How Python is an Essential Tool for Astronomy

Sophia Marie Guevara Torres - GCER 2026 - Houston, Texas :Team 840

I. Introduction:

It is no secret that computers are a powerful tool that have achieved world domination, and with such power comes the need to learn the language of the computers. Python is a programming language that is widely used at the industry, academic, and research level. At the junction of all these areas of python lies AstroPy – a python library made specifically for astronomy and the interpretation of data such as Hubble's FITS (Flexible Image Transport System) files (raw images taken by the space telescope). However, another area in which AstroPy shines is AstroQuery. The intention of this paper is to utilize AstroQuery to reverse search a star as an example.

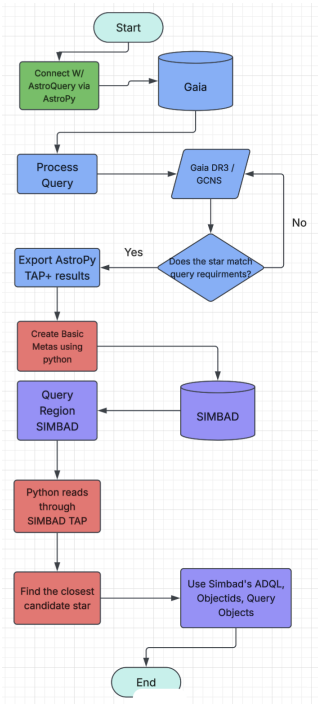


Figure 1

II. The Gaia Archive:

On December 19, 2013, the European Space Agency (ESA) launched the retired probe, Gaia, to survey around 2 billion stars in the Milky Way and beyond. Gaia saved the specific details of each star and saved them to the ESA's archive. The probe released 3 sets of data, each more in detail containing more stars than the last; however gathering data on a vast amount of stars (especially cepheids, binary stars, etc) is not light work. Gaia is equipped with special cameras that take detailed pictures of stars and pick up on changes in the electromagnetic waves coming from the star (this is called radial velocity). As of right now, Gaia is preparing for the 4th Data Release coming December of this year.

III. Introducing ADQL Queries:

AstroPy is the Python library that contains Gaia and the ability to integrate it into Jupyter Notebooks or regular code. Astroquery has many types of ways to access the information (asynchronous search, cone search, metas, query object, etc.) Each comes with their own advantages and disadvantages for specific projects. Async searches are best for a general search for a specific parameter.

```
example = Gaia.launch_job_async("SELECT source_id, ra, dec, parallax FROM external.gaiaedr3_gcns_main_1 WHERE parallax < 20")
job = example.get_results()
print(job)
```

Figure 2

The variable 'example' is the main star of the code. It is what holds the specific query that is sent to the Gaia database and returns the results. It acts for the specific

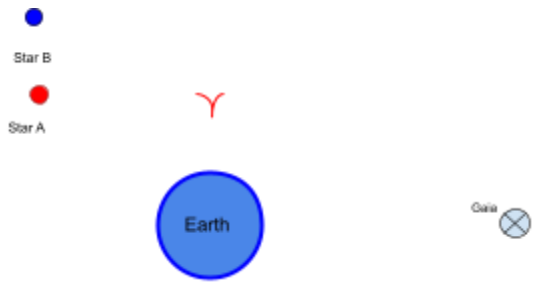
code name of the star, its RA/ Dec, and the parallax. However, it doesn't return every star in GCNS DR3 (Gaia's Catalogue of Nearby Stars, Data Release 3), rather a small portion that has a parallax angle of less than 20 degrees.

source_id	ra deg	dec deg	parallax mas
1965068351068911104	318.00757031414492	38.66293836300947	10.238396
1871996341045033728	317.74889238102924	38.662314050394414	17.139744
1965039282729164416	318.87762732300018	38.78505974138512	11.670069
2164957812656435840	319.10633591349716	40.68274339460371	10.178628
2078740673508865152	297.50624018637745	43.441799238785755	12.731906
2078162085573002240	293.89483719512623	43.24102866517855	13.626671
2078361921807528192	293.87252719429813	44.19994147855522	19.5994
2085320529045977088	300.6473314309398	46.25523793950865	10.148985
...	...	...	...
5048638781190282752	41.851984400709775	-37.67913972813241	10.313603
4953551435071857408	40.20240907944029	-37.54855358427599	14.654312
5048960044743620224	42.29328354687579	-37.00697082149442	12.251339
5049260348857259904	44.21462405227625	-36.03303982898499	14.522108
5045272763780716288	43.7889359073 41	-39.148875518124775	10.486472
5045542247209123072	42.44187881590382	-38.42844330215578	16.841446
5048596965388711296	42.54741871790751	-37.86773023875688	10.942806
5048713135664368768	43.21487278228703	-37.39588443349475	13.3256445
5049106451589286784	43.57706091049634	-36.86332985025791	14.192232
5049005640116420992	42.79479372183268	-36.36494107076062	10.866963

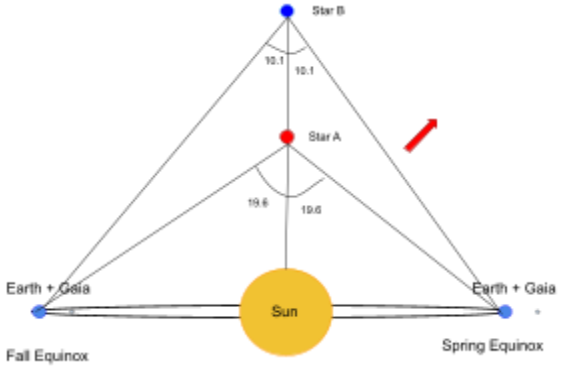
Length = 291081 rows

Figure 3

This is the output of the asynchronous query. The highlighted star is what will be examined. To the most left hand side, it has its source ID (Gaia does NOT get stars by names, that is the job for Simbad which deals with the specifics of each star, not searching for a stars.) RA and DEC give ICRS coordinates of the star in order to be read universally. In this case, the star is 293 degrees away from Ares and it is almost halfway up the sky from its point of view. Since the declination is above 0, it is visible from the Northern Hemisphere only. The parallax is the interesting part: it is the highest (of the table that is shown and not truncated) with 19.6 degrees of difference.



Graphic 1



Graphic 2

Graphic 1 is a representation of the stars' right ascension from the view of Polaris, giving a full 360 degrees of where Star A and Star B lie. Graphic 2 is a view from Ares showing the parallax. Both graphics show that Star A is closer to Earth than Star B because Star A's parallax is larger than Star B's. However, these graphics are only sketches and not proven facts– *yet.* According to Gaia, both stars are extremely close to each other in the night sky, but their main difference is their distance since it is the only parameter that wasn't queried for. The closer a star is, the more noticeable their shift is in 6 months when the Earth is on the other side of its orbit around the Sun.

```python
def distance_from_earth(parallax):
    acension = math.radians(180 - (parallax + 90))
    #Get the missing angle of acension (earth to star)
    #Angel of depression is is star to earth

    s_s = math.atan(acension)
    #Gets the distance from sun to star
    #Shortest leg

    e_s = math.sqrt((1)**2 + (s_s)**2)


    return e_s
#Distance from earth to star (hypotenuse)
```

Figure 4

```python
star_a_info = Gaia.launch_job_async("SELECT source_id, ra, dec, parallax \
                                    FROM external.gaiaedr3_gcns_main_1 \
                                    WHERE source_id = 2078361921807528192")
star_a_info = star_a_info.get_results()
parallax_a = star_a_info["parallax"][0]
d_a = distance_from_earth(parallax_a)
#Gets the generall info for star A to create a trig function




star_b_info = Gaia.launch_job_async("SELECT source_id, ra, dec, parallax \
                                    FROM external.gaiaedr3_gcns_main_1 \
                                    WHERE source_id = 2085320529045977088")
star_b_info = star_b_info.get_results()
parallax_b = star_b_info["parallax"][0]
d_b = distance_from_earth(parallax_b)
#does the same thing for star b, however

print("Distance From Earth to Star A in Astronomical Units:", d_a)
print("Distance from Earth to Star B in Astronomical Units:", d_b)
```

Figure 5

```
INFO: Retrieving tables... [astroquery.utils.tap.core]
INFO: Parsing tables... [astroquery.utils.tap.core]
INFO: Done. [astroquery.utils.tap.core]
INFO: Query finished. [astroquery.utils.tap.core]
INFO: Query finished. [astroquery.utils.tap.core]

Distance From Earth to Star A in Astronomical Units: 1.3371421680362593
Distance from Earth to Star B in Astronomical Units: 1.3782896891405387
```

Figure 6

These calculations prove the initial statement correct: Star A is indeed closer than Star B to Earth. Gaia Async search is very useful and can aid in the general facts of stars, but what if one needs information on a certain star. That's where synchronous searches come in.

While async acts as a general brush, sync works as a comb for finer details. According to AstroPy documentation, async acts on the server's side of the query, so it has more power for longer queries (that sometimes take 5 minutes to load). However, synchronous is where the number of rows that are returned is finite.

```python
star_a = Gaia.launch_job("SELECT top 1 source_id, phot_variable_flag, ref_epoch \
                         FROM gaiadr3.gaia_source \
                         WHERE source_id = 2078361921807528192")
star_a = star_a.get_results()
print(star_a)

print()

star_b = Gaia.launch_job("SELECT top 1 source_id, phot_variable_flag, ref_epoch \
                         FROM gaiadr3.gaia_source \
                         WHERE source_id = 2085320529045977088")
star_b = star_b.get_results()
print(star_b)
```

Figure 7

```
     source_id      phot_variable_flag ref_epoch
                                             yr
------------------- ------------------ ---------
2078361921807528192       NOT_AVAILABLE    2016.0

     source_id      phot_variable_flag ref_epoch
                                             yr
------------------- ------------------ ---------
2085320529045977088       NOT_AVAILABLE    2016.0
```

Figure 8

Above shown is an example of the synchronous search which can handle different AP's (astrophysical parameters) because it is asking for the specifics of star(s) and not specific stars. In this case, both stars are NOT cepheids since their photometry variable is not a number, but both their epoch years are 2016 which means their sky coordinates are verified that year. Async and Sync are very similar in function but have different uses. The most appropriate for this situation of comparing Star A and B would be the cone search and AstroQuery's Simbad. Since, Simbad searches for stars by name while Gaia searches for stars by APs.

IV. SIMBAD

The Set of Identifications, Measurements and Bibliography of Astronomical Data was founded in 1979 (longer than Gaia) in

Strasbourg, France. As the name states, it is another database of stars that has about 5.8 million stars and 5.5 million non stellar objects (11.3 million objects total); however, every star is identified and has a name unlike how Gaia simply tags them with an ID. On python, SIMBAD follows the same ADQL format (SELECT…FROM…WHERE…BY…etc.) as well as use TAP (Table Access Protocol) that Gaia does to ensure that both databases are compatible, which is the intended goal.

V. Gaia, SIMBAD, AND Python

How can one reverse search an engine that relies on the names of stars? Two words: Cone Search. Simbad has a cone search that holds all the names and information of the stars within a designated radius around a specific set of coordinates or a named star. The radius of this cone has to be the distance between star A and B.
Basic astrometry was used to find the distance between Star A and B in degrees, not astronomical units. This was used to find the radius of the query for Simbad. It must be noted that Simbad does not have nearly the same amount as Gaia does in its database because it is a lot more specific on stars. Thus, this search was based not on exactness, but rather an approximation because maybe Simbad does not have the exact star in its database. The archive uses something that Gaia doesn't: the relationships between stars/ hierarchy. It gives insight on how other sources view the celestial body and what it even is. More on that later

```
from astroquery.simbad import Simbad

print()


#The calculation using the parallaxes to find their distance will be used to find the distance of the stars
hori = ask_a["ra"][0] - ask["ra"][0]
print("DISTANCE IN RA:",hori)
print()
verti = ask["dec"][0] - ask_a["dec"][0]
print("DISTANCE IN DEC:",verti)
print()
dist_in_deg = math.sqrt((hori **2) + (verti ** 2))
print("DISTANCE IN DEGREES:" ,dist_in_deg)


query = Simbad.query_region(coordinates= coords, radius = dist_in_deg * u.deg)
```

Figure 9

Below is the distance function but modified to accommodate for degrees instead of astronomical units. Right Ascension and Declination are not compatible with AU, so the formula must be tweaked. Like all cone search jobs, the radius units must be modified– the default for Simbad was arc minutes which is a more galactic approach compared to ICRS which is more earth based.

```
def dist_rad(x1,y1,a = False, b = False):
    if a:
        hori = ask_a["ra"][0] - x1
        verti = ask_a["dec"][0] - y1
        hypo = math.sqrt((hori ** 2) + (verti ** 2))
        return hypo
    elif b:
        hori = ask["ra"][0] - x1
        verti = ask["dec"][0] - y1
        hypo = math.sqrt((hori ** 2) + (verti ** 2))
        return hypo

print()
```

Figure 10

Now it is time for Python to do some heavy-lifting: read over 200,000 rows of celestial objects concentrated in this cone (3.6% of Simbad's entire archive). This loop checks for the Right Ascension only because it is already a very tight condition to follow. Python acts like a sifter on steroids that gets smaller and smaller until the most accurate readings can go through. Through every iteration, the code prints the closest

candidates to be star B in huge chunks of text.

```
for i in range(len(query["ra"])):
    if query["ra"][i] > 300.6473314309397 and query["ra"][i] <300.648:


        if ask["ra"][0] < query["ra"][i]:
            dif = (ask["ra"][0] - query["ra"][i]) * -1


        else:
            dif = (query["ra"][i] - ask["ra"][0]) * -1

        differ = (ask["dec"][0] - query["dec"][i])
        tot = math.sqrt((dif **2) + (differ **2))



        print("NARROWED RA:", query["ra"][i],)
        print("NARROWED DEC:", query["dec"][i])
        print("NAME", query["main_id"][i],)
        print("RA OFF BY", dif, "DEGREES",)
        print("DEC OFF BY", differ, "DEGREES")
        print("TOTAL DISPLACEMENT:", tot, "DEGREES")
        print("DISTANCE FROM A:", dist_rad(query["ra"][i],query["dec"][i], a = True))
        print()
        print()
```

Figure 11

```
DISTANCE IN RA: 2.678823616663067

DISTANCE IN DEC: 5.610264681973931

DISTANCE IN DEGREES: 6.217006174276799
```

Figure 12

```
NARROWED RA: 300.64796026234
NARROWED DEC: 44.68617182788
NAME UCAC4 674-080225
RA OFF BY 0.0006288314002063089 DEGREES
DEC OFF BY 1.5690661116286506 DEGREES
TOTAL DISPLACEMENT: 1.5690662376363793 DEGREES
DISTANCE FROM A: 4.848093768974556


NARROWED RA: 300.64785968378
NARROWED DEC: 45.79040218734
NAME UCAC4 679-076473
RA OFF BY 0.0005282528401835407 DEGREES
DEC OFF BY 0.46483575216865347 DEGREES
TOTAL DISPLACEMENT: 0.4648360523294864 DEGREES
DISTANCE FROM A: 5.800750376248941


NARROWED RA: 300.64771001235994
NARROWED DEC: 44.52877221322
NAME 2MASS J20023545+4431436
RA OFF BY 0.0003785814201364701 DEGREES
DEC OFF BY 1.7264657262886516 DEGREES
TOTAL DISPLACEMENT: 1.726465767796539 DEGREES
DISTANCE FROM A: 4.717834475159082
```

Figure 13

The two screenshots above show the output from figure 10. The first shows the legs and hypotenuse of the triangle used to find the radius for the cone search. It is not as crucial as the second output: the candidates. The most crucial parameter is enclosed in a box and it is the total difference from these coordinates to the Gaia's coordinates of Star B. The farthest is highlighted in orange, the second farthest in yellow, and the most accurate in green. The first time this code was written it was only checking the right ascension, but declination was considered the "tiebreaker". It must be remembered that space (to our perception) is 3 dimensions (with a few more added but it won't be measured in the paper for simplicity purposes). This means that Star B's real name is UCAC4 679-076473. Hooray!

VI. What now?

What comes next is the deep dive on the identity of UCAC4 679-076473. Simbad fully supports queries via the name of a star with query objects and objectids. Both get the specifics of a star but the second gets the hierarchy system briefly explained before. It gathers siblings, parents, children, etc. which are code names for the relationships of stars. If two galaxies are in a pair, they are siblings, the stars in a cluster are the children, etc. Knowing who UCAC4 679-076473 to other stars is, helps identify what it is.

```
para = Simbad.query_objectids("UCAC4 679-076473")
print(para)
```
✓ 0.0s

```
                id
----------------------------
              TIC 240219056
               KIC 9307591
       AP J20023548+4547254
           UCAC4 679-076473
    WISE J200235.48+454725.4
Gaia DR3 2082304465573827712
     2MASS J20023548+4547254
Gaia DR2 2082304465573827712
```

Figure 14

There it is! The final confirmation. Simbad's objectids function helped identify star B as UCAC4 679-076473. Since Simbad is a collection of many sources (including Gaia), it shows that UCAC4 679-076473 is in the DR3 thus proving it is the mystery star B. If one was to review this paper extremely closely, then there may be discrepancies with the source ID in figure 2 and 4 and this screenshot above. This is because the source used for Gaia was not the complete DR3, but the GCNS which is a subcategory.

With this in mind, the endless interrogation of UCAC4 679-076473 can go on. Below will be a few examples of how Simbad can be used with small explanations.

```
para1 = Simbad.query_hierarchy("UCAC4 679-076473", hierarchy= "parents")
print(para1)
print()

para2 = Simbad.query_hierarchy("UCAC4 679-076473", hierarchy= "siblings")
print(para2)
print()

para3 = Simbad.query_hierarchy("UCAC4 679-076473", hierarchy= "children")
print(para3)
```

Figure 15

This code is an example of Simbad's relationships between items in the archive. Parents are the star clusters the star is a part of, siblings are a pair, and children are the individual stars inside the cluster.

Figure 16

Unfortunately, the results are very unsatisfying. UCAC4 679-076473 is just and not related to any other items in the archive. The highlighted red is actually a warning saying that the star is not a part of any group. The query was sited and done correctly, but it does not yield any results.

```
adql = "SELECT basic.ra, basic.dec, basic.otype, basic.rvz_redshift FROM basic WHERE main_id = 'UCAC4 679-076473'"
#Basic ADQL query for simbad
query = Simbad.query_tap(adql)
print(query)

print()

extra = "SELECT basic.main_id, fe_h FROM basic JOIN mesFe_H ON basic.oid = mesFe_H.oidref WHERE main_id = 'UCAC4 679-076473'"
#The Simbad chart wants the user to connect to basic first \
#then connect to the extra table using the oid or the arrow pointing to it
#And then vice verso
ex = Simbad.query_tap(extra)
print(ex)
```

Figure 17

```
       ra            dec        otype       rvz_redshift
      deg            deg
--------------- -------------- ------ ------------------------
300.64785968378 45.79040218734   RG* -0.00017237373989398286

    main_id       fe_h
--------------- ------
UCAC4 679-076473  0.164
UCAC4 679-076473  0.133
UCAC4 679-076473  0.198
UCAC4 679-076473   0.09
UCAC4 679-076473 0.1064
UCAC4 679-076473  0.089
UCAC4 679-076473  0.196
```

Figure 18

Simbad, like Gaia, uses the language of ADQL to find even more specifics of a star. What is being searched are the advanced and basic ways of getting a query. Simbad has multiple TAPs that have different specifics of a star. Basic is the most basic that has the generals of a star while the other tables (and in this case Fe_H) are connected through OID and OIDREF. Both must be called and be set to equal each other in order to reach through the web of tables to gather it. Figure 16 shows how in

order to call fe_h (the metallicity of a star, or how much of it is composed of elements heavier than Helium), mesFe_H (the name of the table) must be connected via basic.oid (the basic table) and mesFe_h.oidref (the added table).

What the data in figure 17 shows is how UCAC4 679-076473 is a Red Giant star (shown by its otype) and that it is coming towards the earth because of its blueshift (negative radial velocity). This information would not have been gathered and confirmed if not for the bridge between Gaia. Simbad, and Python.

VII. Real Life Application

The most obvious question is how this applies to today's world. Python is actively being pushed to be more mainstream thus its uses and libraries are being exposed as the demand from different career options such as psychology, graphic design, mathematics, etc. experts create their own libraries (PsychoPy, MatPlotLib, NumPy, etc.). AstroPy is just part of that demand and the result of multiple volunteers asking for better accessibility for astronomy.

What AI is doing is both simplifying how one can learn to read these libraries but oversimplifying them to the point it misses key details especially in the ADQL format. On the official Gaia website, direct queries can be made but they both have different APs with different data types. Gemini, (Google's AI), reports that in order to retrieve a star's luminosity the AP was "lum_flame"; however AstroQuery did not accept the keyword, but Gaia's Direct Processing did. To Gemini, both AstroQuery's Gaia and ESA's Gaia are the same: that is not true. Thus, a beginner can have aid from AI to be introduced into the python library, but a resistance to completely rely on it must be made in order to avoid getting stuck trying to figure out what works and what doesn't.

VIII. Sources:

ESA - Gaia
AandA.org
AstroPy TAP
AstroQuery - Gaia
IVOA - ADQL Syntax
SIMBAD Article
AstroQuery - SIMBAD
SIMBAD TAP Search - Tables