SPRAWOZDANIE - LISTA 5

Małgorzata Kowalczyk

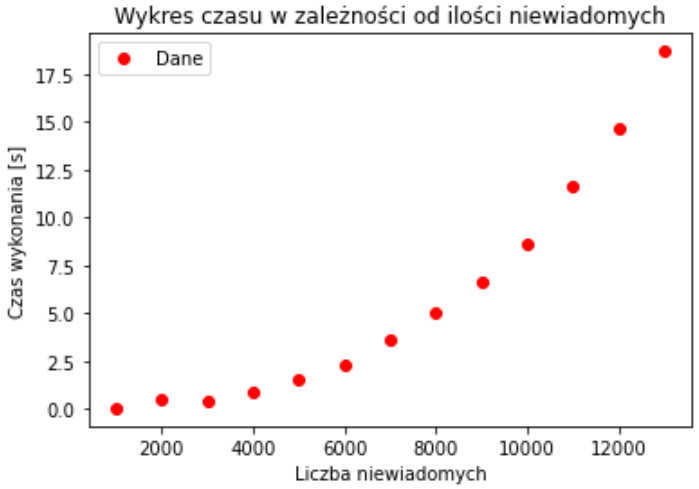
Kamil Kowalski

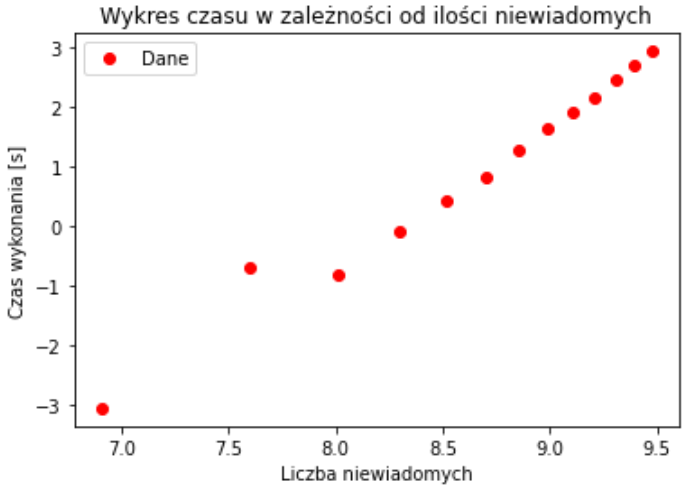
21.12.2021

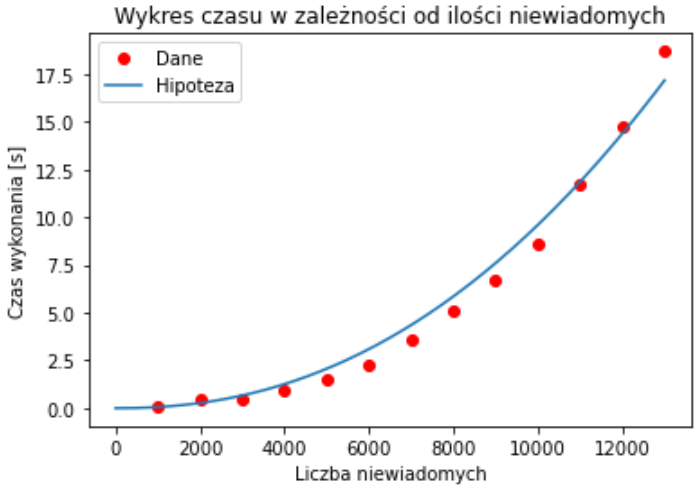
Zadanie 1

In [22]:
```
import numpy as np
from scipy import linalg
import random
import time
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit
import csv


class Fit_class:
    def __init__(self):
        pass

    def fit_fun(self, x, a):
        return a * x ** self.power

def time_checker(n):
    a = np.array([[random.randint(-100, 100) for _ in range(n)] for _ in range(n)])
    b = np.array([random.randint(-100, 100) for _ in range(n)])
    start = time.time()
    linalg.solve(a, b)
    stop = time.time()
    return stop - start


def plot(x, y):
    plt.plot(x, y, 'ro', label="Dane")
    plt.xlabel("Liczba niewiadomych")
    plt.ylabel("Czas wykonania [s]")
    plt.legend(loc='upper left')
    plt.title("Wykres czasu w zależności od ilości niewiadomych")
    plt.show()



def hypothesis_plot(x, y, func, popt):
    x2 = np.arange(1, x[-1])

    plt.plot(x, y, 'ro', label="Dane")
    plt.plot(x2, func(x2, *popt), label="Hipoteza")
    plt.xlabel("Liczba niewiadomych")
    plt.ylabel("Czas wykonania [s]")
    plt.legend(loc='upper left')
    plt.title("Wykres czasu w zależności od ilości niewiadomych")
    plt.show()


if __name__ == "__main__":
    n = [1000*i for i in range(1,14)]

    execution_times = []
    # for i in n:
    #     execution_times.append(time_checker(i))
    #
    # with open('execution_times', 'w', newline='') as myfile:
```

```
#     wr = csv.writer(myfile, quoting=csv.QUOTE_ALL)
#     wr.writerow(execution_times)

with open('execution_times.csv', newline='') as f:
    reader = csv.reader(f)
    execution_times = [float(i) for i in list(reader)[0]]

inst = Fit_class()
inst.power = np.polyfit(np.log(n), np.log(execution_times),1)[0]

plot(n, execution_times)
plot(np.log(n), np.log(execution_times))
popt, pcov = curve_fit(inst.fit_fun, n, execution_times)
hypothesis_plot(n, execution_times,inst.fit_fun, popt)
```





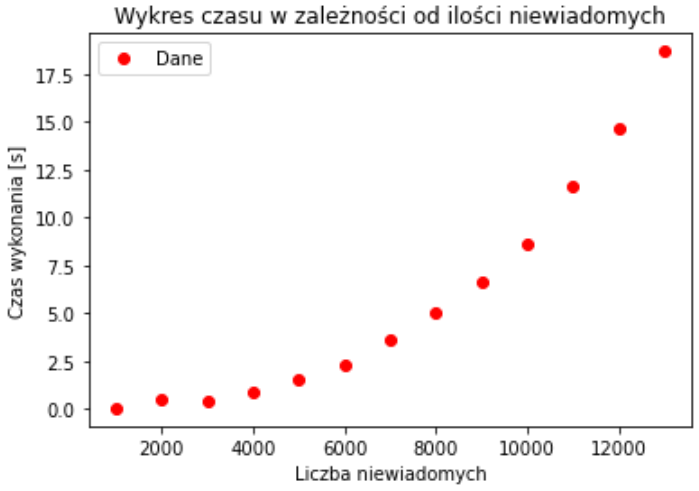


W tym zadaniu musieliśmy przeprowadzić analizę eksperymentalną złożoności obliczeniowej funkcji solve z modułu scipy.linalg. W pierwszym etapie napisaliśmy funkcję time_checker(n) tworzącą i rozwiązującą równania liniowe z n niewiadomymi. Tworzy ona macierze z losowymi wartościami dla współczynników

znajdujących się w równaniach liniowych, a następnie rozwiązuje i mierzy czas potrzebny na rozwiązanie takiego układu równań. W tablicy n przechowujemy liczbę niewiadomych, dla których chcemy zbadać czas wykonywania się programu, a w tablicy execution_times - czas wykonania dla poszczególnych niewiadomych. Dodatkowo stworzyliśmy klasę Fit_class, która umożliwia nam utworzenie funkcji $a \cdot x^b$ o zadanym przez nas parametrze b, który będziemy wyliczać.
Analizę przeprowadziliśmy dla n ze zbioru $[1000, 2000, \ldots, 13000]$. Ponieważ taki czas wykonywania takiego programu trwa stosunkowo długo, wyniki zapisaliśmy do pliku execution_times.csv.

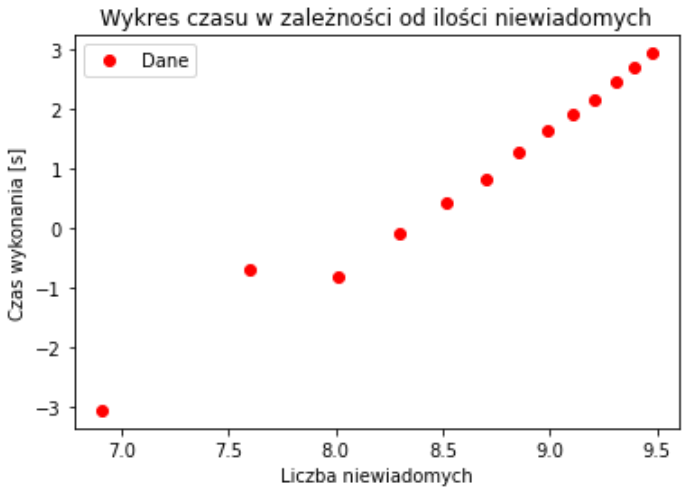
Oto przedstawione dane na wykresie:

In [23]:
```
plot(n, execution_times)
```



Możemy sprawdzić jak wygląda wykres logarytmiczny.

In [24]:
```
plot(np.log(n), np.log(execution_times))
```



Ponieważ punkty na wykresie logarytmicznym, układają się w linie prostą, możemy spodziewać się zależności potęgowej $y = ax^b$.
Za pomocą funkcji numpy.polyfit obliczymy współczynnik nachylenia prostej przedstawionej powyżej:

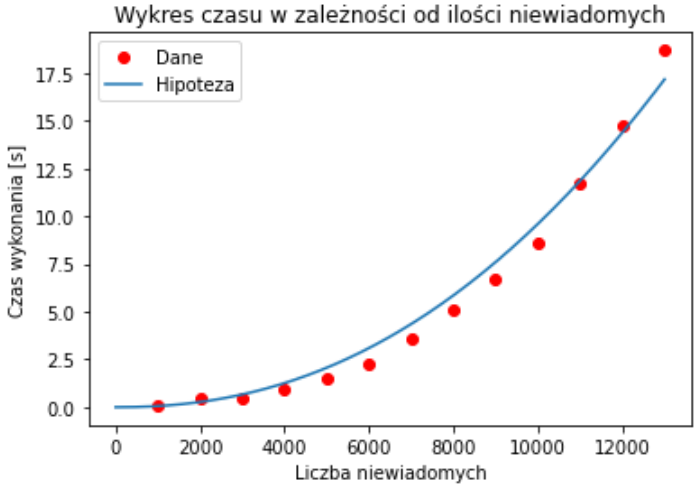
In [5]:
```
np.polyfit(np.log(n), np.log(execution_times),1)[0]
```

Out[5]:
```
2.2160006323290373
```

Wynika z tego, że nasza funkcja będzie miała wzór $T(N) = aN^{2.2160006323290373}$
Teraz za pomocą funkcji curve_fit z moduły scipy dopasowujemy wzór do wykresu.

In [25]:
```
print("a = "+str(popt[0]))
hypothesis_plot(n, execution_times,inst.fit_fun, popt)
```

```
a = 1.3126418966071992e-08
```



Wynika z tego, że czas wykonania algorytmu spełnia zależność

$$T(N) = 0.0000000131264189 \cdot N^{2.2160006323290373}$$

Możemy sprawdzić czy nasz wzór pokrywa się z rzeczywistością.

In [26]:
```
inst.fit_fun(510, popt[0])
```

Out[26]:
```
0.01312573053530281
```

In [27]:
```
time_checker(510)
```

Out[27]:
```
0.015581130981445312
```

Dane empiryczne potwierdzają naszą hipotezę. Funkcja solve jest złożoności $O(N^{2.2160006323290373})$

Zadanie 2

W tym zadaniu, naszym celem było napisanie programu rozwiązującego zagadnienie wieży z Hanoi przy użyciu trzech stosów do przechowywania krążków. W związku z tym stworzyliśmy klasę Tower_Stack, która reprezentuje nasze 3 słupki (wyjściowy - na którym znajduje się określona liczba krążków - source, pomocniczy - helper i docelowy - target).

Do klasycznej definicji stosu dopisaliśmy reprezentację tekstową _ *str* _.

Stworzyliśmy główną funkcję hanoi_stack(), która przy przenoszeniu krążków, będzie nam zwracać dokładną informację, który krążek jest przenoszony z jakiego słupka na jaki. Dodatkowo przy pomocy zmiennej globalnej move - przy każdym przeniesieniu, liczba ruchów zwiększa się o 1. Dzięki temu możemy na końcu zauważyć ile jest wszystkich potrzebnych ruchów do rozwiązania problemu, co potwierdza nam wzór dany z wykładu:

$$2^n - 1$$

Dodaliśmy także funkcję show_hanoi(), która jest odpowiedzialna za lepsze zwizualizowanie wyniku. Pokazuje dokładnie, co znajduje się na naszych słupkach po każdym ruchu.

In [16]:
```
class Tower_Stack:
    '''Represents a tower'''
    def __init__(self, name, num_disks = 0):
        self.name = name
        self.disks = []
        for i in range(num_disks, 0, -1):
            self.push(str(i))
    def __str__(self):
        disks = ''.join('{:<2}'.format(d) for d in self.disks)
        return '{}[ {}]'.format(self.name, disks)
    def push(self, disk):
        self.disks.append(disk)
    def pop(self):
        return self.disks.pop()

move = 0

def hanoi_tower(n, source, helper, target):
    global move
    if n < 0:
        raise ValueError('The amount of discs has to be positive')
    if n >= 1:
        hanoi_tower(n - 1, source, target, helper)
        move += 1
        print('Move disk - {} from {} to {} (count of moves: {}).'.format(str(n), source.name,
        target.push(source.pop())
        show_hanoi(source, helper, target)
        hanoi_tower(n - 1, helper, source, target)

def show_hanoi(A, B, C):
     for t in [A, B, C]:
            print(t)
```

Poniżej przedstawiamy przykładowe wywołania.

In [7]:
```
def main():
    number = 0
    if type(number) != int:
        raise TypeError('The amount of discs has to be an integer')
    a = Tower_Stack("Source-A ", number)
    b = Tower_Stack("Helper-B ")
    c = Tower_Stack("Target-C ")
    show_hanoi(a, b, c)
    hanoi_tower(number, a, b, c)
```

```
if __name__=="__main__":
    main()
```

```
Source-A [ ]
Helper-B [ ]
Target-C [ ]
```

In [9]:

```
def main():
    number = 1
    if type(number) != int:
        raise TypeError('The amount of discs has to be an integer')
    a = Tower_Stack("Source-A ", number)
    b = Tower_Stack("Helper-B ")
    c = Tower_Stack("Target-C ")
    show_hanoi(a, b, c)
    hanoi_tower(number, a, b, c)

if __name__=="__main__":
    main()
```

```
Source-A [ 1 ]
Helper-B [ ]
Target-C [ ]
Move disk - 1 from Source-A  to Target-C  (count of moves: 1).
Source-A [ ]
Helper-B [ ]
Target-C [ 1 ]
```

In [11]:

```
def main():
    number = 3
    if type(number) != int:
        raise TypeError('The amount of discs has to be an integer')
    a = Tower_Stack("Source-A ", number)
    b = Tower_Stack("Helper-B ")
    c = Tower_Stack("Target-C ")
    show_hanoi(a, b, c)
    hanoi_tower(number, a, b, c)

if __name__=="__main__":
    main()
```

```
Source-A [ 3 2 1 ]
Helper-B [ ]
Target-C [ ]
Move disk - 1 from Source-A  to Target-C  (count of moves: 1).
Source-A [ 3 2 ]
Helper-B [ ]
Target-C [ 1 ]
Move disk - 2 from Source-A  to Helper-B  (count of moves: 2).
Source-A [ 3 ]
Target-C [ 1 ]
Helper-B [ 2 ]
Move disk - 1 from Target-C  to Helper-B  (count of moves: 3).
Target-C [ ]
Source-A [ 3 ]
Helper-B [ 2 1 ]
Move disk - 3 from Source-A  to Target-C  (count of moves: 4).
Source-A [ ]
Helper-B [ 2 1 ]
Target-C [ 3 ]
Move disk - 1 from Helper-B  to Source-A  (count of moves: 5).
Helper-B [ 2 ]
Target-C [ 3 ]
Source-A [ 1 ]
Move disk - 2 from Helper-B  to Target-C  (count of moves: 6).
Helper-B [ ]
Source-A [ 1 ]
Target-C [ 3 2 ]
Move disk - 1 from Source-A  to Target-C  (count of moves: 7).
Source-A [ ]
Helper-B [ ]
Target-C [ 3 2 1 ]
```

In [13]:

```
def main():
    number = 5
```

```
    if type(number) != int:
        raise TypeError('The amount of discs has to be an integer')
    a = Tower_Stack("Source-A ", number)
    b = Tower_Stack("Helper-B ")
    c = Tower_Stack("Target-C ")
    show_hanoi(a, b, c)
    hanoi_tower(number, a, b, c)

if __name__=="__main__":
    main()
```

```
Source-A [ 5 4 3 2 1 ]
Helper-B [ ]
Target-C [ ]
Move disk - 1 from Source-A  to Target-C  (count of moves: 1).
Source-A [ 5 4 3 2 ]
Helper-B [ ]
Target-C [ 1 ]
Move disk - 2 from Source-A  to Helper-B  (count of moves: 2).
Source-A [ 5 4 3 ]
Target-C [ 1 ]
Helper-B [ 2 ]
Move disk - 1 from Target-C  to Helper-B  (count of moves: 3).
Target-C [ ]
Source-A [ 5 4 3 ]
Helper-B [ 2 1 ]
Move disk - 3 from Source-A  to Target-C  (count of moves: 4).
Source-A [ 5 4 ]
Helper-B [ 2 1 ]
Target-C [ 3 ]
Move disk - 1 from Helper-B  to Source-A  (count of moves: 5).
Helper-B [ 2 ]
Target-C [ 3 ]
Source-A [ 5 4 1 ]
Move disk - 2 from Helper-B  to Target-C  (count of moves: 6).
Helper-B [ ]
Source-A [ 5 4 1 ]
Target-C [ 3 2 ]
Move disk - 1 from Source-A  to Target-C  (count of moves: 7).
Source-A [ 5 4 ]
Helper-B [ ]
Target-C [ 3 2 1 ]
Move disk - 4 from Source-A  to Helper-B  (count of moves: 8).
Source-A [ 5 ]
Target-C [ 3 2 1 ]
Helper-B [ 4 ]
Move disk - 1 from Target-C  to Helper-B  (count of moves: 9).
Target-C [ 3 2 ]
Source-A [ 5 ]
Helper-B [ 4 1 ]
Move disk - 2 from Target-C  to Source-A  (count of moves: 10).
Target-C [ 3 ]
Helper-B [ 4 1 ]
Source-A [ 5 2 ]
Move disk - 1 from Helper-B  to Source-A  (count of moves: 11).
Helper-B [ 4 ]
Target-C [ 3 ]
Source-A [ 5 2 1 ]
Move disk - 3 from Target-C  to Helper-B  (count of moves: 12).
Target-C [ ]
Source-A [ 5 2 1 ]
Helper-B [ 4 3 ]
Move disk - 1 from Source-A  to Target-C  (count of moves: 13).
Source-A [ 5 2 ]
Helper-B [ 4 3 ]
Target-C [ 1 ]
Move disk - 2 from Source-A  to Helper-B  (count of moves: 14).
Source-A [ 5 ]
Target-C [ 1 ]
Helper-B [ 4 3 2 ]
Move disk - 1 from Target-C  to Helper-B  (count of moves: 15).
Target-C [ ]
Source-A [ 5 ]
Helper-B [ 4 3 2 1 ]
Move disk - 5 from Source-A  to Target-C  (count of moves: 16).
Source-A [ ]
```

```
Helper-B [ 4 3 2 1 ]
Target-C [ 5 ]
Move disk - 1 from Helper-B  to Source-A  (count of moves: 17).
Helper-B [ 4 3 2 ]
Target-C [ 5 ]
Source-A [ 1 ]
Move disk - 2 from Helper-B  to Target-C  (count of moves: 18).
Helper-B [ 4 3 ]
Source-A [ 1 ]
Target-C [ 5 2 ]
Move disk - 1 from Source-A  to Target-C  (count of moves: 19).
Source-A [ ]
Helper-B [ 4 3 ]
Target-C [ 5 2 1 ]
Move disk - 3 from Helper-B  to Source-A  (count of moves: 20).
Helper-B [ 4 ]
Target-C [ 5 2 1 ]
Source-A [ 3 ]
Move disk - 1 from Target-C  to Helper-B  (count of moves: 21).
Target-C [ 5 2 ]
Source-A [ 3 ]
Helper-B [ 4 1 ]
Move disk - 2 from Target-C  to Source-A  (count of moves: 22).
Target-C [ 5 ]
Helper-B [ 4 1 ]
Source-A [ 3 2 ]
Move disk - 1 from Helper-B  to Source-A  (count of moves: 23).
Helper-B [ 4 ]
Target-C [ 5 ]
Source-A [ 3 2 1 ]
Move disk - 4 from Helper-B  to Target-C  (count of moves: 24).
Helper-B [ ]
Source-A [ 3 2 1 ]
Target-C [ 5 4 ]
Move disk - 1 from Source-A  to Target-C  (count of moves: 25).
Source-A [ 3 2 ]
Helper-B [ ]
Target-C [ 5 4 1 ]
Move disk - 2 from Source-A  to Helper-B  (count of moves: 26).
Source-A [ 3 ]
Target-C [ 5 4 1 ]
Helper-B [ 2 ]
Move disk - 1 from Target-C  to Helper-B  (count of moves: 27).
Target-C [ 5 4 ]
Source-A [ 3 ]
Helper-B [ 2 1 ]
Move disk - 3 from Source-A  to Target-C  (count of moves: 28).
Source-A [ ]
Helper-B [ 2 1 ]
Target-C [ 5 4 3 ]
Move disk - 1 from Helper-B  to Source-A  (count of moves: 29).
Helper-B [ 2 ]
Target-C [ 5 4 3 ]
Source-A [ 1 ]
Move disk - 2 from Helper-B  to Target-C  (count of moves: 30).
Helper-B [ ]
Source-A [ 1 ]
Target-C [ 5 4 3 2 ]
Move disk - 1 from Source-A  to Target-C  (count of moves: 31).
Source-A [ ]
Helper-B [ ]
Target-C [ 5 4 3 2 1 ]
```

Program zwraca błędy, gdy podane są złe dane.

In [15]:

```python
def main():
    number = -1
    if type(number) != int:
        raise TypeError('The amount of discs has to be an integer')
    a = Tower_Stack("Source-A ", number)
    b = Tower_Stack("Helper-B ")
    c = Tower_Stack("Target-C ")
    show_hanoi(a, b, c)
    hanoi_tower(number, a, b, c)
```

```
if __name__=="__main__":
    main()
```

```
Source-A [ ]
Helper-B [ ]
Target-C [ ]
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-15-7610a57a1884> in <module>
     10
     11 if __name__=="__main__":
---> 12     main()

<ipython-input-15-7610a57a1884> in main()
      7     c = Tower_Stack("Target-C ")
      8     show_hanoi(a, b, c)
----> 9     hanoi_tower(number, a, b, c)
     10
     11 if __name__=="__main__":

<ipython-input-14-8ea65e8e5697> in hanoi_tower(n, source, helper, target)
     19     global move
     20     if n < 0:
---> 21         raise ValueError('The amount of discs has to be positive')
     22     if n >= 1:
     23         hanoi_tower(n - 1, source, target, helper)

ValueError: The amount of discs has to be positive
```

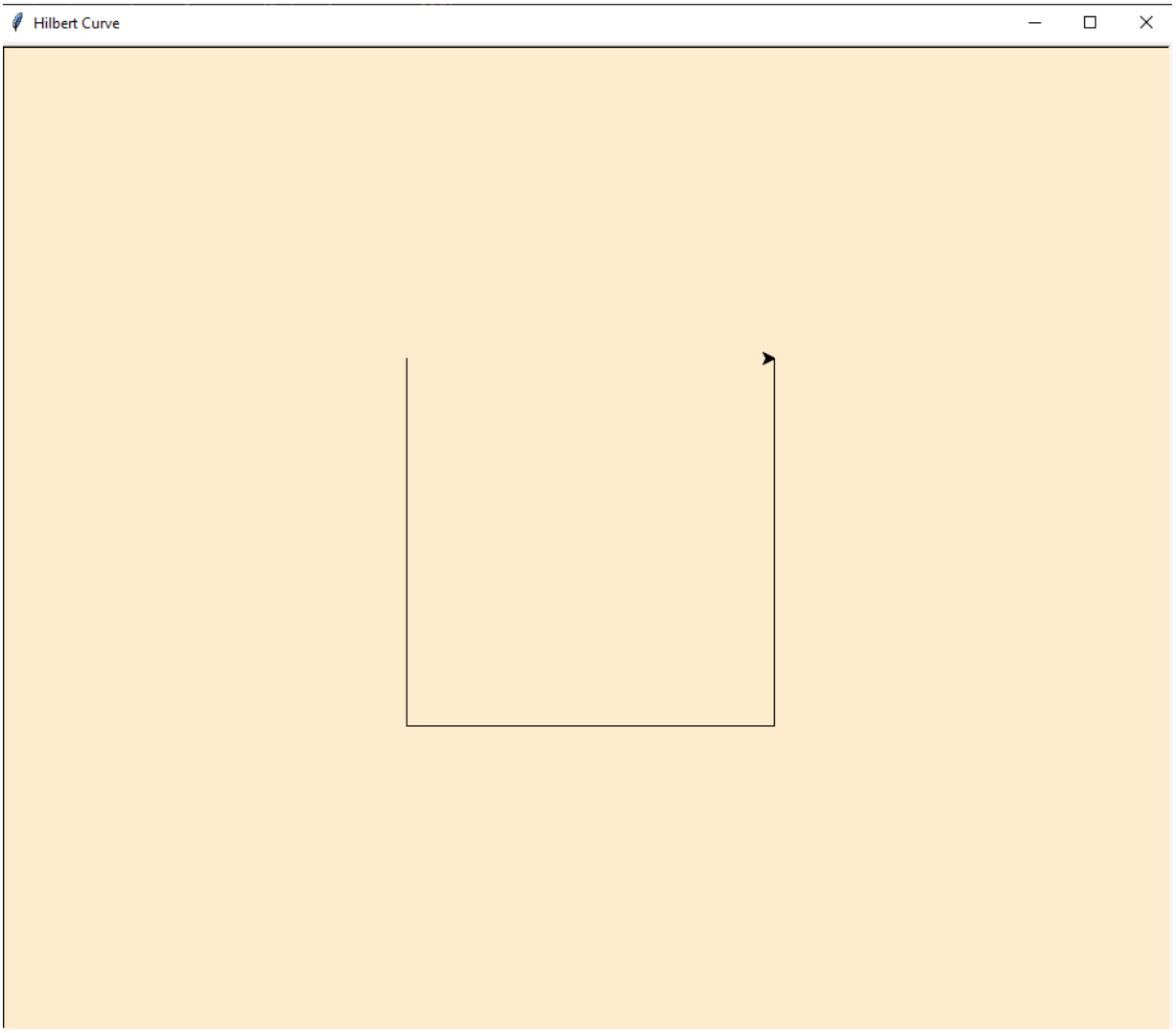
In [17]:
```
def main():
    number = "-1"
    if type(number) != int:
        raise TypeError('The amount of discs has to be an integer')
    a = Tower_Stack("Source-A ", number)
    b = Tower_Stack("Helper-B ")
    c = Tower_Stack("Target-C ")
    show_hanoi(a, b, c)
    hanoi_tower(number, a, b, c)

if __name__=="__main__":
    main()
```

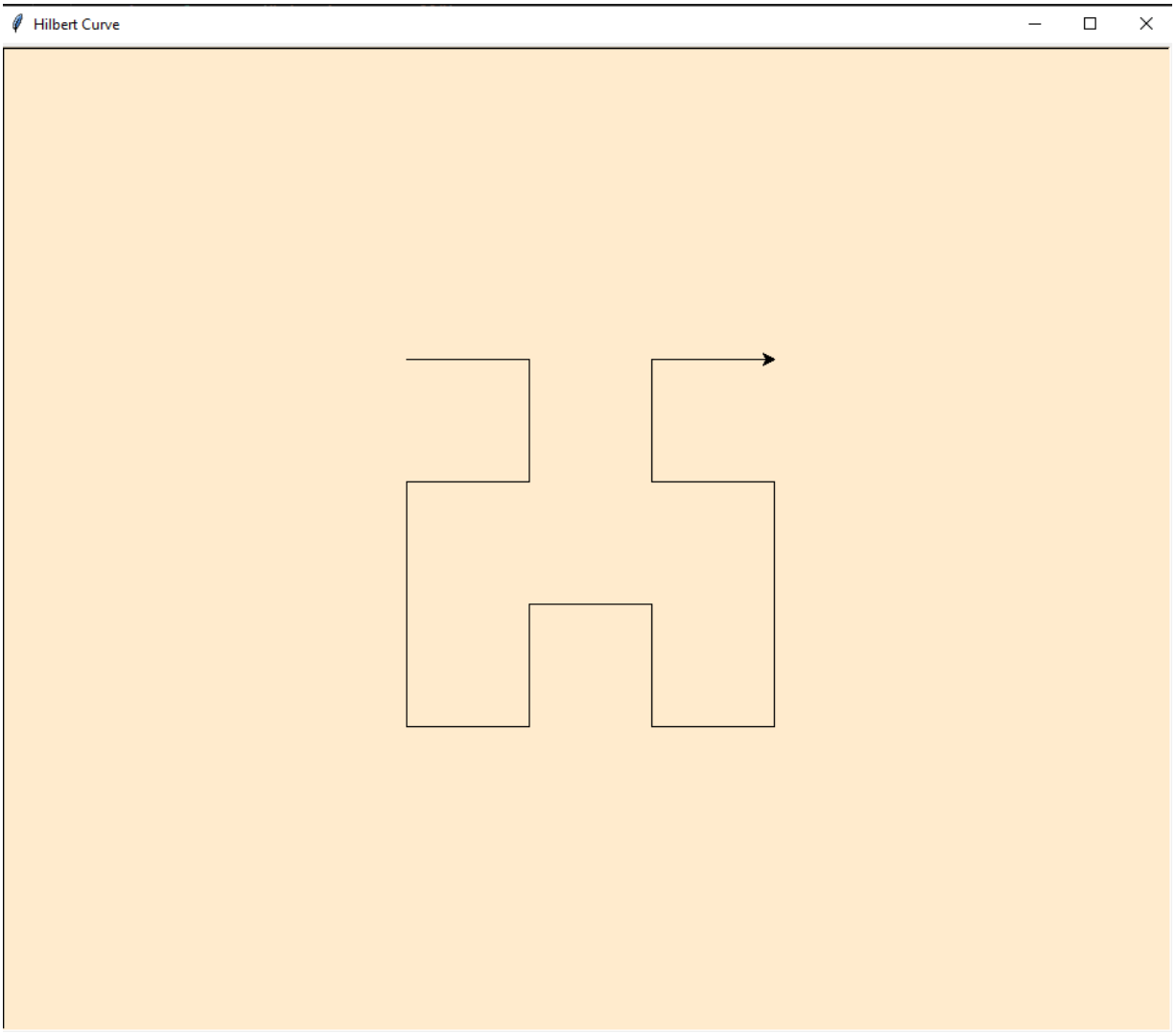
```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-17-2976ca320c2e> in <module>
     10
     11 if __name__=="__main__":
---> 12     main()

<ipython-input-17-2976ca320c2e> in main()
      2     number = "-1"
      3     if type(number) != int:
----> 4         raise TypeError('The amount of discs has to be an integer')
      5     a = Tower_Stack("Source-A ", number)
      6     b = Tower_Stack("Helper-B ")

TypeError: The amount of discs has to be an integer
```

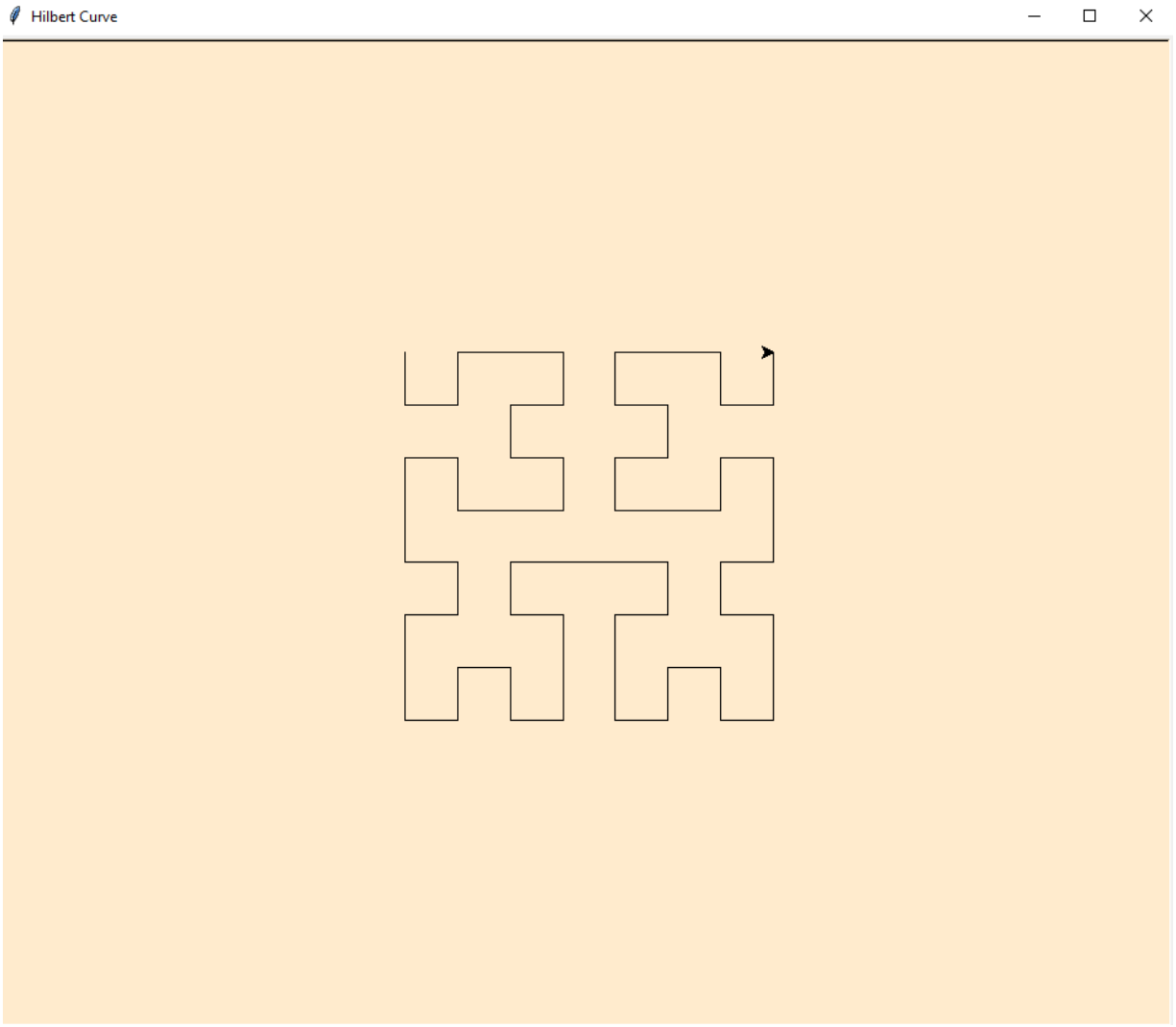
Zadanie 3

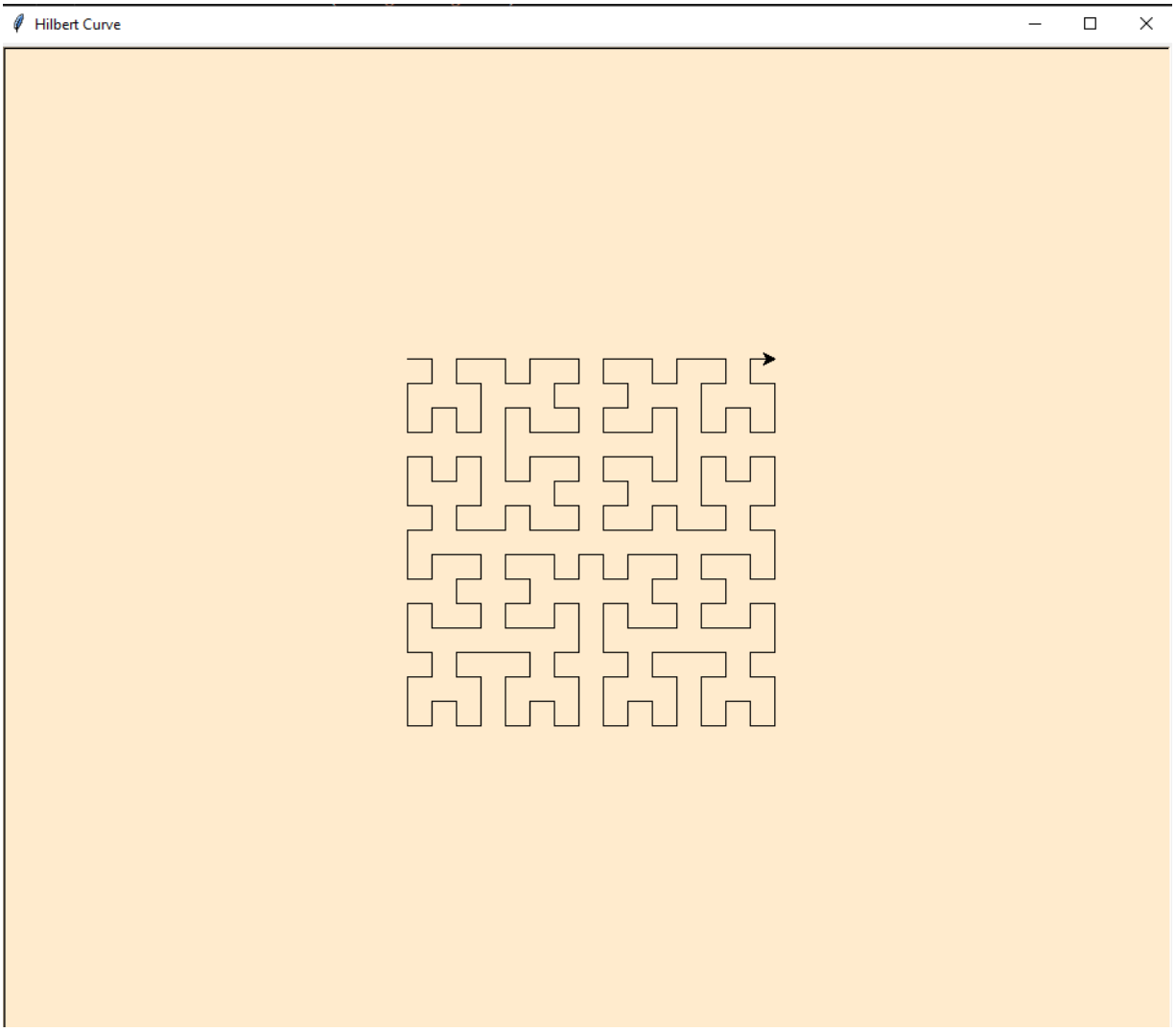
W tym zadaniu naszym celem było narysowanie Krzywej Hilberta, przy pomocy modułu turtle. Napisaliśmy funkcję hilbert_curve(), której zadaniem jest poruszanie się naszego żółwia według określonego schematu. Na początku obracamy naszym żółwiem w prawo o kąt 90°. Wywołujemy funkcję dla $level - 1$, kąta równego -90° oraz dla określonej długości odcinka krzywej, która wynosi $\frac{size}{2^{level}-1}$, gdzie size jest obszarem kwadratu do wypełnienia, a level poziomem naszej rekurencji. Poruszamy się do przodu o tą długość, obracamy w lewo i powtarzamy cały algorytm dla $level - 1$, kąta 90° i długości odcinka. Żółw porusza się do przodu. Ponownie wywołujemy ten sam algorytm, a następnie obracamy żółwia w lewo i przesuwamy do przodu. Na końcu wywołujemy znów algorytm dla $level - 1$ oraz kąta równego -90° i obracamy w prawo. Funkcja draw_hilbert_curve() rysuje i wyświetla nam naszą krzywą.

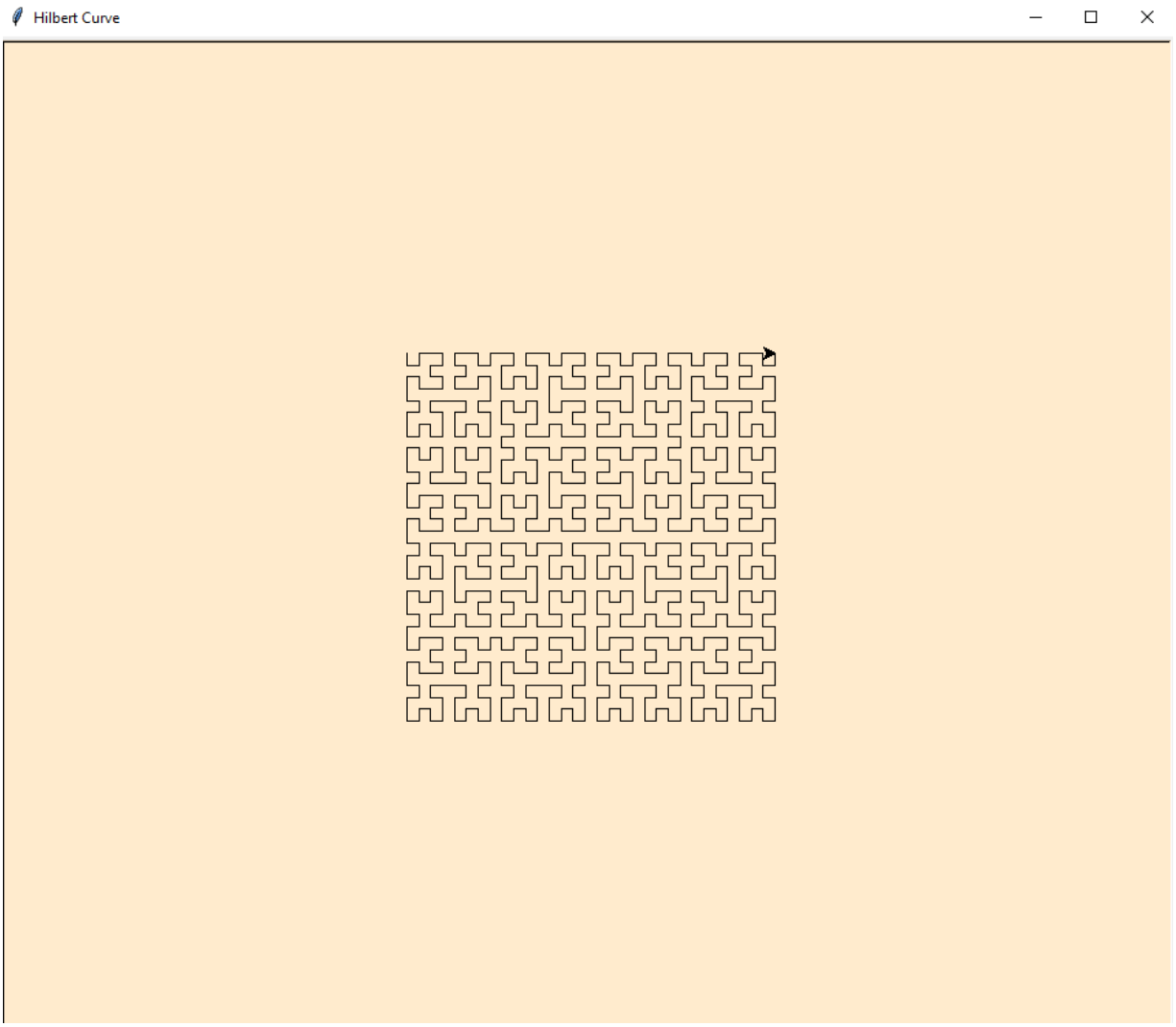
Poniżej przedstawiamy wywołania naszej funkcji draw_hilbert_curve() dla poziomu rekurencji(level) równego kolejno 1,2,3,4,5.




Hilbert Curve


Hilbert Curve


Hilbert Curve



Program zwraca błędy, gdy podane są złe dane.

In [65]:
```
draw_hilbert_curve(0)
```

```
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
<ipython-input-65-b303792f742f> in <module>
----> 1 draw_hilbert_curve(0)

<ipython-input-63-f5d7b35d8b03> in draw_hilbert_curve(level, size)
     23         raise TypeError("Wrong data given")
     24     if level == 0:
---> 25         raise ZeroDivisionError("Wrong data given")
     26     if size <= 0:
     27         raise ValueError("Size is too small")

ZeroDivisionError: Wrong data given
```

In [66]:
```
draw_hilbert_curve(-1)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-66-0fca6271c14d> in <module>
----> 1 draw_hilbert_curve(-1)

<ipython-input-63-f5d7b35d8b03> in draw_hilbert_curve(level, size)
     32     goto(-size/2, size/2)
     33     pendown()
---> 34     hilbert_curve(level, 90, size/(2 ** level - 1))
     35     mainloop()
     36

<ipython-input-63-f5d7b35d8b03> in hilbert_curve(level, angle, step)
      3 def hilbert_curve(level, angle, step):
      4     if level < 0:
----> 5         raise ValueError("The value of level of the recursion must be positive number")
```

```
      6     elif level == 0:
      7         return

ValueError: The value of level of the recursion must be positive number
```

```
In [64]: draw_hilbert_curve("a")
```

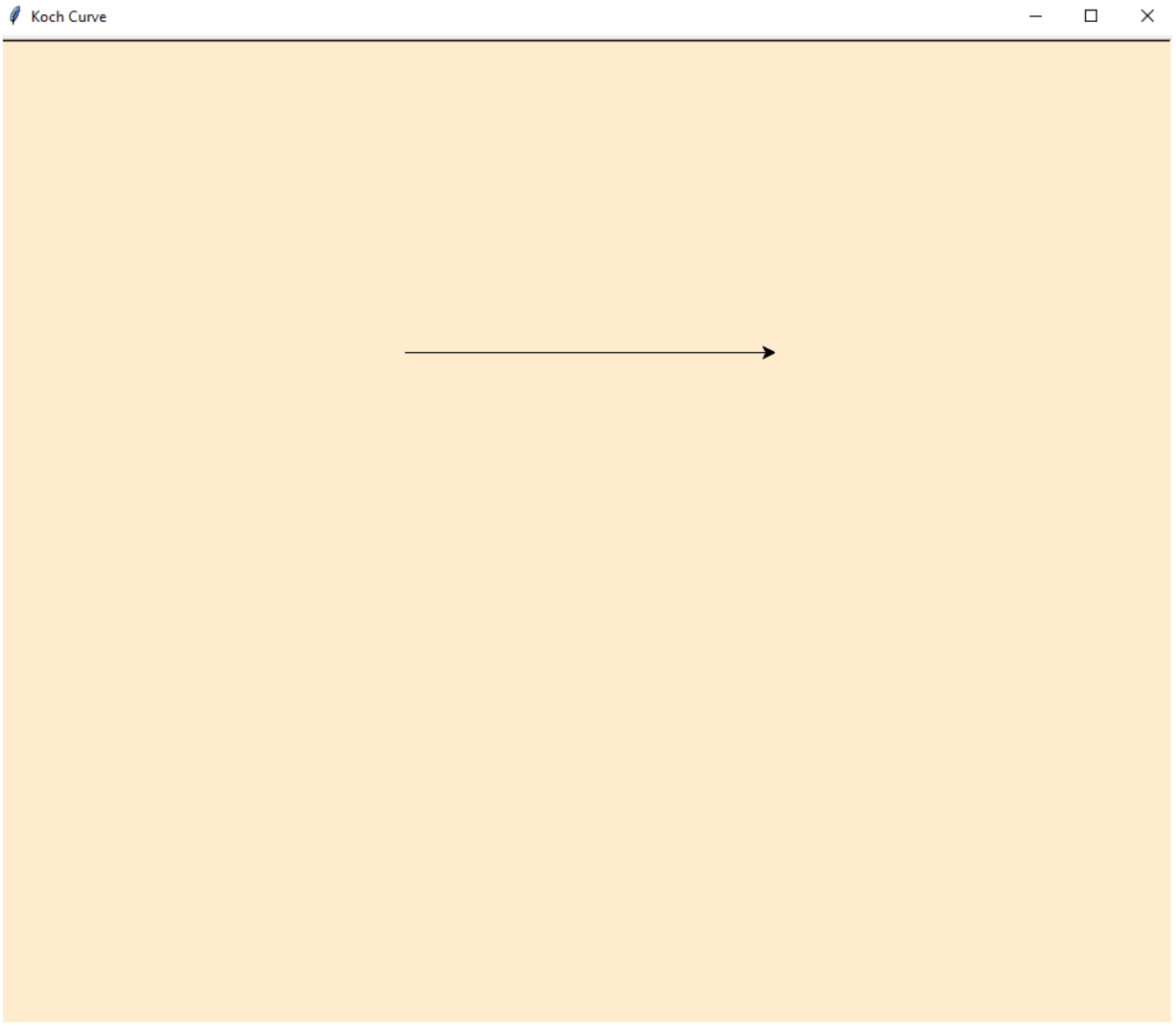
```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-64-6ffcd357481b> in <module>
----> 1 draw_hilbert_curve("a")

<ipython-input-63-f5d7b35d8b03> in draw_hilbert_curve(level, size)
     21 def draw_hilbert_curve(level = 3, size = 300):
     22     if not (isinstance(level, int) and isinstance(size, int)):
---> 23         raise TypeError("Wrong data given")
     24     if level == 0:
     25         raise ZeroDivisionError("Wrong data given")

TypeError: Wrong data given
```

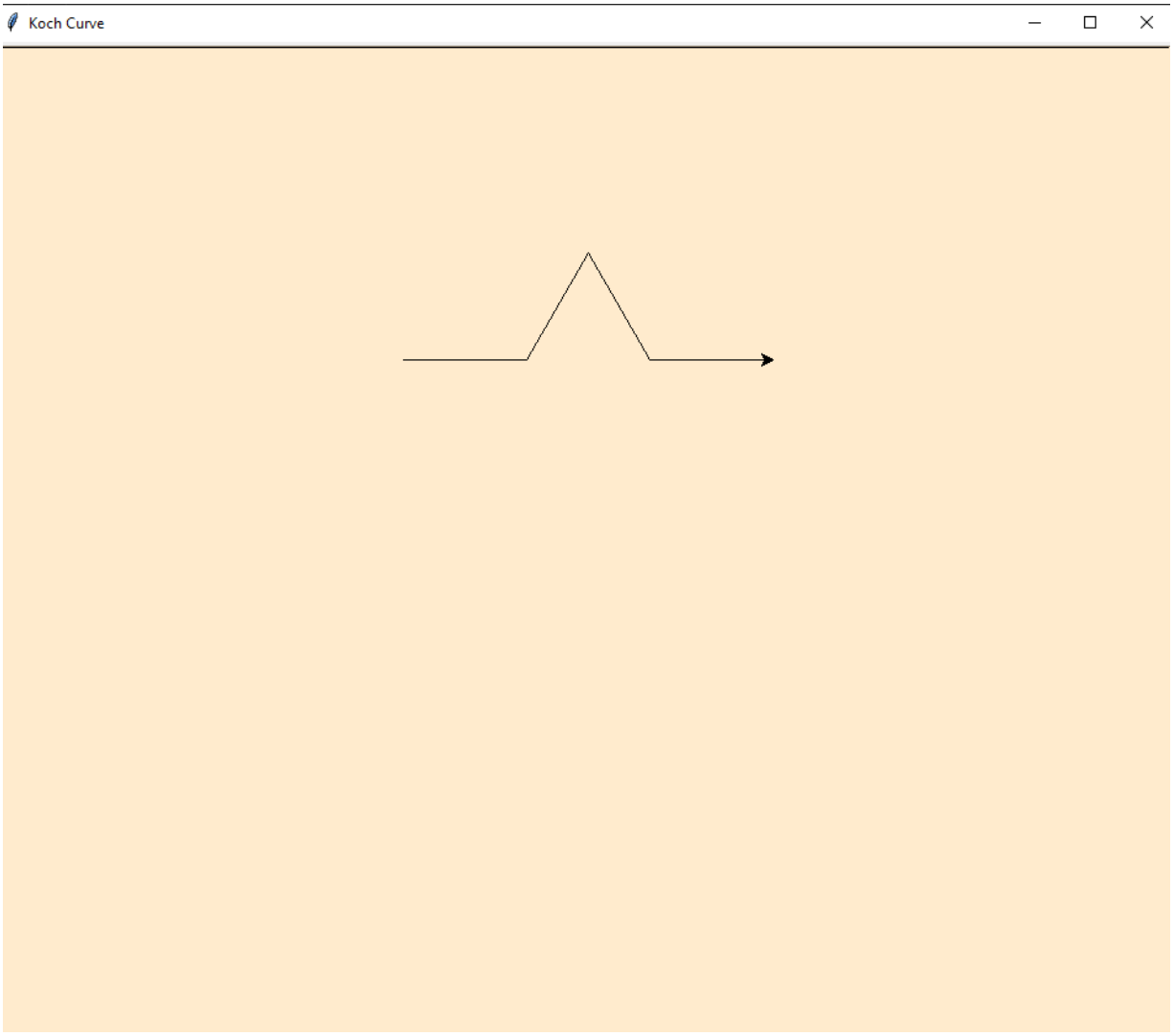
Zadanie 4

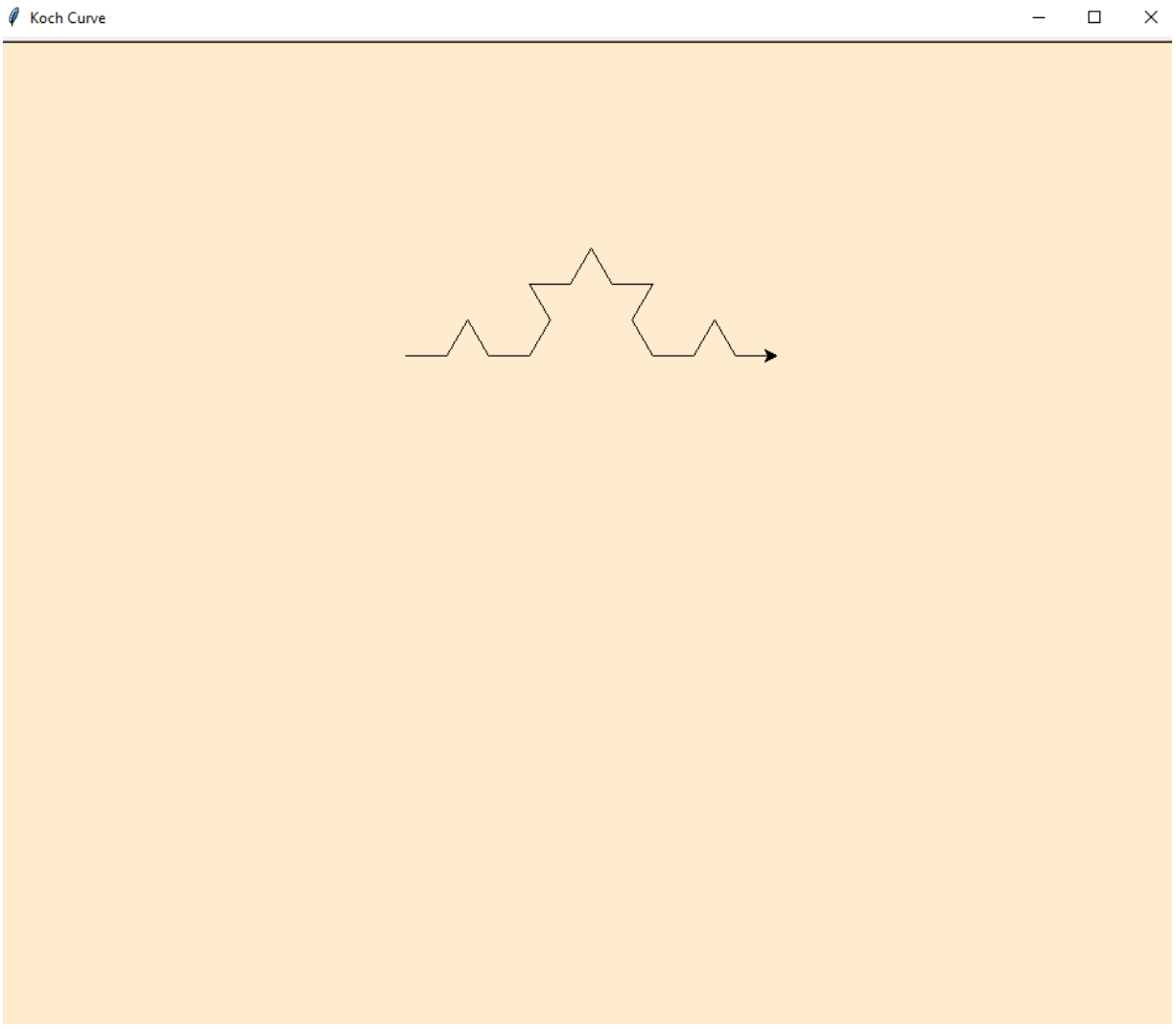
Tutaj naszym zadaniem było narysowanie Płatka Kocha, przy pomocy modułu turtle. Aby tego dokonać stworzyliśmy funkcję koch_curve(), która jest odpowiedzialna za poruszanie żółwiem, w celu stworzenia Krzywej Kocha. Krzywa ta, powstaje z odcinka, poprzez podzielenie go na 3 części i zastąpienie środkowej "ząbkiem" (o ramieniu długości równej $\frac{1}{3}$ odcinka stąd mamy nasze $\frac{step}{3}$). Następnie wywołujemy naszą funkcję dla stopnia $level - 1$. Obracamy w lewo o 60°. Powtarzamy algorytm i obracamy w prawo o 120°. Znowu wykonujemy ponownie algorytm. Na końcu pozostaje nam obrócić znowu o 60° w lewo i kolejny raz powtórzyć algorytm.

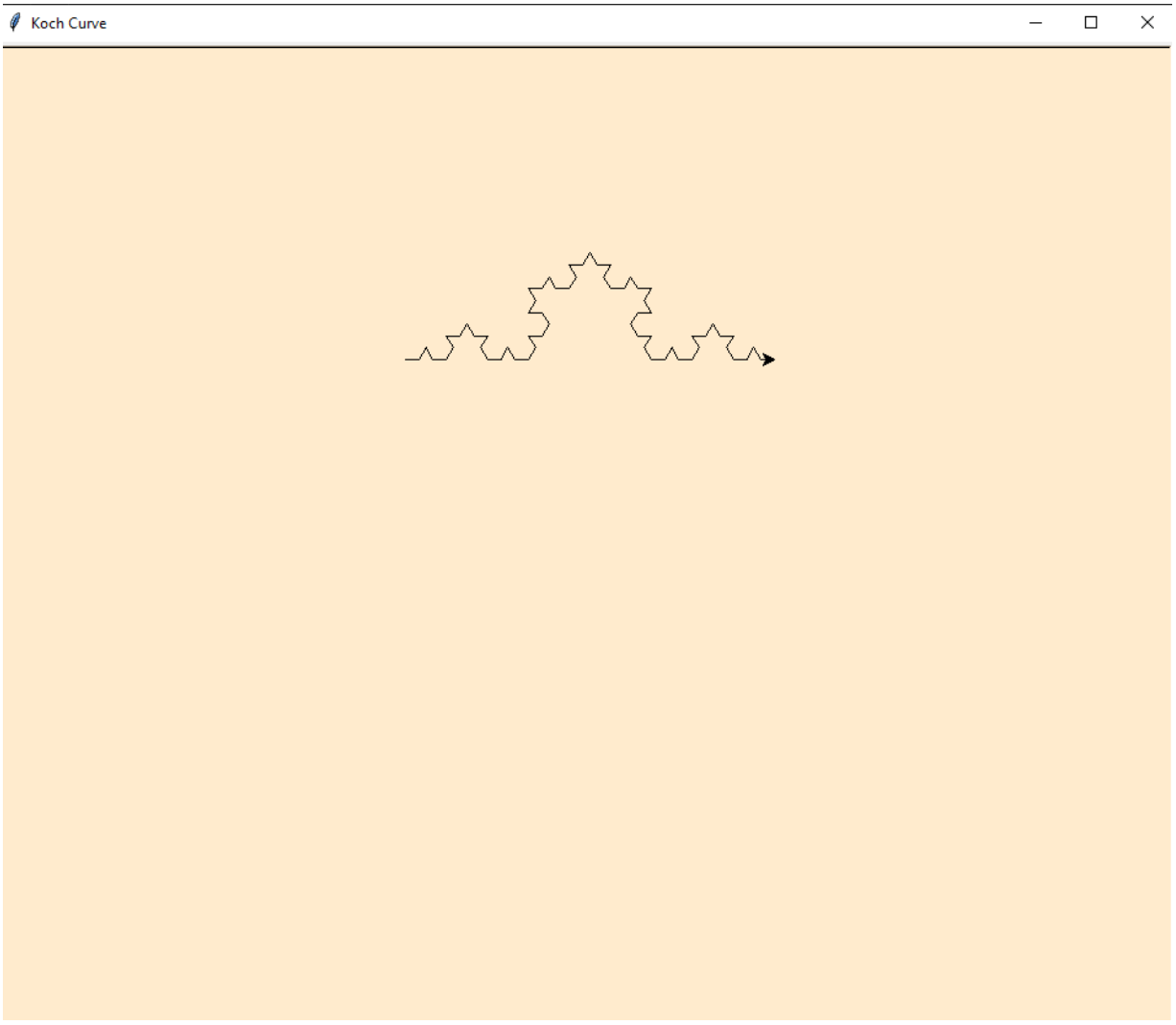
- Krzywa Kocha w kroku zerowym (level = 0) jest odcinkiem.
- Krzywa Kocha w kroku pierwszym (level = 1) została podzielona na 3 równe części, a środkową zastąpiły dwa odcinki długości $\frac{step}{3}$, nachylone względem niej pod kątem 60°. Teraz nasza krzywa zawiera 4 odcinki, każdy równy $\frac{step}{3}$.
- Krzywa Kocha w kroku drugim (level = 2) ponownie została podzielona - każdy z 4 odcinków został podzielony na 3 części, a środkową znów zastąpiono dwoma odcinkami. Teraz zawiera już ona 16 odcinków, każdy długości $\frac{step}{9}$.
- W kolejnym kroku (level = 3) powstanie 64 odcinków, każdy długości $\frac{step}{27}$, itd. dla coraz większych powtórzeń.

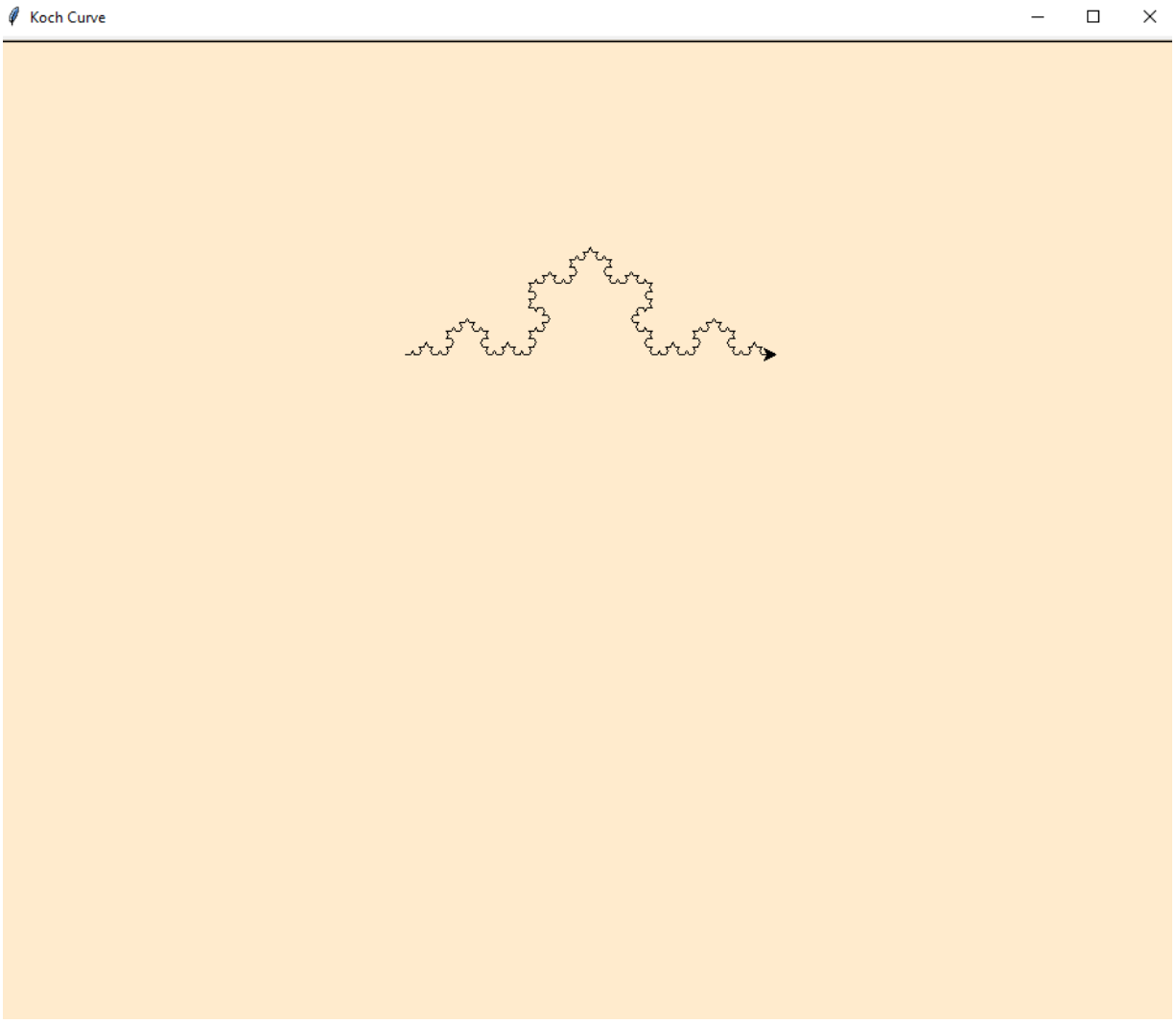
Poniżej przedstawiamy wywołanie funkcji draw_koch() dla poziomu rekurencji(level) odpowiednio równemu 0,1,2,3,4.


Koch Curve

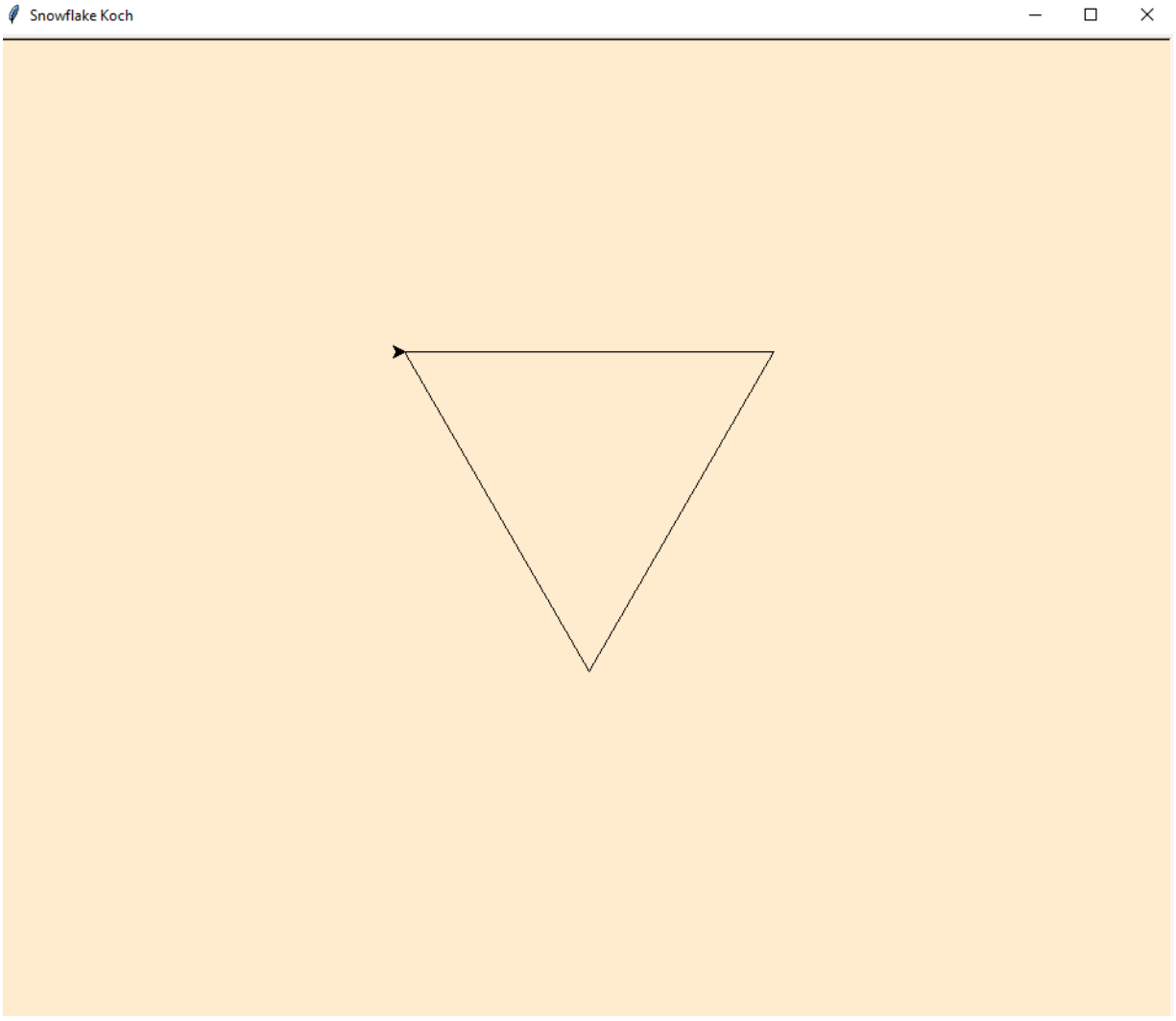

Koch Curve

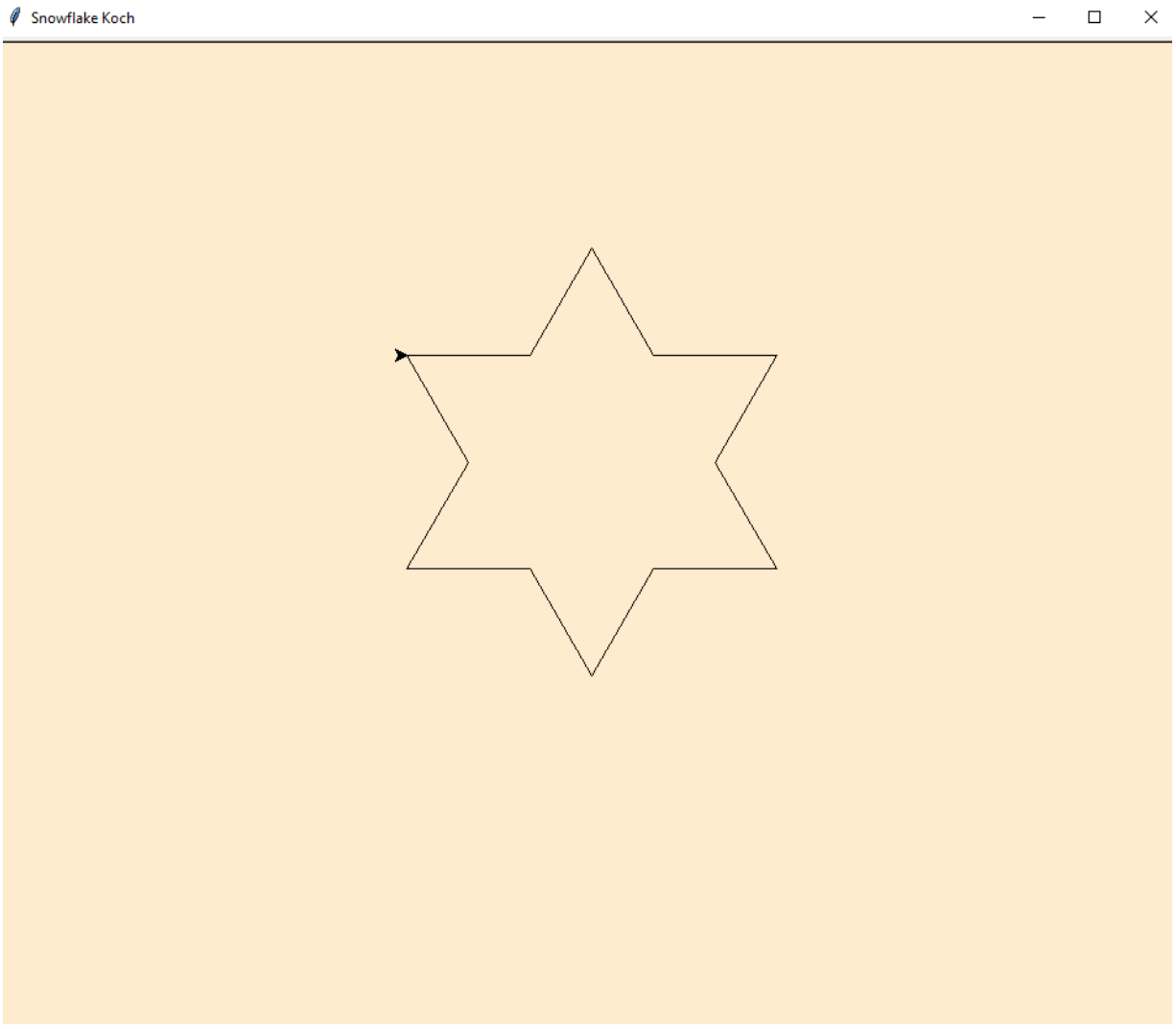

Koch Curve

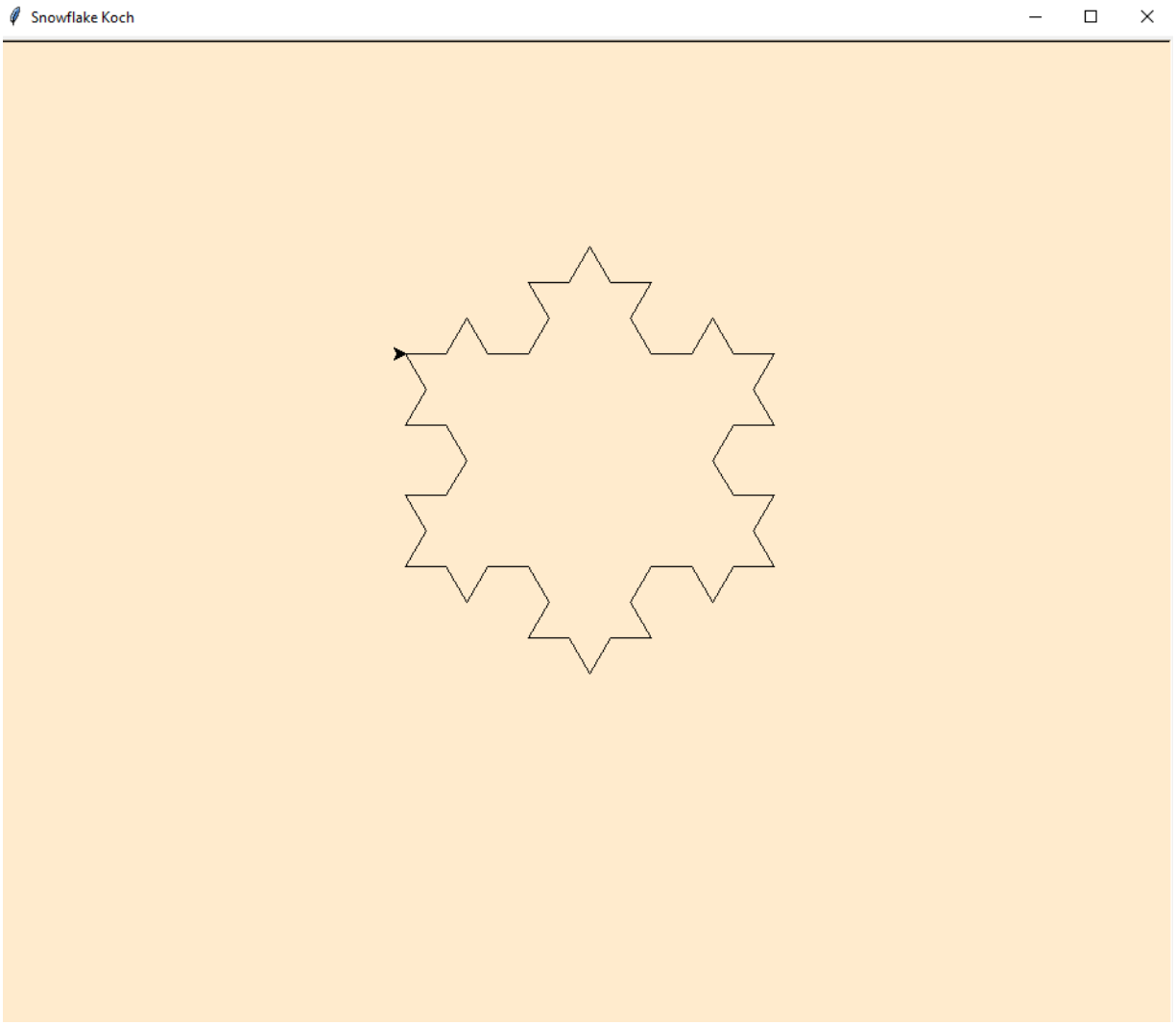

Koch Curve

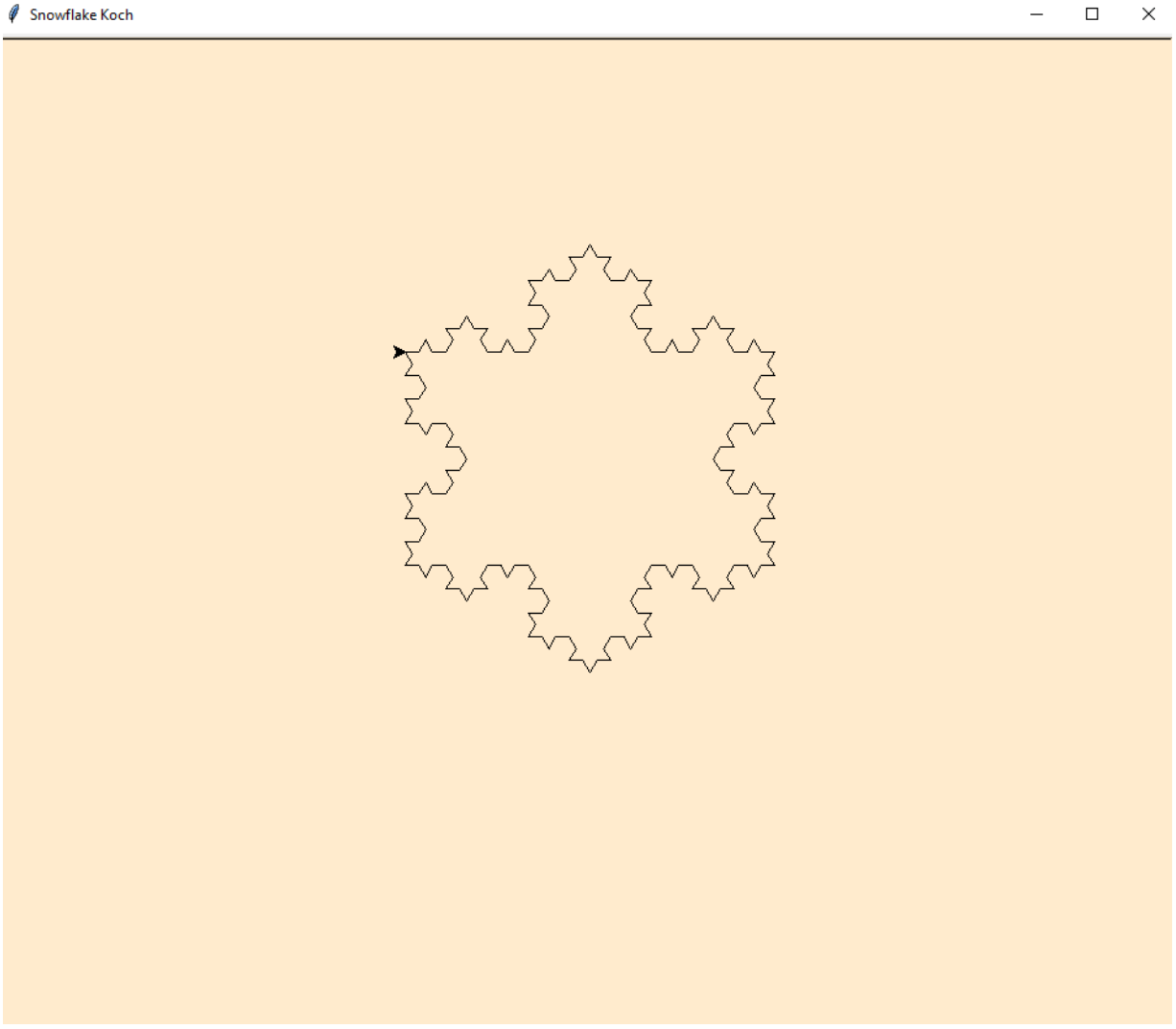


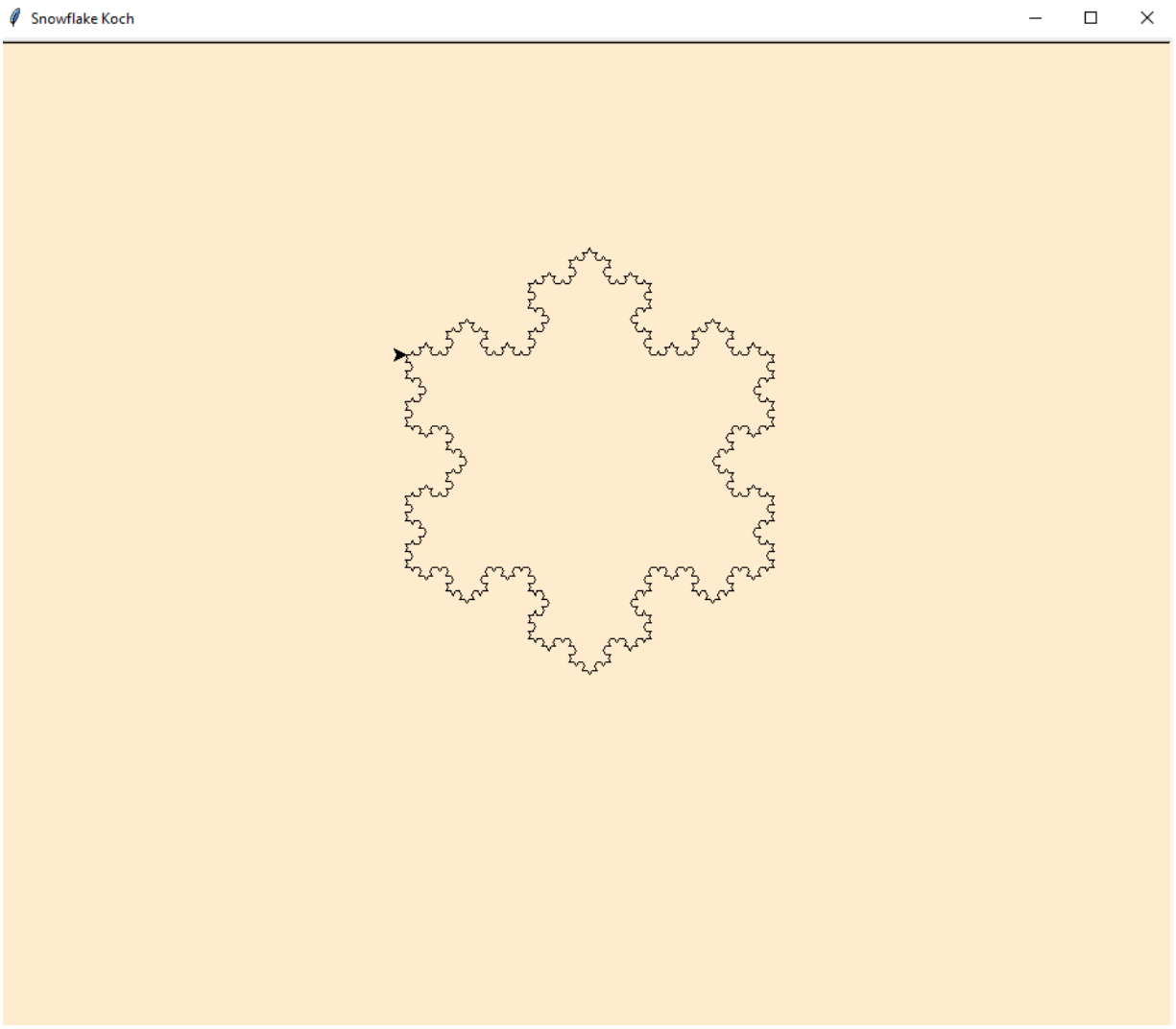
Płatek Kocha powstaje poprzez narysowanie trzech Krzywych Kocha w taki sposób, że razem tworzą jedną figurę. Dlatego w funkcji draw_snowflake_koch() w pętli wywołujemy trzykrotnie koch_curve() oraz obracamy w prawo o 120°. Poniżej znajdują się wywołania dla poziomu rekurencji(level) równemu odpowiednio 0,1,2,3,4.


Snowflake Koch


Snowflake Koch


Snowflake Koch


Snowflake Koch



Program zwraca błędy, gdy podane są złe dane.

In [2]:
```
draw_snowflake_koch(-1)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-2-59e148e4f19f> in <module>
----> 1 draw_snowflake_koch(-1)

<ipython-input-1-1e65d84c1612> in draw_snowflake_koch(level, size)
     43     pendown()
     44     for i in range(3):
---> 45         koch_curve(level, 60, size)
     46         right(120)
     47     mainloop()

<ipython-input-1-1e65d84c1612> in koch_curve(level, angle, step)
      3 def koch_curve(level, angle, step):
      4     if level < 0:
----> 5         raise ValueError("The value of level of the recursion must be positive number")
      6     elif level == 0:
      7         forward(step)

ValueError: The value of level of the recursion must be positive number
```

In [3]:
```
draw_snowflake_koch('a')
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-3-263514b95e78> in <module>
----> 1 draw_snowflake_koch('a')

<ipython-input-1-1e65d84c1612> in draw_snowflake_koch(level, size)
     33 def draw_snowflake_koch(level = 3, size = 300):
     34     if not (isinstance(level, int) and isinstance(size, int)):
---> 35         raise TypeError("Wrong data given")
```

```
     36     if size <= 0:
     37         raise ValueError("Size is too small")

TypeError: Wrong data given
```

In [2]:
```
draw_snowflake_koch(3, -10)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-2-dbde20e69029> in <module>
----> 1 draw_snowflake_koch(3, -10)

<ipython-input-1-8fb67bac0ce3> in draw_snowflake_koch(level, size)
     35         raise TypeError("Wrong data given")
     36     if size <= 0:
---> 37         raise ValueError("Size is too small")
     38     title("Snowflake Koch")
     39     bgcolor("#FFEBCD")

ValueError: Size is too small
```

Linki

https://github.com/github-kamilk/AiSD/tree/main/Lista_5