Mutable vs Immutable Objects

A **mutable** object can be changed after it's created, and an **immutable** object can't.

For example, let's look at **lists** and **tuples** in Python. Lists are mutable and tuples are immutable:

```python
# Python

int_list  = [4, 9]
int_tuple = (4, 9)

int_list[0] = 1
# list is now [1, 9]

int_tuple[0] = 1
# raises: TypeError: 'tuple' object does not support item assignment
```

Different languages have different policies on whether strings should be mutable. Ruby has mutable strings:

```ruby
test_string = 'mutable?'
test_string[7] = '!'
# string is now 'mutable!'
```

But strings are *immutable* in Python:

```python
test_string = 'mutable?'
test_string[7] = '!'
# TypeError: 'str' object does not support item assignment
```

And strings are also immutable in JavaScript:

```javascript
var testString = 'mutable?';
testString[7] = '!';
// string is still 'mutable?'
// (but no error is raised!)
```

In C++ and C, strings can either be mutable or immutable, depending on whether the string is declared with the **const** modifier:

```c++
string testString("mutable?");
testString[7] = '!';
// testString is now "mutable!"

const string testString2("mutable?");
testString2[7] = '!'; // compile-time error
```

C

```
char testString[16] = "mutable?";
testString[7] = '!';
// testString is now "mutable!"

const char testString2[16] = "mutable?";
testString2[7] = '!'; // compile-time error
```

In Swift, strings can either be mutable or immutable, depending on whether the string is declared with the `var` keyword:

Swift (beta)

```
var testString = "mutable?"
if let range = testString.range(of: "?") {
    testString.replaceSubrange(range, with: "!")
    // testString is now "mutable!"
}

let testString = "mutable?"
if let range = testString.range(of: "?") {
    testString.replaceSubrange(range, with: "!")
    // Cannot use mutating member on immutable value
}
```

Mutable objects are nice because you can make changes "in-place," without allocating a new object. But be careful—whenever you make an in-place change to an object, *all* references to that object will now reflect the change (whether you like it or not)!

What's next?

If you're ready to start applying these concepts to some problems, check out our mock coding interview questions (/next).

They mimic a real interview by offering hints when you're stuck or you're missing an optimization.

Try some questions now ➜
