



SORBONNE
UNIVERSITÉ

Nouvelles technologies du web

LI 385

Introduction to iOS development with Swift



Adrien Humilière

iOS Developer
Engineering Manager

adhumidant@gmail.com



Licence DANT

Promotion 2011 – 2012



Music World Media – Applications musicales



Captain Train (acquis par Trainline) – Vente de billets de train



Music World Media – Applications musicales



Captain Train (acquis par Trainline) – Vente de billets de train



Brut. – Média vidéo en ligne



Captain Train (acquis par Trainline) – Vente de billets de train

Brut.

Brut. – Média vidéo en ligne



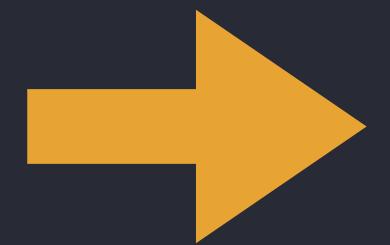
Dashlane – Logiciel de gestion de mots de passe

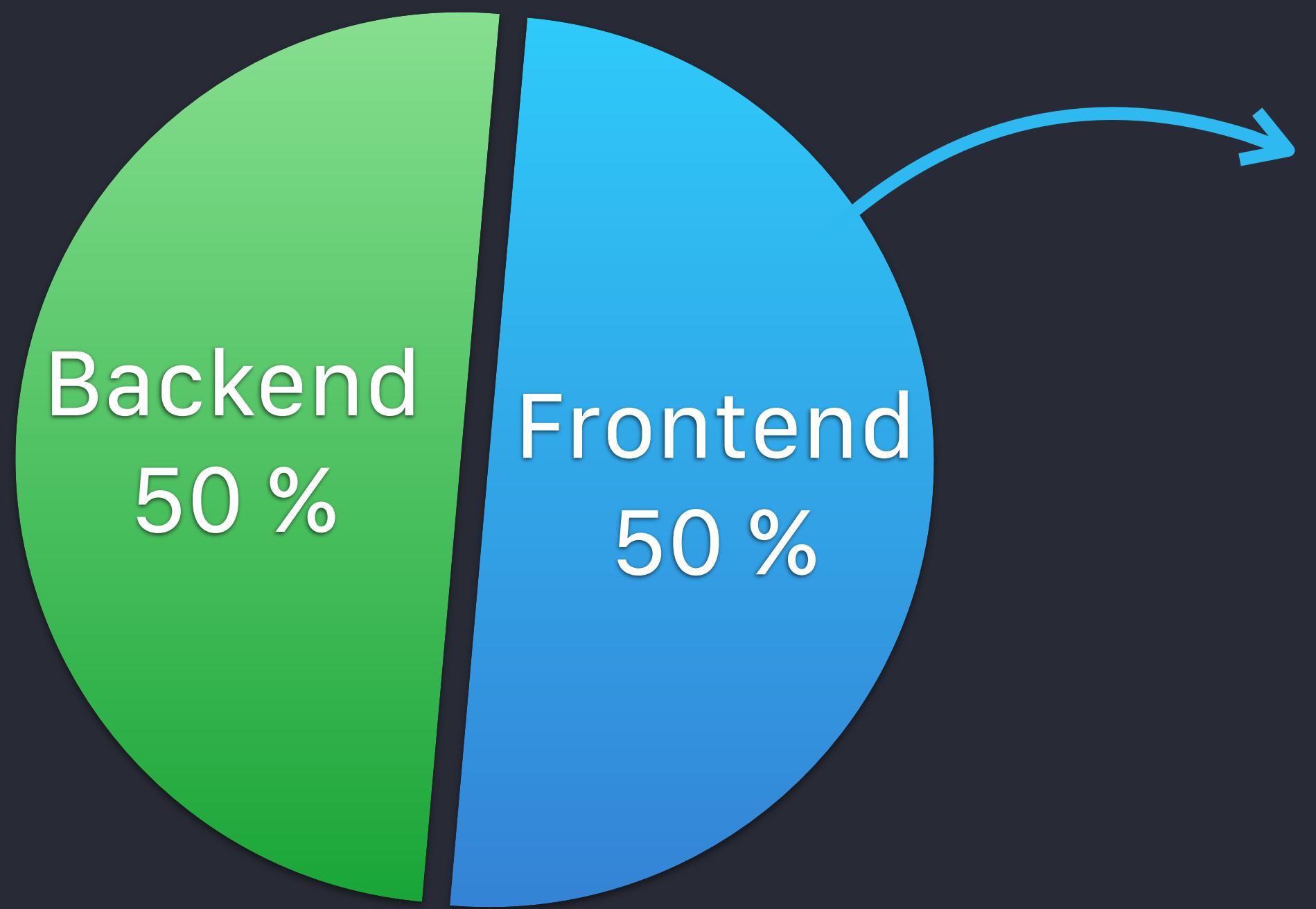
Development tools

Swift 4

User interfaces

iOS SDK





1 project / 2 steps

- Swift library (March to April)
- Application (April to June)

Practice at home

- Have a mac ? **Install Xcode**.
- Swift code can be written and built on Mac, Linux, iPad, and the web.
- Developper account (free) on developer.apple.com needed to build an app on a device.

Practice at university

Salle 14-15, 409

available for you (if not in use)

Introduction to iOS development with Swift

Lesson 1



Adrien Humilière
Dashlane

adhum+dant@gmail.com



- Swift and playgrounds
- Constants, Variables, and Data Types
- Operators
- Control Flow
- Strings
- Functions
- Structures
- Classes and inheritance
- Collections
- Loops

Swift and playgrounds



A modern language

Fast

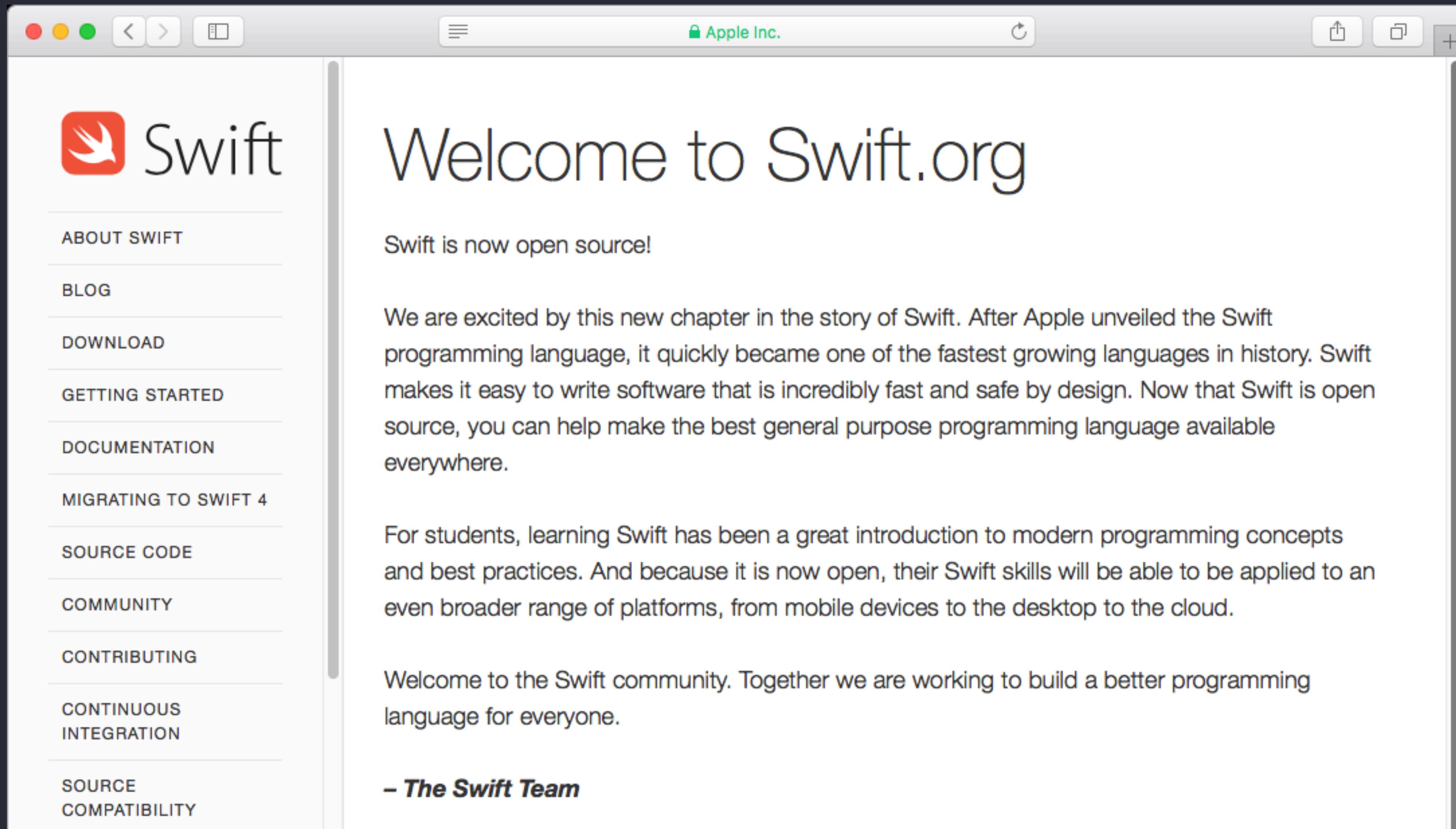
Safe

Expressive

A safe language

- Explicit object « types »
- Type inference
- Optionals
- Error handling

Open Source

A screenshot of a web browser window showing the Swift.org homepage. The browser has a dark mode interface with a light gray header bar. The address bar shows "Apple Inc." with a lock icon. The main content area displays the "Welcome to Swift.org" page. On the left, there is a sidebar with a red square logo containing a white bird icon, followed by the word "Swift". Below the logo is a vertical list of menu items: "ABOUT SWIFT", "BLOG", "DOWNLOAD", "GETTING STARTED", "DOCUMENTATION", "MIGRATING TO SWIFT 4", "SOURCE CODE", "COMMUNITY", "CONTRIBUTING", "CONTINUOUS INTEGRATION", and "SOURCE COMPATIBILITY".

Welcome to Swift.org

Swift is now open source!

We are excited by this new chapter in the story of Swift. After Apple unveiled the Swift programming language, it quickly became one of the fastest growing languages in history. Swift makes it easy to write software that is incredibly fast and safe by design. Now that Swift is open source, you can help make the best general purpose programming language available everywhere.

For students, learning Swift has been a great introduction to modern programming concepts and best practices. And because it is now open, their Swift skills will be able to be applied to an even broader range of platforms, from mobile devices to the desktop to the cloud.

Welcome to the Swift community. Together we are working to build a better programming language for everyone.

– The Swift Team

Hello, world!



```
print("Hello, world!")
```

Playgrounds



Constants, Variables, and Data Types



Constants

- Defined using the let keyword

```
let name = "John"
```

- Defined using the let keyword

```
let pi = 3.14159
```

- Can't assign a constant a new value

```
let name = "John"  
name = "James"
```

Variables

- Defined using the var keyword

```
var age = 29
```

- Can assign a new value to a variable

```
var age = 29
```

```
age = 30
```

Naming constants and variables

- No mathematical symbols
- No spaces
- Can't begin with a number

```
let π = 3.14159
let 一百 = 100
let ⚡ = 6
let mañana = "Tomorrow"
let anzahlDerBücher = 15 //numberOfBooks
```

Naming constants and variables

- Clear and descriptive
- camelCase if multiple words

Types

```
struct Person {  
    let firstName: String  
    let lastName: String  
  
    func sayHello() {  
        print("Hello there! My name is \(firstName) \(lastName).")  
    }  
}
```

Most common types

Int

Double

String

Bool

Type safety

```
let playerName: String = "Julian"  
var playerScore: Int = 1000  
var gameOver: Bool = false  
playerScore = playerName
```



Cannot assign value of type 'String' to type 'Int'

```
var wholeNumber: Int = 30  
var numberWithDecimals: Double = 17.5  
wholeNumber = numberWithDecimals
```



Cannot assign value of type 'Double' to type 'Int'

Type inference

```
let cityName = "San Francisco"  
let pi = 3.1415927
```

Type annotation

```
let cityName: String = "San Francisco"  
let pi: Double = 3.1415927
```

```
let number: Double = 3  
print(number) // ~> 3.0
```

Mandatory type annotation

- When you create a constant or variable before assigning it a value

```
let firstName: String  
//...  
firstName = "Layne"
```

Mandatory type annotation

- When you create a constant or variable that could be inferred as two or more different types

```
let middleInitial: Character = "J"  
var remainingDistance: Float = 30
```

Mandatory type annotation

- When you add properties to a type definition

```
struct Car {  
    let make: String  
    let model: String  
    let year: Int  
}
```

Operators



Assign a value

- Use the = operator to assign a value

```
var favoritePerson = "Luke"
```

- Use the = operator to modify or reassign a value

```
var shoeSize = 8  
shoeSize = 9
```

Basic arithmetic

- You can use the +, -, *, and / operators to perform basic math functions

```
var opponentScore = 3 * 8  
var myScore = 100 / 4
```

Basic arithmetic

→ Use Double values for decimal precision

```
let totalDistance = 3.9
var distanceTravelled = 1.2
var remainingDistance = totalDistance - distanceTravelled
print(remainingDistance) // ~> 2.7
```

Basic arithmetic

```
let x = 51
let y = 4
let z = x / y
print(z) // ~> 12
```

Basic arithmetic

```
let x: Double = 51
let y: Double = 4
let z = x / y
print(z) // ~> 12.75
```

Compound assignment

```
var myScore = 10  
myScore = myScore + 3
```

```
myScore += 3  
myScore -= 5  
myScore *= 2  
myScore /= 2
```

Numeric type conversion

```
let x = 3
let y = 0.1415927
let pi = x + y    ← ! Binary operator '+' cannot be applied to operands of type 'Int' and 'Double'
```

```
let x = 3
let y = 0.1415927
let pi = Double(x) + y
```



Control Flow



Logical operators

==	Two items must be equal
!=	The values must not be equal to each other
>	Value on the left must be greater than the value on the right
>=	Value on the left must be greater than or equal to the value on the right
<	Value on the left must be less than the value on the right
<=	Value on the left must be less than or equal to the value on the right
&&	AND—The conditional statement on the left and right must be true
	OR—The conditional statement on the left or right must be true
!	Returns the opposite of the conditional statement immediately following the operator

if statements

```
if condition {  
    code  
}
```

```
let temperature = 100  
if temperature >= 100 {  
    print("The water is boiling.")  
}
```

if-else statements

```
if condition {  
    code  
} else {  
    code  
}
```

```
let temperature = 100  
if temperature >= 100 {  
    print("The water is boiling.")  
} else {  
    print("The water is not boiling.")  
}
```

switch statement

```
switch value {  
    case n:  
        code  
    case n:  
        code  
    case n:  
        code  
    default:  
        code  
}
```

switch statement

```
let numberOfWheels = 2
switch numberOfWheels {
    case 1:
        print("Unicycle")
    case 2:
        print("Bicycle")
    case 3:
        print("Tricycle")
    case 4:
        print("Quadcycle")
    default:
        print("That's a lot of wheels!")
}
```

switch statement

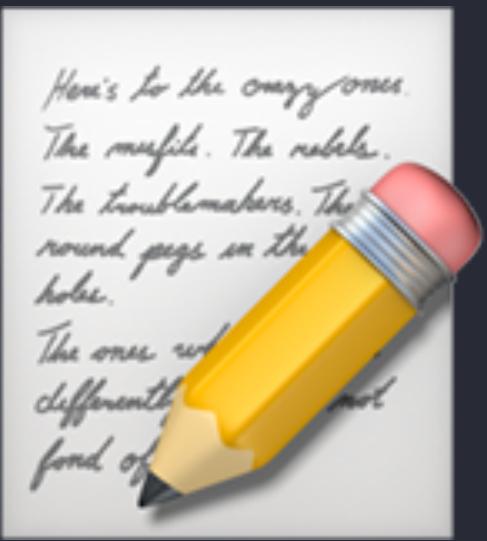
```
let character = "z"

switch character {
    case "a", "e", "i", "o", "u", "y":
        print("This character is a vowel.")
    default:
        print("This character is not a vowel.")
}
```

switch statement

```
switch distance {  
    case 0...9:  
        print("Your destination is close.")  
    case 10...99:  
        print("Your destination is a medium distance from here.")  
    case 100...999:  
        print("Your destination is far from here.")  
    default:  
        print("Are you sure you want to travel this far?")  
}
```

Strings



Basics

```
let greeting = "Hello"  
var otherGreeting = "Salutations"
```

```
let joke = """  
Q: Why did the chicken cross the road?  
A: To get to the other side!  
"""  
  
print(joke)  
// Q: Why did the chicken cross the road?  
// A: To get to the other side!
```

Basics – escaping

```
let greeting = "It is traditional in programming to print  
\"Hello, world!\""
```

\ "

Double quote

\ \

Backslash

\t

Tab

\r

Carriage return (return to beginning of the next line)

Basics – Empty

```
var myString = ""  
  
if myString.isEmpty {  
    print("The string is empty")  
}
```

Basics – Characters

```
let a = "a" // 'a' is a string
let b: Character = "b" // 'b' is a Character
```

Concatenation

```
let string1 = "Hello"  
let string2 = ", world!"  
var myString = string1 + string2 // "Hello, world!"  
  
myString += " Hello!" // "Hello, world! Hello!"
```

Interpolation

```
let name = "Rick"  
let age = 30  
print("\$(name) is \$age years old")
```

// Rick is 30 years old

```
let a = 4  
let b = 5  
print("If a is \$a and b is \$b, then a + b equals \$a+\$b")
```

String equality and comparison

```
let name = "Johnny Appleseed"
if name.lowercased() == "joHnnY aPPleseeD".lowercased() {
    print("The two names are equal.")
}
```

String equality and comparison

```
let greeting = "Hello, world!"  
  
print(greeting.hasPrefix("Hello"))  
print(greeting.hasSuffix("world!"))  
print(greeting.hasSuffix("World!"))
```

String equality and comparison

```
let greeting = "Hi Rick, my name is Amy."  
if greeting.contains("my name is") {  
    print("Making an introduction")  
}
```

String equality and comparison

```
let name = "Ryan Mears"
let count = name.count
let newPassword = "1234"

if newPassword.count < 8 {
    print("This password is too short. Passwords should have
at least 8 characters.")
}
```

String equality and comparison

```
let cow = "🐮"  
let credentials = "résumé"  
let myBook = "私の本"  
print("∞".characters.count)
```

String equality and comparison

```
let someCharacter: Character = "e"
switch someCharacter {
    case "a", "e", "i", "o", "u":
        print("\u{someCharacter} is a vowel.")
    default:
        print("\u{someCharacter} is not a vowel.")
}
```

Functions



```
tieMyShoes()
```

```
makeBreakfast(food: "scrambled eggs", drink: "orange juice")
```

Defining a function

```
func functionName (parameters) -> ReturnType {  
    // Body of the function  
}
```

Defining a function

```
func displayPi() {  
    print("3.1415926535")  
}  
  
displayPi() // 3.1415926535
```

Parameters

```
func triple(value: Int) {  
    let result = value * 3  
    print("If you multiply \(value) by 3, you'll get \  
        (result).")  
}  
  
triple(value: 10) // If you multiply 10 by 3, you'll get 30.
```

Multiple parameters

```
func multiply(firstNumber: Int, secondNumber: Int) {  
    let result = firstNumber * secondNumber  
    print("The result is \(result).")  
}
```

```
multiply(firstNumber: 10, secondNumber: 5)  
// The result is 50.
```

Return values

```
func multiply(firstNumber: Int, secondNumber: Int) -> Int {  
    let result = firstNumber * secondNumber  
    return result  
}
```

Return values

```
func multiply(firstNumber: Int, secondNumber: Int) -> Int {  
    return firstNumber * secondNumber  
}
```

```
let myResult = multiply(firstNumber: 10, secondNumber: 5)  
print("10 * 5 is \(myResult)")
```

```
print("10 * 5 is \(multiply(firstNumber: 10, secondNumber: 5))")
```

Argument labels

```
func sayHello(firstName: String) {  
    print("Hello, \(firstName)!")  
}  
  
sayHello(firstName: "Amy")
```

Argument labels

```
func sayHello(to: String, and: String) {  
    print("Hello \(to) and \(and)")  
}  
  
sayHello(to: "Luke", and: "Dave")
```

Argument labels

```
func sayHello(to person: String, and otherPerson: String) {  
    print("Hello \(person) and \(otherPerson)")  
}  
  
sayHello(to: "Luke", and: "Dave")
```

Argument labels

```
print("Hello, world!")
```

```
func add(_ firstNumber: Int, to secondNumber: Int) -> Int {  
    return firstNumber + secondNumber  
}
```

```
let total = add(14, to: 6)
```

Default parameter values

```
func display(teamName: String, score: Int = 0) {  
    print("\(teamName): \(score)")  
}  
  
display(teamName: "Wombats", score: 100)  
display(teamName: "Wombats")
```

Structures



```
struct Person {  
    var name: String  
}
```

- Capitalize type names
- Use lowercase for property names

Accessing property values

```
struct Person {  
    var name: String  
}  
  
let person = Person(name: "Jasmine")  
print(person.name) // Jasmine
```

Adding functionality

```
struct Person {  
    var name: String  
  
    func sayHello() {  
        print("Hello there! My name is \(name)!")  
    }  
  
}  
  
let person = Person(name: "Jasmine")  
person.sayHello() // Hello there! My name is Jasmine!
```

Instances

```
struct Shirt {  
    var size: String  
    var color: String  
}  
  
let myShirt = Shirt(size: "XL", color: "blue")  
  
let yourShirt = Shirt(size: "M", color: "red")
```

```
struct Car {  
    var brand: String  
    var year: Int  
    var color: String  
  
    func startEngine() {...}  
  
    func drive() {...}  
  
    func park() {...}  
  
    func steer(direction: Direction) {...}  
}  
  
let firstCar = Car(brand: "Peugeot", year: 2010, color: "blue")  
let secondCar = Car(brand: "Ford", year: 2013, color: "black")  
  
firstCar.startEngine()  
firstCar.drive()
```

Initializers

```
let string = String.init() // ""
let integer = Int.init() // 0
let bool = Bool.init() // false
```

Initializers

```
let string = String() // ""
let integer = Int() // 0
let bool = Bool() // false
```

Default values

```
struct Odometer {  
    var count: Int = 0  
}
```

```
let odometer = Odometer()  
print(odometer.count) // 0
```

```
let odometer = Odometer(count: 27000)  
print(odometer.count) // 27000
```

```
struct Person {  
    let name: String  
    let age: Int  
}
```

```
let aPerson = Person(name: "Adrien", age: 32)
```

```
struct Car {  
    let brand: String  
    let year: Int  
    let color: String  
}
```

```
let firstCar = Car(brand: "Honda", year: 2010, color: "blue")
```

Custom initializers

```
struct Temperature {  
    var celsius: Double  
}  
  
let temperature = Temperature(celsius: 30.0)
```

```
let fahrenheitValue = 98.6  
let celsiusValue = (fahrenheitValue - 32) / 1.8  
  
let newTemperature = Temperature(celsius: celsiusValue)
```

```
struct Temperature {  
    var celsius: Double  
  
    init(celsius: Double) {  
        self.celsius = celsius  
    }  
  
    init(fahrenheit: Double) {  
        celsius = (fahrenheit - 32) / 1.8  
    }  
}  
  
let tempFromCelsius = Temperature(celsius: 18.5)  
let tempFromFahrenheit = Temperature(fahrenheit: 212.0)
```


Instance methods

```
struct Size {  
    var width: Double  
    var height: Double  
  
    func area() -> Double {  
        return width * height  
    }  
}  
  
var someSize = Size(width: 10.0, height: 5.5)  
  
let area = someSize.area() // Area is assigned a value of 55.0
```

Mutating methods

```
struct Odometer {  
    var count: Int = 0 // Assigns a default value to the 'count'  
}
```

Need to:

- Increment the mileage
- Reset the mileage

```
struct Odometer {  
    var count: Int = 0  
  
    mutating func increment() {  
        count += 1  
    }  
  
    mutating func increment(by amount: Int) {  
        count += amount  
    }  
  
    mutating func reset() {  
        count = 0  
    }  
}
```

Computed properties

```
struct Temperature {  
    let celsius: Double  
    let fahrenheit: Double  
    let kelvin: Double  
}  
  
let temperature = Temperature(celsius: 0, fahrenheit: 32, kelvin: 273.15)
```

```
struct Temperature {  
    var celsius: Double  
    var fahrenheit: Double  
    var kelvin: Double  
  
    init(celsius: Double) {  
        self.celsius = celsius  
        fahrenheit = celsius * 1.8 + 32  
        kelvin = celsius + 273.15  
    }  
  
    init(fahrenheit: Double) {  
        self.fahrenheit = fahrenheit  
        celsius = (fahrenheit - 32) / 1.8  
        kelvin = celsius + 273.15  
    }  
  
    init(kelvin: Double) {  
        self.kelvin = kelvin  
        celsius = kelvin - 273.15  
        fahrenheit = celsius * 1.8 + 32  
    }  
}
```

Computed properties

```
struct Temperature {  
    let celsius: Double  
  
    var fahrenheit: Double {  
        return celsius * 1.8 + 32  
    }  
  
    var kelvin: Double {  
        return celsius + 273.15  
    }  
}
```

Property observers

```
struct StepCounter {  
    var totalSteps: Int = 0 {  
        willSet {  
            print("About to set totalSteps to \(newValue)")  
        }  
        didSet {  
            if totalSteps > oldValue {  
                print("Added \(totalSteps - oldValue) steps")  
            }  
        }  
    }  
}
```

Property observers

```
var stepCounter = StepCounter()  
stepCounter.totalSteps = 40  
stepCounter.totalSteps = 100  
  
// About to set totalSteps to 40  
// Added 40 steps  
// About to set totalSteps to 100  
// Added 60 steps
```

Type properties and methods

```
struct Temperature {  
    static var boilingPoint = 100.0  
  
    static func convertedFromFahrenheit(_ temperatureInFahrenheit:  
Double) -> Double {  
        return(((temperatureInFahrenheit - 32) * 5) / 9)  
    }  
  
}  
  
let boilingPoint = Temperature.boilingPoint  
let currentTemperature = Temperature.convertedFromFahrenheit(99)  
let positiveNumber = abs(-4.14)
```

Copying

```
var someSize = Size(width: 250, height: 1000)
var anotherSize = someSize

someSize.width = 500

print(someSize.width)
print(anotherSize.width)
```

self

```
struct Car {  
    var color: Color  
  
    var description: String {  
        return "This is a \(self.color) car."  
    }  
}
```

self

```
struct Car {  
    var color: Color  
  
    var description: String {  
        return "This is a \(color) car."  
    }  
}
```

- Not required when property or method names exist on the current object

self

```
struct Temperature {  
    var celsius: Double  
  
    init(celsius: Double) {  
        self.celsius = celsius  
    }  
}
```

adhum.i.fr/teaching

Credentials

- login: m2sar
- password: sarM2