Pattern Matching, Part 4: if case, guard case, for case

May 16, 2016
Updated: Oct 17, 2016
7 minute read — Swift 3.0
Also available in: CN Chinese

Now that we've revisited the various syntaxes for pattern matching in part 1, part 2 and part 3, let's finish this blog post series with some advanced syntax using `if case let`, `for case where` and all!

Let's use what we saw in previous articles and apply them all to some advanced expressions.

> *This post is part of an article series. You can read all the parts here: part 1, part 2, part 3, part 4*

if case let

The `case let x = y` pattern allows you to check if `y` does match the pattern `x`.

Writing `if case let x = y { … }` is strictly equivalent to writing `switch y { case let x: … }`: it's just a more compact syntax which is useful when you only want to pattern-match against one case — as opposed to a `switch` which is more adapted to multiple cases matching.

For example, let's use an `enum` similar to the one from the previous articles:

```
enum Media {
  case book(title: String, author: String, year: Int)
  case movie(title: String, director: String, year: Int)
  case website(urlString: String)
}
```

Then we can write this[1]:

```
let m = Media.movie(title: "Captain America: Civil War", director: "Russo

if case let Media.movie(title, _, _) = m {
  print("This is a movie named \(title)")
}
```

This is equivalent to the more verbose code:

```
switch m {
  case let Media.movie(title, _, _):
    print("This is a movie named \(title)")
  default: () // do nothing, but this is mandatory as all switch in Swift
}
```

if case let where

We can combine the `if case let` with a comma (`,`) – where each condition is separated by `,` – to create a multi-clause condition:

```
  if case let Media.movie(_, _, year) = m, year < 1888 {
    print("Something seems wrong: the movie's year is before the first mov
  }
```

That can lead to quite powerful expressions that would otherwise need a complex `switch` and multiple lines only to test one specific case.

guard case let

Of course, `guard case let` is similar to `if case let`. You can use `guard case let` and `guard case let … , …` to ensure something matches a pattern and a condition and exit otherwise.

```
enum NetworkResponse {
  case response(URLResponse, Data)
  case error(Error)
}

func processRequestResponse(_ response: NetworkResponse) {
  guard case let .response(urlResp, data) = response,
    let httpResp = urlResp as? HTTPURLResponse,
    200..<300 ~= httpResp.statusCode else {
      print("Invalid response, can't process")
      return
  }
```

```
    print("Processing \(data.count) bytes…")
    /* … */
  }
```

for case

Combining `for` and `case` can also let you iterate on a collection conditionally. Using `for case …` is semantically similar to using a `for` loop and wrapping its whole body in an `if case` block: It will only iterate and process the elements that match the pattern.

```
let mediaList: [Media] = [
  .book(title: "Harry Potter and the Philosopher's Stone", author: "J.K. R
  .movie(title: "Harry Potter and the Philosopher's Stone", director: "Chr
  .book(title: "Harry Potter and the Chamber of Secrets", author: "J.K. Ro
  .movie(title: "Harry Potter and the Chamber of Secrets", director: "Chri
  .book(title: "Harry Potter and the Prisoner of Azkaban", author: "J.K. R
  .movie(title: "Harry Potter and the Prisoner of Azkaban", director: "Alf
  .movie(title: "J.K. Rowling: A Year in the Life", director: "James Runci
  .website(urlString: "https://en.wikipedia.org/wiki/List_of_Harry_Potter-
]

print("Movies only:")
for case let Media.movie(title, _, year) in mediaList {
  print(" - \(title) (\(year))")
}

/* Output:
Movies only:
 - Harry Potter and the Philosopher's Stone (2001)
 - Harry Potter and the Chamber of Secrets (2002)
 - Harry Potter and the Prisoner of Azkaban (2004)
 - J.K. Rowling: A Year in the Life (2007)
*/
```

for case where

Adding a `where` clause to that all can make it even more powerful:

```
print("Movies by C. Columbus only:")
for case let Media.movie(title, director, year) in mediaList where directo
  print(" - \(title) (\(year))")
}

/* Output:
Movies by C. Columbus only:
 - Harry Potter and the Philosopher's Stone (2001)
 - Harry Potter and the Chamber of Secrets (2002)
*/
```

Note: Using `for … where` without the `case` pattern matching part is also a valid Swift syntax. For example you can write:

```
for m in listOfMovies where m.year > 2000 { … }
```

It is not using pattern matching (no `case` nor `~=`) so that's a bit out of the scope of this article series, but it's still totally valid and as useful as the other constructs presented here — as it avoids wrapping the whole body of your `for` within a big `if` (or starting it with a `guard … else { continue }`).

Combining them all

Let's finish this series with the Grand Finale: combine all that we learned from the beginning (including some syntactic sugar like `x?` we learned in the previous article):

```
extension Media {
  var title: String? {
    switch self {
    case let .book(title, _, _): return title
    case let .movie(title, _, _): return title
    default: return nil
    }
  }
  var kind: String {
    // Remember part 1 where we said we can omit the `(…)` associated valu
    switch self {
    case .book: return "Book"
    case .movie: return "Movie"
    case .website: return "Web Site"
    }
  }
}

print("All mediums with a title starting with 'Harry Potter'")
for case let (title?, kind) in mediaList.map({ ($0.title, $0.kind) })
  where title.hasPrefix("Harry Potter") {
    print(" - [\(kind)] \(title)")
}
```

This look might look a little complex, so let's split it down:

- It uses `map` to transform the `Array<Media>` array `mediaList` into an array of tuples `[(String?, String)]` containing the title (if any) + the kind of item (as text)
- It only matches if `title?` matches — which is syntactic sugar to say if `.Some(title)` matches — the `$0.title` of each media. This means that it discards any

media for which `$0.title` returns `nil` (a.k.a. `Optional.None`) — excluding any `WebSite` in the process, as those don't have any `title`)
- Then it filters the results to only iterate on those for which `title.hasPrefix("Harry Potter")` is true.

So in the end this will loop on every medium that has a title starting with "Harry Potter", discarding any medium that don't have a title — like `WebSite` — as well as any medium having a title that doesn't start with `"Harry Potter"` — excluding the J.K. Rowling documentary from that iteration as well.

The code will thus output this, only listing Harry Potter books and movies:

```
All medium with a title starting with 'Harry Potter'
 - [Book] Harry Potter and the Philosopher's Stone
 - [Movie] Harry Potter and the Philosopher's Stone
 - [Book] Harry Potter and the Chamber of Secrets
 - [Movie] Harry Potter and the Chamber of Secrets
 - [Book] Harry Potter and the Prisoner of Azkaban
 - [Movie] Harry Potter and the Prisoner of Azkaban
```

Using neither pattern matching nor any `where` clause nor syntactic sugar that we learned in this article series, the code might have looked like this instead:

```
print("All mediums with a title and starting with 'Harry Potter'")
for media in mediaList {
  guard let title = media.title else {
    continue
  }
  guard title.hasPrefix("Harry Potter") else {
    continue
  }
  print(" - [\(media.kind)] \(title)")
}
```

Some might find it more readable, but you can't argue that using `for case let (title?, kind) in … where …` is really powerful and allow you to ~~impress your friends~~ make great use of for loops + pattern matching + `where` clauses altogether. ✨

Conclusion

This is the end of this "Pattern Matching" series. Hope you enjoyed it and learned some interesting stuff ☺

Next articles will be more focused back on some nice Swifty design patterns and architecture than on Swift syntax and language.

Don't hesitate to tell me on Twitter if you have any particular subject on Swift you want me blog about and give me some ideas for what to write about next!

> *Thanks to Frank Manno for updating the code samples of this article to Swift 3!*

---

1. The order of arguments (pattern vs. variable) in that syntax can be troubling. To remember the order, just think of it as using the same `case let Media.movie(…)` syntax you use in a `switch`. That way, you'll remember to write `if case let Media.movie(…) = m` instead of `if case let m = Media.movie(…)` which wouldn't compile anyway — grouping the `case` with the **pattern** (`Media.movie(title, _, _)`) like you do in `switch`, and not with the variable to compare it to (`m`). ↩

Share this post:

Theme crafted with <3 by John Otander (@4lpine). — </> available on Github.