Tutorial básico de Zig

Tutorial básico del lenguaje de programación Zig

Introducción

Zig es un lenguaje de programación de propósito general así como una cadena de herramientas para crear y mantener programas robustos, optimizados y reutilizables.

Zig es un lenguaje de programación de sistemas compilado, imperativo y de tipos estáticos. Pretende ser un reemplazo para el lenguaje C, ofreciendo mayor simplicidad y sencillez pero aportando mayor funcionalidad.

Para poder seguir este tutorial es recomendable que tengas conocimientos previos de programación así como conocer algunos conceptos de programación de bajo nivel.

Instalación de Zig

Como vimos en la introducción, Zig es un lenguaje pero también una cadena de herramientas. Para empezar a trabajar en Zig necesitamos instalar la cadena de herramientas de Zig. A continuación veremos cómo podemos instalar estas herramientas en nuestro ordenador.

Instalación de Zig en Windows

La manera más sencilla de instalar Zig en Windows es usando alguno de los gestores de paquetes disponibles para Windows, como por ejemplo chocolatey, winget o scoop. A continuación te mostramos los comandos necesarios:

```
choco install zig
```

```
winget install zig.zig
```

```
scoop install zig
```

Instalación manual de Zig en Windows

La instalación manual de Zig en Windows se hace a partir de la distribución binaria. Descarga desde la web oficial de Zig el archivo comprimido que sea adecuado para tu plataforma, por ejemplo `x86_64`. Al momento de escribir este tutorial la versión estable de Zig es 0.13.0.

Descomprime el archivo en cualquier directorio de tu equipo, por ejemplo: `C:\Zig`. Para finalizar, añade el directorio donde descomprimiste Zig a la variable Path.

Instalación de Zig en macOS

Para instalar Zig en macOS el método más recomendable es usando un gestor de paquetes como brew. Veamos a continuación los comandos necesarios para instalar Zig en macOS usando brew:

```
brew install zig
```

Instalación de Zig en Linux

El método más sencillo de instalación de Zig en las distribuciones Linux es a través de los gestores de paquetes. Veamos un ejemplo usando apt:

```
sudo apt install zig
```

Instalación manual de Zig en Linux

La instalación manual de Zig en Linux se hace a partir de la distribución binaria. Descarga desde la web oficial de Zig el archivo comprimido que sea adecuado para tu plataforma, por ejemplo `x86_64`. Al momento de escribir este tutorial la versión estable de Zig es 0.13.0.

Descomprime el archivo en cualquier directorio de tu equipo, por ejemplo:

```
tar xf zig-linux-x86_64-0.13.0.tar.xz
```

Para finalizar, añade el directorio donde descomprimiste Zig a la variable Path:

```
echo 'export PATH="$HOME/zig-linux-x86_64-0.13.0:$PATH"' >> ~/.bashrc
```

Verificar la instalación de Zig

Una vez instalados los componentes y registrado el directorio en la variable Path ya podemos verificar que Zig esté correctamente instalado.

> Ten en cuenta que si has registrado el directorio de Zig en la variable Path usando la consola, es recomendable cerrar y volver a abrir la consola.

Abre una ventana de la consola e introduce el siguiente comando:

```
zig version
```

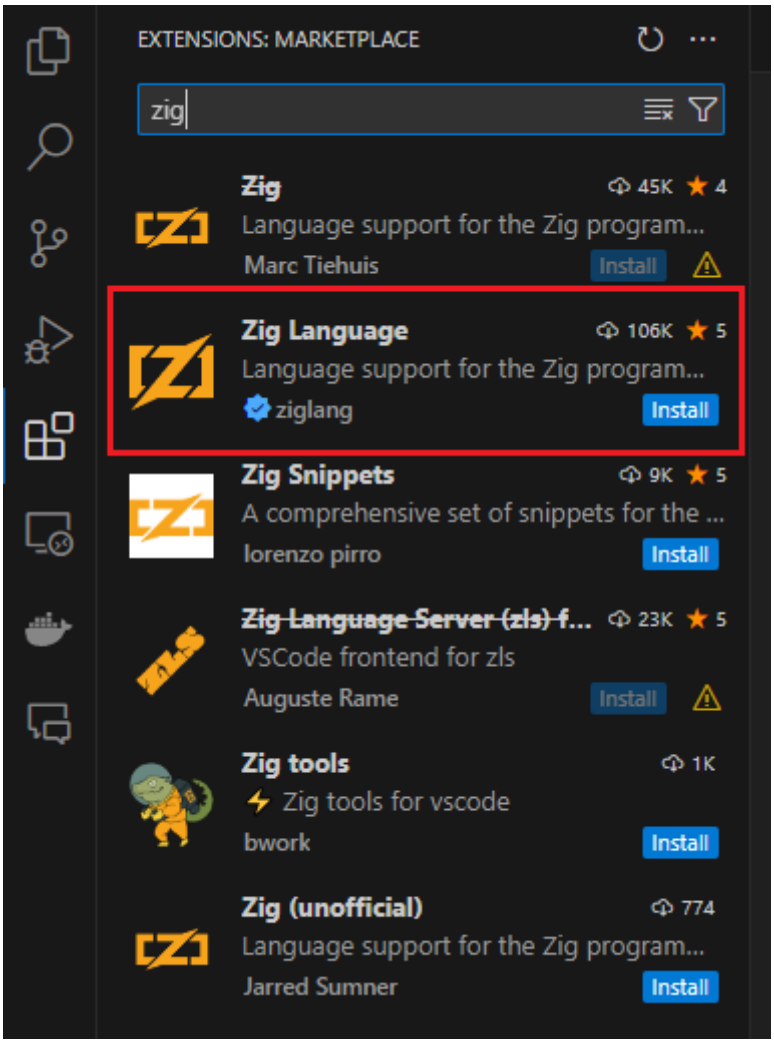
Si la instalación es correcta debería devolver la versión de Zig instalada, por ejemplo:

```
0.13.0
```

Configuración de Visual Studio Code para Zig

Visual Studio Code es un editor código gratuito muy versátil y funcional. VS Code no trae por defecto ningún tipo de soporte para el lenguaje Zig, pero esto no es ningún problema ya que existe un paquete de extensión.

Ve a la galería de extensiones de Visual Studio Code y en el campo de búsqueda escribe `zig`. Instala la extensión oficial, enmarcada en rojo:



Hola Mundo

Una vez instalado Zig es la hora de probar la instalación, para ello vamos a crear un nuevo archivo al que llamaremos `main.zig`. En él, incluye el siguiente código.

```
const std = @import("std");

pub fn main() void {
  std.debug.print("Hola, {s}!\n", .{"mundo"});
}
```

Abre una ventana de consola y navega hasta la carpeta donde has creado el archivo `main.zig` y ejecuta el siguiente comando:

```
zig run main.zig
```

La primera vez que ejecutemos el comando, el código se compilará y luego se ejecutará, produciendo como resultado:

```
Hola mundo!
```

Ya tenemos Zig funcionando en nuestro ordenador y ya hemos visto como es el programa más básico. En los próximos capítulos vamos a ir viendo los diferentes aspectos del lenguaje de programación Zig y su biblioteca de clases.

Comentarios

El lenguaje de programación Zig soporta tres tipos de comentarios: comentarios normales, comentarios de documentación y comentarios de documentación de módulo. Veamos a continuación los tres tipos.

Comentarios normales

Los comentarios normales son aquellos que se usan para anotar el código e introducir explicaciones sobre su funcionamiento o cualquier otra información relativa al mismo. Este tipo de comentarios, al igual que en otros lenguajes, son ignorados y son sólo visibles en el código fuente.

> Zig no dispone de ninguna estructura para definir comentarios de varias líneas.

Los comentarios en Zig se definen usando dos barras oblicuas `//`. Los comentarios pueden aparecer solos en una línea o después de una expresión. Varias líneas seguidas pueden estar comentadas. Veamos algunos ejemplos de comentarios en Zig.

```
const std = @import("std");

// Este es un comentario en una sola línea.
// Y este también.
// De esta forma puedo poner varias líneas,
// una a continuación de la otra.
pub fn main() void {
    // Este comentario también es de una sola línea
    std.debug.print("Hola {s}!\n", .{"mundo"}); // Este comentario está después de una
expresión
    // std.debug.print("Hola {s}!\n", .{"mundo"}); Este código no se ejecutará porque está
comentado
}
```

Comentarios de documentación

A diferencia de los comentarios de código, que son de uso interno del equipo de desarrollo, los comentarios de documentación son extraídos para crear la documentación del programa. Esta documentación puede ser distribuida a los usuarios del programa para que tengan una referencia

para su uso. Los comentarios de documentación tienen la doble ventaja de permitir por un lado la documentación del código y, al mismo tiempo se define el contenido de la documentación formal.

> Atención: al momento de escribir este tutorial, versión 0.13 de Zig, el módulo de generación de documentación está en fase experimental.

Para definir los comentarios de documentación Zig dispone de dos tipos especializados: comentarios de documentación de módulo y comentarios de documentación general. A continuación los veremos con más detalles.

Comentarios de documentación de módulos

Los comentarios de documentación de módulos se definen con `//!` y se usa para documentar un módulo. Este tipo de comentarios solo pueden aparecer al inicio del módulo, siempre antes de cualquier expresión. Si este tipo de comentario se usa en otra parte del código, se producirá un error de compilación. Si bien, el compilador soporta que los comentarios de módulos se usen dentro de un contenedor, estos son ignorados, con lo cual no tiene ninguna utilidad práctica.

Veamos un ejemplo de comentarios de módulo:

```
//! Este es un comentario de módulo.
//! Se pueden definir tantas líneas como sea necesario.
//! Siempre deben aparecer al inicio del módulo, antes de cualquier
//! otra expresión.
const std = @import("std");

//! Este es un comentario que no está permitido aquí
//! ya que está definido después de una expresión.
//! Este producirá un error de compilación.
pub fn main() void {
    std.debug.print("Hola {s}!\n", .{"mundo"});
}
```

Comentarios de documentación general

Este tipo de comentarios son usados para documentar cualquier estructura del código que se encuentre a continuación. Para definir un comentario de documentación general se usan tres barras oblicuas `///`, pero exactamente tres, cuatro o más barras oblicuas causará que el comentario sea tratado como un comentario normal. Veamos un ejemplo.

```
//! Este es un comentario de módulo.
//! Se pueden definir tantas líneas como sea necesario.
```

```
//! Siempre deben aparecer al inicio del módulo, antes de cualquier
//! otra expresión.
const std = @import("std");

/// Este es un comentario de documentación general.
/// Aquí definimos la información que queremos
/// documentar sobre nuestra función main()
pub fn main() void {
    std.debug.print("Hola {s}!\n", .{"mundo"});
}
```

Generación de la documentación

Los comentarios que han sido definidos como documentación, son usados para crear una página web estática. Esta web estática permite navegar por la documentación y visualizarla.

Para generar la documentación debemos añadir la opción `-femit-docs` a los comandos `zig build-{exe, lib, obj}`, `zig run` o `zig test`.

Veamos un ejemplo práctico. Crea un nuevo archivo y llámalo `docs_comments.zig` y escribe el código del ejemplo anterior:

```
//! Este es un comentario de módulo.
//! Se pueden definir tantas líneas como sea necesario.
//! Siempre deben aparecer al inicio del módulo, antes de cualquier
//! otra expresión.
const std = @import("std");

/// Este es un comentario de documentación general.
/// Aquí definimos la información que queremos
/// documentar sobre nuestra función main()
pub fn main() void {
    std.debug.print("Hola {s}!\n", .{"mundo"});
}
```

Guarda el archivo y ejecuta el siguiente comando desde la carpeta donde se encuentra:

```
zig test -femit-docs docs_comments.zig
```

Zig compilará el código, generará la documentación y tratará de ejecutar los tests. No hemos definido ningún test, pero esto no es un problema y no provocará fallo alguno. El módulo de generación de documentación ha creado una carpeta llamada `docs` que contiene los archivos del

sitio web estático. Recuerda que este módulo es experimental y que su funcionalidad es limitada y está sujeto a cambios.