Intro to Objective-C

-relatively thin layer on top of C

-strict superset of C (a C program will happily compile on an Obj-C compiler)

object declaration syntax is similar to C structs:

```
struct point {
    int x;
    int y;
};
```

C

```
@interface Point: NSObject {
    int x;
    int y;
}
@end
```

Obj-C

A Point object declaration would be more like:

```
@interface Point: NSObject {
   int x;
   int y;
}
-(int) getXvalue;
-(int) getYvalue;
-(void) setXvalue:(int)n;
-(void) setYvalue:(int)n;

@end
```

Method declaration syntax:

```
-( retType ) methodName: arg1
```

Simplifying the object a little:

```
@interface Xvalue: NSObject {
    int x;
}
-(int) getXvalue;
-(void) setXvalue:(int)n;

@end
```

Typically these are in separate header and implementation files:

in the .h file

```
@implementation Xvalue
    int x;
}
-(int) getXvalue {
    return x;
}
-(void) setXvalue:(int)n {
    x = n;
}

@end
```

in the .m file

```
@interface Xvalue: NSObject {
    int x;
}
-(int) getXvalue;
-(void) setXvalue:(int n);
@end

@implementation Xvalue
-(int) getXvalue {
    return x;
}
-(void) setXvalue:(int n) {
    x = n;
}
@end

int main() {
        Xvalue *xvOb = [[Xvalue alloc] init];
        [xvOb setXvalue:34];
        printf("xvOb x-value= %d\n", [xvOb getXvalue]);
}
```

Using this object in a main()

Method calling syntax: `[Object method:arg1]`

Common Objective-C Types: NSString

```
NSString *s = @"boot camp";

NSLog(@"Welcome to %@",s);

s=[NSString stringWithUTF8String:"C_str"];
```

```
char *str = [s UTF8String];

int len = [s length];

char c = [s characterAtIndex:2];
```

Common Objective-C Types: NSNumber

```
NSNumber *fn = [NSNumber numberWithFloat:9.5];
NSNumber *fn = @9.5; // new syntax

float f = [fn floatValue];

int num = [fn intValue]; // this gives you 9
```

```
NSNumber *b = [NSNumber numberWithBool:YES];
NSNumber *b = @YES; // new syntax

BOOL b2 = [b boolValue];
```

Why bother wrapping numbers in an object??

So you can use them in container types like NSArray

Common Objective-C Types: NSArray

```
NSArray *a = @[ @"hello",@YES,@49,someObj ];

BOOL b = [a[1] boolValue]; // b=YES

int c = [a count]; // c=4

NSLog(@"%@",a[0]); // prints "hello"

id ob_x = [a objectAtIndex:3 ]; // huh?
```

Common Objective-C Types: NSDictionary

```
NSDictionary *dict = @{
    @"name" : @"Fred",
    @"age : [NSNumber numberWithInt:55],
    @"theDate" : [NSDate date]
};
```

```
int his_age = [dict[@"age"] intValue];

NSLog(@"name= %@",[dict objectForKey:@"name"];

NSArray *k = [dict allKeys];

NSArray *v = [dict allValues];
```

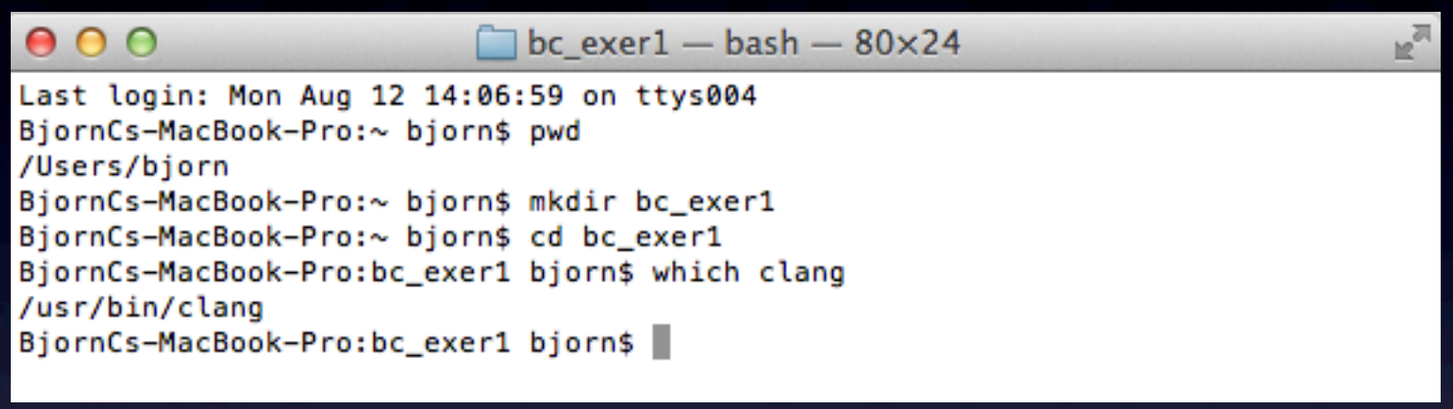
Objective-C

fun with clang

Open a terminal window(or 2)

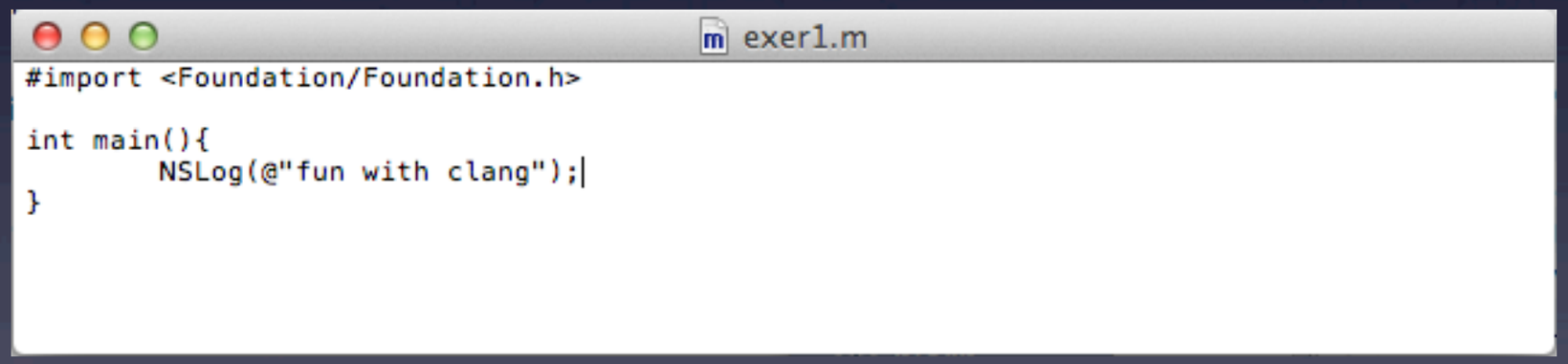
If you need to go to Launchpad->Other, you might add it to your dock.

-Xcode is an IDE, but the command line is still very useful for development.



open TextEdit and enter this code

note: you'll need to do TextEdit->Format->Make Plain Text



clang -framework Foundation exer1.m
./a.out

You should see:

```
BjornCs-MacBook-Pro:exer1 bjorn$ ls
exer1.m
BjornCs-MacBook-Pro:exer1 bjorn$
BjornCs-MacBook-Pro:exer1 bjorn$
BjornCs-MacBook-Pro:exer1 bjorn$
BjornCs-MacBook-Pro:exer1 bjorn$
BjornCs-MacBook-Pro:exer1 bjorn$ clang -framework Foundation exer1.m
BjornCs-MacBook-Pro:exer1 bjorn$ ls
a.out   exer1.m
BjornCs-MacBook-Pro:exer1 bjorn$ ./a.out
2013-08-19 21:04:23.249 a.out[75695:707] fun with clang
BjornCs-MacBook-Pro:exer1 bjorn$
```

tells the clang preprocessor to add the Foundation framework header to the code

```
#import <Foundation/Foundation.h>

int main(){
        NSLog(@"fun with clang");
}
```

NSLog() is the standard Obj-C way to generate debugging output (also gives you a timestamp and process ID)

Now we'll work through some examples