

Técnicas de Programación

CFP Programador full-stack

Introducción al Programa

Acerca del Programa

**C
F
P**

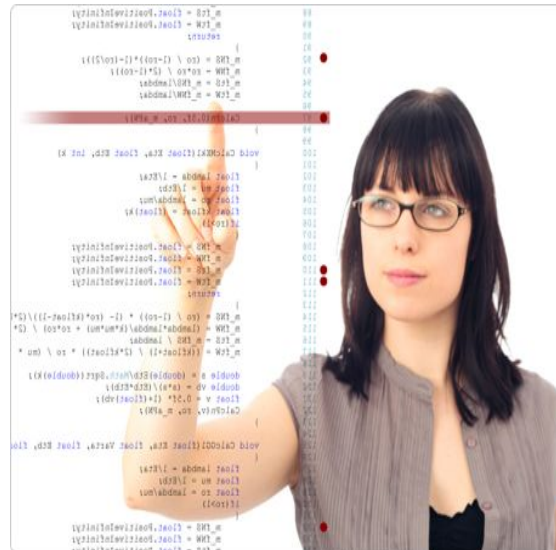
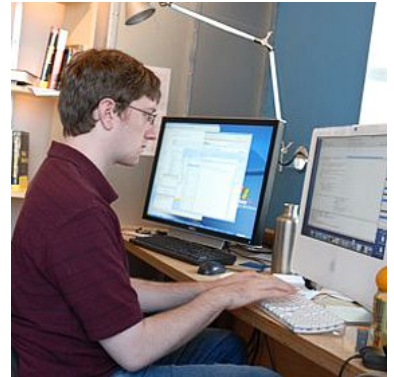


Acerca del Programa

| Módulo | Horas Dictado |
|---------------------------------|---------------|
| Técnicas de Programación | 50 |
| Programación de Interfaces WEB | 40 |
| Programación Avanzada y Objetos | 40 |
| Base de Datos | 40 |
| Programación de servidores | 40 |
| Práctica, exámenes y repasos | 36 |
| Total | 246 |

Sobre Uds.

- Antecedentes
- Tipo de trabajo
- Interés en el curso
- ...



HTML <http:>
 Java php
 RSS <http>
 xml

Curso

Modalidad de las Clases

- Las clases serán teórico-prácticas
 - Por cada tema realizaremos muchos ejercicios
 - Los temas son incrementales (siempre usamos lo aprendido anteriormente)
 - Trabajaremos en la computadora
 - Interactivas (es importante preguntar y participar)



Técnicas de Programación

CFP Programador full-stack

Breve Evolución de las Computadoras

Breve Evolución de las Computadoras

Primera Generación

- Década del 50
- Maquinas grandes y costosas
- Construidas con válvulas de vacío



Breve Evolución de las Computadoras

Segunda Generación

- Se reduce su tamaño y crece el poder de procesamiento
- Empieza a definirse la forma de comunicarse con la computadora (lenguaje)
- Construidas con transistores



Breve Evolución de las Computadoras

Tercera Generación

- Década del 70
- Se manejan por medio de los lenguajes de control de los sistemas operativos
- Construidas con circuitos integrados



Breve Evolución de las Computadoras

Cuarta Generación

- Aparecen los microprocesadores
- Surgen las computadoras personales
- Surgen los procesadores de texto, hojas de cálculo, etc.



Breve Evolución de las Computadoras

Quinta Generación

- Procesamiento en paralelo
- Manejo de lenguaje natural
- Inteligencia artificial



Técnicas de Programación

CFP Programador full-stack

Introducción al Módulo

Técnicas de Programación

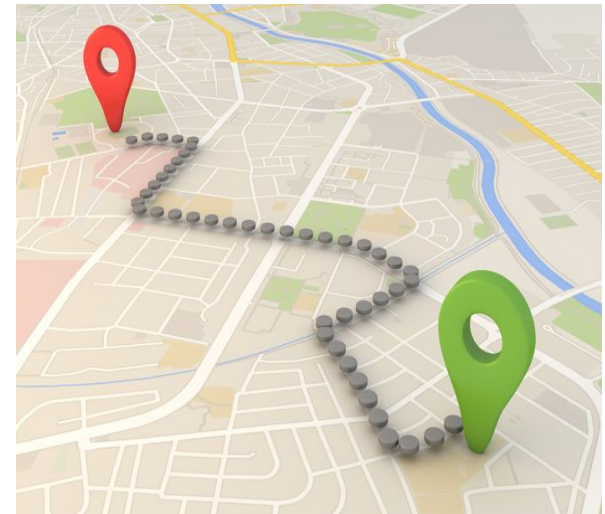
Objetivo

- Interpretar las **especificaciones** de diseño o requisitos de las asignaciones a programar
- Comprendiendo en su contexto inmediato cuál es el **problema a resolver**
- Determinar el **alcance del problema** y convalidar su interpretación a fin de identificar aspectos faltantes.
- **Desarrollar algoritmos** que dan soluciones a los problemas asignados o derivados de los mismos.

Técnicas de Programación

Principales Temas

- Secuencia
- Condicionales
- Ciclos
- Métodos y parámetros
- Arreglos
- Matrices
- Métodos de ordenamiento



El Programador Web

La web fue evolucionando rápidamente a nivel de desarrollo, el webmaster multiusos, paso a dividirse en dos grandes y muy diferenciados roles:

- **Diseño**
 - Encargado de hacer los diseños básicos.
 - En ocasiones también se encargaba de animaciones y transiciones.
- **Programación**
 - Realizaba todas las tareas de desarrollo: JavaScript, PHP, Bases de datos, formularios, hosting, etc...

Las webs de entonces no eran muy complejas, gran parte de la lógica se hacía en el servidor y el verdadero reto era lograr los objetivos con la tecnología de la época.

Al evolucionar la web, su complejidad creció exponencialmente. Por eso, la programación se dividió en dos grandes áreas: Front End y Back End.

El Programador Web

- **Diseñador/Maquetador**
 - Antes con el diseño era suficiente. Luego, el diseñador asume competencias básicas para convertir los diseños en HTML y CSS.
- **Front-End developer**
 - Desarrolladores que asumen las funciones de interacción del lado del cliente (JavaScript) y dejando el servidor. En ocasiones, el diseño quedará fuera de sus competencias.
- **Back-End developer**
 - El desarrollo en el servidor también sufre muchos cambios. Poco a poco, se migrará de proyectos web que basan la mayor parte de su programación en HTML, CSS y JavaScript, desde el servidor a la creación de APIs (especificación y código para que las aplicaciones puedan comunicarse entre ellas).
- **Full Stack Developer**
 - Surge una nueva clase de desarrolladores, que no se encasillan en el back o en el front. Son capaces de adentrarse en ambos mundos y suplir las necesidades de los equipos en estos dos frentes.
 - Cada Full Stack Developer será diferente, cada uno será especialista en unas áreas, y en otras pasará de largo.

Introducción

CFP Programador full-stack

Conceptos Fundamentales

Software

- Es el conjunto de los programas de cómputo, procedimientos, reglas, documentación y datos asociados, que forman parte de las operaciones de un sistema de computación



¿Qué es Programar?

- Programar es un arte y el programador debería ser un artesano.
- Las máquinas y los sistemas son geniales haciendo una única cosa, seguir pasos....
- La responsabilidad de todo programador en relación a las máquinas es ser capaz de guiarlas con las instrucciones más precisas.

¿Qué es Programar?

- Usar vs. Controlar
- Crear Programas
 - Acciones (comandos)
- Solucionar problemas

```
96      .</p>
97      <p><a class="btn btn-lg btn-primary" href="#" role="button">View gallery</a>
98      </div>
99      </div>
100     </div>
101     <a class="left carousel-control" href="#myCarousel" role="button" data-slide="prev">
102       <span class="glyphicon glyphicon-chevron-left" aria-hidden="true"></span>
103       <span class="sr-only">Previous</span>
104     </a>
105     <a class="right carousel-control" href="#myCarousel" role="button" data-slide="next">
106       <span class="glyphicon glyphicon-chevron-right" aria-hidden="true"></span>
107       <span class="sr-only">Next</span>
108     </a>
109   </div><!-- /.carousel -->
110
111   <!--Featured Content Section-->
112   <div class="container">
113     <div class="row">
114       <div class="col-md-4"></div>
115       <div class="col-md-4">FEATURED CONTENT</div>
116     </div>
117   </div>
```

Lenguajes de Programación

- Lenguaje especial para desarrollar programas
- Hay muchos lenguajes según lo que queramos hacer
 - Desarrollo de aplicaciones y juegos: C, C++, Java
 - Bases de datos: MySQL, SQL
 - Drivers: Assembler, C
 - Web: HTML, JavaScript, Python, PHP



Lenguajes de Programación

- Los programas están formados por secuencias de **instrucciones**
- Las instrucciones están escritas para que la computadora realice una **tarea específica**
- La secuencia de instrucciones son escritas por un programador usando un lenguaje de programación



Lenguaje de Programación

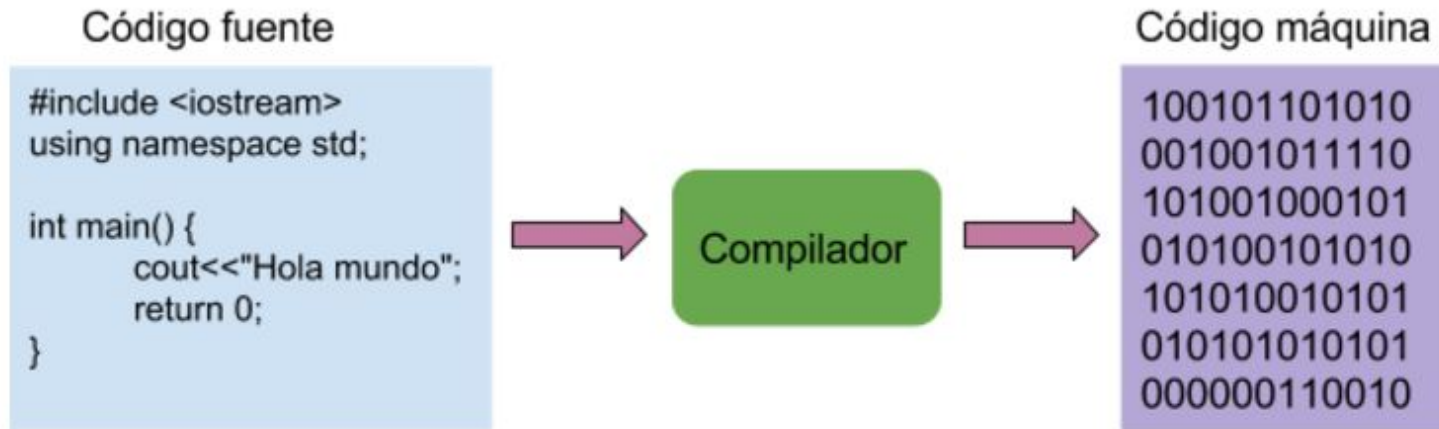
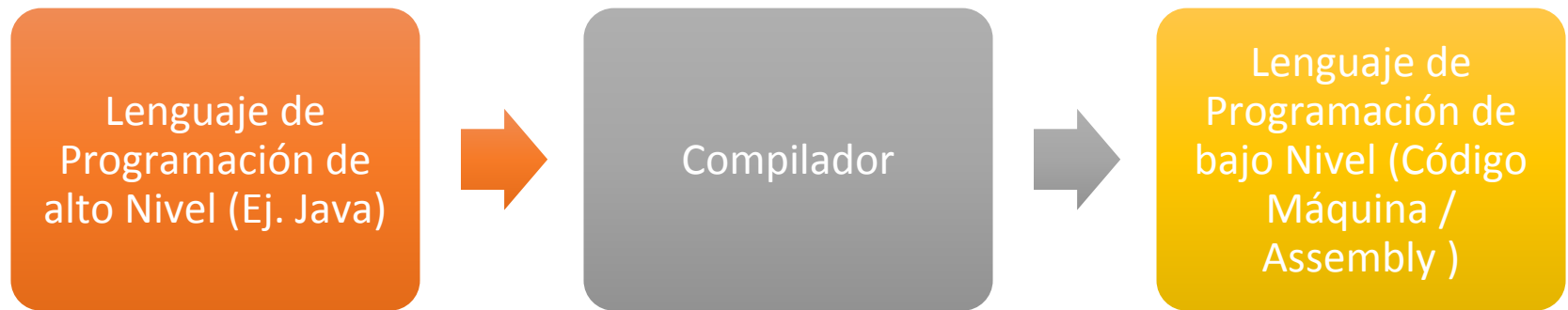
Lenguaje de Máquina

- El lenguaje máquina está compuesto de ceros y unos lo que hace que programar en lenguaje máquina sea un proceso tedioso y sujeto a errores.
- El lenguaje Assembly (ensamblador) hace de traductor entre ese **lenguaje máquina** y uno que es más natural para el humano (**lenguaje de alto nivel**)

| Assembly Language | Machine Code |
|----------------------|---------------------------|
| add \$t1, \$t2, \$t3 | 04CB: 0000 0100 1100 1011 |
| addi \$t2, \$t3, 60 | 16BC: 0001 0110 1011 1100 |
| and \$t3, \$t1, \$t2 | 0299: 0000 0010 1001 1001 |
| andi \$t3, \$t1, 5 | 22C5: 0010 0010 1100 0101 |
| beq \$t1, \$t2, 4 | 3444: 0011 0100 0100 0100 |
| bne \$t1, \$t2, 4 | 4444: 0100 0100 0100 0100 |
| j 0x50 | F032: 1111 0000 0011 0010 |
| lw \$t1, 16(\$s1) | 5A50: 0101 1010 0101 0000 |
| nop | 0005: 0000 0000 0000 0101 |
| nor \$t3, \$t1, \$t2 | 029E: 0000 0010 1001 1110 |
| or \$t3, \$t1, \$t2 | 029A: 0000 0010 1001 1010 |
| ori \$t3, \$t1, 10 | 62CA: 0110 0010 1100 1010 |
| ssl \$t2, \$t1, 2 | 0455: 0000 0100 0101 0101 |
| srl \$t2, \$t1, 1 | 0457: 0000 0100 0101 0111 |
| sw \$t1, 16(\$t0) | 7050: 0111 0000 0101 0000 |
| sub \$t2, \$t1, \$t0 | 0214: 0000 0010 0001 0100 |

Lenguaje de Programación

Compilador



Sistema Operativo

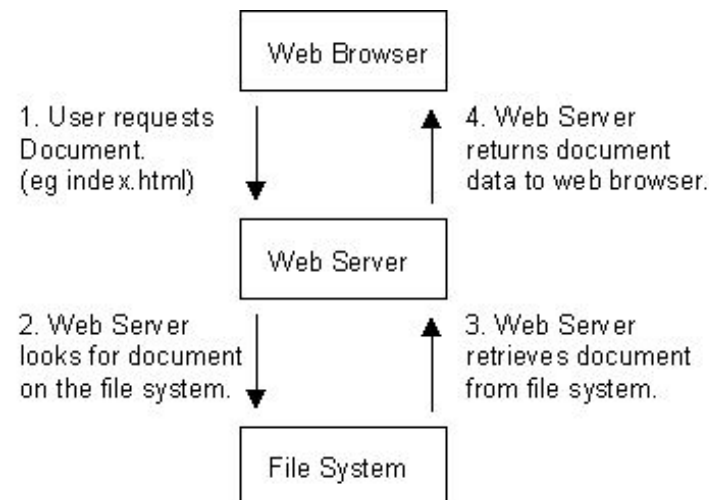
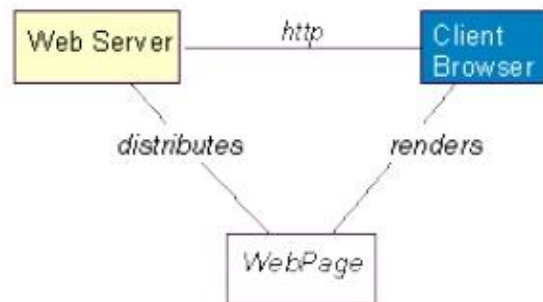
- Gestiona los recursos de **hardware**
- Provee servicios a los **programas de aplicación**



Lenguaje de Programación

Arquitectura WEB

- La arquitectura de un sitio web tiene 3 componentes principales: un servidor Web, una conexión de red y uno o más clientes (browsers)
- El servidor Web distribuye páginas de información formateada a los clientes que las solicitan. Los requerimientos son hechos a través de una conexión de red, y para ello se usa el protocolo HTTP



Desarrollo de un Programa



Desarrollo de un Programa

- **Definir el problema**
 - Determinar la información inicial para la elaboración del mismo
- **Análisis del problema**
 - Datos de entrada, de salida, métodos y fórmulas
- **Diseño del algoritmo**
 - Usar las herramientas de representación de algoritmos
- **Codificación**
 - Escribir la solución del problema, en instrucciones detalladas, en un lenguaje reconocible por la computadora
- **Prueba y depuración**
 - Se toman escenarios posibles, validos o inválidos y se corre la secuencia del algoritmo para ver si cumple con los resultados esperados

CFP

Programador

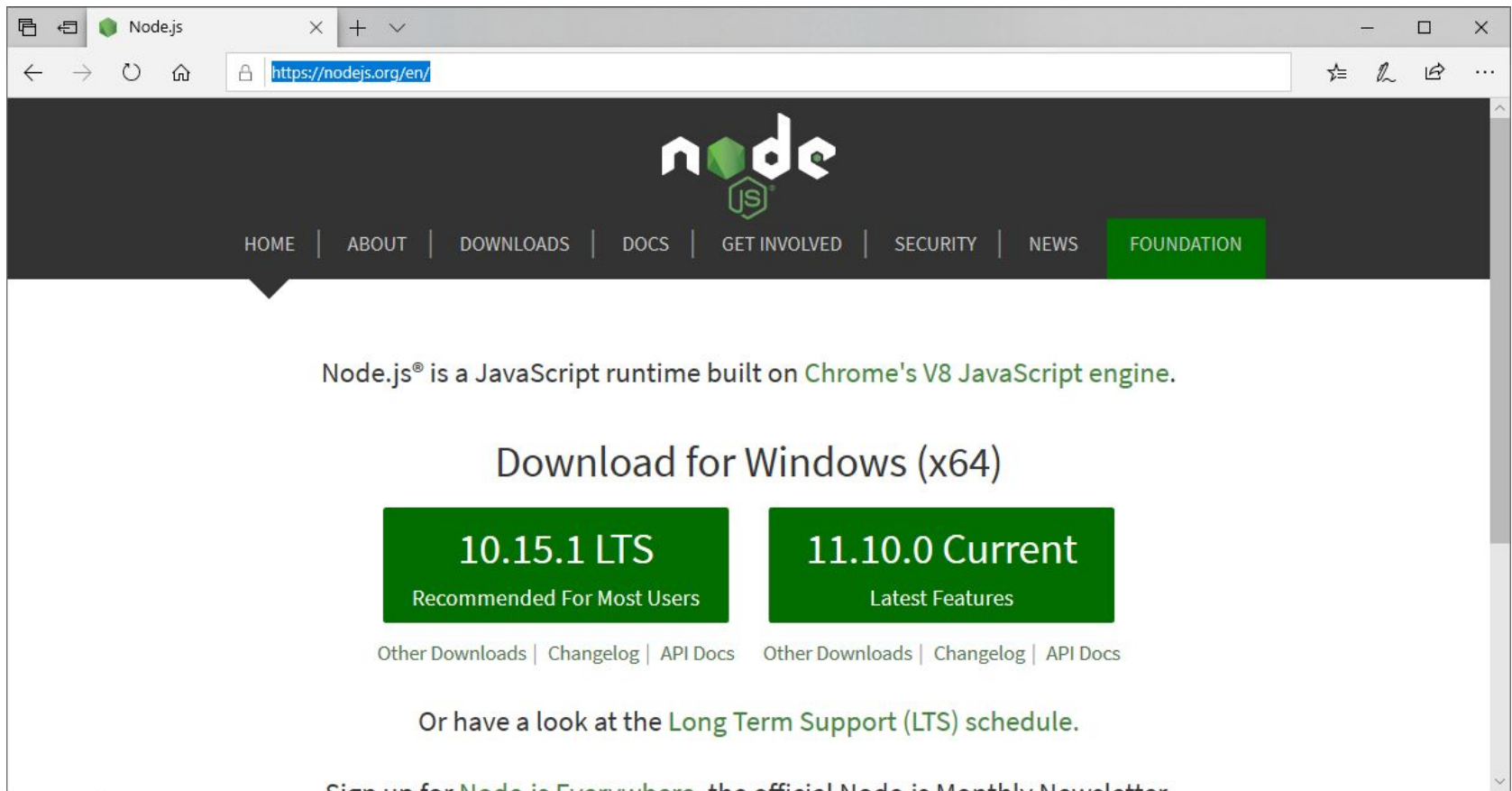
full-stack

Herramientas

Javascript

- JavaScript es un lenguaje de programación que nació para crear páginas web dinámicas.
- Una página web dinámica es aquella que incorpora efectos como texto que aparece y desaparece, animaciones, acciones que se activan al pulsar botones y ventanas con mensajes de aviso al usuario.
- JavaScript es un lenguaje de programación interpretado, por lo que no es necesario compilar los programas para ejecutarlos. En otras palabras, los programas escritos con JavaScript se pueden probar directamente en cualquier navegador sin necesidad de procesos intermedios.
- A pesar de su nombre, JavaScript no guarda ninguna relación directa con el lenguaje de programación Java.

Instalación del intérprete/compilador



Instalamos Node.JS: www.nodejs.org

NodeJS

Node.js es una plataforma para crear aplicaciones utilizando JavaScript.

- **Node.js** es para ejecutar scripts JavaScript
- **npm** es el Administrador de paquetes para los módulos Node.js.

Abrir una consola (Command Prompt)

En el Command Prompt ejecutar:

- **node --help**

Para chequear la instalación de NodeJS

```

Microsoft Windows [Version 10.0.17763.134]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\guillermo.islas>node --help
Usage: node [options] [ -e script | script.js | - ] [arguments]
       node inspect script.js [arguments]

Options:
  -                  script read from stdin (default if no file name is
  --                provided, interactive mode if a tty)
  --abort-on-uncaught-exception
                    indicate the end of node options
                    aborting instead of exiting causes a core file to
                    be generated for analysis
  -c, --check       syntax check script without executing
  --completion-bash  print source-able bash completion script
  --diagnostic-report-directory=...
                    define custom report pathname. (default: current
                    working directory of Node.js process)
  --diagnostic-report-filename=...
                    define custom report file name. (default:
                    YYYYMMDD.HHHMSS.PID.SEQUENCE#.txt)
  --diagnostic-report-on-fatalerror
                    generate diagnostic report on fatal (internal)
                    errors
  --diagnostic-report-on-signal
                    generate diagnostic report upon receiving signals
  --diagnostic-report-signal=...
                    causes diagnostic report to be produced on provided
                    signal, unsupported in Windows. (default: SIGUSR2)
  --diagnostic-report-uncaught-exception
                    generate diagnostic report on uncaught exceptions
  --diagnostic-report-verbose
                    verbose option for report generation(true|false).
                    (default: false)
  -e, --eval=...    evaluate script
  --experimental-modules
                    experimental ES Modules support and caching modules
  --experimental-policy=...
                    use the specified file as a security policy
  --experimental-repl-await
                    experimental await keyword support in REPL
  --experimental-report
                    enable report generation
  --experimental-vm-modules
                    experimental ES Module support in vm module
  -h, --help         print node command line options (currently set)
  --http-parser=...  Select which HTTP parser to use; either 'legacy' or
                    'llhttp' (default: legacy).
  --icu-data-dir=...
                    set ICU data load path to dir (overrides
                    NODE_ICU_DATA)
  --inspect[=[host:]port]
                    activate inspector on host:port (default:
                    127.0.0.1:9229)
  --inspect-brk[=[host:]port]
                    activate inspector on host:port and break at start
                    of user script
  --debug-port, --inspect-port=[host:]port
                    set host:port for inspector
  -i, --interactive  always enter the REPL even if stdin does not appear
                    to be a terminal
  --loader=...       (With --experimental-modules) use the specified
                    file as a custom loader
  --max-http-header-size=...
                    set the maximum size of HTTP headers (default: 8KB)
  --no-deprecation   silence deprecation warnings
  --no-force-async-hooks-checks
                    disable checks for async_hooks
  --no-warnings       silence all process warnings
  --openssl-config=...
                    load OpenSSL configuration from the specified file
                    (overrides OPENSSL_CONF)
  --pending-deprecation
                    emit pending deprecation warnings
  --preserve-symlinks
                    preserve symbolic links when resolving
  --preserve-symlinks-main
                    preserve symbolic links when resolving the main
                    module
  -p, --print [...]  evaluate script and print result
  --prof-process      process V8 profiler output generated using --prof
  --redirect-warnings=...
                    write warnings to file instead of stderr
  -r, --require=...  module to preload (option can be repeated)
  --throw-deprecation
                    throw an exception on deprecations
  --title=...        the process title to use on startup
  --tls-cipher-list=...
                    use an alternative default TLS cipher list
  
```

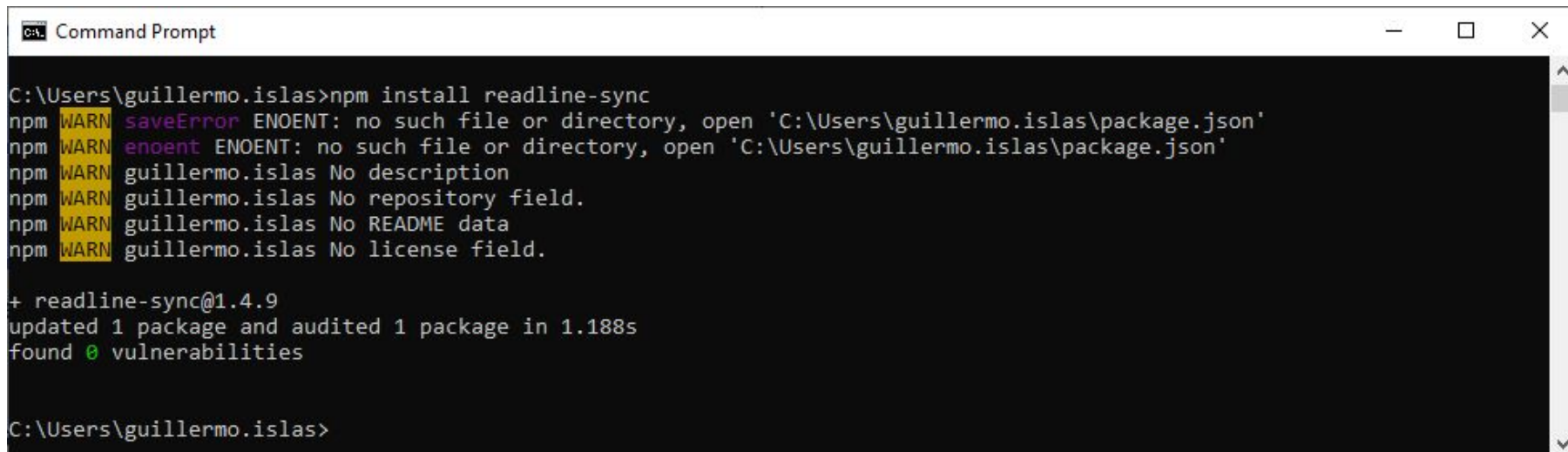

NodeJS - Instalación de paquete “*readline-sync*” usando el comando “*npm*”

En el Command Prompt ejecutar:

- **npm install** readline-sync

Este paquete “readline-sync” permite ejecutar de forma interactiva una conversación con el usuario a través de una consola

De esta manera se puede ingresar datos a nuestros scripts

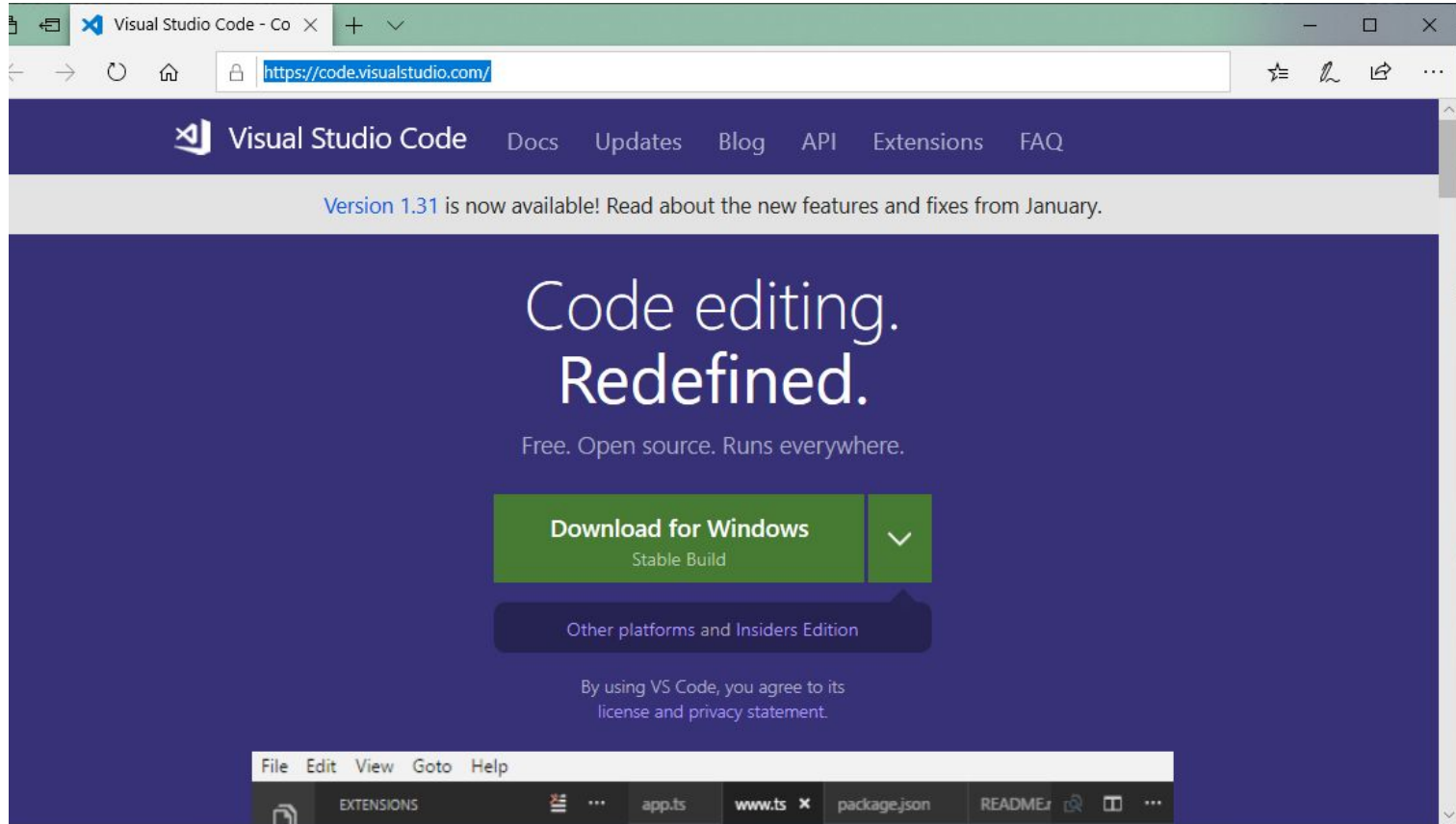


```
C:\Users\guillermo.islas>npm install readline-sync
npm WARN saveError ENOENT: no such file or directory, open 'C:\Users\guillermo.islas\package.json'
npm WARN enoent ENOENT: no such file or directory, open 'C:\Users\guillermo.islas\package.json'
npm WARN guillermo.islas No description
npm WARN guillermo.islas No repository field.
npm WARN guillermo.islas No README data
npm WARN guillermo.islas No license field.

+ readline-sync@1.4.9
updated 1 package and audited 1 package in 1.188s
found 0 vulnerabilities

C:\Users\guillermo.islas>
```

Instalación de editor de textos



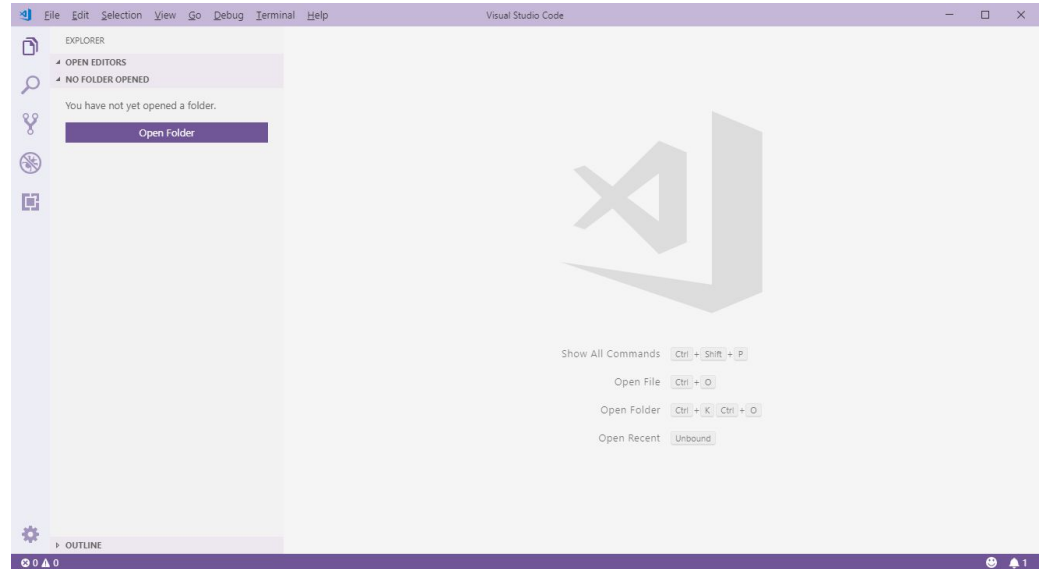
Instalamos Node.JS: <https://code.visualstudio.com/>

Editor de textos Visual Studio Code

Para hacer código se usan editores de textos.

Un Editor de texto especial para código se llama IDE (Integrated Development Enviroment - Ambiente de Desarrollo Integrado)

VSCoode es compatible con los lenguajes JavaScript y TypeScript



Ejecutando un script en VSC

Hola Mundo en JS

- **console.log()** muestra un mensaje en la consola web (o del intérprete JavaScript)

```
console.log("Hola Mundo");
```

- Grabar el archivo con nombre y extensión "HolaMundo.js"
- Abrir la solapa Terminal
- Ejecutar el comando:

```
node c:\temp\HolaMundo.js
```

- Este comando permite ejecutar nuestros scripts desde el path donde los almacenamos

