Cheatsheets / **Learn Intermediate Swift**

Optionals

Optional Types

Optionals types are declared with a `?` after the type name. Optionals either contain a value or nil which represents the *absence* of a value.

```
var email: String? = nil
email = "codey@codecademy.com"
```

Force Unwrapping Optionals

The `!` operator force unwraps an optional. If the underlying value is not nil, it can then be used. If the underlying value is nil, then the program will crash.

```
var name: String? = "Codey"
var email: String? = nil

print("The user's name is \(userName!)")
// Prints "The user's name is Codey"
print("The user's email is \
(userEmail!)") // Crashes!
```

Optional Binding

Safely unwrap optionals by using an `if let` statement to bind the optional to a new variable. If the optional is nil, then the code in the `else` block will run.

```
var name: String? = "Codey"
var email: String? = nil

if let name = name {
  print("The user's name is \(name)")
} else {
  print("No name available")
}

if let unwrappedEmail = email {
  print("The user's email is \
(unwrappedEmail)")
} else {
```

```
  print("No email available")
}

// Prints:
// "The user's name is Codey"
// No email available
```

Multiple Optional Bindings

Multiple optionals can be bound in the same `if let` statement, separating each binding with ",". `if let` statements can also check to see if a boolean expression evaluates to `true`.

```
var name: String? = "Codey"
var email: String? =
"codey@codecademy.com"

if let name = name, let email = email,
email.contains("@") {
  print("Welcome to Codecademy \(name)!
Your email address is \(email)")
} else {
  print("Name is nil, email is nil, or
the email is invalid")
}

// Prints "Welcome to Codecademy Codey!
Your email address is
codey@codecademy.com"
```

Guard Statements

A `guard` block is another way to write a conditional in Swift. All guard statements must have an else block that exits the current scope if the boolean expression is false. If the `guard` statement is true, the code below continues executing. Optionals can be bound in a guard block using the `guard let` syntax.

```
var name: String? = "Codey"
var email: String? =
"codey@codecademy.com"

func displayMessageIfValid() {
  guard let name = name, let email =
email, email.contains("@") else {
    return
```

```
  }
  print("Welcome \(name)!  Your email is
\(email)")
}

displayMessageIfValid()
// Prints: "Welcome Codey!  Your email
is codey@codecademy.com"
```

Nil-Coalescing Operator

The nil-coalescing operator `??` unwraps an optional value and provides a default if the optional is nil.

```
var email: String? = nil
print("Welcome!  Your email is \(email
?? "unknown").")

// Prints: "Welcome!  Your email is
unknown."
```

Optionals and Functions

Functions can take in optional types and return optional types. A function should return nil when it isn't able to return a sensible instance of the requested type.

```
func getFirstInitial(from name: String?)
-> String? {
  return name?.first
}
```