როგორ ავაშენოთ ჯულიას ნაკრები

დღეს მინდა განვიხილო ფრიად საინტერესო თემა, ფრაქტალების აგების გზები **L-სისტემების** გამოყენებით, რომელიც შემოთავაზებული იყო უნგრელი ბიოლოგის **Aristid Lindenmayer**-ის მიერ **1968** წელს და განმეორებითი ფუნქციური სისტემების გამოყენებით, რომლებიც შეიმუშავეს **ჯონ ჰაჩინსონმა** და **მაიკლ ბარნსლიმ 1981** წელს. მაგრამ არსებობს ფრაქტალების აგების კიდევ ერთი საკმაოდ ცნობილი მეთოდი, რამაც, ფაქტობრივად, დაიწყო ამ სამეცნიერო მიმართულების აღზევება - ფრაქტალური მრუდების ფორმირება, როგორც დინამიური სისტემების მიმზიდველი.

1917-1919 წლებში ფრანგი მათემატიკოსი **გასტონ ჯულია** (**1893-1978**) და **პიერ ფატუ** (**1878-1929**) სწავლობდნენ რთული ცვლადი ფუნქციების გამეორებას. მათ შეისწავლეს ხარისხის მრავალწევრების ქცევა $n \geq 2$ ტიპი:

$$f(z) = a_n z^n + a_{n-1} z^{n-1} + \ldots + a_1 z + a_0, \quad a_n \neq 0$$

სადაც $z, a_0, \ldots, a_n$ - რთული რიცხვებია. მაგალითად, უმარტივეს შემთხვევაში, შეგიძლიათ აიღოთ ფორმის ფუნქცია:

$$f(z) = z^2$$

და ნახეთ, როგორ იქცევა ის გამეორებების დროს:

$$f_1(z) = \left[f_0(z) \right]^2 = \left(z^2 \right)^2$$

$$\ldots$$

$$f_n(z) = z^{2n}$$

ეს მოგვაგონებს **SIF**-ს ერთი შეკუმშვის რუქით. მაგრამ ასეთ მარტივ ვერსიაშიც კი შეგვიძლია მივიღოთ საოცრად ლამაზი სურათები. ჩვენ ამაში მალე დავრწმუნდებით.

მოდით სანამ გადავიდოთ უშუალოდ ჯულიას ნაკრების აშენების პროცეზე, მანამდე მინდა მოკლედ განვიხილი რაში მდგომარეობს **L-სისტემა** და **ფრაქტალი**, რა კავშირებია მათ შორის.

L-სისტემა (**Lindenmayer**-სისტემა) არის ალგორითმი, რომელიც გამოიყენება **ფოთლის**, **ხეებისა** და სხვა ბუნებრივი სტრუქტურების მოდელირებისთვის. ეს არის წარმოშობის რეცეპტი, რომელიც გამოიყენება თვითწარმოებაში ანიმაციებისთვის. **L-სისტემა** იწყება იმიტირებული სიგნალით, რომელსაც მიერთდება წესი, რაც უზრუნველყოფს ახალი ფორმის შექმნას, ხშირად ბუნებრივი სიმეტრიისა და ფორმების საფუძველზე.

ფრაქტალი არის გეომეტრიული ფორმა, რომელიც ხასიათდება თვითგავრცელების თვისებით — ანუ მისი სტრუქტურა იგივე რჩება სხვადასხვა სიდიდეზე ან მასშტაბზე. ფრაქტალის რუკა ხშირად გამოიყენება ბუნებაში არსებული სტრუქტურების, მაგალითად, **ტალღების**, **მთების**, თუ **მცენარეების** მოდელირებისთვის.

კავშირი:

L-სისტემები ხშირად გამოიყენება ფრაქტალების გენერირებაში, რადგან მათი გამოთვლითი წესები ხშირად გამოიწვევენ თვითგავრცელებას, რომელიც აუცილებელია ფრაქტალებისთვის. **L-სისტემებით** ხშირად მიიღება მრგვალი ან სხვა კომპლექსური ფორმები, რომლებიც ერთმანეთში იცვლებიან, რაც არის ერთ-ერთი თვისება, რომელიც ფრაქტალებს ახასიათებს.

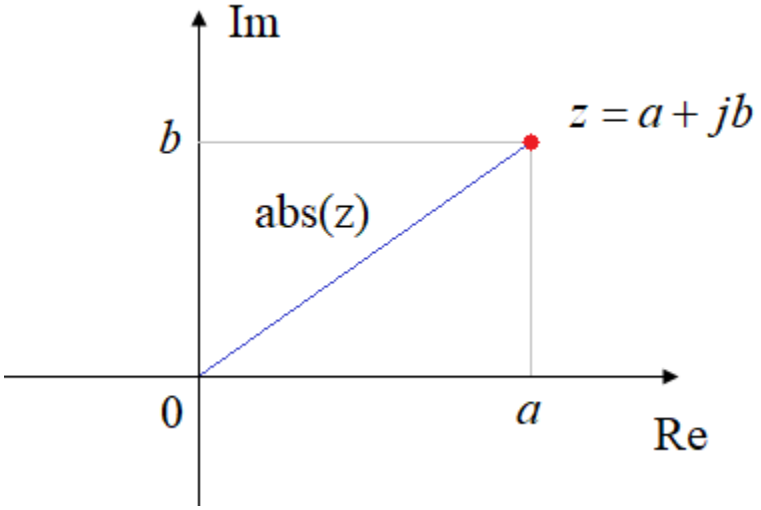
მოდით გავაგრძელოთ ჩვენი თემის განხილვა: **როგორ ავაშენოთ ჯულიას ნაკრები?**

როგორ შეუძლია რთული ცვლადის ამ ფუნქციას ფრაქტალური მრუდების გენერირება? ამ შემთხვევაში ჩვენ შეგვიძლია მივიღოთ მხოლოდ ორი მიმზიდველი: **პირველი** ერთეულ წრეზე წერტილების სიმრავლის სახით; **მეორე** არის წერტილი ნული.

რატომ ზუსტად ეს მიმზიდველები? თუ რომელიმე თქვენგანმა არ იცის, ფორმის ნებისმიერი რთული რიცხვი:

$$z = a + jb$$

სადაც **a** არის რიცხვის რეალური ნაწილი; **b** – წარმოსახვითი ნაწილი; $j = \sqrt{-1}$ - წარმოსახვითი ერთეული. შეიძლება წარმოდგენილი იყოს წერტილის სახით კომპლექსურ სიბრტყეზე:



ამ ფიგურიდან ნათლად ჩანს, რომ წერტილები, რომლებიც გამეორებების დიდი რაოდენობით არ მიდიან უსასრულობამდე:

$$p = \lim_{n \to \infty} z^{2n} < \infty$$

ეს არის ის, რომლებისთვისაც **z** კომპლექსური რიცხვის მოდული არის ერთზე ნაკლები ან ტოლი:

$$|z| \leq 1$$

მაგალითად, თუ რიცხვის აბსოლუტური მნიშვნელობა არის **0,7**, მაშინ მისი გამუდმებით კვადრატში მივიღებთ უფრო და უფრო მცირე მნიშვნელობებს, სანამ ზღვარში ნულს მივაღწევთ და ერთი ნებისმიერი ძალაუფლებისთვის ყოველთვის ერთია:

$$\begin{cases} 0,7^{2n} \to 0 \\ 1^{2n} = 1 \end{cases}$$

აქედან გამომდინარე მივიღებთ, რომ ნებისმიერი რთული წერტილი, რომელიც მდებარეობს ერთეულ წრეზე, რჩება მასზე რთული ცვლადის ფუნქციის ნებისმიერი რაოდენობის გამეორებისთვის და იგივე ნულთან

ერთად. ანუ, მიმზიდველი შედგება "**სტაბილური**" წერტილებისგან, რომლებიც არ ცვლიან თავიანთ პოზიციას ფუნქციის გამეორებისას.

კომპლექსურ სიბრტყეზე წერტილების სიმრავლე, რომლებიც არ მიდიან უსასრულობამდე n გამეორებების რაოდენობის გაზრდისას ქმნიან ჯულიას სიმრავლეებს.

თუმცა, წრეების დახატვა რთული ცვლადის ფუნქციის განმეორებით არ არის საუკეთესო რამ. **კიდევ რა შეუძლია ამ ფორმულებს?**

მოდით გავაკეთოთ რაიმე მარტივი - დავამატოთ რთული ტერმინი და განვიხილოთ ფორმის ფუნქცია:

$$f(z) = z^2 + c$$

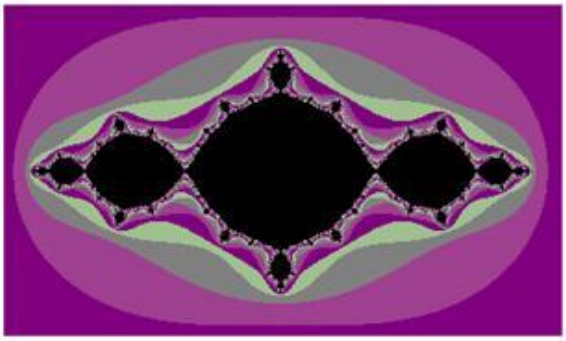
როგორც ჩანს, **რა შეიძლება შეცვალოს ამ ტერმინმა?** მაგრამ ეს ფაქტიურად არღვევს წრეს და იწვევს ბევრ უცნაურ ფრაქტალ ფორმას! სანამ მომავალ სასწაულს შეხვდებით, უნდა გავარკვიოთ რა ღირებულებებით

$$abs(f(z)) = |f(z)| = \sqrt{a^2 + b^2}$$

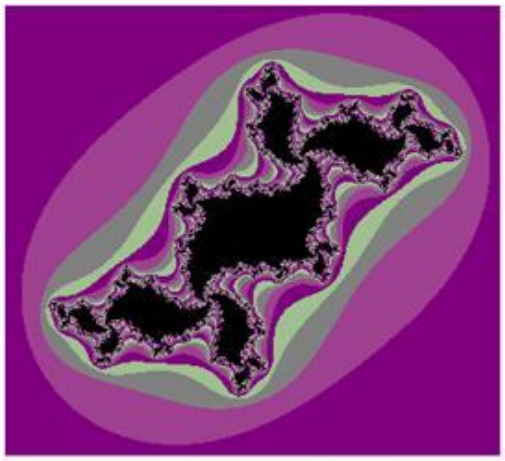
ფუნქცია არ გადავა უსასრულობამდე გამეორებების დროს. როგორ ფიქრობთ, ეს არის იგივე მნიშვნელობა **1**, **როგორც წინა შემთხვევაში?** არა, აქ უნდა ავიღოთ მნიშვნელობა **2**. **რატომ?** არსებობს თეორემა, რომელიც ამტკიცებს, რომ ნებისმიერი რთული რიცხვი ასეთ ფუნქციაში,

$$|f(z)| > 2$$

აშკარად წავა უსასრულობამდე. ეს ნიშნავს, რომ ის არ ეკუთვნის ჯულიას კომპლექტს. (ამ თეორემის ფორმულირება და დადასტურება შეგიძლიათ იხილოთ ფრაქტალური პროცესების ნებისმიერ პრაქტიკულ სახელმძღვანელოში).



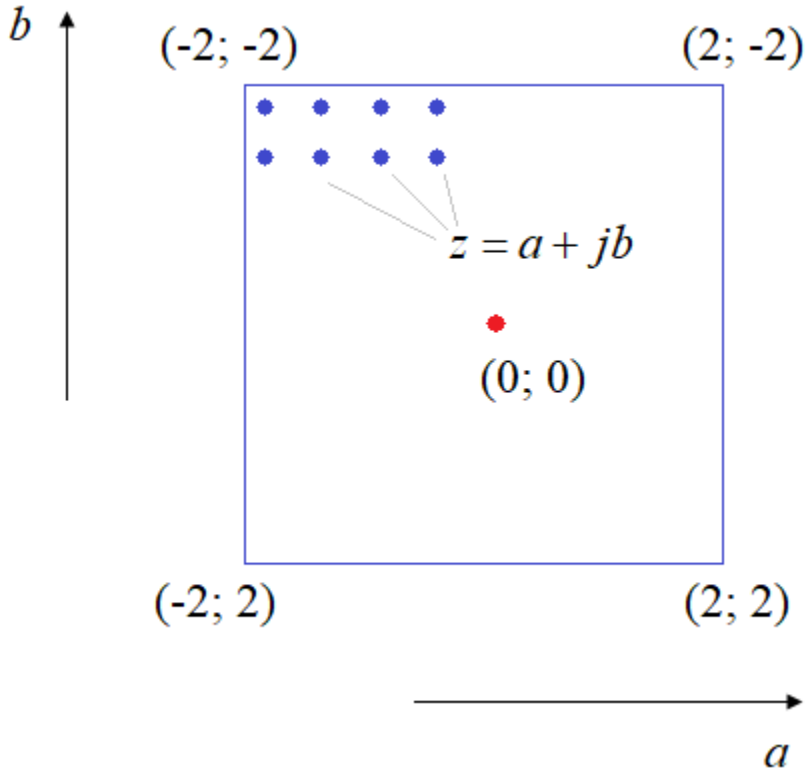
$c = -1$



$c = -0,2 + 0,75j$

მაგალითად, ასე გამოიყურება ჯულიას ნაკრები ტერმინის სხვადასხვა მნიშვნელობებისთვის **c** წერტილები, რომლებიც შეღებილია შავად, არის წერტილები ნაკრებში, ხოლო დანარჩენი წერტილები ნაკრების გარეთ.

დადგა დრო, რომ ეს ქმნილება საკუთარი ხელით შევქმნათ. ალგორითმების იდეა, რომლებიც ქმნიან ასეთ კომპლექტს, შემდეგია.



კომპლექსურ სიბრტყეზე ჩვენ ვსკანირებთ წერტილების ერთობლიობას, მაგალითად, დიაპაზონში [-2; 2] წარმოსახვითი და რეალური ღერძების გასწვრივ. თითოეული წერტილი არის რთული რიცხვი:

$$z = a + jb$$

ჩვენი ამოცანაა დავადგინოთ, ეკუთვნის თუ არა ეს პუნქტი ჯულიას კომპლექტს. ამისათვის ჩვენ ვაკეთებთ ფუნქციის **n** გამეორებას:

$$f(z) = z^2 + c$$

სადაც **c** არის მოცემული რთული რიცხვი (**მუდმივი**). თუ აღმოჩნდება, რომ **n** გამეორების შემდეგ მიღებული რთული რიცხვის მოდული ორზე ნაკლებია, მაშინ ასეთი წერტილი უნდა მიენიჭოს ჯულიას სიმრავლეს:

$$|f_n(z)| \leq 2$$

წინააღმდეგ შემთხვევაში, ეს არის ფონური წერტილი.

ჯულიას სეტების აშენება პითონში

მოდით განვახორციელოთ ასეთი ალგორითმი პითონში.

1. იმპორტირებული ბიბლიოთეკები:

```
import numpy as np

import matplotlib.pyplot as plt

from matplotlib.animation import FuncAnimation

import cv2

from PIL import Image

import io
```

- **numpy:** ეს ბიბლიოთეკა გამოიყენება მათემატიკური და სტრუქტურული ოპერაციებისთვის, როგორიცაა მასივების ოპერაციები და წრფივი ალგებრა. პროგრამაში გამოიყენება **numpy** ფუნქციები, როგორიცაა **np.linspace**, **np.meshgrid**, **np.zeros** და **np.abs**.

- **matplotlib.pyplot:** ეს ბიბლიოთეკა გთავაზობთ ფუნქციებს გრაფიკის და ვიზუალიზაციის შესაქმნელად. კონკრეტულად **plt.imshow** გამოიყენება გამოსახულებების დასათვალიერებლად, ხოლო **plt.savefig** - გამოსახულების შენახვისთვის.
- **matplotlib.animation.FuncAnimation:** ეს უზრუნველყოფს ანიმაციების შექმნას. მისი დახმარებით, გამოსახულება თანდათან იცვლება, რაც ქმნის ვიდეო ანიმაციას.
- **cv2: OpenCV**-ს ბიბლიოთეკა, რომელიც გამოიყენება გამოსახულებების დამუშავებისთვის და ვიდეოების შექმნისთვის. აქ იგი გამოიყენება ვიდეო ფაილების შექმნისთვის (**cv2.VideoWriter**).
- **PIL: Pillow** (**PIL**) არის გამოსახულების დამუშავების ბიბლიოთეკა. მას იყენებენ გამოსახულების დასაბეჭდად და დამუშავების პროცესში (**Image.open**, **Image.save**).
- **io:** მეხსიერებაში გამოსახულების ხატვის დამუშავება და გადასვლა **BytesIO** ობიექტით.

2. **ჯულიას ნაკრების გენერაციის ფუნქცია:**

def julia_set(c, x_min=-1.5, x_max=1.5, y_min=-1.5, y_max=1.5, width=800, height=800, max_iter=256):

- **c:** კომპლექსური რიცხვი, რომელიც განსაზღვრავს ჯულიას ნაკრებს. ეს რიცხვი განსაზღვრავს მოდიფიკაციას, რომელიც გავლენას ახდენს ნაკრების ფორმაზე.
- **x_min, x_max, y_min, y_max:** ეს მნიშვნელობები განსაზღვრავს იმ სივრცის მახლობლად, რომლის გარშემოც ჯულიას ნაკრები იქნება გამოსახული. ეს არის ფართობი, რომელშიც კომპლექსური რიცხვების მრიცხველის ზღვარი განისაზღვრება.
- **width, height:** გამოსახულების განზომილება (**სიგანე** და **სიმაღლე**), რომელსაც პროგრამა შექმნის.
- **max_iter:** მაქსიმალური რაოდენობა, რომლის განმავლობაში ჯულიას ნაკრები გამოთვლების პროცესში დაიფარება.

3. **ჯულიას ნაკრების გენერაციის პროცესი:**

```
x, y = np.linspace(x_min, x_max, width), np.linspace(y_min, y_max, height)
X, Y = np.meshgrid(x, y)
Z = X + 1j * Y
img = np.zeros(Z.shape, dtype=float)
```

- **np.linspace(x_min, x_max, width) და np.linspace(y_min, y_max, height):** ეს ფუნქციები ქმნიან ერთგვარად განაწილებულ მნიშვნელობებს **x** და **y** ღერძებზე, რაც გამოიყენება კომპლექსური რიცხვების განლაგებისთვის.
- **np.meshgrid(x, y):** ქმნის **2D** მახსოვრობას, სადაც **X** და **Y** შეიცავენ მთელ **x** და **y** ღერძებზე მდებარე მაჩვენებლებს.
- **Z = X + 1j * Y:** კომპლექსური რიცხვების შემქმნელი, სადაც **X** არის რეალური ნაწილი, ხოლო **Y** იმიჯინარული.
- **img = np.zeros(Z.shape, dtype=float):** გამოყოფს გამოსახულების შენახვისთვის მახსოვრობას.

4. **ძირითადი გამოთვლები და ნორმალიზაცია:**

```
for i in range(max_iter):
    Z = Z ** 2 + c
    mask = np.abs(Z) < 10  # მხოლოდ იმაზე ვსაუბრობთ, რაც ცოტა მეტია
    mask = mask & np.isfinite(Z)  # მხოლოდ ფინიტური მნიშვნელობები გამოიყენეთ
    img += mask
```

- **Z = Z ** 2 + c:** ეს არის ჯულიას ნაკრების ძირითადი გამოთვლა. ყოველი გამოთვლა ზემოქმედებს კომპლექსური რიცხვის ახალი მნიშვნელობის მისაღებად. თუ **Z** ძალიან დიდდება, პროცესი ვეღარ ხდება განსაზღვრული.

- **mask = np.abs(Z) < 10:** უზრუნველყოფს, რომ მხოლოდ ისეთთაგანს არ გამოვლენს სურათები, რომლებიც ძალიან დიდია (ეს არის პროცესი, რომელიც უარყოფს უსასრულო ან არვალიდურ რიცხვებს).
- **mask & np.isfinite(Z):** ეს დამატებითი ნაბიჯი იცავს **Z**-ის არეული (**NaN**) ან უსასრულო მნიშვნელობების თავიდან აცილებას.
- **img += mask:** ეს ქმნის **img** გამოსახულების პიქსელებს, რომელიც ნებადართული მნიშვნელობებით განახლდება.

5. **ლოგარითმული გარდაქმნა და ნორმალიზაცია:**

```
img = np.log(img + 1)  # ფოტო დახატვა - რომ არ იყოს უსასრულო
return img / img.max()  # ნორმალიზაცია
```

- **np.log(img + 1):** ლოგარითმული გარდაქმნა, რომელიც გამოსახულებას უფრო მკაფიო და გამოკვეთილად აჩენს.
- **img / img.max():** ნორმალიზაცია, რომელიც მიმართულია იმაზე, რომ გამოსახულების თითოეული პიქსელი იყოს **0**-დან **1**-მდე. ეს ეხმარება შექმნას სუფთა და თანაბარი გამოსახულება.

6. **ანიმაციის შექმნის ფუნქცია:**

```
def animate(julia_set_func, c_values,
video_filename="julia_set_animation.mp4", duration=900, fps=30):
```

- **julia_set_func:** ეს არის ფუნქცია, რომელსაც ჩვენ გადავცემთ **julia_set**-ს, რაც გამოიწვევს ჯულიას ნაკრების გენერაციას.
- **c_values:** ეს არის კოეფიციენტების სიები, რომელთაგან თითოეული იმოქმედებს ჯულიას ნაკრების ფორმაზე.
- **video_filename:** გამოსავალი ვიდეო ფაილის სახელი.
- **duration და fps:** ვიდეოს ხანგრძლივობა წამებში და ჩარჩოების რიცხვი წამში.

7. **ვიდეო განლაგება:**

```
fourcc = cv2.VideoWriter_fourcc(*'mp4v')
```

```
out = cv2.VideoWriter(video_filename, fourcc, fps, (800, 800))  # ვიდეოს გამოტანა
```

- **cv2.VideoWriter_fourcc(*'mp4v'):** ეს ინიციალიზებს ვიდეო ფორმატს (MP4).
- **cv2.VideoWriter(video_filename, fourcc, fps, (800, 800)):** ამ ნაწილში შექმნილია ვიდეო გამომცემელი, რომელსაც მივაწოდებთ გამოსახულებების ჩარჩოების მახასიათებლებს, ფორმატს, **FPS**-ს და **სიგანე/სიმაღლეს**.

8. **ანიმაციის პროცესი:**

```
for frame in range(num_frames):
    t = frame / num_frames
    c_idx = int(t * len(c_values))
    c = c_values[c_idx % len(c_values)]  # ცვლის კოეფიციენტი
    img = julia_set_func(c)  # გენერირება ჯულიას ნაკრების გამოსახულება
```

- თითოეული ჩარჩოს გენერაციისთვის, დროის **t** გავლენას ახდენს, რაც განსაზღვრავს რომელ კოეფიციენტზე მოხდება გამოთვლა.

9. **გამოსახულების გადაღება და ვიდეოში ჩადება:**

```
    plt.imshow(img, cmap="twilight", origin="lower")
    plt.axis("off")
    plt.draw()
    plt.pause(1. / fps)

    buf = io.BytesIO()  # იმოგზაურეთ მეხსიერებაში
    plt.savefig(buf, format="png")  # PNG ფორმატში გადარჩენა
```

```
        buf.seek(0)
        pil_img = Image.open(buf)  # გამოსახულების გადაკითხვა PIL-ისთვის
        frame_img = np.array(pil_img)  # numpy array-ში გარდაქმნა
        frame_img = cv2.cvtColor(frame_img, cv2.COLOR_RGBA2BGR)  # RGBA -> BGR
        out.write(frame_img)  # ვიდეოში ჩადება
        plt.clf()  # გაწმენდა შემდეგი ფრაგმენტისათვის
```

- **გამოსახულების ჩარჩოს გადაღება**: **plt.savefig(buf, format="png")** - აღნიშნავს, რომ თითოეული ჩარჩო ჩამოყალიბდება **PNG** ფორმატში და მოთავსდება მეხსიერებაში.
- **OpenCV-ით ვიდეოში ჩადება: out.write(frame_img)** - აქ ყოველი ფრაგმენტი დაემატება ვიდეოს ფაილს.

10. **ვიდეო ფაილის დასრულება:**

```
    out.release()  # ვიდეო ფაილის დახურვა
```

- **out.release():** ეს გაათავისუფლებს ყველა ვიდეო რესურსს, რაც განასხვავებს დასრულებულ ვიდეო ფაილს.

11. **მთავარი ფუნქცია:**

```
if __name__ == "__main__":
    c_values = [ ... ]
    animate(julia_set, c_values)
```

- ამ ხაზში, პროგრამა იყენებს კოეფიციენტების სიას, შემდეგ კი უწოდებს **animate()** ფუნქციას, რომელიც წარმოშობს ანიმაციას.

ეს არის პროგრამის ყოველი ბრძანების დეტალური ახსნა.

დამატებითი ინფორმაციისათვის გთხოვთ დაგვიკავშირდეთ, ქვემოთ მითითებულ კოორდინატებზე.

ნუ დაგეზარებათ, ყოველი თქვენთაგანის აზრი ჩვენთვის მნიშვნელოვანია.

ჩვენი კოორდინატები:

ელ. ფოსტა: isheriphadze@gmail.com

მობ:, ვოცაპი: +995(555)45-92-70

პატივისცემით იმედა შერიფაძე

ინფორმაციული ტექნოლოგიებისა და პროგრამული უზრუნველყოფის დარგის სპეციალისტი, ასევე ნეირონული ქსელების დიზაინისა და თანამედროვე ცხოვრებაში მათი გამოყენების სპეციალისტი