: The Swift Programming Language 5.7 : Quick Reference Guide : **Enumerations** :

Enumerations

Enumerations define a common type for a group of related values.
Enumerations can have properties, methods, initialization, extensions, and protocols.

Enumeration Syntax

```
enum CompassPoint {
    case north
    case south
    case east
    case west
}
```

Cases can appear comma-separated on a single line:

```
enum Planet {
    case mercury, venus, earth, mars, jupiter, saturn, uranus, neptune
} // Sorry pluto.
```

Convention suggests to name enum types with capitals,
and naming them as singular rather than plural names.

```
var directionToHead = CompassPoint.west
```

The variable directionToHead receives an inferred type of CompassPoint.

Dot syntax can be used on reassignments:

```
directionToHead = .east
```

Matching Enumeration Values with a Switch Statement

```
directionToHead = .south
switch directionToHead {
case .north:
    print( "Lots of planets have a north" )
case .south:
    print( "Watch out for penguins" )
case .east:
    print( "Where the sun rises" )
case .west:
    print( "Where the skies are blue" )
}
// Prints "Watch out for penguins"
```

Switch statements must be exhaustive.
default can be used to cover non-addressed cases.

```
let somePlanet = Planet.earth
switch somePlanet {
case .earth
    print( "Mostly harmless" )
default:
    print( "Not a safe place for humans" )
}
// Prints "Mostly harmless"
```

Iterating over Enumeration Cases

```
enum Beverage: CaseIterable { // Beverage conforms to the CaseIterable protocol
    case coffee, tea, juice
}
let numberOfChoices = Beverage.allCases.count
print( "\(numberOfChoices) beverages available" )
// Prints "3 beverages available"

for beverage in Beverage.allCases {
    print(beverage)
}
// coffee
// tea
// juice
```

Associated Values

Associated enumeration values are known as discriminated unions, tagged unions, or variants in other programming languages.

```
enum Barcode {
    case upc(Int, Int, Int, Int)
    case qrCode(String)
}
```

The enum definition does not need values of the associated types when defined.

```
var productBarcode = Barcode.upc(8, 85909, 51226, 3) \\ tuple of four Int

productBarcode = .qrCode( "ABCDEFGHIJKLMNOP" )

switch productBarcode {
case .upc(let numberSystem, let manufacturer, let product, let check):
    print( "UPC: \(numberSystem), \(manufacturer), \(product), \(check)." )
case .qrCode(let productCode):
    print( "QR code: \(productCode)." )
}
// Prints "QR code: ABCDEFGHIJKLMNOP."
```

If all associated values of a case are var, or int, a single var or int can be written before the case name:

```
switch productBarcode {
case let .upc(numberSystem, manufacturer, product, check):
    print( "UPC: \(numberSystem), \(manufacturer), \(product), \(check)." )
case let .qrCode(productCode):
    print( "QR code: \(productCode)." )
}
// Prints "QR code: ABCDEFGHIJKLMNOP."
```

Raw Values

Raw values can be prepopulated in the enum definition:

```
enum ASCIIControlCharacter: Character {
    case tab = "\t"
    case lineFeed = "\n"
    case carriageReturn = "\r"
}
\\ The type Character is inferred for the raw values.
```

A raw value can be a unique string, character, integer or floating point number type.
Raw values act like constants. Associated values act like variables.

Implicitly Assigned Raw Values

```
enum Planet: Int {
    case mercury = 1, venus, earth, mars, jupiter, saturn, uranus, neptune
}
// Planet.mercury has an explicit raw value of 1.
// Planet.venus has an implicit raw value of 2.
// Planet.earth has an implicit raw value of 3. And so on.
enum CompassPoint: String {
   case north, south, east, west
}
// CompassPoint.north has an implicit raw value of "north"
// CompassPoint.south has an implicit raw value of "south". And so on.
let earthsOrder = Planet.earth.rawValue
// earthsOrder is 3
let sunsetDirection = CompassPoint.west.rawValue
// sunsetDirection is "west"
```

Initializing from a Raw Value

```
let possiblePlanet = Planet(rawValue: 7)
// possiblePlanet is of type Planet?, or "optional Planet", and equals Planet.uranus
// The raw value initializer is a failable initializer.
let positionToFind = 11
if let somePlanet = Planet(rawValue: positionToFind) {
    switch somePlanet {
    case .earth:
        print( "Mostly harmless" )
    default:
        print( "Not a safe place for humans" )
    }
} else {
    print( "There isn't a planet at position \(positionToFind)" )
}
// Prints "There isn't a planet at position 11"
// somePlanet is assigned the type Planet? or "optional Planet"
```

Recursive Enumerations (indirect)

```
enum ArithmeticExpression {
    case number(Int)
    indirect case addition(ArithmeticExpression, ArithmeticExpression)
    indirect case multiplication(ArithmeticExpression, ArithmeticExpression)
}
// indirect can also be written before enum:
indirect enum ArithmeticExpression {
    case number(Int)
    case addition(ArithmeticExpression, ArithmeticExpression)
    case multiplication(ArithmeticExpression, ArithmeticExpression)
}
// (5 + 4) * 2
let five = ArithmeticExpression.number(5)
let four = ArithmeticExpression.number(4)
let sum = ArithmeticExpression.addition(five, four)
let product = ArithmeticExpression.multiplication(sum, ArithmeticExpression.number(2))
func evaluate(_ expression: ArithmeticExpression) -> Int {
    switch expression {
    case let .number(value):
        return value
    case let .addition(left, right):
        return evaluate(left) + evaluate(right)
    case let .multiplication(left, right):
        return evaluate(left) * evaluate(right)
    }
}
print(evaluate(product))
// Prints "18"
```