Zadanie Szyfr Bacona – LOGIA 17 (2016/17), etap 2

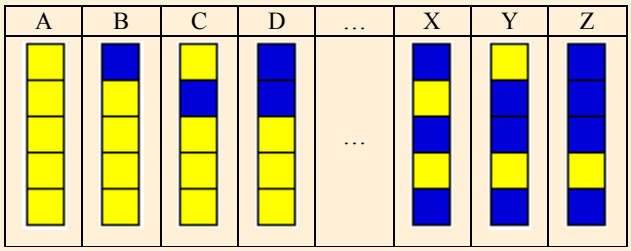
Treść zadania

Szyfr Bacona polega na zastępowaniu liter alfabetu łacińskiego pięcioznakowymi ciągami złożonymi z liter a i b zgodnie z poniższą tabelą:

A	B	C	D	E	F	G	H	I oraz J	K	L	M
aaaaa	aaaab	aaaba	aaabb	aabaa	aabab	aabba	aabbb	abaaa	abaab	ababa	ababb

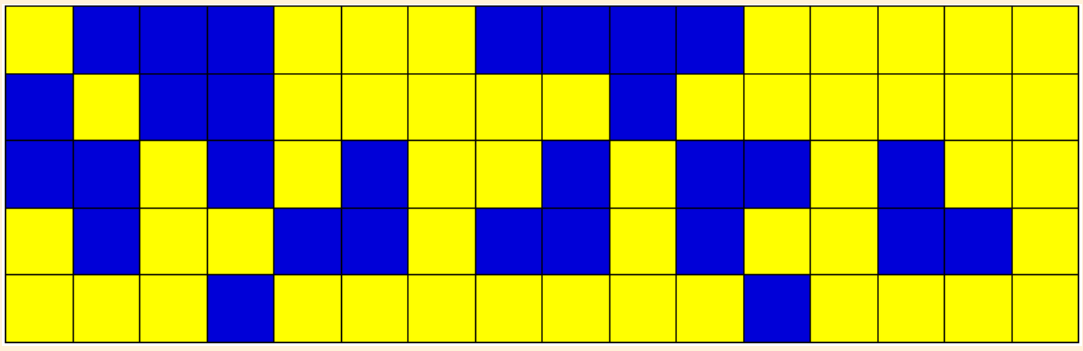
N	O	P	Q	R	S	T	U i V	W	X	Y	Z
abbaa	abbab	abbba	abbbb	baaaa	baaab	baaba	baabb	babaa	babab	babba	babbb

Adaś postanowił szyfrować kolejne litery słowa. Zamiast liter a i b użył odpowiednio zamalowanych kwadratów:



Napisz jednoparametrową procedurę/funkcję szyfruj, po wywołaniu której na środku ekranu powstanie rysunek zaszyfrowanego słowa (od lewej do prawej). Parametrem jest niepuste słowo złożone z wielkich liter alfabetu łacińskiego o maksymalnej długości 70. Szerokość rysunku wynosi 700 lub wysokość wynosi 400. Do zamalowania użyj dwóch dowolnych kolorów różnych od koloru krawędzi kwadratów.

Przykład:



efekt wywołania:
w Logo – `szyfruj "GODZINAKODOWANIA`
w Pythonie – `szyfruj('GODZINAKODOWANIA')`

Omówienie rozwiązania

Mamy do czynienia z szyfrem podstawieniowym. Każdej literze w danej odpowiada „słupek" składający się z odpowiednio zamalowanych kwadratów. Wystarczy tylko przejrzeć całe słowo litera po literze i rysować słupki odpowiadające aktualnej literze.

Najpierw warto zapamiętać sposób kodowania poszczególnych liter. W ty celu tworzymy zmienną **alfabet**, której wartością jest lista 26 kodów kolejnych liter. Litera **A** została zakodowana jako **aaaaa**, litera **B** jako **aaaab**, litera **C** jako **aaaba** itd. Ostatniej literze alfabetu łacińskiego **Z**, odpowiada kod **babbb**.

Kody ASCII wielkich liter są liczbami z zakresu od 65 (kod ASCII litery A) do 90 (kod ASCII litery Z). Kod ASCII aktualnej litery danego słowa pomniejszony o 65 jest indeksem zmiennej alfabet. Przeglądając kolejne znaki słowa podanego jako parametr znajdujemy jego kod. Następnie przeglądamy znaki kodu i sprawdzamy jaką wartość reprezentują. Jeśli napotkamy **b**, rysujemy kwadrat zamalowany kolorem niebieskim, jeśli **a**, to rysujemy kwadrat zamalowany na żółto. Kolejne kwadraty rysujemy jeden nad drugim, po zakodowaniu jednej litery przesuwamy żółwia do początkowej pozycji kolejnego słupka.

W treści zadania znajduje się informacja, że rysunek ma mieć szerokość 700 lub wysokość 400. Dla wszystkich wartości parametrów rysunek musi mieścić się na ekranie. Wytyczne do rozwiązywania zadań określają wymiary ekranu, które obowiązują podczas trwania zawodów. Każdy rysunek musi się mieścić na stronie (w Logomocji), przy czym nie wolno zmieniać standardowego rozmiaru strony lub w prostokącie o szerokości 796 i wysokości 499 oraz o środku w punkcie (0,0) (w Pythonie).

Samo wyliczenie długości boku jednego kwadratu jako iloraz 700 przez długość danego słowa nie wystarczy, trzeba sprawdzić, czy słupek pięciu kwadratów zmieści się w pionie na ekranie.

Rozwiązanie w języku Python

```
1.  from turtle import *
2.
3.  def kw(a, kolor):
4.      fillcolor(kolor)
5.      begin_fill()
6.      for i in range(4):
7.          fd(a); rt(90)
8.      end_fill()
9.
10. def szyfruj(s):
11.     # obliczenia wielkości boku jednego kwadratu
12.     a = 700 / len(s)
13.     # sprawdzenie, czy rysunek mieści się w pionie na ekranie
14.     # jeśli nie, to obliczmy wielkość boku zakładając, że wysokość rysunku = 400
15.     if a * 5 > 498:
16.         a = 400 / 5
17.     # przejście do punktu rozpoczęcia rysowania
18.     pu(); bk(a * len(s)/2); lt(90); bk(a * 5 / 2); pd()
19.
20.     # definiowanie zakodowanych liter
21.     alfabet = ["aaaaa", "aaaab", "aaaba", "aaabb", "aabaa", "aabab", "aabba",
    "aabbb", "abaaa", "abaaa", "abaab", "ababa", "ababb", "abbaa", "abbab", "abbba",
    "abbbb", "baaaa", "baaab", "baaba", "baabb", "baabb", "babaa", "babab", "babba",
    "babbb"]
22.     # przeglądanie kolejnych liter danego słowa w pętli for
23.     for litera in s:
24.         # kod litery
```

```
        x = alfabet[ord(litera) - 65]
        # rysowanie słupka danej litery
        # przeglądanie kolejnych znaków kodu litery w pętli for
        for zn in x:
            if zn == "b":
                kw(a, "blue")
                fd(a)
            else:
                kw(a, "yellow")
                fd(a)
        # powrót po narysowaniu słupka litery
        pu(); bk(5*a); rt(90); fd(a); lt(90); pd()
```

Rozwiązanie w języku Logo

```
oto kw :a :kolor
 ukm :kolor
 wielokąt [powtórz 4[np :a pw 90]]
już

oto szyfruj :s
 ; obliczenia wielkości boku jednego kwadratu
 niech "a 700 / długość :s
 ; sprawdzenie, czy rysunek mieści się w pionie na ekranie
 ; jeśli nie, to obliczmy wielkość boku zakładając, że wysokość rysunku = 400
 jeśli :a * 5 > 498 [niech "a 400 / 5]
 ; przejście do punktu rozpoczęcia rysowania
 pod pw 90 ws (:a * długość :s) / 2
 lw 90 ws 5 * :a / 2 opu
 ; definiowanie zakodowanych liter
 niech "alfabet [aaaaa aaaab aaaba aaabb aabaa aabab aabba aabbb abaaa
                 abaaa abaab ababa ababb abbaa abbab abbba abbbb baaaa
                 baaab baaba baabb baabb babaa babab babba babbb]
 ; przeglądanie kolejnych liter danego słowa w pętli powtórz
 powtórz długość :s[
         niech "x element ((ascii pierw :s) - 64) :alfabet
         ; numerowanie elementów zmiennej alfabet zaczyna się od 1
         ; dlatego odejmujemy 64, a nie 65 jak było w Pythonie
         ; rysowanie słupka odpowiadającego jednej literze
         ; przeglądanie kolejnych znaków kodu litery
         powtórz 5[jeżeli pierw :x = "b[kw :a "jasnoniebieski np :a]
                                       [kw :a "żółty np :a]
                   niech "x bp :x]
         niech "s bp :s
         ; przejście do rysowania kolejnego słupka
         pod ws 5 * :a pw 90 np :a lw 90 opu
           ]
już
```

Testy

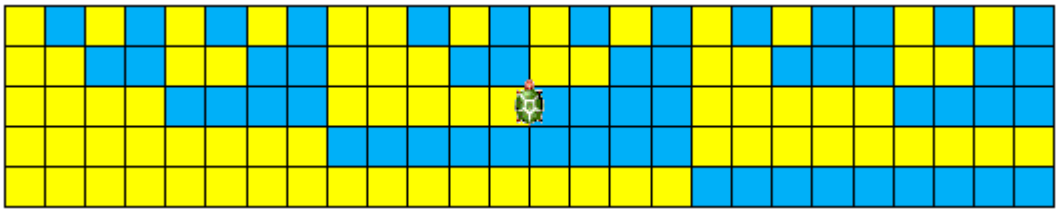
Testowanie rozwiązania powinno obejmować wszystkie litery i przypadki szczególne. Pierwszy test może obejmować cały alfabet. Tutaj należy sprawdzić prawidłowość zaszyfrowania każdej litery. Należy zwrócić uwagę na to, że zaszyfrowane litery **I** i **J** są identyczne. To samo dotyczy liter **U** i **V**.

Python

```
szyfruj("ABCDEFGHIJKLMNOPQRSTUVWXYZ")
```

Logo

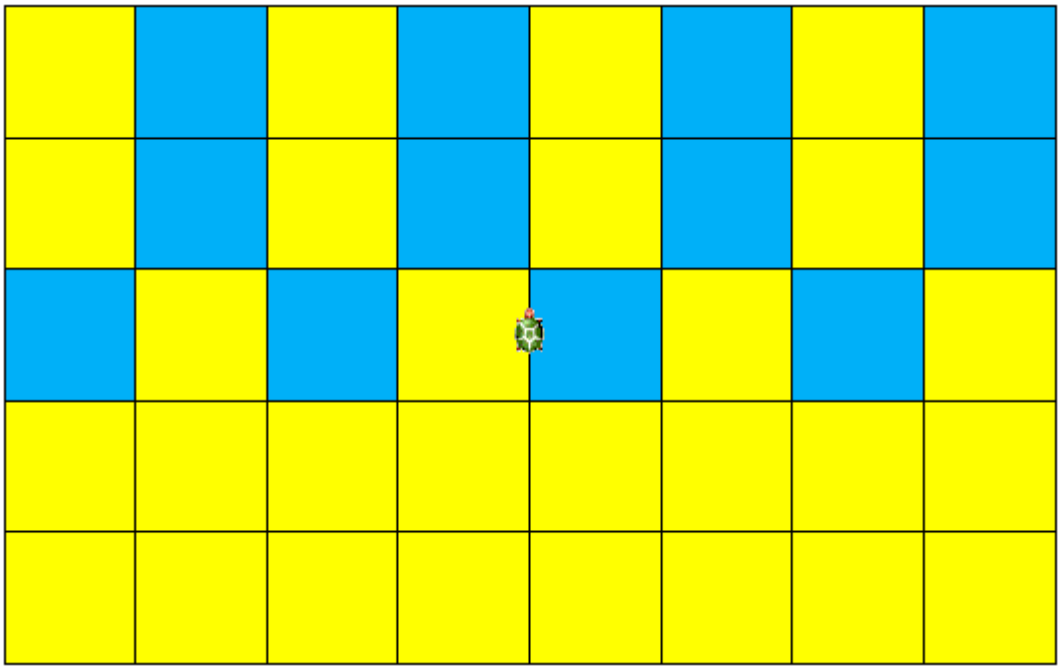
```
szyfruj "ABCDEFGHIJKLMNOPQRSTUVWXYZ
```



Uwaga! Na zrzucie żółw stoi na środku ekranu. Sprawdzamy w ten sposób wyśrodkowanie rysunku. Dane, to cały alfabet. Ten rysunek ma szerokość 700.

```
szyfruj("EDEDEDED")
```

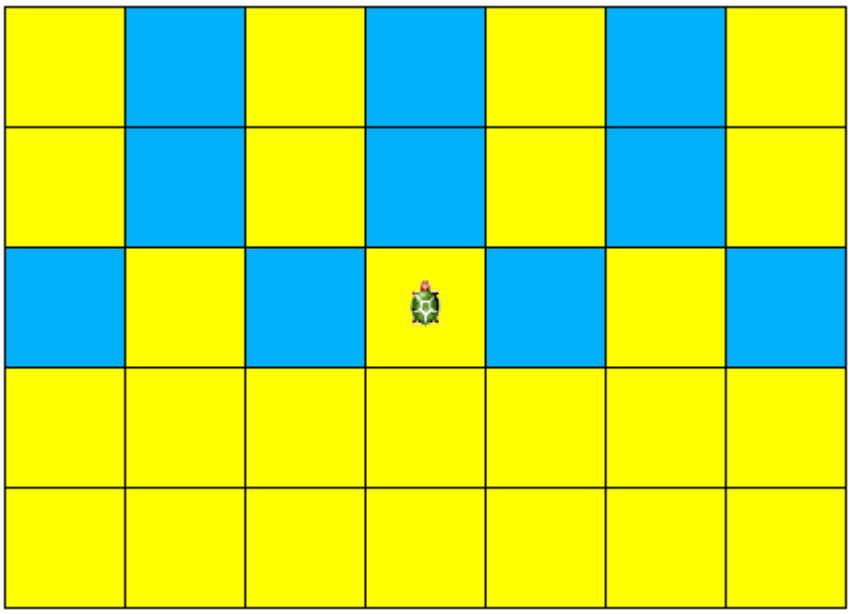
```
szyfruj "EDEDEDED
```



Uwaga! Ten rysunek ma szerokość 700.

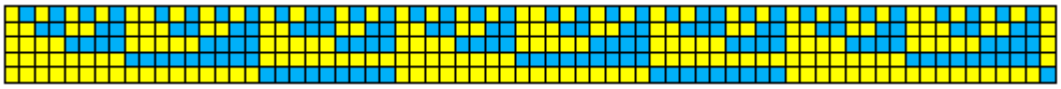
```
szyfruj("EDEDEDE")
```

```
szyfruj "EDEDEDE
```



Uwaga! Ten rysunek ma wysokość 400, ponieważ gdyby miał szerokość 700, to nie mieściłby się na ekranie w pionie.

```
szyfruj("ABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMNOPQR")
 szyfruj "ABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMNOPQR
```



Uwaga! Ten rysunek ma szerokość 700.

```
szyfruj("S")
```

```
szyfruj "S
```

Uwaga! Ten rysunek ma wysokość 400, ponieważ gdyby miał szerokość 700, to nie mieściłby się na ekranie w pionie.

Zadanie Klocki – LOGIA 17 (2016/17), etap 2

Treść zadania

Staś układa na stole sześcienne klocki tej samej wielkości. Na każdym widnieje jedna z wielkich liter alfabetu łacińskiego. Klocki układane są rzędami (wierszami). Każdy rząd rozpoczyna się przy lewej krawędzi stołu. Pierwszy rząd zawiera jeden klocek, drugi – dwa, trzeci – trzy, …, itd. Ostatni rząd może być krótszy. Napisz jednoparametrową funkcję **kolit**, której parametrem jest niepuste słowo długości co najwyżej 1000, zawierające litery widniejące na kolejnych klockach. Wynikiem funkcji jest liczba tych kolumn w układance Stasia, w których wszystkie litery są identyczne.

Przykłady:

Python:	Logo:
wynikiem **kolit('ABCDEFGH')** jest **1**,	wynikiem **kolit "ABCDEFGH** jest **1**,
wynikiem **kolit('ALAMAKRABY')** jest **2**.	wynikiem **kolit "ALAMAKRABY** jest **2**.

W pierwszym przykładzie warunek zadania spełnia ostatnia kolumna, w drugim przykładzie druga i ostatnia.

Omówienie rozwiązania

Warto rozpisać ułożenie liter w kolumnach, a tak naprawdę numery (pozycje) liter w słowie układających się w kolumny. Na przykładach z treści zadania ustawienie jest następujące:

A B C D E F G H	1 2 3 4 5 6 7 8
A L A M A K R A B Y	1 2 3 4 5 6 7 8 9 10

Rozpiszmy pozycje liter trochę szerzej i zauważmy pewne zależności związane z położeniem litery w danym wierszu i kolumnie.

	1	2	3	4	5	6
1	1					
2	2	3				
3	4	5	6			
4	7	8	9	10		
5	11	12	13	14	15	
6	16	17	18	19	20	21

Będziemy musieli sprawdzić poszczególne litery w każdej kolumnie, a więc musimy wiedzieć, jaki jest numer pierwszej litery w danej kolumnie oraz jak wyznaczyć numer kolejnej litery w tej kolumnie. Pierwsze litery kolejnych kolumn mają odpowiednio numery: 1, 3, 6, 10, 15, 21, …. Łatwo można zauważyć, że numer litery rozpoczynającej kolejną kolumnę otrzymamy dodając do numeru pierwszej litery poprzedniej kolumny numer bieżącej (dla drugiej kolumny mamy 3=1+2, dla trzeciej 6=3+3, dla czwartej 10=6+4, itd.). Natomiast numer kolejnej litery w obrębie danej kolumny uzyskamy dodając numer aktualnego wiersza.

Należy policzyć liczbę kolumn, w których wszystkie litery są takie same. Potrzebny więc będzie licznik kolumn oraz pomocnicze zmienne określające numer kolumny, numer wiersza, numer pierwszej litery w danej kolumnie, numer badanej litery w danej kolumnie. Rozwiązanie możemy zapisać w dwóch pętlach, zewnętrzna przebiega po kolumnach oraz wewnętrzna sprawdzająca litery aktualnej kolumny. Zapis algorytmu może być następujący:

1. licznik kolumn ← 0
2. numer kolumny ← 1
3. początkowy numer litery ← 1
4. dopóki początkowy numer litery ≤ długość słowa
   - numer wiersza ← numer kolumny
   - numer litery ← początkowy numer litery + numer wiersza
   - dopóki numer litery ≤ długość słowa i
     słowo[początkowy numer litery] = słowo[numer litery]
     - numer wiersza ← numer wiersza + 1
     - numer litery ← numer litery + numer wiersza
   - jeśli numer litery > długość słowa
     - licznik kolumn ← licznik kolumn + 1
   - numer kolumny ← numer kolumny + 1
   - początkowy numer litery ← początkowy numer litery + numer kolumny
5. wynik licznik kolumn

Zwróćmy uwagę, że długość ostatniego wiersza (rzędu) może być krótsza, ale nie ma to znaczenia, ponieważ sprawdzamy, czy numer litery mieści się w obrębie słowa. Jeśli porównywanie liter zakończy się w sytuacji, że numer litery będzie większy od długości słowa, oznacza to, że sprawdziliśmy wszystkie litery w danej kolumnie i wszystkie były takie same. Powiększamy wtedy licznik kolumn.

Rozwiązanie w języku Python

```
1.  def kolit(slowo):
2.      ilek = 0
3.      nrk = 1
4.      ipocz = 1
5.      while ipocz <= len(slowo):
6.          nrw = nrk
7.          i = ipocz + nrw
8.          while i <= len(slowo) and slowo[ipocz-1] == slowo[i-1]:
9.              nrw += 1
10.             i += nrw
11.         if i > len(slowo):
12.             ilek += 1
13.         nrk += 1
14.         ipocz += nrk
15.     return ilek
```

W Pythonie indeksowanie (numerowane) rozpoczyna się zawsze od zera, dlatego od numerów porównywanych liter zostało odjęte 1.

Rozwiązanie w języku Logo

```
oto kolit :słowo
  niech "ilek 0
  niech "nrk 1
  niech "ipocz 1
  dopóki [ :ipocz <= długość :słowo ]
         [
           niech "nrw :nrk
           niech "i :ipocz + :nrw
           niech "dobrze "prawda
           dopóki [ i :dobrze :i <= długość :słowo ]
                  [
                    niech "dobrze (element :ipocz :słowo) = element :i :słowo
                    zwiększ "nrw
                    (zwiększ "i :nrw)
                  ]
           jeśli :dobrze [ zwiększ "ilek ]
           zwiększ "nrk
           (zwiększ "ipocz :nrk)
         ]
  wynik :ilek
już
```

Warunki logiczne w Logo są zawsze wyliczane „do końca”, tzn. jeśli pierwszy element koniunkcji daje wartość fałsz, to i tak, mimo że znamy już wartość całego wyrażenia, wyliczany jest drugi element. W związku z tym warunek

numer litery ≤ długość słowa i słowo[początkowy numer litery] = słowo[numer litery]

dla numeru litery większego od długości słowa powodowałby błąd. Dlatego wprowadzona została pomocnicza zmienna logiczna `dobrze` i porównanie liter słowa odbywa się wewnątrz pętli, kiedy wiemy, że numer litery jest nie większy od długości słowa.

Testy

Testowanie rozwiązania warto rozpocząć od krótkich słów, dla których jest łatwo ręcznie policzyć wynik.

Python	Logo	
kolit('AAAAAA')	kolit "AAAAAA	3
kolit('AAAABA')	kolit "AAAABA	2
kolit('X')	kolit "X	1
kolit('OPQRSTUVW')	kolit "OPQRSTUVW	0
kolit('QQAQBVQCVGQDVHXQEVIXKQFVJXLZ')	kolit "QQAQBVQCVGQDVHXQEVIXKQFVJXLZ	4

kolit('AABABCABCDABCDEABCDEFABCDEFGA BCDEFGHABCDEFGHIABCDEFGHIJABCDEFGHI JKABCDEFGHIJKLABCDEFGHIJKLMABCDEFGHI JKLMNABCDEFGHIJKLMNOABCDEFGHIJKLMN OPABCDEFGHIJKLMNOPQABCDEFGHIJKLMN OPQRABCDEFGHIJKLMNOPQRSABCDEFGHIJK LMNOPQRSTABCDEFGHIJKLMNOPQRSTUABC DEFGHIJKLMNOPQRSTUVABCDEFGHIJKLMNO PQRSTUVWABCDEFGHIJKLMNOPQRSTUVWX ABCDEFGHIJKLMNOPQRSTUVWXYABCDEFGH IJKLMNOPQRSTUVWXYZABCDEFGHIJKLMNOP QRSTUVWXYZAABCDEFGHIJKLMNOPQRSTUV WXYZAAABCDEFGHIJKLMNOPQRSTUVWXYZA AAABCDEFGHIJKLMNOPQRSTUVWXYZAAAAA BCDEFGHIJKLMNOPQRSTUVWXYZAAAAAABC DEFGHIJKLMNOPQRSTUVWXYZAAAAAAABCD EFGHIJKLMNOPQRSTUVWXYZAAAAAAAABCD EFGHIJKLMNOPQRSTUVWXYZAAAAAAAAABC DEFGHIJKLMNOPQRSTUVWXYZAAAAAAAAAA BCDEFGHIJKLMNOPQRSTUVWXYZAAAAAAAA AAABCDEFGHIJKLMNOPQRSTUVWXYZAAAAA AAAAAAABCDEFGHIJKLMNOPQRSTUVWXYZA AAAAAAAAAAAABCDEFGHIJKLMNOPQRSTUV WXYZAAAAAAAAAAAAAABCDEFGHIJKLMNOP QRSTUVWXYZAAAAAAAAAAAAAAABCDEFGHI JKLMNOPQRSTUVWXYZAAAAAAAAAAAAAAA ABCDEFGHIJKLMNOPQRSTUVWXYZAAAAAAA AAAAAAAAAABCDEFGHIJKLMNOPQRSTUVWX YZAAAAAAAAAAAAAAAAAABCDEFGHIJKLMN OPQRSTUVWXYZAAAAAAAAAAAAAAAAAZZ ZZZZZZZZ')	kolit "AABABCABCDABCDEABCDEFABCDEFGA BCDEFGHABCDEFGHIABCDEFGHIJABCDEFGHI JKABCDEFGHIJKLABCDEFGHIJKLMABCDEFGHI JKLMNABCDEFGHIJKLMNOABCDEFGHIJKLMN OPABCDEFGHIJKLMNOPQABCDEFGHIJKLMN OPQRABCDEFGHIJKLMNOPQRSABCDEFGHIJK LMNOPQRSTABCDEFGHIJKLMNOPQRSTUABC DEFGHIJKLMNOPQRSTUVABCDEFGHIJKLMNO PQRSTUVWABCDEFGHIJKLMNOPQRSTUVWX ABCDEFGHIJKLMNOPQRSTUVWXYABCDEFGH IJKLMNOPQRSTUVWXYZABCDEFGHIJKLMNOP QRSTUVWXYZAABCDEFGHIJKLMNOPQRSTUV WXYZAAABCDEFGHIJKLMNOPQRSTUVWXYZA AAABCDEFGHIJKLMNOPQRSTUVWXYZAAAAA BCDEFGHIJKLMNOPQRSTUVWXYZAAAAAABC DEFGHIJKLMNOPQRSTUVWXYZAAAAAAABCD EFGHIJKLMNOPQRSTUVWXYZAAAAAAAABCD EFGHIJKLMNOPQRSTUVWXYZAAAAAAAAABC DEFGHIJKLMNOPQRSTUVWXYZAAAAAAAAAA BCDEFGHIJKLMNOPQRSTUVWXYZAAAAAAAA AAABCDEFGHIJKLMNOPQRSTUVWXYZAAAAA AAAAAAABCDEFGHIJKLMNOPQRSTUVWXYZA AAAAAAAAAAAABCDEFGHIJKLMNOPQRSTUV WXYZAAAAAAAAAAAAAABCDEFGHIJKLMNOP QRSTUVWXYZAAAAAAAAAAAAAAABCDEFGHI JKLMNOPQRSTUVWXYZAAAAAAAAAAAAAAA ABCDEFGHIJKLMNOPQRSTUVWXYZAAAAAAA AAAAAAAAAABCDEFGHIJKLMNOPQRSTUVWX YZAAAAAAAAAAAAAAAAAABCDEFGHIJKLMN OPQRSTUVWXYZAAAAAAAAAAAAAAAAAZZ ZZZZZZZZ	34

Zadanie Ulica – LOGIA 17 (2016/17), etap 2

Treść zadania

Logolandia ma tylko jedną owalną ulicę, przy której stoją domy podzielone na osiedla. Dwa domy stojące obok siebie są w jednym osiedlu, jeśli różnica ich numerów jest nie większa niż 3. Dom o największym numerze jest w tym samym osiedlu, co dom o numerze 1, 2 lub 3 (o ile istnieje). W Logolandii stoją co najmniej 2 domy, ale nie więcej niż 10000. Największy numer domu jest nie większy niż 30 000. Numery domów nie powtarzają się.

Napisz jednoparametrową funkcję **maksos**, której parametrem jest uporządkowana rosnąco lista numerów domów. Wynikiem jest liczba domów w osiedlu składającym się z największej liczby domów.

Przykłady:

Język	**Wywołanie**	**Wynik**	Uzasadnienie
Python	`maksos([1,4,7,13,14,15,20])`	4	osiedle 20-1-4-7
Logo	`maksos [1 4 7 13 14 15 20]`		
Python	`maksos([1,4,7,11,13,14,15,16,20])`	5	osiedle 11-13-14-15-16
Logo	`maksos [1 4 7 11 13 14 15 16 20]`		

Omówienie rozwiązania

Rozwiązanie polega na przeglądaniu kolejnych par elementów oraz sprawdzaniu, czy domy z rozpatrywanej pary należą do tego samego osiedla. Dokładniej, zmiennej `licznik` przypisujemy wartość 1 – początkową wielkość pierwszego osiedla, a zmiennej `naj` – wielkość najdłuższego dotąd znalezionego osiedla. Jeżeli dwa sąsiednie domy są dostatecznie blisko – należą do tego samego osiedla, czyli ich numery nie różnią się więcej niż 3, zwiększamy licznik o 1, w przeciwnym przypadku mamy do czynienia z końcem osiedla. Wtedy sprawdzamy, czy nie znaleźliśmy większego osiedla od zapamiętanego, jeżeli tak, to uaktualniamy wartość zmiennej `naj` i ustawiamy `licznik` na 1. W ten sposób przeglądamy wszystkie pary sąsiednich domów.

Musimy jeszcze rozpatrzeć przypadek, gdy do jednego osiedla należą domy o najwyższych i najniższych numerach, czyli osiedle jest „zawijane”. W tym celu sprawdzamy, czy pierwszy dom należy do tego samego osiedla co ostatni. Jeśli tak, to rozpatrujemy to osiedle. Tak długo, dopóki kolejne pary domów są blisko siebie, zwiększamy wartość zmiennej licznik. Kończymy, gdy dojdziemy do końca listy lub gdy napotkamy dom z nowego osiedla.

Jeżeli ostatnio rozpatrywane osiedle jest największe, to musimy jeszcze uaktualnić wartość zmiennej `naj`.

Może się zdarzyć, że wszystkie domy należą do tego samego osiedla, wtedy wynikiem funkcji jest długość listy.

OEiiZK
Ośrodek Edukacji Informatycznej
i Zastosowań Komputerów w Warszawie

Rozwiązanie w języku Python

```python
def maksos(lista):
    licznik = 1
    naj = 0

    # zliczanie osiedli bez zawijania
    for i in range(1, len(lista)):
        if lista[i]-lista[i-1] <= 3:
            licznik += 1
        else:
            if licznik > naj:
                naj = licznik
            licznik = 1

    # zawijanie
    if lista[0] < 3:
        licznik += 1
        i = 1
        while i < len(lista) and lista[i] - lista[i-1] <= 3:
            licznik += 1
            i += 1
    # ostatnie osiedle największe
    if licznik > naj:
        naj = licznik

    # wszystkie domy w jednym osiedlu
    if naj == 2 * len(lista):
      return len(lista)

    return naj
```

Rozwiązanie w języku Logo

```
oto maksos :lista
  niech "licznik 1
  niech "naj 0

  ; zliczanie osiedli bez zawijania
  powtórz długość bp :lista
  [
    jeżeli (element npw+1 :lista) - (element npw :lista) <= 3
      [zwiększ "licznik]
      [jeśli :licznik > :naj [niech "naj :licznik]
       niech "licznik 1
      ]
  ]

  ; zawijanie
  jeśli pierw :lista < 3
  [ zwiększ "licznik
    niech "ii 1
    dopóki [ :ii < długość :lista]
    [
      jeżeli (element npw+1 :lista) - (element npw :lista) > 3
        [niech "ii długość :lista]
        [zwiększ "licznik
         zwiększ "ii]
    ]
  ]

  ; ostatnie osiedle największe
  jeśli :licznik > :naj [niech "naj :licznik]


  ; wszystkie domy w jednym osiedlu
  jeśli :naj = (2 * długość :lista)
    [wy długość :lista]

  wy :naj
już
```

Testy

Rozwiązanie testujemy dla różnych przypadków. Najprostszy wydaje się test dla najdłuższego osiedla w środku listy. Sprawdzamy też przypadki, kiedy najdłuższe osiedle jest na końcu listy oraz zawijane. Wśród testów nie powinno zabraknąć przykładu, gdy wszystkie domy są w jednym osiedlu.

Dla języka Python

Wywołanie	Wynik
`maksos([4,7,10,13,16,19,22,25,28,34,37,40,43,46,49,52,55,59,60,63])`	9
`maksos([4,7,10,13,16,19,22,25,28,34,37,40,43,46,49,52,55,59,60,63,64,65,66,67,68,69,70,100])`	10
`maksos([10,20]+[i for i in range(100,1000,1)])`	900
`maksos([1,2,3,4,7,13,14,15,19,20])`	7

`maksos([1,2,3,4,7,10,13,14,15,19,20])`	11
`maksos([1,2])`	2
`maksos([1,20])`	2
`maksos([1,10,20])`	2
`maksos([i for i in range(1,101,1)]+[i for i in range(200,300,1)]` `+[i for i in range(400,500,1)])`	200
`maksos([i for i in range(1,101,1)])`	100

Dla języka Logo

Wywołanie	Wynik
`maksos [4 7 10 13 16 19 22 25 28 34 37 40 43 46 49 52 55 59 60 63]`	9
`maksos [4 7 10 13 16 19 22 25 28 34 37 40 43 46 49 52 55 59 60 63 64 65 66 67 68 69 70 100]`	10
`maksos zd [10 20] generuj 900 [:%+1] 100`	900
`maksos [1 2 3 4 7 13 14 15 19 20]`	7
`maksos [1 2 3 4 7 10 13 14 15 19 20]`	11
`maksos [1 2]`	2
`maksos [1 20]`	2
`maksos [1 10 20]`	2
`maksos (zd (generuj 100 [:%+1] 1) (generuj 100 [:%+1] 200) (generuj 100 [:%+1] 400))`	200
`maksos generuj 100 [:%+1] 1`	100