
SIGMA
Software


SIGNL
Software
UNIVERSITY


SIGMA
Software
UNIVERSITY

ЛЕКЦІЯ 11

“Замкнення”

11

SIGMA SOFTWARE UNIVERSITY

Замкнення

Замкнення в Objective-C

Замкнення в Swift

Замкнення

Замкнення в Objective-C

Замкнення в Swift

Замкнення

Замкнення (closure) - це автономна (замкнута) частина коду, яка виконує деяку функціональність.

Замкнення знають захоплювати посилання на різні об’єкти із контексту, де вони описані, що називається процесом замкнення.

Замкнення - це тип за посиланням.

Глобальні та вкладені функції є спеціальним видом замкнень.

Замкнення

Замкнення в Objective-C

Замкнення в Swift

Замкнення в Objective-C

OBJ-C

```
^{
   //block implementation
}
```

OBJ-C

```
void (^someBlockName)() = ^{
   //block implementation
}
```

SIGMA SOFTWARE UNIVERSITY

Блоки

block соже повертати значення

OBJ-C

```
int (^someBlockName)(int a, int b) = ^(int a,
int b) {
   return a+ b;
}

int sum = someBlockName(1,2); //3
```

можна використовувати значення з поза блоку

OBJ-C

```
int var = 9;

int (^someBlockName)(int a, int b) = ^(int
a, int b) {
   return a+ b + var;
}
int sum = someBlockName(1,2); //11
```

SIGMA SOFTWARE UNIVERSITY

Блок як тип

оголошення блоку як типу

OBJ-C

```
return_type (^block_name)(parameters)

typedef int(^SomeBlock)(BOOL flag, int value);

SomeBlock block = ^(BOOL flag, int value){
    // Implementation
};
```

Блоки як параметри

блок можу бути параметром методу

```
- (void)startWithCompletionHandler:(void(^)(NSData *data, NSError *error))completion;
```

OBJ-C

або визначеним типом

```
typedef void(^CompletionHandler) (NSData *data, NSError *error);

- (void)startWithCompletionHandler:(CompletionHandler)completion;
```

OBJ-C

SIGMA SOFTWARE UNIVERSITY

Блок як властивість

```
//someObject interface
@property (copy, nonatomic) void (^connectCompletion)(BOOL success);
```

OBJ-C

```
- (void)someMethod
{
    SomeObj *newObj = [[SomeObj alloc] init];
    if (newObj.connectCompletion) {
        newObj.connectCompletion(YES);
        newObj.connectCompletion = nil;
    }
}
```

OBJ-C

SIGMA SOFTWARE UNIVERSITY

Блок як змінна

```
int (^describeNumber)(int x) = ^(int x){
    return x * 2
};
```

OBJ-C

```
- (void)localVariable
{
    int (^describeNumber)(int x) = ^(int x){
        return x * 2;
    };
    int b = describeNumber(3);
    int c = b + 2;
    NSLog(@"number + 2 - %i", c);
}
```

OBJ-C

SIGMA SOFTWARE UNIVERSITY

__block

змінні захоплені до блоку не можуть бути змінені

OBJ-C

```
int var = 9;

int (^someBlockName)(int a, int b) = ^(int a,
int b) {
   var += 10; //error
   return a+ b + var;
}

int sum = someBlockName(1,2); //11
```

OBJ-C

```
__block int var = 9;

int (^someBlockName)(int a, int b) = ^(int
a, int b) {
   var += 10; //ok
   return a+ b + var;
}

int sum = someBlockName(1,2); //11
```

SIGMA SOFTWARE UNIVERSITY

__block

self натомість може бути зміненим

OBJ-C

```
- (void)anInstanceMethod {
    // ...
    void (^someBlock)() = ^{
        _anInstanceVariable = @"Something";
        NSLog(@"_anInstanceVariable = %@",
_anInstanceVariable);
    };
    // ...
}
```

SIGMA SOFTWARE UNIVERSITY

Strong Reference Cycles з блоками

OBJ-C

```
- (instancetype)init
{
    if ((self = [super init])) {
      self.block = ^{
            NSLog(@"object is %@", self); // self retain cycle
        };
    }
    return self;
}
```

SIGMA SOFTWARE UNIVERSITY

Strong Reference Cycles with blocks

використовуйте __weak (or __weak+__strong) для уникнення retain cycles

OBJ-C

```
- (instancetype)init
{
    if ((self = [super init])) {
        NSLog(@"init");
        __weak typeof(self) weakSelf = self;
        self.block = ^{
            __strong __typeof(weakSelf) strongSelf = weakSelf;
            NSLog(@"object is %@", strongSelf);
        };
    }
    return self;
}
```

SIGMA SOFTWARE UNIVERSITY

Блоки в деталях

Objective-C’s anonymous functions

Distinct segments of code that can be passed around to methods or functions as if they were values

A block is similar to a function but is defined inline to another function and shares the scope of that within which it is defined

Blocks it’s object

Created in stack local memory

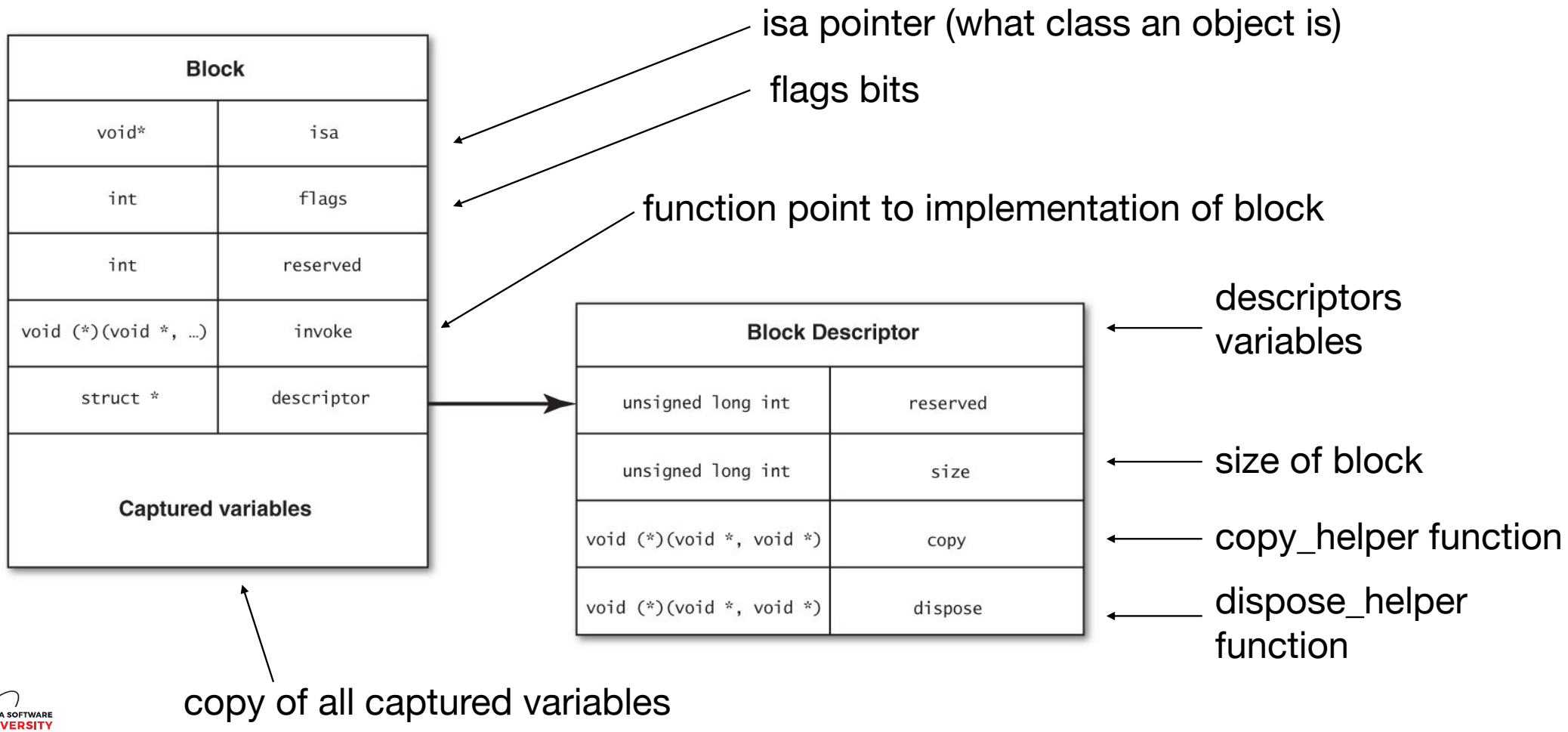
With copy block moved to heap

Блоки в деталях

OBJ-C

```
struct Block_literal_1 {
    void *isa; // initialized to &_NSConcreteStackBlock or &_NSConcreteGlobalBlock
    int flags;
    int reserved;
    void (*invoke)(void *, ...);
    struct Block_descriptor_1 {
        unsigned long int reserved;                 // NULL
        unsigned long int size;                     // sizeof(struct Block_literal_1)
        // optional helper functions
        void (*copy_helper)(void *dst, void *src);  // IFF (1<<25)
        void (*dispose_helper)(void *src);          // IFF (1<<25)
        // required ABI.2010.3.16
        const char *signature;                      // IFF (1<<30)
    } *descriptor;
    // imported variables
};
```

Блоки в деталях - layout



Типи блоків

__NSGlobalBlock__	never copied or disposed
__NSStackBlock__	stack allocated, cant be retained
__NSMallocBlock__	([__NSStackBlock__ copy]), cant be copied

OBJ-C

```
int x = 10; //some scope variable
int (^someBlock)() = counterBlock(); //copy x
int (^someBlockCopy)() = [someBlock copy]; //same x like and in someBlock
int (^anotherBlock)() = counterBlock(); //copy x
```

Замкнення

Замкнення в Objective-C

Замкнення в Swift

Замкнення

Замкнення в Objective-C

Замкнення в Swift

Замкнення

Swift надає наступний синтаксису опису замкнень:

```
{ (<Parameters>) -> <ReturnType> in
    // body
}
```

Замкнення: змінна або константа

```
8   var closure:(String) -> ()
9
10  func printText(text:String) -> () {
        print(text)                                   "Test string\n"
12  }
13
    closure = printText                               (String) -> ()
15  closure("Test string")
16
    var anotherClosure = { (text:String) -> String in (String) -> String
        return text                                   "Test string 2"
19  }
20
    anotherClosure("Test string 2")                   "Test string 2"
```

SIGMA SOFTWARE UNIVERSITY

Замкнення: користувацький тип

```
22  //typealias
23
24  typealias SomeClosure = (String) -> ()
25
26  func doSomething(_ text:String, closure:SomeClosure) {
        let newText = text + text
28      closure(newText)
29  }
30
31  doSomething("test") { (text) in
        print(text)
33  }
```

SIGMA SOFTWARE UNIVERSITY

Замкнення: тип значення, що повертає фнк.

```
    var something:SomeClosure = { (text:String) -> () in
        print(text)
37  }
38
39  func someOperation() -> SomeClosure {
        return something
41  }
42
43  someOperation()("Print text")
44
```

```
(String) -> ()
()


(String) -> ()
```

SIGMA SOFTWARE UNIVERSITY

Замкнення: параметр функції

```swift
//: Closure - As Type Alias

import Foundation

typealias Closure = (String) -> Void

var closure: Closure = { (text: String) -> Void in
    print(text)
}

func doSomething(closure: Closure) {
    closure("Hello World")
}

func doSomethingElse(closure: (String) -> Void) -> Void {
    closure("Hello Again")
}

doSomething(closure)
doSomethingElse(closure)
```

String -> ()
(2 times)

```
Hello World
Hello Again
```

SIGMA SOFTWARE UNIVERSITY

Замкнення: аргумент функції #1

```
47  // function argument
48
49  func printValue(values:(Int, Int), swapFunction:((Int, Int)) -> (Int, Int)) {
        print(swapFunction(values))                                          "(1, 0)\n"
51  }
52
    let value = (0,1)                                                        (.0 0, .1 1)
54
55  printValue(values: value) { (a, b) -> (Int, Int) in
        return (b, a)                                                        (.0 1, .1 0)
57  }
58
59
```

```
Test string
testtest
Print text
(1, 0)
```

SIGMA SOFTWARE UNIVERSITY

Замкнення: аргумент функції #2

```
55 printValue(values: value) { (a, b) -> (Int, Int) in
       return (b, a)                                          (.0 1, .1 0)
57 }
58
59 printValue(values: value) { (a, b)  in
       return (b, a)                                          (.0 1, .1 0)
61 }
62
63 printValue(values: value) {
       return ($0.1, $0.0)                                    (.0 1, .1 0)
65 }
```

SIGMA SOFTWARE UNIVERSITY

Замкнення: аргумент функції #3

```
67
    printValue(values: value) { ($0.1, $0.0) }
69
```

(.0 1, .1 0)

SIGMA SOFTWARE UNIVERSITY

Замкнення: trailing

Swift визначає т. зв. trailing closure - замкнення, яке представлене замість останнього параметра функції, який є замкненням:

```
accessLevel func funcName(…, <closure>) {
   // body
}

funcName(…) { <closure content>
}
```

Замкнення: аргумент функції (trailing)

```
67
    printValue(values: value) { ($0.1, $0.0) }
69
```

(.0 1, .1 0)

SIGMA SOFTWARE UNIVERSITY

Замкнення: список захоплення

Замкнення захоплюють сутності описані поза його межами.

При роботі із захопленням сутностей потрібно враховувати можливість виникнення витоків пам’яті пов’язаних з утворенням зацикленням посилань.

Swift дозволяє контролювати процес того, як захоплюють сутності за допомогою списку захоплень наступного вигляду:

```
{ [ <weak/unowned attribute> capturedElement, …] (…) -> ReturnType in
    // closure body
}
```

Замкнення: список захоплення

weak посилання - це посилання, яке дозволяє ARC звільнити пам’ять з-під екземпляра, якщо на нього більше ніхто не посилається.

weak посилання представлене optional типом.

unowned посилання - це посилання, як також дозволяє ARC звільнити пам’ять з-під екземпляра, якщо на нього більше ніхто не посилається.

unowned посилання, на відміну від weak посилання, завжди має значення.

Замкнення: @escaping

Якшо блок може виконатися після того як функція повернула значення то таке замкнення може бути позначеним словом @escaping

@escaping блок означає що якшо ми хочемо використати self всередині блоку то маємо робити це явно

```
accessLevel func funcName(…, @escaping <closure>) {
   // body
}
```

Замкнення: @escaping

```
var completionHandlers: [() -> Void] = []
func someFunctionWithEscapingClosure(completionHandler: @escaping () -> Void) {
    completionHandlers.append(completionHandler)
}
```

```
[]

[() -> ()]
```

SIGMA SOFTWARE UNIVERSITY

Замкнення: noescape

```
75  func someFunctionWithNonescapingClosure(closure: () -> Void) {
76      closure()
77  }
78
79  class SomeClass {
80      var x = 10
81      func doSomething() {
            someFunctionWithEscapingClosure { self.x = 100 }                    ()
            someFunctionWithNonescapingClosure { x = 200 }                      ()
84      }
85  }
86
    let instance = SomeClass()                                                  SomeClass
    instance.doSomething()                                                      SomeClass
    print(instance.x)                                                           "200\n"
90  // Prints "200"
91
    completionHandlers.first?()                                                 ()
    print(instance.x)                                                           "100\n"
94  // Prints "100"
```

SIGMA SOFTWARE UNIVERSITY

Замкнення: @autoclosure

@autoclosure - це замкнення наступного вигляду:

@autoclosure() -> ReturnType

Автозамкнення не приймає жодних параметрів і автоматично генерується для обгортки виразів які передані до функції як аргумент

зазвичай ми не створюємо @autoclosure

@autoclosure може бути @escaping

Замкнення: @autoclosure

```
97  //autoclosure
98
    var customersInLine = ["Chris", "Alex", "Ewa", "Barry", "Daniella"]
    print(customersInLine.count)
101 // Prints "5"
102
    let customerProvider = { customersInLine.remove(at: 0) }
    print(customersInLine.count)
105 // Prints "5"
106
    print("Now serving \(customerProvider())!")
108 // Prints "Now serving Chris!"
    print(customersInLine.count)
110 // Prints "4"
```

```
["Chris", "Alex", "Ewa", "...
"5\n"

(2 times)
"5\n"

"Now serving Chris!\n"

"4\n"
```

SIGMA SOFTWARE UNIVERSITY


SIGMA
Software
UNIVERSITY