Laziness in Swift

Maciej Konieczny
narf.pl · macoscope.com

narf.pl

macoscope

SwiftWarsaw.com

Python 😍

Django 😋

JavaScript 😩

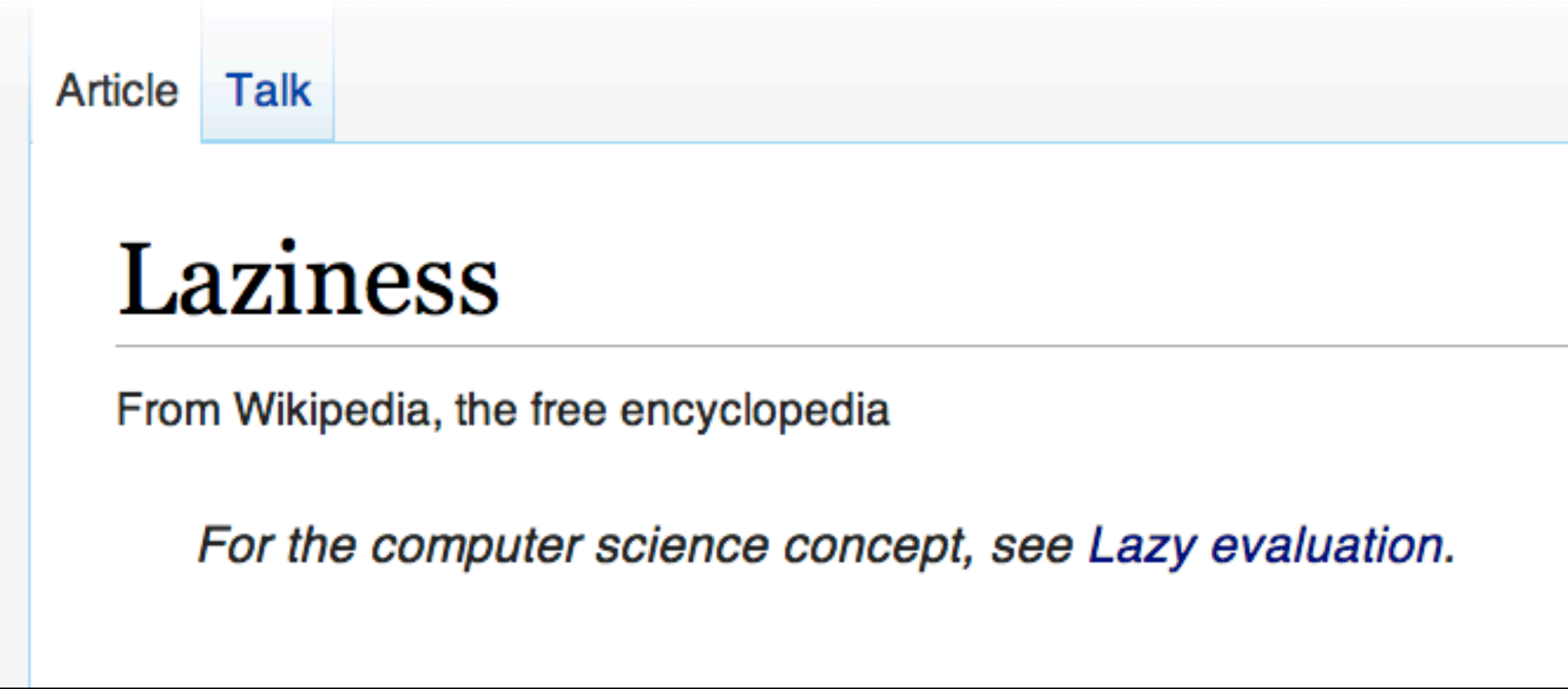
CoffeeScript 😉

Objective-C 😕

Swift 😮

Python 😍

Laziness in Swift



Laziness

From Wikipedia, the free encyclopedia

For the computer science concept, see Lazy evaluation.

delaying computation until necessary

never necessary
never computed

potential for
removing
needless computation

potential for reducing memory footprint

potential for infinite data structures

Laziness allows the expression of programs that would otherwise not terminate not one pattern

Swift

```
lazy var
```

SequenceType

@autoclosure

`lazy var`

```swift
class BlogPost {
    var filename: String

    init(filename: String) {
        self.filename = filename
    }
}
```

```
class BlogPost {
    var filename: String
    var foo = Foo()

    init(filename: String) {
        self.filename = filename
    }
}
```

```
class BlogPost {
    var filename: String
    lazy var foo = Foo()

    init(filename: String) {
        self.filename = filename
    }
}
```

```
class BlogPost {
    var filename: String

    init(filename: String) {
        self.filename = filename
    }
}
```

```
class BlogPost {
    var filename: String
    lazy var markdown: String = {
        markdownForFile(self.filename)
    }()

    init(filename: String) {
        self.filename = filename
    }
}
```

```
Swift.nil != ObjC.nil
```

```
- (NSString *)markdown {
    if (!_markdown) {
        _markdown = markdownForFile(self.filename);
    }

    return _markdown;
}
```

SequenceType

```
for x in xs {
    // ...
}
```

```
for x in xs {
    // ...
}

var _g = xs.generate()
while let x = _g.next() {
    // ...
}
```

```
class Integers: SequenceType {
    func generate() -> GeneratorOf<Int> {
        var n = -1
        return GeneratorOf { ++n }
    }
}
```

```
class Integers: SequenceType {
    func generate() -> GeneratorOf<Int> {
        var n = -1
        return GeneratorOf { ++n }
    }
}

for i in Integers() {
    println(i)  // 0, 1, 2, 3, ...
}
```

```
var integers = lazy(Integers())
var integers = lazy(Integers())

integers.filter
integers.map
```

```
extension LazySequence {
    var first: LazySequence.Generator.Element? {
        for x in self {
            return x
        }

        return nil
    }
}

integers.first!  // 0
```

```
var x = integers
var x = integers \
    .filter { $0 % 2 == 1 } \
var x = integers \
    .filter { $0 % 2 == 1 } \
    .map { $0 * $0 }
```

```
var x = integers \
    .filter { $0 % 2 == 1 } \
    .map { $0 * $0 } \
    .filter { $0 > 100 } \
var x = integers \
    .filter { $0 % 2 == 1 } \
    .map { $0 * $0 } \
    .filter { $0 > 100 } \
    .first!
```

```
var x = integers \
    .filter { $0 % 2 == 1 } \
    .map { $0 * $0 } \
    .filter { $0 > 100 } \
    .first!

println(x)  // 121
```

```
var x = integers.filter {
    return $0 % 2 == 1
}.map {
    return $0 * $0
}.filter {
    return $0 > 10
}.first!

println(x)  // 25
```

```
var x = integers.filter {
    println("\n\($0)")
    println("even?")
    return $0 % 2 == 1
}.map {
    println("square")
    return $0 * $0
}.filter {
    println("threshold")
    return $0 > 10
}.first!

println(x)  // 25
```

```
integers.filter { $0 % 2 == 1 } \
       .map { $0 * $0 }
       .filter { $0 > 10 } \
       .first!

0 even?
1 even? square threshold
2 even?
3 even? square threshold
4 even?
5 even? square threshold
```

@autoclosure

```
// without @autoclosure:
f({ x })

// with @autoclosure:
f(x)
```

```
func foo(bar: () -> ()) {
    bar()
}

foo({ println("baz") })
```

```
func foo(bar: @autoclosure () -> ()) {
    bar()
}

foo(println("baz"))
```

while not x / until x / dopóki x

```
func dopóki(condition: @autoclosure () -> Bool,
            body: () -> ()) {
    if !condition() {
        body()
        dopóki(condition(), body)
    }
}


var i = 3

dopóki (i == 0) {
    println(i)
    i -= 1
}
```

BTW: compiler performs tail call optimisation

not one pattern

removing needless computation

reducing memory footprint

expressiveness

```
lazy var foo = Foo()
lazy var markdown: String = {
    markdownForFile(self.filename)
}()
```

```
for x in xs {
    // ...
}

var _g = xs.generate()
while let x = _g.next() {
    // ...
}
```

```
// without @autoclosure:
f({ x })

// with @autoclosure:
f(x)
```

That's all folks!

narf.pl

macoscope

References (1 of 2)

- *Understand and implement laziness*, Matt Might http://matt.might.net/articles/implementing-laziness/
- *WWDC 2014, Session 404: Advanced Swift* https://developer.apple.com/videos/wwdc/2014/

References (2 of 2)

- *Lazy by name, lazy by nature*, airspeedvelocity http://airspeedvelocity.net/2014/07/26/lazy-by-name-lazy-by-nature/
- */r/aww* http://www.panoptikos.com/r/aww/top

Questions?