cf-python
Cheat Sheet

https:ncas-cms.github.io/cf-python

Install

Install cf-python, including any required dependencies, and the cf-plot visualisation package using pip:

```
$ pip install cf-python cf-plot
```

Install cf-python with all of its required and optional dependencies, and the cf-plot visualisation package using conda:

```
$ conda install -c conda-forge cf-python cf-plot \
> udunits2 esmpy>=8.0.0
```

Import

Import cf-python and cf-plot:

```
import cf
import cfplot as cfp
```

Read

Read the field constructs from a single file:

```
fl_1 = cf.read("file.nc")
fl_2 = cf.read("umfile.pp")
```

Read the field constructs from two or more files:

```
fl = cf.read(['temperature.nc', 'humidity.nc'])
```

*Read the field constructs using the * wildcard character (which reads all files in the directory that match the specified pattern):*

```
fl = cf.read('~/file*.nc')
```

*Return field constructs by selecting the elements of the resulting * wildcard file list:*

```
fl = cf.read('~/file*.nc')[0:3]
f = cf.read('~/file*nc')[-1]
```

Return field constructs whose identities match the given values:

```
fl = cf.read('~/file*.nc', select='units=hPa')
fl = cf.read('~/file*.nc', select='temperature')
fl = cf.read('~/files/file*.nc', select='ncvar%q')
```

Selecting fields:

Select a field directly by index while reading the file:

```
temp = cf.read("file.nc")[0]
```

Select a field from a field list by identity:

```
temp = fl.select_field('air_temperature')
```

Select more than one field from a field list:

```
data = fl.select('precipitation', 'ncvar%stn')
```

Write

Write one or more field constructs to a netCDF file:

```
cf.write(tas, 'temperature.nc')
```

Convert PP and UM fields files to netCDF files:

```
cf.write(pp, 'umfile1.nc')
```

Collapse

Basic collapses:

Write one or more field constructs to a netCDF file:

```
min = f.collapse('minimum')
v = f.collapse('variance')
i = f.collapse('integral', measure=True)
```

Weighted collapses:

Weighted time average:

```
avg = f.collapse('T: mean', weights=True)
```

Mean over the time and latitude axes, with weights only applied to the latitude axis:

```
m = f.collapse('T: Y: mean', weights='Y')
```

Weighted area means:

```
mean = f.collapse('area: mean', weights=True)
```

Specifying Axes:

Temporal maxima

```
max = f.collapse('maximum', axes='T')
# or equivalently:
max = f.collapse('T: maximum')
```

Horizontal maximum:

```
max = f.collapse('maximum', axes=['X', 'Y'])
# or equivalently:
max = f.collapse('X: Y: maximum')
```

Grouped collapses:

Annual maxima starting 1st December:

```
y_max = f.collapse('T: maximum', cf.Y(month=12))
```

December, January, February maxima:

```
djf_max = f.collapse('T: maximum', group=cf.djf())
```

Maxima for each 3-month season (DJF, MAM, JJA, SON):

```
seas_max = f.collapse('T: maximum', group=cf.seasons())
```

Maxima for each 4-month season (JJAS, ONDJ, FMAM):

```
seas_max = f.collapse('T: maximum', group=cf.seasons(3, 6))
```

Multiannual average of the seasonal means:

```
avg = f.collapse('T: mean within years T: mean over years',
                 within_years=cf.seasons(), weights=True)
```

Subspace

Subspacing by index:

Write one or more field constructs to a netCDF file:

```
new = temp[:, :, 0]
new = temp[0, [2, 3, 9], [4, 8]]
new = temp[..., [True, False, True, True, False]]
```

Subspacing by metadata:

Subspace to a new field construct whose 'longitude' coordinate spans only 45 degrees east, with the other domain axes remaining unchanged:

```
new = temp.subspace(longitude=45) # or equivalently:
new = temp.subspace(X=45)
```

Subspace to a new field construct whose longitude coordinate is 45 degrees east and latitude coordinate is -60 degrees north, with the other domain axes remaining unchanged:

```
new = temp.subspace(X=45, Y=-60)
```

Subspace to a new field construct whose domain spans 34 to 72 degrees north and -25 to 45 degrees east:

```
new = temp.subspace(Y=cf.wi(34, 72), X=cf.wi(-25, 45))
```

Subspaces based on the time dimension:

```
new = temp.subspace(T=cf.dt('1996-01-20'))
jan_2023 = temp.subspace(T=cf.year(2023) & cf.month(1))
avg = global_avg.subspace(T=cf.year(cf.wi(1961, 1990)))
```

Accessing constructs

Select constructs by identity:

```
x = f.dimension_coordinate('latitude')
x = f.auxiliary_coordinate('altitude')
```

Select constructs by property:

```
d = f.dimension_coordinate(units='degrees_east')
d = f.dimension_coordinate(long_name='height', units='m')
```

Select constructs by cell method:

```
d = f.cell_method('mean')
d = f.cell_method('mean', 'maximum')
```

Select constructs by axis:

```
d = f.constructs.filter_by_axis('domainaxis1')
```

Select constructs by type:

```
d = f.dimension_coordinates()
```

```
d = f.constructs.filter_by_type('dimension_coordinate',
'field_ancillary')
```

Select constructs by netCDF variable name:

```
d = f.constructs.filter_by_ncvar('time')
d = f.constructs.filter_by_ncvar('time', 'lat')
```

Regridding

Regridding in spherical polar coordinates:

Regrid observational data to that of the model data using bilinear interpolation:

```
obs_regrid = obs.regrids(model, method='linear')
```

Regrid observational data to that of the model data conservatively:

```
obs_regrid = obs.regrids(model, method='conservative')
```

Regrid observational data onto two-dimensional dimension coordinates latitude and longitude using bilinear interpolation:

```
import numpy
d = cf.Domain.create_regular((0, 360, 5.0), (-90, 90, 2.5))
obs_regrid = obs.regrids(d, method='linear')
```

Regridding in cartesian coordinates:

Regrid the time axis 'T' of observational data with the linear method onto the grid specified in the dimension coordinate time:

```
time = cf.DimensionCoordinate.create_regular(
     (0.5, 60.5, 1),
     units=cf.Units("days since 1860-01-01",
     calendar="360_day"),
     standard_name="time",
)
obs_new = obs.regridc([time], axes="T", method="linear")
```

MORE RESOURCES:

API reference | Tutorial | Recipes | GitHub