Zadanie Własny System Kodów Paskowych – miniLOGIA 16 (2017/18), etap 3

Treść zadania

Michał przygotowuje Własny System Kodów Paskowych (WSKP) i dlatego szuka liczb, których suma cyfr jest większa od **a** i mniejsza od **b**. Przegląda kolejne liczby począwszy od 1 do 100000 (włącznie). Napisz trójparametrową funkcję **jaka**. Pierwszym parametrem funkcji jest lewy koniec badanego przedziału (**a**), drugim prawy koniec (**b**). Wynikiem funkcji jest **n**-ta kolejna liczba (określona przez trzeci parametr), której suma cyfr jest większa niż **a** i mniejsza niż **b**. Jeśli taka liczba nie istnieje, to wynikiem jest -1. Przyjmij, że trzeci parametr **n** wynosi co najwyżej 100, natomiast **a** < **b** < 40.

Przykłady:

Python:	Logo:
wynikiem **jaka(1, 10, 4)** jest **5**,	wynikiem **jaka 1 10 4** jest **5**,
wynikiem **jaka(1, 3, 100)** jest **-1**.	wynikiem **jaka 1 3 100** jest **-1**.

W pierwszym przykładzie kolejne liczby spełniające warunek to 2, 3, 4, 5. W drugim przykładzie istnieje tylko 15 liczb spełniających warunek.

Omówienie rozwiązania

Rozwiązanie zadania polega na przeglądaniu kolejnych liczb począwszy od 1. Dla każdej liczby **x** liczymy sumę cyfr i jeśli mieści się w podanym przedziale **a** < **suma** oraz **suma** < **b**, to zwiększamy licznik o 1. Jeśli licznik jest równy podanej liczbie **n**, to kończymy działanie funkcji. Wynikiem jest wartość zmiennej **x**. W przeciwnym przypadku rozpatrujemy kolejną liczbę. Postępowanie kontynuujemy tak długo, aż nie osiągniemy podanego progu 100000.

Warto zdefiniować pomocniczą funkcję `suma_cyfr`, której wynikiem dla danej liczby jest suma jej cyfr. Sumę liczymy, dodając kolejno wartości ostatniej cyfry.

Rozwiązanie w języku Python

```
1.  def suma_cyfr(liczba):
2.      suma = 0
3.      while liczba != 0:
4.          suma += liczba % 10
5.          liczba = liczba // 10
6.      return suma
7.
8.  def jaka(a, b, n):
9.      licznik = 0
10.     x = 1
11.     while x <= 100000:
12.         suma = suma_cyfr(x)
13.         if a < suma and suma < b:
14.             licznik += 1
15.             if licznik == n:
16.                 return x
17.         x += 1
18.     return -1
```

Rozwiązanie w języku Logo

```
oto suma_cyfr :liczba
   niech "suma 0
   dopóki [:liczba<>0]
     [
       niech "suma :suma + reszta :liczba 10
       niech "liczba ilorazc :liczba 10
     ]
   wy :suma
już

oto jaka :a :b :n
  niech "licznik  0
  niech "x 1
  dopóki [:x <= 100000]
    [ niech "suma suma_cyfr :x
      jeśli  i :a < :suma :suma < :b
        [ zwiększ "licznik
          jeśli :licznik = :n  [wy  :x]
        ]
      zwiększ "x
    ]
  wy -1
już
```

Testy

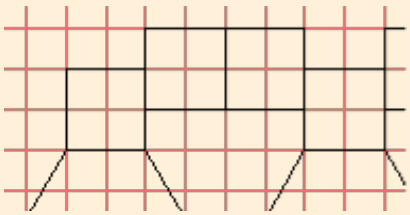
Testowanie rozwiązania warto rozpocząć od przykładów, których wyniki można sprawdzić ręcznie np. jaka(1, 3, 5). Dla tego przykładu liczby spełniające warunek tzn. takie, których suma cyfr wynosi 2, to 2, 11, 20, 101, 110,… Czyli wynikiem jest liczba 110. Potem wykonujemy kolejne testy zmieniając zakres sumy i numer kolejnej liczby. Nie można zapomnieć o przypadkach, w których nie istnieje **n**-ta liczba w podanym przedziale np. `jaka(2, 4, 60)` oraz o warunkach brzegowych.

Python	Logo	
jaka(1, 3, 5) jaka(21, 23, 20)	jaka 1 3 5 jaka 21 23 20	110 985
jaka(3, 17, 34) jaka(23, 39, 58)	jaka 3 17 34 jaka 23 39 58	43 2598
jaka(2, 4, 60) jaka(1, 4, 90)	jaka 2 4 60 jaka 1 4 90	-1 -1
jaka(1, 40, 100) jaka(20, 40, 100)	jaka 1 40 100 jaka 20 40 100	103 1589
jaka(25, 30, 100) jaka(28, 32, 100)	jaka 25 30 100 jaka 28 32 100	4896 7967

Zadanie Ogniwa – miniLOGIA 16 (2017/18), etap 3

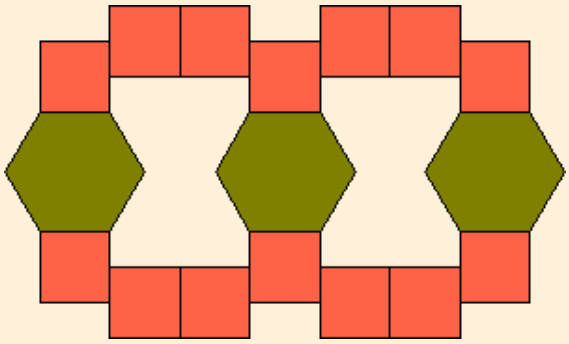
Treść zadania

Napisz dwuparametrową procedurę/funkcję **ogniwa**, po wywołaniu której na środku ekranu powstanie rysunek łańcuszka złożonego z dwukolorowych ogniw powstałych z kwadratowych i sześciokątnych koralików. Pierwszy parametr określa liczbę sześciokątnych koralików w łańcuszku i może przyjmować wartości od **3** do **10**. Drugi parametr określa liczbę kwadratowych koralików w najdłuższym rządku ogniwa i może przyjmować wartości od **2** do **14**. Szerokość rysunku wynosi **700**.

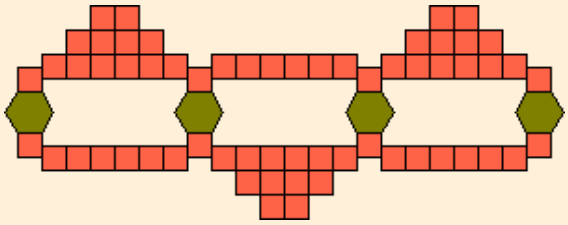


Rysunek pomocniczy

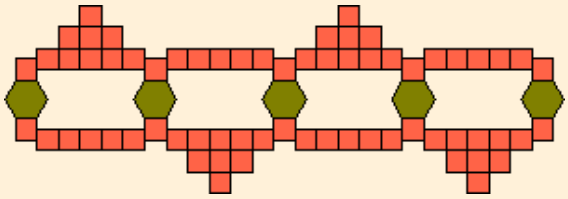
Przykłady:



efekt wywołania:
Logo – **ogniwa 3 2**
Python – **ogniwa (3, 2)**



efekt wywołania:
Logo – **ogniwa 4 6**
Python – **ogniwa (4, 6)**



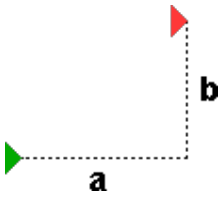
efekt wywołania:
Logo – **ogniwa 5 5**
Python – **ogniwa (5, 5)**

Omówienie rozwiązania

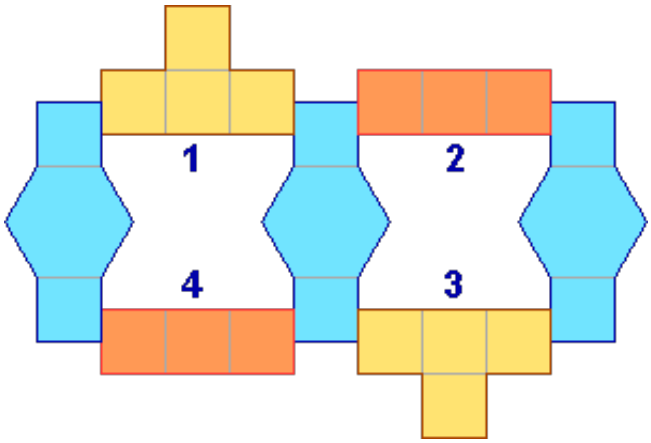
Przyjrzyjmy się uważnie treści zadania. Rysowane są dwa rodzaje figur: kwadraty i sześciokąty o takiej samej długości boku. Długość boków figur zależy od wartości obu parametrów funkcji. Szerokość rysunku wynosi 700, liczba tworzących łańcuszek ogniw jest o jeden mniejsza od liczby sześciokątów. Aby otrzymać długość boku kwadratu należy policzyć ich liczbę na całej długości łańcuszka. W tym celu liczbę ogniw mnożymy przez liczbę kwadratów w najdłuższym rządku ogniwa zwiększoną o 1 (kwadrat przy lewym sześciokącie ogniwa). Do otrzymanej wartości dodajemy 2, biorąc pod uwagę kwadrat rysowany przy ostatnim sześciokącie po prawej stronie łańcuszka oraz wystające fragmenty

sześciokątów na końcach łańcuszka. Następnie dzielimy 700 przez uzyskaną liczbę. Znając długość boku kwadratu i sześciokąta możemy przystąpić do rysowania.

Zauważmy, że przydatna może okazać się funkcja **skok**, przemieszczająca żółwia o podaną liczbę kroków w poziomie i pionie. Na rysunku pomocniczym widzimy wynik działania funkcji **skok(a, b)** w Pythonie. Korzystanie z tej funkcji znacznie uprasza opis poruszania się żółwia, jednak wymaga dokładnego wyliczenia wartości, o jakie chcemy go przemieścić.



Rysunek ogniw opisanych w treści zadania warto podzielić na powtarzające się fragmenty. Zaczynamy od napisania i przetestowania pomocniczych funkcji rysujących kwadrat i sześciokąt. Kolejnym krokiem będzie napisanie funkcji rysującej układ złożony z sześciokąta i dwóch kwadratów (kolor niebieski na rysunku pomocniczym). Przydatna może się także okazać funkcja rysująca pas o podanej długości oraz piramidę złożone z kwadratów (zaznaczone odpowiednio na żółto i pomarańczowo).



Przykładowy podział rysunku na powtarzające się fragmenty

Pisząc funkcję **piramida()** wygodnie jest wykorzystać przygotowaną wcześniej funkcję rysującą pasek z kwadratów, pamiętając, że każdy kolejny rządek kwadratów jest o 2 krótszy, a przed narysowaniem każdego rządka należy odpowiednio przemieścić żółwia. Ostatnim krokiem będzie zbudowanie całego rysunku z poszczególnych elementów. Najpierw możemy narysować wszystkie układy sześciokątów i kwadratów, następnie leżące pomiędzy nimi pasy i piramidy. Zauważmy, że, jeśli zaczniemy od lewego górnego elementu idąc najpierw w prawo, a następnie w dolnej części ogniw od prawej do lewej – pasy i piramidy występują na przemian. Zatem do narysowania ich wystarczy jedna pętla, w której w połowie przemieścimy żółwia do początku pierwszego dolnego elementu. Kolejność rysowania została pokazana na przykładowym rysunku. Wybór elementu do rysowania możemy uzależnić od parzystości zmiennej sterującej pętlą.

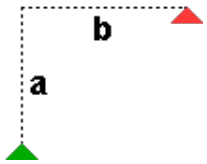
Należy pamiętać, by pisząc funkcję **ogniwa** opisaną w treści zadania zadbać nie tylko o prawidłową wielkość rysunku, ale także o jego wyśrodkowanie – zarówno w poziomie (co jest proste ze względu na stałą szerokość rysunku), jak i w pionie.

Bardzo istotne w tego typu zadaniach jest także ustalenie punktu początkowego, od którego zaczynamy rysowanie danego elementu. Wpływa to na miejsce, do którego będziemy przemieszczać żółwia, składając poszczególne elementy w całość. Najwygodniej jest zwykle kończyć rysowanie elementu w tym samym punkcie, w którym zaczynaliśmy.

Rozwiązanie w języku Python

```
1.  from turtle import *
2.  from math import *
3.
4.  def skok(a, b):
5.      pu(); fd(a); lt(90); fd(b); rt(90); pd()
6.
7.  def kwadrat(a):
8.      fillcolor("tomato");
9.      begin_fill()
10.     for i in range(4):
11.         fd(a); lt(90)
12.     end_fill()
13.
14. def szesc(a):
15.     fillcolor("olive");
16.     begin_fill()
17.     for i in range(6):
18.         fd(a); lt(60)
19.     end_fill()
20.
21. def pion(a):
22.     kwadrat(a)
23.     skok(0, a)
24.     szesc(a)
25.     skok(0, sqrt(3) * a)
26.     kwadrat(a)
27.     skok(0, -(1 + sqrt(3)) * a)
28.
29. def pas(a, n):
30.     for i in range(n):
31.         kwadrat(a)
32.         skok(a, 0)
33.     skok(-n * a, 0)
34.
35. def piramida(a, n):
36.     for i in range(n, 0, -2):
37.         pas(a,i)
38.         skok(a, a)
39.     x = n//2 + n%2
40.     skok(-x * a, -x * a)
41.
42. def ogniwa(n, m):
43.     a = 700 / ((n - 1) * (m + 1) + 2)
44.     skok(-350 + a / 2, -a - sqrt(3) * a / 2)
45.     for i in range(n):
46.         pion(a); skok((m + 1) * a, 0)
47.     skok(-n * ( m + 1) * a, 0)
48.     skok(a, (3 / 2 + sqrt(3)) * a)
49.     for i in range(2 * n - 2):
50.         if i%2 == 0:
51.             piramida(a, m)
52.         else:
53.             pas(a, m)
54.         skok((m + 1) * a, 0)
55.         if i == n - 2:
56.             lt(180); skok(a, (1 + sqrt(3)) * a)
57.     skok(-350 + a / 2, -a / 2 - sqrt(3) * a / 2); lt(180)
```

Rozwiązanie w języku Logo




```
oto skok :a :b
    pod np :a pw 90 np :b lw 90 opu
już

oto kwadrat :a
    powtórz 4 [np :a pw 90]
    skok :a/2 :a/2 ukm "jasnoczerwony zamaluj skok (-:a/2) (-:a/2)
już

oto szesc :a
    powtórz 6 [np :a pw 60]
    skok (pwk 3)*:a/2 :a/2 ukm "oliwkowy zamaluj skok (-(pwk 3)*:a/2) (-:a/2)
już

oto pion :a
    kwadrat :a
    skok :a 0
    lw 30 szesc :a pw 30
    skok (pwk 3)*:a 0
    kwadrat :a
    skok (-(1+pwk 3)*:a) 0
już

oto pas :a :n
    powtórz :n [kwadrat :a skok 0 :a]
    skok 0 (-:n*:a)
już

oto piramida :a :n
    niech "ile (ilorazc :n 2)+reszta :n 2
    powtórz :ile [pas :a :n  skok :a :a niech "n :n-2]
    skok (-:ile*:a) (-:ile*:a)
już

oto ogniwa1 :n :m
    niech "a 700/((:n-1)*(:m+1)+2)
    skok (-:a-(pwk 3)*:a/2) (-350+:a/2)
    powtórz :n [pion :a skok 0 (:m+1)*:a] skok 0 (-:n*(:m+1)*:a)
    skok (3/2+pwk 3)*:a :a
    powtórz 2*:n-2 [
                    jeżeli (reszta npw 2)=1 [piramida :a :m]
                                            [pas :a :m]
                    skok 0 (:m+1)*:a
                    jeżeli npw=:n-1 [pw 180 skok (1+pwk 3)*:a :a]
                   ]
    pw 180
    skok ((1+pwk 3)*:a/2) (350-:a/2)
już
```

Różnice w przemieszczaniu żółwia między rozwiązaniem w języku Logo i Python wynikają z innego jego położenia startowego. Na początku, po uruchomieniu programu, żółw w Pythonie jest skierowany w prawo, w Logo do góry. Także kwadrat i sześciokąt w Logo są rysowane w prawą stronę, a w Pythonie były w lewą. Możliwe jest dosłowne przeniesienie definicji z jednego języka do drugiego, należy jednak wtedy pamiętać, by przed rozpoczęciem rysowania ogniw dodać dodatkowy obrót żółwia o 90 stopni.

Testy

Testowanie rozwiązania rozpoczynamy od przykładów zawartych w treści zadania. Potem testujemy działanie programu dla wartości skrajnych parametrów, wartości parzystych i nieparzystych itp. Warto sprawdzić poprawność rozwiązania dla wszystkich możliwych kombinacji wartości parametrów określonych w treści zadania: pierwszy parametr od 3 do 10, drugi od 2 do 14.

W języku Python, aby przyspieszyć tworzenie rysunku przez żółwia, stosujemy wywołanie złożone z funkcji **tracer()** – rysownie w pamięci, właściwego wywołania funkcji **ogniwa()** i na końcu uaktualniamy ekran za pomocą funkcji **update()**. Przykład:

```
tracer(0)
ogniwa(3, 4)
update()
```

Powrót do standardowego trybu rysowania uzyskamy wywołując funkcję **tracer()** z parametrem równym 1.

Zadanie Robot – miniLOGIA 16 (2017/18), etap 3

Treść zadania

Robot porusza się od lewego do prawego końca korytarza. Został tak zaprogramowany, że wykonuje tylko dwie podstawowe czynności: idzie naprzód o 1 krok lub zawraca, gdy nie może pójść naprzód. Na początku robot stoi w lewym końcu korytarza, jest skierowany w kierunku prawego i jest naładowany.

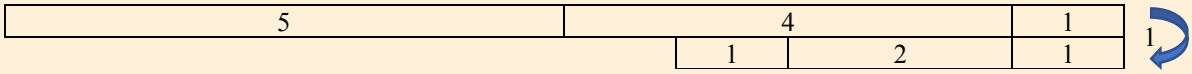
Po uruchomieniu porusza się do drugiego końca, a potem zawraca. Wykonując czynności (ruch lub zawracanie) traci energię, więc co jakiś czas musi zostać doładowany. Doładowanie nie jest jednak pełne, każde kolejne umożliwia wykonanie o jedną czynność mniej niż poprzednie.

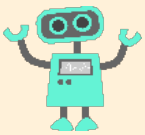
Napisz dwuparametrową funkcję **robot**. Pierwszy parametr określa liczbę czynności, jaką robot może wykonać na pełnym naładowaniu, drugi długość korytarza wyrażoną w krokach robota. Oba parametry są liczbami naturalnymi od 1 do 1000. Wynikiem funkcji jest odległość robota od lewego końca po wykonaniu wszystkich możliwych czynności.

Przykład 1:

Dla wartości pierwszego naładowania równej 5 i długości korytarza 10.

Robot po pierwszym naładowaniu idzie 5 kroków, po drugim naładowaniu 4 kroki, po trzecim 1 krok, zawraca i idzie 1 krok, po czwartym naładowaniu 2 kroki, a po ostatnim 1 krok. Odległość od lewego końca wynosi **6**.

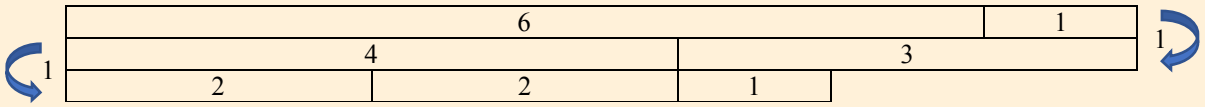




Przykład 2:

Dla wartości pierwszego naładowania równej 6 i długości korytarza 7.

Robot po pierwszym naładowaniu idzie 6 kroków, po drugim naładowaniu 1 krok, zawraca, a potem idzie 3 kroki, po trzecim naładowaniu idzie 4 kroki, po czwartym zawraca i idzie 2 kroki, po piątym idzie 2 kroki, a po ostatnim 1. Odległość od lewego końca wynosi **5**.



Logo:

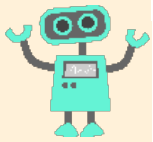
wynikiem **robot 5 10** jest **6**,

wynikiem **robot 6 7** jest **5**.

Python:

wynikiem **robot(5,10)** jest **6**,

wynikiem **robot(6,7)** jest **5**.



Omówienie rozwiązania

Zadanie można rozwiązać implementując symulację ruchu robota. Zapamiętujemy bieżące położenie robota i rozpatrujemy krok po kroku jego pozycję. Jest to rozwiązanie wynikające wprost z treści zadania, ale pracochłonne do implementacji.

Można jednak zauważyć pewną zależność. Jeśli pełne naładowanie wynosi 6, to kolejne wartości naładowania wynoszą: 6, 5, 4, 3, 2, 1. Można liczby zapisać w innej kolejności 6, 1, 5, 2, 4, 3. Teraz łatwo zauważyć, że sumując liczby parami, otrzymamy takie same wartości 7, a średnio (n+1)/2. Takich liczb jest n. Wobec tego wzór na łączną energię można zapisać n*(n+1)/2. Podobnie będzie w przypadku nieparzystej wartości pełnego naładowania: 5, 4, 3, 2, 1; średnio otrzymujemy (5+1)/2=3. Jeśli długość korytarza wynosi dl, to pełny cykl – ruch tam i z powrotem z obrotami – wynosi 2*dl+2.

Obliczamy położenie robota ze wzoru:

energia = n*(n+1)/2, gdzie n - początkowe naładowanie
polozenie = *energia* **mod** (2*dl+2), gdzie dl – długość korytarza, a **mod** oznacza resztę z dzielenia.

i rozpatrujemy cztery przypadki:

- robot idzie z lewej do prawej
  Jeśli `polozenie < dl + 1`, wynikiem jest wartość `polozenie`
- robot jest na prawym końcu
  Jeśli `polozenie = dl + 1`, wynikiem jest wartość `polozenie - 1`
- robot wraca – idzie z prawej do lewej
  Jeśli `polozenie > dl + 1` i `polozenie < 2 * dl + 2`,
  wynikiem jest `wartość 2 * dl + 1 - polozenie`
- robot jest w położeniu początkowym (lub po pełnym cyklu)
  Jeśli `polozenie = 2 * dl + 2`, wynikiem jest wartość `0`

Rozwiązanie w języku Python

Warunki zapisane w języku Python są w wersji uproszczonej w stosunku do zapisu w języku naturalnym, gdyż polecenie **return** nie tylko przekazuje wynik, ale kończy działanie funkcji.

```
1.  def robot(n, dl):
2.      polozenie = (n* (n + 1) // 2  ) % (2 * dl + 2)
3.
4.      if polozenie < dl + 1:
5.          return polozenie
6.
7.      if polozenie == dl + 1:
8.          return polozenie - 1
9.
10.     if polozenie < 2 * dl + 2:
11.         return 2 * dl + 1 - polozenie
12.
13.     return 0
```

Rozwiązanie w języku Logo

Warunki zapisane w języku Logo są w wersji uproszczonej w stosunku do zapisu w języku naturalnym, gdyż polecenie **wynik** nie tylko przekazuje wynik, ale kończy działanie funkcji.

```
1.  oto robot :n :dl
2.    niech "polozenie  reszta ((:n + 1) * :n / 2  )  (2 * :dl + 2)
3.
4.    jeśli :polozenie < :dl + 1
5.      [wy :polozenie]
6.
7.    jeśli :polozenie = :dl + 1
8.      [wy :polozenie - 1]
9.
10.   jeśli :polozenie < 2 * :dl + 2
11.     [wy 2 * :dl + 1 - :polozenie]
12.
13.   wy 0
14. już
```

Testy

Najpierw rozwiązanie sprawdzamy dla najprostszych scenariuszy, gdy robot idzie tylko do przodu i nie zawraca, potem dla bardziej skomplikowanych. W testach nie powinno zabraknąć przykładów dla wszystkich przypadków: robot jest w drodze z lewej do prawej, na końcu, w drodze z prawej do lewej i na samym początku. Sprawdzamy działanie funkcji dla małych i dużych wartości ze szczególnym uwzględnieniem wartości brzegowych.

Python	Logo	wynik
robot(42, 100) robot(36, 45)	robot 42 100 robot 36 45	95 22
robot(60, 35) robot(700, 500)	robot 60 35 robot 700 500	30 139
robot(34, 5) robot(423, 7)	robot 34 5 robot 423 7	4 3
robot(42, 6) robot(404, 100)	robot 42 6 robot 404 100	6 0
robot(500, 1000) robot(1000, 1000)	robot 500 1000 robot 1000 1000	875 0