

Trabajo con mapas hash:

Otro tipo de colección común en Rust es el **mapa hash**. El tipo `HashMap<K, V>` almacena los datos asignando cada clave `K` con su valor `V`. Mientras que a los datos de un vector se accede mediante un índice entero, a los datos de un **mapa hash** se accede mediante una clave.

El tipo de **mapa hash** se usa en muchos lenguajes de programación para elementos de datos como objetos, tablas hash y diccionarios.

Al igual que los vectores, los **mapas hash** se pueden aumentar. Los datos se almacenan en el **heap** y el acceso a los elementos de **mapas hash** se comprueba en tiempo de ejecución.


Definición de un mapa hash:

En el ejemplo siguiente se define un **mapa hash** para realizar un seguimiento de las reseñas de libros. Las claves del **mapa hash** son los nombres de los libros y los valores son las reseñas realizadas por el lector.

```
Rust Copiar  
  
use std::collections::HashMap;  
let mut reviews: HashMap<String, String> = HashMap::new();  
  
reviews.insert(String::from("Ancient Roman History"), String::from("Very accurate."));  
reviews.insert(String::from("Cooking with Rhubarb"), String::from("Sweet recipes."));  
reviews.insert(String::from("Programming in Rust"), String::from("Great examples."));
```

Vamos a examinar este código de forma más detallada. En la primera línea, vemos un tipo de sintaxis nuevo:

Rust

 Copiar

```
use std::collections::HashMap;
```

El comando **use** trae la definición **HashMap** de la parte **collections** de la biblioteca estándar de Rust en el ámbito de nuestro programa. Esta sintaxis es parecida a lo que otros lenguajes de programación llaman una **importación**.

Creamos un **mapa hash** vacío con el método **HashMap::new**. Declaramos la variable **reviews** como mutable para que podamos agregar, o quitar, claves y valores, según sea necesario. En nuestro ejemplo, tanto las claves de mapa hash como los valores usan el tipo **String**.

Rust


 Copiar

```
let mut reviews: HashMap<String, String> = HashMap::new();
```

Incorporación de un par clave-valor:

Agregamos elementos al mapa hash mediante el método **insert(<key>, <value>)**. En el código, la sintaxis es **<hash_map_name>.insert()**:

Rust


 Copiar

```
reviews.insert(String::from("Ancient Roman History"), String::from("Very accurate."));
```

Obtención de un valor de clave:

Después de agregar datos a nuestro mapa hash, podemos obtener un valor específico para una clave con el método **get(<key>)**.

Rust

 Copiar

```
// Look for a specific review
let book: &str = "Programming in Rust";
println!("\nReview for '{}': {:?}", book, reviews.get(book));
```

La salida es la siguiente:

Rust

 Copiar

```
Review for 'Programming in Rust': Some("Great examples.")
```

Nota

Observe que en la salida se muestra la reseña del libro como "Some("Great examples.")" en lugar de solo "Great examples". Dado que el método `get` devuelve un tipo `Option<&Value>`, Rust encapsula el resultado de la llamada de método con la notación "Some()".

Eliminación de un par clave-valor:

Se pueden quitar entradas de un mapa hash mediante el método `remove()`. Si usamos el método `get` para una clave de mapa hash no válida, el método `get` devuelve "None".

Rust


 Copiar

```
// Remove book review
let obsolete: &str = "Ancient Roman History";
println!("\n '{}' removed.", obsolete);
reviews.remove(obsolete);

// Confirm book review removed
println!("\nReview for '{}': {:?}", obsolete, reviews.get(obsolete));
```

La salida es la siguiente:

Rust

 Copiar

```
'Ancient Roman History' removed.  
Review for 'Ancient Roman History': None
```