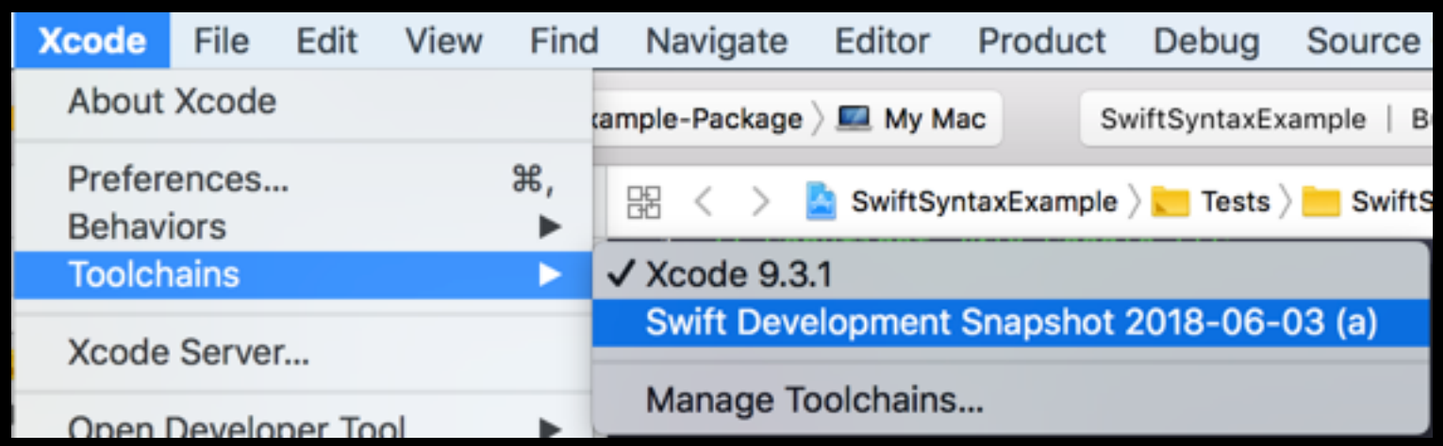
Before the Workshop

- Grab a flash drive
- Copy all contents to Desktop
- Install Swift toolchain

```
$ cd ~/Desktop/Getting-Started-With-SwiftSyntax
$ make open-xcode
```

- Select the latest Swift toolchain in Xcode



SwiftFest 2018

Building Great Tools with SwiftSyntax

Harlan Haskins @harlanhaskins
Alexander Lash @lexlash

Agenda

- Introduce SwiftSyntax
- Learn the API
- Create your own tool

What is SwiftSyntax?

Swift Library for Parsing Swift

Officially Supported

Great for Code Transformation

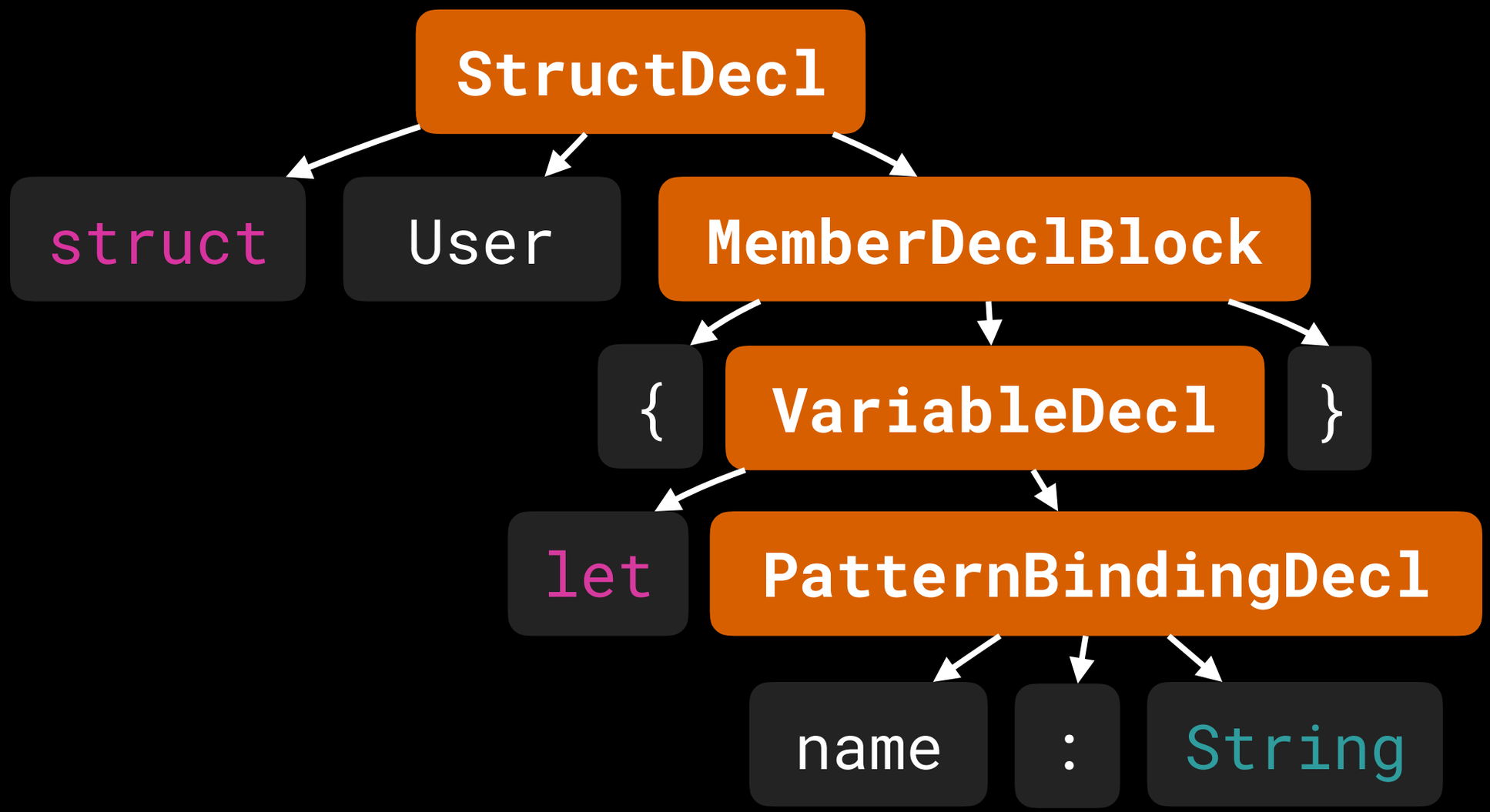
SwiftSyntax Structure

- Uses compiler to parse code
- Tree structure representing Swift grammar
- Comments and whitespace are preserved
- Allows for re-printing files after parse

SwiftSyntax Structure

```
struct User {
    let name: String
}
```

```
struct User {
  let name: String
}
```



```
struct User {
   let name: String
}
```

```
struct   User   {
   let   name   :   String
}
```

```
struct User {
   let name: String
}
```

```
struct_ User_ {
\n__let_ name :_ String
\n}
```

Anatomy of a Token

Anatomy of a Token

```
\n__let_
```

Leading Trivia

```
Trivia([
  .newlines(1),
  .spaces(2)
])
```

Token Kind

```
.letKeyword
```

Trailing Trivia

```
Trivia([
  .spaces(1)
])
```

Anatomy of a Token

```
User_
```

Leading Trivia

```
Trivia([])
```

Token Kind

```
.identifier("User")
```

Trailing Trivia

```
Trivia([
    .spaces(1)
])
```

Trivia Rules

```
\n__ let _
```

Trailing Trivia

A token owns all the trivia after it, up to the first new line or other token.

Leading Trivia

A token owns all the trivia before it from the first newline up to its text.

Trivia Rules

```
/// Represents a user's
/// account in the database
struct User {
    let name: String // User's name
}
```

Trivia Rules

```
/// Represents a user's
/// account in the database
struct User {
    let name: String // User's name
}
```

SwiftSyntax API

Syntax Nodes

- Each node has a `struct`, with:
  - Type-safe accessors for each of the children
  - Type-safe methods to replace each child

```
struct TokenSyntax: Syntax {
  var leadingTrivia: Trivia { get }
  // ...
  func withLeadingTrivia(
    _ trivia: Trivia) -> TokenSyntax
  // ...
}
```

Transforming Syntax Nodes

```
struct User {
    let name: String
}
```

```
struct Account {
    let name: String
}
```

Transforming Syntax Nodes

```
struct User {
  let name: String
}
```

```
struct Account {
  let name: String
}
```

```
let newIdentifier =
  userStruct.identifier.withKind(
    .identifier("Account")
  )

let accountStruct =
  userStruct.withIdentifier(newIdentifier)
```

Parsing a File

```
import Foundation
import SwiftSyntax

let url = URL(...)
let sourceFile =
  try SourceFileSyntax.parse(url)
```

Syntax Visitor

- Allows you to traverse a static parse tree
- Provides visitor methods that you can override

Syntax Visitor

```
class FunctionCounter: SyntaxVisitor {
}
```

Syntax Visitor

```
class FunctionCounter: SyntaxVisitor {
  var numberOfFunctions = 0
}
```

Syntax Visitor

```
class FunctionCounter: SyntaxVisitor {
  var numberOfFunctions = 0

  override func visit(
    _ function: FuncDeclSyntax
  ) {
    self.numberOfFunctions += 1
    super.visit(function)
  }

}
```

Syntax Visitor

```
import Foundation
import SwiftSyntax

let url = URL(...)
let sourceFile =
  try SourceFileSyntax.parse(url)
```

Syntax Visitor

```
import Foundation
import SwiftSyntax

let url = URL(...)
let sourceFile =
  try SourceFileSyntax.parse(url)

let counter = FunctionCounter()
counter.visit(sourceFile)
print(counter.numberOfFunctions)
```

Syntax Rewriter

- Similar to Syntax Visitor
- Transform the tree by replacing Syntax nodes

Syntax Factory

- Static methods to create Syntax nodes
- All in one place
- Discoverable with autocomplete

Syntax Factory

```
// ‘!’ token, with one space
let exclamation =
  SyntaxFactory.makeExclamationMarkToken(
    trailingTrivia: .spaces(1)
  )
```

```
// ‘<expression>!’
let forceUnwrapped =
  SyntaxFactory.makeForcedValueExpr(
    expression: someExpression,
    exclamationMark: exclamation
  )
```

Diagnostics API

Diagnostics

- Just like Swift diagnostics
- Can either be errors or warnings
- Can have notes and Fix-Its attached

Diagnostics

- Just like Swift diagnostics
- Can either be errors or warnings
- Can have notes and Fix-Its attached

Adding Diagnostic Messages

- Similar to `Notification.Name`
- Add `static let` or `func` to `Diagnostic.Message`

```
extension Diagnostic.Message {
   static let noForceUnwrap =
      Diagnostic.Message(
         .error,
         "force unwrapping is not allowed"
      )
}
```

Diagnostic Engine

- Coordinator for diagnostics
- Keeps emitted diagnostics in order
- Forwards diagnostics to ‘consumers’

Diagnostic Engine

```
let engine = DiagnosticEngine()
let printingConsumer =
  PrintingDiagnosticConsumer()
engine.addConsumer(printingConsumer)

let nodeStart = node.startLocation(in: file)
engine.diagnose(.noForceUnwrap,
  location: nodeStart
)
```

Diagnostic Engine

```
let engine = DiagnosticEngine()
let printingConsumer =
  PrintingDiagnosticConsumer()
engine.addConsumer(printingConsumer)

let nodeStart = node.startLocation(in: file)
engine.diagnose(.noForceUnwrap,
   location: nodeStart
){ diag in
  let nodeRange = node.sourceRange(in: file)
  diag.highlight(nodeRange)
}
```

Workshop

Harlan Haskins @harlanhaskins
Alexander Lash @lexlash