Swift Style

@gregheo

Style?

raywenderlich / swift-style-guide

Watch 330 | Star 3,684 | Fork 524

Code | Issues 15 | Pull requests 0 | Pulse | Graphs

The official Swift style guide for raywenderlich.com.

128 commits | 2 branches | 0 releases | 14 contributors

Branch: master | New pull request | New file | Find file | HTTPS | https://github.com/raywe | Download ZIP

gregheo Merge pull request #115 from raywenderlich/swift2 ··· Latest commit da0a801 on Nov 5

screens	Re-indent tip and screenshot	a year ago
README.markdown	enumerate() is a protocol method now	3 months ago

README.markdown

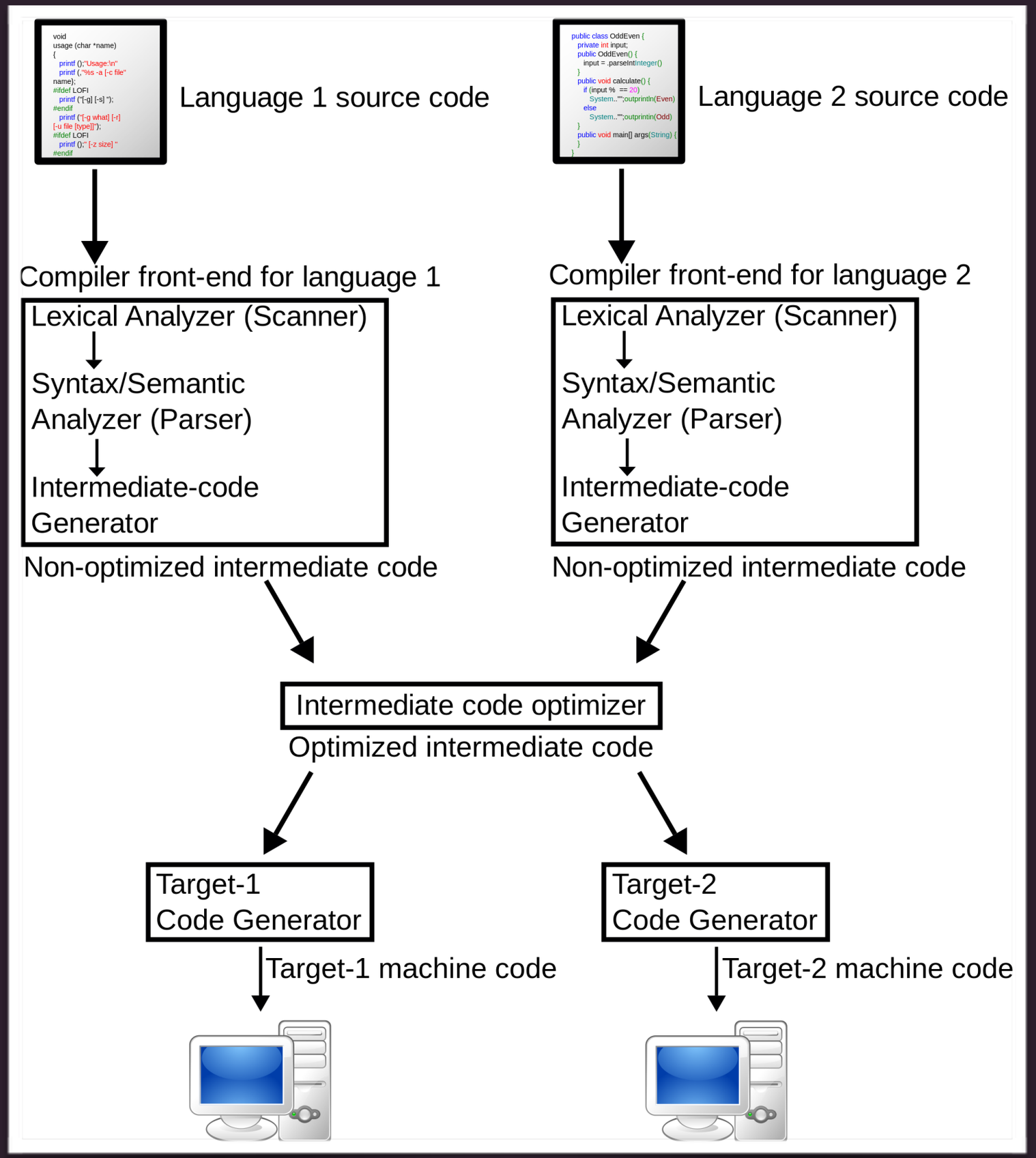
The Official raywenderlich.com Swift Style Guide.

This style guide is different from others you may see, because the focus is centered on readability for print and the web. We created this style guide to keep the code in our books, tutorials, and starter kits nice and consistent — even though we have many different authors working on the books.

Our overarching goals are conciseness, readability, and simplicity.

Writing Objective-C? Check out our Objective-C Style Guide too.

Table of Contents


Language 1 source code
Language 2 source code
Compiler front-end for language 1
Lexical Analyzer (Scanner)
Syntax/Semantic Analyzer (Parser)
Intermediate-code Generator
Compiler front-end for language 2
Lexical Analyzer (Scanner)
Syntax/Semantic Analyzer (Parser)
Intermediate-code Generator
Non-optimized intermediate code
Non-optimized intermediate code
Intermediate code optimizer
Optimized intermediate code
Target-1 Code Generator
Target-2 Code Generator
Target-1 machine code
Target-2 machine code

```
for _ in 0..<indent { print(" ", terminator:
"", toStream: &targetStream) }; let count =
mirror.count; let bullet = count == 0 ? “–" :
maxDepth <= 0 ? "▹" : “▿”; print("\(bullet) ",
terminator: "", toStream: &targetStream) if let
nam = name { print("\(nam): ", terminator: "",
toStream: &targetStream); }
print(mirror.summary, terminator: "", toStream:
&targetStream); if let id =
mirror.objectIdentifier { if let previous =
visitedItems[id] { print(" #\(previous)",
toStream: &targetStream); return; } let
identifier = visitedItems.count;
visitedItems[id] = identifier; print(" #\
(identifier)", terminator: "", toStream:
&targetStream); }
```

```
for _ in 0..<indent { print(" ", terminator: "", toStream: &targetStream) }

let count = mirror.count
let bullet = count == 0    ? "–"
           : maxDepth <= 0 ? "▹" : "▿"
print("\(bullet) ", terminator: "", toStream: &targetStream)

if let nam = name {
  print("\(nam): ", terminator: "", toStream: &targetStream)
}
print(mirror.summary, terminator: "", toStream: &targetStream)

if let id = mirror.objectIdentifier {
  if let previous = visitedItems[id] {
    print(" #\(previous)", toStream: &targetStream)
    return
  }
  let identifier = visitedItems.count
  visitedItems[id] = identifier
  print(" #\(identifier)", terminator: "", toStream: &targetStream)
}
```

Structure and Interpretation of Computer Programs

Second Edition

Harold Abelson and
Gerald Jay Sussman
with Julie Sussman

```
MOV AX, SI                      ; set AX=SI
MOV CX, BX                      ; set CX=BX
DEC CX                          ; set CX=CX-1

@OUTER_LOOP:                    ; loop label
  MOV BX, CX                    ; set BX=CX

  MOV SI, AX                    ; set SI=AX
  MOV DI, AX                    ; set DI=AX
  INC DI                        ; set DI=DI+1

  @INNER_LOOP:                  ; loop label
    MOV DL, [SI]                ; set DL=[SI]

    CMP DL, [DI]                ; compare DL with [DI]
    JNG @SKIP_EXCHANGE          ; jump to label @SKIP_EXCHANGE if DL<[DI]

    XCHG DL, [DI]               ; set DL=[DI], [DI]=DL
    MOV [SI], DL                ; set [SI]=DL

    @SKIP_EXCHANGE:             ; jump label
    INC SI                      ; set SI=SI+1
    INC DI                      ; set DI=DI+1

    DEC BX                      ; set BX=BX-1
  JNZ @INNER_LOOP               ; jump to label @INNER_LOOP if BX!=0
LOOP @OUTER_LOOP
```

SORTING ALGORITHM

```
MOV AX, SI                    ; set AX=SI
MOV CX, BX                    ; set CX=BX
DEC CX                        ; set CX=CX-1

@OUTER_LOOP:                  ; loop label
  MOV BX, CX                  ; set BX=CX

  MOV SI, AX                  ; set SI=AX
  MOV DI, AX                  ; set DI=AX
  INC DI                      ; set DI=DI+1

  @INNER_LOOP:                ; loop label
    MOV DL, [SI]              ; set DL=[SI]

    CMP DL, [DI]              ; compare DL with [DI]
    JNG @SKIP_EXCHANGE        ; jump to label @SKIP_EXCHANGE if DL<[DI]

    XCHG [DI], DL             ; set DL=[DI], [DI]=DL
    MOV [SI], DL              ; set [SI]=DL

    @SKIP_EXCHANGE:           ; jump label
    INC SI                    ; set SI=SI+1
    INC DI                    ; set DI=DI+1

    DEC BX                    ; set BX=BX-1
  JNZ @INNER_LOOP             ; jump to label @INNER_LOOP if BX!=0
LOOP @OUTER_LOOP
```

"Programs must be written for people to read, and only incidentally for machines to execute."

– *Structure and Interpretation of Computer Programs*

"Good programmers write code that humans can understand"

– *Refactoring*

```
for _ in 0..<indent { print(" ", terminator:
"", toStream: &targetStream) }; let count =
mirror.count; let bullet = count == 0 ? “–" :
maxDepth <= 0 ? "▹" : “▿”; print("\(bullet) ",
terminator: "", toStream: &targetStream) if let
nam = name { print("\(nam): ", terminator: "",
toStream: &targetStream); }
print(mirror.summary, terminator: "", toStream:
&targetStream); if let id =
mirror.objectIdentifier { if let previous =
visitedItems[id] { print(" #\(previous)",
toStream: &targetStream); return; } let
identifier = visitedItems.count;
visitedItems[id] = identifier; print(" #\
(identifier)", terminator: "", toStream:
&targetStream); }
```

Concise Code

```
[self beginTaskWithName:@"MyTask"
  completion:^{
    NSLog(@"The task is complete");
}];
```

```
self.beginTaskWithName("MyTask",
  completion: {
    print("The task is complete")
})
```

```
beginTaskWithName("MyTask",
  completion: {
    print("The task is complete")
})
```

```
beginTaskWithName("MyTask") {
  print("The task is complete")
}
```

```
[self beginTaskWithName:@"MyTask"
  completion:^{
    NSLog(@"The task is complete");
}];
```

```
beginTaskWithName("MyTask") {
  print("The task is complete")
}
```

Concise Code

- Value types
- Protocols
- Safety

Value Types: Immutability

NSArray
NSData
NSDictionary
NSIndexSet
NSSet
NSString

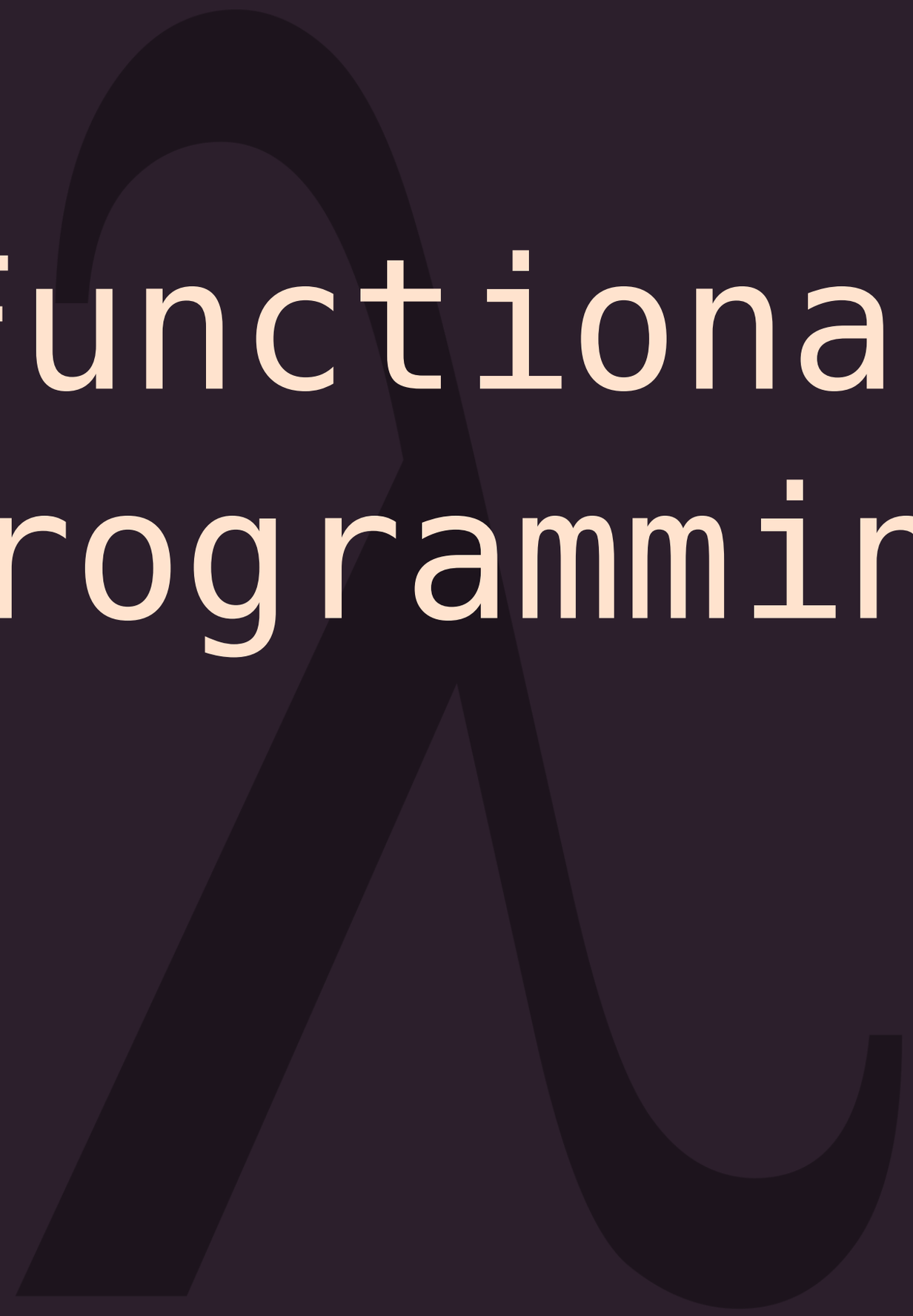
NSArray	NSMutableArray
NSData	NSMutableData
NSDictionary	NSMutableDictionary
NSIndexSet	NSMutableIndexSet
NSSet	NSMutableSet
NSString	NSMutableString

let

var

state!

Functional Programming



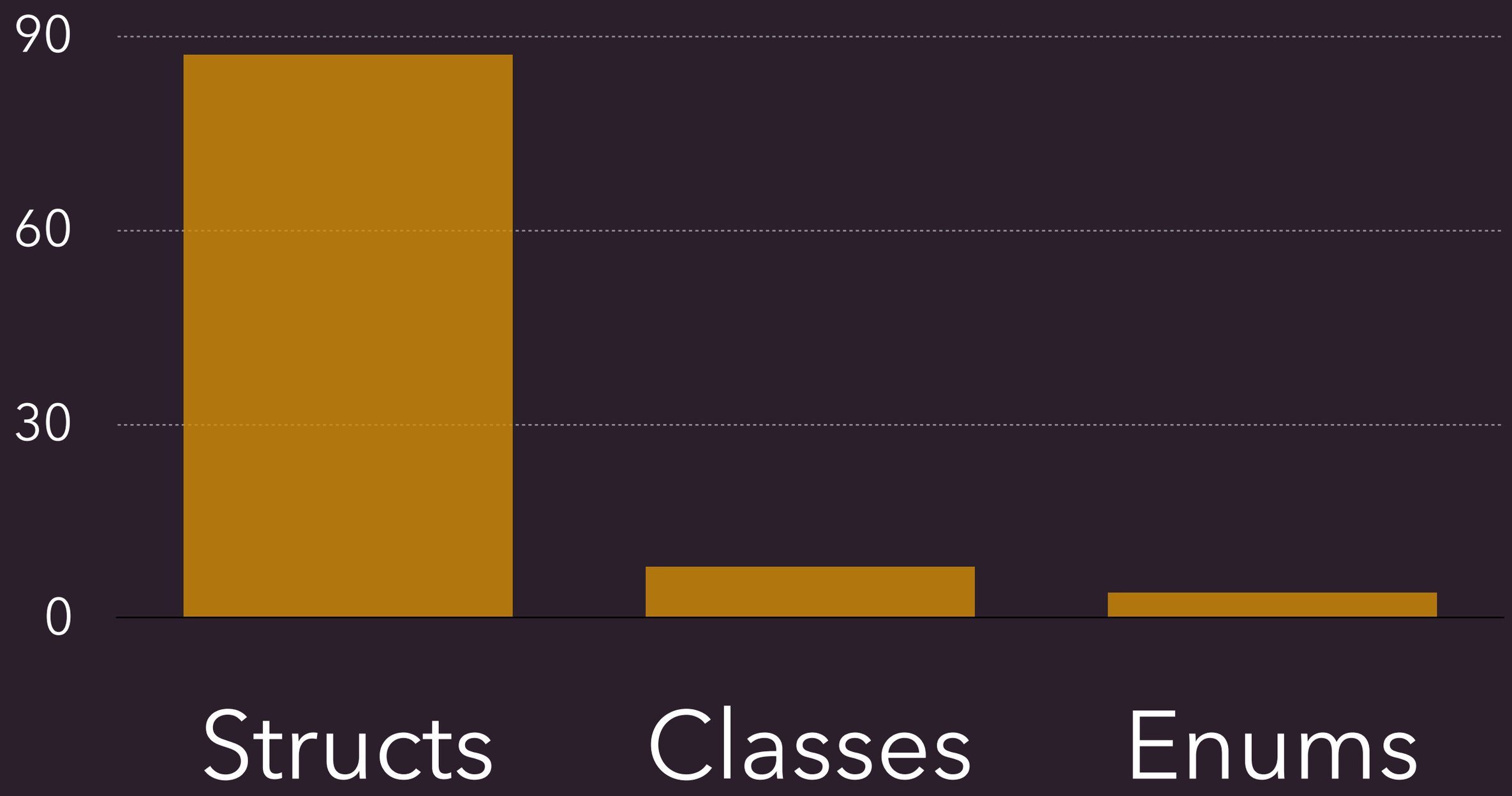
Concurrency

```
var helloString = "您好"
```

⚠ Variable 'helloString' was never mutated; consider changing to 'let' constant

Value Types: Immutability

Value Types: Enumerations


90
60
30
0
Structs
Classes
Enums

```
typedef NS_ENUM(NSInteger, TransportMode) {
  TransportModeAirplane,
  TransportModeBoat,
  TransportModeTruck
};
```

```swift
enum TransportMode {
    case Airplane
    case Boat
    case Truck
}
```

```swift
enum TransportMode {
  case Airplane
  case Boat
  case Truck
}

func emojiForTransportMode(mode: TransportMode)
-> String {
  switch mode {
  case .Airplane: return "✈️"
  case .Boat:     return "⛵"
  case .Truck:    return "🚚"
  }
}
```

```swift
enum TransportMode {
  case Airplane
  case Boat
  case Truck

  func emojiRepresentation() -> String {
    switch self {
    case .Airplane: return "✈️"
    case .Boat:     return "⛵"
    case .Truck:    return "🚚"
    }
  }
}
```

```swift
enum TransportMode {
  case Airplane
  case Boat
  case Truck

  var emojiRepresentation: String {
    switch self {
    case .Airplane: return "🛩"
    case .Boat:     return "⛵"
    case .Truck:    return "🚚"
    }
  }
}
```

```swift
enum TransportMode: String {
  case Airplane = "✈️"
  case Boat     = "⛵"
  case Truck    = "🚚"
}

let t = TransportMode.Airplane
t.rawValue  // "✈️"
```

```
enum TransportMode: String {
  case Airplane = "✈️"
  case Boat     = "⛵"
  case Truck    = "🚚"
  case Rocket   = "✈️"
}
```

Raw value for enum case is not unique

Enumeration Uses

- Well-defined set of values
- Segues
- Asset catalog items

Enumerations

- Computed properties
- Methods
- Raw values
- Associated values

Protocols

```swift
struct Sentence {
  let sentence: String

  init(_ sentence: String) {
    self.sentence = sentence
  }

  // ...
}

let mySentence = Sentence("Hello there")
```

Value Type Behavior

```
mySentence == otherSentence
mySentence >= otherSentence
print("\(mySentence)")
```

```swift
struct Sentence {
  let sentence: String

  init(_ sentence: String) {
    self.sentence = sentence
  }

  // ...
}
```

```
extension Sentence: Equatable {
}

func ==(lhs: Sentence, rhs: Sentence) -> Bool {
  return lhs.sentence == rhs.sentence
}


mySentence == otherSentence
```

```swift
extension Sentence: Comparable {
}

func <(lhs: Sentence, rhs: Sentence) -> Bool {
  return lhs.sentence < rhs.sentence
}



mySentence >= otherSentence
```

```
print("\(mySentence)")
```

```
Sentence(sentence: "Hello there")
```

```
extension Sentence: CustomStringConvertible {
  var description: String {
    return sentence
  }
}

print("\(mySentence)")
```

```
Hello there
```

More Value Type Behavior

```
let sentence = Sentence("Hello there")
let sentence: Sentence = "Hello there"

dictionary[mySentence] = "value"

for word in mySentence { ... }
```

```swift
extension Sentence: StringLiteralConvertible {
  // more code here!

  init(stringLiteral value: StringLiteralType) {
    self.sentence = value
  }
}


let otherSentence: Sentence = "Hello there"
```

```
extension Sentence: Hashable {
  var hashValue: Int {
    return sentence.hashValue
  }
}


dictionary[mySentence] = "value"
set.insert(mySentence)
```

```
extension Sentence: SequenceType {
  func generate() -> SentenceGenerator {
    let words =
      sentence.componentsSeparatedByString(" ")
    return SentenceGenerator(words: words)
  }
}
```

```swift
struct SentenceGenerator: GeneratorType {
  let words: [String]
  var index = 0

  mutating func next() -> String? {
    if index < words.count {
      let thisIndex = index
      index += 1
      return words[thisIndex]
    } else {
      return nil
    }
  }
}
```

```swift
for word in mySentence {
  print("Word: \(word)")
}
```

```
Word: Hello
Word: world
```

Value Type Behavior

• Equality	== !=
• Comparison	> < <= >=
• Printing	print
• Hashing	Set/dict
• String conversion	"..."
• Iteration	for x in ...


SAFETY
FIRST

Undefined Behavior

Optionals?!

!

?

??

if-let

guard

```
func decode() {
  if let x = input.next() {
    if  fastPath((x >> 11) != 0b1101_1) {
      return .Result(UnicodeScalar(x))
    } else {
      return .Error
    }
  }

  return .EmptyInput
}
```

```swift
guard let x = input.next() else
  { return .EmptyInput }
```

❌

```swift
if _fastPath((x >> 11) != 0b1101_1) {
  return .Result(UnicodeScalar(x))
} else {
  return .Error
}
```

Preconditions & Assertions

Assertions

"internal sanity checks that are active during testing but do not impact performance of shipping code"

Preconditions

"Check a necessary condition for making forward progress."

Preconditions

```
precondition(ablMemory != nil,
 "failed to allocate memory")

precondition(index >= 0,
 "vector index out of range")
```

fatalError()

"Unconditionally stop execution." 

	Debug	Release
assert	✔	😴
precondition	✔	✔
fatalError()	✔	✔

- Value types
- Protocols
- Safety


EXIT

@Swift 👀