Objective-C

Simple Data Types

- BOOL - YES, NO
- int - any whole (integer) number
- float - Decimal Number
- char - A single character

Objective-C Data Types

- NSString - @”This is a string”
- NSArray - An ordered set of objects
- NSDictionary - A set of objects and keys
- NSNumber - An object wrapper for numbers
- NSData - “bag of bits”
- NSDate - date object

for loops

```
for (int i=0; i < 100; i++)
{
    NSLog(@”The current number is: %d”, i);
}
```

NSString

- One of the most frequently used Objects
- Can be created without using the standard Objective-C object instantiation syntax
- NSString *test = @”Here is a string”;

Placeholders in NSStrings

Specifier	Description
%d, %i	int
%f	float/double
%%	% character
%c	char
\n	New Line
%@	NSObject Description

```
NSString *logString = [NSString stringWithFormat:@" The current number is: %d", i];
```

updated for loop

```
for (int i=0; i < 100; i++)
{
    NSString *logString = [NSString stringWithFormat:@" The current number is: %d", i];
    NSLog(logString);
}
```

updated for loop

```
for (int i=0; i < 100; i++)
{
    NSString *logString;
    logString = [NSString stringWithFormat:@" The current number is: %d", i];
    NSLog(logString);
}
```

updated for loop

```
for (int i=0; i < 100; i++)
{
    NSString *logString;
    logString = [NSString stringWithFormat:@" The current number is: %d", i];
    NSLog(@"%@",logString);
}
```

if statement

```
if (condition) {
    //Do Something
}
```

```
if (i==3) {
    NSLog(@”i is equal to 3”);
}
```

Relational Operators

Operator	Result
==	Equal To
!=	Not Equal To
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to

Arithmetic Operators

Operator	Result
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulo (Remainder)

Other Operators

Operator	Result
!	Not
&&	Logical AND
||	Logical OR
/	Division
%	Modulo (Remainder)

if else statement

```
if (condition) {
      //Do Something
} else {
    // Do Something else
}

if (i==3) {
      NSLog(@”i is equal to 3”);
} else {
      NSLog(@”i is NOT equal to 3”);
}
```

Putting it together

```
for (int i=0; i < 100; i++)
{
    NSString *logString;

    if (i==3) {
        logString = [NSString stringWithFormat:@"%d IS equal to 3", i];
    } else {
        logString = [NSString stringWithFormat:@"%d IS NOT equal to 3", i];
    }
    NSLog(logString);
}
```

Lets try something

Write a program that iterates over the numbers 1-100. If the number is divisible by 3 print Fizz, if the number is divisible by 5 print Buzz. If the number is divisible by both 5 and 3 print FizzBuzz. Otherwise just print the number.

Lets Work Through It

Demo of data types and control flows

Sending Messages in Objective-C

- Syntax
  - [receiver action]
- Sending Single Argument
  - [receiver action:argument]
- Sending Multiple Arguments
  - [receiver action1:argument1 action2:argument2]

Sending Messages in Objective-C

- Most languages pass parameters to functions or methods in the following manner.
  - sum(X,Y) - Example sum(3,4) would return 7
- Objective-C uses named parameters to do this
  - [self sumX:firstVariable andY:secondVariable]
  - [self sumX:3 andY:4] returns 7

Classes

- Interface (.h)
  - Declaration Part
- Implementation (.m)
  - Definition Part

The Interface

@interface ClassName : SuperClassName

properties

method declarations

@end

The Implementation

```
@implementation ClassName
method definitions
{
}

@end
```

Methods

- Two types of methods
  - Instance Level Methods
  - Class Level Methods

Instance Level Methods

- Can be called by Instances of the class, for example ‘bill’ of the person class can call an instance method
- Prefixed by the (-) sign
- e.g. - (int)ageInDogYears;

Class Level Methods

- Can only be called on the class itself and not on instances, so ‘bill’ cannot call a class method but ‘Person’ can
- e.g. [Person createPersonWithName:@”Sue”];
- Prefixed by a (+) sign

Object Instantiation

```
NSString *string = [[NSString alloc] init];

Person *bill = [[Person alloc] init];
Person *sue = [[Person alloc] init];
Person *bob = [[Person alloc] init];
```

- Allocating - Allocates (in memory) enough space for all the instance variables (properties) in your class
- Initialization - Initializes the class and the allocated properties to default values

Custom Object Initialization

```
Person *bill = [[Person alloc] initWithAge:39];
Person *sue = [[Person alloc] initWithAge:23];
Person *bob = [[Person alloc] initWithAge:27];
```

```
- (id)initWithAge:(int)age
{
    self = [super init];
    if (self) {
        // initialize variables
        self.age = age;
    }
    return self
}
```