

教えそびれたこと

- switch文
- 列挙型
- タプル
- アクセス修飾子

switch文

値によって処理を変えたいときに使うのがswitch文です。

書き方としてはこんな感じです。

Swiftのswitch文は、上の条件式から評価していき、もし条件式に該当した場合はそれ以降のパターンはスキップされます。

次の項で示す列挙体と組み合わせると、switch文のパワーが発揮されると思います。

```
switch 条件式 {  
    case 条件1:  
    case 条件2:  
    case 条件3:  
    default: // どのパターンにも該当しなかった場合  
}
```

列挙型

定義のしかた

列挙型は複数の識別子をまとめるための型です。といってもなんのことかわかりにくいと思うので実例を示します。

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

構造体は、ケース同士が排他的である必要があります(ex. sundayでかつmondayであることは有り得ない)

各ケースについては、構造体の名前.ケース名のようにしてアクセスができます。

```
let day1 = Weekday.sunday

let day2:Weekday = .sunday // 型がわかっている場合は省略可能
```

switch文で条件分岐

switch文で条件分岐をすることもできます。switch文を書く際にSwiftがケースの**網羅性チェック**をしてくれます。もし列挙体に存在するケースをswitch文に記載しなかった場合はビルドエラーになります。

```
switch day2 {
case .sunday:
    print("日")
case .monday:
    print("月")
case .tuesday:
    print("火")
case .wednesday:
    print("水")
case .thursday:
    print("木")
case .friday:
    print("金")
case .saturday:
    print("土")
}
```

rawValue

列挙体のケースにはそれぞれ対応する値を持たせることができます。これをrawValueといいます。

rawValueに指定できる型はInt型,Double型,String型などです。

```
enum BloodType:Int {
    case ab = 0 // 他のケースは自動的に連番が振られる
    case a
    case b
    case o
}

let bloodType:BloodType = .b
print(bloodType.rawValue) // 2
```

列挙型ができること

列挙型はケースを列挙する以外にも以下のことができます

- イニシャライザを持つ

```
enum Weekday {
    case sunday
    case monday
    case tuesday
    case wednesday
    case thursday
    case friday
    case saturday

    init?(japaneseName:String) { // イニシャライザも持てる
        switch japaneseName {
            case "月":
                self = .monday
            case "火":
                self = .tuesday
            case "水":
                self = .wednesday
            case "木":
                self = .thursday
            case "金":
                self = .friday
            case "土":
                self = .saturday
            case "日":
                self = .sunday
            default: // どれにも該当しなかった場合
                return nil
        }
    }
}

let day3 = Weekday(japaneseName: "月")! // イニシャライザを使用した初期化
```

- computed propertyを持つ

```
enum Weekday {
    case sunday
    case monday
    case tuesday
    case wednesday
    case thursday
    case friday
    case saturday

    var name:String { // computed propertyも持てる
        switch self {
            case .sunday:
                return "日"
            case .monday:
                return "月"
        }
    }
}
```

```
        case .tuesday:
            return "火"
        case .wednesday:
            return "水"
        case .thursday:
            return "木"
        case .friday:
            return "金"
        case .saturday:
            return "土"
    }
}
}
let day2:Weekday = .wednesday // 型がわかっている場合は省略可能
print(day2.name)
```

- メソッドを持つ

メソッドを持つこともできます

- プロトコルに適合させる

プロトコルに適合させることもできます。よく列挙型と用いられがちなのがErrorプロトコルですね。

網羅性チェックのおかげで、エラーの場合分けをし忘れる心配がありません。

タプル

複数の型を束ねて使いたい場合、タプルを用いることがあります。

タプルは構造体と異なり、メソッドやプロパティを持つことはできません。

しかし、記述が簡潔であるため、構造体ほどたくさんの情報を渡す必要がない時などに重宝するかもしれません。

```
var tuple:(Int, String) // タプルを定義
tuple = (100, "hoge")
print(tuple.0) // 0番目の要素にアクセス(100)
```

各要素にラベルをつけることもできます。

```
let profile:(name:String, age:Int) = (name:"tanaka", age:25)
print(profile.name) // "tanaka"
```

アクセス修飾子

Javaでprivateとかstaticとかpublicとかfinalとかやりませんでしたか？アレのことです。