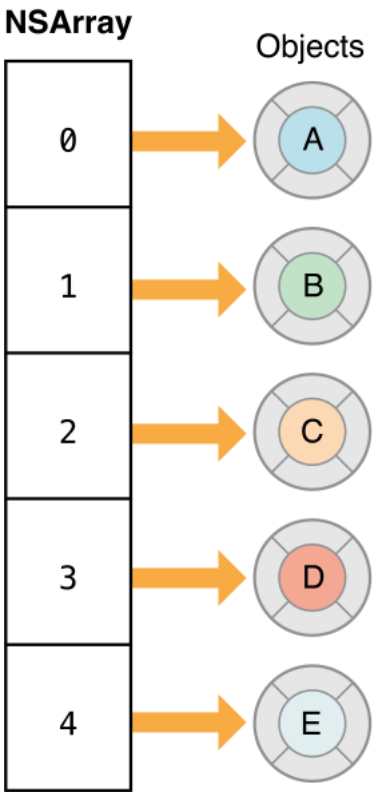
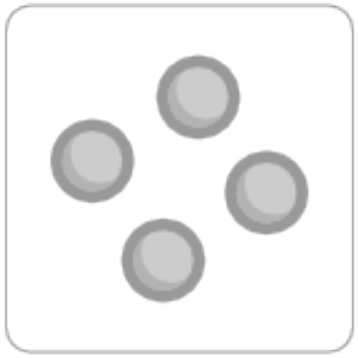
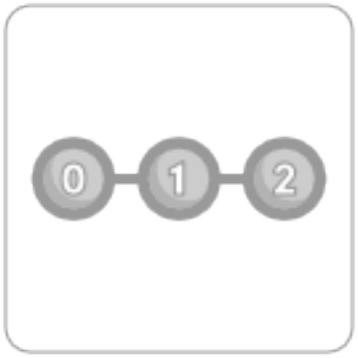
ITP 342
Mobile App Dev

Collections

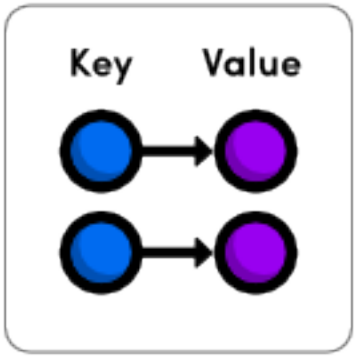




NSSet



NSArray



NSDictionary

Collections

- In Cocoa and Cocoa Touch, a **collection** is a Foundation framework class used for storing and managing groups of objects.
  - Its primary role is to store objects in the form of either an array, a dictionary, or a set.
- Collections share a number of characteristics.
- Most collections hold only objects and have both a mutable and an immutable variant.
- All collections share a number of common tasks, which include:
  - Enumerating the objects in a collection
  - Determining whether an object is in a collection
  - Accessing individual elements in a collection

Collections

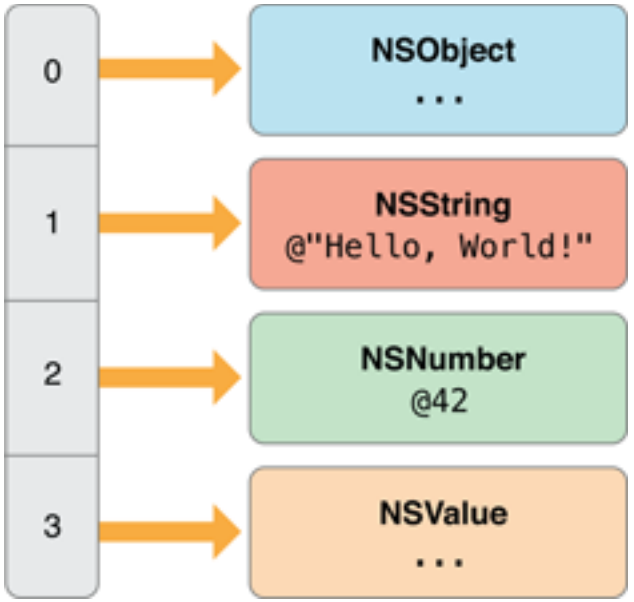
- **Mutable** collections also allow some additional tasks:
  - Adding objects to a collection
  - Removing objects from a collection
- Because how well a collection performs depends on how it is used, you should choose the collection best suited to a particular task.
- Resources
  - https://developer.apple.com/library/ios/documentation/cocoa/conceptual/Collections/Collections.html
  - https://developer.apple.com/library/ios/documentation/cocoa/conceptual/ProgrammingWithObjectiveC/FoundationTypesandCollections/FoundationTypesandCollections.html

Collections

- Array
  - Ordered collection of objects
- Dictionary
  - Manage pairs of keys and values
- Set
  - Unordered collection of objects

Array

- Classes
  - NSArray
  - NSMutableArray
  - NSPointerArray



Array Fundamentals

- Ordered collections that can contain any sort of object
  - Does not have to be homogeneous
    - Could contain any combo of Cat & Dog objects
  - Cannot contain an `int` or a `float`
    - Can use `NSNumber`
- We access the array elements by using an index
  - Index values start at 0
    - To access the first element, use index = 0
    - To access the second element, use index = 1
    - The last index = number of elements - 1
  - If an array has 10 elements, use indices 0 - 9

NSArray

- An **NSArray** object manages a static array
  - Once you have created the array, you cannot add objects to it or remove object from it
  - You can modify individual elements themselves
  - If you have an array of Person objects, can change the properties of the objects
- 2 primitive methods - `count` and `objectAtIndex` - provide the basis for all other methods in its interface
  - `count` returns the number of elements in the array
  - `objectAtIndex` give you access to the array elements by index
- https://developer.apple.com/library/mac/documentation/Cocoa/Reference/Foundation/Classes/NSArray_Class/NSArray.html

Create an Array

- You can create an array through allocation and initialization, class factory methods, or literal syntax.

```
+ (id)arrayWithObject:(id)anObject;
+ (id)arrayWithObjects:(id)firstObject, ...;
- (id)initWithObjects:(id)firstObject, ...;
```

- The `arrayWithObjects:` and `initWithObjects:` methods both take a nil-terminated, variable number of arguments, which means that you must include `nil` as the last value, like this:

```
NSArray *someArray =
  [NSArray arrayWithObjects:someObject, someString,
    someNumber, someValue, nil];
```

Create an Array

- Create an array with a list of objects as its elements
  - Objects are listed in order and are separated by commas
    - This is a special syntax used by methods that can take a variable number of arguments
    - To mark the end of the list, use `nil`

```
// Create an array to contain the month names
NSArray *monthNames = [[NSArray alloc] initWithObjects:
  @"January", @"February", @"March", @"April",
  @"May", @"June", @"July", @"August", @"September",
  @"October", @"November", @"December", nil ];
```

Create an Array

- Literal Syntax
  - Do not terminate the list of objects with `nil` when using this literal syntax

```
NSArray *someArray = @[firstObject, secondObject,
  thirdObject];
```

```
NSArray *monthNames = @[@"January", @"February",
  @"March", @"April", @"May", @"June", @"July",
  @"August", @"September", @"October", @"November",
  @"December"];
```

Create an Array

- Create an array from another array

```
NSArray *sDevices = @[@"iPhone", @"iPad", @"iPod"];

NSArray *aDevices = [NSArray arrayWithObjects:sDevices
                                        count:2];
// aDevices contains { @"iPhone", @"iPad" }
```

Querying Array Objects

- Once you've created an array, you can query it for information like the number of objects, or whether it contains a given item:

```
NSUInteger numberOfItems = [someArray count];
if ([someArray containsObject:someString]) {
  ...
}
```

- You can also query the array for an item at a given index.
  - You'll get an out-of-bounds exception at runtime if you attempt to request an invalid index, so you should always check the number of items first:

```
if ([someArray count] > 0) {
  NSLog(@"First item is: %@", [someArray objectAtIndex:0]);
}
```

Subscripting

- There's also a subscript syntax alternative to using `objectAtIndex:`, which is just like accessing a value in a standard C array. The previous example could be re-written like this:

```
if ([someArray count] > 0) {
  NSLog(@"First item is: %@", someArray[0]);
}
```

Sorting Array Objects

- The `NSArray` class also offers a variety of methods to sort its collected objects.
- Because `NSArray` is immutable, each of these methods returns a new array containing the items in the sorted order.

```
NSArray *unsortedStrings = @[@"gammaString",
                             @"alphaString",
                             @"betaString"];
NSArray *sortedStrings = [unsortedStrings
  sortedArrayUsingSelector:@selector(compare:)];
```

NSMutableArray

- A subclass of `NSArray`
- An **`NSMutableArray`** object manages a dynamic, or mutable, array
  - Allows the addition and deletion of entries at any time, automatically allocating memory as needed
- Primitive methods provide the basis for its ability to add, replace, and remove elements
  - `addObject:`
  - `insertObject:atIndex:`
  - `removeLastObject`
  - `removeObjectAtIndex:`
  - `replaceObjectAtIndex:withObject:`
- https://developer.apple.com/library/mac/documentation/Cocoa/Reference/Foundation/Classes/NSMutableArray_Class/Reference/Reference.html

Initializing a Mutable Array

- @[…] literals create immutable arrays
- Use the older API to create mutable ones
  - `initWithArray:`
  - `initWithArray:copyItems:`
  - `initWithContentsOfFile:`
  - `initWithContentsOfURL:`
  - `initWithObjects:`
  - `initWithObjects:count:`

initWithObjects:

initWithObjects:

Initializes a newly allocated array by placing in it the objects in the argument list.

```
- (id)initWithObjects:(id)firstObj, ...
```

Parameters

firstObj, ...

A comma-separated list of objects ending with `nil`.

Return Value

An array initialized to include the objects in the argument list. The returned object might be different than the original receiver.

Discussion

After an immutable array has been initialized in this way, it can't be modified.

Create NSMutableArray

```
// Create an array to contain names
NSMutableArray *names = [NSMutableArray arrayWithObjects:
  @"Mark Zuckerberg", @"Jimmy Wales", @"Steve Jobs",
  @"Chad Hurley", @"Steve Chen", @"Evan Williams",
  @"Larry Page", @"Sergey Brin", @"Craig Newmark",
  @"Bill Gates", nil ];
```

- You can easily create an instance of one type of array from the other using the initializer `initWithArray:` or `arrayWithArray:`

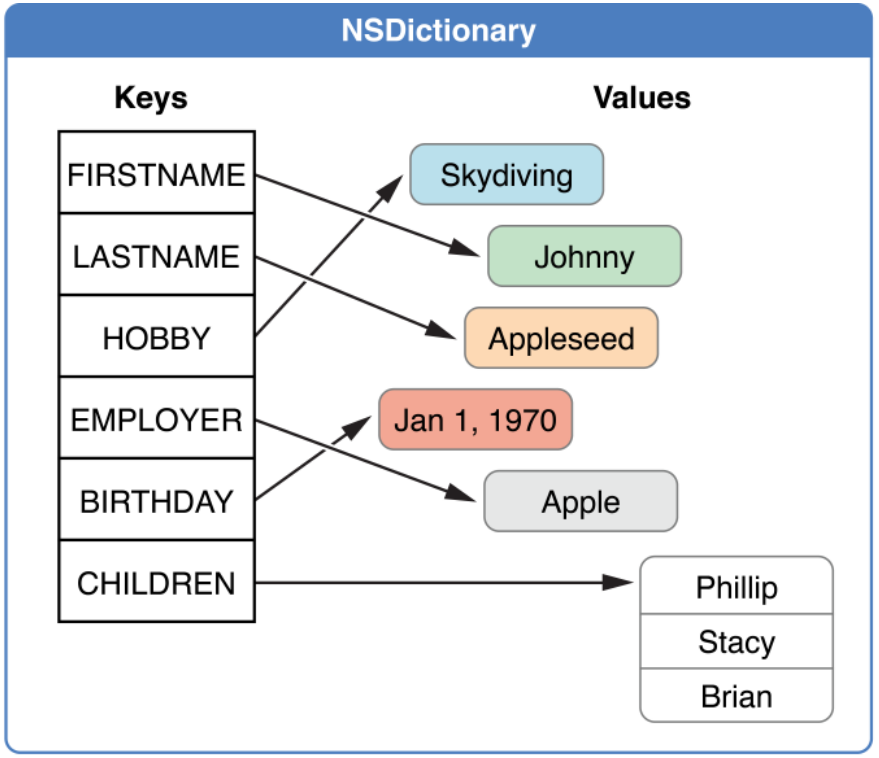
```
NSArray *pets = [NSArray arrayWithObjects:
  @"bird", @"cat", @"dog", @"fish", nil ];
NSMutableArray *morePets = [NSMutableArray arrayWithArray:pets];
```

NSPointerArray

- `NSPointerArray` is a mutable collection modeled after `NSArray` but it can also hold NULL values, which can be inserted or extracted (and which contribute to the object's count).
- Moreover, unlike traditional arrays, you can set the count of the array directly.
- https://developer.apple.com/library/mac/documentation/Cocoa/Reference/Foundation/Classes/NSPointerArray_Class/Introduction/Introduction.html

Dictionary

- Classes
  - NSDictionary
  - NSMutableDictionary



NSDictionary

- An NSDictionary object manages an immutable dictionary – that is, after you create the dictionary, you cannot add, remove or replace keys and values.
- You can, however, modify individual values themselves (if they support modification), but the keys must not be modified.
- The mutability of the collection does not affect the mutability of the objects inside the collection.
- You should use an immutable dictionary if the dictionary rarely changes, or changes wholesale.

NSDictionary

- You can easily create an instance of one type of dictionary from the other using the initializer `initWithDictionary:` or the convenience constructor `dictionaryWithDictionary:`.
- Internally, a dictionary uses a hash table to organize its storage and to provide rapid access to a value given the corresponding key.
  - However, the methods defined for dictionaries insulate you from the complexities of working with hash tables, hashing functions, or the hashed value of keys.
  - The methods take keys directly, not in their hashed form.

Keys

- In most cases, Cocoa-provided objects such as `NSString` objects should be sufficient for use as keys.
- In some cases, however, it may be necessary to use custom objects as keys in a dictionary.
  - Keys must conform to the NSCopying protocol.
  - Keys must implement the `hash` and `isEqual:` methods because a dictionary uses a hash table to organize its storage and to quickly access contained objects.

NSDictionary

- Create a dictionary using literal syntax

```
NSDictionary *inventory = @{
  @"iPodTouch" : [NSNumber numberWithInt:15],
  @"iPhone" : [NSNumber numberWithInt:5],
  @"iPad" : [NSNumber numberWithInt:20]
};
```

```
NSDictionary *info = @{
  @"name" : @"Tommy Trojan",
  @"id" : @"1234567890",
  @"username" : @"ttrojan"
};
```

NSDictionary

- Values and keys as arguments

```
NSDictionary *devices;
devices = [NSDictionary dictionaryWithObjectsAndKeys:
   [NSNumber numberWithInt:15], @"iPodTouch",
   [NSNumber numberWithInt:12], @"iPad2",
   [NSNumber numberWithInt:8], @"iPad3",
   nil];
```

```
NSDictionary *classes;
devices = [NSDictionary dictionaryWithObjectsAndKeys:
   @"Mobile App Development", @"ITP342",
   @"Mobile App Technologies", @"ITP140",
   @"Mobile Game Programming", @"ITP382",
   nil];
```

NSDictionary

- Values and key as arrays

```
NSArray *models = @[@"iPodTouch", @"iPad2", @"iPad3"];
NSArray *counts = @[[NSNumber numberWithInt:15],
   [NSNumber numberWithInt:12],
   [NSNumber numberWithInt:8]];

NSDictionary *devices = [NSDictionary
   dictionaryWithObjects: counts
   forKeys: models];
```

```
NSArray *courses = @[@"ITP140", @"ITP342", @"ITP382"];
NSArray *names = @[@"Mobile App Technologies",
   @"Mobile App Development",
   @"Mobile Game Programming"];

NSDictionary *classes = [NSDictionary
   dictionaryWithObjects: names
   forKeys: courses];
```

NSDictionary

- Accessing values and keys
  - You can use the same subscripting syntax as arrays (`someDict[key]`) to access the value for a particular key.
  - The `objectForKey:` method is the other common way to access values.

```
NSLog(@"ITP has %@ iPods", devices[@"iPodTouch"]);
```

```
NSLog(@"ITP has %@ iPods",
  [devices objectForKey:@"iPodTouch"]);
```

NSDictionary

- Enumerating

```
NSLog(@"We currently have %ld models available",
  [devices count]);

for (id key in devices) {
  NSLog(@"There are %@ %@ devices", devices[key], key);
}
```

- Get keys and values

```
NSLog(@"Models %@", [devices allKeys]);

NSLog(@"Stock %@", [devices allValues]);
```

NSMutableDictionary

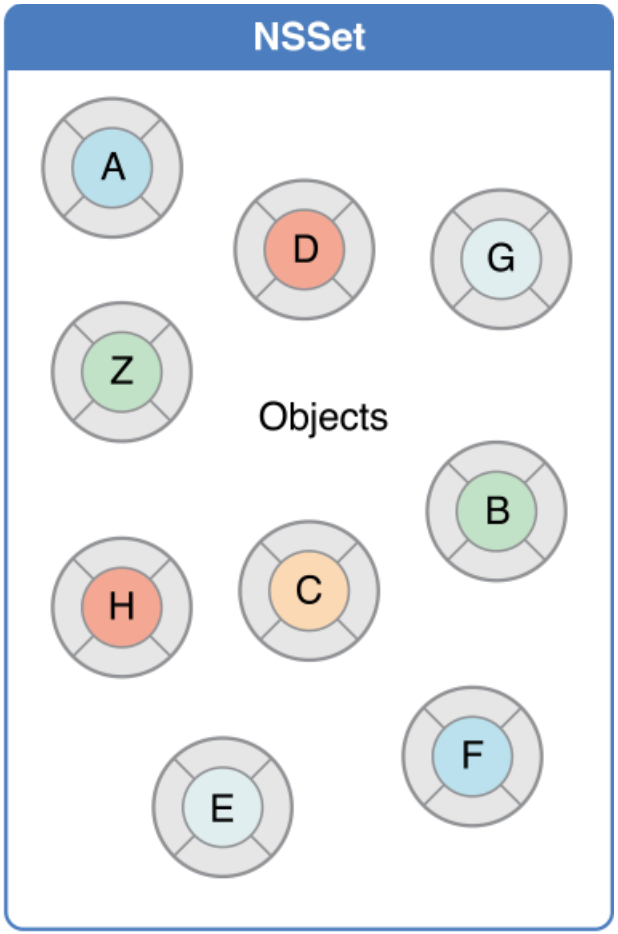
- An NSMutableDictionary object manages a mutable dictionary, which allows the addition and deletion of entries at any time, automatically allocating memory as needed.
- You should use a mutable dictionary if the dictionary changes incrementally, or is very large – as large collections take more time to initialize.

Dictionary

- NSDictionary
  - https://developer.apple.com/library/mac/documentation/Cocoa/Reference/Foundation/Classes/NSDictionary_Class/Reference/Reference.html
- NSMutableDictionary
  - https://developer.apple.com/library/mac/documentation/Cocoa/Reference/Foundation/Classes/NSMutableDictionary_Class/Reference/Reference.html

Set

- Classes
  - NSSet
  - NSMutableSet
  - NSMapTable



Set

- NSSet
  - https://developer.apple.com/library/mac/documentation/Cocoa/Reference/Foundation/Classes/NSSet_Class/Reference/Reference.html
- NSMutableSet
  - https://developer.apple.com/library/mac/documentation/Cocoa/Reference/Foundation/Classes/NSMutableSet_Class/Reference/NSMutableSet.html
- NSCountedSet
  - https://developer.apple.com/library/mac/documentation/Cocoa/Reference/Foundation/Classes/NSCountedSet_Class/Reference/Reference.html
- NSMapTable
  - https://developer.apple.com/library/mac/documentation/Cocoa/Reference/NSMapTable_class/Reference/NSMapTable.html