NSPredicate Cheatsheet

Presented by Realm: a mobile database that replaces Core Data and SQLite. Learn more at http://realm.io

- supported by Realm

Format String Summary

- **`@"attributeName == %@"`**
  object's attribute name is equal to value passed in
- **`@"%K == %@"`**
  pass a string variable to %K, it will be represented as a keypath
- **`@"%name IN $NAME_LIST"`**
  templated for predicate, checks if the value of key name is in $NAME_LIST. Uses predicateWithSubstitutionVariables
- **`@"'name' IN $NAME_LIST"`**
  checks if the constant value 'name' is in $NAME_LIST. Uses predicateWithSubstitutionVariables

```
predicateWithFormat: @"title == %@", @"minecraft"
```

Keypath collection queries

`@avg`	returns the average of the objects in the collection as an NSNumber
`@count`	returns the number of objects in a collection as an NSNumber
`@min`	returns the minimum value of the objects in the collection as an NSNumber
`@max`	returns the maximum value of the objects in the collection as an NSNumber
`@sum`	returns the sum of the objects in the collection based on the property

```
predicateWithFormat:
        @"expenses.@avg.doubleValue < 200"
```

Subqueries

SUBQUERY(collection, variableName, predicateFormat)

Iterates through the collection to return qualifying queries

Collection - array or set of objects

variableName - variable that represents an iterated object

predicateFormat - predicate that runs using the variableName

```
predicateWithFormat: @"SUBQUERY(tasks, $task,
$task.completionDate != nil AND $task.user = 'Alex')
.@count > 0"
```

Assume this was run on an array of projects. It will return projects with tasks that were not completed by user Alex

Basic Comparisons

`=,==`	Left hand expression is equal to right hand expression
`>=,=>`	Left hand expression is greater than or equal to right hand expression
`<=,=<`	Left hand expression is less than or equal to right hand expression
`>`	Left hand expression is greater than right hand expression
`<`	Left hand expression is less than right hand expression
`!=,<>`	Left hand expression is not equal to right hand expression
`IN`	Left hand expression must appear in collection specified by right hand expression. i.e. name IN {'Milk', 'Eggs', 'Bread'}
`BETWEEN`	Left hand expression is between or equal to right hand expression. i.e. 1 Between {0, 33}

```
predicateWithFormat: @"expenses BETWEEN {200, 400}"
```

Basic Compound Predicates

`AND,&&`	Logical AND
`OR,||`	Logical OR
`NOT,!`	Logical NOT

```
predicateWithFormat: @"age == 40 AND price > 67"
```

String Comparison Operators

`BEGINS WITH`	Left hand expression begins with the right hand expression
`CONTAINS`	Left hand expression contains the right hand expression
`ENDSWITH`	Left hand expression ends with the right hand expression
`LIKE`	Left hand expression equals the right hand expression: ? and * are allowed as wildcard characters, where ? matches 1 character and * matches 0 or more characters
`MATCHES`	Left hand expression equals the right hand expression using a regex - style comparison

```
predicateWithFormat: @"name BEGINS WITH 'm'"
```

Aggregate Operators

`ANY,SOME`	returns objects where ANY or SOME of the predicate results are true.
`ALL`	returns objects where ALL of the predicate results are true.
`NONE`	returns objects where NONE of the predicate results are true.

```
predicateWithFormat:@"ALL expenses > 1000"
```

Tips, Tricks, & Examples

Common mistakes

Using stringWithFormat: to build predicates, prone to have weird non escaped diacritic characters and artifacts like an apostrophe. Use NSPredicate predicateWithFormat instead.

Using OR OR OR instead of IN, results in repeatable code and can be less efficient

When using REGEX and Matches, make sure they are the last part of your predicate statment so it does less work.

Using SELF

When using a predicate on an array, SELF refers to each object in the array. Here's an example: Imagine you are a landlord figuring out which apartments have to pay their water bill. If you have a list of all the city wide apartment's that still need to pay called **addressesThatOweWaterBill**, we can check that against our owned apartments, **myApartmentAddresses**.

```
NSPredicate *billingPredicate = [NSPredicate
predicateWithFormat: @"SELF IN %@",
addressesThatOweWaterBill]

NSArray *myApartmentsThatOweWaterBill =
[myApartmentAddresses
filteredArrayUsingPredicate:billingPredicate]
```

LIKE wildcard match with * and ?

* matches 0 or more characters. For example:
Let's say we have an array of names we want to filter

```
@["Sarah", "Silva", "silva", "Silvy", "Silvia", Si*"]

[NSPredicate predicateWithFormat: @"SELF ==
%@", "Sarah"]
```

Will return "Sarah"

```
[NSPredicate predicateWithFormat: @"SELF LIKE[c]
%@", "Si*"]
```

Will return "Silva", "silva", "Silvy", "Silvia", "Si*"

? matches 1 character only

```
[NSPredicate predicateWithFormat: @"SELF LIKE[c] %@",
"Silv?"]
```

Will return "Silva", "silva", "Silvy"

Quick tips

CFStringTransform normalizes strings if diacritic insensitive isn't enough. For example you could turn Japanese characters into a Latin alphabetic representation. It's extremely powerful with a lot of methods that you can see here: http://nshipster.com/cfstringtransform/

Make sure your columns are indexed to improve performance of using IN operators

[c] case insensitive: lower & uppercase values are treated the same
[d] diacritic insensitive: special characters treated as the base character

```
predicateWithFormat: @"name CONTAINS[c] 'f'"
```

Keypath collection queries

Keypath collection queries work best when you work with a lot of numbers. Being able to easily call the min or max, adding things up, and then filtering results can be much simpler when you only have to append an extra parameter. By having an array of expenses, you can easily do a quick check on if something is below or above a range of allowed expenses.

```
predicateWithFormat: @"expenses.@avg.doubleValue < 200"
```

How Subqueries work

SUBQUERY(collection, variableName, predicate)

A subquery takes a collection then iterates through each object (as variableName) checking the predicate against that object (variableName). It works well if you have a collection (A) objects, and each object has a collection (B) other objects. If you're trying to filter A based on 2 or more varying attributes of B.

```
[NSPredicate predicateWithFormat: @"SUBQUERY(tasks,
$task, $task.completionDate != nil AND $task.user =
'Alex').@count > 0"]
```

SUBQUERY(…) returns an array. We need to check if its count > 0 to return the true or false the predicate expects.