**به نام خدا**

نام استاد: دکتر سعید پارسا

نام درس: کامپایلر

نام و نام خانوادگی: نیوشا یقینی

شماره دانشجویی: 98522346

گزارش تمرین: HW3

تاریخ: 07/02/1403

**هدف تمرین:**

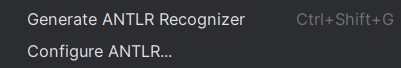
هدف این تمرین این است که داده های ورودی را در قالب جدول خصوصیات آن ها ذخیره کرده و نمایش دهیم. مانند جدول زیر:

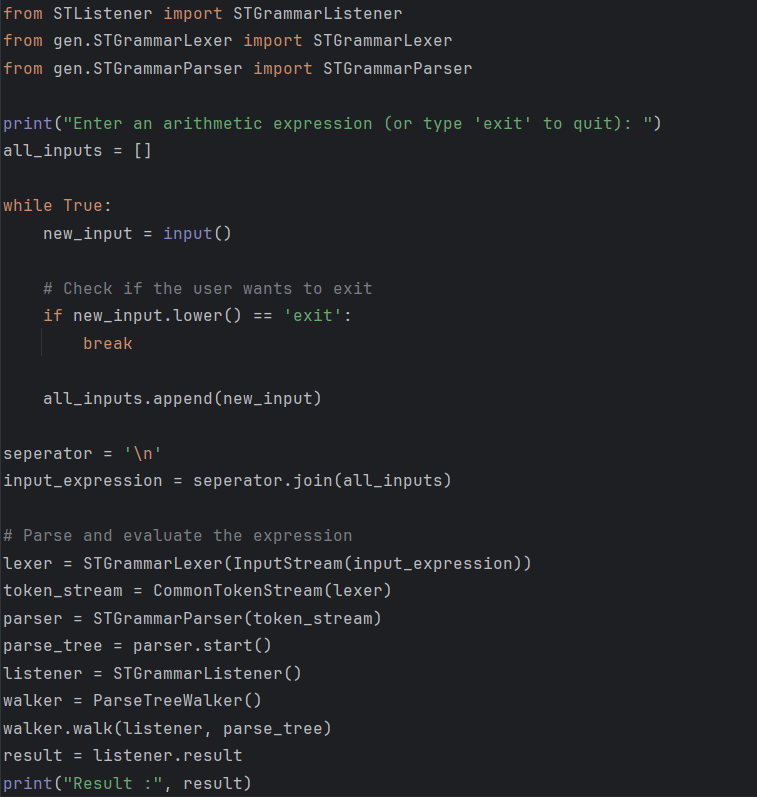
|  |  |  |  |
| --- | --- | --- | --- |
| ID | Name | Type | Value |
| 1 | X | Int | 5 |
| 2 | Y | Float | 6.2 |
| … | … | … | … |

در این گزارش ابتدا مراحل پیاده سازی آورده شده و در انتها نمونه های مثال چک شده و نتیجه گیری آورده شده است.

**مراحل:**

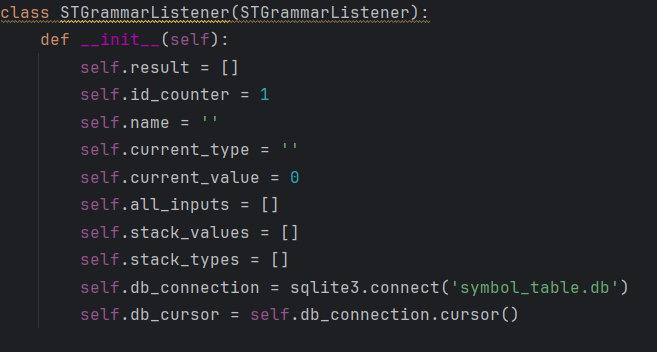
1. ابتدا با استفاده از 2 گزینه زیر فولدر gen را ساختیم.



1. سپس با توجه به تمرین سری قبل فایل پایتون Listener را ایجاد کردیم و توابع exit فایل STGrammarListener از فولدر gen را داخل آن کپی کردیم.
2. مهم ترین نکته برای شروع این است که بتوانیم چند تا ورودی دریافت کنیم، از این رو فایل main.py را به صورت زیر پیاده سازی می‌کنیم:

* برای اینکه بتوان چند تا ورودی دریافت کرد، باید لیستی از تمام ورودی ها ایجاد کرد و در نهایت آنها را بصورت concatenate شده به listener داد، که در اینجا ما با استفاده از دستور join آن را انجام داده‌ایم.
* نکته بعدی آن است که باید یکجا این ورودی گرفتن تمام شود، و شرط خاتمه را وارد کردن exit از طرف کاربر گذاشته‌ایم.
* Separator های مخلتف من الجمله ‘|’استفاده شد، اما هیچکدام قابل separate شدن در داخل listener نبود، برای همین بهترین گزینه استفاده از ‘/n’ به عنوان separator بود.

1. در مرحله بعدی وارد پیاده سازی با توجه به جدولی که باید پر کنیم property های زیر را در تابع initiate اضافه کردیم:



* Id\_counter: این متغیر برای رصد کردن id ورودی هاست.
* Name: این متغیر برای ذخیره کردن نام variable ورودی استفاده می‌شود.
* Current\_type: این متغیر برای ذخیره کردن نوع داده هر variable ورودی استفاده می‌شود.
* Current\_value: این متغیر برای ذخیره کردن مقدار هر variable ورودی استفاده می‌شود.
* All\_inputs: این آرایه برای ذخیره کردن هر ردیف از table نهایی استفاده می‌شود.
* Stack\_values: این پشته برای ذخیره داده های ورودی در پرانتزهای تو در تو تعبیه شده که ترتیب آنها را چک کرده و حفظ کند.
* Stack\_types: این پشته برای ذخیره نوع داده های ورودی در پرانتز های تو در تو تعبیه شده که بتواند با توجه به ترتیب، نوع داده ها را چک کند و در صورت لزوم ارور مناسب را برگرداند.
* دو مورد آخر جهت ذخیره جدول داده ها در sqlite هستند.

1. در مرحله بعد برای پیاده سازی از داخلی ترین توابع شروع می‌کنیم.

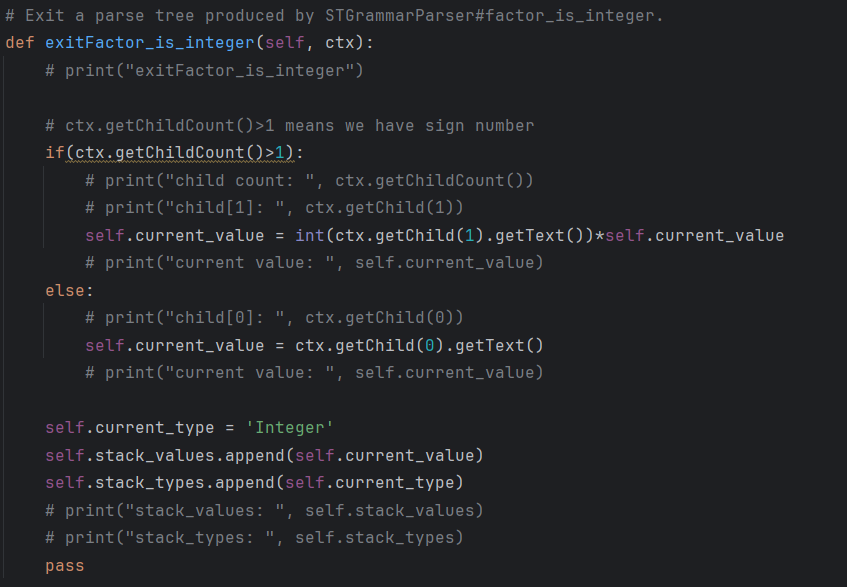
* **exitFactor\_is\_integer:**

زمانی وارد این فانکشن می‌شویم که داده ای از جنس integer به عنوان ورودی دریافت کرده‌ایم، اما نمی‌دانیم آیا این داده یک داده ای از نوع signed است یا unsigned بنابراین با چک کردن تعداد فرزندان ctx این را می‌فهمیم و اگر بیشتر از 1 بود یعنی signed است. برای داده های sign با توجه به تعریفی که در تابع exitSign داریم current\_value را مثبت یا منفی می‌کنیم که اینجا با ضرب داده در current\_value ، قطب عدد مشخص می‌شود.

اگر تعداد فرزندان کمتر از 1 بود یعنی عدد ورودی unsiged است، و دیگر تابع exitSign صدا زده نشده است، برای همین صرفا آن را در current\_value می‌ریزیم. در اینجا بسیار حائز اهمیت است که با استفاده از getText خود داده ها را در stack ها ذخیره کنیم، چرا که بعدا به تداخل خواهیم خورد. (تداخل: به علت وجود پرانتزها و محاسبه مقادیر داخل آنها، بعد از اضافه شدن‌شان به stack دیگر از جنس node نیستند بنابراین هنگامی که می‌خواهیم برابری آنها را که در فانکشن های بعدی مورد نیاز است، چک کنیم، هنگام چک کردن id های تکراری، برابری آنها تایید نمی‌شود.)

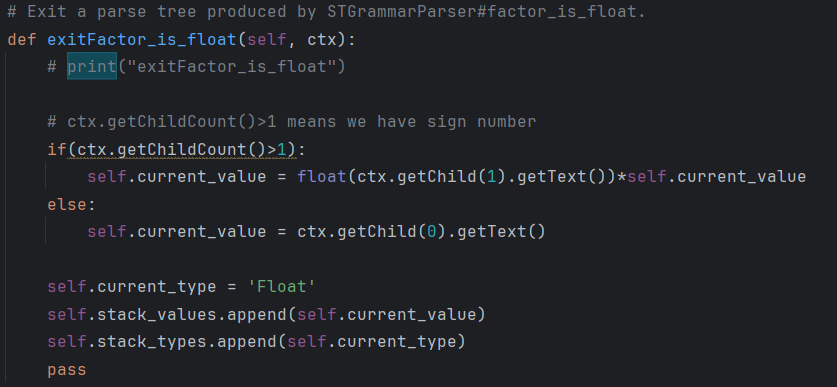
در انتها نیز تایپ داده را در property مربوطه که current\_type است می‌ریزیم.

توجه شود که از آنجایی که این تابع یک تابعی است که ورودی ها را دریافت می‌کند، آن ها را در پشته های مربوطه مقدار و نوع ذخیره می‌کنیم.



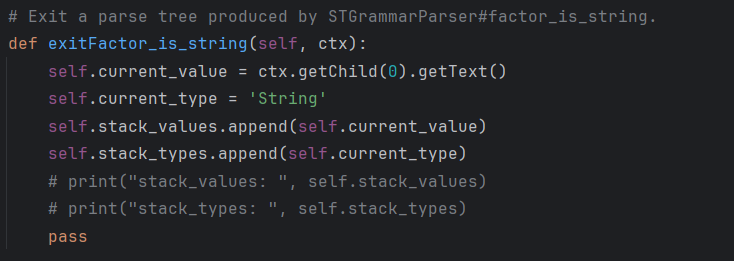
* **exitFactor\_is\_float:**

زمانی وارد این فانکشن می‌شویم که داده ای از جنس float به عنوان ورودی دریافت کرده‌ایم، در این تابع نیز مشابه حالت integer رفتار می‌کنیم.



* **exitFactor\_is\_string:**

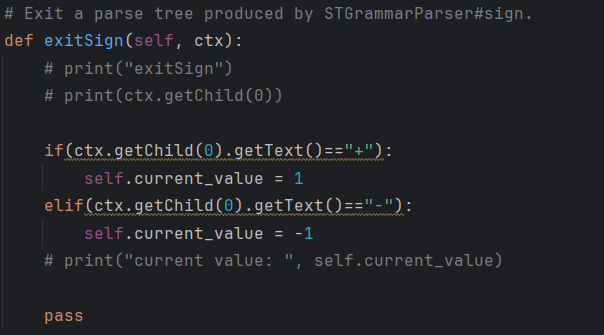
زمانی وارد این فانکشن می‌شویم که داده ای از جنس string به عنوان ورودی دریافت کرده‌ایم، در این تابع صرفا کافیست داده های مربوط به مشخصه های ورودی را در property های مناسب ذخیره کنیم.



* **exitSign:**

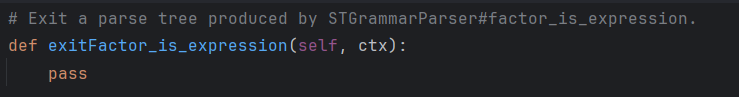
زمانی وارد این فانکشن می‌شویم که داده ی ورودی ما از نوع عددی ست و sign دارد.

برای مشخص شدن آن چک می‌کنیم که sign مثبت است یا منفی، سپس برای مثبت عدد 1 و برای منفی عدد -1 را در current\_value قرار می‌دهیم، این کار انجام می‌شود تا در مرحله بعد که عدد گرفته شد در آن ضرب شود و مثبت یا منفی بودن آن اعمال شود.



* **exitFactor\_is\_expression:**

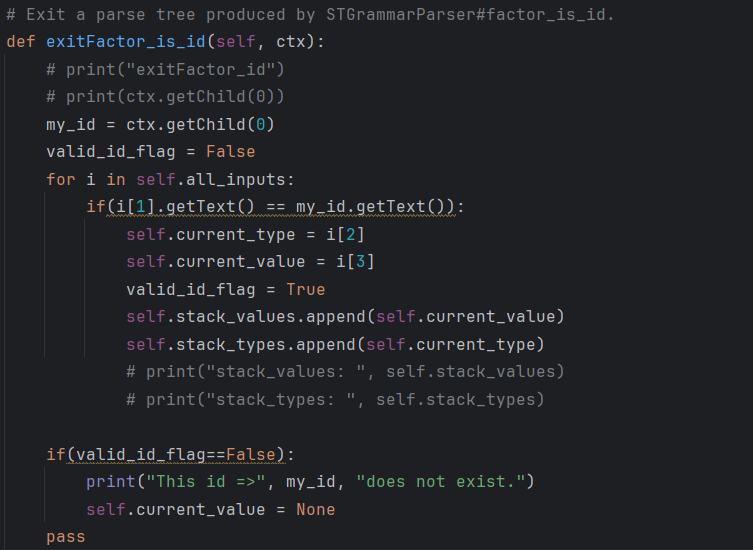
زمانی که داخل پرانتزهای تو در تو باشیم این تابع صدا زده میشود و ctx آن دارای 3 فرزند است، که اولی و آخری پرانتزها هستند، در این تابع تغییری اعمال نمی‌کنیم چرا که هندل کردن اعداد با پشته ها اعمال شده است.



* **exitFactor\_is\_expression:**

این تابع زمانی صدا زده می‌شود که ما داخل ورودی هایمان، ورودی ای از جنس id داشته باشیم، در این حالت باید چک شود که آیا این id داده شده واقعا قبلا در ورودی ها گرفته شده یا نه، و اگر گرفته شده، داده آن بازیابی شود و اگر محاسبه ای نیاز است با آن انجام شود.

در صورت عدم وجود id مقدار دهی شده از قبل، ارور مناسب برگردانده می‌شود.

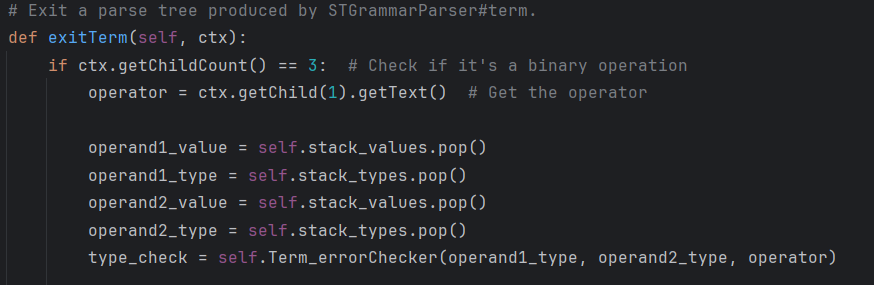


تا اینجا تمام ورودی های مرحله اول گرفته شده و پیاده سازی شده‌اند، حال به مرحله ی پیاده سازی توابع با سلسله مراتب بالاتر می‌رویم.

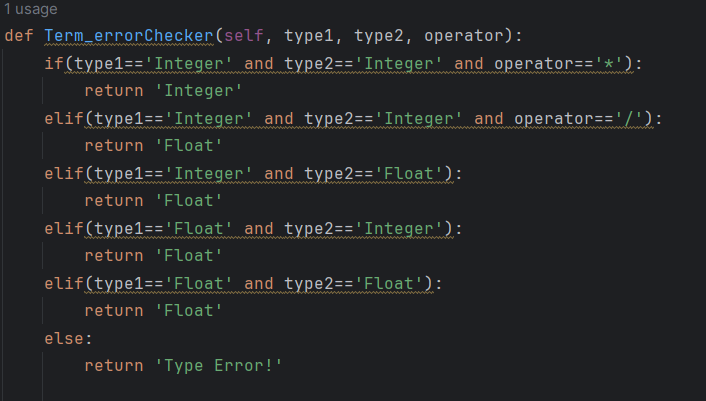
1. در این مرحله ابتدا تابع exitTerm و سپس تابع exitExpr پیاده سازی می‌شود، تابع اول مربوط به محاسبات ضرب و تقسیم است و از لحاظ سلسله مراتبی بالاتر است، تابع دوم نیز مسئول محاسبات جمع و تفریق می‌باشد.

* **exitTerm:**

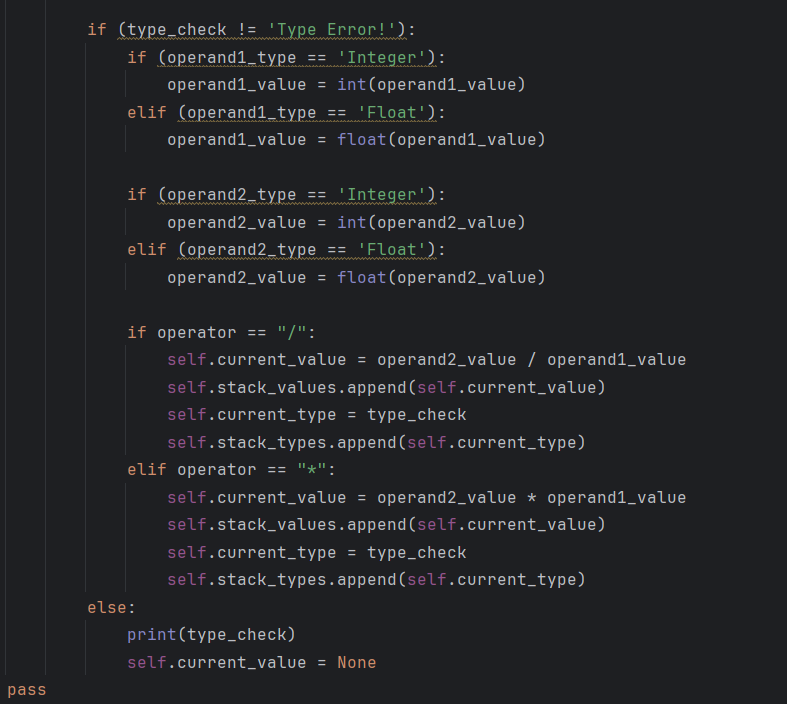
در این تابع ابتدا operator اعمال شده را شناسایی می‌کنیم، سپس دو مقداری که آخر از همه در stack ها ذخیره شده‌اند را pop می‌کنیم، تایپ این مقادیر و operator بدست آمده، تعیین کننده نوع داده بعد از اعمال عملیات است، در ثانی اگر به ارور type error بر بخوریم در اینجا اعمال شده و مشخص می‌شود.



تابع صدا زده شده در اینجا Term\_errorChecker است که برای مشخص کردن نوع داده ی خروجی یا ارور مربوطه در فانکشن exitTerm است. که به صورت زیر شرط گذاری های آن انجام شده است.

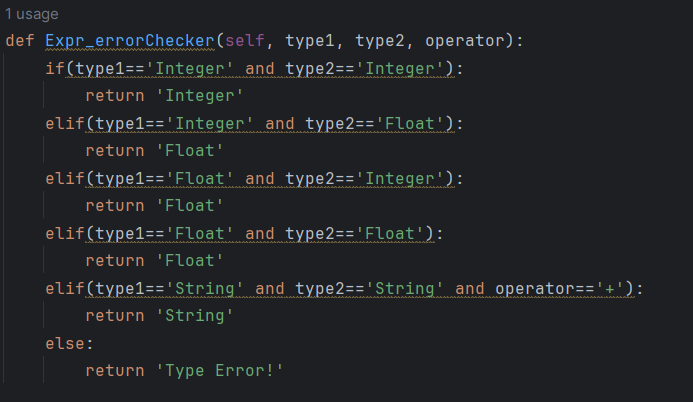


در ادامه تابع exitTerm، در صورت عدم وجود ارور، داده های pop شده را به جنس داده خودشان cast می‌کنیم و سپس عملیات مورد نیاز را انجام می‌دهیم. دقت شود که پس از انجام عملیات ها نوع داده خروجی از این عملیات و جنس آن نیز در property های مربوطه جهت عملیات های آتی ذخیره می‌شود. در انتها نیز در صورت وجود ارور مقدار current\_valut، none می‌شود چرا که برای عملیات بعدی مشخص شود در مرحله قبل ارور داشته‌ایم، و نوع ارور نیز چاپ می‌شود.



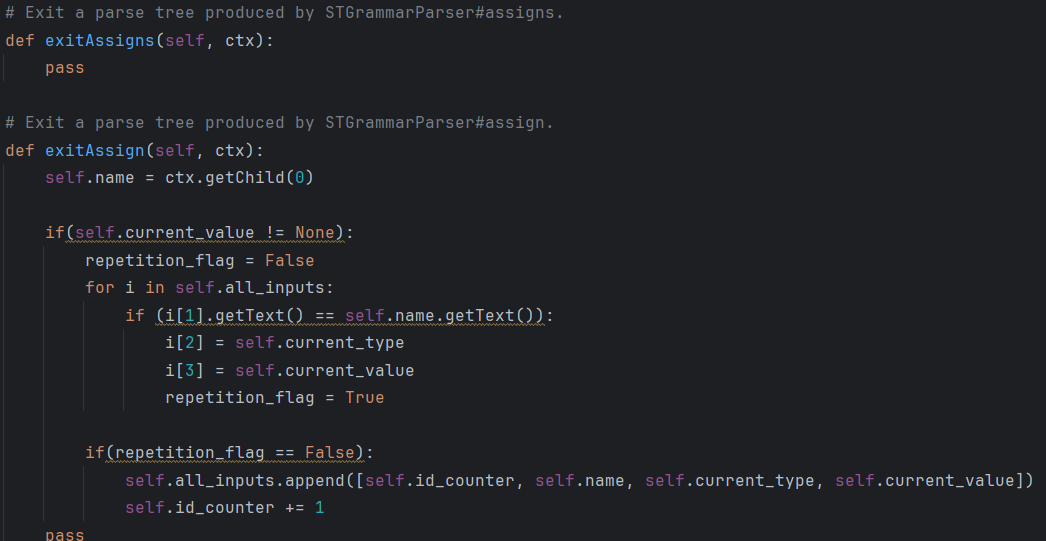
* **exitExpr:**

تابع بعدی تابع exitExpr که همانطور که اشاره شد عملیات های مربوط به جمع و تفریق را انجام می‌دهد. این تابع مشابه تابع exitTerm پیاده سازی شده است، با این تفاوت که از فانکشن Expr\_errorChecker برای چک کردن نوع داده خروجی در تابع exitExpr و وجود یا عدم وجود error type بهره گرفته‌ایم.



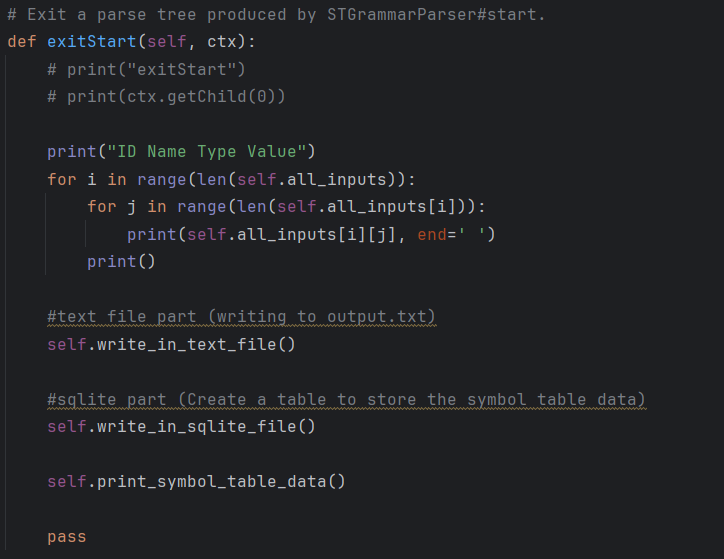
1. مرحله بعدی مرحله پیاده سازی exitAssign و exitAssigns است.

* فانکشن اول، مرحله ای است که ما از یک expression خارج شده‌ایم و حال می‌خواهیم داده های دریافتی و محاسبه شده را در جدول مربوطه بارگزاری کنیم، این جدول یک لیستی است که در property های ما به نام all\_inputs مقدار دهی اولیه شده است، در اینجا همچنین چک می‌کنیم که آیا داده ورودی تکراری هست یا نه، این چک هم برای وارد کردن داده ها به sqlite نیاز است، هم باید چک کنیم که در صورت تکرار id ها و assign دو مقدار متفاوت به آنها مقدار نهایی را به آن اطلاق کنیم.
* فانکشن دوم نیز زمانی صدا زده می‌شود که تمام expression ها تمام شده و در انتهای ورودی ها قرار داریم، پس به عبارتی این فانکشن از لحاظ سلسله مراتبی در سلسله مراتب بالاتری قرار دارد.

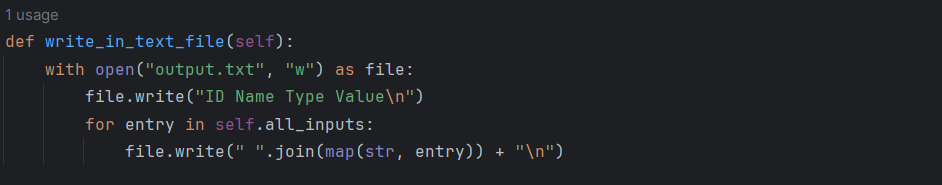


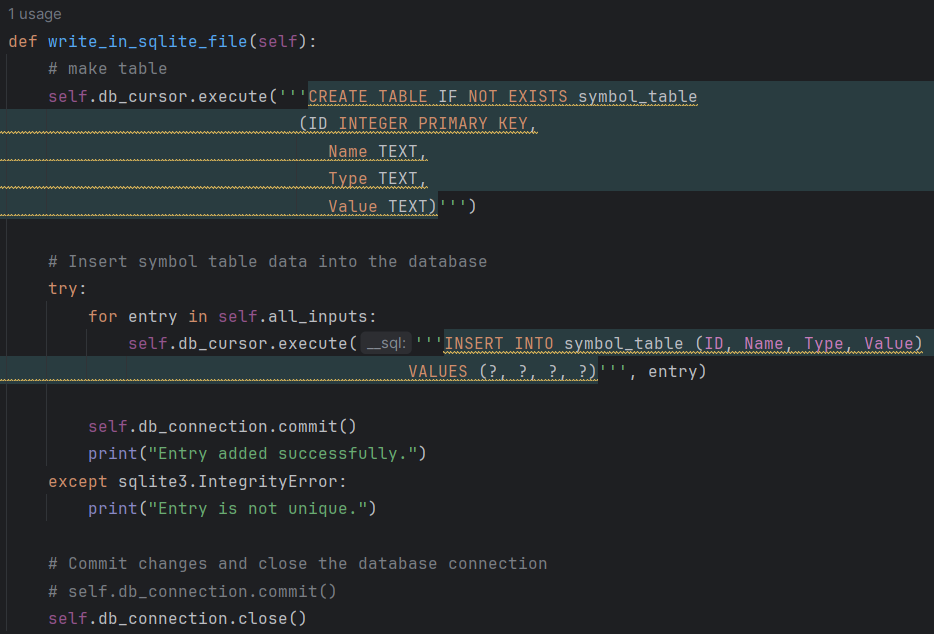
1. مرحله بعدی تابع exitStart است.

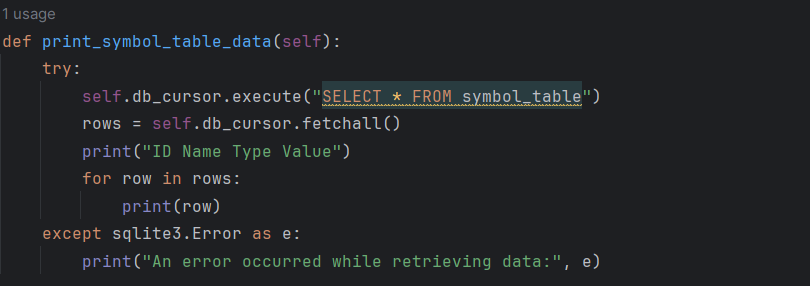
* این تابع زمانی صدا زده می‌شود که کار تمام توابع داخلی به اتمام رسیده باشد، در این تابع ما جدول نهایی را چاپ می‌کنیم، آن را در فایل txt ذخیره می‌کنیم، همچنین در sqlite می‌ریزیم.



* پیاده سازی توابع استفاده شده در این فانکشن نیز به شرح زیر است:

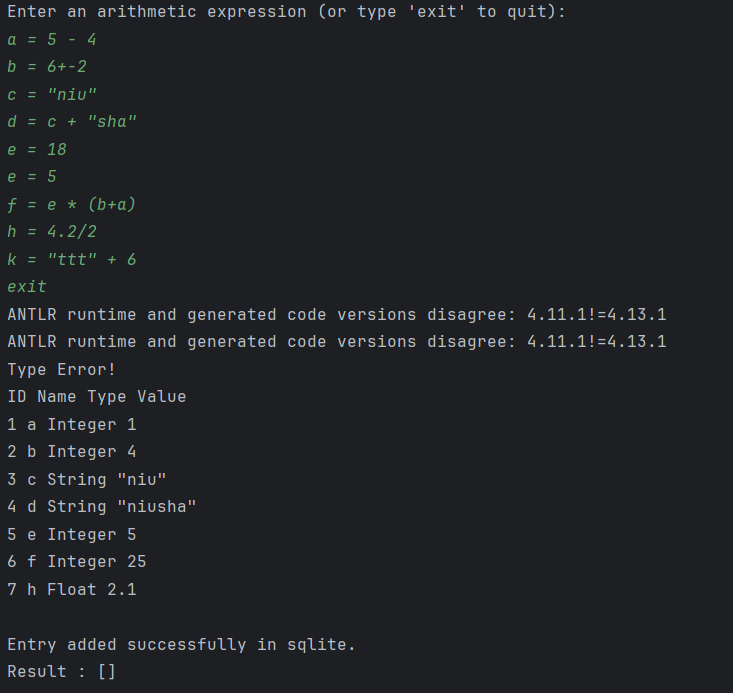




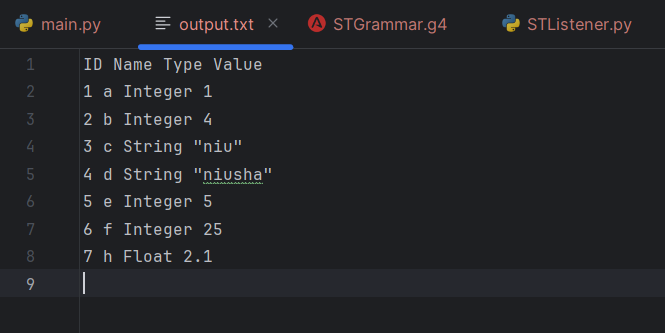


**نمونه اجرا:**

در مثال های زیر تمام حالت های موجود چک شده، نمونه ای هم که ارور داشت در جدول آورده نشده و صرفا ارور آن پرینت شده است.



**خروجی داخل txt:**



**نتیجه گیری:**

* حالت امتیازی برای unary اعمال شده است.
* جلوی حالت های تکراری گرفته شده است.
* در فایل output.txt جدول نهایی ذخیره می‌شود.
* در دیتابیس sqlite جدول نهایی ذخیره می‌شود.