iPhone Programming

CMSC 498i – Spring 2010

Week 2

Chuck Pisula

Note about methods

- In slides and handouts, I'll sometimes use shorthand
- Example shorthand

```
-isEqual:
```

- Instead of

```
-(BOOL)isEqual:(id)other;
```

- I'm expecting you to
  - recognize "-" as starting a method
  - recognize " : " as place a parameter starts
  - be able to look up method specifics online for yourself

Today's Topics

- Foundation Collection Classes
  - Array
  - Dictionary
  - Etc…
- Property Lists

Collection Classes

Collection Classes

- Data structures that hold and organize other objects

Collection Classes

- Data structures that hold and organize other objects
- Generally expected functionality
  - Creation
  - Count objects
  - Object addition / removal
  - Access to stored objects

Collection Classes

- Data structures that hold and organize other objects
- Generally expected functionality
  - Creation
  - Count objects
  - Object addition / removal
  - Access to stored objects
- Collections claim an ownership of their contents
  - e.g. each contained object retained by the collection

Foundation

Foundation

- Collections in Objective-C
  - Hold references to `NSObjects` (e.g. `NSObject *`)
  - Mutability pattern

Foundation

- Collections in Objective-C
  - Hold references to `NSObjects` (e.g. `NSObject *`)
  - Mutability pattern
- We will discuss
  - `NSArray`
  - `NSDictionary`
  - `NSSet`
  - `Enumeration`

Array

NSArray

- Ordered collection of objects
- Retains its contents
- Accessible by index
- Patterns in use
  - Class Clusters
  - Enumeration

NSArray

- `NSNotFound`
  - Value that indicates an item did not exist
  - Used by search methods in array and other collections

NSArray

- `NSNotFound`
  - Value that indicates an item did not exist
  - Used by search methods in array and other collections

```
NSArray *studentsArray = ...;

NSUInteger idx = [studentsArray indexOfObject: person];
if (idx == NSNotFound) {
    NSLog(@"%@ is not enrolled!", person.name);
}
```

NSArray

- Working with `NSArray`

NSArray

- Working with `NSArray`

```
// creation
- (id)arrayWithObjects:(id)firstObj, ...;
- (id)initWithObjects:(id)firstObj, ...;
- (id)initWithObject:(id)anObject;
```

NSArray

- Working with `NSArray`

```
// creation
- (id)arrayWithObjects:(id)firstObj, ...;
- (id)initWithObjects:(id)firstObj, ...;
- (id)initWithObject:(id)anObject;
```

```
// primitives
- (int)count;
- (id)objectAtIndex:(NSUInteger)index;
```

NSArray

- Working with `NSArray`

```
// creation
- (id)arrayWithObjects:(id)firstObj, ...;
- (id)initWithObjects:(id)firstObj, ...;
- (id)initWithObject:(id)anObject;
```

```
// primitives
- (int)count;
- (id)objectAtIndex:(NSUInteger)index;
```

```
// searching
- (NSUInteger)indexOfObject:(id)anObject;
```

NSArray

- Working with `NSArray`

```
// creation
- (id)arrayWithObjects:(id)firstObj, ...;
- (id)initWithObjects:(id)firstObj, ...;
- (id)initWithObject:(id)anObject;
```

```
// primitives
- (int)count;
- (id)objectAtIndex:(NSUInteger)index;
```

```
// searching
- (NSUInteger)indexOfObject:(id)anObject;
```

```
// sorting and filtering
- (NSArray *)sortedArrayUsingSelector:(SEL)comparator;
- (NSArray *)filteredArrayUsingPredicate:(NSPredicate *)...
```

NSArray

- Example Creation

```
Person *p1 = [[Person alloc] initWithName:@”Sally”];
Person *p2 = [[Person alloc] initWithName:@”Fred”];

NSArray *students = [NSArray arrayWithObjects:p1, p2, nil];

// ‘students’ array keeps a retain on p1,p2, so we can release...
[p1 release];
[p2 release];
```

Variable Argument Lists

- Whenever you see "**...**" in a method / function declaration

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

Variable Argument Lists

- Whenever you see "**...**" in a method / function declaration

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

- Works the same as it would in any C, C++ code.
  - Variable number of arguments can be provided
  - End of arguments signified by nil, NULL, or 0
  - Terminate "vararg" properly, or be doomed to crash!

Variable Argument Lists

- Whenever you see "**...**" in a method / function declaration

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

- Works the same as it would in any C, C++ code.
  - Variable number of arguments can be provided
  - End of arguments signified by nil, NULL, or 0
  - Terminate "vararg" properly, or be doomed to crash!

```
NSArray *students = [NSArray arrayWithObjects:p1, p2, nil];
```

NSArray Sorting

- Example

```
NSArray *students = [NSArray arrayWithObjects:@”Sally”, @”Fred”, nil];

students = [students sortedArrayUsingSelector:@selector(compare:)];
```

NSArray Sorting

- Example

```
NSArray *students = [NSArray arrayWithObjects:@”Sally”, @”Fred”, nil];

students = [students sortedArrayUsingSelector:@selector(compare:)];
```

- Comparison method signature

```
- (NSComparisonResult)compare:(id)otherObject {

}
```

NSArray Sorting

- Example

```
NSArray *students = [NSArray arrayWithObjects:@”Sally”, @”Fred”, nil];

students = [students sortedArrayUsingSelector:@selector(compare:)];
```

- Comparison method signature

```
- (NSComparisonResult)compare:(id)otherObject {

}
```

- Valid return values

```
NSOrderedAscending
NSOrderedSame
NSOrderedDescending
```

NSArray Sorting

- `NSString` implements `-compare:`, you can too...

```
Person *p1 = [[Person alloc] initWithFirstName:@”Sally” lastName:@”..”];
Person *p2 = [[Person alloc] initWithFirstName:@”Fred” lastName:@”..”];

NSArray *students = [NSArray arrayWithObjects:p1, p2, nil];
students = [students sortedArrayUsingSelector:@selector(compareFirstName:)];
```

NSArray Sorting

- `NSString` implements `-compare:`, you can too...

```
Person *p1 = [[Person alloc] initWithFirstName:@”Sally” lastName:@”..”];
Person *p2 = [[Person alloc] initWithFirstName:@”Fred” lastName:@”..”];

NSArray *students = [NSArray arrayWithObjects:p1, p2, nil];
students = [students sortedArrayUsingSelector:@selector(compareFirstName:)];
```

- Then implement

```
@implementation Person

- (NSComparisonResult)compareFirstName:(id)other {
   return [[self firstName] compare:[other firstName]];
}
```

NSArray

- `indexOfObject:` details
  - Iterates until a match is found
  - match if `-isEqual:` returns `YES`
  - O(n) search time

NSArray - Sorting

- Sort time is O(n log(n))
- `SEL` comparator is of the form

```
- (NSComparisonResult)compare:(id)otherObject;
- (NSComparisonResult)localizedCompare:(id)otherObject;
```

NSArray - Sorting

- Sort time is O(n log(n))
- `SEL` comparator is of the form

```
- (NSComparisonResult)compare:(id)otherObject;
- (NSComparisonResult)localizedCompare:(id)otherObject;
```

- `-sortedArrayUsing`... variations
  - `UsingFunction: /* function pointer */fx`
  - `UsingDescriptor: (NSSortDescriptor *)descriptor`

NSArray

- Quickly message each object

```
- (void)makeObjectsPerformSelector:(SEL)selector;
```

- Example

```
// foreach object, invoke [student saveToDatabase]
[students_ makeObjectsPerformSelector:
                @selector(saveToDatabase)];
```

NSMutableArray

- `NSMutableArray` is a subclass of `NSArray`
- Allows mutation of contents
  - Add
  - Remove
- Create using any inherited `NSArray` init method

NSMutableArray

- `NSMutableArray` is a subclass of `NSArray`
- Allows mutation of contents
  - Add
  - Remove
- Create using any inherited `NSArray` init method

```
NSArray *nums = [NSArray arrayWithObjects:@”one”, @”two”, nil];
NSArray *nums = [[NSArray alloc] initWithObject:@”one”];
```

NSMutableArray

- `NSMutableArray` is a subclass of `NSArray`
- Allows mutation of contents
  - Add
  - Remove
- Create using any inherited `NSArray` init method

```
NSArray *nums = [NSArray arrayWithObjects:@”one”, @”two”, nil];
NSArray *nums = [[NSArray alloc] initWithObject:@”one”];
```

```
NSMutableArray *nums = [NSMutableArray arrayWithObjects:@”one”, @”two”, nil];
NSMutableArray *nums = [[NSMutableArray alloc] initWithObject:@”one”]
```

NSMutableArray

- Need a mutable array?
  - Use any inherited `NSArray` init method
  - Or ask for a “mutable” copy of an existing array!

```
NSMutableArray *mutableArray = [array mutableCopy];
```

- Common methods

```
- (void)insertObject:(id)object atIndex:(NSUInteger)idx;

- (void)removeObject:(id)object; // find using isEqual:
```

NSMutableArray

- Example Usage

```
@interface iPhoneCourse : NSObject {
    NSMutableArray *students_;
}
@end
```

```
@implementation iPhoneCourse

- (void)insertStudent:(Person *)person {
    // insert into students_ ivar
    // students_ mutable array will -retain the person
    [students_ addObject:person];
}

@end
```

Patterns In Use

Enumeration

Enumeration

- Mechanism for traversing the items in a collection

Enumeration

- Mechanism for traversing the items in a collection
- Options for enumerating collection classes
  - Iterate using an index
  - `NSEnumerator`
  - `NSFastEnumeration`

Enumeration

- Mechanism for traversing the items in a collection
- Options for enumerating collection classes
  - Iterate using an index
  - `NSEnumerator`
  - `NSFastEnumeration`
- One rule to follow – Don’t modify an array while iterating
  - Bad idea to modify state of something being iterated
    - Also, the docs say not to do this…
  - If you must, make a copy!

NSArray iteration

- for loop construct should look similar to that of other languages (except the brackets of course)

```
NSUInteger idx = 0;
NSUInteger count = [students count];

for (idx = 0; idx < count; idx++) {
    Person *person = [students objectAtIndex: idx];

    NSLog(@"student %d: %@", idx, person.name);
}
```

NSEnumerator

- Object encapsulation of state needed to visit all items
- `NSArray` and other objects can return an iterator

```
- (NSEnumerator *)objectEnumerator;
- (NSEnumerator *)reverseObjectEnumerator;
```

NSEnumerator

- Object encapsulation of state needed to visit all items
- `NSArray` and other objects can return an iterator

```
- (NSEnumerator *)objectEnumerator;
- (NSEnumerator *)reverseObjectEnumerator;
```

- To enumerate

```
NSEnumerator *enumerator = [students objectEnumerator];
Person *person = nil;

while ((person = [enumerator nextObject])) {
    NSLog(@"student %d: %@", person.name);
}
```

Fast Enumeration

- New ObjC Language Feature
- Safe Iteration of a Collection
  - Mutation Guard
    - Exception raised if mutated during iteration - no such exception with other iteration approaches
- Concise Syntax

Fast Enumeration

- New ObjC Language Feature
- Safe Iteration of a Collection
  - Mutation Guard
    - Exception raised if mutated during iteration - no such exception with other iteration approaches
- Concise Syntax

```
for (Person *person in students) {
    NSLog(@”student: %@”, person);
}
```

Fast Enumeration

Fast Enumeration

- Any class conforming to NSFastEnumeration protocol can be used with "fast enumeration" syntax
    - Formalizes how to support the feature in any class

Fast Enumeration

- Any class conforming to NSFastEnumeration protocol can be used with "fast enumeration" syntax
  - Formalizes how to support the feature in any class
- Benefits
  - Considerably more efficient than NSEnumerators
  - Mutation guard implies safe concurrent iteration

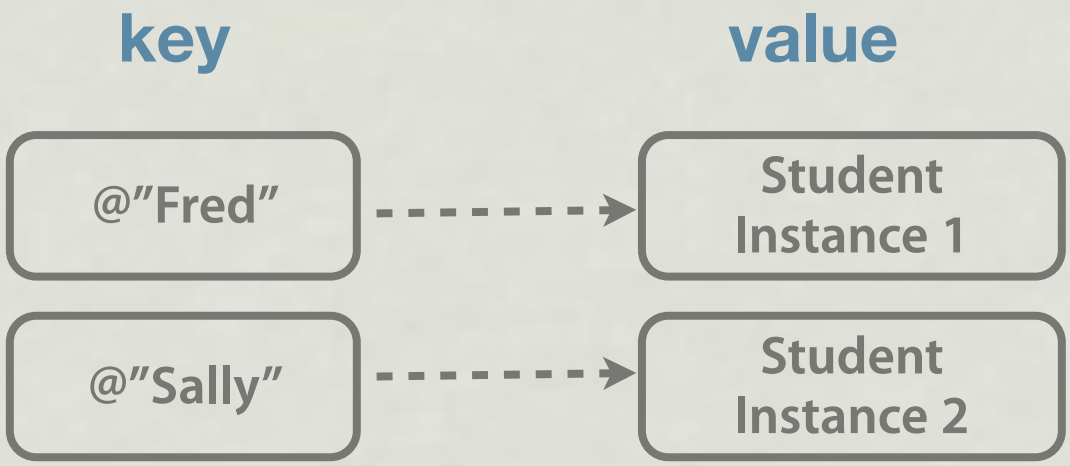
Fast Enumeration

- Any class conforming to NSFastEnumeration protocol can be used with "fast enumeration" syntax
  - Formalizes how to support the feature in any class
- Benefits
  - Considerably more efficient than NSEnumerators
  - Mutation guard implies safe concurrent iteration
- Negative
  - If you need the “index” must track yourself

Dictionary

NSDictionary

- Collection of objects organized by “key”
- Each entry is a key-value pair mapping



NSDictionary

NSDictionary

- A Hash Table
  - All entries have a unique key
  - Hash function is `-hash`
  - Bucket searched using `isEqual:`

NSDictionary

- A Hash Table
  - All entries have a unique key
  - Hash function is `-hash`
  - Bucket searched using `isEqual:`
- Keys are usually `NSStrings`, but can be anything

NSDictionary

- A Hash Table
  - All entries have a unique key
  - Hash function is `-hash`
  - Bucket searched using `isEqual:`
- Keys are usually `NSStrings`, but can be anything
- Keys are copied, and must implement:
  - `-hash`, `-isEqual:`, `NSCopying` `protocol` (can you think why?)

NSDictionary

- A Hash Table
  - All entries have a unique key
  - Hash function is `-hash`
  - Bucket searched using `isEqual:`
- Keys are usually `NSStrings`, but can be anything
- Keys are copied, and must implement:
  - `-hash`, `-isEqual:`, `NSCopying` `protocol` (can you think why?)
- Values are retained

NSDictionary

- Enumerating Keys
  - `NSDictionary` Supports fast enumeration of keys
  - Can access array of all keys
  - Order of keys in enumeration is unspecified
    - I.e. not necessarily sorted, or the same order each time

NSDictionary

- Common Methods

NSDictionary

- Common Methods

```
// Creation
+ (id)dictionaryWithObjectsAndKeys:(id)obj1, ...;
```

NSDictionary

- Common Methods

```
// Creation
+ (id)dictionaryWithObjectsAndKeys:(id)obj1, ...;
```

```
// Primitives
- (NSUInteger)count;
- (id)objectForKey:(id)key;
```

NSDictionary

- Common Methods

```
// Creation
+ (id)dictionaryWithObjectsAndKeys:(id)obj1, ...;
```

```
// Primitives
- (NSUInteger)count;
- (id)objectForKey:(id)key;
```

```
// Enumerating
- (NSArray *)allObjects;
- (NSArray *)allKeys;
- (NSEnumerator *)keyEnumerator;
```

NSDictionary

- Example Use

```
- (id)init {
    if ((self = [super init])) {
        students_ = [NSDictionary dictionaryWithObjectsAndKeys:
                                studentInfo1, @”fred”,
                                studentInfo2, @”sally”,
                                nil];
        [students_ retain];
    }
}
```

NSDictionary

- Example Use

```
- (id)init {
    if ((self = [super init])) {
        students_ = [NSDictionary dictionaryWithObjectsAndKeys:
                                studentInfo1, @"fred",
                                studentInfo2, @"sally",
                                nil];
        [students_ retain];
    }
}
```

```
- (StudentInfo *)studentInfoForName:(NSString *)name
{
    StudentInfo *info = nil;
    info = [students_ objectForKey: name];
    if (!info) {
        NSLog(@"Could not find info for %@", name);
    }
    return info;
}
```

NSMutableDictionary

- Common Methods

```
// Mutation
- (void)setObject:(id)object forKey:(id)key;
- (void)removeObjectForKey:(id)key;
```

- Example

```
NSDictionary *bestFriends = [NSMutableDictionary dictionary];

// Sally's best friend is Fred
[bestFriends setObject:@"fred" forKey:@"sally"];

// Uh oh....Sally is mad at Fred...
// Sally's best friend is now Frank!
[bestFriends setObject:@"frank" forKey:@"sally"];
```

Sets

NSSet

NSSet

- Collection of unordered objects

NSSet

- Collection of unordered objects
- Members must implement
  - `-hash`
  - `-isEqual:`

NSSet

- Collection of unordered objects
- Members must implement
  - `-hash`
  - `-isEqual:`
- Functionality
  - Creation
  - Accessing and enumerating members
  - Testing for membership
  - Retains its contents

NSSet

- Has a mutable subclass, `NSMutableSet`
  - Adding, Removing
  - Set operations: Union, Intersect, …
- Usage very similar to `NSArray`

NSIndexSet

- A Collection of unique unsigned integers - `NSUInteger`
- Efficient storage and enumeration of indexes
  - Very efficient, better than `NSSet` of `NSNumbers`
- Has a mutable subclass, `NSMutableIndexSet`

NSIndexSet

- Example – Gathering index of students getting an “A”

```
NSArray *allStudents = ....; // all the students
NSMutableIndexSet *AStudentIndexes = [[NSMutableIndexSet alloc] init];

NSUInteger idx = 0;
for (idx = 0; idx < [allStudents count]; idx++) {
    Student *student = [allStudents objectAtIndex:idx];
    if ([student isLetterGradeA]) {
        [AStudentIndexes addIndex: idx];
    }
}
```

NSIndexSet

- Example – Gathering index of students getting an “A”

```
NSArray *allStudents = ....; // all the students
NSMutableIndexSet *AStudentIndexes = [[NSMutableIndexSet alloc] init];

NSUInteger idx = 0;
for (idx = 0; idx < [allStudents count]; idx++) {
    Student *student = [allStudents objectAtIndex:idx];
    if ([student isLetterGradeA]) {
        [AStudentIndexes addIndex: idx];
    }
}
```

```
NSUInteger idx = [AStudentIndexes firstIndex];
while (idx != NSNotFound) {
    NSLog(@”%@ is doing great!”, [allStudents objectAtIndex:idx])
    idx = [AStudentIndexes indexGreaterThanIndex:idx];
}
```

Property Lists

Property Lists

Property Lists

- Representation of a data hierarchy

Property Lists

- Representation of a data hierarchy
- Data items are of limited types
  - Strings, Numbers, Dates, Binary Data
  - Arrays, Dictionaries

Property Lists

- Representation of a data hierarchy
- Data items are of limited types
  - Strings, Numbers, Dates, Binary Data
  - Arrays, Dictionaries
- Store as standard XML (plist)

Property Lists

- Representation of a data hierarchy
- Data items are of limited types
  - Strings, Numbers, Dates, Binary Data
  - Arrays, Dictionaries
- Store as standard XML (plist)
- Common iPhone, Mac OS X storage mechanism
  - User Defaults
  - App Configuration Files

Property List

- XML elements
  - `NSArray` – <array>
  - `NSDictionary` – <dictionary>
  - `NSString` – <string>
  - `NSData` – <data>
  - `Boolean` – <true/>, or <false/>

Property Lists

- Example

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN"
"http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
    <key>Fred</key>
    <dict>
        <key>Assignment1</key>
        <integer>95</integer>
        <key>IsGraduateStudent</key>
        <true/>
    </dict>
</dict>
</plist>
```

Property Lists

- Example

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN"
"http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
    <key>Fred</key>
    <dict>
        <key>Assignment1</key>
        <integer>95</integer>
        <key>IsGraduateStudent</key>
        <true/>
    </dict>
</dict>
</plist>
```

Property Lists

- Example

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN"
"http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
    <key>Fred</key>
    <dict>
        <key>Assignment1</key>
        <integer>95</integer>
        <key>IsGraduateStudent</key>
        <true/>
    </dict>
</dict>
</plist>
```

Property Lists

- Example

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN"
"http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
    <key>Fred</key>
    <dict>
        <key>Assignment1</key>
        <integer>95</integer>
        <key>IsGraduateStudent</key>
        <true/>
    </dict>
</dict>
</plist>
```

Property Lists

- Example

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN"
"http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
    <key>Fred</key>
    <dict>
        <key>Assignment1</key>
        <integer>95</integer>
        <key>IsGraduateStudent</key>
        <true/>
    </dict>
</dict>
</plist>
```

Property Lists

- Example

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN"
"http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
    <key>Fred</key>
    <dict>
        <key>Assignment1</key>
        <integer>95</integer>
        <key>IsGraduateStudent</key>
        <true/>
    </dict>
</dict>
</plist>
```

Persistence

- `NSArray`, `NSDictionary`

```
// Reading
+ (id)arrayWithContentsOfURL:(NSURL *)aURL;
+ (id)dictionaryWithContentsOfURL:(NSURL *)aURL;

// Writing
- (BOOL)writeToURL:(NSURL *)aURL atomically:(BOOL)flag
```

- `NSPropertyListSerialization` Class
  - Fine control
  - Convert between plist serialization format

An Example

- Create and save a property list

```
NSDictionary *fredsInfo = [NSDictionary dictionaryWithObjectsAndKeys:
                                [NSNumber numberWithInt:95], @"Assignment1",
                                [NSNumber numberWithBool:NO], @"IsGradStudent",
                                nil];

NSDictionary *classInfo = [NSDictionary dictionaryWithObjectsAndKeys:
                                fredsInfo, @"Fred",
                                nil];

[classInfo writeToFile:@"/tmp/classInfo.plist" atomically:YES];
```

An Example

- Create and save a property list

```
NSDictionary *fredsInfo = [NSDictionary dictionaryWithObjectsAndKeys:
                                [NSNumber numberWithInt:95], @"Assignment1",
                                [NSNumber numberWithBool:NO], @"IsGradStudent",
                                nil];

NSDictionary *classInfo = [NSDictionary dictionaryWithObjectsAndKeys:
                                fredsInfo, @"Fred",
                                nil];

[classInfo writeToFile:@"/tmp/classInfo.plist" atomically:YES];
```

- Loading a property list

```
NSDictionary *classInfo =
          [NSDictionary dictionaryWithContentsOfFile:@"/tmp/classInfo.plist"];
```

Reading

- CocoaFundamentals.pdf
  - A Bit of History: P.57, 58
- “Collections Programming”
  - http://developer.apple.com/mac/library/documentation/Cocoa/Conceptual/Collections/Collections.pdf
- Online Class References
  - http://developer.apple.com/iphone/library/documentation/Cocoa/Reference/Foundation/ObjC_classic/index.html
  - Focus on NSArray, NSDictionary