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

<table>
<thead>
<tr>
<th>Specifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>%d, %i</td>
<td>int</td>
</tr>
<tr>
<td>%f</td>
<td>float/double</td>
</tr>
<tr>
<td>%%</td>
<td>% character</td>
</tr>
<tr>
<td>%c</td>
<td>char</td>
</tr>
<tr>
<td>\n</td>
<td>New Line</td>
</tr>
<tr>
<td>%@</td>
<td>NSObject Description</td>
</tr>
</tbody>
</table>

NSString *logString = [@"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);
}

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] stringValue];
    NSLog(@"%@", logString);
}
if statement

if (condition) {
    //Do Something
}

if (i==3) {
    NSLog(@"i is equal to 3");
}
## Relational Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>==</td>
<td>Equal To</td>
</tr>
<tr>
<td>!=</td>
<td>Not Equal To</td>
</tr>
<tr>
<td>&gt;</td>
<td>Greater than</td>
</tr>
<tr>
<td>&lt;</td>
<td>Less than</td>
</tr>
<tr>
<td>&gt;=</td>
<td>Greater than or equal to</td>
</tr>
<tr>
<td>&lt;=</td>
<td>Less than or equal to</td>
</tr>
</tbody>
</table>
# Arithmetic Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>Addition</td>
</tr>
<tr>
<td>-</td>
<td>Subtraction</td>
</tr>
<tr>
<td>*</td>
<td>Multiplication</td>
</tr>
<tr>
<td>/</td>
<td>Division</td>
</tr>
<tr>
<td>%</td>
<td>Modulo (Remainder)</td>
</tr>
</tbody>
</table>
# Other Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>!</td>
<td>Not</td>
</tr>
<tr>
<td>&amp;&amp;</td>
<td>Logical AND</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>/</td>
<td>Division</td>
</tr>
<tr>
<td>%</td>
<td>Modulo (Remainder)</td>
</tr>
</tbody>
</table>
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

```swift
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);
}
```
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.
Let's 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

@interface 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
}