Swift

ОСНОВЫ

Королев Юрий
AnjLab
г. Владимир

Enum

```
enum Example {
    case A
    case B
    case C
    case D
}
```

Enum и ассоциированные значения

```
enum JSValue {
    case Obj([String: JSValue])
    case Num(Number)
    case Arr([JSValue])
    case Str(String)
    case Null
    case True
    case False
}

// Example
let jsonString = JSValue.Str(“Hello”)
```

Как работать - Pattern matching

```
switch jsonString {
case let .Str(value):
    print(value)
default: break
}

// Swift 2.0 или 1.2 - не важно :)
if case let .Str(value) = jsonString {
    print(value)
}
```

Generic Enums

```
enum Box<T> {
    case empty
    case filledWith(T)
}

let emptyBox:Box<Int> = Box.empty
let filledBox:Box<Int> = Box.filledWith(5)
```

значения не могут быть nil

Проверка на этапе компиляции

```
// Swift
func alert(message: String) {
    // No need to check message is nil
    // Do ...
}

// C#
void Alert(String message) {
    if (message == null)
        throw new ArgumentNullException("message");

    // Do ...
}
```

nil - это не значение, это литерал

```
// Wrapped!
enum ImplicitlyUnwrappedOptional<Wrapped> :
NilLiteralConvertible {
    case None
    case Some(Wrapped)
}

// Wrapped?
enum Optional<Wrapped> : NilLiteralConvertible {
    case None
    case Some(Wrapped)
}
```

Optionals примеры

```
let o1:Optional<String> = Optional.Some("nice")
let o2:Optional<String> = .Some("nice")
let o3 = Optional.Some("nice")
let o4:String? = Optional.Some("nice")
let o5:String? = .Some("nice")
let o6:String? = "nice"


let o7:Optional<String> = Optional.None
let o8:Optional<String> = .None
let o10:String? = Optional.None
let o11:String? = .None
let o12:String? = nil
```

Optionals примеры

```
let o1:ImplicitlyUnwrappedOptional<String> = ImplicitlyUnwrappedOptional.Some("nice")
let o2:ImplicitlyUnwrappedOptional<String> = .Some("nice")
let o3 = ImplicitlyUnwrappedOptional.Some("nice")
let o4:String! = ImplicitlyUnwrappedOptional.Some("nice")
let o5:String! = .Some("nice")
let o6:String! = "nice"
```

```
let o7:ImplicitlyUnwrappedOptional<String> = Optional.None
let o8:ImplicitlyUnwrappedOptional<String> = .None
let o10:String! = ImplicitlyUnwrappedOptional.None
let o11:String! = .None
let o12:String! = nil
```

Optional может быть любой тип

Ошибки

```
func foo() -> Int {
    return 10
}
```

Ошибки

```
enum Result<T> {
    case ok(T)
    case error(NSError)
}
```

```
func foo() -> Result<Int> {
    // Ok
    return Result.ok(10)
    // Error
    return Result.error(NSError())
}
```

Работа с ошибками в swift

```
func foo() -> Result<Int> {
    // Ok
    return Result.ok(10)
    // Error
    return Result.error(NSError())
}
```

```
func foo() throws -> Int {
    // Ok
    return 10
    // Error
    throw NSError()
}
```

Вызов методов

```
// C#
try {
  foo();
  bla();
}
catch (Exception exception) {
   // …
}
```

```
// Swift
do {
  try foo()
  bla()
}
catch let error {
   // …
}
```

Вызов методов try?

```
let x = try? foo() // x:Int?
```

```
let x:Int?
if case let Result.ok(value) = foo() {
    x = value
} else {
    x = nil
}
```

Вызов методов try!

```
let x = try! foo() // x:Int or crash
```

```
let x: Int
let result = foo()
switch result {
case let .ok(value):
    x = value
case let .error(error):
    fatalError(error) // примерно так :)
}
```

Конец

Вопросы?