PyVO

1st ASTERICS-OBELICS International School

Markus Demleitner *(msdemlei@ari.uni-heidelberg.de)*

Hendrik Heinl *(heinl@ari.uni-heidelberg.de)*

Agenda

- Short introduction to the VO
- Introduction to PyVO
- Synchronous TAP queries on three services
- Asynchronous TAP queries and SED selection
- Get PyVO

A Definition

The Virtual Observatory (VO) is (or will be), a

comprehensive set of
data and **services**
relevant to **astronomy**
accessible from **clients** of **your choice**
regardless of where you are and
preserving products of digital astronomy.

What's PyVO?

Astropy affiliated package for accessing Virtual Observatory data and services.

PyVO provides APIs for:

- Simple Image Access (SIA)
- Simple Spectral Access Protocol (SSAP)
- Simple Cone Search (SCS)
- Simple Line Access Protocol (SLAP)
- Table Access Protocol (TAP)*
- Simple Application Messaging Protocol (SAMP)*

The Use Case

Integrate results from potentially lots of different sources.

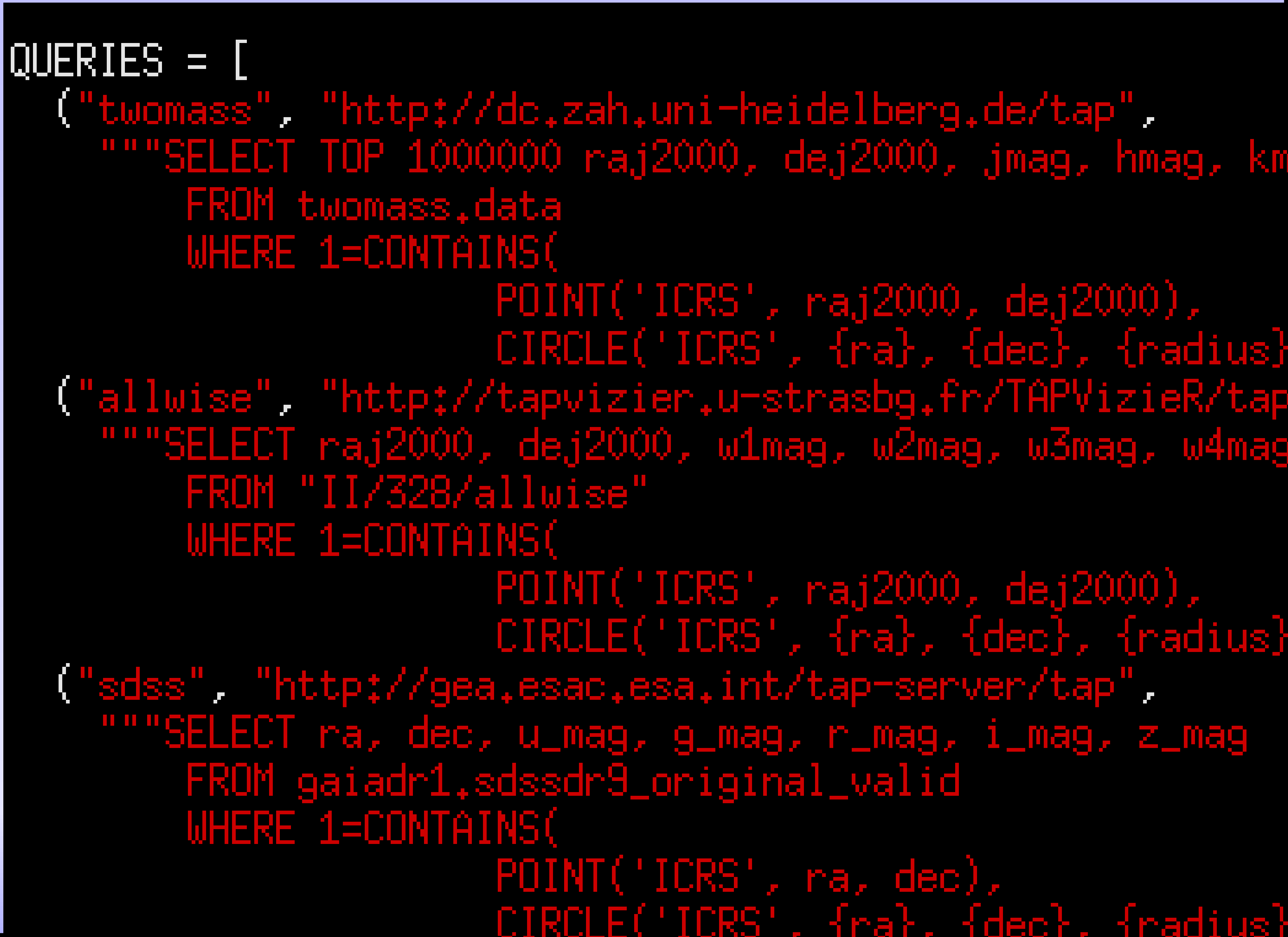
The example: Take photometry from three catalogs from three services using TAP.

Use different methods:

- Synchronous Queries + SAMP
- Asynchronous Queries combined with „your“ code
- Search for Spectra

Synchronous Queries

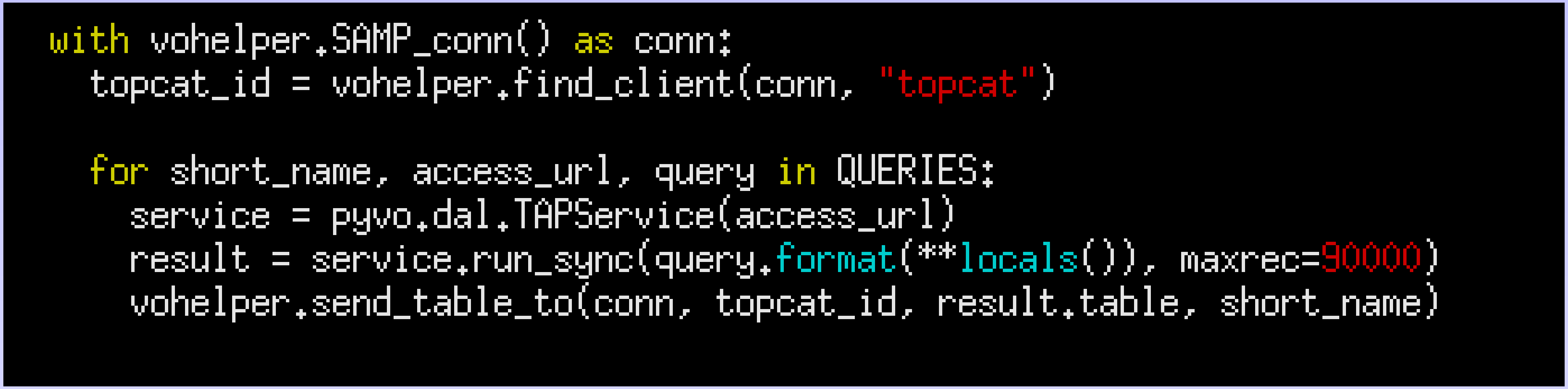
Query three catalogs and send the result to TOPCAT via SAMP.



```
QUERIES = [
  ("twomass", "http://dc.zah.uni-heidelberg.de/tap",
    """SELECT TOP 1000000 raj2000, dej2000, jmag, hmag, km
        FROM twomass.data
        WHERE 1=CONTAINS(
                      POINT('ICRS', raj2000, dej2000),
                      CIRCLE('ICRS', {ra}, {dec}, {radius}
  ("allwise", "http://tapvizier.u-strasbg.fr/TAPVizieR/tap
    """SELECT raj2000, dej2000, w1mag, w2mag, w3mag, w4mag
        FROM "II/328/allwise"
        WHERE 1=CONTAINS(
                      POINT('ICRS', raj2000, dej2000),
                      CIRCLE('ICRS', {ra}, {dec}, {radius}
  ("sdss", "http://gea.esac.esa.int/tap-server/tap",
    """SELECT ra, dec, u_mag, g_mag, r_mag, i_mag, z_mag
        FROM gaiadr1.sdssdr9_original_valid
        WHERE 1=CONTAINS(
                      POINT('ICRS', ra, dec),
                      CIRCLE('ICRS', {ra}, {dec}, {radius}
```

Run Queries and Use SAMP

Running the queries and sending the results to TOPCAT is actually close to trivial with a bit of support code:



```
with vohelper.SAMP_conn() as conn:
  topcat_id = vohelper.find_client(conn, "topcat")

  for short_name, access_url, query in QUERIES:
    service = pyvo.dal.TAPService(access_url)
    result = service.run_sync(query.format(**locals()), maxrec=90000)
    vohelper.send_table_to(conn, topcat_id, result.table, short_name)
```

Asynchronous Jobs and „your“ code

When doing a lot of queries or long-running queries, it may be a good move to run them asynchonously and in parallel. We will

- run Asynchronoues Jobs on three services,
- use your code to cluster the received positions,
- build SEDs,
- plot SEDs and select objects of interest.

Given async is a fairly complex pattern and this kind of orchestration is fairly advanced, the actual code is quite simple.

Asynchronous Jobs continued

Create jobs, start them, memorize short names and jobs:

```
jobs = set()
for short_name, access_url, query in QUERIES:
  job = pyvo.dal.TAPService(access_url).submit_job(
    query.format(**locals()), maxrec=9000000)
  job.run()
  jobs.add((short_name, job))
```

Asynchronous Jobs continued

The poll the status of the jobs until all are done:

```
with vohelper.SAMP_conn() as conn:
  topcat_id = vohelper.find_client(conn, "topcat")

  while jobs:
    for short_name, job in list(jobs):
      print short_name, job.phase
      if job.phase not in ('QUEUED', 'EXECUTING'):
        jobs.remove((short_name, job))
        vohelper.send_table_to(
          conn,
          topcat_id,
          job.fetch_result().table,
          short_name)
        job.delete()

    time.sleep(0.5)
```

Combine with „your“ Code

The cool thing about doing this in python is that you can easily do your own logic now.

Here: Cluster by position so you get SEDs. Then, display the curves and let the user interactively select „interesting“ cases.

Cluster by Position

Get coordinates through UCDs:

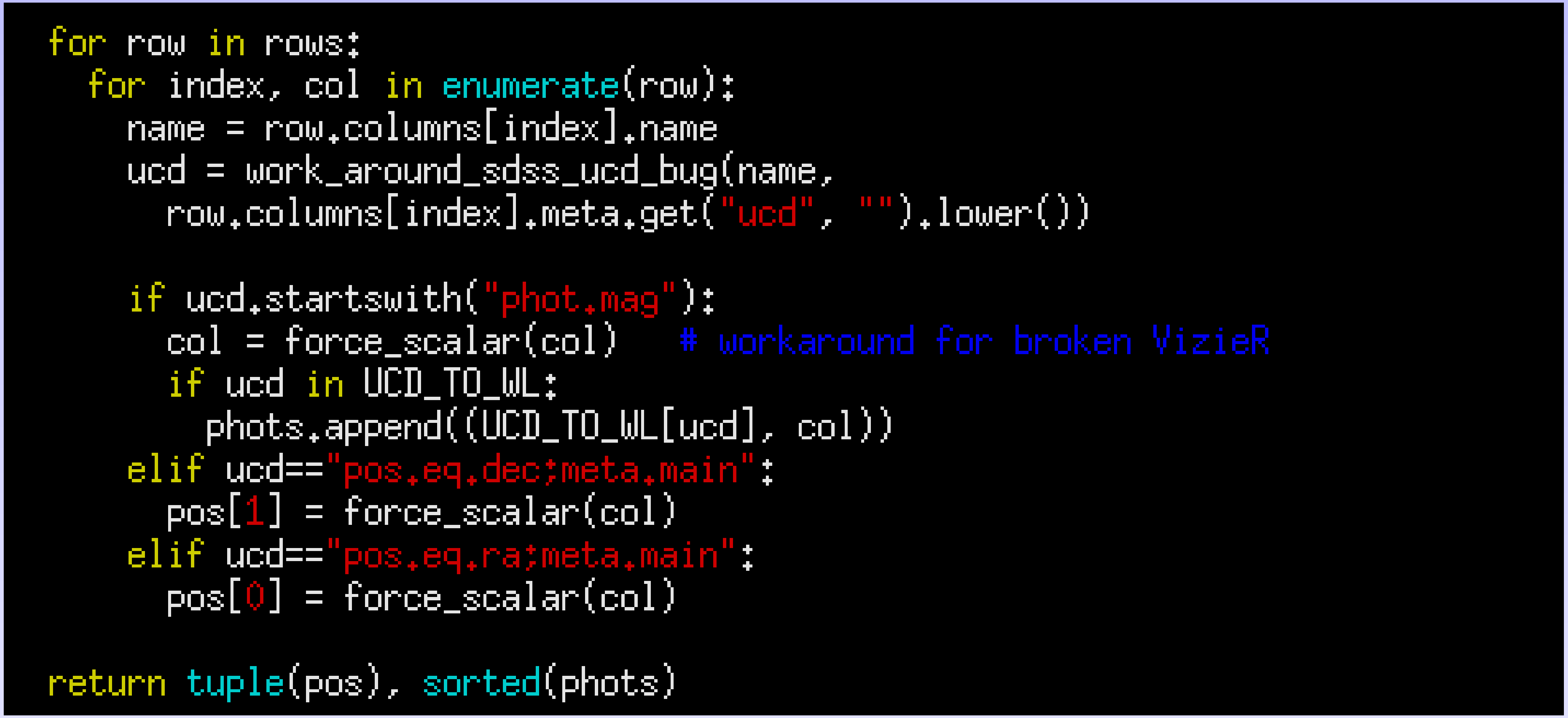
```
def get_coordinates_for_table(table):
  ra_column = vohelper.get_name_for_ucd(
    "pos.eq.ra;meta.main", table)
  dec_column = vohelper.get_name_for_ucd(
    "pos.eq.dec;meta.main", table)

  # fix broken metadata (sigh)
  if table[ra_column].unit=="Angle[deg]":
    table[ra_column].unit = "deg"
  if table[dec_column].unit=="Angle[deg]":
    table[dec_column].unit = "deg"

  return coordinates.SkyCoord(
    work_around_vizast_bug(table[ra_column]),
    work_around_vizast_bug(table[dec_column]))
```

Build SEDs

Get magnitudes through UCDs (and workaround):



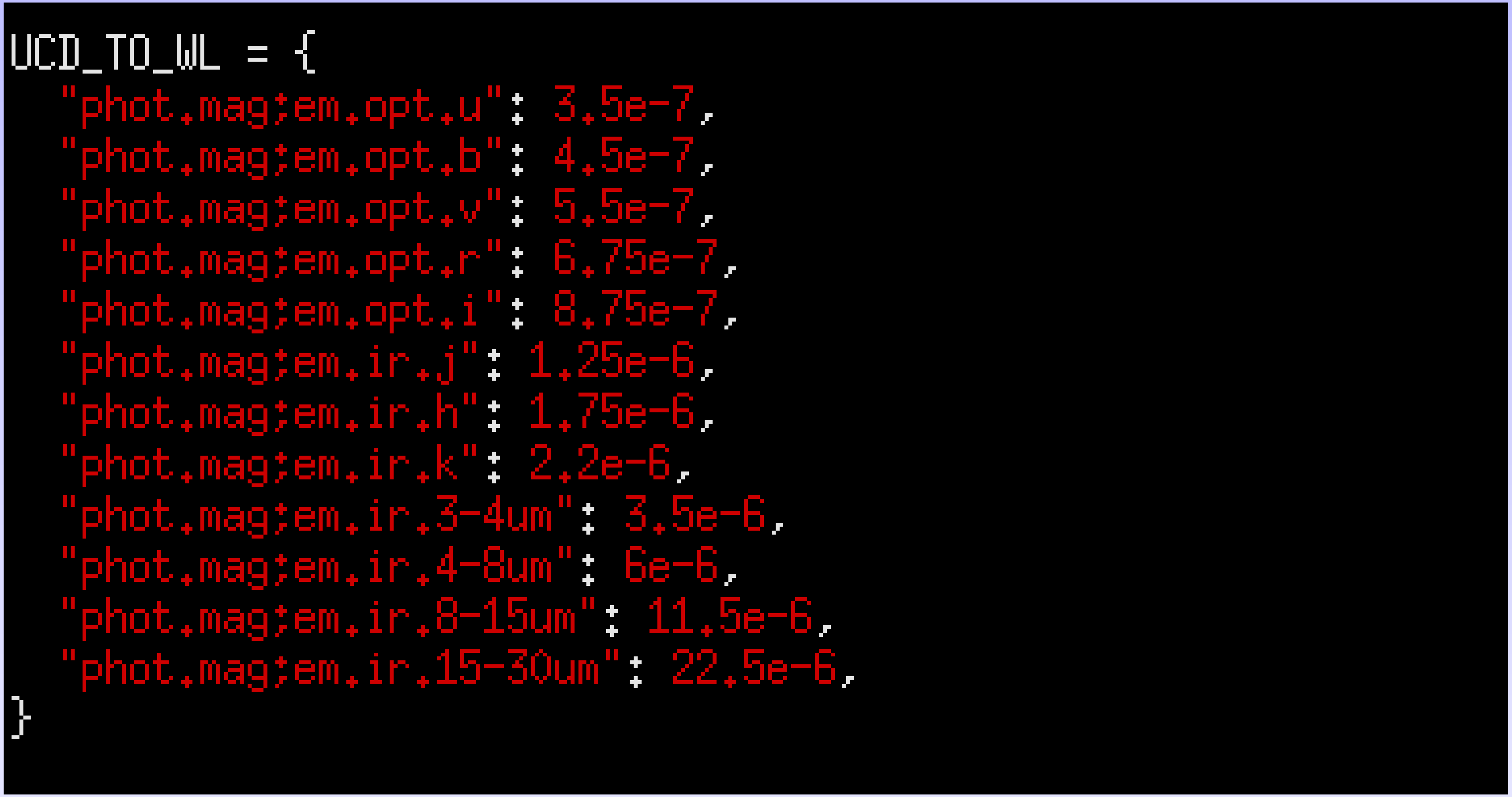
```
for row in rows:
  for index, col in enumerate(row):
    name = row.columns[index].name
    ucd = work_around_sdss_ucd_bug(name,
      row.columns[index].meta.get("ucd", "").lower())

    if ucd.startswith("phot.mag"):
      col = force_scalar(col)   # workaround for broken VizieR
      if ucd in UCD_TO_WL:
        phots.append((UCD_TO_WL[ucd], col))
    elif ucd=="pos.eq.dec;meta.main":
      pos[1] = force_scalar(col)
    elif ucd=="pos.eq.ra;meta.main":
      pos[0] = force_scalar(col)

return tuple(pos), sorted(phots)
```

Build SEDs continued

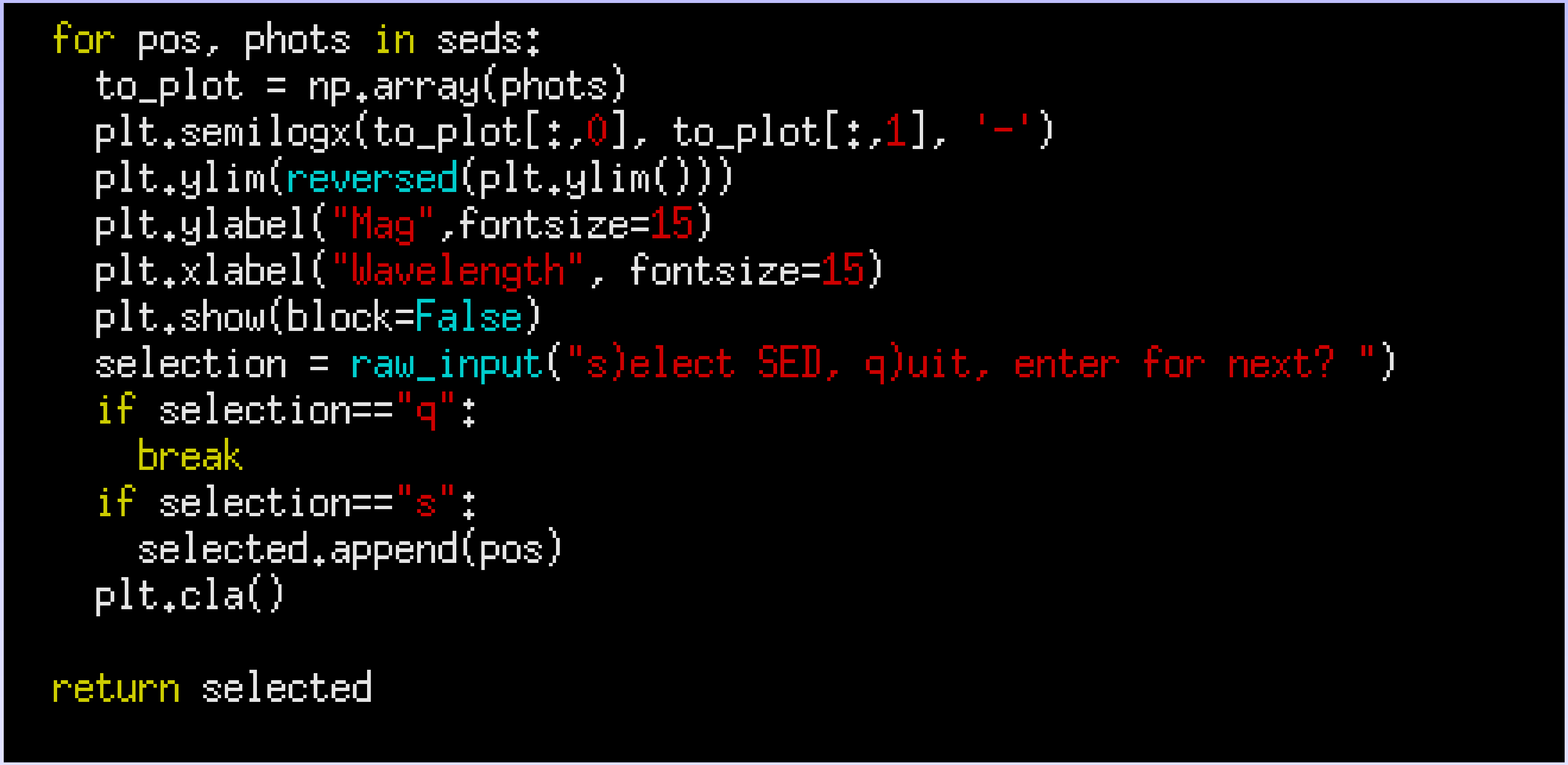
Mapping UCDs to band width:



```
UCD_TO_WL = {
  "phot.mag;em.opt.u": 3.5e-7,
  "phot.mag;em.opt.b": 4.5e-7,
  "phot.mag;em.opt.v": 5.5e-7,
  "phot.mag;em.opt.r": 6.75e-7,
  "phot.mag;em.opt.i": 8.75e-7,
  "phot.mag;em.ir.j": 1.25e-6,
  "phot.mag;em.ir.h": 1.75e-6,
  "phot.mag;em.ir.k": 2.2e-6,
  "phot.mag;em.ir.3-4um": 3.5e-6,
  "phot.mag;em.ir.4-8um": 6e-6,
  "phot.mag;em.ir.8-15um": 11.5e-6,
  "phot.mag;em.ir.15-30um": 22.5e-6,
}
```

Plot SEDs

Use matplotlib to select SEDs of interest.

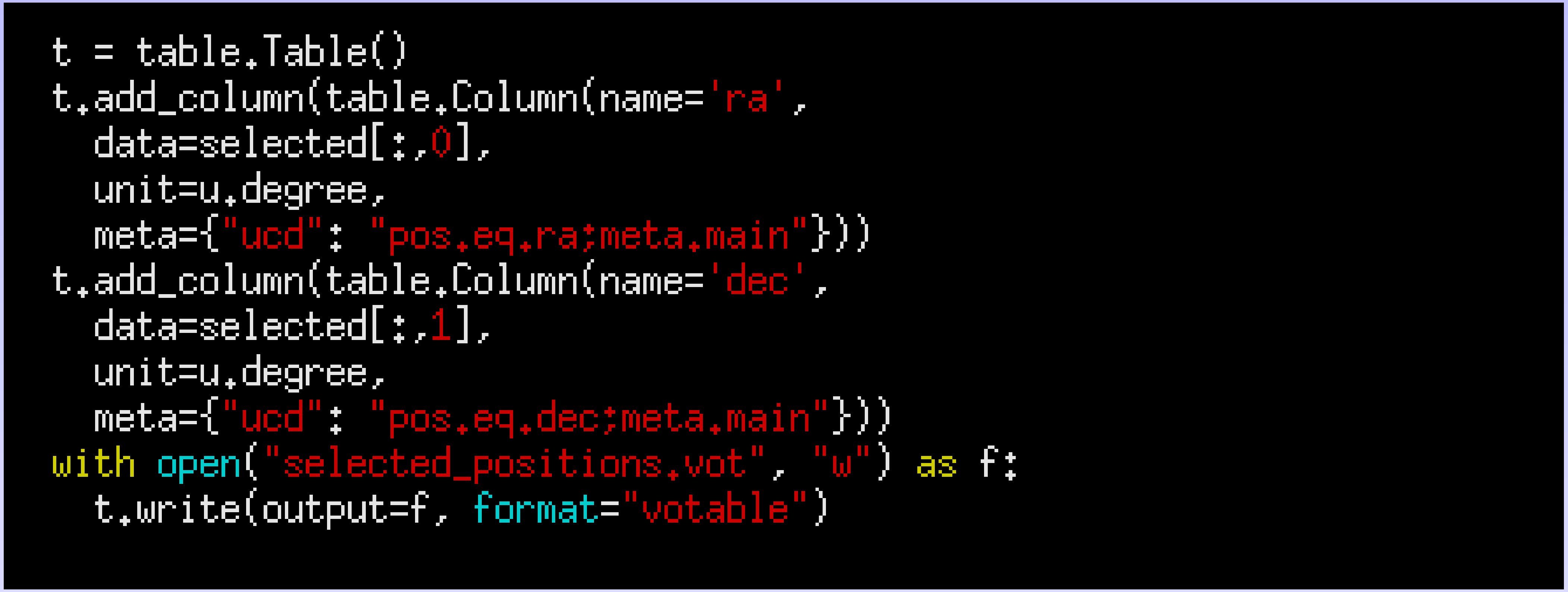


```
for pos, phots in seds:
  to_plot = np.array(phots)
  plt.semilogx(to_plot[:,0], to_plot[:,1], '-')
  plt.ylim(reversed(plt.ylim()))
  plt.ylabel("Mag",fontsize=15)
  plt.xlabel("Wavelength", fontsize=15)
  plt.show(block=False)
  selection = raw_input("s)elect SED, q)uit, enter for next? ")
  if selection=="q":
    break
  if selection=="s":
    selected.append(pos)
  plt.cla()

return selected
```

Save selected Positions

Make a votable from the selected positions.



```
t = table.Table()
t.add_column(table.Column(name='ra',
  data=selected[:,0],
  unit=u.degree,
  meta={"ucd": "pos.eq.ra;meta.main"}))
t.add_column(table.Column(name='dec',
  data=selected[:,1],
  unit=u.degree,
  meta={"ucd": "pos.eq.dec;meta.main"}))
with open("selected_positions.vot", "w") as f:
  t.write(output=f, format="votable")
```

Overview

- 200 lines of code
- .. of which almos half are workarounds for interoparability bugs

Imagine how great things will be when people will have embraced interoperable services and annotations.

Looking for Spectra

SSAP is the VO protocol to access spectra. Hence, it only let's you access one object at y time, which is kind of tedious. Let's use `obscore` instead. Luckily `obscore` is just TAP with a special table structure. It's good to find spectra, cubes, timeseries, etc.

- Search for `obscore` services
- Use TAP upload to search to collect spectra
- Send spectra to SPLAT

You can query all Obscore services (essentially) in a uniform way.

Query the registry

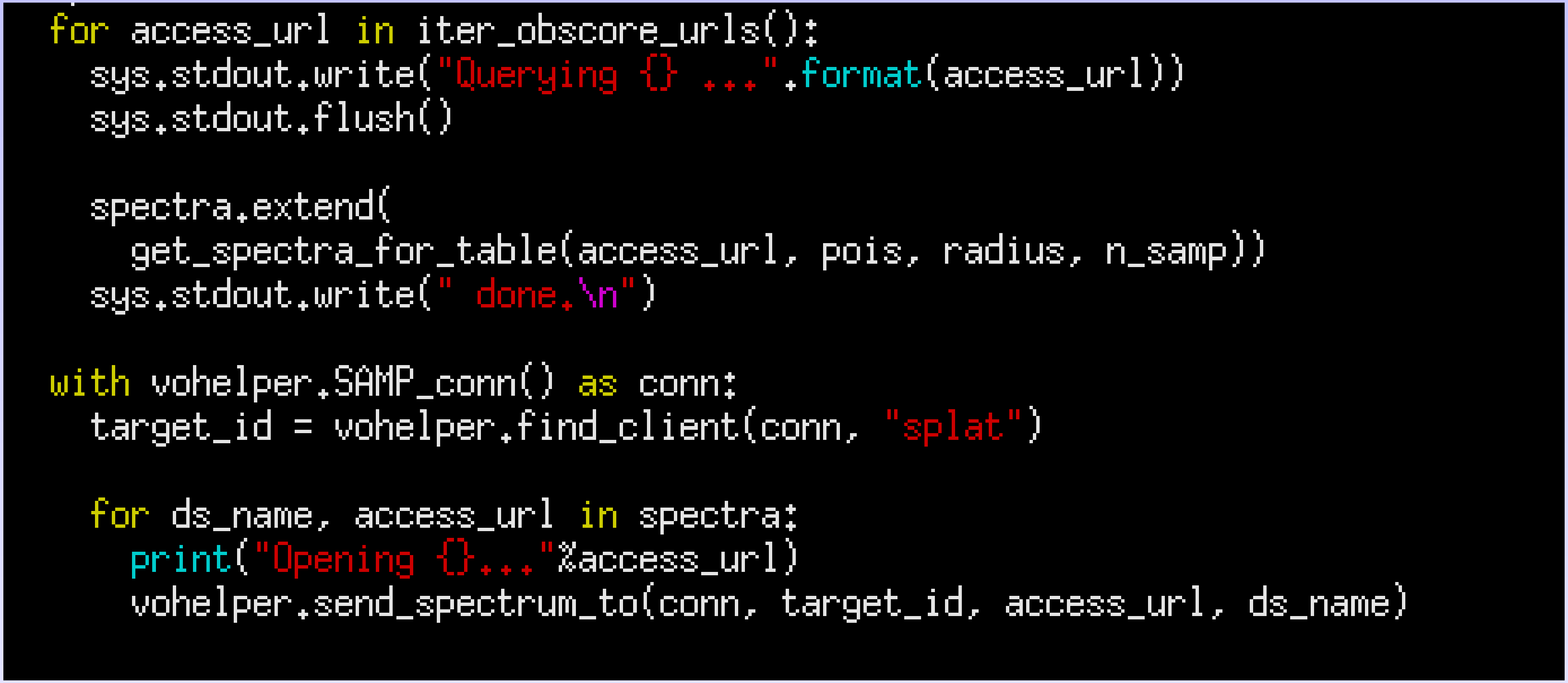
First query the Registry for all Obscore services:

```
# use raw RegTAP until pyVO registry is up to the task
result = pyvo.dal.TAPService("http://reg.g-vo.org/tap"
  ).run_sync("""
  SELECT DISTINCT access_url AS url
  FROM rr.interface
  NATURAL JOIN rr.capability
  NATURAL JOIN rr.res_detail
  WHERE standard_id='ivo://ivoa.net/std/tap'
     AND intf_type='vs:paramhttp'
     AND detail_xpath='/capability/dataModel/@ivo-id'
     AND 1=ivo_nocasematch(detail_value,
       'ivo://ivoa.net/std/obscore%')""")

for url in result["url"]:
  yield url
```

Collect Spectra

Collect spectra and send them to SPLAT to display and investigate



```
for access_url in iter_obscore_urls():
  sys.stdout.write("Querying {} ...".format(access_url))
  sys.stdout.flush()

  spectra.extend(
    get_spectra_for_table(access_url, pois, radius, n_samp))
  sys.stdout.write(" done.\n")

with vohelper.SAMP_conn() as conn:
  target_id = vohelper.find_client(conn, "splat")

  for ds_name, access_url in spectra:
    print("Opening {}..."%access_url)
    vohelper.send_spectrum_to(conn, target_id, access_url, ds_name)
```

Collect Spectra continued

The TAP upload query:

```
result = vohelper.run_sync_resilient(svc,
  """SELECT TOP {samplesize} obs_publisher_did, access_url
      FROM ivoa.obscore
      JOIN TAP_UPLOAD.pois AS up
      ON 1=INTERSECTS(
        s_region,
        CIRCLE('ICRS', up.{ra_column_name}, up.{dec_column_name}, {radius}))
      WHERE dataproduct_type='spectrum'
      """.format(**locals()),
      # add more constraints (spectral region, resolution... here)
      uploads = {"pois": ('inline', pois)})

if result is None:
  return

for row in result.table:
  yield unicode(row[0]), unicode(row[1])
```

Where to get PyVO?

- `pip install pyvo`
- Debian package available at: `http://www.g-vo.org/`

Thank you for your attention.