

## Dictionary(TKey, TValue)

Una clase de Diccionario es una estructura de datos que representa una colección de claves y valores de pares de datos. La clave es idéntica en un par clave-valor y puede tener como máximo un valor en el diccionario, pero un valor puede asociarse con muchas claves diferentes. Esta clase se define en el espacio de nombres **System.Collections.Generic**.

### Espacio de Nombres

```
using System.Collections.Generic
```

### Sintaxis

```
Dictionary<TKey, TValue>
```

### Parámetros

- **TKey** – El tipo de la clave en el diccionario.
- **TValue** – El tipo de valor en el diccionario.

### Ejemplo

```
Dictionary<string, string>
```

```
Dictionary<string, int>
```

La colección **Dictionary<TKey, TValue>** es igual que un diccionario de cualquier otro idioma, es decir, un diccionario es una colección de palabras y sus definiciones, a menudo enumeradas alfabéticamente, de la misma manera, el Diccionario en C # es una colección de claves y valores, donde la clave es como la palabra y el valor es como la definición.

**TKey** denota el tipo de clave y **TValue** es el tipo de los valores.

Un objeto de diccionario puede asignarse a una variable de **IDictionary<Tkey, TValue>** o a la clase **Dictionary<TKey, Tvalue>**.

## Ejemplo de Inicialización

```
IDictionary<int, string> diccionario = new Dictionary<int, string>();  
  
//o  
  
Dictionary<int, string> diccionario = new Dictionary<int, string>();
```

En este ejemplo, se han especificado tipos de clave y valor al declarar un objeto de diccionario. El **int** es un tipo de clave y el **string** es un tipo de valor que se almacenará en un objeto de diccionario llamado *diccionario*. Puede usar cualquier tipo de datos de C# válido para claves y valores.

Se recomienda usar la interface en lugar de la clase, es decir, se recomienda usar la variable de tipo **IDictionary<TKey, TValue>** para inicializar un objeto de diccionario.

El diccionario no puede incluir claves duplicadas o nulas, donde los valores se pueden duplicar o establecer como nulos. Las claves deben ser únicas, de lo contrario lanzará una excepción de tiempo de ejecución.

Propiedad	Descripción
<b>Count</b>	Obtiene el número total de elementos existentes en el Dictionary <TKey, TValue>.
<b>IsReadOnly</b>	Devuelve un valor booleano que indica si el Dictionary <TKey, TValue> es de solo lectura.
<b>Item</b>	Obtiene o establece el elemento con la clave especificada en el Dictionary <TKey, TValue>.
<b>Keys</b>	Devuelve la colección de claves del Dictionary <TKey, TValue>.
<b>Values</b>	Devuelve la colección de valores en el Dictionary <TKey, TValue>.

Método	Descripción
<b>Add</b>	Agrega un elemento a la colección de diccionarios.
<b>Add</b>	Agregue pares clave-valor en la colección Dictionary <TKey, TValue>.
<b>Remove</b>	Elimina la primera aparición de un elemento especificado del Dictionary <TKey, TValue>.
<b>Remove</b>	Elimina el elemento con la clave especificada.
<b>ContainsKey</b>	Comprueba si la clave especificada existe en el Dictionary <TKey, TValue>.
<b>ContainsValue</b>	Comprueba si la clave especificada existe en el Dictionary <TKey, TValue>.
<b>Clear</b>	Elimina todos los elementos del Dictionary <TKey, TValue>.
<b>TryGetValue</b>	Devuelve verdadero y asigna el valor con la clave especificada, si la clave no existe, devuelve falso.

## Añadir elementos

Para ello se usa el método **Add()** para agregar el par clave-valor en el diccionario.

```
IDictionary<int, string> diccionario = new Dictionary<int, string>();
diccionario.Add(1, "Uno");
diccionario.Add(2, "Dos");
diccionario.Add(3, "Tres");
```

La instancia de tipo **IDictionary** tiene una sobrecarga más para el método **Add()**. Acepta una estructura **KeyValuePair<TKey, TValue>** como parámetro.

Ejemplo

```
IDictionary<int, string> diccionario = new Dictionary<int, string>();
diccionario.Add(new KeyValuePair<int, string>(1, "Uno"));
diccionario.Add(new KeyValuePair<int, string>(2, "Dos"));

//también es válido

diccionario.Add(3, "Tres");
```

También se puede inicializar utilizando la sintaxis inicializadora con claves y valores como se muestra a continuación.

Ejemplo

```
IDictionary<int, string> diccionario = new Dictionary<int, string>()
{
    {1, "Uno"},
    {2, "Dos"},
    {3, "Tres"}
};
```

## Acceso a los elementos

Podemos acceder a los elementos del diccionario de muchas maneras, por ejemplo, usando los bucles **foreach**, **for** o **indexer**.

Podemos usar los bucles **foreach** o **for** para iterar sobre todos los elementos del diccionario. El diccionario almacena pares clave-valor, por lo tanto, podemos usar un tipo **KeyValuePair <TKey, TValue>** o una variable implícitamente escrita en el bucle **foreach** como se muestra a continuación.

## Ejemplo

```
using System;
using System.Collections.Generic;

namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            Dictionary<int, string> diccionario = new Dictionary<int, string>()
            {
                {1, "Uno"},
                {2, "Dos"},
                {3, "Tres"}
            };

            foreach (KeyValuePair<int, string> elemento in diccionario)
            {
                Console.WriteLine("Clave: {0}, Valor: {1}", elemento.Key,
elemento.Value);
            }
            Console.ReadKey();
        }
    }
}
```

## Resultado

Clave: 1, Valor: Uno

Clave: 2, Valor: Dos

Clave: 3, Valor: Tres

Utilizamos el bucle **for** para acceder a todos los elementos, para ello usamos la propiedad **Count** del diccionario para obtener el número total de elementos en el diccionario.

## Ejemplo

```
using System;
using System.Collections.Generic;
using System.Linq;

namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
```

```

    {
        Dictionary<int, string> diccionario = new Dictionary<int, string>()
            {
                {1, "Uno"},
                {2, "Dos"},
                {3, "Tres"}
            };

        for (int i = 0; i < diccionario.Count; i++)
        {
            Console.WriteLine("Clave: {0}, Valor: {1}",
diccionario.Keys.ElementAt(i), diccionario[diccionario.Keys.ElementAt(i)]);
        }

        Console.ReadKey();
    }
}

```

## Resultado

Clave: 1, Valor: Uno

Clave: 2, Valor: Dos

Clave: 3, Valor: Tres

El diccionario se puede utilizar como una matriz para acceder a sus elementos individuales, para ello debemos especificar la clave para obtener un valor de un diccionario usando un indexador como una matriz.

## Ejemplo

```

using System;
using System.Collections.Generic;

namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            Dictionary<int, string> diccionario = new Dictionary<int, string>()
                {
                    {1, "Uno"},
                    {2, "Dos"},
                    {3, "Tres"}
                };

            Console.WriteLine(diccionario[1]); //devuelve Uno
            Console.WriteLine(diccionario[2]); // devuelve Dos
        }
    }
}

```

```
        Console.ReadKey();
    }
}
}
```

## Resultado

Uno

Dos

El indexador coge la clave como parámetro y si la clave especificada no existe, lanzará una excepción **KeyNotFoundException**.

Si no estamos seguros de la clave, es mejor usar el método **TryGetValue()**. El método **TryGetValue()** nos devolverá false si no puede encontrar las claves en lugar de lanzar una excepción.

## Ejemplo

```
using System;
using System.Collections.Generic;

namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            Dictionary<int, string> diccionario = new Dictionary<int, string>()
            {
                {1, "Uno"},
                {2, "Dos"},
                {3, "Tres"}
            };

            string resultado;
            if (diccionario.TryGetValue(4, out resultado))
            {
                Console.WriteLine(resultado);
            }
            else
            {
                Console.WriteLine("No se ha podido encontrar la Clave
especificada");
            }

            Console.ReadKey();
        }
    }
}
```

```
}  
}
```

## Resultado

No se ha podido encontrar la Clave especificada

## Comprobar si existen elementos

El diccionario incluye varios métodos para determinar si un diccionario contiene elementos o claves específicos. Use el método **ContainsKey()** para verificar si existe una clave específica en el diccionario o no.

Use el método **Contains()** para verificar si un par de clave y valor especificado existe en el diccionario o no.

### Ejemplo

```
using System;  
using System.Collections.Generic;  
using System.Linq;  
  
namespace ConsoleApplication1  
{  
    class Program  
    {  
        static void Main(string[] args)  
        {  
            Dictionary<int, string> diccionario = new Dictionary<int, string>()  
                {  
                    {1, "Uno"},  
                    {2, "Dos"},  
                    {3, "Tres"}  
                };  
  
            diccionario.ContainsKey(1); // devuelve true  
            diccionario.ContainsKey(4); // devuelve false  
            Console.WriteLine(diccionario.Contains(new KeyValuePair<int,  
string>(1, "Uno"))); // devuelve true  
  
            Console.ReadKey();  
        }  
    }  
}
```

## Resultado

```
true
```

Otra sobrecarga del método **Contains()** coge a **IEqualityComperer** como un segundo parámetro. Utilizaremos una instancia de **IEqualityComparer** cuando desea personalizar la comparación de igualdad.

## Ejemplo

```
using System;
using System.Collections.Generic;
using System.Linq;

namespace ConsoleApplication1
{
    public class Persona
    {
        public int PersonaID { get; set; }
        public string PersonaName { get; set; }
    }

    class PersonaDictionaryComparer : IEqualityComparer<KeyValuePair<int,
Persona>>
    {
        public bool Equals(KeyValuePair<int, Persona> x, KeyValuePair<int,
Persona> y)
        {
            if (x.Key == y.Key && (x.Value.PersonaID == y.Value.PersonaID) &&
(x.Value.PersonaName == y.Value.PersonaName))
                return true;

            return false;
        }

        public int GetHashCode(KeyValuePair<int, Persona> obj)
        {
            return obj.Key.GetHashCode();
        }
    }

    class Program
    {
        static void Main(string[] args)
        {
            IDictionary<int, Persona> personaDiccionario = new
Dictionary<int, Persona>()
            {
                { 1, new Persona(){ PersonaID =1, PersonaName = "Juan"}},
                { 2, new Persona(){ PersonaID =2, PersonaName =
"Marcos"}},
                { 3, new Persona(){ PersonaID =3, PersonaName = "Paco"}}
            };

            Persona persona = new Persona() { PersonaID = 1, PersonaName =
"Juan" };
        }
    }
}
```



```

        KeyValuePair<int, Persona> elementToFind = new KeyValuePair<int,
Persona>(1, persona);

        bool resultado = personaDiccionario.Contains(elementToFind, new
PersonaDictionaryComparer()); // devuelve true

        Console.WriteLine(resultado);
        Console.ReadKey();
    }
}
}

```

## Resultado

True

En el ejemplo anterior, hemos utilizado **PersonaDictionaryComparer** que deriva de **IEqualityComparer** para comparar los objetos de Persona en el diccionario. El comparador predeterminado solo funcionará con tipos de datos primitivos.

## Eliminar elementos

Utilice el método **Remove()** para eliminar un elemento existente del diccionario. **Remove()** tiene dos sobrecargas, un método de sobrecarga acepta una clave y el otro método de sobrecarga acepta un **KeyValuePair** <> como parámetro.

### Ejemplo

```

using System;
using System.Collections.Generic;
using System.Linq;

namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            Dictionary<int, string> diccionario = new Dictionary<int, string>()
                {
                    {1, "Uno"},
                    {2, "Dos"},
                    {3, "Tres"}
                };

            Diccionario.Remove(1); //elimina el elemento que tiene 1 como clave
        }
    }
}

```

```
}  
}
```

Tanto la clave como el valor deben coincidir para eliminar un elemento. El elemento no se eliminará si ambos no coinciden. Por ejemplo, el siguiente ejemplo no eliminará ningún elemento:

```
using System;  
using System.Collections.Generic;  
using System.Linq;  
  
namespace ConsoleApplication1  
{  
    class Program  
    {  
        static void Main(string[] args)  
        {  
            Dictionary<int, string> diccionario = new Dictionary<int, string>()  
                {  
                    {1, "Uno"},  
                    {2, "Dos"},  
                    {3, "Tres"}  
                };  
            //no elimina nada porque no encuentra el valor Dos1  
            Diccionario.Remove(new KeyValuePair<int, string>(2, "Dos1"));  
        }  
    }  
}
```

Para ordenar colección de diccionarios según las claves podemos usar la colección genérica **SortedDictionary**.

## Agregar valores

El método de adición en el diccionario toma dos parámetros, uno para la clave y otro para el valor.

### Sintaxis

```
public void Add(TKey key, TValue value)
```

Por ejemplo

```
diccionario.Add("docena",12);
```

La clave en un diccionario no debe ser nula, pero un valor puede serlo, si TValue es un tipo de referencia.

## Recuperar par clave-valor

Podemos recuperar los valores del diccionario utilizando el bucle **foreach**.

```
using System;
using System.Collections.Generic;

public class ReflectionExample
{
    static void Main()
    {
        Dictionary<string, int> diccionario = new Dictionary<string, int>();
        diccionario.Add("Uno", 1);
        diccionario.Add("Dos", 2);
        diccionario.Add("Tres", 3);
        diccionario.Add("cuatro", 4);

        foreach (KeyValuePair<string, int> par in diccionario)
        {
            Console.WriteLine(par.Key.ToString() + " - " +
par.Value.ToString());
        }
        Console.ReadKey();
    }
}
```

## Resultado

```
Uno - 1
```

```
Dos - 2
```

```
Tres - 3
```

```
cuatro - 4
```

## Buscar una clave

Podemos buscar una clave en el diccionario usando el método **ContainsKey** para probar si existe una clave o no. **ContainsKey** calcula el **hashcode** para el argumento y verifica las estructuras internas en el Diccionario.

```
using System;
using System.Collections.Generic;

public class ConsoleApplication1
{
    static void Main()
    {
        Dictionary<string, int> diccionario = new Dictionary<string, int>();
        diccionario.Add("Uno", 1);
        diccionario.Add("Dos", 2);
        diccionario.Add("Tres", 3);
        diccionario.Add("cuatro", 4);

        if (diccionario.ContainsKey("cuatro") == true)
        {
            Console.WriteLine(diccionario["cuatro"].ToString());
        }
        else
        {
            Console.WriteLine("La Clave no Existe");
        }

        Console.ReadKey();
    }
}
```

## Resultado

4

FIN