Generics

Write a name inside angle brackets to make a generic function or type.

```
func makeArray<Item>(repeating item: Item, numberOfTimes: Int) -> [Item] {
    var result: [Item] = []
    for _ in 0..<numberOfTimes {
        result.append(item)
    }
    return result
}
makeArray(repeating: "knock", numberOfTimes: 4)
```

You can make generic forms of functions and methods, as well as classes, enumerations, and structures.

```
// Reimplement the Swift standard library's optional type
enum OptionalValue<Wrapped> {
    case none
    case some(Wrapped)
}
var possibleInteger: OptionalValue<Int> = .none
possibleInteger = .some(100)
```

Use `where` right before the body to specify a list of requirements—for example, to require the type to implement a protocol, to require two types to be the same, or to require a class to have a particular superclass.

```
func anyCommonElements<T: Sequence, U: Sequence>(_ lhs: T, _ rhs: U) ->
  Bool
    where T.Element: Equatable, T.Element == U.Element
{
    for lhsItem in lhs {
        for rhsItem in rhs {
            if lhsItem == rhsItem {
                return true
            }
        }
    }
    return false
}
anyCommonElements([1, 2, 3], [3])
```

> EXPERIMENT
>
> Modify the `anyCommonElements(_:_:)` function to make a function that returns an array of the elements that any two sequences have in common.

Writing `<T: Equatable>` is the same as writing `<T> ... where T: Equatable`.