```
1   from collections import namedtuple
2   from primitives import primitives
3
4   # ----- Expression types -----
5
6   fundef    = namedtuple('fundef',    ('name', 'argnames', 'body'))
7   vardef    = namedtuple('vardef',    ('name', 'exp'))
8   call      = namedtuple('call',      ('funname', 'args'))
9   kyif      = namedtuple('kyif',      ('cond', 'iftrue', 'iffalse'))
10  begin     = namedtuple('begin',     ('exps'))
11  kyfun     = namedtuple('kyfun',     ('argnames', 'body', 'env'))
12  gradfun   = namedtuple('gradfun',   ('fun', 'argnum', 'env'))
13  grad      = namedtuple('grad',      ('funname', 'argnum'))
14  tape      = namedtuple('tape',      ('value', 'env'))
15  primitive = namedtuple('primitive', ('fun', 'grad'))
16
17  # ----- Evaluator -----
18
19  def kyeval(exp, env):
20      if isinstance(exp, str): # variable lookup
21          return env[exp] if exp in env else globalenv[exp]
22      elif isinstance(exp, vardef):
23          env[exp.name] = kyeval(exp.exp, env)
24      elif isinstance(exp, fundef):
25          env[exp.name] = kyfun(exp.argnames, exp.body, env)
26      elif isinstance(exp, grad):
27          return gradfun(env[exp.funname], exp.argnum, env)
28      elif isinstance(exp, kyif):
29          return kyeval(exp.iftrue if kyeval(exp.cond, env) else exp.iffalse, env)
30      elif isinstance(exp, call):
31          return kyapply(kyeval(exp.funname, env), [kyeval(arg, env) for arg in
    exp.args])
32      elif isinstance(exp, begin):
```

```
            return [kyeval(subexp, env) for subexp in exp.exps][-1]
        else:
            return exp

    def kyapply(fun, args):
        localenv = {'outgrad' : kyfun((), 0.0, {})}
        if isinstance(fun, kyfun):
            localenv.update(fun.env)
            localenv.update(zip(fun.argnames, args))
            return kyeval(fun.body, localenv)
        elif isinstance(fun, gradfun):
            args[fun.argnum] = tape(args[fun.argnum], localenv)
            getval(kyapply(fun.fun, args), 1.0, localenv)
            return kyapply(localenv['outgrad'], ())
        elif any([isinstance(arg, tape) for arg in args]):
            argvals = [getval(arg, grad, localenv) for arg, grad in zip(args,
    fun.grads)]localenv.update({'arg_' + str(i) : val for i, val in
    enumerate(argvals)})result'] = kyapply(fun, argvals)
            return tape(localenv['result'], localenv)
        else:
            return fun.fun(*args)

    def getval(arg, grad, localenv):
        if isinstance(arg, tape):
            arg.env['outgrad'] = kyfun((), call('add',
                (call(kyfun((), grad, localenv), ()), call(arg.env['outgrad'],
    ())))), {})return arg.value
        else:
            return arg

    # ----- Parser -----

    def parse(string):
        s_list = string.replace('(', ' ( ').replace(')', ' ), ').split()
        s_list = [s if s in ['(', '),'] else "'" + s + "'," for s in s_list]
        tuples = eval("".join(['("begin", '] + s_list + [')']))
        return kyexp(tuples)

    def kyexp(obj):
        tag = obj[0]
        if isinstance(obj, str):
            return int(obj) if obj.isdigit() else obj
        elif tag == 'def' and isinstance(obj[1], tuple):
            return fundef(obj[1][0], obj[1][1:], begin(map(kyexp, obj[2:])))
        elif tag == 'def':
            return vardef(obj[1], kyexp(obj[2]))
        elif tag == 'grad':
            return grad(obj[1], int(obj[2]))
        elif tag == 'if':
            return kyif(*map(kyexp, obj[1:4]))
        elif tag == 'begin':
            return begin(map(kyexp, obj[1:]))
        else:
```

```
        else:
            return call(tag, tuple(map(kyexp, obj[1:])))

    globalenv = {name : primitive(val[0], [parse(s) for s in val[1]])
                 for name, val in primitives.iteritems()}

    # ----- Python interface -----

    def get_function(string, fun_name, global_vars={}):
        env = global_vars.copy()
        kyeval(parse(string), env)
        return lambda *args : kyapply(env[fun_name], list(args))
```