Agenda

- Start Swift on Linux
  - docker start swiftfun
  - docker attach swiftfun
- Language basics
  - Variables, Constants, Strings and Operators

Swift feature	Description
Type inference	Swift can automatically deduce the type of the variable or constant, based on the initial value.
Generics	Generics allow us to write the code only once to perform identical tasks for different types of objects.
Collection mutability	Swift does not have separate objects for mutable or nonmutable containers. Instead, you define mutability by defining the container as a constant or variable.
Closure syntax	Closures are self-contained blocks of functionality that can be passed around and used in our code.
Optionals	Optionals define a variable that might not have a value.
Switch statement	The Switch statement has been drastically improved.
Multiple return types	Functions can have multiple return types using tuples.
Operator overloading	Classes can provide their own implementation of existing operators.
Enumerations with Associated values	In Swift, we can do a lot more than just defining a group of related values with enumerations.

Hello World

```
var name = "Jon"
var language = "Swift"

var message1 = "Welcome to the wonderful world of "
var message2 = "\(name) Welcome to the wonderful world of \(language)!"

print(name, message1, language, "!")
print(message2)
```

vi main.swift
swiftc main.swift
./main

REPL

```
Welcome to Swift version 3.0 ({your-swift-version}). Type :help for
    assistance
1>

1>  var x = 10
x: Int = 10
2>  x += 5
3>  print(x)
15

:quit
```

swiftc options

```
swiftc main.swift
```

```
swiftc main.swift file1.swift file2.swift file3.swift
```

```
swiftc main.swift file1.swift file2.swift -o myexecutable
```

Constant and Variables

- What variables and constants are
- The difference between explicit and inferred typing
- Explaining numeric, string, and boolean types
- Defining optional types
- Explaining how enumerations work in Swift
- Explaining how Swift's operators work

Constant and Variables

- An identifier must not contain any whitespace
- An identifier must not contain any mathematical symbols
- An identifier must not contain any arrows
- An identifier must not contain private use or invalid Unicode characters
- An identifier must not contain line- or box-drawing characters
- An identifier must not start with a number, but it can contain numbers
- If you use a Swift keyword as an identifier, surround it with back ticks

Constant and Variables

```
// Constants
let freezingTemperatureOfWaterCelsius = 0
let speedOfLightKmSec = 300000

// Variables
var currentTemperature = 22
var currentSpeed = 55
```

```
// Constants
let freezingTempertureOfWaterCelsius = 0, speedOfLightKmSec = 300000

// Variables
var currentTemperture = 22, currentSpeed = 55
```

Constant and Variables

```
let speedOfLightKmSec = 300000
var highTemperture = 93

highTemperture = 95
speedOfLightKmSec = 29999
```

Type Safety

```
var integerVar = 10

integerVar = "My String"
```

error: cannot assign value of type 'String' to type 'Int' integerVar = "My String"

Type Inference

```
var x = 3.14       // Double type
var y = "Hello"    // String type
var z = true       // Boolean type


print(type(of: x))
```

Explicit Types

var pi : Float = 3.14

var x: Int

Integers

Type	Minimum	Maximum
Int8	−128	127
Int16	−32,768	32,767
Int32	−2,147,483,648	2,147,483,647
Int64	−9,223,372,036,854,775,808	9,223,372,036,854,775,807
Int	−9,223,372,036,854,775,808	9,223,372,036,854,775,807
UInt8	0	255
UInt16	0	65,535
UInt32	0	4,294,967,295
UInt64	0	18,446,744,073,709,551,615
UInt	0	18,446,744,073,709,551,615

Integers

```
print("UInt8 max \(UInt8.max)")
print("UInt8 min \(UInt8.min)")
```

Integers

```
var a = 95

var b = 0b1011111

var c = 0o137

var d = 0x5f


let speedOfLightKmSec = 300_000
```

Floating Point

```
let f: Float = 0.111_111_111 + 0.222_222_222

let d: Double = 0.111_111_111 + 0.222_222_222
```

Floating Point

```
var a : Int = 3

var b : Double = 0.14

var c = a + b
```

```
var a : Int = 3

var b : Double = 0.14

var c = Double(a) + b
```

Boolean

```
let swiftIsCool = true

let swiftIsHard = false

var itIsWarm = false

var itIsRaining = true
```

Boolean

```
let isSwiftCool = true
let isItRaining = false
if isSwiftCool {
    print("YEA, I cannot wait to learn it")
}
if isItRaining {
    print("Get a rain coat")
}
```

Strings

var stringOne = "Hello"

var stringTwo = " World"

Strings

```
var stringOne = "Hello"
for char in stringOne.characters {
    print(char)
}
```

Strings

var stringC = stringA + stringB

stringA += stringB

Strings

```
var stringA = "Jon"

var stringB = "Hello \(stringA)"
```

Strings – Mutable vs Immutable

```
var x = "Hello"

let y = "HI" var

z = "World"

//This is valid, x is mutable

x += z

//This is invalid, y is not mutable.

y += z
```

Strings – Converting

```
var stringOne = "hElLo"

print("Lowercase String: " + stringOne.lowercased())

print("Uppercase String: " + stringOne.uppercased())
```

```
Lowercase String: hello

Uppercase String: HELLO
```

Strings – Comparison

```
var stringOne = "Hello Swift"
var stringTwo = ""
stringOne.isEmpty //false
stringTwo.isEmpty //true
stringOne == "hello swift" //false
stringOne == "Hello Swift" //true
stringOne.hasPrefix("Hello") //true
stringOne.hasSuffix("Hello") //false
```

Strings – Replacing

```
var stringOne = "one,to,three,four"

print(stringOne.replacingOccurrences(of: "to", with: "two"))
```

Optional variables

```
var optionalString: String?
var nonoptionalString: String


var name: String?
name = "Jon"
if name != nil {
        var newString = "Hello " + name!
}
```

Enumerations

```
enum Planets {

        case Mercury
        case Venus
        case Earth
        case Mars
        case Jupiter
        case Saturn
        case Uranus
        case Neptune

}

var planetWeLiveOn = Planets.Earth
var furthestPlanet = Planets.Neptune
planetWeLiveOn = .Mars
```

Enumerations

```
// Using the traditional == operator
if planetWeLiveOn == .Earth {
        print("Earth it is")
}
// Using the switch statement
switch planetWeLiveOn {
        case .Mercury:
                print("We live on Mercury, it is very hot!")
        case .Venus:
                print("We live on Venus, it is very hot!")
        case .Earth:
                print("We live on Earth, just right")
        case .Mars:
                print("We live on Mars, a little cold")
        default:
                print("Where do we live?")
}
```

Enumerations

```
enum Devices: String {
        case MusicPlayer = "iPod"
        case Phone = "iPhone"
        case Tablet = "iPad"
}
print("We are using an " + Devices.Tablet.rawValue)
```

Enumerations

```
enum Product {
    case Book(Double, Int, Int)
    case Puzzle(Double, Int)
}
var masterSwift = Product.Book(49.99, 2016, 310)
var worldPuzzle = Product.Puzzle(9.99, 200)

switch masterSwift {
case .Book(let price, let year, let pages):
    print("Mastering Swift was published in \(year) for the price of
        \(price) and has \(pages) pages")
case .Puzzle(let price, let pieces):
    print("Master Swift is a puzze with \(pieces) and sells for
         \(price)")
}

switch worldPuzzle {
case .Book(let price, let year, let pages):
    print("World Puzzle was published in \(year) for the price of
        \(price) and has \(pages) pages")
case .Puzzle(let price, let pieces):
    print("World Puzzle is a puzze with \(pieces) and sells for
        \(price)")
}
```

Operators

Assignment

Comparison

Arithmetic

Remainder

Compound Assignment

Ternary

NOT

AND

OR

Assignment

Prototype:

```
varA = varB
```

Example:

```
let x = 1
var y = "Hello"
a = b
```

Comparison

Prototypes:

```
Equality:  varA == varB
Not equal:  varA != varB
Greater than:  varA > varB
Less than:  varA < varB
Greater than or equal to:  varA >= varB
Less than or equal to:  varA <= varB
```

Example:

```
2 == 1 //false, 2 does not equal 1
2 != 1 //true, 2 does not equal 1
2 > 1  //true, 2 is greater than 1
2 < 1  //false, 2 is not less than 1
2 >= 1 //true, 2 is greater or equal to 1
2 <= 1 //false, 2 is not less or equal to 1
```

Arithmetic

Prototypes:

```
Addition:  varA + varB
Subtraction:  varA - varB
Multiplication:  varA * varB
Division:  varA / varB
```

Example:

```
var x = 4 + 2  //x will equal 6
var x = 4 - 2  //x will equal 2
var x = 4 * 2  //x will equal 8
var x = 4 / 2  //x will equal 2
var x = "Hello " + "world"  //x will equal "Hello World"
```

Remainder

Prototype:

```
varA % varB
```

Example:

```
var x = 10 % 3  //x will equal 1
var x = 10 % 2.6  //x will equal 2.2
```

Compound Assignment

Prototypes:

```
varA += varB
varA -= varB
varA *= varB
varA /= varB
```

Example:

```
var x = 6
x += 2  //x is equal to 8
x -= 2  //x is equal to 4
x *= 2  //x is equal to 12
x /= 2  //x is equal to 3
```

Ternary

Prototype:

```
(boolValue ? valueA : valueB)
```

Example:

```
var x = 2
var y = 3
var z = (y > x ? "Y is greater" : "X is greater")  //z equals   "Y is greater"
```

Logical NOT

Prototype:

```
varA = !varB
```

Example:

```
var x = true
var y = !x  //y equals false
```

Logical AND

Prototype:

```
varA && varB
```

Example:

```
var x = true
var y = false
var z = x && y  //z equals false
```

Logical OR

Prototype:

```
varA || varB
```

Example:

```
var x = true
var y = false
var z = x || y  //z equals true
```