```
ops = { '+': lambda x,y: x+y,
        '-': lambda x,y: x-y,
        '*': lambda x,y: x*y,
        '/': lambda x,y: x//y }

def make_node(op, name):
  class node:
    def __init__(self, op1, op2):
      self.__operand1 = op1
      self.__operand2 = op2
    def eval(self):
      return op(self.__operand1.eval(), self.__operand2.eval())
    def combine(self):
      if isinstance(self.__operand1, leaf) and
         isinstance(self.__operand2, leaf):
        return leaf(str(self.eval()))
      else:
        self.__operand1 = self.__operand1.combine()
        self.__operand2 = self.__operand2.combine()
        return self
    def __str__(self): return "("+self.__operand1.__str__() +
                               name + self.__operand2.__str__()+")"
  return node

class leaf:
  def __init__(self, value):
    self.__value = value
  def eval(self):
    return int(self.__value)
  def combine(self): return self
  def __str__(self): return self.__value

translator = {op:make_node(fop, op) for op, fop in ops.items()}
translator.update({str(x):leaf(str(x)) for x in range(10)})

class calculator:
  def __init__(self, expr):
    self.__root, dropped = self.__convert(expr, 0)
  def is_value(self): return isinstance(self.__root,leaf)
  def __convert(self, expr, n):
    if n < len(expr):
       if expr[n] in {'+', '*', '-', '/'}:
         op1,n1 = self.__convert(expr,n+1)
         op2,n2 = self.__convert(expr,n1+1)
         return translator[expr[n]](op1,op2),n2
       else: return translator[expr[n]],n
  def combine(self):
    self.__root = self.__root.combine()
    return self
  def eval(self): return self.__root.eval()
  def __str__(self): return self.__root.__str__()

def print_reduction(e):
  print(e)
  if not e.is_value(): print_reduction(e.combine())
```