Functional Programming in Objective C and Go

Concepts of Programming Languages

Simon Treutlein

Agenda

- Objective C
- Functional Programming
- Closures
- Lazy Evaluation
- Immutability
- Recursion
- Conclusion

Objective C

- first appeared 1984
- OS X, iOS → Apple acquiring NeXT (1996)
- introduction Swift 2014
- modern <=> legacy
- object-oriented

Functional Programming

- Relevance
  - Erlang (Facebook Chat)
  - Haskell (AT&T,Facebook,Google)
-
- Key Concepts

Pure Functions

- takes input
- reproduces output
- do not rely on global state/outside variables
- same output for same input

Pure Functions

Go

```
func sum(a, b int) {
    return a + b
}
```

Objective C

```
- (int) sum: (int) a: (int) b{
    return  a+b;
```

Closures

- function that takes multiple argument → sequence of functions
- enables currying

```
function add (a, b) {
  return a + b;
}

add(3, 4);
```

```
function add (a) {
  return function (b) {
    return a + b;
  }
}
```

Closures - Go

```
func sequence() func() int {
    i := 0
    return func() int {
        i++
        return i
    }
}
```

Closures – Objective C

blocks

```
^{
    NSLog(@"This is a block");
}
```

```
double (^multiplyTwoValues)(double, double) =
                          ^(double firstValue, double secondValue) {
                              return firstValue * secondValue;
                          };

double result = multiplyTwoValues(2,4);
```

Alternative to blocks

- blocks come with OS X v10.6 and later, and iOS 4.0 and later
- alternatives for blocks
  - function pointer
  - protocol pattern
  - selectors

Alterntive – function pointer

```
void print() {
    NSLog(@"Printed!");
}

void printTwice(void (*toDo)(void)) {
    toDo();
    toDo();
}

int main(void) {
    printTwice(print);
    return 0;
}
```

Alternative – protocol pattern

```
@protocol Command <NSObject>
- (void) printSomething;
@end

@interface DoPrint : NSObject <Command> {
}
@end

@implementation DoPrint
- (void) printSomething {
    NSLog(@"Printed!");
}
@end

void printTwice(id<Command> command) {
    [command printSomething];
    [command printSomething];
}

int main(void) {
    DoPrint* doPrint = [[DoPrint alloc] init];
    printTwice(doPrint);
    [doPrint release];
    return 0;
}
```

Alternative - selector

```
@interface DoPrint : NSObject {
}
- (void) printSomething;
@end

@implementation DoPrint
- (void) printSomething {
    NSLog(@"Printed!");
}
@end

void printTwice(id<NSObject> obj, SEL selector) {
    [obj performSelector:selector];
    [obj performSelector:selector];
}

int main(void) {
    DoPrint* doPrint = [[DoPrint alloc] init];
    printTwice(doPrint, @selector(printSomething));
    [doPrint release];
    return 0;
}
```

Lazy Evaluation

- call-by-need
- evaluation when value is needed
- not native implemented in Go and Objective C
- can be implemented

Immutability

- initialized variable can not be modified
- possible in Go and Objective C

Immutability - Go

mutable

```
type Person struct {
    Name           string
    FavoriteColors []string
}
```

immutable

```
type Person struct {
    name           string
    favoriteColors []string
}
```

- getter and setter added → control over which properties are allowed to change

Immutability – Objective C

- objects mutable by default
- Foundation framework: mutable and immutable variant
- immutable classes are superclasses
- NSMutableArray <=> NSArray

Recursion

- no loops in functional programming
- more difficult to understand
- less performant
- possible in Go and Objective C

Recursion - Go

```
func fib(input int) int {
	fn := make(map[int]int)
	for i := 0; i <= input; i++ {
		var fibonacci int
		if i <= 2 {
			fibonacci = 1
		} else {
			fibonacci = fn[i-1] + fn[i-2]
		}
		fn[i] = fibonacci
	}
	return fn[input]
}
```

Recursion – Objective C

```
-(int) fib: (int) num {

    if (num == 0) {
        return 0;aim
    }
    if (num == 1) {
        return 1;
    }
    return [self fib:num - 1] + [self fib:num - 2];
}
```

Conclusion

Language	Pure Functions	Closure	Lazy Evaluation	Immutability	Recursion
Go	yes	yes	Generally not, can be implemented	yes	yes
Objective C	yes	yes	No, can be implemented	yes	yes