کلاس GPTree:
این کلاس اصلی ترین بخش برنامه ماست که قرار است با آن هر یک از اعضای جمعیت را بازنمایی کنیم.
هر عضو جمعیت درختیست با ارتفاع 2 تا 5 که گره های میانی آن توابع و گره های برگ، اعداد بین -10 تا 10 و یا متغیر ها را تشکیل میدهند.
در این کلاس ما یک classmethod با اسم random_tree داریم که درخت رندمی میسازد.
این اولین مرحله کار ماست!
بعد از ساخت چندین درخت اولیه که در کد ما این عدد 60 تعیین شده، ما باید بتوانیم این درخت ها را mutate کنیم و یا روی دو درخت عمل crossover را انجام دهیم.
پیاده سازی این توابع هم در این کلاس انجام شده.
مختصری از عملکرد mutation , crossover:
متد mutation با احتمال 0.2 اجرا میشود. در صورت Mutate شدن یک درخت، یا زیرشاخه سمت راست و یا زیرشاخه سمت چپ آن تغییر پیدا میکنند و یک زیردرخت رندم جایگزین آن میشود.
متد crossover به صورت رندوم دو گره از دو درخت را انتخاب میکند، سپس زیردرخت سمت چپ انها را با هم جابجا میکند تا درخت جدیدی از ترکیب این دو درخت به دست آید.

در سراسر این کلاس برای اکثر عملیات ها از احتمال بهره بردیم که با استفاده از random() آنرا پیاده سازی کردیم.

سپس یک تابع برای ساخت جمعیت اولیه را طراحی کردیم که درخت های متنوع با ارتفاع های مختلف میسازد.

تابع fitness روی هر درخت، محاسبه میکند که مقدار واقعی تابع با مقدار خروجی درخت چه تفاوتی دارد؟ برای این منظور از 100 نقطه اولیه استفاده کردیم و میانگین این تفاوت ها را در محاسبه فیتنس استفاده کردیم.
برای اینکه ماکسیمم مقدار تابع fitness برایمان مهم است، ولی اروری که در مرحله قبل توضیح دادیم نیاز به مینیمم شدن دارد، این ارور را در مخرج کسری با صورت یک قرار دادیم تا همواره دنبال بیشتر کردن تابع فیتنس باشیم.

اما چالش اصلی در پیاده سازی این تابع محسابه جمعیت جدید از روی جمعیت نسل قبلی بود.
من در این کد الگوریتم roulette wheel را پیاده سازی کردیم که در آن هر عضو از جمعیت قبلی، در جمعیت بعدی با احتمالی حضور دارد که این احتمال رابطه مستقیم با fitness درخت مورد نظر دارد.
من ابتدا بک جمعیت با تعداد اعضای برابر با نسل قبلی ساختم. سپس با ترکیب هر دو ژن کنار هم در جمعیت، یک عضو جدید در نسل جدید را با crossover دو ژن قبلی ایجاد کردم. بدین ترتیب هر عضو از نسل قبلی ، با عمل selection امکان حضور چند بار را هم در جمعیت جدید دارد.

درمورد چالش ها: ابتدا توابعی که در نظر گرفتم شامل توابع دو عملوندی میشدند و sin , cos را کنار گذاشتم. با افزودن این دو تابع مجبور شدم که برای گره هایی از درخت که تک عملوندی هستند، فقط یک فرزند راست یا چپ بگذارم. و فقط فرزندی از آن که none نیست را برای محاسبه مقدارش در نظر بگیرم.

پس از افزودن توابع جدید ، نوبت به در نظر گرفتن توابع چند متغیره بود که برای این کار تابعی دو متغیره با دو متغیر x , y را در نظر گرفتم، بدین ترتیب تابعی که برای محاسبه dataset نوشته بودم را هم دستخوش تغییر کردم. ولی در کل کار سختی نبود!!!

این الگوریتم به ازای هر تابع ورودی با تا 250 نسل اجرا میشود مگر در طی این حلقه، به درختی بربخوریم که همان تابع اصلی باشد یا به عبارتی فیتنس 1 بدهد. که این کار با چند بار اجرا کردن چندین تابع مختلف رخ نداد و مقدار fitness از 0.8 در تمامی دفعات اجرا به ازای توابع مختلف عبور نکرد.