







**UNIVERSITY**

ЛЕКЦІЯ 3

# “Вступ до мов програмування орієнтованих на iOS платформу”

# Вступ до мов програмування орієнтованих на iOS

Філософія

«Hello, World!»

Синтаксис

Основні типи даних

Константи та змінні

Optional

# Філософія

Використання мов програмування має бути зручним

Висока ступінь розуміння і підтримка коду

Використання найкращих можливостей існуючих мов

Здатність інтеграції з існуючими мовами

Простота використання і написання

Підтримка декількох парадигм програмування

Перевикористання існуючого коду

# Філософія

## Використання мов програмування має бути зручним

Висока ступінь розуміння і підтримка коду

Використання найкращих можливостей існуючих мов

Здатність інтеграції з існуючими мовами

Простота використання і написання

Підтримка декількох парадигм програмування

Перевикористання існуючого коду

```
- (SCNNode *)walkInDirection:(vector_float3)direction time:
(NSTimeInterval)time scene:(SCNScene *)scene
groundTypeFromMaterial:(AAPLGroundType(^)(SCNMaterial
*))groundTypeFromMaterial;
```

OBJ-C

```
func walk(in direction:vector_float3, time:TimeInterval,
scene:SCNScene, groundType:(
_ material:SCNMaterial)->(AAPLGroundType)))
```



# Філософія

## Використання мов програмування має бути зручним

Висока ступінь розуміння і підтримка коду

Використання найкращих можливостей існуючих мов

Здатність інтеграції з існуючими мовами

Простота використання і написання

Підтримка декількох парадигм програмування

Перевикористання існуючого коду

OBJ-C

```
@property (strong, nonatomic) NSString *someVariable;
```



```
var someVariable:String?
```



# Філософія

## Використання мов програмування має бути зручним

Висока ступінь розуміння і підтримка коду

Використання найкращих можливостей існуючих мов

Здатність інтеграції з існуючими мовами

Простота використання і написання

Підтримка декількох парадигм програмування

Перевикористання існуючого коду

```
@protocol ProtocolNameDelegate <NSObject>
    @required
    - (void)someMethod();
@end
```

OBJ-C

```
protocol ProtocolNameDelegate {
    func someMethod()
}
```



# Філософія

## Використання мов програмування має бути зручним

Висока ступінь розуміння і підтримка коду

Використання найкращих можливостей існуючих мов

Здатність інтеграції з існуючими мовами

Простота використання і написання

Підтримка декількох парадигм програмування

Перевикористання існуючого коду

```
Book *newBook = [Book bookNamed: «Awesome Book»]
```

OBJ-C

```
let book = Book(named: «Awesome Book»)
```



# Філософія

## Висока ступінь розуміння і підтримка коду

Використання найкращих можливостей існуючих мов

Здатність інтеграції з існуючими мовами

Простота використання і написання

Підтримка декількох парадигм програмування

Перевикористання існуючого коду

читаючи код розумієш функціональність  
детальні довідкові матеріали  
допомога Apple інженерів в вирішенні проблем\*  
велика база прикладів реалізацій  
офіційна підтримка  
універсальність коду  
високотехнологічні інструменти для розробки  
велика кількість готових бібліотек

# Філософія

## Висока ступінь розуміння і підтримка коду

Використання найкращих можливостей існуючих мов

Здатність інтеграції з існуючими мовами

Простота використання і написання

Підтримка декількох парадигм програмування

Перевикористання існуючого коду

має кілька вбудованих компіляторів  
підтримує безліч мов і скриптів  
надає унікальні можливості для відлагодження коду  
високофункціональні симулятори  
підтримка сторонніх пакетів  
підтримка тестування  
підтримка CI

# Філософія

## Використання найкращих можливостей існуючих мов

Здатність інтеграції з існуючими мовами

Простота використання і написання

Підтримка декількох парадигм програмування

Перевикористання існуючого коду

//Languages:

C

SmallTalk

OBJ-C

//Improvements

ARC

Blocks

Categories

Dynamic types

MessageForwarding

Posing

Delegation

Swizzling

Improved runtime

etc

# Філософія

## Використання найкращих можливостей існуючих мов

Здатність інтеграції з існуючими мовами

Простота використання і написання

Підтримка декількох парадигм програмування

Перевикористання існуючого коду

//Languages:

Objective-C

Rust

Haskell

Ruby

Python

C#

CLU

//Improvements

Almost all from Obj-C

Improved syntax

Access control

Tuples

Optionals, chaining

Generics

RuntimeDevelopment

etc



# Філософія

## Здатність інтеграції з існуючими мовами

Простота використання і написання

Підтримка декількох парадигм програмування

Перевикористання існуючого коду

C  
C++  
Objective-C  
Objective-C++  
Java  
AppleScript  
Python  
Ruby  
ResEdit (Rez)  
Swift  
GNU Pascal  
Free Pascal  
Ada  
C#  
Perl  
D

# Філософія

## Простота використання і написання

Підтримка декількох парадигм програмування

Перевикористання існуючого коду

всі плюси від мови C  
простий і самодекларуючий стиль написання коду  
виправлення помилок коду ще на стадії написання  
простота побудови графічного інтерфейсу  
великий набір інструментів розробника



# Філософія

## Підтримка декількох парадигм програмування

Перевикористання існуючого коду

object-oriented  
class-based

**Objective-C**  
reflective

**Swift**  
protocol-oriented  
functional  
imperative  
block structured

# Філософія

## Перевикористання існуючого коду

CocoaPods

Carthage

Swift Package Manager

Xcode Maven

Xcode Gradle

# «Hello, World!»

OBJ-C

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

int main(int argc, const char * argv[]) {
    @autoreleasepool {
        // insert code here...
        NSLog(@"Hello, World!");
    }
    return 0;
}
```

# «Hello, World!»



```
import Foundation  
print("Hello, World!")
```

# Синтаксис

//пронаслідуваний від C  
//побудований на принципах токєну

NSLog(@"Hello, World! \n»); // 6 токєнів

NSLog	//1
@	//2
(	//3
"Hello, World! \n»	//4
)	//5
;	//6

OBJ-C

# Синтаксис

OBJ-C

```
//Крапка з комою ;  
// термінатор для виразу
```

```
NSLog(@"Hello, World! \n");  
return 0;
```

```
//коментарі  
/* my first program in Objective-C */  
// this is also one-line comment
```

# Синтаксис

OBJ-C

//ідентифікатори

//чутливий до реєстру

//A до Z або a до z чи/та нижнього підкреслення \_ з 0 або більше символами, \_, та цифри (0 - 9)

//символи пунктуації не дозволено використовувати в назвах

# Синтаксис

//ключові слова

auto	else	long	switch
break	enum	register	typedef
case	extern	return	union
char	float	short	unsigned
const	for	signed	void
continue	goto	sizeof	volatile
default	if	static	while
do	int	struct	_Packed
double	protocol	interface	implementation
NSObject	NSInteger	NSNumber	CGFloat
property	nonatomic;	retain	strong
weak	unsafe_unretained;	readwrite	readonly

OBJ-C



# Синтаксис



```
//використовує принципи токенів  
print(«test!»)
```

```
//індивідуальні токени  
print(           //1  
«Hello World!»  //2  
)              //3
```

# Синтаксис



//Крапка з комою ;  
//необов'язковий термінатор для виразу

```
print(@"Hello, World! \n»); return 0;
```

\\новий рядок трактується як вираз  

```
print(@"Hello, World! \n»)  
return 0
```

//коментарі  
/\* multiline in Swift \*/  
// this is also one-line comment

# Синтаксис



//ідентифікатори

//чутливий до реєстру

//A до Z або a до z чи/та нижнього підкреслення \_ з 0 або більше символами, \_ та цифри (0 - 9)

//символи пунктуації не дозволено використовувати в назвах

//пробіли використовуються для ідентифікації літералів та виразів, мають бути присутні між операндами

# Синтаксис

## //ключові слова

### Keywords used in declarations

Class	deinit	Enum	extension
Func	import	Init	internal
Let	operator	private	protocol
public	static	struct	subscript
typealias	var		

### Keywords used in statements

break	case	continue	default
do	else	fallthrough	for
if	in	return	switch
where	while		

### Keywords used in expressions and types

as	dynamicType	false	is
nil	self	Self	super
true	_COLUMN_	_FILE_	_FUNCTION_
_LINE_			

### Keywords used in particular contexts

associativity	convenience	dynamic	didSet
final	get	infix	inout
lazy	left	mutating	none
nonmutating	optional	override	postfix
precedence	prefix	Protocol	required
right	set	Type	unowned
weak	willSet		



# Основні типи даних

Характеристика, яка надається об'єкту та визначає множину припустимих значень, формат їхнього збереження, кількість виділеної пам'яті, а також набір операцій, які можна виконувати над даними заданого типу.

Типи за значенням або value types

Типи за посиланням або reference types

# Основні типи даних

Цілочислові типи (integer)

Дійсні типи (floating-point)

Логічний тип (boolean)

Символьний тип (character)

Рядковий тип (string)

# Основні типи даних

OBJ-C

S.N.	Types and Description
1	<b>Basic Types:</b> They are arithmetic types and consist of the two types: (a) integer types and (b) floating-point types.
2	<b>Enumerated types:</b> They are again arithmetic types and they are used to define variables that can only be assigned certain discrete integer values throughout the program.
3	<b>The type void:</b> The type specifier <i>void</i> indicates that no value is available.
4	<b>Derived types:</b> They include (a) Pointer types, (b) Array types, (c) Structure types, (d) Union types and (e) Function types.

# Основні типи даних

OBJ-C

Type	Storage size	Value range
char	1 byte	-128 to 127 or 0 to 255
unsigned char	1 byte	0 to 255
signed char	1 byte	-128 to 127
int	2 or 4 bytes	-32,768 to 32,767 or -2,147,483,648 to 2,147,483,647
unsigned int	2 or 4 bytes	0 to 65,535 or 0 to 4,294,967,295
short	2 bytes	-32,768 to 32,767
unsigned short	2 bytes	0 to 65,535
long	4 bytes	-2,147,483,648 to 2,147,483,647
unsigned long	4 bytes	0 to 4,294,967,295

Type	Storage size	Value range	Precision
float	4 byte	1.2E-38 to 3.4E+38	6 decimal places
double	8 byte	2.3E-308 to 1.7E+308	15 decimal places
long double	10 byte	3.4E-4932 to 1.1E+4932	19 decimal places

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

```
int main()
```

```
{  
    NSLog(@"Storage size for int : %d \n", sizeof(int));
```

```
    return 0;
```

```
}
```

```
//2018-02-17 22:23:51.806534+0200
```

```
dataTypes[6836:2921407] Storage size for int : 4
```



# Основні типи даних



S.N.	Types and Description
1	<b>Function returns as void</b> There are various functions in Objective-C which do not return value or you can say they return void. A function with no return value has the return type as void. For example, <b>void exit (int status);</b>
2	<b>Function arguments as void</b> There are various functions in Objective-C which do not accept any parameter. A function with no parameter can accept as a void. For example, <b>int rand(void);</b>

# Основні типи даних



- **Int or UInt** – This is used for whole numbers. More specifically, you can use Int32, Int64 to define 32 or 64 bit signed integer, whereas UInt32 or UInt64 to define 32 or 64 bit unsigned integer variables. For example, 42 and -23.
- **Float** – This is used to represent a 32-bit floating-point number and numbers with smaller decimal points. For example, 3.14159, 0.1, and -273.158.
- **Double** – This is used to represent a 64-bit floating-point number and used when floating-point values must be very large. For example, 3.14159, 0.1, and -273.158.
- **Bool** – This represents a Boolean value which is either true or false.
- **String** – This is an ordered collection of characters. For example, "Hello, World!"
- **Character** – This is a single-character string literal. For example, "C"
- **Optional** – This represents a variable that can hold either a value or no value.
- **Tuples** – This is used to group multiple values in single Compound Value.

# Основні типи даних



Type	Typical Bit Width	Typical Range
Int8	1byte	-127 to 127
UInt8	1byte	0 to 255
Int32	4bytes	-2147483648 to 2147483647
UInt32	4bytes	0 to 4294967295
Int64	8bytes	-9223372036854775808 to 9223372036854775807
UInt64	8bytes	0 to 18446744073709551615
Float	4bytes	1.2E-38 to 3.4E+38 (~6 digits)
Double	8bytes	2.3E-308 to 1.7E+308 (~15 digits)

```
let intSize = MemoryLayout<Double>.size  
print(«Storage size for int: \((intSize)»)
```

```
//Storage size for int: 8
```

# Основні типи даних



```
//безпека типів  
var varA = 42  
varA = "This is hello» // -> Cannot assign value of type 'String' to type 'Int'  
print(varA)
```

# Типи даних: Tuple

Tuple - представлення сукупності значень одного або різних типів.

Опис кортежу можна представити у наступному вигляді:  
(тип значення1, тип значення 2, ...)

# Типи даних: Tuple



```
3 (Int, String).self
4
5 (12, "twelve")
6 (number: 12, string: "twelve")
7
8 (red: 255, green: 255, blue: 255, hex: 0xffffff, colorName: "white")
```

```
(Int, String).Type
(.0 12, .1 "twelve")
(.0 12, .1 "twelve")
(.0 255, .1 255, .2 255, .3 16777215, .4 "white")
```

# Константи та змінні

Константа - сутність, значення якої *не може* бути змінено після встановлення.

Змінна - сутність, значення якої може змінювати протягом її життя.

# Константи та змінні

Кожна сутність в Objective-C має явно або неявно вказаний тип даних.  
Неявне задання типу визначається ключовим словом *id*.



# Константи та змінні

Objective-C не використовує ключові слова для створення сутностей

КОНСТАНТИ - ключове слово *const*

ЗМІННІ - явно вказаний тип або `id` або `NSObject` (базовий для більшості об'єктів)

# Константи та змінні: Синтаксис

Objective-C визначає наступний синтаксис опису змінної:

```
@property (<#attributes#>) <#type#> <#name#>;
```

```
@property(copy) NSString *title;
```

# Константи та змінні

OBJ-C

```
@property(copy) NSString *title;
```

```
const int constIntValue = 12;
```

```
constIntValue = 3; //Cannot assign to variable 'constIntValue' with const-qualified type 'const int'
```

```
id someObject = [[Book alloc] init];
```

```
Book *someBookObject = [[Book alloc] init];
```

# Константи та змінні

Swift використовує два ключові слова для поділу сутностей на:

константи - ключове слово *let*

змінні - ключове слово *var*

# Константи та змінні: Типизація

Кожна сутність в Swift має явно або неявно вказаний тип даних.  
Неявне задання типу визначається компілятором.

# Константи та змінні: Синтаксис

Swift визначає наступний синтаксис опису константи:

```
let constantName[: Type] = constantValue
```

```
let constantName = constantValue
```

```
let constantName: Type = constantValue
```

# Константи та змінні: Синтаксис

Swift визначає наступний синтаксис опису змінної:

```
var variableName[: Type] = variableValue
```

```
var variableName = variableValue
```

```
var variableName: Type = variableValue
```

# Константи та змінні



1	//: Constants and Variables	
2		
3	// Numbers (Integers, Floating-Points)	
4		
5	let twentyDecimal = 20	20
6	let twentyBinary = 0b10100	20
7	let twentyOctal = 0o24	20
8	let twentyHexidecimal = 0x14	20
9	let twoPoint20: Float = 2.20	2.2
10	let minus24: Int8 = -24	-24
11	let plus256: UInt64 = 255	255
12	var tenPoint63 = 10.63	10.63
13	var twoHundred = 2_00	200
14		
15	twoHundred = 2_000	2000
16		
17	type(of: twentyDecimal)	Int.Type
18	type(of: minus24)	Int8.Type
19	type(of: tenPoint63)	Double.Type
20	type(of: twoPoint20)	Float.Type
21		
22	// Booleans	
23		
24	let valueTrue = true	true
25	var valueFalse: Bool = false	false
26		
27	type(of: valueTrue)	Bool.Type
28	type(of: valueFalse)	Bool.Type



# Константи та змінні



29		
30	let chA: Character = "a"	"a"
31	let chAnotherA = Character("a")	"a"
32	var chB: Character = "c"	"c"
33		
34	chB = "b"	"b"
35	type(of: chA)	Character.Type
36	type(of: chAnotherA)	Character.Type
37		
38	// Strings	
39		
40	let strA = "a"	"a"
41		
42	type(of: strA)	String.Type
43		
44	let hello = "Hello"	"Hello"
45	let world: String = "world"	"world"
46		
47	type(of: hello)	String.Type
48	type(of: world)	String.Type

# Optionals

Optional - це “надбудова” над певним типом даних, яка каже

сутність містить певне значення цього типу *або*  
сутність не містить жодного значення цього типу.

# Optionals: ?

Optionals у Swift описуються за допомогою знаку питання “?” після назви типу.

```
let constantName: Type? [ = constantValue]  
var variableName: Type? [ = variableValue ]
```

# Optionals



<pre>3 // Optional constants 4 5 let someConstatnt: Int? 6 7 someConstatnt = 5 8 9 // error if 10 // someConstatnt = 6 11 12 // Optional variable 13 14 var someVar: String? 15 var anotherVar: String? = "another" 16 17 someVar = "some" 18 19 // Optional tuples 20 21 let someConstTuple: (Int, String)? 22 var someVarTuple: (Int8, Character, String)? = (6, "6", "six") 23 24 someConstTuple = (5, "five") 25 26 // Type of optional 27 28 type(of:someConstTuple) 29 type(of:someVar)</pre>	<pre>5 nil "another" "some" (.0 6, .1 "6", .2 "six") (.0 5, .1 "five") Optional&lt;(Int, String)&gt;.Type Optional&lt;String&gt;.Type</pre>
---	---

# Optionals: nil

Swift використовує ключове слово *nil*, аби вказати на відсутність значення.

```
var variableName: Type? = nil
```

# Optionals



2		
3	// nil while using Optionals	
4		
5	var optInt: Int?	nil
6	var anotherOptInt: Int? = nil	nil
7		
8	optInt = 5	5
9	anotherOptInt = 55	55
10		
11	// nil and let	
12		
13	let constOptInt: Int? = nil	nil
14		
15	// error if	
16	//constOptInt = 555	

# Optionals: Unwrapping

Unwrapping - процес отримання значення, що містить у сутності optional типу.  
Для того, щоб *розгорнути* значення, яке містить optional-сутність, необхідно:

- 1) перевірити, чи сутність optional-типу містить значення,
- 2) використати forced unwrapping, якщо таке значення існує  
або використати механізм optional binding

# Optionals: Forced Unwrapping

Forced Unwrapping - процес примусового отримання значення, що містить у сутності optional типу.

Force Unwrapping може спричинювати виникнення помилки, якщо сутність optional типу не містить значення.

Swift представляє наступний синтаксис використання forced unwrapping:

*constantName!*

*variableName!*



# Optionals: Forced Unwrapping



```
3 // Forced Unwrapping (Situation 1)
4
5 var a: Int
6 var b: Int? = 5
7
8 a = b!
9
10 print(a, terminator:"")
11 print(b!)
```

```
5
5
"5"
"5\n"
```

# Optionals: Forced Unwrapping



```
2
3 // Forced Unwrapping (Situation 2)
4
5 var a: Int
6 var b: Int?
7
8 // error if: no value behind b
9 // a = b!
10 // print(b!)
11
12 // unwrap optional only if there is a value (not nil)
13
14 if b != nil {
15     a = b!
16 }
17
18 // error if: are might not have any value set by default
19 // print(a)
20
21 a = 0
22
23 if b != nil {
24     a == b!
25 }
26
27 print(a, terminator:"")
```

nil

0

"0"

# Optionals: Forced Unwrapping



```
3 // Forced Unwrapping (Situation 2)
4
5 var a: Int = 0
6 var b: Int?
7
8 // error if: no value behind b
9 // a = b!
10 // print(b!)
11
12 // unwrap optional only if there is a value (not nil)
13
14 if b != nil {
15     a = b!
16 }
17
18 print(a, terminator:"")
```

0  
nil

"0"

# Optionals: Forced Unwrapping



```
3 // Forced Unwrapping (Situation 3)
4
5 var a: Int = 0
6 var b: Int?
7
8 b = 10
9
10 // error if: no value behind b
11 // a = b!
12 // print(b!)
13
14 // unwrap optional only if there is a value (not nil)
15
16 if b != nil {
17     a = b!
18 }
19
20 print(a, terminator:"")
```

0
nil
10
10

# Optionals: Optional Binding

Optional Binding - процес безпечного отримання значення, що міститься у сутності optional типу.

Optional Binding *не* спричиняє виникнення помилки, якщо сутність optional типу не містить значення.

Swift представляє наступний синтаксис використання optional binding:

```
if let constantName = optionalEntity {  
    // unwrapped statement  
} else {  
    // nil statement  
}
```

# Optionals: Optional Binding



```
3 // Optional Binding
4
5 var a: Int
6 var b: Int? = 5
7
8 if let unwrappedB = b {
9     a = unwrappedB
10 } else {
11     a = 0
12 }
13
14 print(a, terminator:"")
```

5

5

"5"

# Optionals: Optional Binding



```
3 // Optional Binding
4
5 var a: Int
6 var b: Int? = 5
7
8 if let _ = b {
9     a = b!
10 } else {
11     a = 0
12 }
13
14 print(a, terminator:"")
```

5

5

"5"

# Optionals: Optional Binding



```
3 // Optional Binding
4
5 var a: Int
6 var b: Int? = 5
7
8 if let b = b {
9     a = b
10 } else {
11     a = 0
12 }
13
14 print(a, terminator:"")
```

5

5

"5"



# Optionals: Implicitly Unwrapped Optionals

Implicitly Unwrapped Optional - спосіб опису сутності optional-типу, значення якої вже задано.

Swift представляє наступний синтаксис опису сутності як `implicitly unwrapped optional`:

```
let constantName: Type! [ = constantValue ]  
var variableName: Type! [ = variableValue ]
```

# Optionals: Optional Binding



3 // Implicitly Unwrapped Optionals	
4	
5 var a: Int!	nil
6	
7 // error if: no default value set	
8 a = 5	5
9	
10 print(a, terminator:"")	"5"
11 type(of: a)	ImplicitlyUnwrappedOptional<Int>.Type
12	

# Список рекомендованих ресурсів до перегляду

Вступ до Swift - ([https://developer.apple.com/library/content/documentation/Swift/Conceptual/Swift\\_Programming\\_Language/index.html#//apple\\_ref/doc/uid/TP40014097-CH3-ID0](https://developer.apple.com/library/content/documentation/Swift/Conceptual/Swift_Programming_Language/index.html#//apple_ref/doc/uid/TP40014097-CH3-ID0))

Вступ до Objective-C -  
(<http://blog.teamtreehouse.com/an-introduction-to-objective-c> )

Про Objective-C - ([https://developer.apple.com/library/content/documentation/Cocoa/Conceptual/OOP\\_ObjC/Introduction/Introduction.html#//apple\\_ref/doc/uid/TP40005149-CH1-SW2](https://developer.apple.com/library/content/documentation/Cocoa/Conceptual/OOP_ObjC/Introduction/Introduction.html#//apple_ref/doc/uid/TP40005149-CH1-SW2))

The Swift Programming Language by Apple (ch Welcome to Swift)





**UNIVERSITY**