Functional Programming in SWIFT

Functional Programming?

- Not related to SWIFT
- SWIFT can be use in both OOP and FP
- FP is a programming paradigm that emphasizes calculations via mathematical-style functions,
- Immutability and minimizes the use of variables and state.
- Function is easier to test
- Concurrency and parallel processing

void Function

```
func printHello(message: String)
{
        println("hello \(message)")
}


printHello("world")    =>  hello world
```

Function return String

```
func hello(message: String) -> String
{
        return "hello \(message)"
}
```

```
hello("world")  => hello world
```

Nested Function

```
func salut(message: String) -> String
{
        func nestedHello(message: String) -> String    {
                return "Hello \(message)"
        }
        return nestedHello(message)
 }


 salut("World !!!")   => Hello World !!!
```

Function return Function

```
func hola(startMessage: String) -> String -> String
{
        func nestedHello(endMessage: String) -> String
        {
                return "\(startMessage) \(endMessage)"
        }
        return nestedHello
}
```

```
hola("Hello")("World !!!")  =>  Hello World !!!
```

Function as Parameter

```
func morgen(startMessage:String, endMessage: String -> String) -> String
{
        let param = "World"
        return "\(startMessage) \(endMessage(param))"
}

func message(msg: String) -> String
{
         return "\(msg) !!!"
}


morgen("Hello", message) =>  Hello World !!!
```

Array Filtering

```
//The imperative way
var evens = [Int]()
for i in 1...10
{
        if i % 2 == 0
        {
                evens.append(i)
        }
}

println(evens)  => [2, 4, 6, 8, 10]
```

Array Filtering

```
//The fonctionnal way
func isEven(number: Int) -> Bool
{
        return number % 2 == 0
}
evens = Array(1...10).filter(isEven)

//functions are just named closures
evens = Array(1...10).filter { (myNmbr) in myNmbr % 2 == 0 }

println(evens)  => [2, 4, 6, 8, 10]
```

Array Filtering

```
//My own filter
func myFilter<T>(source: [T], predicate:(T) -> Bool) -> [T]
{
        var result = [T]()
        for i in source
        {
                if predicate(i)
                {
                        result.append(i)
                }
        }
        return result
 }
evens = myFilter(Array(1...10)){(myNmbr) in myNmbr % 2 == 0}

println(evens)  => [2, 4, 6, 8, 10]
```

Example : Build Index

let words = ["Cat", "Chicken", "fish", "Dog", "Mouse", "Guinea Pig", "monkey"]

=> [(C, [Cat, Chicken]), (F, [fish]), (D, [Dog]), (M, [Mouse, monkey]), (G, [Guinea Pig])]

typealias Entry = (Character, [String])

Example : Build Index

```
func buildIndex(words: [String]) -> [Entry]
{
        var result = [Entry]()
        var letters = [Character]()
        for word in words
        {
                let firstLetter = Character(word.substringToIndex( advance(word.startIndex, 1)).uppercaseString)
                if !contains(letters, firstLetter)
                {
                        letters.append(firstLetter)
                 }
        }
        for letter in letters
         {
                var wordsForLetter = [String]()
                for word in words
                {
                        let firstLetter = Character(word.substringToIndex( advance(word.startIndex, 1)).uppercaseString)
                        if firstLetter == letter
                         {
                                wordsForLetter.append(word)
                         }
                 }
                result.append((letter, wordsForLetter))
         }
        return result
 }

println(buildIndex(words))  => [(C, [Cat, Chicken]), (F, [fish]), (D, [Dog]), (M, [Mouse, monkey]), (G, [Guinea Pig])]
```

Example : Build Index

```
//Words to Array of First Letter
func buildIndex(words: [String]) -> [Entry]
{
        let letters = words.map
        {
                (word) -> Character in
                        Character(word.substringToIndex
                (advance(word.startIndex, 1) ).uppercaseString)
        }
        return [Entry]()
}


println(letters)  => [C, C, F, D, M, G, M]
```

Example : Build Index

```
//My own disctinct Function
func distinct<T: Equatable>(source: [T]) -> [T] {
        var unique = [T]()
        for item in source
        {
                if !contains(unique, item)
                        {
                        unique.append(item)
                }
        }
        return unique
}
```

Example : Build Index

```
//Get Distinct letter
func buildIndex(words: [String]) -> [Entry]
{
        let letters = words.map
         {
                (word) -> Character in
                        Character(word.substringToIndex
                        (advance(word.startIndex, 1) ).uppercaseString)
        }
        let distinctLetters = distinct(letters)
        return [Entry]()
}

println(distinctLetters)   => [C, F, D, M, G]
```

Example : Build Index

```
//Return Entry
func buildIndex(words: [String]) -> [Entry]
{
        let letters = words.map
        {
                (word) -> Character in
                Character(word.substringToIndex(advance(word.startIndex, 1) ).uppercaseString)
        }

        let distinctLetters = distinct(letters)

        return distinctLetters.map
        {
                (letter) -> Entry in return (letter, [])
        }
}


println(buildIndex(words))  => [(C, []), (F, []), (D, []), (M, []), (G, [])]
```

Example : Build Index

```
//Use filtrer to return entry
func buildIndex(words: [String]) -> [Entry]
{
        let letters = words.map
         {
                (word) -> Character in
                Character(word.substringToIndex(advance(word.startIndex, 1) ).uppercaseString)
        }

        let distinctLetters = distinct(letters)

        return distinctLetters.map
        {
                (letter) -> Entry in return (letter, words.filter
                {
                        (word) -> Bool in
                        Character(word.substringToIndex(advance(word.startIndex, 1) ).uppercaseString) == letter
                })
        }
}

println(buildIndex(words))   => [(C, [Cat, Chicken]), (F, [fish]), (D, [Dog]), (M, [Mouse, monkey]), (G, [Guinea Pig])]
```

Example : Build Index

```
//Clean the code
func buildIndex(words: [String]) -> [Entry]
{
        func firstLetter(str: String) -> Character
        {
                return Character(str.substringToIndex(advance(str.startIndex, 1)).uppercaseString)
        }

        let letters = words.map
        {
                (word) -> Character in firstLetter(word)
        }

        let distinctLetters = distinct(letters)

        return distinctLetters.map
        {
                (letter) -> Entry in return (letter, words.filter
                 {
                        (word) -> Bool in firstLetter(word) == letter
                 })
        }
}
```

Example : Build Index

```
//Clean the code
func buildIndex(words: [String]) -> [Entry]
{
        func firstLetter(str: String) -> Character
        {
                return Character(str.substringToIndex(advance(str.startIndex, 1)).uppercaseString)
        }

        return distinct(words.map(firstLetter)) .map
                {
                 (letter) -> Entry in return (letter, words.filter
                        {
                                (word) -> Bool in firstLetter(word) == letter
                        })
                 }
        }
}
```