Državna tekma Python mar 2024

Naloge

1. **Kazalca na uri** Napišite funkcijo, katere argument je čas v 12-urni obliki (hh:mm:ss) in vrne manjši kot med urnim in minutnim kazalcem v stopinjah, zaokroženo na tri decimalna mesta.

   Primeri

   ```
   clock("12:00:00") --> 0.0
   clock("12:15:00") --> 82.5
   clock("12:32:44") --> 179.967
   clock("03:33:33") --> 94.525
   ```

   Ogrodje programa:

   ```python
   #!/usr/bin/env python3

   import sys
   def angle(h):
       hour, minute, sec = [int(x) for x in h.split(':')]
       hour_angle = 30 * hour + minute * 0.5 + sec * 0.0083333
       minute_angle = minute * 6 + 0.1 * sec
       angle = (hour_angle - minute_angle) % 360
       if angle > 180:
           angle = 360 - angle
       return angle

   if __name__ == '__main__':
       for line in sys.stdin:
           target = line.rstrip()
           print(round(angle(target),3))
   ```

2. **Nogavice**

   Koliko parov nogavic lahko izberemo iz danega nabora nogavic različnih barv, če predpostavimo, da par tvorita nogavici enakih barv.

   Podan je seznam celih števil, ki predstavlja nogavice različnih barv. Določite, koliko parov enakih števil lahko izberemo. Na primer, med sedmimi nogavicami z barvami $['1', '2', '1', '2', '1', '3', '2']$ obstaja en par barve $'1'$ in en par barve $'2'$. Ostale tri nogavice $['1', '2', '3']$ nimajo para. Število parov je 2.

   Napišite funkcijo `socks()`, ki vrne število ujemajočih se parov nogavic, ki so na voljo.

   Seznami nogavic so predstavljeni z nizom številk, ločenih s presledkom.

```python
#!/usr/bin/env python3

import sys

def sock(l):
    return sum([l.count(x) // 2 for x in set(l)])

if __name__ == '__main__':
   for line in sys.stdin:
        target = line.rstrip().split()
        print(sock(target))
```

3. **Palindromski anagrami**

   Napišite funkcijo, ki vrne, ali je mogoče ustvariti palindrom s prerazporeditvijo črk v dani besedi.

   Primeri

```
is_palindrome_possible("rearcac") -->  True
# Beseda  "racecar"

is_palindrome_possible("suhbeusheff") -->  True
# Beseda "sfuehbheufs"

is_palindrome_possible("palindrom") -->  False
# To ni mogoče

is_palindrome_possible("atakidnissindikata") --> True
# Je že palidrom
```

```python
#!/usr/bin/env python3

import sys

def is_palindrome_possible(s):
    s = s.lower()
    m = [s.count(x) % 2 for x in set(s)]
    return m.count(1) < 2

if __name__ == '__main__':
    for line in sys.stdin:
        target = line.rstrip()
        print(is_palindrome_possible(target))
```

4. **Koda TAP**

Koda `TAP` ali koda s trkanjem je način sporočanja s trkanjem črk sporočila. Ta način sporočanja so uporabljali zaporniki, ki so si izmenjevali sporočila tako, da so trkali po steni celice.

Črke so razporejene v kvadrat 5×5. Vsaka črka je podana z dvema zaporedjema trkov, ki sta ločeni z dvojnim premorom. V našem primeru trke označimo s piko in premor s presledkom. Prvo zaporedje vsebuje toliko pik, kot je številka vrstice v tabeli, kjer se nahaja črka. Drugo zaporedje pik pa pove, v katerem stolpcu je črka.

Kodiramo posamezne besede, ki so izpisane z velikimi tiskanimi črkami. Male tiskane črke pretvorimo v velike tiskane.

Napišite funkcijo `tap(s)`, ki vrne zakodirano besedo, če je ta izpisana s črkami. Besede, izpisane s pikami, izpiše v dekodirani obliki z velikimi tiskanimi črkami.

```
  1 2 3 4 5
1 A B C Č D          . .      . ..      . ...      . ....      . .....
2 E F G H I          .. .     .. ..     .. ...     .. ....     .. .....
3 J K L M N          ... .    ... ..    ... ...    ... ....    ... .....
4 O P R S Š          .... .   .... ..   .... ...   .... ....   .... .....
5 T U V Z Ž          ..... .  ..... ..  ..... ...  ..... ....  ..... .....
```

Primeri:

```
tap("naprej")   "... ..... . . .... .. .... ... .. . ... ."
tap("... ..... . . ..... .... . . ... .")   "NAZAJ"
```

```python
#!/usr/bin/env python3

import sys

M = [['A', 'B', 'C', 'Č', 'D'],
     ['E', 'F', 'G', 'H', 'I'],
     ['J', 'K', 'L', 'M', 'N'],
     ['O', 'P', 'R', 'S', 'Š'],
     ['T', 'U', 'V', 'Z', 'Ž']]


def make_code(M):
    code_dic = {}
    for i in range(len(M)):
        for j in range(len(M[0])):
            code_dic[M[i][j]] = (i + 1) * '.' + ' ' + (j + 1) * '.'
    return code_dic

chifrant = make_code(M)
dechifrant = dict(zip(chifrant.values(), chifrant.keys()))
```

```
def tap_code(s: str) -> str:
    res = ''
    for ch in s:
        res += (chifrant[ch] + ' ')
    return res[:-1]

def tap_decode(s: str) -> str:
    ss = s.split()
    ss2 = [ss[i] + ' ' + ss[i + 1] for i in range(0, len(ss) - 1, 2)]
    res = ''
    for x in ss2:
        res += dechifrant[x]
    return res

def tap(s:str) -> str:
    if '.' in target:
        return tap_decode(s)
    else:
        return tap_code(s)

if __name__ == '__main__':
    for line in sys.stdin:
        target = line.rstrip()
        print(tap(target))
```

5. **Kontrolna vsota**

Preverjamo točnost zapisa identifikacijske številke s pomočjo formule kontrolne vsote.

Formula kontrolne vsote:

Vzemimo številko: 3 - 7 - 5 - 6 - 2 - 1 - 9 - 8 - 6 - 7 - X, kjer je X kontrolna številka.
Postopek se začne od zadaj, pri kontrolni številki X.

a) Številska vrednost vsake druge številke se podvoji. Nova številka bo: 3 - 14 - 5 - 12 - 2 - 2 - 9 - 16 - 6 - 14 - X.

b) Če postane vrednost večja od 9, se odšteje 9. Številka postane: 3 - 5 - 5 - 3 - 2 - 2 - 9 - 7 - 6 - 5 - X.

c) Izračuna se vsota številskih vrednosti števk, 47 + X.

d) Nekontrolni del se pomnoži z 9, 47 * 9 = 423.

e) Števka enote v rezultatu množenja je kontrolna številka, X = 3.

f) Veljavna številka bi bila potemtakem 37562198673.

Napišite funkcijo `validate`, ki preveri pravilnost zapisa številke. Če je kontrolna številka enaka '0', namesto nje zapiše številko, ki ustreza kontrolni vsoti, in vrne tako popravljeno številko. Sicer pa preveri, ali je kontrolna številka pravilno zapisana. V tem primeru izpiše `True` oziroma `False`.

```
#!/usr/bin/env python3

import sys

def validate(ns):
    n = [int(x) for x in ns[::-1]]
    for i in range(len(n)):
        if i % 2:
            x = 2 * n[i]
            if x > 9:
                x -= 9
            n[i] = x
    return sum(n * 9) % 10

def validate_(ns):
    c = validate(ns)
    if ns[-1] == '0':
        ns = ns[:-1] + str(c)
        return ns
    else:
        return c == 0

if __name__ == '__main__':
    for line in sys.stdin:
        line = line.rstrip()
        print(validate_(line))
```

6. **Kako do cilja**

Začenši s 3 ali 5 in z uporabo operacij:

a) prištejemo 5 oziroma

b) množimo s 3,

poskušamo doseči ciljno vrednost n.

Napišite funkcijo z imenom `is_possible(n)`, ki vrne `True`, če je ciljno vrednost mogoče doseči, oziroma `False`, če ciljne vrednosti ni mogoče doseči.

Primeri:

```
is_possible(14) -> True
# 14 = 3*3 + 5

is_possible(25) -> True
# 25 = 5+5+5+5+5

is_possible(7)  -> False
# Števila 7 ni mogoče doseči.
```

```python
#!/usr/bin/env python3
import sys

def can_reach_target(start, target):
    if start == target:
        return True
    elif start > target:
        return False
    else:
        return can_reach_target(start + 5, target) or
        can_reach_target(start * 3, target)

def only_5_and_3(target: int) -> bool:
    return can_reach_target(3, target) or can_reach_target(5, target)

if __name__ == '__main__':
    for line in sys.stdin:
        target = int(line.rstrip())
        print(only_5_and_3(target))
```