iOS development

iPhone OS 1, 2007

June 29, 2007, $499 (4GB) / $599 (8GB) iPhones

- SMS (Texting), Calendar, Photos, Camera, YouTube, Stocks, Maps, Weather, Clock, Calculator, Notes, Phone, Mail, Safari, iPod, and Settings.
- missing was the App Store or any way for developers to create apps

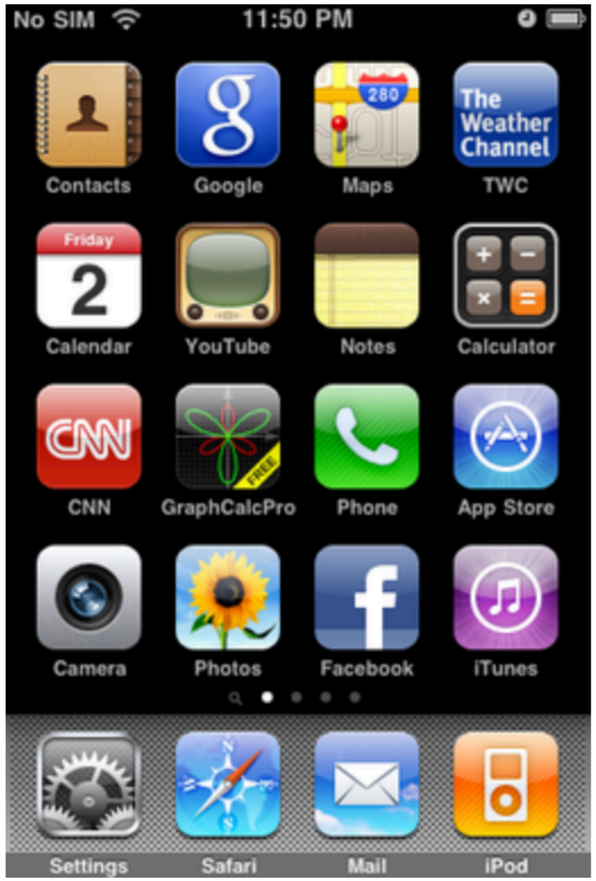
iPhone OS 2

March 2008

- enterprise-level features for businesses such as push email, calendar and contacts, VPN, and ActiveSync and other Microsoft integrations, and more
- The SDK was based on the Cocoa libraries used to create Mac OS X apps, but revamped and redesigned for touch screens.
- the initial SDK wasn’t much and many developers resorted to still making jail broken apps that could support additional features that Apple didn’t allow on the official App Store

iPhone OS 3

- Core Data, no longer did you need to learn SQLite and spend hours debugging why something wasn’t saving
- Apple released Game Kit for creating games that could play with nearby devices, MapKit for taking advantage of Google maps in apps, and new media features like working with AVRecorder



iOS 4

June 21, 2010 and was the first version of iOS that dropped “iPhone” from the name and started going by “iOS” only

- User wallpapers (yes, 4 versions in for this feature, and yes, this is the most important feature)
- Multitasking for Internet calling, location use, and audio playback using “Fast App Switching”
- System-wide spell checking
- Game Center
- FaceTime for video calling
- Digital zoom for the camera with 5x zoom
- Home screen added support for folders, increasing the number of apps that could be shown on the Home Screen from 180 to 2,160

iOS 5

After iOS 5, you could set up, backup, and install updates for your iPhone without the need for a Mac or Windows computer. Siri!

- Notifications were revamped and Notification Center was introduced to show recent notifications
- iCloud was introduced with iCloud wireless backups, contact syncing, and more, and replaced Apple's MobileMe features
- iMessage was introduced to provide user-to-user rich text instant messaging through the Messages app
- Newsstand was introduced and developers were given access to provide Newsstand apps that could be updated from the background to provide updated magazine stories, newspapers, and more.
- Reminders app was introduced that allowed users to created lists of reminders for later and use location-based reminders

iOS 6

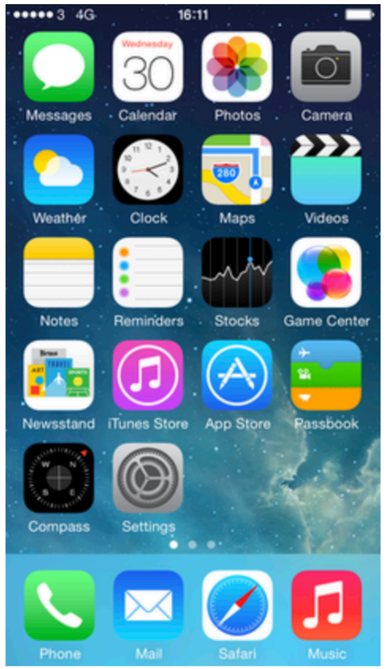
First without Steve Jobs, Apple's own Maps app

- A dedicated Podcasts app
- Passbook, which allowed developers to write passes that could be updated and provide details to users through the app like tickets, member cards, and more (the precursor to the Wallet app)
- Like Twitter integration the year before, the Social framework added Facebook integration and contact syncing
- New privacy controls allowed users to fine-grain control things like advertising, camera access, and more
- Developers got AutoLayout, changing the paradigm for laying out user interfaces in Interface Builder (goodbye struts and springs!)

iOS 7

2013, was the first version of iOS that included flat design that was redesigned from the ground up by Jony Ive

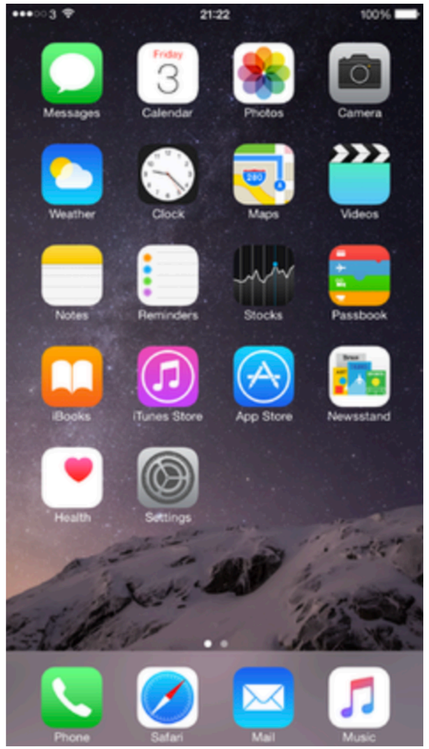
- CarPlay integration
- AirDrop
- Automatic App Store updates and automatic iOS updates
- Control Center for quickly accessing commonly used settings on the iPhone and iPad
- App Switcher for quickly accessing previously opened apps (before this was a bar at the bottom of the screen that didn’t show an app preview)



iOS 8

2014, incorporate shared SMS and phone calls between Mac and iPhone and iPad

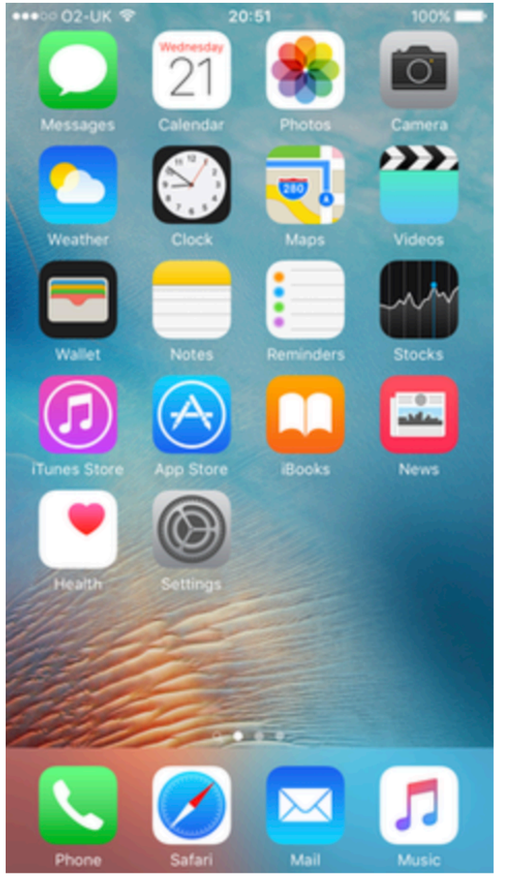
- Extensions for sharing content in Apple apps (such as Photo extensions)
- Widgets in the Notification Center
- Ability to make third party keyboards that could finally replace the default iOS keyboard
- Health app that could aggregate data from amongst third party apps
- iCloud Drive for storing files, and CloudKit for a zero-API way for developers to write an deploy cloud data services
- Biometrics frame that allowed for supporting Touch ID in third party apps
- HomeKit, which allowed licensed manufacturers to produce HomeKit equipment and manage it through the HomeKit gateway, though there was not yet a Home app
- Safari was updated to support WebGL graphics



iOS 9

2015, dedicated this release to fixing some performance issues

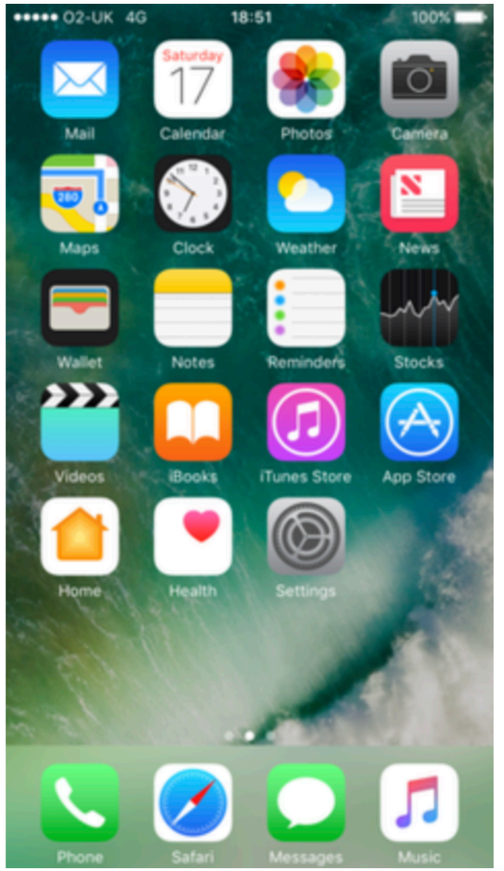
- The short-lived 3D feature on supported devices (this is now replaced by long press gestures for “peeking” and “popping” views)
- Battery savings with a feature to not light the screen for notifications when placed face-down and also the introduction of Low Power mode
- San Francisco was announced as the new system font replacing Helvetica Neue
- Multitasking changes allowed Picture-in-Picture (PiP) for iPad, and also slide over and split screen features for the first time on iOS. This made use of Apple’s much-touted size classes in AutoLayout
- Performance improvements were expanded to the entire system through the introduction of Metal and iOS 9 made use of Metal to handle the core user interface elements and graphics rendering
- On the security front, Apple stopped using 4-digits as the standard passcode option and moved to 6-digits instead. Apple also supported 2FA (Two Factor Authentication) for the first time in iCloud and Apple IDs



iOS 10

2016, Siri integrations and iMessage extensions were huge then and continue to be huge now.

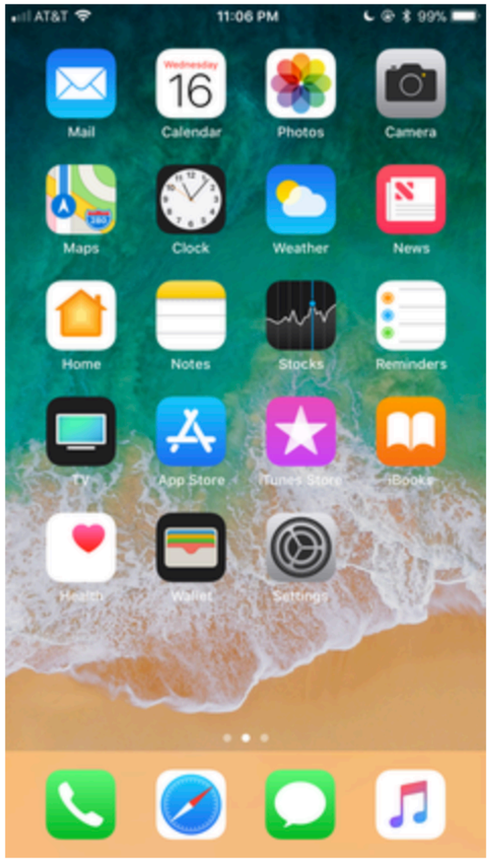
- The last version of iOS to support 32-bits apps — it's a 64-bit world hereafter, and Apple started warning users about apps that hadn't been updated upon launch
- Apple introduced the Home app to manage HomeKit equipment, Sherlocking a new HomeKit apps that had previously done this job
- A Universal Clipboard feature used iCloud and Continuity to share a clipboard between iOS and Mac device; it also allowed Apple TV's to use an iPhone to enter text



iOS 11

2017

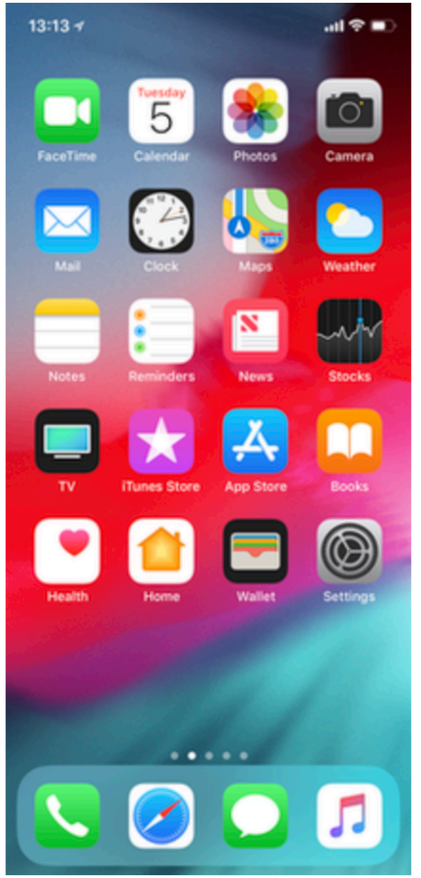
- Do very powerful stuff on-device instead of in the cloud where you risk user privacy.
- ARKit, Apple’s augmented reality framework that made use of the advanced CPUs and GPUs available on iOS devices running the A9 chipset and newer.
- Apple Also unlocked Core NFC framework that let developers write apps that could finally take advantage of the NFC reader built into iPhones that support Apple Pay.
- We said goodbye to 32-bit apps, which was bittersweet for the apps no longer supported but still useable until now
- iOS 11 also dropped the Social Frameworks introduced in iOS 5 by dropping native support for Twitter, Facebook, Flickr, and Vimeo.
- iCloud Drive app was removed and replaced by the Files app, which included APIs for integrating third party file apps into



iOS 12

2018, devices launched apps 40%, Screen Time

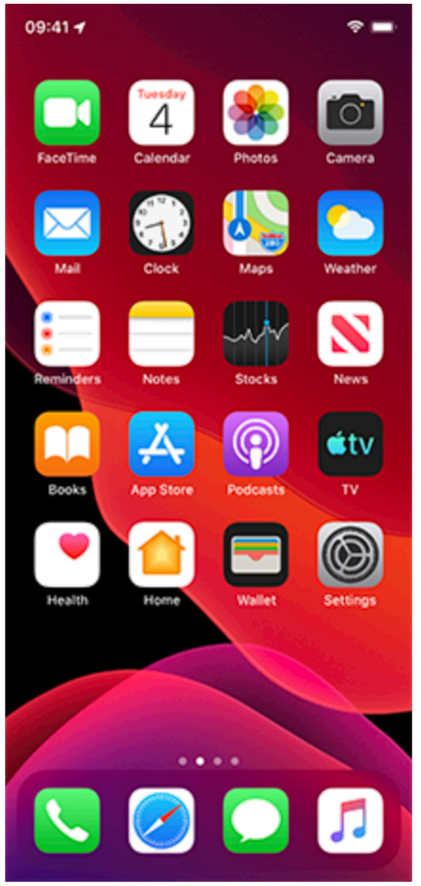
- Shortcuts app that allowed Siri to perform actions that the user defined in the app. This was a revamped version of the Workflow app that Apple had acquired the year before.
- ARKit2 with 3D object detection
- CarPlay third party app integration for Waze, Google Maps, and more
- Trackpad mode in the Keyboard for 3D Touch devices



iOS 13

2019, privacy

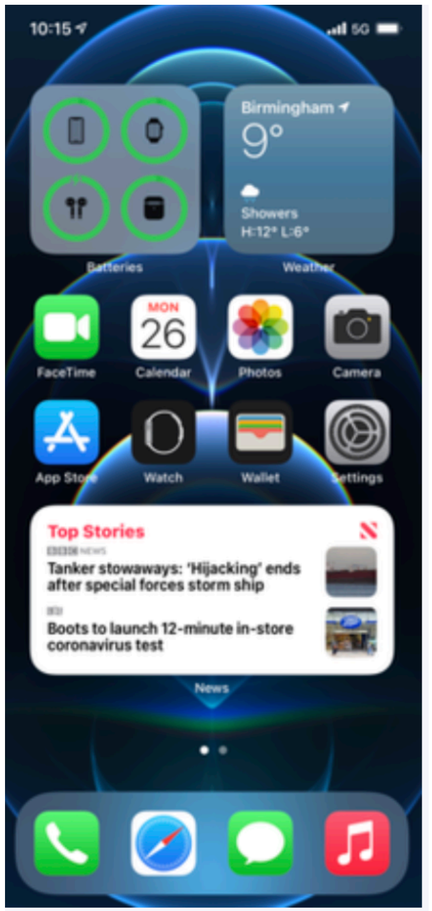
- iPad and iOS have now split their codebases. iPad now runs iPadOS while iPhones and iPod touch run iOS
- Sign In with Apple is a new Single Sign On (SSO) option, and a requirement if SSO is available in third party apps from Google or other providers
- External storage support in the Files app to read and write files in the Files app and third party apps with connected USB mass storage devices
- Further performance improvements, which included a new app download format making 50% smaller apps and app launches up to 2x as fast as before



iOS 14

2020

- App Clips are a new feature expanding on the functionality of the App Store. Intended as a dynamic feature rather than a permanently installed app
- Car keys allow an iPhone to act as a virtual car key using NFC technology with compatible cars.
- To the left of the first page, the Today View is replaced by a scrollable widget UI. Widgets may be placed on the home screen to sit amongst app icons
- Improvements to the Search feature on the home screen were made, including a refined UI, quick launcher for apps, more detailed web search, shortcuts to in-app search, and improved as-you-type search suggestions.[
- Wi-Fi MAC address randomization



iOS 15

2021

- The Focus allows users to set their "state," such as work, sleep, do not disturb or a custom focus
- Notifications receive a new look with contact photos for all communication apps and larger app icons.
- Devices with an A12 chip or later support Live Text in all apps which can transcribe text from the camera in the real world, images, photos
- The new version allows apps to build more immersive AR experiences using new APIs to capture objects even faster, custom shaders, dynamic assets, custom systems and character control.
- Safari was completely redesigned, moving the tab bar and address bar to the bottom of the screen.

Android vs iOS

- Abstracting from specific technological differences:
- users’ behaviour: fewer users, but spending approx. 2x much on Apps consistent hardware - 9 devices vs. thousands
- typically need to support just last 2 iOS versions
- these factors result in approx. 30% shorter development time

Monetization $$$

Apple typically has 30% margin. What about Google Play?

1 Paid Apps

    fixed price tiers, same price in all countries

2 In-App Purchases (freemium model)

    selling via website is possible but it may not be linked from App

3 Subscriptions

4 Auto-Renewable Subscriptions

    Apple’s share drops to 15% after first subscription year

5 Apple Pay

    real world products only

    Apple’s margin is approx. 2%

6 In App Ads: iAds, Facebook Ads, Google Ads, ...

7 Affiliates, B2B (most profitable model)

Swift?

Objective C vs Swift

Example (Objective-C Method Signature)

```
+ (NSString *)myMethod:(NSString *)firstArg and:(NSArray
*)secondArg { ... }
```

Example (Swift Method Signature)

```
func myMethod(firstArg: String, secondArg: [Int]) -> String {
... }
```

- Swift
  - much newer, modern syntax and features, less verbose, no header files
  - build with functional programming in mind: Closures (lambda expressions) are first-class members
  - has generics, strongly typed language
  - does not have null pointer
  - syntax similar to major languages, such as Java or C# faster development

- ObjectiveC
  - more stable, Swift changes syntax a little with every major release still has more libraries

Architecture

Model View Controller

Model

- The central component of the pattern. It is the application's dynamic data structure, independent of the user interface. It directly manages the data, logic and rules of the application.

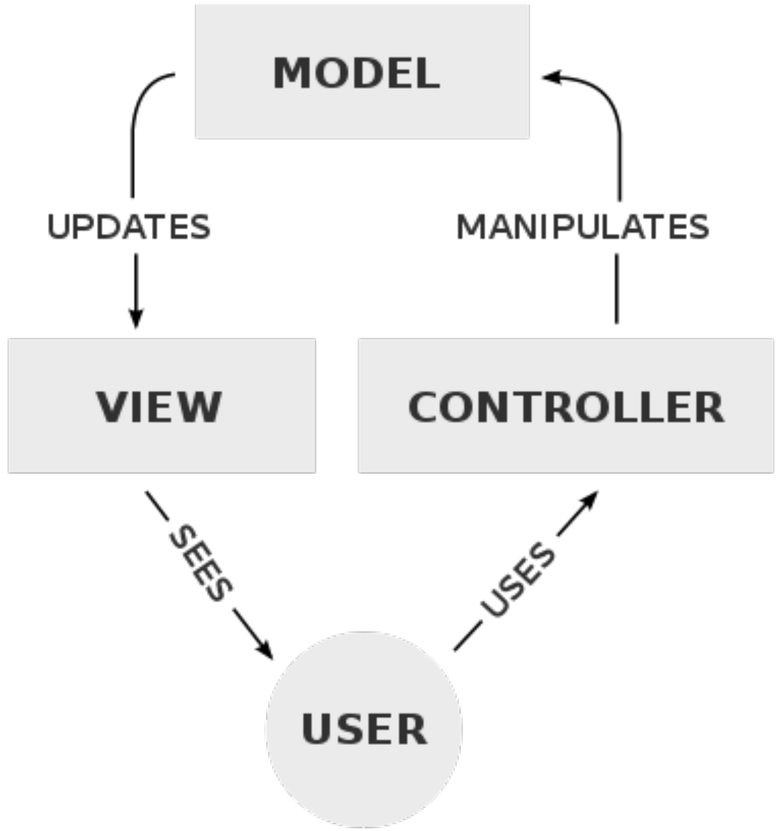
View

- Any representation of information such as a chart, diagram or table. Multiple views of the same information are possible, such as a bar chart for management and a tabular view for accountants.

Controller

- Accepts input and converts it to commands for the model or view.

- The model is responsible for managing the data of the application. It receives user input from the controller.
- The view renders presentation of the model in a particular format.
- The controller responds to the user input and performs interactions on the data model objects. The controller receives the input, optionally validates it and then passes the input to the model.



Declaring Constants and Variables

```swift
// Declaring a constant using the let keyword
let double: Double = 2.0
// double = 3.0 // Error: You can't reassign a let
let inferredDouble = 2.0 // Inferred as a Double

// Declaring a variable using the var keyword
var mutableInt: Int = 1
mutableInt = 2 // OK: You can reassign a var
```

Numeric Type Conversion

```
let integerValue = 8
let doubleValue = 8.0
// let sum = integerValue + double
// Error: type mismatch

// Use an opt-in approach that prevents hidden
// conversion errors and helps make type conversion
// intentions explicit
let sum = Double(integerValue) + double
// OK: Both values have the same type
```

Strings

```swift
// Using a string literal as an initial value for
// a constant or variable
let helloWorld = "Hello, World!"

// Using a multiline string literal to span
// over several lines
let helloWorldProgram = """
A "Hello, World!" program generally is a computer
program that outputs or displays the message
"Hello, World!"
"""

// Empty string
let emptyString = "" // Using string literal
// Initializer syntax
let anotherEmptyString = String()

// Mutating a string
var mutableString = "Swift"
mutableString += " is awesome!"

// String interpolation
// Interpolating a Double
print("The value is \(double)")
// Interpolating a String
print("This is my opinion: \(mutableString)")
```

Tuples

```
// Group multiple values into
// a single compound value
let httpError = (503, "Server Error")

// Decomposing a tuple's contents
let (code, reason) = httpError
// Another way to decompose
let codeByIndex = httpError.0
let reasonByIndex = httpError.1
// Ignoring parts of the tuple using _
let (_, justTheReason) = httpError
```

Optionals

```
// catchphrase can hold a String or nil
var catchphrase: String? // Automatically set to nil
catchphrase = "Hey, what's up, everybody?"

// Forced unwrapping operator (!)
// count1 contains catchphrase's count if
// catchphrase isn't nil; crashes otherwise
let count1: Int = catchphrase!.count

// Optional binding
// If the optional Int returned by
// catchphrase?.count contains a value,
// set a new constant called count to the value
// contained in the optional
if let count = catchphrase?.count {
  print(count)
}

// Coalescing operator (??)
// count2 contains catchphrase's count if
// catchphrase isn't nil; 0 otherwise
let count2: Int = catchphrase?.count ?? 0

// Chaining operator (?)
// count3 contains catchphrase's count if
// catchphrase isn't nil; nil otherwise
let count3: Int? = catchphrase?.count

// Implicitly unwrapped optionals
let forcedCatchphrase: String! =
  "Hey, what's up, everybody?"
let implicitCatchphrase = forcedCatchphrase
// No need for an exclamation mark
```

Collection Types: Array

```
let immutableArray: [String] = ["Alice", "Bob"]
// Type of mutableArray inferred as [String]
var mutableArray = ["Eve", "Frank"]
// Test the membership
let isEveThere = immutableArray.contains(“Eve")
// Access by index
let name: String = immutableArray[0]
// Update item in list;
// crashes if the index is out of range
mutableArray[1] = "Bart"
// immutableArray[1] = "Bart" // Error: can't change
mutableArray.append("Ellen") // Add an item
// Add an item at index
mutableArray.insert("Gemma", at: 1)
// Delete by index
let removedPerson = mutableArray.remove(at: 1)
// You can't reassign a let collection nor change
// its content; you can reassign a var collection
// and change its content
mutableArray = ["Ilary", "David"]
mutableArray[0] = “John"
```

Collection Types: Dictionary

```
let immutableDict: [String: String] =
  ["name": "Kirk", "rank": "captain"]
// Type of mutableDict inferred as [String: String]
var mutableDict =
  ["name": "Picard", "rank": "captain"]
// Access by key, if the key isn't found returns nil
let name2: String? = immutableDict["name"]
// Update value for key
mutableDict["name"] = "Janeway"
// Add new key and value
mutableDict["ship"] = "Voyager"
// Delete by key, if the key isn't found returns nil
let rankWasRemoved: String? =
mutableDict.removeValue(forKey: "rank")
```

Collection Types: Set

```
// Sets ignore duplicate items, so immutableSet
// has 2 items: "chocolate" and "vanilla"
let immutableSet: Set =
  ["chocolate", "vanilla", "chocolate"]
var mutableSet: Set =
  ["butterscotch", "strawberry"]
// A way to test membership
immutableSet.contains("chocolate")
// Add item
mutableSet.insert("green tea")
// Remove item, if the item isn't found returns nil
let flavorWasRemoved: String? =
mutableSet.remove("strawberry")
```

Control Flow: Loops

```swift
// Iterate over list or set
for item in listOrSet {
    print(item)
}

// Iterate over dictionary
for (key, value) in dictionary {
    print("\(key) = \(value)")
}

// Iterate over ranges

// Closed range operator (...)
for i in 0...10 {
  print(i) // 0 to 10
}
// Half-open range operator (..<)
for i in 0..<10 {
  print(i) // 0 to 9
}

// while
var x = 0
while x < 10 {
  x += 1
  print(x)
}

// repeat-while
repeat {
  x -= 1
  print(x)
} while(x > 0)
```

Control Flow: Conditionals

```swift
// Using if to choose different paths
let number = 88
if (number <= 10) {
  // If number <= 10, this gets executed
} else if (number > 10 && number < 100) {
  // If number > 10 && number < 100,
  // this gets executed
} else {
  // Otherwise this gets executed
}

// Ternary operator
// A shorthand for an if-else condition
let height = 100
let isTall = height > 200 ? true : false
```

```swift
// Using guard to transfer program control
// out of a scope if one or more conditions
// aren't met
for n in 1...30 {
  guard n % 2 == 0 else {
    continue
  }
  print("\(n) is even")
}

// Using switch to choose different paths
let year = 2012
switch year {
case 2003, 2004:
  // Execute this statement if year is 2003 or 2004
  print("Panther or Tiger")
case 2010:
  // Execute this statement if year is exactly 2010
  print("Lion")
case 2012...2015:
  // Execute this statement if year is
  // within the range 2012-2015,
  // range boundaries included
  print("Mountain Lion through El Captain")
default:
  // Every switch statement must be exhaustive
  print("Not already classified")
}
```

Functions

```swift
// A Void function
func sayHello() {
  print("Hello")
}

// Function with parameters
func sayHello(name: String) {
  print("Hello \(name)!")
}

// Function with default parameters
func sayHello(name: String = "Lorenzo") {
  print("Hello \(name)!")
}

// Function with mix of default and
// regular parameters
func sayHello(name: String = "Lorenzo", age: Int) {
  print("\(name) is \(age) years old!")
}
// Using just the non default value
sayHello(age: 35)
```

```swift
// Function with parameters and return value
func add(x: Int, y: Int) -> Int {
  return x + y
}
let value = add(x: 8, y: 10)

// If the function contains a single expression,
// the return value can be omitted
func multiply(x: Int, y: Int) -> Int {
  x * y
}

// Specifying arguments labels
func add(x xVal: Int, y yVal: Int) -> Int {
  return xVal + yVal
}

// Omitting the argument label for one
// (or more) parameters
func add(_ x: Int, y: Int) -> Int {
    return x + y
}
let value = add(8, y: 10)

// A function that accepts another function
func doMath(operation: (Int, Int) -> Int, a: Int, b:
Int) -> Int {
  return operation(a, b)
}
```

Closures

```
let adder: (Int, Int) -> Int = { (x, y) in x + y }

// Closures with shorthand argument name
let square: (Int) -> Int = { $0 * $0 }

// Passing a closure to a function
let addWithClosure = doMath(operation: adder, a: 2,
b: 3)
```

Enumerations

```
enum Taste {
  case sweet, sour, salty, bitter, umami
}
let vinegarTaste = Taste.sour

// Iterating through an enum class
enum Food: CaseIterable {
  case pasta, pizza, hamburger
}

for food in Food.allCases {
  print(food)
}

// enum with String raw values
enum Currency: String {
  case euro = "EUR"
  case dollar = "USD"
  case pound = "GBP"
}

// Print the backing value
let euroSymbol = Currency.euro.rawValue
print("The currency symbol for Euro is \
(euroSymbol)")

// enum with associated values
enum Content {
  case empty
  case text(String)
  case number(Int)
}

// Matching enumeration values with a switch
statement
let content = Content.text("Hello")
switch content {
case .empty:
  print("Value is empty")
case .text(let value): // Extract the String value
  print("Value is \(value)")
case .number(_): // Ignore the Int value
  print("Value is a number")
}
```

Structs

```
struct User {
  var name: String
  var age: Int = 40
}

// A memberwise initializer is automatically
// created to accept parameters matching the
// properties of the struct
let john = User(name: "John", age: 35)

// Memberwise initializer uses default parameter
// values for any properties that have them
let dave = User(name: “Dave")

// Accessing properties
print("\(john.name) is \(john.age) years old")
```

Classes

```
class Person {
  let name: String
  // Class initializer
  init(name: String) {
    self.name = name
  }

  // Using deinit to perform
  // object's resources cleanup
  deinit {
    print("Perform the deinitialization")
  }

  var numberOfLaughs: Int = 0
  func laugh() {
    numberOfLaughs += 1
  }

  // Define a computed property
  var isHappy: Bool {
    return numberOfLaughs > 0
  }
}

let david = Person(name: "David")
david.laugh()
let happy = david.isHappy
```

Inheritance

```
class Student: Person {
  var numberOfExams: Int = 0

  // Override isHappy computed property
  // providing additional logic
  override var isHappy: Bool {
    numberOfLaughs > 0 && numberOfExams > 2
  }
}

let ray = Student(name: "Ray")
ray.numberOfExams = 4
ray.laugh()
let happy = ray.isHappy
// Mark Child as final to prevent subclassing
final class Child: Person { }
```

Designated & Convenience Initializers

```swift
// A class must have at least one
// designated initializer and may have one or more
// convenience initializers
class ModeOfTransportation {
  let name: String
  // Define a designated initializer
  // that takes a single argument called name
  init(name: String) {
    self.name = name
  }

  // Define a convenience initializer
  // that takes no arguments
  convenience init() {
    // Delegate to the internal
    // designated initializer
    self.init(name: "Not classified")
  }
}

class Vehicle: ModeOfTransportation {
  let wheels: Int
  // Define a designated initializer
  // that takes two arguments called name and wheels
  init(name: String, wheels: Int) {
    self.wheels = wheels
    // Delegate up to the superclass
    // designated initializer
    super.init(name: name)
  }
  // Override the superclass convenience initializer
  override convenience init(name: String) {
    // Delegate to the internal
    // designated initializer
    self.init(name: name, wheels: 4)
  }
}
```

Extensions

```
// Extensions add new functionality to an existing
// class, structure, enumeration, or protocol type
extension String {

  // Extending String type to calculate
  // if a String instance is truthy or falsy
  var boolValue: Bool {
    if self == "1" {
      return true
    }
    return false
  }
}

let isTrue = "0".boolValue
```

Error Handling

```swift
// Representing an error
enum BeverageMachineError: Error {
  case invalidSelection
  case insufficientFunds
  case outOfStock
}

func selectBeverage(_ selection: Int) throws ->
String {
  // Some logic here
  return "Waiting for beverage..."
}

// If an error is thrown by the code in the do
// clause, it is matched against the catch clauses
// to determine which one of them can handle
// the error
let message: String
do {
  message = try selectBeverage(20)
} catch BeverageMachineError.invalidSelection {
  print("Invalid selection")
} catch BeverageMachineError.insufficientFunds {
  print("Insufficient funds")
} catch BeverageMachineError.outOfStock {
  print("Out of stock")
} catch {
  print("Generic error")
}

// If an error is thrown while evaluating the try?
// expression, the value of the expression is nil
let nillableMessage = try? selectBeverage(10)

// If an error is throws you'll get a runtime error,
// otherwise the value
let throwableMessage = try! selectBeverage(10)
```

Coding Protocols

```
import Foundation

// Codable conformance is the same as conforming
// separately to Decodable and Encodable protocols
struct UserInfo: Codable {
  let username: String
  let loginCount: Int
}

// Conform to CustomStringConvertible to provide
// a specific representation when converting the
// instance to a string
extension UserInfo: CustomStringConvertible {
  var description: String {
    return "\(username) has tried to login \
(loginCount) time(s)"
  }
}

// Define multiline string literal to represent JSON
let json = """
{ "username": "David", "loginCount": 2 }
"""

// Using JSONDecoder to serialize JSON
let decoder = JSONDecoder()

// Transform string to its data representation
let data = json.data(using: .utf8)!
let userInfo = try! decoder.decode(UserInfo.self,
from: data)
print(userInfo)

// Using Encodable to serialize a struct
let encoder = JSONEncoder()
let userInfoData = try! encoder.encode(userInfo)

// Transform data to its string representation
let jsonString = String(data: userInfoData,
encoding: .utf8)!
print(jsonString)
```

Access Control

```
// A module - a framework or an application - is
// a single unit of code distribution that can be
// imported by another module with import keyword

// Class accessible from other modules
public class AccessLevelsShowcase {

  // Property accessible from other modules
  public var somePublicProperty = 0

  // Property accessible from the module
  // is contained into
  var someInternalProperty = 0

  // Property accessible from its own
  // defining source file
  fileprivate func someFilePrivateMethod() {}

  // Property accessible from its
  // enclosing declaration
  private func somePrivateMethod() {}
}
```