Мова програмування Swift

Виконали студенти групи ПМАМ-12 :
Зровко Володимир
Плекан Сергій
Терещук Олег

Swift з'явився в 2014 році. Творцем мови програмування є компанія Apple. Ця мова програмування створена насамперед для розробки додатків на iOS і macOS. Вона відноситься до МП загального призначення, тобто на ній можна розробляти не тільки мобільні додатки, але і програми для десктопних платформ.

Компілятор Swift побудований з використанням технологій вільного проєкту LLVM. Swift успадковує найкращі елементи мов C і Objective-C, тому синтаксис звичний для знайомих з ними розробників, але водночас відрізняється використанням засобів автоматичного розподілу пам'яті і контролю переповнення змінних і масивів, що значно збільшує надійність і безпеку коду. При цьому Swift-програми компілюються у машинний код, що дозволяє забезпечити високу швидкодію. За заявою Apple, код Swift виконується в 1.3 рази швидше коду на Objective-C. Замість збирача сміття Objective-C в Swift використовуються засоби підрахунку посилань на об'єкти, а також надані у LLVM оптимізації, такі як автовекторизація.

Основним застосуванням Swift є розробка користувацьких застосунків для macOS, iOS, tvOS, watchOS з використанням тулкіта Cocoa і Cocoa Touch. Swift щільно інтегровано до власницького середовища розробки Xcode, проте може бути викликано з терміналу, що уможливлює її використання на операційний системах, відмінних від macOS, наприклад, на Linux.

ТИПИ ДАНИХ

Type and use
Int Represents an integer. Internally Int refers to Int32 in 32 bit platform and Int64 in 64 bit platform.
Int64 Represents 64 bit integer.
Int32 Represents 32 bit integer.
Int16 Represents to 16 bit integer.
Int8 Represents to 8 bit integer.
UInt64 Represents 64 bit unsigned integer.
UInt32 Represents 32 bit unsigned integer.
UInt16 Represents 16 bit unsigned integer.
UInt8 Represents 8 bit unsigned integer.
Float Represents 32 bit floating point number.
Bool Represents a boolean and has two states *true* and *false*.
Character Represents a single character. It accepts unicode characters.
String Represents strings which is actually a collection of type Character.

Типи змінних і констант можна визначати явно і неявно

```
1  var age: Int = 32
2  var name: String = "Tom"
```

```
1  var name = "Tom"
```

Swift являється типобезпечною мовою зі строгою типізацією, тому після того як змінній буде встановлено тип, ми його змінити не можемо. В даній ситуації виникне помилка

```
1  var age: Int
2  age = "Tom"
```

ОПЕРАТОРИ

Мова Swift підтримує більшість стандартних операторів C, а також ряд можливостей для усунення типових помилок в коді. Оператор присвоювання (=) не повертає значення, що дозволяє уникнути плутанини з оператором перевірки на рівність (==). Арифметичні оператори (+, -, *, /,% і т. Д.) можуть виявляти і запобігати переповнення типу, щоб числовій змінної не можна було присвоїти занадто велике або занадто мале значення.

Оператор присвоєння:

```
let b = 10
var a = 5
a = b
```

```
let (x, y) = (1, 2)
```

Арифметичні оператори (+, -, *, /):

```
1 + 2 // equal 3
5 - 3 // equal 2
2 * 3 // equal 6
10.0 / 2.5 // equal 4.0
```

```
"hello, " + "world"  // equal "hello, world"
```

Оператор цілочисельного ділення:

```
9 % 4    // equal 1
```

Унарні оператори (+, -):

```
let three = 3
let minusThree = -three // minusThree equal -3
```

Складові оператори присвоювання (+=, -=, *=, /=):

```
var a = 1
a += 2
// a equal 3
```

Оператори порівняння:

- equal (a == b)
- not equal (a != b)
- more (a > b)
- less (a < b)
- more or equal (a >= b)
- less or equal (a <= b)

Логічні оператори (!, ||, &&)
Тернарний умовний оператор (var variable = condition ? true : false)

Оператор обєднання по nil:

```
var colorNameToUse = undefinedColor ?? defaultColor
```

Оператор замкнутого діапазону:

```
for index in 1...5 {
 print("\(index) multiply on 5 will be \(index * 5)")
}
// 1 multiply on 5 will be  5
// 2 multiply on 5 will be 10
// 3 multiply on 5 will be 15
// 4 multiply on 5 will be 20
// 5 multiply on 5 will be 25
```

Оператор півзамкненого діапазону:

```
let names = ["Anna", "Alex", "Brian", "Jack"]
let count = names.count
for i in 0..<count {
 print("Person \(i + 1) is \(names[i])")
}
// Person 1 is Anna
// Person 2 is Alex
// Person 3 is Brian
// Person 4 is Jack
```

```
for name in names[2...] {
    print(name)
}
// Brian
// Jack

for name in names[...2] {
    print(name)
}
// Anna
// Alex
// Brian
```

КОРТЕЖІ

Кортежі являють собою набір значень, який розглядають як один обєкт. Тип даних в кортежі можна визначити явно і неявно.

```
let props = (22, "age")
var userInfo = (true, 34, "Tom")
```

```
let props: (Int, String) = (22, "age")
var userInfo: (Bool, Int, String) = (true, 34, "Tom")
```

Можна присвоювати значення кортежа іншій змінній або константі:

```
var userInfo: (Bool, Int, String) = (true, 34, "Tom")
let(isMarried, age, name) = userInfo
print(name)     // "Tom"
```

```
var userInfo: (Bool, Int, String) = (true, 34, "Tom")
let(_, age, name) = userInfo
```

```
var userInfo: (Bool, Int, String) = (true, 34, "Tom")
let age = userInfo.1
let isMarried = userInfo.0
var name = userInfo.2
```

```
var userInfo = (married: true, age: 34, name: "Tom")
let age = userInfo.age
var name = userInfo.name    // Tom
```

ФУНКЦІЇ

Функції — це самостійні фрагменти коду, які вирішують певне завдання. Кожній функції присвоюється унікальне ім’я, за яким її можна ідентифікувати і «викликати» в потрібний момент.

Кожна функція у Swift має тип, що описує тип параметрів функції і тип значення, що повертається функцією. Тип функції може бути використаний аналогічно будь-яким іншим типам в Swift, тобто одна функція може бути параметром іншої функції…

ОГОЛОШЕННЯ Й ВИКЛИК ФУНКЦІЇ

У кожної функції повинне бути ім'я, що відображає завдання яке вона виконує. При оголошенні функції можна задати одне або декілька іменованих типізованих значень, які будуть її вхідними даними(параметрами), а також тип значення, яке функція буде повертати в якості результату виконання

```
func greetAgain(person: String) -> String
{
    return "Hello again, " + person + "!"
}
print(greetAgain(person: "Anna"))
```

Функція може не мати повертаємого значення. В такому випадку при оголошенні функції відсутня результуюча стрілка і повертаємий тип.

```
func greet(person: String) {
```

Можна використовувати кортежний тип в якості повертаємого типу для функції, що повертає зразу декілька значень. Розглянемо приклад функції, що шукає й повертає максимальний мінімальний елементи массива.

```
func minMax(array: [Int]) -> (min: Int, m
ax: Int) {
    var currentMin = array[0]
    var currentMax = array[0]
    for value in array[1..<array.count] {
        if value < currentMin {
            currentMin = value
        } else if value > currentMax {
            currentMax = value
        }
    }
    return (currentMin, currentMax)
}
```

```
let bounds = minMax(array: [8, -6, 2, 10
9, 3, 71])
print("min is \(bounds.min) and max is \
(bounds.max)")
```

При оголошенні функції будь-якому з її параметрів можна присвоїти значення за замовчуванням, що буде використаний у вападку упущення аргументу при виклику функції.

Використання функціональних типів

У Swift з функціональними типами можна працювати так, як і з іншими типами. Наприклад, можна оголосити консанту чи змінну функціонального типу і присвоїти їй функцію відповідного типу.

```
var mathFunction: (Int, Int) -> Int = add
TwoInts
```

ООП у Swift

Swift є об'єктно-орієнтованою мовою, а отже дозволяє представити програму як набір взаємодіючих між собою об'єктів. Класс являється описом об'єкта, а об'єкт представляє екземпляр цього класу.

Для оголошення класу використовується ключове слово class, після якого йде назва классу

```
class User {


}
```

Класс може містити змінні і константи, які зберігають стан об'єкту.

```
class User {

    var age: Int = 18
    var name: String = ""
}
```

```
class User {

    var age: Int = 18
    var name: String = ""

    func move(){

        print("\(name) передвигается")
    }
}
```

Крім констант і змінних в класс може мати методи. Методи являють собою функції, асоційовані з певним типом – классом.

Після оголошення классу ми можемо використовувати його в програмі створюючи його об'єкти. Щоб створити об'єкт класу використовується ініціалізатор: `User()`

```
var tom: User = User()
tom.age = 22
tom.name = "Том"
print(tom.name) // Том
tom.move()  // "Том передвигается"
```

Наслідування

Наслідування у Swift реалізується наступним чином. Нехай у нас є клас Instrument і клас Piano. То щоб наслідувати клас Piano выд класу Instrument анм потрібно при оголошенні першого через двокрапку вказати батьківський клас:

```
class Piano:  Instrument  {
```

Таким чином всі властивості і методи батьківського класу наслідуються дочірнім й можуть ним використовуватись.

Модифікатори доступу

Private: Доступний тільки всередині класу.
Fileprivate: Доступний в будь-якому місці в межах файлу.
Internal: Доступний в будь-якому місці модуля чи додатку.
Public: Доступний поза модулем.

Додатково існує ще два модифікатора:

Open: Може бути доступний не тільки поза модулем, а також може бути унаслідуваний та перевизначений.
Final: Неможливо перевизначити чи зробити які-небуть зміни

Поліморфізм

Одною з сильних сторін ООП являється можливість використання різних об'єктів через один інтерфейс, в той час як кожен з них реалізовує свою логіку.
Створюємо клас групи:

```
class Band  {
    let instruments: [Instrument]

    init(instruments: [Instrument]) {
        self.instruments = instruments
    }

    func perform(_ music: Music) {
        for instrument in instruments {
        instrument.perform(music)
        }
    }
}
```

Тоді створюєм масив об'єктів інструментів, що є екземплярами відповідних класів і передаєм їх нашій групі:

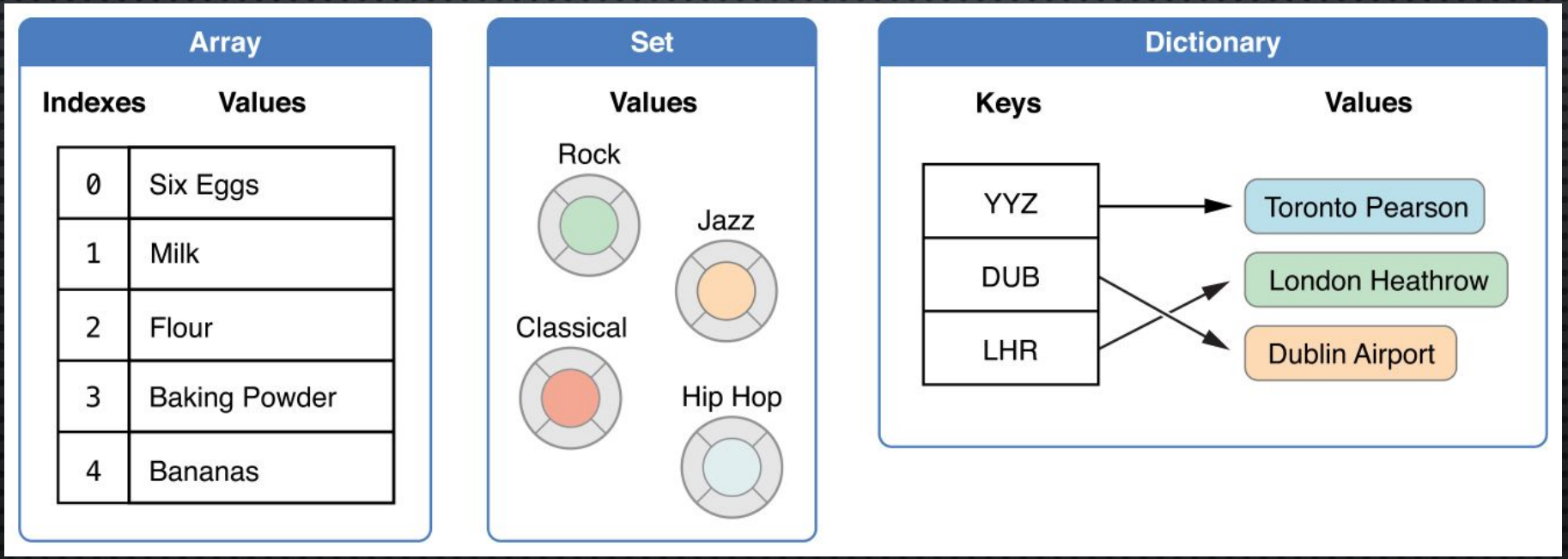
```
let instruments = [piano, acousticGuitar,
                   electricGuitar, bassGuitar]

let band = Band(instruments: instruments) band.perform(music)
```

Викликаючи метод perform екземпляра класу band, ми для кожного екземпляра інструменту викликаємо метод, який присутній в кожному з них, оскільки всі інструменти наслідуються від класу Instrument, що дозволяє ‘зіграти’ музику, яка в свою чергу є також екземпляром відповідного класу

КОЛЕКЦІЇ

Мова Swift надає три основні *типи колекцій*, відомих як масиви (Array), множини (Set) та словники (Dictionary), для зберігання колекцій значень.



МАСИВИ

Масиви - це впорядковані колекції значень.

Створерння пустого масиву:

1
```
// The full type name is also allowed
var emptyFloats: Array<Float> = Array()
```

2
```
// Shortened forms are preferred
var emptyDoubles: [Double] = []
```

Створення масиву зі значенням за замовчанням:

3
```
var digitCounts = Array(repeating: 0, count: 10)
print(digitCounts)
// Prints "[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]"
```

Доступ до елементів масиву:

```
print(oddNumbers[0], oddNumbers[3], separator: ", ")
// Prints "1, 7"
```

4
```
for street in streets {
    print("I don't live on \(street).")
}
// Prints "I don't live on Albemarle."
// Prints "I don't live on Brandywine."
// Prints "I don't live on Chesapeake."
```

МНОЖИНИ

Множини - це невпорядковані колекції унікальних значень.

Створення та ініціалізація порожньої множини:

```
var letters = Set<Character>()
```

непорожньої:

```
var favoriteGenres: Set = ["Rock", "Classical", "Hip hop"]
```

Доступ до елементів множини:

```
print("I have \(favoriteGenres.count) favorite music genres.")
```

```
favoriteGenres.insert("Jazz")
```

```
if let removedGenre = favoriteGenres.remove("Rock") {
    print("\(removedGenre)? I'm over it.")
} else {
    print("I never much cared for that.")
}
```

```
if favoriteGenres.isEmpty {
    print("As far as music goes, I'm not picky.")
} else {
    print("I have particular music preferences.")
}
```

Ітерування множини:

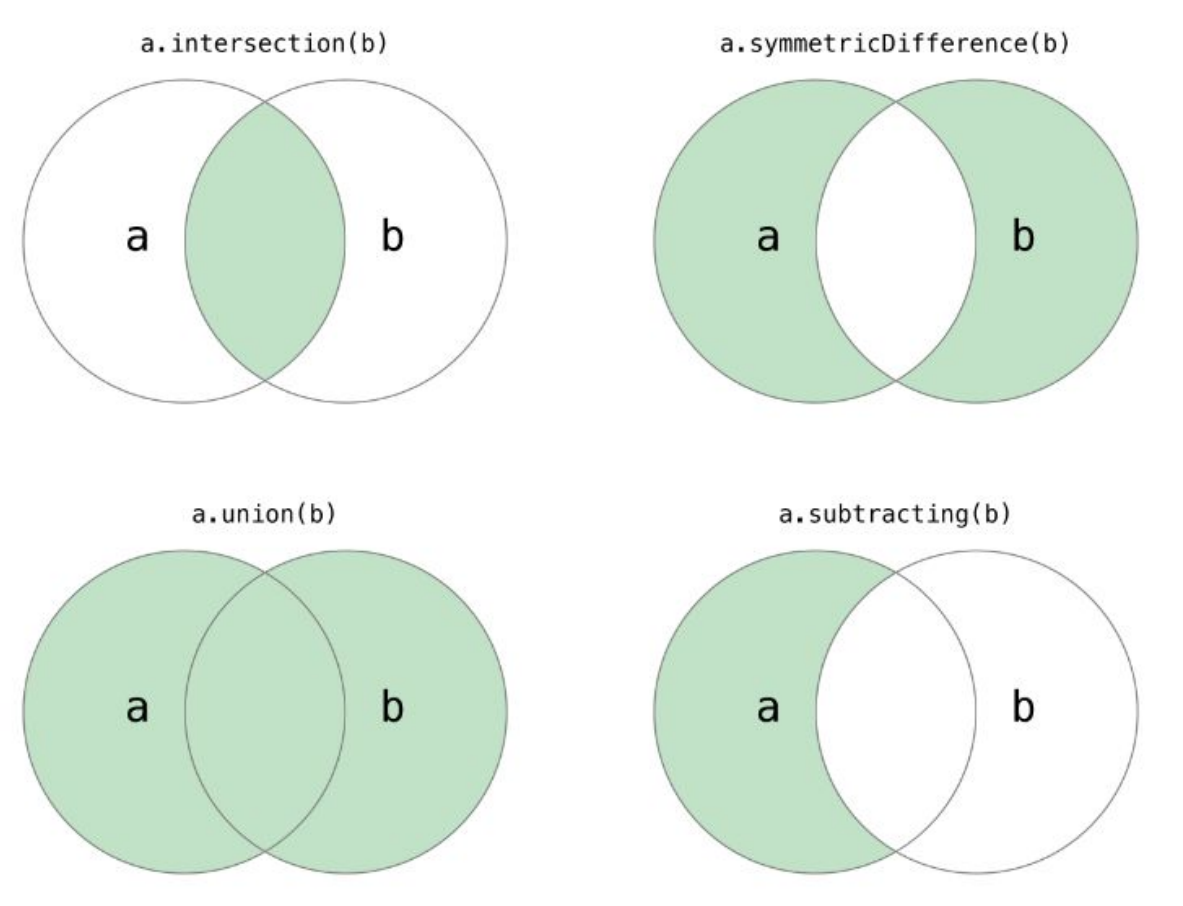
```
for genre in favoriteGenres {
    print("\(genre)")
}
```

Операції над множинами

У Swift ефективно реалізовані операції над множинами, такі як об'єднання двох множин, знаходження спільних елементів двох множин, та визначення, чи містять дві множини всі, деякі чи жодного спільного елемента.

Методи:

- a.intersection(b)
- a.symmetricDifference(b)
- a.union(b)
- a.subtracting(b)



Входження та рівність множин

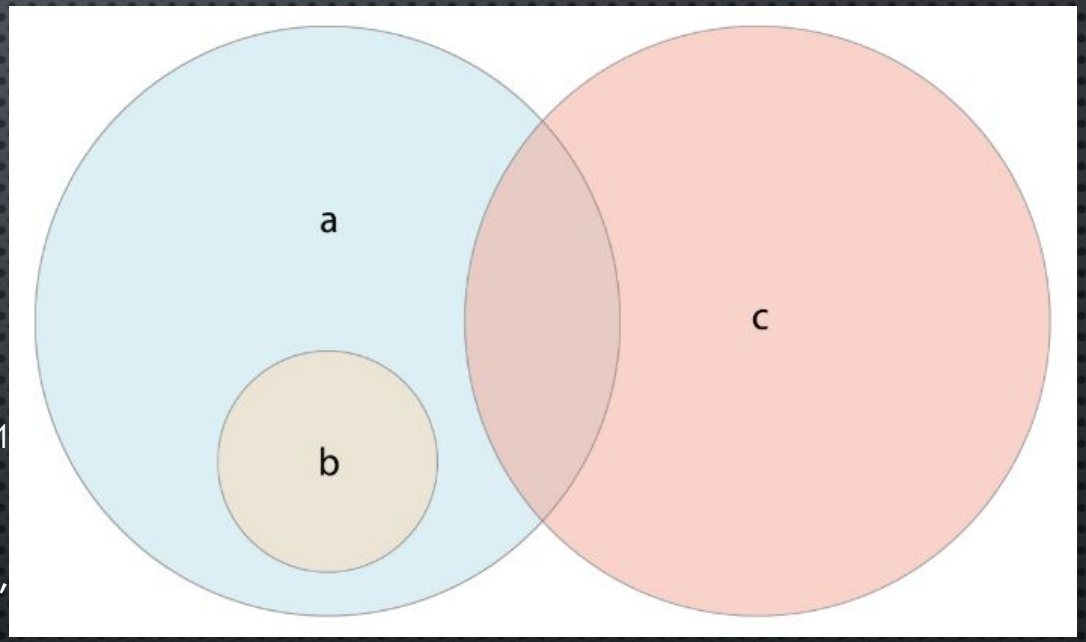
Оператор “дорівнює” (==) визначає, чи всі елементи двох множин співпадають.

Метод isSubset(of:) визначає, чи є дана множина підмножиною вказаної, тобто чи всі значення даної множини містяться у вказаній множині.

Метод isSuperset(of:) визначає, чи є дана множина надмножиною вказаної, тобто чи всі значення вказаної множини містяться у даній множині.

Методи isStrictSubset(of:) та isStrictSuperset(of:) визначають, чи є дана множина підмножиною/надмножиною вказаної, яка при цьому не дорівнює вказаній множині.

Метод isDisjoint(with:) визначає, чи перетинаються множини, тобто чи мають вони спільні елементи.



Словники

Словники зберігають асоціації між ключами одного типу та значеннями одного типу у колекції без визначеного впорядкування. Кожне значення асоціюється з унікальним ключем key, що діє як ідентифікатор для цього значення у словнику. На відміну від масивів, елементи у словнику не мають вказаного порядку.

Створення та ініціалізація порожнього словника:

```
var namesOfIntegers: [Int: String] = [:]
```

Доступ до елементів словника:

```
airports["LHR"] = "London"
```

```
let oldValue = airports.updateValue("Dublin Airport", forKey: "DUB")
```

```
let removedValue = airports.removeValue(forKey: "DUB")
```

Ітерування словника:

```
for (airportCode, airportName) in airports {
    print("\(airportCode): \(airportName)")
}
```

Середовища розробки

- Atom - безкоштовний текстовий редактор з відкритим вихідним кодом для Windows, хоча його також можна використовувати на інших платформах, таких як macOS і Linux.

- Sublime Text - текстовий редактор коду, сумісний з широким спектром мов програмування, а також з мовою розмітки, що робить його одним з кращих варіантів для професіоналів. Для Swift пропонує відомий пакет під назвою Swift-Sublime-Пакет.
- Visual Studio Code - у Microsoft також є власний редактор вихідного коду під назвою VS Code, який відповідає за сумісність з великою кількістю мов програмування. Щоб працювати в середовищі Swift, ми повинні встановити розширення для Swift.

- Xcode - інтегроване середовище розробки виробництва Apple.

- AppCode - IDE для розробки Swift, Objective-C, C і C + для iOS і macOS, побудованої на платформі IntelliJ IDEA від JetBrains.

- CodeRunner - полегшене середовище IDE програмування для iOS і macOS, призначеної для підтримки всіх найбільш часто використовуваних мов програмування і миттєвого їх виконання.