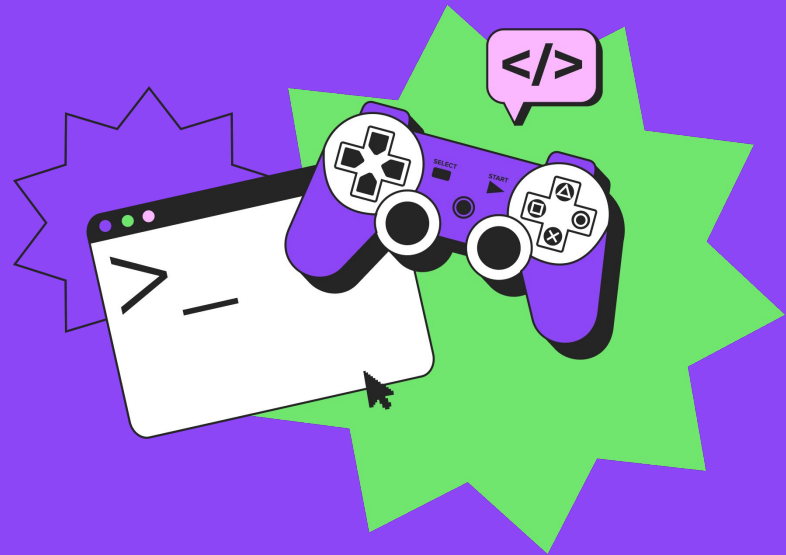


Знакомство с языком Objective-C

Урок 1

Знакомство с языком Objective-C. Изучение его предшественников. Отличия от других языков. Основные типы данных и арифметические операции. Обзор среды разработки Xcode. Организация файлов.












План курса



Что будет на уроке сегодня

-  Немного истории языка
-  Различия Objective – C vs Swift
-  Основы синтаксиса Objective – C
-  Арифметические операции
-  Условные выражения
-  Циклы
-  Приведение типов





Немного истории



История языка

Он был разработан в начале 1980-х Брэдом Коксом и Томом Лавом. Основная идея заключалась в том, чтобы объединить возможности Smalltalk-80 и популярность языка Си. В 1988 году NeXT лицензировала Objective-C и расширила компилятор GCC для поддержки языка. Разработал библиотеки AppKit и Foundation Kit, на которых основан NeXTSTEP.



Brad Cox



Tom Love



История языка

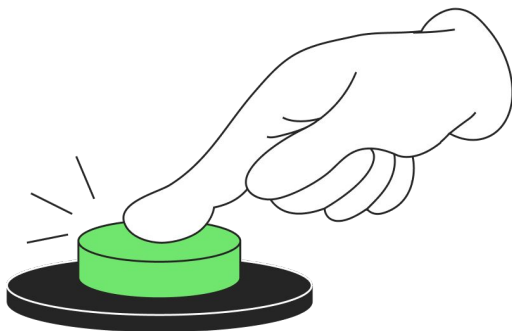
Apple Computer (Apple Inc.) приобрела NeXT в 1996 году и использовала ОС NeXTSTEP в качестве основы для разработки Mac OS X. Это включало Objective-C, его среду выполнения и инструменты разработки (позже Xcode).





Objective – C vs Swift

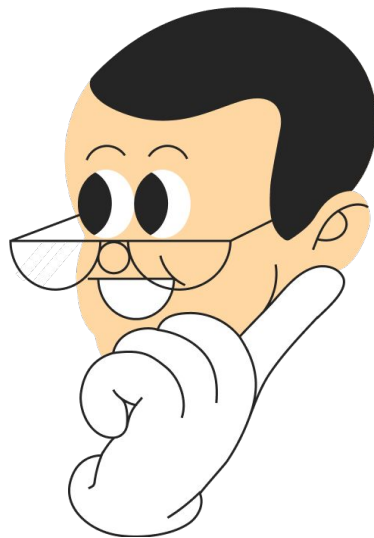
- ✓ Вывод типа (Type inference).
- ✓ Перегрузка оператора (Operator overloading).
- ✓ Расширения протокола (Protocol extensions).
- ✓ Интерполяция строк (String interpolation).
- ✓ Пространства имен (Namespaces).
- ✓ Кортежи (Tuples).
- ✓ Опционалы (Optionals).
- ✓ Playgrounds.
- ✓ Конструкций guard and defer.
- ✓ Закрытых и полуоткрытых диапазонов.
- ✓ Перечисления со связанными значениями.





Великое переименование Swift

Ранние версии Swift — от 1.0 до 2.2 — использовали почти те же соглашения об именах для методов и свойств, что и Objective-C. В Swift 3.0 Apple представила Великое переименование Cocoa, которое включало переименование почти каждого метода и свойства в «более Swifty», что приводило к тому, что почти каждый существующий проект ломался, пока он не был обновлен для использования новых соглашений об именах.



Пространство имен Objective – C

В Objective-C нет концепции пространств имен, а это означает, что все имена классов должны быть глобально уникальными. Это создает множество проблем, и решение Apple простое и универсальное: используйте двух-, трех- или четырехбуквенные префиксы, чтобы сделать каждое имя класса уникальным.

Подумайте об этом: UITableView, SKSpriteNode, MKMapView, XCTestCase — вы все время использовали этот префиксный подход, возможно, даже не осознавая, что он был разработан для устранения недостатка Objective-C.





Разделение кода

Когда вы создаете класс в Objective-C, он состоит из `YourClass.h` (заголовочный файл) и `YourClass.m` (файл реализации). Первоначально «m» означало «сообщения», но сегодня большинство людей считают его файлом «Implementation». Ваш заголовочный файл описывает, что класс предоставляет внешнему миру: свойства, к которым можно получить доступ, и методы, которые можно вызвать. В вашем файле реализации вы пишете фактический код для этих методов.





Разделение кода

Этого разделения между H и M нет в Swift, где весь класс или структура создается внутри одного файла. Но в Objective-C это важно: когда вы хотите использовать другой класс, компилятору достаточно прочитать H-файл, чтобы понять, как можно использовать этот класс.

Это позволяет вам использовать компоненты с закрытым исходным кодом, такие как аналитическая библиотека Google: они предоставляют вам файл H, который описывает, как работают их компоненты, и файл «.a», который содержит их скомпилированный исходный код.





NSLog vs Print

```
1 NSLog(@"Hello %@", @"World");
2 print("Hello Woeld")
3
4
5 NSLog(@"My age → %d", 12);
6 print("Hello \ (12)")
7
8 NSLog(@"Hello %@", @"World");
9 print("Hello Woeld")
10
11
12 NSLog(@"My age → %@", AnyObject);
13 print("Hello \ (AnyObject)")
```



Список спецификаций формата NSString

Specifier	Description
%@	Objective-C object, printed as the string returned by <code>descriptionWithLocale:</code> if available, or <code>description</code> otherwise. Also works with <code>CTypeRef</code> objects, returning the result of the <code>CFCopyDescription</code> function.
%%	'%' character.
%d, %D	Signed 32-bit integer (<code>int</code>).
%u, %U	Unsigned 32-bit integer (<code>unsigned int</code>).
%x	Unsigned 32-bit integer (<code>unsigned int</code>), printed in hexadecimal using the digits 0-9 and lowercase a-f.
%X	Unsigned 32-bit integer (<code>unsigned int</code>), printed in hexadecimal using the digits 0-9 and uppercase A-F.
%o, %O	Unsigned 32-bit integer (<code>unsigned int</code>), printed in octal.
%f	64-bit floating-point number (<code>double</code>).
%e	64-bit floating-point number (<code>double</code>), printed in scientific notation using a lowercase e to introduce the exponent.
%E	64-bit floating-point number (<code>double</code>), printed in scientific notation using an uppercase E to introduce the exponent.
%g	64-bit floating-point number (<code>double</code>), printed in the style of %e if the exponent is less than -4 or greater than or equal to the precision, in the style of %f otherwise.
%G	64-bit floating-point number (<code>double</code>), printed in the style of %E if the exponent is less than -4 or greater than or equal to the precision, in the style of %f otherwise.
%c	8-bit unsigned character (<code>unsigned char</code>).
%C	16-bit UTF-16 code unit (<code>unichar</code>).
%s	Null-terminated array of 8-bit unsigned characters. Because the %s specifier causes the characters to be interpreted in the system default encoding, the results can be variable, especially with right-to-left languages. For example, with RTL, %s inserts direction markers when the characters are not strongly directional. For this reason, it's best to avoid %s and specify encodings explicitly.
%S	Null-terminated array of 16-bit UTF-16 code units.
%p	Void pointer (<code>void *</code>), printed in hexadecimal with the digits 0-9 and lowercase a-f, with a leading 0x.
%a	64-bit floating-point number (<code>double</code>), printed in scientific notation with a leading 0x and one hexadecimal digit before the decimal point using a lowercase p to introduce the exponent.
%A	64-bit floating-point number (<code>double</code>), printed in scientific notation with a leading 0x and one hexadecimal digit before the decimal point using an uppercase P to introduce the exponent.
%F	64-bit floating-point number (<code>double</code>), printed in decimal notation.



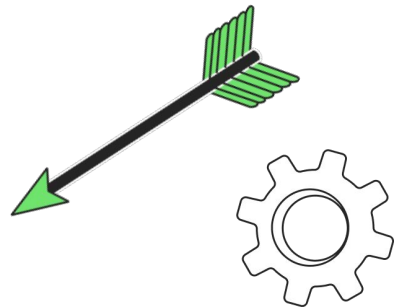
Функция main

```
int main(int argc, const char * argv[]) { }
```

И вот что означает каждая вещь:

- `int`: эта функция возвращает целое число.
- `main`: Функция называется `main()`.
- `int argc`: первый параметр представляет собой целое число с именем `argc`.
- `const char * argv[]`: Второй параметр представляет собой массив строк с именем `argv`.

Эта функция `main()` с этим параметром является стандартным способом создания программ командной строки, и она будет автоматически вызываться при запуске программы.





Синтаксические особенности

Еще одна синтаксическая особенность Objective-C – отправка сообщений (обращение к методу объекта). Для этого в квадратных скобках указывается объект и через пробел – его метод:

```
[object method];
```





Типы данных

Тип	Размер	Диапазон значений
char	1 байт	От -127 до 127
bool	1 байт	true, false
short int	2 байта	От -32767 до 32767
unsigned short int	2 байта	От 0 до 65535
int	4 байта	От -32767 до 32767
unsigned int	4 байта	От 0 до 65535



Типы данных

Тип	Размер	Диапазон значений
long int	4 байта	От -2147483647 до 2147483647
unsigned long int	4 байта	От 0 до 4294967295
float	4 байта	От $1E-37$ до $1E+37$ с точностью не менее 6 значащих десятичных цифр
double	8 байт	От $1E-37$ до $1E+37$ с точностью не менее 10 значащих десятичных цифр
long double	10 байт	От $1E-37$ до $1E+37$ с точностью не менее 10 значащих десятичных цифр



Указатели

Указатели – это переменная, значением которой является адрес другой переменной.

Есть **две основные операции**, которые применяются к указателям:

1. Захват адреса через `&`
2. Разыменование через `*`





```
#include <stdio.h>

int main(int argc, const char * argv[]) {
    int *a;
    int b = 222;
    a = &b;

    printf("%d\n", b);

    printf("%d\n", *a);

    *a = 1;
    // same as
    // b = 1;

    printf("%d\n", b);
    printf("%d\n", *a);

    return 0;
}
```

Объявление указателя



```
#include <stdio.h>
```

```
int main(int argc, const char * argv[]) {
```

```
    int b = 222;
```

```
    a = &b;
```

Присвоение
значения (адреса)

```
    printf("%d\n", b);
```

```
    printf("%d\n", *a);
```

```
    *a = 1;
```

```
    // same as
```

```
    // b = 1;
```

```
    printf("%d\n", b);
```

```
    printf("%d\n", *a);
```

```
    return 0;
```

```
}
```



```
#include <stdio.h>
```

```
int main(int argc, const char * argv[]) {
```

```
    int *a;
```

```
    int b = 123;
```

```
    a = &b;
```

```
    printf("%d\n", b);  
    printf("%d\n", *a);
```

Prints "123"

```
    *a = 432;
```

```
    // same as
```

```
    // b = 432;
```

```
    printf("%d\n", b);
```

```
    printf("%d\n", *a);
```

```
    return 0;
```

```
}
```



```
#include <stdio.h>

int main(int argc, const char * argv[]) {
    int *a;
    int b = 123;
    a = &b;

    printf("%d\n", b);

    printf("%d\n", *a);
    *a = 432;
    // same as
    // b = 432;

    printf("%d\n", b);
    printf("%d\n", *a);

    return 0;
}
```

Присвоение значения



```
#include <stdio.h>

int main(int argc, const char * argv[]) {
    int *a;
    int b = 123;
    a = &b;

    printf("%d\n", b);

    printf("%d\n", *a);

    *a = 432;
    // same as
    // b = 432;

    printf("%d\n", b);
    printf("%d\n", *a);

    return 0;
}
```

Prints "432"



Создание переменных

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

int main(int argc, const char * argv[]) {
    @autoreleasepool {
        NSInteger integer = 3;
        // Создание NSNumber из NSInteger
        NSNumber *integerNumber = [NSNumber numberWithInt:integer];
        // Создание NSNumber из BOOL
        NSNumber *boolNumber = [NSNumber numberWithBool:NO];
        // Создание NSNumber, используя литерал
        NSNumber *number = @1;
        // 3, 0, 1
        NSLog(@"%@, %@, %@", integerNumber, boolNumber, number);
    }
    return 0;
}
```




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

int main(int argc, const char * argv[]) {
    @autoreleasepool {
        int intValue = 10;
        char *charValue = "s";
        bool boolValue = false;
        float floatValue = 1.2;
        double doubleValue = 2.3;

        BOOL boolObjc = YES;
        NSInteger integer = 3;
        CGFloat cgFloat = 3.1;
        NSNumber *number = @1;
        NSString *string = @"Hello";

        NSLog(@"%d", intValue);          // 10
        NSLog(@"%s", charValue);         // s
        NSLog(@"%d", boolValue);         // 0
        NSLog(@"%f", floatValue);        // 1.200000
        NSLog(@"%f", doubleValue);       // 2.300000
        NSLog(@"%d", boolObjc);          // 1
        NSLog(@"%ld", (long)integer);    // 3
        NSLog(@"%f", cgFloat);           // 3.100000
        NSLog(@"%@", number);            // 1
        NSLog(@"%@", string);            // Hello
    }
    return 0;
}
```



Создание переменных

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

int main(int argc, const char * argv[]) {
    @autoreleasepool {
        NSNumber *number = @1;
        NSString *str = [NSString stringWithFormat:@"%d", number];
        NSString *anotherStr = @"Hello";

        NSLog(@"%@, %@", str, anotherStr);
    }
    return 0;
}
```



Арифметические операции

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

int main(int argc, const char * argv[]) {
    @autoreleasepool {
        int number1 = 10 + 15; // 25
        int number2 = 15 - 10; // 5
        int number3 = 10 * 15; // 150
        int number4 = 10 / 15; // 0
        int number5 = 10 % 2; // 0
    }
    return 0;
}
```



Арифметические операции

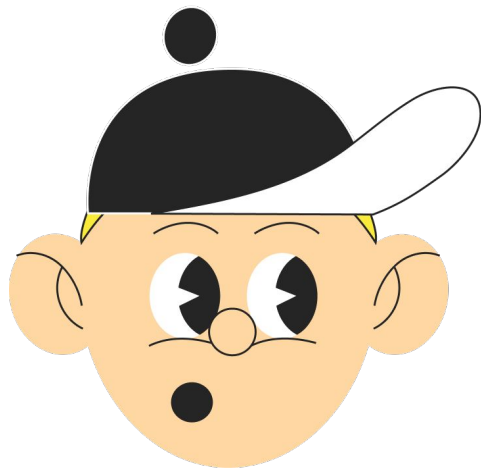
```
#import <Foundation/Foundation.h>

int main(int argc, const char * argv[]) {
    @autoreleasepool {
        int a = 0;
        int b = 1;
        b--;
        a++;
        NSLog(@"%d, %d", a, b);
    }
    return 0;
}
```



Условные выражения

Условные операторы в основном работают так же, как и в Swift, хотя вы всегда должны заключать свои условия в скобки. Эти круглые скобки, как и точка с запятой в конце строки, часто случайно пропускаются, когда вы переходите из Swift, но Xcode откажется компилироваться, пока вы это не исправите.





Условные выражения (if/else)

```
1 #import <Foundation/Foundation.h>
2
3 int main(int argc, const char * argv[]) {
4     @autoreleasepool {
5         int i = 10;
6         if (i == 10) {
7             NSLog(@"Привет, мир!");
8         } else {
9             NSLog(@"До свидания!");
10        }
11        if (i == 10)
12            NSLog(@"Привет, мир!");
13        else {
14            NSLog(@"До свидания!");
15        }
16
17        return 0;
18 }
```



Условные выражения(switch/case)

```
1 #import <Foundation/Foundation.h>
2
3 int main(int argc, const char * argv[]) {
4     @autoreleasepool {
5         int i = 20;
6         switch (i) {
7             case 20:
8                 NSLog(@"It's 20!");
9                 break;
10            case 40:
11                NSLog(@"It's 40!");
12                break;
13            case 60:
14                NSLog(@"It's 60!");
15                break;
16            default:
17                NSLog(@"It's something else.");
18        }
19
20    return 0;
21 }
```



Цикл while

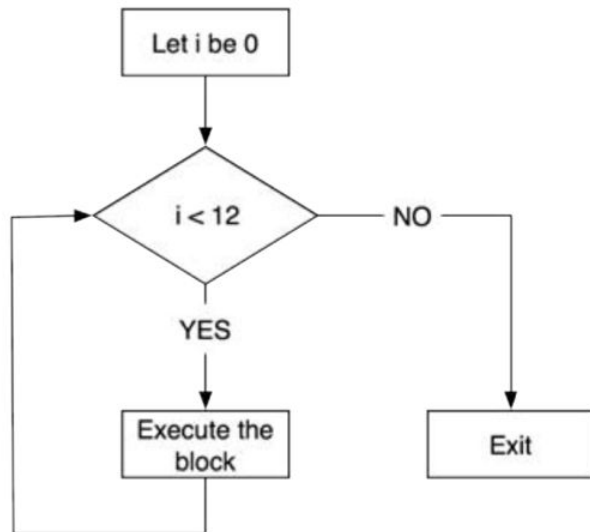
```
#include <stdio.h>
int main(int argc, const char * argv[]) {
    int i = 0;
    while (i < 12) {
        NSLog("%d. Aaron is Cool\n", i);
        i++;
    }
    return 0;
}
```




Цикл while

Результат:

0. Aaron is Cool
1. Aaron is Cool
2. Aaron is Cool
3. Aaron is Cool
4. Aaron is Cool
5. Aaron is Cool
6. Aaron is Cool
7. Aaron is Cool
8. Aaron is Cool
9. Aaron is Cool
10. Aaron is Cool
11. Aaron is Cool





Цикл do — while

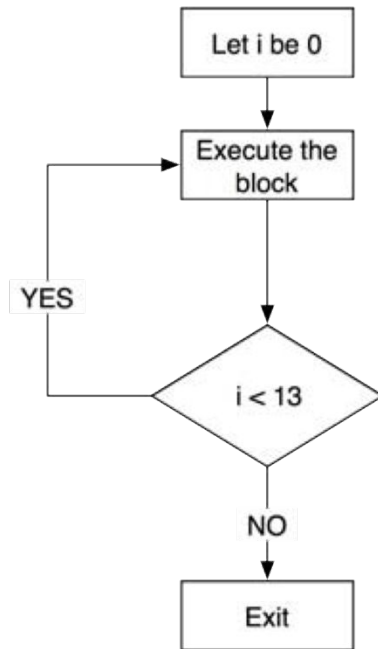
```
#include <stdio.h>
int main(int argc, const char * argv[]) {
{
    int i = 0;
    do {
        printf("%d. Aaron is Cool\n", i);
        i++;
    } while (i < 13);
    return 0;
}
```



Цикл while

Результат:

0. Aaron is Cool
1. Aaron is Cool
2. Aaron is Cool
3. Aaron is Cool
4. Aaron is Cool
5. Aaron is Cool
6. Aaron is Cool
7. Aaron is Cool
8. Aaron is Cool
9. Aaron is Cool
10. Aaron is Cool
11. Aaron is Cool





Цикл for

Objective-C имеет полный набор параметров цикла, включая цикл for в стиле C, который устарел в Swift 2.2. Начнем с наиболее распространенного типа цикла, известного как быстрое перечисление.

```
#include <stdio.h>
int main(int argc, const char * argv[]) {
    NSArray *names = @[@"Laura", @"Janet", @"Kim"];
    for (NSString *name in names) {
        NSLog(@"Hello, %@", name);
    }
    return 0;
}
```



Инструкция break

Иногда бывает нужно прервать выполнение цикла изнутри. Предположим, мы хотим перебрать все положительные числа в поисках такого числа x , для которого выполняется условие $x + 90 = x$. План действий: перебираем целые числа от 0 до 11 и при обнаружении нужного числа прерываем выполнение цикла.





Инструкция break

```
#include <stdio.h>
int main(int argc, const char * argv[]) {
    int i;
    for (i = 0; i < 12; i++) {
        printf("Checking i = %d\n", i);
        if (i + 90 == i * i) {
            break;
        }
    }
    printf("The answer is %d.\n", i);
    return 0;
}
```



Инструкция break

Результат:

Checking i = 0

Checking i = 1

Checking i = 2

Checking i = 3

Checking i = 4

Checking i = 5

Checking i = 6

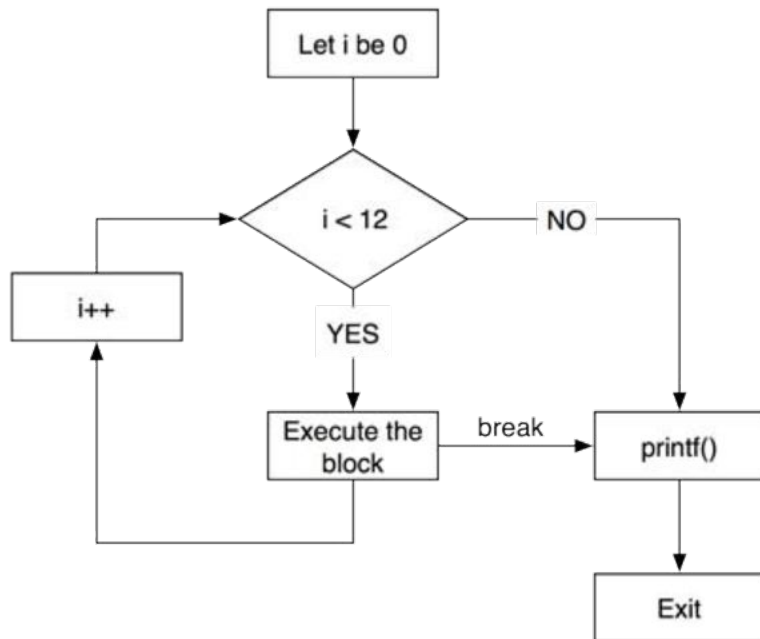
Checking i = 7

Checking i = 8

Checking i = 9

Checking i = 10

The answer is 10.

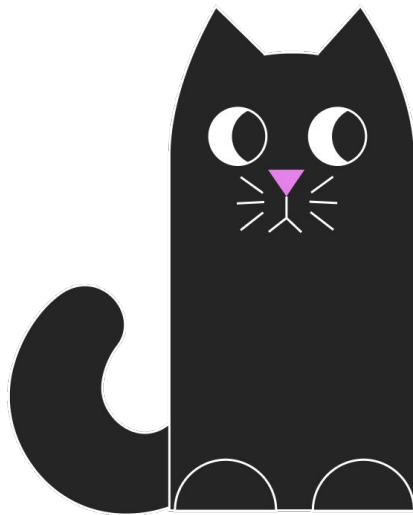




Инструкция `continue`

Иногда во время выполнения блока в цикле нужно сказать программе. «А теперь пропусти все, что осталось выполнить в блоке, и начини следующий проход». Эта задача решается командой `continue`. Допустим, вы твердо уверены в том, что для чисел, кратных 3, условие никогда не выполняется.

Как избежать напрасной потери времени на их проверку?





Инструкция continue

```
#include <stdio.h>
int main(int argc, const char * argv[]) {
    int i;
    for (i = 0; i < 12; i++) {
        if (i % 3 == 0) {
            continue;
        }
        printf("Checking i = %d\n", i);
        if (i + 90 == i * i) {
            break;
        }
    }
    printf("The answer is %d.\n", i);
    return 0;
}
```



Инструкция continue

Результат:

Checking i = 1

Checking i = 2

Checking i = 4

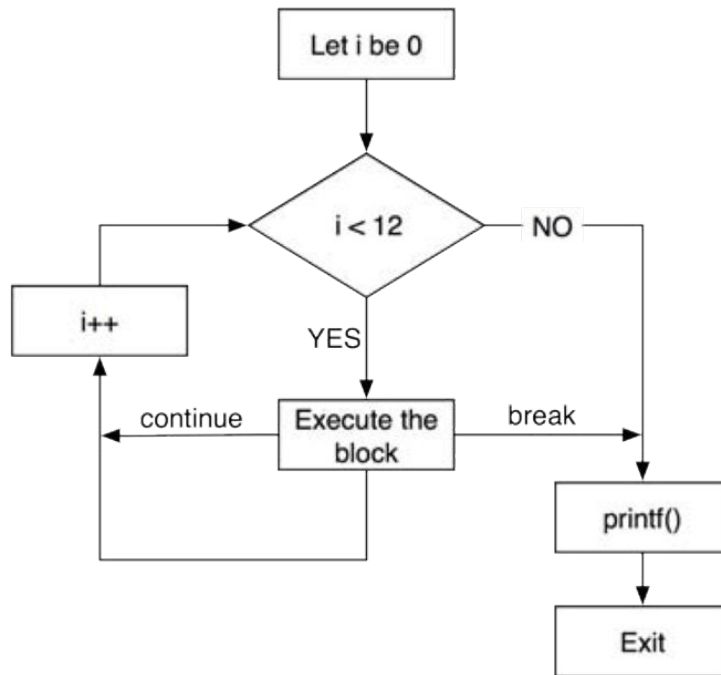
Checking i = 5

Checking i = 7

Checking i = 8

Checking i = 10

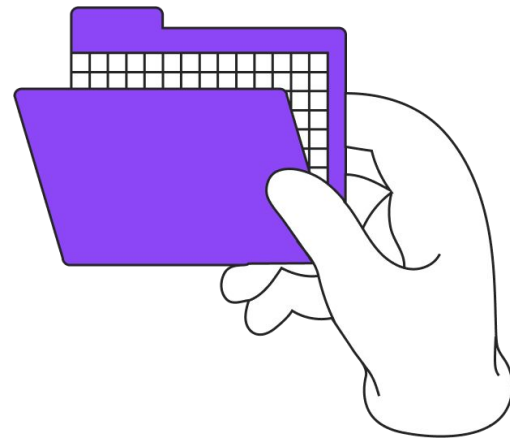
The answer is 10.





Инструкция switch case

В Objective-C менее мощен, чем Swift, поэтому вам нужно проделать больше работы самостоятельно, а во-вторых, операторы case имеют неявные отказы. Это противоположность Swift и означает, что вы почти всегда хотите написать break в конце блоков case, чтобы избежать провала.





Инструкция switch case

```
#include <stdio.h>
int main(int argc, const char * argv[]) {
    int i = 20;
    switch (i) {
        case 20:
            NSLog(@"It's 20!");
            break;
        case 40:
            NSLog(@"It's 40!");
            break;
        case 60:
            NSLog(@"It's 60!");
            break;
        default:
            NSLog(@"It's something else.");
    }
    return 0;
}
```



Приведение типов

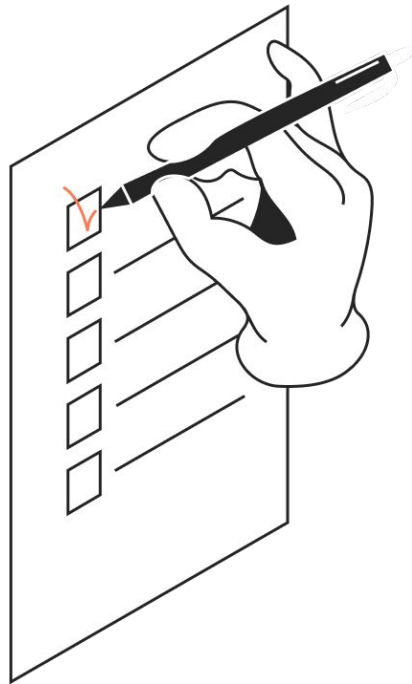
Чтобы получить переменную определенного типа из другого, можно воспользоваться приведением типов. Допустим, что нам дано десятичное число, но необходимо, чтобы результат был в виде целого числа.

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

int main(int argc, const char * argv[]) {
    @autoreleasepool {
        double value = 1.2;
        int number = (int)value;
        NSLog(@"%d", number);
    }
    return 0;
}
```

Что почитать?

- 📌 Стивен Кочан. «Программирование на Objective-C».
- 📌 Скотт Кнастер, Вакар Малик, Марк Далримпл. «Objective-C. Программирование для Mac OS X и iOS».
- 📌 <https://www.objc.io/>
- 📌 <https://developer.apple.com/library/archive/documentation/Cocoa/Conceptual/ProgrammingWithObjectiveC/>





Вопросы?

Вопросы?



Вопросы?

