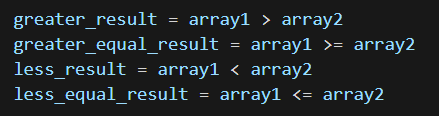
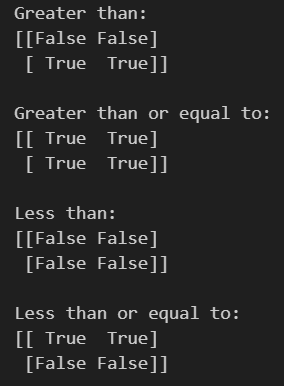
سوال اول)

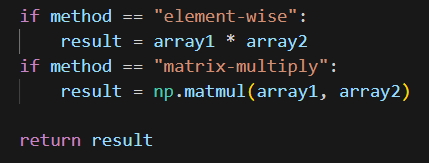


وقتی از numpy استفاده میکنیم، میتوانیم از امکاناتی که این کتابخانه در اختیار ما می‌گذارد، بهره ببریم، یکی از آنها نوع تفسیر عملگرهای مقایسه‌ای است. اگر بین دو آرایه‌ی numpy از عملگر مقایسه‌ای استفاده کنیم، خروجی یک آرایه جدید با همان ابعاد است که به ازای مقادیری از دو آرایه اصلی که آن عملگر مقایسه‌ای برای آنها درست است، مقدار True و در غیر اینصورت مقدار False در آرایه جدید ذخیره می‌شود.

خروجی:



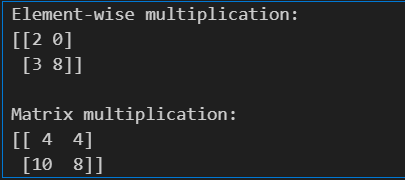
سوال دوم)



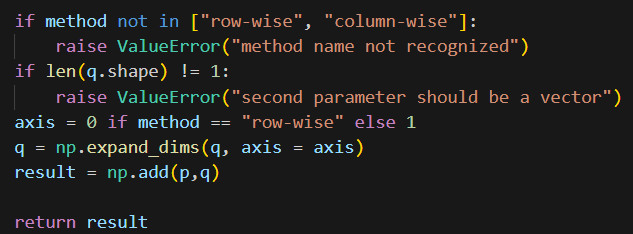
علاوه بر عملگرهای مقایسه‌ای، دیگر عملگرهای ریاضی مثل عملگرهای مقایسه‌ای هم به همان شکل عمل میکنند، برای مثال ضرب دو آرایه با ابعاد یکسان، یک آرایه جدید با همان ابعاد است که مقدار هر خانه‌ی آن برابر با ضرب مقادیر همان خانه در دو آرایه اولیه است.

برای ضرب ماتریسی هم می‌توان از تابع np.matmul استفاده کرد که ورودی آن دو آرایه و خروجی ماتریس حاصلضرب است.

خروجی:



سوال سوم)



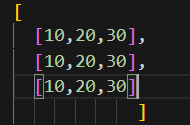
توضیح تابع np.expand\_dims :

این تابع روی یک آرایه numpy اعمال شده و در محور مشخص شده axis ابعاد آرایه را افزایش میدهد، برای مثال اگر ابعاد آرایه ما (3,) باشد، بعد از اعمال این تابع با axis=0 ، ابعاد جدید آن (1,3) خواهد بود و اگر axis=1 بگذاریم، ابعاد جدید آن (3,1) خواهد شد.

همچنین حالات ارور را هم در نظر گرفتم و ارور مناسب را برگرداندم. ارور مربوط به اسم متد و ابعاد ورودی دوم که باید وکتور باشد.

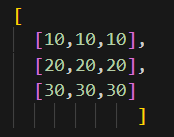
توضیح تابع np.add :

این تابع دو آرایه نامپای را ورودی میگیرد و عملیات جمع را روی آنها انجام می‌دهد و اگر ابعاد دو آرایه یکسان نباشد، باید قابلیت broadcastشدن به یک ابعاد را داشته باشند. برای مثال آرایه q در سوال پس از broadcast به شکل زیر می‌شود:

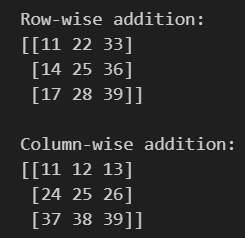


برای همین پس از جمع شدن، به شکلی در می‌آید که صورت سوال خواسته است.

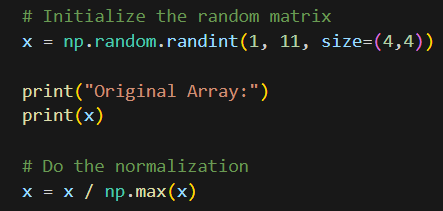
برای حالتی که آرایه q یک بردار عمودی باشد یعنی ابعادش (3,1) باشد، پس از broadcast به شکل زیر در میاید:



خروجی نهایی:



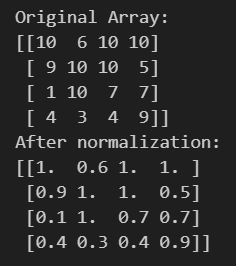
سوال چهارم)



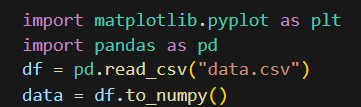
از تابع np.random.randint برای تولید عدد صحیح رندوم استفاده میکنیم. دو پارامتر اول محدوده‌ای را مشخص می‌کنند که اعداد رندوم در آن بازه انتخاب می‌شوند و پارامتر سایز هم ابعاد آرایه خروجی را مشخص میکند. بعد از این دستور، یک ماتریس 4\*4 از اعداد صحیح بین 1 تا 10 خواهیم داشت.

برای نرمالسازی هم کافیست همه مقادیر آرایه را بر مقدار ماکزیمم آن تقسیم کنیم. برای یافتن مقدار حداکثر در آرایه از np.max استفاده می‌کنیم.

خروجی:



سوال پنجم)



برای خواندن فایل csv از کتابخانه pandas استفاده کردم که خروجی آن یک شی از کلاس دیتافریم است که میتوان با استفاده از تابع to\_numpy آن را به یک آرایه numpy تبدیل کرد.

الف) محاسبه بازده روزانه



برای این کار نیاز داشتم، قیمت پایانی همه روزها به جز روز آخر (data[:-1][:,1]) را از قیمت پایانی روز بعدی (data[1:][:,1]) کم کنم، که برای این کار از قابلیت index slicing در نامپای استفاده کردم.

در آخر هم بر همان مقادیر روزهای قبل تقسیم کردم.

ب) میانگین بازده روزانه



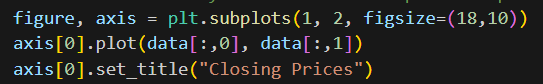
برای میانگین گیری کافیست از تابع np.mean استفاده کنیم و آرایه‌ی بازده روزانه را به عنوان پارامتر به آن بدهیم.

ج) انحراف معیار بازده روزانه



از تابع np.std برای محاسبه standard deviation یا همان انحراف معیار استفاده می‌شود.

د) نمایش قیمتهای پایانی در مرور زمان



برای نمایش نمودار، یک نمودار خطی رسم کردم و این کار با کتابخانه matplotlib و تابع plot در آن ممکن بود. این تابع برای پارامتر اول مقادیر x را میگیرد که همان تاریخ ها هستند که در ستون اول آرایه data هستند و پارامتر دوم مقادیر y هستند که همان قیمتهای پایانی هستند که ستون دوم آرایه data را اشغال میکند.

علت استفاده از تابع subplots این است که میخواستم یک نمودار دیگر را هم نشان بدهم و این توی یک سلول امکان پذیر نیست مگر اینکه از این تابع استفاده کنیم. پارامتر اول آن تعداد سطر و پارامتر دوم تعداد ستون خروجی است و پارامتر سوم سایز کل پنجره است.

از تابع set\_title برای تنظیم عنوان برای نمودار استفاده میشود.

ذ) نمایش بازده روزانه در مرور زمان



مشابه با مورد (د)

ر) محاسبه روزهای با کمترین و بیشترین بازده روزانه



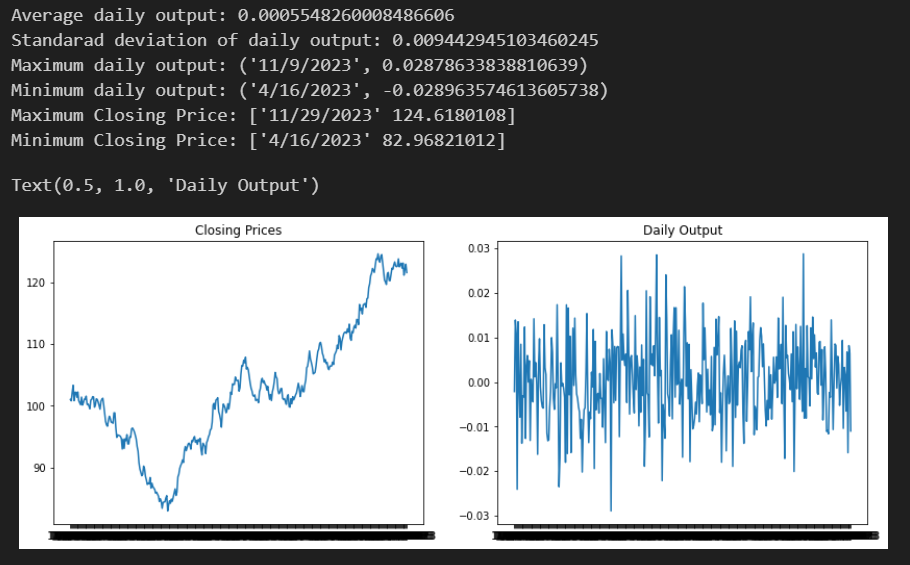
از تابع argmax برای استخراج ایندکسی از آرایه استفاده میشود که حاوی مقدار ماکزیمم است، این عدد را با 1 جمع میکنم، زیرا بازده روزانه برای روز اول تعریف نمی‌شود. سپس از این ایندکس برای یافتن تاریخ مربوط به آن در آرایه data استفاده میکنم. همچنین برای اینکه خود این مقدار ماکزیمم را هم نشان دهم از تابع np.max استفاده کردم که خود مقدار ماکزیمم را خروجی میدهد.

ز) محاسبه روزهای با کمترین و بیشترین قیمت پایانی

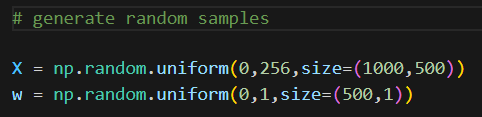


از روشی مشابه با مورد (ر) حل کردم.

خروجی نهایی:

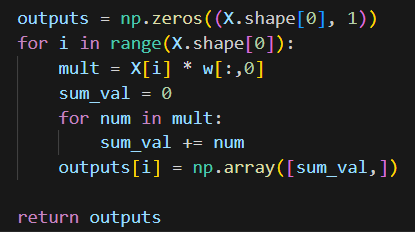


سوال ششم)



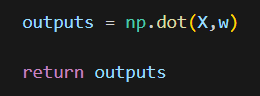
برای تولید اعداد رندوم از تابع np.random.uniform استفاده کردم که مقادیر تولیدی آن از یک توزیع نرمال استفاده میکنند که میانگین آنها صفر است. پارامتر اول و دوم مشخص کننده بازه اعداد رندوم است و پارامتر سوم هم سایز آرایه یا ماتریس خروجی است. وزنها را با مقادیر بین صفر و یک مقداردهی اولیه کردم که در مقداردهی اولیه وزنها در شبکه عصبی معمولا به همین شکل است.

روش اول (استفاده از حلقه for):



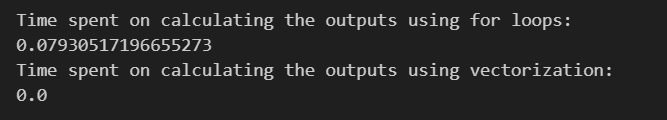
در این روش یک حلقه بر روی تعداد سطرهای ماتریس X میزنیم و هر سطر این ماتریس را در هر ستون ماتریس w ضرب میکنیم و مقادیر حاصل را با استفاده از یک حلقه دیگر جمع میزنیم و در سطر موردنظر در آرایه outputs می‌ریزیم.

روش دوم (vectorization):



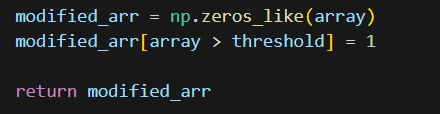
در این روش کافیست از تابع np.dot برای ضرب ماتریسی استفاده کنیم. این کار از لحاظ سرعت در آرایه های بسیار بزرگ، تاثیر بسزایی میگذارد زیرا کتابخانه نامپای سعی کرده پیاده‌سازی این محاسبات سنگین را بهینه انجام دهد.

خروجی:



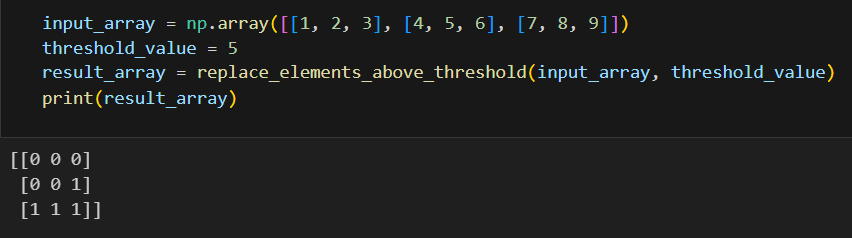
همانطور که مشخص است، زمان انجام عملیات در حالت vectorization بسیار ناچیز است در مقابل روش حلقه زدن.

سوال هفتم)

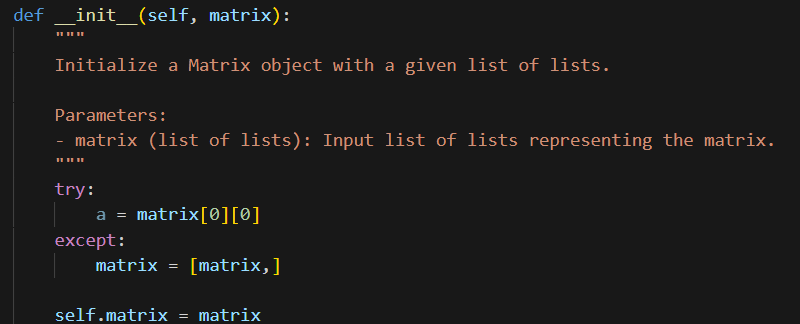


از قابلیت masking در نامپای استفاده کردم، همانطور که قبلا گفتم، عبارت array > threshold یک آرایه جدید با ابعاد array برمیگرداند که در خانه‌هایی که مقدار array از threshold بزرگتر است، مقدار True و در بقیه خانه‌های مقدار False میگیرد. حال این آرایه حاوی مقادیر True/False را که به عنوان ایندکس به یک آرایه دیگر بدهیم، به شکل یک mask عمل کرده، خانه‌هایی که مقدار True دارند، در خانه متناظر در آرایه modified\_arr مقدار 1 را ذخیره کردم. از آنجا مقدار اولیه این آرایه صفر است، دیگر لازم نیست مقادیر کمتر از threshold را برابر با صفر قرار دهم.

خروجی:

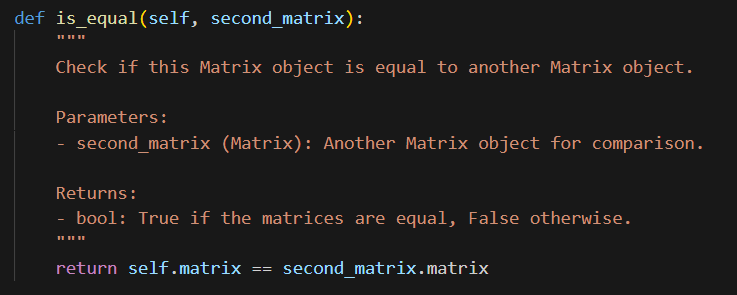


سوال هشتم)



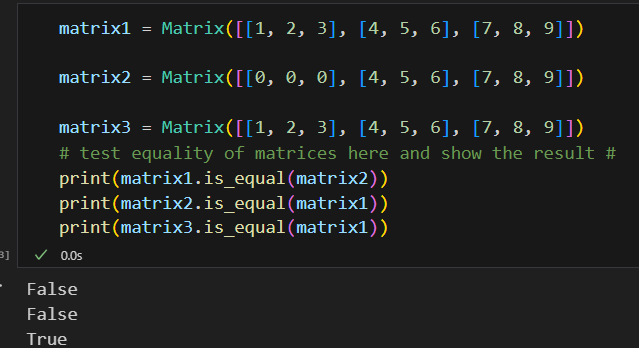
در تابع init ماتریس ورودی را به یک متغیر درون آبجکت self.matrix ریختم. البته اگر ماتریس ورودی یک vector باشد، ابعاد آن را از (n,) به (1,n) تغییر میدهم. این کار در پیاده سازی توابع کلاس کمک میکند.

الف) تساوی دو ماتریس

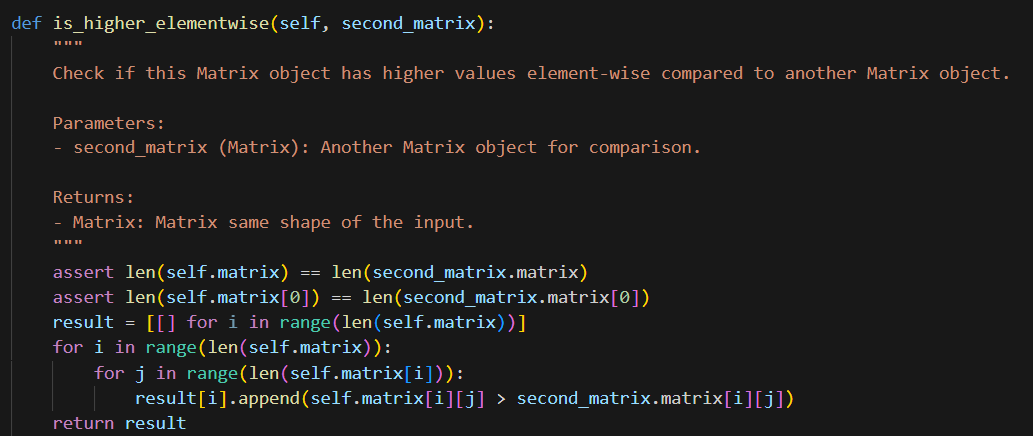


این مورد رو خود پایتون هندل میکرد و کافیست از عملگر == بین ماتریس های درون آبجکت استفاده کنیم.

خروجی:

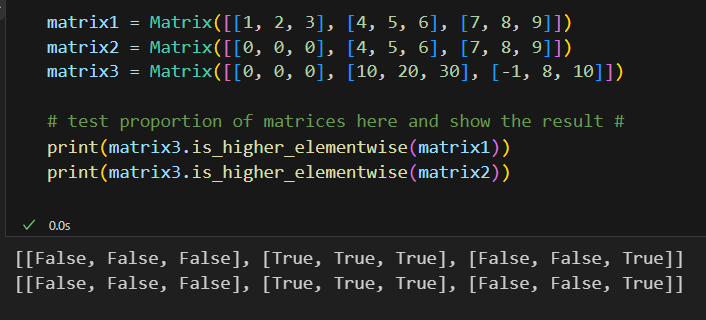


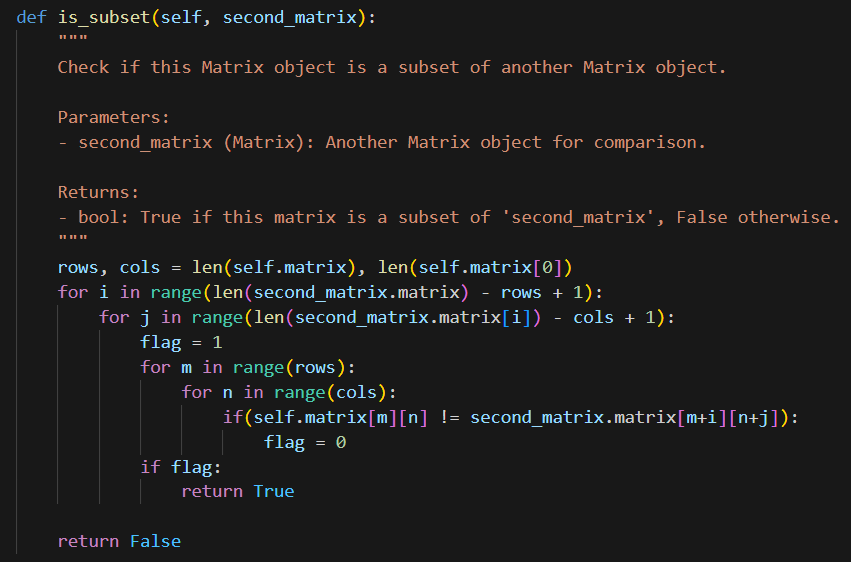
ب) بزرگتر بودن عنصری



ابتدا با استفاده از دستور assert چک کردم که ابعاد دو ماتریس یکسان باشد، سپس آرایه خروجی result را با همان ابعاد ساختم. سپس با استفاده از دو حلقه for به عناصر هردو ماتریس دسترسی پیدا کردم و حاصل مقایسه آن دو را درون خانه متناظر در آرایه خروجی قرار دادم.

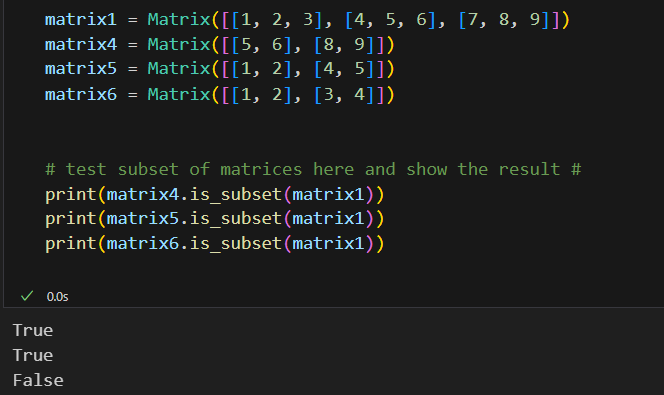
خروجی:



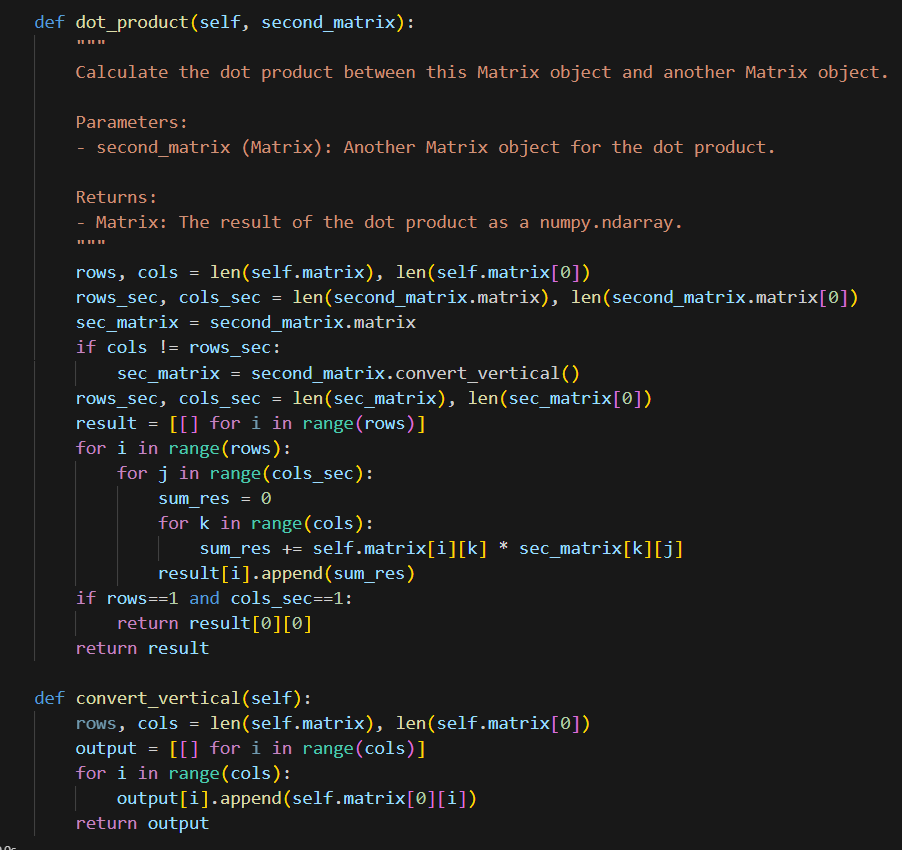
ج) بررسی زیرمجموعه بودن

ابتدا ابعاد ماتریسی که قرار است زیرمجموعه بودن آن چک شود را بدست آوردم. حال فرض میکنیم ماتریس کوچکتر قرار است روی ماتریس بزرگتر بلغزد. برای شبیه سازی این لغزش، باید نقاطی از ماتریس بزرگتر که نقطه بالا چپ ماتریس کوچکتر میتواند روی آن قرار بگیرد را بدست آوریم و یک حلقه روی آن بزنیم که دو حلقه اول برای همین امر است. حال که ماتریس کوچکتر را در یک نقطه از ماتریس بزرگتر ثابت کردیم، وقت آن است که همه اعضای قرار گرفته درون این پنجره را با اعضای خود پنجره مقایسه کنیم و اگر همه آنها مقدار یکسان داشتند، یعنی این ماتریس زیرمجموعه ماتریس دوم است و در نتیجه flag برابر با 1 باقی مانده و وقتی چک میشود که مقدار آن 1 است، مقدار True برگردانده میشود. اگر درون حلقه هرگز چیزی برگردانده نشود، یعنی این پنجره در هیچ نقطه ای منطبق نشده، در این حالت مقدار False برگردانده میشود.

خروجی:



د) ضرب ماتریس



برای ضرب، طبق گفته های اساتید حل تمرین، چهار حالت داشتیم: ضرب ماتریس در ماتریس، ضرب وکتور در ماتریس، ضرب ماتریس در وکتور و در نهایت ضرب وکتور در وکتور.

در دو خط اول، تعداد سطر و ستون هر دو ماتریس را بدست آوردم، حال چک می‌کنیم که تعداد ستونهای ماتریس اول با تعداد سطرهای ماتریس دوم برابر است یا خیر. اگر برابر بودند، مشکلی نداریم و ضرب نرمال ماتریسی را انجام میدهیم که آن قسمت از کد را جلوتر توضیح میدهم. اگر برابر نبودند پس یا ضرب وکتور در وکتور داریم و یا ضرب ماتریس در وکتور، در هر دوی این حالات اگر وکتور دوم را Transpose کنیم، دوباره با ضرب نرمال قابل حل است. برای Transpose کردن هم یک تابع جدا نوشتم به اسم convert\_vertical که روی همان آبجکت اعمال میشود و خروجی Transpose شده را برمیگرداند.

حال که دیگر ابعاد مشکلی ندارند، میتوان ضرب ماتریسی معمولی را انجام داد. از آنجا که ابعاد خروجی ما تعداد سطرهای ماتریس اول و تعداد ستون ماتریس دوم را میگیرد، پس حلقه for را روی این دو مقدار میزنیم.

حال یک حلقه دیگر روی بعد مشترک دو ماتریس میزنیم زیرا میخواهیم اعضای حاصلضرب نقطه ای هر سطر ماتریس اول با هر ستون ماتریس دوم را با یکدیگر جمع بزنیم تا در خانه موردنظر در ماتریس خروجی قرار دهیم.

در نهایت چک میکنم اگر ضرب وکتور در وکتور بود، فقط همون عدد را برمیگردانم تا اینکه یک لیست باشد که فقط یک عضو دارد.

خروجی:

