

Palindromzahlen

Die Ziffernfolge einer Zahl ist die Abfolge ihrer (in unserem Zahlensystem) Dezimalstellen von vorne nach hinten. Somit wird 263 zu [2,6,3].

Die Zahl 362 mit der umgekehrten (gestürzten, reversierten, gespiegelten...) Ziffernfolge nennen wir reversierte Zahl.

Eine Palindromzahl ist nun eine Zahl, die von vorne und rückwärts gelesen gleich ist. Etwa sind 12321, 99, 666, 9090909, 1000000000001 und 7 Palindrome, während 12, 5555155, 100 und 98 es nicht sind. Eine Palindromzahl ist also identisch mit ihrer reversierten Zahl. (palindrome Worte unserer Sprache wären etwa ANNA, RENNER oder RELIEFPFEILER)

Nicht alle Zahlen sind palindrom. Aber vielleicht können wir sie zu solchen machen!
Folgende Vorschrift wird gestellt:

- 1.) Wähle eine beliebige natürliche Zahl.
- 2.) Ist die Zahl palindrom: Dann bist du fertig
- 3.) Bilde die Summe dieser Zahl mit ihrer reversierten.
- 4.) Mach weiter bei 2.)

Beispiel: **195** -> 195+591=786 -> 786+687=1473 -> 1473+3741=5214 -> 5214+4125=**9339** palindrom!

Die Frage ist nun: **wird jede Zahl**, die wir dieser Behandlung unterziehen, irgendwann **zu einer Palindromzahl**?

Programmierung:

Offenbar durchlaufen wir beim Erzeugen dieser Zahlenfolge eine Schleife:

```
solange Zahl nicht palindrom:
    Zahl durch nächste in Folge ersetzen
```

Aber was passiert, wenn eine Zahl nie zum Palindrom werden sollte? Dann liefere das Programm ewig. Nehmen wir besser eine maximale Schrittzahl mit.

```
schritte = 0
solange schritte < max:
    wenn Zahl palindrom: Rückgabe "Palindrom"
    Zahl = nächste nach Vorschrift berechnen
    schritte += 1
Rückgabe 'bisher kein Palindrom'
```

Unsere Funktion benötigt vermutlich also:

- 2 Eingaben zahl und max
- eine Hilfsfunktion istpalindrom zum Testen ob ein Palindrom vorliegt
- eine Funktion next, die die nächste Zahl in der Folge liefert. Dazu addiert sie die Zahl zu ihrer reversierten Zahl.

```
def palfolge(zahl,max=100):
    schritte = 0
    while schritte<max:
        if istpalindrom(zahl): return "Palindrom"
        zahl = next(zahl)
        schritte += 1
    return 'kein Palindrom'
```

Damit handeln wir uns allerdings ein Problem ein, das man beim Nachvollziehen der Vorschrift mit Zettel und Bleistift gar nicht bemerkt: Wir vermischen die Begriffe 'Wert einer Zahl' und 'Zifferndarstellung einer Zahl'!

Das Rechenzeichen + für die Addition bezieht sich auf den Wert der Zahl. Datentyp Integer oder Long. Das Feststellen der Palindromeigenschaft und das Reversieren der Zahl bezieht sich jedoch auf ihre Zifferndarstellung!

	a und b sind String	a und b sind Integer
a + b	long(a)+long(b)	a+b
reversieren	l = list(a) l.reverse() b = "".join(l)	
istpalindrom	siehe reversieren	

Was fällt uns noch auf:

- zum Bestimmen des nächsten Elementes addieren wir die reversierte Zahl, und zum Überprüfen der Palindromeigenschaft vergleichen wir den Zahl mit genau dieser reversierten. Da werden wir die Arbeit nicht doppelt machen. Wir verzichten auf gesonderte Funktionen next und istpalindrom. Stattdessen erledigen wir alles gleich in der Hauptfunktion
- was ist günstiger für unsere 'Zahl' – sollen wir sie als String behandeln und bei der Addition zu Integer oder Long übergehen, oder sie als String einer Ziffernfolge bearbeiten? Beide Varianten sind möglich. Versuchen wir es als einmal als String.
- Nett wäre auch eine Anzeige der berechneten Zahlenfolge.

<pre>def palfolge(zahl,max=100): schritte = 0 while schrite<=max: print "schritte:",zahl schritte += 1 l = list(zahl) l.reverse() rev = "".join(l) if rev==zahl: return 1 zahl = str(long(zahl)+long(rev)) return 0</pre>	<p>Ausgabe</p> <p>String als Liste umdrehen reversierter String</p> <p>ja, palindrom</p> <p>nächstes Element berechnen</p> <p>nein, kein Palindrom</p>
---	--

Aufruf: palfolge("159")

Erweiterungen:

- will man als Eingabe der Funktion eine normale Zahl (also keinen String) haben, so kann man ganz simpel

```
def palfolge(startzahl,max=100):
    zahl=str(startzahl)
```

 schreiben und alles andere bleibt wie gehabt.
- ganz elegant erlaubt man beide Eingabetypen!

```
def palfolge(zahl,max=100):
    if type(zahl)!=type(" "): zahl=str(zahl)
```

 falls der Typ von zahl nicht String ist, dann wandeln wir ihn eben einfach um.
- für systematische Untersuchungen wäre eine Funktion geschickt, die nicht alle Werte der Folge anzeigt, sondern nur ihre Schlussfolgerung zurückgibt. z.B. return schritte bei Erreichen eines Palindroms und return

- schritte bei Nichterreichen. Die negative Zahl signalisiert 'nicht erreicht'.
- Wir lösen die Aufgabe jetzt mit einer while 1: Schleife und zwei if-Abfragen zum Verlassen der Funktion. Sonst bekommen wir bei Nicht-Palindromen nicht die zuletzt berechnete Zahl zurück, sondern die danach.

```
def palfolge(zahl,max=100):

    if type(zahl)!=type(" "): zahl=str(zahl)
    schritte = 0
    while 1:

        schritte += 1
        l = list(zahl)
        l.reverse()
        rev = "".join(l)
        if rev==zahl: return (schritte,zahl)
        if schritte>=max: return (-schritte,zahl)

    zahl = str(long(zahl)+long(rev))
```

Für Könner: Wie sieht die Sachlage eigentlich aus, wenn wir nicht mit Dezimalzahlen arbeiten, sondern beliebige Zahlenstrings zahl in einer Basis basis zulassen! Die Wandlung String->Wert macht die Pythonfunktion int oder long(zahl,basis). Die Wandlung Wert->String müssen wir allerdings selbst programmieren...

Die folgende Lösung der allgemeinen Aufgabe arbeitet zur Abwechslung mit den Zahlenwerten und bestimmt falls nötig den Ziffernstring

```
digits="0123456789ABCDEFGHIJKLMNPOQRSTUVWXYZ" # max Basis 36

## Zahl n in Basis wandeln, Rückgabe ist String

def numstr(n,base):
    s = ""
    while n>0:
        n,d = divmod(n,base)
        s = digits[d]+s
    return s

## Rückgabe (+-ordnung, Ziffernstring,Wert der letzten Zahl)

def PalFolge(zahl,basis=10,max=50):
    if type(zahl)==type(" "): zahl=long(zahl,basis)
    ordnung = 0
    while 1:
        n = numstr(zahl,basis)
        print "%4d : %40s"%(ordnung,n)

        n1 = list(n)
        n1.reverse()
        nrev = "".join(n1)
        nn = long(nrev,basis)

        if n==nrev: return (ordnung,n,nn)
        if ordnung>=max: return (-ordnung,n,nn)

    zahl += nn
    ordnung += 1
```

zahl ist Wert

n ist Ziffernstring

nrev ist Ziffernstring

nn sein Wert

nächstes Folgeelement

```
>>> PalFolge(121,2,16)
  0 :          1111001
  1 :          1100100
  2 :          11011011
(2, '11011011', 219L)

>>> PalFolge(22,2,16)
  0 :          10110
  1 :          100011
  2 :          1010100
  3 :          1101001
  4 :          10110100
  5 :          11100001
  6 :          101101000
  7 :          110010101
  8 :          1011101000
  9 :          1101000101
 10 :          10111010000
 11 :          11000101101
 12 :          101111010000
 13 :          110010001101
 14 :          1011110100000
 15 :          1100001011101
 16 :          10111110100000
(-16, '10111110100000', 381L)
```

Geschichte:

Die Vermutung, alle Zahlen würden einmal zu einem Palindrom werden, wurde bereits um 1930 formuliert. Sie wurde eher als richtig eingeschätzt, obwohl kein Beweis vorlag. Erst 1967 unterzog der Mathematiker Charles Trigg die Zahlen bis 10000 einer genaueren Untersuchung. 1975 wurden 237310 Element der von 196 ausgehenden Folge durch Harry Saal mit Computerhilfe berechnet, wobei kein Palindrom entdeckt wurde. Trigg selbst hielt die Vermutung übrigens für falsch.

Heiko Harborth bewies 1973, dass die Vermutung in allen Zahlenbasen, die reine Zweierpotenzen sind, falsch ist. Für die Basis 10 gibt es bis heute kein Resultat.

Palindrome und Sprache:

einige Beispiele des britischen Spezialisten für Wortspiele, J. A. Lindon. (Beachte die entstehenden Mehrdeutigkeiten und Bedeutungswandel der Worte durch die veränderte Leserichtung. Wie erklärt sich der (gleichzeitig englische und deutsche) Titel des Gedichtes?

- palindrome sentences: reverse the sequence of words
 - You can cage a swallow, can't you, but you can't swallow a cage, can you?
 - Girl, bathing on Bikini, eyeing boy, finds boy eying bikini on bathing girl.
- palindrome poems: reverse the sequence of lines

DOPPELGÄNGER

Entering the lonely house with my wife,
 I saw him for the first time
 Peering furtively from behind a bush -
 Blackness that moved,
 A shape amid the shadows,
 A momentary glimpse of gleaming eyes
 Revealed in the ragged moon.
 A closer look (he seemed to turn) might have
 Put him to flight forever -
 I dared not
 (For reasons that I failed to understand),
 Though I knew I should act at once.

I puzzled over it, hiding alone,
 Watching the woman as she neared the gate.
 He came, and I saw him crouching
 Night after night,
 Night after night
 He came, and I saw him crouching,
 Watching the woman as she neared the gate.

I puzzled over it, hiding alone -
 Though I knew I should act at once,
 For reasons that I failed to understand
 I dared not
 Put him to flight forever.

A closer look (he seemed to turn) might have
 Revealed in the ragged moon
 A momentary glimpse of gleaming eyes
 A shape amid the shadows,
 Blackness that moved.

Peering furtively from behind a bush,
 I saw him, for the first time,
 Entering the lonely house with my wife,