

# An Introduction to the Swift Programming Language

Sarah Reichelt

# What is Swift?

- Language introduced by Apple in 2014
- Open-sourced 2015
- Used for all Apple devices
- Now extending to Linux and server-side apps

*Swift is a general-purpose programming language built using a modern approach to safety, performance, and software design patterns.*

# How to follow along

- If you have a Mac, use Xcode and a Swift Playground
- If you have an iPad, use Swift Playgrounds
- If you do not have a Mac, use the “We ❤️ Swift” online sandbox
- Go to <http://troz.net/ncss2018> for a direct link

```
1 print("👋 🌍 !")
```

```
2
```

```
3
```

```
4
```

👋 🌍 !

We ❤️ Swift



# Using the sandbox:

- Swift is designed for writing apps.
- In apps, the input comes from controls: buttons, sliders, text fields etc. or databases, or the internet.
- You will set inputs using variables & constants. Just imagine that these are coming from user input and then change them manually to test different things.
- You will display your results using the "print" command.

**Data**

# Variables

```
var hitPoints = 29  
print(hitPoints)
```

```
hitPoints = 12  
print(hitPoints)
```

```
29  
12
```

# Constants

```
let playerName = "Maximus"  
print(playerName)  
playerName = "Minimus"
```

```
code:4:10: error: cannot assign to value:  
'playerName' is a 'let' constant  
playerName = "Minimus"  
~~~~~ ^
```

```
code:1:1: note: change 'let' to 'var' to make it  
mutable  
let playerName = "Maximus"  
^~~  
var
```



# Naming variables & constants

- Variable names can contain almost any character, including emojis.
- They cannot contain spaces, punctuation or maths symbols.
- They cannot start with a number.

# Names

```
// Valid Names
```

```
let myVariable = 42  
let next_number = 43  
let 🐛 = "pile of poo"  
let phase_1_active = true
```

```
// Invalid names
```

```
let first number = 37  
let 2Much = 4_000_000  
let a+b = "AB"
```

*Strings MUST be surrounded by double quotes.  
Single quotes are not valid for Swift strings.*

# Types

```
let anotherPlayerName = "Shocker"  
print(type(of: anotherPlayerName))  
  
var weaponStrength = 104  
print(type(of: weaponStrength))  
  
let damagePerSecond = 13.54  
print(type(of: damagePerSecond))  
  
var isDead = false  
print(type(of: isDead))
```

String  
Int  
Double  
Bool

# Comments

```
// This is a single line comment
```

```
let x = 3    // Comment on the end of a line
```

```
/*  
This is a multi-line comment.  
You start and end these with  
slashes & asterisks.  
*/
```

*Comments are very important when you have to come back to code  
or are trying to understand someone else's code.*

# Strings

# Strings

```
let string1 = "I am a mage!"  
print(string1)
```

```
let robotHead = "🤖"  
print("I am fighting a demonic \"(robotHead).")
```

```
let robotHitPoints = 427  
print("The robot has \"(robotHitPoints) hit  
points!")
```

I am a mage!

I am fighting a demonic 🤖.

The robot has 427 hit points!

# Looking at Strings

```
let spellName = "Fire"  
for character in spellName.characters {  
    print(character)  
}  
  
print(spellName.characters.count)  
print(spellName.isEmpty)
```

```
F  
i  
r  
e  
  
4  
false
```

# Parts of Strings

**Swift 4 syntax - works in Xcode, not in sandbox!**

```
let greeting = "I am a fire mage"
```

```
let firstWord = greeting.prefix(4)  
print(firstWord)
```

```
let lastWords = greeting.suffix(9)  
print(lastWords)
```

```
let words = greeting.components(separatedBy: " ")  
print(words)
```

```
I am  
fire mage  
["I", "am", "a", "fire", "mage"]
```



# Collections

# Arrays

```
var classTypes = ["Mage", "Warrior", "Priest"]  
print(classTypes)
```

```
classTypes.append("Rogue")  
print(classTypes)
```

```
let warriorIndex = classTypes.index(of: "Warrior")  
let warlockIndex = classTypes.index(of: "Warlock")
```

```
classTypes.remove(at: 1)  
print(classTypes)
```

```
["Mage", "Warrior", "Priest"]  
["Mage", "Warrior", "Priest", "Rogue"]  
["Mage", "Priest", "Rogue"]
```

# Sets

```
var raceSet: Set = ["Human", "Dwarf", "Orc"]  
print(raceSet)
```

```
raceSet.insert("Troll")  
print(raceSet)
```

```
raceSet.insert("Dwarf")  
print(raceSet)
```

```
raceSet.remove("Goblin")  
print(raceSet)
```

```
["Human", "Dwarf", "Orc"]  
["Orc", "Human", "Dwarf", "Troll"]  
["Orc", "Human", "Dwarf", "Troll"]  
["Orc", "Human", "Dwarf", "Troll"]
```

# Dictionaries

```
var weaponSkills = [  
    "Sword": 7, "Staff": 36, "Bow": 17  
]  
print(weaponSkills)
```

```
weaponSkills["Axe"] = 13  
print(weaponSkills)
```

```
["Bow": 17, "Sword": 7, "Staff": 36]
```

```
["Bow": 17, "Sword": 7, "Staff": 36, "Axe": 13]
```

# Dictionaries 2

```
let staffSkills = weaponSkills["Staff"]  
print("I have \(staffSkills) staff skill points.")
```

```
let maceSkills = weaponSkills["Perth"]  
print("I have \(maceSkills) mace skill points.")
```

```
weaponSkills.removeValue(forKey: "Sword")  
print(weaponSkills)
```

```
weaponSkills.removeValue(forKey: "Crossbow")  
print(weaponSkills)
```

I have Optional(36) staff skill points.

I have nil mace skill points.

["Bow": 17, "Staff": 36, "Axe": 13]

["Bow": 17, "Staff": 36, "Axe": 13]

# Decisions

# if

```
var enemyStrength = 37  
  
if enemyStrength > 30 {  
    print("Too big for me to fight.")  
}
```

Too big for me to fight.

# if ... else

```
var anotherEnemy = 24

if anotherEnemy > 30 {
    print("I should run away.")
} else {
    print("I think I can beat this enemy.")
}
```

I think I can beat this enemy.



# if ... else if ... else

```
var enemyIsDead = false
var enemyPetIsDead = true

if enemyIsDead == true {
    print("I WIN!!!!")
} else if enemyPetIsDead == true {
    print("Making progress.")
} else {
    print("This could be a long fight...")
}
```

Making progress.

# switch

```
var newDirection = "S"

switch newDirection {
case "N":
    print("🚶 North")
case "E":
    print("🚶 East")
case "S":
    print("🚶 South")
case "W":
    print("🚶 West")
default:
    print("Just wandering around...")
}
```

🚶 South

# Loops

# for

```
for i in 0 ..< 3 {  
    print(i)  
}
```

```
var classTypes = ["Mage", "Warrior", "Priest"]  
for type in classTypes {  
    print(type)  
}
```

0

1

2

Mage

Warrior

Priest

# while

```
var enemyHealth = 4

while enemyHealth > 0 {
    print(enemyHealth)
    enemyHealth -= 1
}
```

```
4
3
2
1
```

# repeat ... while

```
var myStamina = 0  
  
repeat {  
    print(myStamina)  
    myStamina += 2  
} while myStamina < 10
```

```
0  
2  
4  
6  
8
```

# Looping through Dictionaries

```
var weaponSkills = [  
    "Sword": 7, "Staff": 36, "Bow": 17  
]  
  
var totalSkills = 0  
for (key, value) in weaponSkills {  
    print("My skill in \(key) is \(value).")  
    totalSkills += value  
}  
let averageSkill = totalSkills /  
    weaponSkills.count  
print(averageSkill)
```

```
My skill in Bow is 17.  
My skill in Sword is 7.  
My skill in Staff is 36.  
20
```

# Functions



# Simple Function

```
func showCharacterName() {  
    print("My name is Maximus")  
}  
  
showCharacterName()
```

My name is Maximus

# Function parameters

```
func showCharacterName(name: String) {  
    print("My name is \(name)")  
}  
showCharacterName(name: "Griselda")  
  
func showCharacter(name: String,  
                  hitPoints: Int) {  
    print("My name is \(name)")  
    print("    and I have \(hitPoints) hit  
points.")  
}  
showCharacter(name: "Bob", hitPoints: 28)
```

```
My name is Griselda  
My name is Bob  
    and I have 28 hit points.
```

# Better function calls

```
func changeHitPoints(newHitPoints: Int) {  
    print("Changing hit points to \  
(newHitPoints)")  
}  
changeHitPoints(newHitPoints: 12)  
  
func adjustHitPoints(to newHitPoints: Int) {  
    print("Adjusting hit points to \  
(newHitPoints)")  
}  
adjustHitPoints(to: 34)
```

Changing hit points to 12  
Adjusting hit points to 34

# Returning values

```
func square(of number: Int) -> Int {  
    return number * number  
}
```

```
let result = square(of: 4)  
print(result)
```

# Un-named parameters

```
func convertCtoF(_ degreesC: Double) -> Double {  
    let degreesF = degreesC / 5 * 9 + 32  
    return degreesF  
}
```

```
let boiling = convertCtoF(100)  
let human = convertCtoF(37)
```

```
print(boiling)  
print(human)
```

212.0

98.6

# Default parameters

```
func showInfo(_ name: String,  
              useUpperCase: Bool = false) {  
    var info = "My name is \(name)."  
    if useUpperCase {  
        info = info.uppercased()  
    }  
    print(info)  
}
```

```
showInfo("Griselda", useUpperCase: true)  
showInfo("Bob")
```

```
MY NAME IS GRISELDA.  
My name is Bob.
```

# Optionals

# Optional Variables

```
var characterDescription: String?  
print(characterDescription)
```

```
characterDescription = "Charming but deadly."  
print(characterDescription)
```

```
nil  
Optional("Charming but deadly.")
```



# Un-wrapping Optionals

```
var bonusPoints: Int?  
  
// Un-comment next line to see error  
// let triple1 = bonusPoints * 3  
// error: value of optional type 'Int?' not  
// unwrapped; did you mean to use '!' or '?'?  
  
// Un-comment next line to see error  
// let triple2 = bonusPoints! * 3  
// fatal error: unexpectedly found nil while  
// unwrapping an Optional value  
  
bonusPoints = 7  
let triple3 = bonusPoints! * 3  
print(triple3)
```

# Checking for optionals

```
var otherDescription: String?  
  
if let description = otherDescription {  
    print("\(description)")  
} else {  
    print("No description available.")  
}
```

No description available.

# Programming Style

# Structure

- Pick a naming style and stick to it: camelCase or snake\_case.
- Use descriptive names even if they are long.
- Be consistent with how you place your braces.
- USE WHITE SPACE!
- Comment "why", not "what".
- **Remember that the person who has to work out this code in 6 months time might be you!!!**

# Apple's Guidelines

- Clarity at the point of use is your most important goal.
- Clarity is more important than brevity.
- Concise code is a consequence of using contextual cues.