Optionals

Optionals say one of two things:

1. There **is** a value and it equals x.
2. There **is no** value at all.

```
let possibleNumber = "123"
let convertedNumber = Int(possibleNumber)
// convertedNumber is inferred to be of type "Int?", or "optional Int"
```

- `convertedNumber` is referred to as an `Int?` or an “optional Int”.
- The initializer for converting `possibleNumber` to an Int **may fail** and return `nil`.

nil

- An optional can be set to a valueless state by setting it equal to `nil`.
- The default value of any optional (set if a value is not defined) is `nil`.

```
var serverResponseCode: Int? = 404
// serverResponseCode contains an actual Int value of 404
serverResponseCode = nil
// serverResponseCode now contains no value
```

Optionals in `if` Statements and Forced Unwrapping

- You can check if an optional has a value by comparing it to `nil`.
- Once you are sure an optional has a value, you can explicitly unwrap it with the `!` operator.

```
if convertedNumber != nil {
    print("convertedNumber has an integer value of \(convertedNumber!).")
}
```

Optional Binding

- Lets you check an optional for a value and store that value into a constant or variable in one step.
- Allows you to rewrite the example a few cells above as the following.

```
if let actualNumber = Int(possibleNumber) {
    print("\'\(possibleNumber)\' has an integer value of \(actualNumber)")
} else {
```

```
    print("\'\(possibleNumber)\' could not be converted to an integer")
}
```

This code can be read as:

- “If the optional `Int` returned by `Int(possibleNumber)` contains a value, set a new constant called `actualNumber` to the value contained in the optional.”

You can include multiple optional bindings in a single if statement and use a where clause to check for a Boolean condition

```
if let firstNumber = Int("4"), secondNumber = Int("42") where firstNumber < secondNumber {
    print("\(firstNumber) < \(secondNumber)")
}
```

Implicitly Unwrapped Optionals

- A specific type of optional that will *always* have a value.
    - Declared with a `!` instead of a `?`.
- These are useful when an optional’s value is confirmed to exist immediately after definition and at every point thereafter.
- Primary use of this concept is in **class initialization.**

```
let possibleString: String? = "An optional string."
let forcedString: String = possibleString! // requires an exclamation mark

let assumedString: String! = "An implicitly unwrapped optional string."
let implicitString: String = assumedString // no need for an exclamation mark
```

- The difference here is that the implicity unwrapped optional, `assumedString`, never needs to be unwrapped since its declaration, `let assumedString: String!`, guarantees it to have a value.
- **Never** use an implicitly unwrapped optional if there is a chance of the optional becoming `nil`.

NOTE: If an implicitly unwrapped optional is nil and you try to access its wrapped value, you’ll trigger a runtime error. The result is exactly the same as if you place an exclamation mark after a normal optional that does not contain a value.