Enumerations

Outline

1. Definition
2. Enumeration Syntax
3. Enumeration Usage
4. Associated Values
5. Raw Values
6. Recursive Enumerations

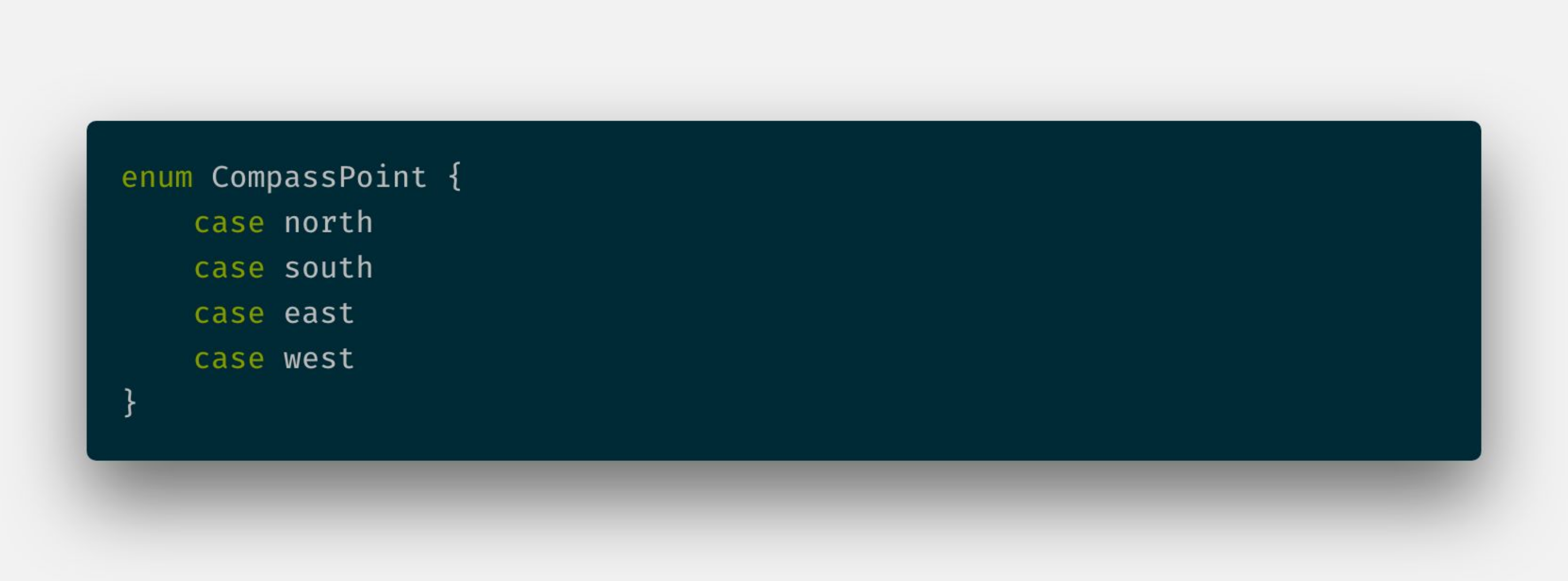
1. Definition

- An *enumeration* defines a common type for a group of related values and enables you to work with those values in a type-safe way within your code.

- Enumerations in Swift are first-class types in their own right.

2. Enumeration Syntax

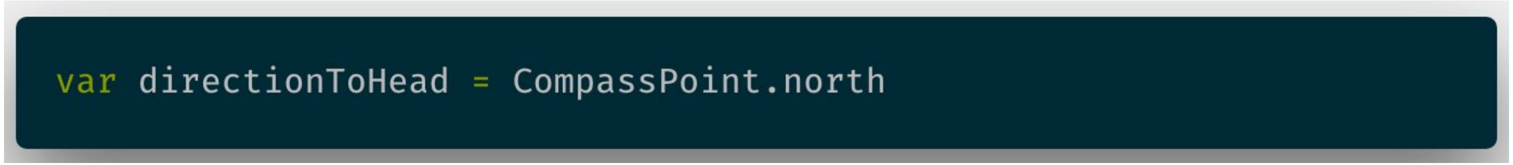
❖ You introduce enumerations with the `enum` keyword and place their entire definition within a pair of braces.



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

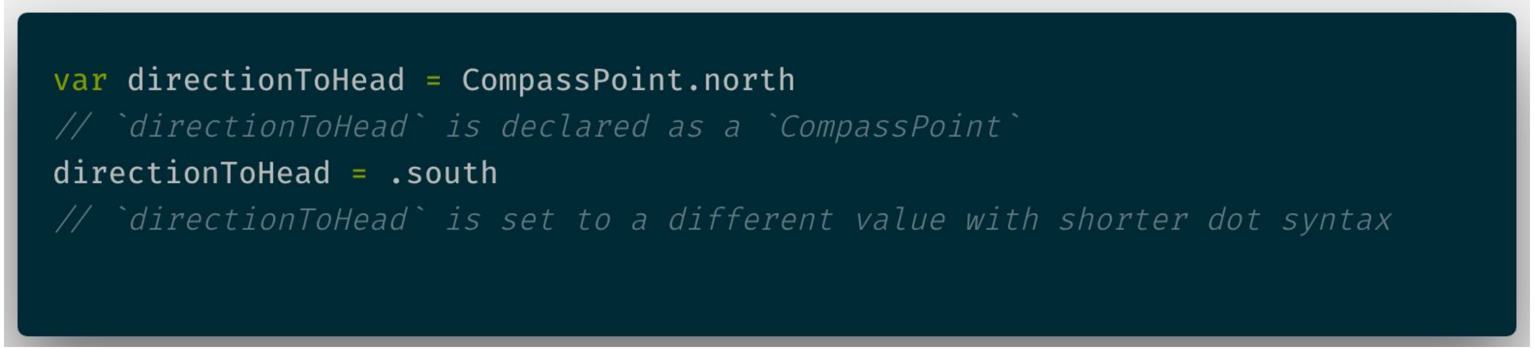
2. Enumeration Syntax (cont)

- Each enumeration definition defines a new type.



```swift
var directionToHead = CompassPoint.north
```

- Once a variable is declared as enum type, you can set it to a different value provided in enum using a shorter dot syntax:



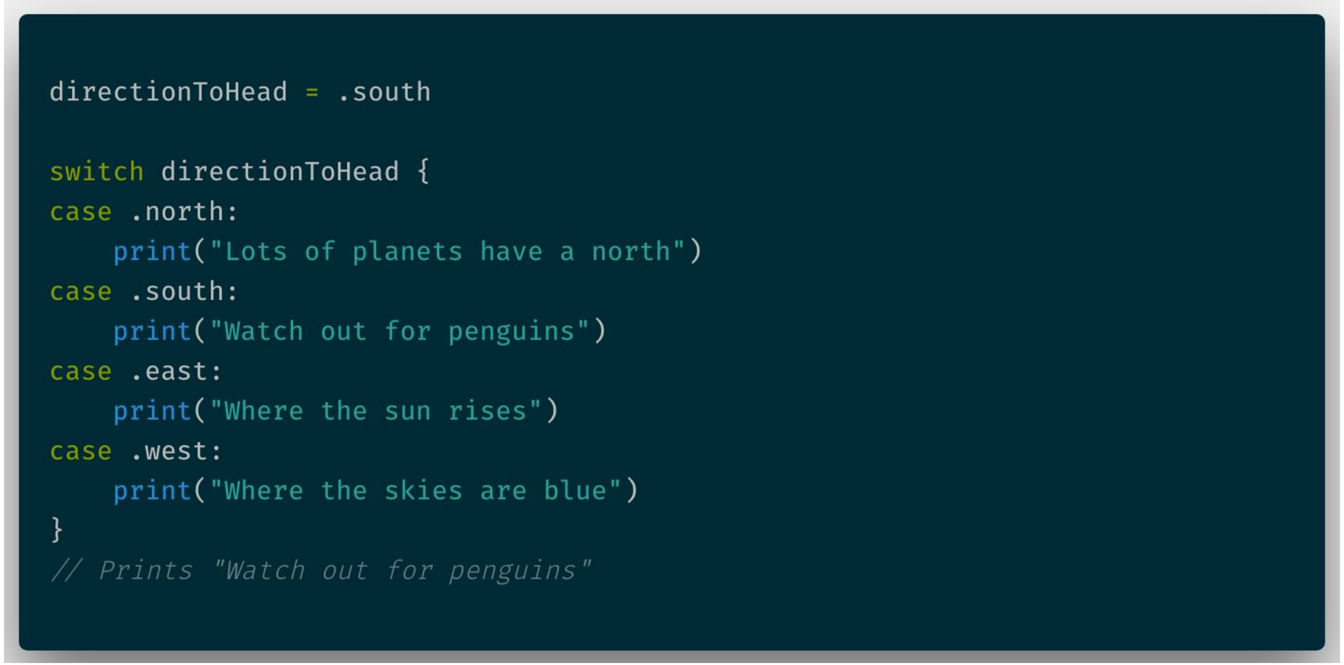
```swift
var directionToHead = CompassPoint.north
// `directionToHead` is declared as a `CompassPoint`
directionToHead = .south
// `directionToHead` is set to a different value with shorter dot syntax
```

3. Enumeration Usage

1. In Switch Statement
2. Iterating over Enumeration Cases

3.1 In Switch Statement

❖ You can match individual enumeration values with a switch statement:

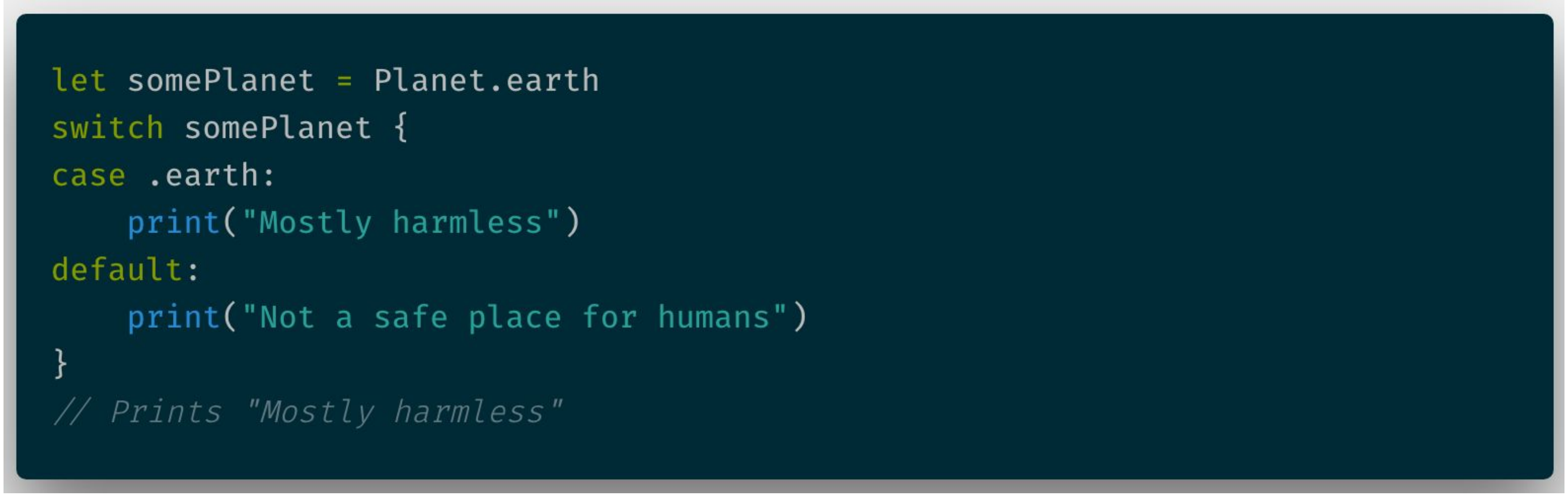


```
directionToHead = .south

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"
```

3.1 In Switch Statement (cont)

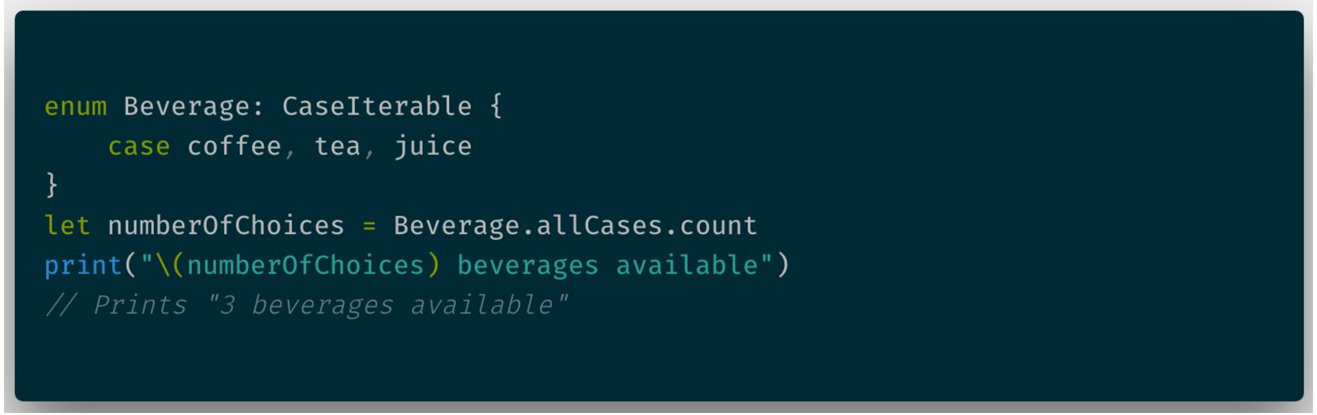
- When it isn’t appropriate to provide a **case** for every enumeration case, you can provide a **default** case to cover any cases that aren’t addressed explicitly:



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

3.2 Iterating over Enumeration Cases

- For some enumerations, it’s useful to have a collection of all of that enumeration’s cases.
- You enable this by writing `: CaseIterable` after the enumeration’s name. Swift exposes a collection of all the cases as an **`allCases`** property of the enumeration type.



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

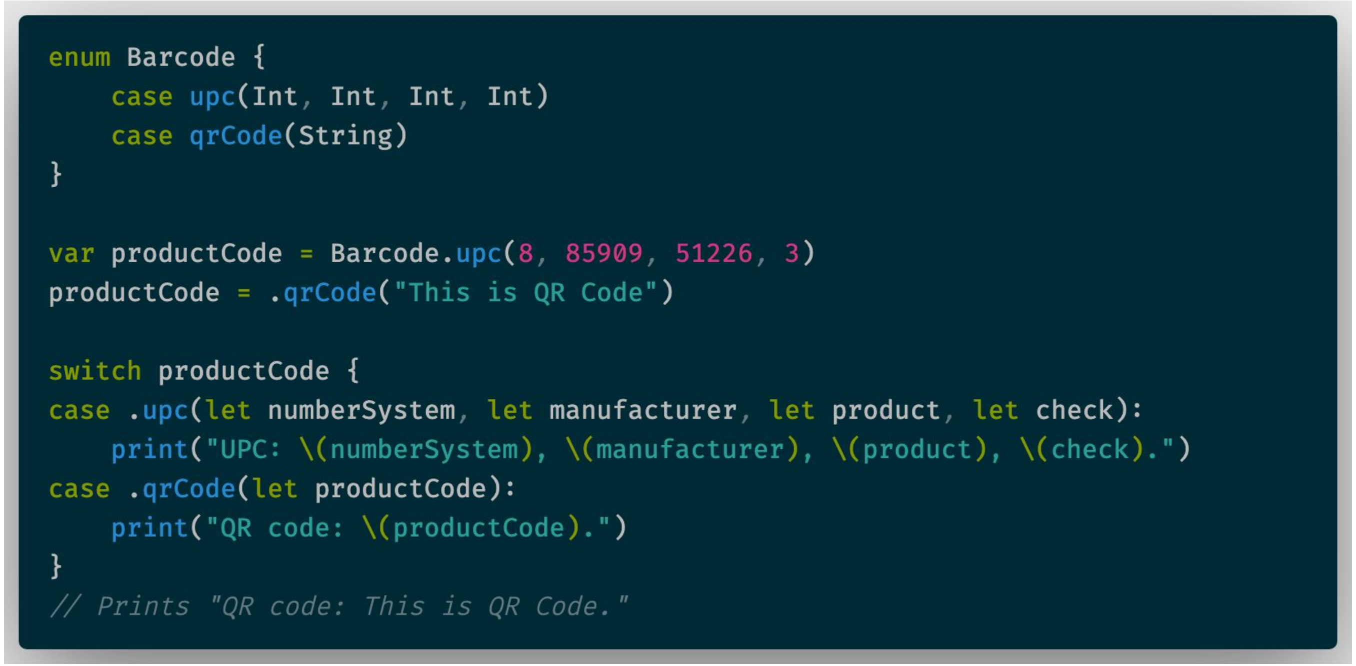
4. Associated Values

- You can define Swift enumerations to store associated values of any given type, and the value types can be different for each case of the enumeration if needed.

- Enumerations similar to these are known as *discriminated unions*, *tagged unions*, or *variants* in other programming languages.

4. Associated Values (cont)

- You can check the different barcode types using a switch statement:



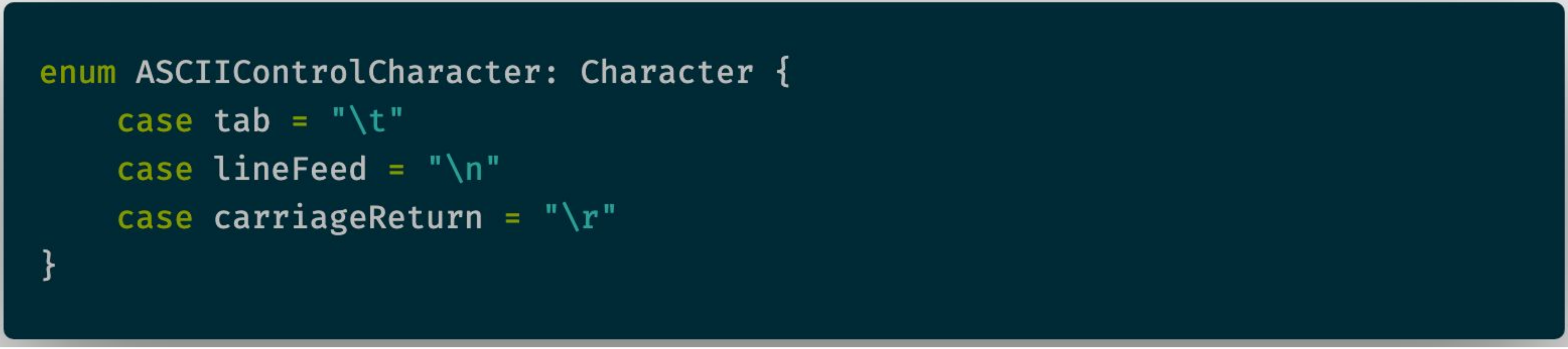
```swift
enum Barcode {
    case upc(Int, Int, Int, Int)
    case qrCode(String)
}

var productCode = Barcode.upc(8, 85909, 51226, 3)
productCode = .qrCode("This is QR Code")

switch productCode {
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: This is QR Code."
```

5. Raw Values

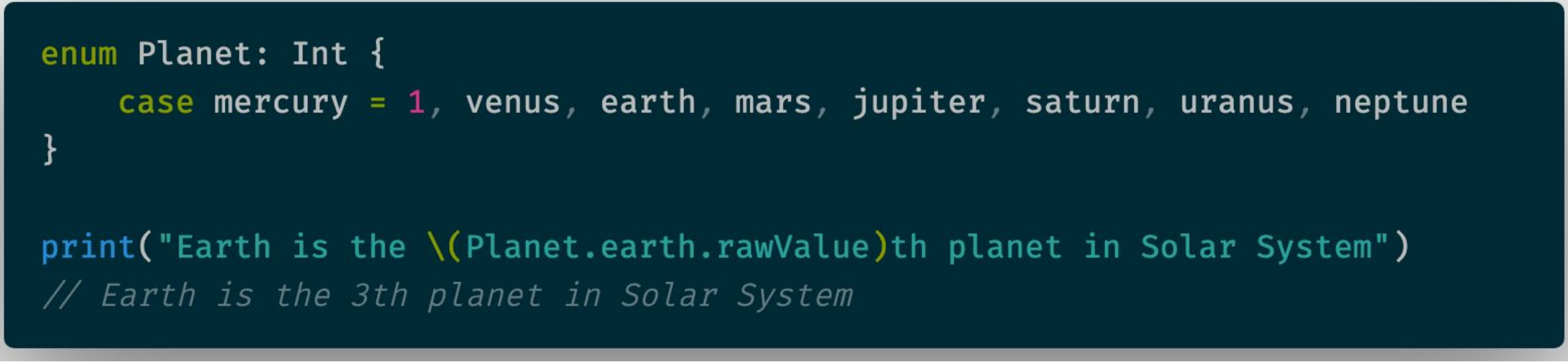
- As an alternative to associated values, enumeration cases can come prepopulated with default values (called *raw values*), which are all of the same type.



```
enum ASCIIControlCharacter: Character {
    case tab = "\t"
    case lineFeed = "\n"
    case carriageReturn = "\r"
}
```

5. Raw Values (cont)

❖ When you’re working with enumerations that store integer or string raw values, you don’t have to explicitly assign a raw value for each case. When you don’t, Swift automatically assigns the values for you.

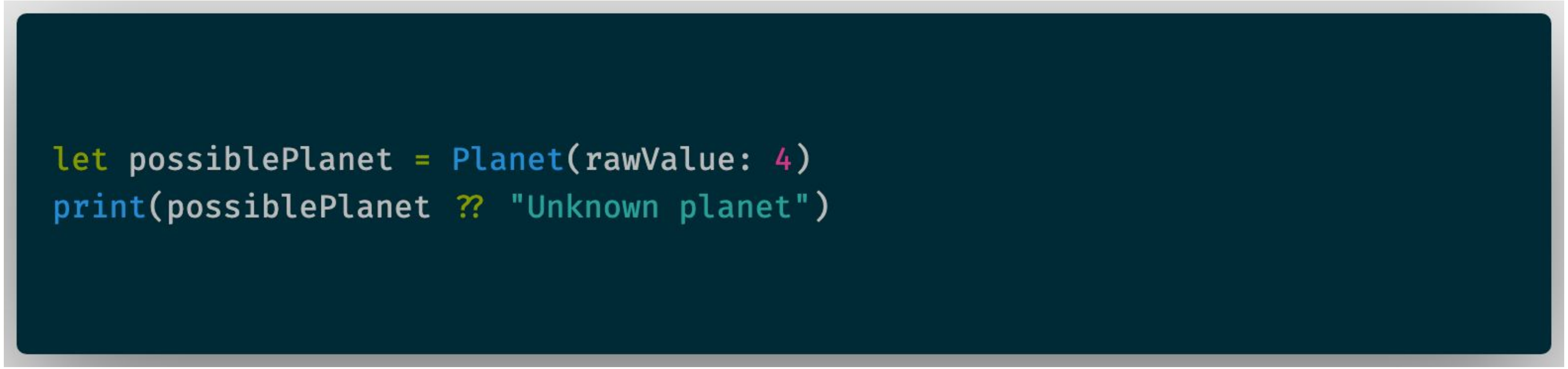


```swift
enum Planet: Int {
    case mercury = 1, venus, earth, mars, jupiter, saturn, uranus, neptune
}

print("Earth is the \(Planet.earth.rawValue)th planet in Solar System")
// Earth is the 3th planet in Solar System
```

5. Raw Values (cont)

- If you define an enumeration with a raw-value type, the enumeration automatically receives an initializer that takes a value of the raw value’s type (as a parameter called `rawValue`) and returns either an enumeration case or `nil`. You can use this initializer to try to create a new instance of the enumeration.



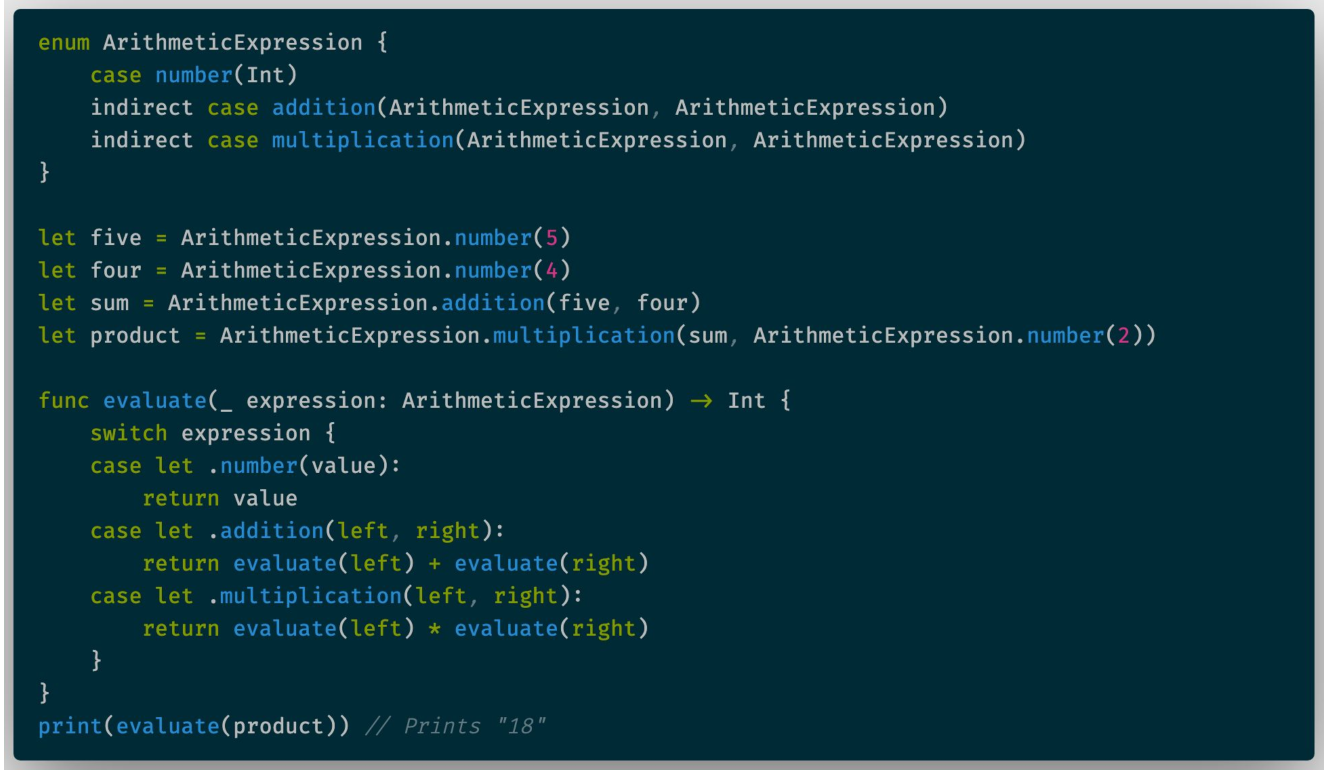
```
let possiblePlanet = Planet(rawValue: 4)
print(possiblePlanet ?? "Unknown planet")
```

6. Recursive Enumerations

- A recursive enumeration is an enumeration that has another instance of the enumeration as the associated value for one or more of the enumeration cases.

- You indicate that an enumeration case is recursive by writing `indirect` before it, which tells the compiler to insert the necessary layer of indirection.

6. Recursive Enumerations (cont)



```swift
enum ArithmeticExpression {
    case number(Int)
    indirect case addition(ArithmeticExpression, ArithmeticExpression)
    indirect case multiplication(ArithmeticExpression, ArithmeticExpression)
}

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"
```

Question & Answer?

THANK YOU

you're my cup<T>...
you're my cup<'t>...!