

Objective-C®

Notes for Professionals

Chapter 2: Basic Data Types

Section 2.1: SEL

Selectors are used as method identifiers in Objective-C.

In the example below, there are two selectors, `new` and `setName`:

```
Person *customer = [Person new];  
[customer setName:@"John Doe"];
```

Each pair of brackets corresponds to a message send. On the first line we send a message selector to the `Person` class and on the second line we send a message containing the selector to the `customer` instance. In these cases, the `SEL` type can be used to hold a reference to the selector itself. In these cases, the `SEL` type can be used to hold a reference to the selector.

If the selector is available at compile time, you can use `#selector()` to get a reference:

```
SEL s = #selector(setName);  
Add if you need to find the selector at runtime, use NSSelectorFromString.
```

```
SEL s = [NSSelectorFromString(@"setName")];  
When using NSSelectorFromString, make sure to wrap the selector name in a NSString.
```

It is commonly used to check if a delegate implements an optional method:

```
if ([self.delegate respondsToSelector:#selector(doesSomething)]) {  
    // self.delegate does something;  
}
```

Section 2.2: BOOL

The `BOOL` type is used for boolean values in Objective-C. It has two values, `YES` and `NO`, which are identical to the C language.

Its behavior is straightforward and identical to the C language.

```
BOOL areEqual = (1 == 1); // areEqual is YES  
BOOL areNotEqual = (1 != 2); // areNotEqual is NO  
NSAssert(areEqual, @"Mathematics is a lie!"); // Assertion failed
```

```
BOOL shouldInitiateOrder = YES;  
if (shouldInitiateOrder) {  
    NSLog(@"Only the very smartest programmers read this kind  
of code.");  
}
```

A `BOOL` is a primitive, and so it cannot be stored directly in a Foundation collection. It must be wrapped in an `NSNumber`. Clang provides special syntax for this:

```
NSNumber *yes = YES; // Equivalent to [NSNumber numberWithInt:YES]
```

Chapter 5: Classes and Objects

Section 5.1: Difference between allocation and initialization

In most object-oriented languages, allocating memory for an object and initializing it is an atomic operation.

```
// Both allocate memory and call the constructor  
MyClass object = new MyClass();
```

In Objective-C, these are separate operations. The class method `alloc` (and its historic sibling `allocWithZone:`) makes the Objective-C runtime reserve the required memory and clears it. Except for a few internal values, all properties and variables are set to `0` or `nil`.

The object then is already "alive" but we always want to call a method to actually set up the object, which we call an initializer. These serve the same purpose as constructors in other languages. By convention, these methods start with `init`. From a language point of view, they are just normal methods.

```
// Allocate memory and set all properties and variables to 0/NO/nil.  
MyClass *object = [MyClass alloc];  
// Initialize the object.  
object = [object init];  
  
// Shortcut:  
object = [[MyClass alloc] init];
```

Section 5.2: Creating classes with initialization values

```
#import <Foundation/Foundation.h>  
#interface Car {  
    NSString *carMotorCode;  
    NSString *carChassisCode;  
};  
  
- (instancetype)initWithMotorValue:(NSString *) motorCode andChassisValue:(NSInteger)chassisCode;  
- (void) startCar;  
- (void) stopCar;  
  
@end  
  
@implementation Car  
  
- (instancetype)initWithMotorValue:(NSString *) motorCode andChassisValue:(NSInteger)chassisCode {  
    Car *motorCode = motorCode;  
    Car *chassisCode = chassisCode;  
    return self;  
}  
  
- (void) startCar (...) {  
- (void) stopCar (...) {  
  
@end
```

The method `initWithMotorValue: type andChassisValue: type` will be used to initialize the `Car` object.

Objective-C Notes for Professionals

Chapter 13: NSString

The `NSString` class is a part of Foundation framework to work with strings (series of characters). It also includes methods for comparing, searching and modifying strings.

Section 13.1: Encoding and Decoding

```
// encode  
NSString *string = [NSString alloc] initWithBytes:UTF8Data  
encoding:NSUTF8StringEncoding];  
  
// decode  
NSData *utf8Data = [string dataUsingEncoding:NSUTF8StringEncoding];  
Some supported encodings are:
```

- `NSStringEncoding`
- `NSStringEncoding`
- `NSStringEncoding`

Note that `utf8Data` bytes does not include a terminating null character, which is necessary for C strings. If you need a C string, use `UTF8String`:

```
const char *string = [string UTF8String];  
printf("%s", string);
```

Section 13.2: String Length

`NSString` has a `length` property to get the number of characters.

```
NSString *string = @"example";  
NSInteger length = string.length; // length equals 7
```

As in the `Splitting Example`, keep in mind that `NSString` uses `UTF-16` to represent characters. The length is actually just the number of UTF-16 code units. This can differ from what the user perceives as characters.

Here are some cases that might be surprising:

```
NSString *string = @"é";  
string.length == 1 // LATIN SMALL LETTER E WITH ACUTE (U+00E9)  
NSString *string = @"é";  
string.length == 2 // LATIN SMALL LETTER E (U+0045) + COMBINING ACUTE ACCENT (U+0301)  
NSString *string = @"♥";  
string.length == 4 // REGIONAL INDICATOR SYMBOL LETTER J (U+2600) + REGIONAL INDICATOR SYMBOL LETTER J (U+2600)
```

In order to get the number of user-perceived characters, known technically as "grapheme clusters", you must iterate over the string with `enumerateStringPairsWithOptions:usingBlock:`, and keep a count. This is demonstrated in an answer by Nikolai Buhler on Stack Overflow.

Section 13.3: Comparing Strings

Strings are compared for equality using `isEqualToString:`.

The `==` operator just tests for object identity and does not compare the logical values of objects, so it can't be used.

Objective-C Notes for Professionals

100+ pages
of professional hints and tricks

Contents

About	1
Chapter 1: Getting started with Objective-C Language	2
Section 1.1: Hello World	2
Chapter 2: Basic Data Types	4
Section 2.1: SEL	4
Section 2.2: BOOL	4
Section 2.3: id	5
Section 2.4: IMP (implementation pointer)	5
Section 2.5: NSInteger and NSUInteger	6
Chapter 3: Enums	8
Section 3.1: typedef enum declaration in Objective-C	8
Section 3.2: Converting C++ std::vector<Enum> to an Objective-C Array	8
Section 3.3: Defining an enum	9
Chapter 4: Structs	10
Section 4.1: Defining a Structure and Accessing Structure Members	10
Section 4.2: CGPoint	10
Chapter 5: Classes and Objects	12
Section 5.1: Difference between allocation and initialization	12
Section 5.2: Creating classes with initialization values	12
Section 5.3: Specifying Generics	13
Section 5.4: Singleton Class	13
Section 5.5: The "instancetype" return type	14
Chapter 6: Inheritance	15
Section 6.1: Car is inherited from Vehicle	15
Chapter 7: Methods	17
Section 7.1: Class methods	17
Section 7.2: Pass by value parameter passing	17
Section 7.3: Pass by reference parameter passing	17
Section 7.4: Method parameters	18
Section 7.5: Create a basic method	18
Section 7.6: Return values	19
Section 7.7: Calling methods	19
Section 7.8: Instance methods	20
Chapter 8: Properties	21
Section 8.1: Custom getters and setters	21
Section 8.2: Properties that cause updates	22
Section 8.3: What are properties?	23
Chapter 9: Random Integer	26
Section 9.1: Basic Random Integer	26
Section 9.2: Random Integer within a Range	26
Chapter 10: BOOL / bool / Boolean / NSCFBoolean	27
Section 10.1: BOOL/Boolean/bool/NSCFBoolean	27
Section 10.2: BOOL VS Boolean	27
Chapter 11: Continue and Break!	28
Section 11.1: Continue and Break Statement	28

Chapter 12: Key Value Coding / Key Value Observing	29
Section 12.1: Most Common Real Life Key Value Coding Example	29
Section 12.2: Querying KVC Data	29
Section 12.3: Collection Operators	30
Section 12.4: Key Value Observing	32
Chapter 13: NSString	35
Section 13.1: Encoding and Decoding	35
Section 13.2: String Length	35
Section 13.3: Comparing Strings	35
Section 13.4: Splitting	36
Section 13.5: Searching for a Substring	37
Section 13.6: Creation	37
Section 13.7: Changing Case	38
Section 13.8: Removing Leading and Trailing Whitespace	38
Section 13.9: Joining an Array of Strings	38
Section 13.10: Formatting	39
Section 13.11: Working with C Strings	39
Section 13.12: Reversing a NSString Objective-C	39
Chapter 14: NSArray	41
Section 14.1: Creating Arrays	41
Section 14.2: Accessing elements	41
Section 14.3: Using Generics	41
Section 14.4: Reverse an Array	42
Section 14.5: Converting between Sets and Arrays	42
Section 14.6: Converting NSArray to NSMutableArray to allow modification	42
Section 14.7: Looping through	42
Section 14.8: Enumerating using blocks	43
Section 14.9: Comparing arrays	43
Section 14.10: Filtering Arrays With Predicates	43
Section 14.11: Sorting array with custom objects	44
Section 14.12: Sorting Arrays	44
Section 14.13: Filter NSArray and NSMutableArray	45
Section 14.14: Add objects to NSArray	45
Section 14.15: Finding out the Number of Elements in an Array	45
Section 14.16: Creating NSArray instances	45
Chapter 15: NSMutableArray	46
Section 15.1: Sorting Arrays	46
Section 15.2: Creating an NSMutableArray	46
Section 15.3: Adding elements	46
Section 15.4: Insert Elements	46
Section 15.5: Deleting Elements	46
Section 15.6: Move object to another index	47
Section 15.7: Filtering Array content with Predicate	47
Chapter 16: NSDictionary	48
Section 16.1: Create	48
Section 16.2: Fast Enumeration	48
Section 16.3: Creating using literals	48
Section 16.4: Creating using dictionaryWithObjectsAndKeys:	48
Section 16.5: NSDictionary to NSArray	49
Section 16.6: NSDictionary to NSData	49

Section 16.7: NSDictionary to JSON	49
Section 16.8: Creating using plists	49
Section 16.9: Setting a Value in NSDictionary	49
Section 16.10: Getting a Value from NSDictionary	50
Section 16.11: Check if NSDictionary already has a key or not	50
Section 16.12: Block Based Enumeration	50
Chapter 17: NSMutableDictionary	52
Section 17.1: NSMutableDictionary Example	52
Section 17.2: Removing Entries From a Mutable Dictionary	53
Chapter 18: NSDate	55
Section 18.1: Convert NSDate that is composed from hour and minute (only) to a full NSDate	55
Section 18.2: Converting NSDate to NSString	55
Section 18.3: Creating an NSDate	56
Section 18.4: Date Comparison	56
Chapter 19: NSURL	58
Section 19.1: Create	58
Section 19.2: Compare NSURL	58
Section 19.3: Modifying and Converting a File URL with removing and appending path	58
Chapter 20: NSURL send a post request	60
Section 20.1: Simple POST request	60
Section 20.2: Simple Post Request With Timeout	60
Chapter 21: NSData	61
Section 21.1: Create	61
Section 21.2: NSData and Hexadecimal String	61
Section 21.3: Get NSData length	62
Section 21.4: Encoding and decoding a string using NSData Base64	62
Chapter 22: NSPredicate	63
Section 22.1: Filter By Name	63
Section 22.2: Find movies except given ids	64
Section 22.3: Find all the objects which is of type movie	64
Section 22.4: Find Distinct object ids of array	64
Section 22.5: Find movies with specific ids	64
Section 22.6: Case Insensitive comparison with exact title match	64
Section 22.7: Case sensitive with exact title match	64
Section 22.8: Case Insensitive comparison with matching subset	65
Chapter 23: NSRegularExpression	66
Section 23.1: Check whether a string matches a pattern	66
Section 23.2: Find all the numbers in a string	66
Chapter 24: NSJSONSerialization	67
Section 24.1: JSON Parsing using NSJSONSerialization Objective-C	67
Chapter 25: NSCalendar	69
Section 25.1: System Locale Information	69
Section 25.2: Initializing a Calendar	69
Section 25.3: Calendrical Calculations	69
Chapter 26: NSAttributedString	71
Section 26.1: Using Enumerating over Attributes in a String and underline part of string	71
Section 26.2: Creating a string that has custom kerning (letter spacing) editshare	71
Section 26.3: Create a string with text struck through	71
Section 26.4: How you create a tri-color attributed string	72

Chapter 27: NSTimer	73
Section 27.1: Storing information in the Timer	73
Section 27.2: Creating a Timer	73
Section 27.3: Invalidating a timer	73
Section 27.4: Manually firing a timer	74
Chapter 28: NSObject	75
Section 28.1: NSObject	75
Chapter 29: NSSortDescriptor	76
Section 29.1: Sorted by combinations of NSSortDescriptor	76
Chapter 30: NSTextAttachment	77
Section 30.1: NSTextAttachment Example	77
Chapter 31: NSCache	78
Section 31.1: NSCache	78
Chapter 32: NSUserDefaults	79
Section 32.1: Simple example	79
Section 32.2: Clear NSUserDefaults	79
Chapter 33: Subscripting	80
Section 33.1: Subscripts with NSArray	80
Section 33.2: Custom Subscripting	80
Section 33.3: Subscripts with NSDictionary	80
Chapter 34: Low-level Runtime Environment	82
Section 34.1: Augmenting methods using Method Swizzling	82
Section 34.2: Attach object to another existing object (association)	83
Section 34.3: Calling methods directly	83
Chapter 35: Fast Enumeration	85
Section 35.1: Fast enumeration of an NSArray with index	85
Section 35.2: Fast enumeration of an NSArray	85
Chapter 36: Categories	86
Section 36.1: Conforming to protocol	86
Section 36.2: Simple Category	86
Section 36.3: Declaring a class method	86
Section 36.4: Adding a property with a category	87
Section 36.5: Create a Category on XCode	87
Chapter 37: Protocols	91
Section 37.1: Optional and required methods	91
Section 37.2: Checking existence of optional method implementations	91
Section 37.3: Forward Declarations	91
Section 37.4: Conforming to Protocols	92
Section 37.5: Basic Protocol Definition	92
Section 37.6: Check conforms Protocol	92
Chapter 38: Protocols and Delegates	93
Section 38.1: Implementation of Protocols and Delegation mechanism	93
Chapter 39: Blocks	94
Section 39.1: Block Typedefs	94
Section 39.2: Blocks as Properties	94
Section 39.3: Blocks as local variables	95
Section 39.4: Blocks as Method Parameters	95
Section 39.5: Defining and Assigning	95

[Click here to download full PDF material](#)