Practical Techniques for Functional Programming in Swift and Objective C

Chris Woodard
Tampa Bay Cocoaheads

Functional Programming

A language is functional if it has…

- Anonymous functions
- Recursion
- Programming with expressions rather than statements
- Closures
- Currying / partial functions
- Lazy evaluation
- Algebraic data types
- Parametric polymorphism

Functional Programming

- "…***a style of building the structure and elements of computer programs that treats computation as the evaluation of mathematical functions and avoids changing state and mutable data.*** **" - Wikipedia**

- ***"a style of programming which models computations as the evaluation of expressions."*** **-** www.haskell.org

Functional Programming

For our purposes:

- Programming with *functions* and *expressions* rather than *statements*
- Functions are first-class objects
- Anonymous functions (closures)
- Functions do not share mutable state
- Process *lists* instead of *arrays*

Functional Programming

Functions

Functional Programming

Mathematical Functions

This type of function is written as:

$$y = f(x)$$

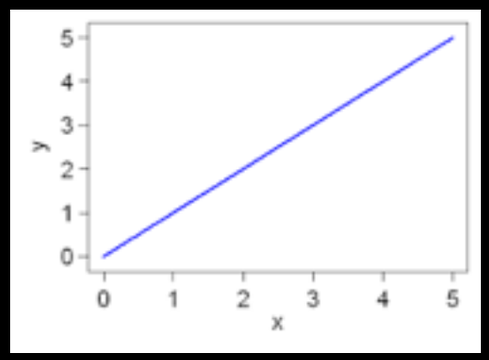
where x is the *domain*, y is the *range*, and *f* is the function.

Functional Programming

Mathematical Functions $y = f(x)$

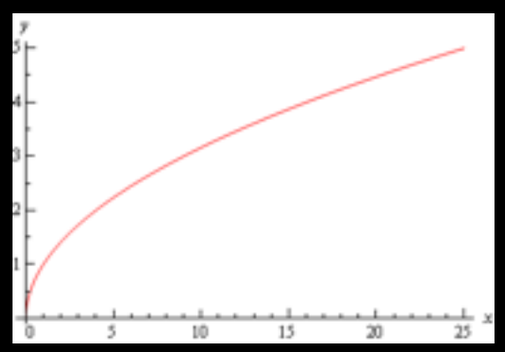
Straight line:

$$y = mx + b$$



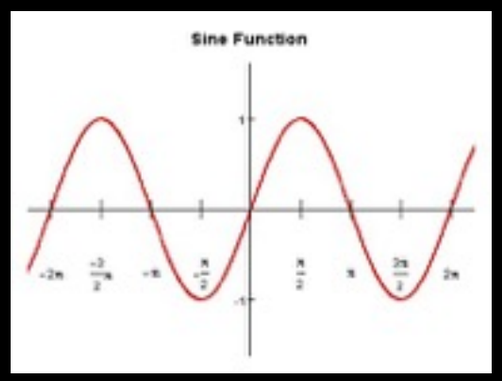
Square root:

$$y = sqrt(x)$$



Sine curve:

$$y = sin(x)$$



Functional Programming

Predicate Functions

$y = f(x)$

(used to compute a TRUE or FALSE output from some type of input, such as whether a number is even)

$$f(x) = \begin{cases} \textbf{TRUE} \text{ if x / 2 has no remainder} \\ \textbf{FALSE} \text{ if x / 2 has a nonzero remainder} \end{cases}$$

Functional Programming

Linguistic Functions

$y = f(x)$

English to Klingon:

greeting = toKlingon(“have a nice day”)

Word Classifier:

type = wordType(“pedantic”)

Functional Programming

Three different ways of saying the same thing:

- A function declares a mapping from one set of values to another set of values.
- A function transforms one set of values to another set of values.
- **A function transforms the values of its parameters to its returned values.**

Functional Programming

“Functions are first-class objects”

- Functions can be *declared* and *passed into other functions* the same as parameter values.

- Functions can be generated inside other functions and returned from those functions to be used in another expression.

- A function passed in or returned this way can be called and will return values the same way as functions declared in the source code

Functional Programming

Declaring and Generating Functions in Objective C

This is done with *blocks*, which are executable chunks of code that are assigned a signature, which is defined by the block type of the variable that holds the reference to the block. Declaring a variable that holds a block reference requires *block type syntax*:

return-type(^*BlockType*)(*param1*, *param2*)

Functional Programming

Declaring and Generating Functions in Objective C

For convenience, it's best to use the typedef keyword to let you shorthand the declaration later.

Some examples are:

```
typedef BOOL (^FloatPredicate)(NSNumber *num);

typedef BOOL (^StringPredicate)(NSNumber *num);

typedef NSString *(^StringBleeper)(NSString *num);
```

These can be used to declare block reference variables.

Functional Programming

Declaring and Generating Functions in Objective C

For convenience, it's best to use the typedef keyword to let you shorthand the declaration later.

Some examples are:

```
typedef BOOL (^FloatPredicate)(NSNumber *num);

typedef BOOL (^StringPredicate)(NSNumber *num);

typedef NSString *(^StringBleeper)(NSString *num);
```

These can be used to declare block reference variables.

Sample code - Generating blocks in Objective C

Sample code - Generating closures in Swift

Functional Programming

“**Anonymous functions**”

- Functions can be declared and passed to other functions as parameters.

- Functions can be generated inside other functions and returned from those functions.

- Functions like this are called closures or blocks.

Functional Programming

“**Curried functions**”

- Curried functions are basically function factories. Outer function with all its parameters is main function.

- Call it with one of its parameters filled and it *curries* the function by generating a closure with that parameter’s value fixed.

Sample code - Curried functions in Objective C

Sample code - Curried functions in Swift

Functional Programming

“**Functions do not share mutable state**”

- state = collective values of variables in a function that control its operation

- shared state = state that is accessible to more than one function

- mutable shared state = state that can be changed by the functions that share it

Sample code - Generating blocks in Objective C

Sample code - Generating closures in Swift

Functional Programming

Benefits of not having shared mutable state:

- One function can’t change a value another function depends on.
- A function’s output is completely determined by the values in its parameters and its program logic.

These make it trivial to unit-test functions.

Functional Programming

One more benefit…

- Functions that don't require shared state are *fully decoupled*.
- The decoupling that those practices produce make it easy to run them i**n parallel**.

These make it trivial to unit-test functions.

Sample code - State in Objective C Functions

Sample code - State in Swift Functions

Functional Programming

"**Process *lists* rather than *arrays***"

- A *lot* of programmatic operations can be broken down into a few basic operations on lists:
  - map(*input-list, transform*) - applies *transform* to each element of *input-list* and returns a new list with those transformed elements in it
  - filter(*input-list, predicate*) - applies *predicate* to each element of *input-list* and returns a subset of its elements in a new list
  - reduce(*input-list, reductor*) - applies *reductor* to each element of *input-list* to compute and return a single value

Sample code - processing lists versus arrays in Objective C

Sample code - map, filter, reduce in Objective C

Sample code - map, filter, reduce in Swift

Functional Programming

Rules (of thumb) to live by:

- Organize your code in terms of *functions*, not *objects*
- Functions *should* only need the values passed in as parameters to compute their output (return).
- One function cannot mutate a value another function depends on.

Functional Programming

More rules (of thumb) to live by:

- Pattern your functions after *map*, *filter* and *reduce*
  - map() - creating a list of objects out of another list of objects and modifying them along the way
  - filter() - fetching a subset of objects from a list
  - reduce() - reducing a list of objects to a single value.