Removing Currying from Swift

- Frederik Lohner
- David Zhuzhunashvili
- Derek Holland
- Justin Olson

What is Swift

- Swift is Apples new programming language, it allows you to program for all Apple platforms.
- iOS, OS X, tvOS, watchOS.
- First introduced 2014
- Open Source as of Dec 3 2015!!

What are the goals of Swift?

- Easy to understand

```
func greetings(name: String) -> String {
    return "Greetings \(name)!"
}
```

Swift is fairly Pythonic in nature

```
for car in cars {
    print(car)
}

for (key, value) in someDictionary {
    print("Key: \(key) has value \(value).")
}
```

Where are Swift changes discussed

Mailing lists:

- Swift-dev (Swift compiler discussion):
  https://lists.swift.org/mailman/listinfo/swift-dev
- Swift-evolution (Swift language evolution discussion):
  https://lists.swift.org/mailman/listinfo/swift-evolution

Community Structure

Advancing the Swift programming language with a coherent, clear view of its evolution requires strong leadership. The leadership is taken from the community, and works closely with the much broader group of contributors and users.

There is also a code of conduct working group to help in matters of community, culture, and the code of conduct. Most importantly, everyone that uses Swift is a valued member of our extended community.

Community Structure (cont.)

- Project Lead: Apple Inc.
- Core Team: Small group of engineers responsible for strategic decision (currently all Apple engineers)
- Code Owner: Individual responsible for a specific area of the Swift code base
- Commiter: Anybody who has commit access to the Swift code base
- Contributor: Anyone who contributes a patch or helps with code review.

What are the format of Swift changes

Github Pull Request with proposal in markdown format is created.

Categories:

- Introduction
- Motivation
- Proposed Solution
- Detailed Design
- Alternatives Considered
- Rationale (Final team decision)

What is Currying

- Break an operation down to multiple functions based on inputs
- More specifically, translating the evaluation of a multiple-argument function into evaluating a sequence of functions, each with one argument

Can Currying be Useful?

- Yes, it allows the user to pass one variable at a time in case all the variables are not known, but this is not too helpful.
- This is not too helpful because this does not happen very often and there are many ways to avoid the issue without currying.

Example

```
func curried(x: Int)(y: String) -> Float {
    return Float(x) + Float(y)!
}
```

- Currying splits this into two portions the add x portion and add y portion.

Example cont

- So this would break down to

```
func curried(x: Int) -> (String) -> Float {
    return {(y: String) -> Float in
      return Float(x) + Float(y)!
    }
}
```

Why would we remove currying

- Often causes issues with other features of a language without any real benefit
- Causes ambiguous syntax

Example of confusing currying syntax

```
func curry<A, B, C>(f: (A, B) -> C) -> (A -> (B -> C)
  return { a in
             { b in
               return f(a, b)
             }
          }
}
```

Impact on the language

- Removing a language feature will affect the existing code.
- The effects on the existing code, however, are not noticeable enough to bar the removal of currying.

What Warrants the Removal?

- Currying is of very small utility in Swift and can only be applied in very specific cases.
- Currying goes against emerging language practice.

What Warrants the Removal? (cont.)

- Swift strives to be an easy to understand language, and currying does quite the opposite by adding confusion over keyword rules and declaration names of functions
- Removal of currying will greatly simplify the language and remove this ambiguity, bringing Swift closer to what it strives for

Alternatives to Removal

- More explicit Currying such as Python's:

```
def func(x, y=None):
    def inner(y):
        return x + y
    if y is None:
        return inner
    else:
        return inner(b)
```

- This allows use of partial functions without support of currying.

Alternatives to Removal (cont.)

- Another solution would be to keep currying but make it more readable by adding a Scala-like ad-hoc partial application syntax
- This way the language will be more idiomatic as a whole and would add flexibility to currying.

Alternatives to Removal (cont.)

Make syntax more idiomatic:

For example: `foo(_, bar: 2)`

would be a shorthand for: `{ x in foo(x, bar: 2) }`

Greatly increasing readability