

## 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.

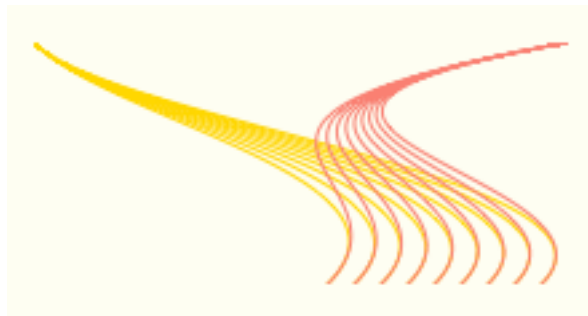
It is not always the case that arguments are so nicely ordered that it is the first ones that need to be held constant. To handle the general case with arbitrary arguments, a special symbol<sup>1</sup> can designate free variables i.e. those arguments that are not fixed.

In the code below<sup>2</sup> 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()
```



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

<sup>2</sup>To 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...) ->
  func (for arg in a then arg ?= b.shift())...

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())...
```

Reversing the `b...` arguments are only required because `pop` returns the last element. Noé Rubinstein joined the *code reduction* fun by noticing that the `Array::shift` function removes and returns the first argument.

```
_ = undefined
partial3lines = (func, a...) -> (b...) ->
  func (for arg in a then arg ?= b.shift())...
```

## Test

A couple of test cases and an example of `partial`. In the interactive HTML you can try substituting the number in `partial3lines` 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 = partial3lines
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, partial3lines, partial4lines, partial5lines;
3   var __slice = Array.prototype.slice;
4
5   show = console.log;
6
7   showDocument = function(doc, width, height) {
8     return show(doc);
9   };
10
11  draw = function(ctx) {
12    var ix;
13    ctx.beginPath();
14    ctx.strokeStyle = 'gold';
15    drawMove(ctx, (function() {
16      var _results;
```

```

17     _results = [];
18     for (ix = 0; ix < 90; ix += 10) {
19         _results.push(ix);
20     }
21     return _results;
22 })();
23 ctx.beginPath();
24 ctx.strokeStyle = 'salmon';
25 return drawPath(ctx, (function() {
26     var _results;
27     _results = [];
28     for (ix = 0; ix < 90; ix += 10) {
29         _results.push(ix);
30     }
31     return _results;
32 })());
33 };
34
35 movement = function(ctx, ax, ay, cp1x, cp1y, cp2x, cp2y, x, y) {
36     ctx.moveTo(ax, ay);
37     return ctx.bezierCurveTo(cp1x, cp1y, cp2x, cp2y, x, y);
38 };
39
40 drawMove = function(ctx, args) {
41     args.forEach(function(ix) {
42         return movement(ctx, 0, 0, 30, 30, 150 + ix, 50, 110 + ix, 90);
43     });
44     return ctx.stroke();
45 };
46
47 _ = void 0;
48
49 partialFree = function() {
50     var a, func;
51     func = arguments[0], a = 2 <= arguments.length ? __slice.call(arguments, 1) : [];
52     return function() {
53         var arg, b;
54         b = 1 <= arguments.length ? __slice.call(arguments, 0) : [];
55         return func.apply(null, (function() {
56             var _i, _len, _results;
57             _results = [];
58             for (_i = 0, _len = a.length; _i < _len; _i++) {
59                 arg = a[_i];
60                 _results.push(arg != null ? arg : arg = b.shift());
61             }
62             return _results;
63         })());
64     };
65 };
66
67 swirl = partialFree(movement, _, _, 0, 30, 30, _, 50, _, 90);
68
69 drawPath = function(ctx, args) {
70     args.forEach(function(ix) {
71         return swirl(ctx, 200, 150 + ix, 110 + ix);
72     });
73     return ctx.stroke();
74 };
75
76 _ = {};
77
78 partial15lines = function() {
79     var args, func, wrapper;
80     func = arguments[0], args = 2 <= arguments.length ? __slice.call(arguments, 1) : [];
81     return wrapper = function() {
82         var i, j, res_args;
83         i = 0;
84         j = 0;
85         res_args = [];
86         while (i < args.length) {
87             if (args[i] === _) {
88                 res_args.push(arguments[j]);

```

```

89         j++;
90     } else {
91         res_args.push(args[i]);
92     }
93     i++;
94 }
95 return func.apply(null, res_args);
96 };
97 };
98
99 _ = {};
100
101 partial12lines = function() {
102     var args, func;
103     func = arguments[0], args = 2 <= arguments.length ? __slice.call(arguments, 1) : [];
104     return function() {
105         var i, j, moreargs, res_args;
106         moreargs = 1 <= arguments.length ? __slice.call(arguments, 0) : [];
107         i = j = 0;
108         res_args = [];
109         while (i < args.length) {
110             if (args[i] === _) {
111                 res_args.push(moreargs[j++]);
112             } else {
113                 res_args.push(args[i]);
114             }
115             i++;
116         }
117         return func.apply(null, res_args);
118     };
119 };
120
121 _ = {};
122
123 partial10lines = function() {
124     var args, func;
125     func = arguments[0], args = 2 <= arguments.length ? __slice.call(arguments, 1) : [];
126     return function() {
127         var i, j, moreargs;
128         moreargs = 1 <= arguments.length ? __slice.call(arguments, 0) : [];
129         i = j = 0;
130         return func.apply(null, (function() {
131             var _results;
132             _results = [];
133             while (i++ < args.length) {
134                 if (args[i - 1] === _) {
135                     _results.push(moreargs[j++]);
136                 } else {
137                     _results.push(args[i - 1]);
138                 }
139             }
140             return _results;
141         })());
142     };
143 };
144
145 _ = {};
146
147 partial8lines = function() {
148     var a, func;
149     func = arguments[0], a = 2 <= arguments.length ? __slice.call(arguments, 1) : [];
150     return function() {
151         var arg, b, i;
152         b = 1 <= arguments.length ? __slice.call(arguments, 0) : [];
153         i = 0;
154         return func.apply(null, (function() {
155             var _i, _len, _results;
156             _results = [];
157             for (_i = 0, _len = a.length; _i < _len; _i++) {
158                 arg = a[_i];
159                 if (arg === _) {
160                     _results.push(b[i++]);
161                 }
162             }
163             return _results;
164         })());
165     };
166 };

```

```

161         } else {
162             _results.push(arg);
163         }
164     }
165     return _results;
166 })();
167 };
168 };
169
170 _ = {};
171
172 partial5lines = function() {
173     var a, func;
174     func = arguments[0], a = 2 <= arguments.length ? __slice.call(arguments, 1) : [];
175     return function() {
176         var arg, b;
177         b = 1 <= arguments.length ? __slice.call(arguments, 0) : [];
178         b.reverse();
179         return func.apply(null, (function() {
180             var _i, _len, _results;
181             _results = [];
182             for (_i = 0, _len = a.length; _i < _len; _i++) {
183                 arg = a[_i];
184                 if (arg === _) {
185                     _results.push(b.pop());
186                 } else {
187                     _results.push(arg);
188                 }
189             }
190             return _results;
191         })());
192     };
193 };
194
195 _ = void 0;
196
197 partial4lines = function() {
198     var a, func;
199     func = arguments[0], a = 2 <= arguments.length ? __slice.call(arguments, 1) : [];
200     return function() {
201         var arg, b;
202         b = 1 <= arguments.length ? __slice.call(arguments, 0) : [];
203         b.reverse();
204         return func.apply(null, (function() {
205             var _i, _len, _results;
206             _results = [];
207             for (_i = 0, _len = a.length; _i < _len; _i++) {
208                 arg = a[_i];
209                 _results.push(arg != null ? arg : arg = b.pop());
210             }
211             return _results;
212         })());
213     };
214 };
215
216 _ = void 0;
217
218 partial3lines = function() {
219     var a, func;
220     func = arguments[0], a = 2 <= arguments.length ? __slice.call(arguments, 1) : [];
221     return function() {
222         var arg, b;
223         b = 1 <= arguments.length ? __slice.call(arguments, 0) : [];
224         return func.apply(null, (function() {
225             var _i, _len, _results;
226             _results = [];
227             for (_i = 0, _len = a.length; _i < _len; _i++) {
228                 arg = a[_i];
229                 _results.push(arg != null ? arg : arg = b.shift());
230             }
231             return _results;
232         })());

```

```

233     };
234 };
235
236 test = function() {
237     var f, fold, g, max, min, partial;
238     f = function(x, y, z) {
239         return x + 2 * y + 5 * z;
240     };
241     g = partialFree(f, _, 1, _);
242     show("g 3, 5 => " + (g(3, 5)) + " Expected: 30");
243     fold = function(f, z, xs) {
244         var x, _i, _len;
245         for (_i = 0, _len = xs.length; _i < _len; _i++) {
246             x = xs[_i];
247             z = f(z, x);
248         }
249         return z;
250     };
251     max = partialFree(fold, Math.max, -Infinity, _);
252     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");
253     partial = function() {
254         var a, f;
255         f = arguments[0], a = 2 <= arguments.length ? __slice.call(arguments, 1) : [];
256         return function() {
257             var b;
258             b = 1 <= arguments.length ? __slice.call(arguments, 0) : [];
259             return f.apply(null, __slice.call(a).concat(__slice.call(b)));
260         };
261     };
262     min = partial(fold, Math.min, Infinity);
263     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");
264 };
265
266 partialFree = partial3lines;
267
268 test();
269
270 }).call(this);

```

---

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

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