Exercise 1

Get our example basicsiap.py from the notes.

Now find an image service publishing the ROSAT survey and pointed observations and see if it has an image for the position given (or try some other service and position you are actually interested in).

Use WIRR to search the VO Registry for now.

What is coming back from SIAService's `search` is a sequence of SIARecords. Have a quick look at its pyvo documentation and make your program print the file size and the instrument name rather than calling `cachedataset`.

Exercise 2

Get the globalsiap.py script from the attachment and change it so it skips 90% of the services discovered randomly (use `random.random()`). Also, remove the constraint on the date (we don't need that here) and change the position to something you are interested in or expect to have pretty pictures (M1 or M51 are always good candiates). Run the thing and see what you find.

Exercise 3

Get the pyVO source code and find the source of pyvo.samp. Start TOPCAT, find the implementation of the `connection` context manager, and then open a SAMP connection manually from an interactive Python prompt. And then again, and a third time. What do you observe in TOPCAT?

Hint: To get the source code, try:

```
git clone https://github.com/astropy/pyvo.
```

Or, on Debian-derviced boxes:

```
apt source python3-pyvo
```

Exercise 4

Still in `samp.py`, inspect how `send_image_to` is implemented. From reading the code, can you figure out how to only send the image to Aladin? If you can, try your solution in globalsiapsamp.py by having Aladin and ds9 (Debian package: saods9) open at the same time.

Hint: To find out Aladin's client name, check TOPCAT's SAMP status window.

Exercise 5

Write a program that prints the number of rows in the table `arihip.main` in the TAP service at http://dc.g-vo.org/tap (do *not* pull all the rows and use python's len).

Hint: With ADQL's `AS` construct you can control the names of table columns.

Exercise 6

The following program should print URIs and titles for images in some collection for whatever names are in `OBJECTS`:

```
import pyvo

OBJECTS = ["IC 4756", "NGC 3377"]
QUERY = """select accref, imagetitle
    from maidanak.reduced
    where object={object}"""

svc = pyvo.dal.TAPService("https://dc.g-vo.org/tap")
for object in OBJECTS:
    print(svc.run_sync(QUERY.format(**locals())).to_table())
```

What really happens: An error message. Can you figure out where it comes from and how to fix things?

Exercise 7

Use TOPCAT's TAP data browser to locate services and table names for TGAS and RAVE. Also figure out where the positions and some usable magnitude are, plus the proper motions from TGAS and the radial velocities from RAVE.

Re-write fetch3.py to query the retrieve all stars between 8 and 8.2 mags from each table. Also, send the results to Aladin (which is known as *Aladin* (capitalised) on the SAMP bus). See if you can get a nice plot of rv, pmra, and pmdec.

Hint: Check Aladin's Catalog/Create filter for fancy plotting options.

Exercise 8

Go through the source code of fetch3-cluster.py. You will see we have put in two workarounds for where the data providers messed up. Can you see in each case what might have gone wrong? Have the service operators fixed their software or do things still fail when you remove a workaround? In a course setting, coordinate with your neighbours and split up the work so each only looks at one workaround.

Exercise 9

Run fetch3-cluster.py and select a couple of objects. Keep the resulting file (`selected_positions.vot`) – we will want to reuse it later.

Exercise 10

You can use URLs in a query's upload argument. To try this out, review the TGAS and RAVE exercise 7. Let the initial RAVE query be asynchronous. On the resulting job, call `wait` as above. Once it is done, upload what is job's `result_uri` attribute into the TGAS server with a normal positional upload join.

Exercise 11

Can you change `get_spectra.py` such that only spectra of resolving power 10000 or greater are retrieved?

Hint: Use TOPCAT or the `tables` property of your `TAPService` to inspect the metadata of the `ivoa.obscore` table to figure out which column to query against. Just in case: It is almost always better to filter on the remote side rather than the local side. And chuck the "almost" if the constraint can be expressed as a single condition in a WHERE clause.

Exercise 12

The action of the SAMP handler is in the `make_response_table` method; have a brief look at it to appreciate what is going on. Then, replace what is there with something that does a SIAP search on the service at http://dc.g-vo.org/lswscans/res/positions/siap/siap.xml and returns the corresponding table for sending to Aladin (hint: remember the `to_table` method of DAL results).

Exercise 13

Listening to the SAMP message *coord.pointAt.sky*, implement an "odometer" computing and printing after each step the distance travelled by the pointer.

To do this, you will need to keep the SAMP connection, the last position and the distance travelled so far as state; take the vicinitysearcher, remove the code keeping the state and behaviour used for its function, and insert our new logic.

Hints: Look at SkyCoord in Astropy and the mtypes page; when re-using SAMP bindings, make sure you handle messages, not calls.

Exercise 14

Can you figure out the default output limit (i.e., in effect an implied TOP) for the TAP service at http://dc.g-vo.org/tap? How far can you raise it?

Can you write a program that figures it out for all TAP services out there that talk about tgas?

Exercise 15

Which IAU constellation is the least massive exoplanet in the exoplanet merged catalogue in? Try solving this using pyVO's registry API; hint: to figure out constellations, having the constellations as ADQL polygons is really handy.

Exercise 16

(You will need to have looked at the vocabularies sidetrack for this)

Take new-constraint.py and add support for query expansion: add a keyword argument `expand`. If that is true, include the narrower concepts of what was passed in, too.

Hint: You can leave (something like) this to the server with a UDF, or you can do the query expansion locally; the first way is simpler, the second perhaps more instructive.

Exercise 17

Write a function `get_available_semantics(dl) -> set` returning a set of the semantics available for a given datalink.

Try your program on the SSA example from the lecture.

Exercise 18

Get the soda-with-rows.py script for doing cutouts on CALIFA DR3 and make a false colour image for IC 1151 by taking the slices from the COMB cube (see the setup column) at 400 nm as blue, at 550 nm as green, and at 700 nm as red. Do not download the whole cube, use SODA to just retrieve exactly what you need.

Exercise 19

In multitap.py, have a look at `get_services_and_tables`; in there, we are doing a grouping operation on the client (i.e., our) side. Can you move to to the server side using GROUP BY and the `ivo_string_agg` UDF?

Exercise 20

Can you find out the strings you need to pass to `get_feature` find find out whether a service supports the nifty `IN_UNIT` function?

Exercise 21

There is one glaring hole in our multitap script: Units. Try to improve on this: If the service supports `IN_UNIT`, use it in about the way we have been using CAST.

If you actually need something like this, you can of course also compute the conversion factors locally (using astropy.units) and bake them into the queries. Feel free to try that, too.

Exercise 22

Get the epnquery.py and change it to only discover spectra (that's dataproduct type *sp* in EPN-TAP). then send the first two spectra your program finds to TOPCAT (or SPLAT, or CASSIS, if you have one of them).

Exercise 23

The SSAP service at http://dc.g-vo.org/theossa/q/ssa/ssap.xml? houses theoretical spectra mostly of hot, compact stars.

See if you can retrieve three spectra for stars with `log_g` between 4.5 and 5.5, an effective temperature between 7×10^4 and 10^5 Kelvin, and a Nitrogen mass fraction larger than 0.015 dex (write `+Inf` for "no upper limit").

Send the spectra retrieved to splat.

Hints: Use `viewparams.py`, start from `siapextra.py`, remember `dal.ssa.SSAService`, and pass in `FORMAT='VOTable'` to avoid retrieving spectra in both FITS and VOTable.

Exercise 24

Add full Gaia records from ivo://esavo/gaia/tap's DR3 `gaia_source` to some records from the hdgaia.main table on GAVO's data centre. This does not need any slicing; still, only upload what you actually need for matching; for that, the smart-tap-upload.py example should be helpful.

Hint: for our simple table.join to work (which needs the same name in both tables), it is probably smart to rename `source_id3` in hdgaia at the ADQL level.