Objective-C Cheatsheet

A quick reference for Objective-C syntax, data types, control structures, and object-oriented programming concepts. This cheat sheet provides a concise overview for both beginners and experienced developers.

Fundamentals

Basic Syntax

Header File	`#import <Foundation/Foundation.h>`
Implementation File	`#import "YourClass.h"`
Comments	`// Single-line comment` `/* Multi-line comment */`
NSLog	`NSLog(@"Hello, World!");`
Variables	`int age = 30;` `NSString *name = @"John";`

Data Types

Integer	`int age = 30;`
Float	`float price = 99.99;`
Double	`double pi = 3.14159265359;`
Boolean	`BOOL isTrue = YES;` (`YES` or `NO`)
Character	`char initial = 'J';`
NSString	`NSString *message = @"Hello, Objective-C!";`

Operators

Arithmetic	`+, -, *, /, %`
Assignment	`=, +=, -=, *=, /=, %=`
Comparison	`==, !=, >, <, >=, <=`
Logical	`&&, ||, !`
Bitwise	`&, |, ^, ~, <<, >>`

Control Structures

Conditional Statements

If Statement

```
if (condition) {
    // Code to execute if condition is true
}
```

If-Else Statement

```
if (condition) {
    // Code if condition is true
} else {
    // Code if condition is false
}
```

Else If Statement

```
if (condition1) {
    // Code if condition1 is true
} else if (condition2) {
    // Code if condition2 is true
} else {
    // Code if all conditions are false
}
```

Switch Statement

```
switch (variable) {
    case constant1:
        // Code to execute if variable == constant1
        break;
    case constant2:
        // Code to execute if variable == constant2
        break;
    default:
        // Code to execute if variable doesn't match any constant
        break;
}
```

Looping

For Loop

```
for (int i = 0; i < 10; i++) {
    // Code to be executed
}
```

While Loop

```
while (condition) {
    // Code to be executed while condition is true
}
```

Do-While Loop

```
do {
    // Code to be executed at least once
} while (condition);
```

For-In Loop (Fast Enumeration)

```
NSArray *array = @[@"apple", @"banana", @"cherry"];
for (NSString *item in array) {
    NSLog(@"%@", item);
}
```

Object-Oriented Programming

Classes

Defining a Class (.h file)

```
@interface MyClass : NSObject
{
    // Instance variables
    int myInteger;
    NSString *myString;
}

// Methods
- (void)myMethod;
+ (int)myClassMethod;

@end
```

Implementing a Class (.m file)

```
#import "MyClass.h"

@implementation MyClass

- (void)myMethod {
    NSLog(@"Instance method called");
}

+ (int)myClassMethod {
    return 42;
}

@end
```

Properties

Defining Properties in .h file

```
@interface MyClass : NSObject

@property (nonatomic, strong) NSString *name;
@property (nonatomic, assign) int age;

@end
```

Synthesizing Properties in .m file (Prior to Xcode 4.4)

```
@implementation MyClass

@synthesize name, age;

@end
```

Accessing Properties

```
MyClass *myObject = [[MyClass alloc] init];
myObject.name = @"John";
int age = myObject.age;
```

Methods

Instance Method	`-(returnType)methodName:(parameterType)parameterName;` Example: `-(void)printName:(NSString *)name;`
Class Method	`+(returnType)methodName:(parameterType)parameterName;` Example: `+(MyClass*)createObject;`
Method Implementation	`-(void)printName:(NSString *)name { NSLog(@"Name: %@", name); }`

Protocols

Defining a Protocol

```
@protocol MyProtocol

- (void)myMethod;
@optional
- (void)optionalMethod;

@end
```

Adopting a Protocol

```
@interface MyClass : NSObject <MyProtocol>

@end

@implementation MyClass

- (void)myMethod {
    NSLog(@"Method implemented from
MyProtocol");
}

@end
```

Memory Management

Automatic Reference Counting (ARC)

ARC automates memory management by tracking object ownership using reference counting. The compiler inserts retain and release calls as needed.

To enable ARC, set the compiler flag `-fobjc-arc`.

When ARC is enabled, you no longer need to manually call `retain`, `release`, or `autorelease`.

Strong and Weak References

Strong Reference	A strong reference increases the retain count of an object. It keeps the object alive as long as the strong reference exists. `@property (nonatomic, strong) NSObject *myObject;`
Weak Reference	A weak reference does *not* increase the retain count. When the object is deallocated, the weak reference is automatically set to `nil`. Used to prevent retain cycles. `@property (nonatomic, weak) NSObject *myObject;`
Unretained Reference	An unretained reference is similar to a weak reference, but it is *not* set to `nil` when the object is deallocated. Accessing an unretained reference to a deallocated object will result in a crash (dangling pointer). `@property (nonatomic, unsafe_unretained) NSObject *myObject;` (Use with caution.)

Autorelease Pool

An `autoreleasepool` is a mechanism to defer the `release` of objects. Objects sent the `autorelease` message are added to the pool and released when the pool is drained.

In ARC, you rarely need to use `@autoreleasepool` blocks directly, as the system often manages them for you (e.g., in the main event loop).

Example:

```
@autoreleasepool {
    // Code that creates autoreleased objects
}
// Objects are released when the pool is
drained
```