iOS alapú szoftverfejlesztés

Dr. Blázovics László

Blazovics.Laszlo@aut.bme.hu

2021. Szeptember 07.

BME AUT

Automatizálási és
Alkalmazott
Informatikai Tanszék

Bevezetés

Oktatók

- Dr. Blázovics László
  blazovics.laszlo@aut.bme.hu
- Albert István
  ialbert@aut.bme.hu

Követelmények

- Ebben a félévben 12 előadás
- Aláírásért
  - Laborok legalább 60%-án részt kell venni (7 labor)
  - 12. héten ZH az előadás idejében, legalább 40% (elméletibb)
- Félév végi jegy (20%-ban beszámít a ZH eredmény)
  - Házi feladatra megajánlott jegy (opcionális, de nagyon ajánlott)
  - Vizsga (gyakorlatibb)
- Előnyös, ha van saját Mac vagy Hackintosh, de enélkül is teljesíthető a tárgy

Információk

- Tárgy honlap
- Teams csatorna
- Github organization: https://github.com/AUT-VIAUAV15

Házi feladat (opcionális)

- Saját projekt
- Beadási határidő: utolsó oktatási hét szerda 24:00
  - 6. hét végéig előre fel kell tölteni a specifikációt (részletek később)
- A BME benne van az iOS Developer University Programban
  - A félévre fejlesztői licencet lehet igényelni, amivel a saját eszközön való tesztelésen felül pár Apple szolgáltatás, például a *Push Notification*, vagy a *CloudKit* is tesztelhető
  - Hallgatónként maximum 1 eszköz
  - Jelentkezés a hamarosan (opcionális)
- Saját eszközön tesztelés sima Apple ID-val is lehetséges, teljesen ingyen

Ajánlott segédanyagok 📖

- The Swift Programming Language – **ingyenes**
  https://docs.swift.org/swift-book/
- Paul Hudson - Hacking with Swift – **ingyenes**
  https://www.hackingwithswift.com/read
- Paul Hudson - Pro Swift
  https://www.hackingwithswift.com/store/pro-swift
- Chris Eidhof, Ole Begemann, Airspeed Velocity - Advanced Swift
  https://www.objc.io/books/advanced-swift/
- Chris Eidhof, Florian Kugler, Wouter Swierstra - Functional Swift
  https://www.objc.io/books/functional-swift/
- Lőrentey Károly - Optimizing Collections
  https://www.objc.io/books/optimizing-collections/
- Keith Harrison - Modern Auto Layout
  https://useyourloaf.com/autolayout/

Ajánlott segédanyagok 📖 II

- Paul Hudson - 100 Days of Swift – **ingyenes**
  https://www.hackingwithswift.com/100
- Ray Wenderlich tutorialok - **egy része ingyenes**
  https://www.raywenderlich.com/
- Swift by Sundell – **ingyenes**
  https://www.swiftbysundell.com/
  - > Basics
    https://www.swiftbysundell.com/basics
- Use Your Loaf - **ingyenes cikkek**
  https://useyourloaf.com/
- objc.io
  https://www.objc.io/
- Magyar nyelven jelenleg nem tudunk aktualizált, jó könyvről ☹

iOS elhelyezése

Apple platformok

- iOS
- iPadOS
- macOS
- tvOS
- watchOS

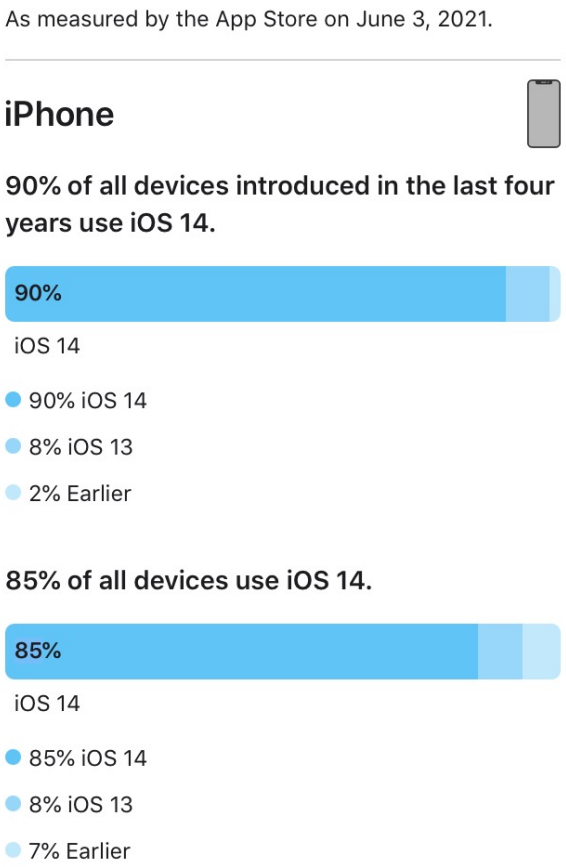
iOS alapok

- iPhone, iPad operációs rendszere
  - iPadeken iOS 13-tól ugyan *iPadOS* névre hallgat, azonban ez egyelőre csak egy marketing elnevezés, valójában továbbra is iOS
- Ugyanazon alapokra épül mint macOS Kernel (XNU) nagy része közös (UNIX/BSD alapok)
  - Az iOS alkalmazások alapból nem kompatibilisek a macOS-el
  - iOS 13-tól a Catalyst technológia segítségével lehetőség nyílt iPad alkalmazások futtatására macOS-en
- Alapvetően zárt rendszer, alkalmazásokra megkötések
- Programozás iOS SDK-val, csak macOS alatt

Az iOS rövid története

- 2007. június: megjelenik az iPhone 1.0
- 2008. március: megjelenik az iPhone SDK
  - > 3rd party alkalmazások fejlesztése elkezdődhet
- 2008. július iPhone 3G és iOS 2.0, elindul az App Store
- 2010. április: megjelenik az iPad (innentől iOS az oprendszer neve, eddig iPhone OS volt)
- 2021. szeptember: iOS 15, watchOS 8, tvOS 15

iOS verziók



- Sokkal kisebb a fragmentáció mint az Androidnál

Hardver

- ARM architektúrájú processzorok
  - ARMv8 utasításkészlettel (64-bites architektúra, 32-bites támogatással)
  - A processzor hiába támogatja a 32-bites architektúrát, iOS 11-től minden alkalmazásnak kötelezően támogatnia kell a 64-bites architektúrát
- Egyre több különböző kijelzőméret
  - Még mindig kevesebb, mint Androidon

iPad

- Kategóriateremtő készülék
- iOS 13-tól iPadOS fut rajta
- Kezdetben az iPhone-hoz hasonló funkciókkal rendelkezett,
  - > Az utóbbi fejlesztések a MacBook funkcionalitása felé mutatnak(?)
- Használható hozzá az Apple Pencil

Apple Watch

- watchOS fut rajta, mélyen iOS-re épül
- Az Apple Watch teljes értékű használatához kell egy iPhone is
- watchOS 2: natív alkalmazások futnak az órán
- watchOS 3, 4, 5: kisebb-nagyobb fejlesztések, újratervezett alkalmazások
- watchOS 6: különálló, csak az órán futó alkalmazások

Apple TV

- tvOS fut rajta, mélyen iOS-re épül
- Apple smart TV platformja
- iPad 8 hardvere, kijelző nélkül
- Siri Remote
- Apple mércével "friendly price"

Szoftverfejlesztés és eszközök

BME AUT
Automatizálási és
Alkalmazott
Informatikai Tanszék

Natív szoftverfejlesztés

- A félév során natív alkalmazásokat fejlesztünk
- Natív: Swiftben vagy Objective-C-ben írt, gépi kódra lefordított alkalmazás
  - Nem virtuális gépen/interpreter által futtatott a kód
  - Felhasználhatunk C/C++ kódban írt könyvtárakat/kódrészleteket is
  - Minden iOS funkció elérhető, gyors
  - Webes tartalmakat is megjeleníthet (WebKit)

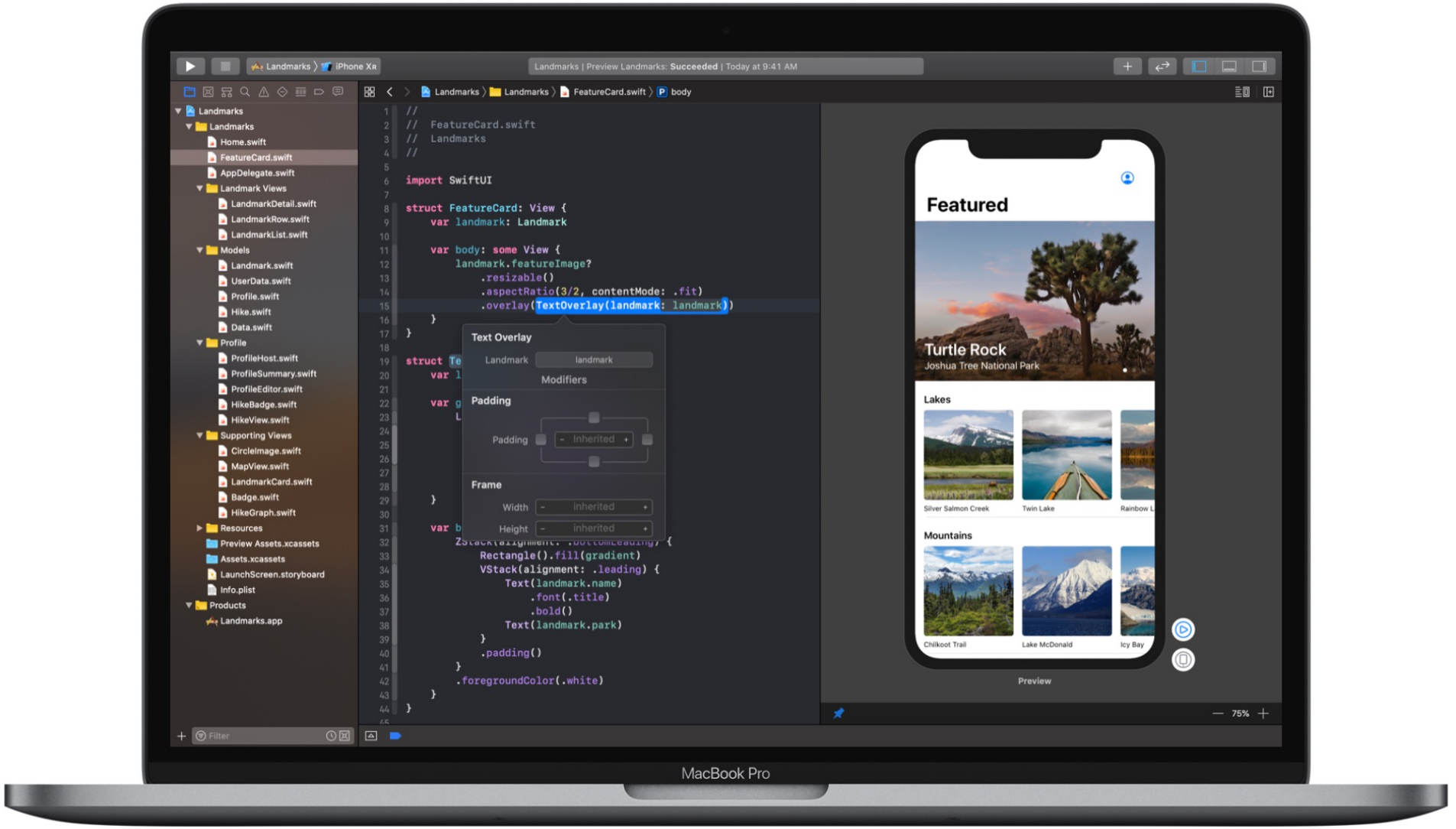
Megkötések, különbségek

- Sandbox környezet
  - Nem tudunk akárhova írni a háttértáron
- Alkalmazások közötti kommunikáció erősen korlátozott
- Általában csak egy ablakunk van, limitált képernyőmérettel
- Speciális hardver eszközök (szenzorok, GPS, stb.)
- Limitált erőforrások (memória)
- Nincs garbage collection
  - ARC (Automatic Reference Counting - Automatikus referencia számlálás) van, fordítási időben

Xcode

- iOS, iPadOS, watchOS, tvOS és macOS alkalmazások készítéséhez
- Mac App Store-ból ingyenesen letölthető
- Tartalmaz mindent, ami szükséges
  - Compiler: LLVM
    - Swift, Objective-C, C, C++
  - iOS, watchOS, tvOS, macOS SDK
  - iOS, watchOS, tvOS Simulator
  - Code Editor
  - Interface Builder
  - [Visual] Debugger
  - Instruments: alkalmazás tesztelés és elemzés (pl. memory leak szűrés)

Xcode


Landmarks | Preview Landmarks: Succeeded | Today at 9:41 AM
Landmarks
Landmarks
Home.swift
FeatureCard.swift
AppDelegate.swift
Landmark Views
LandmarkDetail.swift
LandmarkRow.swift
LandmarkList.swift
Models
Landmark.swift
UserData.swift
Profile.swift
Hike.swift
Data.swift
Profile
ProfileHost.swift
ProfileSummary.swift
ProfileEditor.swift
HikeBadge.swift
HikeView.swift
Supporting Views
CircleImage.swift
MapView.swift
LandmarkCard.swift
Badge.swift
HikeGraph.swift
Resources
Preview Assets.xcassets
Assets.xcassets
LaunchScreen.storyboard
Info.plist
Products
Landmarks.app
Landmarks › Landmarks › FeatureCard.swift › body
// FeatureCard.swift // Landmarks //
import SwiftUI
struct FeatureCard: View {
var landmark: Landmark
var body: some View {
landmark.featureImage?
.resizable()
.aspectRatio(3/2, contentMode: .fit)
.overlay(TextOverlay(landmark: landmark))
}
}
Text Overlay
Landmark landmark
Modifiers
Padding
Padding - Inherited +
Frame
Width - inherited +
Height - inherited +
Rectangle().fill(gradient)
VStack(alignment: .leading) {
Text(landmark.name)
.font(.title)
.bold()
Text(landmark.park)
}
.padding()
}
.foregroundColor(.white)
}
Featured
Turtle Rock
Joshua Tree National Park
Lakes
Silver Salmon Creek
Twin Lake
Mountains
Chilkoot Trail
Lake McDonald
Icy Bay
Preview
75%
MacBook Pro

iOS Simulator

- iPhone és iPad macOS alatt való szimulálására
  - Az alkalmazások kipróbálhatók célhardver nélkül is
  - Valójában x86-os architektúrára, macOS-re fordul a kód
- Hardver képességek szimulálása
  - Készülék orientáció, hardver gombok és „rázás" szimuláció
  - Limitált multi-touch és gesture támogatás
  - Helymeghatározás szimulálása
  - Szenzorok támogatása csak külön programokkal

iOS Simulator

iOS alkalmazások

- iOS (és macOS) alkalmazások úgynevezett bundle-ök formájában léteznek
  - Egy meghatározott felépítésű könyvtár AppName.app néven
  - Benne találhatók az alkalmazás binárisok, erőforrások, stb.
  - Minden alkalmazáshoz egy globálisan egyedi, programozó által választott azonosító tartozik: *Bundle ID*, pl. hu.bme.aut.ios.MyFirstApp
- Az alkalmazás telepítése leegyszerűsítve a bundle egy meghatározott helyre történő bemásolása

Fordítás/tesztelés készüléken

- 2015-től elég egy ingyenes Apple ID a saját készüléken való teszteléshez
  - Sok Apple szolgáltatás (pl. Push notification, In-app payment, Game Kit azonban továbbra is csak iOS Developer Program előfizetéssel)
  - Apple University Programba belépve van lehetőség Push Notification tesztelésre
- USB kábellel összekötjük telefont a Mac-el
  - Legújabb iPad Pro-t kivéve egyelőre *USB <-> Lightning*
  - Remélhetőleg hamarosan *USB-C <-> USB-C*
  - Xcode 9 és iOS 11-től akár Wireless Development
- Fordítás után automatikusan felmásolódik az app a készülékre
  - On-device debug: készüléken fut, közben a Mac-en debuggolunk Xcode-ban

Framework

- Az alkalmazások frameworkökön keresztül érik el az iOS funkcióit
- Framework: headerök, binárisok és erőforrások
- Framework használata
  - A legtöbb esetben a fordító automatikusan linkeli őket
  - Vagy explicit hozzáadjuk a projekthez
    (Project settings -> Linked Frameworks)

Példák Framework-ökre

- Foundation
  - Alapvető segédosztályok (tömbök, sztringek, idő kezelés, stb.)
  - Xcode-dal generált projektek alapból linkelik
- UIKit
  - Legnagyobb és legfontosabb framework:
  - Alkalmazás életciklus támogatása, multitasking,
  - UI elemek,
  - Gesztúrák,
  - ...
  - Xcode-dal generált projektek alapból linkelik
- Core Location
  - Helymeghatározás
- ...

Szoftverfejlesztés és iOS verziók

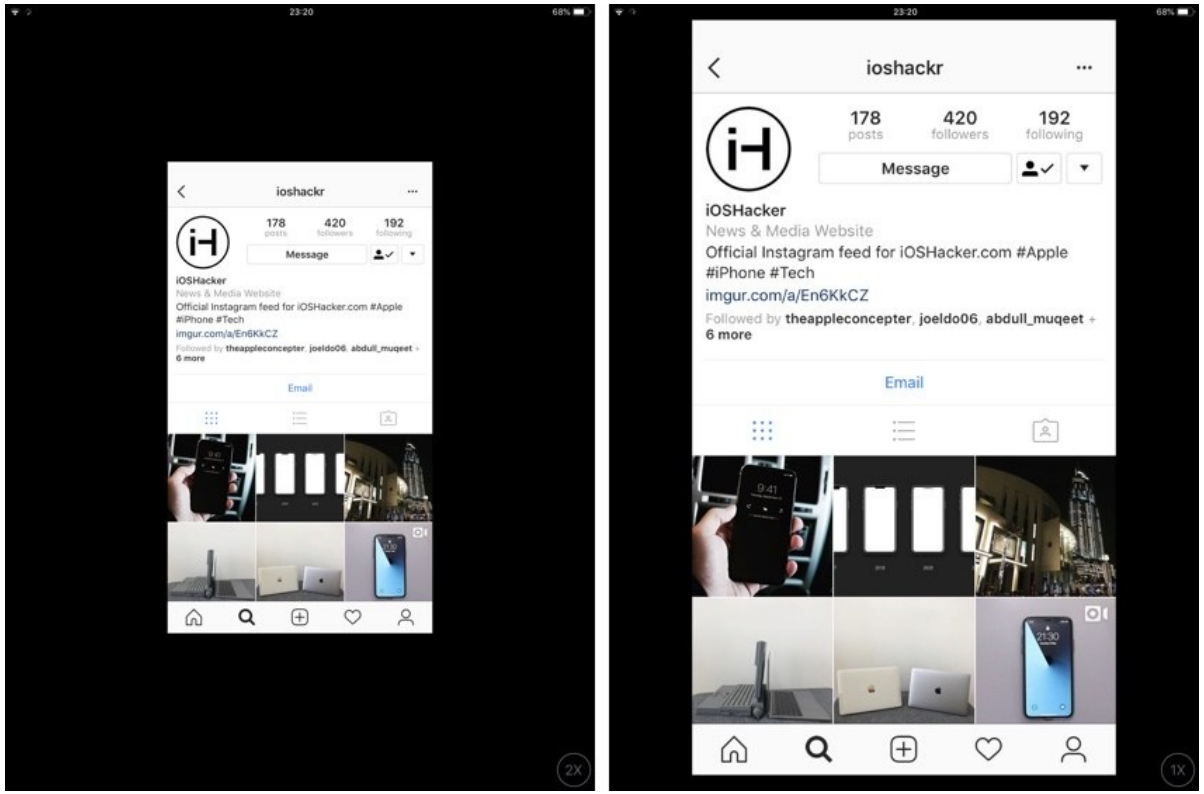
- Az iOS folyamatosan frissül, jelenleg 14.7.1 (hamarosan itt a 15.0)
- Verziók között jelentős API eltérések
- Régebbi SDK-val fordított alkalmazások futnak újabb iOS-en
- Újabb SDK-val fordított alkalmazás fut régebbi iOS-en is, ha nem használtunk fel új API-kat
  - Ha olyan API-t használunk amit még nem támogat a készüléken futó iOS verzió, akkor crash
  - Beállítható melyik a legrégebbi iOS verzió, amit támogatni szeretnénk (Deployment Target)

iPad vs. iPhone

- Alapvetően egyelőre nincs nagy különbség a fejlesztésben
- iPaden van néhány speciális UI elem
  - Split View Controller
  - Pop Over
- iPaden jobban oda kell figyelni az ablakkezelésre
  - Több screen, változó méret
- Alkalmazás támogatás, projekt típusok:
  - iPhone – Csak telefonon, iPad-en képernyő közepén
  - iPad – Egyedi UI iPad-re szabva
  - *Mac* – Csak ha iPad is target*

iPhone only alkalmazások iPaden

- iOS 12-ig ha iPhone only alkalmazást futtattunk iPaden, akkor az egy iPhone 4-es felbontású ablakban jelent meg
- iOS 12-től kezdve egy iPhone 6-os felbontású ablakban



macOS vs. iOS fejlesztés

- Ugyan sok a közös API, de nagyok a különbségek
  - Cocoa vs. Cocoa Touch
  - Külön dokumentáció, külön példakód
  - iOS alatt sok osztály hiányzik vagy le van butítva
  - DE! Pl.: AppKit vs. UIKit: UIKit sokkal fejlettebb
- Mac App Store-ban való terjesztéshez is Apple Developer Program regisztráció szükséges
  - App Store-on kívül viszont ingyenesen terjeszthetők az appok, bár hamarosan kötelező lesz Apple által hitelesíteni

Swift alapok

BME AUT
Automatizálási és
Alkalmazott
Informatikai Tanszék

Objective-C

- Xcode 6 előtt ez volt a natív iOS fejlesztés egyetlen támogatott programozási nyelve
- Sokan idegenkedtek tőle
  - Smalltalk-ból származó szintaxis, nem hasonlít a ma népszerű programozási nyelvekre
- Hiányoznak belőle bizonyos modern programozási konstrukciók (pl. generics) és elég "bőbeszédű”
- Továbbra is 100%-ban támogatott iOS alkalmazások fejlesztésénél
- Bizonyos funkciók csak a „közvetítésével” érhetők el

Objective-C

```
#import <UIKit/UIKit.h>

@interface MessagesViewController : UITableViewController

@end

#import "MessagesViewController.h"

@interface MessagesViewController ()

@property (nonatomic, copy) NSArray<Message *> *messages;

@end

@implementation MessagesViewController

- (UITableViewCell *)tableView:(UITableView *)tableView cellForRowAtIndexPath:(NSIndexPath *)indexPath {
    UITableViewCell *cell = [tableView dequeueReusableCellWithIdentifier:@"Cell" forIndexPath:indexPath];
    cell.textLabel.text = self.messages[indexPath.row];
    return cell;
}

@end
```

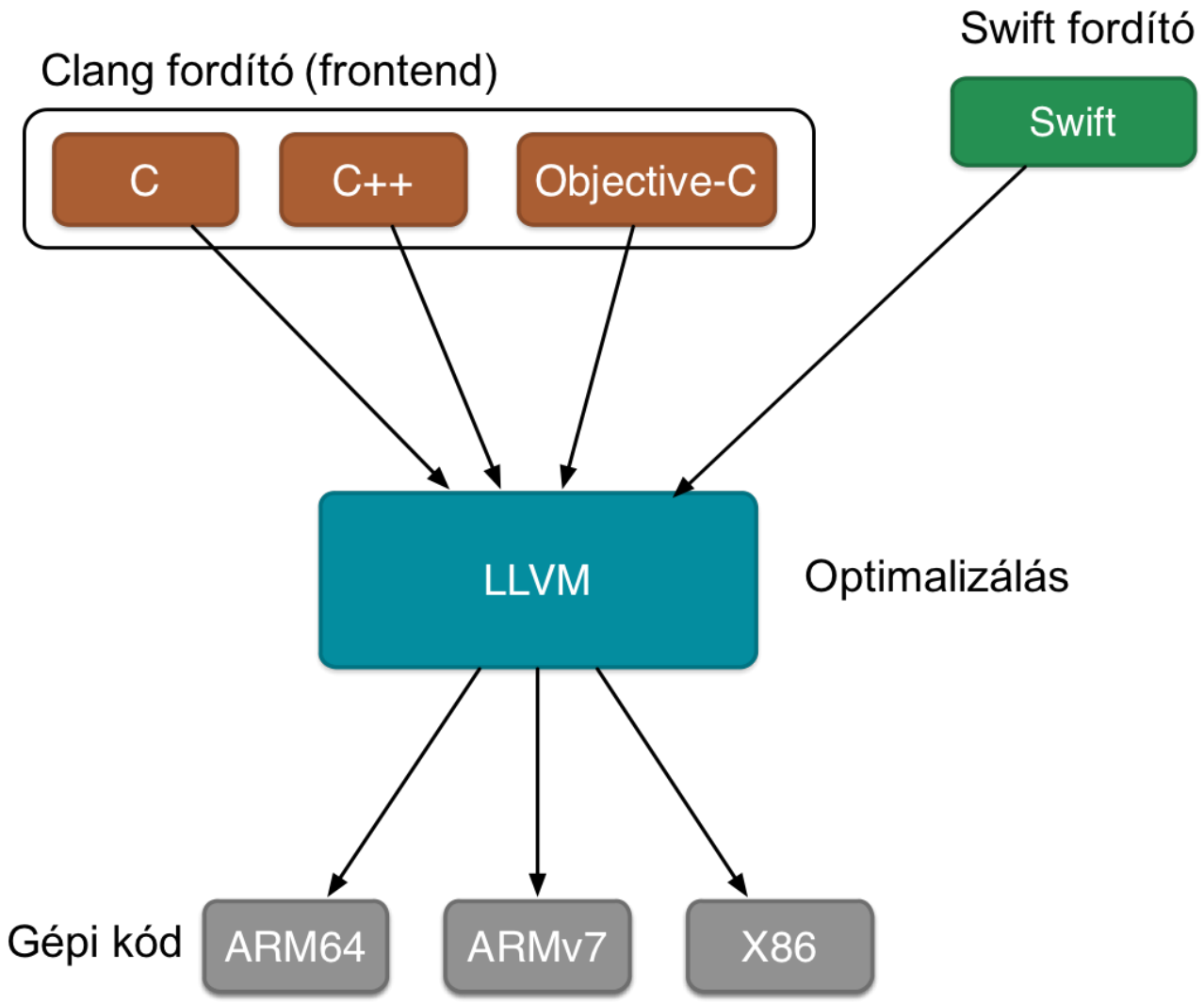
Swift

```
import UIKit

class MessagesViewController: UITableViewController {
    var messages = [Message]()

    override func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) ->
UITableViewCell {
        let cell = tableView.dequeueReusableCell(withIdentifier: "Cell", for: indexPath)
        cell.textLabel?.text = messages[indexPath.row]
        return cell
    }
}
```

Swift dióhéjban

- Apple platformok új fejlesztési nyelve
- Script-szerű nyelv, modern lehetőségekkel, kompakt szintaxissal
- Natív kódra fordul
  - Mint C, C++ vagy Objective-C
- Sok más nyelvből merít: Python, Haskell, Ruby, C#
  - Objective-C without the C
- Könnyen összekapcsolható Objective-C-ben írt kóddal (*interoperability*)
- Elsőre egyszerűnek tűnik, de sok funkciója kifejezetten komplex és ezek helyes elsajátítása nem kevés időt igényel

Fordító/toolchain



Szintaxis

- Pontosvesszők opcionálisak
- Zárójelek a legtöbb esetben elhagyhatók
- Kód blokkoknál mindig kötelezők a kapcsos zárójelek (egy utasítás esetén is!)

```
if i > 10 {
    print("That's more than nothing")
}
```

- Bármilyen Unicode karakter használható az azonosítókban
  - Kódolásnál maradjunk az angol ABC betűinél… (Ctrl + ⌘ + Space! 😎)
- Általános tanács: használjuk ki a Swift nyújtotta lehetőségeket és törekedjünk a legtömörebb kódra (kivéve, ha a megértés vagy helyes működés megkívánja a bővebb kódot)

Konstansok és változók

- Konstans: egyszer adhatunk értéket
  - **`let`** kulcsszóval deklarálva

```
let pi = 3.14159265
pi = 3 // ERROR: pi értéke nem változhat
```

- Változó: megváltozhat az értéke
  - **`var`** kulcsszóval deklarálva

```
var i = 12
i += 1 // OK i egy változó
```

Típusrendszer

- A Swift **erősen típusos**: sehol sincs automatikus (implicit) típus konverzió (pl. **Int** és **Double** között vagy **Bool**ra valamilyen számértékről)
- Aritmetikai operátorok csak azonos típusú attribútumokkal működnek

```
let doubleNum = 7.13
let intNum = 3
let result = doubleNum + intNum // ERROR
let result = doubleNum + Double(intNum)
```

- Minden konverziót explicit jelölni kell

Típusrendszer II.

- A Swift **statikusan típusos**: minden változónak/konstansnak deklarálástól kezdve konkrét típusa van, ami később nem változhat
- **Type inference**: a Swift megpróbálja kikövetkeztetni a változó típusát a kezdeti értékből

```
var str = "Hello swift" //String
var d = 3.14 //Double
let green = UIColor.green //UIColor
```

- Explicit is megadhatók a típusok:

```
var color: UIColor = .green
```

Alaptípusok

- Egész szám: **Int**
  - Ahol csak lehet, használjunk sima **Int**et! Csak ott használjunk specifikusabb típusokat, ahol kifejezetten szükséges
  - Előjel nélküli integer: **UInt**, explicit méretű integer: **Int8**, **Int64**, **UInt8**, ...
- Lebegőpontos számok: **Double** vagy **Float**
  - Double 64 bites, Float 32 bites
  - Az alapértelmezett lebegőpontos típus a Double

```
var d = 3.14
```

- Logikai érték: Bool
  - Két lehetséges érték: **true** vagy **false**

String

- Ugyanúgy karakterek sorozata, de használata más programozási nyelvektől némileg eltérő!

```
var str: String = "Hello"
```

- String összefűzés

```
str = "Hello " + "BME"
str += ". How are you?"
str.append("\n I'm fine!")
```

- String interpoláció

```
let studentsStr = "The number of students is \(studentCount)"
```

- Karakterszám:

```
str.count
```

- A String érték típus (**value type**): mindig egy másolat készül és adódik át

String II.

```
let palinka = "🇭🇺 p\u{00E1}linka"
print(palinka) // "🇭🇺 pálinka"
```

- Karakterszám

```
print(palinka.count) // "9"
```

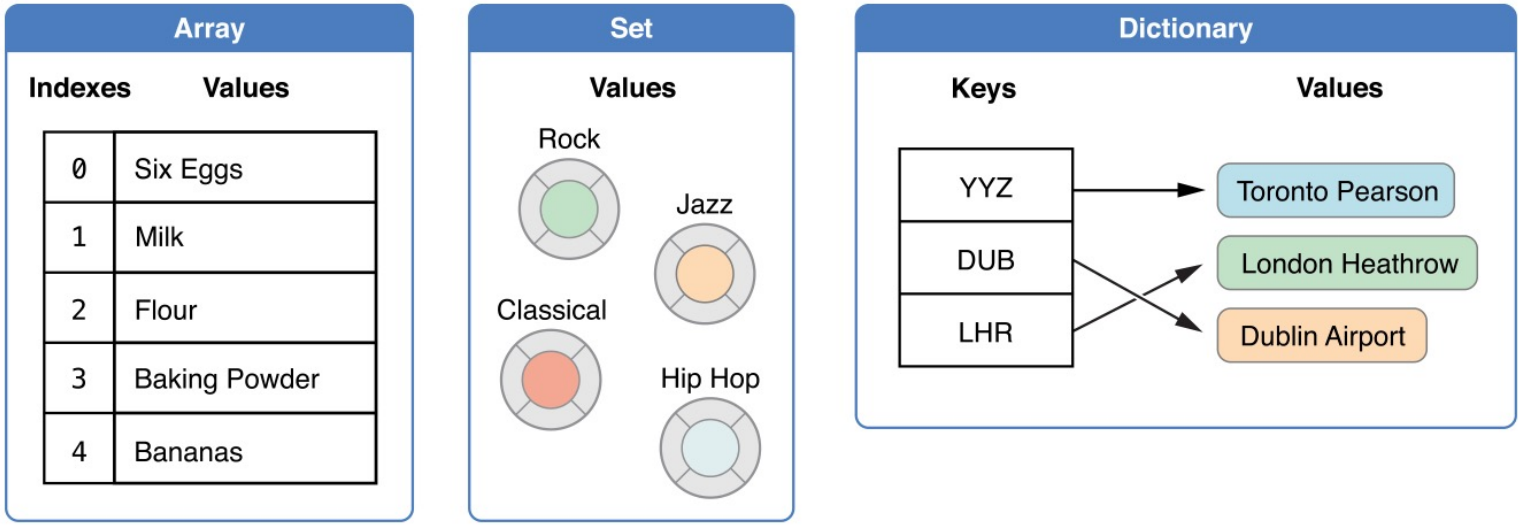
- Valóban csak 1 karakter pl. a 🇭🇺?

```
let hun = "🇭🇺"
print(hun.count) // "1"
print(hun.unicodeScalars.count) // "2" --> \u{1F1ED} és \u{1F1FA}
print(hun.utf16.count) // "4" --> 0xD83C 0xDDED és 0xD83C 0xDDFA
print(hun.utf8.count) // "8" --> 0xF0 0x9F 0x87 0xAD és 0xF0 0x9F 0x87 0xBA
```

- Ha üres Stringre vizsgálunk használjuk mindig az **isEmpty** metódust a count > 0 helyett!
  - > A fentiekből következően a count ugyanis költséges lehet

Collection Types

- Generikus tárolók, de mindig egyértelmű, hogy milyen típusú elemeket tárol éppen
  - A fordító nem enged rossz típust beilleszteni
- A létrehozás módja dönti el, a kollekció módosíthatóságát
  - **`var`-**ként létrehozva *mutable*
  - **`let`-**ként pedig *immutable* --> mindig induljunk lettel!



Array

- Tömb: ugyanolyan típusú elemeket tárol rendezett listában
  - Egy érték többször is szerepelhet különböző pozíciókban
  - (AnyObject vagy Any használatával akár különböző típusú elemeket is tárolhatunk)
- Tömb létrehozási módok

```
let animals: Array<String> = ["🐕", "🐈", "🐻", "🐊"]
let animals: [String] = ["🐕", "🐈", "🐻", "🐊"] // shorthand
let animals = ["🐕", "🐈", "🐻", "🐊"] // shorthand + type inference
```

- Tömb elemek elérése:

```
let dog = animals[0] // "🐕"
let pets = animals[0...1] // ["🐕", "🐈"]
```

Set

- Halmaz: ugyanolyan típusú *egyedi* elemeket tartalmaz *rendezetlenül*
  - A halmazban tárolt elemeknek meg kell felelnie a `Hashable` protokollnak
  - Alaptípusok alapból megfelelnek
- Set létrehozási módok

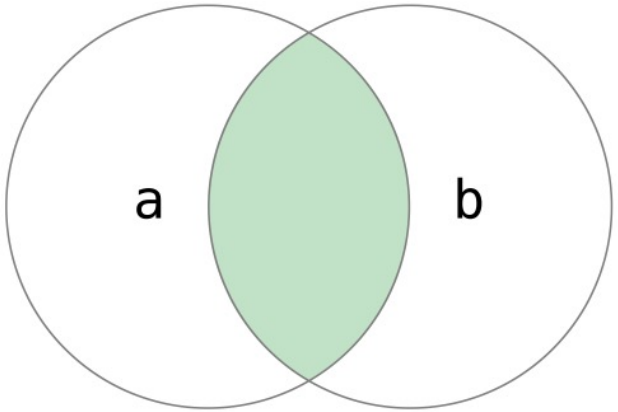
```
var favoriteGenres: Set<String> = ["Rock", "Metal", "Classical"]
var favoriteGenres: Set = ["Rock", "Metal", "Classical"] // type inference
```

- Set manipuláció

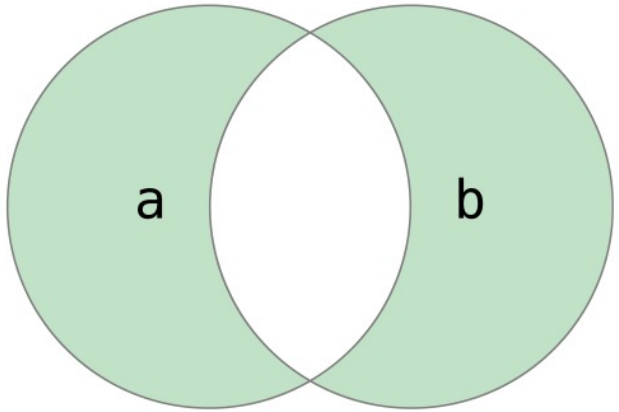
```
favoriteGenres.insert("Jazz")
favoriteGenres.insert("Rock") // nem történik semmi
favoriteGenres.remove("Rock")
favoriteGenres.contains("Funk") // false
```

Halmazműveletek

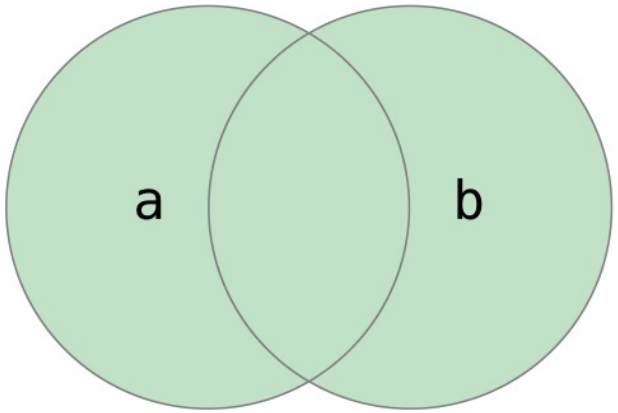
a.intersection(b)



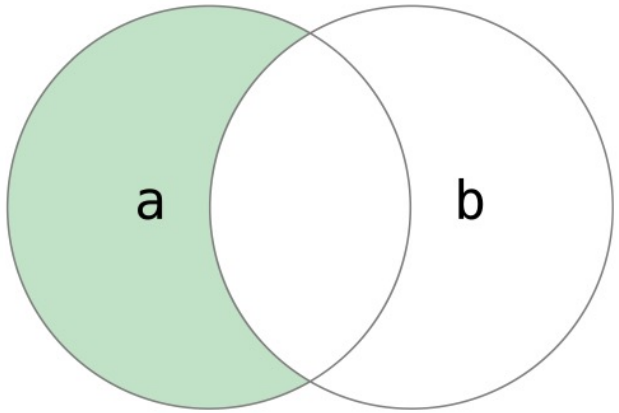
a.symmetricDifference(b)



a.union(b)

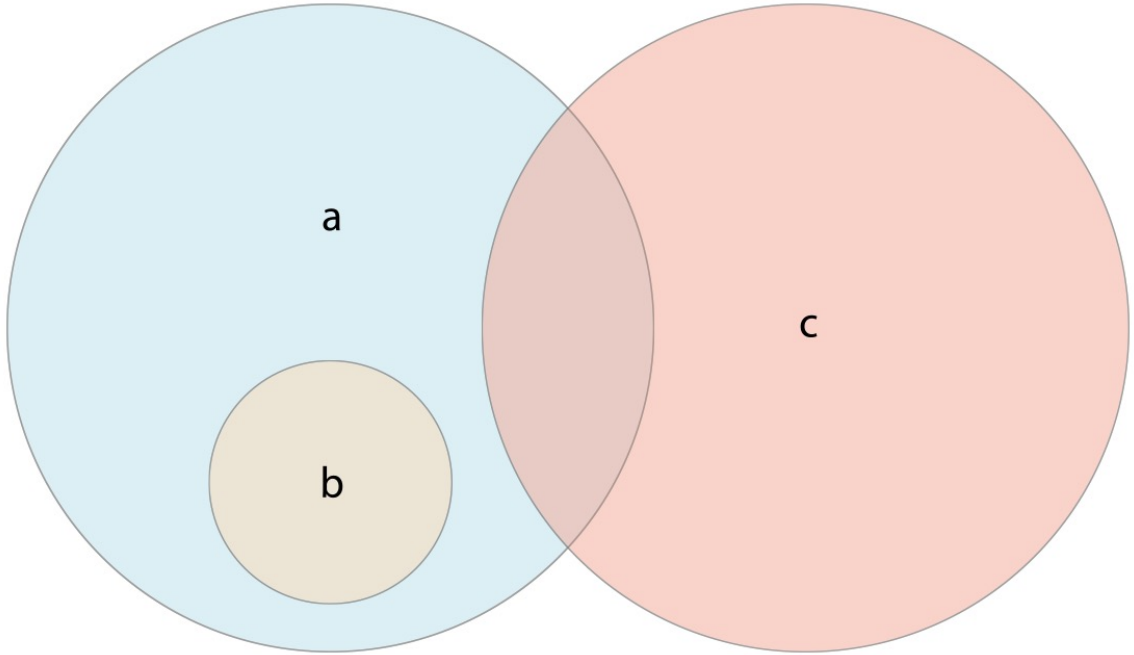


a.subtracting(b)



Halmazműveletek II.

- `isSubset(of:), isStrictSubset(of:)`
- `isSuperset(of:), isStrictSuperset(of:)`



Dictionary

- Kulcs-érték párokat tárol
- Mind a kulcs, mind az értékek típusa rögzített
- A kulcsnak meg kell valósítani a `Hashable` protokollt
- Létrehozási módok

```
var applicants: Dictionary<String, Int> = ["Márton" : 23, "Laura" : 33]
var applicants: [String: Int] = ["Márton" : 23, "Laura" : 33]
var applicants = ["Márton" : 23, "Laura" : 33]
```

- Műveletek

```
applicants["Zoltán"] = 40 // hozzáadás
let age = applicants["Zoltán"] // hozzáférés
print(age) // 40
applicants["Zoltán"] = nil // törlés
```

Enum

```
enum PartnerType {
    case customer
    case employee
    case owner
}
var type: PartnerType = .customer
```

- Az Enum értékek mögött tetszőleges "nyers/mögöttes" típus (**raw value**) állhat (nem csak Int)
- Enum értékekhez hozzárendelhetők társértékek (**associated value**)

```
case employee(jobgrade: Int)
```

- Enumok kiterjeszthetők metódusokkal és property-kkel, sőt még protokollokat (interfészeket) is megvalósíthatnak!

Vezérlési szerkezetek

- A kapcsos zárójelek `{ }` használata mindenhol kötelező!
- `for in`
  - C stílusú for ciklusok használata megszűnt
- `while` és `repeat-while`
- `if` (`else`)
- `guard`
- `switch`

for in

- Array (vagy a Set)

```
let names = ["Anna", "Alex", "Brian",
"Jack"]
for name in names {
  print("Hello, \(name)!")
}
```

- Dictionary (key-value)

```
let numberOfLegs = ["spider": 8, "ant": 6, "cat": 4]
for (animalName, legCount) in numberOfLegs {
  print("\(animalName)s have \(legCount) legs")
}
```

- Új típusú for ciklus C-s stílus helyett

```
for index in 1...5 {
  print("\(index) times 5 is \(index * 5)")
}
```

`while` és `repeat-while`

- Leggyakrabban használt, "sima" while ciklus

```
while condition {
  statements
}
```

- C-ben do-while néven fut, ritkán használt

```
repeat {
  statements
} while condition
```

`if else`

- Csak akkor hajtja végre az utasítást, ha teljesül a feltétel
- Opcionálisan megadható `else` ág, ami akkor fut le, ha NEM teljesül az eredeti feltétel
- Az `else` és az `if` kombinálható is az első feltétel után

```
let temperatureInCelsius = 30
if temperatureInCelsius <= -5 {
  print("It's very cold. Consider wearing a scarf.")
} else if temperatureInCelsius >= 30 {
  print("It's really warm. Don't forget to wear sunscreen.")
} else {
  print("It's not that cold. Wear a t-shirt.")
}
// "It's really warm. Don't forget to wear sunscreen."
```

guard

```
guard someParameter != nil else {
    fatalError("someParam cannot be nil")
}
```

- Ha teljesül a feltétel: nem történik semmi
- Ha nem teljesül: lefut a kódblokk, melynek **kötelező kilépnie a függvényből**
- A példában *crash*

switch

```
let someCharacter: Character = "z"
switch someCharacter {
case "a":
    print("The first letter of the alphabet")
case "z":
    print("The last letter of the alphabet")
default:
    print("Some other character")
}
// "The last letter of the alphabet"
```

- automatikus **break** a következő **case** ág előtt
- **fallthrough** és **break** utasítások
- minden lehetséges értéket le kell fedni **case** ágakkal (vagy felvenni **default** ágat)
- lehetőség van tartományok, több érték lefedésére, valamint további feltételek vizsgálatára (**where**)
- Stringekre is működik

Függvények

- Függvényeket a **func** kulcsszóval deklarálunk

```
func countNumbers(in string: String) -> Int {
    //Implementation goes here
}
```

- A paraméter neveket (címkéket) ki kell írni híváskor

```
countNumbers(in: "iOS 13")
```

- A címke elhagyható ‘_’, ekkor nem kell hívásnál kiírni
  - Alapértelmezetten a paraméternév

```
func greet(_ students: [String], greeting: String)

greet(students, greeting: "Szia ")
```

Függvények II.

- Paraméterekhez megadható alapértelmezett érték

```
func greet(students: [String], withGreeting greeting: String = "Hi ")
```

- Változó hosszúságú paraméter lista
  - Pl.: print

```
func print(_ items: Any..., separator: String = " ", terminator: String = "\n")
```

  - A függvény törzsében tömbre képződik le

```
func sum(_ numbers: Int...) -> Int
sum(1, 2, 3, 4)
```

Optional

- Adattípus, melynek nem biztos, hogy van értéke (képes kifejezni az érték hiányát)
- Valójában egy enum, ami nyelvi szintre lett emelve

```
public enum Optional<Wrapped>: ExpressibleByNilLiteral {
  ...
  case none
  ...
  case some(Wrapped)
  ...
}
```

- Tartalma: `some(Wrapped)`, vagy `none` (**`nil`**-re képződik)

```
var points: Int?
```

Optional II.

- A **?** operátorral deklaráljuk
  - Bármilyen Swift típus Optionallé tehető egy ?-t a típus neve után írva
  - `Int?`, `String?`, `UIColor?`, …
- Egyik legalapvetőbb Swift mechanizmus
  - Ha egy property-nek nincs mindig értéke
  - Ha egy metódus nem mindig tér vissza konkrét értékkel

Optional típusok használata

```
var points: Int?
```

- Mielőtt felhasználhatnánk egy Optional típusú értéket, először ellenőrizni kell, hogy van-e értéke:

```
if points != nil { }
```

  - Ha egy üres (`nil`) értékű Optional-t próbálunk használni, **az alkalmazás crash-el ha erőltetjük a kicsomagolását**

- Az Optional típusú értékeket ki kell csomagolni (**unwrap**)
  - Force unwrapping operátor: **!**

```
print("The value of points: \(points!)")
```

  - Ellenőrzés és kicsomagolás Optional bindinggal:

```
if let pointsValue = points { }
guard let pointsValue = points else { return }
```

  - Optional chaining

Optional Chaining

- Biztonságos és kényelmes módja egy esetleg `nil` értékű Optional egy metódusának meghívására vagy érték átadására
- A `?` operátorral "kicsomagolva" az Optional-t, majd erre meghívva egy metódust vagy lekérve egy property-t: ha az Optional `nil`, a művelet nem hajtódik végre (nincs crash sem)

```
var str: String? // str egy Optional
let capitalizedStr = str?.capitalized
```

Implicitly Unwrapped Optional

- Ha egy olyan property-re van szükségünk, mely definiáláskor még üres, de első használata előtt biztos értéket kap
- **Implicitly Unwrapped Optional**: egy olyan opcionális érték, mely használatkor automatikusan "kicsomagolódik" (nem kell '**!**')
  - A fordító nem panaszkodik, hogy nincs kezdeti értéke
  - Miután később értéket adtunk neki, úgy használható mint egy standard (nem opcionális) érték
- A '**!**' jelet kell hozzáadni egy típushoz

```
var someText: String!
```

- FONTOS: `nil` esetén továbbra is elszáll az app!

`class` és `struct`

```
class SomeClass { }
struct SomeStruct { }
```

- Újrafelhasználható, általános célú típusok property-kkel és metódusokkal
- *class* és *struct* közötti különbségek
  - *class*: **reference type** (referencia adódik át)
  - *struct*: **value type** (értékadáskor mindig másolat adódik át)
  - Öröklés csak osztályoknál (de protokollokat struct is megvalósíthat)
- Csak az osztályoknak lehet "deinicializálója" (~destruktor)
- Castolás csak osztályoknál

struct-ok (érték típusok)

- `Int`, `Double`, … (numerikus típusok)
- `String`
- `Array`
- `Dictionary`
- `Enum`
- `Tuples`
- …

Konstans class vs. konstans struct

- Konstans class (referencia típus)
  - Csak az objektumra hivatkozó változó (referencia) ami konstans (~ "konstans pointer" más nyelvekben) – nem lehet új értéket adni neki
  - A hivatkozott objektum állapota (pl. property-k) ettől még módosítható!

```
let constLabel = UILabel() // konstans referencia
constLabel = UILabel(frame: someFrame) // COMPILE ERROR
constLabel.text = "Owls are funny" // a property módosítható
```

- Konstans struct (érték típus)
  - Mind a változó, mind a hivatkozott objektum állapota konstans
  - Egy konstans Struct propertyjei NEM módosíthatók

```
let constPoint = CGPoint(x: 0, y: 0)
constPoint.x = 10 //COMPILE ERROR
```

Osztályok deklarálása

```
class Book {

    //Property-k var vagy let kulcsszóval deklarálhatók
    var title: String
    var pageCount: Int?

    //Inicializálók (~konstruktor) felelősek az osztály példányainak kiindulási
    //állapotának beállításáért. Mindig init kulcsszó.
    init(title: String, pageCount: Int? = nil) {
        self.title = title
        self.pageCount = pageCount
    }

    //Metódusok func kulcsszóval deklarálva
    func info() {
        print("\(title) book is \(pageCount) long")
    }
}
```

Osztályok példányosítása

- Osztálynév + ()

```
var myColor = UIColor()
```

- A paramétereknek meg kell felelniük az osztály egyik inicializálójának

```
var rating = Rating(user: "Fanboy", stars: 5)
```

```
init(user: String, stars: Int)
```

Tárolt és kiszámított property-k

- **Stored property**: az értéke az osztály memóriaterületén tárolódik (mint változóknál)

```
var title: String
```

- **Computed property**: egy metódus pár (kötelező **get**, opcionális **set**), mely meghívódik, ha a property értékét lekérdezzük vagy megváltoztatjuk

```
var titleWithAuthor: String {
  get {
   return "\(author): \(title)"
  }
}
```

Metódusok

- Az osztály törzsén belül **func** kulcsszóval deklarált függvények
  - **self** (~this pointer) kulcsszóval hivatkozhatunk az objektumra, melyre a metódust meghívták (elhagyható, hagyjuk is el, ha lehet!)

```
func addRating(with stars: Int, _ user: String) {
    self.ratings.append(Rating(user: user, stars: stars))
}
```

- Metódushívásnál minden paraméter neve (címkéje) kiírásra kerül (explicit jelölhető '_'-val, ha el szeretnénk hagyni)

```
myBook.addRating(with: 5, "Tim")
```

Függvények fajtái

- (Globális) függvény
- Metódus
  - Olyan függvény, mely mindig egy adott típushoz (`class`, `struct`, `enum`) tartozik
- Closure
  - Egy anonim függvény / lambda kifejezés
  - Automatikusan "begyűjti" (capture) a behivatkozott változókat
- Minden függvény/metódus/Closure **referencia típus** és **elsőrendű eleme a nyelvnek**: paraméterként átadható, változóhoz rendelhető, visszatérési érték lehet, stb.

Inicializálók – init

- Példányosításkor az osztály minden property-jét kötelező inicializálni
- Vagy a property deklarálásával együtt a kezdeti értékének megadásával

```
var stars: Int = 0
```

- Vagy inicializálás során, az init metódus(ok)ban:

```
init(stars: Int) {
  self.stars = stars
}
```

- Egy osztálynak tetszőleges számú inicializálója lehet (a paramétereiknek legalább névben el kell térniük)
- Példányosításkor alapesetben az inicializáló minden paraméterének nevét ki kell írni

```
var rating = Rating(stars: 5)
```

Protocol

- **Protokoll**: metódusok és property deklarációk listája (más nyelvekben interfész)
  - Metódusok és property-k implementáció nélkül
- A protokollokat adoptálhatják (megvalósíthatják) az osztályok, structok és enumok
  - Tetszőleges számú protocol-t megvalósíthatnak

Protocol példa

```
protocol Attackable {
    var life: Int { get set }
    func take(damage: Int)
}

//Goblin adoptálja az Attackable protokollt:
class Goblin: Attackable {
  var life = 10
  func take(damage: Int) {
      life -= damage
  }
}
```