

Partial with Free Variables — *Smooth CoffeeScript*

This literate program is *interactive* in its HTML form. Edit a CoffeeScript segment to try it.

Partial function application with free variables

Partial function application is presented in [Smooth CoffeeScript partial application](#). It is a way to create a function from another function where the first arguments are filled in. With the new function we can then ignore those arguments so subsequent calls become easier to read and write.

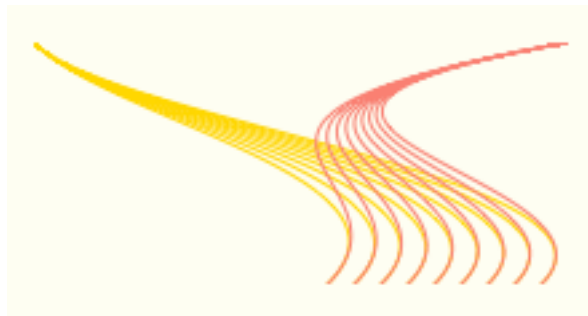
It is not always the case that arguments are so nicely ordered that it is the first ones that need to be held constant. To handle the general case with arbitrary arguments, a special symbol¹ can designate free variables i.e. those arguments that are not fixed.

In the code below² the `movement` function is inspired by the canvas function `bezierCurveTo`. It takes nine arguments, clearly outdoing `bezierCurveTo`'s measly six arguments. When calling such a function from many places, several of the arguments may well be the same.

```
draw = (ctx) -> # Try changing colors below
  ctx.beginPath(); ctx.strokeStyle = 'gold'
  drawMove ctx, (ix for ix in [0...90] by 10)
  ctx.beginPath(); ctx.strokeStyle = 'salmon'
  drawPath ctx, (ix for ix in [0...90] by 10)

movement = (ctx, ax, ay, cp1x, cp1y, cp2x, cp2y, x, y) ->
  ctx.moveTo ax, ay
  ctx.bezierCurveTo cp1x, cp1y, cp2x, cp2y, x, y

drawMove = (ctx, args) ->
  args.forEach (ix) -> movement ctx,
    0, 0, 30, 30, 150+ix, 50, 110+ix, 90
  ctx.stroke()
```



¹In the examples underscore `_` is used. You may want to choose another symbol to avoid clashes with commonly used libraries such as [Underscore](#).

²To run this example standalone you can prepend this [CoffeeKup](#) CoffeeScript:

```
show = if exports? then console.log else alert
(require 'fs').writeFileSync './bezier.html',
  webpage = (require 'coffeescript').render ->
    doctype 5
    html -> meta charset: 'utf-8',
      head -> title 'Bezier path'
      body ->
        canvas id: 'drawCanvas', width: 300, height: 200
        coffeescript ->
          window.onload = ->
            canvas = document.getElementById 'drawCanvas'
            ctx = canvas.getContext '2d'
            alert 'No canvas in this browser.' unless ctx?
            draw ctx if draw?
```

A wrapper function `swirl` that takes only those arguments that might change can cut down on the repetition. The `partialFree` function returns a new function when given a function, its fixed arguments and the placeholder symbol for the variable arguments.

```
_ = undefined
partialFree = (func, a...) -> (b...) ->
  b.reverse()
  func (for arg in a then arg ?= b.pop())...

swirl = partialFree movement, _, _, 0, 30, 30, _, 50, _, 90

drawPath = (ctx, args) ->
  args.forEach (ix) -> swirl ctx, 200, 150+ix, 110+ix
  ctx.stroke()
```

Prior Art

Where did the definition of `partialFree` come from? It started with a [search](#) for existing implementations. The search eventually turned up this JavaScript version from [Angus Croll's blog](#).

```
window.___ = {}; //argument placeholder

Function.prototype.partial = function() {
  if (arguments.length<1) {
    //nothing to pre-assign - return the function as is
    return this;
  }
  var __method = this;
  var args = arguments;
  return function() {
    //build up new arg list, for placeholders use current arg,
    //otherwise copy original args
    var argIndex = 0, myArgs = [];
    for (var i = 0; i < args.length; i++) {
      myArgs[i] = window.___==args[i] ?
        arguments[argIndex++] : args[i];
    }
    return __method.apply(this, myArgs);
  }
}
```

And this CoffeeScript version from [Mirotin](#).

```
_ = {}
partial15lines = () ->
  [func, args...] = arguments
  wrapper = () ->
    i = 0
    j = 0
    res_args = []
    while i < args.length
      if args[i] == _
        res_args.push arguments[j]
        j++
      else
        res_args.push args[i]
        i++
    return func.apply null, res_args
```

Stepwise CoffeeScript Improvements

Those implementations are fine and usable as they are. But here comes one of the most fun activities in CoffeeScript: *code reduction*. It is also useful because less code makes maintenance easier — up to a point — too clever tricks and the code can become harder to understand. In the following line of code reductions, which one would you choose as the best balance between brevity and readability?

In [partial15lines](#) there are some redundant words that can be removed. The use of [arguments](#) can also be replaced with a splat ...

```
_ = {}
partial12lines = (func, args...) ->
  (moreargs...) ->
    i = j = 0
    res_args = []
    while i < args.length
      if args[i] == _
        res_args.push moreargs[j++]
      else
        res_args.push args[i]
      i++
    func.apply null, res_args
```

In CoffeeScript `while` is an expression that returns the value of its inner block, so there is no need for pushing values to a results array.

```
_ = {}
partial10lines = (func, args...) ->
  (moreargs...) ->
    i = j = 0
    func.apply null,
      while i++ < args.length
        if args[i-1] == _
          moreargs[j++]
        else
          args[i-1]
```

A `for` loop instead of the `while` gets rid of the `length` check. A splat can also be used in a call which eliminates the `apply`. The old school `==` can be replaced with `is`.

```
_ = {}
partial8lines = (func, a...) -> (b...) ->
  i = 0
  func (for arg in a
    if arg is _
      b[i++]
    else
      arg)...
```

The low level counter `i` is only used to get the next argument from `b`. The same effect can be achieved by treating `b` as a LIFO (Last In First Out) buffer. To do that `b` has to be reversed.

```
_ = {}
partial5lines = (func, a...) -> (b...) ->
  b.reverse()
  func (for arg in a
    if arg is _ then b.pop() else arg)...
```

Instead of using an empty object as the placeholder, using `undefined` allows the `if` test to be replaced with an [existential assignment](#) `?`.

```
_ = undefined
partial4lines = (func, a...) -> (b...) ->
  b.reverse()
  func (for arg in a then arg ?= b.pop())...
```

Test

A couple of test cases and an example of `partial`. In the interactive HTML you can try substituting the number in `partial4lines` to test the other versions.

```
test = ->
  f = (x, y, z) -> x + 2*y + 5*z
  g = partialFree f, _, 1, _
  show "g 3, 5 => #{g 3, 5} Expected: 30"

  # Modified from an alexkg example
  fold = (f, z, xs) ->
    z = f(z, x) for x in xs
    z
  max = partialFree fold, Math.max, -Infinity, _
  show "max [-10..10] => #{max [-10..10]} Expected: 10"

  # Without free vars
  partial = (f, a...) -> (b...) -> f a..., b...
  min = partial fold, Math.min, Infinity
  show "min [-10..10] => #{min [-10..10]} Expected: -10"
partialFree = partial4lines
test()
```

Output

```
1 g 3, 5 => 30 Expected: 30
2 max [-10..10] => 10 Expected: 10
3 min [-10..10] => -10 Expected: -10
```

JavaScript

```
1 (function() {
2   var draw, drawMove, drawPath, movement, partial10lines, partial12lines, partial15lines, partial4lines, partial5lines, partial8lines;
3   var __slice = Array.prototype.slice;
4
5   draw = function(ctx) {
6     var ix;
7     ctx.beginPath();
8     ctx.strokeStyle = 'gold';
9     drawMove(ctx, (function() {
10      var _results;
11      _results = [];
12      for (ix = 0; ix < 90; ix += 10) {
13        _results.push(ix);
14      }
15      return _results;
16    })());
17     ctx.beginPath();
18     ctx.strokeStyle = 'salmon';
19     return drawPath(ctx, (function() {
20      var _results;
21      _results = [];
22      for (ix = 0; ix < 90; ix += 10) {
23        _results.push(ix);
24      }
25      return _results;
26    })());
27   });
```

```

28
29 movement = function(ctx, ax, ay, cp1x, cp1y, cp2x, cp2y, x, y) {
30     ctx.moveTo(ax, ay);
31     return ctx.bezierCurveTo(cp1x, cp1y, cp2x, cp2y, x, y);
32 };
33
34 drawMove = function(ctx, args) {
35     args.forEach(function(ix) {
36         return movement(ctx, 0, 0, 30, 30, 150 + ix, 50, 110 + ix, 90);
37     });
38     return ctx.stroke();
39 };
40
41 _ = void 0;
42
43 partialFree = function() {
44     var a, func;
45     func = arguments[0], a = 2 <= arguments.length ? __slice.call(arguments, 1) : [];
46     return function() {
47         var arg, b;
48         b = 1 <= arguments.length ? __slice.call(arguments, 0) : [];
49         b.reverse();
50         return func.apply(null, (function() {
51             var _i, _len, _results;
52             _results = [];
53             for (_i = 0, _len = a.length; _i < _len; _i++) {
54                 arg = a[_i];
55                 _results.push(arg != null ? arg : arg = b.pop());
56             }
57             return _results;
58         })());
59     };
60 };
61
62 swirl = partialFree(movement, _, _, 0, 30, 30, _, 50, _, 90);
63
64 drawPath = function(ctx, args) {
65     args.forEach(function(ix) {
66         return swirl(ctx, 200, 150 + ix, 110 + ix);
67     });
68     return ctx.stroke();
69 };
70
71 _ = {};
72
73 partial15lines = function() {
74     var args, func, wrapper;
75     func = arguments[0], args = 2 <= arguments.length ? __slice.call(arguments, 1) : [];
76     return wrapper = function() {
77         var i, j, res_args;
78         i = 0;
79         j = 0;
80         res_args = [];
81         while (i < args.length) {
82             if (args[i] === _) {
83                 res_args.push(arguments[j]);
84                 j++;
85             } else {
86                 res_args.push(args[i]);
87             }
88             i++;
89         }
90         return func.apply(null, res_args);
91     };
92 };
93
94 _ = {};
95
96 partial12lines = function() {
97     var args, func;
98     func = arguments[0], args = 2 <= arguments.length ? __slice.call(arguments, 1) : [];
99     return function() {

```

```

100     var i, j, moreargs, res_args;
101     moreargs = 1 <= arguments.length ? __slice.call(arguments, 0) : [];
102     i = j = 0;
103     res_args = [];
104     while (i < args.length) {
105         if (args[i] === _) {
106             res_args.push(moreargs[j++]);
107         } else {
108             res_args.push(args[i]);
109         }
110         i++;
111     }
112     return func.apply(null, res_args);
113 };
114 };
115
116 _ = {};
117
118 partial10lines = function() {
119     var args, func;
120     func = arguments[0], args = 2 <= arguments.length ? __slice.call(arguments, 1) : [];
121     return function() {
122         var i, j, moreargs;
123         moreargs = 1 <= arguments.length ? __slice.call(arguments, 0) : [];
124         i = j = 0;
125         return func.apply(null, (function() {
126             var _results;
127             _results = [];
128             while (i++ < args.length) {
129                 if (args[i - 1] === _) {
130                     _results.push(moreargs[j++]);
131                 } else {
132                     _results.push(args[i - 1]);
133                 }
134             }
135             return _results;
136         })());
137     };
138 };
139
140 _ = {};
141
142 partial8lines = function() {
143     var a, func;
144     func = arguments[0], a = 2 <= arguments.length ? __slice.call(arguments, 1) : [];
145     return function() {
146         var arg, b, i;
147         b = 1 <= arguments.length ? __slice.call(arguments, 0) : [];
148         i = 0;
149         return func.apply(null, (function() {
150             var _i, _len, _results;
151             _results = [];
152             for (_i = 0, _len = a.length; _i < _len; _i++) {
153                 arg = a[_i];
154                 if (arg === _) {
155                     _results.push(b[i++]);
156                 } else {
157                     _results.push(arg);
158                 }
159             }
160             return _results;
161         })());
162     };
163 };
164
165 _ = {};
166
167 partial5lines = function() {
168     var a, func;
169     func = arguments[0], a = 2 <= arguments.length ? __slice.call(arguments, 1) : [];
170     return function() {
171         var arg, b;

```

```

172     b = 1 <= arguments.length ? __slice.call(arguments, 0) : [];
173     b.reverse();
174     return func.apply(null, (function() {
175         var _i, _len, _results;
176         _results = [];
177         for (_i = 0, _len = a.length; _i < _len; _i++) {
178             arg = a[_i];
179             if (arg === _) {
180                 _results.push(b.pop());
181             } else {
182                 _results.push(arg);
183             }
184         }
185         return _results;
186     })());
187 };
188
189
190 _ = void 0;
191
192 partial4lines = function() {
193     var a, func;
194     func = arguments[0], a = 2 <= arguments.length ? __slice.call(arguments, 1) : [];
195     return function() {
196         var arg, b;
197         b = 1 <= arguments.length ? __slice.call(arguments, 0) : [];
198         b.reverse();
199         return func.apply(null, (function() {
200             var _i, _len, _results;
201             _results = [];
202             for (_i = 0, _len = a.length; _i < _len; _i++) {
203                 arg = a[_i];
204                 _results.push(arg != null ? arg : arg = b.pop());
205             }
206             return _results;
207         })());
208     };
209 };
210
211 test = function() {
212     var f, fold, g, max, min, partial;
213     f = function(x, y, z) {
214         return x + 2 * y + 5 * z;
215     };
216     g = partialFree(f, _, 1, _);
217     show("g 3, 5 => " + (g(3, 5)) + " Expected: 30");
218     fold = function(f, z, xs) {
219         var x, _i, _len;
220         for (_i = 0, _len = xs.length; _i < _len; _i++) {
221             x = xs[_i];
222             z = f(z, x);
223         }
224         return z;
225     };
226     max = partialFree(fold, Math.max, -Infinity, _);
227     show("max [-10..10] => " + (max([-10, -9, -8, -7, -6, -5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10])) + " Expected: 10");
228     partial = function() {
229         var a, f;
230         f = arguments[0], a = 2 <= arguments.length ? __slice.call(arguments, 1) : [];
231         return function() {
232             var b;
233             b = 1 <= arguments.length ? __slice.call(arguments, 0) : [];
234             return f.apply(null, __slice.call(a).concat(__slice.call(b)));
235         };
236     };
237     min = partial(fold, Math.min, Infinity);
238     return show("min [-10..10] => " + (min([-10, -9, -8, -7, -6, -5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10])) + " Expected: -10");
239 };
240
241 partialFree = partial4lines;
242
243 test();

```

```
244
245 }).call(this);
```

Formats [CoffeeScript](#) [Markdown](#) [PDF](#) [HTML](#)

License [Creative Commons Attribution Share Alike](#) by [autotelicum](#) © 2554/2011