```
from types import SimpleNamespace
import numpy as np

class Peano(object):

    def __init__(self, *args):

        self.peanos_annahme = SimpleNamespace()
        self.peanos_annahme.kennung = []
        self.peanos_annahme.wert = []

        self.Abmessung = len(args)
        self.definiert = False

        for i in range(self.Abmessung):
            if isinstance(args[i], str):
                self.peanos_annahme.kennung.append(args[i])

    def definieren(self, *args):

        if len(args) == len(self.peanos_annahme.kennung):
            self.definiert = True
            self.peanos_annahme.wert = []

            for i in range(self.Abmessung):
                self.peanos_annahme.wert.append(abs(args[i]))

    def gleich(self):

        if self.definiert:
            return self.peanos_annahme.wert.count(self.auf(0)) == self.Abmessung

    def auf(self, i):

        if self.definiert:
            return self.peanos_annahme.wert[i]

    def s(self, i):

        if self.definiert:
            return self.auf(i) + 1

#Alle diese Funktionen mussen immer wahr sein
    def einzel_aquivalenz_axiom(self):

        if self.definiert and self.Abmessung == 1:
            return self.auf(0) == self.auf(0)

    def doppelt_aquivalenz_axiom(self):
```

```
        if self.definiert and self.Abmessung == 2 and self.auf(0) == self.auf(1):
            return self.auf(1) == self.auf(0)

    def mehrere_aquivalenz_axiom(self):

        if self.definiert and self.Abmessung > 2 and self.gleich():
            return self.auf(0) == self.auf(self.Abmessung)

    def aquivalenter_nachfolger_axiom(self):

        if self.definiert and self.Abmessung == 2 and self.auf(0) == self.auf(1):
            return self.s(1) == self.s(0)

    def aquivalenter_nachfolger_zusatz_axiom(self):

        if self.definiert and self.Abmessung > 2 and
        (self.peanos_annahme.wert.count(self.auf(0)) == self.Abmessung - 1
        or self.peanos_annahme.wert.count(self.auf(1)) == self.Abmessung - 1):

            unique = np.unique(np.array(self.peanos_annahme.wert))
            return unique[0] + unique[1] == unique[1] + unique[0]

    def aquivalenter_nachfolger_multiplikation_axiom(self):

        if self.definiert and self.Abmessung > 2 and
        (self.peanos_annahme.wert.count(self.auf(0)) == self.Abmessung - 1
        or self.peanos_annahme.wert.count(self.auf(1)) == self.Abmessung - 1):

            unique = np.unique(np.array(self.peanos_annahme.wert))
            return unique[0] * unique[1] == unique[1] * unique[0]

    def null_zusatz_axiom(self):

        if self.definiert and self.Abmessung == 1:
            return 0 + self.auf(0) == self.auf(0)

    def doppelt_nachfolger_zusatz_axiom(self):

        if self.definiert and self.Abmessung == 2:
            return self.s(0) + self.auf(1) == ((self.auf(0)) + self.auf(1)) + 1

    def null_multiplikation_axiom(self):

        if self.definiert and self.Abmessung == 1:
            return 0 * self.auf(0) == 0

    def doppelt_nachfolger_multiplikation_axiom(self):
```

```
    if self.definiert and self.Abmessung == 2:
        return self.s(0) * self.auf(1) ==
        ((self.auf(0)) * self.auf(1)) + self.auf(1)

def aquivalenter_peanos_axiom(self):

    if self.definiert and self.Abmessung == 2 and self.s(0) == self.s(1):
        return self.auf(0) == self.auf(1)

def null_nachfolger_axiom(self):

    if self.definiert and self.Abmessung == 1:
        return not(self.s(0) == 0)
```