Objective-C Object Literals Quick Reference

Object literals ease the construction of NSString and NSNumber object representations of basic types (known as "boxing") as well as NSArray's and NSDictionary' s. They also allow for subscripting of arrays and dictionaries and even of custom class instances.

The Old Way

```
NSNumber *v1 = [NSNumber numberWithFloat: 1.2358f];
NSMutableArray *mArr = [NSMutableArray arrayWithObjects:
    v1, [NSNumber numberWithUnsignedInt:255u],
nil];
[mArr replaceObjectAtIndex:1 withObject:[NSNumber numberWithChar:'z']];

NSString *k1 = [NSString stringWithUTF8String:"key1"];
NSString *k2 = [NSString stringWithUTF8String:"key2"];
NSMutableDictionary *mDict = [NSMutableDictionary
    dictionaryWithObjectAndKeys:[NSString stringWithUTF8String:"Hello", k1,
                                [NSString stringWithUTF8String:"World"],k2,
nil];
[mDict setObject:[NSString stringWithUTF8String:"Ciao" forKey:k1]];
```

With New Object Literals

```
NSNumber *v1 = @1.2358F;
NSMutableArray *mArr = [@[v1, @255U] mutableCopy];
         // array/dict literals are immutable so we need a mutable copy
mArr[1] = @'z';


NSMutableDictionary *mDict = [ @{
    @"key1" : @"Hello",
    @"key2" : @"World"
} mutableCopy];
mDict[@"key1"] = "@Ciao";
```

Syntax Reference

string (char *)	@"mystring"
int	@5
unsigned int	@5U
long int	@5L
long long int	@5LL
float (single precision)	@1.234F
float (double precision)	@1.234 // the default
float (long double)	not supported!
BOOL[1]	@YES @NO
C char	@'a' // [NSNumber numberWithChar:]
C++ bool	@true @false
enums	@(UIControlEventTouchUpInside) typedef enum : unsigned char { Red, Green, Blue } Color; NSNumber *c = @(col);
compiler constants (with exceptions[2])	@(INT_MAX) @(INT_MIN)

const variables	const int kMyConstant = 5;
	NSNumber *n = @(kMyConstant);
	
	NSString * const kTouchUpEvent
	= @"touchUp";
	NSString *myString = @(ktouchUpEvent);

Boxed Expressions	@(-INT_MAX - 1);
	@(M_PI / 2);
	
	char *cCharPtr = "abcdef";
	@(cCharPtr);
	
	@(getenv("PATH"));

Arrays, Defining (immutable only)	NSArray *arr = @[@"string", @2L, @(kUp), self.view];

Arrays, Subscripting	NSLog(@"%@", arr[0]);
	UIView *lastView = arr[arr.count-1];

Arrays, Substitution
(mutable only)
	int idx = 0
	arr[idx] = @"newEntry";
	NSNumber *n = @2;
	arr[[n intValue]] = @(kDown);

Dictionaries, Defining
(immutable only)
	NSDictionary *dict = @{
	@"key1" : @"value1",
	@"key2" : @3.14F,
	@5.01 : [NSNull null],
	self : [NSDate date]
	};

Dictionaries, Subscripting	NSLog(@"%@", dict[@"key1"]);
	NSDate *later = [dict[self] dateByAdding-TimeInterval:60];

Dictonaries, Substitution
(mutable only)
	dict[@"key1"] = @"newValue";
	dict[self] = [NSDate distantFuture];

[1] Currently not supported in iOS
[2] Compiler expressions must resolve to a compatible type. See "Caveats"
[3] Keys must conform to <NSCopying>

Courtesy of Club15CC

Subscripting Custom Objects

Array Style

```
@implementation MyViewCollectionView

//...
- (id)objectAtIndexedSubscript:(NSUInteger)idx {
    return self.subviews[idx];
}
- (void)setObject:(id)anObject atIndexedSubscript:(NSUInteger)idx {
    [self insertSubview:anObject belowSubview:self.subviews[idx]];
    [self.subviews[idx] removeFromSuperView];
}
//...
@end

// usage for MyViewCollection *viewColl;
// Replace lowest level subview with new view
viewColl[0] = [[UIView alloc] initWithFrame: CGRect(0, 0, 100, 100)];
```

Dictionary Style

```
extern NSString * const kButtonUp = @"up";
extern NSString * const kButtonDown = @"down";

@implementation MyImageButton {
    NSMutableDictionary *btnImages;
}
//...
- (id)objectForKeyedSubscript:(id)aKey {
    return btnImages[aKey];
}
- (void)setObject:(id)anObject forKeyedSubscript:(id)aKey {
    btnImages[aKey] = anObject;

    // Update button images
    [self setImage:(UIImage *)anObject forState:(aKey == kButtonUp ? UI-
ControlStateNormal : UIControlStateHighlighted)]
}
//...
@end

// usage for MyImageButton *btn;
// Set the up and down state images
btn[kButtonUp] = myUpImg;
btn[kButtonDown] = myDownImg;
```

Caveats

NSMutableArray *arr = ...
arr[1]++; // wrong
arr[1] = @([arr[1] intValue]+1); // correct
	Can't use a binary operator (arr[1] + 1) on an object

@INT_MAX // works but not advisable @INT_MIN // error @(INT_MIN) // correct	@INT_MIN yields a compiler error as it's defined as #define INT_MIN (-2147483647-1) which requires a boxed expression

NSMutableArray *arr = @[@1, @2]; // warning
NSMutableArray *arr
= [@[@1, @2] mutableCopy]; // correct
	Compiler warning since array and dictionary literals are immutable

NSNumber *n1 = @123;
NSNumber *n2 = @123;
if (@n1 == @n2) ... // dangerous
if ([n1 isEqualToNumber:n2])... // better
if (@n1 > @n2) ... // wrong
if ([n1 compare:n2] // correct
== NSOrderedAscending)
	Comparison with ==, >,<,>=,<= actually compares pointer addresses rather than values. Object literals, both strings and numbers are not guaranteed to have the same object represent the same value. Currently the compiler implements NSString this way (all @"myString" code will reference the same memory space) hence why string1 == string2 works however the docs warn against relying on this behavior as it is subject to change