```python
import numpy as np
import operator as op
from abc import ABCMeta
from core import Node, Setter, getval, zeros_like

class NumericNode(Node):
    __array_priority__ = 100.0 # Ensure precedence of Node's __rmul__ over numpy's __mul__
    __metaclass__ = ABCMeta
    def __add__(self, other):   return op.add(self, other)
    def __radd__(self, other):  return op.add(self, other)
    def __sub__(self, other):   return op.sub(self, other)
    def __rsub__(self, other):  return op.sub(other, self)
    def __mul__(self, other):   return op.mul(self, other)
    def __rmul__(self, other):  return op.mul(other, self)
    def __neg__(self):          return op.neg(self)
    def __pow__(self, power):   return op.pow(self, power)
    def __rpow__(self, power):  return op.pow(power, self)
    def __div__(self, other):   return op.div(self, other)
    def __rdiv__(self, other):  return op.div(other, self)
    def __lt__(self, other):    return getval(self) < getval(other)
    def __gt__(self, other):    return getval(self) > getval(other)

class FloatNode(NumericNode):
    _value_types = [float, np.float64]
    def zeros(self):
        return 0.0

class ArrayNode(NumericNode):
    _value_types = [np.ndarray]
    def zeros(self):
        return np.zeros(self.shape)
    def reshape(self, shape, order=None):
        return np.reshape(self, shape, order=order)
    def ravel(self, order=None):
        return np.ravel(self, order=order)
    def squeeze(self, axis=None):
        return np.squeeze(self, axis=axis)
    @property
    def T(self): return np.transpose(self)
    @property
    def shape(self): return self.value.shape
    @property
    def ndim(self): return self.value.ndim
    @property
    def size(self): return self.value.size

class DictNode(Node):
    _value_types = [dict]
    def zeros(self):
        return {k : zeros_like(v) for k, v in getval(self).iteritems()}

class ListNode(Node):
    _value_types = [list]
    def zeros(self):
        return [zeros_like(v) for v in getval(self)]

    def __len__(self): return len(self.value)

class SetterNode(Node):
    _value_types = [Setter]
    def zeros(self):
        raise Exception("Shouldn't get zeros of setter")

node_types = [FloatNode, ArrayNode, DictNode, ListNode, SetterNode]
type_mappings = {}
for node_type in node_types:
    type_mappings[node_type] = node_type
    for value_type in node_type._value_types:
        type_mappings[value_type] = node_type
```