گزارش تمرین نخست

نام: محمد جمشیدی

شماره دانش‌جویی: ۹۸۱۰۰۷۱۸

۱. برفدانه‌ی کوخ:

برای ساختن این برفدانه ابتدا ۴ تابع جداگانه برای انجام تبدیل‌های لازم می‌سازیم. هر یک از این توابع اعمال تجانس، دوران و انتقال مورد نیاز را انجام می‌دهند. ساز و کار این توابع به این صورت است:

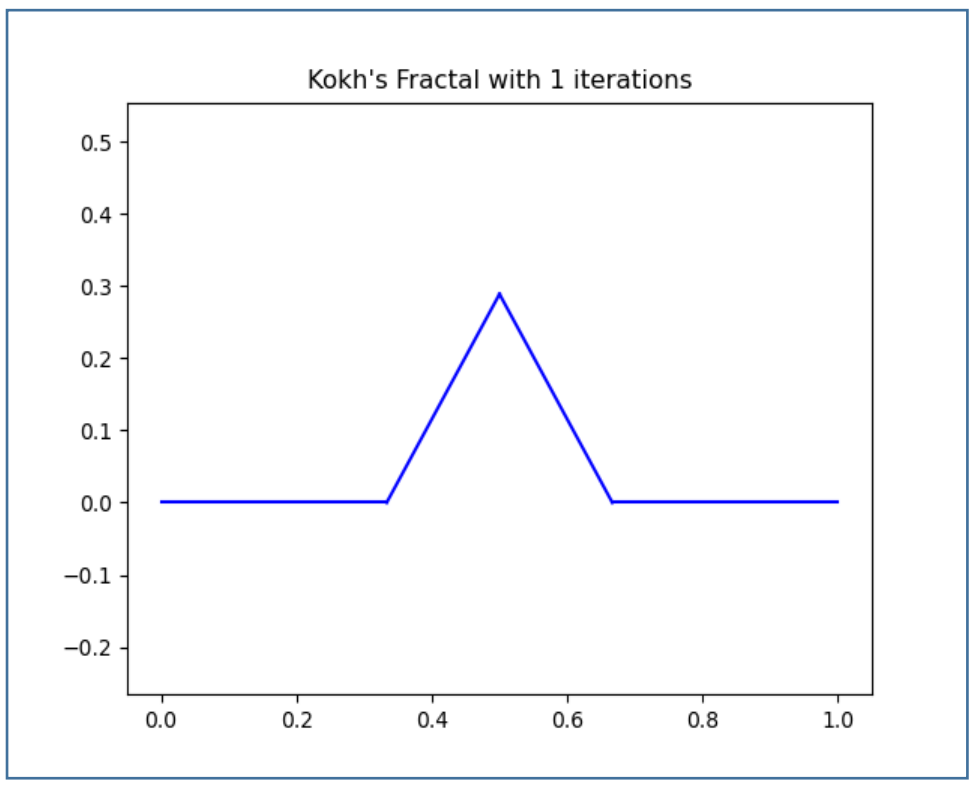
تابع f1(): تجانس با ضریب یک سوم.

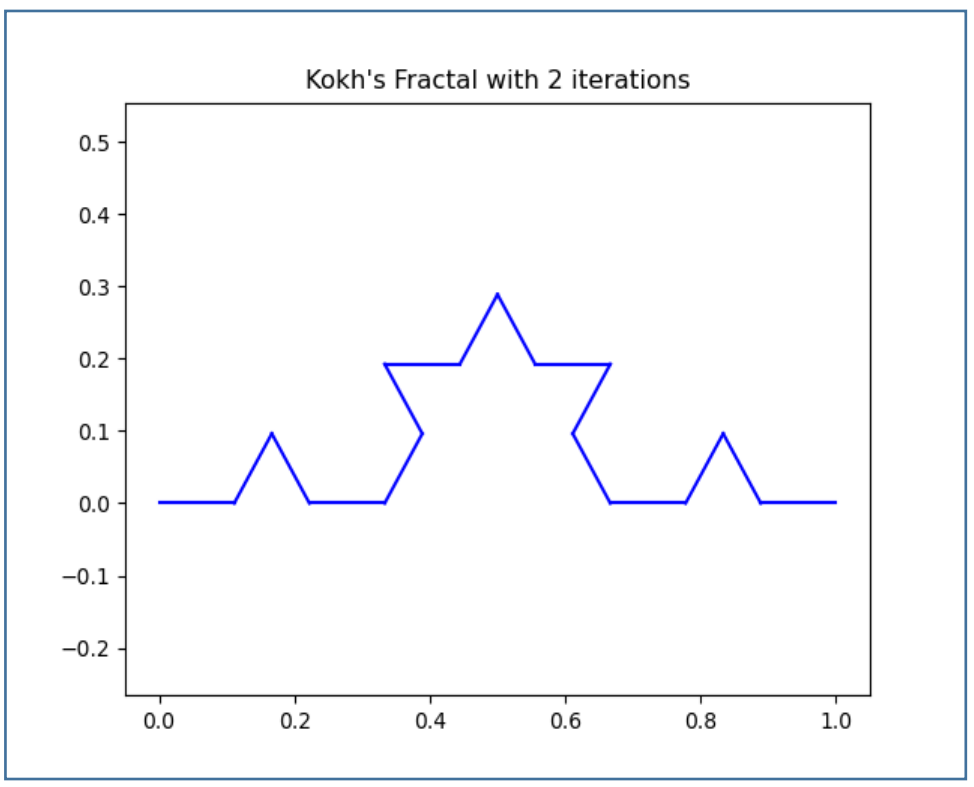
تابع f2(): دوران پادساعتگرد به اندازه‌ی ۶۰ درجه و تجانس با ضریب یک سوم. سپس انتقال به اندازه‌ی یک سوم در جهت مثبت محور ایکس.

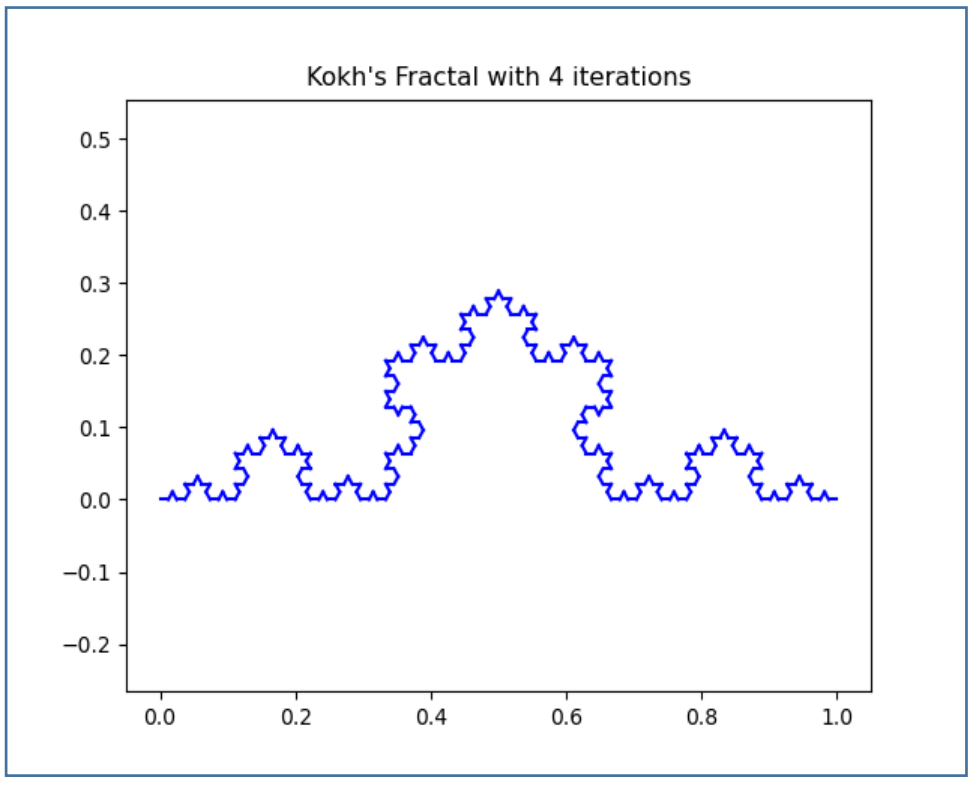
تابع f3(): دوران ساعتگرد به اندازه‌ی ۶۰ درجه و تجانس با ضریب یک سوم. سپس یک انتقال به اندازه‌ی یک دوم در جهت مثبت محور ایکس و یک انتقال به اندازه‌ی رادیکال سه ششم در جهت مثبت محور وای.

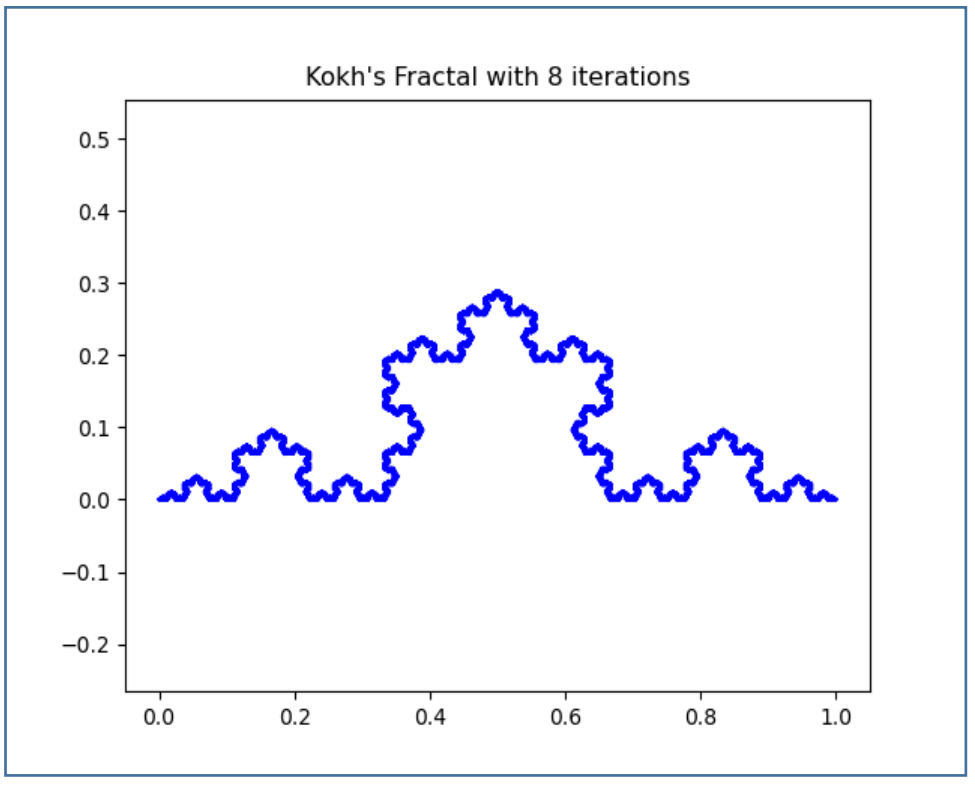
تابع f4(): تجانس با ضریب یک سوم و سپس انتقال به اندازه‌ی دو سوم در جهت مثبت محور ایکس.

در نهایت با کمک این توابع تا درجه‌ی تکراری که مطلوب کاربر باشد، فراکتال مورد نظر ساخته می‌شود.





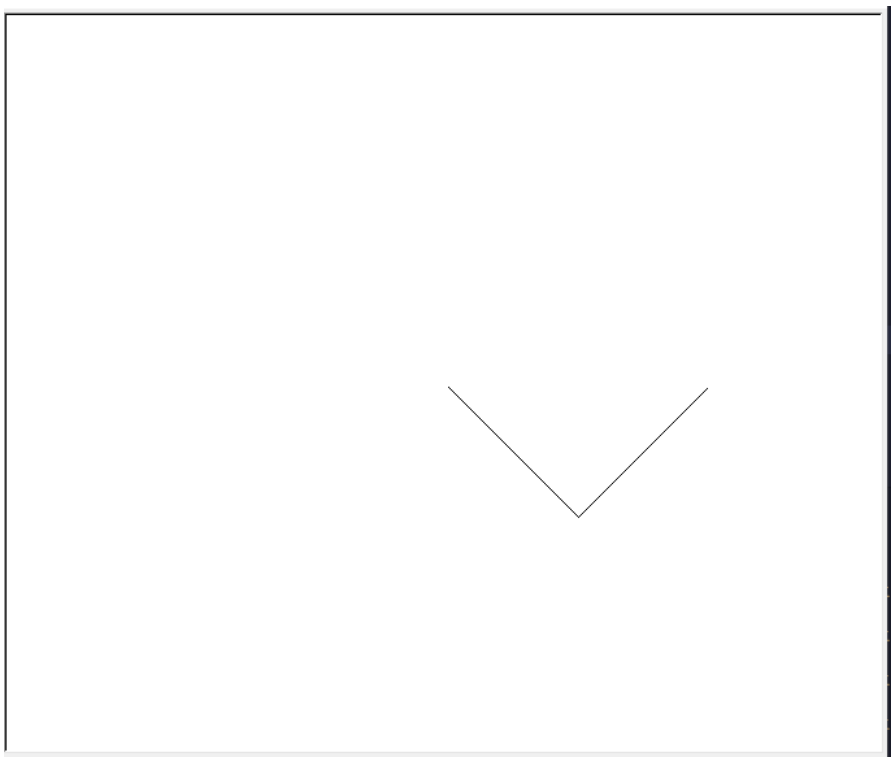




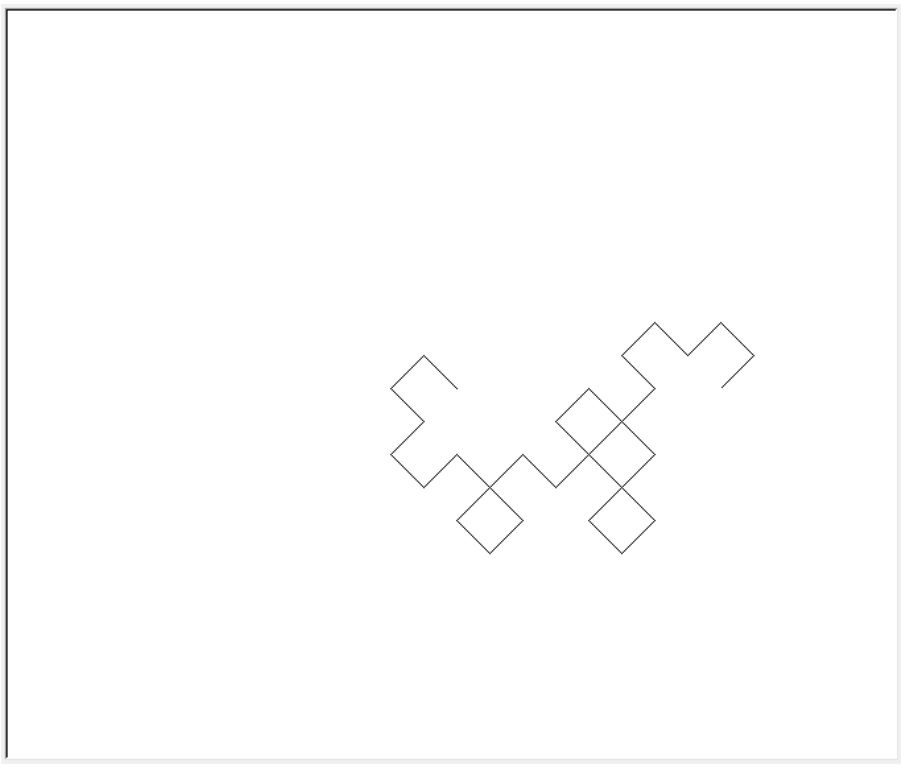
۲. اژدهای هی وی

سعی داشتم اژدهای هی وی را هم مانند دیگر شکل‌ها تنها به کمک کتابخانه‌ی matplotlib.pyplot و numpy رسم کنم. در انجام این کار هم تا حد خوبی موفق شدم اما کد نهایی، شکل مطلوب را با خطا تولید می‌کرد که دیگر فرصت برطرف کردن خطا و دیباگ کردن کد و الگوریتم خود را نداشتم. اما با استفاده از کتابخانه‌ی turtle این کار را می‌توان به سادگی انجام داد و تعداد خط‌های کد آن نیز کم‌تر است.

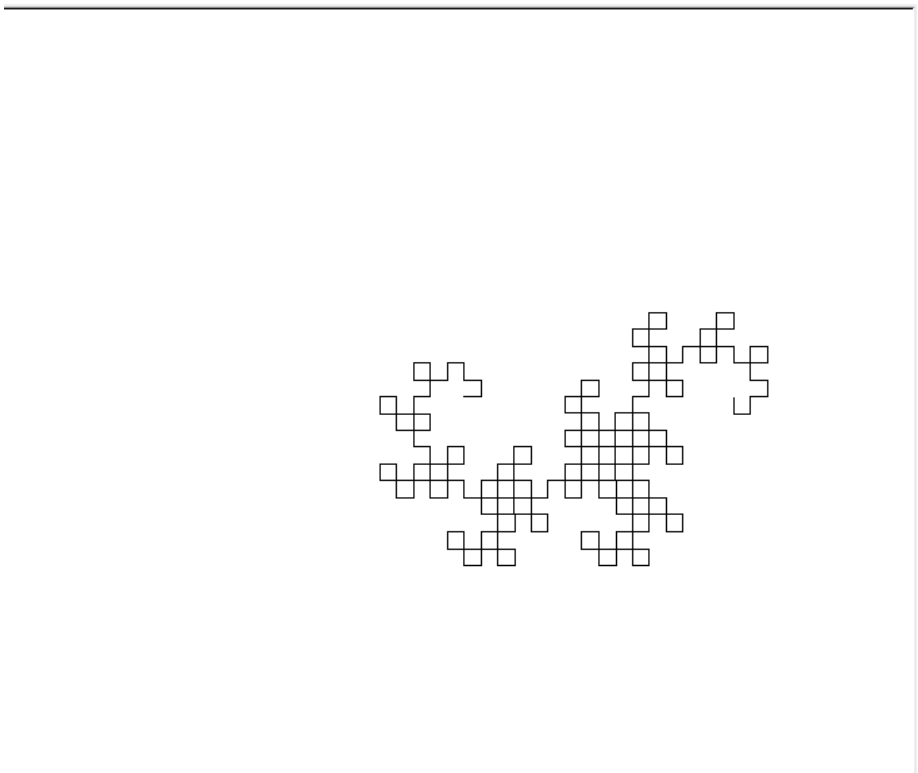
برای ۱ تکرار:



برای ۵ تکرار:



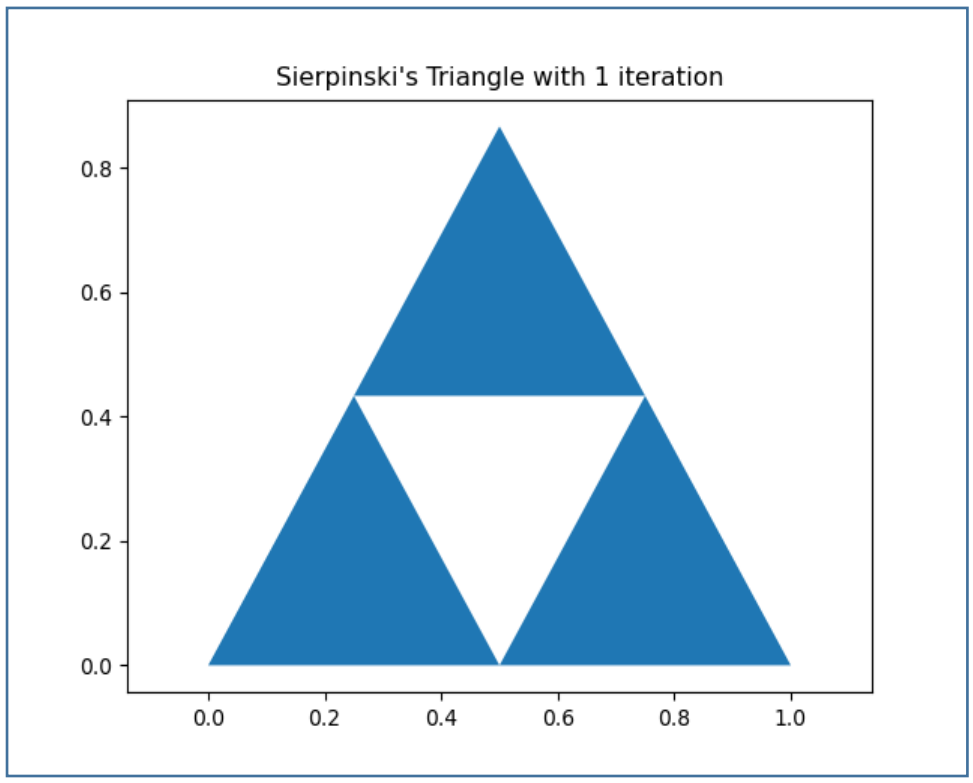
برای ۸ تکرار:

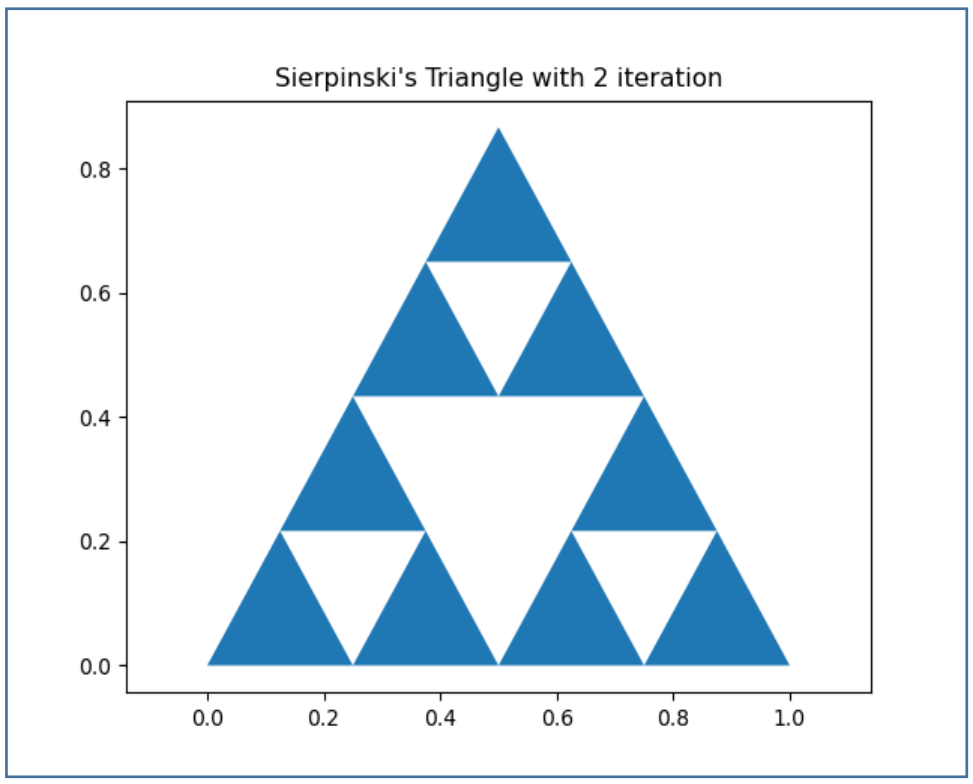


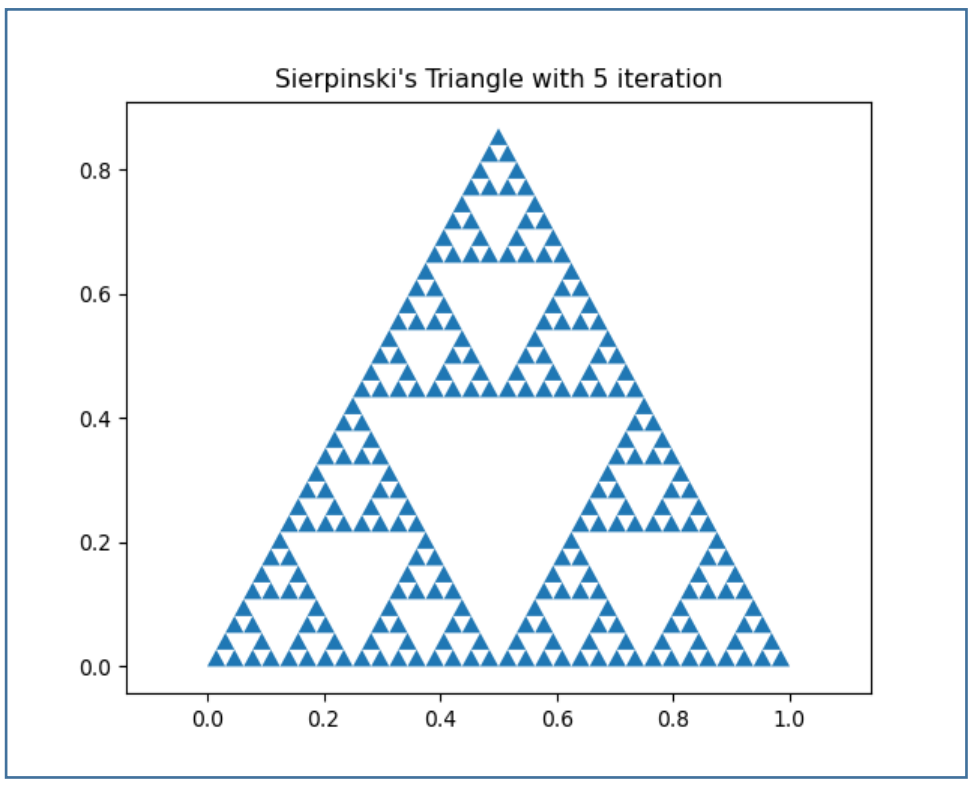
۳. مثلث سیرپینسکی:

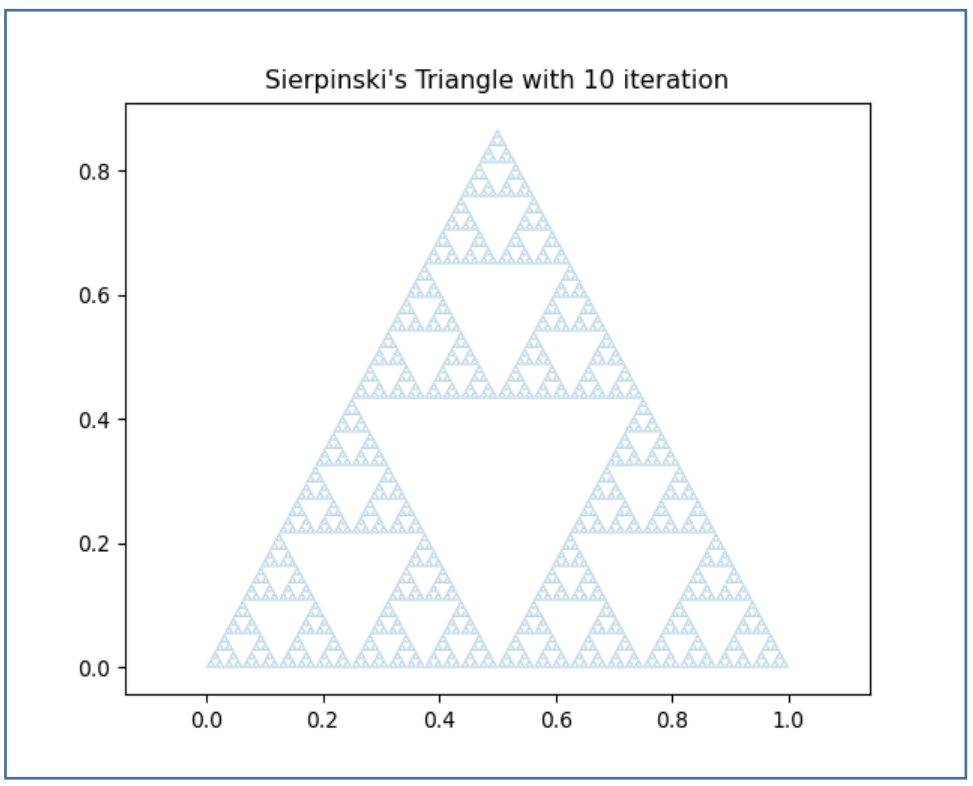
برای ساختن این مثلث، در هر مرحله از هر یک از مثلث‌های کوچک‌تر، سه مثلث جدید می‌سازم. برای انجام این کار از تابع newTriangleMaker() استفاده می‌کنم. این تابع یک مثلث می‌گیرد و با استفاده از سه تابع دیگر newTriangleMaker0()، newTriangleMaker1() و newTriangleMaker2()، سه مثلث جدید درست می‌کند.

هر کدام از این سه تابع، خود از تبدیل‌های تجانس و انتقال برای ساختن مثلث‌های جدید استفاده می‌کنند. ضریب تجانس هربار، یک دوم است. تبدیل‌های انتقال هم به گونه‌ای است که هر کدام از سه مثلث جدید در یک گوشه‌ی مثلث اولیه قرار گیرند.









۴. مثلث سرپینسکی با استفاده از مثلث خیام

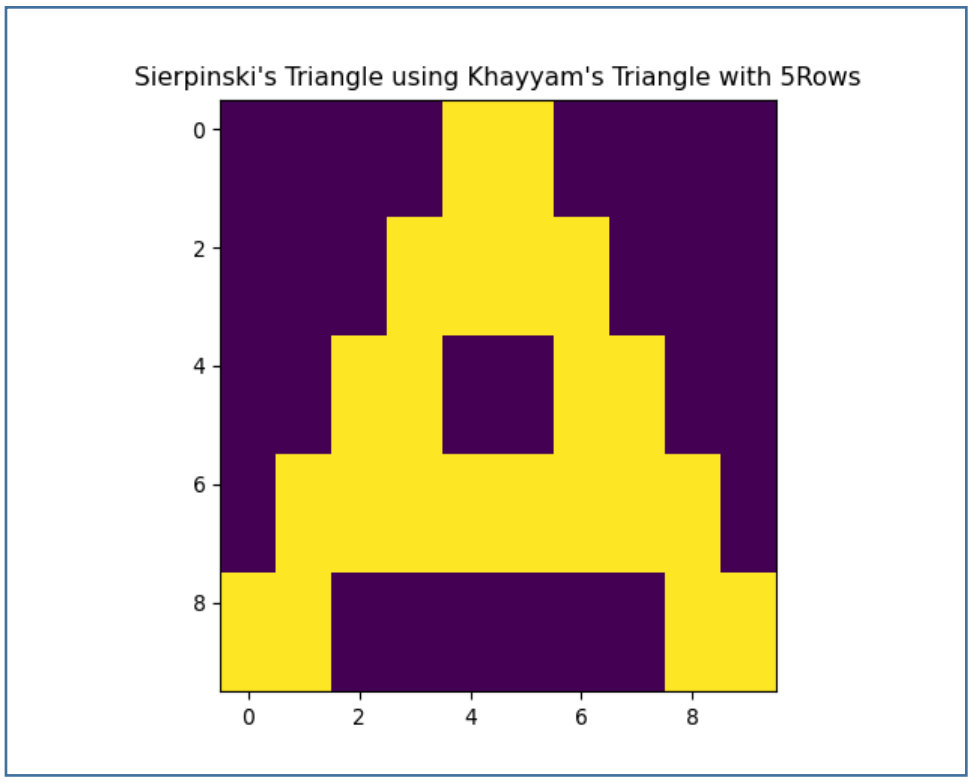
با استفاده از مثلث خیام می‌خواهیم مثلث سیرپینسکی را تولید کنیم. برای این کار ابتدا تابع makeKhayyamNumbers() را تعریف می‌کنم. ورودی این تابع شماره‌ی سطری است که می‌خواهیم اعداد خیام تا آن سطر تولید شوند (به عبارتی تعداد سطرهای مثلث مطلوب). خروجی این تابع یک لیست است که اعداد هر سطر را به صورت یک لیست جداگانه در خود جای داده است. در ادامه چون قرار است از تابع imshow() برای نمایش استفاده کنم، لیست بدست آمده از مثلث خیام را بازتولید می‌کنم. برای این منظور تابع makeArrayOfKhayyamNumbers() را تعریف می‌کنم. این تابع در قدم اول، طول همه‌ی لیست‌های درونی را دو برابر می‌کند؛ به این ترتیب که از هر عدد موجود (به جز یک‌های مرزی) یک کپی در کنار آن می‌سازد. نمونه:

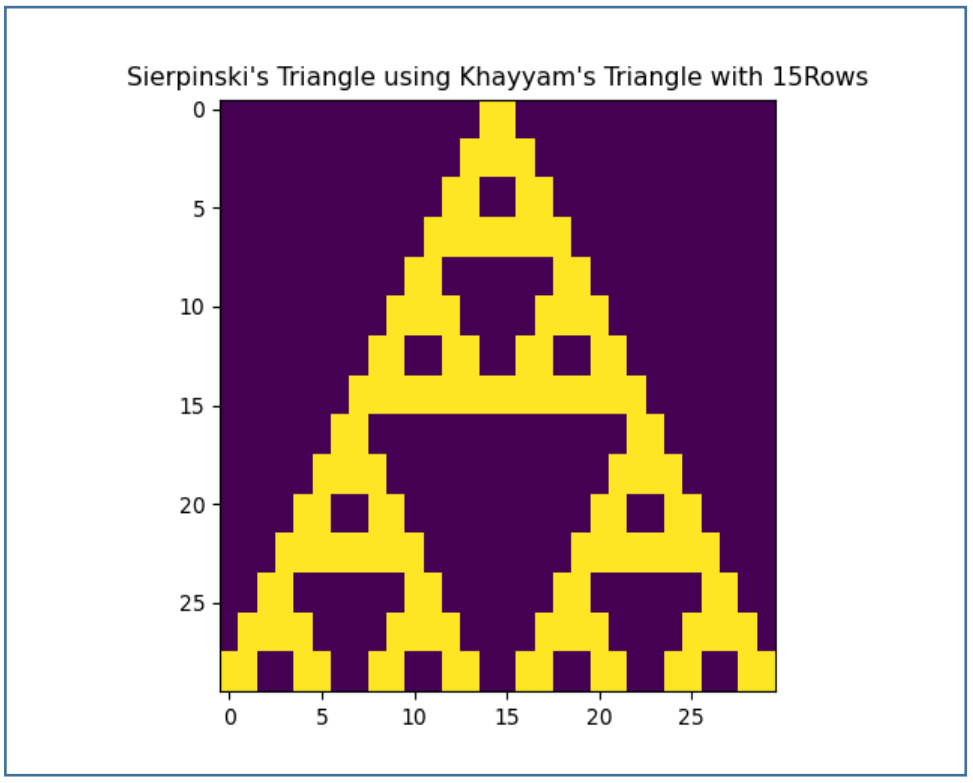
[[1], [1, 1], [1, 2, 1]] ➔ [[1, 1], [1, 1, 1, 1], [1, 1, 2, 2, 1, 1]]

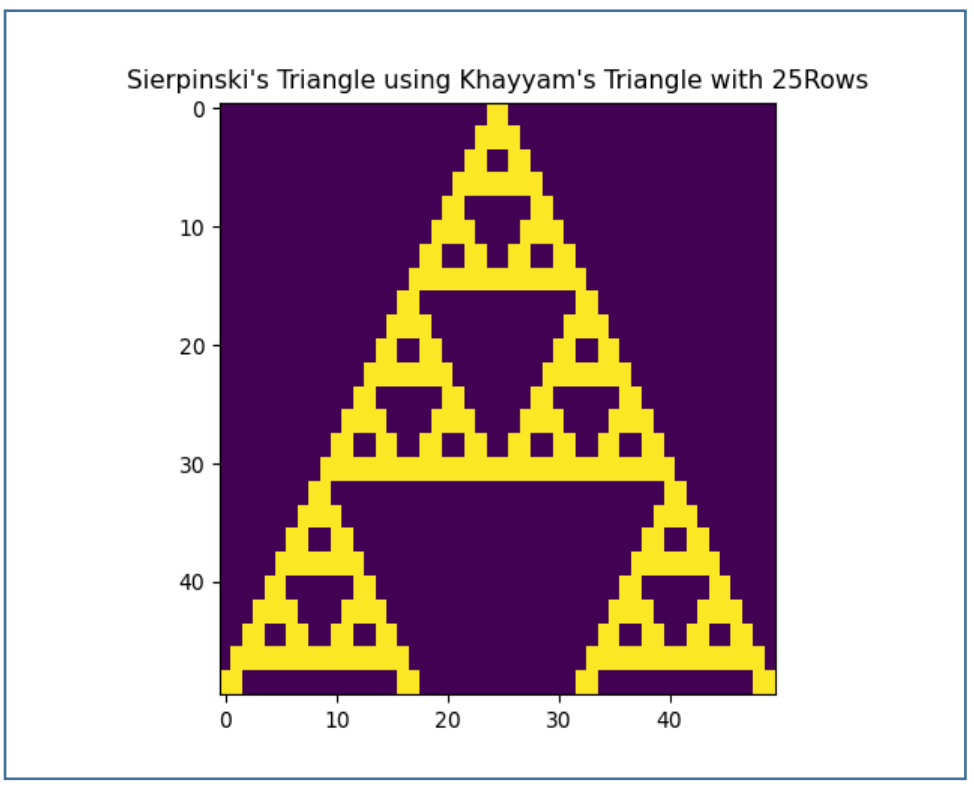
در قدم بعدی برای یکسان شدن طول همه‌ی لیست‌ها، جاهای خالی را با صفر پر می‌کند.

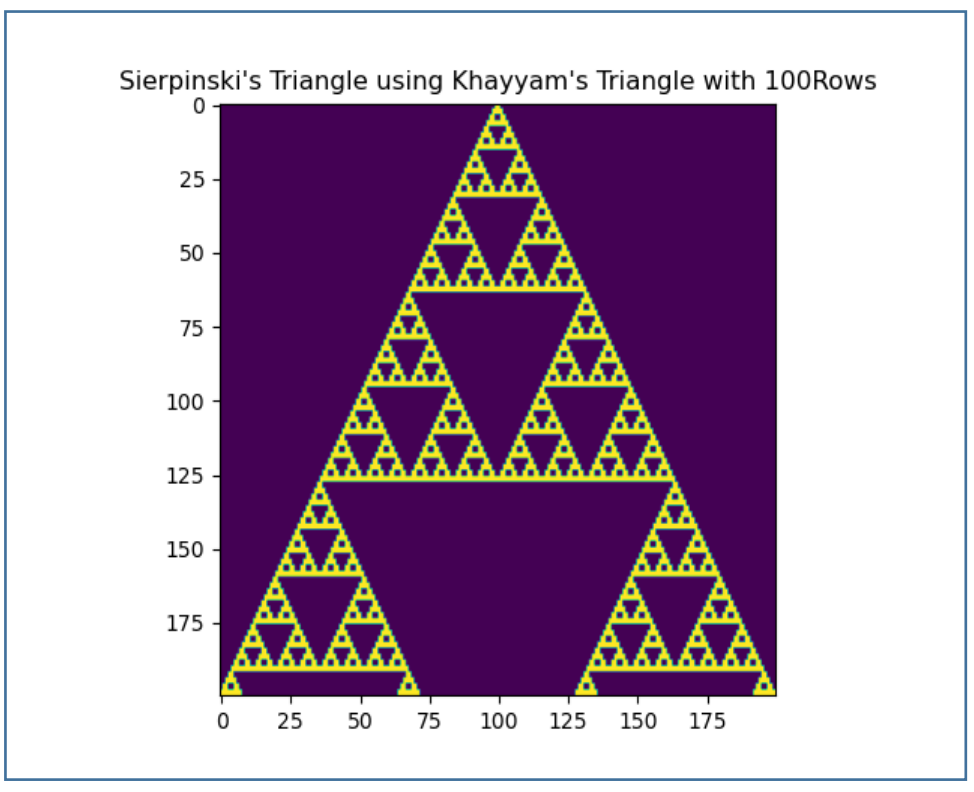
➔[[0, 0, 1, 1, 0, 0], [0, 1, 1, 1, 1, 0], [1, 1, 2, 2, 1, 1]]

اکنون از هر کدام از این لیست‌ها یک کپی در کنار آن ساخته می‌شود و در لیست اصلی جای می‌گیرد. در نهایت به کمک تابع showKhayyamTriangle() به جای هر کدام از اعداد موجود در این لیست‌ها، باقی‌مانده‌ی آن‌ها به پیمانه‌ی ۲ جایگذاری می‌شود و سپس به کمک تابع imshow() خروجی مورد نظر نمایش داده می‌شود.





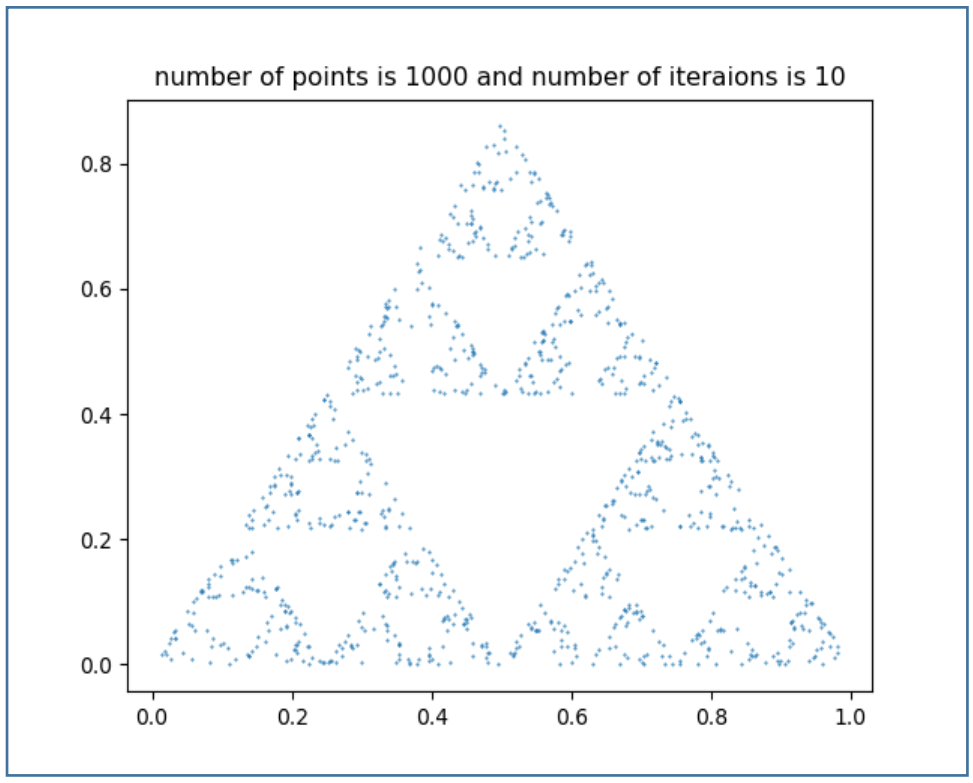


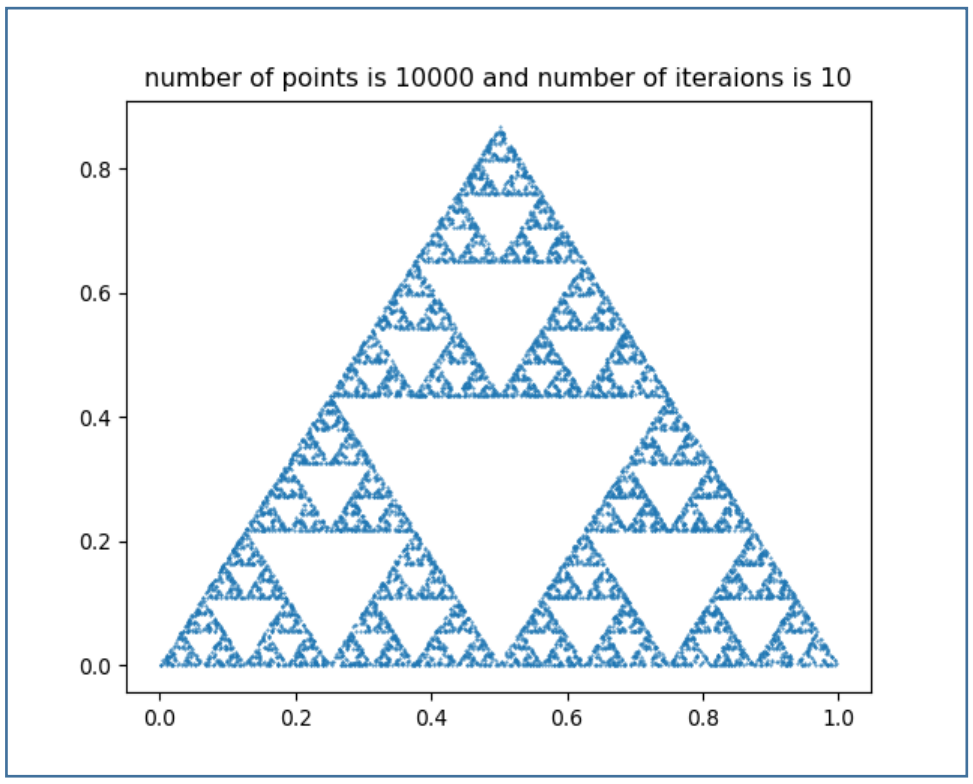


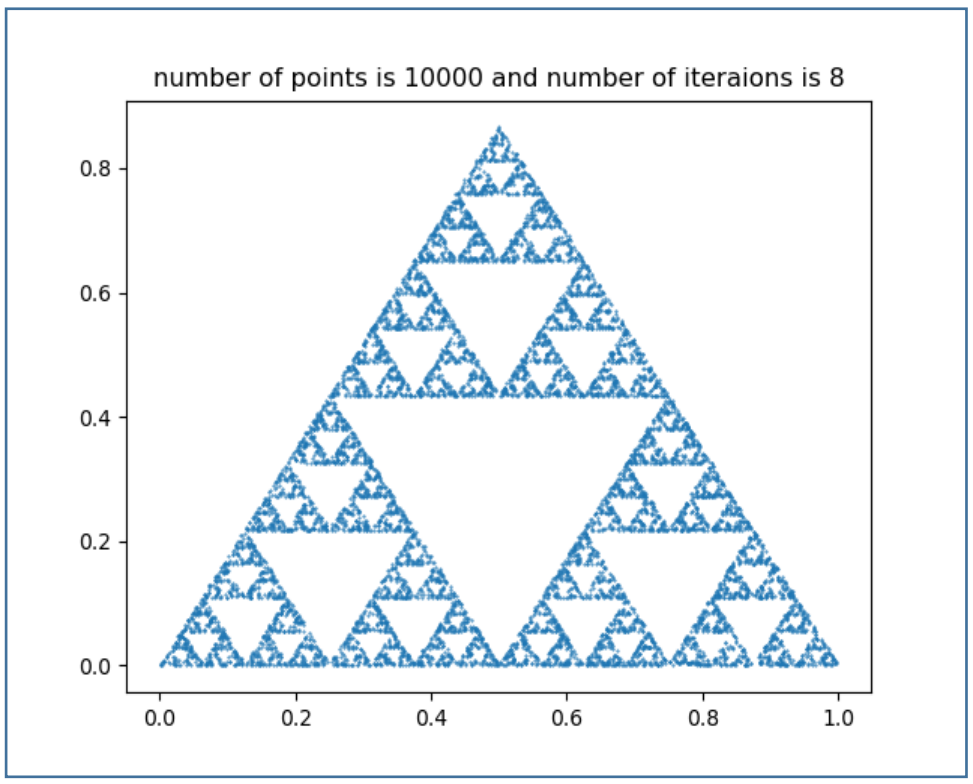
یک چیزی که در این شکل‌ها دیده می‌شود این است که انگار یک تیکه‌ی اضافی در پایین‌ترین ضلع وجود دارد. نکته این است که تنها برای توان‌های دو یک مثلث کامل و دقیق خواهیم داشت.

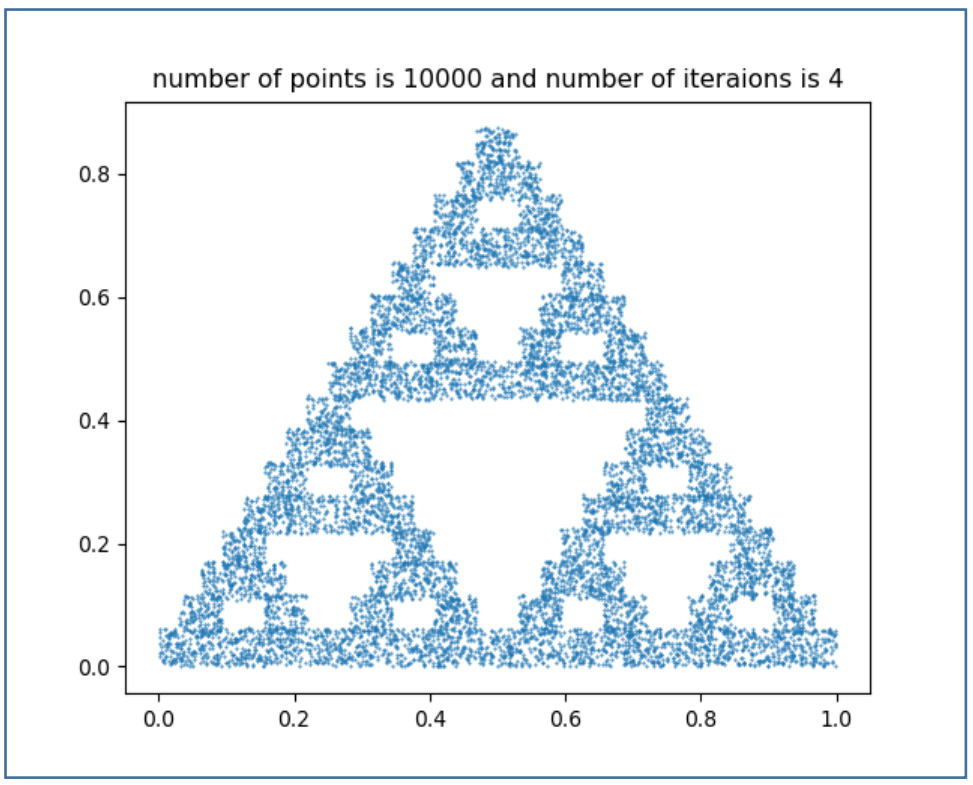
۵. مثلث سیرپینسکی با الگوریتم تصادفی

با کمک الگوریتم تصادفی مثلث سیرپینسکی را تولید می‌کنیم. ابتدا سه تابعf1, f2 و f3 را تعریف می‌کنیم. کار این توابع انجام تبدیل‌های مقیاس و انتقال مورد نیاز بر روی یک نقطه‌ی ورودی است. الگوریتم به این شکل است که برای تولید مثلث با مرتبه‌ی دلخواه، به تعداد این مرتبه و به صورت رندوم این توابع را بر روی نقطه‌ی ورودی اثر می‌دهیم. سپس کل این عملیات را برای تعداد مناسبی نقطه تکرار می‌کنیم و هر نقطه‌ی حاصل را بر روی صفحه نمایش می‌دهیم. ۴ نمونه خروجی برای تعداد متفاوتی از نقاط و و برای درجه‌ی تکرارهای ۸ و ۱۰ در زیر آمده است.









۶. برگ سرخس با الگوریتم تصادفی

ابتدا ۴ تابع f1, f2, f3 و f4 را می‌سازم. وظیفه‌ی هر کدام از این توابع انجام تبدیلات لازم برای تولید بخش‌های گوناگون سرخس است. تابع f1، بخش زرد شکل را می‌سازد. تابع f2، بخش قرمز، تابع f3، بخش آبی پررنگ و تابع f4 هم بخش آبی کم‌رنگ را می‌سازد. اعداد داخل کد قابل مشاهده است. در ادامه برای افزایش کیفیت شکل خروجی و شبیه‌تر کردن آن به شکل کتاب با جستجوی اینترنتی متوجه شدم که می‌توانم با کمک یک توزیع احتمال این کار را انجام دهم. تابع probabilityDisturb() همین وظیفه را دارد. اعداد مشخص کننده‌ی توزیع احتمال در داخل کد مربوط به این تابع مشخص است. در ادامه برای تعداد زیادی نقطه الگوریتم تصادفی را به کار می‌بریم و هر نقطه‌ی حاصل را رسم می‌کنیم. در نهایت شکل زیبای سرخس حاصل می‌شود. این کار در نهایت به وسیله‌ی تابع makeRandomFern() انجام می‌شود. ورودی‌های این تابع تعداد نقاط مورد نظر کاربر و تعداد تکرارهاست.