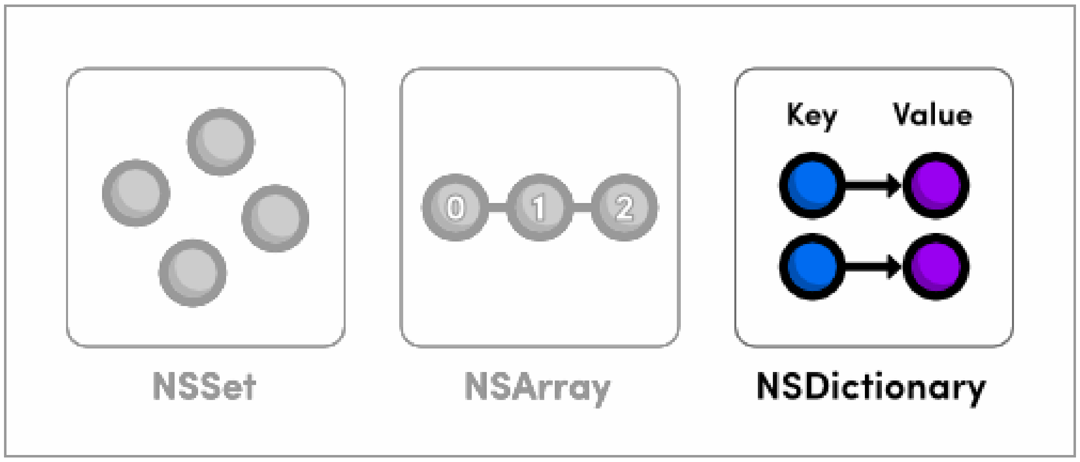
NSDictionary

Like an `NSSet`, the `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.



The basic collection classes of the Foundation Framework

`NSDictionary` is immutable, but the `NSMutableDictionary` data structure lets you dynamically add and remove entries as necessary.

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. All of these are presented below.

```
// Literal syntax
NSDictionary *inventory = @{
    @"Mercedes-Benz SLK250" : [NSNumber numberWithInt:13],
    @"Mercedes-Benz E350" : [NSNumber numberWithInt:22],
    @"BMW M3 Coupe" : [NSNumber numberWithInt:19],
    @"BMW X6" : [NSNumber numberWithInt:16],
};
// Values and keys as arguments
inventory = [NSDictionary dictionaryWithObjectsAndKeys:
             [NSNumber numberWithInt:13], @"Mercedes-Benz SLK250",
             [NSNumber numberWithInt:22], @"Mercedes-Benz E350",
             [NSNumber numberWithInt:19], @"BMW M3 Coupe",
             [NSNumber numberWithInt:16], @"BMW X6", nil];
// Values and keys as arrays
NSArray *models = @[@"Mercedes-Benz SLK250", @"Mercedes-Benz E350",
                    @"BMW M3 Coupe", @"BMW X6"];
NSArray *stock = @[[NSNumber numberWithInt:13],
                   [NSNumber numberWithInt:22],
                   [NSNumber numberWithInt:19],
                   [NSNumber numberWithInt:16]];
inventory = [NSDictionary dictionaryWithObjects:stock forKeys:models];
NSLog(@"%@", inventory);
```

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 you should be careful to ensure that the key array is the same length as the value array.

For common programming tasks, you'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 the `isEqual:` 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).

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.

```
NSDictionary *inventory = @{
    @"Mercedes-Benz SLK250" : [NSNumber numberWithInt:13],
    @"Mercedes-Benz E350" : [NSNumber numberWithInt:22],
    @"BMW M3 Coupe" : [NSNumber numberWithInt:19],
    @"BMW X6" : [NSNumber numberWithInt:16],
};
NSLog(@"There are %@ X6's in stock", inventory[@"BMW X6"]);
NSLog(@"There are %@ E350's in stock",
      [inventory objectForKey:@"Mercedes-Benz E350"]);
```

Typically, you'll want to access values from keys, as shown above; however, 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 (but not vice versa). For example, the following extracts all of the keys with a value of 0.

```
NSDictionary *inventory = @{
    @"Mercedes-Benz SLK250" : [NSNumber numberWithInt:0],
    @"Mercedes-Benz E350" : [NSNumber numberWithInt:0],
    @"BMW M3 Coupe" : [NSNumber numberWithInt:19],
    @"BMW X6" : [NSNumber numberWithInt:16],
};
NSArray *outOfStock = [inventory allKeysForObject:
                       [NSNumber numberWithInt:0]];
NSLog(@"The following cars are currently out of stock: %@",
      [outOfStock componentsJoinedByString:@", "]);
```

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.

```
NSDictionary *inventory = @{
    @"Mercedes-Benz SLK250" : [NSNumber numberWithInt:13],
    @"Mercedes-Benz E350" : [NSNumber numberWithInt:22],
    @"BMW M3 Coupe" : [NSNumber numberWithInt:19],
    @"BMW X6" : [NSNumber numberWithInt:16],
};
NSLog(@"We currently have %ld models available", [inventory count]);
for (id key in inventory) {
    NSLog(@"There are %@ %@'s in stock", inventory[key], key);
}
```

You can isolate a dictionary's keys/values with the `allKeys`/`allValues` methods, 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.

```
NSLog(@"Models: %@", [inventory allKeys]);
NSLog(@"Stock: %@", [inventory allValues]);
```

For block-oriented developers, `enumerateKeysAndObjectsUsingBlock:` offers yet another way to enumerate entries. The block is called for each entry, and both the key and the value are passed as arguments:

```
[inventory enumerateKeysAndObjectsUsingBlock:^(id key, id obj, BOOL *stop) {
    NSLog(@"There are %@ %@'s in stock", obj, key);
}];
```

Comparing Dictionaries

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

```
NSDictionary *warehouseInventory = @{
    @"Mercedes-Benz SLK250" : [NSNumber numberWithInt:13],
    @"Mercedes-Benz E350" : [NSNumber numberWithInt:22],
    @"BMW M3 Coupe" : [NSNumber numberWithInt:19],
    @"BMW X6" : [NSNumber numberWithInt:16],
};
NSDictionary *showroomInventory = @{
    @"Mercedes-Benz SLK250" : [NSNumber numberWithInt:13],
    @"Mercedes-Benz E350" : [NSNumber numberWithInt:22],
    @"BMW M3 Coupe" : [NSNumber numberWithInt:19],
    @"BMW X6" : [NSNumber numberWithInt:16],
};
if ([warehouseInventory isEqualToDictionary:showroomInventory]) {
    NSLog(@"Why are we storing so many cars in our showroom?");
}
```

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 with `keysSortedByValueUsingComparator:`, which accepts a block that should return one of the `NSComparisonResult` enumerators described in the NSArray module. The following example sorts the models from most expensive to most affordable.

```
NSDictionary *prices = @{
    @"Mercedes-Benz SLK250" : [NSDecimalNumber decimalNumberWithString:@"42900.00"],
    @"Mercedes-Benz E350" : [NSDecimalNumber decimalNumberWithString:@"51000.00"],
    @"BMW M3 Coupe" : [NSDecimalNumber decimalNumberWithString:@"62000.00"],
    @"BMW X6" : [NSDecimalNumber decimalNumberWithString:@"55015.00"]
};
NSArray *sortedKeys = [prices keysSortedByValueUsingComparator:
                       ^NSComparisonResult(id obj1, id obj2) {
                           return [obj2 compare:obj1];
                       }];
NSLog(@"%@", sortedKeys);
```

It's important to note that the dictionary's *values* are passed to the block—not the keys themselves. Alternatively, you can always manually sort the results of `allKeys` and `allValues`.

Filtering Dictionary Keys

Similarly, you can't directly filter a dictionary, but the aptly named `keysOfEntriesPassingTest:` method lets you extract the keys passing a particular test. The test is defined as a block that returns `YES` if the current entry's key should be included and `NO` if it shouldn't. The resulting keys are returned as an `NSSet`. For example, the following snippet finds all of the cars that are under $50,000.

```
NSDictionary *prices = @{
    @"Mercedes-Benz SLK250" : [NSDecimalNumber decimalNumberWithString:@"42900.00"],
    @"Mercedes-Benz E350" : [NSDecimalNumber decimalNumberWithString:@"51000.00"],
    @"BMW M3 Coupe" : [NSDecimalNumber decimalNumberWithString:@"62000.00"],
    @"BMW X6" : [NSDecimalNumber decimalNumberWithString:@"55015.00"]
};
NSDecimalNumber *maximumPrice = [NSDecimalNumber decimalNumberWithString:@"50000.00"]
NSSet *under50k = [prices keysOfEntriesPassingTest:
                   ^BOOL(id key, id obj, BOOL *stop) {
                       if ([obj compare:maximumPrice] == NSOrderedAscending) {
                           return YES;
                       } else {
                           return NO;
                       }
                   }];
NSLog(@"%@", under50k);
```

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:@{
                                 @"Audi TT" : @"John",
                                 @"Audi Quattro (Black)" : @"Mary",
                                 @"Audi Quattro (Silver)" : @"Bill",
                                 @"Audi A7" : @"Bill"
                             }];
NSLog(@"%@", jobs);
```

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. As an alternative, you can assign values to keys using the dictionary subscripting syntax, also shown below.

```
NSMutableDictionary *jobs = [NSMutableDictionary
                             dictionaryWithDictionary:@{
                                 @"Audi TT" : @"John",
                                 @"Audi Quattro (Black)" : @"Mary",
                                 @"Audi Quattro (Silver)" : @"Bill",
                                 @"Audi A7" : @"Bill"
                             }];
// Transfer an existing job to Mary
[jobs setObject:@"Mary" forKey:@"Audi TT"];
// Finish a job
```

```
[jobs removeObjectForKey:@"Audi A7"];
// Add a new job
jobs[@"Audi R8 GT"] = @"Jack";
```

Combining Dictionaries

Mutable dictionaries can be expanded by adding the contents of another dictionary to its collection via the `addEntriesFromDictionary:` method. This can be used, for example, to combine jobs from two auto shop locations:

```
NSMutableDictionary *jobs = [NSMutableDictionary
                             dictionaryWithDictionary:@{
                                 @"Audi TT" : @"John",
                                 @"Audi Quattro (Black)" : @"Mary",
                                 @"Audi Quattro (Silver)" : @"Bill",
                                 @"Audi A7" : @"Bill"
                             }];
NSDictionary *bakerStreetJobs = @{
    @"BMW 640i" : @"Dick",
    @"BMW X5" : @"Brad"
};
[jobs addEntriesFromDictionary:bakerStreetJobs];
```

This method also presents another option for creating mutable dictionaries:

```
// Create an empty mutable dictionary
NSMutableDictionary *jobs = [NSMutableDictionary dictionary];
// Populate it with initial key-value pairs
[jobs addEntriesFromDictionary:@{
    @"Audi TT" : @"John",
    @"Audi Quattro (Black)" : @"Mary",
    @"Audi Quattro (Silver)" : @"Bill",
    @"Audi A7" : @"Bill"
}];
```

Enumeration Considerations

Dictionaries should not be mutated while you're iterating over them. Please see the Enumeration Considerations section of the *NSSet* module for details.

Depending on how you need to alter the collection, you might be able to use the `allKeys` or `allValues` methods to create snapshots of just the keys or values. If those aren't sufficient you should use the `dictionaryWithDictionary:` class method to create a shallow copy of the entire dictionary.

Continue to *Date Programming* ›