SCIENCES SORBONNE UNIVERSITÉ

Swift, tuples and filtering

Fabrice.Kordon@lip6.fr

LIP6

As an introduction…

Nice to have all these together

- Already experimented in some programming languages

Tuple = grouping values together

- Similar to Ada aggregates
  - Already some similar form with dictionaries

```
let dico = ["aa":1, "bb":2]
for (key,value) in dico {
    print("key: \(key), value:\(value)")
}
```

Filtering

- Similar to OCaml «pattern matching»

Tuples by the example…

Tuple construction

```
let http404pb = (404, "page not found")
print ("Let's have a look at this error : «\(http404pb)»")
```

Tuples by the example...

Tuple construction

```
let http404pb = (404, "page not found")
print ("Let's have a look at this error : «\(http404pb)»")
```

Dispatching tuple elements

```
let (code, msg) = http404pb
print ("code     = \(code)")
print ("message  = \(msg)")
```

Tuples by the example...

Tuple construction

```
let http404pb = (404, "page not found")
print ("Let's have a look at this error : «\(http404pb)»")
```

Dispatching tuple elements

```
let (code, msg) = http404pb
print ("code     = \(code)")
print ("message  = \(msg)")
```

Retrieving Tuples elements

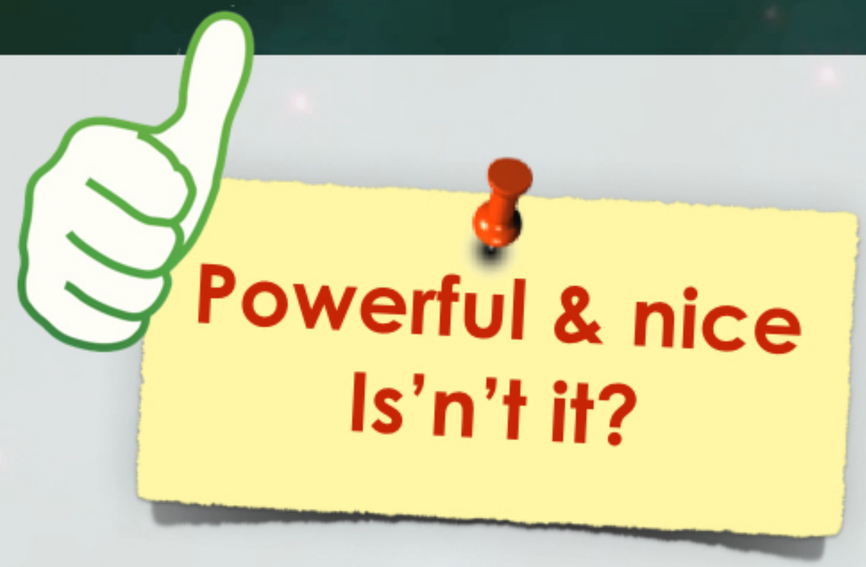
```
print ("Only the message = \(http404pb.1)")
print ("Only the code    = \(http404pb.0)")

let http404pb2 = (code : 404, desc : "page not found")
print ("Only the code    = \(http404pb2.code)")
print ("Only the message = \(http404pb2.desc)")
```

Tuples and filtering (first look)


```swift
func localise (point : (Int, Int)) -> String {
    switch point {
    case (0, 0):
        return ("(0, 0) is the origin")
    case (_, 0):
        return ("(\(point.0), 0) is on the X axis")
    case (0, _):
        return ("(0, \(point.1)) is on the Y axis")
    case (-2...2, -2...2):
        return ("(\(point.0), \(point.1)) is in the box")
    default:
        return ("(\(point.0), \(point.1)) is out of the box")
    }
}

let pt0 = (0, 0)
let pt1 = (1, 1)
let pt2 = (1, 0)
let pt3 = (0, 1)
let pt4 = (3, 1)
let pt5 = (3, 0)

print ("\(pt0) => \(localise(point:pt0))")
print ("\(pt1) => \(localise(point:pt1))")
print ("\(pt2) => \(localise(point:pt2))")
print ("\(pt3) => \(localise(point:pt3))")
print ("\(pt4) => \(localise(point:pt4))")
print ("\(pt5) => \(localise(point:pt5))")
```

Tuples and filtering (first look)


```swift
func localise (point : (Int, Int)) -> String {
    switch point {
    case (0, 0):
        return ("(0, 0) is the origin")
    case (_, 0):
        return ("(\(point.0), 0) is on the X axis")
    case (0, _):
        return ("(0, \(point.1)) is on the Y axis")
    case (-2...2, -2...2):
        return ("(\(point.0), \(point.1)) is in the box")
    default:
        return ("(\(point.0), \(point.1)) is out of the box")
    }
}

let pt0 = (0, 0)
let pt1 = (1, 1)
let pt2 = (1, 0)
let pt3 = (0, 1)
let pt4 = (3, 1)
let pt5 = (3, 0)

print ("\(pt0) => \(localise(point:pt0))")
print ("\(pt1) => \(localise(point:pt1))")
print ("\(pt2) => \(localise(point:pt2))")
print ("\(pt3) => \(localise(point:pt3))")
print ("\(pt4) => \(localise(point:pt4))")
print ("\(pt5) => \(localise(point:pt5))")
```




```
(0, 0) => (0, 0) is the origin
(1, 1) => (1, 1) is in the box
(1, 0) => (1, 0) is on the X axis
(0, 1) => (0, 1) is on the Y axis
(3, 1) => (3, 1) is out of the box
(3, 0) => (3, 0) is on the X axis
```

More on tuples & filtering


```swift
func diagonales (point : (Int, Int)) -> String {
    switch point {
    case let (x, y) where x == y:
        return ("(\(x), \(y)) is on the diagonal")
    case let (x, y) where x == -y:
        return ("(\(x), \(y)) is on the other diagonal")
    case let (x, y):
        return ("(\(x), \(y)) is not on a diagonal")
    }
}

let pt6 = (3, -3)
let pt7 = (2, 2)
let pt8 = (3, 1);

print ("\(pt6) => \(diagonales(point:pt6))")
print ("\(pt7) => \(diagonales(point:pt7))")
print ("\(pt8) => \(diagonales(point:pt8))")
```

```
(3, -3) => (3, -3) is on the other diagonal
(2, 2) => (2, 2) is on the diagonal
(3, 1) => (3, 1) is not on a diagonal
```

As a conclusion…

- **This is only a glance on it**
  - Numerous possibilities
- **So, I strongly recommend you to RTFM!**