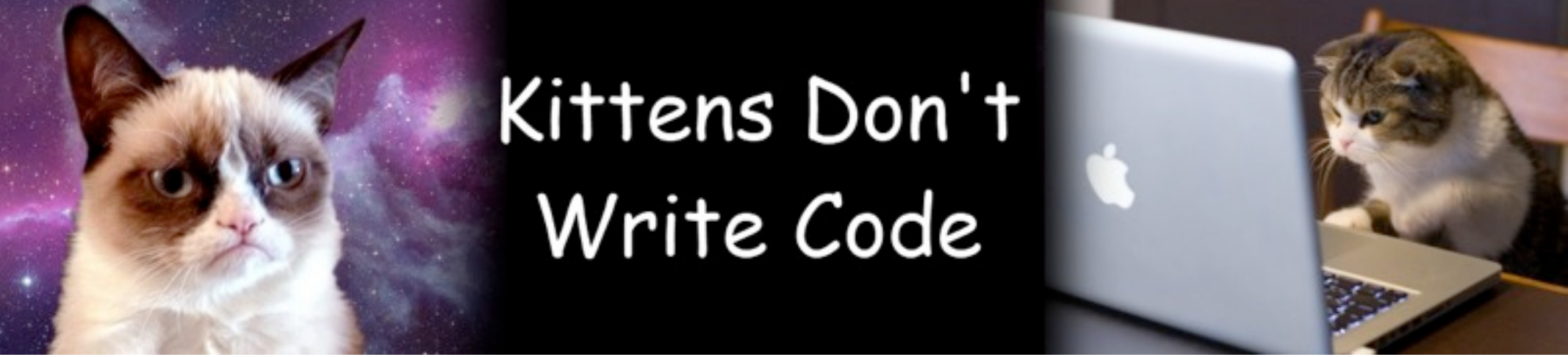


Episode 9: Introducing NSDog

First man invented the Wheel

Then other stuff happened

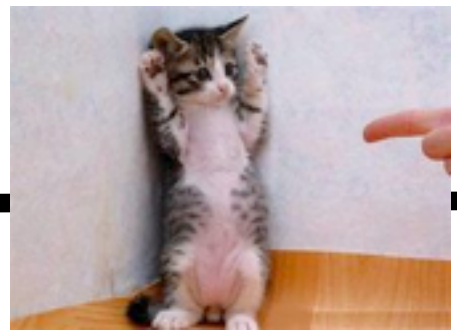
and then there was NSDog!

The Trouble with Kittens...

So your object isn't doing what you expected, so you throw in an NSLog and try to catch the bug just before it crashes...

```
Kitty* _kitty =
   [[Kitty alloc] init];
```

```
if (kitty.barfed == NO)
NSLog(@"Good Kitty!");
```



> Good Kitty!
(hmm, seems ok...)

```
kitty.barfed = YES;
kitty = nil;
```

Uh oh ...

CRASH!

Oops. Looks like we missed it,
should we just add a log after every single line?

There has to be a better way of doing this...

NSLog only checks an object at a single point in time,
stepping through code in the debugger is slow and tedious.

How about ...

something that tracks an object over time
something that tells you when it changes,
something that tells you it deallocated,
something you just set-it-and-forget-it,
something really ... furry?

Enter the Dog!

```
NSDog(id objectToObserve, NSString* keypathObserved);
```

OK Kitty, lets try that again ...

```
Kitty* _kitty =
   [[Kitty alloc] init];
```

```
NSDog(_kitty,@"barfed")
;
```

```
_kitty.barfed = YES;
_kitty = nil;
```

Debugger Output:
> Bark! Kitty barfed: YES
> Bark! Kitty deallocated

Gotcha!

BREAKPOINT

NSDog *observes* your object
barks whenever it changes
barks when it gets deallocated

Cool, so how does this Dog thing work?

NSDog is just a #define macro that creates a Dog object,
and **attaches** it to your observed object as an associated object

```
void NSDog(id object, NSString* keypath)
{
    [Dog dogAttachedTo: object keypath: keypath];
}
```

Srsly, what r u doing?

Properties and iVars are really just keypaths

The Dog sets itself up as a KVO observer for the keypath you select as a NSString.

```
@interface Kitten : NSObject

@property BOOL  behavingBadly;

@end


NSDog(_badKitty, @"behavingBadly");
```

Pick some property (or ivar), use it's name as a string for keypath

Get this dog off me. Now.

Setting up NSDog in your project

```
#include “NSDog.h”

@implementation ViewController

- (void)viewDidLoad
{
    [super viewDidLoad];

    Kitten* kitty = [[Kitten alloc] init];

    kitty.behavingBadly = NO;

    NSDog(kitty, @"behavingBadly");

    kitty.behavingBadly = YES;
    kitty = nil;

}

@end
```

All you need is this

Then you can do this

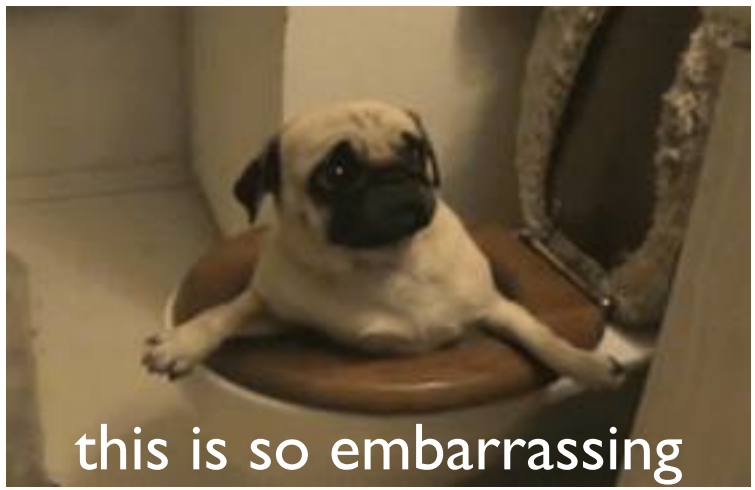
And get this

Debugger Output:
> Bark! Kitten changed behavingBadly: YES
> Bark! Kitten deallocated

Don’t worry, Dogs clean up after themselves

When an object is deallocated,
the Dog removes itself as a KVO observer just BEFORE it goes down the drain.

This neat little trick prevents what would usually result in the classic “KVO dealloc observer” problem.



But wait, there's more!

You can directly create a Dog yourself if you like:

```
@interface Dog : NSObject

@property (assign) BOOL barkWhenObjectIsDeallocated;
@property (assign) BOOL breakpointOnBark;
@property (assign) BOOL breakpointOnDealloc;

+ (Dog*)dogAttachedTo:(id)object keypath:(NSString*)keypath;
+ (int)removeDogsFrom:(id)object forKeypath:(NSString*)keypath;
```

Maybe you'd like a Dog to automatically create a breakpoint for you,
so you can see what's going on *at that exact moment it changed*.

Teaching the Dog Tricks

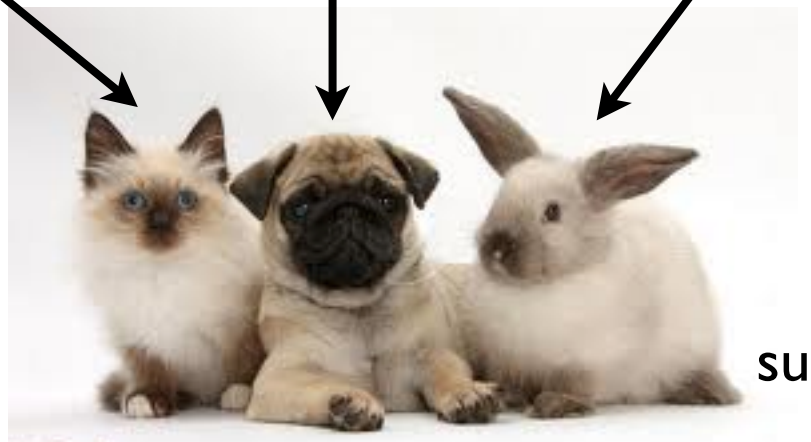
Because a Dog is really just a KVO observer, there are some cool tricks it can do:

```
+ (Dog*)watchDogForObject:(id)object
                  keypath:(NSString*)keypath
   relayObservedChangesTo:(id)receiver;
```

Like safely relaying KVO observations to an interested third party.

```
// standard KVO Override in class Bunny.h
- (void)observeValueForKeyPath:(NSString *)keyPath
                      ofObject:(id)object
                        change:(NSDictionary *)change
                       context:(void *)context {
    NSLog(@"Sup kitty.");
}
```

object Dog receiver

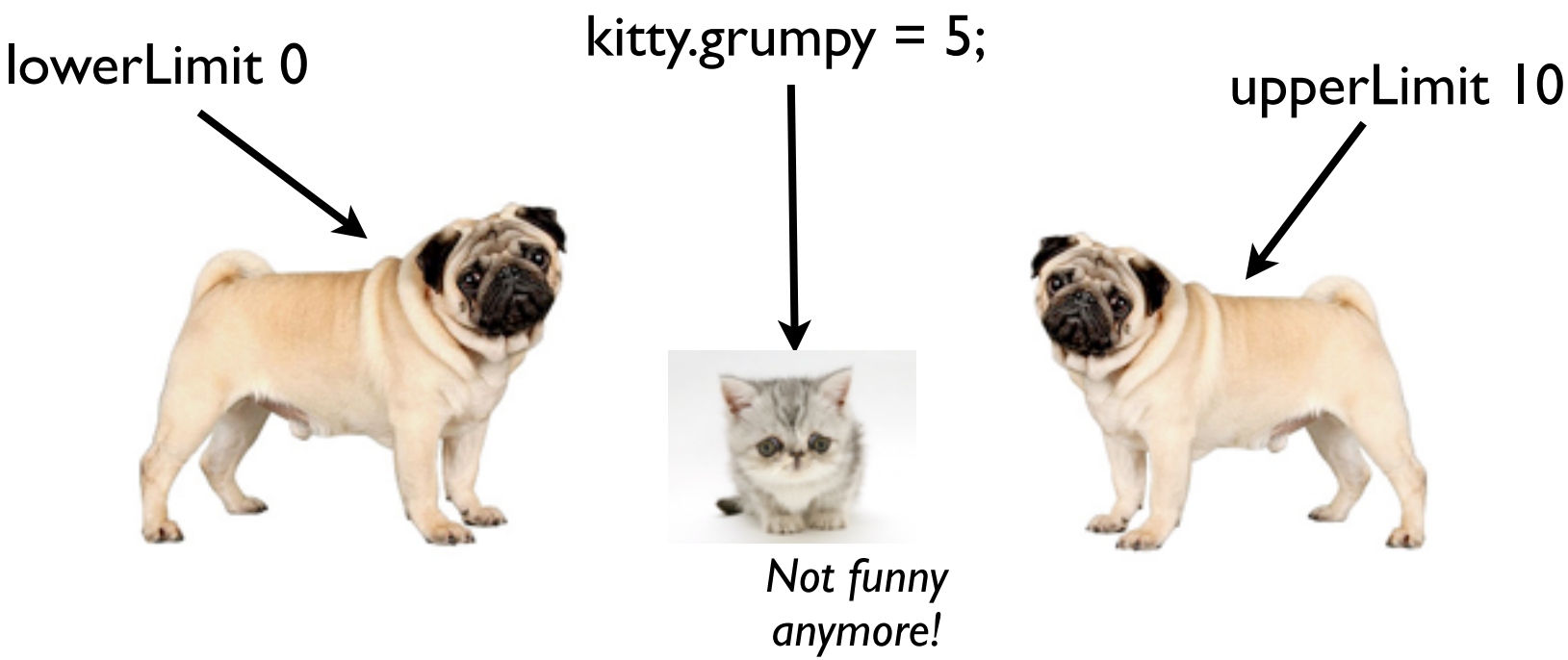


sup.

Guard Dogs

If your property is just a scalar (int's, float's, BOOL's), you can make a Guard Dog that will only bark if the observed value exceeds limit's you set for it.

```
+ (Dog*)guardDogForObject:(id)object
                  keypath:(NSString*)keypath
               lowerLimit:(CGFloat)lowerLimit
               upperLimit:(CGFloat)upperLimit;
```



Easy KVO for ANY object

Because Dog's do all the hard work of setting up and taking down KVO for you, it's dead simple to execute custom code whenever the object changes.

Two additions to NSObject add callbacks or blocks to ANY object:

```
@interface NSObject (NSDogCategory)

- (BOOL)addObserver:(id)observer forKeyPath:(NSString *)keyPath callback:(SEL)callback;
- (BOOL)addObserver:(__weak id)weakObserver forKeyPath:(NSString *)keyPath
              block:(void(^)(void))executionBlock;

@end
```

```
[kitty addObserver:bunny forKeyPath:@"isAttacking" callback:@selector(jumpAway)];
```

any object

any other object

```
- (void)jumpAway
{
    self.gotAwaySafely = TRUE;
}
```

One More Kitten...

With a bit of Duct Tape and Voodoo,
Dog's are even capable of observing changes to some ***concrete*** & ***struct*** types!

CGRect
CGPoint
CGSize
NSValue
NSNumber

```
NSDog(_kitty, @"frame");
```

Wait, kitty has no frame ...
uh oh.

Adopt a Dog

You can test drive NSDog, clone the example project* from GitHub:

https://github.com/xtreme-christopher-larsen/NSDog

Bugs & Feature Requests ...
you know the drill ;)

please don't try Dogs

* *May contain Kittens.*