EBook Gratis

APRENDIZAJE

Objective-C Language

Free unaffiliated eBook created from
Stack Overflow contributors.

#objective-c

Tabla de contenido

Acerca de

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: objective-c-language

It is an unofficial and free Objective-C Language ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official Objective-C Language.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Capítulo 1: Comenzando con el lenguaje Objective-C

Versiones

Versión	Fecha de lanzamiento
1.0	1983-01-01
2.0	2007-10-27
Moderno	2014-03-10

Examples

Hola Mundo

Este programa dará salida a "Hello World!"

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

int main(int argc, char * argv[]) {
    NSLog(@"Hello World!");
}
```

`#import` es una directiva de preprocesador, que indica que queremos *importar* o incluir la información de ese archivo en el programa. En este caso, el compilador copiará el contenido de `Foundation.h` en el marco de `Foundation` en la parte superior del archivo. La principal diferencia entre #import y #include es que #import es lo suficientemente "inteligente" para no reprocesar los archivos que ya se han incluido en otros #includes.

La documentación del lenguaje C explica la función `main` .

La función `NSLog()` imprimirá la cadena proporcionada a la consola, junto con cierta información de depuración. En este caso, usamos un literal de cadena de Objective-C: `@"Hello World!"` . En C, escribirías esto como `"Hello World!"` Sin embargo, Foundation Framework de Apple agrega la clase `NSString` , que proporciona una gran cantidad de funcionalidades útiles, y es utilizada por NSLog. La forma más sencilla de crear una instancia de `NSString` es la siguiente: `@" string content here "` .

> Técnicamente, NSLog () es parte de Foundation Framework de Apple y en realidad no es parte del lenguaje Objective-C. Sin embargo, Foundation Framework está presente en toda la programación de Objective-C. Dado que Foundation Framework no es de código abierto y no se puede usar fuera del desarrollo de Apple, existen alternativas de código abierto al marco que están asociadas con OPENStep y GNUStep .

Compilando el programa

Suponiendo que deseamos compilar nuestro programa Hello World, que consiste en un solo archivo `hello.m` , el comando para compilar el ejecutable es:

```
clang -framework Foundation hello.m -o hello
```

Entonces puedes ejecutarlo:

```
./hello
```

Esto dará como resultado:

```
Hello World!
```

Las opciones son:

- `-framework` : especifica un marco para usar para compilar el programa. Como este programa utiliza Foundation, incluimos el marco Foundation.
- `-o` : Esta opción indica a qué archivo nos gustaría enviar nuestro programa. En nuestro caso `hello` . Si no se especifica, el valor predeterminado es `a.out` .

Lea Comenzando con el lenguaje Objective-C en línea: https://riptutorial.com/es/objective-c/topic/199/comenzando-con-el-lenguaje-objective-c

Capítulo 2: Análisis XML

Examples

Análisis XML

```
<?xml version="1.0" encoding="UTF-8"?>
<GeocodeResponse>
 <status>OK</status>
 <result>
  <type>premise</type>
  <formatted_address>4201 Oak Lawn Ave, Dallas, TX 75219, USA</
      formatted_address>
  <address_component>
   <long_name>4201</long_name>
   <short_name>4201</short_name>
   <type>street_number</type>
  </address_component>
  <address_component>
   <long_name>Oak Lawn Avenue</long_name>
   <short_name>Oak Lawn Ave</short_name>
   <type>route</type>
  </address_component>
  <address_component>
   <long_name>Oak Lawn</long_name>
   <short_name>Oak Lawn</short_name>
   <type>neighborhood</type>
   <type>political</type>
  </address_component>
  <address_component>
   <long_name>Dallas</long_name>
   <short_name>Dallas</short_name>
   <type>locality</type>
   <type>political</type>
  </address_component>
 </result>
</GeocodeResponse>
```

Analizaremos los datos de etiqueta resaltados a través de `NSXMLParser`

Hemos declarado pocas propiedades de la siguiente manera

```
@property(nonatomic, strong)NSMutableArray *results;
@property(nonatomic, strong)NSMutableString *parsedString;
@property(nonatomic, strong)NSXMLParser *xmlParser;

//Fetch xml data
NSURLSession *session=[NSURLSession sessionWithConfiguration:[NSURLSessionConfiguration
```

```
defaultSessionConfiguration]];

NSURLSessionDataTask *task=[session dataTaskWithRequest:[NSURLRequest requestWithURL:[NSURL
URLWithString:YOUR_XMLURL]] completionHandler:^(NSData * _Nullable data, NSURLResponse *
_Nullable response, NSError * _Nullable error) {

self.xmlParser=[[NSXMLParser alloc] initWithData:data];
self.xmlParser.delegate=self;
if([self.xmlParser parse]){
   //If parsing completed successfully

        NSLog(@"%@",self.results);

}

}];

[task resume];
```

Luego definimos el `NSXMLParserDelegate`

```
- (void)parser:(NSXMLParser *)parser didStartElement:(NSString *)elementName
namespaceURI:(nullable NSString *)namespaceURI qualifiedName:(nullable NSString *)qName
attributes:(NSDictionary<NSString *, NSString *> *)attributeDict{

    if([elementName isEqualToString:@"GeocodeResponse"]){
        self.results=[[NSMutableArray alloc] init];
    }

    if([elementName isEqualToString:@"formatted_address"]){
        self.parsedString=[[NSMutableString alloc] init];
    }

}


- (void)parser:(NSXMLParser *)parser foundCharacters:(NSString *)string{

    if(self.parsedString){
        [self.parsedString appendString:[string
stringByTrimmingCharactersInSet:[NSCharacterSet whitespaceAndNewlineCharacterSet]]];
    }


}

- (void)parser:(NSXMLParser *)parser didEndElement:(NSString *)elementName
namespaceURI:(nullable NSString *)namespaceURI qualifiedName:(nullable NSString *)qName{

    if([elementName isEqualToString:@"formatted_address"]){
        [self.results addObject:self.parsedString];

        self.parsedString=nil;
    }

}
```

Lea Análisis XML en línea: https://riptutorial.com/es/objective-c/topic/8048/analisis-xml

Capítulo 3: Bloques

Sintaxis

- // Declarar como una variable local:

  returnType (^ blockName) (parametersType1, parametersType2, ...) = ^ returnType (argumento1, argumento2, ...) {...};

- // Declarar como una propiedad:

  @propiedad (nonatomic, copy, nullability) returnType (^ blockName) (parametersTypes);

- // Declarar como un parámetro del método:

  - (void) someMethodThatTakesABlock: (returnType (^ nullability) (parametersTypes)) blockName;

- // Declarar como un argumento a una llamada de método:

  [someObject someMethodThatTakesABlock: ^ returnType (parámetros) {...}];

- // Declarar como un typedef:

  typedef returnType (^ TypeName) (parametersTypes);

  TypeName blockName = ^ returnType (parámetros) {...};

- // Declarar una función en C devolver un objeto de bloque:

  BLOCK_RETURN_TYPE (^ function_name (parámetros de función)) (BLOCK_PARAMETER_TYPE);

Observaciones

Los bloques se especifican en la Especificación de lenguaje para bloques para C, Objective-C, C ++ y Objective-C ++.

Además, la ABI de bloques está definida por la Especificación de implementación de bloques .

Examples

Bloques como parámetros del método

```
- (void)methodWithBlock:(returnType (^)(paramType1, paramType2, ...))name;
```

Definiendo y Asignando

Un bloque que realiza la adición de dos números de doble precisión, asignados a la `addition` variables:

```
double (^addition)(double, double) = ^double(double first, double second){
    return first + second;
};
```

El bloque puede ser llamado posteriormente así:

```
double result = addition(1.0, 2.0); // result == 3.0
```

Bloques como propiedades

```
@interface MyObject : MySuperclass

@property (copy) void (^blockProperty)(NSString *string);

@end
```

Al asignar, dado que `self` conserva `blockProperty` , el bloque no debe contener una referencia fuerte a self. Esas referencias fuertes mutuas se denominan "ciclo de retención" y evitarán la liberación de cualquiera de los objetos.

```
__weak __typeof(self) weakSelf = self;
self.blockProperty = ^(NSString *string) {
    // refer only to weakSelf here.  self will cause a retain cycle
};
```

Es muy poco probable, pero el `self` podría ser desasignado dentro del bloque, en algún lugar durante la ejecución. En este caso `weakSelf` vuelve `nil` y todos los mensajes no tienen el efecto deseado. Esto podría dejar la aplicación en un estado desconocido. Esto puede evitarse conservando `weakSelf` con un `__strong` ivar durante la ejecución del bloque y la limpieza posterior.

```
__weak __typeof(self) weakSelf = self;
self.blockProperty = ^(NSString *string) {
    __strong __typeof(weakSelf) strongSelf = weakSelf;
    // refer only to strongSelf here.
    // ...
    // At the end of execution, clean up the reference
    strongSelf = nil;
};
```

Bloquear Typedefs

```
typedef double (^Operation)(double first, double second);
```

Si declara un tipo de bloque como typedef, puede usar el nuevo nombre de tipo en lugar de la

descripción completa de los argumentos y los valores de retorno. Esto define la `Operation` como un bloque que toma dos dobles y devuelve un doble.

El tipo se puede utilizar para el parámetro de un método:

```
- (double)doWithOperation:(Operation)operation
                    first:(double)first
                   second:(double)second;
```

o como un tipo de variable:

```
Operation addition = ^double(double first, double second){
    return first + second;
};

// Returns 3.0
[self doWithOperation:addition
                first:1.0
               second:2.0];
```

Sin el typedef, esto es mucho más desordenado:

```
- (double)doWithOperation:(double (^)(double, double))operation
                    first:(double)first
                   second:(double)second;

double (^addition)(double, double) = // ...
```

Bloques como variables locales.

```
returnType (^blockName)(parameterType1, parameterType2, ...) = ^returnType(argument1,
argument2, ...) {...};

float (^square)(float) = ^(float x) {return x*x;};

square(5); // resolves to 25
square(-7); // resolves to 49
```

Aquí hay un ejemplo sin retorno y sin parámetros:

```
NSMutableDictionary *localStatus;
void (^logStatus)() = ^(void){ [MYUniversalLogger logCurrentStatus:localStatus]};

// Insert some code to add useful status information
// to localStatus dictionary

logStatus(); // this will call the block with the current localStatus
```

Lea Bloques en línea: https://riptutorial.com/es/objective-c/topic/540/bloques

Capítulo 4: BOOL / bool / Boolean / NSCFBoolean

Examples

BOOL / Boolean / bool / NSCFBoolean

1. bool es un tipo de datos definido en C99.
2. Los valores booleanos se utilizan en condicionales, como las declaraciones if o while, para realizar condicionalmente la lógica o repetir la ejecución. Al evaluar una declaración condicional, el valor 0 se considera "falso", mientras que cualquier otro valor se considera "verdadero". Debido a que NULL y nil se definen como 0, las declaraciones condicionales sobre estos valores inexistentes también se evalúan como "falsas".
3. BOOL es un tipo Objective-C definido como char firmado con las macros YES y NO para representar verdadero y falso

De la definición en objc.h:

```
#if (TARGET_OS_IPHONE && __LP64__)  ||  TARGET_OS_WATCH
typedef bool BOOL;
#else
typedef signed char BOOL;
// BOOL is explicitly signed so @encode(BOOL) == "c" rather than "C"
// even if -funsigned-char is used.
#endif

#define YES ((BOOL)1)
#define NO  ((BOOL)0)
```

4. NSCFBoolean es una clase privada en el clúster de clase NSNumber. Es un puente al tipo CFBooleanRef, que se utiliza para envolver valores booleanos para las listas y colecciones de propiedades de Core Foundation. CFBoolean define las constantes kCFBooleanTrue y kCFBooleanFalse. Debido a que CFNumberRef y CFBooleanRef son tipos diferentes en Core Foundation, tiene sentido que estén representados por diferentes clases puente en NSNumber.

BOOL VS Boolean

BOOL

- Los frameworks Objective-C de Apple y la mayoría de los usos del código Objective-C / Cocoa
  BOOL.
- Utilice BOOL en object-C, cuando trate con cualquier API CoreFoundation

Booleano

- Boolean es una palabra clave antigua de Carbon, definida como un carácter sin firma

Lea BOOL / bool / Boolean / NSCFBoolean en línea: https://riptutorial.com/es/objective-c/topic/7267/bool---bool---boolean---nscfboolean

Capítulo 5: Categorías

Sintaxis

- @interface ClassName (categoryName) // ClassName es la clase que se extenderá
- // Declaraciones de método y propiedad
- @fin

Observaciones

Para evitar choques de nombres de métodos, se recomienda usar prefijos (como `xyz_` en el ejemplo). Si existen métodos con el mismo nombre, no está definido cuál se usará en el tiempo de ejecución.

Examples

Categoría simple

Interfaz e implementación de una categoría simple en NSArray, llamada Filter, con un solo método que filtra los números.

Es una buena práctica agregar un prefijo (`PF`) al método para garantizar que no sobrescribamos ningún método futuro de `NSArray` .

```
@interface NSArray (PFFilter)

- (NSArray *)pf_filterSmaller:(double)number;

@end

@implementation NSArray (PFFilter)

- (NSArray *)pf_filterSmaller:(double)number
{
    NSMutableArray *result = [NSMutableArray array];
    for (id val in self)
    {
        if ([val isKindOfClass:[NSNumber class] && [val doubleValue] >= number)
        {
            [result addObject:val];
        }
    }
    return [result copy];
}

@end
```

Declarar un método de clase

Archivo de encabezado `UIColor+XYZPalette.h` :

```
@interface UIColor (XYZPalette)

+(UIColor *)xyz_indigoColor;

@end
```

y la implementación de `UIColor+XYZPalette.m` :

```
@implementation UIColor (XYZPalette)

+(UIColor *)xyz_indigoColor
{
    return [UIColor colorWithRed:75/255.0f green:0/255.0f blue:130/255.0f alpha:1.0f];
}

@end
```

Añadiendo una propiedad con una categoría

Las propiedades se pueden agregar con categorías usando objetos asociados, una característica del tiempo de ejecución de Objective-C.

Tenga en cuenta que la declaración de propiedad de `retain, nonatomic` coincide con el último argumento con `objc_setAssociatedObject` . Consulte Adjuntar objeto a otro objeto existente para obtener explicaciones.

```
#import <objc/runtime.h>

@interface UIViewController (ScreenName)

@property (retain, nonatomic) NSString *screenName;

@end

@implementation UIViewController (ScreenName)

@dynamic screenName;

- (NSString *)screenName {
    return objc_getAssociatedObject(self, @selector(screenName));
}

- (void)setScreenName:(NSString *)screenName {
    objc_setAssociatedObject(self, @selector(screenName), screenName,
OBJC_ASSOCIATION_RETAIN_NONATOMIC);
}

@end
```

De conformidad con el protocolo.

Puede agregar protocolos a las clases estándar para ampliar su funcionalidad:

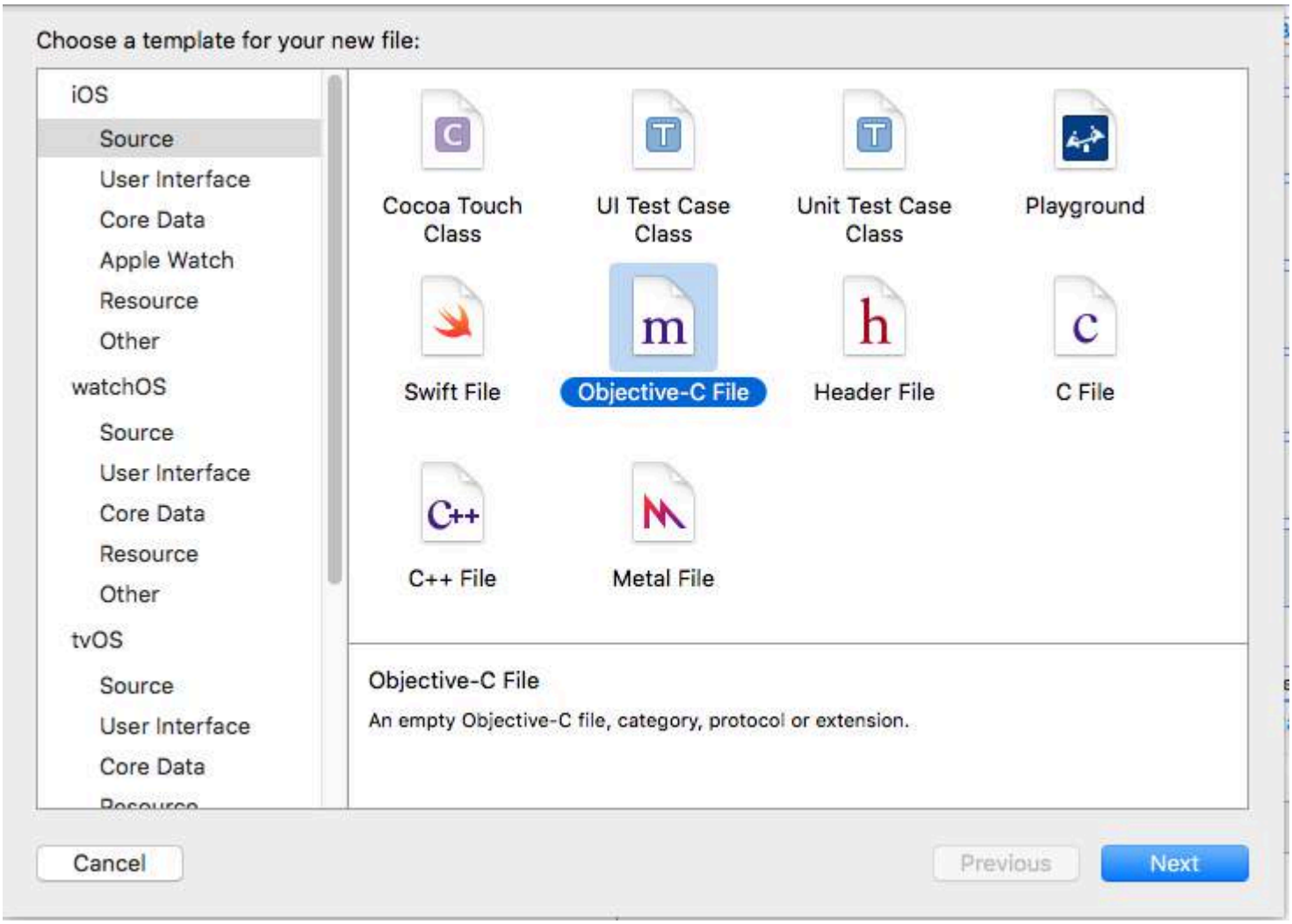
```
@protocol EncodableToString <NSObject>
- (NSString *)toString;
@end

@interface NSDictionary (XYZExtended) <EncodableToString>
@end

@implementation NSDictionary (XYZExtended)
- (NSString *)toString {
    return self.description;
}
@end
```

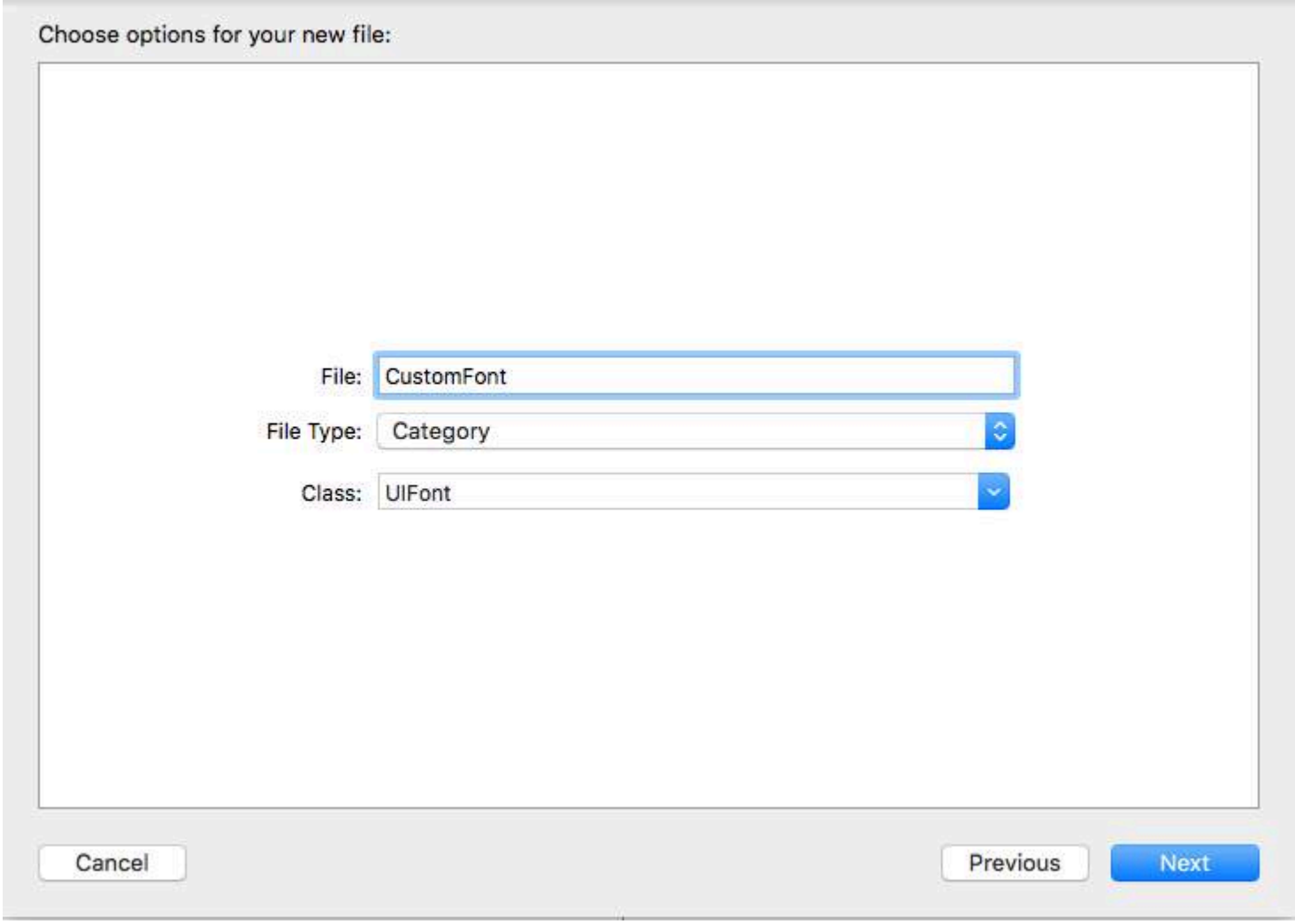
donde `XYZ` prefijo tu proyecto

Crear una categoría en XCode

Las categorías proporcionan la capacidad de agregar alguna funcionalidad adicional a un objeto sin subclasificar o cambiar el objeto real.

Por ejemplo, queremos establecer algunas fuentes personalizadas. Vamos a crear una categoría que agregue funcionalidad a la clase `UIFont` . Abra su proyecto XCode, haga clic en `File` -> `New` -> `File` y elija `File Objective-C file` , haga clic en Siguiente, ingrese el nombre de su categoría, diga "CustomFont" elija el tipo de archivo como Categoría y Clase como UIFont, luego haga clic en "Siguiente" seguido de "Crear". "


Choose a template for your new file:
iOS
Source
User Interface
Core Data
Apple Watch
Resource
Other
watchOS
Source
User Interface
Core Data
Resource
Other
tvOS
Source
User Interface
Core Data
Cocoa Touch Class
UI Test Case Class
Unit Test Case Class
Playground
Swift File
Objective-C File
Header File
C File
C++ File
Metal File
Objective-C File
An empty Objective-C file, category, protocol or extension.
Cancel
Previous
Next



Declare el método de la categoría: -

Haga clic en "UIFont + CustomFonts.h" para ver el archivo de encabezado de la nueva categoría. Agregue el siguiente código a la interfaz para declarar el método.

```
@interface UIFont (CustomFonts)

+(UIFont *)productSansRegularFontWithSize:(CGFloat)size;

@end
```

Ahora implemente el método de la categoría: -

Haga clic en "UIFont + CustomFonts.m" para ver el archivo de implementación de la categoría. Agregue el siguiente código para crear un método que establezca la fuente ProductSansRegular.

```
+(UIFont *)productSansRegularFontWithSize:(CGFloat)size{

    return [UIFont fontWithName:@"ProductSans-Regular" size:size];

}
```

Importa tu categoría

```
#import "UIFont+CustomFonts.h"
```

Ahora establece la fuente de la etiqueta

```
[self.label setFont:[UIFont productSansRegularFontWithSize:16.0]];
```

Lea Categorías en línea: https://riptutorial.com/es/objective-c/topic/550/categorias

Capítulo 6: Clases y objetos

Sintaxis

- Cat * cat = [[Cat alloc] init]; // Crear el objeto cat del tipo Cat
- Dog * dog = [[alloc del perro] init]; // Crear un objeto de tipo perro.
- NSObject * someObject = [NSObject alloc]; [someObject init]; // **no hagas esto**
- XYZObject * object = [XYZObject new]; // Use new para crear objetos si NO se necesitan argumentos para la inicialización
- NSString * someString = @ "¡Hola mundo!"; // Creando una NSString con *sintaxis literal*
- NSNumber * myFloat = @ 3.14f; // Otro ejemplo para crear un NSNumber usando sintaxis literal
- NSNumber * myInt = @ (84/2); // Crear un objeto usando una expresión en caja

Examples

Creación de clases con valores de inicialización.

```
#import <Foundation/Foundation.h>
@interface Car:NSObject {
    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{
    CarMotorCode = motorCode;
    CarChassisCode = chassisCode;
    return self;
}

- (void) startCar {...}
- (void) stopCar {...}

@end
```

El método `initWithMotorValue: type andChassisValue: type` se utilizará para inicializar los objetos Car.

Clase Singleton

¿Qué es una Clase Singleton?

Una clase de singleton devuelve la misma instancia sin importar cuántas veces la solicite una aplicación. A diferencia de una clase regular, un objeto singleton proporciona un punto global de acceso a los recursos de su clase.

¿Cuándo usar las clases de Singleton?

Los Singletons se usan en situaciones donde este único punto de control es deseable, como en las clases que ofrecen algún servicio o recurso general.

Cómo crear clases singleton

Primero, crea un archivo Nuevo y `NSObject` de `NSObject` . Nombra algo, usaremos `CommonClass` aquí. Xcode ahora generará los archivos CommonClass.h y CommonClass.m para usted.

En su archivo `CommonClass.h` :

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

@interface CommonClass : NSObject {
}
+ (CommonClass *)sharedObject;
@property NSString *commonString;
@end
```

En su archivo `CommonClass.m` :

```
#import "CommonClass.h"

@implementation CommonClass

+ (CommonClass *)sharedObject {
    static CommonClass *sharedClass = nil;
    static dispatch_once_t onceToken;
    dispatch_once(&onceToken, ^{
        sharedClass = [[self alloc] init];
    });
    return sharedClass;
}

- (id)init {
    if (self = [super init]) {
        self.commonString = @"this is string";
    }
    return self;
}

@end
```

Cómo usar las clases de Singleton

Podrá acceder a la Clase Singleton que creamos anteriormente desde cualquier parte del proyecto siempre que haya importado el archivo `CommonClass.h` en el módulo correspondiente.

Para modificar y acceder a los datos compartidos en Singleton Class, tendrá que acceder al Objeto compartido de esa clase, al que se puede acceder mediante el método `sharedObject` , como se `sharedObject` a continuación:

```
[CommonClass sharedObject]
```

Para leer o modificar los elementos en Clase compartida, haga lo siguiente:

```
NSString *commonString = [[CommonClass sharedObject].commonString; //Read the string in
singleton class

NSString *newString = @"New String";
[CommonClass sharedObject].commonString = newString;//Modified the string in singleton class
```

El tipo de retorno "instancetype"

Objective-C admite un tipo especial llamado `instancetype que solo se puede usar como tipo devuelto por un método. Se evalúa a la clase del objeto receptor.

Considere la siguiente jerarquía de clases:

```
@interface Foo : NSObject

- (instancetype)initWithString:(NSString *)string;

@end

@interface Bar : Foo
@end
```

Cuando se `[[Foo alloc] initWithString:@"abc"]` , el compilador puede inferir que el tipo de retorno es `Foo *` . La clase `Bar` derivó de `Foo` pero no anuló la declaración del inicializador. Sin embargo, gracias al tipo de `instancetype` , el compilador puede inferir que `[[Bar alloc] initWithString:@"xyz"]` devuelve un valor de tipo `Bar *` .

Considere el tipo de retorno de `-[Foo initWithString:]` siendo `Foo *` lugar: si llamara a `[[Bar alloc] initWithString:]` , el compilador `[[Bar alloc] initWithString:]` que se devuelve un `Foo *` , no un `Bar *` como es la intención de desarrollador. El tipo de `instancetype` resolvió este problema.

Antes de la introducción de `instancetype` , inicializadores, métodos estáticos como accesores singleton y otros métodos que desean devolver una instancia de la clase receptora necesaria para devolver un `id` . El problema es que `id` significa *"un objeto de cualquier tipo"* . Por lo tanto, el compilador no puede detectar que `NSString *wrong = [[Foo alloc] initWithString:@"abc"];` está asignando a una variable con un tipo incorrecto.

Debido a este problema, los **inicializadores siempre deben usar `instancetype` lugar de `id`** como valor de retorno.

Especificando genéricos

Puede mejorar sus propias clases con *genéricos* como `NSArray` o `NSDictionary` .

```
@interface MyClass<__covariant T>

@property (nonnull, nonatomic, strong, readonly) NSArray<T>* allObjects;

- (void) addObject:(nonnull T)obj;

@end
```

Diferencia entre asignación e inicialización.

En la mayoría de los lenguajes orientados a objetos, asignar memoria a un objeto e inicializarlo es una operación atómica:

```
// Both allocates memory and calls the constructor
MyClass object = new MyClass();
```

En Objective-C, estas son operaciones separadas. Los métodos de clase `alloc` (y su histórico hermano `allocWithZone:` hacen que el tiempo de ejecución de Objective-C reserve la memoria requerida y la borre. Excepto por unos pocos valores internos, todas las propiedades y variables se establecen en 0 / `NO` / `nil` .

El objeto entonces ya es "válido", pero siempre queremos llamar a un método para configurar realmente el objeto, que llamamos un *inicializador* . Estos tienen el mismo propósito que los *constructores* en otros idiomas. Por convención, estos métodos comienzan con `init` . Desde el punto de vista del lenguaje, son solo métodos normales.

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

// Shorthand:
object = [[MyClass alloc] init];
```

Lea Clases y objetos en línea: https://riptutorial.com/es/objective-c/topic/758/clases-y-objetos

Capítulo 7: Codificación de valor clave / Observación de valor clave

Examples

Ejemplo más común de codificación de valor clave de la vida real

La codificación de valor clave se integra en **NSObject** mediante el protocolo **NSKeyValueCoding** .

¿Lo que esto significa?

Significa que cualquier objeto de identificación es capaz de llamar al método valueForKey y sus diversas variantes como valueForKeyPath, etc.

También significa que cualquier objeto id puede invocar el método setValue y sus diversas variantes también.

Ejemplo:

```
id obj = [[MyClass alloc] init];
id value = [obj valueForKey:@"myNumber"];

int myNumberAsInt = [value intValue];
myNumberAsInt = 53;
[obj setValue:@(myNumberAsInt) forKey:@"myNumber"];
```

Excepciones:

El ejemplo anterior supone que MyClass tiene una propiedad NSNumber llamada myNumber. Si myNumber no aparece en la definición de la interfaz de MyClass, puede generarse una NSUndefinedKeyException en posiblemente las líneas 2 y 5, conocida popularmente como:

```
this class is not key value coding-compliant for the key myNumber.
```

Por qué esto es TAN poderoso:

Puede escribir código que pueda acceder a las propiedades de una clase dinámicamente, sin necesidad de interfaz para esa clase. Esto significa que una vista de tabla puede mostrar valores de cualquier propiedad de un objeto derivado de NSObject, siempre que sus nombres de propiedad se proporcionen dinámicamente en tiempo de ejecución.

En el ejemplo anterior, el código también puede funcionar sin que MyClass esté disponible y el tipo de identificación obj esté disponible para el código de llamada.

Valor clave observando

Configuración de la observación del valor clave.

En este caso, queremos observar el `contentOffset` en un objeto que posee nuestro observador

```
//
// Class to observe
//
@interface XYZScrollView: NSObject
@property (nonatomic, assign) CGPoint contentOffset;
@end

@implementation XYZScrollView
@end


//
// Class that will observe changes
//
@interface XYZObserver: NSObject
@property (nonatomic, strong) XYZScrollView *scrollView;
@end

@implementation XYZObserver

// simple way to create a KVO context
static void *XYZObserverContext = &XYZObserverContext;


// Helper method to add self as an observer to
// the scrollView's contentOffset property
- (void)addObserver {

    // NSKeyValueObservingOptions
    //
    // - NSKeyValueObservingOptionNew
    // - NSKeyValueObservingOptionOld
    // - NSKeyValueObservingOptionInitial
    // - NSKeyValueObservingOptionPrior
    //
    // can be combined:
    // (NSKeyValueObservingOptionNew | NSKeyValueObservingOptionOld)

    NSString *keyPath = NSStringFromSelector(@selector(contentOffset));
    NSKeyValueObservingOptions options = NSKeyValueObservingOptionNew;

    [self.scrollView addObserver: self
                      forKeyPath: keyPath
                         options: options
                         context: XYZObserverContext];
}

- (void)observeValueForKeyPath:(NSString *)keyPath ofObject:(id)object
change:(NSDictionary<NSString *,id> *)change context:(void *)context {

    if (context == XYZObserverContext) { // check the context

        // check the keyPath to see if it's any of the desired keyPath's.
        // You can observe multiple keyPath's
        if ([keyPath isEqualToString: NSStringFromSelector(@selector(contentOffset))]) {
```

```
            // change dictionary keys:
            // - NSKeyValueChangeKindKey
            // - NSKeyValueChangeNewKey
            // - NSKeyValueChangeOldKey
            // - NSKeyValueChangeIndexesKey
            // - NSKeyValueChangeNotificationIsPriorKey

            // the change dictionary here for a CGPoint observation will
            // return an NSPoint, so we can take the CGPointValue of it.
            CGPoint point = [change[NSKeyValueChangeNewKey] CGPointValue];

            // handle point
        }

    } else {

        // if the context doesn't match our current object's context
        // we want to pass the observation parameters to super
        [super observeValueForKeyPath: keyPath
                             ofObject: object
                               change: change
                              context: context];
    }
}

// The program can crash if an object is not removed as observer
// before it is dealloc'd
//
// Helper method to remove self as an observer of the scrollView's
// contentOffset property
- (void)removeObserver {
    NSString *keyPath = NSStringFromSelector(@selector(contentOffset));
    [self.scrollView removeObserver: self forKeyPath: keyPath];
}

@end
```

Consulta de datos KVC

```
if ([[dataObject objectForKey:@"yourVariable"] isEqualToString:"Hello World"]) {
    return YES;
} else {
    return NO;
}
```

Puede consultar los valores almacenados mediante KVC de forma rápida y sencilla, sin necesidad de recuperarlos o convertirlos en variables locales.

Operadores de cobro

Los operadores de colección se pueden usar en una ruta de acceso de clave KVC para realizar una operación en una propiedad de "tipo de colección" (es decir, `NSArray` , `NSSet` y similares). Por ejemplo, una operación común a realizar es contar los objetos en una colección. Para lograr esto, utiliza el *operador de colección* `@count` :

```
self.array = @[@5, @4, @3, @2, @1];
NSNumber *count = [self.array valueForKeyPath:@"@count"];
NSNumber *countAlt = [self valueForKeyPath:@"array.@count"];
// count == countAlt == 5
```

Si bien esto es completamente redundante aquí (podríamos haber accedido a la propiedad de `count`), en ocasiones *puede* ser útil, aunque rara vez es necesario. Hay, sin embargo, algunos operadores de recolección que son mucho más útiles, a saber `@max` , `@min` , `@sum` , `@avg` y la `@unionOf` familia. Es importante tener en cuenta que estos operadores *también* requieren una ruta de acceso por separado que *siga* al operador para funcionar correctamente. Aquí hay una lista de ellos y el tipo de datos con los que trabajan:

Operador	Tipo de datos
`@count`	(ninguna)
`@max`	`NSNumber` , `NSDate` , `int` (y relacionado), etc.
`@min`	`NSNumber` , `NSDate` , `int` (y relacionado), etc.
`@sum`	`NSNumber` , `int` (y relacionado), `double` (y relacionado), etc.
`@avg`	`NSNumber` , `int` (y relacionado), `double` (y relacionado), etc.
`@unionOfObjects`	`NSArray` , `NSSet` , etc.
`@distinctUnionOfObjects`	`NSArray` , `NSSet` , etc.
`@unionOfArrays`	`NSArray<NSArray*>`
`@distinctUnionOfArrays`	`NSArray<NSArray*>`
`@distinctUnionOfSets`	`NSSet<NSSet*>`

`@max` y `@min` devolverán el valor más alto o más bajo, respectivamente, de una propiedad de objetos en la colección. Por ejemplo, mira el siguiente código:

```
// “Point” class used in our collection
@interface Point : NSObject

@property NSInteger x, y;

+ (instancetype)pointWithX:(NSInteger)x y:(NSInteger)y;

@end

...

self.points = @[[Point pointWithX:0 y:0],
                [Point pointWithX:1 y:-1],
                [Point pointWithX:5 y:-6],
                [Point pointWithX:3 y:0],
                [Point pointWithX:8 y:-4],
```

```
];

NSNumber *maxX = [self valueForKeyPath:@"points.@max.x"];
NSNumber *minX = [self valueForKeyPath:@"points.@min.x"];
NSNumber *maxY = [self valueForKeyPath:@"points.@max.y"];
NSNumber *minY = [self valueForKeyPath:@"points.@min.y"];

NSArray<NSNumber*> *boundsOfAllPoints = @[maxX, minX, maxY, minY];

...
```

En solo 4 líneas de código y Foundation puro, con el poder de los operadores de recopilación de Key-Value Coding, pudimos extraer un rectángulo que encapsula todos los puntos de nuestra matriz.

Es importante tener en cuenta que estas comparaciones se realizan invocando el método `compare:` en los objetos, por lo que si alguna vez desea que su propia clase sea compatible con estos operadores, debe implementar este método.

`@sum` , como probablemente pueda adivinar, sumará todos los valores de una propiedad.

```
@interface Expense : NSObject

@property NSNumber *price;

+ (instancetype)expenseWithPrice:(NSNumber *)price;

@end

...

self.expenses = @[[Expense expenseWithPrice:@1.50],
                  [Expense expenseWithPrice:@9.99],
                  [Expense expenseWithPrice:@2.78],
                  [Expense expenseWithPrice:@9.99],
                  [Expense expenseWithPrice:@24.95]
];

NSNumber *totalExpenses = [self valueForKeyPath:@"expenses.@sum.price"];
```

Aquí, usamos `@sum` para encontrar el precio total de todos los gastos en la matriz. Si en su lugar quisiéramos encontrar el precio promedio que estamos pagando por cada gasto, podemos usar `@avg` :

```
NSNumber *averagePrice = [self valueForKeyPath:@"expenses.@avg.price"];
```

Finalmente, está la familia `@unionOf` . Hay cinco operadores diferentes en esta familia, pero todos funcionan casi igual, con solo pequeñas diferencias entre cada uno. Primero, hay `@unionOfObjects` que devolverá una matriz de las propiedades de los objetos en una matriz:

```
// See "expenses" array above

NSArray<NSNumber*> *allPrices = [self valueForKeyPath:
    @"expenses.@unionOfObjects.price"];
```

```
// Equal to @[ @1.50, @9.99, @2.78, @9.99, @24.95 ]
```

`@distinctUnionOfObjects` funciona igual que `@unionOfObjects` , pero elimina duplicados:

```
NSArray<NSNumber*> *differentPrices = [self valueForKeyPath:
    @"expenses.@distinctUnionOfObjects.price"];

// Equal to @[ @1.50, @9.99, @2.78, @24.95 ]
```

Y finalmente, los últimos 3 operadores de la familia `@unionOf` irán un paso más allá y devolverán una serie de valores encontrados para una propiedad contenida dentro de matrices anidadas:

```
NSArray<NSArray<Expense*,Expense*>*> *arrayOfArrays =
    @[
        @[ [Expense expenseWithPrice:@19.99],
           [Expense expenseWithPrice:@14.95],
           [Expense expenseWithPrice:@4.50],
           [Expense expenseWithPrice:@19.99]
         ],

        @[ [Expense expenseWithPrice:@3.75],
           [Expense expenseWithPrice:@14.95]
         ]
     ];

// @unionOfArrays
NSArray<NSNumber*> allPrices = [arrayOfArrays valueForKeyPath:
    @"@unionOfArrays.price"];
// Equal to @[ @19.99, @14.95, @4.50, @19.99, @3.75, @14.95 ];

// @distinctUnionOfArrays
NSArray<NSNumber*> allPrices = [arrayOfArrays valueForKeyPath:
    @"@distinctUnionOfArrays.price"];
// Equal to @[ @19.99, @14.95, @4.50, @3.75 ];
```

El que falta en este ejemplo es `@distinctUnionOfSets` , sin embargo, esto funciona exactamente igual que `@distinctUnionOfArrays` , pero funciona con `NSSet` s (no hay `distinct` versión que no sea `distinct` porque en un conjunto, cada objeto debe ser distinto de todos modos).

¡Y eso es! Los operadores de recolección pueden ser realmente poderosos si se usan correctamente, y pueden ayudar a evitar tener que recorrer cosas innecesariamente.

Una última nota: también puede usar los operadores de recopilación estándar en matrices de `NSNumber` s (sin acceso adicional a la propiedad). Para hacer esto, accede a la `self` pseudopropiedad que simplemente devuelve el objeto:

```
NSArray<NSNumber*> *numbers = @[@0, @1, @5, @27, @1337, @2048];

NSNumber *largest = [numbers valueForKeyPath:@"@max.self"];
NSNumber *smallest = [numbers valueForKeyPath:@"@min.self"];
NSNumber *total = [numbers valueForKeyPath:@"@sum.self"];
NSNumber *average = [numbers valueForKeyPath:@"@avg.self"];
```

Lea Codificación de valor clave / Observación de valor clave en línea: https://riptutorial.com/es/objective-c/topic/556/codificacion-de-valor-clave---observacion-de-valor-clave

Capítulo 8: Continuar y romper!

Examples

Continuar y romper declaración

La instrucción continue en el lenguaje de programación de Objective-C funciona de manera similar a la instrucción break. Sin embargo, en lugar de forzar la terminación, continuar fuerza la siguiente iteración del bucle, omitiendo cualquier código intermedio.

Para el bucle for, la instrucción continue hace que la prueba condicional y las porciones de incremento del bucle se ejecuten. Por el momento y haz ... while, la instrucción continue hace que el control del programa pase a las pruebas condicionales.

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

int main ()
{
   /* local variable definition */
   int a = 10;

   /* do loop execution */
   do
   {
      if( a == 15)
      {
         /* skip the iteration */
         a = a + 1;
         continue;
      }
      NSLog(@"value of a: %d\n", a);
      a++;

   }while( a < 20 );

   return 0;
}
```

Salida:

```
2013-09-07 22:20:35.647 demo[29998] value of a: 10
2013-09-07 22:20:35.647 demo[29998] value of a: 11
2013-09-07 22:20:35.647 demo[29998] value of a: 12
2013-09-07 22:20:35.647 demo[29998] value of a: 13
2013-09-07 22:20:35.647 demo[29998] value of a: 14
2013-09-07 22:20:35.647 demo[29998] value of a: 16
2013-09-07 22:20:35.647 demo[29998] value of a: 17
2013-09-07 22:20:35.647 demo[29998] value of a: 18
2013-09-07 22:20:35.647 demo[29998] value of a: 19
```

Consulte este enlace para más información.

Lea Continuar y romper! en línea: https://riptutorial.com/es/objective-c/topic/8709/continuar-y-romper-

Capítulo 9: Declare el método de clase y el método de instancia

Introducción

Los métodos de instancia son métodos que son específicos de clases particulares. Los métodos de instancia se declaran y definen seguidos de un símbolo - (menos).

Los métodos de clase pueden llamarse por el mismo nombre de clase. Los métodos de clase se declaran y definen mediante el signo + (más).

Sintaxis

1. - (void) testInstanceMethod; // Los métodos de clase declaran con el signo "+"
2. (void) classMethod; // los métodos de instancia declaran con el signo "-"

Examples

Cómo declarar método de clase y método de instancia.

Los métodos de instancia utilizan una instancia de una clase.

```
@interface MyTestClass : NSObject

- (void)testInstanceMethod;

@end
```

Entonces podrían ser utilizados como tal:

```
MyTestClass *object = [[MyTestClass alloc] init];
[object testInstanceMethod];
```

El método de clase se puede usar solo con el nombre de la clase.

```
@interface MyClass : NSObject

+ (void)aClassMethod;

@end
```

Entonces podrían ser utilizados como tal:

```
[MyClass aClassMethod];
```

los métodos de clase son los métodos de conveniencia en muchas clases de Foundation como [NSString's + stringWithFormat:] o NSArray's + arrayWithArray

Lea Declare el método de clase y el método de instancia en línea: https://riptutorial.com/es/objective-c/topic/8214/declare-el-metodo-de-clase-y-el-metodo-de-instancia

Capítulo 10: Entero aleatorio

Examples

Entero aleatorio basico

La función `arc4random_uniform()` es la forma más sencilla de obtener enteros aleatorios de alta calidad. Según el manual:

> arc4random_uniform (upper_bound) devolverá un número aleatorio distribuido uniformemente menos que upper_bound.
>
> Se recomienda arc4random_uniform () sobre construcciones como " arc4random ()% upper_bound ", ya que evita el "sesgo de módulo" cuando el límite superior no es una potencia de dos.

```
uint32_t randomInteger = arc4random_uniform(5); // A random integer between 0 and 4
```

Entero aleatorio dentro de un rango

El siguiente código demuestra el uso de `arc4random_uniform()` para generar un entero aleatorio entre 3 y 12:

```
uint32_t randomIntegerWithinRange = arc4random_uniform(10) + 3; // A random integer between 3
and 12
```

Esto funciona para crear un rango porque `arc4random_uniform(10)` devuelve un número entero entre 0 y 9. Agregar 3 a este número entero aleatorio produce un rango entre `0 + 3` y `9 + 3` .

Lea Entero aleatorio en línea: https://riptutorial.com/es/objective-c/topic/1573/entero-aleatorio

Capítulo 11: Entorno de tiempo de ejecución de bajo nivel

Observaciones

Para utilizar el tiempo de ejecución de Objective-C, debe importarlo.

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

Examples

Adjuntar objeto a otro objeto existente (asociación)

Es posible adjuntar un objeto a un objeto existente como si hubiera una nueva propiedad. Esto se denomina *asociación* y permite extender objetos existentes. Puede usarse para proporcionar almacenamiento cuando se agrega una propiedad a través de una extensión de clase o, de lo contrario, se agrega información adicional a un objeto existente.

El tiempo de ejecución libera automáticamente el objeto asociado una vez que se desasigna el objeto de destino.

```
#import <objc/runtime.h>

// "Key" for association. Its value is never used and doesn't
// matter. The only purpose of this global static variable is to
// provide a guaranteed unique value at runtime: no two distinct
// global variables can share the same address.
static char key;

id target = ...;
id payload = ...;
objc_setAssociateObject(target, &key, payload, OBJC_ASSOCIATION_RETAIN);
// Other useful values are OBJC_ASSOCIATION_COPY
// and OBJ_ASSOCIATION_ASSIGN

id queryPayload = objc_getAssociatedObject(target, &key);
```

Métodos de aumento utilizando el método Swizzling

El tiempo de ejecución de Objective-C le permite cambiar la implementación de un método en tiempo de ejecución. Esto se llama *método swizzling* y se usa a menudo para intercambiar las implementaciones de dos métodos. Por ejemplo, si se intercambian los métodos `foo` y `bar` , el envío del mensaje `foo` ejecutará la implementación de `bar` y viceversa.

Esta técnica se puede utilizar para aumentar o "parchar" los métodos existentes que no puede editar directamente, como los métodos de las clases proporcionadas por el sistema.

En el siguiente ejemplo, el método `-[NSUserDefaults synchronize]` se aumenta para imprimir el tiempo de ejecución de la implementación original.

IMPORTANTE: muchas personas intentan hacer swizzling usando `method_exchangeImplementations`. Sin embargo, este enfoque es peligroso si necesita llamar al método que está reemplazando, porque lo estará usando con un selector diferente del que espera recibir. Como resultado, su código puede dividirse de formas extrañas e inesperadas, especialmente si varias partes dominan un objeto de esta manera. En su lugar, siempre debe hacer swizzling usando `setImplementation` junto con una función C, permitiéndole llamar al método con el selector original.

```
#import "NSUserDefaults+Timing.h"
#import <objc/runtime.h> // Needed for method swizzling

static IMP old_synchronize = NULL;

static void new_synchronize(id self, SEL _cmd);

@implementation NSUserDefaults(Timing)

+ (void)load
{
    Method originalMethod = class_getInstanceMethod([self class], @selector(synchronize:));
    IMP swizzleImp = (IMP)new_synchronize;
    old_synchronize = method_setImplementation(originalMethod, swizzleImp);
}
@end

static void new_synchronize(id self, SEL _cmd);
{
    NSDate *started;
    BOOL returnValue;

    started = [NSDate date];

    // Call the original implementation, passing the same parameters
    // that this function was called with, including the selector.
    returnValue = old_synchronize(self, _cmd);


    NSLog(@"Writing user defaults took %f seconds.", [[NSDate date]
timeIntervalSinceDate:started]);

    return returnValue;
}

@end
```

Si necesita cambiar un método que toma parámetros, simplemente los agrega como parámetros adicionales a la función. Por ejemplo:

```
static IMP old_viewWillAppear_animated = NULL;
static void new_viewWillAppear_animated(id self, SEL _cmd, BOOL animated);

...

Method originalMethod = class_getClassMethod([UIViewController class],
@selector(viewWillAppear:));
```

```
IMP swizzleImp = (IMP)new_viewWillAppear_animated;
old_viewWillAppear_animated = method_setImplementation(originalMethod, swizzleImp);

...

static void new_viewWillAppear_animated(id self, SEL _cmd, BOOL animated)
{
    ...

    old_viewWillAppear_animated(self, _cmd, animated);

    ...
}
```

Métodos de llamada directamente

Si necesita llamar a un método Objective-C desde el código C, tiene dos formas: usar `objc_msgSend` u obtener el `IMP` (puntero de la función de implementación del método) y llamar a eso.

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

@implementation Example

- (double)negate:(double)value {
    return -value;
}

- (double)invert:(double)value {
    return 1 / value;
}

@end

// Calls the selector on the object. Expects the method to have one double argument and return
a double.
double performSelectorWithMsgSend(id object, SEL selector, double value) {
    // We declare pointer to function and cast `objc_msgSend` to expected signature.
    // WARNING: This step is important! Otherwise you may get unexpected results!
    double (*msgSend)(id, SEL, double) = (typeof(msgSend)) &objc_msgSend;

    // The implicit arguments of self and _cmd need to be passed in addition to any explicit
arguments.
    return msgSend(object, selector, value);
}

// Does the same as the above function, but by obtaining the method's IMP.
double performSelectorWithIMP(id object, SEL selector, double value) {
    // Get the method's implementation.
    IMP imp = class_getMethodImplementation([self class], selector);

    // Cast it so the types are known and ARC can work correctly.
    double (*callableImp)(id, SEL, double) = (typeof(callableImp)) imp;

    // Again, you need the explicit arguments.
    return callableImp(object, selector, value);
}

int main() {
    Example *e = [Example new];
```

```
    // Invoke negation, result is -4
    double x = performSelectorWithMsgSend(e, @selector(negate:), 4);

    // Invoke inversion, result is 0.25
    double y = performSelectorWithIMP(e, @selector(invert:), 4);
}
```

`objc_msgSend` trabaja obteniendo el IMP para el método y llamando a eso. Los `IMP` para los últimos métodos llamados se almacenan en caché, por lo que si está enviando un mensaje de Objective-C en un bucle muy cerrado, puede obtener un rendimiento aceptable. En algunos casos, el almacenamiento en caché manual del IMP puede ofrecer un rendimiento ligeramente mejor, aunque esta es una optimización de último recurso.

Lea Entorno de tiempo de ejecución de bajo nivel en línea: https://riptutorial.com/es/objective-c/topic/1180/entorno-de-tiempo-de-ejecucion-de-bajo-nivel

Capítulo 12: Enumeración rápida

Examples

Enumeración rápida de un NSArray

Este ejemplo muestra cómo usar la enumeración rápida para atravesar una NSArray.

Cuando tienes una matriz, como

```
NSArray *collection = @[@"fast", @"enumeration", @"in objc"];
```

Puede usar `for ... in` sintaxis para recorrer cada elemento de la matriz, comenzando automáticamente con el primero en el índice `0` y deteniéndose con el último elemento:

```
for (NSString *item in collection) {
    NSLog(@"item: %@", item);
}
```

En este ejemplo, la salida generada se vería como

```
// item: fast
// item: enumeration
// item: in objc
```

Enumeración rápida de una NSArray con índice.

Este ejemplo muestra cómo usar la enumeración rápida para atravesar una NSArray. De esta manera, también puede realizar un seguimiento del índice del objeto actual mientras se atraviesa.

Supongamos que tienes una matriz,

```
NSArray *weekDays = @[@"Monday", @"Tuesday", @"Wednesday", @"Thursday", @"Friday",
@"Saturday", @"Sunday"];
```

Ahora puedes atravesar la matriz como abajo,

```
[weekDays enumerateObjectsUsingBlock:^(id obj, NSUInteger idx, BOOL *stop) {

    //... Do your usual stuff here

    obj  // This is the current object
    idx  // This is the index of the current object
    stop // Set this to true if you want to stop

}];
```

Lea Enumeración rápida en línea: https://riptutorial.com/es/objective-c/topic/5583/enumeracion-

rapida

Capítulo 13: Enums

Sintaxis

- typedef NS_ENUM (type, name) {...} - *type* es el tipo de enumeración y *name* es el nombre del enum. los valores están en "...". Esto crea una enumeración básica y un tipo para ir con él; los programas como Xcode asumirán que una variable con el tipo de enumeración tiene uno de los valores de enumeración

Examples

Definiendo una enumeración

Las enumeraciones se definen por la siguiente sintaxis.

```
typedef NS_ENUM(NSUInteger, MyEnum) {
    MyEnumValueA,
    MyEnumValueB,
    MyEnumValueC,
};
```

También puede establecer sus propios valores en bruto para los tipos de enumeración.

```
typedef NS_ENUM(NSUInteger, MyEnum) {
    MyEnumValueA = 0,
    MyEnumValueB = 5,
    MyEnumValueC = 10,
};
```

También puede especificar en el primer valor y todo lo siguiente lo usará con el incremento:

```
typedef NS_ENUM(NSUInteger, MyEnum) {
    MyEnumValueA = 0,
    MyEnumValueB,
    MyEnumValueC,
};
```

Las variables de esta enumeración pueden ser creadas por `MyEnum enumVar = MyEnumValueA`.

Declaración de enumeración typedef en Objective-C

Una enumeración declara un conjunto de valores ordenados; typedef simplemente agrega un nombre práctico a esto. El 1er elemento es 0 etc.

```
 typedef enum {
        Monday=1,
        Tuesday,
        Wednesday
```

```
    } WORKDAYS;

    WORKDAYS today = Monday;//value 1
```

Convertir C ++ std :: vector a una matriz Objective-C

Muchas bibliotecas de C ++ usan enumeraciones y devuelven / reciben datos utilizando vectores que contienen enumeraciones. Como C enums no son objetos de Objective-C, las colecciones de Objective-C no se pueden usar directamente con C enums. El siguiente ejemplo se ocupa de esto mediante el uso de una combinación de NSArray y genéricos y un objeto de envoltura para la matriz. De esta manera, la colección puede ser explícita sobre el tipo de datos y no hay ninguna preocupación acerca de las posibles fugas de memoria con matrices C Se utilizan objetos Objective-C.

Aquí está el objeto C enum y Objective-C equivalente:

```
typedef enum
{
  Error0 = 0,
  Error1 = 1,
  Error2 = 2
} MyError;

@interface ErrorEnumObj : NSObject

@property (nonatomic) int intValue;

+ (instancetype) objWithEnum:(MyError) myError;
- (MyError) getEnumValue;

@end

@implementation ErrorEnumObj

+ (instancetype) objWithEnum:(MyError) error
{
    ErrorEnumObj * obj = [ErrorEnumObj new];
    obj.intValue = (int)error;
    return obj;
}

- (MyError) getEnumValue
{
    return (MyError)self.intValue;
}

@end
```

Y aquí hay un posible uso de este en Objective-C ++ (la NSArray resultante se puede usar en archivos de Objective-C solo si no se usa C ++).

```
class ListenerImpl : public Listener
{
public:
```

```
    ListenerImpl(Listener* listener) : _listener(listener) {}
    void onError(std::vector<MyError> errors) override
    {
        NSMutableArray<ErrorEnumObj *> * array = [NSMutableArray<ErrorEnumObj *> new];
        for (auto&& myError : errors)
        {
            [array addObject:[ErrorEnumObj objWithEnum:myError]];
        }
        [_listener onError:array];
    }

private:
    __weak Listener* _listener;
}
```

Si este tipo de solución se va a utilizar en múltiples enumeraciones, la creación de EnumObj (declaración e implementación) se puede hacer usando una macro (para crear una plantilla como solución).

Lea Enums en línea: https://riptutorial.com/es/objective-c/topic/1461/enums

Capítulo 14: Especificadores de formato

Introducción

Los Especificadores de formato se utilizan en Objective-c para implantar valores de objeto en una cadena.

Sintaxis

- %@ //Cuerda
- % d // Entero de 32 bits firmado
- % D // Entero de 32 bits firmado
- % u // Entero de 32 bits sin signo
- % U // Entero de 32 bits sin signo
- % x // Entero de 32 bits sin signo en formato hexadecimal en minúsculas
- % X // Entero de 32 bits sin signo en MAYÚSCULAS en formato hexadecimal
- % o // Entero de 32 bits sin signo en formato octal
- % O // Entero de 32 bits sin signo en formato octal
- % f // número de punto flotante de 64 bits
- % F // Número de punto flotante de 64 bits impreso en notación decimal
- % e // número de punto flotante de 64 bits en formato de notación científica en minúsculas
- % E // número de punto flotante de 64 bits en formato de notación científica MAYÚSCULAS
- % g // caso especial% e que usa% f cuando hay menos de 4 sig-figs disponibles, de lo contrario% e
- % G // caso especial% E que usa% f cuando hay menos de 4 sig-figs disponibles, de lo contrario% E
- % c // carácter sin signo de 8 bits
- Unidad de código UTF-16 de% C // 16 bits
- % s // cadena UTF8
- % S // variante de 16 bits de% s
- % p // Void Pointer en formato hexadecial en minúsculas con '0x' al principio
- % zx // caso especial% p que elimina el '0x' inicial (para uso con conversión sin tipo)
- % a // número de punto flotante de 64 bits en notación científica con '0x' inicial y un dígito hexadecimal antes del punto decimal utilizando una 'p' para indicar el exponente.
- % A // número de punto flotante de 64 bits en notación científica con '0x' inicial y un dígito hexadecimal antes del punto decimal utilizando una 'P' para indicar el exponente.

Observaciones

Debido a la naturaleza de los especificadores de formato, si desea incluir el símbolo de porcentaje (%) en su cadena, debe escapar usando un segundo símbolo de porcentaje.

Ejemplo:

```
int progress = 45;//percent
NSString *progressString = [NSString stringWithFormat:@"Progress: %i%%", (int)progress];

NSLog(progressString);//logs "Progress: 45%"
```

No existe ningún especificador de formato para el tipo BOOL.

Las soluciones de uso común incluyen:

```
BOOL myBool = YES;
NSString *boolState = [NSString stringWithFormat:@"BOOL state: %@", myBool?@"true":@"false"];

NSLog(boolState);//logs "true"
```

Que utiliza un operador ternario para lanzar una cadena equivalente.

```
BOOL myBool = YES;
NSString *boolState = [NSString stringWithFormat:@"BOOL state: %i", myBool];

NSLog(boolState);//logs "1" (binary)
```

Que utiliza un molde (int) para implantar un equivalente binario.

Examples

Ejemplo de entero -% i

```
int highScore = 57;
NSString *scoreBoard = [NSString stringWithFormat:@"HighScore: %i", (int)highScore];

NSLog(scoreBoard);//logs "HighScore: 57"
```

Lea Especificadores de formato en línea: https://riptutorial.com/es/objective-c/topic/9048/especificadores-de-formato

Capítulo 15: Estructuras

Sintaxis

- typedef struct { *typeA propertyA* ; *typeB propertyB* ; ...} *StructName*

Observaciones

En el Objetivo C, casi siempre debes usar un objeto en lugar de una estructura. Sin embargo, todavía hay casos en los que es mejor usar una estructura, como:

- Cuando vaya a crear y destruir muchos valores del tipo (estructura), necesitará un buen rendimiento y un pequeño uso de memoria.
    - Las estructuras son más rápidas de crear y usar porque al llamar a un método en un objeto, el método debe determinarse en tiempo de ejecución
    - Las estructuras ocupan menos tamaño porque los objetos tienen una propiedad extra `isa` , que mantiene su clase
- Cuando el valor solo tiene un par de propiedades y un tamaño total pequeño (tome `CGSize` ; tiene 2 flotadores que tienen 4 bytes cada uno, por lo que puede ocupar 8 bytes), y se usará mucho (se vincula con el primer punto)
- Cuando podría usar uniones o campos de bits y, lo que es más importante, *necesita el tamaño guardado por ellos* porque necesita un pequeño uso de memoria (concuerda con el primer punto)
- Cuando *realmente* desea almacenar una matriz dentro de la estructura, ya que los objetos Objective-C no pueden almacenar directamente las matrices-C. Sin embargo, tenga en cuenta que todavía puede obtener "indirectamente" una matriz en un objeto Objective-C al convertirla en una referencia (es decir, `type *` en lugar del `type[]` matriz C `type[]`)
- Cuando necesita comunicarse con algún otro código, como una biblioteca, se codifica en C; Las estructuras están completamente implementadas en C, pero los objetos no están

Examples

CGPoint

Un ejemplo realmente bueno de una estructura es `CGPoint` ; Es un valor simple que representa un punto bidimensional. Tiene 2 propiedades, `x` y `y` , y puede ser escrito como

```
typedef struct {
    CGFloat x;
    CGFloat y;
} CGPoint;
```

Si usó anteriormente Objective-C para el desarrollo de aplicaciones Mac o iOS, es casi seguro

que se haya topado con `CGPoint` ; `CGPoint` mantiene la posición de casi todo en la pantalla, desde vistas y controles a objetos en un juego hasta cambios en un degradado. Esto significa que los `CGPoint` se usan mucho. Esto es aún más cierto con juegos realmente pesados; Estos juegos tienden a tener muchos objetos, y todos estos objetos necesitan posiciones. Estas posiciones suelen ser `CGPoint` o algún otro tipo de estructura que transmite un punto (como un punto tridimensional para juegos 3D).

Puntos como `CGPoint` podrían representarse fácilmente como objetos, como

```
@interface CGPoint {
    CGFloat x;
    CGFloat y;
}

... //Point-related methods (e.g. add, isEqualToPoint, etc.)

@property(nonatomic, assign)CGFloat x;
@property(nonatomic, assign)CGFloat y;

@end

@implementation CGPoint

@synthesize x, y;

...

@end
```

Sin embargo, si `CGPoint` se usara de esta manera, tomaría mucho más tiempo crear y manipular puntos. En programas más pequeños y rápidos, esto realmente no causaría una diferencia, y en esos casos sería correcto o incluso mejor usar puntos de objeto. Pero en los programas grandes donde los puntos se usan mucho, usar objetos como puntos realmente puede afectar el rendimiento, hacer que el programa sea más lento y también desperdiciar memoria, lo que podría forzar al programa a fallar.

Definición de una estructura y acceso a los miembros de la estructura

El formato de la sentencia struct es este:

```
struct [structure tag]
{
   member definition;
   member definition;
   ...
   member definition;
} [one or more structure variables];
```

Ejemplo: declarar la estructura de ThreeFloats:

```
   typedef struct {
    float x, y, z;
```

```
} ThreeFloats;

@interface MyClass
- (void)setThreeFloats:(ThreeFloats)threeFloats;
- (ThreeFloats)threeFloats;
@end
```

Al enviar una instancia de MyClass, el mensaje valueForKey: con el parámetro @ "threeFloats" invocará el método MyClass threeFloats y devolverá el resultado envuelto en un NSValue.

Lea Estructuras en línea: https://riptutorial.com/es/objective-c/topic/3792/estructuras

Capítulo 16: Explotación florestal

Sintaxis

- NSLog (@ "text to log"); // Registro de texto básico
- NSLog (@ "data:% f -% .2f", myFloat, anotherFloat); // Registro de texto incluyendo números flotantes.
- NSLog (@ "data:% i", myInteger); // Registro de texto incluyendo número entero.
- NSLog (@ "data:% @", myStringOrObject); // Registro de texto que hace referencia a otra cadena o cualquier objeto derivado de NSObject.

Observaciones

Para registrar diversos tipos de objetos y tipos de datos, consulte: Objective-C, Especificadores de formato

Examples

Explotación florestal

```
NSLog(@"Log Message!");
NSLog(@"NSString value: %@", stringValue);
NSLog(@"Integer value: %d", intValue);
```

El primer argumento de `NSLog` es un `NSString` contiene el formato de mensaje de registro. El resto de los parámetros se utilizan como valores para sustituir en lugar de los especificadores de formato.

El formato funciona exactamente igual que `printf` , excepto por el especificador de formato adicional `%@` para un objeto Objective-C arbitrario. Esta:

```
NSLog(@"%@", object);
```

es equivalente a:

```
NSLog(@"%s", [object description].UTF8String);
```

NSLog vs printf

```
NSLog(@"NSLog message");
printf("printf message\n");
```

Salida:

```
2016-07-16 08:58:04.681 test[46259:1244773] NSLog message
printf message
```

`NSLog` genera la fecha, la hora, el nombre del proceso, la ID del proceso y la ID del hilo, además del mensaje de registro. `printf` acaba de enviar el mensaje.

`NSLog` requiere una `NSString` y agrega automáticamente una nueva línea al final. `printf` requiere una cadena C y no agrega automáticamente una nueva línea.

`NSLog` envía la salida a `stderr` , `printf` envía la salida a la salida `stdout` .

Algunos `format-specifiers` en `printf` vs `NSLog` son diferentes. Por ejemplo, cuando se incluye una cadena anidada, se producen las siguientes diferencias:

```
NSLog(@"My string: %@", (NSString *)myString);
printf("My string: %s", [(NSString *)myString UTF8String]);
```

Formato de salida NSLog

```
NSLog(@"NSLog message");
```

El mensaje que se imprime al llamar a `NSLog` tiene el siguiente formato cuando se ve en Console.app:

Fecha	Hora	Nombre del programa	Identificacion de proceso		ID de hilo	Mensaje
2016-07-16	08:58:04.681	test	[46259	:	1244773]	NSLog message

Registro de valores variables

No deberías llamar a `NSLog` sin una cadena de formato literal como esta:

```
NSLog(variable);    // Dangerous code!
```

Si la variable no es una `NSString` , el programa se bloqueará, porque `NSLog` espera una `NSString` .

Si la variable es una `NSString` , funcionará a menos que su cadena contenga un `%` . `NSLog` analizará la secuencia `%` como un especificador de formato y luego leerá un valor de basura de la pila, causando un bloqueo o incluso ejecutando código arbitrario .

En su lugar, siempre haga que el primer argumento sea un especificador de formato, como este:

```
NSLog(@"%@", anObjectVariable);
NSLog(@"%d", anIntegerVariable);
```

El mensaje vacío no se imprime

Cuando se le pide a `NSLog` que imprima una cadena vacía, omite el registro por completo.

```
NSString *name = @"";
NSLog(@"%@", name);  // Resolves to @""
```

El código anterior no imprimirá **nada** .

Es una buena práctica prefijar los registros con etiquetas:

```
NSString *name = @"";
NSLog(@"Name: %@", name);  // Resolves to @"Name: "
```

El código de arriba se imprimirá:

```
2016-07-21 14:20:28.623 App[87711:6153103] Name:
```

Eliminar declaraciones de registro de versiones de lanzamiento

Los mensajes impresos desde `NSLog` se muestran en Console.app incluso en la versión de lanzamiento de su aplicación, lo que no tiene sentido para las impresiones que solo son útiles para la depuración. Para solucionar esto, puede usar esta macro para el registro de depuración en lugar de `NSLog` .

```
#ifdef DEBUG
#define DLog(...) NSLog(__VA_ARGS__)
#else
#define DLog(...)
#endif
```

Usar:

```
NSString *value = @"value 1";
DLog(@"value = %@", value);
// little known fact: programmers look for job postings in Console.app
NSLog(@"We're hiring!");
```

En las versiones de depuración, `DLog` llamará `NSLog` . En las versiones de lanzamiento, `DLog` no hará nada.

Usando __FUNCION __

```
NSLog(@"%s %@",__FUNCTION__, @"etc etc");
```

Inserta la clase y el nombre del método en la salida:

```
2016-07-22 12:51:30.099 loggingExample[18132:2971471] -[ViewController viewDidLoad] etc etc
```

Tipo NSLog y BOOL

No hay un especificador de formato para imprimir tipo booleano utilizando NSLog. Una forma de imprimir un valor booleano es convertirlo en una cadena.

```
BOOL boolValue = YES;
NSLog(@"Bool value %@", boolValue ? @"YES" : @"NO");
```

Salida:

```
2016-07-30 22:53:18.269 Test[4445:64129] Bool value YES
```

Otra forma de imprimir un valor booleano es convertirlo en entero, logrando una salida binaria (1 = sí, 0 = no).

```
BOOL boolValue = YES;
NSLog(@"Bool value %i", boolValue);
```

Salida:

```
2016-07-30 22:53:18.269 Test[4445:64129] Bool value 1
```

Registro de metadatos NSLog

```
NSLog(@"%s %d %s, yourVariable: %@", __FILE__, __LINE__, __PRETTY_FUNCTION__, yourVariable);
```

Registrará el archivo, el número de línea y los datos de la función junto con cualquier variable que desee registrar. Esto puede hacer que las líneas de registro sean más largas, particularmente con nombres de archivos y métodos detallados, sin embargo, puede ayudar a acelerar el diagnóstico de errores.

También puede envolver esto en una Macro (guárdela en un Singleton o donde más la necesite);

```
#define ALog(fmt, ...) NSLog((@"%s [Line %d] " fmt), __PRETTY_FUNCTION__, __LINE__,
##__VA_ARGS__);
```

Luego, cuando desee iniciar sesión, simplemente llame

```
ALog(@"name: %@", firstName);
```

Lo que te dará algo como;

```
 -[AppDelegate application:didFinishLaunchingWithOptions:] [Line 27] name: John
```

Iniciar sesión agregando a un archivo

NSLog es bueno, pero también puede iniciar sesión agregando un archivo en su lugar, utilizando un código como:

```
NSFileHandle* fh = [NSFileHandle fileHandleForWritingAtPath:path];
if ( !fh ) {
    [[NSFileManager defaultManager] createFileAtPath:path contents:nil attributes:nil];
    fh = [NSFileHandle fileHandleForWritingAtPath:path];
}
if ( fh ) {
    @try {
        [fh seekToEndOfFile];
        [fh writeData:[self dataUsingEncoding:enc]];
    }
    @catch (...) {
    }
    [fh closeFile];
}
```

Lea Explotación florestal en línea: https://riptutorial.com/es/objective-c/topic/724/explotacion-florestal

Capítulo 17: Gestión de la memoria

Examples

Conteo automático de referencias

Con el recuento automático de referencias (ARC), las inserciones del compilador `retain` , `release` y `autorelease` donde se necesitan, por lo que no tiene que escribirlas usted mismo. También escribe métodos `dealloc` para usted.

El programa de muestra de Manual Memory Management se ve así con ARC:

```
@interface MyObject : NSObject {
    NSString *_property;
}
@end

@implementation MyObject
@synthesize property = _property;

- (id)initWithProperty:(NSString *)property {
    if (self = [super init]) {
        _property = property;
    }
    return self;
}

- (NSString *)property {
    return property;
}

- (void)setProperty:(NSString *)property {
    _property = property;
}

@end
```

```
int main() {
    MyObject *obj = [[MyObject alloc] init];

    NSString *value = [[NSString alloc] initWithString:@"value"];
    [obj setProperty:value];

    [obj setProperty:@"value"];
}
```

Aún puede anular el método dealloc para limpiar los recursos no gestionados por ARC. A diferencia de cuando se usa la administración de memoria manual, no se llama `[super dealloc]` .

```
-(void)dealloc {
   //clean up
}
```

Referencias fuertes y débiles.

Moderno

Una referencia débil se parece a una de estas:

```
@property (weak) NSString *property;
NSString *__weak variable;
```

Si tiene una referencia débil a un objeto, entonces bajo el capó:

- No lo estás reteniendo.
- Cuando se desasigna, cada referencia se establecerá automáticamente en `nil`

Las referencias a objetos son siempre fuertes por defecto. Pero puedes especificar explícitamente que son fuertes:

```
@property (strong) NSString *property;
NSString *__strong variable;
```

Una referencia fuerte significa que mientras exista esa referencia, está reteniendo el objeto.

Gestión de memoria manual

Este es un ejemplo de un programa escrito con administración de memoria manual. Realmente no deberías escribir tu código de esta manera, a menos que por alguna razón no puedas usar ARC (como si necesitas soportar 32 bits). El ejemplo evita la notación de `@property` para ilustrar cómo solía tener que escribir getters y setters.

```
@interface MyObject : NSObject {
    NSString *_property;
}
@end

@implementation MyObject
@synthesize property = _property;

- (id)initWithProperty:(NSString *)property {
    if (self = [super init]) {
        // Grab a reference to property to make sure it doesn't go away.
        // The reference is released in dealloc.
        _property = [property retain];
    }
    return self;
}

- (NSString *)property {
    return [[property retain] autorelease];
}

- (void)setProperty:(NSString *)property {
    // Retain, then release. So setting it to the same value won't lose the reference.
    [property retain];
```

```
    [_property release];
    _property = property;
}

- (void)dealloc {
    [_property release];
    [super dealloc]; // Don't forget!
}

@end
```

```
int main() {
    // create object
    // obj is a reference that we need to release
    MyObject *obj = [[MyObject alloc] init];

    // We have to release value because we created it.
    NSString *value = [[NSString alloc] initWithString:@"value"];
    [obj setProperty:value];
    [value release];

    // However, string constants never need to be released.
    [obj setProperty:@"value"];
    [obj release];
}
```

Reglas de gestión de memoria cuando se utiliza el conteo manual de referencias.

Estas reglas solo se aplican si utiliza el recuento manual de referencias

1. **Eres dueño de cualquier objeto que crees**

   Al llamar a un método cuyo nombre comienza con `alloc` , `new` , `copy` o `mutableCopy` . Por ejemplo:

   ```
   NSObject *object1 = [[NSObject alloc] init];
   NSObject *object2 = [NSObject new];
   NSObject *object3 = [object2 copy];
   ```

   Eso significa que usted es responsable de liberar estos objetos cuando haya terminado con ellos.

2. **Puedes tomar posesión de un objeto usando retener**

   Para tomar posesión de un objeto, se llama el método de retención.

   Por ejemplo:

   ```
   NSObject *object = [NSObject new]; // object already has a retain count of 1
   [object retain]; // retain count is now 2
   ```

   Esto solo tiene sentido en algunas situaciones raras.

Por ejemplo, cuando implementas un descriptor de acceso o un método init para tomar posesión:

```
- (void)setStringValue:(NSString *)stringValue {
    [_privateStringValue release]; // Release the old value, you no longer need it
    [stringValue retain]; // You make sure that this object does not get deallocated
outside of your scope.
    _privateStringValue = stringValue;
}
```

3. **Cuando ya no lo necesite, debe renunciar a la propiedad de un objeto que posee**

```
NSObject* object = [NSObject new]; // The retain count is now 1
[object performAction1]; // Now we are done with the object
[object release]; // Release the object
```

4. **No debe renunciar a la propiedad de un objeto que no posee**

   Eso significa que cuando no te apropiaste de un objeto, no lo liberas.

5. **Autoreleasepool**

   El autoreleasepool es un bloque de código que libera cada objeto en el bloque que recibió un mensaje de autorelease.

   Ejemplo:

```
@autoreleasepool {
    NSString* string = [NSString stringWithString:@"We don't own this object"];
}
```

   Hemos creado una cadena sin tomar posesión. El `NSString` método `stringWithString:` tiene que asegurarse de que la cadena se desasigna correctamente después de que ya no es necesaria. Antes de que el método devuelva la cadena recién creada, se llama al método autorelease para que no tenga que tomar posesión de la cadena.

   Así es como se implementa el `stringWithString:` :

```
+ (NSString *)stringWithString:(NSString *)string {
    NSString *createdString = [[NSString alloc] initWithString:string];
    [createdString autorelease];
    return createdString;
}
```

   Es necesario usar bloques de autoreleasepool porque a veces tienes objetos que no tienes (las cuartas reglas no siempre se aplican).

   El conteo automático de referencias cuida automáticamente las reglas para que no tenga que hacerlo.

Lea Gestión de la memoria en línea: https://riptutorial.com/es/objective-c/topic/2751/gestion-de-la-

memoria

Capítulo 18: Grand Central Dispatch

Introducción

Grand Central Dispatch (GCD) En iOS, Apple ofrece dos formas de realizar tareas múltiples: Grand Central Dispatch (GCD) y los marcos de NSOperationQueue. Discutiremos aquí sobre GCD. GCD es una forma liviana de representar unidades de trabajo que se ejecutarán simultáneamente. No se programan estas unidades de trabajo; El sistema se encarga de la programación por usted. Agregar dependencia entre bloques puede ser un dolor de cabeza. ¡Cancelar o suspender un bloque crea un trabajo adicional para usted como desarrollador!

Examples

Qué es Grand Central Dispatch.

¿Qué es la concurrencia?

- Haciendo múltiples cosas al mismo tiempo.
- Aprovechando la cantidad de núcleos disponibles en CPUs multinúcleo.
- Ejecutando múltiples programas en paralelo.

Objetivos de concurrencia.

- Ejecutando el programa en segundo plano sin acaparar la CPU.
- Defina tareas, defina reglas y deje que el sistema asuma la responsabilidad de realizarlas.
- Mejore la capacidad de respuesta asegurándose de que el hilo principal sea libre de responder a los eventos de los usuarios.

COLAS DE DESPACHO

Grand Central Dispatch: las colas de despacho nos permiten ejecutar bloques de código arbitrarios de forma asíncrona o síncrona. Todas las colas de despacho primero en entrar, primero en salir Todas las tareas agregadas a la cola de despacho se inician en el orden en que se agregaron a la cola de despacho.

Lea Grand Central Dispatch en línea: https://riptutorial.com/es/objective-c/topic/8280/grand-central-dispatch

Capítulo 19: Herencia

Sintaxis

1. @interface nombre-clase-derivado: nombre-clase-base

Examples

El coche se hereda del vehículo.

Considere un **vehículo de** clase base y su **automóvil de** clase derivada de la siguiente manera:

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

@interface Vehicle : NSObject

{
    NSString *vehicleName;
    NSInteger vehicleModelNo;
}

- (id)initWithName:(NSString *)name andModel:(NSInteger)modelno;
- (void)print;
@end

@implementation Vehicle

- (id)initWithName:(NSString *)name andModel:(NSInteger)modelno{
    vehicleName = name;
    vehicleModelNo = modelno;
    return self;
}

- (void)print{
    NSLog(@"Name: %@", vehicleName);
    NSLog(@"Model: %ld", vehicleModelNo);
}

@end

@interface Car : Vehicle

{
    NSString *carCompanyName;
}

- (id)initWithName:(NSString *)name andModel:(NSInteger)modelno
  andCompanyName:(NSString *)companyname;
- (void)print;

@end


@implementation Car
```

```
- (id)initWithName:(NSString *)name andModel:(NSInteger) modelno
  andCompanyName: (NSString *) companyname
  {
    vehicleName = name;
    vehicleModelNo = modelno;
    carCompanyName = companyname;
    return self;
}
- (void)print
{
    NSLog(@"Name: %@", vehicleName);
    NSLog(@"Model: %ld", vehicleModelNo);
    NSLog(@"Company: %@", carCompanyName);
}

@end

int main(int argc, const char * argv[])
{
    NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];
    NSLog(@"Base class Vehicle Object");
    Vehicle *vehicle = [[Vehicle alloc]initWithName:@"4Wheeler" andModel:1234];
    [vehicle print];
    NSLog(@"Inherited Class Car Object");
    Car *car = [[Car alloc]initWithName:@"S-Class"
    andModel:7777 andCompanyName:@"Benz"];
    [car print];
    [pool drain];
    return 0;
}
```

Cuando el código anterior se compila y ejecuta, produce el siguiente resultado:

2016-09-29 18: 21: 03.561 Herencia [349: 303] Objeto de vehículo de clase base

2016-09-29 18: 21: 03.563 Herencia [349: 303] Nombre: 4Wheeler

2016-09-29 18: 21: 03.563 Herencia [349: 303] Modelo: 1234

2016-09-29 18: 21: 03.564 Herencia [349: 303] Objeto heredado de clase de automóvil

2016-09-29 18: 21: 03.564 Herencia [349: 303] Nombre: Clase S

2016-09-29 18: 21: 03.565 Herencia [349: 303] Modelo: 7777

2016-09-29 18: 21: 03.565 Herencia [349: 303] Compañía: Benz

Lea Herencia en línea: https://riptutorial.com/es/objective-c/topic/7117/herencia

Capítulo 20: Macros predefinidas

Introducción

ANSI C define una serie de macros. Aunque cada una está disponible para su uso en la programación, las macros predefinidas no deben modificarse directamente.

Sintaxis

1. **FECHA** La fecha actual como un literal de carácter en formato "MMM DD YYYY"
2. **HORA** La hora actual como un literal de carácter en formato "HH: MM: SS"
3. **ARCHIVO** Contiene el nombre de archivo actual como una cadena literal.
4. **LÍNEA** Contiene el número de línea actual como una constante decimal.
5. **STDC** Definido como 1 cuando el compilador cumple con el estándar ANSI.

Examples

Macros predefinidas

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

int main()
{
   NSLog(@"File :%s\n", __FILE__ );
   NSLog(@"Date :%s\n", __DATE__ );
   NSLog(@"Time :%s\n", __TIME__ );
   NSLog(@"Line :%d\n", __LINE__ );
   NSLog(@"ANSI :%d\n", __STDC__ );

   return 0;
}
```

Cuando el código anterior en un archivo main.m se compila y ejecuta, produce el siguiente resultado:

```
2013-09-14 04:46:14.859 demo[20683] File :main.m
2013-09-14 04:46:14.859 demo[20683] Date :Sep 14 2013
2013-09-14 04:46:14.859 demo[20683] Time :04:46:14
2013-09-14 04:46:14.859 demo[20683] Line :8
2013-09-14 04:46:14.859 demo[20683] ANSI :1
```

Lea Macros predefinidas en línea: https://riptutorial.com/es/objective-c/topic/8254/macros-predefinidas

Capítulo 21: Manejo de errores

Sintaxis

- NSAssert (condition, fmtMessage, *arg1, arg2, ...*) (los argumentos en cursiva son opcionales): afirma que la *condición se* evalúa a un valor verdadero. Si no es así, la aserción generará una excepción (NSAssertionException), con el formato *fmtMessage* con los *argumentos* proporcionados

Examples

Afirmando

```
@implemenetation Triangle

...

-(void)setAngles:(NSArray *)_angles {
    self.angles = _angles;

    NSAssert((self.angles.count == 3), @"Triangles must have 3 angles. Array '%@' has %i",
self.angles, (int)self.angles.count);

    CGFloat angleA = [self.angles[0] floatValue];
    CGFloat angleB = [self.angles[1] floatValue];
    CGFloat angleC = [self.angles[2] floatValue];
    CGFloat sum = (angleA + angleB + angleC);
    NSAssert((sum == M_PI), @"Triangles' angles must add up to pi radians (180°). This
triangle's angles add up to %f radians (%f°)", (float)sum, (float)(sum * (180.0f / M_PI)));
}
```

Estas afirmaciones aseguran que no se le den ángulos incorrectos a un triángulo, lanzando una excepción si lo hace. Si no lanzaron una excepción que el triángulo, el no ser un triángulo verdadero podría causar algunos errores en el código posterior.

Manejo de errores y excepciones con el bloque try catch

Las excepciones representan errores a nivel de programador, como intentar acceder a un elemento de matriz que no existe.

Los errores son problemas de nivel de usuario, como intentar cargar un archivo que no existe. Porque se esperan errores durante la ejecución normal de un programa.

Ejemplo:

```
    NSArray *inventory = @[@"Sam",
                           @"John",
                           @"Sanju"];
    int selectedIndex = 3;
```

```
    @try {
        NSString * name = inventory[selectedIndex];
        NSLog(@"The selected Name is: %@", name);
    } @catch(NSException *theException) {
        NSLog(@"An exception occurred: %@", theException.name);
        NSLog(@"Here are some details: %@", theException.reason);
    } @finally {
        NSLog(@"Executing finally block");
    }
```

SALIDA:

> Se produjo una excepción: NSRangeException
>
> Aquí hay algunos detalles: *** - [__ NSArrayI objectAtIndex:]: índice 3 más allá de los límites [0 .. 2]
>
> Ejecutando finalmente el bloque

Lea Manejo de errores en línea: https://riptutorial.com/es/objective-c/topic/1459/manejo-de-errores

Capítulo 22: Métodos

Sintaxis

- `-` o `+` : El tipo de método. ¿Instancia o clase?
- (): Donde va el tipo de retorno. ¡Usa void si no quieres devolver nada!
- El siguiente es el nombre del método. Use camelCase y haga que el nombre sea fácil de recordar y comprender.
- Si tu método necesita parámetros, ¡ahora es el momento! El primer parámetro viene justo después del nombre de la función como esta `:(type)parameterName` . Todos los demás parámetros se realizan de esta manera `parameterLabel:(type)parameterName`
- ¿Qué hace tu método? ¡Ponlo todo aquí, entre llaves {}!

Examples

Parámetros del método

Si desea pasar valores a un método cuando se le llama, use parámetros:

```
- (int)addInt:(int)intOne toInt:(int)intTwo {
  return intOne + intTwo;
}
```

Los dos puntos (`:`) separa el parámetro del nombre del método.

El tipo de parámetro va entre paréntesis `(int)` .

El nombre del parámetro va después del tipo de parámetro.

Crea un método básico.

Así es como se crea un método básico que registra "Hello World" en la consola:

```
- (void)hello {
  NSLog(@"Hello World");
}
```

El `-` al principio denota este método como un método de instancia.

El `(void)` denota el tipo de retorno. Este método no devuelve nada, así que entras en `void` .

El 'hola' es el nombre del método.

Todo en el `{}` es el código que se ejecuta cuando se llama al método.

Valores de retorno

Cuando desea devolver un valor de un método, coloque el tipo que desea devolver en el primer conjunto de paréntesis.

```
- (NSString)returnHello {
  return @"Hello World";
}
```

El valor que desea devolver va después de la palabra clave `return` ;

Métodos de clase

Se llama a un método de clase en la clase a la que pertenece el método, no a una instancia de él. Esto es posible porque las clases de Objective-C también son objetos. Para denotar un método como un método de clase, cambie el `-` a un `+` :

```
+ (void)hello {
  NSLog(@"Hello World");
}
```

Métodos de llamada

Llamando a un método de instancia:

```
[classInstance hello];

 @interface Sample
 -(void)hello; // exposing the class Instance method
 @end

 @implementation Sample
     -(void)hello{
        NSLog(@"hello");
      }
 @end
```

Llamando a un método de instancia en la instancia actual:

```
[self hello];

@implementation Sample

     -(void)otherMethod{
       [self hello];
     }

     -(void)hello{
        NSLog(@"hello");
      }
```

```
 @end
```

Llamando a un método que toma argumentos:

```
[classInstance addInt:1 toInt:2];

 @implementation Sample
     -(void)add:(NSInteger)add to:(NSInteger)to
        NSLog(@"sum = %d",(add+to));
      }
 @end
```

Llamando a un método de clase:

```
[Class hello];

 @interface Sample
 +(void)hello; // exposing the class method
 @end

 @implementation Sample
     +(void)hello{
        NSLog(@"hello");
      }
 @end
```

Metodos de instancia

Un método de instancia es un método que está disponible en una instancia particular de una clase, después de que la instancia haya sido instanciada:

```
MyClass *instance = [MyClass new];
[instance someInstanceMethod];
```

Así es como se define uno:

```
@interface MyClass : NSObject

- (void)someInstanceMethod; // "-" denotes an instance method

@end

@implementation MyClass

- (void)someInstanceMethod {
    NSLog(@"Whose idea was it to have a method called \"someInstanceMethod\"?");
}

@end
```

Pase por el paso del parámetro valor

Al pasar por el valor del parámetro que pasa a un método, el valor del parámetro real se copia al

valor del parámetro formal. Por lo tanto, el valor del parámetro real no cambiará después de regresar de la función llamada.

```
@interface SwapClass : NSObject

-(void) swap:(NSInteger)num1 andNum2:(NSInteger)num2;

@end

@implementation SwapClass

-(void) num:(NSInteger)num1 andNum2:(NSInteger)num2{
    int temp;
    temp = num1;
    num1 = num2;
    num2 = temp;
}
@end
```

Llamando a los métodos:

```
NSInteger a = 10, b =20;
SwapClass *swap = [[SwapClass alloc]init];
NSLog(@"Before calling swap: a=%d,b=%d",a,b);
[swap num:a andNum2:b];
NSLog(@"After calling swap: a=%d,b=%d",a,b);
```

Salida:

```
2016-07-30 23:55:41.870 Test[5214:81162] Before calling swap: a=10,b=20
2016-07-30 23:55:41.871 Test[5214:81162] After calling swap: a=10,b=20
```

Pase por parámetro de referencia pasando

En el paso por referencia del parámetro que pasa a un método, la dirección del parámetro real se pasa al parámetro formal. Por lo tanto, el valor del parámetro real se cambiará después de regresar de la función llamada.

```
@interface SwapClass : NSObject

-(void) swap:(int)num1 andNum2:(int)num2;

@end

@implementation SwapClass

-(void) num:(int*)num1 andNum2:(int*)num2{
    int temp;
    temp = *num1;
    *num1 = *num2;
    *num2 = temp;
}
@end
```

Llamando a los métodos:

```
int a = 10, b =20;
SwapClass *swap = [[SwapClass alloc]init];
NSLog(@"Before calling swap: a=%d,b=%d",a,b);
[swap num:&a andNum2:&b];
NSLog(@"After calling swap: a=%d,b=%d",a,b);
```

Salida:

```
2016-07-31 00:01:47.067 Test[5260:83491] Before calling swap: a=10,b=20
2016-07-31 00:01:47.070 Test[5260:83491] After calling swap: a=20,b=10
```

Lea Métodos en línea: https://riptutorial.com/es/objective-c/topic/1457/metodos

Capítulo 23: Multihilo

Examples

Creando un hilo simple

La forma más sencilla de crear un hilo es llamando a un selector "en el fondo". Esto significa que se crea un nuevo hilo para ejecutar el selector. El objeto receptor puede ser cualquier objeto, no solo `self` , sino que debe responder al selector dado.

```
- (void)createThread {
    [self performSelectorInBackground:@selector(threadMainWithOptionalArgument:)
                           withObject:someObject];
}

- (void)threadMainWithOptionalArgument:(id)argument {
    // To avoid memory leaks, the first thing a thread method needs to do is
    // create a new autorelease pool, either manually or via "@autoreleasepool".
    @autoreleasepool {
        // The thread code should be here.
    }
}
```

Crear hilo más complejo.

El uso de una subclase de `NSThread` permite la implementación de subprocesos más complejos (por ejemplo, para permitir pasar más argumentos o para encapsular todos los métodos de ayuda relacionados en una clase). Además, la instancia de `NSThread` se puede guardar en una propiedad o variable y se puede consultar sobre su estado actual (si todavía se está ejecutando).

La clase `NSThread` admite un método llamado `cancel` que se puede llamar desde cualquier subproceso, que luego establece la propiedad `cancelled` en `YES` de una manera segura para subprocesos. La implementación del hilo puede consultar (y / u observar) la propiedad `cancelled` y salir de su método `main` . Esto se puede utilizar para cerrar con gracia un subproceso de trabajo.

```
// Create a new NSThread subclass
@interface MyThread : NSThread

// Add properties for values that need to be passed from the caller to the new
// thread. Caller must not modify these once the thread is started to avoid
// threading issues (or the properties must be made thread-safe using locks).
@property NSInteger someProperty;

@end

@implementation MyThread

- (void)main
{
    @autoreleasepool {
        // The main thread method goes here
```

```
        NSLog(@"New thread. Some property: %ld", (long)self.someProperty);
    }
}

@end


MyThread *thread = [[MyThread alloc] init];
thread.someProperty = 42;
[thread start];
```

Almacenamiento de hilo local

Cada hilo tiene acceso a un diccionario mutable que es local al hilo actual. Esto permite almacenar información en caché de una manera fácil sin la necesidad de bloquear, ya que cada hilo tiene su propio diccionario mutable dedicado:

```
NSMutableDictionary *localStorage = [NSThread currentThread].threadDictionary;
localStorage[someKey] = someValue;
```

El diccionario se libera automáticamente cuando el hilo termina.

Lea Multihilo en línea: https://riptutorial.com/es/objective-c/topic/3350/multihilo

Capítulo 24: NSArray

Sintaxis

- NSArray * palabras; // Declarando matriz inmutable
- NSMutableArray * palabras; // Declarando matriz mutable
- NSArray * palabras = [NSArray arrayWithObjects: @ "one", @ "two", nil]; // Sintaxis de inicialización de matriz
- NSArray * words = @ [@ "list", @ "of", @ "words", @ 123, @ 3.14]; // Declarando matrices literales
- NSArray * stringArray = [NSArray arrayWithObjects: [NSMutableArray array], [NSMutableArray array], [NSMutableArray array], nil]; // Creando matrices multidimensionales

Examples

Creando Arrays

Creando arreglos inmutables:

```
NSArray *myColors = [NSArray arrayWithObjects: @"Red", @"Green", @"Blue", @"Yellow", nil];

// Using the array literal syntax:
NSArray *myColors = @[@"Red", @"Green", @"Blue", @"Yellow"];
```

Para matrices mutables, vea NSMutableArray .

Averiguar el número de elementos en una matriz

```
NSArray *myColors = [NSArray arrayWithObjects: @"Red", @"Green", @"Blue", @"Yellow", nil];
NSLog (@"Number of elements in array = %lu", [myColors count]);
```

Elementos de acceso

```
NSArray *myColors = @[@"Red", @"Green", @"Blue", @"Yellow"];
// Preceding is the preferred equivalent to [NSArray arrayWithObjects:...]
```

Obtener un solo artículo

El método `objectAtIndex:` proporciona un solo objeto. El primer objeto en una `NSArray` es el índice 0. Dado que una `NSArray` puede ser homogénea (que contiene diferentes tipos de objetos), el tipo de retorno es `id` ("cualquier objeto"). (Un `id` puede asignarse a una variable de cualquier otro tipo de objeto). Es importante destacar que `NSArray` s solo puede contener objetos. No pueden contener valores como `int` .

```
NSUInteger idx = 2;
NSString *color = [myColors objectAtIndex:idx];
// color now points to the string @"Green"
```

Clang proporciona una mejor sintaxis de subíndices como parte de su funcionalidad de literales de matriz :

```
NSString *color = myColors[idx];
```

Ambos lanzan una excepción si el índice pasado es menor que 0 o mayor que `count - 1` .

Primer y último artículo

```
NSString *firstColor = myColors.firstObject;
NSString *lastColor = myColors.lastObject;
```

`firstObject` y `lastObject` son propiedades computadas y devuelven `nil` lugar de fallar para matrices vacías. Para matrices de elementos individuales devuelven el mismo objeto. Aunque, el método `firstObject` no se introdujo en `NSArray` hasta iOS 4.0.

```
NSArray *empty = @[]
id notAnObject = empty.firstObject;    // Returns `nil`
id kaboom = empty[0];    // Crashes; index out of bounds
```

Filtrado de matrices con predicados

```
NSArray *array = [NSArray arrayWithObjects:@"Nick", @"Ben", @"Adam", @"Melissa", nil];

NSPredicate *aPredicate = [NSPredicate predicateWithFormat:@"SELF beginswith[c] 'a'"];
NSArray *beginWithA = [array filteredArrayUsingPredicate:bPredicate];
     // beginWithA contains { @"Adam" }.

NSPredicate *ePredicate = [NSPredicate predicateWithFormat:@"SELF contains[c] 'e'"];
[array filterUsingPredicate:ePredicate];
    // array now contains { @"Ben", @"Melissa" }
```

Más sobre

NSPredicate :

Apple doc: NSPredicate

Convertir NSArray a NSMutableArray para permitir modificaciones

```
NSArray *myColors = [NSArray arrayWithObjects: @"Red", @"Green", @"Blue", @"Yellow", nil];

// Convert myColors to mutable
NSMutableArray *myColorsMutable = [myColors mutableCopy];
```

Ordenando matriz con objetos personalizados

Método de comparación

O implementas un método de comparación para tu objeto:

```
- (NSComparisonResult)compare:(Person *)otherObject {
    return [self.birthDate compare:otherObject.birthDate];
}

NSArray *sortedArray = [drinkDetails sortedArrayUsingSelector:@selector(compare:)];
```

NSSortDescriptor

```
NSSortDescriptor *sortDescriptor;
sortDescriptor = [[NSSortDescriptor alloc] initWithKey:@"birthDate"
                                             ascending:YES];
NSArray *sortDescriptors = [NSArray arrayWithObject:sortDescriptor];
NSArray *sortedArray = [drinkDetails sortedArrayUsingDescriptors:sortDescriptors];
```

Puede ordenar fácilmente por varias claves agregando más de una a la matriz. También es posible usar métodos de comparación personalizados. Echa un vistazo a la documentación .

Bloques

```
NSArray *sortedArray;
sortedArray = [drinkDetails sortedArrayUsingComparator:^NSComparisonResult(id a, id b) {
    NSDate *first = [(Person*)a birthDate];
    NSDate *second = [(Person*)b birthDate];
    return [first compare:second];
}];
```

Actuación

El `-compare:` y los métodos basados en bloques serán un poco más rápido, en general, que el uso de `NSSortDescriptor` ya que éste se basa en KVC. La principal ventaja del método `NSSortDescriptor` es que proporciona una manera de definir su orden de clasificación utilizando datos, en lugar de código, lo que facilita, por ejemplo, configurar las cosas para que los usuarios puedan ordenar un `NSTableView` haciendo clic en la fila del encabezado.

Conversión entre conjuntos y matrices

```
NSSet *set = [NSSet set];
NSArray *array = [NSArray array];

NSArray *fromSet = [set allObjects];
NSSet *fromArray = [NSSet setWithArray:array];
```

Revertir una matriz

```
NSArray *reversedArray = [myArray.reverseObjectEnumerator allObjects];
```

En bucle

```
NSArray *myColors = @[@"Red", @"Green", @"Blue", @"Yellow"];

// Fast enumeration
// myColors cannot be modified inside the loop
for (NSString *color in myColors) {
    NSLog(@"Element %@", color);
}

// Using indices
for (NSUInteger i = 0; i < myColors.count; i++) {
    NSLog(@"Element %d = %@", i, myColors[i]);
}

// Using block enumeration
[myColors enumerateObjectsUsingBlock:^(id obj, NSUInteger idx, BOOL * stop) {
    NSLog(@"Element %d = %@", idx, obj);

    // To abort use:
    *stop = YES
}];

// Using block enumeration with options
[myColors enumerateObjectsWithOptions:NSEnumerationReverse usingBlock:^(id obj, NSUInteger
idx, BOOL * stop) {
    NSLog(@"Element %d = %@", idx, obj);
}];
```

Usando Genéricos

Para mayor seguridad, podemos definir el tipo de objeto que contiene la matriz:

```
NSArray<NSString *> *colors = @[@"Red", @"Green", @"Blue", @"Yellow"];
NSMutableArray<NSString *> *myColors = [NSMutableArray arrayWithArray:colors];
[myColors addObject:@"Orange"]; // OK
[myColors addObject:[UIColor purpleColor]]; // "Incompatible pointer type" warning
```

Cabe señalar que esto se verifica solo durante el tiempo de compilación.

Enumerar utilizando bloques

```
NSArray *myColors = @[@"Red", @"Green", @"Blue", @"Yellow"];
[myColors enumerateObjectsUsingBlock:^(id obj, NSUInteger idx, BOOL *stop) {
    NSLog(@"enumerating object %@ at index %lu", obj, idx);
}];
```

Al establecer el parámetro de `stop` en `YES` , puede indicar que no se necesita más enumeración. para hacer esto simplemente establece `&stop = YES` .

NSEnumerationOptions
Puede enumerar la matriz en sentido inverso y / o concurrentemente:

```
[myColors enumerateObjectsWithOptions:NSEnumerationConcurrent | NSEnumerationReverse
                              usingBlock:^(id obj, NSUInteger idx, BOOL *stop) {
                                  NSLog(@"enumerating object %@ at index %lu", obj, idx);
                              }];
```

Enumerar subconjunto de matriz

```
NSIndexSet *indexSet = [NSIndexSet indexSetWithIndexesInRange:NSMakeRange(1, 1)];
[myColors enumerateObjectsAtIndexes:indexSet
                            options:kNilOptions
                         usingBlock:^(id obj, NSUInteger idx, BOOL *stop) {
                              NSLog(@"enumerating object %@ at index %lu", obj, idx);
                            }];
```

Comparando arrays

Las matrices se pueden comparar para **igualarlas con el** método acertadamente llamado **isEqualToArray:** que devuelve **SÍ** cuando ambas matrices tienen el mismo número de elementos y cada par pasa una comparación **isEqual :** .

```
NSArray *germanMakes = @[@"Mercedes-Benz", @"BMW", @"Porsche",
                       @"Opel", @"Volkswagen", @"Audi"];
NSArray *sameGermanMakes = [NSArray arrayWithObjects:@"Mercedes-Benz",
                          @"BMW", @"Porsche", @"Opel",
                          @"Volkswagen", @"Audi", nil];

if ([germanMakes isEqualToArray:sameGermanMakes]) {
    NSLog(@"Oh good, literal arrays are the same as NSArrays");
}
```

Lo importante es que cada par debe pasar la prueba isEqual:. Para los objetos personalizados, este método debe implementarse. Existe en el protocolo NSObject.

Añadir objetos a NSArray

```
NSArray *a = @[@1];
a = [a arrayByAddingObject:@2];
a = [a arrayByAddingObjectsFromArray:@[@3, @4, @5]];
```

Estos métodos están optimizados para recrear la nueva matriz de manera muy eficiente, generalmente sin tener que destruir la matriz original o incluso asignar más memoria.

Lea NSArray en línea: https://riptutorial.com/es/objective-c/topic/736/nsarray

Capítulo 25: NSArray

Examples

Creando instancias de NSArray

```
NSArray *array1 = [NSArray arrayWithObjects:@"one", @"two", @"three", nil];
NSArray *array2 = @[@"one", @"two", @"three"];
```

Ordenando matrices

La forma más flexible de ordenar una matriz es con el método sortedArrayUsingComparator:. Esto acepta un **bloque ^ NSComparisonResult (id obj1, id obj2)** .

```
 Return Value            Description
 NSOrderedAscending      obj1 comes before obj2
 NSOrderedSame           obj1 and obj2 have no order
 NSOrderedDescending     obj1 comes after obj2
```

Ejemplo:

```
   NSArray *categoryArray = @[@"Apps", @"Music", @"Songs",
                      @"iTunes", @"Books", @"Videos"];

    NSArray *sortedArray = [categoryArray sortedArrayUsingComparator:
^NSComparisonResult(id obj1, id obj2) {
    if ([obj1 length] < [obj2 length]) {
        return NSOrderedAscending;
    } else if ([obj1 length] > [obj2 length]) {
        return NSOrderedDescending;
    } else {
        return NSOrderedSame;
    }
  }];

 NSLog(@"%@", sortedArray);
```

Filtro NSArray y NSMutableArray

```
NSMutableArray *array =
    [NSMutableArray arrayWithObjects:@"Ken", @"Tim", @"Chris", @"Steve",@"Charlie",@"Melissa",
nil];

NSPredicate *bPredicate =
    [NSPredicate predicateWithFormat:@"SELF beginswith[c] 'c'"];
NSArray *beginWithB =
    [array filteredArrayUsingPredicate:bPredicate];
// beginWith "C" contains { @"Chris", @"Charlie" }.

NSPredicate *sPredicate =
    [NSPredicate predicateWithFormat:@"SELF contains[c] 'a'"];
```

```
[array filterUsingPredicate:sPredicate];
// array now contains { @"Charlie", @"Melissa" }
```

Lea NSArray en línea: https://riptutorial.com/es/objective-c/topic/1181/nsarray

Capítulo 26: NSAttributedString

Examples

Creación de una cadena que tiene kerning personalizado (espacio entre letras)

`NSAttributedString` (y su hermano mutable `NSMutableAttributedString`) le permite crear cadenas que son complejas en su apariencia para el usuario.

Una aplicación común es usar esto para mostrar una cadena y agregar un espaciado de letras / kerning personalizado.

Esto se lograría de la siguiente manera (donde label es una `UILabel`), dando un kerning diferente para la palabra "kerning"

```
NSMutableAttributedString *attributedString;
attributedString = [[NSMutableAttributedString alloc] initWithString:@"Apply kerning"];
[attributedString addAttribute:NSKernAttributeName value:@5 range:NSMakeRange(6, 7)];
[label setAttributedText:attributedString];
```

Crear una cadena con texto golpeado a través

```
NSMutableAttributedString *attributeString = [[NSMutableAttributedString alloc]
initWithString:@"Your String here"];
[attributeString addAttribute:NSStrikethroughStyleAttributeName
                    value:@2
                    range:NSMakeRange(0, [attributeString length])];
```

Uso de Enumerar sobre los atributos en una cadena y subrayar parte de la cadena

```
 NSMutableDictionary *attributesDictionary = [NSMutableDictionary dictionary];
 [attributesDictionary setObject:[UIFont systemFontOfSize:14] forKey:NSFontAttributeName];
 //[attributesDictionary setObject:[UIColor redColor] forKey:NSForegroundColorAttributeName];
 NSMutableAttributedString *attributedString = [[NSMutableAttributedString
alloc]initWithString:@"Google www.google.com link" attributes:attributesDictionary];

 [attributedString enumerateAttribute:(NSString *) NSFontAttributeName
                              inRange:NSMakeRange(0, [attributedString length])

options:NSAttributedStringEnumerationLongestEffectiveRangeNotRequired
                           usingBlock:^(id value, NSRange range, BOOL *stop) {
                               NSLog(@"Attribute: %@, %@", value, NSStringFromRange(range));
                              }];

  NSMutableAttributedString *attributedStr = [[NSMutableAttributedString alloc]
initWithString:@"www.google.com "];

  [attributedString addAttribute:NSUnderlineStyleAttributeName
```

```
                        value:[NSNumber numberWithInt:NSUnderlineStyleDouble]
                        range:NSMakeRange(7, attributedStr.length)];

  [attributedString addAttribute:NSForegroundColorAttributeName
                        value:[UIColor blueColor]
                        range:NSMakeRange(6,attributedStr.length)];

  _attriLbl.attributedText = attributedString;//_attriLbl (of type UILabel) added in
storyboard
```

Salida:

Cómo crear una cadena de tres colores atribuida.

```
 NSMutableAttributedString * string = [[NSMutableAttributedString alloc]
initWithString:@"firstsecondthird"];
[string addAttribute:NSForegroundColorAttributeName value:[UIColor redColor]
range:NSMakeRange(0,5)];
[string addAttribute:NSForegroundColorAttributeName value:[UIColor greenColor]
range:NSMakeRange(5,6)];
[string addAttribute:NSForegroundColorAttributeName value:[UIColor blueColor]
range:NSMakeRange(11,5)];
```

Rango: de principio a fin cadena

Aquí tenemos la primera tercera secuencia, así que en primer lugar hemos establecido el rango (0,5) para que, desde el primer carácter hasta el quinto carácter, se muestre en color verde.

Lea NSAttributedString en línea: https://riptutorial.com/es/objective-c/topic/3725/nsattributedstring

Capítulo 27: NSCache

Examples

NSCache

Lo usas de la misma manera que usarías NSMutableDictionary. La diferencia es que cuando NSCache detecta una presión de memoria excesiva (es decir, está almacenando demasiados valores en la memoria caché) liberará algunos de esos valores para dejar espacio.

Si puede recrear esos valores en tiempo de ejecución (descargando desde Internet, haciendo cálculos, lo que sea), entonces NSCache puede satisfacer sus necesidades. Si los datos no se pueden volver a crear (por ejemplo, es una entrada del usuario, es sensible al tiempo, etc.), entonces no debe almacenarlo en un NSCache porque se destruirá allí.

Lea NSCache en línea: https://riptutorial.com/es/objective-c/topic/8257/nscache

Capítulo 28: NSCalendar

Examples

Información local del sistema

`+currentCalendar` devuelve el calendario lógico para el usuario actual.

```
    NSCalendar *calender = [NSCalendar currentCalendar];
    NSLog(@"%@",calender);
```

`+autoupdatingCurrentCalendar` devuelve el calendario lógico actual para el usuario actual.

```
NSCalendar *calender = [NSCalendar autoupdatingCurrentCalendar];
NSLog(@"%@",calender);
```

Inicializando un calendario

`- initWithCalendarIdentifier:` inicializa un objeto NSCalendar recién asignado para el calendario especificado por un identificador dado.

```
NSCalendar *calender = [[NSCalendar alloc]initWithCalendarIdentifier:@"gregorian"];
NSLog(@"%@",calender);
```

`- setFirstWeekday:` establece el índice del primer día de la semana para el receptor.

```
NSCalendar *calender = [NSCalendar autoupdatingCurrentCalendar];
[calender setFirstWeekday:1];
NSLog(@"%d",[calender firstWeekday]);
```

`- setLocale:` establece la configuración regional del receptor.

```
NSCalendar *calender = [NSCalendar autoupdatingCurrentCalendar];
[calender setLocale:[NSLocale currentLocale]];
NSLog(@"%@",[calender locale]);
```

`- setMinimumDaysInFirstWeek:` establece el número mínimo de días en la primera semana del destinatario.

```
NSCalendar *calender = [NSCalendar autoupdatingCurrentCalendar];
[calender setMinimumDaysInFirstWeek:7];
NSLog(@"%d",[calender minimumDaysInFirstWeek]);
```

`- setTimeZone:` establece la zona horaria del receptor.

```
NSCalendar *calender = [NSCalendar autoupdatingCurrentCalendar];
[calender setTimeZone:[NSTimeZone timeZoneForSecondsFromGMT:0]];
```

```
NSLog(@"%@",[calender timeZone]);
```

Cálculos Calendarios

`- components:fromDate:` devuelve un objeto NSDateComponents que contiene una fecha determinada descompuesta en componentes especificados

```
NSCalendar *calender = [NSCalendar autoupdatingCurrentCalendar];
[calender setTimeZone:[NSTimeZone timeZoneForSecondsFromGMT:0]];
NSLog(@"%@",[calender components:NSCalendarUnitDay fromDate:[NSDate date]]);
NSLog(@"%@",[calender components:NSCalendarUnitYear fromDate:[NSDate date]]);
NSLog(@"%@",[calender components:NSCalendarUnitMonth fromDate:[NSDate date]]);
```

`- components:fromDate:toDate:options:` Devuelve, como un objeto NSDateComponents que usa componentes específicos, la diferencia entre dos fechas proporcionadas.

```
 NSCalendar *calender = [NSCalendar autoupdatingCurrentCalendar];
[calender setTimeZone:[NSTimeZone timeZoneForSecondsFromGMT:0]];
NSLog(@"%@",[calender components:NSCalendarUnitYear fromDate:[NSDate
dateWithTimeIntervalSince1970:0] toDate:[NSDate dateWithTimeIntervalSinceNow:18000]
options:NSCalendarWrapComponents]);
```

`- dateByAddingComponents:toDate:options:` devuelve un nuevo objeto NSDate que representa el tiempo absoluto calculado al agregar componentes dados a una fecha determinada.

```
NSCalendar *calender = [NSCalendar autoupdatingCurrentCalendar];
NSDateComponents *dateComponent = [[NSDateComponents alloc]init];
[dateComponent setYear:10];
NSLog(@"%@",[calender dateByAddingComponents:dateComponent toDate:[NSDate
dateWithTimeIntervalSinceNow:0] options:NSCalendarWrapComponents] );
```

`- dateFromComponents:` devuelve un nuevo objeto NSDate que representa el tiempo absoluto calculado a partir de los componentes dados.

```
NSCalendar *calender = [NSCalendar autoupdatingCurrentCalendar];
NSDateComponents *dateComponent = [[NSDateComponents alloc]init];
[dateComponent setYear:2020];
NSLog(@"%@",[calender dateFromComponents:dateComponent]);
```

Lea NSCalendar en línea: https://riptutorial.com/es/objective-c/topic/2903/nscalendar

Capítulo 29: NSData

Examples

Crear

De NSString:

```
NSString *str = @"Hello world";
NSData *data = [str dataUsingEncoding:NSUTF8StringEncoding];
```

Desde int:

```
int i = 1;
NSData *data = [NSData dataWithBytes: &i length: sizeof(i)];
```

También puede utilizar los siguientes métodos:

```
+ dataWithContentsOfURL:
+ dataWithContentsOfURL:options:error:
+ dataWithData:
- initWithBase64EncodedData:options:
- initWithBase64EncodedString:options:
- initWithBase64Encoding:
- initWithBytesNoCopy:length:
- initWithBytesNoCopy:length:deallocator:
- initWithBytesNoCopy:length:freeWhenDone:
- initWithContentsOfFile:
- initWithContentsOfFile:options:error:
- initWithContentsOfMappedFile:
- initWithContentsOfURL:
- initWithContentsOfURL:options:error:
- initWithData:
```

Obtener NSData longitud

```
NSString *filePath = [[NSFileManager defaultManager] pathForRessorce: @"data" ofType:@"txt"];
NSData *data = [NSData dataWithContentsOfFile:filePath];
int len = [data length];
```

Codificación y decodificación de una cadena usando NSData Base64

Codificación

```
 //Create a Base64 Encoded NSString Object
 NSData *nsdata = [@"iOS Developer Tips encoded in Base64"
dataUsingEncoding:NSUTF8StringEncoding];

 // Get NSString from NSData object in Base64
```

```
 NSString *base64Encoded = [nsdata base64EncodedStringWithOptions:0];
 // Print the Base64 encoded string
 NSLog(@"Encoded: %@", base64Encoded);
```

Descodificación:

```
// NSData from the Base64 encoded str
NSData *nsdataFromBase64String = [[NSData alloc]initWithBase64EncodedString:base64Encoded
options:0];

// Decoded NSString from the NSData
NSString *base64Decoded = [[NSString alloc] initWithData:nsdataFromBase64String
encoding:NSUTF8StringEncoding];
NSLog(@"Decoded: %@", base64Decoded);
```

NSData y cadena hexadecimal

Obtener NSData de cadena hexadecimal

```
+ (NSData *)dataFromHexString:(NSString *)string
{
    string = [string lowercaseString];
    NSMutableData *data= [NSMutableData new];
    unsigned char whole_byte;
    char byte_chars[3] = {'\0','\0','\0'};
    int i = 0;
    int length = (int) string.length;
    while (i < length-1) {
        char c = [string characterAtIndex:i++];
        if (c < '0' || (c > '9' && c < 'a') || c > 'f')
            continue;
        byte_chars[0] = c;
        byte_chars[1] = [string characterAtIndex:i++];
        whole_byte = strtol(byte_chars, NULL, 16);
        [data appendBytes:&whole_byte length:1];
    }
    return data;
}
```

Obtener cadena hexadecimal de los datos:

```
+ (NSString *)hexStringForData:(NSData *)data
{
    if (data == nil) {
        return nil;
    }

    NSMutableString *hexString = [NSMutableString string];

    const unsigned char *p = [data bytes];

    for (int i=0; i < [data length]; i++) {
        [hexString appendFormat:@"%02x", *p++];
    }

    return hexString;
```

```
}
```

Lea NSData en línea: https://riptutorial.com/es/objective-c/topic/1532/nsdata

Capítulo 30: NSDate

Observaciones

`NSDate` es un tipo de valor muy simple, que representa un momento exacto en el tiempo universal. No contiene información sobre zonas horarias, horario de verano, calendarios o configuración regional.

`NSDate` es realmente solo una envoltura inmutable alrededor de un `NSTimeInterval` que es un `double`. No hay una subclase mutable, como ocurre con otros tipos de valores en Foundation.

Examples

Creando un NSDate

La clase `NSDate` proporciona métodos para crear objetos `NSDate` correspondientes a una fecha y hora determinadas. Se puede inicializar un `NSDate` utilizando el inicializador designado, que:

> Devuelve un objeto `NSDate` inicializado relativo a las 00:00:00 UTC del 1 de enero de 2001 en un número determinado de segundos.

```
NSDate *date = [[NSDate alloc] initWithTimeIntervalSinceReferenceDate:100.0];
```

`NSDate` también proporciona una manera fácil de crear un `NSDate` igual a la fecha y hora actuales:

```
NSDate *now = [NSDate date];
```

También es posible crear un `NSDate` una cantidad determinada de segundos a partir de la fecha y hora actuales:

```
NSDate *tenSecondsFromNow = [NSDate dateWithTimeIntervalSinceNow:10.0];
```

Comparación de fechas

Hay 4 métodos para comparar `NSDate` s en Objective-C:

- `- (BOOL)isEqualToDate:(NSDate *)anotherDate`
- `- (NSDate *)earlierDate:(NSDate *)anotherDate`
- `- (NSDate *)laterDate:(NSDate *)anotherDate`
- `- (NSComparisonResult)compare:(NSDate *)anotherDate`

Considere el siguiente ejemplo utilizando 2 fechas, fecha de `NSDate date1 = July 7, 2016` y `NSDate date2 = July 2, 2016` :

```
NSDateComponents *comps1 = [[NSDateComponents alloc]init];
comps.year = 2016;
```

```
comps.month = 7;
comps.day = 7;

NSDateComponents *comps2 = [[NSDateComponents alloc]init];
    comps.year = 2016;
    comps.month = 7;
    comps.day = 2;

NSDate* date1 = [calendar dateFromComponents:comps1]; //Initialized as July 7, 2016
NSDate* date2 = [calendar dateFromComponents:comps2]; //Initialized as July 2, 2016
```

Ahora que se han creado los `NSDate` , se pueden comparar:

```
if ([date1 isEqualToDate:date2]) {
    //Here it returns false, as both dates are not equal
}
```

También podemos usar los `earlierDate:` y `laterDate:` de la clase `NSDate` :

```
NSDate *earlierDate = [date1 earlierDate:date2];//Returns the earlier of 2 dates. Here
earlierDate will equal date2.
NSDate *laterDate = [date1 laterDate:date2];//Returns the later of 2 dates. Here laterDate
will equal date1.
```

Por último, podemos usar el `NSDate` de `compare: NSDate` :

```
NSComparisonResult result = [date1 compare:date2];
    if (result == NSOrderedAscending) {
        //Fails
        //Comes here if date1 is earlier than date2. In our case it will not come here.
    }else if (result == NSOrderedSame){
        //Fails
        //Comes here if date1 is the same as date2. In our case it will not come here.
    }else{//NSOrderedDescending

        //Succeeds
        //Comes here if date1 is later than date2. In our case it will come here
    }
```

Convierta NSDate que se compone de hora y minuto (solo) a un NSDate completo

Hay muchos casos en los que uno ha creado una fecha de emisión (NSDate) a partir de solo un formato de hora y minuto, es decir: 08:12

El inconveniente de esta situación es que su NSDate está casi completamente "desnudo" y lo que debe hacer es crear: día, mes, año, segundo y huso horario para que este objeto "siga el ritmo" con otros tipos de NSDate.

En aras del ejemplo, digamos que hourAndMinute es el tipo NSDate que se compone de formato de hora y minuto:

```
NSDateComponents *hourAndMintuteComponents = [calendar components:NSCalendarUnitHour |
```

```
NSCalendarUnitMinute
                                                        fromDate:hourAndMinute];
NSDateComponents *componentsOfDate = [[NSCalendar currentCalendar]
components:NSCalendarUnitDay | NSCalendarUnitMonth | NSCalendarUnitYear
                                                                fromDate:[NSDate date]];

NSDateComponents *components = [[NSDateComponents alloc] init];
[components setDay: componentsOfDate.day];
[components setMonth: componentsOfDate.month];
[components setYear: componentsOfDate.year];
[components setHour: [hourAndMintuteComponents hour]];
[components setMinute: [hourAndMintuteComponents minute]];
[components setSecond: 0];
[calendar setTimeZone: [NSTimeZone defaultTimeZone]];

NSDate *yourFullNSDateObject = [calendar dateFromComponents:components];
```

Ahora tu objeto es el opuesto total de estar "desnudo".

Convertir NSDate a NSString

Si ww tiene el objeto NSDate, y queremos convertirlo en NSString. Hay diferentes tipos de cadenas de fecha. ¿Cómo podemos hacer eso ?, es muy simple. Sólo 3 pasos.

1. Crear objeto NSDateFormatter

```
NSDateFormatter *dateFormatter = [[NSDateFormatter alloc] init];
```

2. Establece el formato de fecha en el que quieres tu cadena.

```
dateFormatter.dateFormat = @"yyyy-MM-dd 'at' HH:mm";
```

3. Ahora, consigue la cadena formateada

```
NSDate *date = [NSDate date]; // your NSDate object
NSString *dateString = [dateFormatter stringFromDate:date];
```

Esto dará salida a algo como esto: `2001-01-02 at 13:00`

Nota:

Crear una instancia de `NSDateFormatter` es una operación costosa, por lo que se recomienda crearla una vez y reutilizarla cuando sea posible.

Lea NSDate en línea: https://riptutorial.com/es/objective-c/topic/1981/nsdate

Capítulo 31: NSDiccionario

Examples

Crear

```
NSDictionary *dict = [[NSDictionary alloc] initWithObjectsAndKeys:@"value1", @"key1",
@"value2", @"key2", nil];
```

o

```
NSArray *keys = [NSArray arrayWithObjects:@"key1", @"key2", nil];
NSArray *objects = [NSArray arrayWithObjects:@"value1", @"value2", nil];
NSDictionary *dictionary = [NSDictionary dictionaryWithObjects:objects
                                                       forKeys:keys];
```

o usando la sintaxis literal apropiada

```
NSDictionary *dict = @{@"key": @"value", @"nextKey": @"nextValue"};
```

NSDictionary to NSArray

```
NSDictionary *myDictionary = [[NSDictionary alloc] initWithObjectsAndKeys:@"value1", @"key1",
@"value2", @"key2", nil];
NSArray *copiedArray = myDictionary.copy;
```

Obtener llaves:

```
NSArray *keys = [myDictionary allKeys];
```

Obtener valores:

```
NSArray *values = [myDictionary allValues];
```

NSDictionary a NSData

```
NSDictionary *myDictionary = [[NSDictionary alloc] initWithObjectsAndKeys:@"value1", @"key1",
@"value2", @"key2", nil];
NSData *myData = [NSKeyedArchiver archivedDataWithRootObject:myDictionary];
```

Ruta de reserva:

```
NSDictionary *myDictionary = (NSDictionary*) [NSKeyedUnarchiver
unarchiveObjectWithData:myData];
```

NSDictionary a JSON

```
NSDictionary *myDictionary = [[NSDictionary alloc] initWithObjectsAndKeys:@"value1", @"key1",
@"value2", @"key2", nil];

NSMutableDictionary *mutableDictionary = [myDictionary mutableCopy];
NSData *data = [NSJSONSerialization dataWithJSONObject:myDictionary
options:NSJSONWritingPrettyPrinted error:nil];
NSString *jsonString = [[NSString alloc] initWithData:data encoding:NSUTF8StringEncoding];
```

Enumeración basada en bloques

La enumeración de diccionarios le permite ejecutar un bloque de código en cada par de clave-valor del diccionario utilizando el método `enumerateKeysAndObjectsUsingBlock:(void (^)(id key, id obj, BOOL *stop))block`

Ejemplo:

```
NSDictionary stockSymbolsDictionary = @{
                                        @"AAPL": @"Apple",
                                        @"GOOGL": @"Alphabet",
                                        @"MSFT": @"Microsoft",
                                        @"AMZN": @"Amazon"
                                      };
NSLog(@"Printing contents of dictionary via enumeration");
[stockSymbolsDictionary enumerateKeysAndObjectsUsingBlock:^(id key, id obj, BOOL *stop) {
    NSLog(@"Key: %@, Value: %@", key, obj);
}];
```

Enumeración rápida

`NSDictionary` se puede enumerar utilizando la enumeración rápida, al igual que otros tipos de colección:

```
NSDictionary stockSymbolsDictionary = @{
                                       @"AAPL": @"Apple",
                                       @"GOOGL": @"Alphabet",
                                       @"MSFT": @"Microsoft",
                                       @"AMZN": @"Amazon"
                                     };

for (id key in stockSymbolsDictionary)
{
    id value = dictionary[key];
    NSLog(@"Key: %@, Value: %@", key, value);
}
```

Debido a que `NSDictionary` está intrínsecamente desordenado, el orden de las claves que en el ciclo for no está garantizado.

Lea NSDiccionario en línea: https://riptutorial.com/es/objective-c/topic/847/nsdiccionario

Capítulo 32: NSDiccionario

Sintaxis

- @{ valor clave, ... }
- [NSDictionary dictionaryWithObjectsAndKeys: value, key, ..., nil];
- dict [*clave*] = *valor* ;
- *valor de* id = dict [*clave*];

Observaciones

La clase NSDictionary declara la interfaz programática a los objetos que administran asociaciones inmutables de claves y valores. Utilice esta clase o su subclase NSMutableDictionary cuando necesite una forma conveniente y eficiente de recuperar datos asociados con una clave arbitraria. NSDictionary crea diccionarios estáticos, y NSMutableDictionary crea diccionarios dinámicos. (Por conveniencia, el término diccionario se refiere a cualquier instancia de una de estas clases sin especificar su pertenencia a la clase exacta).

Un par clave-valor dentro de un diccionario se llama una entrada. Cada entrada consta de un objeto que representa la clave y un segundo objeto que es el valor de esa clave. Dentro de un diccionario, las claves son únicas. Es decir, no hay dos claves en un solo diccionario iguales (según lo determinado por isEqual :). En general, una clave puede ser cualquier objeto (siempre que cumpla con el protocolo NSCopying; consulte a continuación), pero tenga en cuenta que cuando se utiliza la codificación de clave-valor, la clave debe ser una cadena (consulte Fundamentos de codificación de clave-valor). Ni una clave ni un valor pueden ser nulos; Si necesita representar un valor nulo en un diccionario, debe usar NSNull.

NSDictionary es un "puente gratuito" con su contraparte de Core Foundation, CFDictionaryRef. Consulte la sección Puente sin cargo para obtener más información sobre el puente sin cargo.

Examples

Creando utilizando literales.

```
NSDictionary *inventory = @{
    @"Mercedes-Benz SLK250" : @(13),
    @"BMW M3 Coupe" : @(self.BMWM3CoupeInventory.count),
    @"Last Updated" : @"Jul 21, 2016",
    @"Next Update"  : self.nextInventoryUpdateString
};
```

Creación utilizando dictionaryWithObjectsAndKeys:

```
NSDictionary *inventory = [NSDictionary dictionaryWithObjectsAndKeys:
    [NSNumber numberWithInt:13], @"Mercedes-Benz SLK250",
    [NSNumber numberWithInt:22], @"Mercedes-Benz E350",
    [NSNumber numberWithInt:19], @"BMW M3 Coupe",
    [NSNumber numberWithInt:16], @"BMW X6",
    nil];
```

`nil` debe pasarse como el último parámetro como centinela que indica el final.

Es importante recordar que al crear diccionarios de esta manera, los valores van primero y luego las claves. En el ejemplo anterior, las cadenas son las claves y los números son los valores. El nombre del método refleja esto también: `dictionaryWithObjectsAndKeys` . Si bien esto no es incorrecto, se prefiere la forma más moderna de instanciar diccionarios (con literales).

Creando utilizando plistas

```
NSString *pathToPlist = [[NSBundle mainBundle] pathForResource:@"plistName"
    ofType:@"plist"];
NSDictionary *plistDict = [[NSDictionary alloc] initWithContentsOfFile:pathToPlist];
```

Estableciendo un valor en NSDictionary

Hay varias formas de establecer el objeto de una clave en un NSDictionary, correspondiente a las formas en que obtiene un valor. Por ejemplo, para agregar un lamborghini a una lista de coches

Estándar

```
[cars setObject:lamborghini forKey:@"Lamborghini"];
```

Al igual que cualquier otro objeto, llame al método de NSDictionary que establece un objeto de una clave, `objectForKey:` Tenga cuidado de no confundir esto con `setValue:forKey:` Eso es para una cosa completamente diferente, Key Value Coding

Taquigrafía

```
cars[@"Lamborghini"] = lamborghini;
```

Esta es la sintaxis que utiliza para los diccionarios en la mayoría de los otros idiomas, como C #, Java y Javascript. Es mucho más conveniente que la sintaxis estándar, y posiblemente más legible (especialmente si codifica en estos otros idiomas), pero por supuesto, no es *estándar* . También está disponible solo en las versiones más recientes de Objective C

Obtener un valor de NSDictionary

Hay varias formas de obtener un objeto de un NSDictionary con una clave. Por ejemplo, para obtener un lamborghini de una lista de coches

Estándar

```
Car * lamborghini = [cars objectForKey:@"Lamborghini"];
```

Al igual que cualquier otro objeto, llame al método de NSDictionary que le da un objeto para una clave, `objectForKey:` Tenga cuidado de no confundir esto con `valueForKey:` Eso es para una cosa completamente diferente, Key Value Coding

Taquigrafía

```
Car * lamborghini = cars[@"Lamborghini"];
```

Esta es la sintaxis que utiliza para los diccionarios en la mayoría de los otros idiomas, como C #, Java y Javascript. Es mucho más conveniente que la sintaxis estándar, y posiblemente más legible (especialmente si codifica en estos otros idiomas), pero por supuesto, no es *estándar* . También está disponible solo en las versiones más recientes de Objective C

Compruebe si NSDictionary ya tiene una clave o no

C objetivo:

```
   //this is the dictionary you start with.
 NSDictionary *dict = [NSDictionary dictionaryWithObjectsAndKeys:@"name1", @"Sam",@"name2",
@"Sanju",nil];

//check if the dictionary contains the key you are going to modify.  In this example, @"Sam"
if (dict[@"name1"] != nil) {
    //there is an entry for Key name1
}
else {
    //There is no entry for name1
}
```

Lea NSDiccionario en línea: https://riptutorial.com/es/objective-c/topic/875/nsdiccionario

Capítulo 33: NSJSONSerialización

Sintaxis

- (id) JSONObjectWithData: (NSData *) opciones de datos: (NSJSONReadingOptions) opt error: (NSError * _Nullable *) error

Parámetros

Operador	Descripción
datos	Un objeto de datos que contiene datos JSON
optar	Opciones para leer los datos JSON y crear los objetos de Foundation.
error	Si se produce un error, al regresar contiene un objeto NSError que describe el problema.

Observaciones

NSJSONSerialization está disponible en iOS 5.0 y versiones posteriores. Un objeto que se puede convertir a JSON debe tener las siguientes propiedades:

- El objeto de nivel superior es un NSArray o NSDictionary.
- Todos los objetos son instancias de NSString, NSNumber, NSArray, NSDictionary o NSNull.
- Todas las claves del diccionario son instancias de NSString.
- Los números no son NaN o infinito.

Examples

Análisis JSON utilizando NSJSONSerialization Objective c

```
NSError *e = nil;
NSString *jsonString = @"[{\"id\": \"1\", \"name\":\"sam\"}]";
NSData *data = [jsonString dataUsingEncoding:NSUTF8StringEncoding];

NSArray *jsonArray = [NSJSONSerialization JSONObjectWithData: data options:
NSJSONReadingMutableContainers error: &e];

if (!jsonArray) {
    NSLog(@"Error parsing JSON: %@", e);
} else {
    for(NSDictionary *item in jsonArray) {
        NSLog(@"Item: %@", item);
```

```
    }
}
```

Salida:

```
Item: {
    id = 1;
    name = sam;
}
```

Ejemplo 2: Uso de contenidos de url:

```
//Parsing:

NSData *data = [NSData dataWithContentsOfURL:@"URL HERE"];
NSError *error;
NSDictionary *json = [NSJSONSerialization JSONObjectWithData:data options:kNilOptions
error:&error];
NSLog(@"json :%@",json);
```

Respuesta de muestra:

```
json: {
    MESSAGE = "Test Message";
    RESPONSE =(
                {
            email = "test@gmail.com";
            id = 15;
            phone = 1234567890;
            name = Staffy;
        }
    );
    STATUS = SUCCESS;
}


 NSMutableDictionary *response = [[[json valueForKey:@"RESPONSE"]
objectAtIndex:0]mutableCopy];
 NSString *nameStr = [response valueForKey:@"name"];
 NSString *emailIdStr = [response valueForKey:@"email"];
```

Lea NSJSONSerialización en línea: https://riptutorial.com/es/objective-c/topic/2587/nsjsonserializacion

Capítulo 34: NSMutableArray

Examples

Añadiendo elementos

```
NSMutableArray *myColors;
myColors = [NSMutableArray arrayWithObjects: @"Red", @"Green", @"Blue", @"Yellow", nil];
[myColors addObject: @"Indigo"];
[myColors addObject: @"Violet"];

//Add objects from an NSArray
NSArray *myArray = @[@"Purple",@"Orange"];
[myColors addObjectsFromArray:myArray];
```

Insertar elementos

```
NSMutableArray *myColors;
int i;
int count;
myColors = [NSMutableArray arrayWithObjects: @"Red", @"Green", @"Blue", @"Yellow", nil];
[myColors insertObject: @"Indigo" atIndex: 1];
[myColors insertObject: @"Violet" atIndex: 3];
```

Borrando elementos

Eliminar en el índice específico:

```
[myColors removeObjectAtIndex: 3];
```

Eliminar la primera instancia de un objeto específico:

```
[myColors removeObject: @"Red"];
```

Eliminar todas las instancias de un objeto específico:

```
[myColors removeObjectIdenticalTo: @"Red"];
```

Eliminar todos los objetos:

```
[myColors removeAllObjects];
```

Eliminar el último objeto:

```
[myColors removeLastObject];
```

Ordenando matrices

```
NSMutableArray *myColors = [NSMutableArray arrayWithObjects: @"red", @"green", @"blue",
@"yellow", nil];
NSArray *sortedArray;
sortedArray = [myColors sortedArrayUsingSelector:@selector(localizedCaseInsensitiveCompare:)];
```

Mueve el objeto a otro índice

Mueve *azul* al principio de la matriz:

```
NSMutableArray *myColors = [NSMutableArray arrayWithObjects: @"Red", @"Green", @"Blue",
@"Yellow", nil];

NSUInteger fromIndex = 2;
NSUInteger toIndex = 0;

id blue = [[[self.array objectAtIndex:fromIndex] retain] autorelease];
[self.array removeObjectAtIndex:fromIndex];
[self.array insertObject:blue atIndex:toIndex];
```

`myColors` ahora es `[@"Blue", @"Red", @"Green", @"Yellow"]` .

Filtrado de contenido de matriz con Predicado

Uso de **filterUsingPredicate:** esto evalúa un predicado dado en comparación con el contenido de las matrices y devuelve los objetos que coinciden.

Ejemplo:

```
      NSMutableArray *array = [NSMutableArray array];
      [array setArray:@[@"iOS",@"macOS",@"tvOS"]];
      NSPredicate *predicate = [NSPredicate predicateWithFormat:@"SELF beginswith[c] 'i'"];
      NSArray *resultArray = [array filteredArrayUsingPredicate:predicate];
      NSLog(@"%@",resultArray);
```

Creando un NSMutableArray

`NSMutableArray` se puede inicializar como una matriz vacía como esta:

```
NSMutableArray *array = [[NSMutableArray alloc] init];
// or
NSMutableArray *array2 = @[].mutableCopy;
// or
NSMutableArray *array3 = [NSMutableArray array];
```

`NSMutableArray` se puede inicializar con otra matriz como esta:

```
NSMutableArray *array4 = [[NSMutableArray alloc] initWithArray:anotherArray];
// or
NSMutableArray *array5 = anotherArray.mutableCopy;
```

Lea NSMutableArray en línea: https://riptutorial.com/es/objective-c/topic/2008/nsmutablearray

Capítulo 35: NSMutableDictionary

Parámetros

objetos	llaves
Una matriz que contiene los valores para el nuevo diccionario.	CellAn matriz que contiene las claves para el nuevo diccionario. Cada clave se copia y la copia se agrega al diccionario.

Examples

Ejemplo de NSMutableDictionary

+ diccionario con capacidad:

Crea y devuelve un diccionario mutable, inicialmente otorgándole suficiente memoria asignada para contener un número dado de entradas.

```
NSMutableDictionary *dict =  [NSMutableDictionary dictionaryWithCapacity:1];
NSLog(@"%@",dict);
```

- en eso

Inicializa un diccionario mutable recién asignado.

```
NSMutableDictionary *dict =  [[NSMutableDictionary alloc] init];
NSLog(@"%@",dict);
```

+ dictionaryWithSharedKeySet:

Crea un diccionario mutable que está optimizado para tratar con un conjunto conocido de claves.

```
id sharedKeySet = [NSDictionary sharedKeySetForKeys:@[@"key1", @"key2"]]; // returns
NSSharedKeySet
NSMutableDictionary *dict = [NSMutableDictionary dictionaryWithSharedKeySet:sharedKeySet];
dict[@"key1"] = @"Easy";
dict[@"key2"] = @"Tutorial";
//We can an object thats not in the shared keyset
dict[@"key3"] = @"Website";
NSLog(@"%@",dict);
```

SALIDA

```
{
    key1 = Eezy;
    key2 = Tutorials;
```

```
    key3 = Website;
}
```

Adición de entradas a un diccionario mutable

- setObject: forKey:

Agrega un par clave-valor dado al diccionario.

```
NSMutableDictionary *dict =  [NSMutableDictionary dictionary];
[dict setObject:@"Easy" forKey:@"Key1"];
NSLog(@"%@",dict);
```

SALIDA

```
{
    Key1 = Eezy;
}
```

- setObject: forKeyedSubscript:

Agrega un par clave-valor dado al diccionario.

```
NSMutableDictionary *dict =  [NSMutableDictionary dictionary];
[dict setObject:@"Easy" forKeyedSubscript:@"Key1"];
NSLog(@"%@",dict);
```

SALIDA {Clave1 = Fácil; }

Eliminar entradas de un diccionario mutable

- removeObjectForKey:

Elimina una clave dada y su valor asociado del diccionario.

```
NSMutableDictionary *dict =  [NSMutableDictionary
dictionaryWithDictionary:@{@"key1":@"Easy",@"key2": @"Tutorials"}];
[dict removeObjectForKey:@"key1"];
NSLog(@"%@",dict);
```

SALIDA

```
{
    key2 = Tutorials;
}
```

- removeAllObjects

Vacía el diccionario de sus entradas.

```
NSMutableDictionary *dict =  [NSMutableDictionary
dictionaryWithDictionary:@{@"key1":@"Eezy",@"key2": @"Tutorials"}];
[dict removeAllObjects];
NSLog(@"%@",dict);
```

SALIDA

```
{
}
```

- removeObjectsForKeys:

Se elimina de las entradas del diccionario especificadas por elementos en una matriz dada.

```
NSMutableDictionary *dict =  [NSMutableDictionary
dictionaryWithDictionary:@{@"key1":@"Easy",@"key2": @"Tutorials"}];
[dict removeObjectsForKeys:@[@"key1"]];
NSLog(@"%@",dict);
```

SALIDA

```
{
    key2 = Tutorials;
}
```

Lea NSMutableDictionary en línea: https://riptutorial.com/es/objective-c/topic/3103/nsmutabledictionary

Capítulo 36: NSObject

Introducción

`NSObject` es la clase raíz de `Cocoa` , sin embargo, el lenguaje `Objective-C` en sí mismo no define ninguna clase raíz en absoluto. `Cocoa` , el Framework de Apple. El sistema de tiempo de ejecución y la capacidad de comportarse como objetos Objective-C.

Esta clase tiene todas las propiedades básicas del objeto de clase `Objective'C` como:

yo.

clase (nombre de la clase).

superclase (superclase de la clase actual).

Sintaxis

- yo
- superclase
- en eso
- asignar
- nuevo
- es igual
- isKindOfClass
- isMemberOfClass
- descripción

Examples

NSObject

`@interface NSString : NSObject` (`NSObject` es una clase base de la clase NSString).

Puede usar los siguientes métodos para la asignación de la clase de cadena:

```
- (instancetype)init

+ (instancetype)new

+ (instancetype)alloc
```

Para copiar cualquier objeto:

```
- (id)copy;
```

```
- (id)mutableCopy;
```

Para comparar objetos:

```
- (BOOL)isEqual:(id)object
```

Para obtener superclase de la clase actual:

```
superclass
```

¿Para comprobar qué clase de clase es esta?

```
- (BOOL)isKindOfClass:(Class)aClass
```

Algunas propiedades de las clases NO ARC:

```
- (instancetype)retain OBJC_ARC_UNAVAILABLE;

- (oneway void)release OBJC_ARC_UNAVAILABLE;

- (instancetype)autorelease OBJC_ARC_UNAVAILABLE;

- (NSUInteger)retainCount
```

Lea NSObject en línea: https://riptutorial.com/es/objective-c/topic/9355/nsobject

Capítulo 37: NSPredicate

Sintaxis

- Operador CONTAINS: Permite filtrar objetos con subconjunto coincidente.

```
NSPredicate *filterByName = [NSPredicate predicateWithFormat:@"self.title CONTAINS[cd] %@",@"Tom"];
```

- LIKE: Su filtro de comparación simple.

```
NSPredicate *filterByNameCIS = [NSPredicate predicateWithFormat:@"self.title LIKE[cd] %@",@"Tom and Jerry"];
```

- = operador: devuelve todos los objetos con valor de filtro coincidente.

```
NSPredicate *filterByNameCS = [NSPredicate predicateWithFormat:@"self.title = %@",@"Tom and Jerry"];
```

- Operador IN: Le permite filtrar objetos con un conjunto de filtros específico.

```
NSPredicate *filterByIds = [NSPredicate predicateWithFormat:@"self.id IN %@",@[@"7CDF6D22-8D36-49C2-84FE-E31EECCECB79", @"7CDF6D22-8D36-49C2-84FE-E31EECCECB76"]];
```

- Operador NO EN: Le permite encontrar objetos Inversos con un conjunto específico.

```
NSPredicate *filterByNotInIds = [NSPredicate predicateWithFormat:@"NOT (self.id IN %@)",@[@"7CDF6D22-8D36-49C2-84FE-E31EECCECB79", @"7CDF6D22-8D36-49C2-84FE-E31EECCECB76"]];
```

Observaciones

Para más detalles, lea NSPredicate en la documentación de Apple.

Examples

Filtrar por nombre

```
NSArray *array = @[
                    @{
                        @"id": @"7CDF6D22-8D36-49C2-84FE-E31EECCECB71",
                        @"title": @"Jackie Chan Strike Movie",
                        @"url": @"http://abc.com/playback.m3u8",
                        @"thumbnailURL": @"http://abc.com/thumbnail.png",
                        @"isMovie" : @1
                    },
                    @{
                        @"id": @"7CDF6D22-8D36-49C2-84FE-E31EECCECB72",
                        @"title": @"Sherlock homes",
                        @"url": @"http://abc.com/playback.m3u8",
                        @"thumbnailURL": @"http://abc.com/thumbnail.png",
                        @"isMovie" : @0
                    },
```

```
                    @{
                        @"id": @"7CDF6D22-8D36-49C2-84FE-E31EECCECB73",
                        @"title": @"Titanic",
                        @"url": @"http://abc.com/playback.m3u8",
                        @"thumbnailURL": @"http://abc.com/thumbnail.png",
                        @"isMovie" : @1
                    },
                    @{
                        @"id": @"7CDF6D22-8D36-49C2-84FE-E31EECCECB74",
                        @"title": @"Star Wars",
                        @"url": @"http://abc.com/playback.m3u8",
                        @"thumbnailURL": @"http://abc.com/thumbnail.png",
                        @"isMovie" : @1
                    },
                    @{
                        @"id": @"7CDF6D22-8D36-49C2-84FE-E31EECCECB75",
                        @"title": @"Pokemon",
                        @"url": @"http://abc.com/playback.m3u8",
                        @"thumbnailURL": @"http://abc.com/thumbnail.png",
                        @"isMovie" : @0
                    },
                    @{
                        @"id": @"7CDF6D22-8D36-49C2-84FE-E31EECCECB76",
                        @"title": @"Avatar",
                        @"url": @"http://abc.com/playback.m3u8",
                        @"thumbnailURL": @"http://abc.com/thumbnail.png",
                        @"isMovie" : @1
                    },
                    @{
                        @"id": @"7CDF6D22-8D36-49C2-84FE-E31EECCECB77",
                        @"title": @"Popey",
                        @"url": @"http://abc.com/playback.m3u8",
                        @"thumbnailURL": @"http://abc.com/thumbnail.png",
                        @"isMovie" : @1
                    },
                    @{
                        @"id": @"7CDF6D22-8D36-49C2-84FE-E31EECCECB78",
                        @"title": @"Tom and Jerry",
                        @"url": @"http://abc.com/playback.m3u8",
                        @"thumbnailURL": @"http://abc.com/thumbnail.png",
                        @"isMovie" : @1
                    },
                    @{
                        @"id": @"7CDF6D22-8D36-49C2-84FE-E31EECCECB79",
                        @"title": @"The wolf",
                        @"url": @"http://abc.com/playback.m3u8",
                        @"thumbnailURL": @"http://abc.com/thumbnail.png",
                        @"isMovie" : @1
                    }
                    ];

// *** Case Insensitive comparision with excate title match ***
NSPredicate *filterByNameCIS = [NSPredicate predicateWithFormat:@"self.title LIKE[cd]
%@",@"Tom and Jerry"];
NSLog(@"Filter By Name(CIS) : %@",[array filteredArrayUsingPredicate:filterByNameCIS]);
```

Buscar películas, excepto los identificadores dados

```
// *** Find movies except given ids ***
```

```
NSPredicate *filterByNotInIds = [NSPredicate predicateWithFormat:@"NOT (self.id IN
%@)",@[@"7CDF6D22-8D36-49C2-84FE-E31EECCECB79", @"7CDF6D22-8D36-49C2-84FE-E31EECCECB76"]];
NSLog(@"Filter movies except given Ids : %@",[array
filteredArrayUsingPredicate:filterByNotInIds]);
```

Encuentra todos los objetos de tipo película.

```
// *** Find all the objects which is of type movie, Both the syntax are valid ***
NSPredicate *filterByMovieType = [NSPredicate predicateWithFormat:@"self.isMovie = %@",@1];
// OR
//NSPredicate *filterByMovieType = [NSPredicate predicateWithFormat:@"self.isMovie =
%@",[NSNumber numberWithBool:YES]];
NSLog(@"Filter By Movie Type : %@",[array filteredArrayUsingPredicate:filterByMovieType]);
```

Encuentra identificadores de objetos distintos de la matriz

```
// *** Find Distinct object ids of array ***
NSLog(@"Distinct id : %@",[array valueForKeyPath:@"@distinctUnionOfObjects.id"]);
```

Encuentra películas con identificadores específicos

```
// *** Find movies with specific ids ***
NSPredicate *filterByIds = [NSPredicate predicateWithFormat:@"self.id IN %@",@[@"7CDF6D22-
8D36-49C2-84FE-E31EECCECB79", @"7CDF6D22-8D36-49C2-84FE-E31EECCECB76"]];
NSLog(@"Filter By Ids : %@",[array filteredArrayUsingPredicate:filterByIds]);
```

Comparación entre mayúsculas y minúsculas con el título exacto

```
// *** Case Insensitive comparison with exact title match ***
NSPredicate *filterByNameCIS = [NSPredicate predicateWithFormat:@"self.title LIKE[cd]
%@",@"Tom and Jerry"];
NSLog(@"Filter By Name(CIS) : %@",[array filteredArrayUsingPredicate:filterByNameCIS]);
```

Caso sensible con coincidencia de título exacto

```
// *** Case sensitive with exact title match ***
NSPredicate *filterByNameCS = [NSPredicate predicateWithFormat:@"self.title = %@",@"Tom and
Jerry"];
NSLog(@"Filter By Name(CS) : %@",[array filteredArrayUsingPredicate:filterByNameCS]);
```

Comparación entre mayúsculas y minúsculas con subconjunto coincidente

```
// *** Case Insensitive comparison with matching subset ***
NSPredicate *filterByName = [NSPredicate predicateWithFormat:@"self.title CONTAINS[cd]
%@",@"Tom"];
NSLog(@"Filter By Containing Name : %@",[array filteredArrayUsingPredicate:filterByName]);
```

Lea NSPredicate en línea: https://riptutorial.com/es/objective-c/topic/2004/nspredicate

Capítulo 38: NSRegularExpression

Sintaxis

- NSRegularExpression * regex = [NSRegularExpression regularExpressionWithPattern: opciones de PATTERN: error de OPCIONES: ERROR];
- NSArray <NSTextCheckingResult *> * results = [regex matchesInString: STRING options: OPTIONS range: RANGE_IN_STRING];
- NSInteger numberOfMatches = [regex numberOfMatchesInString: opciones de STRING: rango de OPCIONES: RANGE_IN_STRING];

Examples

Encuentra todos los números en una cadena

```
NSString *testString = @"There are 42 sheep and 8672 cows.";
NSError *error = nil;
NSRegularExpression *regex = [NSRegularExpression regularExpressionWithPattern:@"(\\d+)"

options:NSRegularExpressionCaseInsensitive
                                                                        error:&error];

NSArray *matches = [regex matchesInString:testString
                                  options:0
                                    range:NSMakeRange(0, testString.length)];

for (NSTextCheckingResult *matchResult in matches) {
    NSString* match = [testString substringWithRange:matchResult.range];
    NSLog(@"match: %@", match);
}
```

La salida será `match: 42` y `match: 8672` .

Compruebe si una cadena coincide con un patrón

```
NSString *testString1 = @"(555) 123-5678";
NSString *testString2 = @"not a phone number";

NSError *error = nil;
NSRegularExpression *regex = [NSRegularExpression regularExpressionWithPattern:@"^\\(\\d{3}\\)
\\d{3}\\-\\d{4}$"

options:NSRegularExpressionCaseInsensitive error:&error];

NSInteger result1 = [regex numberOfMatchesInString:testString1 options:0 range:NSMakeRange(0,
testString1.length)];
NSInteger result2 = [regex numberOfMatchesInString:testString2 options:0 range:NSMakeRange(0,
testString2.length)];

NSLog(@"Is string 1 a phone number? %@", result1 > 0 ? @"YES" : @"NO");
NSLog(@"Is string 2 a phone number? %@", result2 > 0 ? @"YES" : @"NO");
```

La salida mostrará que la primera cadena es un número de teléfono y la segunda no lo es.

Lea NSRegularExpression en línea: https://riptutorial.com/es/objective-c/topic/2484/nsregularexpression

Capítulo 39: NSSortDescriptor

Examples

Ordenado por combinaciones de NSSortDescriptor

```
NSArray *aryFName = @[ @"Alice", @"Bob", @"Charlie", @"Quentin" ];
NSArray *aryLName = @[ @"Smith", @"Jones", @"Smith", @"Alberts" ];
NSArray *aryAge = @[ @24, @27, @33, @31 ];

//Create a Custom class with properties for firstName & lastName of type NSString *,
//and age, which is an NSUInteger.

NSMutableArray *aryPerson = [NSMutableArray array];
[firstNames enumerateObjectsUsingBlock:^(id obj, NSUInteger idx, BOOL *stop) {
    Person *person = [[Person alloc] init];
    person.firstName = [aryFName objectAtIndex:idx];
    person.lastName = [aryLName objectAtIndex:idx];
    person.age = [aryAge objectAtIndex:idx];
    [aryPerson addObject:person];
}];

NSSortDescriptor *firstNameSortDescriptor = [NSSortDescriptor
sortDescriptorWithKey:@"firstName"
                                            ascending:YES
                                            selector:@selector(localizedStandardCompare:)];

NSSortDescriptor *lastNameSortDescriptor = [NSSortDescriptor sortDescriptorWithKey:@"lastName"
                                           ascending:YES
                                           selector:@selector(localizedStandardCompare:)];

NSSortDescriptor *ageSortDescriptor = [NSSortDescriptor sortDescriptorWithKey:@"age"
                                       ascending:NO];

NSLog(@"By age: %@", [aryPerson sortedArrayUsingDescriptors:@[ageSortDescriptor]]);
// "Charlie Smith", "Quentin Alberts", "Bob Jones", "Alice Smith"


NSLog(@"By first name: %@", [aryPerson
sortedArrayUsingDescriptors:@[firstNameSortDescriptor]]);
// "Alice Smith", "Bob Jones", "Charlie Smith", "Quentin Alberts"


NSLog(@"By last name, first name: %@", [aryPerson
sortedArrayUsingDescriptors:@[lastNameSortDescriptor, firstNameSortDescriptor]]);
// "Quentin Alberts", "Bob Jones", "Alice Smith", "Charlie Smith"
```

Lea NSSortDescriptor en línea: https://riptutorial.com/es/objective-c/topic/5833/nssortdescriptor

Capítulo 40: NSString

Introducción

La clase *NSString* es una parte del marco de Foundation para trabajar con cadenas (series de caracteres). También incluye métodos para comparar, buscar y modificar cadenas.

Observaciones

Para anidar varios tipos de objetos y tipos de datos en NSStrings, consulte: Objective-C, Especificadores de formato

Examples

Creación

Sencillo:

```
NSString *newString = @"My String";
```

De múltiples cadenas:

```
NSString *stringOne = @"Hello";
NSString *stringTwo = @"world";
NSString *newString = [NSString stringWithFormat:@"My message: %@ %@",
                     stringOne, stringTwo];
```

Usando cadena mutable

```
NSString *stringOne = @"Hello";
NSString *stringTwo = @"World";
NSMutableString *mutableString = [NSMutableString new];
[mutableString appendString:stringOne];
[mutableString appendString:stringTwo];
```

De NSData:

Al inicializar desde `NSData` , se debe proporcionar una codificación explícita, ya que `NSString` no puede adivinar cómo se representan los caracteres en el flujo de datos sin procesar. La codificación más común hoy en día es UTF-8, que es incluso un requisito para ciertos datos como JSON.

Evite utilizar `+[NSString stringWithUTF8String:]` ya que espera una cadena C terminada explícitamente en NULL, que `-[NSData bytes]` *no* proporciona.

```
NSString *newString = [[NSString alloc] initWithData:myData encoding:NSUTF8StringEncoding];
```

De NSArray:

```
NSArray *myArray = [NSArray arrayWithObjects:@"Apple", @"Banana", @"Strawberry", @"Kiwi",
nil];
NSString *newString = [myArray componentsJoinedByString:@" "];
```

Longitud de la cuerda

NSString tiene una propiedad de `length` para obtener el número de caracteres.

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

Como en el ejemplo de división , tenga en cuenta que `NSString` usa UTF-16 para representar caracteres. La longitud es en realidad sólo el número de unidades de código UTF-16. Esto puede diferir de lo que el usuario percibe como personajes.

Aquí hay algunos casos que pueden ser sorprendentes:

```
@"é".length == 1   // LATIN SMALL LETTER E WITH ACUTE (U+00E9)
@"é".length == 2   // LATIN SMALL LETTER E (U+0065) + COMBINING ACUTE ACCENT (U+0301)
@"❤".length == 2  // HEAVY BLACK HEART (U+2764) + VARIATION SELECTOR-16 (U+FE0F)
@"".length == 4  // REGIONAL INDICATOR SYMBOL LETTER I (U+1F1EE) + REGIONAL INDICATOR SYMBOL
LETTER T (U+1F1F9)
```

Para obtener el número de caracteres percibidos por el usuario, conocidos técnicamente como " agrupamientos de grafemas ", debe iterar sobre la cadena con `-enumerateSubstringsInRange:options:usingBlock:` y mantener un recuento. Esto se demuestra en una respuesta de Nikolai Ruhe en Stack Overflow .

Caso cambiante

Para convertir una cadena a mayúsculas, use `uppercaseString` :

```
NSString *myString = @"Emphasize this";
NSLog(@"%@", [myString uppercaseString]; // @"EMPHASIZE THIS"
```

Para convertir una cadena a minúsculas, use `lowercaseString` :

```
NSString *myString = @"NORMALIZE this";
NSLog(@"%@", [myString lowercaseString]; // @"normalize this"
```

Para poner en mayúscula el primer carácter de letra de cada palabra en una cadena, use `capitalizedString` :

```
NSString *myString = @"firstname lastname";
NSLog(@"%@", [myString capitalizedString]); // @"Firstname Lastname"
```

Comparando cuerdas

Las cadenas se comparan para la igualdad usando `isEqualToString:`

El operador `==` solo prueba la identidad del objeto y no compara los valores lógicos de los objetos, por lo que no se puede usar:

```
NSString *stringOne = @"example";
NSString *stringTwo = [stringOne mutableCopy];

BOOL objectsAreIdentical = (stringOne == stringTwo);          // NO
BOOL stringsAreEqual = [stringOne isEqualToString:stringTwo]; // YES
```

La expresión `(stringOne == stringTwo)` comprueba si las direcciones de memoria de las dos cadenas son iguales, lo que generalmente no es lo que queremos.

Si las variables de cadena pueden ser `nil` también debe tener cuidado con este caso:

```
BOOL equalValues = stringOne == stringTwo || [stringOne isEqualToString:stringTwo];
```

Esta condición devuelve `YES` cuando las cadenas tienen valores iguales o ambos son `nil` .

Para ordenar dos cadenas alfabéticamente, utilice `compare` :.

```
NSComparisonResult result = [firstString compare:secondString];
```

`NSComparisonResult` puede ser:

- `NSOrderedAscending` : la primera cadena aparece antes de la segunda.
- `NSOrderedSame` : Las cadenas son iguales.
- `NSOrderedDescending` : la segunda cadena aparece antes de la primera.

Para comparar la igualdad de dos cadenas, use `isEqualToString:`

```
BOOL result = [firstString isEqualToString:secondString];
```

Para comparar con la cadena vacía (`@""`), use mejor la `length` .

```
BOOL result = string.length == 0;
```

Unirse a una matriz de cuerdas

Para combinar una `NSArray` de `NSString` en una nueva `NSString` :

```
NSArray *yourWords = @[@"Objective-C", @"is", @"just", @"awesome"];
NSString *sentence = [yourWords componentsJoinedByString:@" "];

// Sentence is now: @"Objective-C is just awesome"
```

Codificación y decodificación

```
// decode
NSString *string = [[NSString alloc] initWithData:utf8Data
                                          encoding:NSUTF8StringEncoding];

// encode
NSData *utf8Data = [string dataUsingEncoding:NSUTF8StringEncoding];
```

Algunas codificaciones soportadas son:

- `NSASCIIStringEncoding`
- `NSUTF8StringEncoding`
- `NSUTF16StringEncoding` (== `NSUnicodeStringEncoding`)

Tenga en cuenta que `utf8Data.bytes` no incluye un carácter nulo de terminación, que es necesario para las cadenas C. Si necesita una cadena en C, use `UTF8String` :

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

Terrible

Puede dividir una cadena en una matriz de partes, dividida por **un carácter separador** .

```
NSString * yourString = @"Stack,Exchange,Network";
NSArray * yourWords = [yourString componentsSeparatedByString:@","];
// Output: @[@"Stack", @"Exchange", @"Network"]
```

Si necesita dividir en un conjunto de **varios delimitadores diferentes** , use `-[NSString componentsSeparatedByCharactersInSet:]` .

```
NSString * yourString = @"Stack Overflow+Documentation/Objective-C";
NSArray * yourWords = [yourString componentsSeparatedByCharactersInSet:
                       [NSCharacterSet characterSetWithCharactersInString:@"+/"]];
// Output: @[@"Stack Overflow", @"Documentation", @"Objective-C"]`
```

Si necesita dividir una cadena en sus **caracteres individuales** , recorra la longitud de la cadena y convierta cada carácter en una nueva cadena.

```
NSMutableArray * characters = [[NSMutableArray alloc] initWithCapacity:[yourString length]];
for (int i = 0; i < [myString length]; i++) {
    [characters addObject: [NSString stringWithFormat:@"%C",
                                      [yourString characterAtIndex:i]];
}
```

Como en el Ejemplo de longitud , tenga en cuenta que un "carácter" aquí es una unidad de código UTF-16, no necesariamente lo que el usuario ve como un carácter. Si usa este bucle con `@"👨👩"` , verá que está dividido en cuatro partes.

Para obtener una lista de los caracteres percibidos por el usuario, use `-enumerateSubstringsInRange:options:usingBlock:`

```
NSMutableArray * characters = [NSMutableArray array];
[yourString enumerateSubstringsInRange:(NSRange){0, [yourString length]}
                               options:NSStringEnumerationByComposedCharacterSequences
                            usingBlock:^(NSString * substring, NSRange r, NSRange s, BOOL *
b){
                                [characters addObject:substring];
                            }];
```

Esto conserva grupos de grafemas como la bandera italiana como una sola subcadena.

Buscando una subcadena

Para buscar si una cadena contiene una subcadena, haga lo siguiente:

```
NSString *myString = @"This is for checking substrings";
NSString *subString = @"checking";

BOOL doesContainSubstring = [myString containsString:subString];  // YES
```

Si se dirige a iOS 7 o OS X 10.9 (o anterior):

```
BOOL doesContainSubstring = ([myString rangeOfString:subString].location != NSNotFound);  //
YES
```

Trabajar con cuerdas C

Para convertir `NSString` en uso de caracteres `const char` `-[NSString UTF8String]` :

```
NSString *myNSString = @"Some string";
const char *cString = [myNSString UTF8String];
```

También puede usar `-[NSString cStringUsingEncoding:]` si su cadena está codificada con algo distinto a UTF-8.

Para el uso de la ruta inversa `-[NSString stringWithUTF8String:]` :

```
const *char cString = "Some string";
NSString *myNSString = [NSString stringWithUTF8String:cString];
myNSString = @(cString); // Equivalent to the above.
```

Una vez que tenga el `const char *` , puede trabajar con él de manera similar a una serie de `chars` :

```
printf("%c\n", cString[5]);
```

Si desea modificar la cadena, haga una copia:

```
char *cpy = calloc(strlen(cString)+1, 1);
strncpy(cpy, cString, strlen(cString));
// Do stuff with cpy
free(cpy);
```

Eliminar espacios en blanco iniciales y finales

```
NSString *someString = @"   Objective-C Language  \n";
NSString *trimmedString = [someString stringByTrimmingCharactersInSet:[NSCharacterSet
whitespaceAndNewlineCharacterSet]];
//Output will be - "Objective-C Language"
```

El método stringByTrimmingCharactersInSet devuelve una nueva cadena creada al eliminar de ambos extremos los caracteres de cadena contenidos en un conjunto de caracteres determinado.

También podemos eliminar solo espacios en blanco o nueva línea.

```
// Removing only WhiteSpace
NSString *trimmedWhiteSpace = [someString stringByTrimmingCharactersInSet:[NSCharacterSet
whitespaceCharacterSet]];
//Output will be - "Objective-C Language  \n"

// Removing only NewLine
NSString *trimmedNewLine = [someString stringByTrimmingCharactersInSet:[NSCharacterSet
newlineCharacterSet]];
//Output will be - "   Objective-C Language  "
```

Formateo

El formato `NSString` admite todas las cadenas de formato disponibles en la `printf` ANSI-C de `printf` . La única adición hecha por el lenguaje es el símbolo `%@` usado para formatear todos los objetos de Objective-C.

Es posible formatear enteros.

```
int myAge = 21;
NSString *formattedAge = [NSString stringWithFormat:@"I am %d years old", my_age];
```

O cualquier objeto subclasificado de NSObject

```
NSDate *now = [NSDate date];
NSString *formattedDate = [NSString stringWithFormat:@"The time right now is: %@", now];
```

Para obtener una lista completa de los Especificadores de formato, consulte: Objective-C, Especificadores de formato, Sintaxis

Invirtiendo un Objective-C de NSString

```
// myString is "hi"
```

```
NSMutableString *reversedString = [NSMutableString string];
NSInteger charIndex = [myString length];
while (charIndex > 0) {
    charIndex--;
    NSRange subStrRange = NSMakeRange(charIndex, 1);
    [reversedString appendString:[myString substringWithRange:subStrRange]];
}
NSLog(@"%@", reversedString); // outputs "ih"
```

Lea NSString en línea: https://riptutorial.com/es/objective-c/topic/832/nsstring

Capítulo 41: NSTextEnganche

Sintaxis

1. NSTextAttachment * attachmentName = [[NSTextAttachment alloc] init];

Observaciones

`NSTextAttachment` objetos `NSTextAttachment` son utilizados por el `NSAttributedString` clase `NSAttributedString` como valores para los atributos de adjunto. Los objetos que creas con esta clase se conocen como objetos de adjuntos de texto, o cuando no hay confusión, como adjuntos de texto o simplemente adjuntos.

Examples

Ejemplo NSTextAttachment

```
NSTextAttachment *attachment = [[NSTextAttachment alloc] init];
attachment.image = [UIImage imageNamed:@"imageName"];
attachment.bounds = CGRectMake(0, 0, 35, 35);
NSAttributedString *attachmentString = [NSAttributedString
attributedStringWithAttachment:attachment];
```

Lea NSTextEnganche en línea: https://riptutorial.com/es/objective-c/topic/7143/nstextenganche

Capítulo 42: NSTimer

Examples

Creando un temporizador

Esto creará un temporizador para llamar al método `doSomething` en `self` en `5.0` segundos.

```
[NSTimer scheduledTimerWithTimeInterval:5.0
         target:self
         selector:@selector(doSomething)
         userInfo:nil
         repeats:NO];
```

Establecer el parámetro de `repeats` en `false/NO` indica que queremos que el temporizador se active solo una vez. Si configuramos esto en `true/YES` , se dispararía cada cinco segundos hasta que se invalide manualmente.

Invalidando un temporizador

```
[timer invalidate];
timer = nil;
```

Esto detendrá el temporizador de disparar. **Debe llamarse desde el hilo en el que se creó el temporizador,** consulte las notas de Apple :

> Debe enviar este mensaje desde el hilo en el que se instaló el temporizador. Si envía este mensaje desde otro hilo, es posible que la fuente de entrada asociada con el temporizador no se elimine de su ciclo de ejecución, lo que podría impedir que el hilo salga correctamente.

Establecer `nil` te ayudará a comprobar si se está ejecutando o no.

```
if(timer) {
    [timer invalidate];
    timer = nil;
}

//Now set a timer again.
```

Manualmente disparando un temporizador

```
[timer fire];
```

Llamar al método de `fire` hace que un NSTimer realice la tarea que normalmente habría realizado en un horario.

En un **temporizador que no se repite** , esto invalidará automáticamente el temporizador. Es decir, activar el `fire` antes de que `fire` el intervalo de tiempo dará lugar a una sola invocación.

En un **temporizador de repetición** , esto simplemente invocará la acción sin interrumpir el horario habitual.

Almacenando información en el temporizador

Al crear un temporizador, puede configurar el parámetro `userInfo` para que incluya información que desea pasar a la función a la que llama con el temporizador.

Al tomar un temporizador como parámetro en dicha función, puede acceder a la propiedad `userInfo` .

```
NSDictionary *dictionary = @{
                              @"Message" : @"Hello, world!"
                             }; //this dictionary contains a message
[NSTimer scheduledTimerWithTimeInterval:5.0
     target:self
     selector:@selector(doSomething)
     userInfo:dictionary
     repeats:NO]; //the timer contains the dictionary and later calls the function

...

- (void) doSomething:(NSTimer*)timer{
    //the function retrieves the message from the timer
    NSLog("%@", timer.userInfo["Message"]);
}
```

Lea NSTimer en línea: https://riptutorial.com/es/objective-c/topic/3773/nstimer

Capítulo 43: NSURL

Examples

Crear

De NSString:

```
NSString *urlString = @"https://www.stackoverflow.com";
NSURL *myUrl = [NSURL URLWithString: urlString];
```

También puede utilizar los siguientes métodos:

```
- initWithString:
+ URLWithString:relativeToURL:
- initWithString:relativeToURL:
+ fileURLWithPath:isDirectory:
- initFileURLWithPath:isDirectory:
+ fileURLWithPath:
- initFileURLWithPath:
 Designated Initializer
+ fileURLWithPathComponents:
+ URLByResolvingAliasFileAtURL:options:error:
+ URLByResolvingBookmarkData:options:relativeToURL:bookmarkDataIsStale:error:
- initByResolvingBookmarkData:options:relativeToURL:bookmarkDataIsStale:error:
+ fileURLWithFileSystemRepresentation:isDirectory:relativeToURL:
- getFileSystemRepresentation:maxLength:
- initFileURLWithFileSystemRepresentation:isDirectory:relativeToURL:
```

Compara NSURL

```
NSString *urlString = @"https://www.stackoverflow.com";

NSURL *myUrl = [NSURL URLWithString: urlString];
NSURL *myUrl2 = [NSURL URLWithString: urlString];

if ([myUrl isEqual:myUrl2]) return YES;
```

Modificación y conversión de una URL de archivo con la eliminación y la adición de la ruta

1. URLByDeletingPathExtension:

Si el receptor representa la ruta raíz, esta propiedad contiene una copia de la URL original. Si la URL tiene varias extensiones de ruta, solo se eliminará la última.

2. URLByAppendingPathExtension:

Devuelve una nueva URL hecha agregando una extensión de ruta a la URL original.

Ejemplo:

```
    NSUInteger count = 0;
        NSString *filePath = nil;
        do {
            NSString *extension = ( NSString *)UTTypeCopyPreferredTagWithClass((
CFStringRef)AVFileTypeQuickTimeMovie, kUTTagClassFilenameExtension);
            NSString *fileNameNoExtension = [[asset.defaultRepresentation.url
URLByDeletingPathExtension] lastPathComponent];//Delete is used
            NSString *fileName = [NSString stringWithFormat:@"%@-%@-%u",fileNameNoExtension ,
AVAssetExportPresetLowQuality, count];
            filePath = NSTemporaryDirectory();
            filePath = [filePath stringByAppendingPathComponent:fileName];//Appending is used
            filePath = [filePath stringByAppendingPathExtension:extension];
            count++;

        } while ([[NSFileManager defaultManager] fileExistsAtPath:filePath]);

        NSURL *outputURL = [NSURL fileURLWithPath:filePath];
```

Lea NSURL en línea: https://riptutorial.com/es/objective-c/topic/1187/nsurl

Capítulo 44: NSUrl enviar una solicitud de publicación

Examples

Solicitud POST simple

```
// Create the request.
NSMutableURLRequest *request = [NSMutableURLRequest requestWithURL:[NSURL
URLWithString:@"http://google.com"]];

// Specify that it will be a POST request
request.HTTPMethod = @"POST";

// This is how we set header fields
[request setValue:@"application/xml; charset=utf-8" forHTTPHeaderField:@"Content-Type"];

// Convert your data and set your request's HTTPBody property
NSString *stringData = @"some data";
NSData *requestBodyData = [stringData dataUsingEncoding:NSUTF8StringEncoding];
request.HTTPBody = requestBodyData;

// Create url connection and fire request
NSURLConnection *conn = [[NSURLConnection alloc] initWithRequest:request delegate:self];
```

Solicitud de publicación simple con tiempo de espera

```
// Create the request.
NSMutableURLRequest *request = [NSMutableURLRequest requestWithURL:[NSURL
URLWithString:@"http://google.com"]];

// Specify that it will be a POST request
request.HTTPMethod = @"POST";

// Setting a timeout
request.timeoutInterval = 20.0;
// This is how we set header fields
[request setValue:@"application/xml; charset=utf-8" forHTTPHeaderField:@"Content-Type"];

// Convert your data and set your request's HTTPBody property
NSString *stringData = @"some data";
NSData *requestBodyData = [stringData dataUsingEncoding:NSUTF8StringEncoding];
request.HTTPBody = requestBodyData;

// Create url connection and fire request
NSURLConnection *conn = [[NSURLConnection alloc] initWithRequest:request delegate:self];
```

Lea NSUrl enviar una solicitud de publicación en línea: https://riptutorial.com/es/objective-c/topic/7243/nsurl-enviar-una-solicitud-de-publicacion

Capítulo 45: NSUserDefaults

Examples

Ejemplo simple

Por ejemplo:

PARA AHORRAR:

```
  NSUserDefaults *prefs = [NSUserDefaults standardUserDefaults];

  // saving an NSString
  [prefs setObject:txtUsername.text forKey:@"userName"];
  [prefs setObject:txtPassword.text forKey:@"password"];

  [prefs synchronize];
```

PARA RECUPERAR

```
  NSUserDefaults *prefs = [NSUserDefaults standardUserDefaults];

  // getting an NSString
 NSString *savedUsername = [prefs stringForKey:@"userName"];
 NSString *savedPassword = [prefs stringForKey:@"password"];
```

Borrar NSUserDefaults

```
NSString *appDomain = [[NSBundle mainBundle] bundleIdentifier];
[[NSUserDefaults standardUserDefaults] removePersistentDomainForName:appDomain];
```

Lea NSUserDefaults en línea: https://riptutorial.com/es/objective-c/topic/10713/nsuserdefaults

Capítulo 46: Objetivo moderno-C

Examples

Literales

El Modern Objective C proporciona formas de reducir la cantidad de código que necesita para inicializar algunos tipos comunes. Esta nueva forma es muy similar a cómo se inicializan los objetos NSString con cadenas constantes.

NSNumber

Vieja forma:

```
NSNumber *number = [NSNumber numberWithInt:25];
```

Manera moderna:

```
NSNumber *number = @25;
```

Nota: también se puede almacenar `BOOL` valores en `NSNumber` objetos usando `@YES` , `@NO` o `@(someBoolValue)` ;

NSArray

Vieja forma:

```
NSArray *array = [[NSArray alloc] initWithObjects:@"One", @"Two", [NSNumber numberWithInt:3],
@"Four", nil];
```

Manera moderna:

```
NSArray *array = @[@"One", @"Two", @3, @"Four"];
```

NSDiccionario

Vieja forma:

```
NSDictionary *dictionary = [NSDictionary dictionaryWithObjectsAndKeys: array, @"Object",
[NSNumber numberWithFloat:1.5], @"Value", @"ObjectiveC", @"Language", nil];
```

Manera moderna:

```
NSDictionary *dictionary = @{@"Object": array, @"Value": @1.5, @"Language": @"ObjectiveC"};
```

Suscripción de contenedores

En la sintaxis moderna de Objective C, puede obtener valores de los contenedores `NSArray` y `NSDictionary` utilizando subíndices de contenedor.

Vieja forma:

```
NSObject *object1 = [array objectAtIndex:1];
NSObject *object2 = [dictionary objectForKey:@"Value"];
```

Manera moderna:

```
NSObject *object1 = array[1];
NSObject *object2 = dictionary[@"Value"];
```

También puede insertar objetos en matrices y establecer objetos para claves en diccionarios de una manera más limpia:

Vieja forma:

```
// replacing at specific index
[mutableArray replaceObjectAtIndex:1 withObject:@"NewValue"];
// adding a new value to the end
[mutableArray addObject:@"NewValue"];

[mutableDictionary setObject:@"NewValue" forKey:@"NewKey"];
```

Manera moderna:

```
mutableArray[1] = @"NewValue";
mutableArray[[mutableArray count]] = @"NewValue";

mutableDictionary[@"NewKey"] = @"NewValue";
```

Lea Objetivo moderno-C en línea: https://riptutorial.com/es/objective-c/topic/9486/objetivo-moderno-c

Capítulo 47: Propiedades

Sintaxis

- @property (*optional_attributes, ...*) *identificador de tipo* ;
- @synthesize *identifier* = *optional_backing_ivar* ;
- *identificador* @dynamic;

Parámetros

Atributo	Descripción
`atomic`	*Implícito.* Habilita la sincronización en métodos de acceso sintetizados.
`nonatomic`	Desactiva la sincronización en los métodos de acceso sintetizados.
`readwrite`	*Implícito.* Sintetiza getter, setter y respaldo ivar.
`readonly`	Sintetiza solo el método getter y el respaldo de ivar, que se pueden asignar directamente.
`getter=` *nombre*	Especifica el nombre del método getter, implícito es `propertyName` .
`setter=` *nombre*	Especifica el nombre del método de establecimiento, implícito es `setPropertyName:` Colon `:` debe ser parte del nombre.
`strong`	*Implícito para objetos bajo ARC* . El respaldo ivar se sintetiza utilizando `__strong` , lo que evita la desasignación del objeto referenciado.
`retain`	Sinónimo de `strong` .
`copy`	Igual que `strong` , pero el definidor sintetizado también llama a la `-copy` en el nuevo valor.
`unsafe_unretained`	*Implícito, excepto para los objetos bajo ARC.* El soporte ivar se sintetiza utilizando `__unsafe_unretained` , que (para objetos) da como resultado un puntero colgante una vez que el objeto referenciado se desasigna.
`assign`	Sinónimo para `unsafe_unretained` . Adecuado para tipos sin objeto.
`weak`	El respaldo de ivar se sintetiza utilizando `__weak` , por lo que el valor se `__weak` una vez que se desasigne el objeto al que se hace referencia.
`class`	Los accesores de propiedades se sintetizan como métodos de clase, en lugar de métodos de instancia. No se sintetiza ningún almacenamiento de respaldo.

Atributo	Descripción
`nullable`	La propiedad acepta valores `nil` . Utilizado principalmente para puentes rápidos.
`nonnull`	La propiedad no acepta valores `nil` . Utilizado principalmente para puentes rápidos.
`null_resettable`	La propiedad acepta valores `nil` en el establecedor, pero nunca devuelve valores `nil` de getter. Su implementación personalizada de getter o setter debe garantizar este comportamiento. Utilizado principalmente para puentes rápidos.
`null_unspecified`	*Implícito.* La propiedad no especifica el manejo de valores `nil` . Utilizado principalmente para puentes rápidos.

Examples

¿Qué son las propiedades?

Aquí hay una clase de ejemplo que tiene un par de variables de instancia, sin usar propiedades:

```
@interface TestClass : NSObject {
    NSString *_someString;
    int _someInt;
}

-(NSString *)someString;
-(void)setSomeString:(NSString *)newString;

-(int)someInt;
-(void)setSomeInt:(NSString *)newInt;

@end


@implementation TestClass

-(NSString *)someString {
    return _someString;
}

-(void)setSomeString:(NSString *)newString {
    _someString = newString;
}

-(int)someInt {
    return _someInt;
}

-(void)setSomeInt:(int)newInt {
    _someInt = newInt;
}
```

```
@end
```

Esto es bastante código repetitivo para crear una variable de instancia simple. Debe crear la variable de instancia y crear métodos de acceso que no hagan nada, excepto establecer o devolver la variable de instancia. Así que con Objective-C 2.0, Apple introdujo propiedades, que generan automáticamente parte o la totalidad del código repetitivo.

Aquí está la clase anterior reescrita con propiedades:

```
@interface TestClass

@property NSString *someString;
@property int someInt;

@end


@implementation testClass

@end
```

Una propiedad es una variable de instancia emparejada con captadores y definidores generados automáticamente. Para una propiedad llamada `someString` , el getter y setter se llaman `someString` y `setSomeString:` respectivamente. El nombre de la variable de instancia es, de forma predeterminada, el nombre de la propiedad con un guión bajo (por lo que la variable de instancia para `someString` se llama `_someString` , pero esto se puede anular con una directiva `@synthesize` en la sección `@implementation` :

```
@synthesize someString=foo;    //names the instance variable "foo"
@synthesize someString;    //names it "someString"
@synthesize someString=_someString;        //names it "_someString"; the default if
                                           //there is no @synthesize directive
```

Se puede acceder a las propiedades llamando a los captadores y configuradores:

```
[testObject setSomeString:@"Foo"];
NSLog(@"someInt is %d", [testObject someInt]);
```

También se puede acceder a ellos usando notación de puntos:

```
testObject.someString = @"Foo";
NSLog(@"someInt is %d", testObject.someInt);
```

Capturadores personalizados y setters

Los captadores y definidores de propiedades predeterminados se pueden anular:

```
@interface TestClass

@property NSString *someString;
```

```
@end

@implementation TestClass

// override the setter to print a message
- (void)setSomeString:(NSString *)newString {
    NSLog(@"Setting someString to %@", newString);
    // Make sure to access the ivar (default is the property name with a _
    // at the beginning) because calling self.someString would call the same
    // method again leading to an infinite recursion
    _someString = newString;
}

- (void)doSomething {
    // The next line will call the setSomeString: method
    self.someString = @"Test";
}

@end
```

Esto puede ser útil para proporcionar, por ejemplo, una inicialización perezosa (anulando al getter para establecer el valor inicial si aún no se ha establecido):

```
- (NSString *)someString {
    if (_someString == nil) {
        _someString = [self getInitialValueForSomeString];
    }
    return _someString;
}
```

También puedes hacer una propiedad que calcula su valor en el captador:

```
@interface Circle : NSObject

@property CGPoint origin;
@property CGFloat radius;
@property (readonly) CGFloat area;

@end

@implementation Circle

- (CGFloat)area {
    return M_PI * pow(self.radius, 2);
}

@end
```

Propiedades que causan actualizaciones

Este objeto, `Shape` tiene una `image` propiedad que depende de `numberOfSides` y `sideWidth` . Si cualquiera de ellos está configurado, entonces la `image` debe ser recalculada. Pero el recálculo es presumiblemente largo, y solo debe hacerse una vez si ambas propiedades están establecidas, por lo que la `Shape` proporciona una manera de establecer ambas propiedades y solo recalcularlas una vez. Esto se hace estableciendo la propiedad ivars directamente.

En `Shape.h`

```
@interface Shape {
    NSUInteger numberOfSides;
    CGFloat sideWidth;

    UIImage * image;
}

// Initializer that takes initial values for the properties.
- (instancetype)initWithNumberOfSides:(NSUInteger)numberOfSides withWidth:(CGFloat)width;

// Method that allows to set both properties in once call.
// This is useful if setting these properties has expensive side-effects.
// Using a method to set both values at once allows you to have the side-
// effect executed only once.
- (void)setNumberOfSides:(NSUInteger)numberOfSides andWidth:(CGFloat)width;

// Properties using default attributes.
@property NSUInteger numberOfSides;
@property CGFloat sideWidth;

// Property using explicit attributes.
@property(strong, readonly) UIImage * image;

@end
```

En `Shape.m`

```
@implementation AnObject

// The variable name of a property that is auto-generated by the compiler
// defaults to being the property name prefixed with an underscore, for
// example "_propertyName". You can change this default variable name using
// the following statement:
// @synthesize propertyName = customVariableName;

- (id)initWithNumberOfSides:(NSUInteger)numberOfSides withWidth:(CGFloat)width {
    if ((self = [self init])) {
       [self setNumberOfSides:numberOfSides andWidth:width];
    }

    return self;
}

- (void)setNumberOfSides:(NSUInteger)numberOfSides {
    _numberOfSides = numberOfSides;

    [self updateImage];
}

- (void)setSideWidth:(CGFloat)sideWidth {
    _sideWidth = sideWidth;

    [self updateImage];
}

- (void)setNumberOfSides:(NSUInteger)numberOfSides andWidth:(CGFloat)sideWidth {
    _numberOfSides = numberOfSides;
```

```
    _sideWidth = sideWidth;

    [self updateImage];
}

// Method that does some post-processing once either of the properties has
// been updated.
- (void)updateImage {
    ...
}

@end
```

Cuando las propiedades se asignan a (usando `object.property = value`), se llama al método `setProperty:` Este configurador, incluso si lo proporciona `@synthesize` , puede ser anulado, como lo es en este caso para `numberOfSides` y `sideWidth` . Sin embargo, si configura el ivar de una propiedad directamente (a través de la `property` si el objeto es self, o `object->property`), no llama al captador ni al establecedor, lo que le permite hacer cosas como conjuntos de propiedades múltiples que solo llaman una actualización o Evita los efectos secundarios causados por el colocador.

Lea Propiedades en línea: https://riptutorial.com/es/objective-c/topic/1818/propiedades

Capítulo 48: Protocolos

Examples

Definición de protocolo básico

Definiendo un nuevo protocolo:

```
@protocol NewProtocol

- (void)protocolMethod:(id)argument;

- (id)anotherMethod;

@end
```

Métodos opcionales y requeridos.

Por defecto, todos los métodos declarados en un protocolo son obligatorios. Esto significa que cualquier clase que se ajuste a este protocolo debe implementar esos métodos.

También es posible declarar métodos *opcionales* . Estos métodos solo se pueden implementar si es necesario.

Se marcan métodos opcionales con la directiva `@optional` .

```
@protocol NewProtocol
- (void)protocolMethod:(id)argument;
@optional
- (id)anotherMethod;
@end
```

En este caso, solo otro `anotherMethod` está marcado como opcional; se supone que los métodos sin la directiva `@optional` son requeridos.

La directiva `@optional` aplica a los métodos que siguen, hasta el final de la definición del protocolo o, hasta que se encuentre otra directiva.

```
@protocol NewProtocol
- (void)protocolMethod:(id)argument;
@optional
- (id)anotherMethod;
- (void)andAnotherMethod:(id)argument;
@required
- (void)lastProtocolMethod;
@end
```

Este último ejemplo define un protocolo con dos métodos opcionales y dos métodos requeridos.

Conforme a los Protocolos

La siguiente sintaxis indica que una clase adopta un protocolo, utilizando paréntesis angulares.

```
@interface NewClass : NSObject <NewProtocol>
...
@end
```

Esto significa que cualquier instancia de NewClass responderá a los métodos declarados en su interfaz, pero también proporcionará una implementación para todos los métodos requeridos de `NewProtocol` .

También es posible que una clase se ajuste a múltiples protocolos, separándolos con coma.

```
@interface NewClass : NSObject <NewProtocol, AnotherProtocol, MyProtocol>
...
@end
```

Al igual que cuando se ajusta a un solo protocolo, la clase debe implementar cada método requerido de cada protocolo y cada método opcional que elija implementar.

Declaraciones Forward

Es posible declarar el nombre del protocolo sin métodos:

```
@protocol Person;
```

utilízalo tu código (definición de clase, etc):

```
@interface World : NSObject
@property (strong, nonatomic) NSArray<id<some>> *employees;
@end
```

y luego defina el método del protocolo en algún lugar de su código:

```
@protocol Person
- (NSString *)gender;
- (NSString *)name;
@end
```

Es útil cuando no necesita conocer los detalles de los protocolos hasta que importe ese archivo con la definición del protocolo. Por lo tanto, el archivo de encabezado de su clase permanece claro y solo contiene detalles de la clase.

Comprobando la existencia de implementaciones de métodos opcionales.

```
if ([object respondsToSelector:@selector(someOptionalMethodInProtocol:)])
{
    [object someOptionalMethodInProtocol:argument];
```

```
}
```

Verifica el protocolo de conformidad

Devuelve un valor booleano que indica si la clase cumple con el protocolo:

```
[MyClass conformsToProtocol:@protocol(MyProtocol) ];
```

Lea Protocolos en línea: https://riptutorial.com/es/objective-c/topic/448/protocolos

Capítulo 49: Protocolos y Delegados.

Observaciones

Protocolos y *delegados* son dos conceptos relacionados pero diferentes:

Un *protocolo* es una interfaz que una clase puede cumplir, lo que significa que la clase implementa los métodos enumerados.

Un *delegado* suele ser un objeto anónimo que se ajusta a un protocolo.

La aplicación del *Delegado* llamado *Delegación* es un patrón de diseño.

En un extremo, tenemos el concepto de *herencia,* que crea un acoplamiento estrecho entre la subclase y su superclase, mientras que el patrón de diseño de *Delegación* proporciona una alternativa para evitar este acoplamiento estrecho mediante el cual podemos crear una relación mucho más flexible basada en objetos *Delegados* anónimos.

Examples

Implementación de Protocolos y mecanismo de Delegación.

Supongamos que tiene dos vistas `ViewA` y `ViewB`

La instancia de `ViewB` se crea dentro de `ViewA` , por lo que `ViewA` puede enviar un mensaje a `ViewB's` instancia `ViewB's` , pero para que ocurra lo contrario, necesitamos implementar una delegación (de modo que el uso de `ViewB's` instancia delegada de `ViewB's` pueda enviar un mensaje a `ViewA`)

Siga estos pasos para implementar la delegación.

1. En `ViewB` crear protocolo como

```
 @protocol ViewBDelegate

-(void) exampleDelegateMethod;

 @end
```

2. Declarar al delegado en la clase de remitente.

```
 @interface ViewB : UIView
 @property (nonatomic, weak) id< ViewBDelegate > delegate;
 @end
```

3. Adoptar el protocolo en Class ViewA.

```
@interfac ViewA: UIView < ViewBDelegate >
```

4. Establecer el delegado

```
-(void) anyFunction
{
    // create Class ViewB's instance and set the delegate
    [viewB setDelegate:self];
}
```

5. Implementar el método delegado en la clase `ViewA`

```
-(void) exampleDelegateMethod
{
    // will be called by Class ViewB's instance
}
```

6. Utilice el método en la clase `ViewB` para llamar al método delegado como

```
-(void) callDelegateMethod
{
    [delegate exampleDelegateMethod];
    //assuming the delegate is assigned otherwise error
}
```

Lea Protocolos y Delegados. en línea: https://riptutorial.com/es/objective-c/topic/7832/protocolos-y-delegados-

Capítulo 50: Pruebas unitarias utilizando Xcode

Observaciones

Dependencias :

- Si la aplicación utiliza bibliotecas de terceros o pods de cacao, entonces esas bibliotecas o pods también deben instalarse para prueba.
- La clase de prueba (Traje de prueba) extiende XCTestCase.

Cepillarse antes de comenzar:

- Todas las clases de prueba tienen dos métodos en común setUp y tearDown.
- setUp se ejecuta antes de cada testcase y tearDown después de cada testcase.
- Los casos de prueba se ejecutan alfabéticamente.
- En Test Driven Development, es bueno crear datos de prueba ficticios primero.
- Los métodos de caso de prueba comienzan con la palabra clave "prueba".
- Los métodos de prueba no aceptan parámetros y no devuelven ningún valor.

Apéndice:

Hay varios otros métodos para comparar el resultado esperado y el resultado real de una operación. Algunos de esos métodos se enumeran a continuación:

- XCTAssertNil (expresión, comentario) genera un error si la expresión! = Nil.
- XCTAssertNotNil (expresión, comentario) genera un error si expresión = nil.
- XCTAssert (expresión, comentario) genera un error si la expresión == falsa.
- XCTAssertTrue (expresión, comentario) genera un error si la expresión == falsa.
- XCTAssertFalse (expresión, comentario) genera un error si la expresión! = Falso.
- XCTAssertEqualObjects (expresión1, expresión2, comentario) genera una falla si expresión1 no es igual a expresión2.
- XCTAssertEqualObjects (expresión1, expresión2, comentario) genera una falla si expresión1 es igual a expresión2.
- XCTAssertNotEqual (expresión1, expresión2, comentario) genera un error si expresión1 == expresión2.
- XCTAssertEqual (expresión1, expresión2, comentario) genera un error si expresión1! = Expresión2.
- XCTAssertGreaterThanOrEqual (expresión1, expresión2, comentario) genera un error cuando (expresión1 <expresión2).

Examples

Probando un bloque de código o algún método:

- Importe la clase, que contiene el método a probar.
- Realizar la operación con datos ficticios.
- Ahora compare el resultado de la operación con el resultado esperado.

```
- (void)testReverseString{
NSString *originalString = @"hi_my_name_is_siddharth";
NSString *reversedString = [self.someObject reverseString:originalString];
NSString *expectedReversedString = @"htrahddis_si_eman_ym_ih";
XCTAssertEqualObjects(expectedReversedString, reversedString, @"The reversed string did not
match the expected reverse");
}
```

> **Alimente los datos ficticios al método bajo prueba si es necesario y luego compare los resultados esperados y reales.**

Prueba de bloque asíncrono de código:

```
- (void)testDoSomethingThatTakesSomeTime{
XCTestExpectation *completionExpectation = [self expectationWithDescription:@"Long method"];
[self.someObject doSomethingThatTakesSomeTimesWithCompletionBlock:^(NSString *result) {
    XCTAssertEqualObjects(@"result", result, @"Result was not correct!");
    [completionExpectation fulfill];
}];
[self waitForExpectationsWithTimeout:5.0 handler:nil];
}
```

- Alimente los datos ficticios al método bajo prueba si es necesario.
- La prueba se detendrá aquí, ejecutando el ciclo de ejecución, hasta que se alcance el tiempo de espera o se cumplan todas las expectativas.
- El tiempo de espera es el tiempo esperado para que responda el bloque asíncrono.

Medición del rendimiento de un bloque de código:

1. Para los métodos sincrónicos:

```
- (void)testPerformanceReverseString {
    NSString *originalString = @"hi_my_name_is_siddharth";
    [self measureBlock:^{
        [self.someObject reverseString:originalString];
```

```
    }];
}
```

2. Para los métodos asíncronos:

```
- (void)testPerformanceOfAsynchronousBlock {
   [self measureMetrics:@[XCTPerformanceMetric_WallClockTime] automaticallyStartMeasuring:YES
forBlock:^{

    XCTestExpectation *expectation = [self
expectationWithDescription:@"performanceTestWithResponse"];

    [self.someObject doSomethingThatTakesSomeTimesWithCompletionBlock:^(NSString *result) {
        [expectation fulfill];
    }];
    [self waitForExpectationsWithTimeout:5.0 handler:^(NSError *error) {
    }];
}];
}
```

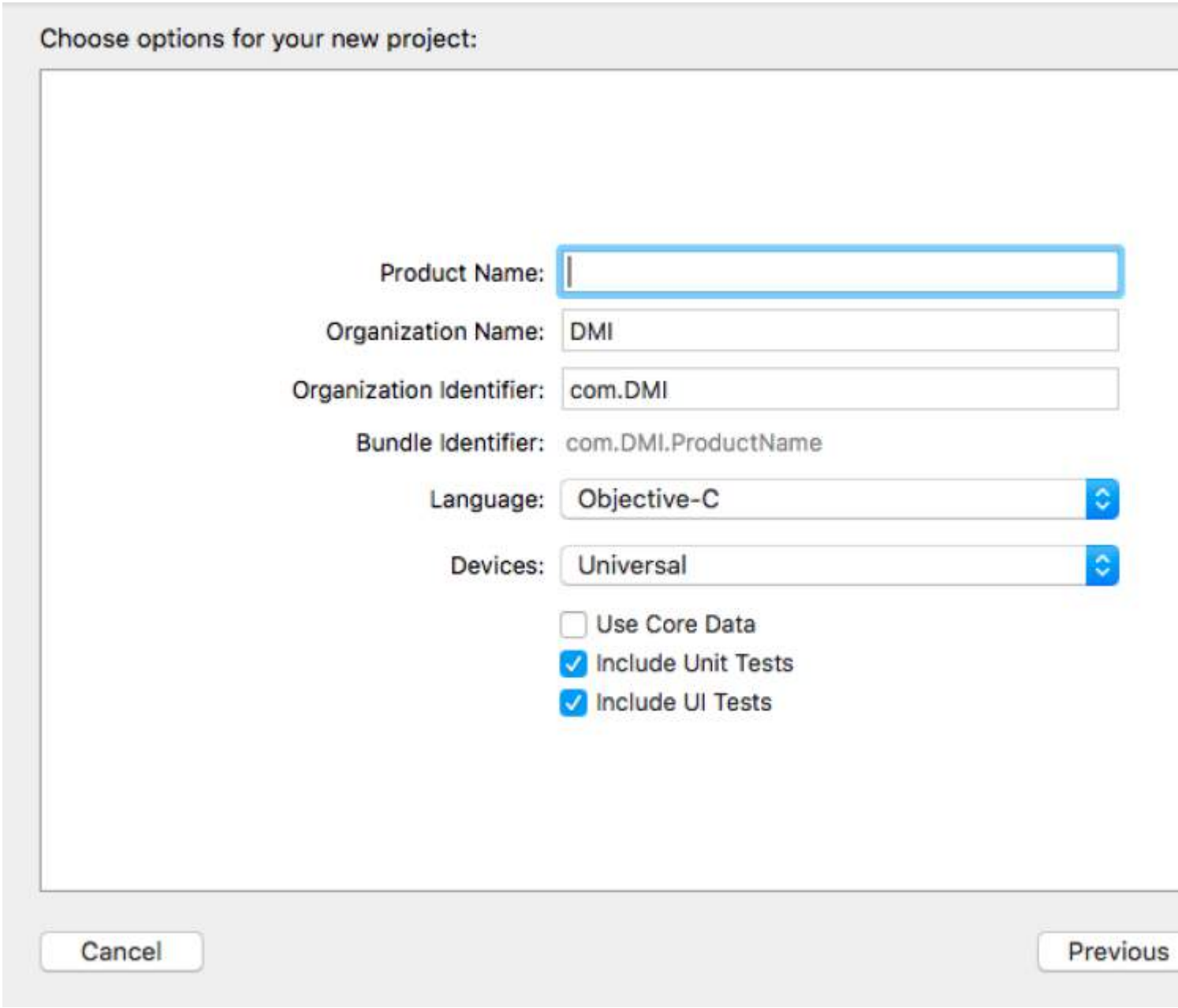
- Este bloque de medida de rendimiento se ejecuta 10 veces consecutivas y luego se calcula el promedio, y sobre la base de este resultado de rendimiento promedio se crea y se acepta la línea de base para una evaluación adicional.
- El resultado de rendimiento se compara con los resultados de la prueba anterior y la línea de base con una desviación estándar máxima personalizable.

Ejecución de trajes de prueba:

Ejecute todas las pruebas seleccionando Producto> Prueba. Haga clic en el icono de Test Navigator para ver el estado y los resultados de las pruebas. Puede agregar un objetivo de prueba a un proyecto (o agregar una clase a una prueba) haciendo clic en el botón Agregar (más) en la esquina inferior izquierda del navegador de prueba. Para ver el código fuente de una prueba en particular, selecciónelo de la lista de pruebas. El archivo se abre en el editor de código fuente.

Nota:

Asegúrese de que la casilla de inclusión de caso de prueba unitaria esté marcada al crear un nuevo proyecto como se muestra a continuación:



Lea Pruebas unitarias utilizando Xcode en línea: https://riptutorial.com/es/objective-c/topic/5160/pruebas-unitarias-utilizando-xcode

Capítulo 51: Singletons

Introducción

Solo asegúrate de leer este hilo (¿Qué tiene de malo Singleton?) Antes de usarlo.

Examples

Usando Grand Central Dispatch (GCD)

GCD garantizará que su singleton solo se ejemplifique una vez, incluso si se llama desde múltiples hilos. Inserta esto en cualquier clase para una instancia de singleton llamada `shared` .

```
+ (instancetype)shared {

    // Variable that will point to the singleton instance. The `static`
    // modifier makes it behave like a global variable: the value assigned
    // to it will "survive" the method call.
    static id _shared;

    static dispatch_once_t _onceToken;
    dispatch_once(&_onceToken, ^{

        // This block is only executed once, in a thread-safe way.
        // Create the instance and assign it to the static variable.
        _shared = [self new];
    });

    return _shared;
}
```

Crear una clase Singleton y también evitar que tenga múltiples instancias usando alloc / init.

Podemos crear la clase Singleton de tal manera que los desarrolladores se vean obligados a usar la instancia compartida (objeto singleton) en lugar de crear sus propias instancias.

```
@implementation MySingletonClass

+ (instancetype)sharedInstance
{
    static MySingletonClass *_sharedInstance = nil;
    static dispatch_once_t oncePredicate;
    dispatch_once(&oncePredicate, ^{
        _sharedInstance = [[self alloc] initClass];
    });

    return _sharedInstance;
}

-(instancetype)initClass
```

```
{
    self = [super init];
    if(self)
    {
        //Do any additional initialization if required
    }
    return self;
}

- (instancetype)init
{
    @throw [NSException exceptionWithName:@"Not designated initializer"
                                   reason:@"Use [MySingletonClass sharedInstance]"
                                 userInfo:nil];
    return nil;
}
@end



/*Following line will throw an exception
  with the Reason:"Use [MySingletonClass sharedInstance]"
  when tried to alloc/init directly instead of using sharedInstance */
MySingletonClass *mySingletonClass = [[MySingletonClass alloc] init];
```

Creando Singleton y también evitando que tenga múltiples instancias usando alloc / init, nuevo.

```
//MySingletonClass.h
@interface MYSingletonClass : NSObject

+ (instancetype)sharedInstance;

-(instancetype)init NS_UNAVAILABLE;

-(instancetype)new NS_UNAVAILABLE;

@end

//MySingletonClass.m

@implementation MySingletonClass

+ (instancetype)sharedInstance
{
    static MySingletonClass *_sharedInstance = nil;
    static dispatch_once_t oncePredicate;
    dispatch_once(&oncePredicate, ^{
        _sharedInstance = [[self alloc]init];
    });

    return _sharedInstance;
}
-(instancetype)init
{
    self = [super init];
    if(self)
    {
        //Do any additional initialization if required
```

```
    }
    return self;
}
@end
```

Lea Singletons en línea: https://riptutorial.com/es/objective-c/topic/2834/singletons

Capítulo 52: Suscripción

Examples

Subíndices con NSArray

Los subíndices se pueden usar para simplificar la recuperación y configuración de elementos en una matriz. Dada la siguiente matriz

```
NSArray *fruit = @[@"Apples", @"Bananas", @"Cherries"];
```

Esta línea

```
[fruit objectAtIndex: 1];
```

Puede ser reemplazado por

```
fruit[1];
```

También se pueden usar para establecer un elemento en una matriz mutable.

```
NSMutableArray *fruit = [@[@"Apples", @"Bananas", @"Cherries"] mutableCopy];
fruit[1] = @"Blueberries";
NSLog(@"%@", fruit[1]); //Blueberries
```

Si el índice del subíndice es igual al conteo de la matriz, el elemento se agregará a la matriz.

Se pueden usar subíndices repetidos para acceder a elementos de matrices anidadas.

```
NSArray *fruit = @[@"Apples", @"Bananas", @"Cherries"];
NSArray *vegetables = @[@"Avocado", @"Beans", @"Carrots"];
NSArray *produce = @[fruit, vegetables];

NSLog(@"%@", produce[0][1]); //Bananas
```

Subíndices con NSDictionary

Los subíndices también se pueden usar con NSDictionary y NSMutableDictionary. El siguiente código:

```
NSMutableDictionary *myDictionary = [@{@"Foo": @"Bar"} mutableCopy];
[myDictionary setObject:@"Baz" forKey:@"Foo"];
NSLog(@"%@", [myDictionary objectForKey:@"Foo"]); // Baz
```

Se puede acortar a:

```
NSMutableDictionary *myDictionary = [@{@"Foo": @"Bar"} mutableCopy];
myDictionary[@"Foo"] = @"Baz";
NSLog(@"%@", myDictionary[@"Foo"]); // Baz
```

Suscripción personalizada

Puede agregar subíndices a sus propias clases implementando los métodos requeridos.

Para subíndices indexados (como matrices):

```
- (id)objectAtIndexedSubscript:(NSUInteger)idx
- (void)setObject:(id)obj atIndexedSubscript:(NSUInteger)idx
```

Para los subíndices codificados (como diccionarios):

```
- (id)objectForKeyedSubscript:(id)key
- (void)setObject:(id)obj forKeyedSubscript:(id <NSCopying>)key
```

Lea Suscripción en línea: https://riptutorial.com/es/objective-c/topic/3825/suscripcion

Capítulo 53: Tipos de datos básicos

Sintaxis

- BOOL tenerPlutonio = SÍ; // Asignación directa
- BOOL fastEnough = (car.speedInMPH> = 88); // Expresión de comparación
- BOOL fluxCapacitorActive = (havePlutonium && fastEnough); // expresión booleana
-
- id somethingWicked = [witchesCupboard lastObject]; // Recuperar objeto sin tipo
- id powder = prepareWickedIngredient (somethingWicked); // Pasar y volver
- if ([ingrediente isKindOfClass: [clase Toad]]) {// tipo de tiempo de ejecución de prueba

Examples

BOOL

El tipo `BOOL` se utiliza para valores booleanos en Objective-C. Tiene dos valores, `YES` y `NO` , en contraste con los más comunes "verdadero" y "falso".

Su comportamiento es directo e idéntico al del lenguaje C.

```
BOOL areEqual = (1 == 1);    // areEqual is YES
BOOL areNotEqual = !areEqual    // areNotEqual is NO
NSCAssert(areEqual, "Mathematics is a lie");    // Assertion passes

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

Un `BOOL` es un elemento primitivo, por lo que no se puede almacenar directamente en una colección de Foundation. Debe estar envuelto en un `NSNumber` . Clang proporciona una sintaxis especial para esto:

```
NSNumber * yes = @YES;    // Equivalent to [NSNumber numberWithBool:YES]
NSNumber * no = @NO;    // Equivalent to [NSNumber numberWithBool:NO]
```

La implementación de `BOOL` se basa directamente en C's, ya que es un typedef del `bool` tipo estándar C99. Los valores de `YES` y `NO` están definidos para `__objc_yes` y `__objc_no` , respectivamente. Estos valores especiales son los compiladores incorporados introducidos por Clang, que se traducen a `(BOOL)1` y `(BOOL)0` . Si no están disponibles, `YES` y `NO` se definen directamente como la forma de entero de conversión. Las definiciones se encuentran en el encabezado de tiempo de ejecución de Objective-C objc.h

carné de identidad

`id` es el puntero de objeto genérico, un tipo Objective-C que representa "cualquier objeto". Una instancia de cualquier clase de Objective-C se puede almacenar en una variable de `id` . Un `id` y cualquier otro tipo de clase pueden asignarse de ida y vuelta sin lanzar:

```
id anonymousSurname = @"Doe";
NSString * surname = anonymousSurname;
id anonymousFullName = [NSString stringWithFormat:@"%@, John", surname];
```

Esto se vuelve relevante cuando se recuperan objetos de una colección. Los tipos de métodos de `objectAtIndex:` como `objectAtIndex:` son `id` por exactamente este motivo.

```
DataRecord * record = [records objectAtIndex:anIndex];
```

También significa que un parámetro de función o método escrito como `id` puede aceptar cualquier objeto.

Cuando un objeto se escribe como `id` , se le puede pasar cualquier mensaje conocido: el envío del método no depende del tipo de tiempo de compilación.

```
NSString * extinctBirdMaybe =
               [anonymousSurname stringByAppendingString:anonymousSurname];
```

Un mensaje al que el objeto no responde realmente causará una excepción en el tiempo de ejecución, por supuesto.

```
NSDate * nope = [anonymousSurname addTimeInterval:10];
// Raises "Does not respond to selector" exception
```

Protección contra excepción.

```
NSDate * nope;
if([anonymousSurname isKindOfClass:[NSDate class]]){
    nope = [anonymousSurname addTimeInterval:10];
}
```

El tipo de `id` está definido en objc.h

```
typedef struct objc_object {
    Class isa;
} *id;
```

SEL

Los selectores se utilizan como identificadores de método en Objective-C.

En el siguiente ejemplo, hay dos selectores. `new` y `setName:`

```
Person* customer = [Person new];
```

```
[customer setName:@"John Doe"];
```

Cada par de corchetes corresponde a un mensaje enviado. En la primera línea enviamos un mensaje que contiene el `new` selector a la clase `Person` y en la segunda línea enviamos un mensaje que contiene el `setName:` selector y una cadena. El receptor de estos mensajes utiliza el selector para buscar la acción correcta para realizar.

La mayoría de las veces, el paso de mensajes con la sintaxis de corchete es suficiente, pero ocasionalmente necesita trabajar con el propio selector. En estos casos, el tipo `SEL` se puede utilizar para mantener una referencia al selector.

Si el selector está disponible en tiempo de compilación, puede usar `@selector()` para obtener una referencia a él.

```
SEL s = @selector(setName:);
```

Y si necesita encontrar el selector en tiempo de ejecución, use NSSelectorFromString.

```
SEL s NSSelectorFromString(@"setName:");
```

Cuando use NSSelectorFromString, asegúrese de envolver el nombre del selector en una NSString.

Se usa comúnmente para verificar si un delegado implementa un método opcional.

```
if ([self.myDelegate respondsToSelector:@selector(doSomething)]) {
    [self.myDelegate doSomething];
}
```

IMP (puntero de implementación)

IMP es un tipo C que se refiere a la implementación de un método, también conocido como puntero de implementación. Es un puntero al inicio de la implementación de un método.

Sintaxis:

```
id (*IMP)(id, SEL, …)
```

IMP se define por:

```
typedef id (*IMP)(id self,SEL _cmd,…);
```

Para acceder a este IMP, se puede usar el mensaje **"methodForSelector"** .

Ejemplo 1:

```
IMP ImpDoSomething = [myObject methodForSelector:@selector(doSomething)];
```

El método dirigido por el IMP se puede llamar mediante la eliminación de la referencia del IMP.

```
ImpDoSomething(myObject, @selector(doSomething));
```

Así que estas llamadas son iguales:

```
 myImpDoSomething(myObject, @selector(doSomething));
[myObject doSomething]
[myObject performSelector:mySelector]
[myObject performSelector:@selector(doSomething)]
[myObject performSelector:NSSelectorFromString(@"doSomething")];
```

Ejemplo: 2:

```
SEL otherWaySelector = NSSelectorFromString(@"methodWithFirst:andSecond:andThird:");

IMP methodImplementation  = [self methodForSelector:otherWaySelector];

result = methodImplementation( self,
                          betterWaySelector,
                          first,
                          second,
                          third );

NSLog(@"methodForSelector : %@", result);
```

Aquí, llamamos a [NSObject methodForSelector, que nos devuelve un puntero a la función C que implementa el método, al que podemos llamar directamente.

NSInteger y NSUInteger

El NSInteger es solo un typedef para un int o un largo dependiendo de la arquitectura. Lo mismo ocurre con un NSUInteger que es un typedef para las variantes sin firmar. Si marca el NSInteger verá lo siguiente:

```
#if __LP64__ || (TARGET_OS_EMBEDDED && !TARGET_OS_IPHONE) || TARGET_OS_WIN32 ||
NS_BUILD_32_LIKE_64
typedef long NSInteger;
typedef unsigned long NSUInteger;
#else
typedef int NSInteger;
typedef unsigned int NSUInteger;
#endif
```

La diferencia entre un int firmado o no firmado o largo es que un int firmado o largo puede contener valores negativos. El rango del int es -2 147 483 648 a 2 147 483 647, mientras que el int sin firma tiene un rango de 0 a 4 294 967 295. El valor se duplica porque el primer bit ya no se usa para decir que el valor es negativo o no. Para un largo y NSInteger en arquitecturas de 64 bits, el rango es mucho más amplio.

La mayoría de los métodos que proporciona Apple están devolviendo un entero NS (U) sobre el int normal. Recibirá una advertencia si intenta convertirlo en un int normal porque perderá

precisión si está ejecutando en una arquitectura de 64 bits. No es que importe en la mayoría de los casos, pero es más fácil usar NS (U) Integer. Por ejemplo, el método de conteo en una matriz devolverá un NSUInteger.

```
NSNumber *iAmNumber = @0;

NSInteger iAmSigned = [iAmNumber integerValue];
NSUInteger iAmUnsigned = [iAmNumber unsignedIntegerValue];

NSLog(@"%ld", iAmSigned); // The way to print a NSInteger.
NSLog(@"%lu", iAmUnsigned); // The way to print a NSUInteger.
```

Al igual que un BOOL, el Entero NS (U) es un tipo de datos primitivo, por lo que a veces necesita envolverlo en un NSNumber, puede usar @ antes del entero para convertirlo como arriba y recuperarlo usando los métodos a continuación. Pero para convertirlo en NSNumber, también puedes usar los siguientes métodos:

```
[NSNumber numberWithInteger:0];
[NSNumber numberWithUnsignedInteger:0];
```

Lea Tipos de datos básicos en línea: https://riptutorial.com/es/objective-c/topic/4564/tipos-de-datos-basicos

Creditos

S. No	Capítulos	Contributors
1	Comenzando con el lenguaje Objective-C	Ali Riahipour, Community, connor, insys, J F, Jeff Wolski, Josh Brown, Josh Caswell, Niraj, StrAbZ, tbodt
2	Análisis XML	iphonic, Stephen Leppik
3	Bloques	BB9z, connor, danh, Fantini, insys, J F, Jeff Wolski, Johannes Fahrenkrug, Josh Caswell, Kote, Lito, Oliver Atkinson, Seán Labastille, tbodt, Yevhen Dubinin
4	BOOL / bool / Boolean / NSCFBoolean	Md. Ibrahim Hassan, user459460
5	Categorías	atroutt, DarkDust, Ekta Padaliya, Faran Ghani, Håvard, insys, Mykola Denysyuk, Orlando, Paulo Fierro, phi, WMios
6	Clases y objetos	Byron, DarkDust, HCarrasko, Jens Meder, Josh Caswell, NSNoob, ok404, RamenChef, tbodt
7	Codificación de valor clave / Observación de valor clave	Cory Wilhite, insys, Jason McDermott, Nirav Bhatt, ThatsJustCheesy, WMios
8	Continuar y romper!	Md. Ibrahim Hassan
9	Declare el método de clase y el método de instancia	Ruby, user459460
10	Entero aleatorio	Josh Caswell, jsondwyer
11	Entorno de tiempo de ejecución de bajo nivel	connor, DarkDust, dgatwood, Grady Player, Josh Caswell, orta, tbodt, Tricertops
12	Enumeración rápida	ff10, shuvo
13	Enums	Daniel Bocksteger, DavidA, Doc, Doron Yakovlev-Golani, lostInTransit, Mikhail Larionov, user459460
14	Especificadores de formato	Albert Renshaw

15	Estructuras	Doc, Sujania
16	Explotación florestal	Albert Renshaw, Chris Prince, connor, Daniel Bocksteger, DarkDust, DavidA, HariKrishnan.P, HCarrasko, Iulian Onofrei, Jason McDermott, Josh Caswell, Kornel, Nicolas Miari, NobodyNada, Peter N Lewis, Samet DEDE, Tapan Prakash, tbodt, Thomas Tempelmann, Tricertops, WMios
17	Gestión de la memoria	James P, ok404, Tamarous, tbodt, Tricertops
18	Grand Central Dispatch	user459460
19	Herencia	BKO
20	Macros predefinidas	user459460
21	Manejo de errores	Doc, Sujania
22	Métodos	Caleb Kleveter, Jon Schneider, Joshua, jsondwyer, mrtnf, Sujania, Tapan Prakash, tbodt
23	Multihilo	DarkDust, jsondwyer
24	NSArray	animuson, AnthoPak, Bharath, DarkDust, Ekta Padaliya, Evan, HCarrasko, Irfan, j.f., James P, Jeff Wolski, Johannes Fahrenkrug, Jon Schneider, Joost, Josh Caswell, Joshua, jsondwyer, Losiowaty, mrtnf, mszaro, Muhammad Zohaib Ehsan, njuri, NSNoob, Paulo Fierro, Ruby, tbodt
25	NSAttributedString	Patrick, Sujania, user459460
26	NSCache	user459460
27	NSCalendar	byJeevan, connor
28	NSData	Patrick, Sujania, WMios
29	NSDate	jsondwyer, Nikolai Ruhe, Patrick, Sujania, WMios
30	NSDiccionario	connor, Fantini, insys, Kevin Stewart, Mykola Denysyuk, Patrick, Sujania
31	NSJSONSerialización	connor, Sujania
32	NSMutableArray	aniket.ghode, animuson, DavidA, HCarrasko, J F, Joost, Ruby, Spidy, Sujania, tbodt, william205
33	NSMutableDictionary	Ravi Dhorajiya

34	NSObject	CodeChanger
35	NSPredicate	Dipen Panchasara
36	NSRegularExpression	Johannes Fahrenkrug
37	NSSortDescriptor	4444, Rahul
38	NSString	Albert Renshaw, animuson, AnthoPak, Cœur, DarkDust, Darshan Kunjadiya, David Mangon, il Malvagio Dottor Prosciutto, James P, Jeff Wolski, Johnny Rockex, Jon Schneider, Josh Caswell, Joshua, jsondwyer, Md. Ibrahim Hassan, Nikolai Ruhe, NSNoob, Orlando, Patrick, Paulo Fierro, RamenChef, Ruby, Srinivasan Saivenkat, Sunil Sharma, tbodt, Tricertops, ViratA, WMios
39	NSTextEnganche	BKO
40	NSTimer	Arc676, DarkDust, Hemang, Jason McDermott, Patrick, Ruby
41	NSURL	Patrick, Sujania
42	NSUrl enviar una solicitud de publicación	Md. Ibrahim Hassan
43	NSUserDefaults	Adriana Carelli
44	Objetivo moderno-C	pckill
45	Propiedades	DarkDust, dgatwood, Doc, J F, Nef10, NobodyNada, tbodt, Tricertops
46	Protocolos	Håvard, insys, jsondwyer, Mykola Denysyuk, Patrick, RamenChef, StrAbZ, tbodt, Tricertops
47	Protocolos y Delegados.	RamenChef, Sanjay Mohnani
48	Pruebas unitarias utilizando Xcode	ajmccall, Amon, Siddharth Sunil
49	Singletons	Amit Kalghatgi, Andrew Hoos, connor, Daniel Bocksteger, DarkDust, Glenn Smith, Itachi, pckill, Peter DeWeese
50	Suscripción	connor, tbodt
51	Tipos de datos básicos	connor, Josh Caswell, Muhammad Zohaib Ehsan, ok404, Sietse, sudo, Sujania, user459460