**Mac Operating System**

Every Mac comes with a collection of great apps for things you do every day, like Safari for surﬁng the web, Photos for managing your photos and videos, Pages for creating documents, Numbers for making spreadsheets and Keynote for preparing presentations. There are apps for sending email and text messages, taking notes, and staying up to date with your contacts and calendar. It even comes with the Mac App Store for finding new apps. Your Mac is more than fully featured, it’s fully loaded.



The Photos app for Mac makes it simple to manage a lifetime’s worth of photos and videos. Powerful yet easy-to-use editing tools let you edit like a pro even if you’re a beginner. And you can add to your toolkit with editing extensions from your favourite developers and access them within the Photos app. Photos has also been fine-tuned to make it even easier to manage your library. Now you can add a location to a single image or a group of photos, and you can batch change photo titles, descriptions and keywords. Naming your favourite people in Faces is faster with a streamlined workflow. And you can sort your albums — and the contents inside them — by date, title and more.



Turn 4K video into movie magic: Browse your clips easily, instantly share your favourite moments, and create Hollywood‑style trailers and beautiful movies at stunning 4K resolution. You can even start editing on iPhone or iPad, and finish on a Mac.

* Browse The home of home video: If you have an iPhone or iPad, not to mention a camcorder or digital camera, you probably have a lot of great video. View all your photos and video clips in the redesigned Media browser and when inspiration strikes, easily turn them into the next great movie or trailer.
* Make movies Rated S for simple: It’s never been easier to make it in the movies. Just choose the clips you want to use. Insert titles, add effects and create a full soundtrack with powerful tools that are as simple as drag and drop. iMovie even supports 4K video for stunning cinema-quality films. And that, ladies and gentlemen, is a wrap.
* iMovie everywhere Start on your iPhone. Then cut to your Mac: iMovie for Mac and iMovie for iOS are designed to work together. So now you can start a project on your iPhone, continue working on your iPad, and then add the final touches on your Mac. And with iMovie Theater, all your finished movies and trailers appear on all your devices. So you’ll always be the star of the show.
* Share Lights. Camera. Reaction: What’s a film without an audience? iMovie gives you even more ways to share with your most loyal fans. Post it on the web for everyone to see. Or email it to a select crowd. Get your movie out there, then sit back and take all the credit.



Make insightful — and beautiful — spreadsheets with Numbers. Drop your data into one of many stunning, Apple-designed templates. Easily add your own images, text and shapes to the flexible, free-form canvas. Create some quick calculations with the help of the formula panel. Then visualise the numbers with a dramatic interactive chart that animates your data.



Make insightful — and beautiful — spreadsheets with Numbers. Drop your data into one of many stunning, Apple-designed templates. Easily add your own images, text and shapes to the flexible, free-form canvas. Create some quick calculations with the help of the formula panel. Then visualise the numbers with a dramatic interactive chart that animates your data.

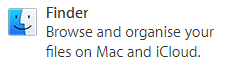


## The smartest way to surf.Safari is the best, fastest and most secure way to surf the web on your Mac, with all kinds of innovative features that make your browsing more enjoyable. Keep favourite websites open and accessible with Pinned Sites. Quickly mute audio without hunting for the tab it’s coming from. And use AirPlay to stream video from a web page to your HDTV with Apple TV. It’s also easy to share the cool pages you find. Just click the Share button to tweet, post to Facebook or send to a friend. Energy-saving technologies let you surf longer. And built-in privacy features protect you from being tracked as you go.

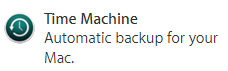


Disk Utility is the name of a utility, created by Apple, for performing disk-related tasks in Mac OS X. These tasks include:

* The creation, conversion, compression and encryption of disk images from a wide range of formats read by Disk Utility to .dmg or .cdr, which is identical to the .iso format
* Mounting, unmounting, and ejecting disks (including both hard disks, removable media and disk images)
* Enabling or disabling journaling
* Verifying a disk's integrity, and repairing it if the disk is damaged
* Verifying and repairing permissions
* Disk erasing, formatting and partitioning
* Secure deletion of free space or disk using a 35-pass Gutmann algorithm
* Adding or changing partition table between Apple Partition Table and GUID Partition Table
* Creating, destroying, and repairing RAID sets
* Restoring volumes from scanned for ASR images
* Burning disk images to CD or DVD in HFS+ format
* Erasing CD-RWs and DVD-RWs
* Checking the S.M.A.R.T status of a hard disk



The Finder is the app that helps you navigate all of the files and folders on your Mac. The Finder lets you browse your apps, disks, files, and folders in a variety of ways. You can use the Finder to organize these items the way you want. You can also use the Finder to search for items, delete files you no longer want, and more.



Back up using Time Machine. After you [set up Time Machine](https://support.apple.com/en-in/HT201250#setup), it automatically makes hourly backups for the past 24 hours, daily backups for the past month, and weekly backups for all previous months. The oldest backups are deleted when your backup drive is full.

* To back up now instead of waiting for the next automatic backup, choose Back Up Now from the Time Machine menu .
* To stop automatic backups, turn off Time Machine in Time Machine preferences. You can still back up manually by choosing Back Up Now from the Time Machine menu.
* To pause a backup, choose Stop Backup from the Time Machine menu. To resume, choose Back Up Now.
* To check backup status, use the Time Machine menu. The icon shows when Time Machine is backing up, idle until the next automatic backup, or unable to complete the backup.
* To exclude items from your backup, open Time Machine preferences from the Time machine menu, click Options, then click Add and select the item to exclude.

 **Search**

## Search with Spotlight. Use Spotlight to search for things like apps, documents, images and other files. In OS X Yosemite, Spotlight suggestions offer additional results like Wikipedia, news sites, Maps, iTunes, movie listings, and more.To open Spotlight, click the magnifying glass icon in the upper-right corner of the menu bar, or press Command-Space from any app.

Spotlight appears front and center when you open it. To search for something, type words related to it like an app name, a word contained in a document, or [a tag](http://support.apple.com/kb/HT202754) you've attached to a file. You can also search for items like a Wikipedia article, or the location of your nearest Apple Store.



The Dock holds your favorite apps, documents, and more. The Dock is a convenient place to access the apps you use the most. The Dock is the bar of icons that sits at the bottom or side of your screen. It provides easy access to many of the apps that come with your Mac (like [Mail](http://support.apple.com/kb/HT5361), [Safari](http://support.apple.com/kb/HT6074), and [Messages](http://support.apple.com/kb/HT5395)). You can add your own apps, documents and folders to the Dock, too.

## Organizing the Dock

## 

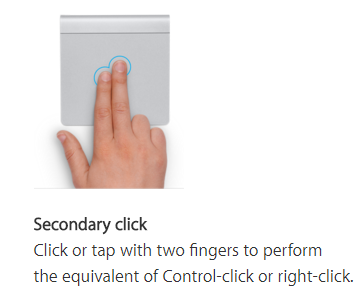
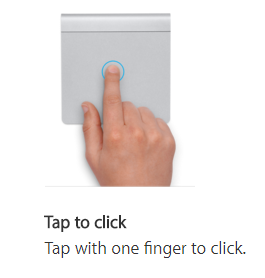
The Dock keeps apps on its left side. Folders, documents, and minimized windows are kept on the right side of the Dock. If you look closely, you can see a vertical separator line that separates these two sides.If you want to rearrange where an icon appears on the Dock, just drag it to another location in the Doc

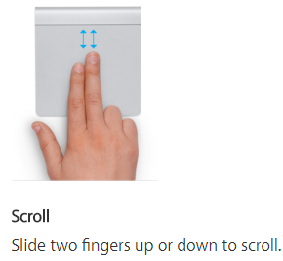


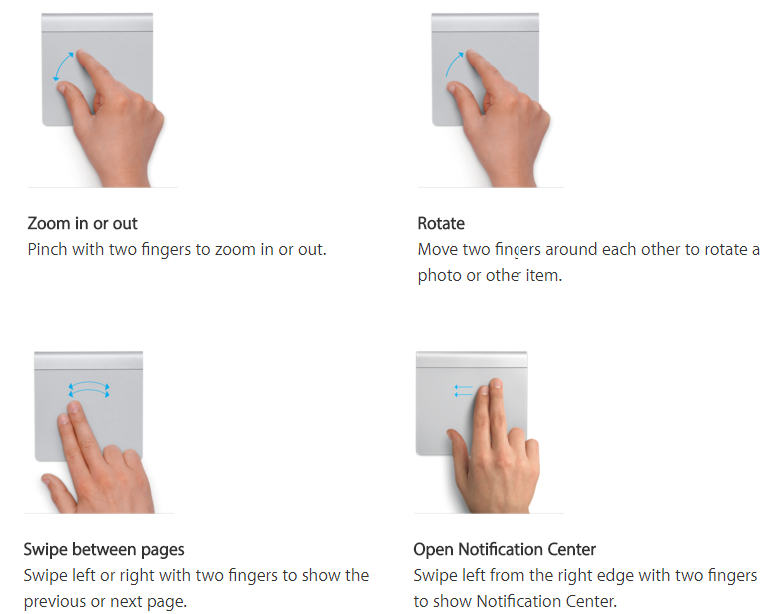
# If you can‘t unlock the Security & Privacy pane, make sure you‘ve set an admin password. You can't use admin accounts with blank passwords to unlock the Security & Privacy pane of System Preferences. You can't unlock the Security & Privacy pane if the admin account you're using has a blank password.



Use Multi-Touch gestures on your Mac. With a Multi-Touch trackpad or Magic Mouse, you can tap, swipe, pinch, or spread one or more fingers to perform useful actions.









Using Parental Controls preferences, People can manage, monitor, and control the time their kids spend on the Mac, the websites they visit, and the people they chat with.

**Objective C**

**Introduction:**

Objective C is a superset of C Language. It is the combination of C and the OOPS features. The C and Objective C are having same basic syntax of declaring and initializing the variables and the control statements etc. The simple Objective C program is as shown below:

#import<Foundation/Foundation.h>

int main()

{

//simple objective C program

NSLog(@"Hello World");

return 0;

}

Here the method NSLog(@" ") present in the library Foundation, hence we import it.

**Comments:**

Comments are used to give extra information about the block of code and are ignored by the compiler.

We have two ways to comment the text:

1) Inline Comment: A single line can commented.

//This is the inline comment

2) Block comment: More than one lines can be commented.

/\*This is Block comment.

We can write comment to more than one line\*/

**Constant :**

The keyword const is used to represent the constant, it tells the compiler that the variable is never allowed to change.

For eg: double const pi = 3.14159;

pi = 4.200000; //compile error

**Objective C Variables:**

* Variables are the container which holds the value and the variables gets the values statically and the programmer shoul mention the type of the value it should hold.
* The syntax for variable Declarartion : "<variable type><variable name>;".eg.: int aInt;
* We have different types of variables like integer type variables, floating type variables etc.
* The variables can be intialized with a value which is compatible.eg.: int bInt=3;
* The variable of one type can be casted to the other type, syntax is <target type><value>.
* The value of the variable is printed using the statement NSLog(@"string format");.eg.:to print the integer value we use: NSLog(@"%d");

**Objective C Primitive Datatypes:**

**1) BOOL:** It is primitive data type which checks whether true or false. It takes the values "YES" or "NO". If we use the string format "%d" it prints '0' for "NO" and '1' for "YES".

Eg: BOOL isbool = YES;

NSLog(@"%d", isbool);

NSLog(@"%@", isbool ? @"YES" : @ "NO");

**2) char:** It is primitive data type which holds both signed and unsigned the character value. The string format "%c" is used to print the single character value given to the variable, "%hhd" to print its ASCII value or "%hhu" to print the unsigned value given to the variable.

Eg: char aChar = 'a';

unsigned char uChar = 256;

NSLog(@"%c ASCII is @%hhd", aChar, aChar);

NSLog(@"%hhu", uChar);

**3) short:** It is primitive data type which holds both signed and unsigned the short value. The string format "%hd" is used to print the integer value given to the variable, "%hu" to print unsigned value given to the variable of type short.

Eg: short aShort = -56789;

unsigned = 256;

NSLog(@"%c ASCII is @%hhd", aChar, aChar);

NSLog(@"%hhu", uChar);

**4) int:** It is primitive data type which holds both signed and unsigned the integer value. The string format "%d" is used to print the integer value given to the variable, "%u" to print unsigned integer value.

Eg: int aInt = -67562;

unsigned int unInt = 76876;

NSLog(@"%d", aInt);

NSLog(@"%u", unInt);

**5) long:** It is primitive data type which holds both signed and unsigned long values. The string format "%ld" is used to print the long value given to the variable, "%u" to print unsigned long value.

Eg: long aLong = -675627864767;

unsigned long unLong = 76876876846;

NSLog(@"%ld", aLong);

NSLog(@"%lu", unLong);

**6) long long:** It is primitive data type which holds both signed and unsigned the long long value. The string format "%lld" is used to print the value given to the variable of type long long, "%llu" to print unsigned long long value.

Eg: long long aLongLong = -67562786476786565327;

unsigned long long unLongLong = 76876876846765475642;

NSLog(@"%lld", aLongLong);

NSLog(@"%llu", unLongLong);

**7) float:** It is primitive data type which holds both float value. The string format "%f" is used to print the value given to the variable of type float, "%8.2f" determines the padding and number of digits to be printed after decimal point.

Eg: float aFloat = -67.01f;

NSLog(@"%f", aFloat);

NSLog(@"%8.2f", aFloat);

**8) double:** By default the literal decimal values are double, we can explicitly mark it as float data type by adding 'f' at the end of the value assigning to the variable. It is primitive data type which holds double value. The string format "%8.2f" is used to print the value given to the variable of type double, "%e" is used to format the value as scientific notation.

Eg: double aDouble = -67.01;

NSLog(@"%8.2f", aDouble);

NSLog(@"%e", aDouble);

**9) long double:** By default the literal decimal values are double, we can explicitly mark it as long double by adding 'L' at the end of the value assigning to the variable. It is primitive data type which holds long double value. The string format "%Lf" is used to print the value given to the variable of type long double, "%Le" is used to format the value as scientific notation.

Eg: long double aLongDouble = -67.01e8L;

NSLog(@"%Lf", aLongDouble);

NSLog(@"%Le", aLongDouble);

**Arithmetic Operations:**

As similar to C programming we can perform the basic arithmetic operations such as, addition, subtraction, multiplication, division and modulus.

For eg.:NSLog(@"6+3=%d", 6+3); //9

NSLog(@"6-3=%d", 6-3); //3

NSLog(@"6\*3=%d", 6\*3); //18

NSLog(@"6/3=%d", 6/3); //2

NSLog(@"6%3=%d", 6%3); //0

Classes and Objects

* Classes are blueprint for creating objects.
* Class will be consisting of variables and methods.
* The members of the class can be accessed by creating the object of that class.
* Like C++, Objective C abstracts a class’s interface from its implementation.
* An **interface** declares the public properties and methods of a class, and the corresponding **implementation** defines the code that actually makes these properties and methods work.

Creating Class

* Consider we are working with the class called Car, its interface resides in Car.h and the implementation resides in Car.m.
* These are the standard file extensions used in Objective C Programming.
* Other classes use header file of class Car to interact with.
* Every Class is a sub class of NSObject, if we want to inherit the properties of other class we make it as Super class.

Interfaces

* Car.h contains some template code.
* The .h file declares the methods to be implemented.
* Interface is created using @interface directive.
* The @property directive declares a public property, and the (copy)attribute defines its memory management behavior.
* The methods declared in @interface are implemented in @implementation.

Implementation

* The first thing any class implementation needs to do is import its corresponding interface.
* It will be consisting of implementation of methods which are declared in interface.

Instantiation and Usage

* Any files that need access to a class must import its header file (Car.h) they should never, ever try to access the implementation file directly.
* The instantiation and usage will be done in main.m file.
* Interface has been imported with the #import directive, you can instantiate objects with the alloc/init as shown below:

Class\_name \*obj1 = [[Class\_name alloc]init];

* All objects must be stored as pointers.

Class Methods and Variables

* Instance level methods are prefixed with - sign and the Class level methods are prefixed with +sign.
* Similarly the class method implementation is also preceded by plus sign and instance method implementation is preceded by minus sign.
* The [parameter copy] call creates a copy of the parameter instead of assigning it directly.
* Class methods use the same square-bracket syntax as instance methods.

**Interface**

The interface of a class is usually defined in a header file. A common convention is to name the header file after the name of the class. E.g Ball.h would contain the interface for the class Ball.

An interface ClassName : Super Class Name

{

//instance variable

}

+ class Method1;

+ (return\_type) Class Method2;

+(return\_type)ClassMethod3:(param1\_type)param1\_var name;

-(return\_type)instanceMethod1:(param1\_type)param1\_varName:(param2\_type)

param2\_var name;

-(return\_type)instanceMethod2 With Parameter:(param1\_type)param1\_var name and other Parameter : (parameter2\_type)param2\_var\_name;

@end

‘+’ sign denotes class methods or methods that can be called without an instance of class.

‘-’sign denotes instance methods, which can only be called within a particular instance of the class.

Class methods also have no aceess to instance variables.

**Implementation**

The interface only declares the class interface and not the method themselves the actual code is written in the implementation file.

Implementation (method)files normally have the file extension.m, which originally signified “messages”.

@implementation ClassName

+ Class Method

{

//implementation

}

* Instance Method

{

//implementation

}

**If-else statement**

If statement checks the condition, if it is true then executes code within the if block, 'else' statement executes if the condition checked is false. The syntax is as follows:

if(condition)

{

//executes if the condition is true

}

else

{

//executes if the condition is false

}

The operators used are as below:

**Operator Description**

a==b Equal to

a!=b Not equal to

a>b Greater than

a>=b Greater than or equal to

a<b Less than

a<=b Less than or equal

a&&b Logical and

a||b Logical or

Example for if-else:

#import <Foundation/Foundation.h>

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

{

NSAutoreleasePool \* pool = [[NSAutoreleasePool alloc] init];

// find the grade of students

BOOL attend = YES;

int marks = 75;

if(!attend)

{

NSLog(@"ABSENT");

}

else

{

if(marks >=90 && marks < 100){

NSLog(@"S grade");

}

else if(marks >=75 && marks < 90){

NSLog(@"A grade");

}

else if(marks >=65 && marks < 75){

NSLog(@"B grade");

}

else if(marks >=55 && marks < 65){

NSLog(@"C grade");

}

else if(marks >=40 && marks < 55){

NSLog(@"D grade");

}

else{

NSLog(@"FAIL");

}

}

[pool drain];

return 0;

}

**For loop**

For loop is used for iterating over values. The loop continues till the condition is reached. Syntax for 'for' loop is as below:

for(i=0;i<5;i++)

{

//loop executes till the condition is reached

}

For eg:

#import <Foundation/Foundation.h>

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

{

NSAutoreleasePool \* pool = [[NSAutoreleasePool alloc] init];

int n=5;

int i;

for(i=0;i<n;i++)

{

NSLog(@"%d", i)

}

[pool drain];

return 0;

}

The above program prints the numbers less than 'n'.

**NSString**

The [NSString](https://developer.apple.com/library/mac/#documentation/Cocoa/Reference/Foundation/Classes/NSString_Class/Reference/NSString.html) class is the basic tool for representing text in an Objective-C. The string in Objective-C programming language is represented using NSString and its subclass NSMutableString provides several ways for creating string objects. String objects represent character strings in Cocoa frameworks.

An immutable string is a text string that is defined when it is created and subsequently cannot be changed. To create and manage an immutable string, use the NSString class. To construct and manage a string that can be changed after it has been created, use NSMutableString. Strings can be compared against one another, search them for substrings, combine them into new strings, and so on.

**Creating a String:**

The most common way to create strings is using the literal @"Some String" syntax, but the stringWithFormat: class method is also useful for generating strings that are composed of variable values. It takes the same kind of format string as NSLog():. A string object is an array of Unicode characters. We can use the @"%@" format specifier in the NSLog() call instead of passing the string directly with NSLog(message).

**For ex :**

#import <Foundation/Foundation.h>

int main ( )

{

NSString \*company = @"NeoRays";

NSLog(@"Company name: %@\n", company );

return 0;

}

**Enumerating Strings:**

The two most basic NSString methods are length andcharacterAtIndex:, which return the number of characters in the string and the character at a given index, respectively. length helps to find the length of given string and characterAtIndex helps to find the character at a perticular index.

**Comparing Strings:**

String comparisons present the same issues as NSNumber comparisons. Instead of comparing pointers with the == operator, The isEqualToString: method is used for a more robust value comparison. Also hasPrefix: and hasSuffix: methods can be used for partial comparisons.

**Combining Strings:**

The methods like stringByAppendingstring and stringByAppendingFormat are used to append two strings.

**Searching String:**

NSString’s search methods all return an NSRange struct, which defines a location and a length field. The location is the index of the beginning of the match, and the length is the number of characters in the match.

**Replacing Sustrings:**

stringByReplacingCharactersInRange method is used to replace a range of a string with a new string.

**Changing case:**

The NSString class also provides a few convenient methods for changing the case of a string. This can be used to normalize user-submitted values. Methods like lowercaseString, uppercaseString and capitalizedString are used for changing the cases of strings.

**String Methods:**

**1] - (NSString \*)capitalizedString;**

Returns a capitalized representation of the receiver.

**2] - (unichar)characterAtIndex:(NSUInteger)index;**

Returns the character at a given array position.

**3] - (double)doubleValue;**

Returns the floating-point value of the receiver’s text as a double.

**4] - (float)floatValue;**

Returns the floating-point value of the receiver’s text as a float.

**5] - (BOOL)hasPrefix:(NSString \*)aString;**

Returns a Boolean value that indicates whether a given string matches the beginning characters of the receiver.

**6] - (BOOL)hasSuffix:(NSString \*)aString;**

Returns a Boolean value that indicates whether a given string matches the ending characters of the receiver

**7] - (id)initWithFormat:(NSString \*)format ...;**

Returns an NSString object initialized by using a given format string as a template into which the remaining argument values are substituted.

**8] - (NSInteger)integerValue;**

Returns the NSInteger value of the receiver’s text.

**9] - (BOOL)isEqualToString:(NSString \*)aString;**

Returns a Boolean value that indicates whether a given string is equal to the receiver using a literal Unicode-based comparison.

**10] - (NSUInteger)length;**

Returns the number of Unicode characters in the receiver.

**11] - (NSString \*)lowercaseString;**

Returns lowercased representation of the receiver.

**12] - (NSRange)rangeOfString:(NSString \*)aString;**

Finds and returns the range of the first occurrence of a given string within the receiver.

**13] - (NSString \*)stringByAppendingFormat:(NSString \*)format ...;**

Returns a string made by appending to the receiver a string constructed from a given format string and the following arguments.

**14] - (NSString \*)stringByTrimmingCharactersInSet:(NSCharacterSet \*)set;**

Returns a new string made by removing from both ends of the receiver characters contained in a given character set.

**15] - (NSString \*)substringFromIndex:(NSUInteger)anIndex;**

Returns a new string containing the characters of the receiver from the one at a given index to the end.

**16] isEqualToString:**

 This method is used for a more robust value comparison.

**NSMutableString**

The [NSMutableString](https://developer.apple.com/library/mac/" \l "documentation/Cocoa/Reference/Foundation/Classes/NSMutableString_Class/Reference/Reference.html) class is a mutable version of NSString. Unlike immutable strings, it’s possible to alter individual characters of a mutable string without creating a new object. NSMutableString inherits from NSString, it can be manipulated like NSString.

NSMutableString \*car = [NSMutableString stringWithString:@"Porsche 911"]

The setString: method lets you assign a new value to the instance:

[car setString:@"Porsche Boxster"];

It’s possible to replace substring via the replaceCharactersInRange:withString: and delete the substring using deleteCharactersInRange: method.

**Creating and Intializing Mutable String:**

**1]** +stringWithCapacity

#### Declaration

+ (NSMutableString \*)stringWithCapacity:([NSUInteger](https://developer.apple.com/library/mac/documentation/Cocoa/Reference/Foundation/Miscellaneous/Foundation_DataTypes/index.html" \l "//apple_ref/doc/c_ref/NSUInteger))capacity

#### Parameters

Capacity : The number of characters the string is expected to initially contain.

#### Return Value

An empty NSMutableString object with initial storage for capacity characters.

2] -initWithCapacity:

- (NSMutableString \*)initWithCapacity:([NSUInteger](https://developer.apple.com/library/mac/documentation/Cocoa/Reference/Foundation/Miscellaneous/Foundation_DataTypes/index.html" \l "//apple_ref/doc/c_ref/NSUInteger))capacity

#### Parameters

Capacity : The number of characters the string is expected to initially contain.

#### Return Value

An initialized NSMutableString object with initial storage for capacity characters. The returned object might be different than the original receiver.

**NSArray**

It represents an ordered collection of objects, and it provides a high-level interface for sorting and otherwise manipulating lists of data. Immutable arrays can be defined as literals using the @[ ] syntax. objectAtIndex: method is used to access the element at a particular index.

**Comparing Arrays:**

Arrays can be compared for equality using the method namedisEqualToArray: , which returns YES when both arrays have the same number of elements and every pair pass an isEqual: comparison. NSArray does not offer the same subset and intersection comparisons as NSSet.

**Sorting Arrays:**

Sorting is one of the main advantages of arrays. thesortedArrayUsingComparator:  method. It is used to sort the array. This accepts an NSComparisonResult(id obj1, id obj2) block, which should return one of the following enumerators depending on the relationship between obj1 and obj2:

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

**Filtering Array:**

Array can be filtered with the filteredArrayUsingPredicate:method.

**Subdividing Array:**

Subdividing an array is essentially the same as extracting substrings from an NSString, but instead of substringWithRange:, we can use subarrayWithRange:.

**Combining Arrays:**

Arrays can be combined via arrayByAddingObjectsFromArray:.

**NSMutableArray**

In The NSMutableArray class dynamically items can be added or removed from arbitrary locations in the collection. it’s slower to insert or delete elements from a mutable array than a set or a dictionary.

**Creating Mutable Array:**

To create mutable arrays is still through the arrayWithObjects: method. You can create empty mutable arrays using the array orarrayWithCapacity: class methods. Or, if you already have an immutable array that you want to convert to a mutable one, you can pass it to the arrayWithArray: class method.

**Adding and Removing objects:**

The two basic methods for manipulating the contents of an array are the addObject: and removeLastObject methods. The former adds an object to the end of the array, and the latter is pretty self-documenting. Insert or delete objects at arbitrary locations usinginsertObject:atIndex: and removeObjectAtIndex:. It’s also possible to replace the contents of an index with thereplaceObjectAtIndex:withObject: method.

**NSDictionary**

NSDictionary class represents an unordered collection of objects; however, they associate each value with a key, which acts like a label for the value. This is useful for modeling relationships between pairs of objects.

**Creating Dictionaries:**

Immutable dictionaries can be defined using the literal @{} syntax. But, like array literals, this was added relatively recently, so you should also be aware of the dictionaryWithObjectsAndKeys: and dictionaryWithObjects:forKeys: factory methods.

The dictionaryWithObjectsAndKeys: method treats its argument list as value-key pairs, so every two parameters define a single entry. This ordering is somewhat counterintuitive, so make sure that the value always comes before its associated key. The dictionaryWithObjects:ForKeys: method is a little bit more straightforward, but we should be careful to ensure that the key array is the same length as the value array. It has same syntax as arrays (someDict[key]) to access the value for a particular key.

**Accessing Values and Keys:**

The objectForKey: method is the other common way to access values. It’s possible to do a reverse lookup to get a value’s key(s) with the allKeysForObject: method. Note that this returns an array because multiple keys can map to the same value.

**Enumerating Dictionaries:**

You can isolate a dictionary’s keys/values with the allKeys/allValuesmethods, which return an NSArray of each key/value in the collection, respectively. Note that there is no guarantee that these methods will return keys and values in the same order.

**Comparing Dictionaries:**

The isEqualToDictionary: method returns YES when both dictionaries contain the same key-value pairs:

**Sorting Dictionary keys:**

keysSortedByValueUsingComparator:, which accepts a block that should return one of the NSComparisonResult enumerators described in the [NSArray](http://rypress.com/tutorials/objective-c/data-types/nsarray.html" \l "sorting-arrays) module.

NSMutable Dictionary

The NSMutableDictionary class lets you add new key-value pairs dynamically. For example, an auto shop application might need to assign broken cars to specific mechanics. One way of modeling this is to treat cars as keys and mechanics as values.

Creating Mutable Dictionaries:

To convert a literal dictionary to a mutable one using dictionaryWithDictionary:.

**Adding and Removing Entries:**

The setObject:forKey: and removeObjectForKey: methods are the significant additions contributed by NSMutableDictionary. The former can be used to either replace existing keys or add new ones to the collection.

**Combining Dictionaries:**

Mutable dictionaries can be expanded by adding the contents of another dictionary to its collection via the addEntriesFromDictionary:method.

**Id type**

* It is generic type for all objects in objective C, This means that the compiler will expect any **object** type there, and will not enforce restrictions.
* It can store a reference to any type of object.

For eg:

id obj = @"Emp name is Bhagya";

NSLog(@"%@",[obj discription]);

obj = @{@"name":@"Bhagya",@"dob" : @"1991"};

NSLog(@"%@",[obj discription]);

* The id type automatically implies that the variable is a pointer, hence asterisk is not used.
* id is a pointer to any type, but unlike void \* it always points to an Objective-C object.
* For example, you can add anything of type id to an NSArray, but those objects must respond to retain and release.
* The compiler implicitly cast any object to id, and id to any object.
* This is unlike any other implicit casting in Objective-C, and is the basis for most container types in Cocoa.

Method overloading

* Method overloading cannot be done in Objective-C.
* When you overload the method, in interface class the compile time error will occur.

**Example:**

-(int) addNumbersMethod : (int)firstNumber : (int) secondNumber; //Line number 1

-(float) addNumbersMethod: (float)firstNumber : (float) secondNumber; //Line Number 2 //**error.**

* In the above example at second line it gives an error of declaration of duplicate methods.
* But method can be declared like as give below example.

**Example:**

-(int) addNumbersMethod : (int)firstNumber : (int) secondNumber; //Line number 1

-(int) addNumbersMethod : (int)firstNumber : (int) secondNumber : (int) thirdNumber;

//Line number 2

* In the above example 1 ,when you pass the parameter as,

[FirtsClass addNumbersMethod : 4 : 4]; it will execute the first method from Line number 1.

* In the above example 2 , when you pass the parameter as,

[FirtsClass addNumbersMethod : 2 : 3 : 4]; it will execute the second method from Line number 2.

Inheritance

* When creating a new class, instead of writing new data members and member functions we can modify the existing class using the concept called inheritance.
* The existing class is known as Base class(Super class) and the new class is called Derived class(Sub Class).
* Inheritance implements the is-a relation.
* Objective C supports Simple, Multilevel, Hierarchical inheritance, Multiple inheritance.

Simple Inheritance

* One new class or Sub class is inherited the properties of Existing Class is also called as Super Class.
* The Super class members can be accessed in the sub class with or without using Super keyword.

Multilevel Inheritance

* Sub Class inherits the properties of Super class and Sub class is the Super class of its sub class.
* Existing class will be the super of other sub classes indirectly.
* Any member of Existing class can be accessed in the sub classes.

Hierarchical Inheritance

* Here any number of classes can inherit the properties of existing class called Super class.
* Only one Super class is present other classes will be the sub classes of that class.

Multiple Inheritance

* Here one new class inherits the properties of more than one existing class.
* Multiple inheritance is not supported in Objective C.
* No more than one class can be a Super class of new class or sub class.
* It gives the compile time error if we try to inherit the properties of two existing classes.

Method overriding

* Method overriding is a language feature in which a class can provide an implementation of a method that is already provided by one of its parent classes.
* The implementation in this class replaces (that is, overrides) the implementation in the parent class.
* When you define a method with the same name as that of a parent class, that new method replaces the inherited definition.
* The new method must have the same return type and take the same number and type of parameters as the method you are overriding.

The following example shows the method overriding:

OverrideSup.h

#import <Foundation/Foundation.h>

@interface OverrideSup : NSObject

-(void) overMethod1;

@end

OverrideSup.m

#import "OverrideSup.h"

@implementation OverrideSup

-(void) overMethod1

{

NSLog(@"IN OVERMETHOD1");

}

OverrideSub.h

@interface OverrideSub : OverrideSup

-(void) overMethod1;

@end

OverrideSub.m

-(void) overMethod1 //Overriding the method overMethod1 from Super class

{

NSLog(@"IN SUB CLASS OVERMETHOD1");

}

@end

main.h

#import <Foundation/Foundation.h>

#import "OvrrideSup.h" //Import Super Class

#import "OverrideSub.h" //Import Sub Class

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

{

@autorelaesepool {

OverrideSub \*sub = [[OverrideSub alloc]init];

[sub overMethod1]; //executes overMethod1 of Sub class

}

return 0;

}

Selector

* Selectors are Objective-C’s internal representation of a method name.
* A selector is the name used to select a method to execute for an object, or the unique identifier that replaces the name when the source code is compiled.
* A selector by itself doesn’t do anything.
* It simply identifies a method.
* The only thing that makes the selector method name different from a plain string is that the compiler makes sure that selectors are unique
* What makes a selector useful is that (in conjunction with the runtime) it acts like a dynamic function pointer that, for a given name, automatically points to the implementation of a method appropriate for whichever class it’s used with
* Suppose you had a selector for the method run, and classes Dog, Athlete, and ComputerSimulation (each of which implemented a method run). The selector could be used with an instance of each of the classes to invoke its run method—even though the implementation might be different for each.

**Note:** You use this technique in special situations, such as when you implement an object that uses the target-action design pattern. Normally, you simply invoke the method directly.

**Getting a Selector**

* There are two ways to get the selector for a method name.
* The **@selector()**directive lets you convert a source-code method name to a selector, and the **NSSelectorFromString()** function lets you convert a string to a selector (the latter is not as efficient).
* Both of these return a special data type for selectors called SEL.
* You can use SEL the exact same way as BOOL, int, or any other data type.

Compiled selectors are of type SEL. There are two common ways to get a selector:

At compile time, you use the compiler directive @selector.

**SEL aSelector = @selector(methodName);**

At runtime, you use the NSSelectorFromString function, where the string is the name of the method:

**SEL aSelector = NSSelectorFromString(@"methodName");**

You use a selector created from a string when you want your code to send a message whose name you may not know until runtime.

**Examples:**

//Using Function NSSelectorFromString

SEL stepOne = NSSelectorFromString(@"startEngine");

//Using directive @selector

SEL stepTwo = @selector(driveForDistance:);

//Using directive @selector

SEL stepThree = @selector(turnByAngle:quickly:);a

References:

* <http://rypress.com/tutorials/objective-c/methods>
* <https://developer.apple.com/library/ios/documentation/General/Conceptual/DevPedia-CocoaCore/Selector.html>

Constructors

* Objective-C enables user to define constructor with the help of self and super keyword.
* It has a parent class and a programmer can access its constructor by statement [super init], this statement returns an instance of parent class, which we assign to the “self” keyword. Actually “self” plays the same role as “this” keyword in java statement.
* The default constructor is “-(id) init” statement
* If(self) is used to check the condition self! = nil to confirm that parent class has returned a new object successfully.

Properties

* Property is a robust way to handle Object’s data.
* We need to define a property if you want single piece of data (or a class object member) to be visible to other classes.

**Definition of simple Property**

**@**interface SimpleProperties : UIViewController

{

int count;

}

@property (read write) int count;

@end

In the class interface we need two things

1. Define a class variable as we normally do.
2. Define a property using the directive @property.

* In the above example ‘readwrite’ is property attribute.
* Property attribute you define decides how the property behaves.

After you define the property on the class interface, you need to also define the property implementation.

@implementation SimpleProperties

@synthesize count

@end

@synthesize will create automatically setter & getter methods for the property.

**Property Declarations:**

* Atomic:
* It means blocking access to accessors a single access both.
* If two threads try to modify the value of property, their access to property is not so simultaneous. Using this behavior helps to avoid potential problems (Ex Values not expected to pass coming two setters).
* On the other hand non atomic accessor render much factor, but make no warranty as to the simultaneous access of threads to access property.
* Both are similar and use multi-threading .In this case non has been selected for factor access and atomic for safer and robust access.
* Non atomic is not thread safe and multi-tasking is allowed.

**Strong:** Class/Object values.

* Reference count of object is always increasing i.e. memory is modified.

**Weak:** Primitive Values.

* Reference count of object is same i.e. memory remains unchanged.

**Copy:** Duplicates values at initialization.

**Assign:** Primitive types.

In .h file

@property (non atomic, Strong )NSString \*str;

In .m file

@implemetation

@synthesize str; // Compile time feature generate setter/getter methods.

Protocols

* Objective C was extended at NEXT to introduce the concept of multiple inheritances of specification, not implementation, through the introduction of protocols.
* This is a pattern achievable either as an abstract multiply-inherited base class in C++ or as an interface (as in java and C#).
* Objective C makes use of ad-hoc protocols, called informal protocols and compiler enforced protocols, called formal protocols.
* Informal protocol is a list of methods, which a class can opt to implement. It's specified in the documentation, since it has no presence in language.
* Informal protocols often include optional methods, where implementing the method can change the behavior of class.
* For Ex: a text field class might have a delegate which should implement an informal protocol with an optional auto complete method. The text field discovers whether the delegate implements that method (via reflection) and if so, calls it to support auto complete.
* Formal protocol is similar to an interface in java or C#. It's a list of methods, which any class can declare itself to implement.
* Versions of Objective C before 2.0 required that a class must implement all methods in a protocol, it declares itself as adopting, the compiler will send an error if the class does not implement every method of its declared protocols.
* Objective C 2.0 added support for making certain methods in a protocol optional, and the compiler will not enforce implementation of optional methods.
* The Objective C concept of protocols is different from the java or C# concept of interfaces in which a class may implement that protocol without being declared to implement that protocol.
* The difference is not detectable from outside code.
* Formal protocols cannot provide any implementation; they simply assure caller that classes which conform to the protocol will provide implementation.

**Defining a protocol:**

@protocol MyProtocolName <NSObject>

//Methods go here

@ends

Replace "MyProtocolName" with name of your choice .There are no curly Braces.

That is because variables go in curly braces, and protocols have Variables associated with them "<NSObject>" means that the amount of protocol is derivation of the NSObject Protocol.

There are Both NSObject class and NSObject Protocol. Pointed brackets are associated with protocols.

**using the Protocol:**

In Java we specify that a class implements an interface with the "implements" keyword. In Objective C we use pointy brackets in the interface declaration.

(In Objective C "interface” means part of the class in the header file or "interface"

In java also same), following the class you extend.

E.x: we usually declare class like this

@interface CustomView:UIView

To specify that it implements a protocol, simply change it to this:

@ interface CustomView:UIView <MyProtocol Name>.

**Protocol as variables**:

Here is where it differs from java the most. In java when declaring a variable, you would use an interface name just you would a class. In Objective -C you declare a variable this way:

id<My Protocol Name> myNewVariable

So the new type is "id<MyProtocolName>", id is the generic object even though it’s a pointer to an object, it does not have an asterisks it's assumed.

You can also use the notation when defining methods

Ex:

-(void)dosomethingWithThisObject :(id<My Protocol Name>)

CATEGORIES

* Categories collect method implementations into separate files.
* The programmer can place groups of related methods into a category to make them more reliable.
* For instances one could create a “Reverse String” category “on” String object, while collecting all of the methods related to Reverse String.
* It contains .h and .m files.
* The methods within a category are added to a class at runtime. Thus, categories permit the programmer to add method to an existing class without the need to recompile that or even have access to its source code.
* When we write Object Oriented Programs, we will often want to add some behavior to an existing class. There are always new hoops for objects to jump through.
* For ex: We might have designed a new kind of tire, so we would subclass Tire & add the new behavior. When we want to add behavior to an existing class, we usually create a sub class.
* But sometimes sub classing isn't convenient. For ex: we may want to add new behaviors to NSString, but we realize that NSString is really the front end for a class cluster, and So it's difficult to sub class.
* In other cases, you might be able to make a subclass, but you are using a toolkit or library that won't be able to handle objects of the new class.
* For Ex: SubClass of NSString won't be returned when we create a new string with the string with format class method. The dynamic run-time dispatch mechanism employed by the Objective-C term for those new methods is 'categories'.
* creating a category.
* A category is a way to add new method to existinfg classes thus can be done to any class, even classes we don't have the source code for.
* Let us say we are writing a cross word puzzle App that takes a series of strings, determines the length of each string, them puts those length into an NSArray or NSDictionary.

NSNumber \*number;

number = [NSNumber numberWithUnsignedInt:[Stringlength]];

//do something with number

@interface

//The declaration of a category looks a lot like the declaration for a class.

@interface NSString(NumberConvenience)

-(NSNumber \*) lengthAsNumber;

@end//Number Convenience

First, an existing class is mentioned, followed by a new name in parenthesis.

This means that the category is called "NumberConvenience", and it adds methods to NSString. Another way to say this is wwe are adding a category

onto NSString called NumberConvenience. You can add as many categories to a class as you want as long as the category names are unique.

We indicate the class you are putting the category onto (NSString),and the name of the category(Number convince),and list the methods you are adding ,following by @end.

Now instance variables cant be added ,so there is no instance variable section as there is with a class declaration .

@implementation

There is an @implementation companion to @interface

@implementation NSString (Number convince )

-(NSNumber \* )lengthAs Number

{

unsigned int length =[self length];

return ([NSNumber number withUnsigned int:length]);

}

@end

@implementation has the names of the class the category, along with the bodies of the new methods.

The lenghtAsNumber method gets the length of the string by calling [self length].This well be string to which you send the length AsNumber. then a new NSNumber is added with length.

numberWithUnsignedInt is not 'alloc', 'copy', or 'new' method.

The NSNumber Object we create will get cleaned up when the currently active auto release pool is destroyed.

**Limitations Of Categories:**

* New instance variable to a class can't created
* The second limitations concerns name collosion ,in which one the category methods has the name as an existing method.when names collide the category
* Category methods will completely replace the original method ,with no way of getting the original back.