Chapter 12: Switch

Parameter	Details
Value to test	The variable that to compare against

Section 12.1: Switch and Optionals

Some example cases when the result is an optional.

```
var result: AnyObject? = someMethod()

switch result {
case nil:
    print("result is nothing")
case is String:
    print("result is a String")
case _ as Double:
    print("result is not nil, any value that is a Double")
case let myInt as Int where myInt > 0:
    print("\(myInt) value is not nil but an int and greater than 0")
case let a?:
    print("\(a) - value is unwrapped")
}
```

Section 12.2: Basic Use

```
let number = 3
switch number {
case 1:
    print("One!")
case 2:
    print("Two!")
case 3:
    print("Three!")
default:
    print("Not One, Two or Three")
}
```

switch statements also work with data types other than integers. They work with any data type.Here's an example of switching on a string:

```
let string = "Dog"
switch string {
case "Cat", "Dog":
  print("Animal is a house pet.")
default:
  print("Animal is not a house pet.")
}
```

This will print the following:

```
Animal is a house pet.
```

Section 12.3: Matching a Range

A single case in a switch statement can match a range of values.

```
let number = 20
switch number {
case 0:
    print("Zero")
case 1..<10:
    print("Between One and Ten")
case 10..<20:
    print("Between Ten and Twenty")
case 20..<30:
    print("Between Twenty and Thirty")
default:
    print("Greater than Thirty or less than Zero")
}
```

Section 12.4: Partial matching

Switch statement make use of partial matching.

```
let coordinates: (x: Int, y: Int, z: Int) = (3, 2, 5)

switch (coordinates) {
case (0, 0, 0): // 1
  print("Origin")
case (_, 0, 0): // 2
  print("On the x-axis.")
case (0, _, 0): // 3
  print("On the y-axis.")
case (0, 0, _): // 4
  print("On the z-axis.")
default:        // 5
  print("Somewhere in space")
}
```

1. Matches precisely the case where the value is (0,0,0). This is the origin of 3D space.
2. Matches y=0, z=0 and any value of x. This means the coordinate is on the x- axis.
3. Matches x=0, z=0 and any value of y. This means the coordinate is on they- axis.
4. Matches x=0, y=0 and any value of z. This means the coordinate is on the z- axis.
5. Matches the remainder of coordinates.

Note: using the underscore to mean that you don't care about the value.

If you don't want to ignore the value, then you can use it in your switch statement, like this:

```
let coordinates: (x: Int, y: Int, z: Int) = (3, 2, 5)

switch (coordinates) {
case (0, 0, 0):
  print("Origin")
case (let x, 0, 0):
  print("On the x-axis at x = \(x)")
case (0, let y, 0):
  print("On the y-axis at y = \(y)")
case (0, 0, let z):
  print("On the z-axis at z = \(z)")
case (let x, let y, let z):
  print("Somewhere in space at x = \(x), y = \(y), z = \(z)")
}
```

Here, the axis cases use the let syntax to pull out the pertinent values. The code then prints the values using string

interpolation to build the string.

Note: you don't need a default in this switch statement. This is because the final case is essentially the default—it matches anything, because there are no constraints on any part of the tuple. If the switch statement exhausts all possible values with its cases, then no default is necessary.

We can also use the let-where syntax to match more complex cases. For example:

```
let coordinates: (x: Int, y: Int, z: Int) = (3, 2, 5)

switch (coordinates) {
case (let x, let y, _) where y == x:
  print("Along the y = x line.")
case (let x, let y, _) where y == x * x:
  print("Along the y = x^2 line.")
default:
break
}
```

Here, We match the "y equals x" and "y equals x squared" lines.

Section 12.5: Using the where statement in a switch

The where statement may be used within a switch case match to add additional criteria required for a positive match. The following example checks not only for the range, but also if the number is odd or even:

```
switch (temperature) {
      case 0...49 where temperature % 2 == 0:
        print("Cold and even")

      case 50...79 where temperature % 2 == 0:
        print("Warm and even")

      case 80...110 where temperature % 2 == 0:
        print("Hot and even")

      default:
        print("Temperature out of range or odd")
}
```

Section 12.6: Matching Multiple Values

A single case in a switch statement can match on multiple values.

```
let number = 3
switch number {
case 1, 2:
    print("One or Two!")
case 3:
    print("Three!")
case 4, 5, 6:
    print("Four, Five or Six!")
default:
    print("Not One, Two, Three, Four, Five or Six")
}
```

Section 12.7: Switch and Enums

The Switch statement works very well with Enum values

```
enum CarModel {
    case Standard, Fast, VeryFast
}

let car = CarModel.Standard

switch car {
case .Standard: print("Standard")
case .Fast: print("Fast")
case .VeryFast: print("VeryFast")
}
```

> Since we provided a case for each possible value of car, we omit the `default` case.

Section 12.8: Switches and tuples

Switches can switch on tuples:

```
public typealias mdyTuple = (month: Int, day: Int, year: Int)

let fredsBirthday =   (month: 4, day: 3, year: 1973)




switch theMDY
{
//You can match on a literal tuple:
case (fredsBirthday):
  message = "\(date) \(prefix) the day Fred was born"

//You can match on some of the terms, and ignore others:
case (3, 15, _):
  message = "Beware the Ides of March"

//You can match on parts of a literal tuple, and copy other elements
//into a constant that you use in the body of the case:
case (bobsBirthday.month, bobsBirthday.day, let year) where year > bobsBirthday.year:
  message = "\(date) \(prefix) Bob's \(possessiveNumber(year - bobsBirthday.year))" +
    "birthday"

//You can copy one or more elements of the tuple into a constant and then
//add a where clause that further qualifies the case:
case (susansBirthday.month, susansBirthday.day, let year)
  where year > susansBirthday.year:
  message = "\(date) \(prefix) Susan's " +
    "\(possessiveNumber(year - susansBirthday.year)) birthday"

//You can match some elements to ranges:.
case (5, 1...15, let year):
  message = "\(date) \(prefix) in the first half of May, \(year)"
}
```

Section 12.9: Satisfy one of multiple constraints using switch

You can create a tuple and use a switch like so:

```
var str: String? = "hi"
var x: Int? = 5

switch (str, x) {
case (.Some,.Some):
    print("Both have values")
case (.Some, nil):
    print("String has a value")
case (nil, .Some):
    print("Int has a value")
case (nil, nil):
    print("Neither have values")
}
```

Section 12.10: Matching based on class - great for prepareForSegue

You can also make a switch statement switch based on the **class** of the thing you're switching on.

An example where this is useful is in `prepareForSegue`. I used to switch based on the segue identifier, but that's fragile. if you change your storyboard later and rename the segue identifier, it breaks your code. Or, if you use segues to multiple instances same view controller class (but different storyboard scenes) then you can't use the segue identifier to figure out the class of the destination.

Swift switch statements to the rescue.

Use Swift `case let var as Class` syntax, like this:

Version < 3.0

```
override func prepareForSegue(segue: UIStoryboardSegue, sender: AnyObject?) {
  switch segue.destinationViewController {
    case let fooViewController as FooViewController:
      fooViewController.delegate = self

    case let barViewController as BarViewController:
      barViewController.data = data

    default:
      break
  }
}
```

Version ≥ 3.0

In Swift 3 the sytax has changed slightly:

```
override func prepare(for segue: UIStoryboardSegue, sender: AnyObject?) {
  switch segue.destinationViewController {
    case let fooViewController as FooViewController:
      fooViewController.delegate = self

    case let barViewController as BarViewController:
      barViewController.data = data

    default:
```

```
        break
    }
}
```

Section 12.11: Switch fallthroughs

It is worth noting that in swift, unlike other languages people are familiar with, there is an implicit break at the end of each case statement. In order to follow through to the next case (i.e. have multiple cases execute) you need to use `fallthrough` statement.

```
switch(value) {
case 'one':
    // do operation one
    fallthrough
case 'two':
    // do this either independant, or in conjunction with first case
default:
    // default operation
}
```

this is useful for things like streams.