Cheatsheets / **Learn Intermediate Swift**

Properties and Access Control

Access Control

Access control specifies which methods and properties can be accessed from outside that scope of a structure, file, or module. Access control makes it possible to hide implementation details and protect properties from being changed at unexpected times.

Access Levels

Swift provides several different levels of access. From least restrictive to most restrictive they are:

- open / public
- internal
- fileprivate
- private

```
// public structures can be accessed in
other modules
public struct User {
  // internal is the default level of
access control
        let name: String
  // fileprivate methods can only be
accessed inside of the file they're
declared in
  fileprivate func incrementVisitCount()
{
    visitCount += 1
  }
  // private properties can only be
accessed inside their structure's
definition
  private let visitCount = 0
}
```

Private Properties and Methods

Mark methods and properties as `private` to prevent them from being accessed outside of the structure, class, or enumeration's definition.

```
struct User {
  let name: String
  init(name: String) {
    self.name = name
    uploadNewUser()
  }
  private func uploadNewUser() {
    print("Uploading the new user...")
  }
}
```

Read-only Computed Properties

Read-only computed properties can be accessed, but not assigned to a new value. To define a read-only computed property, either use the `get` keyword without a `set` keyword, or omit keywords entirely.

```
struct Room {
  let width: Double
  let height: Double
  var squareFeet: Double {
    return width * height
  }
  var description: String {
    get {
      return "This room is \(width) x \
(height)"
    }
  }
}
```

Property Observers

Property observers execute code whenever a property is changed. The `willSet` property observer is triggered right before the property is changed and creates a `newValue` variable within the block's scope. The `didSet` property observer is triggered right after the property is changed and creates an `oldValue` within the block's scope.

```
struct Employee {
  var hourlyWage = 15 {
    willSet {
      print("The hourly wage is about to
be changed from \(hourlyWage) to \
(newValue)")
    }
```

```
    didSet {
      print("The hourly wage has been
changed from \(oldValue) to \
(hourlyWage)")
    }
  }
}

var codey = Employee()
codey.hourlyWage = 20

// Prints:
// The hourly wage is about to be
changed from 15 to 20
// The hourly wage has been changed from
15 to 20
```

Private Setters

Properties marked as `private(set)` can be accessed from outside the scope of its structure, but only assigned within it. This allows the setter to be more restrictive than the getter.

```
struct User {
        private(set) var name: String
  mutating func updateName(to newName:
String) {
        if newName != "" {
        name = newName
    }
  }
}

var currentUser = User(name: "codey")
currentUser.updateName(to: "Codey")
print(currentUser.name)
// currentUser.name = "Bob" // This line
doesn't compile because the 'name'
setter is inaccessible
```

Static Properties and Methods

The `static` keyword is used to declare type methods and properties. These are accessed from the type itself rather than an instance.

```
struct User {
  static var allUsers = [User]()
  let id: Int
  init(id: Int) {
    self.id = id
    User.allUsers.append(self)
  }
}

let userOne = User(id: 1)
let userTwo = User(id: 2)
let userThree = User(id: 3)

print(User.allUsers) // Prints:
[User(id: 1), User(id: 2), User(id: 3)]
```

Extensions

The `extension` keyword is used to continue defining an existing class, structure, or enumeration from anywhere in a codebase. Extensions can have new methods, internal types, and computed properties, but can't contain new stored properties.

```
struct User {
  let name: String
}

extension User {
  var description: String {
    return "This is a user named \
(name)"
  }
}
```