Can we assume that the number of inputs will always be valid, i.e. if f takes 10 variables we can assume that the number of valid values passed in will be 10?

I guess a concrete example might in order:

Given

(define . 'MISSING)

These could be used as a test

((peval f a b c d e))
((peval g . . ) a b c)
((peval h a . ) b)
((peval m . b) a)
((peval n))

This would not

((peval p . . .))
((peval q a b c) a)
((peval r . . c) a)
((peval s . ) .)

Subject: Re: Exercise 2
Posted by lusth on Thu, 29 Sep 2016 11:17:36 GMT

The number of arguments to peval will be one plus the number of arguments of the function being partially evaluated. The number of arguments to the resulting function will be the number of missing arguments in the initial call.

Subject: Re: Exercise 2
Posted by jrmelton on Thu, 06 Oct 2016 02:03:02 GMT
The order of the arguments in the resulting function do matter correct?

Subject: Re: Exercise 2
Posted by lusth on Thu, 06 Oct 2016 12:37:29 GMT

Yes, the order matters. The first argument in the second group should take the place of the first missing argument in the first group (left-to-right), and so on.

Subject: Re: Exercise 2
Posted by tmurphy2 on Thu, 06 Oct 2016 22:12:20 GMT

If I construct a list of arguments to pass to a function, how do I allow that function to use the list as its arguments?

Subject: Re: Exercise 2
Posted by lusth on Thu, 06 Oct 2016 23:17:54 GMT

The apply function. These two expressions are equivalent:

(f a b c d)
(aply f (list a b c d))

Subject: Re: Exercise 2
Posted by jrmelton on Fri, 07 Oct 2016 02:56:44 GMT

When constructing the list of arguments to pass to the function, how can I make the arguments distinct? Obviously, I could say that when I come across a missing parameter, just concatenate "a" (or any other letter) to the previous variable name....the variable names would continue to grow such as "a" "aa" "aaa" "aaaa" but this just seems like the wrong way to go about this. Are there any other ways of doing this that I'm just not thinking about?
Both peval and the function it returns are variadic, so the arguments to peval will be collected in the list \(@\). The arguments to the resulting function will also be collected in the list \(@\). The resulting function will merge those two lists and apply the original function to the merged list. You will need to figure out how to keep the inner \(@\) from shadowing the outer \(@\).

Subject: Re: Exercise 2
Posted by apluth on Tue, 25 Oct 2016 21:06:51 GMT
View Forum Message <> Reply to Message

If we used a dot notation for the outer list would \(@\) work for the inner or would it be better to use dot for the inner args but because we defined \(\cdot\) As 'missing would that not work at all

Subject: Re: Exercise 2
Posted by jarobinson3 on Tue, 25 Oct 2016 21:18:24 GMT
View Forum Message <> Reply to Message

I am not really sure what you are asking. In my solution I explicitly used 'MISSING in my function but to pass in the input I did use the defined \(\cdot\) to make that easier.

Subject: Re: Exercise 2
Posted by apluth on Thu, 27 Oct 2016 21:22:54 GMT
View Forum Message <> Reply to Message

I figured it out using \(@\) twice