Introduction

Given that you can assign a variable of any type to an `interface{}`, often people will try code like the following.

```
var dataSlice []int = foo()
var interfaceSlice []interface{} = dataSlice
```

This gets the error

```
cannot use dataSlice (type []int) as type []interface { } in assignment
```

The question then, "Why can't I assign any slice to an `[]interface{}`, when I can assign any type to an `interface{}`?"

Why?

There are two main reasons for this.

The first is that a variable with type `[]interface{}` is not an interface! It is a slice whose element type happens to be `interface{}`. But even given this, one might say that the meaning is clear.

Well, is it? A variable with type `[]interface{}` has a specific memory layout, known at compile time.

Each `interface{}` takes up two words (one word for the type of what is contained, the other word for either the contained data or a pointer to it). As a consequence, a slice with length N and with type `[]interface{}` is backed by a chunk of data that is N*2 words long.

This is different than the chunk of data backing a slice with type `[]MyType` and the same length. Its chunk of data will be N*sizeof(MyType) words long.

The result is that you cannot quickly assign something of type `[]MyType` to something of type `[]interface{}`; the data behind them just look different.

What can I do instead?

It depends on what you wanted to do in the first place.

If you want a container for an arbitrary array type, and you plan on changing back to the original type before doing any indexing operations, you can just use an `interface{}`. The code will be generic (if not compile-time type-safe) and fast.

If you really want a `[]interface{}` because you'll be doing indexing before converting back, or you are using a particular interface type and you want to use its methods, you will have to make a copy of the slice.

```
var dataSlice []int = foo()
var interfaceSlice []interface{} = make([]interface{}, len(dataSlice))
for i, d := range dataSlice {
    interfaceSlice[i] = d
}
```