**History Of Objective C**

Objective c is basically a object oriented language follows ANSI c syntax with methods from smalltalk.

Small talk was invented in 1981 by two men

1)Brad cox and

2)Tom Love

For the small talk Tom Love added the some OOPS concept and make it as final product i.e Objective c

In 1986 they realsed the objective c language from there company “StepStone”.

In 1986 Steve jobs acquire the licence of NextStep

He used the objective c Program to build the NextStep operating system.

Objective c interface made much easier for NextStep os

In 1995 NextStep get full permission for the objective c from step stone

**OPENSTEP API**

Developed in 1993 by Next step and sun.

**APPLE and MAC OS**

In 1996 NEXT is taken by APPLE put Steve jobs and his objective c libraries to work.

Redesigned the mac os 10 using objective c and released in 2001.

Developed a set of libraries named “coca” for GUI development.

**Dynamic Language**

Everything is done at runtime

Uses dynamic typing,linking and binding.

Minimizes the Ram and cpu usage.

**About Mac OS**

Force click with Force Touch track pad

The Force Touch track pad lets you Force click. You Force click by pressing on the trackpad and then applying more pressure. This allows you to take advantage of added functionality in many apps and system features on your Mac.

To see a video of a Force click, select the Apple menu and choose System Preferences. Then choose Trackpad, click the Point & Click tab, and hover the pointer over the Look up & data detectors checkbox.

What you can do with a Force click

Here are some examples what you can do with a Force click:

* **Look up**: Force click text in a webpage or Mail message to see more information about the text from sources like Dictionary, Wikipedia, and more.
* **Addresses**: Force click an address to see a Maps preview of that location.
* **Spotlight**: You'll feel a notch when moving the Spotlight search bar back to its standard horizontal or vertical position.
* **Preview**: You'll feel a notch when you align shapes, text, and other markup elements with each other.
* **Photo arrowing**: When you arrow through Photos in an Album or a Moment, you can apply additional pressure to go faster.
* **Rotate photos**: In Photos, when you choose Crop and then rotate a photo, and you’ll feel a notch when the rotation of the photo is at zero degrees.
* **File icons**: Force click a file icon to see a Quick Look preview of the file's content.
* **File names**: Force click a file name in the Finder or on your desktop to edit the file name.
* **Dock**: Force click an app in the Dock to access App Exposé. This shows you all open windows for that app.

# Turn on "three finger drag" for your Force Touch trackpad

"Three finger drag" is a [Multi-Touch gesture](https://support.apple.com/en-us/HT4721) in OS X. It lets you use three fingers to move the active window on your screen.

Here's how to turn on three finger drag if your Mac has a [Force Touch trackpad](https://support.apple.com/en-us/HT204352).

1. From the Apple menu, choose System Preferences.
2. Click the Accessibility icon.
3. Choose Mouse & Trackpad from the options on the left.
4. Click Trackpad Options.
5. Place a checkmark next to "Enable dragging."
6. From the related pop-up menu, choose “three finger drag” so there's a checkmark next to it.
7. Click OK.

## What is the Finder?

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. To open a new Finder window, click the Finder icon in the Dock, then select File > New Window. The first window you may see in the Finder is All My Files. This is a special window that shows you all of the documents that you have access to on your Mac.

## Window views

You can change how items are displayed in Finder windows by dragging them, arranging them, or changing their view. To change the view of the current Finder window, use the View menu in the menu bar or the View buttons in the window's [toolbar](https://support.apple.com/en-us/HT201732#toolbar). You can view as [icons](https://support.apple.com/en-us/HT201732#icons), as [list](https://support.apple.com/en-us/HT201732#list), as [colums](https://support.apple.com/en-us/HT201732#columns), or as [Cover Flow](https://support.apple.com/en-us/HT201732#coverflow).

You can also arrange items within the each view. For example, to sort the items in a window alphabetically, choose View > Arrange By > Name. The Finder keeps the window organized this way until you choose another option from the View menu.

### View as icons

Choose View > as Icons to see a small image that represents each file. You can move each item by dragging the icon that represents the file.

### View as list

Choose View > as List to see the items in a consecutive order. You can change the sort order of the list by clicking the headers (Name, Date Modified, Size, Kind) at the top of the list. Click the same header a second time to switch between ascending and descending order.

If an item is within a folder, you can click the triangle next to the folder name to see any items stored within that folder.

## Use the search field

## Use the search field to find files and folders on your Mac and any drives that are available. You can narrow your search further by selecting a search token for your search. Search tokens such as "Name matches" appear as you enter a term.

## 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.

### Search System Preferences

To open System Preferences, choose Apple menu > System Preferences.

1. In the search box, type a word or phrase that describes what you want to do. For example, if you want to share a printer with other computers, type “share printer.”

As you type, possible matches for what you’re looking for appear below the search box, and one or more preference icons are spotlighted in the System Preferences window.

1. Click the item in the list that matches what you want to do. The appropriate preference pane opens.

### Customize your view of System Preferences

Do any of the following:

Choose how to group the icons: Choose View > Organize by Categories, or View > Organize Alphabetically.

Show or hide an icon: Choose View > Customize, then select or deselect an icon. When you’re finished, click Done.

**Set up Time Machine**

Time Machine is the built-in backup feature of OS X. To use it, you need one of these external storage solutions, sold separately:

* External hard drive connected to a USB, FireWire, or Thunderbolt port on your Mac
* Time Capsule or OS X Server on your network
* External hard drive connected to the USB port of an AirPort Extreme (802.11ac) base station on your network

When you connect an external hard drive directly to your Mac, you might be asked if you want to use the drive to back up with Time Machine. Click “Use as Backup Disk.” If you select the option to encrypt, your backups will be accessible only to users with the password.

**Parental Control**

As a parent, you want your kids to have a safe and happy experience on the Mac. Using Parental Controls preferences, you can manage, monitor, and control the time your kids spend on the Mac, the websites they visit, and the people they chat with.

### Turn on parental controls

### Set restrictions

1. Choose Apple menu Choose Apple menu > System Preferences, then click Parental Controls.

Note:   When you open Parental Controls preferences, if you see the message “There are no user accounts to manage,” see [Add a managed user](https://support.apple.com/kb/index?page=link&apdid=mh35530&viewlocale=en_US&bookId=Mac%20Help135122).

1. Click the lock icon  to unlock it, then enter an [administrator](https://support.apple.com/kb/index?page=link&apdid=mchl17f139fc&viewlocale=en_US&bookId=Mac%20Help135122) name and password.
2. Select a user, then click Enable Parental Controls.

If the user isn’t in the list, click Add , then fill in the name, account, and password information to create a new user.

1. > System Preferences, then click Parental Controls.

Note:   When you open Parental Controls preferences, if you see the message “There are no user accounts to manage,” see [Add a managed user](https://support.apple.com/kb/index?page=link&apdid=mh35530&viewlocale=en_US&bookId=Mac%20Help135122).

1. Click the lock icon  to unlock it, then enter an administrator name and password.
2. Select a user, then click one of the tabs along the top.
   * Apps: Specify which apps the child can access. If you allow the child to access the App Store, you can specify a permitted app rating so the child only sees age-appropriate apps. You can also set up a simplified Finder for an inexperienced user.
   * Web: Limit access to websites, or allow unrestricted access.
   * People: Restrict a child’s contact with other people through Game Center, Mail, and Messages.
   * Time Limits: Set time limits for weekdays, weekends, and bedtime.
   * Other: Hide profanity in the dictionary and other sources, and block using the built-in camera, Dictation, burning CDs and DVDs, or changing the password or printer settings.

### Manage parental controls from another Mac

After you set restrictions for a child using a Mac, you can manage parental controls from a different Mac. Both computers must be on the same network.

1. On the Mac the child uses, choose Apple menu > System Preferences, then click Parental Controls.

Note:   When you open Parental Controls preferences, if you see the message “There are no user accounts to manage,” see [Add a managed user](https://support.apple.com/kb/index?page=link&apdid=mh35530&viewlocale=en_US&bookId=Mac%20Help135122).

1. Click the lock icon  to unlock it, then enter an administrator name and password.

Don’t select the child’s account at this time.

1. Select “Manage parental controls from another computer.”
2. On the Mac that will manage the child’s computer, choose Apple menu > System Preferences, then click Parental Controls.
3. Click the lock icon  to unlock it, then enter an administrator name and password.
4. Select the user to be managed.
5. You can now change the child’s parental controls settings and monitor the activity logs.

### Reuse parental controls settings

You can copy a user’s parental controls settings and apply them to another user.

1. Choose Apple menu > System Preferences, then click Parental Controls.

Note:   When you open Parental Controls preferences, if you see the message “There are no user accounts to manage,” see [Add a managed user](https://support.apple.com/kb/index?page=link&apdid=mh35530&viewlocale=en_US&bookId=Mac%20Help135122).

1. Click the lock icon  to unlock it, then enter an administrator name and password.
2. Select the user whose settings you want to copy.
3. Click the Action pop-up menu , then choose Copy Settings.
4. Select the user to whom you want to apply the copied settings.
5. Click the Action pop-up menu , then choose Paste Settings.

### Turn off parental controls

1. Choose Apple menu > System Preferences, then click Parental Controls.

Note:   When you open Parental Controls preferences, if you see the message “There are no user accounts to manage,” see [Add a managed user](https://support.apple.com/kb/index?page=link&apdid=mh35530&viewlocale=en_US&bookId=Mac%20Help135122).

1. Click the lock icon  to unlock it, then enter an administrator name and password.
2. Select the user, click the Action pop-up menu , then choose Turn off Parental Controls.

**Objective c**

**Introduction**:

Objective c is the native language for apple’s iOS and OS X operating system. Objective C was designed to add object-oriented features to C language. Objective C is the super specification of C.

**Comments:**

In objective C there are two ways to declare comments. They are Inline comments and Block comments. Inline comments are the single line comments are begin with //. The Block comments are written inside /\* and \*/. These comments are ignored by the compiler. Comments are used to give the extra information.

**Conditional Statements**

The Conditional statements are used to execute a statement or a group of statement based on certain condition. If is the most used conditional statement.

Syntax:

If(condition)

{

Valid statement

}

Eg:

If(a>b)

{

NSLog(@”a is greator than b”);

}

**If else condition:**

In if else if the condition is true the statements between if and else is executed. If it is false the statement after else is executed.

Syntax:

If(condition)

{

Statement1

}

Else

{

Statement2

}

Eg:

If(a>b)

{

NSLog(@”a is greater than b”);

}

Else

{

NSLog(@”b is greater than a”);

}

**Else if:**

The else if condition contains the multiple blocks of if and else to check the conditions.

Syntax:

If(condition)

{

Statement

}

Else if(condition)

{

Statement

}

Else if(condition)

{

Statement

}

Else

{

Statement

}

Eg:

If(a>b)

{

NSLog(@”a is greater than b”);

}

Else if(b>c)

{

NSLog(@”b is greater than c”);

}

Else if(c>d)

{

NSLog(@”c is greater than d”);

}

Else

{

NSLog(@”d is high value”);

}

**Operators:**

In conditional statement we use different operators to check the different conditions.

The operators are.

|  |  |
| --- | --- |
| operators | Description |
| a == b | Equals to |
| a !=b | Not equals to |
| a>b | A is greator than n |
| a<b | A is less than b |
| a>=b | a>=b |
| a<=b | A<=b |
| !a | Logical negation |
| a&&b | Logical and |
| a||b | Logical or |

**Looping statements**

**For loop:**

If you want to execute group of statements repeatedly then we can use for loop. Group of statements will be executed until the specified condition becomes false.

**Syntax:**

For(initialization;condition;increment/decrement)

{

//Statements;

}

Eg:

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

{

NSLog(@"%d",i);

}

**Data Types**

1. **Id type :**

The id type is one of the data type. It is designed to be generic type which can hold any object. The id type is typically typed (cast) to a specific type after using the introspection methods. The id type is predefined as pointer type there’s no need to add asterisk.

**Syntax:**

Id some object;

2>**NSString:**

The Objective-C class for strings is NSString. Strings are typically created by direct assignment or by calling one of the NSString class methods. The NSString class provides an extensive set of APIs for working with strings, including methods for comparing, searching, and modifying strings. Just like other languages, strings are enclosed and defined by the use of “double quotes”, however in Objective-C an NSString also comes with the prefix of the @ sign.

Syntax:

NSString \*variable name = @”statement”;

E.g.:

NSString string1 \*stirng1 = @”This is NSString”;

Inherits from:

NSObject

NSString

**Comparing String:**

NSString uses the isEqualToString method to compare the two strings and also with this it uses hassuffix and hasprefix methods for partial comparison.

Eg:

NSString \*bus = @”KSRTC red bus”;

If ([bus isEqualToString:@”This is red bus”])

{

NSLog(@”The bus is red”);

}

If ([bus hasprefix:@”KSRTC”])

{

NSLog(@“it is the ksrtc bus of some color”);

}

If ([bus hassuffix:@”bus”])

{

NSLog(@“this is the bus”);

}

**Combining the String:**

NSString is an immutable type so whenever we concatenate the new strings will be created. It uses the two methods to combine the strings they are 1)stringByAppendingString and 2)stringByAppendingFormat.

Eg:

NSString \*fname = @”Harish”;

NSString \*lname = @”Bhardwaj”;

NSString \*name = [ fname stringByAppendingString lname ]

NSLog(@”%@”,name);//HarishBhardwaj

NSString \*name = [ fname stringByAppendingFormat lname ]

NSLog(@”%@”,name);//Harish Bharadwaj

**Searching String:**

The NSString search method returns the NSRange which defines a location and length field. The Location contains the beginning match of the string and length field has toatal number of characters present in the string. if no match was found, location will contain NSNotFound.

**Changing case:**

The NSString change case is used to convert the string from lower case to upper case and upper case to lower case.

Eg:

NSString \*name = @”NeoRays”;

NSLog(@”%@”, [name uppercaseStirng]);//NEORAYS

NSLog(@”%@”, [ name lowercaseString]);//neorays

**2>NSMutabaleString:**

NSObject

NSString

NSMutableString

The NSMutableString class is the mutable version of NSString. The NSMutableString will not create a new string for any changes made for the string. There are so many methods are there to support for the NSMutableString. The mutable string is created with StringewithString Class method. Which turns the NSString object into mutable String

Eg:

NSMutableString \*name = [NSMutableString StringwithString :@”Harish”];

After created a mutable string we can assign the new value for the string using setString method.

**3>NSArray:**

NSObject

NSArray

NSArray is Objective-C’s general-purpose array type. It represents an ordered collection of objects.  NSArray creates static arrays. Immutable arrays can be defined as literals using the @[] syntax.

Eg:

NSArray \*fruits = @ [@”mango”,@”apple”,@”orange”];

NSArray \*colors = @[NSArray arraywithobjects:@”yellow”,@”blue”@”red”];

NSLog(@”The fruits are %@”,fruits[0]);

NSLog(@”The colors are:%@”,[colors objectAtIndex:0]);

* **Comparing arrays:**

The arrays can be compared to check the equality of two different arrays. The array object use isEqualToArray method to compare the two arrays. If the two arrays elements are equal then it returns YES.

Eg:

NSArray \*fruits = @ [@”mango”,@”apple”,@”orange”];

NSArray \*colors = @[NSArray arraywithobjects:@”yellow”,@”blue”@”red”];

If([fruits isEqualToArray:colors])

{

NSLog(@”the given arrays are equal “);

}

**4>NSMutableArray:**

NSObject

NSArray

NSMutableArray

The easiest way to create mutable arrays is arrayWithObjects:method we can create the mutable empty array using ArrayWithCapacity: class method.

Eg:

NSMutableArray \*city = [NSMutableArray arrayWithObjects:@”hassn”,

@”banglore”,@”manglore”]);

* addObject and removeLastObject: adds the elements to the end of the array.
* insertObject:atIndex and removeObjectAtIndex: if we know the index of the object we use remove fromObjectatindex .if we don’t know the particular index means we will use the removeObject:method
* replaceObjectAtIndex:withObject: used to replace the content of the objects.

**5>NSDictionary:**

The NSDictionary represents the unordered collection of Objects. The NSDictionary Contains the key and a value pair. The NSDictionary is immutable class. The NSDictionary can be declared using the literals @{} syntax.

**NSArray:**

NSObject

NSArray

NSArray is Objective-C’s general-purpose array type. It represents an ordered collection of objects.  NSArray creates static arrays. Immutable arrays can be defined as literals using the @[] syntax.

NSArray and its subclass [NSMutableArray](https://developer.apple.com/library/mac/documentation/Cocoa/Reference/Foundation/Classes/NSMutableArray_Class/index.html#//apple_ref/occ/cl/NSMutableArray) manage ordered collections of objects called **arrays**. NSArraycreates static arrays, and NSMutableArray creates dynamic arrays. You can use arrays when you need an ordered collection of objects.

NSArray is “toll-free bridged” with its Core Foundation counterpart, [CFArrayRef](https://developer.apple.com/library/mac/documentation/CoreFoundation/Reference/CFArrayRef/index.html#//apple_ref/c/tdef/CFArrayRef). See [Toll-Free Bridging](https://developer.apple.com/library/mac/documentation/General/Conceptual/CocoaEncyclopedia/Toll-FreeBridgin/Toll-FreeBridgin.html#//apple_ref/doc/uid/TP40010810-CH2) for more information on toll-free bridging.

Eg:

NSArray \*fruits = @ [@”mango”,@”apple”,@”orange”];

NSArray \*colors = @[NSArray arraywithobjects:@”yellow”,@”blue”@”red”];

NSLog(@”The fruits are %@”,fruits[0]);

NSLog(@”The colors are:%@”,[colors objectAtIndex:0]);

* **Comparing arrays:**

The arrays can be compared to check the equality of two different arrays. The array object use isEqualToArray method to compare the two arrays. If the two arrays elements are equal then it returns YES.

Eg:

NSArray \*fruits = @ [@”mango”,@”apple”,@”orange”];

NSArray \*colors = @[NSArray arraywithobjects:@”yellow”,@”blue”@”red”];

If([fruits isEqualToArray:colors])

{

NSLog(@”the given arrays are equal “);

}

**4>NSMutableArray:**

NSObject

NSArray

NSMutableArray

The easiest way to create mutable arrays is arrayWithObjects:method we can create the mutable empty array using ArrayWithCapacity: class method.

The NSMutableArray class declares the programmatic interface to objects that manage a modifiable array of objects. This class adds insertion and deletion operations to the basic array-handling behavior inherited from [NSArray](https://developer.apple.com/library/mac/documentation/Cocoa/Reference/Foundation/Classes/NSArray_Class/index.html#//apple_ref/occ/cl/NSArray).

There is typically little reason to subclass NSMutableArray. The class does well what it is designed to do—maintain a mutable, ordered collection of objects. But there are situations where a custom NSArray object might come in handy. Here are a few possibilities:

* Changing how NSMutableArray stores the elements of its collection. You might do this for performance reasons or for better compatibility with legacy code.
* Acquiring more information about what is happening to the collection (for example, statistics gathering).

Eg:

NSMutableArray \*city = [NSMutableArray arrayWithObjects:@”hassn”,

@”banglore”,@”manglore”]);

* addObject and removeLastObject: adds the elements to the end of the array.
* insertObject:atIndex and removeObjectAtIndex: if we know the index of the object we use remove fromObjectatindex .if we don’t know the particular index means we will use the removeObject:method
* replaceObjectAtIndex:withObject: used to replace the content of the objects.

**NSDictionary**

The NSDictionary represents the unordered collection of Objects. The NSDictionary Contains the key and a value pair. The NSDictionary is immutable class. The NSDictionary can be declared using the literals @{} syntax.

The NSDictionary class declares the programmatic interface to objects that manage immutable associations of keys and values. Use this class or its subclass [NSMutableDictionary](https://developer.apple.com/library/mac/documentation/Cocoa/Reference/Foundation/Classes/NSMutableDictionary_Class/index.html#//apple_ref/occ/cl/NSMutableDictionary) when you need a convenient and efficient way to retrieve data associated with an arbitrary key. NSDictionary creates static dictionaries. A key-value pair within a dictionary is called an entry. Each entry consists of one object that represents the key and a second object that is that key’s value. Within a dictionary, the keys are unique. That is, no two keys in a single dictionary are equal (as determined by [isEqual:](https://developer.apple.com/library/mac/documentation/Cocoa/Reference/Foundation/Protocols/NSObject_Protocol/index.html#//apple_ref/occ/intfm/NSObject/isEqual:)). In general, a key can be any object (provided that it conforms to the NSCopying protocol—see below), but note that when using key-value coding the key must be a string (see [Key-Value Coding Fundamentals](https://developer.apple.com/library/mac/documentation/Cocoa/Conceptual/KeyValueCoding/Articles/BasicPrinciples.html#//apple_ref/doc/uid/20002170)). Neither a key nor a value can be nil; if you need to represent a null value in a dictionary, you should use [NSNull](https://developer.apple.com/library/mac/documentation/Cocoa/Reference/Foundation/Classes/NSNull_Class/index.html#//apple_ref/occ/cl/NSNull).

Eg:

NSDictionary \*Inventory = @{

@”swift “:[NSNumber numberWithInt:10],

@”Alto”:[NSNumber numberWithString:21],

@”Baleno”:[NSNumber numberWithString:34],

@”Maruti 800”:[NSNumber numberWithString:19],

//values and keys as arguments

Inventory = [NSDictionary dictionaryWithObjectsAndkeys:

[NSNumber numberWithInt:10],@”swift”,

[NSNumber numberWithint:21],@”Alto”,

[NSNumber numberWithInt:34],@”Baleno”,

[NSNumber numberWithInt:19],@”Marutin800”];

//Values and Keys as Arrays

NS Array \*model = @[@”swift”,@”Alto”,@”Baleno”,@”Maruti 800”];

NSArray \*stock =@[ [NSNumber numberWithInt:10],

[NSNumber numberWithint:21],

[NSNumber numberWithInt:34],

[NSNumber numberWithInt:19]];

Inventory = [NSDictionary dictionaryWithObjects:stock forKeys:models];

NSLog(@”%@”,inventory);

For common programming tasks, we’ll probably never need to use anything but an NSString as a key, but it’s worth noting that any object that adopts the NSCopying protocol and implements the hash and theisEqual: methods can be used as a key. This is necessary because keys are copied when an entry is added to the array. However, this is not true for values, which are strongly referenced (just like set and array elements).

* Sorting Dictionary Keys

Dictionaries can’t be directly sorted into a new NSDictionary instance, but it is possible to sort the keys of the dictionary withkeysSortedByValueUsingComparator:, 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#sorting-arrays) module. The following example sorts the models from most expensive to most affordable.

## Enumerating Dictionaries

## As with sets and arrays, fast-enumeration is the most efficient way to enumerate a dictionary, and it loops through the keys (not the values).NSDictionary also defines a count method, which returns the number of entries in the collection.

## We 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

## Comparing dictionaries works the same as comparing arrays. TheisEqualToDictionary: method returns YES when both dictionaries contain the same key-value pairs.

# NSMutableDictionary

The NSMutableDictionary class lets you add new key-value pairs dynamically. Mutable dictionaries provide similar performance to mutable sets when it comes to inserting and deleting entries, and remember that both of these are a better choice than mutable arrays if you need to constantly alter the collection.

Mutable collections in general lend themselves to representing system states, and mutable dictionaries are no different. A common use case is to map one set of objects to another set of objects. 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. This arrangement allows a single mechanic to be responsible for multiple cars, but not vice versa.

## Creating Mutable Dictionaries

Mutable dictionaries can be created by calling any of the factory methods defined by NSDictionary on the NSMutableDictionary class. But, since many of these methods aren’t always the most intuitive to work with, you might find it useful to convert a literal dictionary to a mutable one using dictionaryWithDictionary.

NSMutableDictionary \*jobs = [NSMutableDictionary dictionaryWithDictionary:@{

@”scorpio”:@”harish”,

@”swift”,:@”sameer”,

@”xuv”,@”bhaskar}];

NSLog(@”%@”,jobs);

***NSString***

The Objective-C class for strings is NSString. Strings are typically created by direct assignment or by calling one of the NSString class methods. The NSString class provides an extensive set of APIs for working with strings, including methods for comparing, searching, and modifying strings. Just like other languages, strings are enclosed and defined by the use of “double quotes”, however in Objective-C an NSString also comes with the prefix of the @ sign.

Syntax:

NSString \*variable name = @”statement”;

E.g.:

NSString string1 \*stirng1 = @”This is NSString”;

Inherits from:

NSObject

NSString

**Comparing String:**

NSString uses the isEqualToString method to compare the two strings and also with this it uses hassuffix and hasprefix methods for partial comparison.

Eg:

NSString \*bus = @”KSRTC red bus”;

If ([bus isEqualToString:@”This is red bus”])

{

NSLog(@”The bus is red”);

}

If ([bus hasprefix:@”KSRTC”])

{

NSLog(@“it is the ksrtc bus of some color”);

}

If ([bus hassuffix:@”bus”])

{

NSLog(@“this is the bus”);

}

**Combining the String:**

NSString is an immutable type so whenever we concatenate the new strings will be created. It uses the two methods to combine the strings they are 1)stringByAppendingString and 2)stringByAppendingFormat.

Eg:

NSString \*fname = @”Harish”;

NSString \*lname = @”Bhardwaj”;

NSString \*name = [ fname stringByAppendingString lname ]

NSLog(@”%@”,name);//HarishBhardwaj

NSString \*name = [ fname stringByAppendingFormat lname ]

NSLog(@”%@”,name);//Harish Bharadwaj

**Searching String:**

The NSString search method returns the NSRange which defines a location and length field. The Location contains the beginning match of the string and length field has toatal number of characters present in the string. if no match was found, location will contain NSNotFound.

Changing case:

The NSString change case is used to convert the string from lower case to upper case and upper case to lower case.

Eg:

NSString \*name = @”NeoRays”;

NSLog(@”%@”, [name uppercaseStirng]);//NEORAYS

NSLog(@”%@”, [ name lowercaseString]);//neorays

**2>NSMutabaleString:**

NSObject

NSString

NSMutableString

The NSMutableString class is the mutable version of NSString. The NSMutableString will not create a new string for any changes made for the string. There are so many methods are there to support for the NSMutableString. The mutable string is created with StringewithString Class method. Which turns the NSString object into mutableString. The NSMutableString class declares the programmatic interface to an object that manages a mutable string—that is, a string whose contents can be edited—that conceptually represents an array of Unicode characters. To construct and manage an immutable string—or a string that cannot be changed after it has been created—use an object of the [NSString](https://developer.apple.com/library/mac/documentation/Cocoa/Reference/Foundation/Classes/NSString_Class/index.html#//apple_ref/occ/cl/NSString) class.

Eg:

NSMutableString \*name = [NSMutableString StringwithString :@”Harish”];

After created a mutable string we can assign the new value for the string using setString method.

The NSMutableString class adds one primitive method

[replaceCharactersInRange:withString:](https://developer.apple.com/library/mac/documentation/Cocoa/Reference/Foundation/Classes/NSMutableString_Class/index.html#//apple_ref/occ/instm/NSMutableString/replaceCharactersInRange:withString:)—to the basic string-handling behavior inherited from NSString. All other methods that modify a string work through this method. For example, [insertString:atIndex:](https://developer.apple.com/library/mac/documentation/Cocoa/Reference/Foundation/Classes/NSMutableString_Class/index.html#//apple_ref/occ/instm/NSMutableString/insertString:atIndex:) simply replaces the characters in a range of 0 length, while [deleteCharactersInRange:](https://developer.apple.com/library/mac/documentation/Cocoa/Reference/Foundation/Classes/NSMutableString_Class/index.html#//apple_ref/occ/instm/NSMutableString/deleteCharactersInRange:) replaces the characters in a given range with no characters.

**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.

Note that the subclass’s method must have the same name and parameter list as the superclass's overridden method.

Eg:

@interface SuperClass : NSObject

{

-(void)method1:(int n);

}

@end

@implements SuperClass : NSObject

{

Int age;

}

-(void)method1(int n)

{

Age = n;

NSLog(@”The age of Person is %d”,age);

}

@end

@interface SubClass : SuperClass //Extends super class

@end

@implements Subclass :SuperClass

{

int marks;

}

-(void)method1(int n)

{

Marks = n;

NSLog(@”The marks of student is %d”,marks);

}

@end

if we create an instance of SuperClass and send a number, it will print the age of the person. If we create an instance of Subclass and send a number it will print the marks of the student.

**Method OverRiding**

Method over riding is not possible in Objective c Programming. But we ll work with Method over riding by using different set of arguments and type. Using The same type of arguments is shows the error.

Eg:

@interface SuperClass

-(void)method1(int)a : (int)b ;

-(void)method2(int)a : (int)b; //error message

-(void)method2(int)a : (int)b ; (int)c;

@end

**Polymorphism**

The word polymorphism means having many forms. Typically, polymorphism occurs when there is a hierarchy of classes and they are related by inheritance.

Objective-C polymorphism means that a call to a member function will cause a different function to be executed depending on the type of object that invokes the function.

Consider the example, we have a class Aniaml that provides the basic interface for all the Animals. Dog,Lion and Cat are derived from the base class Aniaml.

We have the methods eat,sound and sleep that is going to show about the OOP feature polymorphism.

@interface Animal : NSObject

@Property NSString \*soundType;

@Property int sleepTime;

@Property NSString \*foodType;

-(void)eat;

-(void)sound;

-(void)sleep;

@end

@implements Animal

@synthesise soundType,sleepTime,foodType;

-(void)eat

{

NSLog(@”The Animal will eat both veg and non veg”);

}

-(void)sound

{

NSLog(@”The animals will make different sounds”);

}

-(void)sleep

{

NSLog(@”sleep time will differs depending on the animals”);

}

@interface Dog : Animal

@end

@implements Dog

@synthesise soundType,sleepTime,foodType;

-(void)sound

{

NSLog(@”The Dog sound like %@”,soundType);

}

-(void)eat

{

NSLog(@”The Dog Eats %@”,foodType);

}

-(void)sleep

{

NSLog(@”The Dog Sleeps for %d hours”,sleepTime);

}

@end

@interface Lion : Animal

@end

@implements Lion

@synthesise soundType,sleepTime,foodType;

-(void)sound

{

NSLog(@”The Lion sound like %@”,soundType);

}

-(void)eat

{

NSLog(@”The Lion Eats %@”,foodType);

}

-(void)sleep

{

NSLog(@”The Lion Sleeps for %d hours”,sleepTime);

}

@end

//creating the object in main and message passing using set method

Animal animalObject = [[Lion alloc]init]

//checking the object belongs to which class object using ismemeber method

If([animalobject isMemeberOfClass:[Dog class]])

{

[animalobject setsoundType:@”bowbow”];

[animalobject sleepTime:6];

[animalobject foodType:”vegandNonVeg”];

}

Else

{

[animalobject setsoundType:@”Roars”];

[animalobject sleepTime:10];

[animalobject foodType:”NonVeg”];

}

//method calling

[animalObject eat];

[animalObject sound];

[animalObject sleep];

**Classes and Objects**

In Objective-C classes provide the blueprint for creating objects. First, we have to define a reusable set of properties and behaviors inside of a class. Then, you instantiate objects from that class to interact with those properties and behaviors. A class is used to specify the form of an object and it combines data representation and methods for manipulating that data into one neat package. The data and methods within a class are called members of the class. A class definition starts with the keyword **@interface** followed by the interface(class) name; and the class body, enclosed by a pair of curly braces. In Objective-C, all classes are derived from the base class called **NSObject**. It is the superclass of all Objective-C classes. It provides basic methods like memory allocation and initialization. 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 the class:**

interface resides in a file named Car.h (also called a “header”) and whose implementation resides in Car.m. These are the standard file extensions for Objective-C classes.

The Objective-C syntax used to declare a class interface is:

@interface SimpleClass : NSObject

@end

This example declares a class named SimpleClass, which inherits from NSObject.

The public properties and behavior are defined inside the @interface declaration. In this example, nothing is specified beyond the superclass, so the only functionality expected to be available on instances of SimpleClass is the functionality inherited from NSObject.

It’s important to note that the name of each class must be unique within an app, even across included libraries or frameworks. If you attempt to create a new class with the same name as an existing class in a project, you’ll receive a compiler error.

The basic syntax to provide the implementation for a class is:

@import “xyzperson.h”

@implement XyzPerson

@end

If you declare any methods in the class interface, you’ll need to implement them inside this file

The minus sign (-) at the front of the method name indicates that it is an instance method, which can be called on any instance of the class. This differentiates it from class methods, which can be called on the class itself.

**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

}

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: 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.

**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.

Inheritance

One of the most important concepts in object-oriented programming is that of inheritance. Inheritance allows us to define a class in terms of another class which makes it easier to create and maintain an application. This also provides an opportunity to reuse the code functionality and fast implementation time.

When creating a class, instead of writing completely new data members and member functions, the programmer can designate that the new class should inherit the members of an existing class. This existing class is called the**base** class, and the new class is referred to as the **derived** class.

The idea of inheritance implements the **is a** relationship. For example, mammal IS-A animal, dog IS-A mammal, hence dog IS-A animal as well and so on.

Objective-C allows only Multiple inheritance, i.e., it can have only one base class but allows multilevel inheritance. All classes in Objective-C is derived from the superclass **NSObject**.

Eg:

@interface College : NSObject

NSString \*collegeName;

Int collegeCode;

-(void)print;

@end

@implements College

{

NSString \*name;

Int code;

}

-(void)print

{

name = collegeName;

code = collegeCode;

NSLog(@”The College Name is %@”,name);

NsLog(@”The college code is %d”,code);

}

@end

@interface Course : College

NSString \*courseName;

@end

@implements Course

NSString \*course;

-(void)print

{

course = courseName;

NSLog(@”The course name in college is %@”,course);

}

@end

Protocols

**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>)

**Selector**

A selector is a message that can be sent to a class of an object

It can be used to refer simply to the name of a method when its used in a source-code message to an object.

It also refers to unique identifier that replaces the name when source code is complied. Compiled selectors are of type SEL. All the methods with same name have the same selector.

You can use a selector to invoke a method of an object.

Methods and selectors

For efficiency full ascii names are not used as method selectors in compiled code.

Instead, the compiler writes each method name into table, then pairs the name with a unique identifier that represents the method at runtime.

The runtime system makes our each identifier is unique. No two selectors are same, and all methods with same name have the same selector.

**SEL and @selector**

Compiled selector are assigned to a special type, SEL to distinguish them from another data.

Valid selectors are never 0.

You must let the system assign SEL identifiers to methods; its pointless to assign them arbitrally

The @selector directive lets you refer to the compiled selector, rather than to the full method name. Here, the selector for setWidth:Height: is assigned to the setWidthHeight variable.

Example:

SEL setWidthHeight;

setWidthHeight=@selector(setWidth:height);

this is the most efficient way to assign values to the SEL variables at compile time with the @selector directive.

However in some cases, you may need to convert a character string to a selector at runtime. You can do this with the NSSelectorFromString function

setWidthHeight= NSSelectorFromString(aBuffer);

conversion in the opposite direction is also possible. The NSStringFromSelector function returns a method name for selector

NSString \*method;

Method=NSStringFromSelector(setWidthHeight);

Compiled selectors identify method names, not method implementations

The display method for one class, for example, has the same selector as display methods difined in other classes.

This is essential for polymorphism and dynamic binding; it lets you send the same message to receivers belonging to different class.

If there were one selector per method implementation, a message would be no different than a function call.

A class method and an instance method with same name are assigned with same selector. However because of their separate domains, there is no confusion between two. A class could define a display class method in addition to display instance method.

**return type and argument type of methods**

The messaging routine has access to method implementations only through selector, so it treats all methods with selectors alike. It discovers the return type of a method and data type of its arguments, from the selector.

Therefore, except for the message sent to statically typed receivers, dynamic binding requires all implementations of identically named methods to have the same return type and the same arguments type.

Statically typed receivers are an exception to this rule, since the compiler can learn about the method implementation from the class type.

Although identically named class methods and instance methods are represented by the name selector, they can have different argument and return type.

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'.
* crearing 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.