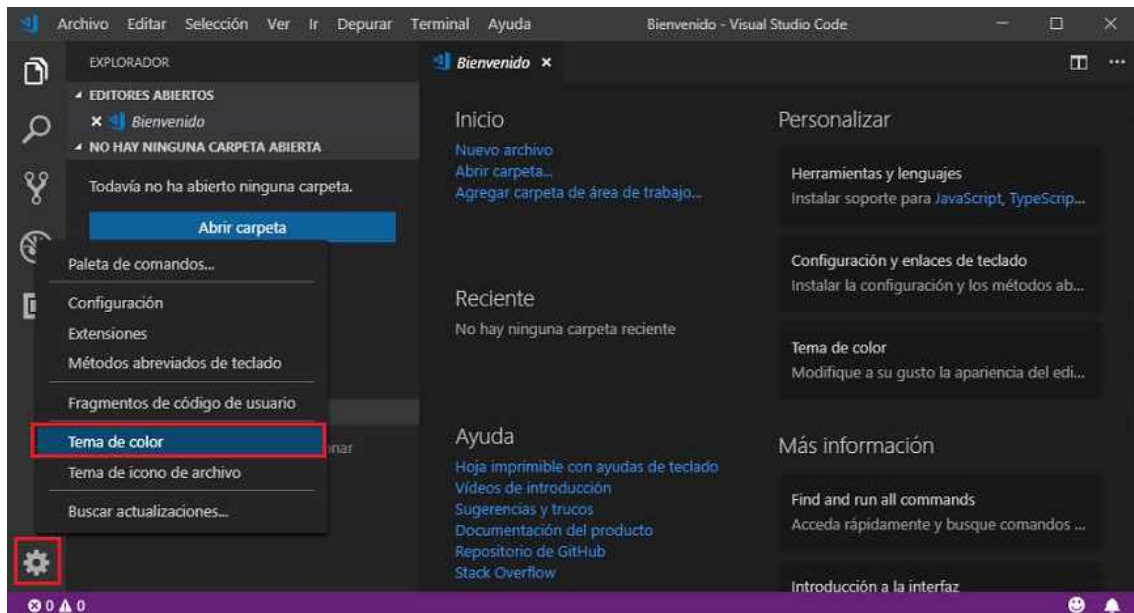


Luego de instalar la extensión debemos reiniciar el VS Code para que tenga efecto. Ahora ya tenemos el editor en castellano:

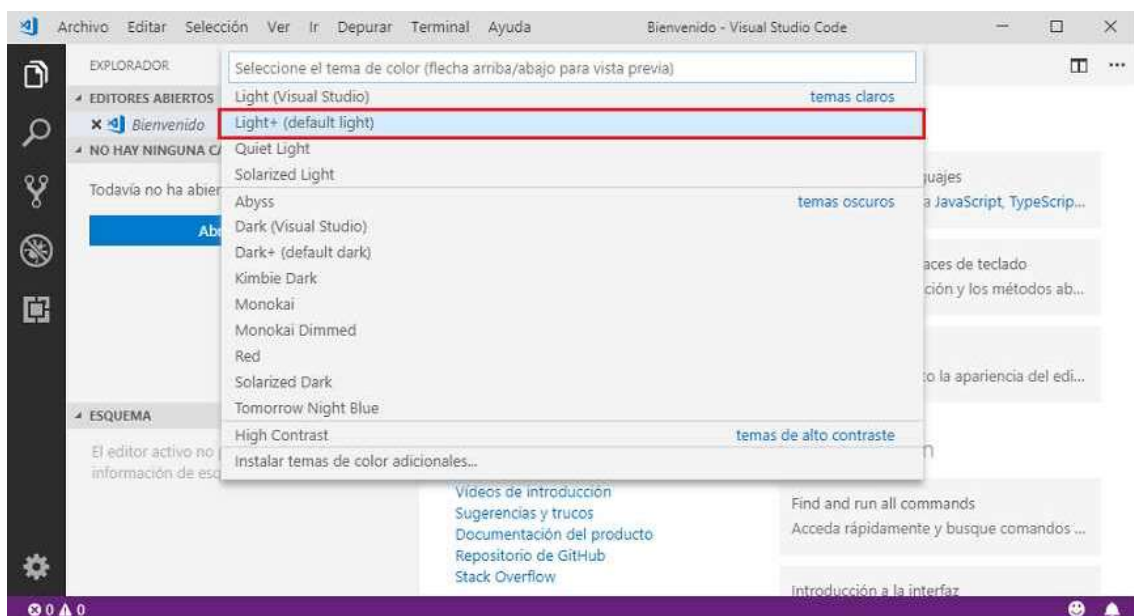


El tema o aspecto visual por defecto es el oscuro, pero para que se vea más fácil en este tutorial lo cambiaremos por un aspecto más claro.

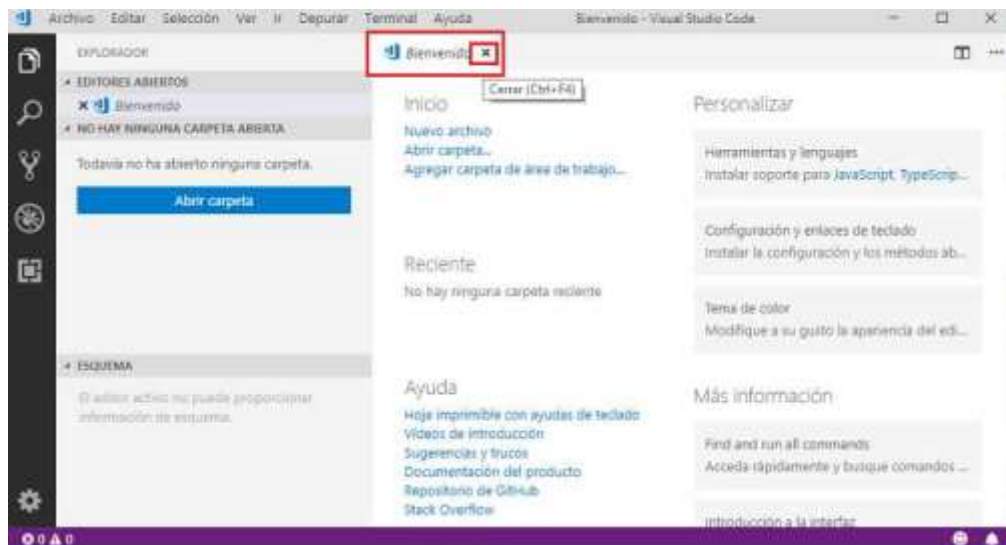
Para cambiar el tema presionamos el botón del engranaje en la parte inferior izquierda y seleccionamos "Tema de color":



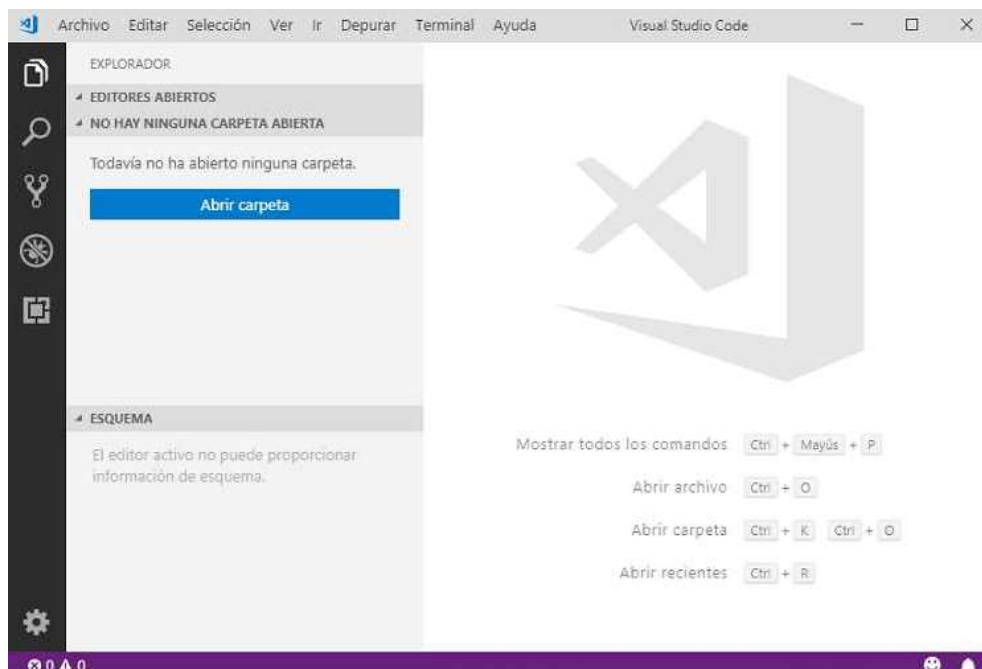
Yo seleccionaré el tema "Light + (default light)":



Podemos cerrar la pestaña de bienvenida y tendremos el editor VSCode en su mínima expresión:



El resultado sin pestañas abiertas es:



Como vemos en el fondo del editor aparecen los 4 comandos más comunes que tenemos que hacer normalmente cuando trabajamos con el editor de texto VSCode como puede ser "Abrir archivo" con el atajo de teclas Ctrl + O.

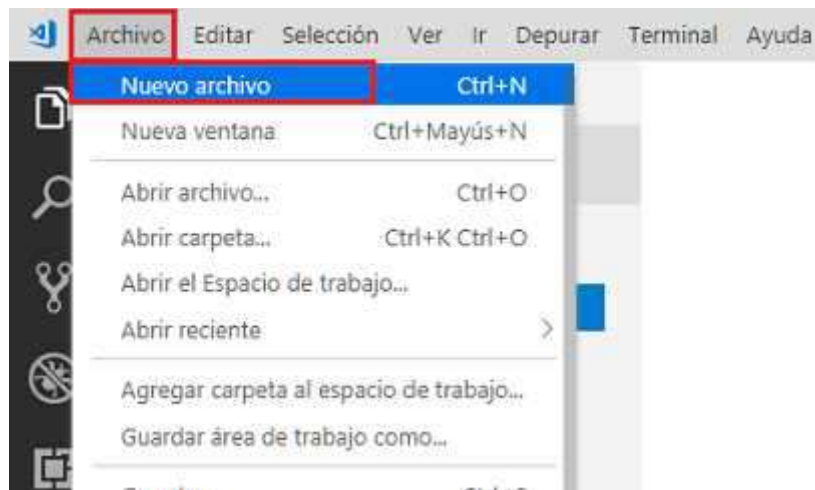
A lo largo del tutorial veremos los atajos de teclas más comunes y otros no tan comunes pero que nos pueden ayudar en un momento preciso.

2 - Archivo independiente : Creación, apertura y modificación

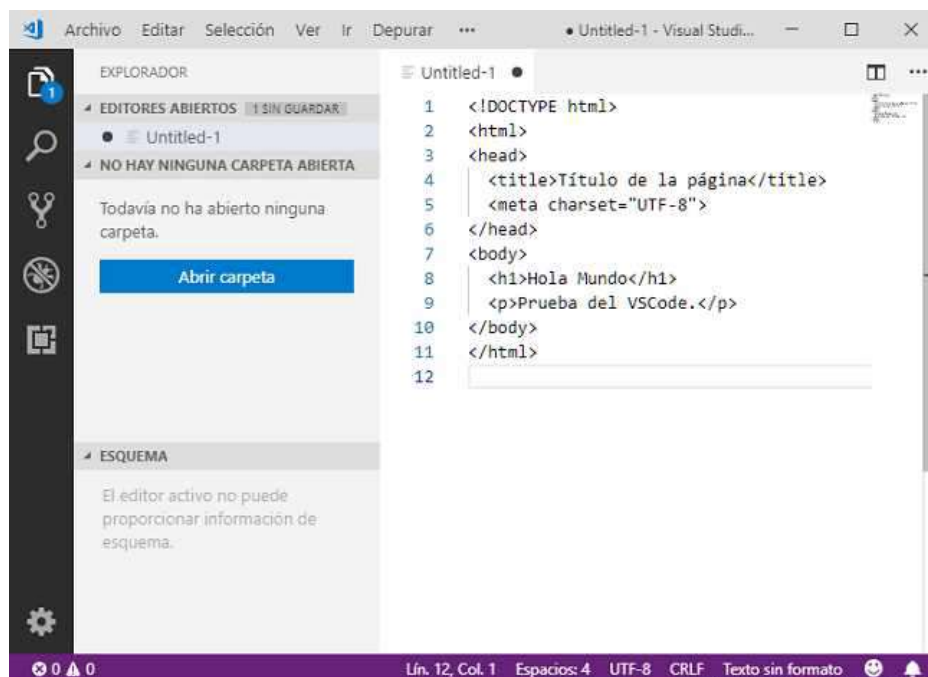
Las actividades básicas de todo editor de texto es permitirnos crear nuevos archivos y modificar archivos existentes.

Creación de un archivo

Desde el menú de opciones de VSCode seleccionamos "Archivo -> Nuevo archivo" (o mediante las teclas de atajo Ctrl + N):

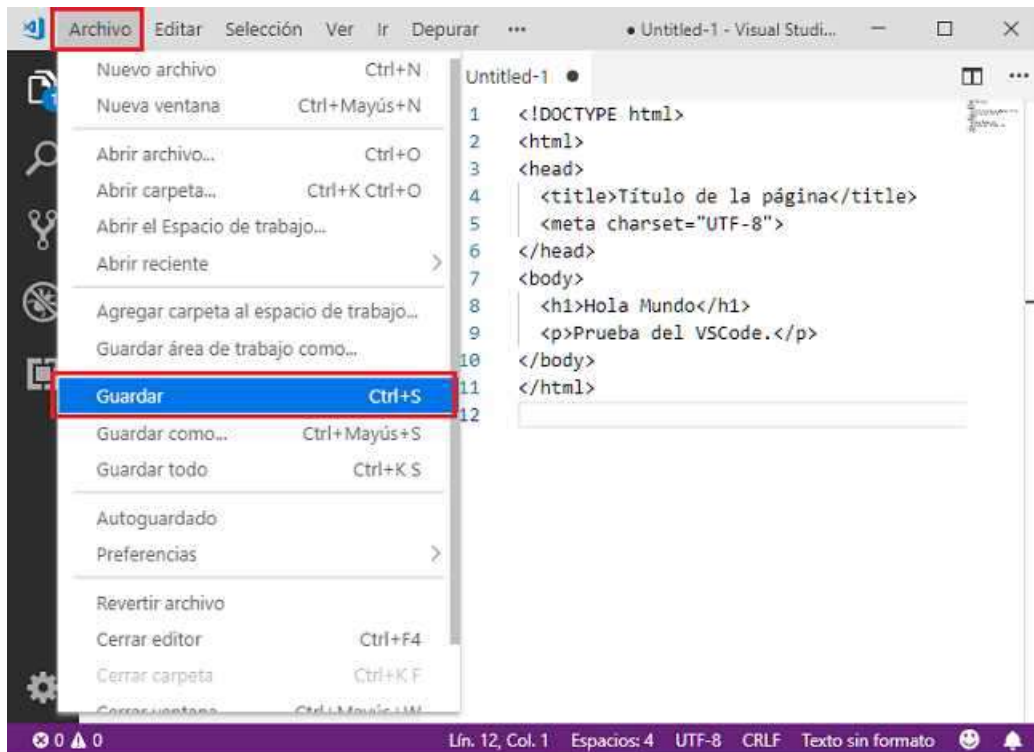


Ahora en la ventana de edición procederemos a codificar un archivo HTML básico:

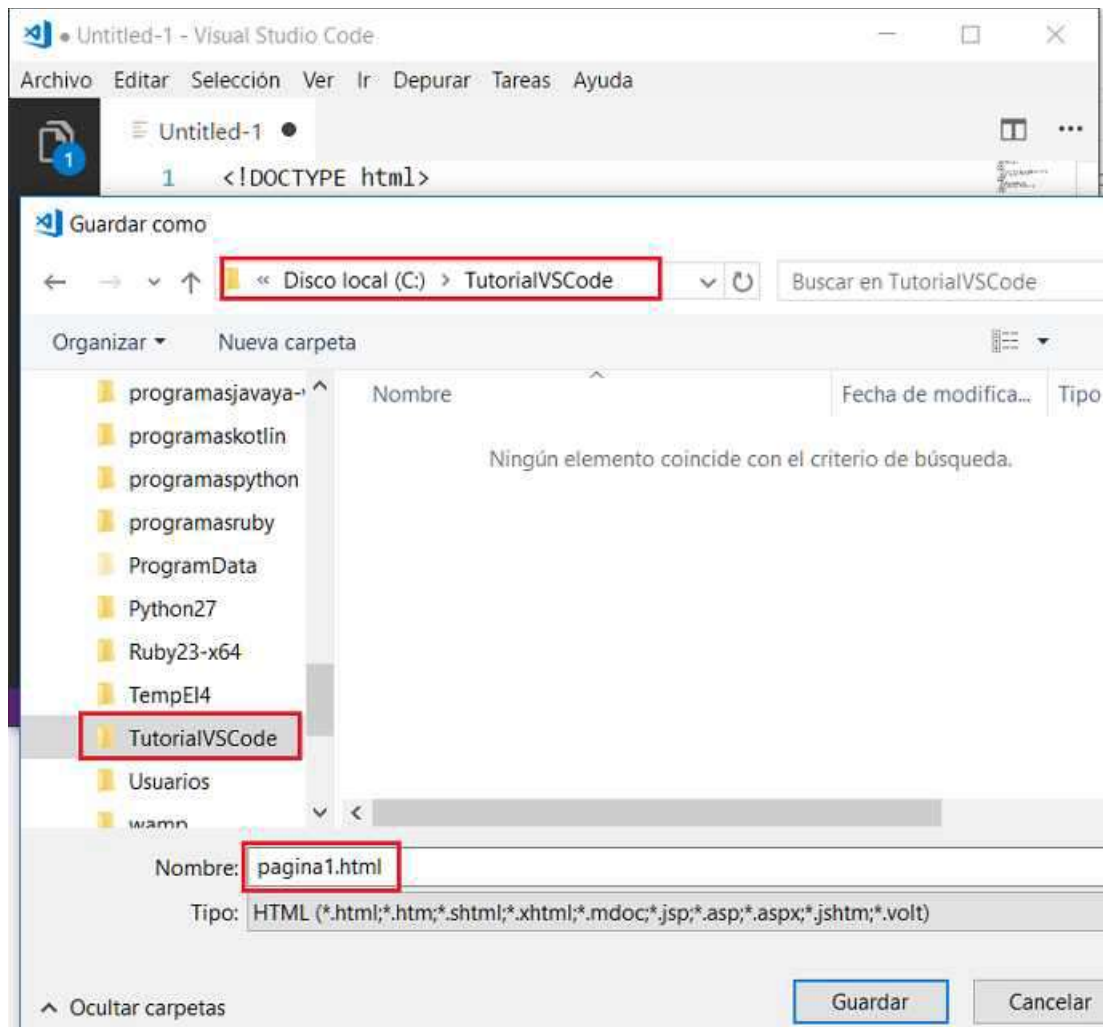


Todavía no aparecen las etiquetas HTML coloreadas ya que no hemos guardado el archivo en el disco duro donde indicaremos con la extensión el tipo de archivo.

Para grabar el archivo desde el menú de opciones elegimos "Archivo->guardar" (o mediante las teclas de atajo Ctrl + S):

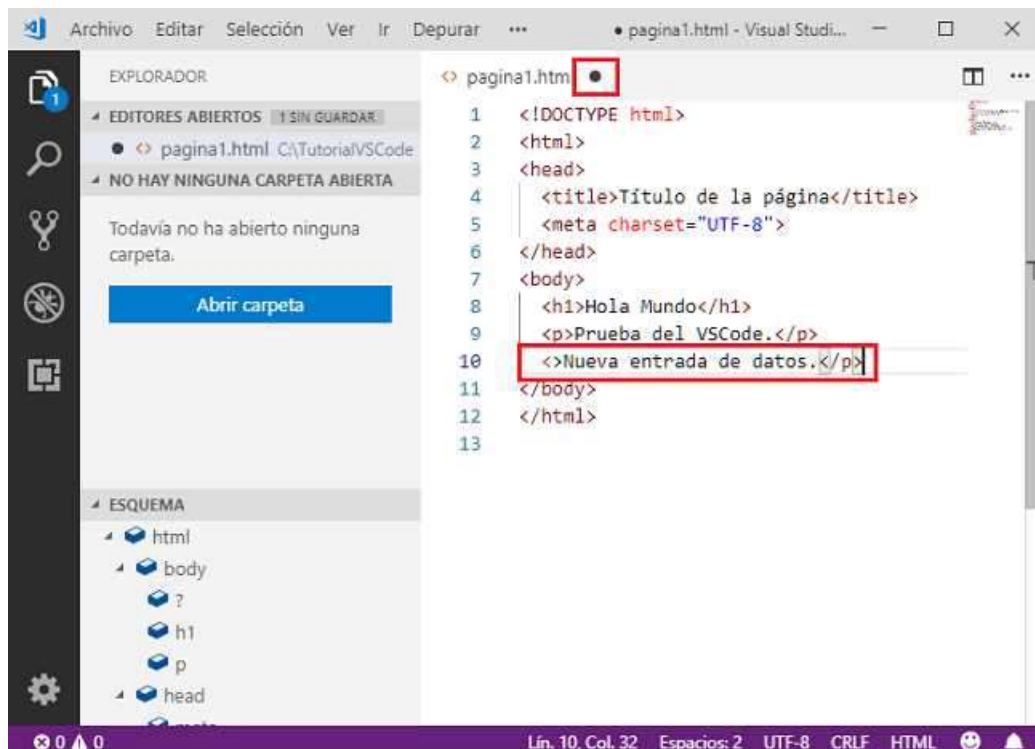


Ahora seleccionamos en el diálogo la carpeta y nombre de archivo que le asignaremos:



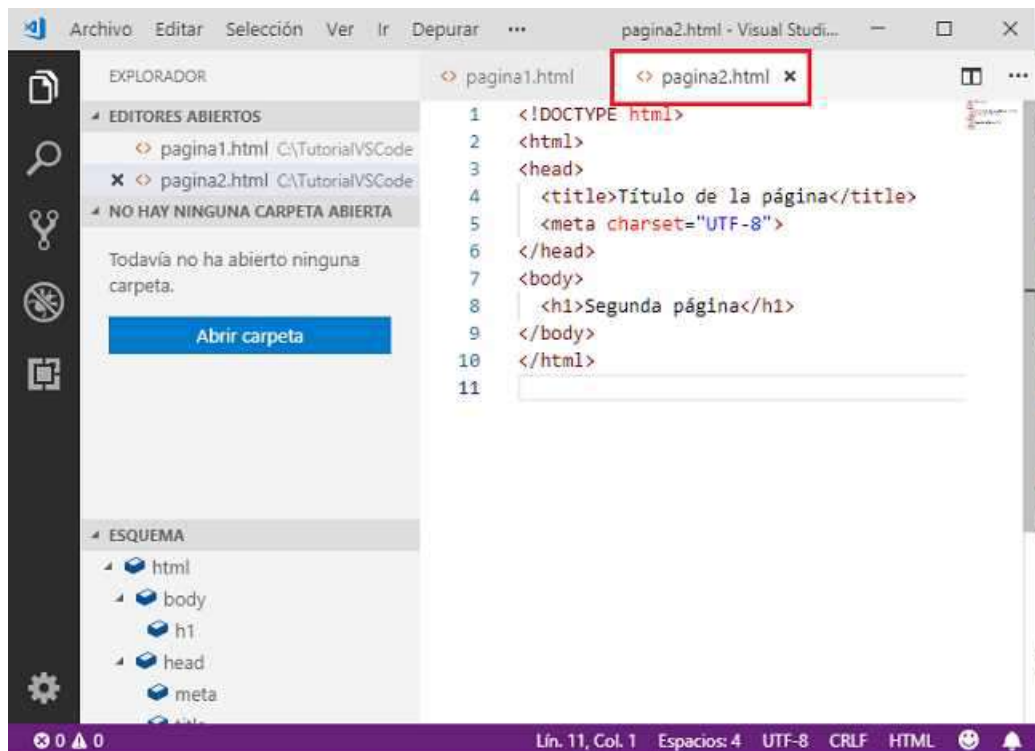
Después de ésta acción el contenido que hemos escrito en VSCode queda almacenado en el archivo 'pagina1.html' en la unidad C:\TutorialVSCoDe

Si introducimos un cambio con el editor VSCode podemos comprobar que en la pestaña cambia la cruz por un círculo indicando que debemos grabar los cambios introducidos:



Para actualizar los cambios en el archivo debemos seleccionar nuevamente la opción "Archivo guardar" o (Ctrl + S). Podemos crear varios archivos y tenerlos abiertos en forma simultánea, cada uno en una pestaña diferente.

Creemos nuestro segundo archivo dando los pasos que ya vimos "Archivo -> Nuevo archivo" o (Ctrl + N), luego escribamos nuestra segunda página HTML y la grabemos con la opción "Archivo->guardar" o (Ctrl + S):



Cambio entre pestañas

1. Mediante el mouse podemos seleccionar la pestaña del archivo que necesitemos editar.

2. También podemos cambiar entre pestañas mediante las teclas (Alt y la tecla de números):



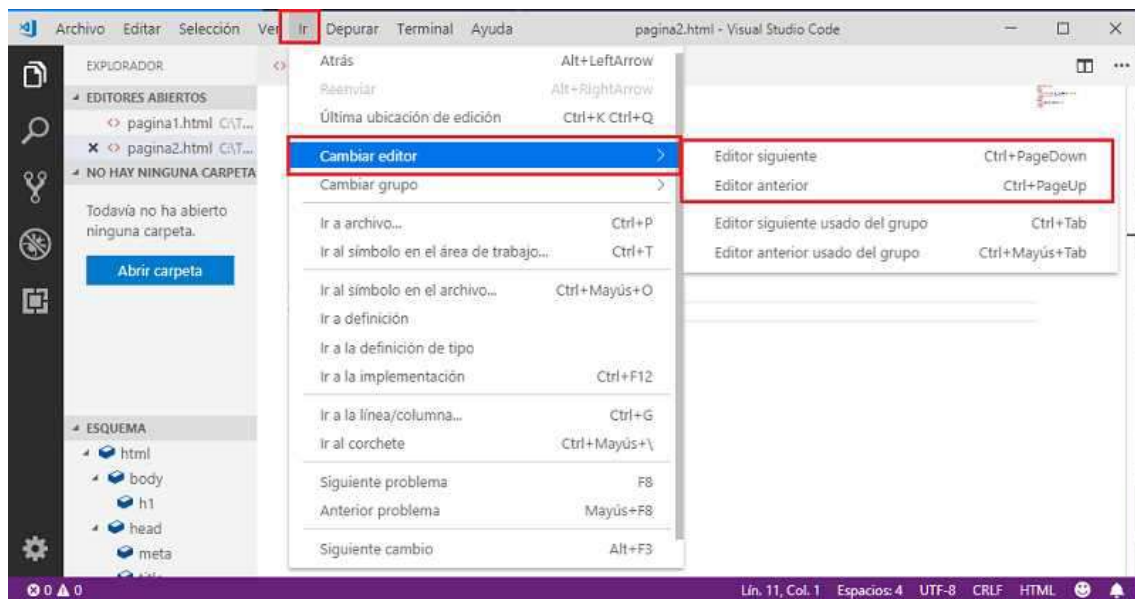
(Seleccionamos la segunda pestaña presionando Alt+2)

3. Podemos también cambiar entre pestañas mediante las teclas (Ctrl y alguna de las teclas Av Pag. y Rt Pag):



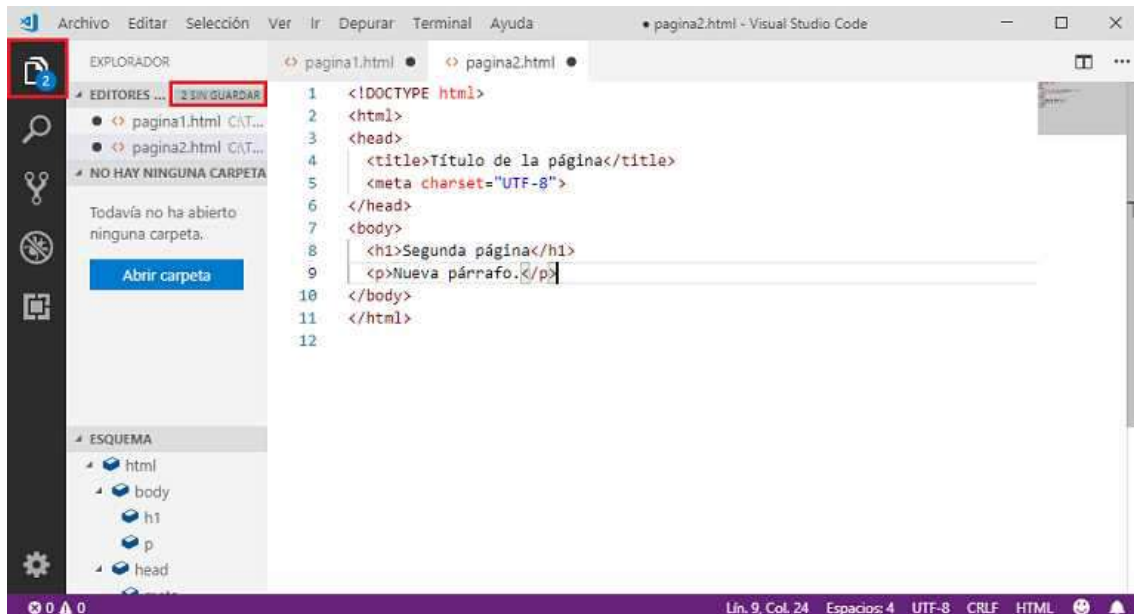
(Se van seleccionando las pestañas de izquierda a derecha o viceversa)

4. Por último desde el menú de opciones de VSCode podemos cambiar entre pestañas seleccionando alguna de las dos opciones: Ir -> Cambiar editor -> Editor siguiente o la Editor anterior:



Grabar todos los archivos modificados.

Para probar esta opción introduzcamos algunos cambios en las dos pestañas de los archivos 'pagina1.html' y 'pagina2.html'. Podemos ver en el primer ícono de la izquierda que hay dos archivos modificados sin grabar:



La opción del menú que graba todos los archivos modificados es (Archivo->Guardar todo) o el atajo de las teclas (Ctrl + K, luego soltamos las teclas control + K y presionamos finalmente la tecla S):



Cerrar pestañas.

Tenemos tres formas de cerrar una pestaña:

Mediante el mouse podemos presionar sobre la cruz de la pestaña (si no se ha grabado nos aparece un diálogo para confirmar que se guarden los cambios)

También podemos cerrar la pestaña activa presionando las teclas Ctrl + W:

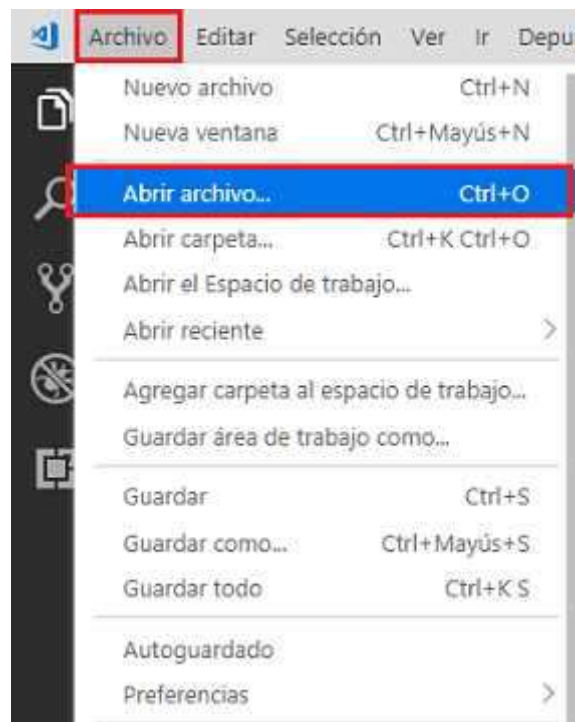


Cerrar la pestaña activa presionando las teclas Ctrl + F4:



Abrir archivos.

Para abrir un archivo almacenado en el disco podemos hacerlo seleccionando desde el menú de opciones: Archivo->Abrir archivo:



El atajo de teclas para que aparezca el diálogo de apertura de archivo es: Ctrl + O.

Crear otro archivo a partir de uno existente.

Es muy común tener que a partir de un archivo existente generar otro. Para esto debemos tener seleccionada la pestaña con el archivo y luego desde el menú de opciones seleccionar Archivo -> Guardar como... Problemas a partir del archivo 'pagina2.html' generar un archivo llamado 'pagina3.html' con el mismo contenido:



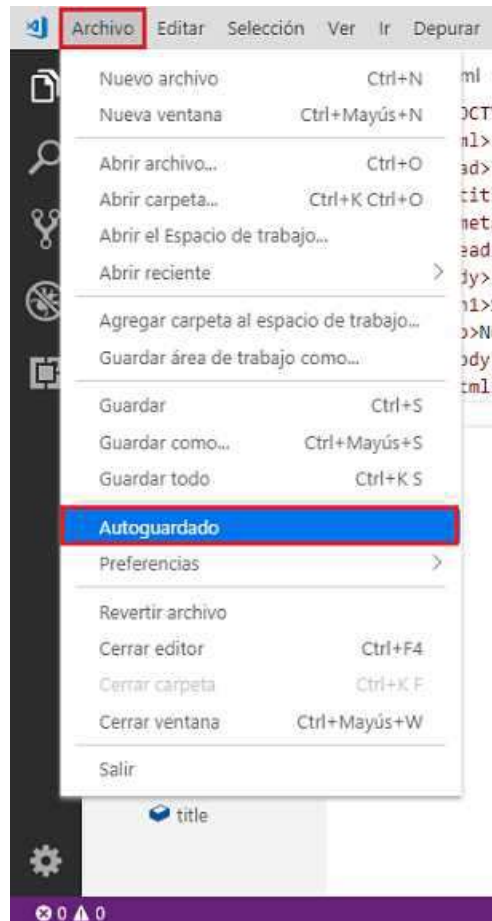
En el diálogo que aparece indicamos el nuevo nombre de archivo: 'pagina3.html'

Esta misma acción la podemos hacer mediante las teclas:



Guardado automático.

Si queremos desentendernos de la grabación de los archivos podemos activar la opción de "Autoguardado". Para activar esta funcionalidad en Visual Studio Code debemos ir a la opción de menú: Archivo -> Autoguardado y dejarla tildada (si la seleccionamos nuevamente se desactiva la opción):



A partir de ese momento cada cambio que hagamos a un archivo luego se ven reflejados en el disco donde se almacena (es decir no necesitamos ejecutar 'Archivo->Guardar')

3 - Carpeta : apertura y cerrado

Abrir carpeta

Vimos en el concepto anterior la metodología para editar archivos que nos presenta VSCode.

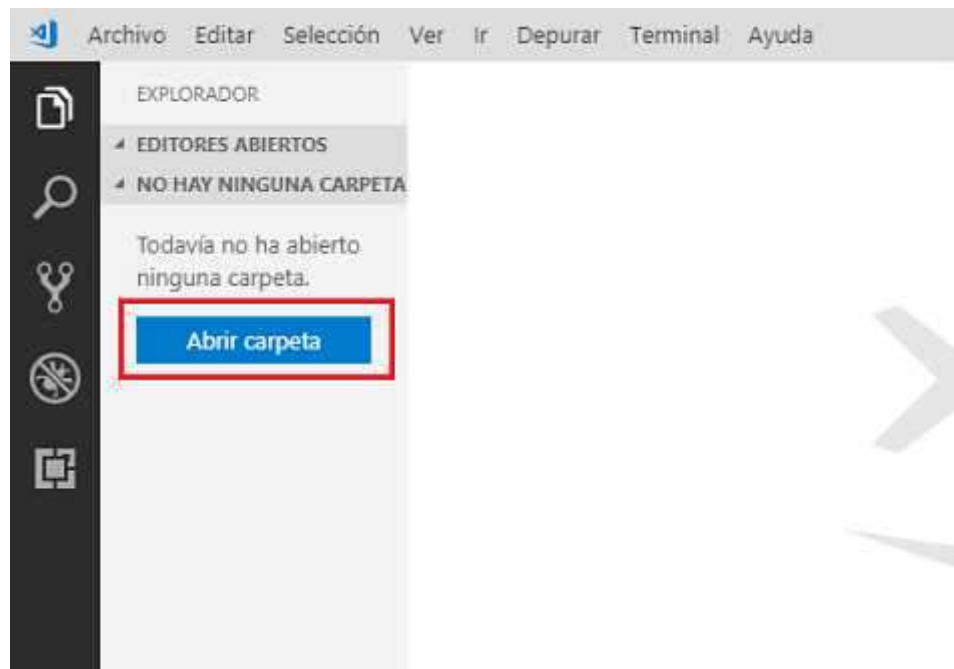
Cuando tenemos que trabajar con un conjunto de archivos que se encuentran en una carpeta lo más conveniente es utilizar la funcionalidad de "Abrir carpeta" para que se nos muestre la lista de archivos contenidos en la misma y no tener que abrir en forma individual cada archivo.

Hay varias formas de poder hacer esta actividad:

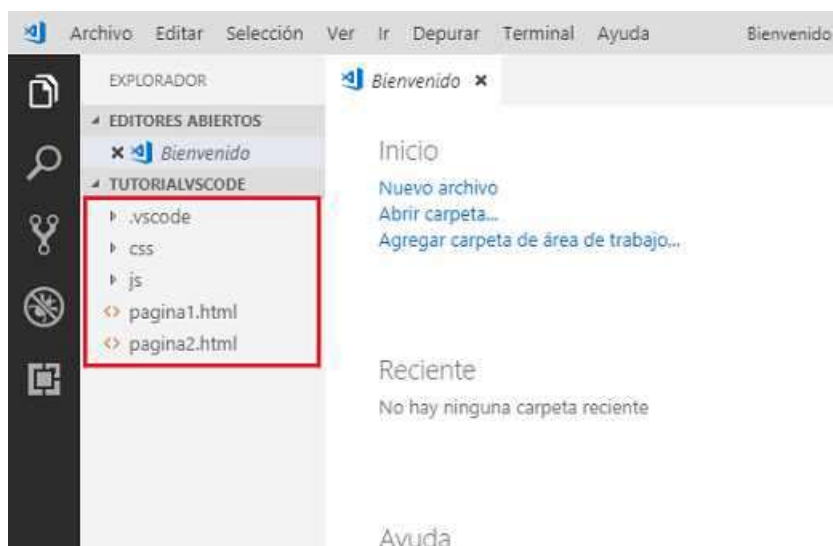
Presionar el botón de "Explorador" de la "barra de actividades" que se encuentra al lado izquierdo:



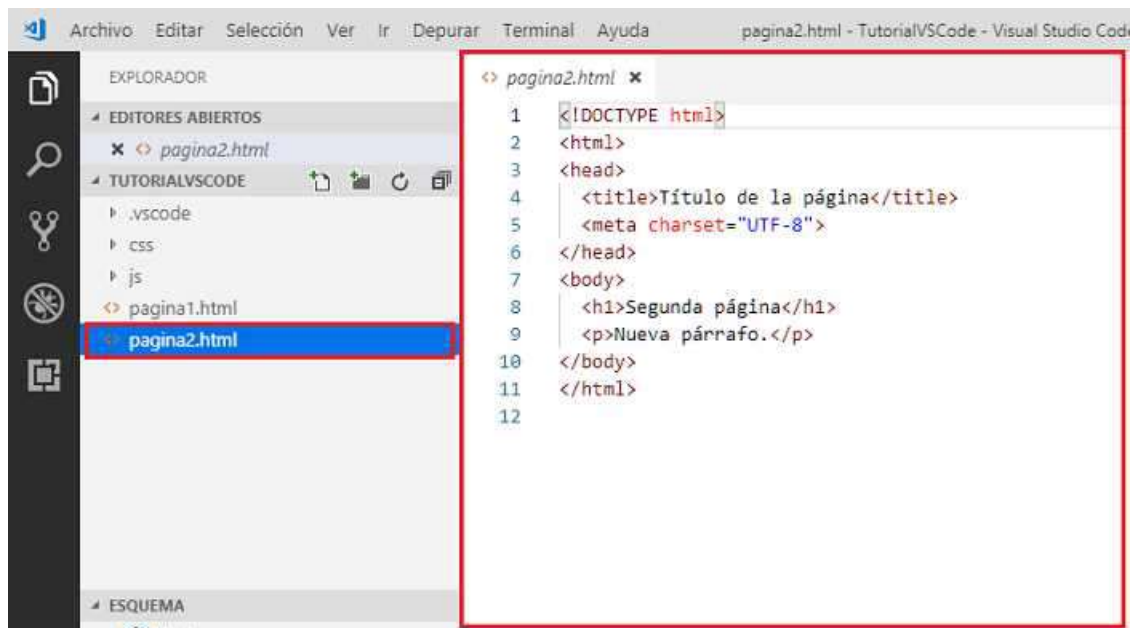
Seguidamente en el panel que aparece presionamos el botón "Abrir carpeta":



Seleccionamos la carpeta que queremos abrir y tenemos como resultado la lista de archivos y carpetas que tiene dicha carpeta:



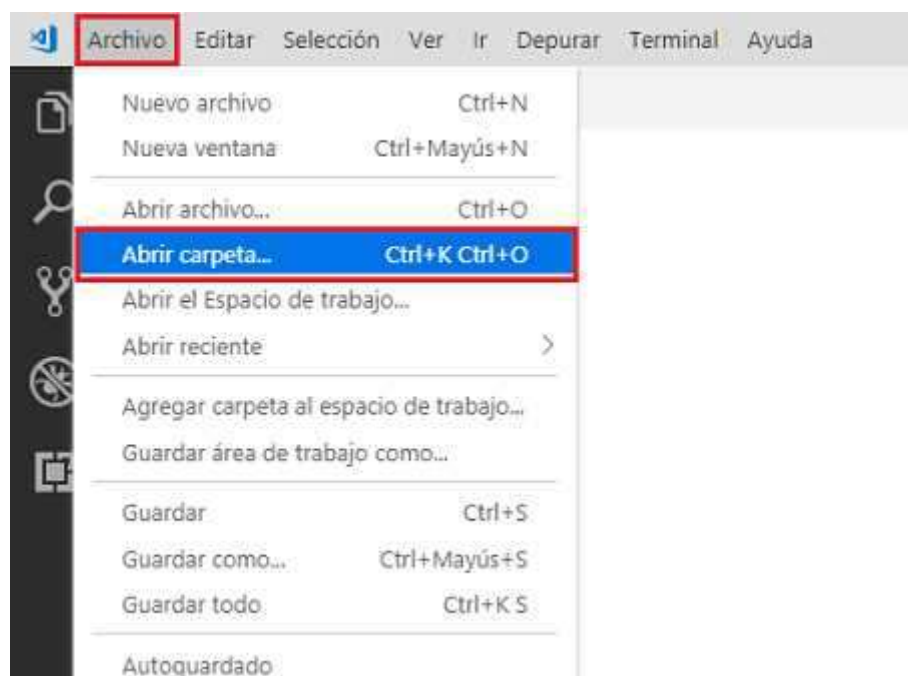
Podemos seleccionar con el mouse cualquiera de los archivos contenidos en el "Explorador" y ver su contenido inmediatamente en una pestaña de VSCode:



Los contenidos que seleccionamos con el mouse aparecen siempre en la misma pestaña, si necesitamos abrir dicho archivo en otra pestaña debemos hacer doble clic sobre el nombre del archivo.

2.

La segunda forma de "Abrir carpeta" es desde el menú de opciones de VSCode seleccionando "Archivo -> Abrir carpeta...":



Podemos acceder a la apertura de una carpeta por medio del atajo de teclado (Ctrl + K, Ctrl O):



Si salimos del editor VSCode, al regresar en otro momento queda almacenada la última carpeta abierta.

Cerrar carpeta

Cuando se abre una nueva carpeta se cierra la carpeta abierta actualmente, pero si queremos podemos cerrar la carpeta abierta actualmente mediante la opción del menú "Archivo -> Cerrar carpeta":



También podemos cerrarlas con las teclas de atajo (Ctrl + K, F):



4 - Area de trabajo : creación, apertura, modificación y cerrado

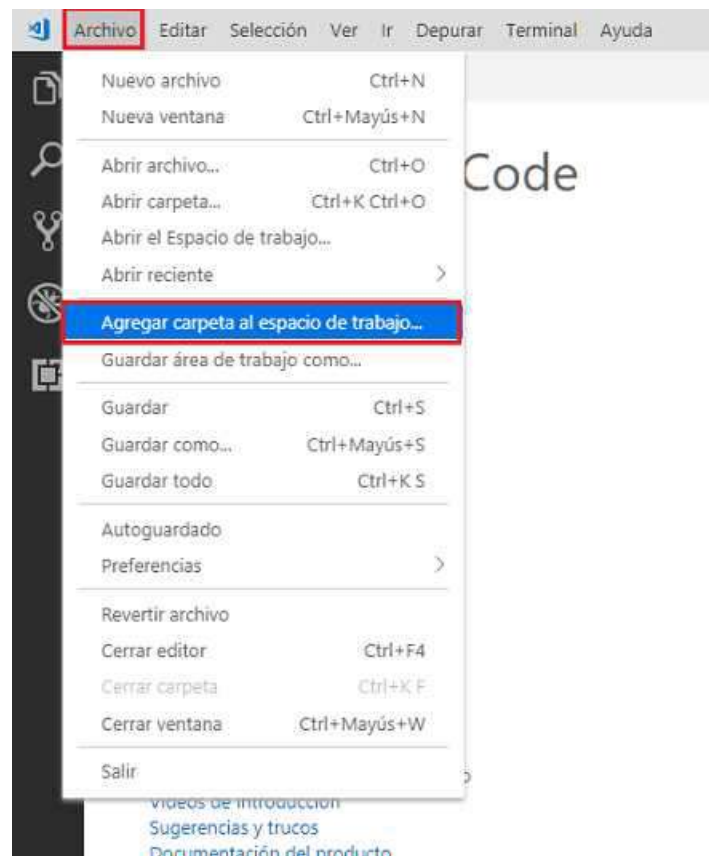
Es la tercera forma de trabajar con VSCode cuando tenemos proyectos más complejos que requieren de más de una carpeta.

Cuando un proyecto es pequeño o mediano es muy común que todos los archivos se encuentren alojados en una misma carpeta con sus subcarpetas.

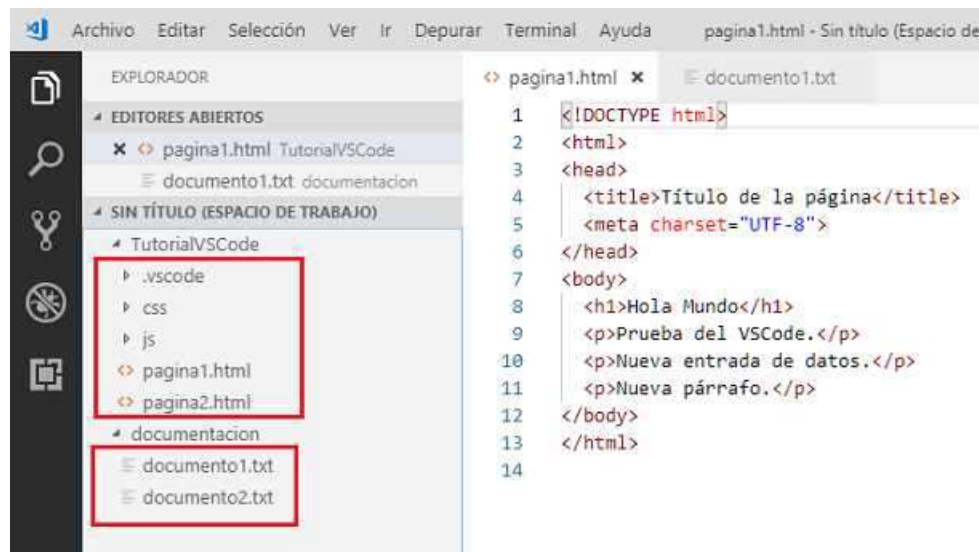
Cuando tenemos que trabajar con un proyecto muy grande, el cual se puede encontrar en carpetas de la misma unidad o que se almacenan en distintos discos duros por ejemplo, lo más adecuado es crear un área de trabajo que haga referencia a dichos directorios.

Creación de área de trabajo

Para crear un área de trabajo desde el menú de opciones seleccionamos "Archivo -> Agregar carpeta al espacio de trabajo":

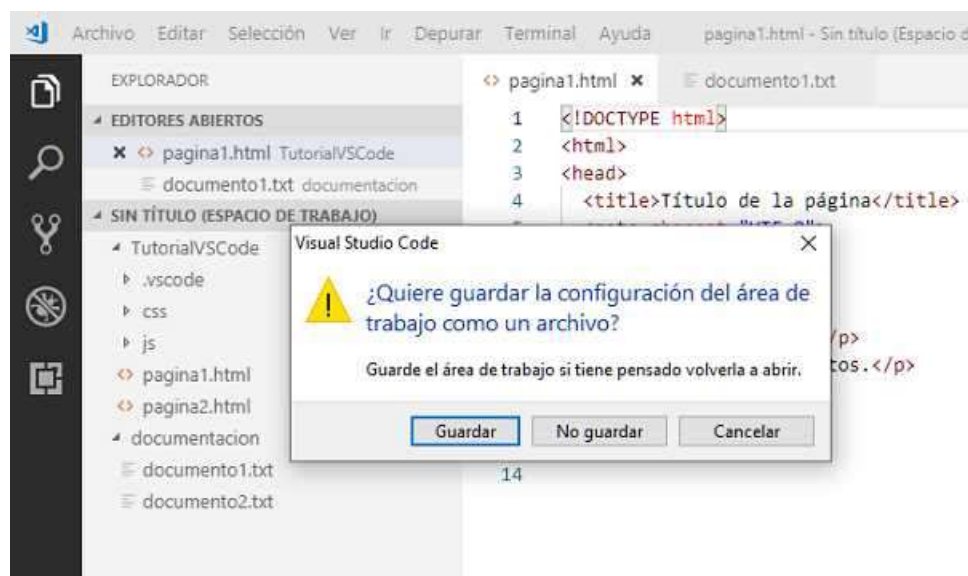


Repetimos la misma operación para agregar más directorios a la misma área de trabajo. Luego podemos ver que todos los directorios añadidos aparecen en la sección de "Explorador":

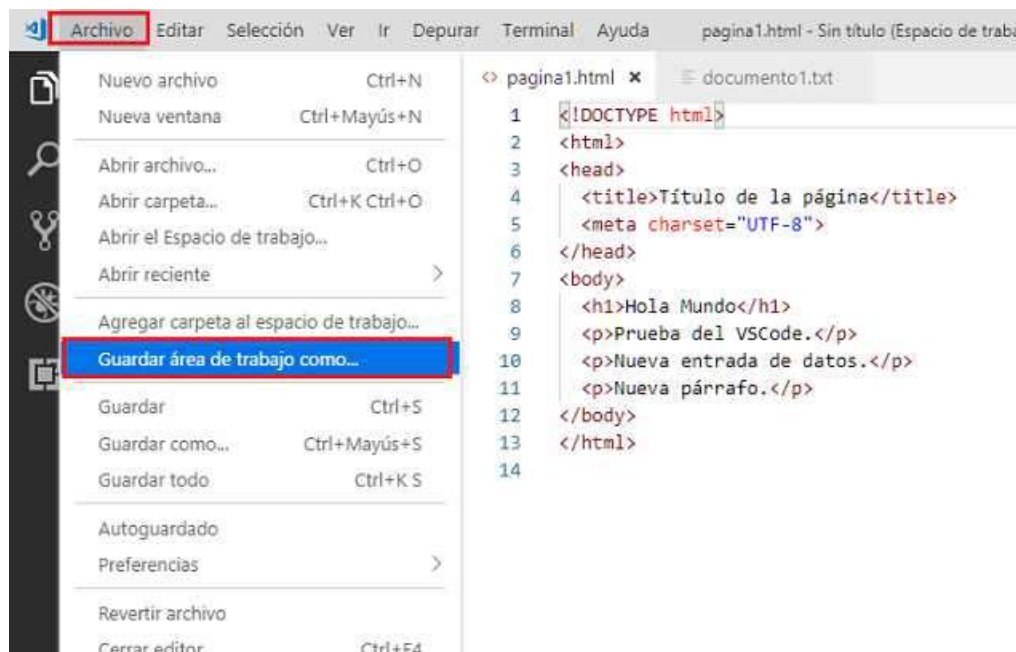


En el ejemplo que se ve en la imagen hemos agregado dos directorios ubicados en c:\TutorialVSCode y c:\documentacion

Podemos abrir ahora cualquiera de los archivos que hace referencia el área de trabajo. Si salimos del VSCode, cuando volvamos a ingresar se mostrará la última área de trabajo que hemos creado, pero si intentamos abrir por ejemplo una carpeta sin añadirla al área de trabajo actual se nos informará que el área de trabajo se perderá:



VSCode nos da la posibilidad de almacenar áreas de trabajo con un nombre en el disco duro de nuestra computadora para poderlas abrir en otro momento. Para almacenar un área de trabajo desde el menú de opciones seleccionamos "Guardar área de trabajo como...":



En el diálogo que aparece indicamos un nombre para poder recuperar esta área de trabajo creada y no tener que crearla nuevamente en otro momento.

Se genera un archivo con el nombre del área de trabajo indicada más la extensión 'code-workspace':

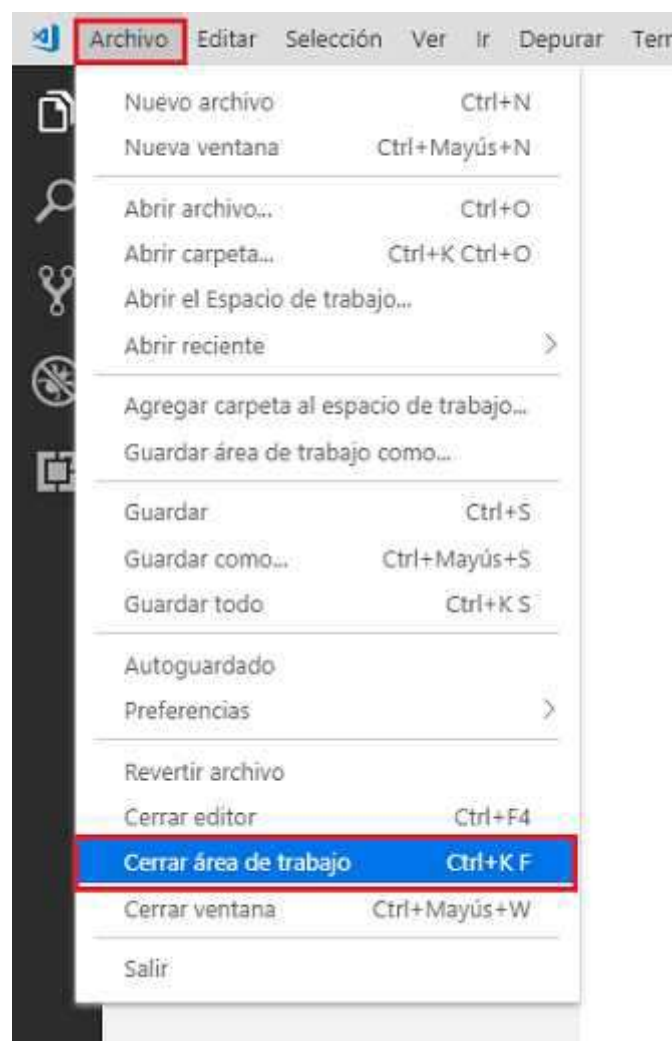
areaproyecto1.code-workspace

Si vemos el contenido de este archivo podremos ver que en su interior hace referencia a todas las carpetas del 'área de trabajo':



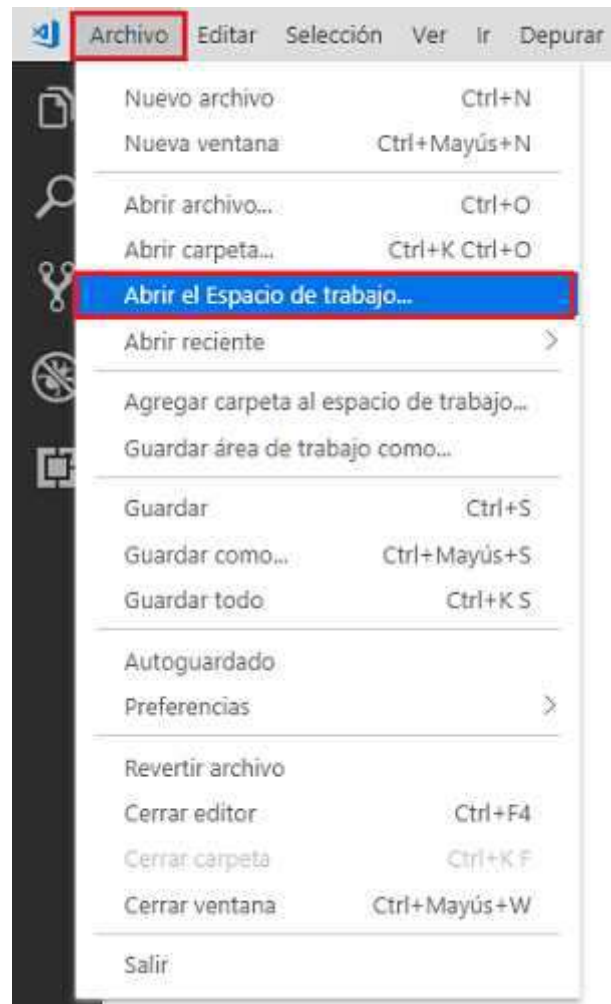
Cerrar un área de trabajo

Si tenemos un área de trabajo abierta podemos cerrarla seleccionando la opción "Archivo -> Cerrar área de trabajo" o el atajo (Ctrl + K, F):



Abrir un área de trabajo

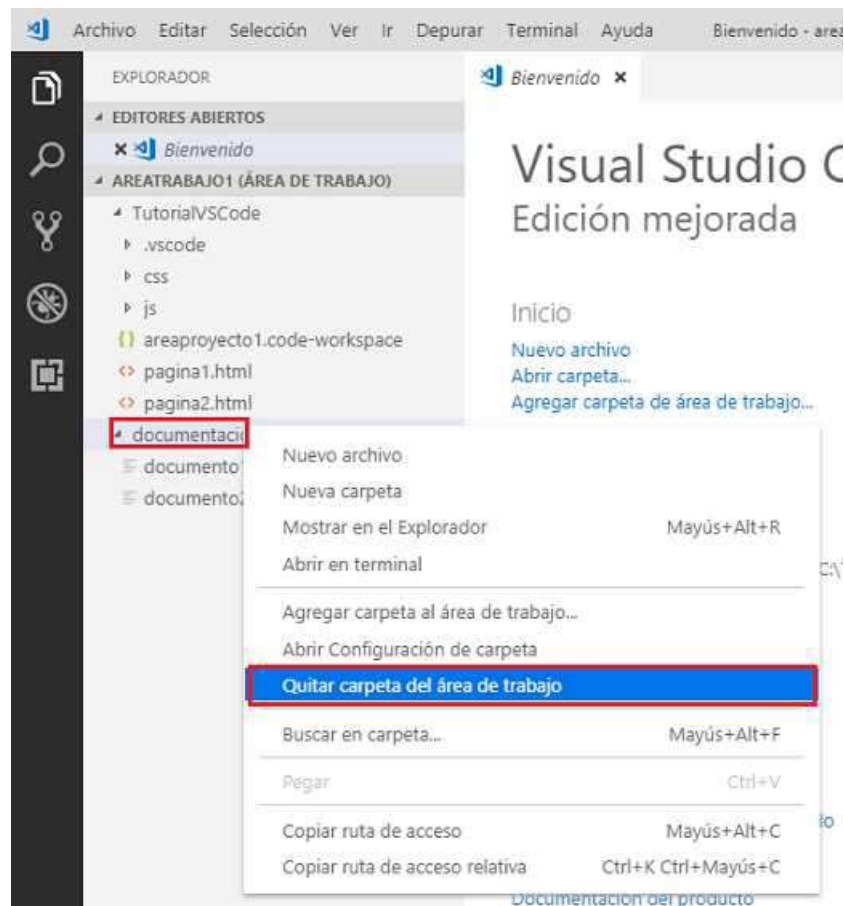
Si en algún momento grabamos un área de trabajo en un archivo con extensión *.code-workspace podemos hacer su apertura seleccionando desde el menú de opciones "Archivo ->Abrir el espacio de trabajo..."



Eliminar carpetas propias de un área de trabajo.

En cualquier momento podemos agregar carpetas a un área de trabajo como vimos anteriormente, pero también podemos borrar la referencia a carpetas (no se borra la carpeta del disco duro, sino solo la referencia en el área de trabajo)

Para desvincular una carpeta de un área de trabajo debemos presionar el botón derecho del ratón sobre el nombre de la carpeta y seleccionar "Quitar carpeta del área de trabajo":



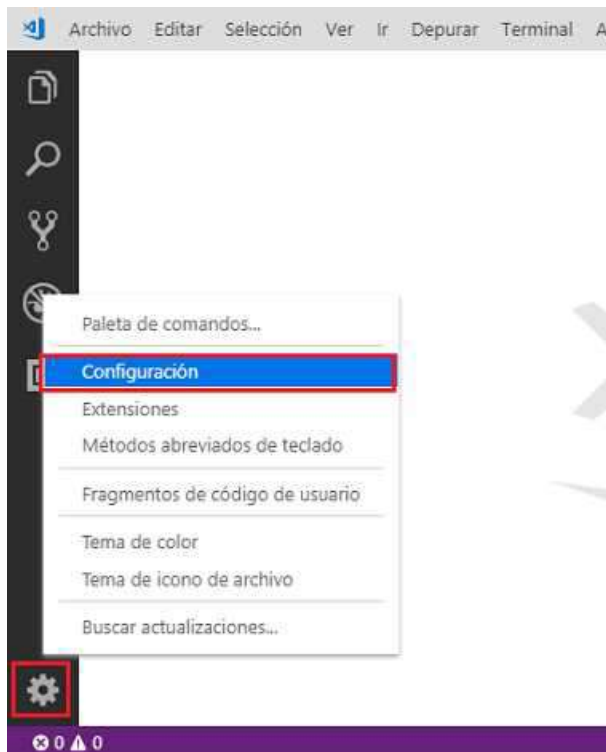
5 - Configuración del VSCode

Otra de las características fundamentales que tiene el editor de texto Visual Studio Code es la posibilidad de configurar sus funcionalidades como pueden ser fuentes, colores, tamaños de fuentes, tabulaciones, modos de presentar en pantalla el editor etc.

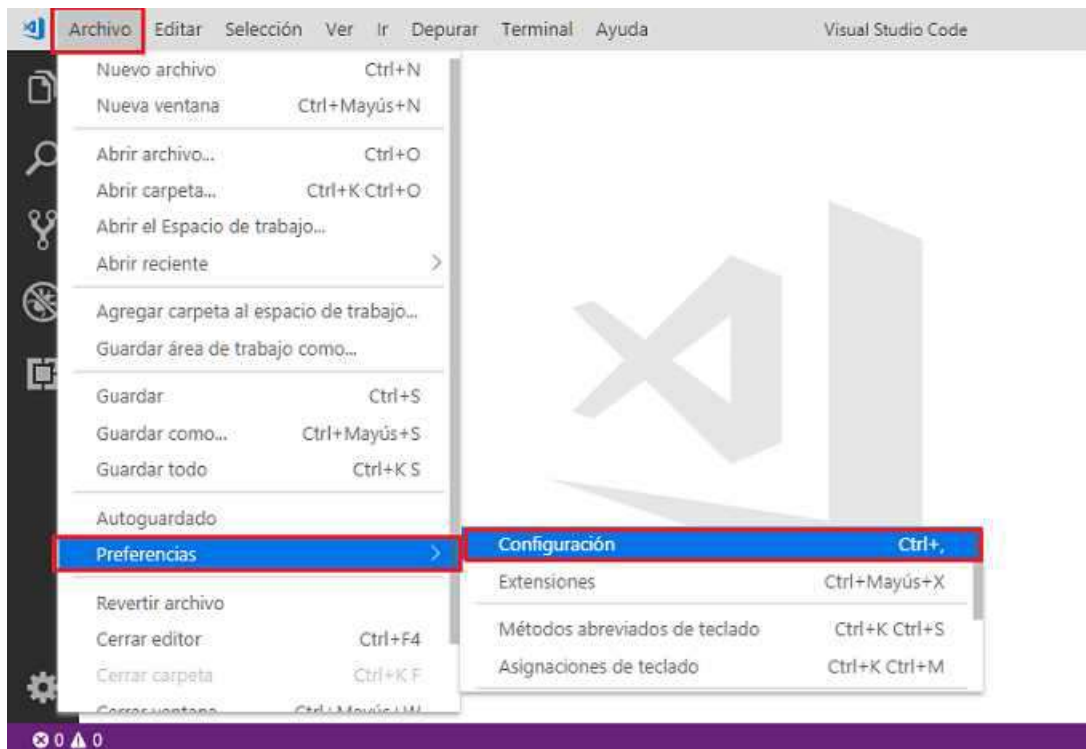
En este momento podemos configurar más de 540 características.

Para acceder a la ventana de configuración del VSCode lo podemos hacer de tres formas:

- Desde el ícono de la rueda dentada que aparece en la parte inferior izquierda:



- También podemos acceder a la pestaña de configuración desde el menú de barra seleccionando (Archivo -> Preferencias -> Configuración):

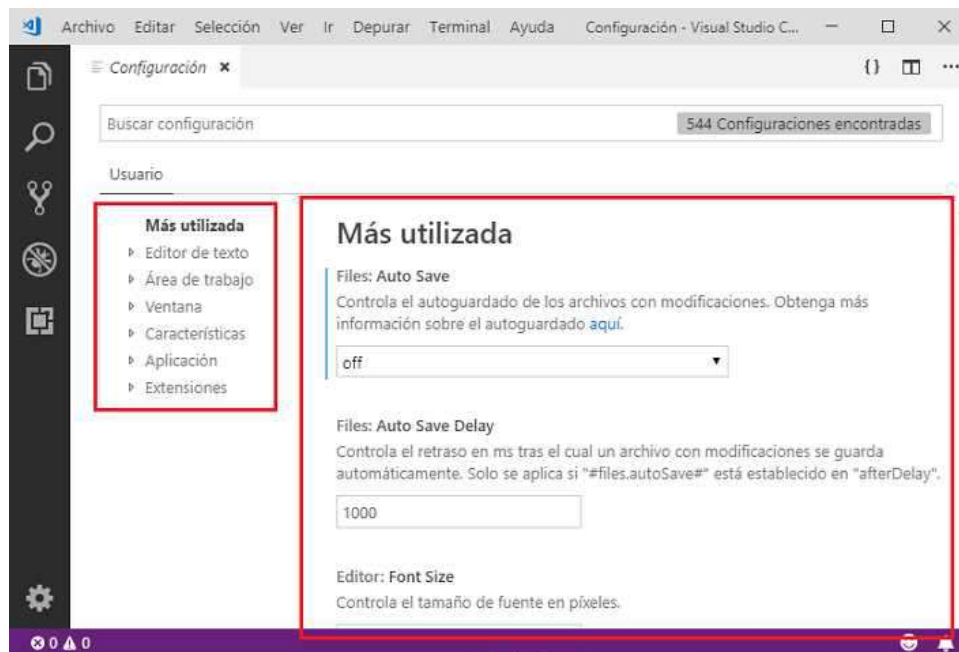


- Finalmente en cualquier momento podemos abrir la pestaña de configuración mediante las teclas (Ctrl + ,):

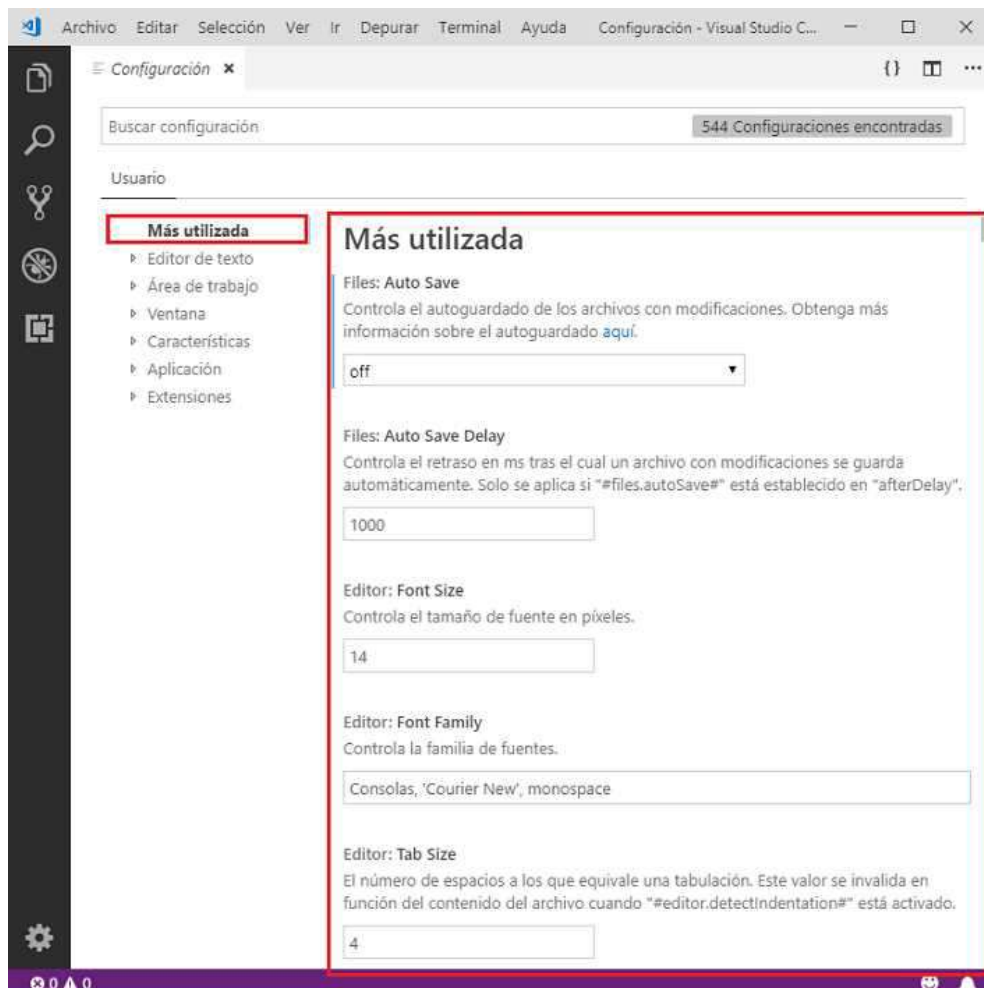


Ventana de configuración

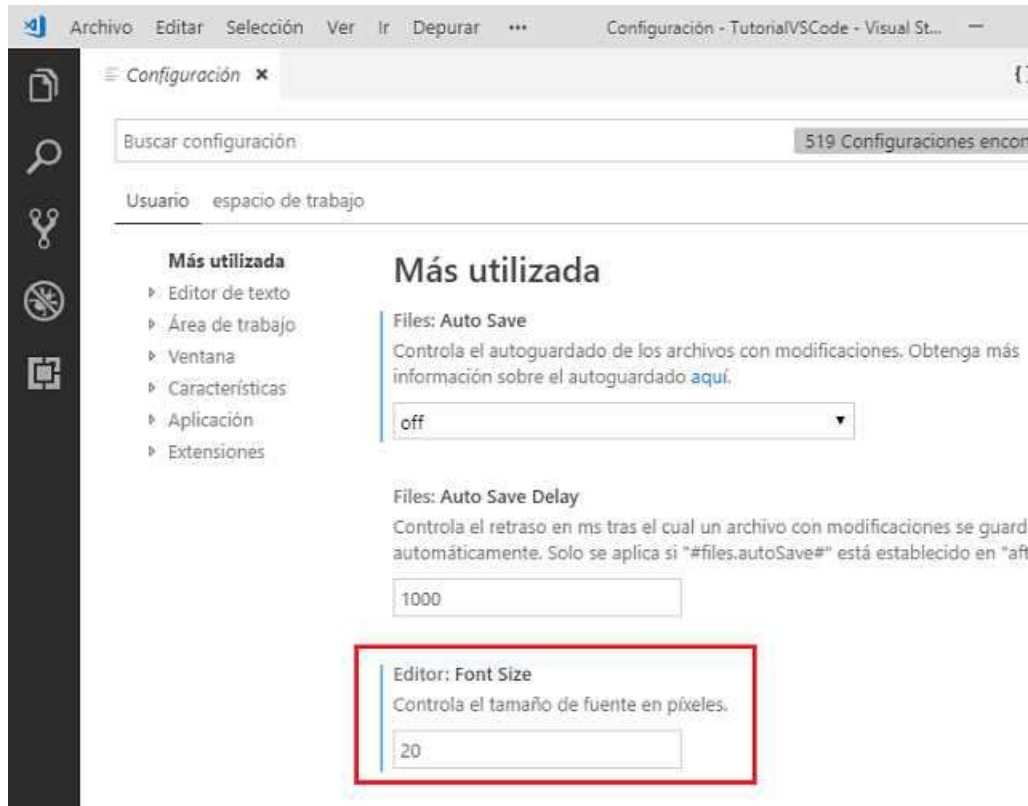
Del lado izquierdo nos muestran las distintas secciones que podemos configurar, y del lado derecho los valores que podemos asignarle:



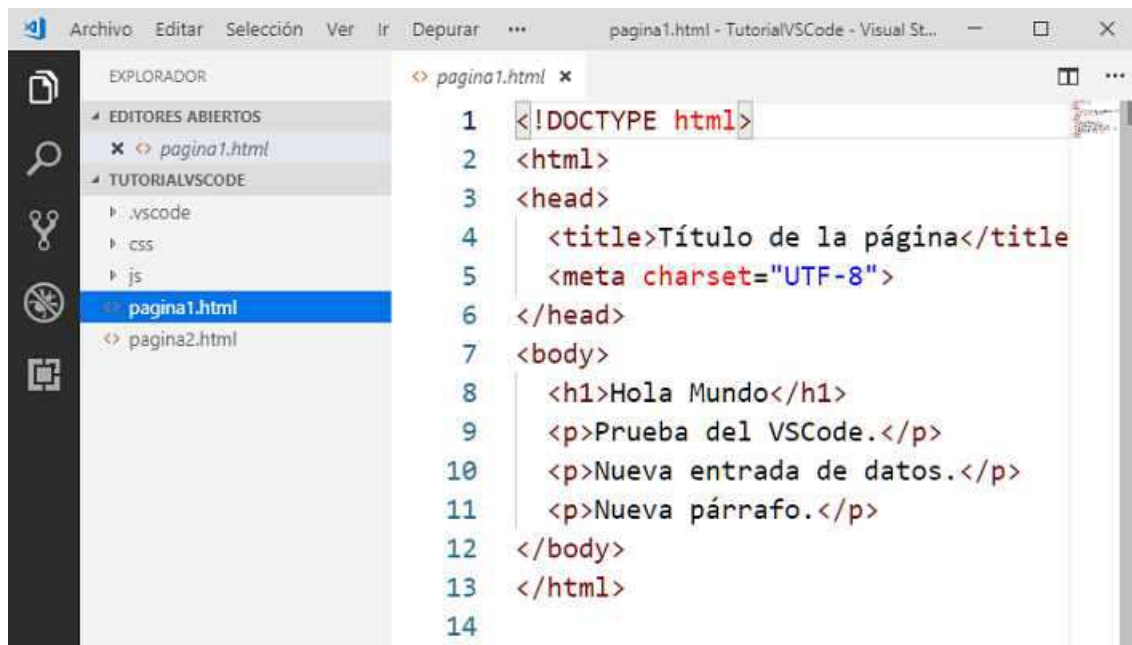
Primero se agrupan las propiedades que podemos configurar más utilizadas como puede ser espacios para tabulación, tamaño de fuente etc:



Luego de efectuar un cambio en la ventana de "Configuración", por ejemplo el tamaño de la fuente



Cerremos la pestaña de configuración y abramos cualquier archivo, luego podemos comprobar que el tamaño de la fuente es ahora de 20 puntos:

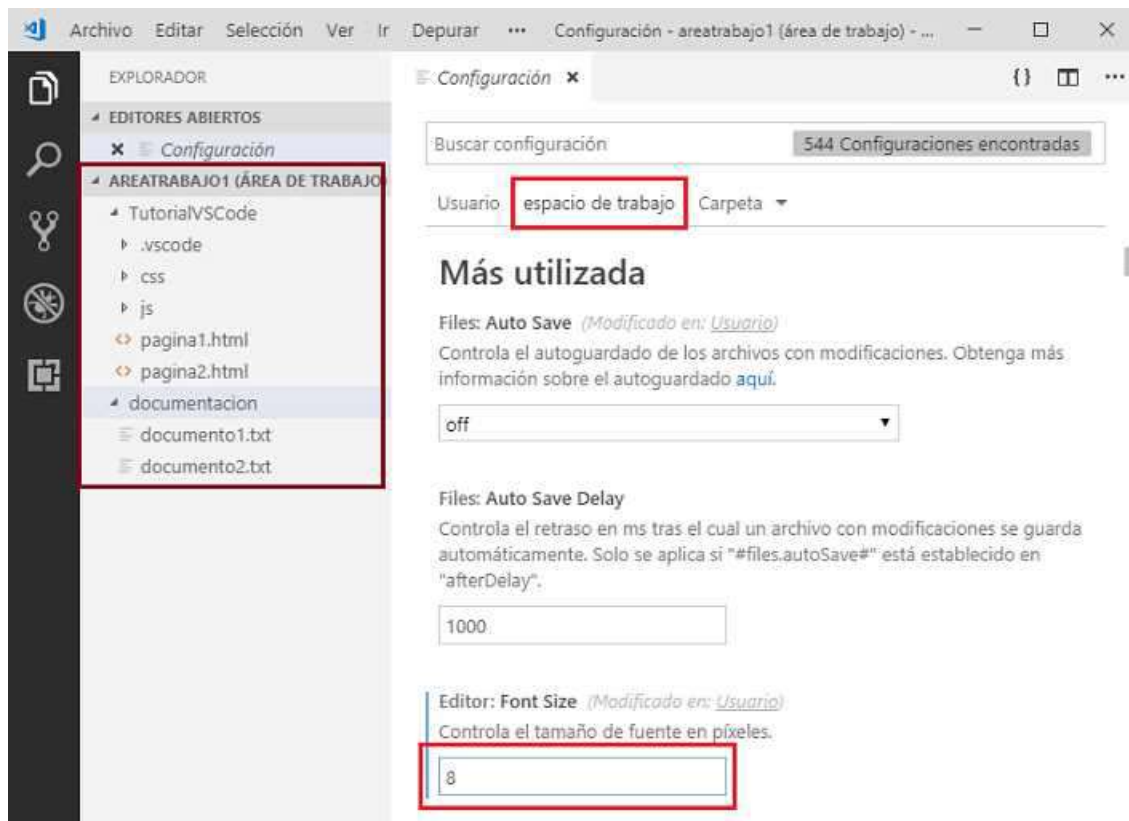


```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <title>Título de la página</title>
5   <meta charset="UTF-8">
6 </head>
7 <body>
8   <h1>Hola Mundo</h1>
9   <p>Prueba del VSCode.</p>
10  <p>Nueva entrada de datos.</p>
11  <p>Nueva párrafo.</p>
12 </body>
13 </html>
14
```

Configuración de usuario y configuración de área de trabajo

En el ejemplo anterior modificamos la "Configuración de usuario", ésta es aplicada a todos los archivos que se editen. Podemos definir configuraciones particulares a un área de trabajo, para eso el área de trabajo debe estar abierta.

Para probar la configuración de un área de trabajo primero hagamos la apertura de un área de trabajo que tengamos guardada y procedamos a entrar a "Configuración":



Debemos seleccionar la opción "Espacio de trabajo" y recién pasar a modificar valores de configuración.

La configuración definida solo tiene efecto en esa área de trabajo, luego si cerramos dicha área de trabajo y abrimos otra veremos que la configuración hecha anteriormente no tiene efecto.

La configuración del área de trabajo VS Code lo resuelve modificando el archivo con extensión '.code-workspace':

```
{ "folders": [ { "path": "C:\\\\TutorialVSCode" }, { "path": "C:\\\\documentacion" } ], "settings": { "editor.fontSize": 10 } }
```

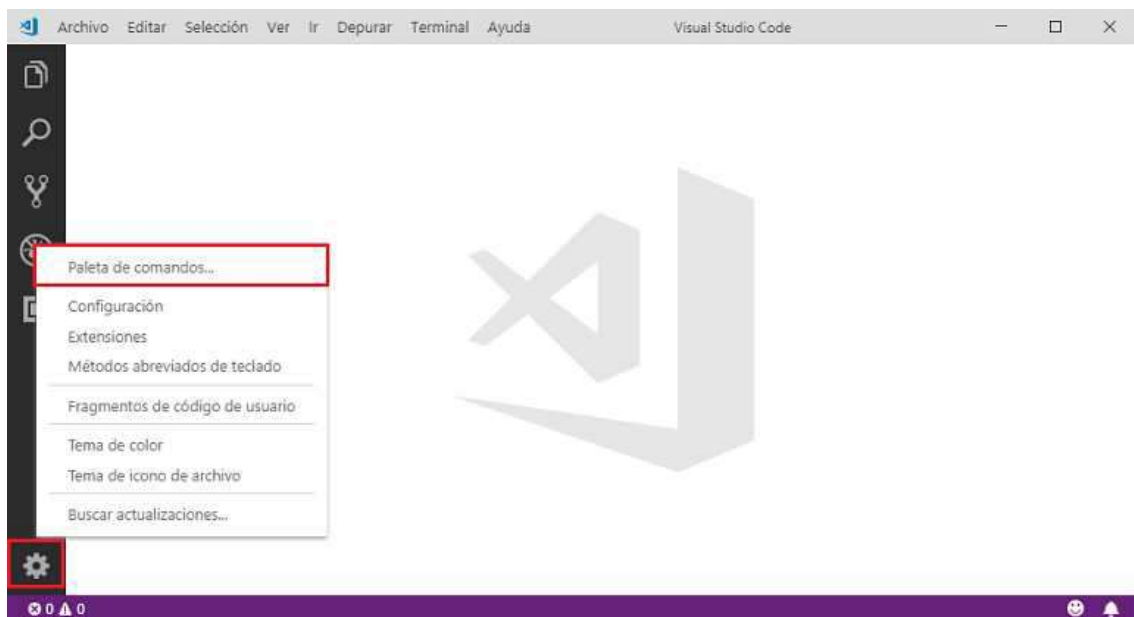
6 - Paleta de comandos

Otra herramienta que nos provee VSCode es una paleta de comandos donde podemos acceder a todas las funcionalidades.

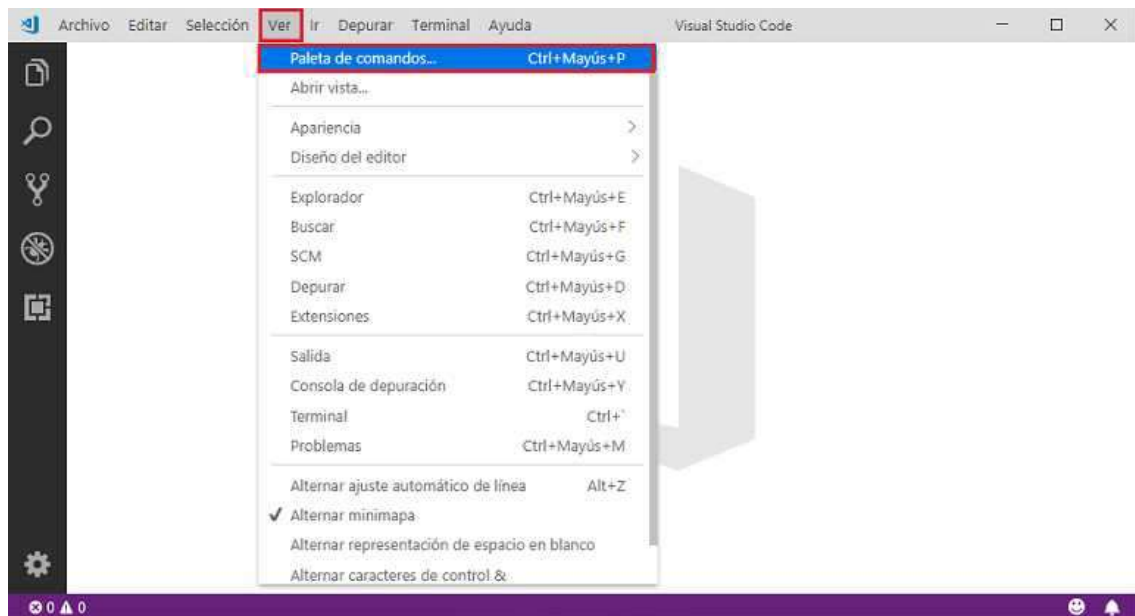
Tenemos que buscar la funcionalidad escribiendo su descripción.

Para hacer que aparezca la paleta de comandos podemos lanzarla de cuatro formas:

- Desde el ícono de la rueda dentada que aparece en la parte inferior izquierda:



- También podemos acceder a la "paleta de comandos" desde el menú de barra seleccionando (Ver -> Paleta de comandos...):



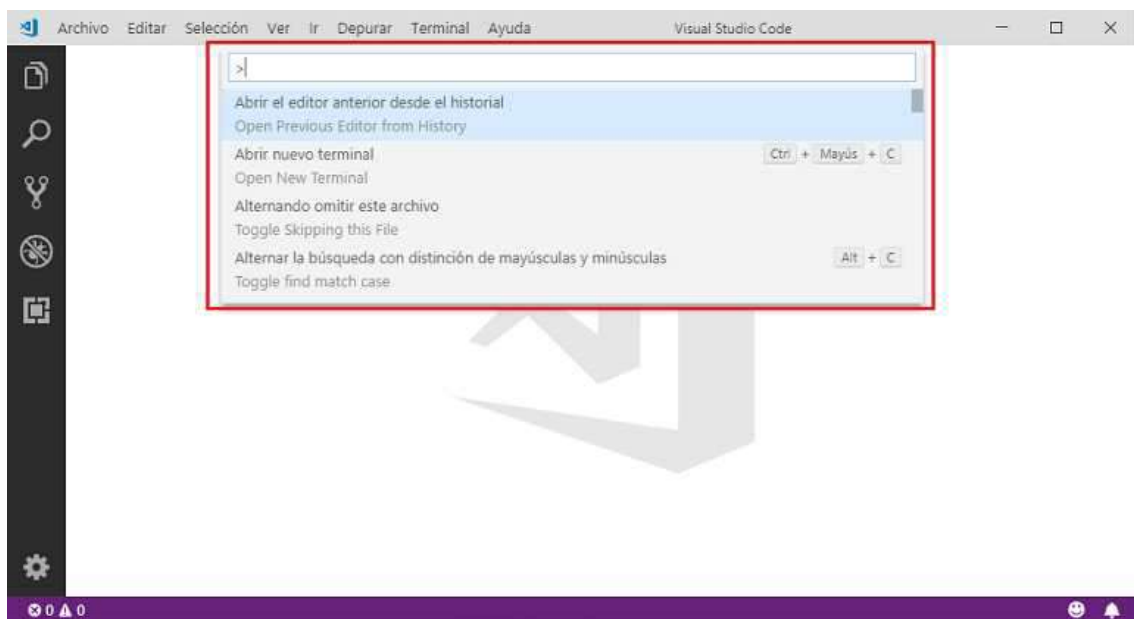
- Podemos abrir la "Paleta de comandos" mediante la tecla (F1):



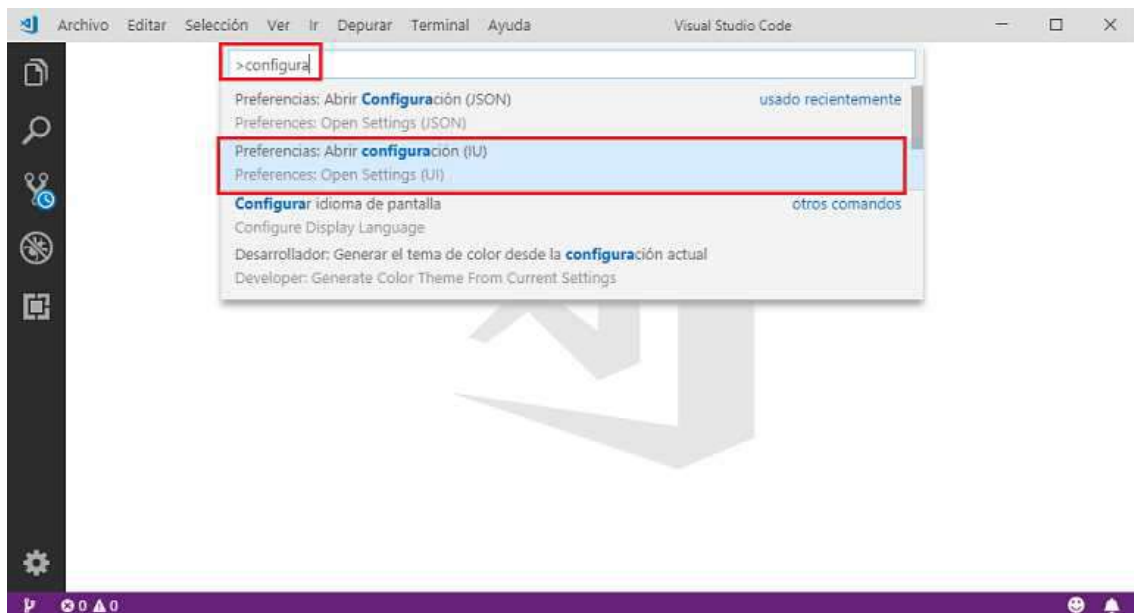
- Finalmente en cualquier momento podemos abrir la "Paleta de comandos" mediante las teclas (Ctrl + Shift + P):



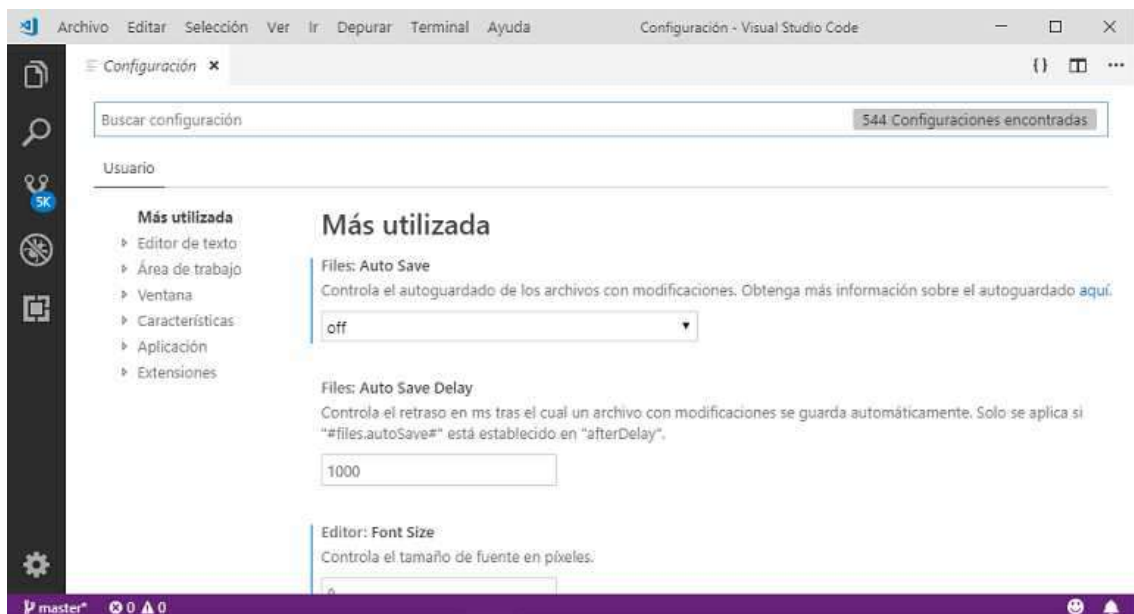
La "Paleta de comandos" aparece en pantalla en forma centrada y en su editor debemos escribir la acción que queremos hacer dentro de VSCode:



Por ejemplo si queremos entrar a la sección de preferencias de configuración de usuario que vimos en el concepto anterior podemos empezar a escribir dicho texto:



La cantidad de comandos se van filtrando a medida que ingresamos una o más palabras. Una vez que la ubicamos hacemos clic sobre la misma, en este ejemplo deberá aparecer la pestaña que ya conocemos de configuración:



La "Paleta de comandos" desaparece inmediatamente luego de lanzar la actividad.

Esta herramienta que nos provee VSCode es muy útil en un principio, a medida que conozcamos las teclas de atajo para cada actividad necesitaremos menos la "Paleta de comandos".

Ejercicios

1. Buscar las siguientes actividades desde la "Paleta de comandos":

- abrir archivo
- abrir carpeta
- abrir área de trabajo
- guardar

2. Teniendo el cursor activo en una línea de un archivo HTML, buscar en la paleta de componentes:

- eliminar línea
- agregar comentario
- buscar
- cerrar editor

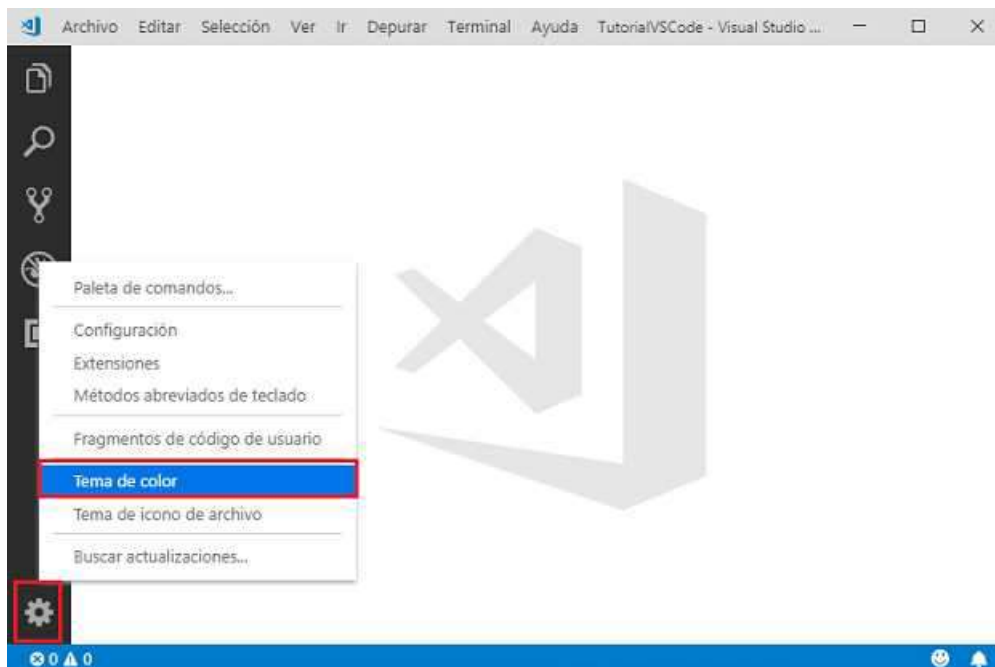
7 - Tema de color y tema de ícono de archivo

Tema de color

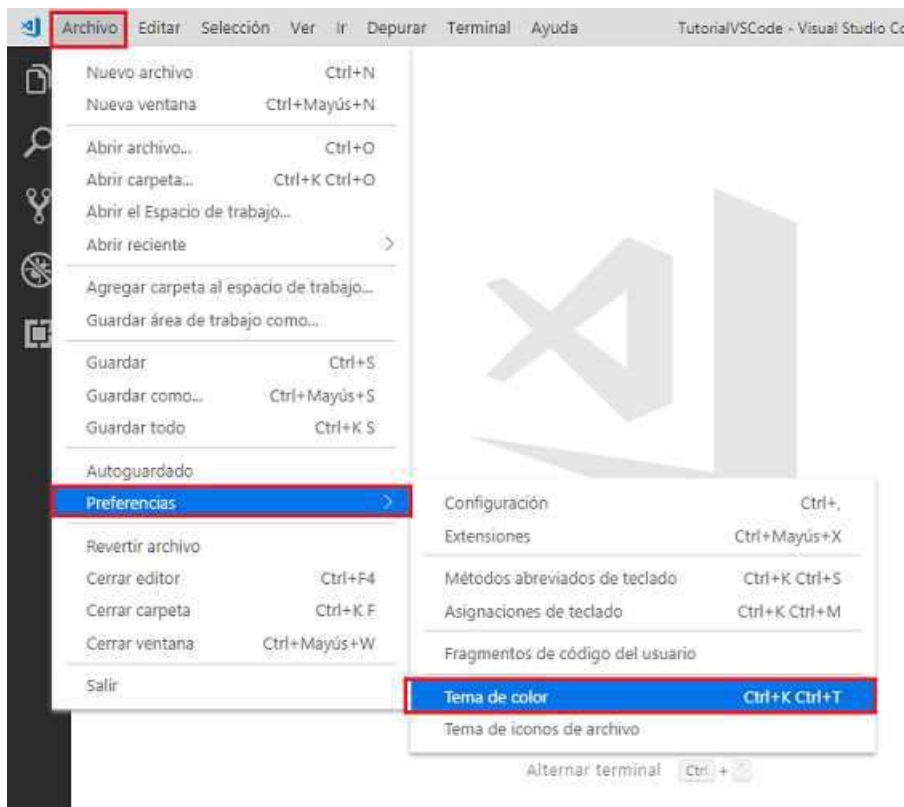
En el primer concepto vimos como cambiar rápidamente el "Tema de color" que trae VSCode con el objetivo de mostrar un tema claro y no el oscuro que trae por defecto.

Para hacer que aparezca el cuadro de selección de tema podemos hacerlo de tres formas:

- Desde el ícono de la rueda dentada que aparece en la parte inferior izquierda:



- También podemos acceder desde el menú de barra seleccionando (Archivo -> Preferencias -> Tema de Color):



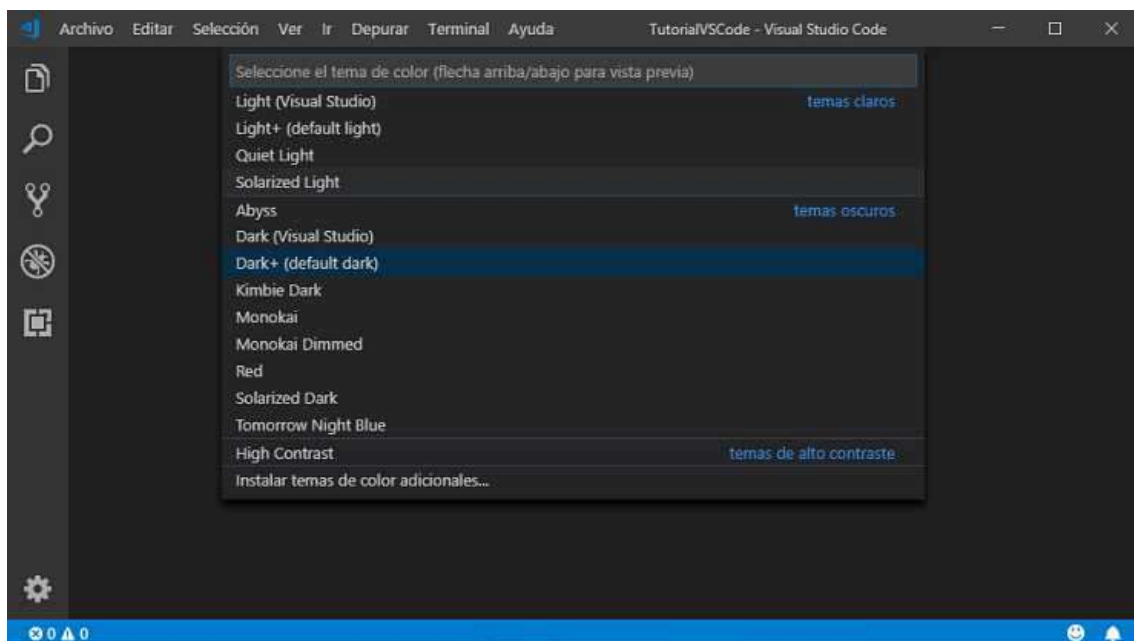
- Mediante las teclas (Ctrl + K, Ctrl + T):



Cuando se abre el panel con los "Temas de color" mediante las teclas de flechas o el mouse podemos ir seleccionando cada uno y ver como se actualiza en tiempo real la interfaz de VSCode:



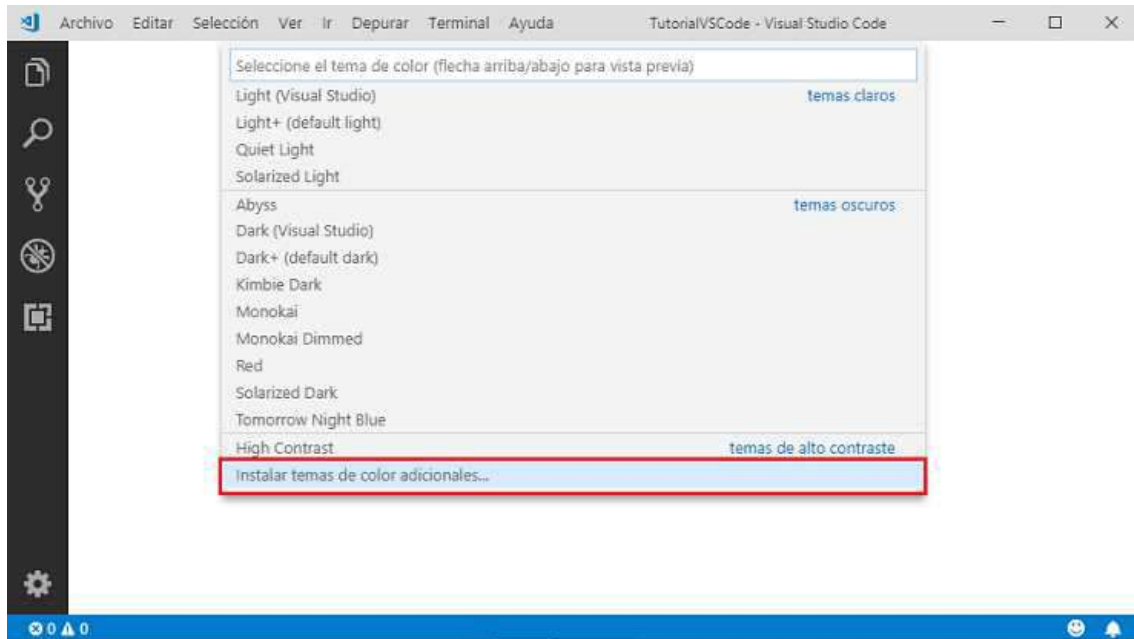
Si seleccionamos "Dark + (Default dark)"



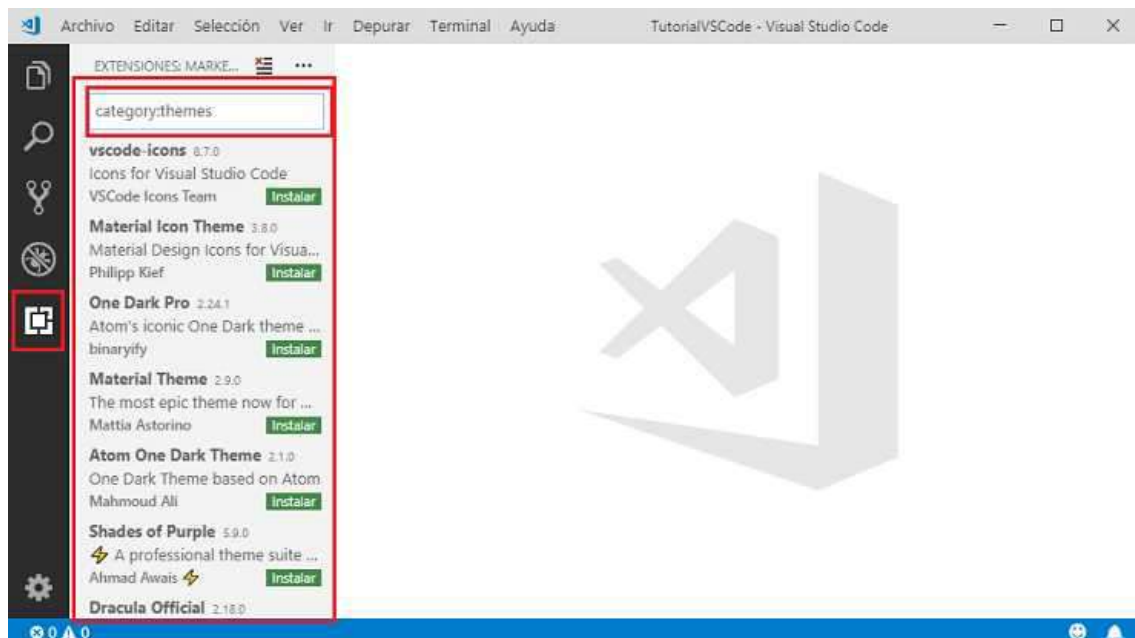
Disponemos una lista limitada de "temas de color" pero podemos descargar de la tienda de complementos de VSCode otros creados por la comunidad.

(<https://marketplace.visualstudio.com/search?target=VSCode&category=Themes&sortBy=Downloads>)

La búsqueda de temas también la podemos hacer desde el mismo VSCode seleccionando la opción "Instalar temas de color adicionales...":

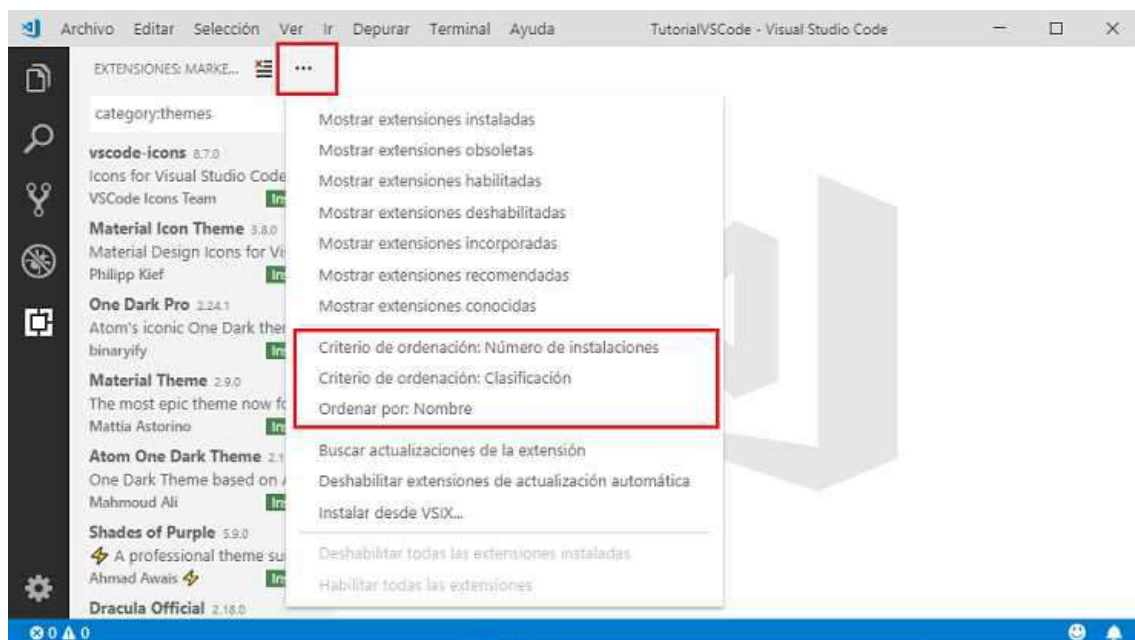


Cuando elegimos esta opción aparece un diálogo a la izquierda donde podemos navegar por los distintos temas disponibles para su instalación:



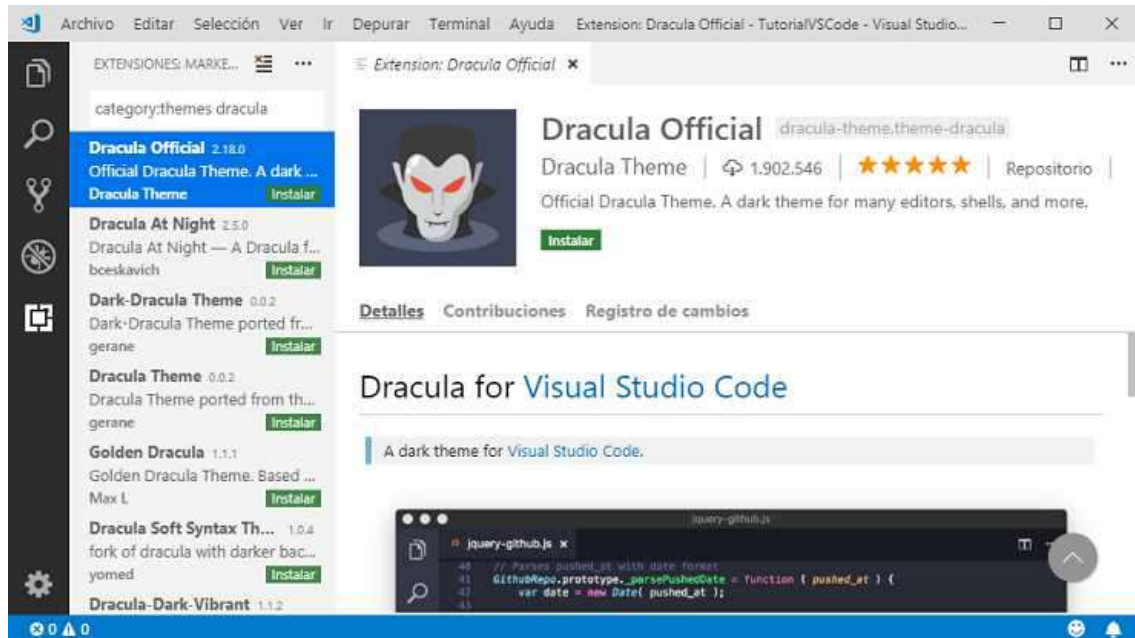
Podemos ver que aparecen seleccionados todos los complementos de tipo "tema de color" debido a que en el cuadro de entrada de datos está filtrado por dicha categoría: "category:themes"

Podemos ordenar los "Temas de color" por diferentes criterios (Número de instalaciones, clasificación, por nombre etc.), para hacer esta actividad presionamos el botón de la parte superior derecha:

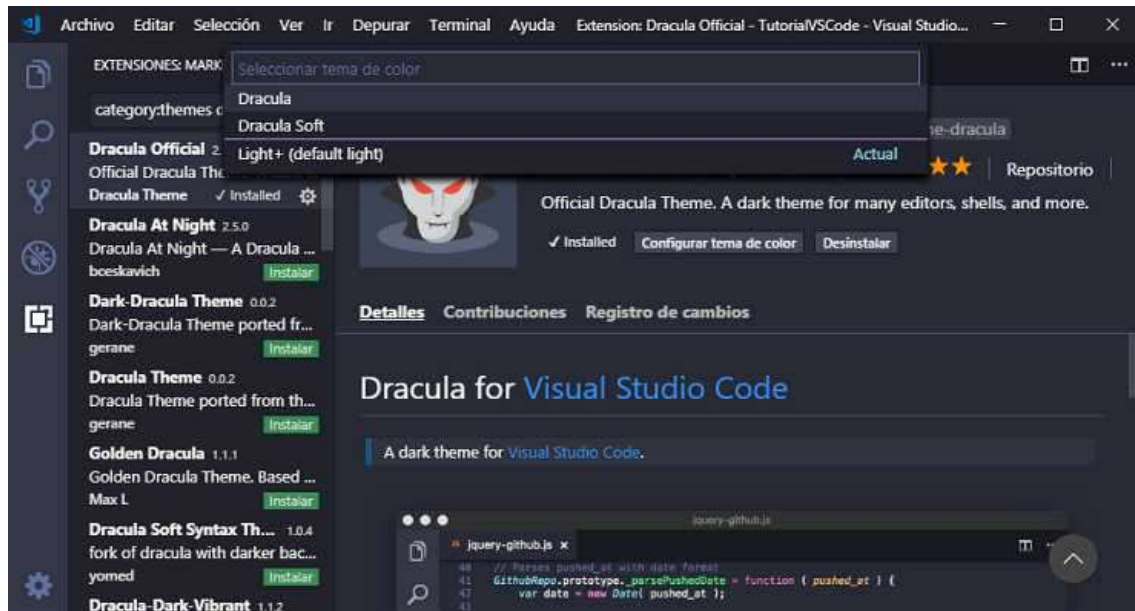


Veremos más adelante que hay otros complementos que podemos instalar en VSCode.

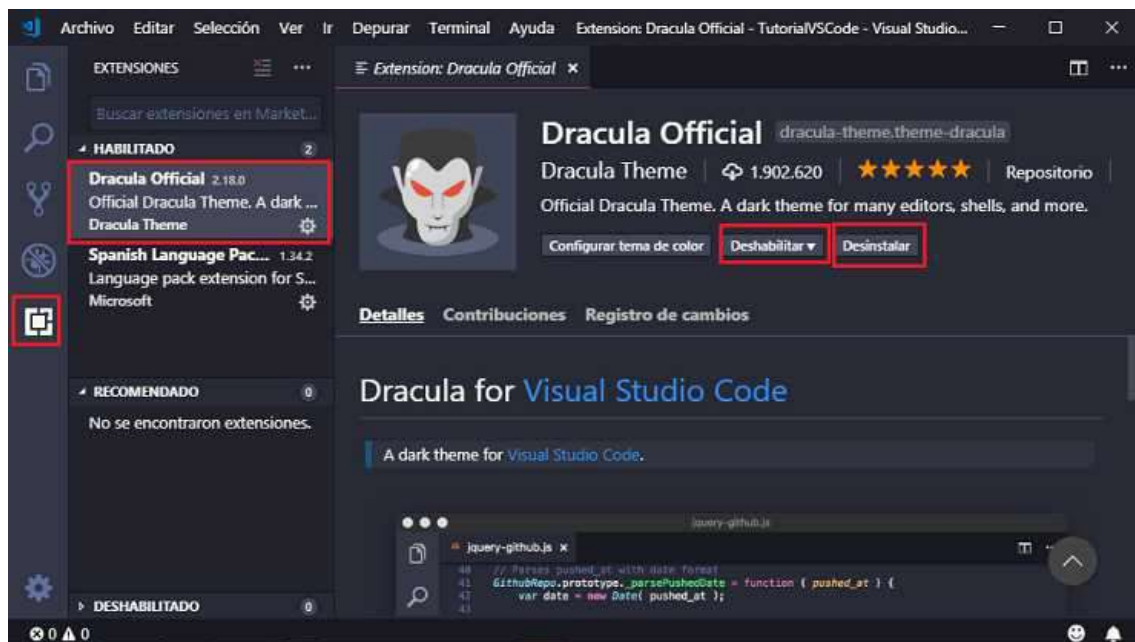
Probemos a descargar un tema de color muy popular llamado "Dracula":



Presionemos el botón de "Instalar", luego de esto en unos pocos segundos lo tendremos instalado y solo nos falta ir al cuadro de selección de temas para su activación:



Para administrar todos los complementos instalados en VSCode debemos seleccionar el icono de "Extensiones" (Ctrl + Shift + X):



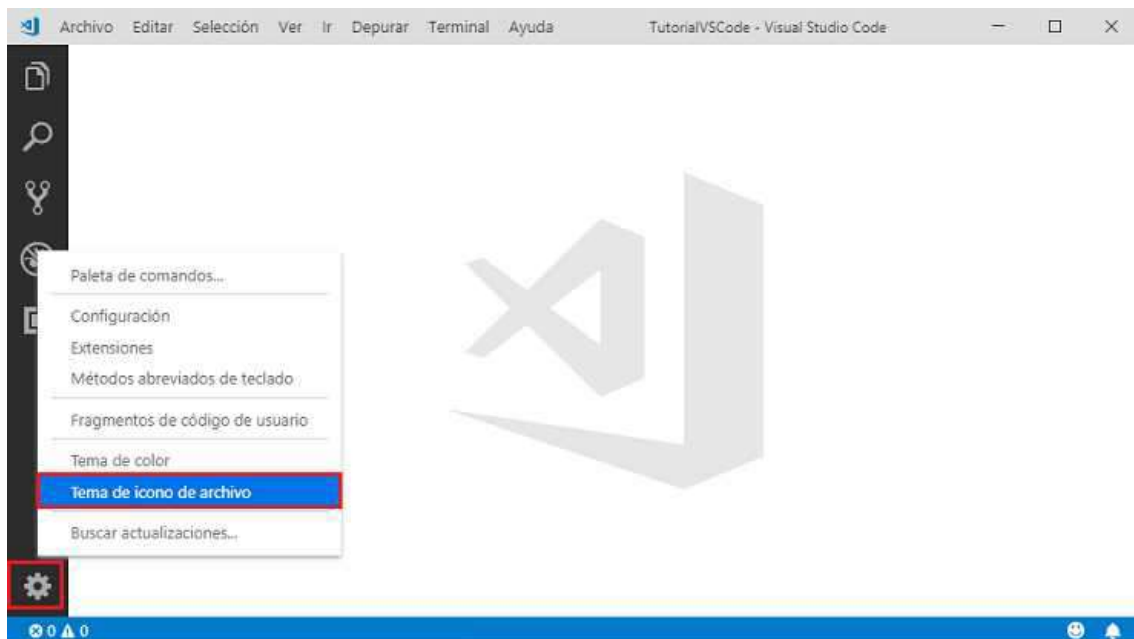
La existencia de tantos "Temas de color" puede parecer algo banal, pero pensemos que un programador pasa miles de horas con la vista puesta en el editor de texto.

Tema de ícono de archivo

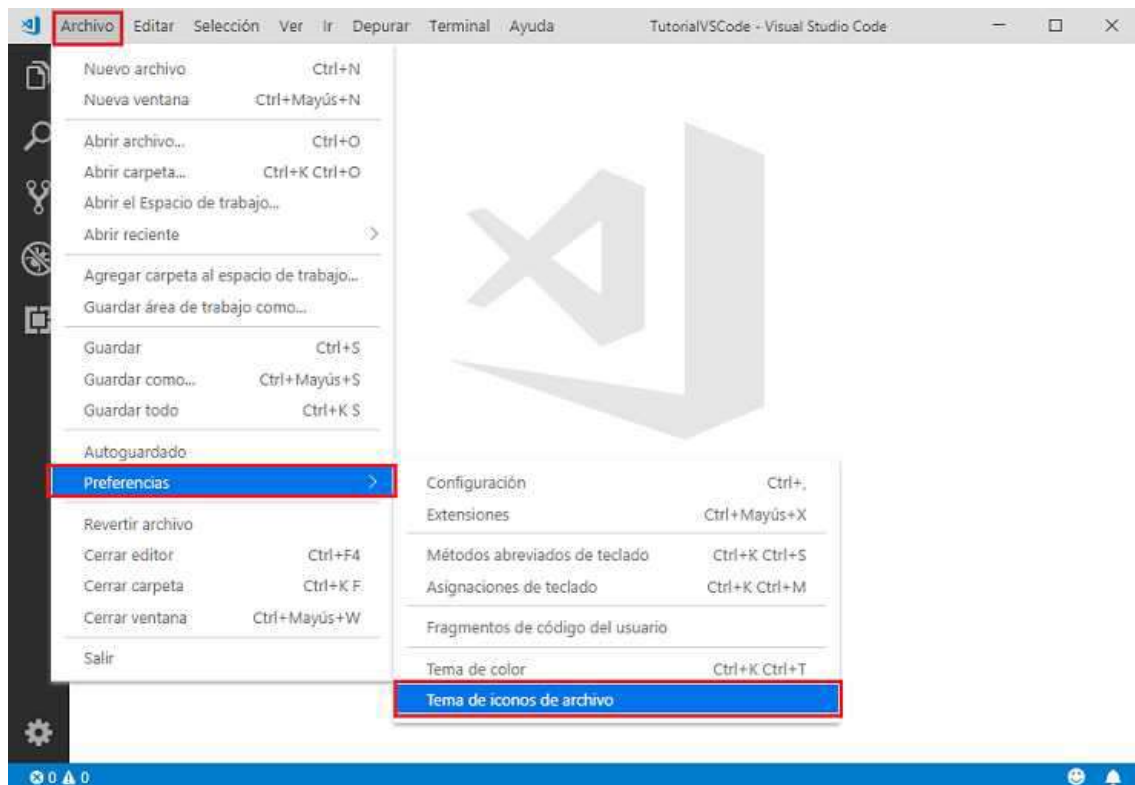
En forma muy similar a los "Tema de color" se administran los "Temas de ícono de archivo".

Para hacer que aparezca el cuadro de selección de "Tema de ícono de archivo" lo podemos hacer de dos formas:

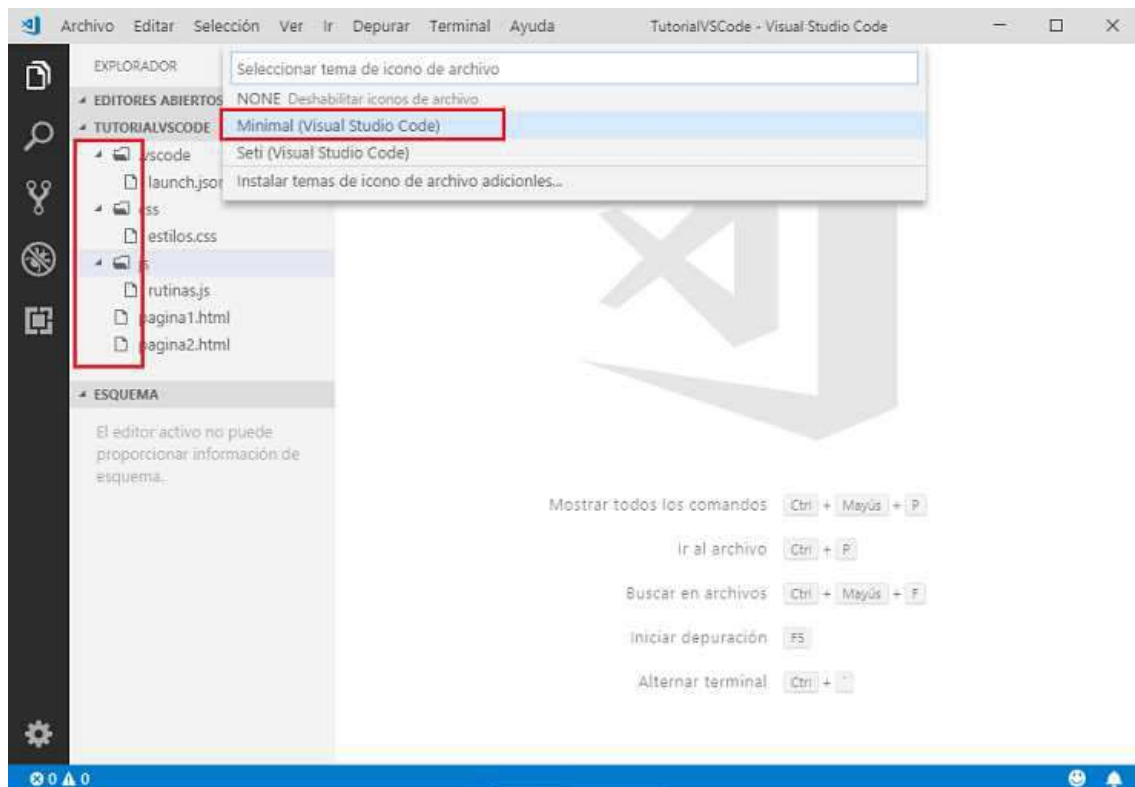
- Desde el ícono de la rueda dentada que aparece en la parte inferior izquierda:



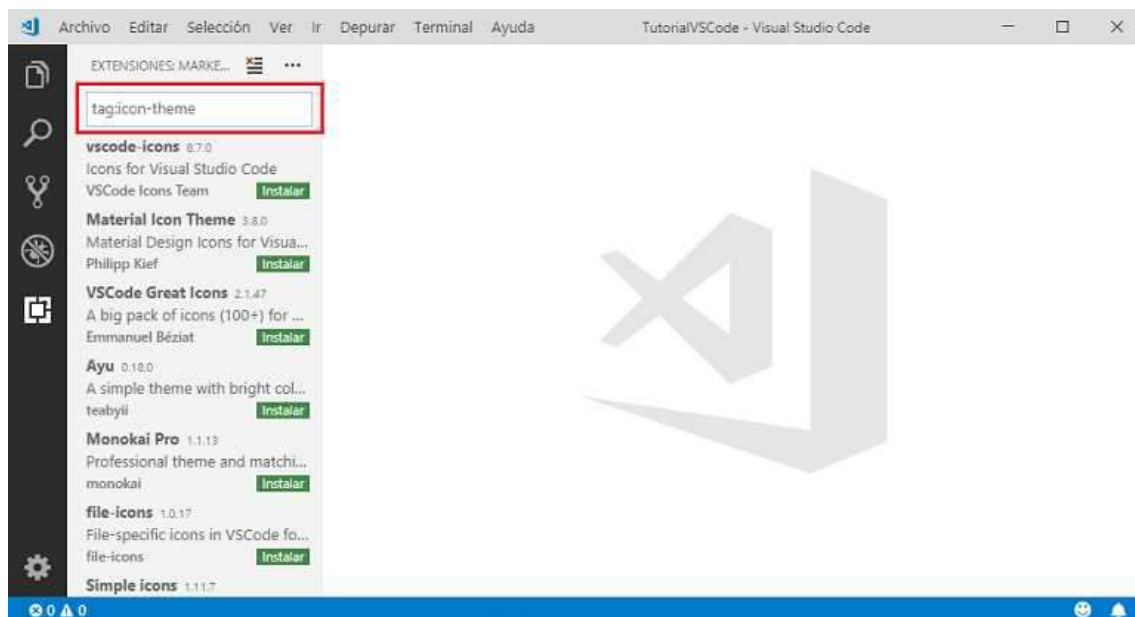
- También podemos acceder desde el menú de barra seleccionando (Archivo -> Preferencias -> Tema de ícono de archivo):



Cuando se abre el panel con los "Temas de ícono de archivo" mediante las teclas de flechas o el mouse podemos ir seleccionando cada uno y ver como se actualiza en tiempo real la interfaz de VSCode:



Igual que los temas de color, la comunidad ha desarrollado "Temas de ícono de archivo" que podemos instalar dando los mismos pasos visto anteriormente:



A medida que seleccionamos un "Tema de ícono de archivo" podemos ver en una pestaña quién es el creador, la cantidad de descargas, sus características etc.:

EXTENSIONES: MARKE...

Extension: Material Icon Theme

Material Icon Theme

Philipp Kief

7.453.391

★★★★★

Repositorio

Licencia

Instalar

Detalles Contribuciones Registro de cambios

Material Icon Theme

VS Marketplace v3.8.0 rating 4.91/5 (98) installs 1.81M downloads 7.45M build passing

The Material Icon Theme provides lots of icons based on Material Design for Visual Studio Code.

File icons

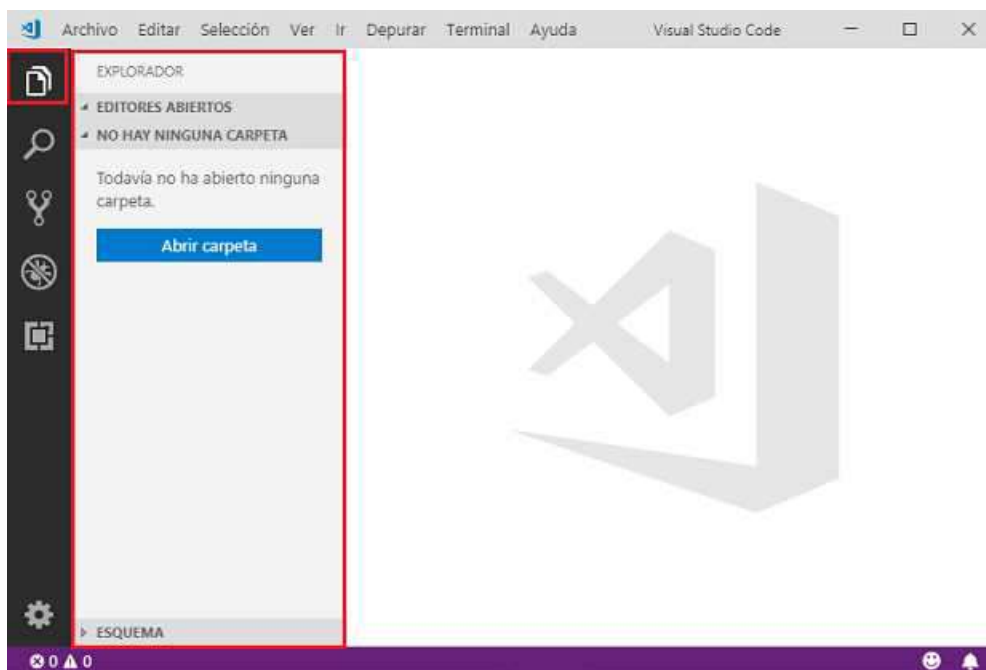
Icon	Name	Icon	Name	Icon	Name	Icon	Name	Icon	Name
3d	3d	Csharp	Csharp	Hexa	Hexa	Nodejs	Nodejs	Smarty	Smarty
Actionscript	Actionscript	Css	Css	Hcl	Hcl	Nodejs_all	Nodejs_all	Snyk	Snyk
Android	Android	Css map	Css map	Helm	Helm	Nodemon	Nodemon	Solidity	Solidity
Angular	Angular	Cucumber	Cucumber	Hetoku	Hetoku	Npm	Npm	Stencil	Stencil
Angular-component	Angular-component	Cuda	Cuda	Hipp	Hipp	Nunjucks	Nunjucks	Storyboard	Storyboard
Angular-directive	Angular-directive	D	D	Html	Html	Next	Next	Stylint	Stylint
Angular-guard	Angular-guard	Dart	Dart	Http	Http	Octant	Octant	Stylus	Stylus
Angular-pipe	Angular-pipe	Database	Database	Http	Http	Pdf	Pdf	Sublime	Sublime
Angular-resolver	Angular-resolver	Diff	Diff	Image	Image	Perl	Perl	Svelte	Svelte
Angular-service	Angular-service	Django	Django	Ionic	Ionic	Php	Php	Vue	Vue

8 - Explorador

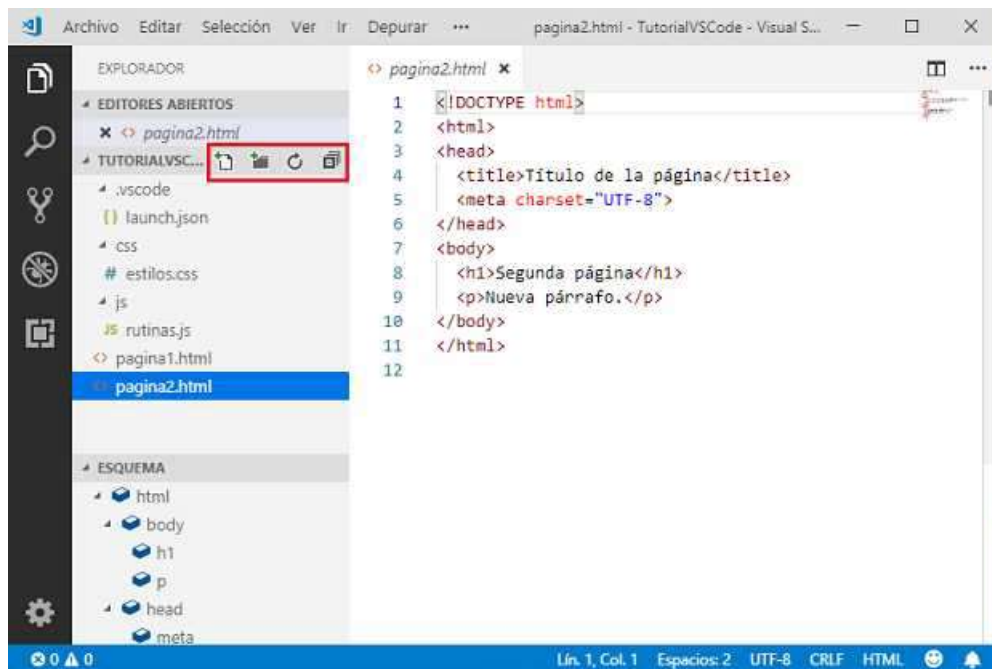
El ícono de "Explorador" se encuentra en la "Barra de actividades" del lado izquierdo por defecto. Ya hemos trabajado con el explorador cuando abrimos una carpeta o un área de trabajo.

Cuando se lo presiona al ícono aparece el "Explorador" y si volvemos a presionarse oculta.

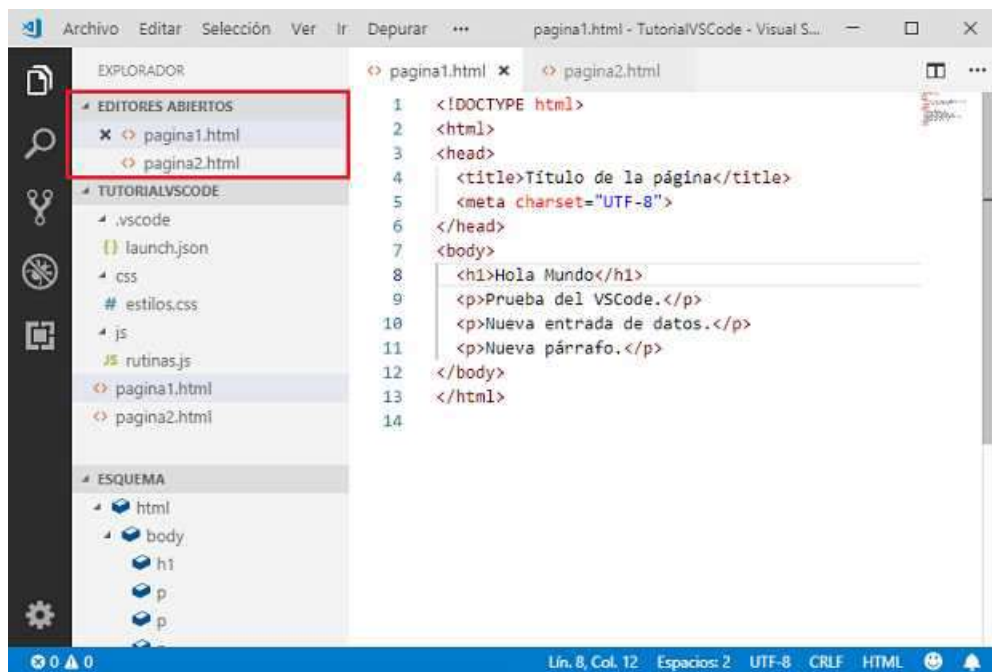
Cuando no hay ninguna carpeta o área de trabajo abierta el "Explorador" nos invita a "Abrir carpeta":



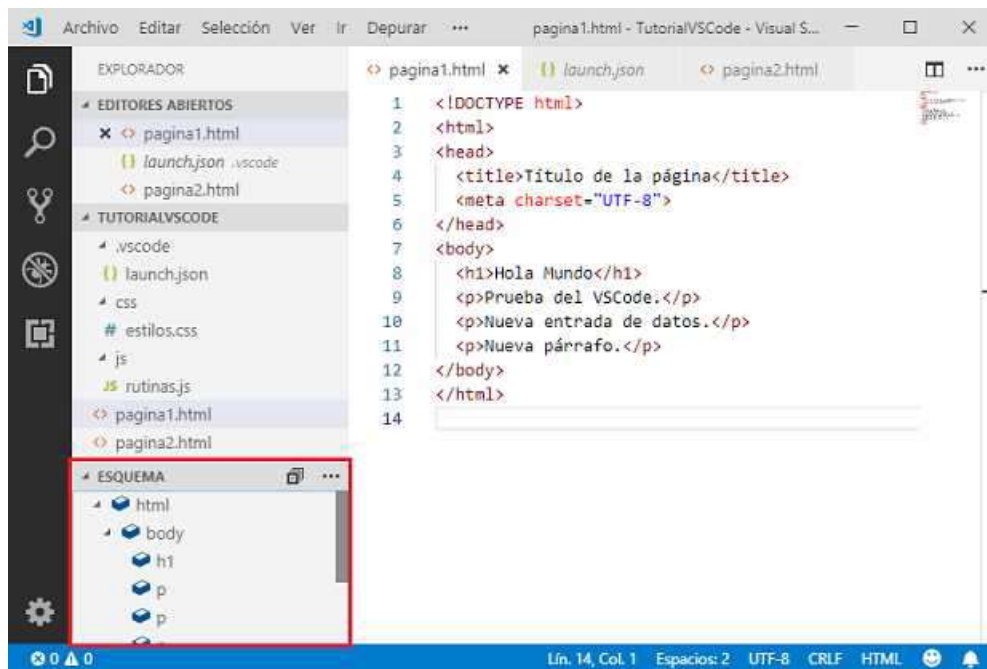
Cuando tenemos abierta una carpeta o área de trabajo podemos desde el mismo explorador crear archivos, carpetas, actualizar y contraer carpetas (los íconos de estas actividades solo son visibles cuando la flecha del mouse se encuentra en la ventana del Explorador):



En el mismo "Explorador" en la parte superior se muestran todos los archivos que se encuentran abiertos en alguna pestaña:



En la parte inferior se muestra un esquema del contenido del archivo que se encuentra abierto (depende del tipo de archivo que se muestra HTML, CSS, JavaScript, C# etc.):

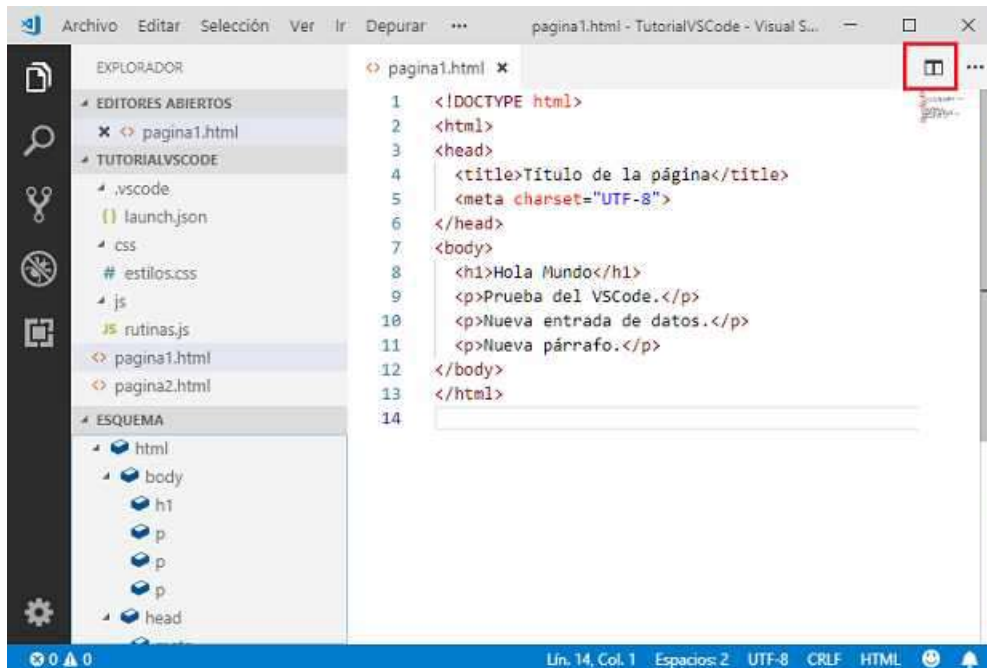


Las teclas de acceso rápido para mostrar el "Explorador" (Ctrl + Shift + E):

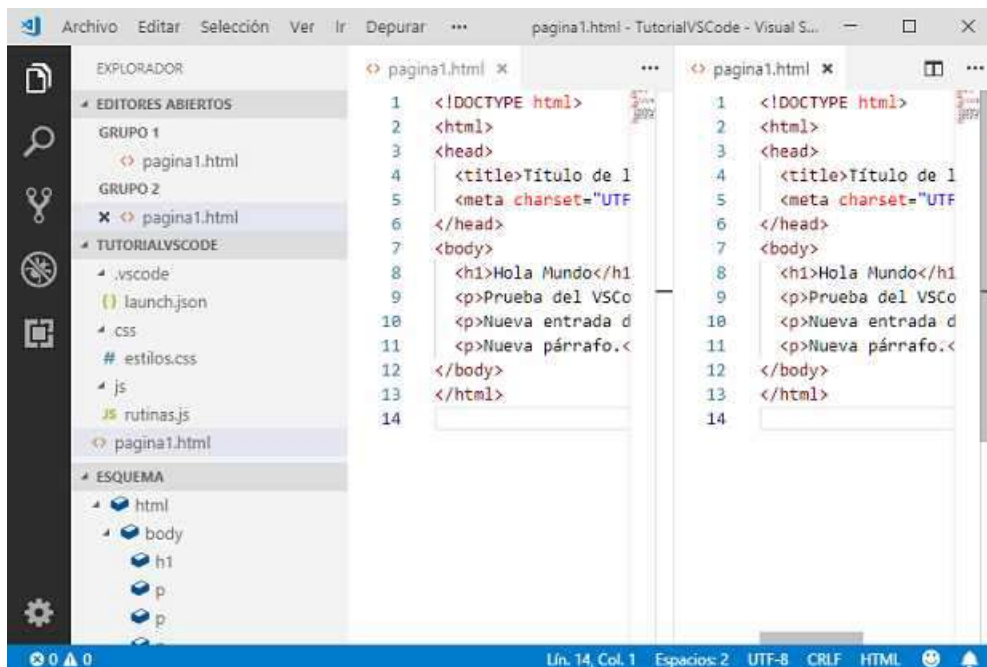


Para abrir un archivo en el área de edición hacemos clic sobre el nombre en el "Explorador" y con doble clic se abre una nueva pestaña(editor)

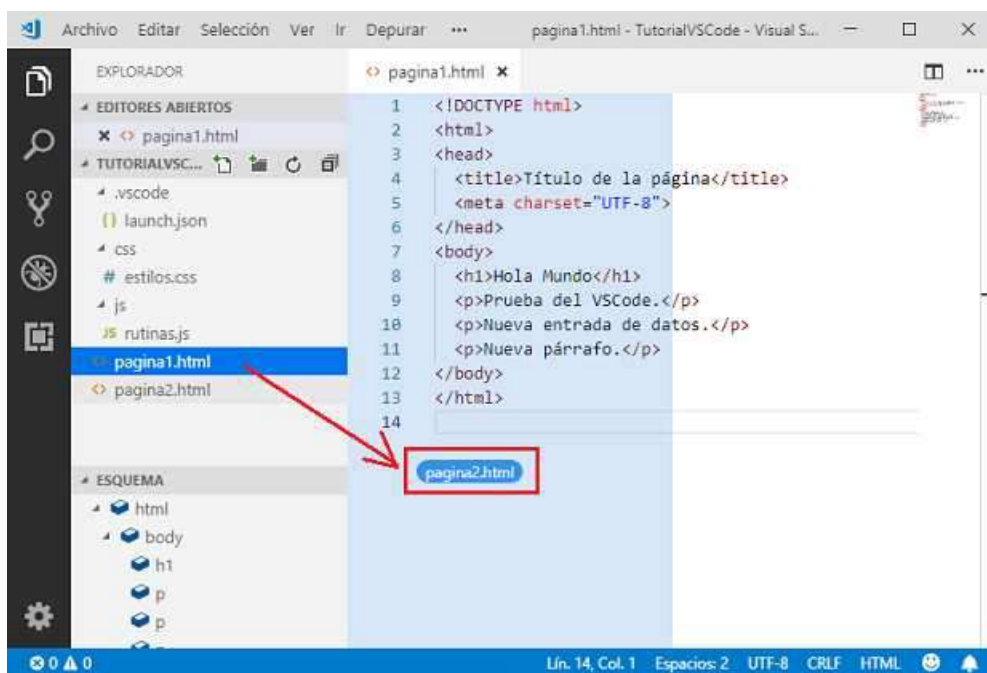
En algunas situaciones si necesitamos tener un mismo archivo abierto en forma simultánea debemos presionar el ícono con un símbolo de dividir de la equina superior derecha:



Luego de dividir tenemos dos ventanas de edición paralelas (tiene sentido si tenemos archivos muy grandes y tenemos que comparar contenidos que se encuentran en líneas muy distantes entre si):

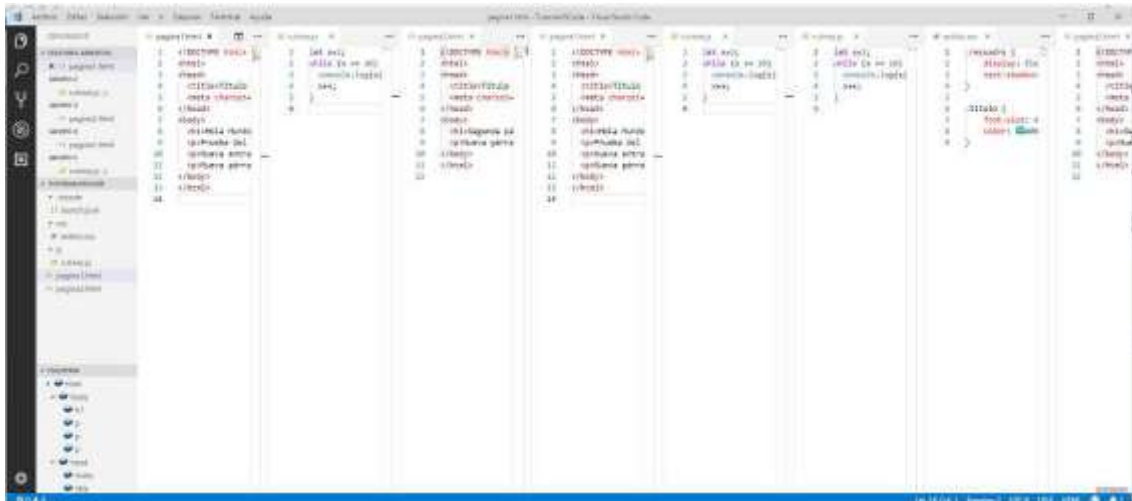


Podemos abrir archivos distintos en forma paralela arrastrando el segundo archivo a un lado o a otro del archivo abierto actualmente:

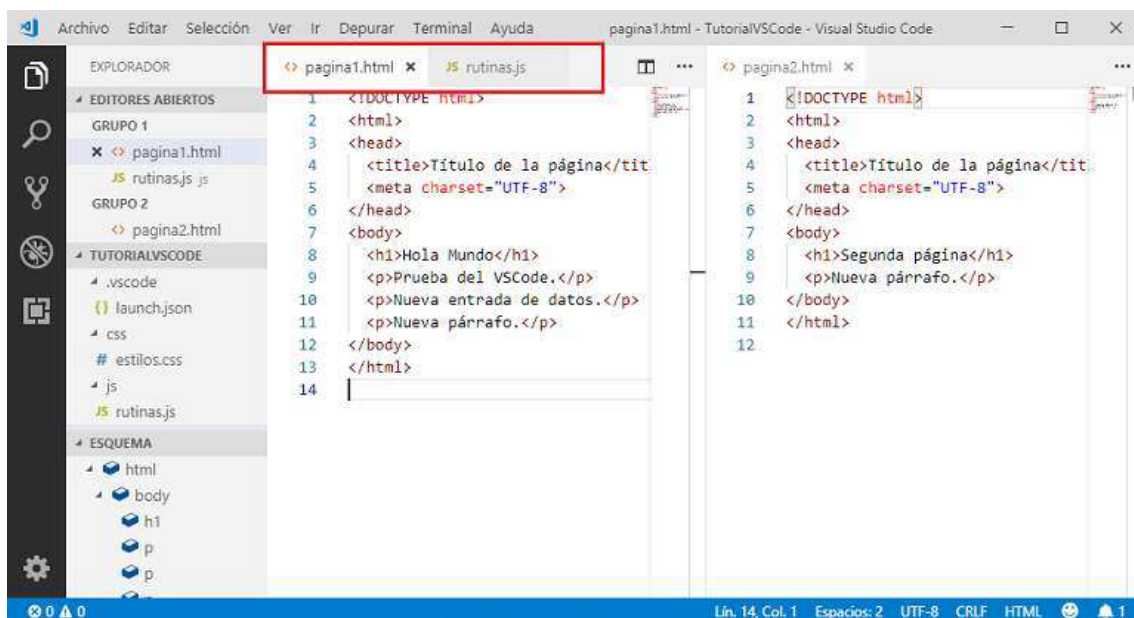


Luego de este proceso tenemos abiertos en simultáneo los dos archivos.

Podemos abrir en forma simultánea muchos archivos en VSCode:

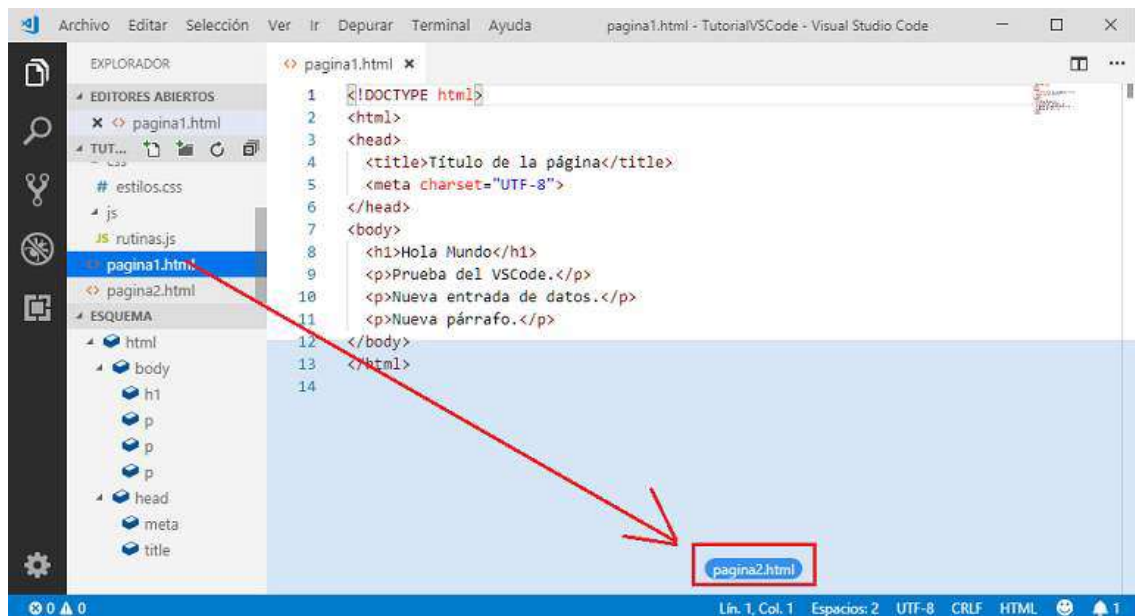


Podemos tener más de un archivo en una de las ventanas de edición:

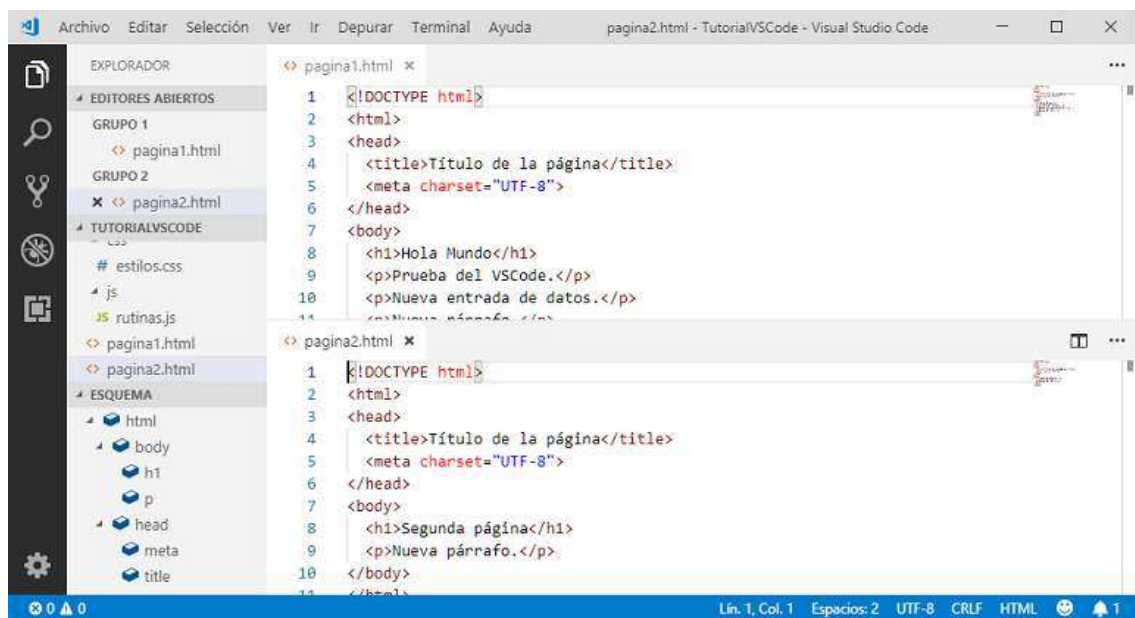


Las teclas de acceso rápido para cambiar entre las distintas ventanas de edición es (Ctrl + 1) o (Ctrl + 2) o (Ctrl + 3)

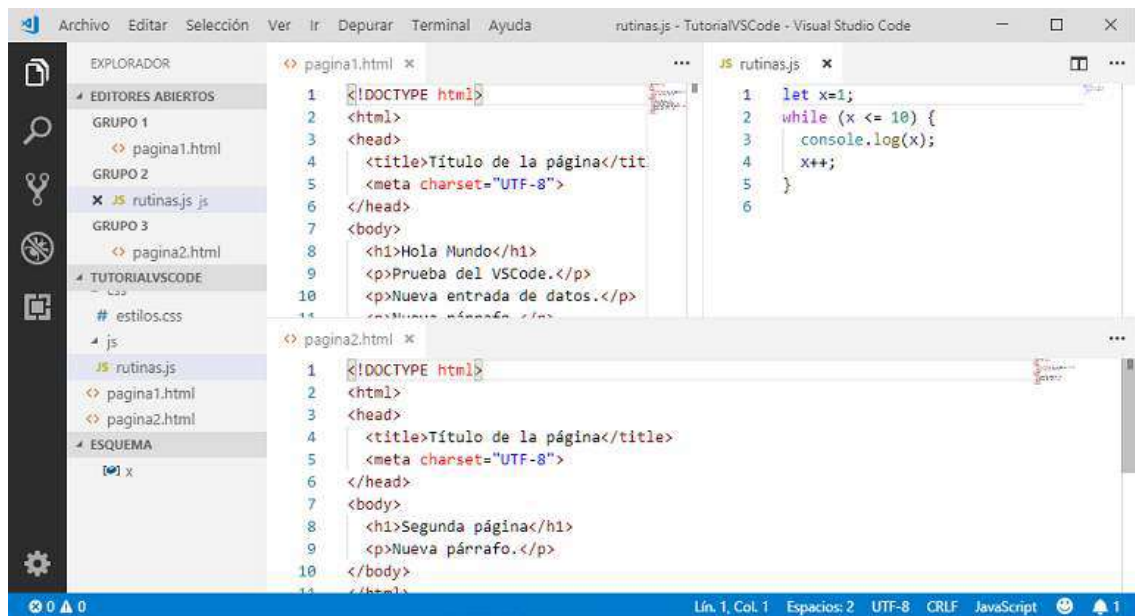
Podemos en VS Code abrir ventanas de edición en forma vertical:



Luego tenemos como resultado:



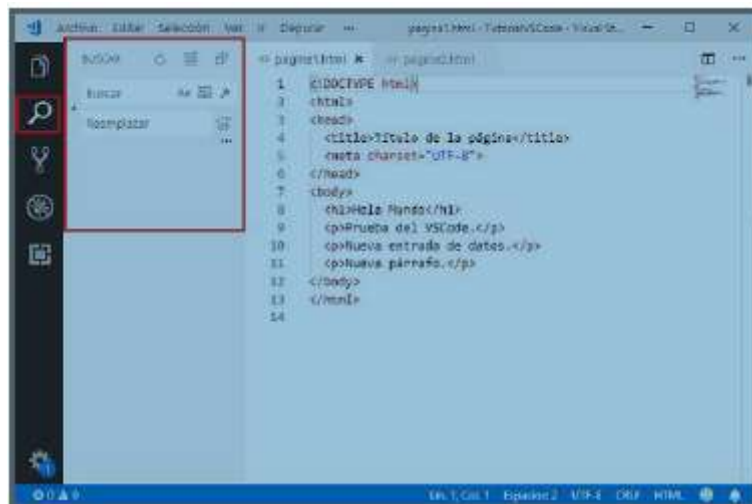
Podemos tener varias ventanas en forma horizontal y vertical:



9 - Buscar en carpetas o áreas de trabajo

Buscar.

La actividad de buscar es muy importante cuando estamos trabajando en un proyecto, por lo que VSCode le reservó un ícono en la "Barra de actividades":



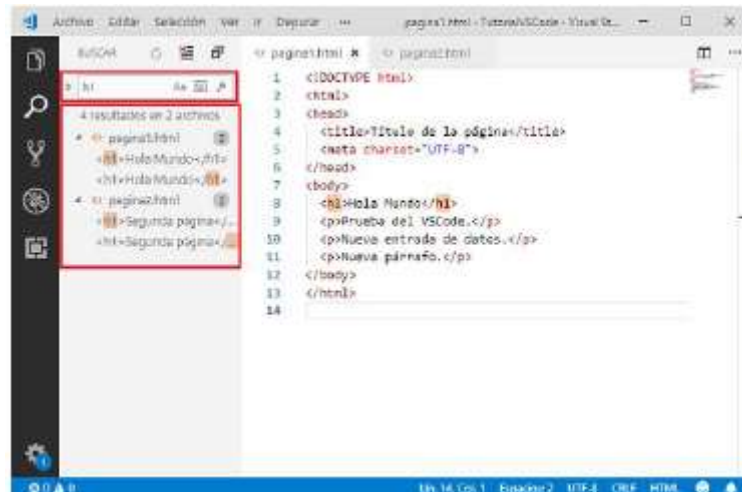
Las teclas de atajos para mostrar la ventana de búsquedas es (Ctrl - Shift - F):



El diálogo de búsqueda tiene tres íconos que nos permiten hacer:

- Coincidir mayúsculas y minúsculas.
- Solo palabras completas.
- Usar expresiones regulares.

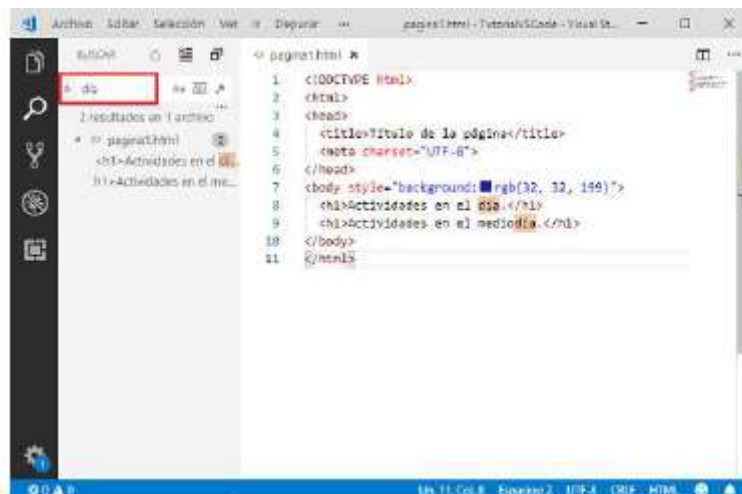
Cuando se confirma la búsqueda se muestra todas las líneas de los diferentes archivos de la carpeta o área de trabajo que coinciden:



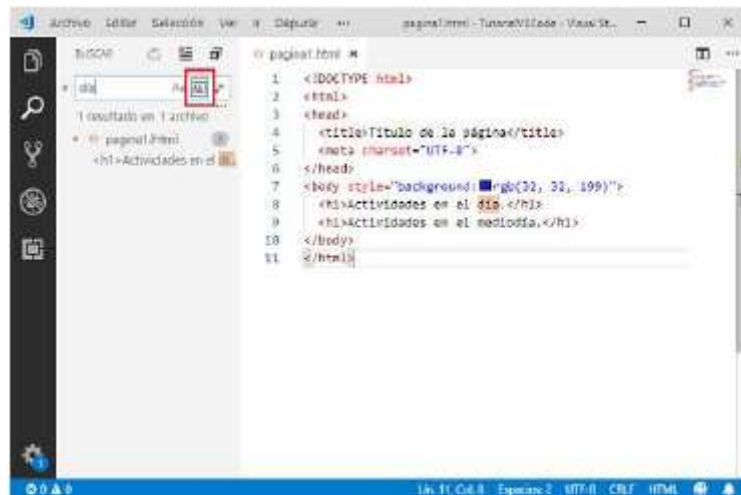
Si tenemos el siguiente archivo dentro de nuestra carpeta abierta en VSCode:

```
<!DOCTYPE html><html><head> <title>Título de la
página</title> <meta charset="UTF-8"></head><body
style="background:rgb(32, 32, 199)"> <h1>Actividades en el
día.</h1> <h1>Actividades en el
mediodía.</h1></body></html>
```

Luego buscamos la cadena 'día' tenemos como resultado:

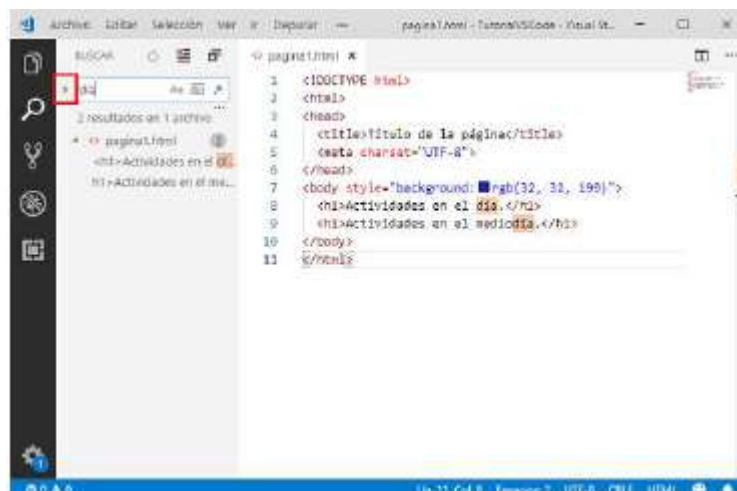


Ahora seleccionemos el ícono de "Solo palabra completa" y el resultado cambia por:

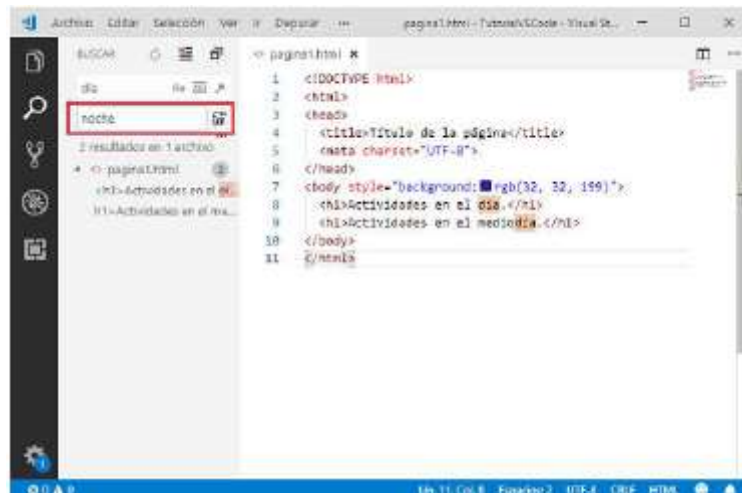


Buscar y reemplazar.

Si necesitamos reemplazar las búsquedas podemos presionar el ícono del lado de la izquierda y nos mostrará un cuadro para indicar por que valor reemplazar:



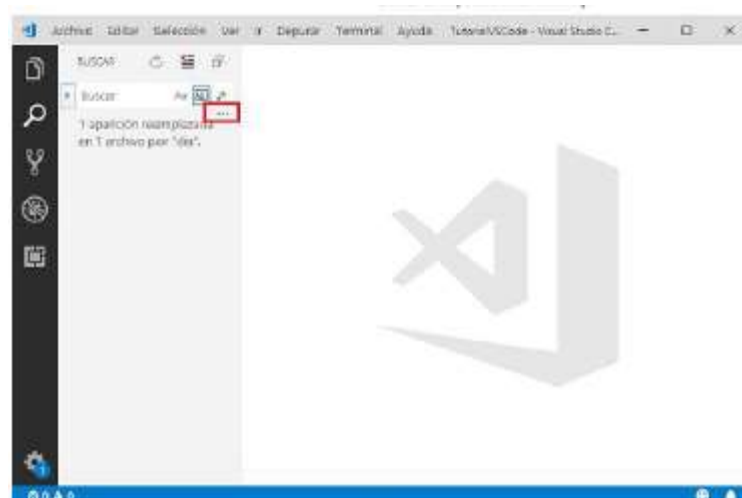
Luego podemos indicar la cadena de reemplazo y efectuar dicha actividad:



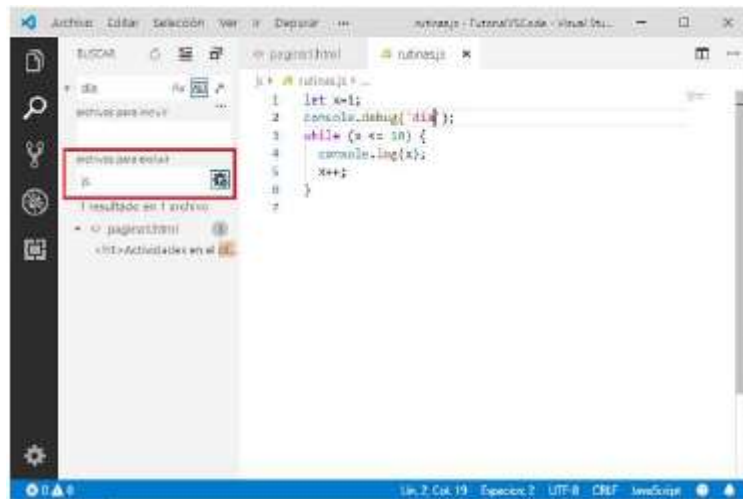
La actividad de remplazo se efectúa cuando presionamos el ícono que aparece inmediatamente después de la cadena de remplazo o mediante las teclas de atajos: Ctrl + Alt+ Enter)

Excluir carpetas y archivos en las búsquedas.

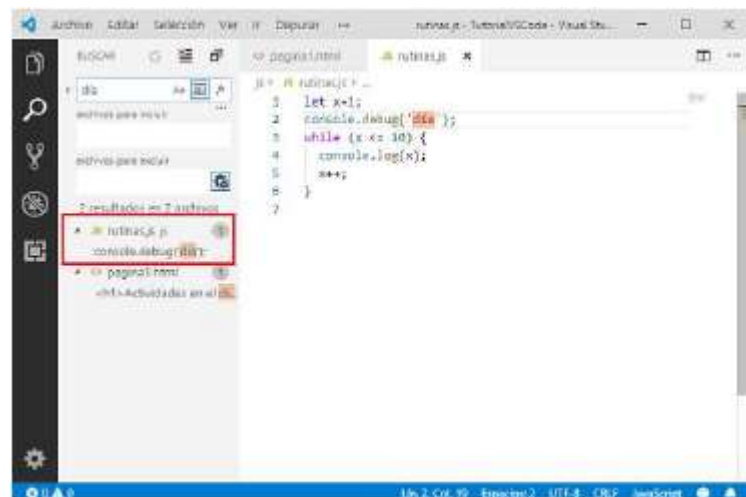
Hay otro ícono en el diálogo de búsqueda que nos permite filtrar en que archivos y carpetas buscar o excluirlos:



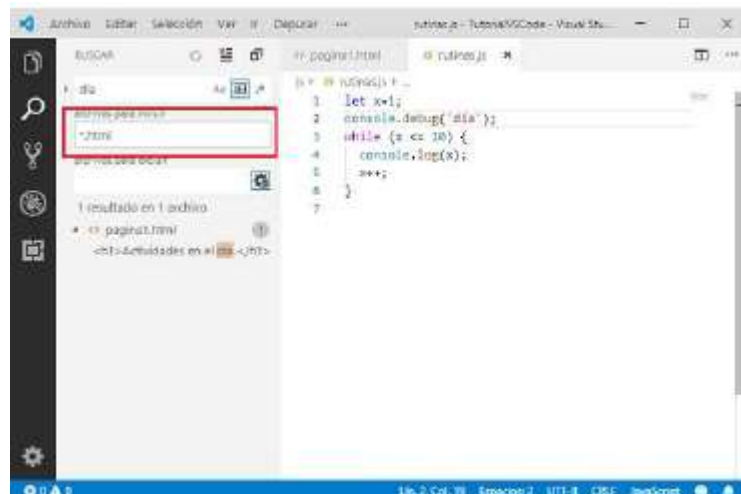
Podemos por ejemplo excluir la búsqueda en los archivos 'js':



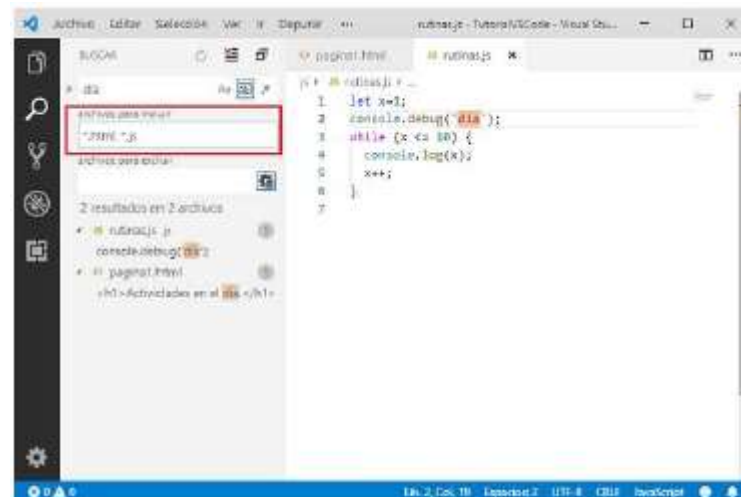
Si desactivamos la exclusión de los archivos js:



Podemos hacer búsquedas en archivos que cumplan que tienen una determinada extensión:

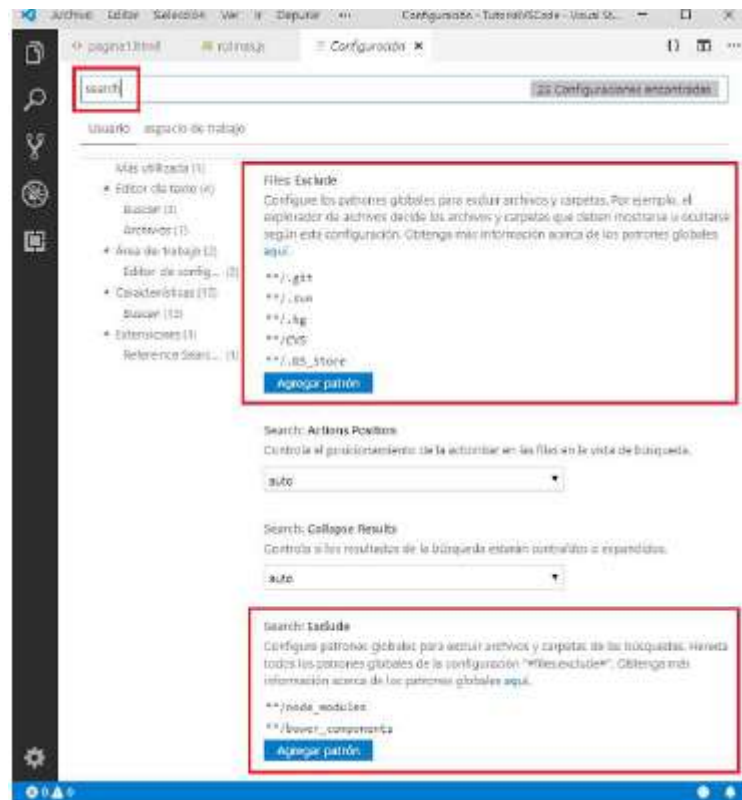


Si tenemos que definir varias reglas de búsqueda debemos separarlas por una coma:



Exclusiones definidas en la configuración.

Por defecto VSCode tiene una serie de carpetas que se excluyen en su búsqueda, podemos acceder a ellos desde "Configuración":



En muchas herramientas web como Angular, Vue, React, NodeJS etc. requieren una serie de librerías que se localizan en la carpeta node_modules y que cada vez que hacemos una búsqueda no tiene sentido procesar los miles de archivos que contiene ésta carpeta.

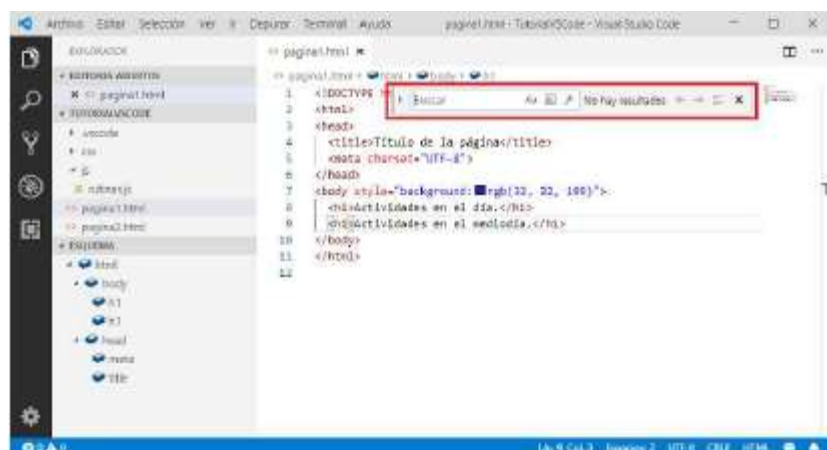
10 - Buscar en un único archivo

Vimos en el concepto anterior como buscar y eventualmente remplazar datos en todos los archivos de una carpeta o área de trabajo.

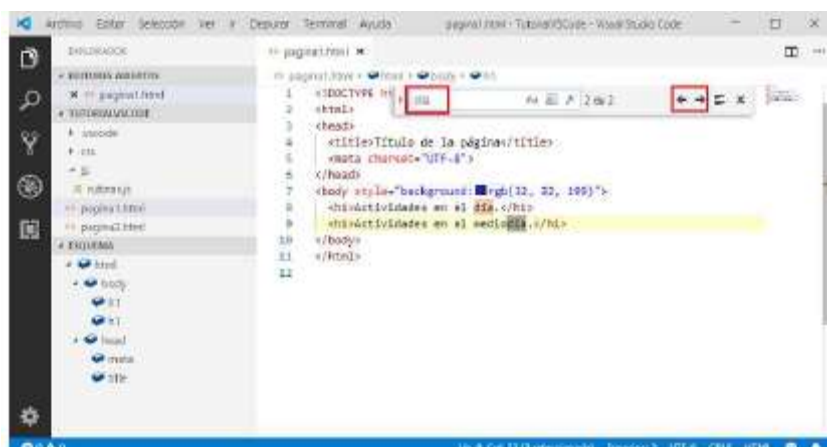
Cuando el proceso de búsqueda lo queremos limitar solo a la pestaña(editor) seleccionada podemos hacerlo mediante las teclas de atajo (Ctrl + F) o desde el menú de barra (Editar ->Buscar):



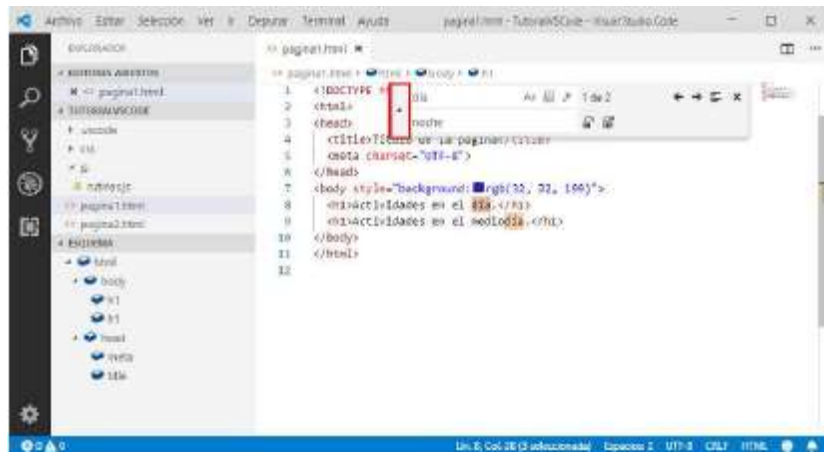
Se nos muestra un diálogo en la parte superior de la pestaña con una interfaz similar a las búsquedas globales:



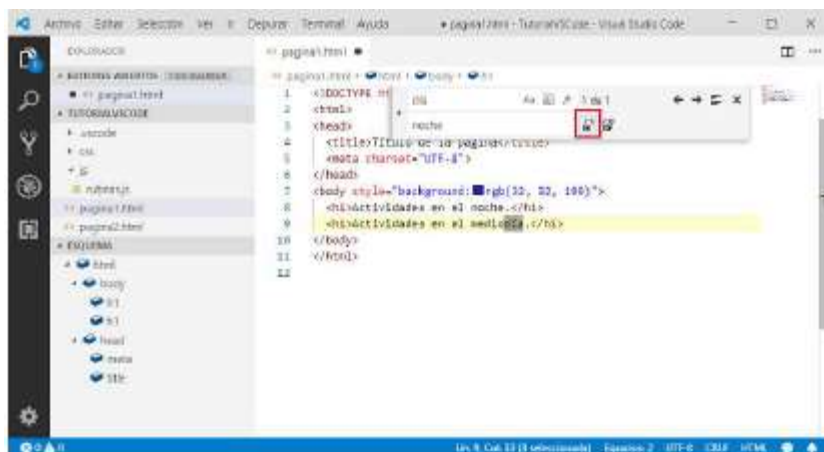
Luego de la búsqueda disponemos de dos botones que nos permiten desplazarnos en cada una de las coincidencias que aparecen dentro del archivo:



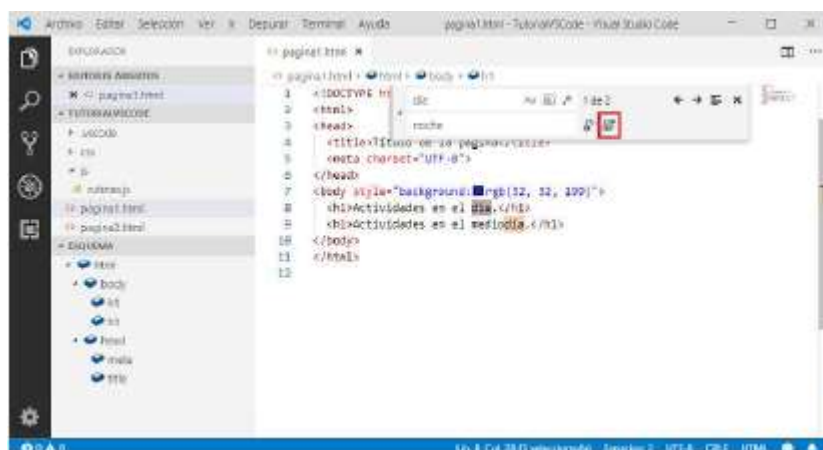
Podemos presionar el botón que aparece a la izquierda de la búsqueda para permitir ingresar un valor de remplazo para las búsquedas que coinciden:



Luego podemos remplazar cada coincidencia una a una mediante el botón:

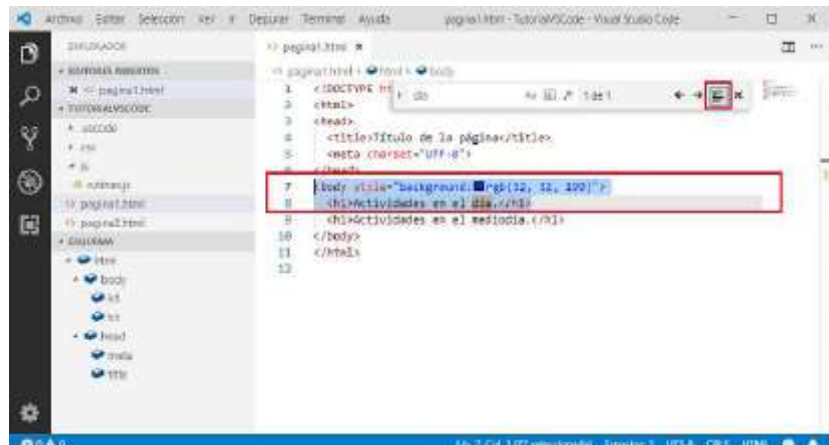


Para un remplazo automático de todas las coincidencias tenemos el otro botón:

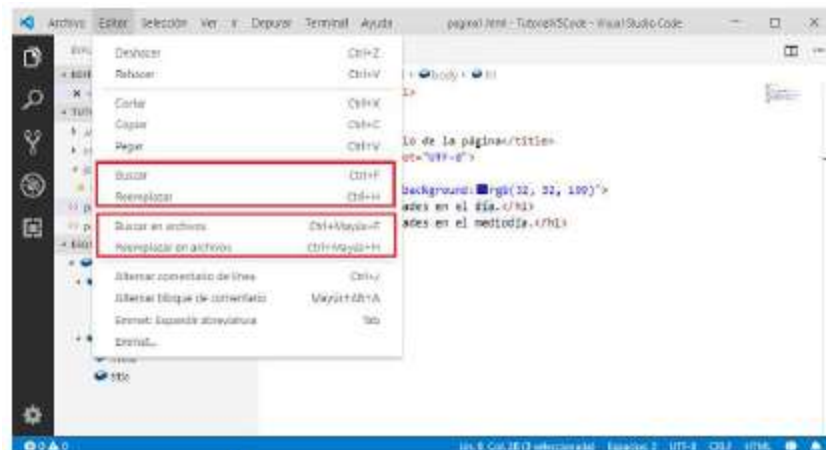


Búsquedas en un bloque

Podemos hacer búsquedas solo en un bloque de líneas seleccionadas, para esto debemos primero seleccionar un conjunto de líneas y seguidamente activar el ícono:



Opciones de buscar en Visual Studio Code.



Todas las opciones de búsqueda que vimos en el concepto anterior y éste se pueden acceder desde "Editar".

Si queremos buscar y reemplazar en la pestaña activa tenemos las opciones:

- Buscar
- Reemplazar

Si queremos buscar y reemplazar en una carpeta o área de trabajo tenemos:

- Buscar en archivos
- Reemplazar en archivos

11 - IntelliSense

El concepto de IntelliSense en Visual StudioCode se lo asocia a un conjunto de funcionalidades de edición de código que incluyen por ejemplo: sugerencia de métodos y propiedades de un objeto, información de parámetros y sus tipos etc.

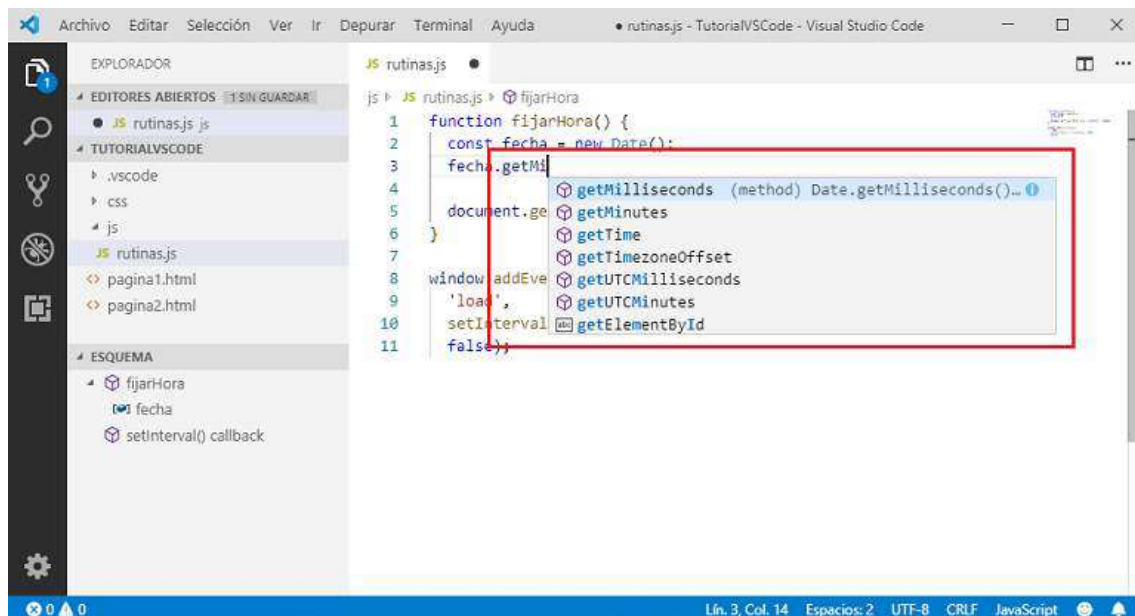
Visual Studio Code trae esta funcionalidad por defecto para los lenguajes JavaScript, TypeScript, CSS, HTML JSON, Less y Sass. Luego veremos que existen muchas "extensiones" que pueden agregar la funcionalidad de IntelliSense a lenguajes tan disímiles como Python, Java, C#, Go, SQL etc.

Para probar su funcionamiento crearemos un archivo JavaScript llamado 'rutinas.js':

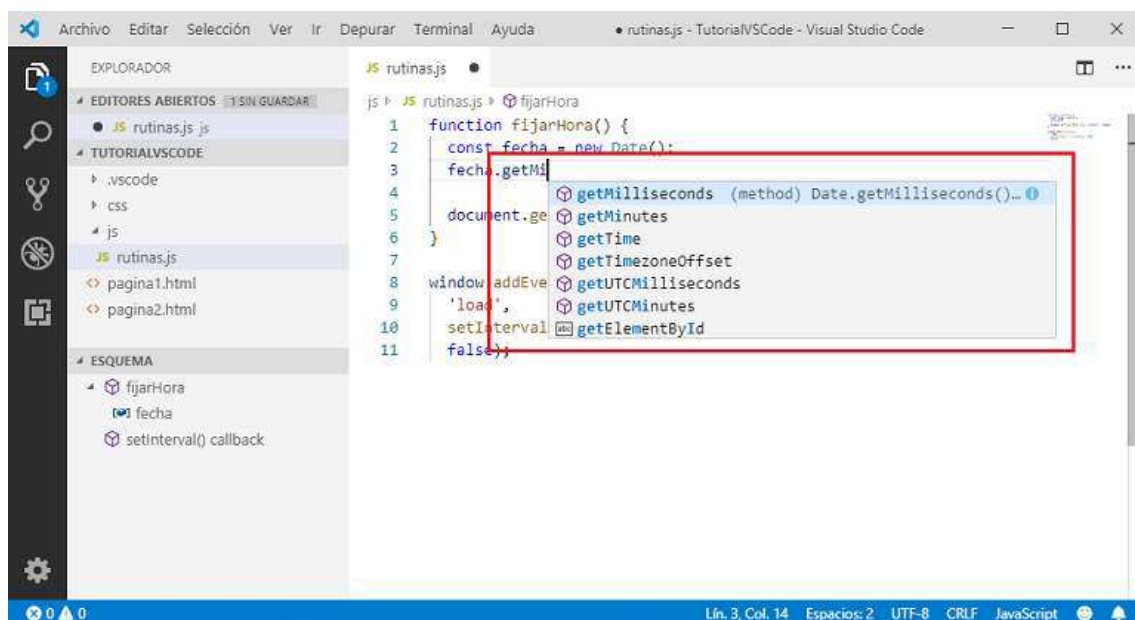
```
function fijarHora() { const fecha = new
Date(); let cadena = fecha.getHours() + ':' +
fecha.getMinutes
document.getElementById('hora').innerText =
cadena; }
```

```
window.addEventListener( 'load', setInterval(()
=> fijarHora(), 1000), false);
```

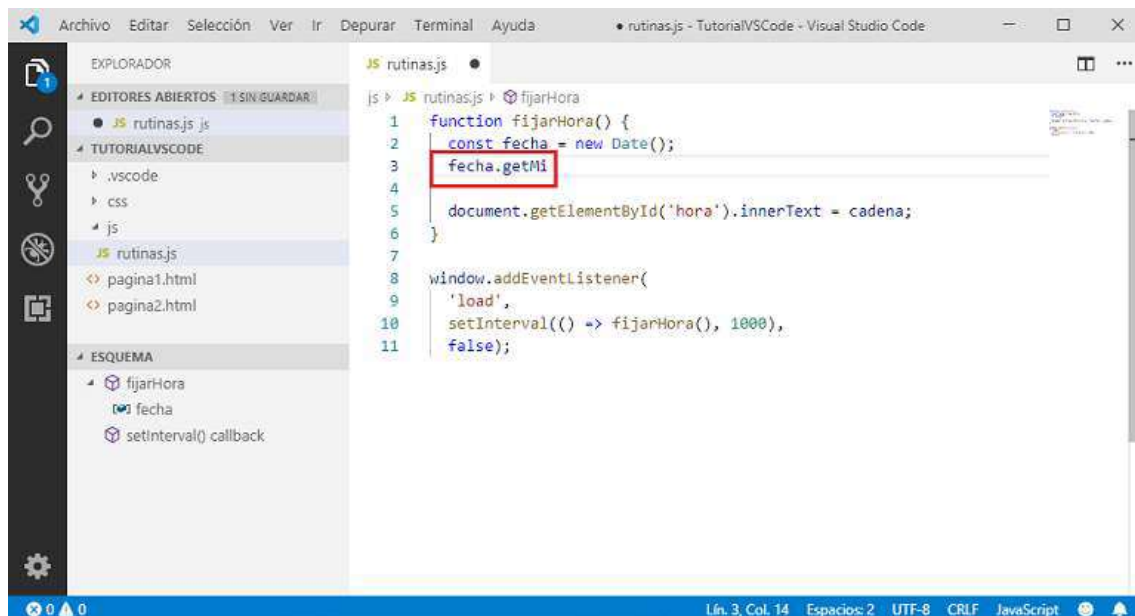
Luego de crear un objeto de una determinada clase cada vez que disponemos un punto luego del nombre del objeto se activa IntelliSense para sugerirnos valores permitidos:



En este ejemplo a medida que escribimos el método que necesitamos llamar veremos que las opciones se van filtrando:



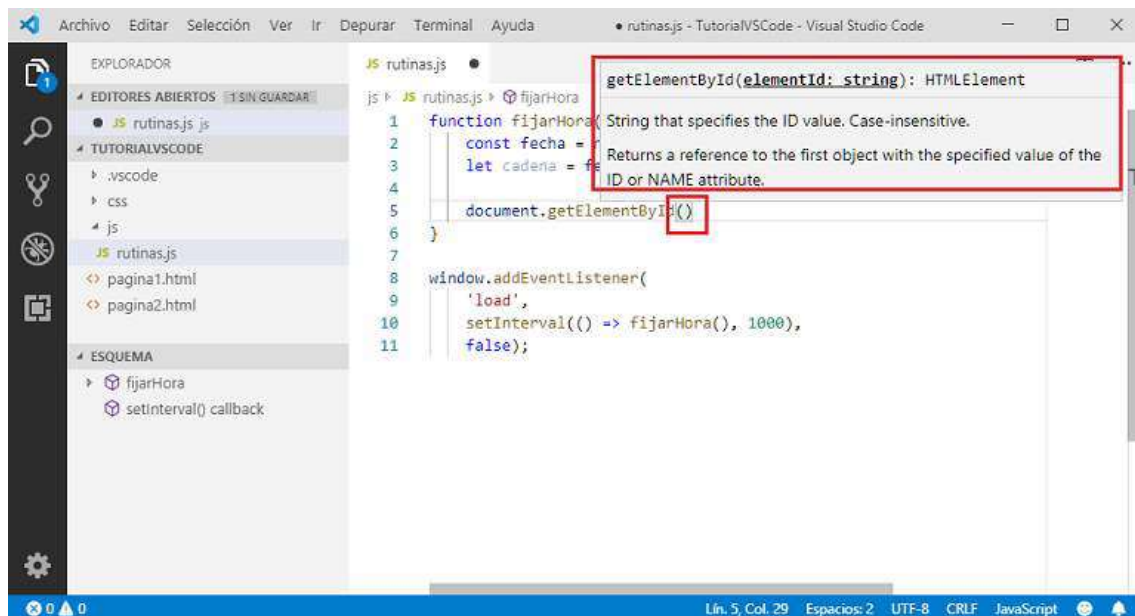
Si se nos cierra el cuadro de sugerencias por ejemplo al cambiar de ventana o cualquier otro motivo:



Podemos activarlo fácilmente por medio del atajo de teclado (Ctrl + Espacio):

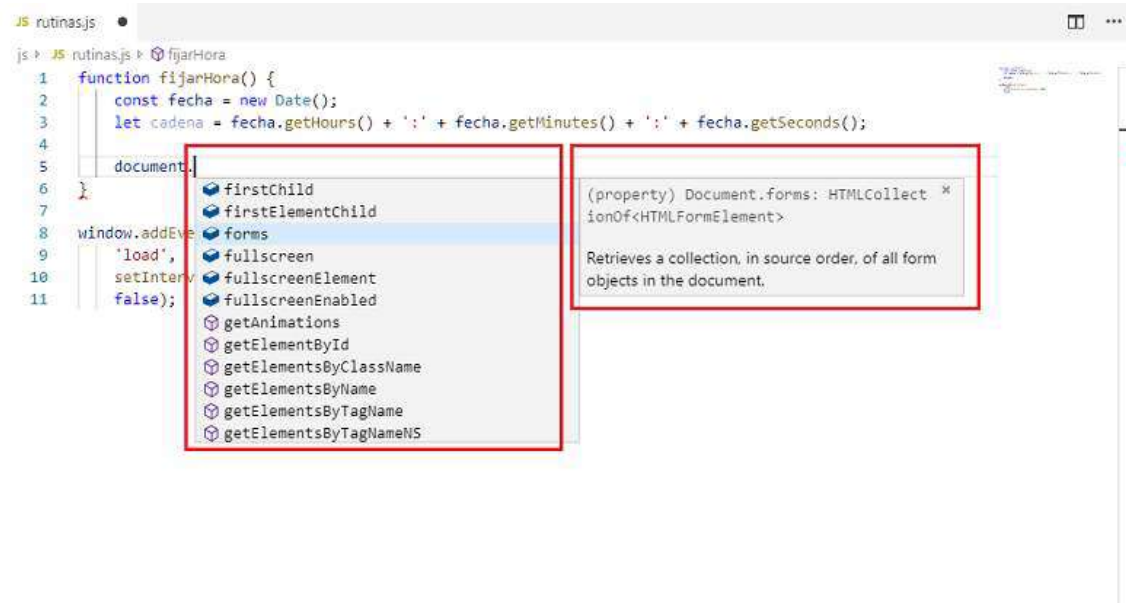


Lo mismo sucede que se activa "IntelliSense" cuando comenzamos a escribir los parámetros de un método o función:

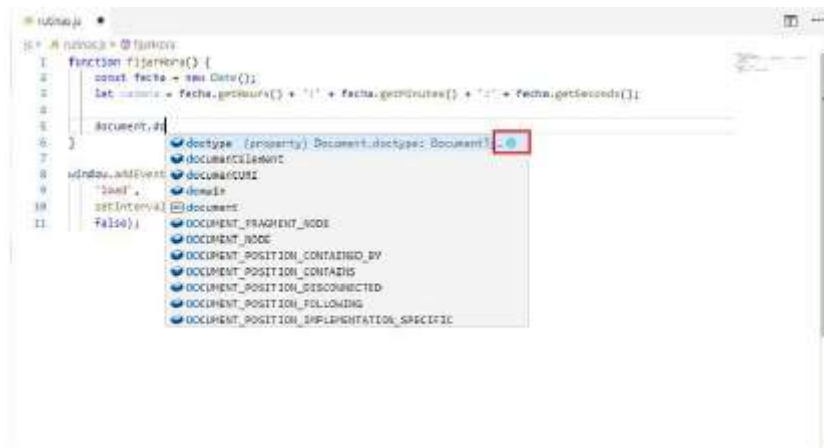


Nos informa la cantidad y tipo de parámetros, así como una descripción del mismo.

Si necesitamos una información ampliada de cada propiedad o método que se sugiere debemos presionar las teclas (Ctrl + Espacio) estando activas las sugerencias y veremos que aparece una segunda ventana con la explicación del método o propiedad, si mediante las teclas de flechas nos desplazamos entre los métodos podremos ir consultando cada definición, para ocultarla debemos presionar nuevamente (Ctrl + Espacio):



También podemos activar la explicación extendida presionando el ícono de color azul:



Iconos

Cuando aparece el cuadro de sugerencias de "IntelliSense", del lado izquierdo se muestran una serie de íconos que tienen un determinado significado:



Detección de errores

Cuando introducimos errores en nuestro código, VSCode nos sugiere posibles correcciones. Probemos de ingresar un objeto literal sin separar sus propiedades por coma: `let actual = { dia: 1 mes: 5 año: 2018 }`

Luego veremos que aparece subrayado en rojo el atributo `mes`. Si disponemos la flecha del mouse sobre el identificador nos sugerirá una posible corrección:



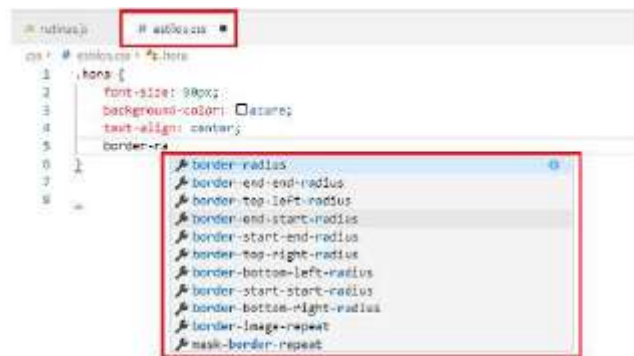
Una vez

que efectuamos las correcciones desaparecen los subrayados rojos:



Sugerencias según el tipo de archivo.

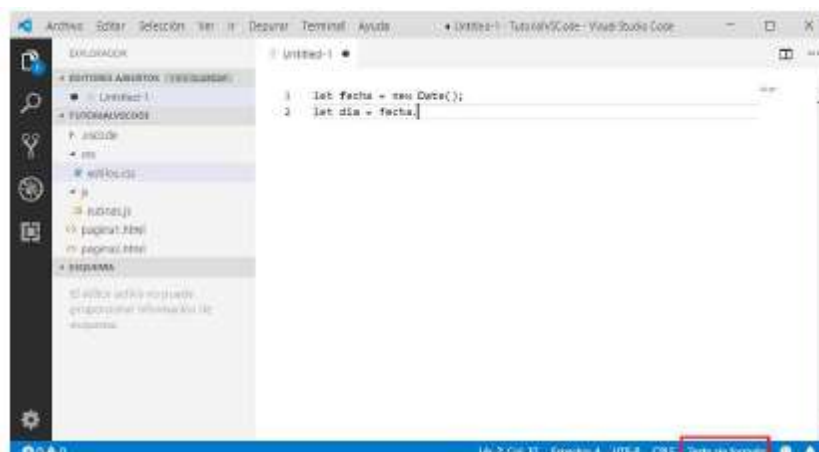
IntelliSense actúa según el tipo de archivo que estamos trabajando, por ejemplo si editamos un archivo CSS las sugerencias serán adecuadas para este formato:



Es
important
el
mensaje
que nos

muestra VSCode en la parte inferior derecha, aquí nos informa el formato de archivo que estamos trabajando.

Es común que a veces creamos un archivo (Ctrl + N) y no le asignemos un nombre al principio, en estos casos VSCode no puede inferir el formato de datos que cargamos y no aparecerá activo IntelliSense:

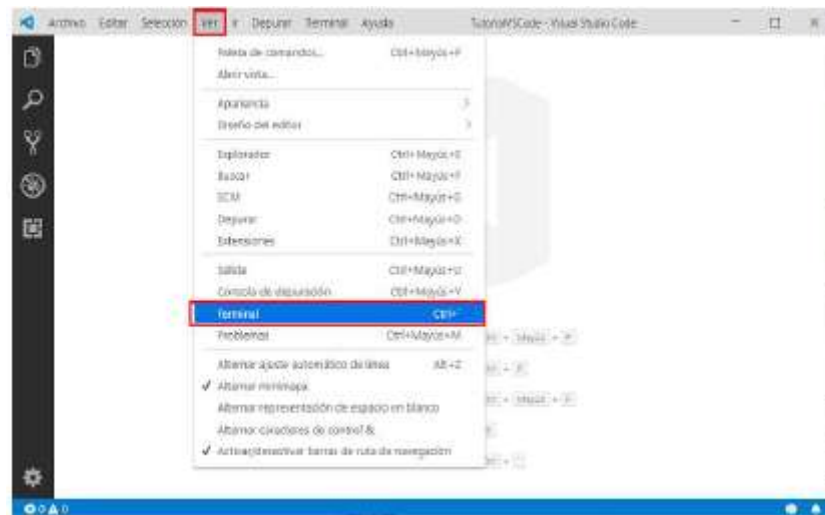


En la parte inferior aparece un mensaje de "Texto sin formato". Podemos cambiar este valor por el formato de nuestro archivo, en ese momento ya tendremos activas algunas funcionalidades de IntelliSense, de todos modos para la activación completa debemos guardarlo en disco.

12 - Terminal integrado en VSCode

VSCode ha incorporado la posibilidad de mostrar un terminal de comandos del sistema operativo donde se ejecuta.

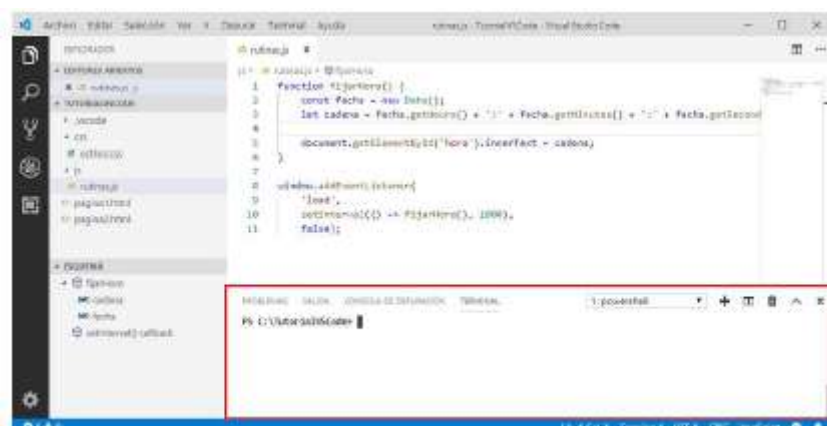
Para activar la terminal podemos hacerlo mediante el menú de opciones:



También lo podemos abrir a la terminal con el atajo de teclas (Ctrl + `):



La consola normalmente se muestra en la parte inferior de pantalla dentro del editor VSCode:



Ejemplo de uso.

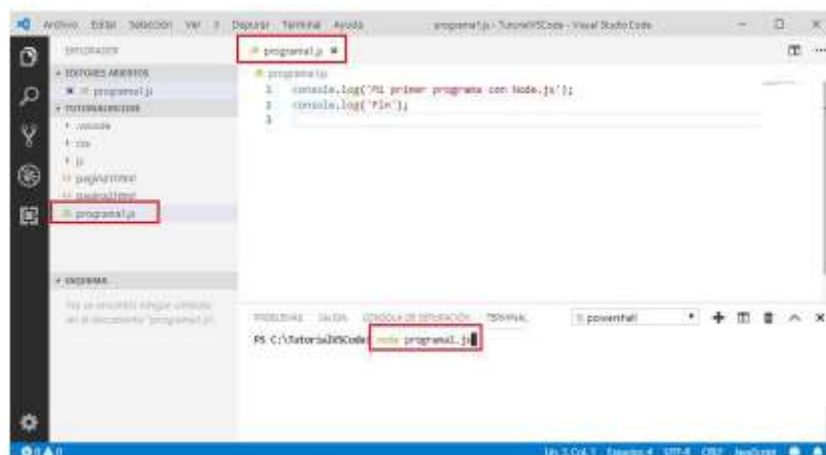
Vamos a instalar NodeJS para hacer unas pruebas como podemos ejecutar las aplicaciones codificadas con esta herramienta

El primer paso será descargar e instalar NodeJS.

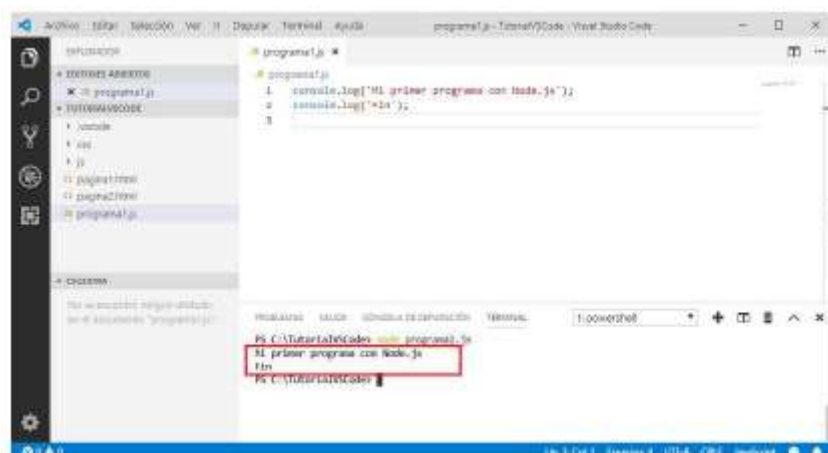
El objetivo no es estudiar NodeJS sino ver como podemos interactuar con el mismo desdeVSCode: codificar los programas y ejecutarlos desde el mismo entorno.

Una vez instalado NodeJS procederemos a codificar un programa mínimo (creamos elarchivo 'programa1.js'):console.log('Mi primer programa con Node.js');console.log('Fin');

Ahora desde la terminal podemos ejecutar nuestra primera aplicación en Node.JS:



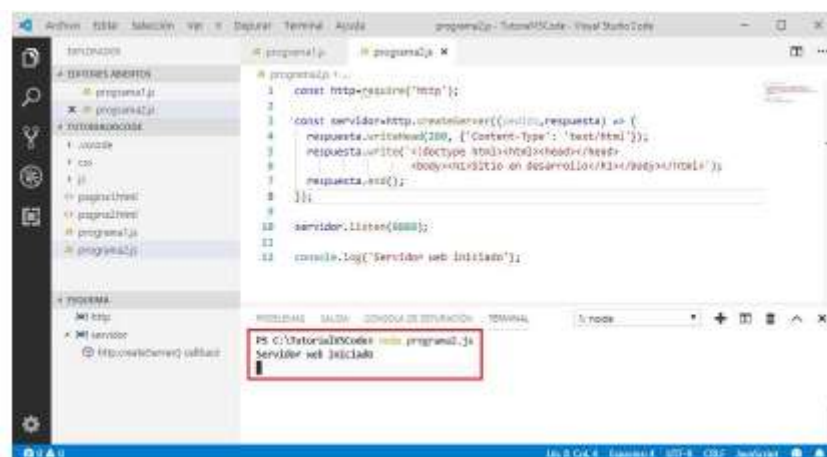
En Node.JS cuando utilizamos el método log del objeto Console los datos se muestran en la misma terminal:



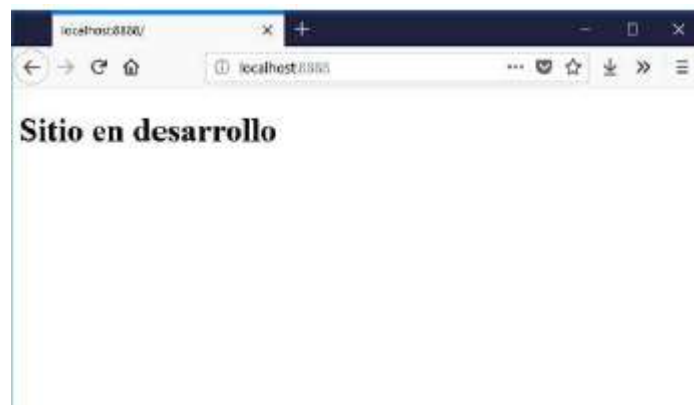
Desarrollaremos una segunda aplicación Node.JS que cree un servidor web ('programa2.js'):

```
const http=require('http');const
servidor=http.createServer((pedido,respuesta)
=> { respuesta.writeHead(200, {'Content-Type':
'text/html'}); respuesta.write(`<!doctype
html><html><head></head> <body><h1>Sitio en
desarrollo</h1></bod
respuesta.end();});servidor.listen(8888);consol
e.log('Servidor web iniciado');
```

Ahora podemos lanzar la aplicación desde el terminal:



El servidor web creado con Node.JS está en ejecución y esperando peticiones en el puerto8888, ahora solo nos queda desde el navegador hacer la petición a dicho servidor:



Hemos visto el uso de la terminal aplicada a Node.JS pero existen muchos otros lenguajes que podemos ejecutarlos y/o compilarlos desde la terminal: Python, Java, Ruby, Go etc. En conceptos futuros veremos otros entornos que requieren la consola integrada de VSCode.

14 - Extensiones

Una de las opciones fundamentales del editor de texto VSCode es la posibilidad de agregar características mediante las "extensiones".

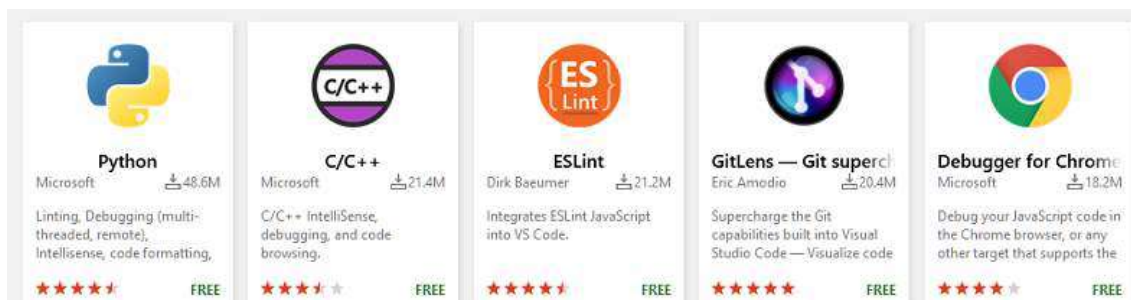
Algo ya hemos trabajado con esto cuando agregamos extensiones sobre "Temas de color" y "Temas de íconos de archivo".

Existen miles de "extensiones" con objetivos tan diversos como:

- Lenguajes de programación
- Depuradores
- Formateadores de código fuente
- Temas
- Teclas de acceso rápido de otros editores

Para consultar la tienda de extensiones para VSCode podemos visitar marketplace.visualstudio.com/vscode (<https://marketplace.visualstudio.com/vscode>)

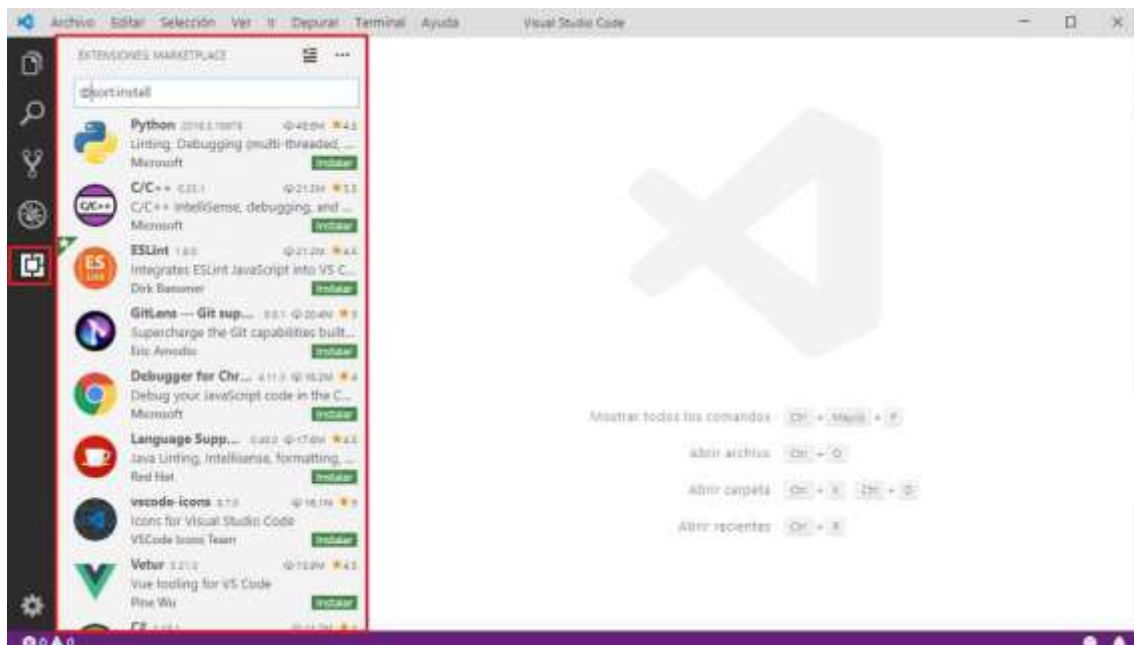
La popularidad de las extensiones la podemos comprobar cuando vemos la cantidad de descargas de las mismas (la extensión de Python tiene más de 48 millones de descargas):



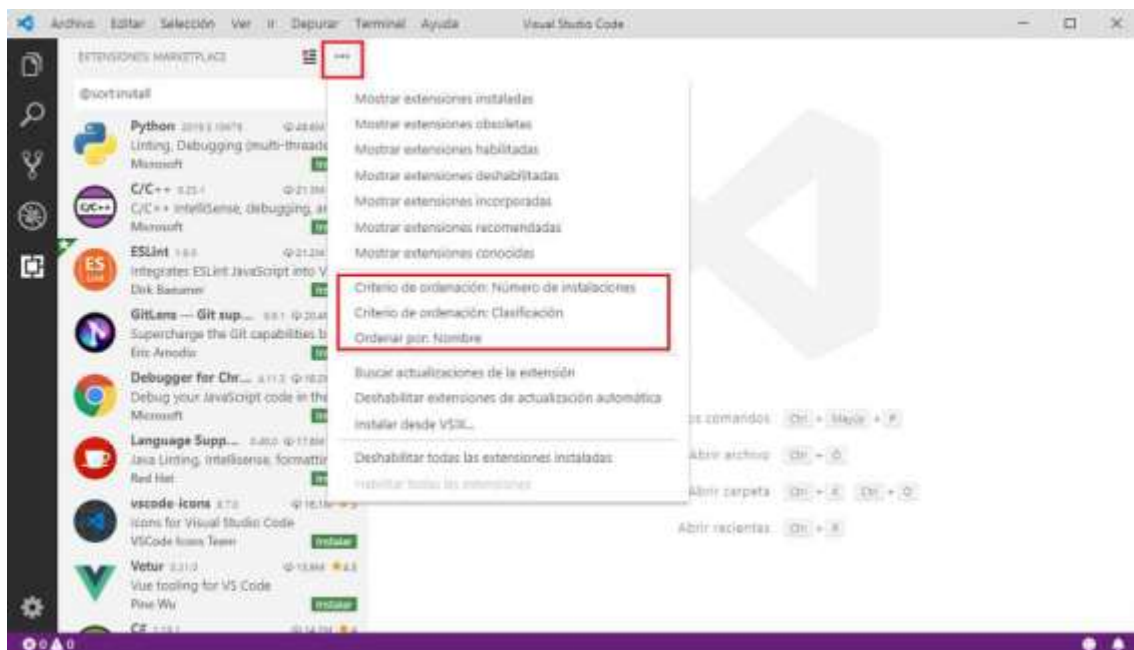
Instalación de una "Extensión"

Para conocer los pasos para la instalación de una extensión instalaremos la de Python, para eso primero descargaremos el lenguaje Python 3.x del sitio [Python.org](https://www.python.org)

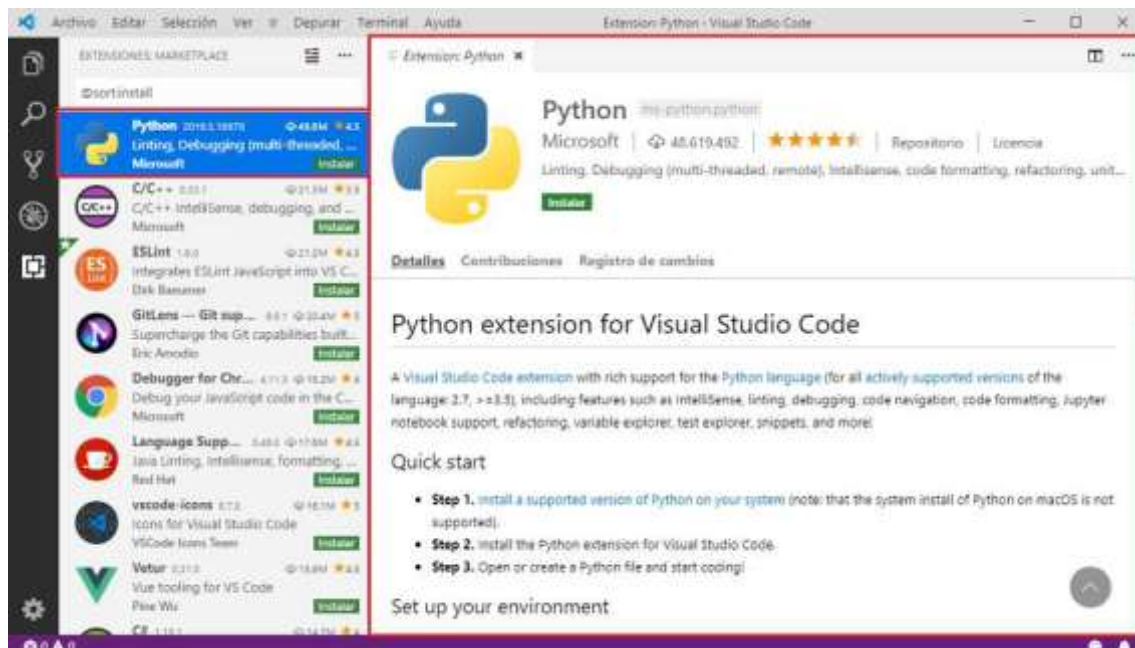
Una vez que hayamos instalado el lenguaje Python procederemos a instalar la extensión de Python en VSCode. Desde la "Barra de actividades" a través del ícono: (<https://www.python.org/downloads/>)



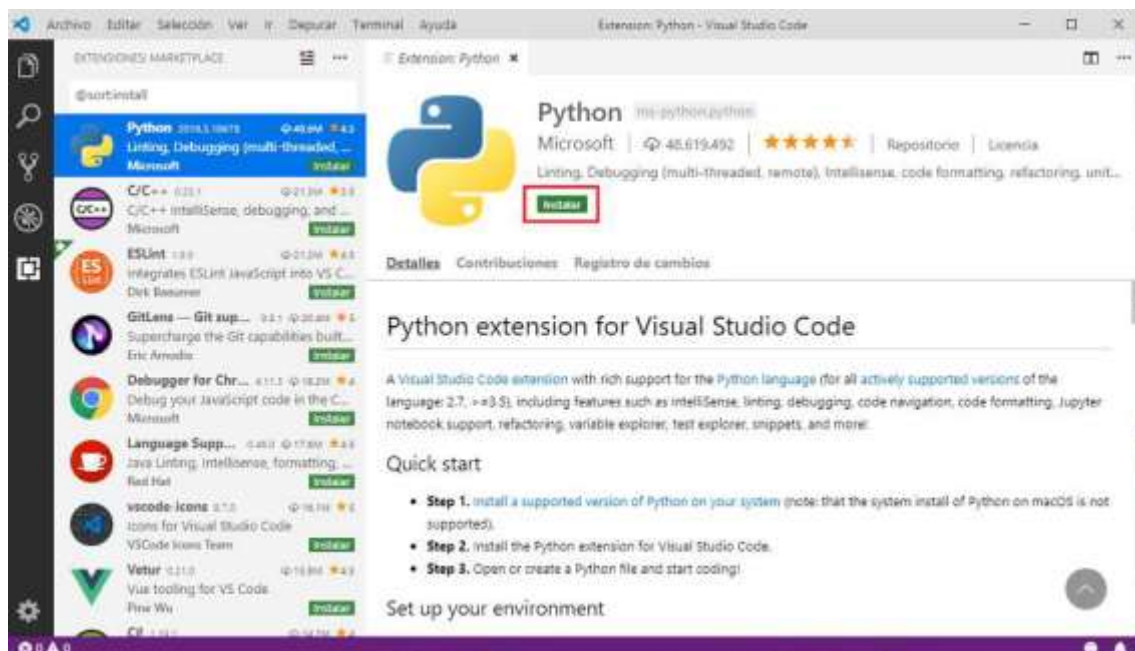
Aparecen ordenadas por popularidad debido al texto ingresado: @sort:installs. Pero podemos mostrarlas ordenadas por otro filtro que podemos activarlo mediante el menú de opciones del botón superior:



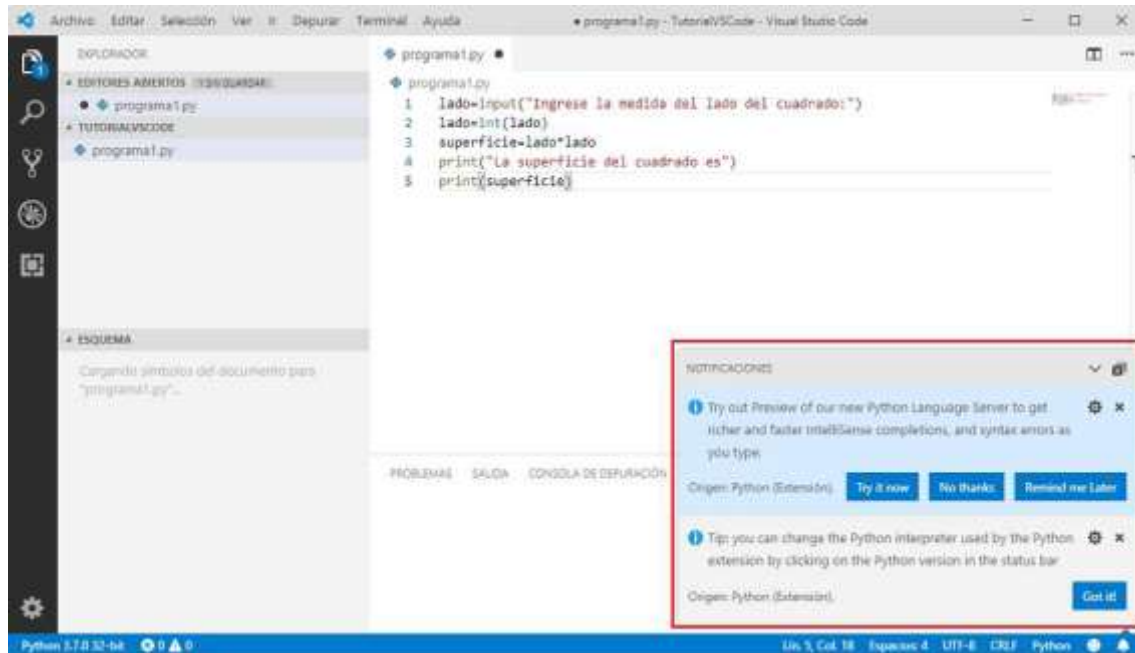
Cuando seleccionamos una extensión se nos muestra una pestaña con toda la información de dicha extensión:



Procederemos a su instalación presionando el botón "instalar" que aparece dentro de la pestaña de la extensión:



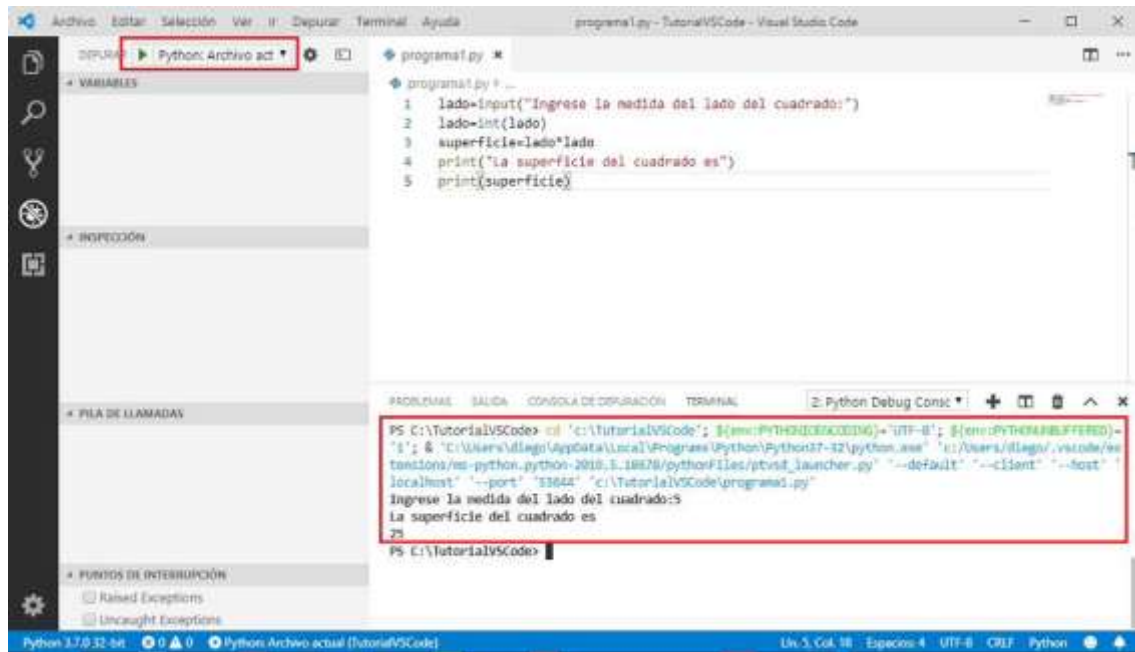
El VSCode dependiendo del ambiente de trabajo nos puede mostrar notificaciones que nos informan de configuraciones necesarias u otras extensiones a instalar (las notificaciones aparecen en la parte inferior derecha):



Ya tenemos todo listo para probar nuestro primer programa en Python, procedemos a crear un archivo 'programa1.py' y codificar:

```
lado=input("Ingrese la medida del lado del cuadrado:")
lado=int(lado)
superficie=lado*lado
print("La superficie del cuadrado es")
print(superficie)
```

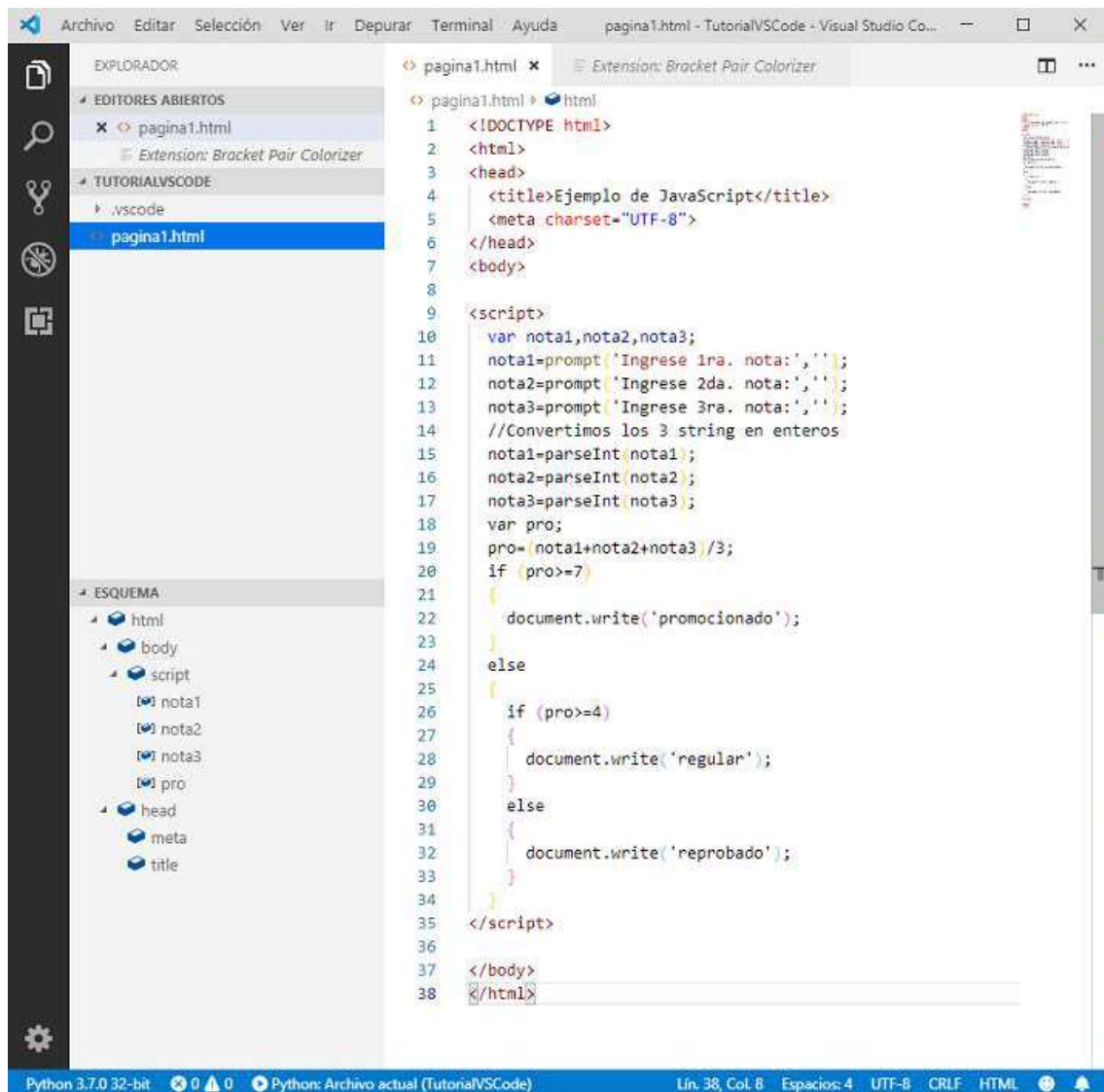
Podemos ahora abrir la opción de depurar y proceder a ejecutar nuestra aplicación en Python:



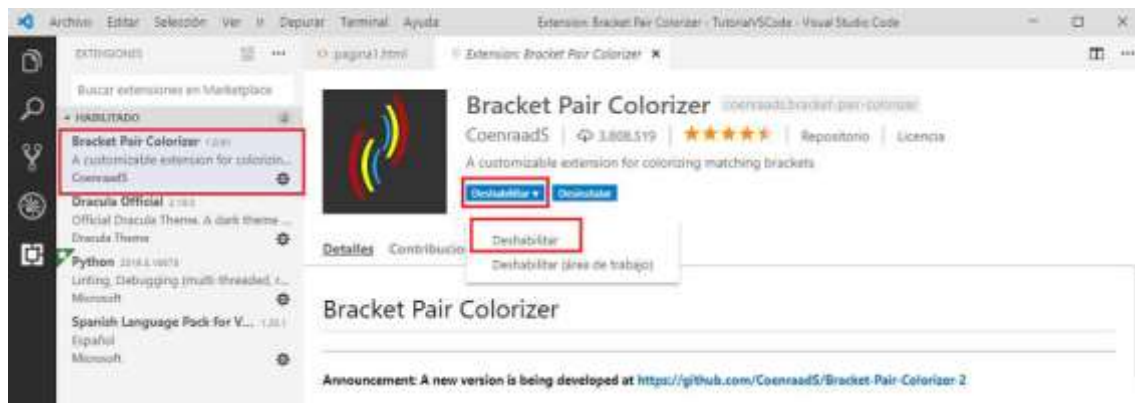
El objetivo no es aprender Python sino entender los pasos de instalación de una "Extensión".

Problema

Buscar la extensión "Bracket Pair Colorizer" y proceder a su instalación. Verificar luego teniendo un archivo HTML con JavaScript si las llaves aparecen de distinto color según su anidamiento. En pantalla debería aparecer algo similar a esto:



Cuando no utilizamos una extensión es conveniente deshabilitar para que VSCode cargue más rápido, para esto seleccionamos el ícono de extensiones y procedemos a deshabilitarla:



Podemos tener distintas áreas de trabajo con ciertas extensiones habilitadas.

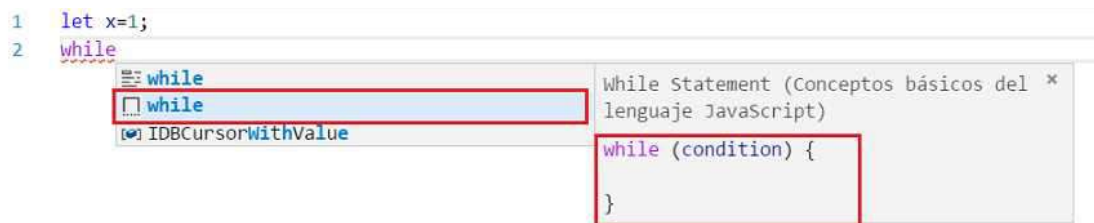
Si no vamos a utilizar más una "Extensión" podemos proceder a su desinstalación.

- Fragmentos de código (snippets)

Los fragmentos de código son un bloque de una o más instrucciones que el programador debe ingresar en forma repetitiva a lo largo de la codificación de un programa. Las estructuras repetitivas, condicionales, bloques de comentarios etc. son instrucciones que debemos codificar constantemente en nuestro programa y los "Fragmentos de código" nos automatizan el ingreso de los mismos.

Los fragmentos de código son particulares de cada lenguaje de programación. Los lenguajes básicos que soporta VS Code como JavaScript, TypeScript, HTML etc. incorporan fragmentos de código básicos. Los fragmentos de código se muestran junto a IntelliSense.

Veamos como insertar un fragmento de código cuando codificamos un programa en JavaScript, supongamos que tenemos que mostrar los números del 1 al 10 por la consola, en el momento que disponemos el bloque while se nos muestra la posibilidad de insertar un "Fragmento de código":



Es decir se nos propone sustituir:

while

Por el fragmento (snippet): `while (condition) { }`

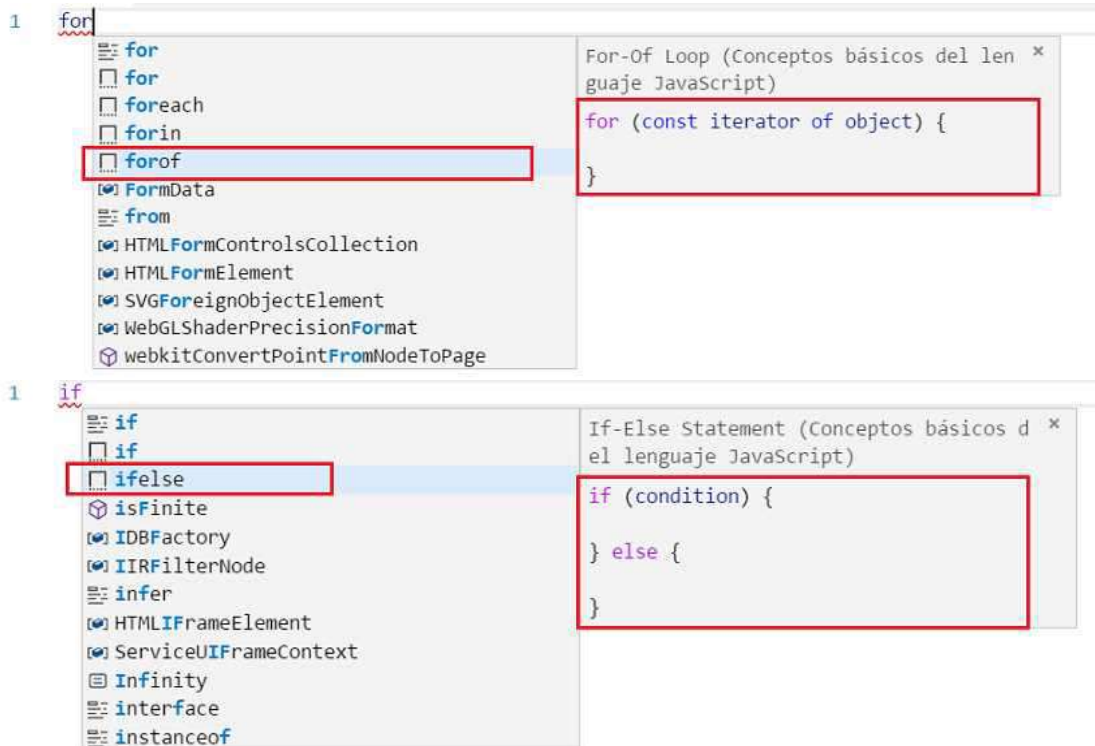
Luego de confirmar el fragmento el cursor se posiciona en la condición del 'while' y podemos codificarla:

```
1 let x=1;
2 while (condition) {
3   |
4 }
```

Finalmente luego de disponer la condición presionamos la tecla 'TAB' y procedemos a codificar el bloque repetitivo:

```
1 let x=1;
2 while (x<=10) {
3   |
4 }
```

El fragmento de código nos agiliza la codificación de bloques de programación. Hay muchos fragmentos configurados por defecto para JavaScript:



Fragmentos de código según el lenguaje.

Los fragmentos de código que propone IntelliSense dependen del lenguaje que estamos codificando. Cuando instalamos extensiones de otros lenguajes las mismas incorporan fragmentos propios.

Si estamos codificando en Python y hemos instalado su extensión luego se nos muestran fragmentos como por ejemplo:

The screenshot shows two examples of Python code snippets suggested by IntelliSense. In the first example, the word 'if' is typed, and a list of suggestions appears: 'if', 'if', 'if(main)', and 'if/else'. The 'if/else' option is highlighted with a red box. To the right, a preview shows the code snippet: 'if condition: pass else: pass'. In the second example, the word 'class' is typed, and suggestions include 'class', 'class', and 'classmethod'. The 'class' option is highlighted with a red box. To the right, a preview shows the code snippet: 'class classname(object): pass'.

```
1 if
```

```
class
```

Si hemos instalado la extensión del lenguaje C# que suministra Microsoft, luego los fragmentos de código serán:

The screenshot shows two examples of C# code snippets suggested by IntelliSense. In the first example, the word 'class' is typed, and a suggestion 'class' is highlighted with a red box. To the right, a preview shows the code snippet: 'class Name { }'. In the second example, the word 'for' is typed within a method body, and suggestions include 'for', 'foreach', and 'forr'. The 'for' option is highlighted with a red box. To the right, a preview shows the code snippet: 'for (int i = 0; i < length; i++) { }'.

```
class
```

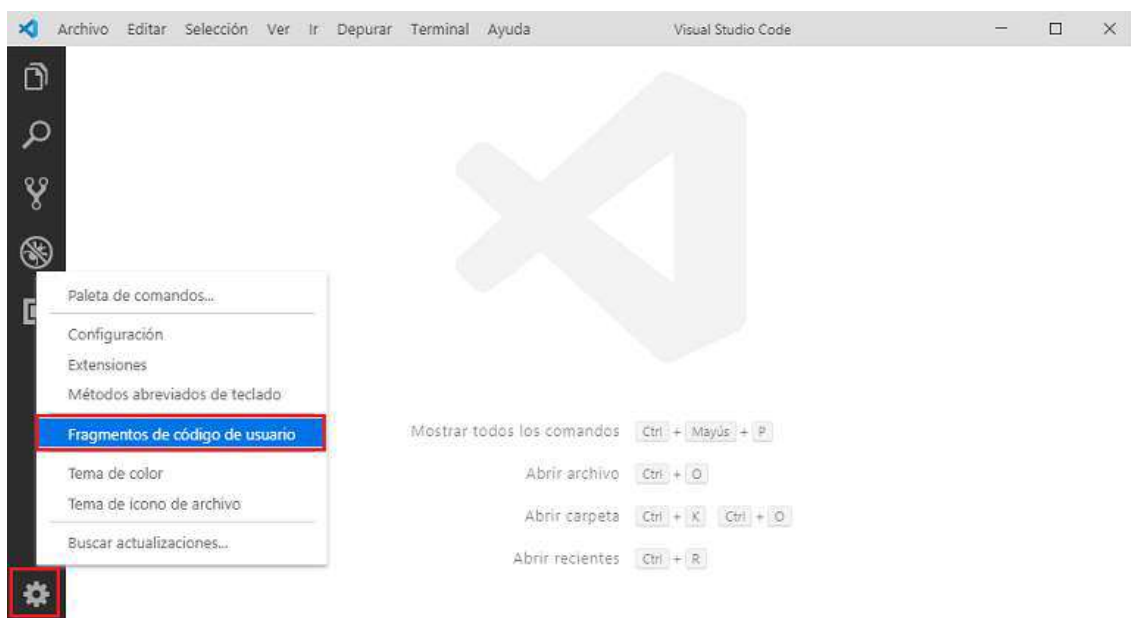
```
class Programa
```

- Fragmentos de código personalizados

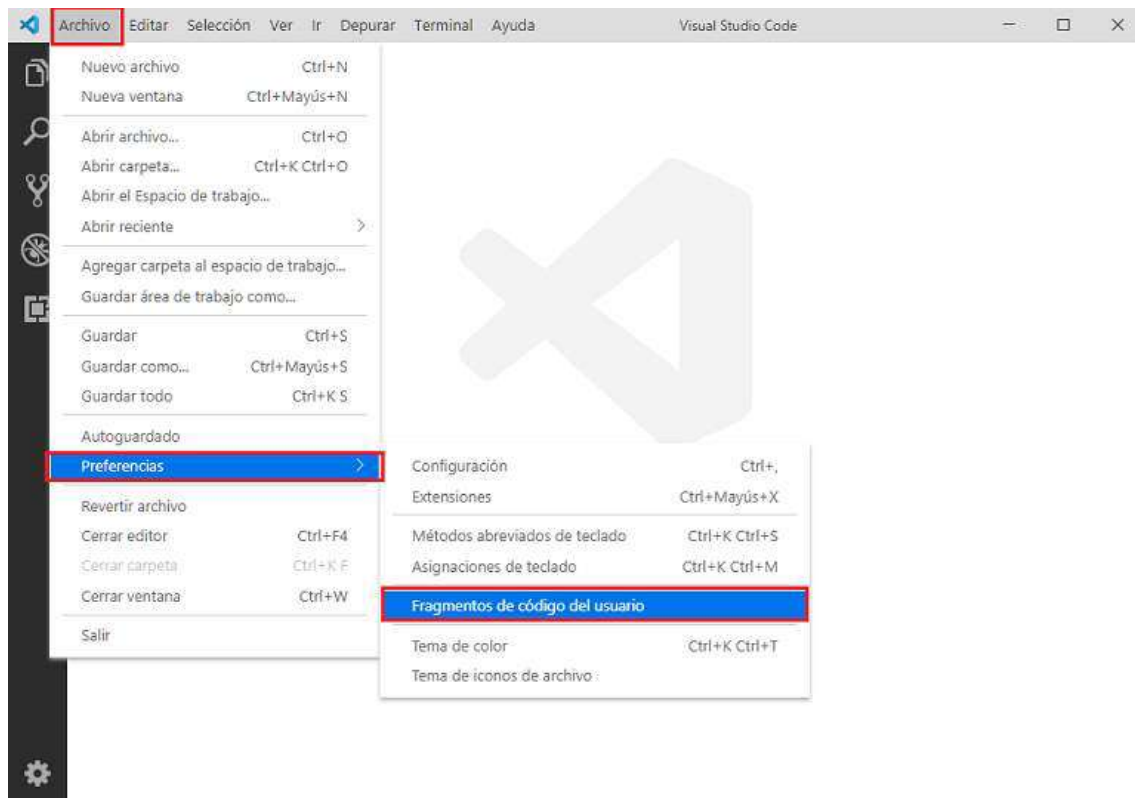
Vimos en el concepto anterior que VS Code trae una serie de fragmentos de código para los lenguajes que apoya por defecto (JavaScript, TypeScript, HTML etc.), también hay extensiones para otros lenguajes como C#, Python etc. Con fragmentos de código.

Ahora veremos que podemos crear nuestros propios fragmentos de código para luego utilizarlos en nuestros proyectos.

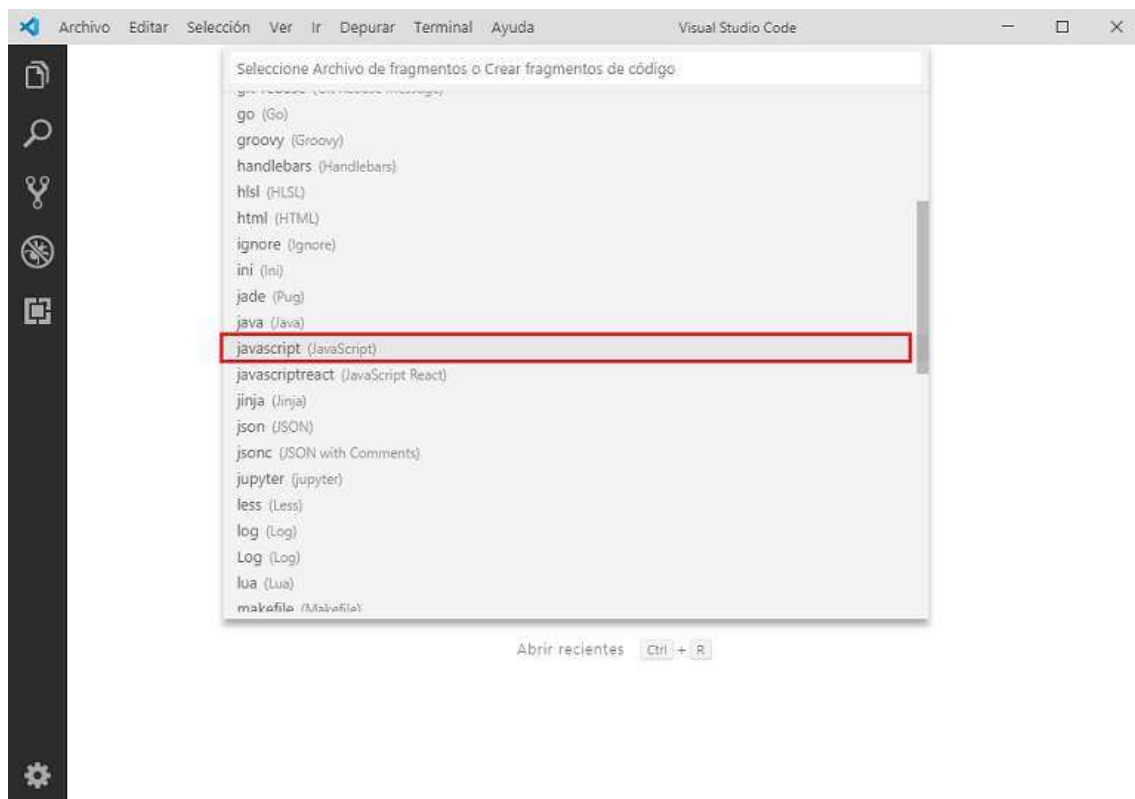
Para crear un fragmento de código lo podemos acceder desde la barra de actividades:



O desde el menú de barra de VS Code:



Una vez elegida esta opción debemos seleccionar el lenguaje al cual le agregaremos nuestro fragmento de código personalizado:



Se abre un archivo llamado 'javascript.json' donde debemos configurar nuestros fragmentos (aparece una explicación y un ejemplo):

```
{ // Place your snippets for javascript here. Each s //  
description. The prefix is what is used to trig // $1,  
$2 for tab stops, $0 for the final cursor p // same ids  
are connected. // Example: // "Print to console": { //  
"prefix": "log", // "body": [ // "console.log('$1');",  
// "$2" // ], // "description": "Log output to console"  
// }}
```

Fragmento de una sola línea

Crearemos un fragmento que inserte un mensaje de 'Hola mundo' por consola cuando el operador ingrese en el editor VS Code la palabra 'hola' :

```
{ // Place your snippets for javascript here. Each
snippet // description. The prefix is what is used to
trigger th // $1, $2 for tab stops, $0 for the final
cursor positio // same ids are connected. // Example: //
"Print to console": { // "prefix": "log", // "body": [
// "console.log('$1');", // "$2" // ], // "description":
"Log output to console" // } "Hola mundo": { "prefix":
"hola", "body": "console.log('Hola mundo');",
"description": "muestra un hola mundo por la consola" }}
```

Luego de grabarlo al archivo 'javascript.json' y comenzar a escribir un programa en 'JavaScript' al ingresar la palabra 'hola' IntelliSense nos muestra el fragmento de código que hemos codificado:



Si confirmamos presionando la tecla 'Enter' se reemplaza el texto que hemos ingresado por el valor del snippet:

```
1 console.log('Hola mundo');
```

Si analizamos el archivo 'javascript.json' hemos definido una propiedad llamada "Hola mundo" que es un objeto que en su interior define tres propiedades:

- prefix: es el nombre que IntelliSense identificará cuando ingresemos en el editor VSCode.
- body: es el fragmento de código propiamente dicho.
- description: es una descripción del objetivo de dicho snippet

Fragmento de varias líneas

Si el fragmento de código a implementar tiene varias líneas la propiedad "body" deberá definir un array con los valores de las distintas líneas. Debemos encerrar entre corchetes y separados por coma cada una de las líneas:

Creemos un fragmento que tenga tres líneas que genere un comentario para recordar revisar el algoritmo de una función:

```
{ // Place your snippets for javascript here. Each snippet // description. The prefix is what is used to trigger th // $1, $2
for tab stops, $0 for the final cursor positio // same ids are connected.
```



```
// Example: // "Print to console": { // "prefix": "log",
// "body": [ // "console.log('$1');", // "$2" // ], //
"description": "Log output to console" // } "Hola
mundo": { "prefix": "hola", "body": "console.log(\"Hola
mundo\");", "description": "muestra un hola mundo por la
consola" }, "Comentario función": { "prefix":
"comentario revisar función", "body":[ "//-----
-----", "//-----Revisar
\"Función\"-----", "//-----
-----" ], "description": "genera un bloque
de comentarios para r } }
```

Lo primero que debemos notar es que hemos agregado el fragmento de código al archivo 'javascript.json' y hemos dejado el fragmento anterior.

En este nuevo fragmento cuando el operador comience a escribir 'comentario revisar función' Intellisense nos propondrá sustituir por el valor definido en 'body'.

La propiedad 'body' define entre corchetes y separados por coma cada una de las líneas del fragmento de código:

```
"body": [ " //------", " //------Revisar \"Función\"-----", " //------" ],
```

Si en un fragmento de código debemos disponer las comillas dobles debemos anteceder el carácter \" cuando escribamos en el editor:

Si confirmamos el fragmento luego se escribe en forma automática:

Ubicación del cursor dentro del fragmento

Podemos generar un fragmento que luego de escribirse dentro del editor nos ubique el cursor en lugares específicos para que el programador los complete y mediante la tecla 'Tab' pase al siguiente punto de configuración.

Desarrollaremos un fragmento de código que nos permita generar una estructura repetitiva for donde podamos definir el valor inicial y final del contador del for (borraremos los otros dos fragmentos de código desarrollados anteriormente para dejar más claro nuestro archivo 'javascript.json' sabiendo que en la realidad luego podremos tener múltiples fragmentos)

```
{ // Place your snippets for javascript here. Each
  snippet // description. The prefix is what is used to
  trigger the // $1, $2 for tab stops, $0 for the final
  cursor position // same ids are connected. // Example: //
  "Print to console": { // "prefix": "log", // "body": [
  // "console.log('$1');", // "$2" // ], // "description":
  "Log output to console" // } "for desde hasta": {
  "prefix": "for desde hasta", "body": [ "for(let x = $1;
  x <= $2; x++) {", " " $0", "}" ], "description": "genera
  un for con incremento unitario }}
```

Para indicar la primera posición donde debe aparecer el cursor utilizamos la sintaxis \$1, luego que el programador presiona la tecla 'Tab' el cursor se posiciona en \$2 y así sucesivamente.

Con \$0 indicamos la posición final donde debe aparecer el cursor.
Para activar este fragmento de código debemos ingresar:

Cuando se presiona la tecla 'Enter' se genera el fragmento y el cursor se posiciona en el lugar donde dispusimos el símbolo \$1:

Luego de ingresar el valor inicial para el contador 'x' presionamos la tecla 'Tab':

Luego de fijar el valor final para el contador presionamos nuevamente la tecla 'Tab' y el cursor se posiciona finalmente en donde definimos el símbolo \$0:

Ubicación del cursor con valores por defecto.

Cuando disponemos el cursor dentro del fragmento podemos definir un valor por defecto para que se muestre, luego es responsabilidad del programador de confirmar dicho valor presionando la tecla 'Tab' o cambiarlo.

Modifiquemos el fragmento de código para que el 'for desde hasta' por defecto sea del número 0 al 10:

```
{ // Place your snippets for javascript here. Each
  snippet // description. The prefix is what is used to
  trigger the // $1, $2 for tab stops, $0 for the final
  cursor position // same ids are connected. // Example: //
  "Print to console": { // "prefix": "log", // "body": [
  // "console.log('$1');", // "$2" // ], // "description":
  "Log output to console" // } "for desde hasta": {
  "prefix": "for desde hasta", "body": [ "for(let x =
  ${1:0}; x <= ${2:10}; x++) {", " $0", "}" ],
  "description": "genera un for con incremento unitario }}
```

Debemos encerrar entre llaves la posición del cursor y luego de dos puntos el valor por defecto a mostrar: \${1:0}

Cuando aceptamos el fragmento de código nos aparecen seleccionados los valores por defecto:

Si presionamos la tecla 'Tab' en forma sucesiva los valores 0 y 10 quedan confirmados.

Aplicación web para generar fragmentos (Snippets) de código

He desarrollado esta pequeña aplicación en Angular para facilitar la creación de fragmentos de código. La misma nos permite no tener que generar manualmente la propiedad 'body':

```
prefix: comentario: fragmento Mostrar ejemplo "" : { "prefix": "", "body": [ ""  
], "description": "" }
```