Swift Pills 💊 #3

Vincent Pradeilles - Worldline

❤️ Optionals ❤️

❤️ Optionals ❤️

```
let date: Date? = Date() // or could be nil
let formatter = DateFormatter()
let label = UILabel()

if let safeDate = date {
    label.text = formatter.string(from: safeDate)
}
```

❤️ Optionals ❤️

```
let date: Date? = Date() // or could be nil
let formatter = DateFormatter()
let label = UILabel()

if let safeDate = date {
    label.text = formatter.string(from: safeDate)
}
```

❤️ Optionals ❤️

```
let date: Date? = Date() // or could be nil
let formatter = DateFormatter()
let label = UILabel()

if let safeDate = date {
    label.text = formatter.string(from: safeDate)
}
```

❤️ Optionals ❤️

```
let date: Date? = Date() // or could be nil
let formatter = DateFormatter()
let label = UILabel()

if let safeDate = date {
    label.text = formatter.string(from: safeDate)
}
```

❤️ Optionals ❤️

```
let date: Date? = Date() // or could be nil
let formatter = DateFormatter()
let label = UILabel()

if let safeDate = date {
    label.text = formatter.string(from: safeDate)
}
```

❤️ Optionals ❤️

```
if let safeDate = date {
    label.text = formatter.string(from: safeDate)
}
```

How about we look at the documentation? 🤔

Generic Instance Method

map(_:)

Evaluates the given closure when this `Optional` instance is not `nil`, passing the unwrapped value as a parameter.

Declaration

```
func map<U>(_ transform: (Wrapped) throws -> U) rethrows -> U?
```

Parameters

`transform`
A closure that takes the unwrapped value of the instance.

Return Value

The result of the given closure. If this instance is `nil`, returns `nil`.

Generic Instance Method

map(_:)

Evaluates the given closure when this `Optional` instance is not `nil`, passing the unwrapped value as a parameter.

Declaration

```
func map<U>(_ transform: (Wrapped) throws -> U) rethrows -> U?
```

Parameters

`transform`

A closure that takes the unwrapped value of the instance.

Return Value

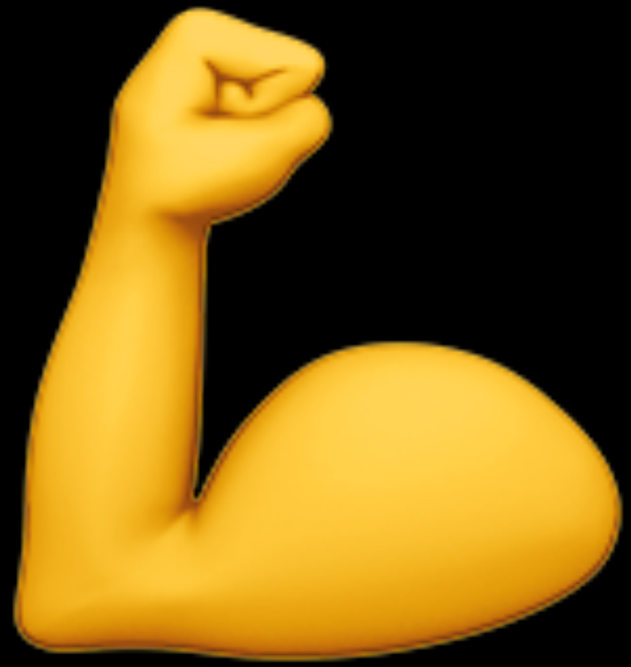
The result of the given closure. If this instance is `nil`, returns `nil`.

❤️ Optionals ❤️

```
label.text = date.map { return formatter.string(from: $0) }
// or
label.text = date.map(formatter.string(from:))
```

❤️ Optionals ❤️

```
label.text = date.map { return formatter.string(from: $0) }
// or
label.text = date.map(formatter.string(from:))
```



❤️ Optionals ❤️

```
func doesNotWorkWithOptionalString(_ param: String) {
    // does something really cool
}

let label = UILabel()
label.text = "This is some text."

doesNotWorkWithOptionalString(label.text ?? "")
```

❤️ Optionals ❤️

```
func doesNotWorkWithOptionalString(_ param: String) {
    // does something really cool
}

let label = UILabel()
label.text = "This is some text."

doesNotWorkWithOptionalString(label.text ?? "")
```

❤️ Optionals ❤️

```
func doesNotWorkWithOptionalString(_ param: String) {
    // does something really cool
}

let label = UILabel()
label.text = "This is some text."

doesNotWorkWithOptionalString(label.text ?? "")
```

❤️ Optionals ❤️

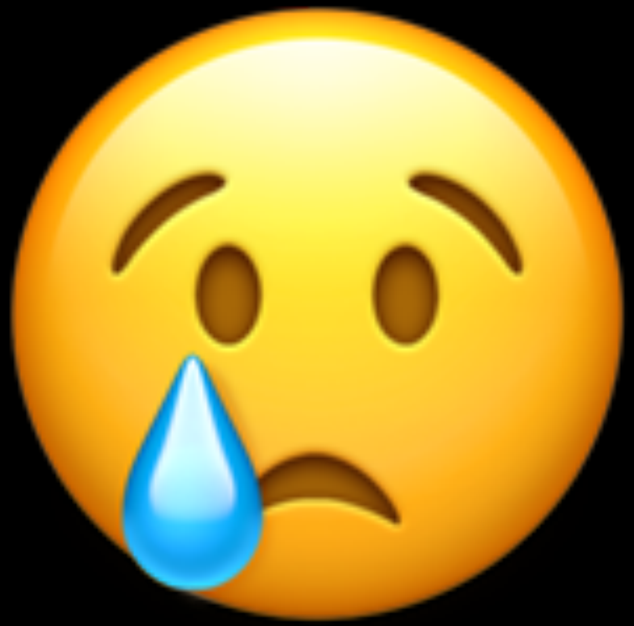
```
func doesNotWorkWithOptionalString(_ param: String) {
    // does something really cool
}

let label = UILabel()
label.text = "This is some text."

doesNotWorkWithOptionalString(label.text ?? "")
```

❤️ Optionals ❤️

```
label.text ?? ""
```



❤️ Optionals ❤️

```
extension Optional where Wrapped == String {
    var orEmpty: String {
        switch self {
        case .some(let value):
            return value
        case .none:
            return ""
        }
    }
}
```

❤️ Optionals ❤️

```
extension Optional where Wrapped == String {
    var orEmpty: String {
        switch self {
        case .some(let value):
            return value
        case .none:
            return ""
        }
    }
}
```

❤️ Optionals ❤️

```
extension Optional where Wrapped == String {
    var orEmpty: String {
        switch self {
        case .some(let value):
            return value
        case .none:
            return ""
        }
    }
}
```

❤️ Optionals ❤️

```
extension Optional where Wrapped == String {
    var orEmpty: String {
        switch self {
        case .some(let value):
            return value
        case .none:
            return ""
        }
    }
}
```

❤️ Optionals ❤️

```
extension Optional where Wrapped == String {
    var orEmpty: String {
        switch self {
        case .some(let value):
            return value
        case .none:
            return ""
        }
    }
}
```

❤️ Optionals ❤️

```
doesNotWorkWithOptionalString(label.text.orEmpty)
```