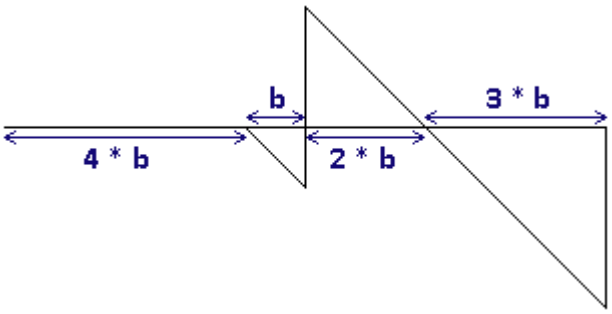
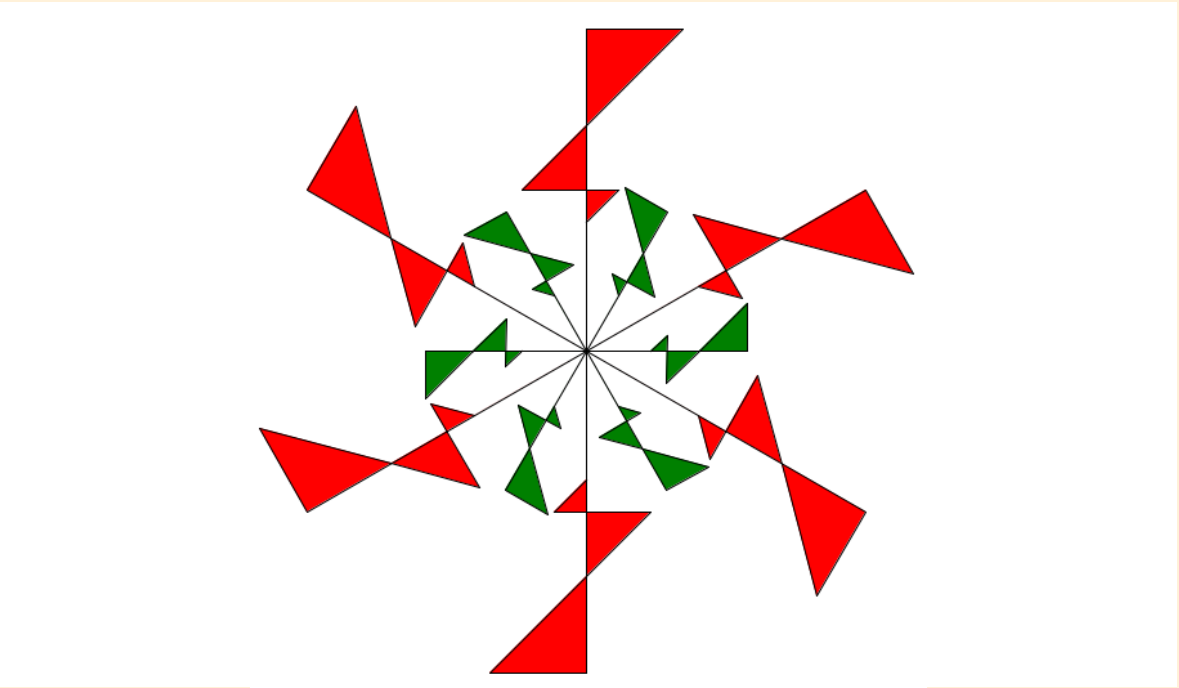
Zadanie Wiatrak – LOGIA 21 (2020/21), etap 1

Treść zadania

Napisz funkcję **wiatrak()**, po wywołaniu której powstanie na ekranie wiatrak złożony z łopat w dwóch kolorach – zielonym i czerwonym. Długość czerwonej łopaty wynosi **250**, a zielonej **125**. Proporcje łopat odczytaj z rysunku pomocniczego.



rysunek pomocniczy



Efekt wywołania: **wiatrak()**

Omówienie rozwiązania

Zadanie polega na narysowaniu układu złożonego z 6 elementów zielonych i 6 elementów czerwonych, nazywanych w treści zadania łopatami. Każda łopata składa się z trzech trójkątów prostokątnych równoramiennych o długościach ramion *b*, *2*b* oraz *3*b* (podanych na rysunku pomocniczym) oraz z odcinka o długości *4*b*. Dla łopaty czerwonej długość odcinka *b* jest równa 250/10, dla zielonej 125/10. Długość trzeciego boku trójkąta (przeciwprostokątnej) obliczamy z twierdzenia Pitagorasa, np.:

$$x^2 = (3 * b)^2 + (3 * b)^2 = 2 * (3 * b)^2$$

$$x = 3 * \sqrt{2} * b$$

Tworząc pojedynczą łopatę możemy rysować każdy trójkąt osobno i zamalowywać go, lub narysować i zamalować od razu cały układ trzech trójkątów. Kąty między dwoma kolejnymi czerwonymi łopatami wynoszą 360/6, czyli 60 stopni. Podobnie jest z łopatami w kolorze zielonym. Kąt między łopatą czerwoną i jej sąsiadem w kolorze zielonym wynosi połowę tej liczby, czyli 30 stopni. Tworząc cały wiatrak możemy rysować najpierw wszystkie łopaty czerwone, następnie obrócić żółwia o kąt 30 stopni i narysować łopaty zielone, albo rysować na przemian łopaty w jednym i w drugim kolorze. Należy przy tym pamiętać, by położenie łopat było identyczne jak w treści zadania, a w szczególności odpowiednie dwie czerwone łopaty były narysowane pionowo, a odpowiednie dwie zielone – poziomo.

Rozwiązanie w języku Python

Wersja programu z oddzielnymi funkcjami pomocniczymi rysującymi zieloną i czerwoną łopatę.

```
1.  from turtle import *
2.  from math import *
3.
4.  def lopata_czerwona(a):
5.      b = a / 10
6.      fillcolor("red")
7.      begin_fill()
8.      fd(a); rt(90); fd(3 * b); rt(135); fd(3 * b * sqrt(2))
9.      fd(2 * b * sqrt(2)); lt(135); fd(3 * b); rt(135); fd(b * sqrt(2))
10.     lt(45); pu(); fd(4 * b); pd(); lt(180)
11.     end_fill()
12.
13. def lopata_zielona(a):
14.     b = a / 10
15.     fillcolor("green")
16.     begin_fill()
17.     fd(a); lt(90); fd(3 * b); lt(135); fd(3 * b * sqrt(2))
18.     fd(2 * b * sqrt(2)); rt(135); fd(3 * b); lt(135); fd(b * sqrt(2))
19.     rt(45); pu(); fd(4 * b); pd(); lt(180)
20.     end_fill()
21.
22. def wiatrak():
23.     lt(90)
24.     for i in range(6):
25.         lopata_czerwona(250)
26.         lt(60)
27.     rt(30)
28.     for i in range(6):
29.         lopata_zielona(125)
30.         lt(60)
```

Jedno z rozwiązań uczniowskich wykorzystujące instrukcję warunkową do ustalenia kierunku rysowania łopaty, dzięki temu zdefiniowano jedną wspólną funkcję rysującą zieloną i czerwoną łopatę.

```
1.  from turtle import *
2.  from math import sqrt
3.
4.  def wind(k):
5.      if k == 1:
6.          fillcolor("red")
7.          b = 25
8.      else:
9.          fillcolor("green")
10.         b = 12.5
11.     begin_fill()
12.     fd(10 * b); rt(90 * k);fd(3 * b)
13.     lt(45 * k); bk(5 * sqrt(2) * b)
14.     rt(45 * k); fd(3 * b)
15.     lt(45 * k); bk(b * sqrt(2))
16.     lt(45 * k); bk(4 * b)
17.     end_fill()
18.
19. def wiatrak():
20.     for i in range(12):
21.         if i % 2 == 0:
22.             wind(-1)
23.         else:
24.             wind(1)
25.         rt(30)
```

Testy

Ponieważ otrzymany rysunek ma stałą wielkość oraz liczbę i położenie łopat, wystarczy raz przetestować działanie funkcji. Sprawdzamy, czy długości, kształt i kolor łopaty są prawidłowe oraz czy ich położenie jest zgodne z rysunkiem w treści zadania.

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

```
tracer(0)
wiatrak()
update()
```

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