Cheatsheets / **Learn Intermediate Swift**

Enumerations

Defining an Enumeration

Enumerations (or enums) are used to define a new custom type with a list of possible cases. When creating an instance of an enumeration, its value must be one of the cases.

```
enum Day {
  case monday
  case tuesday
  case wednesday
  case thursday
  case friday
  case saturday
  case sunday
}

let casualWorkday: Day = .friday
```

Switch Statements

Switch statements are used to determine the case of an enumeration. When switching on an enumeration, all cases must be addressed if a `default` is not provided. Switching on enumerations can also access the associated values of a case.

```
enum Dessert {
    case cake(flavor: String)
    case vanillaIceCream(scoops: Int)
    case brownie
}

let customerOrder: Dessert =
.cake(flavor: "Red Velvet")

switch customerOrder {
  case let .cake(flavor):
        print("You ordered a \(flavor)
cake")
  case let .vanillaIceCream(scoopCount):
```

```
        print("You ordered \(scoopCount)
scoops of vanilla ice cream")
  case .brownie:
        print("You ordered a brownie")
}

// Prints: "You ordered a Red Velvet
cake"
```

CaseIterable

Add conformance to the `CaseIterable` protocol to access an `allCases` property that returns an array of all the cases of an enumeration.

```
enum Season: CaseIterable {
    case winter
    case spring
    case summer
    case fall
}

for season in Season.allCases {
    print(season)
}
```

Raw Values

Enumerations can have a raw value associated with each case by adding `: RawValueType` after the enumeration name. A `String`, `Character`, `Int`, `Double`, or `Float` can be assigned as a raw value. Enumerations with a raw value can be instantiated using the `init(rawValue:)`. Instances of enumerations with a raw value have a `rawValue` property.

```
enum Grade: Character {
  case pass = "P"
  case fail = "F"
}

let mathTest = Grade.pass
print(mathTest.rawValue) // Prints "P"
```

Associated Values

Each case in an enumeration can have a value associated with it. Enumerations can have a raw value or cases with associated values, but not both.

```
enum Dessert {
    case cake(flavor: String)
```

```
    case vanillaIceCream(scoops: Int)
    case brownie
}

let customerOrder: Dessert =
.cake(flavor: "Red Velvet")
```

Instance Methods

Just like classes and structures, enumerations can have instance methods. If an instance method changes the value of the enumeration, it needs to be marked as `mutating`.

```
enum Traffic {
  case light
  case medium
  case heavy

  mutating func reportAccident() {
    self = .heavy
  }
}

var currentTraffic: Traffic = .light
currentTraffic.reportAccident() //
currentTraffic is now .heavy
```

Computed Properties

An enumeration can have computed properties defined within its declaration. Enumerations cannot contain stored properties.

```
enum ShirtSize: String {
  case small = "S"
  case medium = "M"
  case large = "L"
  case extraLarge = "XL"

        var description: String {
    return "This shirt size is \
(self.rawValue)"
  }
}
```


Save
Print
Share