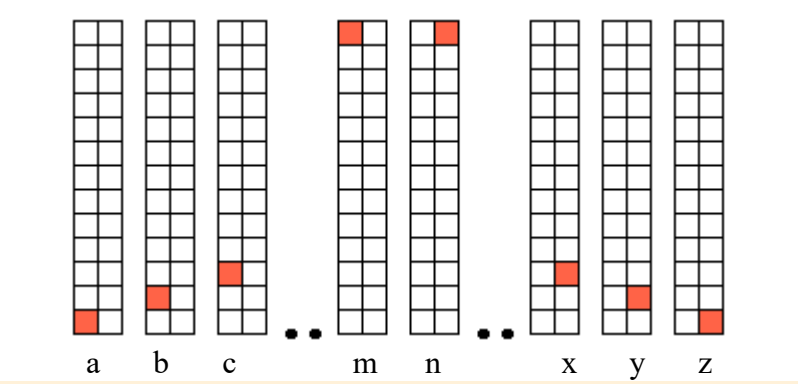
Zadanie Kwadratowy szyfr – LOGIA 20 (2019/20), etap 2

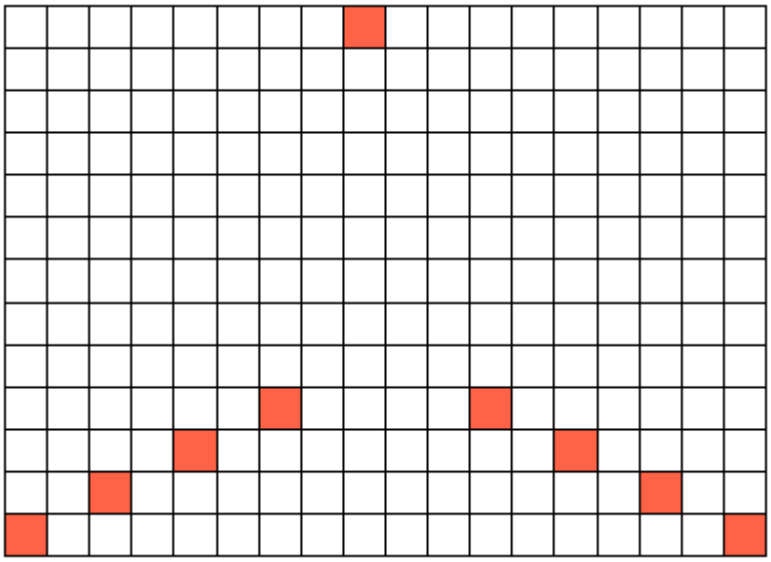
Treść zadania

Kolejnym literom alfabetu łacińskiego (jest ich 26) odpowiadają rysunki przedstawione poniżej:

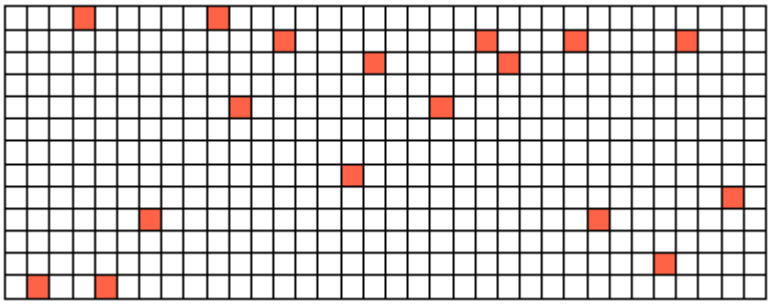


Napisz funkcję **koduj(napis)**, której parametrem jest napis składający się z małych liter alfabetu łacińskiego o długości od **9** do **40** liter. Po jej wywołaniu na środku ekranu powstanie rysunek zakodowanego napisu. Szerokość rysunku wynosi **760**.

Przykłady:



koduj("abcdmwxyz")



koduj("znadnilukrokodyle")

Omówienie rozwiązania

Każda litera jest zaszyfrowana 26 kwadratami, z których jeden jest zamalowany. Do narysowania takich kwadratów potrzebna jest znajomość alfabetu łacińskiego, który składa się z 26 liter. Warto go wygenerować z pomocą kodów ASCII, żeby nie pominąć żadnej litery oraz nie pomylić kolejności liter.

Można to zrobić wstawiając do tablicy kolejne litery począwszy od kodu ASCII 97, który odpowiada litrze "a".

```
1.  tab = []
2.  for i in range(26):
3.      tab += chr(i + 97)
```

Rysowanie zakodowanej litery, to narysowanie dwóch kolumn kwadratów, po 13 kwadratów w każdej z nich. Jeden z kwadratów jest zamalowany kolorem czerwonym. Dla litery "a" jest to pierwszy dolny kwadrat lewej kolumny, dla litery "b" drugi itd. Ostatnią literą z zamalowanym kwadratem w pierwszej kolumnie jest "m". Kolejne litery od "n" do "z" zamalowane są w drugiej kolumnie kwadratów.

```
1.  def rysuj(litera, bok):
2.      tab = alfabet()
3.      for i in range(26):
4.          if litera != tab[i]:
5.              kwadrat(bok)
6.          else:
7.              kwadrat_cz(bok)
8.          fd(bok)
9.          if i == 12:
10.             rt(90); fd(2 * bok); rt(90)
```

Funkcja rysująca zakodowaną literę była najtrudniejszą częścią zadania. Samo kodowanie danego napisu jest proste, wystarczy przeglądać napis dany jako parametr znak po znaku i rysować kwadraty odpowiadające odpowiedniej literze.

Przy tworzeniu rysunku należy uwzględnić skalowanie. Ponieważ szerokość rysunku wynosi 760, można obliczyć długość boku kwadratu:

```
1.  bok = 760 / (2 * len(napis))
```

Rysunek musi być także na środku ekranu, dlatego na początku funkcji głównej należy żółwia przenieść w odpowiednie miejsce. Zakodowany wyraz jest prostokątem o szerokości 760 i wysokości równej trzynastu długościom boku pojedynczego kwadratu.

```
def koduj(napis):
    bok = 760 / (2 * len(napis))
    pu(); bk(760 / 2); lt(90); bk(13 * bok / 2); pd()
    for zn in napis:
        rysuj(zn, bok)
        lt(180)
```

Rozwiązanie w języku Python

```
from turtle import *

def alfabet():
    tab = []
    for i in range(26):
        tab += chr(i + 97)
    return tab

def kwadrat(bok):
    for i in range(4):
        fd(bok); rt(90)

def kwadrat_cz(bok):
    fillcolor("red")
    begin_fill()
    kwadrat(bok)
    end_fill()

def rysuj(litera, bok):
    tab = alfabet()
    for i in range(26):
        if litera != tab[i]:
            kwadrat(bok)
        else:
            kwadrat_cz(bok)
        fd(bok)
        if i == 12:
            rt(90); fd(2 * bok); rt(90)

def koduj(napis):
    bok = 760 / (2 * len(napis))
    pu(); bk(760 / 2); lt(90); bk(13 * bok / 2); pd()
    for zn in napis:
        rysuj(zn, bok)
        lt(180)
```

Testy

Podczas samodzielnego rozwiązywania zadania należy sprawdzać program nie tylko dla przykładów podanych w treści zadania, ale testować rozwiązania dokładnie.

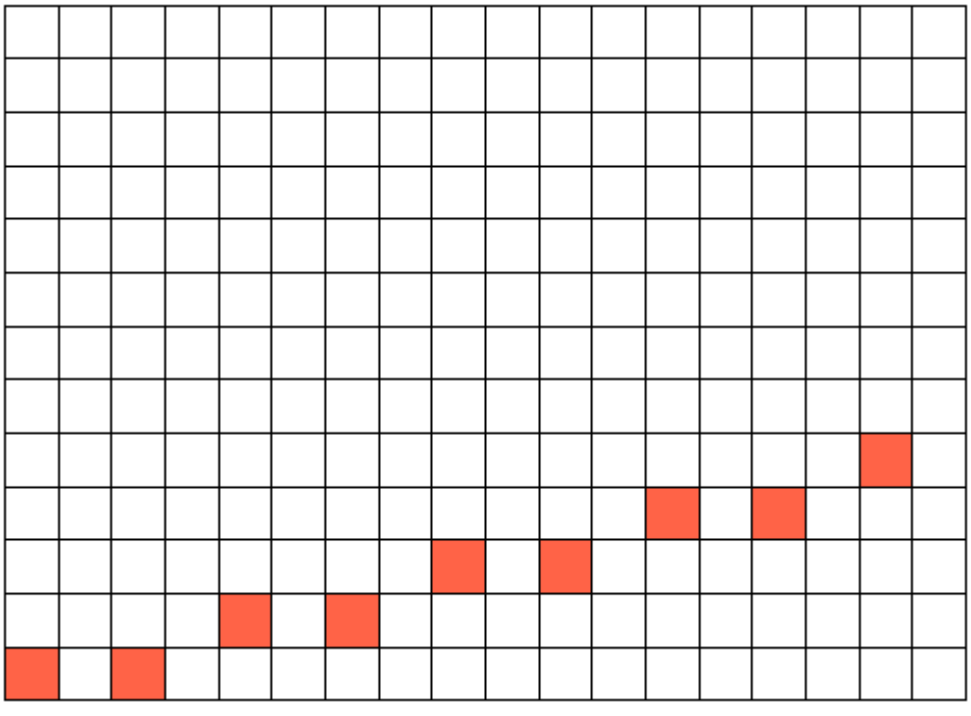
Warto sprawdzić poprawność rysowania wszystkich 26 liter alfabetu. W tym celu można uruchomić dwa niezależne testy dla pierwszych 13-tu liter alfabetu i 13-tu ostatnich.

```
1. koduj("abcdefghijklm")
2. koduj("nopqrstuvwxyz")
```

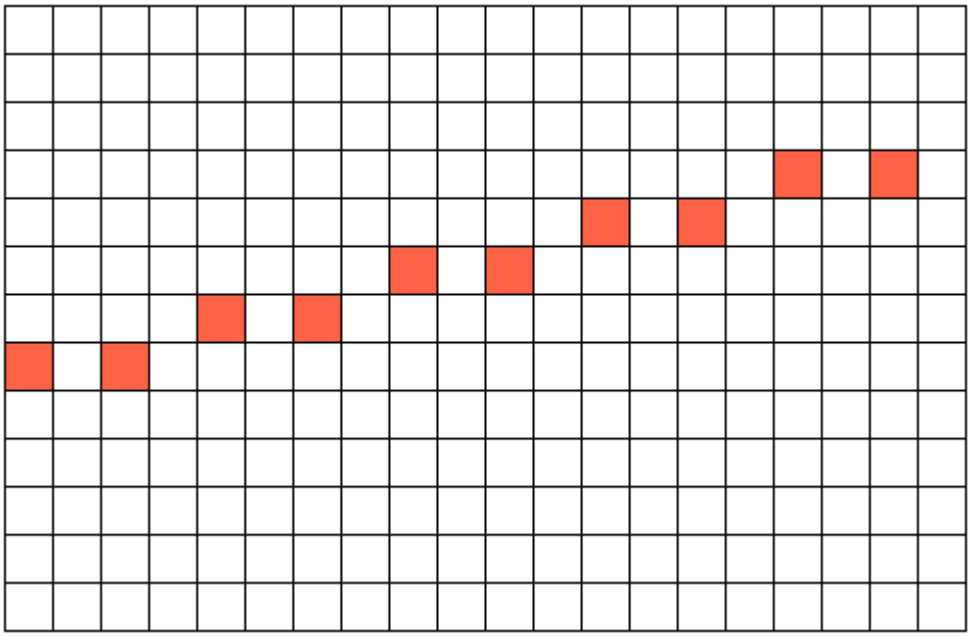
Następnie sprawdzamy wyśrodkowanie rysunku.

```
1. tracer(0)
2. koduj("abcdefghj")
3. pu(); home();pd()
4. update()
```

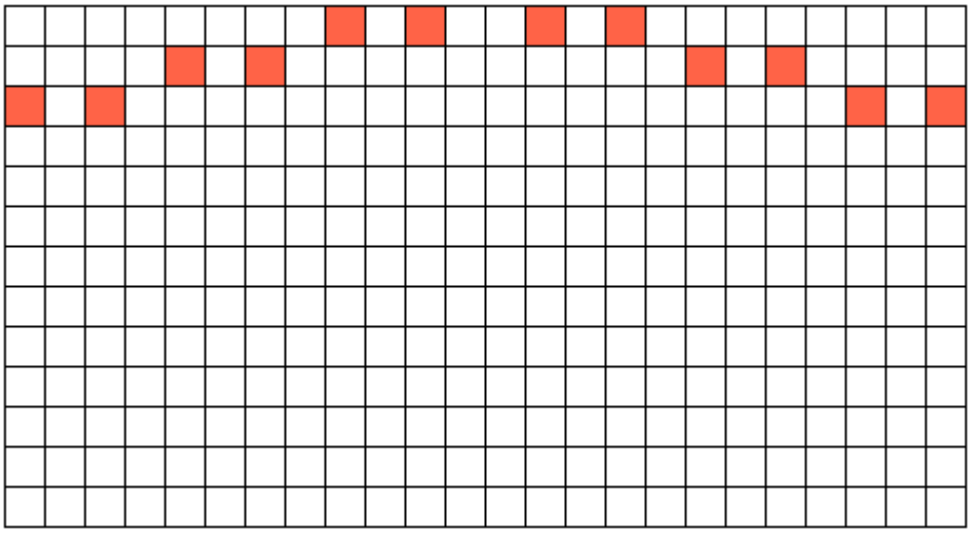
Po zakończeniu rysowania żółw znajdzie się w środku ekranu, który powinien pokrywać się ze środkiem rysunku. Takie testowanie warto przeprowadzić dla kilku wartości parametru, na przykład dla słowa długości 9 (minimalna długość według treści zadania), dla słowa o parzystej i nieparzystej długości oraz dla słowa o długości 40 (maksymalna długość).



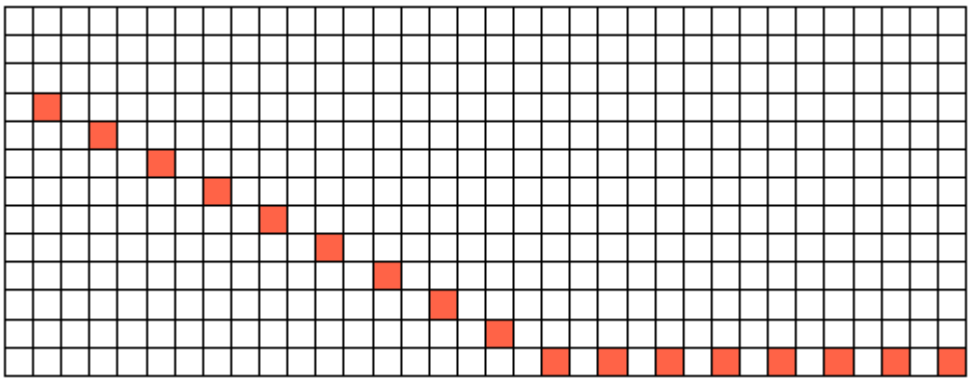
`koduj("aabbccdde")`



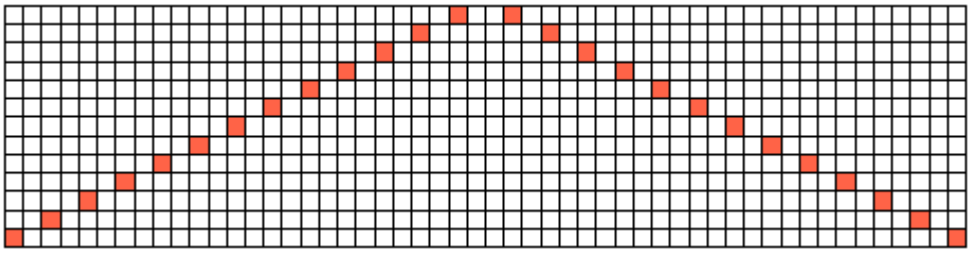
```
koduj("ffgghhiijj")
```



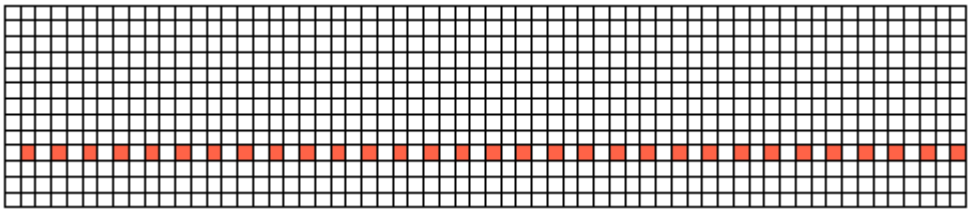
```
koduj("kkllmmnnoopp")
```



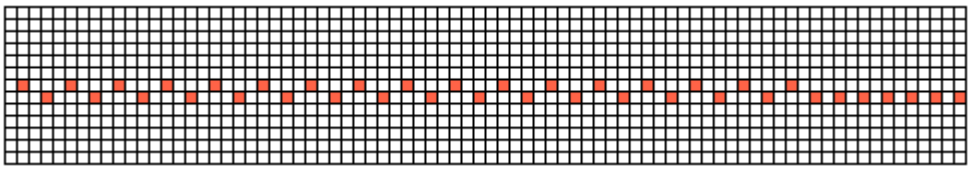
```
koduj("qrstuvwxyzzzzzzzzz")
```



```
koduj("abcdefghijklmnopqrstuvwxyz")
```



```
koduj("wwwwwwwwwwwwwwwwwwwwwwwwwwwwww")
```



```
koduj("tutututututututututututututututututuuuuuuu")
```

Zadanie Liczby – LOGIA 20 (2019/20), etap 2

Treść zadania

Basia szuka liczb, których suma cyfr jest równa iloczynowi tych cyfr np. 123. Napisz funkcję **ile(lista)**, której parametrem jest niepusta lista dodatnich liczb naturalnych, a wynikiem liczba znalezionych przez Basię liczb z podanej listy. Długość listy jest nie większa niż 1 000, a każda z liczb jest nie większa niż 1 000 000.

Przykłady:

Wynikiem **ile([7, 13, 1122, 111, 52, 52111])** jest **2**, ponieważ są dwie takie liczby 7 i 52111.

Wynikiem **ile([11, 1000, 123])** jest **1**, ponieważ jest jedna taka liczba 123.

Wynikiem **ile([321, 152, 2141, 4211])** jest **3**, ponieważ są trzy takie liczby 321, 2141, 4211.

Omówienie rozwiązania

Ustawiamy wartość licznika na 0. Następnie przeglądamy kolejne liczby z listy i dla każdej z nich liczymy sumę cyfr i ich iloczyn. Gdy wyliczone wartości są równe, to zwiększamy licznik. Wynikiem jest wartość licznika.

Obliczanie sumy i iloczynu cyfr liczby można zrealizować przez wyodrębnianie cyfr od prawej strony, czyli najmniej znaczącej cyfry. Ostatnia cyfra liczby to wyniki reszty z dzielenia (`%`) liczby przez 10, a liczba bez ostatniej cyfry to wynik dzielenia całkowitego (`//`) liczby przez 10. Wobec tego sumę cyfr danej liczby można policzyć następująco:

1. Przypisz zmiennej `pom` wartość 0.
2. Dopóki `liczba` jest większa od 0 wykonuj:
   - Wartość zmiennej `pom` zwiększ o `liczba % 10`
   - Zmiennej `liczba` przypisz wartość `liczba // 10`
3. Wynikiem jest wartość zmiennej `pom`.

Analogicznie postępujemy przy obliczaniu iloczynu, z tym że wartością początkową zmiennej `pom` jest 1. Wartość zmiennej `pom` mnożymy w pętli przez wyodrębnioną cyfrę.

1. Przypisz zmiennej `pom` wartość 1.
2. Dopóki `liczba` jest większa od `0` wykonuj:
   - Pomnóż wartość zmiennej `pom` przez `liczba % 10`
   - Zmiennej `liczba` przypisz wartość `liczba // 10`
3. Wynikiem jest wartość zmiennej `pom`.

OEiiZK
Ośrodek Edukacji Informatycznej
i Zastosowań Komputerów w Warszawie

Rozwiązanie w języku Python

Rozwiązanie składa się z dwóch funkcji pomocniczych `suma(liczba)` i `iloczyn(liczba)` oraz głównej funkcji podanej w treści zadania `ile(lista)`.

```
1.  def suma(liczba):
2.      pom = 0
3.      while liczba > 0:
4.          pom = pom + liczba % 10
5.          liczba = liczba // 10
6.      return pom
7.
8.  def iloczyn(liczba):
9.      pom = 1
10.     while liczba > 0:
11.         pom = pom * (liczba % 10)
12.         liczba = liczba // 10
13.     return pom
14.
15. def ile(lista):
16.     licznik = 0
17.     for x in lista:
18.         if suma(x) == iloczyn(x):
19.             licznik += 1
20.     return licznik
```

Można też jednocześnie liczyć sumę i iloczyn. Wtedy rozwiązanie przedstawia się następująco:

```
1.  def czy_rowne(liczba):
2.      suma = 0
3.      iloczyn = 1
4.      while liczba > 0:
5.          x = liczba % 10
6.          suma += x
7.          iloczyn *= x
8.          liczba = liczba // 10
9.      return suma == iloczyn
10.
11. def ile(lista):
12.     licznik = 0
13.     for x in lista:
14.         if czy_rowne(x):
15.             licznik += 1
16.     return licznik
```

Testy

Wywołujemy funkcję **ile** dla różnych testów. W testach powinniśmy uwzględnić sytuację, gdy na liście znajdują się liczby szukane przez Basię oraz gdy nie ma takich liczb.

ile([512, 4121, 1234, 1124, 36, 45, 123, 4121])
4

ile([237, 1412, 2141, 1142, 213, 456, 1421, 7890, 123, 2114, 22, 4121, 312, 9, 4112, 7, 1214, 8, 6790, 231, 1241, 2411, 321, 1124, 132])
21

ile([9385, 111126, 21122, 446, 31311, 4823, 7907, 3124, 31131, 25111, 51112, 111162, 21151, 6857, 31113, 33111, 51121, 21115, 5689, 51211, 52111, 8347, 15211, 3668, 3906])
15

ile([1, 2, 3, 4, 5, 6, 7, 8, 9, 22, 123, 132, 213, 231, 312, 321, 1124, 1142, 1214, 1241, 1412, 1421, 2114, 2141, 2411, 4112, 4121, 4211, 11125, 11133, 11152, 11215, 11222, 11251, 11313, 11331, 11512, 11521, 12115, 12122, 12151, 12212, 12221, 12511, 13113, 13131, 13311, 15112, 15121, 15211, 21115, 21122, 21151, 21212, 21221, 21511, 22112, 22121, 2456, 22211, 25111, 31113, 31131, 31311, 33111, 51112, 51121, 51211, 52111, 111126, 111162, 111216, 111261, 111612, 111621, 112116, 135, 112161, 112611, 116112, 116121, 116211, 121116, 121161, 121611, 126111, 161112, 161121, 161211, 162111, 211116, 211161, 211611, 216111, 261111, 611112, 611121, 611211, 612111, 23578, 621111])
98

oraz
ile([i for i in range(10**4 - 1000, 10**4)])
0

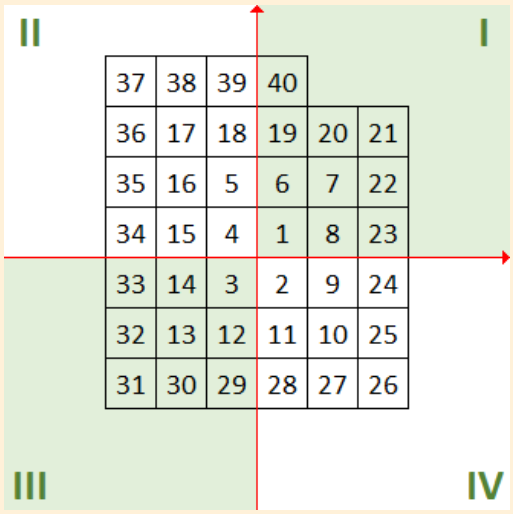
ile([i for i in range(10**4 - 1000, 10**4)] + [1])
1

Do generowania dwóch ostatnich testów zostały wykorzystane listy składane. Zwiększa to czytelność, a przede wszystkim upraszcza zapis.

Zadanie Klocki –LOGIA 20 (2019/20), etap 2

Treść zadania

Tola układa z kwadratowych klocków spiralę na planszy podzielonej na cztery ćwiartki jak na rysunku obok. Pierwszy klocek umieszcza w pierwszej ćwiartce. Kolejne klocki układa spiralnie zgodnie z ruchem wskazówek zegara tak, jak na rysunku pomocniczym. Następnie próbuje policzyć, w której ćwiartce umieści ostatni klocek. Pomóż Toli rozwiązać problem.



Napisz funkcję **klocki(n)**, której parametrem jest liczba klocków nie mniejsza niż **1** i nie większa niż **10 000**. Wynikiem funkcji jest numer ćwiartki układu współrzędnych zapisany liczbą rzymską, w której Tola umieści ostatni klocek.

Przykłady:

Wynikiem **klocki(2)** jest **'IV'**.

Wynikiem **klocki(17)** jest **'II'**.

Wynikiem **klocki(40)** jest **'I'**.

Omówienie rozwiązania

Zadanie Klocki można potraktować jako problem symulacyjny.

Przeanalizujmy sposób układania klocków dla początkowych wartości **n**.

numer ćwiartki	numer ostatniego klocka umieszczonego w danej ćwiartce		
	I cykl	II cykl	III cykl
I	**1**	5 + 3 = **8**	18 + **5** = **23**
IV	1 + 1 = **2**	8 + 3 = **11**	23 + 5 = **28**
III	2 + **1** = **3**	11 + 3 = **14**	28 + 5 = **33**
II	3 + **2** = **5**	14 + **4** = **18**	33 + 6 = **39**

Z powyższej analizy wynikają następujące wnioski:

- z każdym kolejnym cyklem rośnie liczba klocków dokładanych do danej ćwiartki,
- w ćwiartce II liczba dodanych klocków jest o jeden większa od liczby klocków, które zostały dodane w poprzedniej ćwiartce (III) – wyróżnienie kolorem czerwonym w tabeli (na przykład I cykl),
- podobna reguła dotyczy ćwiartki I – wyróżnienie kolorem niebieskim w tabeli (na przykład przejście z II do III cyklu).

Na podstawie obserwacji możemy ustalić schemat postępowania umożliwiający nam wyznaczenie numeru ćwiartki, w której znajdzie się dany klocek:

```
numer = 0
ile_d = 0
dopóki numer < n
    jeśli cwiartka == "I" lub cwiartka == "II"
          ile_d ← ile_d + 1
    numer = numer + ile_d
    jeśli numer >= n
          wynik ćwiartka
```

W pojedynczym obrocie pętli ustalamy numer ostatniego klocka umieszczonego w danej ćwiartce. Jeśli numer ten jest większy lub równy liczbie klocków do ułożenia, to znaleźliśmy numer właściwej ćwiartki.

Pozostaje nam zastanowić się nad tym, jakie powinny być początkowe wartości zmiennych oraz jak w prosty sposób cyklicznie zmieniać numery ćwiartek.

Drugą kwestię rozwiążemy wprowadzając następującą numerację ćwiartek:

0 – I

1 – IV

2 – III

3 – II

Numeracja 0-3 odzwierciedla kolejność przechodzenia ćwiartek. Natomiast przejście z ostatniej ćwiartki (oznaczonej 3) do pierwszej ćwiartki (oznaczonej 0) zrealizujemy operacją modulo:

```
cwiartka = (cwiartka + 1) % 4
```

Pełny zapis algorytmu wraz z inicjalizacją zmiennych może wyglądać tak:

```
numer = 0
cwiartka = 3
ile_d = 0
dopóki numer < n
    cwiartka = (cwiartka + 1) % 4
    jeśli cwiartka == "I" lub cwiartka == "II"
          ile_d ← ile_d + 1
    numer = numer + ile_d
    jeśli numer >= n
          wynik ktora(cwiartka)
```

Funkcja **ktora** z parametrem **cwiartka** odtwarza pierwotne oznaczenie ćwiartek.

Rozwiązanie w języku Python

Funkcja realizująca opisany algorytm, zapisana w języku Python może wyglądać tak:

```
1.  def klocki(n):
2.      numer = 0
3.      cwiartka = 3
4.      ile = 0
5.      while numer < n:
6.          cwiartka = (cwiartka + 1) % 4
7.          if cwiartka == 0 or cwiartka == 3:
8.              ile = ile + 1
9.          numer = numer + ile
10.         if numer >= n:
11.             return ktora(cwiartka)
```

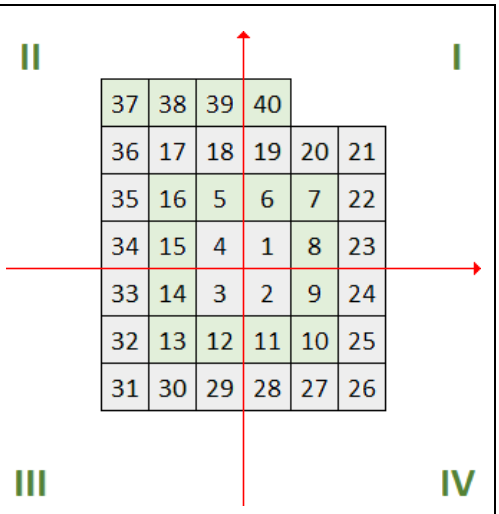
Zamianę numeru ćwiartki z notacji 0-3 (zgodnej z ruchem wskazówek zegara) na oznaczenie ćwiartek z treści zadania (notacja matematyczna) realizuje poniższa funkcja:

```
12. def ktora(cwiartka):
13.     if cwiartka == 0: return 'I'
14.     if cwiartka == 1: return 'IV'
15.     if cwiartka == 2: return 'III'
16.     if cwiartka == 3: return 'II'
```

Rozwiązanie alternatywne

Istnieje rozwiązanie wyznaczające odpowiedź bez użycia iteracji. W rozwiązaniu tym wyznaczany jest numer warstwy, w której leży rozpatrywany klocek:

numery klocków	numer warstwy
1 – 4	0
5 – 16	1
17 – 36	2
37 – 64	3



Numer warstwy dla klocka o numerze **n** obliczymy ze wzoru:

```
int(sqrt((n - 1) // 4))
```

Na podstawie przynależności klocka do warstwy ustalana jest jego pozycja (położenie w danej ćwiartce). Należy przy tym zauważyć, że:

- w warstwach poniżej rozpatrywanej leży `4 * w * w` klocków;
- w każdej ćwiartce w danej warstwie leży `2 * w + 1` klocków.

Rozwiązanie w języku Python

```
1.  from math import sqrt
2.
3.  def klocki(n):
4.      w = int(sqrt((n - 1) // 4))  #warstwa
5.      p = n - 4 * w * w - w        #pozycja
6.      if p <= 0:
7.          return 'II'
8.      p = p - (2 * w + 1)
9.      if p <= 0:
10.         return 'I'
11.     p = p - (2 * w + 1)
12.     if p <= 0:
13.         return 'IV'
14.     p = p - (2 * w + 1)
15.     if p <= 0:
16.         return 'III'
17.     return 'II'
```

Testy

Pierwsza grupa testów sprawdza poprawność dla wartości z rysunku pomocniczego. Wartości te leżą na przekątnych ćwiartek. Druga i trzecia grupa testów została tak dobrana, by sprawdzić poprawność przekraczania ćwiartek (druga grupa dla liczb niewiele przekraczających wartości z rysunku pomocniczego, trzecia grupa dla liczb znacznie je przekraczających). Czwarta i piąta grupa testów miała na celu wyeliminowanie rozwiązań typu jeżeli … to …

Grupa testów	**Test**	**Wynik**
I	klocki(7)	I
	klocki(10)	IV
	klocki(13)	III
	klocki(16)	II
II	klocki(46)	I
	klocki(47)	IV
III	klocki(333)	II
	klocki(334)	I
IV	klocki(9019)	IV
	klocki(9230)	II
V	klocki(9891)	III
	klocki(9999)	II

Zadanie Palindromy – LOGIA (2019/2020), etap 2

Treść zadania

Palindrom to napis, który czytany od lewej do prawej i od prawej do lewej jest identyczny, na przykład: *oko*, *kajak*, *abba*. Napis możemy rozszerzyć poprzez dodawanie liter na początku albo na końcu. Nie można dodawać liter w środku ani jednocześnie z obu stron.

Napisz funkcję **roz(napis)**, której parametrem jest niepusty napis złożony z małych liter alfabetu łacińskiego o długości nie większej niż 1 000. Wynikiem jest napis rozszerzony o minimalną liczbę znaków tak, aby powstał palindrom. Jeśli takich napisów jest więcej niż 1, wynikiem funkcji powinien być napis pierwszy w porządku alfabetycznym. Przy czym rozszerzenie może być o 0 znaków, gdy parametr jest już palindromem.

Przykłady:

Wynikiem **roz("ko")** jest **"kok"** (możliwe palindromy to "oko" i "kok").

Wynikiem **roz("abba")** jest **"abba"**.

Wynikiem **roz("abbaca")** jest **"acabbaca"**.

Omówienie rozwiązania

Rozwiązanie polega na sprawdzaniu, czy dane słowo jest palindromem i w przypadku negatywnej odpowiedzi rozszerzanie napisu o kolejną literę. Sprawdzanie czy napis jest palindromem, realizujemy poprzez porównanie napisu oraz napisu odwróconego. Jeśli są identyczne, napis jest palindromem, w przeciwnym przypadku nie jest.

W pętli zaczynamy od kroku 0, czyli od sprawdzania, czy dane słowo jest palindromem. W i-tym kroku algorytmu rozszerzamy napis o i-tą literę, a dokładniej rozszerzając z tyłu bierzemy i-liter początkowych i dopisujemy je w odwrotnej kolejności na koniec. Następnie sprawdzamy, czy tak powstały napis jest palindromem. Podobnie postępujemy, rozpatrując dopisywanie od przodu, z tym że rozpatrujemy odwrócony napis. Na koniec i-tego kroku pętli sprawdzamy, czy otrzymaliśmy palindrom: tylko w pierwszym przypadku – dopisanie liter na koniec, tylko w drugim przypadku – dopisanie liter na początek, czy w obu. Jeśli odpowiedź jest twierdząca, to otrzymany palindrom jest on wynikiem funkcji. Przy czym, gdy w danym obrocie pętli wygenerowane zostały dwa palindromy, to przekazujemy jako wynik wcześniejszy leksykograficznie. Jeśli nie znaleźliśmy palindromu, postępowanie kontynuujemy tak długo, aż nie otrzymamy palindromu. Górnym ograniczeniem na długość słowa jest dwukrotna długość wyjściowego słowa pomniejszona o 1.

Rozwiązanie w języku Python

W języku Python istnieje możliwość wyodrębnienia fragmentu napisu poprzez notacją nawiasową. Zapis `napis[a:b:c]` oznacza, że ze zmiennej typu napis wybieramy fragment od elementu `a` do `b` co `c` elementów. Jeśli wybieramy od początku napisu, to pomijamy `a`, jeśli do końca to pomijamy b. Wobec tego zapis `napis[::-1]` w istocie powoduje odwrócenie słowa.

```
1.  def czy_pali(napis):
2.      return napis==napis[::-1]
3.
4.  def rozszerz(napis,ile):
5.      return napis + (napis[:ile])[::-1]
6.
7.
8.  def roz(napis):
9.      for i in range(len(napis)):
10.         ile = 0
11.         x = rozszerz(napis,i)
12.         y = rozszerz(napis[::-1],i)
13.         if czy_pali(x):
14.             ile += 1
15.         if czy_pali(y):
16.             ile += 2
17.         if ile == 1:
18.             return x
19.         if ile == 2:
20.             return y
21.         if ile == 3:
22.             if x < y:
23.                 return x
24.             else:
25.                 return y
26.     return -1
```

W powyższym zapisie użyto zmiennej pomocniczej `ile`, która wskazuje jakie palindromy otrzymamy w i-tym obrocie pętli. Jeśli zmienna ile ma wartość 0 – to nie otrzymaliśmy palindromu, jeśli 1 – otrzymano palindrom przez dopisanie liter na końcu, jeśli 2 – na początku. Gdy uzyskano oba palindromy, to 3.

Testy

Testy obejmują różne przypadki – gdy dopisujemy jedną lub więcej liter na końcu, na początku lub w ogóle nie dopisujemy – dana jest palindromem. W Pythonie jest zdefiniowana operacja mnożenia liczby przez napis. Zapis 7*"kajak" oznacza powtórzenie 7 razy napisu "kajak", czyli

'kajakkajakkajakkajakkajakkajakkajak'.

Przykładowe testy:

roz("defdefdeffedfed") → defdefdeffedfedfed

roz("zzzxyxyxy") → zzzxyxyxyxzzz

roz("uuuughturguuuu") → uuuughturguuuugruthguuuu

roz("cecccdc") → cdccccecccdc

roz(7*"kajak") → kajakkajakkajakkajakkajakkajakkajak

roz(15*"okno")
oknooknooknooknooknooknooknooknooknooknooknooknonkoonkoonkoonkoonkoon
koonkoonkoonkoonkoonkoonkoonkoonkoonkoonkoonko

roz(23*"abbac")

cabbacabbacabbacabbacabbacabbacabbacabbacabbacabbacabbacabbacabbacabbacabbacabb
acabbacabbacabbacabbacabbacabbac

roz(99*"o"+"x"+100*"o")

oooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo
ooooooooooooooooooooooxooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo
ooooooooooooooooooooooooooooooooooooooooooooooo

roz("abcdefghijklmnopqrstuvwxyz")

abcdefghijklmnopqrstuvwxyzyxwvutsrqponmlkjihgfedcba

roz(200*"bloki")

blokiblokiblokiblokiblokiblokiblokiblokiblokiblokiblokiblokiblokiblokiblokiblokiblokiblokiblokiblokiblokiblok
iblokiblokiblokiblokiblokiblokiblokiblokiblokiblokiblokiblokiblokiblokiblokiblokiblokiblokiblokiblokiblokiblo
kiblokiblokiblokiblokiblokiblokiblokiblokiblokiblokiblokiblokiblokiblokiblokiblokiblokiblokiblokiblokiblokibl
okiblokiblokiblokiblokiblokiblokiblokiblokiblokiblokiblokiblokiblokiblokiblokiblokiblokiblokiblokiblokiblokib
lokiblokiblokiblokiblokiblokiblokiblokiblokiblokiblokiblokiblokiblokiblokiblokiblokiblokiblokiblokiblokibloki
blokiblokiblokiblokiblokiblokiblokiblokiblokiblokiblokiblokiblokiblokiblokiblokiblokiblokiblokiblokiblokiblok
iblokiblokiblokiblokiblokiblokiblokiblokiblokiblokiblokiblokiblokiblokiblokiblokiblokiblokiblokiblokiblokiblo
kiblokiblokiblokiblokiblokiblokiblokiblokiblokiblokiblokiblokiblokiblokiblokiblokiblokiblokiblokiblokiblokibl
okiblokiblokiblokiblokiblokiblokiblokiblokiblokiblokiblokiblokiblokiblokiblokiblokiblokiblokiblokiblokiblokib
lokiblokiblokiblokiblokiblokiblokiblokiblokiblokiblokiblokiblokiblokiblokiblokiblokiblokiblokiblokiblokikolbikolbikolbikolbikolbikolbikolbikolbik
olbikolbikolbikolbikolbikolbikolbikolbikolbikolbikolbikolbikolbikolbikolbikolbikolbikolbikolbikolbikolbikolbi
kolbikolbikolbikolbikolbikolbikolbikolbikolbikolbikolbikolbikolbikolbikolbikolbikolbikolbikolbikolbikolbikolb
ikolbikolbikolbikolbikolbikolbikolbikolbikolbikolbikolbikolbikolbikolbikolbikolbikolbikolbikolbikolbikolbikol
bikolbikolbikolbikolbikolbikolbikolbikolbikolbikolbikolbikolbikolbikolbikolbikolbikolbikolbikolbikolbikolbiko
lbikolbikolbikolbikolbikolbikolbikolbikolbikolbikolbikolbikolbikolbikolbikolbikolbikolbikolbikolbikolbikolbik
olbikolbikolbikolbikolbikolbikolbikolbikolbikolbikolbikolbikolbikolbikolbikolbikolbikolbikolbikolbikolbikolbi
kolbikolbikolbikolbikolbikolbikolbikolbikolbikolbikolbikolbikolbikolbikolbikolbikolbikolbikolbikolbikolbikolb
ikolbikolbikolbikolbikolbikolbikolbikolbikolbikolbikolbikolbikolbikolbikolbikolbikolbikolbikolbikolbikolbikol
bikolbikolbikolbikolbikolbikolbikolbikolbikolbikolbikolbikolbikolbikolbikolbikolbikolbikolbikolbikolbikolbiko
lbikolbikolbikolbikolb