

# TEMA 1 INTRODUCCIÓN A NODE.JS

---

## 1.5- Depuración de código


# Depurar el código

---

- ☐ Utilizando nuestro propio IDE (Visual Studio Code)
- ☐ Desde terminal
- ☐ Utilizando Google Chrome

# Depurar. Visual Studio Code

---

- ☐ Debes hacer click en el icono  (bug) del panel izquierdo.
- ☐ Hay que configurar el launch.json.
- ☐ Podemos hacer breakpoints
- ☐ Iniciar la depuración con F5 o desde el Menú Depurar > Iniciar depuración o con la flecha verde de play.
- ☐ Podemos ir paso a paso o hasta el siguiente breakpoint visualizando los valores en el panel izquierdo.
- ☐ Finalizar con el botón rojo stop.
- ☐ En la consola de depuración podemos consultar los valores de cualquier variable

# Depurar. Terminal

---

- ❑ Tecleando: `node inspect <nombre js>` se inicial el depurador de `node.js`
- ❑ `list(nlineas)`, donde `nlineas` será un número entero, mostrará en el terminal las `nlineas` líneas de código anteriores y posteriores al punto en el que estamos
- ❑ `n` pasará a ejecutar la siguiente instrucción (ejecución paso a paso)
- ❑ `c` ejecutará el programa hasta su finalización, o hasta encontrar un punto de ruptura (*breakpoint*). Para definir puntos de ruptura en el código, se puede hacer añadiendo la instrucción `debugger`; donde queramos poner el punto de ruptura. Por ejemplo:  

```
var mensaje = 'Hola mundo';  
debugger;  
console.log(mensaje);
```
- ❑ De este modo, el depurador se detendrá en esa línea al ejecutar el comando `c`.
- ❑ `repl` inicia un terminal REPL, como el que hemos visto para Visual Studio Code. El símbolo de prompt cambiará, y podremos comprobar el valor de variables u objetos, o llamar a funciones:
- ❑ Para salir de dicho terminal REPL, pulsaremos `Control+C`, y volveremos al terminal de depuración (`debug>`)
- ❑ Desde el modo de depuración normal (`debug>`), si queremos finalizar la depuración pulsaremos dos veces `Control+C`.

# Depurar. Google Chrome

---

- ❑ Tecleando `node --inspect-brk archivo.js` en el terminal.
- ❑ Accedemos a `chrome://inspect` desde google chrome.
- ❑ Se abrirá una ventana con el depurador. Desde la pestaña *Sources* podemos examinar el código fuente del programa. Y hacer breakpoints.
- ❑ Y en el console drawer podemos ver valores de variables como un REPL.

# Depurar. nodemon

---

- Tecleando:

  - `nodemon inspect archivo.js`

  - `nodemon --inspect-brk archivo.js`

- De esta forma, cada vez que realicemos un cambio en el código se reiniciará la depuración automáticamente, lo que puede resultar bastante cómodo.

# EJERCICIO 5

---

- Ve a la carpeta "PruebasRequire" y utiliza el depurador en cualquiera de las tres formas explicadas (Visual Studio Code, terminal o Google Chrome). Añade un punto de ruptura dentro de cada función del archivo "utilidades.js" (funciones sumar y restar) para que, cuando se les llame, podamos examinar el valor de los parámetros que reciben como entrada.

# EJERCICIO 6

---

- ❑ Crea una carpeta llamada "Ejercicio\_Depu" en la carpeta "ProyectosNode/Ejercicios/", ábrela con Visual Studio Code y crea un archivo llamado "depuracion.js". Dentro, introduce este código y guarda el archivo:

```
1  var m = 1, n = 2;  
2  
3  for(i = 1; i <= 5; i++) {  
4      m = m * i;  
5      n = n + m * n;  
6  }  
7  
8  console.log(n);
```

- ❑ Utiliza el depurador de cualquiera de las formas explicadas (a través de Visual Studio Code, terminal o con Google Chrome) para averiguar el valor de la variable n tras ejecutarse la línea 6 de código (pon un *breakpoint* en la línea 7, por ejemplo). Haz una captura de pantalla mostrando lo que haces para averiguar el valor, y el resultado que te muestra.