SCIENCES SORBONNE UNIVERSITÉ

Swift, basic types

Fabrice.Kordon@lip6.fr

LIP6

As an introduction…

You already know these types

- String, Int, Double, Bool

Let's have a quick look on

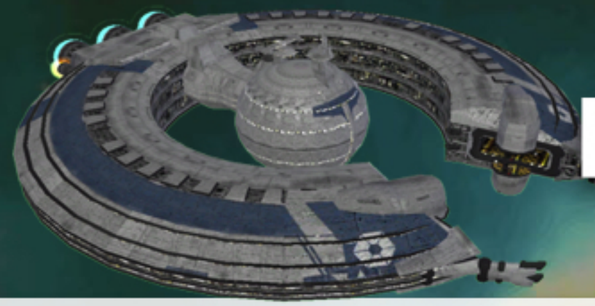
- Enumerated types
- Arrays and Dictionaries

Immutable, Mutable?

- keywords let/var
  - **Be aware that concerns variables!**

Key notion of «optional»

- «type» version «type?»
  - **Just a first glance in this video**



Enumerated types, Episode I

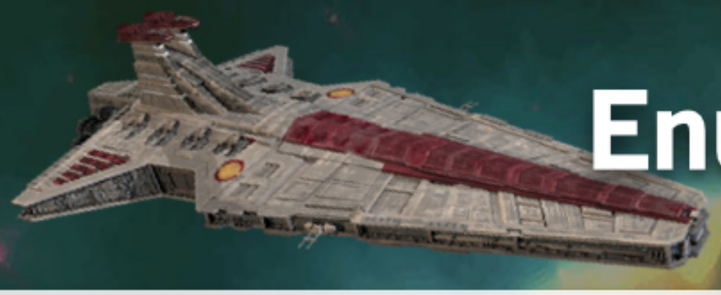
Seen as a series of values

```
enum Suite {case club, diamond, heart, spade}

let cardType1 = club
let cardType2 = Suite.diamond

if cardType1 == Suite.diamond {
    print ("It is a Diamond");
} else {
    print ("It is something else");
}
print ("Here is one : \(cardType1), and a second : \(cardType2)")
```

```
It is something else
Here is one : club, and a second : diamond
```



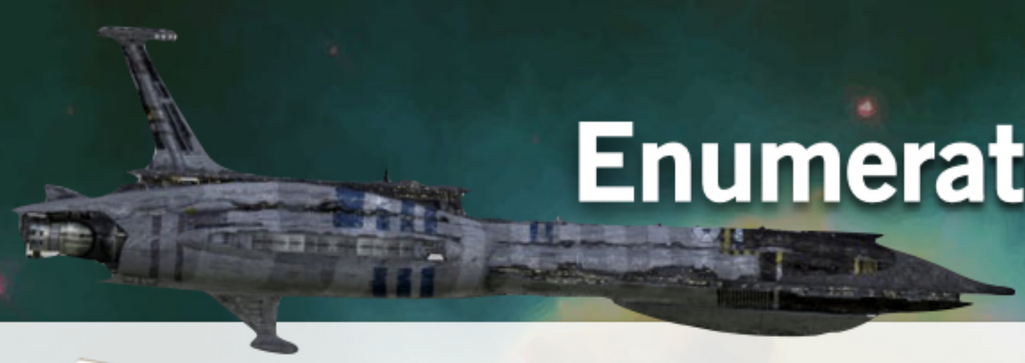
Enumerated types, Episode II

A series of values… potentially associated to functions

```
enum Suite {case club, diamond, heart, spade
    func image() -> String {
        switch self {
        case .clover:
            return "Club (black)"
        case .heart:
            return "Heart (red)"
        case .spade:
            return "Spade (black)"
        case .tile:
            return "Diamond (red)"
        }
    }
}

let v1 = Suite.club
let v2 = Suite.diamond
print ("Here is one : \(v1.image()), and a second : \(v2.image())")
```

```
Here is one : Club (black), and a second : Diamond (red)
```



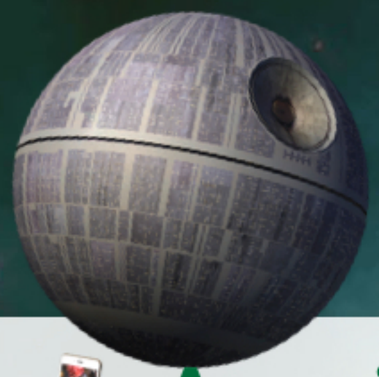
Enumerated types, Episode III

A series of values... connected to Int

```
enum Suite : Int {case club = 1, diamond = 2, heart = 3, spade = 4}

let v1 = Suite.club
print ("here is one : \(v1)")
let v2 = Suite.heart
print ("\(v2) is the \(String(describing: v2.rawValue))rd value...")
let pos = 5
let v3 = Suite(rawValue:pos) // may be nil if pos is too high
if  v3 == Suite.spade {
    print("Spade is the 5th color")
} else {
    print("Oups!!!") // catches also "nil"
}
```

```
here is one : club
heart is the 3rd value...
Oups!!!
```



Enumerated types, Episode IV

A series of values... connected to String

```
enum Suite : String {case club = "C", diamond = "D", heart = "H", spade = "S"}

let v1 = Suite.club
print ("here is one : \(v1)")
let v2 = Suite.heart
print ("\(v2) is represented by character \(String(describing: v2.rawValue))...")
let pos = 5
let v3 = Suite(rawValue:"W") // may be nil if pos is wrong
if  v3 == Suite.spade {
    print("Spade is the W color")
} else {
    print("Oups!!!") // catches also "nil"
}
```

```
here is one : club
heart is represented by character H...
Oups!!!
```

Arrays


```swift
var tab = ["Titi", "Rominet", "Grandma"]
tab.append("Brutus") // impossible with "let"
print("tab = \(tab)")
tab.remove(at: 1) // impossible with "let"
if tab.isEmpty {
    print("Empty array")
} else {
    print("tab = \(tab)")
}
print ("The dog is called \(tab[2])")
```

```
tab = ["Titi", "Rominet", "Grandma", "Brutus"]
tab = ["Titi", "Grandma", "Brutus"]
The dog is called Brutus
```

Arrays iterators


```swift
var tab2 = [Int]()
tab2.append(0)
tab2[0] = 1  // impossible with "let"
tab2 += [3, 5, 7, 11] // impossible with "let"
print("Let's see \(tab2.count) prime numbers: \(tab2)")

var cpt = 0
for v in tab2 {
    print("tab3[\(cpt)] = \(v)")
    cpt+=1
}
```

```
Let's see 5 prime numbers: [1, 3, 5, 7, 11]
tab3[0] = 1
tab3[1] = 3
tab3[2] = 5
tab3[3] = 7
tab3[4] = 11
```

Dictionaries


```swift
var cardsColors = [ "Club" : "Black",
                    "Diamond" : "Red",
                    "Heart" : "Red",
                    "Spade": "Red"]
print("Dictionary has \(cardsColors.count) elemments : \(cardsColors)")
cardsColors["Spade"] = "Black" // correcting a small mistake ;-)
print("Dictionary = \(cardsColors)")
```

```
Dictionary has 4 elemments : ["Club": "Black", "Heart": "Red", "Spade": "Red", "Diamond":
"Red"]
Dictionary = ["Club": "Black", "Heart": "Red", "Spade": "Black", "Diamond": "Red"]
```

Dictionaries and iterators


```swift
var cardsColors = ["Club" : "Black","Diamond" : "Red","Heart" : "Red","Spade": "Black"]

for (key,value) in cardsColors
{
    print("key: \(key) value:\(value)")
}
for key in cardsColors.keys {
    print("Key: \(key)")
}
for values in cardsColors.values {
    print("Value: \(values)")
}
let s1 = cardsColors["Club"]
let s2 = cardsColors["Toto"]
print("s1 = \(String(describing: s1)) et s2 = \(String(describing: s2))")
print("s1 = \(s1 ?? "none") et s2 = \(s2 ?? "none")")
```

```
key: Club value:Black
key: Heart value:Red
key: Spade value:Black
key: Diamond value:Red
Key: Club
Key: Heart
Key: Spade
Key: Diamond
Value: Black
Value: Red
Value: Black
Value: Red
s1 = Optional("Black") et s2 = nil
s1 = Black et s2 = none
```

As a conclusion…

- This is useful stuff
- This is a first approach to Swift facilities
- Play with the «playground»
- More to discover in the RTFM!