Enums

```swift
enum User {
  case Anonymous
  case Registered
}

let user = User.Registered
```

```
enum User {
  case Anonymous
  case Registered(name : String)
}

let user = User.Registered(name: "Ulrik")
```

```
@interface User : NSObject

// Might have no name!!!1
@property (strong, nonatomic) NSString *name;

@end
```

```objc
- (void)greetUser:(User *)user {
  NSLog(@"Hello, %@", user.name);
}
```

```
func greet(user : User) {
  println("Hello, \(user.name)")
}
```

User doesn’t have a name

```
func greet(user : User) {
  switch user {
    case .Anonymous:
      println("Hello, anon")
    case .Registered(let name):
      println("Hello, \(name)")
  }
}
```

```swift
func doStuff(completion :
  (data : NSData?, error : NSError?) -> Void)
```

```swift
func doStuff(completion :
  (data : NSData?, error : NSError?) -> Void) {
  completion(data: nil, error: nil)
}
```

```
enum Result {
  case Data(NSData)
  case Error(NSError)
}
```

```swift
enum Result {
  case Data(NSData)
  case Error(NSError)
}

func doStuff(completion : Result -> Void) {


}
```

```
enum Result {
  case Data(NSData)
  case Error(NSError)
}

func doStuff(completion : Result -> Void) {
  let error = createError()
  completion(.Error(error))
}
```

```swift
struct User {
  let name : String
  let userId : Int
  let info : Info

  enum Type {
  case Local(email : String, session : String)
  case Remote(isFriend : Bool)
  }
}
```

Optionals

```swift
let a : String = nil                     Nope

let b : String? = nil

b.capitalizedString                      Nope

b?.capitalizedString ?? "empty"
```

```swift
enum Optional<T> : NilLiteralConvertible {
    case None
    case Some(T)
}
```

```swift
if let b = b {
  let str = b.capitalizedString
}

let c : String! = nil

c.capitalizedString
```

Careful!

```
let userId : Int = 0 // No user
```

```swift
let userId : Int? = nil // No user
```

```swift
class View : UIView {
  let title : String
}
```

```
class View : UIView {
  let title : String

  required init(coder aDecoder : NSCoder) {
        You need to set title to something

  }
}
```

```
class View : UIView {
  let title : String

  required init(coder aDecoder : NSCoder) {
    // ಠ_ಠ
    title = "Happy now?"
  }
}
```

```
class View : UIView {
  let title : String!

  required init(coder aDecoder : NSCoder) {


  }
}
```

Careful!

Lazyness

```swift
class View : UIView {
  let button : UIButton

  override init() {
    button = setupButton()
    super.init()
  }

  func setupButton() -> UIButton {
    let button = UIButton(frame: CGRectZero)
    button.setTitle("Hello", forState: .Normal)
    return button
  }
}
```

self used before super.init call

```
class View : UIView {
  lazy var button : UIButton =
     self.setupButton()

  override init() {
    super.init()
  }

  func setupButton() -> UIButton {
    let button = UIButton(frame: CGRectZero)
    button.setTitle("Hello", forState: .Normal)
    return button
  }
}
```

```
class View : UIView {
  lazy var button : UIButton =
    self.setupButton()






  func setupButton() -> UIButton {
    let button = UIButton(frame: CGRectZero)
    button.setTitle("Hello", forState: .Normal)
    return button
  }
}
```

```swift
class View : UIView {
  lazy var button : UIButton =
     self.setupButton()

  func setupButton() -> UIButton {
    let button = UIButton(frame: CGRectZero)
    button.setTitle("Hello", forState: .Normal)
    return button
  }
}
```

```swift
class View : UIView {
  lazy var button : UIButton =
    UIButton(title: "Hello")
}
```

Readable functions

```
UIView *view1;
UIView *view2;

NSLayoutConstraint *constraint = [NSLayoutConstraint
constraintWithItem:view1 attribute:NSLayoutAttributeWidth
relatedBy:NSLayoutRelationEqual toItem:view2
attribute:NSLayoutAttributeWidth multiplier:1.0 constant:0.0];
constraint.priority = 750;
```

```
UIView *view1;
UIView *view2;

NSLayoutConstraint *constraint = [NSLayoutConstraint
constraintWithItem:view1 attribute:NSLayoutAttributeWidth
relatedBy:NSLayoutRelationEqual toItem:view2
attribute:NSLayoutAttributeWidth multiplier:1.0 constant:0.0];
constraint.priority = 750;
NSLayoutConstraint *constraint2 = [NSLayoutConstraint
constraintWithItem:view1 attribute:NSLayoutAttributeCenterY
relatedBy:NSLayoutRelationEqual toItem:view2
attribute:NSLayoutAttributeCenterY multiplier:1.0 constant:0.0]
  NSLayoutConstraint *constraint3 = [NSLayoutConstraint
constraintWithItem:view1 attribute:NSLayoutAttributeTrailing
relatedBy:NSLayoutRelationGreaterThanOrEqual toItem:view2
attribute:NSLayoutAttributeLeading multiplier:1.0 constant:10.0];
  NSLayoutConstraint *constraint4 = [NSLayoutConstraint
constraintWithItem:view1 attribute:NSLayoutAttributeWidth
relatedBy:NSLayoutRelationEqual toItem:nil
attribute:NSLayoutAttributeNotAnAttribute multiplier:1.0 constant:
50.0];
```

```
NSLayoutAttribute.Width
```

$$\downarrow$$

```
.Width
```

```swift
let constraint = NSLayoutConstraint(item:
item1, attribute: .Width, relatedBy: .Equal,
toItem: item2, attribute: .Width, multiplier:
1.0, constant: 0.0)
constraint.priority = 750
```

```swift
let constraint = NSLayoutConstraint(item: item1,
attribute: .Width, relatedBy: .Equal, toItem: item2,
attribute: .Width, multiplier: 1.0, constant: 0.0)
constraint.priority = 750

extension NSLayoutConstraint {
  convenience init(
    item1 : UIView,
    attribute1 : NSLayoutAttribute,
    relation : NSLayoutRelation,
    item2 : UIView?,
    attribute2 : NSLayoutAttribute,
    constant : CGFloat,
    multiplier : CGFloat) {
  self.init(
    item: item1,
    attribute: attribute1,
    relatedBy: relation,
    toItem: item2,
    attribute: attribute2,
    multiplier: multiplier,
    constant: constant)
  }
}
```

```swift
let constraint = NSLayoutConstraint(item1, .Width, .Equal,
item2, .Width, 1.0, 0.0)
constraint.priority = 750


extension NSLayoutConstraint {
  convenience init(
    _ item1 : UIView,
    _ attribute1 : NSLayoutAttribute,
    _ relation : NSLayoutRelation,
    _ item2 : UIView?,
    _ attribute2 : NSLayoutAttribute,
    _ constant : CGFloat,
    _ multiplier : CGFloat) {
  self.init(
    item: item1,
    attribute: attribute1,
    relatedBy: relation,
    toItem: item2,
    attribute: attribute2,
    multiplier: multiplier,
    constant: constant)
  }
}
```

```swift
let constraint = NSLayoutConstraint(item1, .Width, .Equal,
item2, .Width, constant: 1.0, multiplier: 0.0)
constraint.priority = 750


extension NSLayoutConstraint {
  convenience init(
    _ item1 : UIView,
    _ attribute1 : NSLayoutAttribute,
    _ relation : NSLayoutRelation,
    _ item2 : UIView?,
    _ attribute2 : NSLayoutAttribute,
    constant : CGFloat,
    multiplier : CGFloat) {
  self.init(
    item: item1,
    attribute: attribute1,
    relatedBy: relation,
    toItem: item2,
    attribute: attribute2,
    multiplier: multiplier,
    constant: constant)
  }
}
```

```
NSLayoutConstraint(item1, .Width, .Equal, item2, .Width,
constant: 1.0, multiplier: 0.0, priority: 750)

extension NSLayoutConstraint {
  convenience init(
    _ item1 : UIView,
    _ attribute1 : NSLayoutAttribute,
    _ relation : NSLayoutRelation,
    _ item2 : UIView?,
    _ attribute2 : NSLayoutAttribute,
    constant : CGFloat,
    multiplier : CGFloat,
    priority : UILayoutPriority) {
  self.init(
    item: item1,
    attribute: attribute1,
    relatedBy: relation,
    toItem: item2,
    attribute: attribute2,
    multiplier: multiplier,
    constant: constant)
    self.priority = priority
  }
}
```

```swift
extension NSLayoutConstraint {
  convenience init(
    _ item1 : UIView,
    _ attribute1 : NSLayoutAttribute,
    _ relation : NSLayoutRelation,
    _ item2 : UIView? = nil,
    _ attribute2 : NSLayoutAttribute = .NotAnAttribute,
    constant : CGFloat = 0.0,
    multiplier : CGFloat = 1.0,
    priority : UILayoutPriority = 1000.0) {
  self.init(
    item: item1,
    attribute: attribute1,
    relatedBy: relation,
    toItem: item2,
    attribute: attribute2,
    multiplier: multiplier,
    constant: constant)
    self.priority = priority
  }
}
```

```
Constraint(item1, .Width, .Equal, item2, .Width, priority: 750)

Constraint(item1, .Width, .Equal, constant: 50)

Constraint(item1, .Width, .Equal, item1, .Height)

Constraint(item1, .Top, .GreaterThanOrEqual, item2, .Top)
```

```swift
UIView.animateWithDuration(0.3, delay: 0,
options: nil, animations: {
  item1.alpha = 0
}, completion: nil)
```

```swift
extension UIView {
  class func animate(
    duration : NSTimeInterval,
    delay : NSTimeInterval = 0.0,
    options : UIViewAnimationOptions = nil,
    animations : Void -> Void,
    completion : (Bool -> Void)? = nil) {}
}
```

```swift
UIView.animate(0.3, animations: {
  item1.alpha = 0
})

UIView.animate(0.5, delay: 1, animations: {
  item1.alpha = 0
})

UIView.animate(0.5, options: .CurveEaseOut, animations: {
  item1.alpha = 0
})
```

```
animations : Void -> Void
```

```
nimate(0.3, animations: { item1.alpha = 0 })
```

```
func setAlphaToZero() {
  self.view.alpha = alpha
}
```

```
nimate(0.3, animations: setAlphaToZero)
```

```
func setAlpha(view : UIView, value : CGFloat) {
  view.alpha = value
}

animations: setAlpha(view, 0)
```

Curried function

```
func setAlpha(view : UIView, value : CGFloat)() {
  view.alpha = value
}
```

```
setAlpha // (UIView, CGFloat) -> () -> Void
setAlpha(item1, 0) // () -> Void
setAlpha(item1, 0)() // Void
```

```
func setAlpha(view : UIView, value : CGFloat)() {
  view.alpha = value
}
```

```
UIView.animate(0.3, animations: setAlpha(item1, 0))
```

Autoclosure

```
class func animate(
  duration : NSTimeInterval,
  animations : @autoclosure () -> Void)
```

```
UIView.animate(0.3, animations: item1.alpha = 0)
```

Operator overloading

```
        item1.alpha = 0     item1.center = CGPointZero

                     ↘    ↙

imate(0.3, animations:         )
```

```
func combine(
  f1 : @autoclosure () -> Void,
  f2 : @autoclosure () -> Void)
  -> (Void -> Void) {
  return { f1(); f2() }
}

combine(item1.alpha = 0, item1.center = CGPointZero)
```

```
func +(
  f1 : @autoclosure () -> Void,
  f2 : @autoclosure () -> Void)
  -> (Void -> Void) {
  return { f1(); f2() }
}

(item1.alpha = 0) + (item1.center = CGPointZero)
```

```swift
          infix operator |> {
            associativity left
            precedence 10
          }


func |>(
  f1 : @autoclosure () -> Void,
  f2 : @autoclosure () -> Void)
  -> (Void -> Void) {
  return { f1(); f2() }
}

item1.alpha = 0 |> item1.center = CGPointZero
```

```
UIView.animate(0.3, animations:
  item1.alpha = 0 |> item1.center = CGPointZero)
```

Generics

```swift
struct Container<T> {
  var value : T
}
```

```
struct Container<T : Printable> {
  var value : T

  func printValue() {
    println("value: " + value.description)
  }
}
```

```
class ArrayDataSource<Type> : NSObject, UITableViewDataSource {
  let values : [Type]

  func numberOfSectionsInTableView(tableView: UITableView) -> Int {
    return 1
  }

  func tableView(tableView: UITableView,
                 numberOfRowsInSection section: Int) -> Int {
    return values.count
  }

  func tableView(tableView: UITableView,
                 cellForRowAtIndexPath indexPath: NSIndexPath) -> UITableViewCell {
    let cell = tableView.dequeueReusableCellWithIdentifier("cell")
               ???

    return cell
  }
}
```

```swift
protocol GenericCell {
  typealias T

  func setValue(value : T)
}

class StringCell : UITableViewCell, GenericCell {
  typealias T = String

  func setValue(value : T) { ... }
}


extension UITableViewCell : GenericCell {
  typealias T = String

  func setValue(value : T?) {
    textLabel.text = value
  }
}
```

```
class ArrayDataSource<Type>
class ArrayDataSource<Type, Cell : GenericCell>
```

```
func tableView(tableView: UITableView,
               cellForRowAtIndexPath indexPath: NSIndexPath) -> UITableViewCell {

   let cell = tableView.dequeueReusableCellWithIdentifier(cellIdentifier) as Cell

   return cell

}
```

Cell is not convertible to UITableViewCell

```
class ArrayDataSource<Type, Cell : GenericCell>
class ArrayDataSource<Type, Cell : GenericCell
                          where Cell : UITableViewCell>
```

```swift
class ArrayDataSource<Type, Cell : GenericCell
                      where Cell : UITableViewCell, Cell.T == Type>
```

```swift
func tableView(tableView: UITableView,
               cellForRowAtIndexPath indexPath: NSIndexPath) -> UITableViewCell {

   let cell = tableView.dequeueReusableCellWithIdentifier(cellIdentifier) as Cell
   cell.value = values[indexPath.row]
   return cell

}
```

Summary

Enums

```
enum User {
  case Anonymous
  case Registered(name : String)
}
```

Optionals

```swift
let userId : Int? = nil // No user
```

Lazyness

```
class View : UIView {
  lazy var button : UIButton =
    UIButton(title: "Hello")
}
```

Readability

```
func doStuff(completion : (Void -> Void)? = nil)
```

```
Constraint(item1, .Width, .Equal, item2, .Width, priority: 750)
Constraint(item1, .Width, .Equal, constant: 50)
```

```
UIView.animate(0.3, animations: setAlpha(item1, 0))
```

```
UIView.animate(0.3, animations: item1.alpha = 0)
```

Generics

```
protocol GenericCell {
  typealias T

  var value : T { get set }
}
```

```
class ArrayDataSource<Type, Cell : GenericCell
                  where Cell : UITableViewCell, Cell.T == Type>
```

Ulrik Damm
ufd.dk

Kaomoji Keyboard