سری 1

1-2) کوخ

برای حل این سوال بجای روش انتقال و مقیاس کردن از الگوریتم دیگری استفاده شده که تعداد نقاط نهایی را کاهش داده و پردازش را بهبود می بخشد.

find_new_points() : با دادن دو نقطه ابتدایی و انتهایی خط به تابع سه نقطه جدید پیدا میکند که عبارتند از:

point1 و point2 که روی خط اصلی هستند و خط را به سه قسمت تقسیم میکنند.

همچنین top_point که به اندازه ارتفاع مثلث متساوی الاضلاع بالاتر از وسط خط یا همان middle_point میباشد.

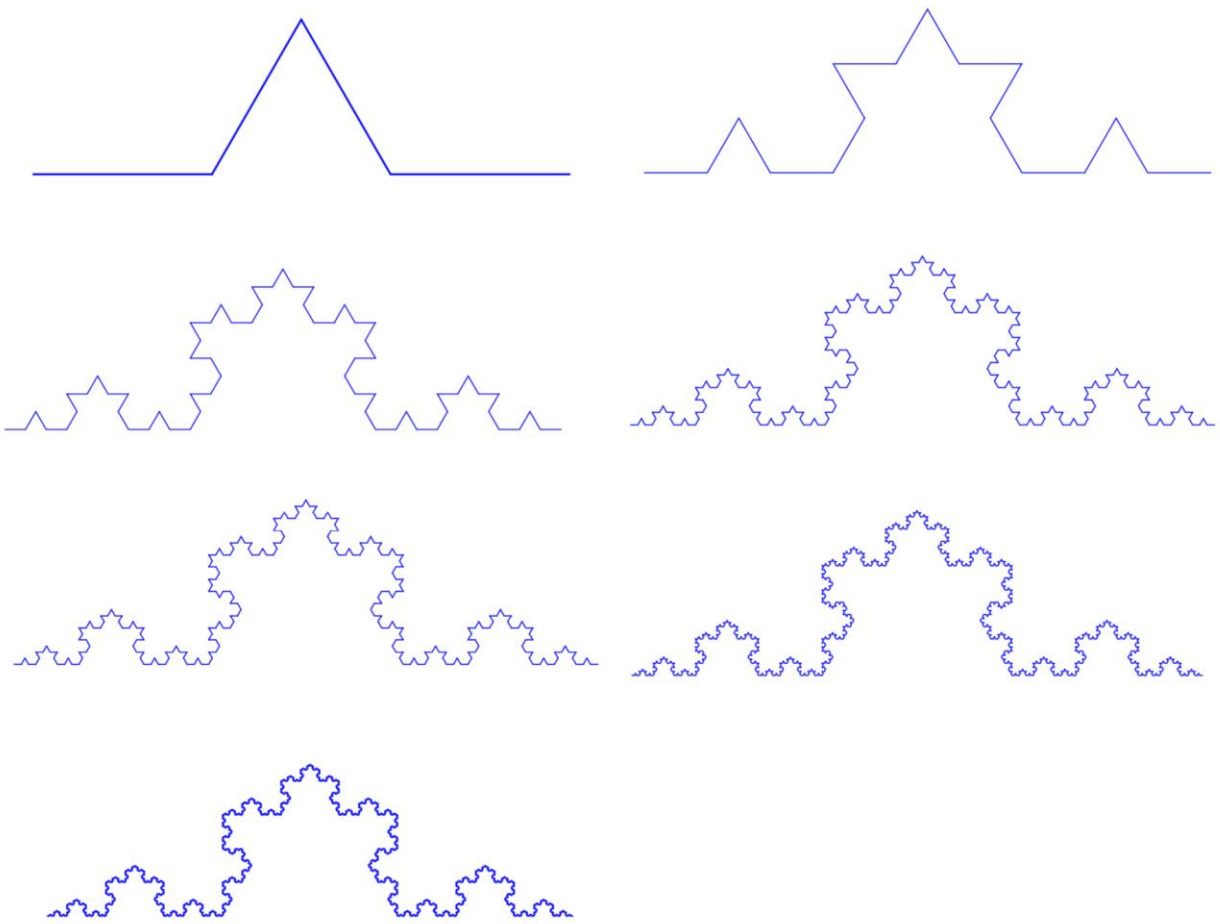
برای پیدا کردن هر کدام از این نقاط تابع جداگانه ای تعریف شده است که با بردار یکه در راستای خط کار میکند و با خط های چرخیده نیز سازگار است.

چالش این روش نحوه ذخیره سازی نقطه ها است. چون برای کشیدن خط ها روی لیست نقطه ها حرکت میکنیم و به ترتیب دو به دو بین نقطه ها خط میکشیم، به همین دلیل ترتیب ذخیره سازی نقطه ها بسیار مهم است.

add_new_points(): برای این کار بعد از پیدا کردن سه نقطه جدید نقطه end_point را در لیست پیدا میکنیم و نقاط جدید را قبل آن insert میکنیم.

Line_between_points() : به دلیل عدم آشنایی کامل با plt این تابع برای ساختن خط بین دو نقطه تعریف شده است که نیازی به آن نبود و روش های ساده تری برای انجام این کار وجود داشت.

نتایج به صورت زیر است.

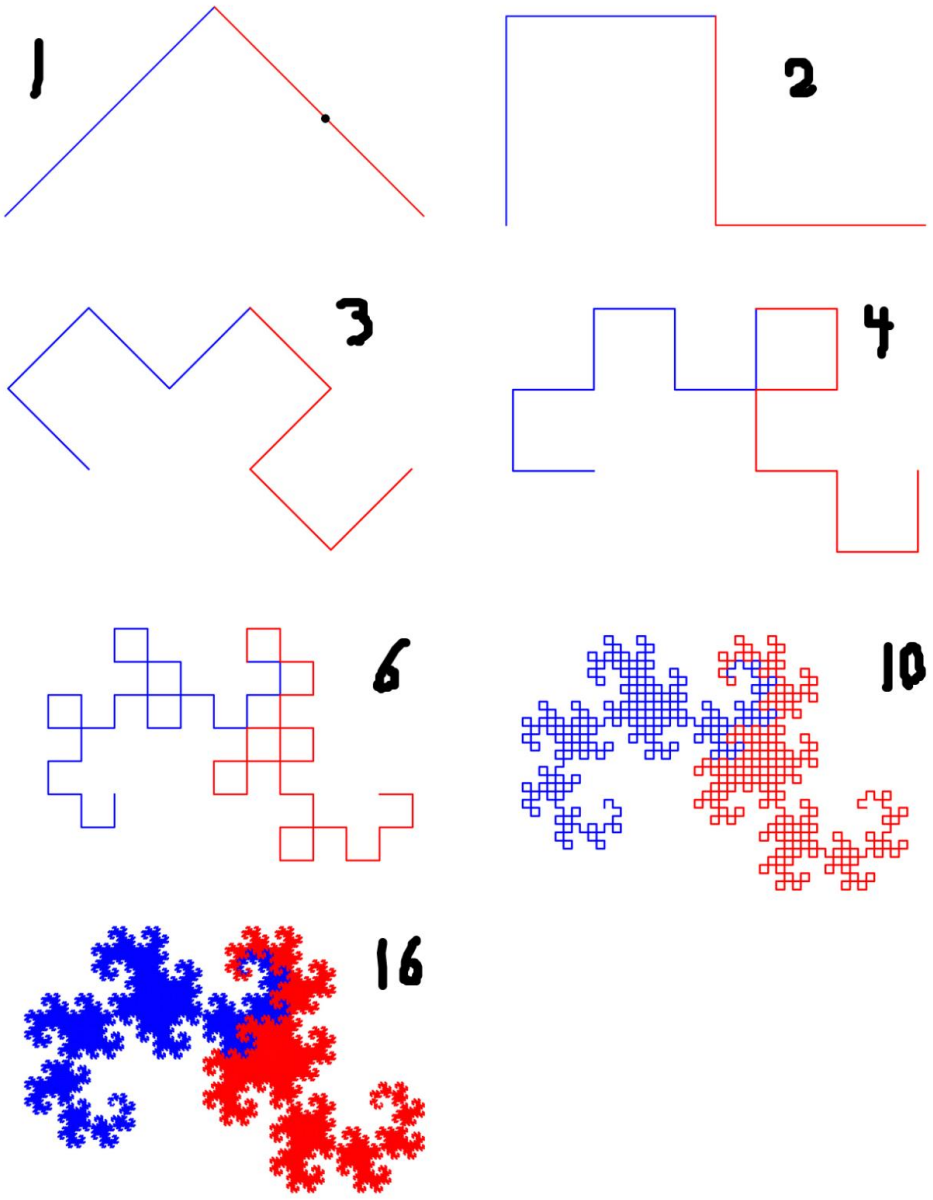


2-2) اژدها هی وی

برای این سوال و سوال های دیگر دو تابع جامع scale_shape و rotate_shape تعریف شده اند که به صورت کلی یکسری نقاط میگیرند و عملیات مورد نظر را انجام میدهند.

dragon_curve() : تابع اصلی این کد است که نقاط آبی را 45 درجه دوران و مقدار 0.5 مقیاس میکنید و نقاط قرمز را 135 درجه دوران و مقدار 0.5 مقیاس میکند.

نکته این است که باز ترتیب ذخیره سازی نقاط مهم است و لیست نقاط قرمز قبل از ذخیره سازی باید برعکس شود.



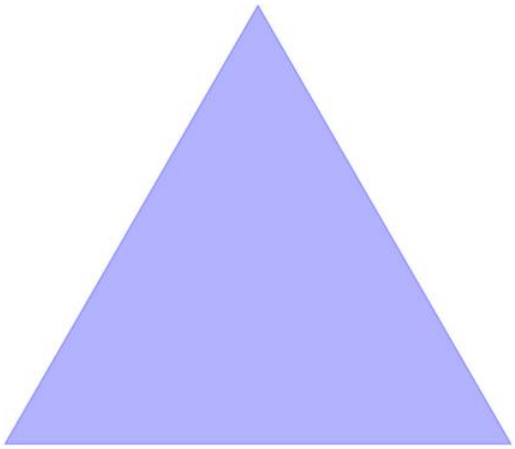
3-2) مثلث سرپینسکی

در این سوال نیز از تابع scale_shape سوال قبل استفاده شده و هر سری center_point یکی از راس هاس مثلث قرار داده شده که نقاط به سمت آن راس به اندازه 0.5، اسکیل شوند.

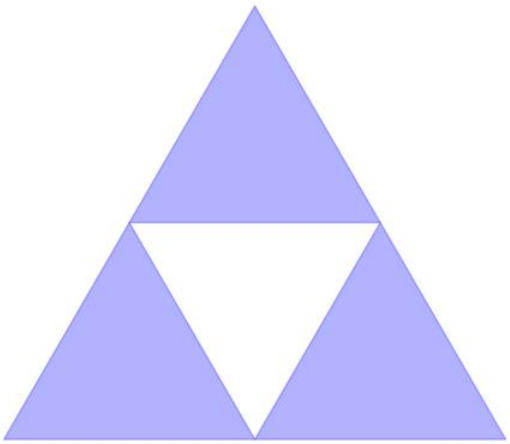
چالش این سوال نمایش درست مثلث ها از بین همه نقاط در لیست all_points بود. برای این کار در کد نمایش داده شده نقاط را به ترتیب به دسته های سه تایی تقسیم بندی کرده و به عنوان یک مثلث نمایش میدهیم.

```
59  for i in range(len(all_points) // 3):
60      triangle = all_points[3 * i : 3 * i + 3]
61      plt.fill(triangle[:, 0], triangle[:, 1], alpha=0.3, color="blue", label="Top")
```

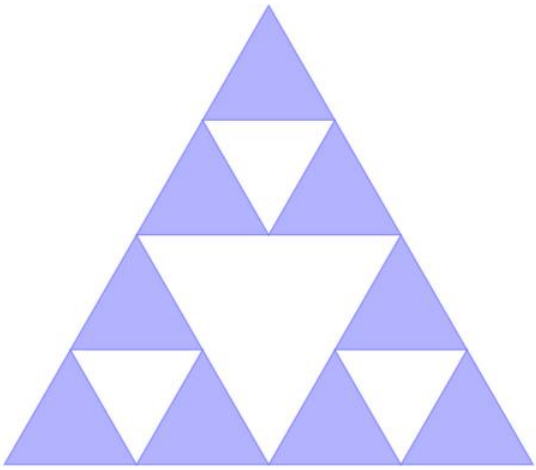
Sierpinski Triangle - Iteration 0



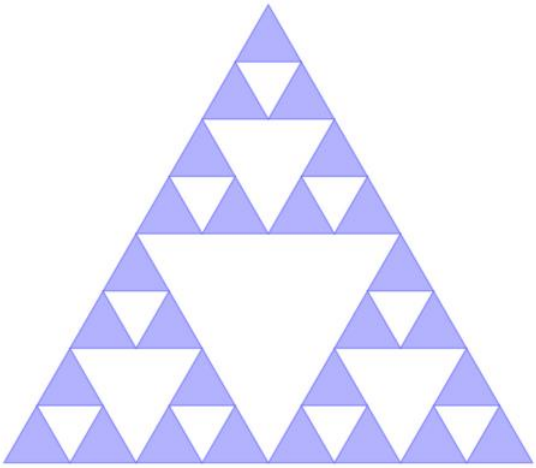
Sierpinski Triangle - Iteration 1



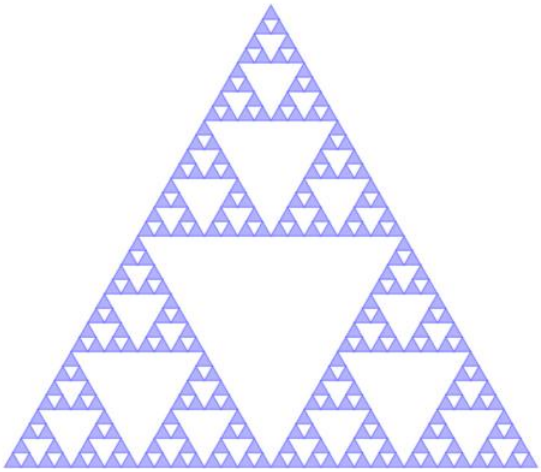
Sierpinski Triangle - Iteration 2



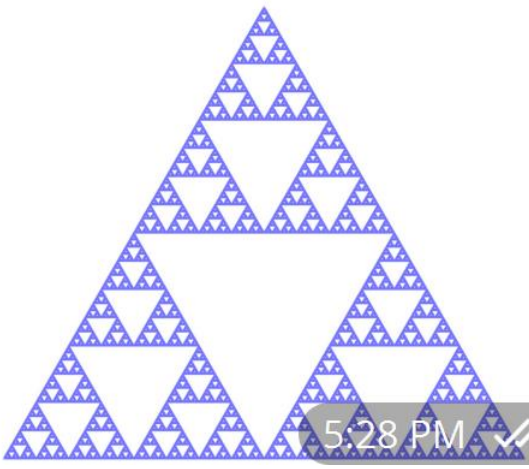
Sierpinski Triangle - Iteration 3



Sierpinski Triangle - Iteration 5



Sierpinski Triangle - Iteration 8

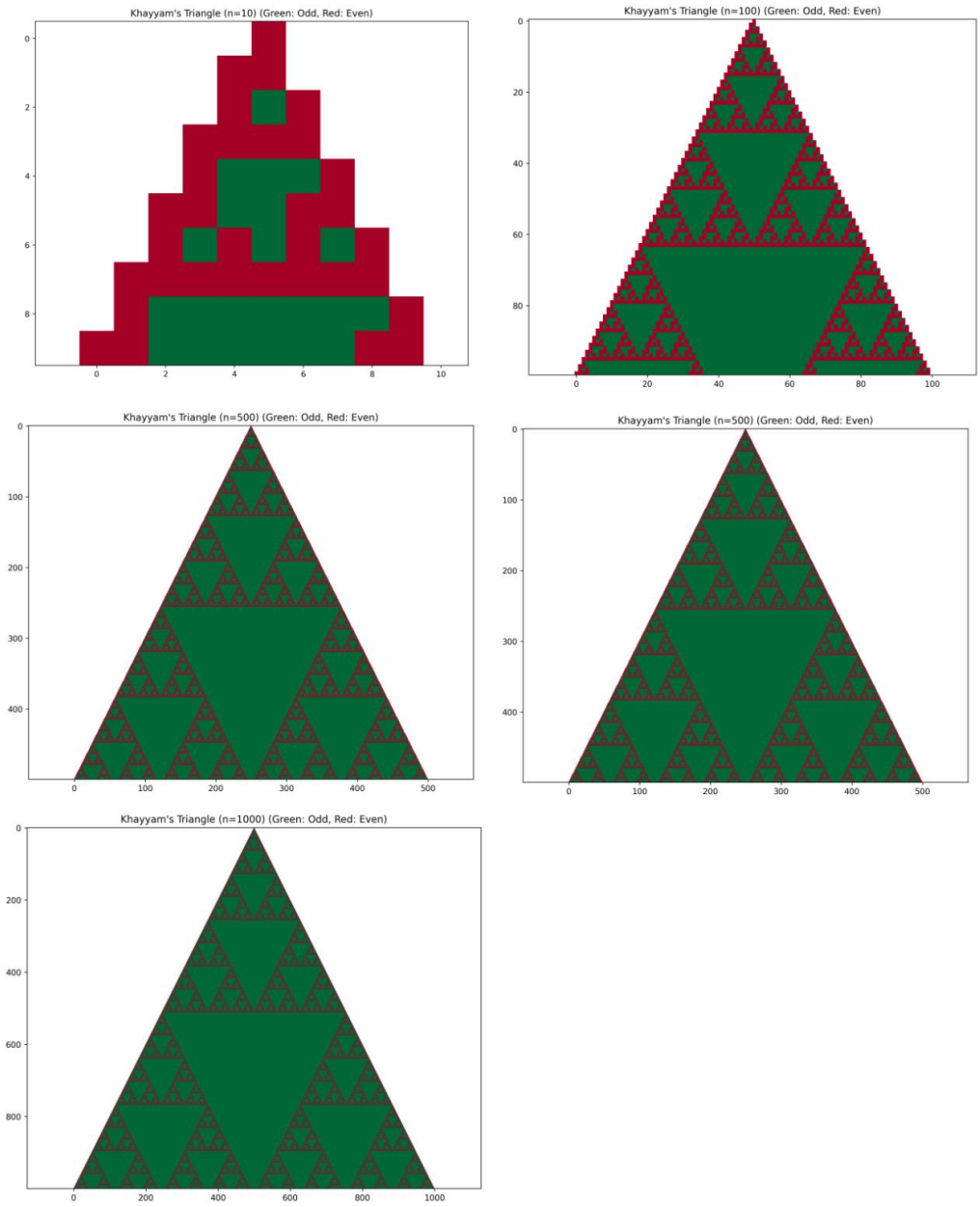


4-2) مثلث خیام

generate_khayyam_triangle(n) : مثلث خیام را به صورت عددی تا مرتبه n تولید میکند.

generate_point(triangle) : برای رنگ ها ماتریس colors تعریف شده که به اعداد فرد 1 و به اعداد زوج 2 نسبت میدهد. جایگاه های خالی اعداد 0 میمانند که در نهایت mask میشوند.

برای نمایش تصویر نیز اعداد زوج با پیکسل قرمز و اعداد فرد با پیکسل سبز نمایش داده میشوند.



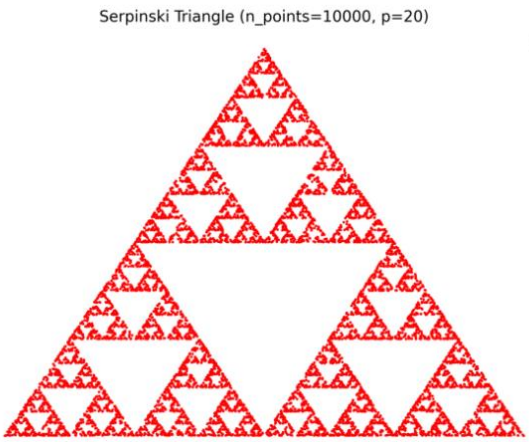
پیکسل ها از وسط تصویر شروع میشوند ولی همانطور که برای n های کم قابل مشاهده است، وقتی یک ردیف تعدادی زوج عدد داشته باشد پیکسل ها یک خانه به سمت چپ متمایل میشوند که در تعداد بالا قابل مشاهده نیست.

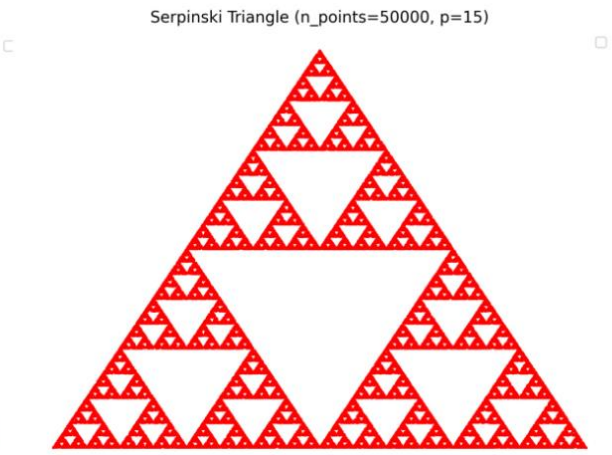
5-2) سرپینسکی تصادفی

باز هم ازscale_shape استفاده میکنیم.

Random_point_generator() : یک نقطه به صورت رندوم در مثلث اولیه تولید میکند.

Serpinski_generator() : به تعداد p بار یکی از راس ها را انتخاب میکند و نقطه را به سمت راس انتخاب شده اسکیل میکند، سپس نقطه نهایی را ذخیره میکند.





6-2) سرخس

از توابع scale_shape و rotate_shape استفاده شده است.

تابع random_point_generator تغییر پیدا کرده است تا در مستطیل اولیه یک نقطه رندوم تولید کند.

چهار توابع اصلی با نام های right_func, left_func, top_func, tail_func تعریف شده اند که مستطیل اصلی را به مستطیل های داده شده در شکل تبدیل میکنند. زوایا و مقیاس ها اندازه گیری شده است.

Srakhs() : این تابع به تعداد p بار یکی از توابع اصلی را انتخاب میکند (tail_func با احتمال 4% و بقیه با احتمال 32% انتخاب میشوند) و روی نقطه رندوم اثر میدهد و نقطه نهایی را ذخیره میکند.

متاسفانه به دلیل اشتباه بود زوایا و مقیاس ها این شکل خیلی مطلوب در نیامد، ولی کارایی الگوریتم درست است.

7-2) مجموعه ژولیا

این تمرین برای من نیاز به یادگیری بالایی داشت که بتوانم محاسبات را به صورت موازی روی پیکسل ها پیاده سازی کنم.

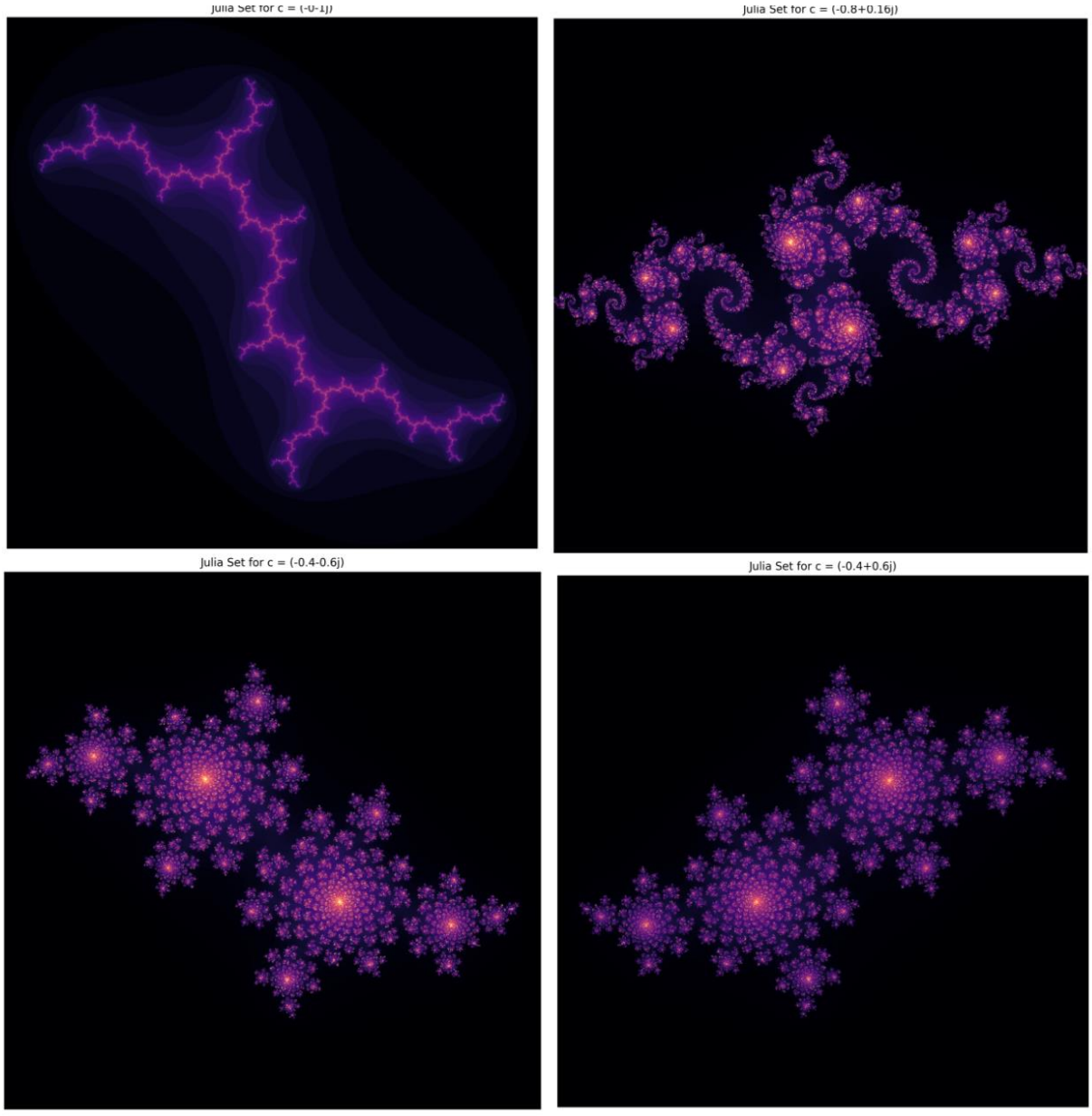
ابتدا صفحه را با ogrid گرید بندی میکنیم و آن را به مختصات مختلط تبدیل میکنیم که به صورت یک ماتریس z ذخیره میشود.

برای این ماتریس دو ماتریس دیگر تعریف میکنیم.

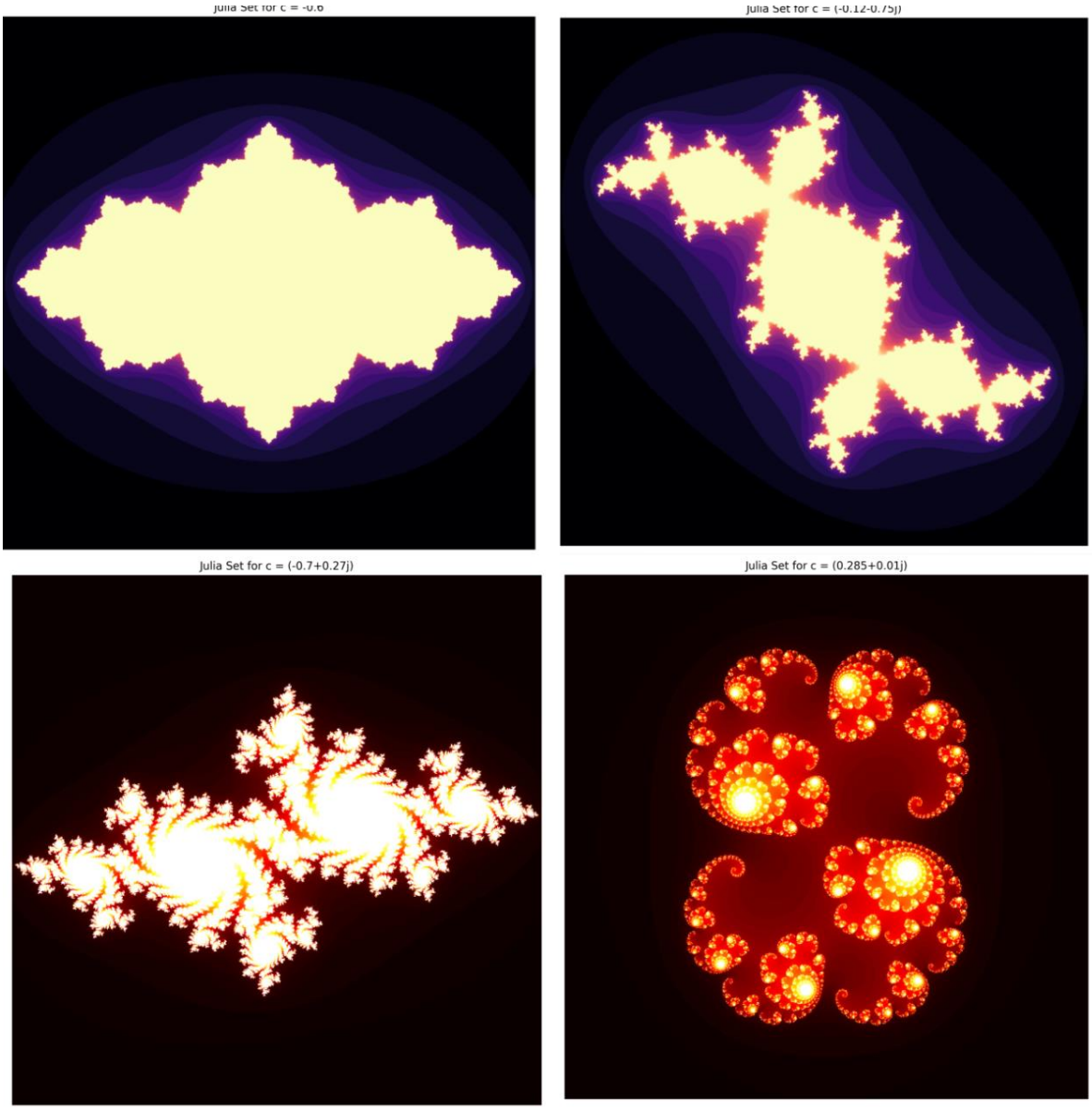
Iteration: تعداد تکرار انجام شده روی هر درایه ماتریس z. هر بار یک درایه از دایره خارج شد تعداد iteration آن در درایه متناظر آن با z ذخیره میشود. برای درایه هایی که هیچ وقت خارج نشدند یک max_iter نسبت داده میشود.

mask: فلگ اینکه آیا هر درایه ماتریس z هنوز از دایره خارج شده است یا خیر.

با این اوصاف و عوض کردن cmap شکل های زیبایی تولید میشوند.



در اینجا مشاهده میشود که عوض کرد علامت قسمت مختلط c فقط روی عوض شدن جهت شکل نهایی تاثیر میگذارد.



با افزایش p نقاط بیشتری فرار میکنند و شکل نهایی تاریک تر میشود.