Xarray Part 1

https://xarray.pydata.org/en/stable/index.html

xarray is a python package which allows us to handle multi-dimensional datasets in a simple way. It provides a huge set of functions for advanced analytics and visualization. It is part of higher level package ecosystems like Pangeo.

xarray's underlying data model is borrowed from the data format NetCDF. This data format in combination with the Climate and Forecast conventions is the standard for the climate science community. A large part of DKRZ's data is available in netCDF. Therefore, `xarray` allows fast and intuitive data analysis on this kind of data.

xarray data structure deals with scientific data by using labels, attributes, dimensions and coordinates, and extend the capabilities of **NumPy** and **pandas**.

Content:

- DataArrays
- Dimensions
- Coordinates
- Variable attributes
- Datasets
- Read and open files
- Indexing and selecting data

Requirements:

- numpy

Overview: Xarray's data model

A **data model** describes how the package organizes elements of data and standardizes how they relate to one another. On code level, a graph of a data

model shows the interconnections of classes, types and methods. **Xarray's** data model consists of the classes *Dataset*, *DataArray*, *Dimension*, *Coordinate* and *attributes*.

Dataset (file):

```
Dict-like collection of DataArray objects with aligned dimensions. Similar use of variables
```

DataArray (= variable in the file):

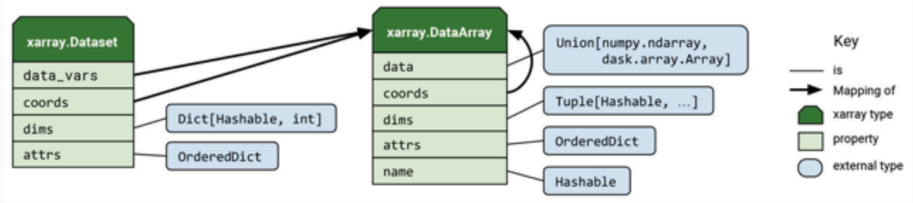
```
N-dimensional array with dimensions. The objects add dimension names, coordinates, and attib
```

Dimensions:

```
Named dimension axes, if missing the dimension names are dim_0, dim_1, ...
```

Coordinates:

```
An array which labels a dimension. Two types are defined a) dimension coordinates - 1-dimens
```



From https://xarray-contrib.github.io/xarray-tutorial/online-tutorial-series/01_xarray_fundamentals.html

Importing modules

In this notebook, the Python libraries Numpy, Pandas, and cfgrib are needed for the examples.

```
import xarray as xr

import numpy as np
import pandas as pd
import cfgrib
```

If you work with jupyter lite, *before* importing the packages do:

```
import micropip
await micropip.install(['xarray','cfgrib'])
```

```
import xarray as xr
import numpy as np
import pandas as pd
import cfgrib
```

DataArray

As a start, we compare the `numpy` array with an `xarray`'s **DataArray** type. You can directly convert a `numpy` array into an `xarray` **DataArray** type by using it as input for `xarray`'s function `DataArray`. We use the data from the file `pr.dat` by loading it with `numpy`.

```
pr_data = np.loadtxt('pr.dat', usecols=(1,2,3), skiprows=1)
pr_data_xr = xr.DataArray(pr_data)
pr_data_xr
```

In Jupyterlite, you can do:

```
from js import fetch
res = await fetch('https://swift.dkrz.de/v1/dkrz_0b2a0dcc-1430-4a8a-9f25-a6cb8924d92b/pythor
text = await res.text()

from io import StringIO
f = StringIO(text)
```

where f can be opened by numpy.

```
pr_data = np.loadtxt('pr.dat', usecols=(1,2,3), skiprows=1)
pr_data_xr = xr.DataArray(pr_data)
pr_data_xr
```

```
<xarray.DataArray (dim_0: 32, dim_1: 3)>
array([[  36.75,  -97.25,   48.  ],
       [  36.41,  -97.69,   46.3 ],
       [  36.63,  -96.81,   49.8 ],
       [  34.11,  -94.29,   45.  ],
       [  40.08,  -97.31,   38.2 ],
       [  35.85,  -97.48,   46.6 ],
       [  38.31,  -97.29,   34.3 ],
       [  35.68,  -95.86,   50.2 ],
       [  33.02, -100.98,   39.8 ],
       [  36.69,  -97.48,   47.1 ],
       [  39.58,  -94.17,   40.4 ],
       [  36.79,  -97.75,   46.7 ],
       [  37.3 ,  -95.6 ,   39.9 ],
       [  36.03,  -96.5 ,   48.8 ],
       [  34.98,  -97.52,   46.5 ],
       [  36.36,  -97.15,   50.4 ],
       [  40.09, -100.65,   26.4 ],
       [  36.56, -100.61,   22.4 ],
       [  36.6 ,  -97.49,   49.  ],
       [  36.43,  -98.28,   43.2 ],
       [  36.88,  -98.29,   41.4 ],
```

```
       [  37.33,  -99.31,   33.6 ],
       [  38.2 ,  -99.32,   33.9 ],
       [  38.12,  -97.51,   36.3 ],
       [  37.84,  -97.02,   40.2 ],
       [  38.2 ,  -95.59,   31.7 ],
       [  37.38,  -96.18,   42.8 ],
       [  34.88,  -98.2 ,   45.9 ],
       [  35.36,  -98.98,   45.3 ],
       [  35.56,  -98.02,   47.9 ],
       [  35.26,  -97.48,   46.8 ],
       [  36.07,  -99.22,   38.6 ]])
Dimensions without coordinates: dim_0, dim_1
```

`pr_data_xr` has got more structure and descriptive information than `pr_data`. In contrast to the `numpy` data array, the `Xarray's` DataArray can separate the variable of interest, `pr`, as a *data variable* from *coordinate* variables. In summary, it contains:

- **dimensions** with names (`pr_data_xr.dims`)
- **coordinates** pointing to variables (`pr_data_xr.coords`)
- and **attributes** (`pr_data_xr.attrs`)

Not only `xarray` but other software tools require and use the **labeld geospatial** information from coordinates, for example for

- **plotting**: mapping of data on a real world grid point
- **analysis**: implemented routines for e.g. area *weighted* means can be run

This information is not correctly parsed from the input numpy array per default when executing `xr.DataArray()`. But we know them so we need to configure the call `xr.DataArray()` via the function parameters (arguments + keyword arguments):

```
xr.DataArray(data,
             coords=,
             dims=,
             name=,
             attrs
            )
```

Parsing numpy data with labels to xarray

Let's define a clear structure for the `xarray.DataArray()` for the numpy data first:

1. The actual **data** for the data variable is in the first column of the `numpy` array.

2. The **coords** are the second and third column of the `numpy` array. They have the same dimension as the data array.
3. We have one dimension (**dims**) which refers to the *station*. It is an index which runs from 0 to the length of the a column minus 1.
4. The **name** of the data variable is *Precipitation*.
5. In the **attrs**, we can store variable attributes like *units*.

Let's bring that into context with `xr.DataArray()`:

```
pr_data_xr = xr.DataArray(pr_data[:,0],
                          coords={"lon":("Station",pr_data[:,1]),
                                  "lat":("Station",pr_data[:,2])},
                          dims=["Station"],
                          name="Precipitation",
                          attrs={"units":"mm",
                                 "coords":"lon lat"})

pr_data_xr = xr.DataArray(pr_data[:,2],
                          coords={"lon":("Station",pr_data[:,1]),
                                  "lat":("Station",pr_data[:,0])},
                          dims=["Station"],
                          name="Precipitation",
                          attrs={"units":"mm",
                                 "coords":"lon lat"})
```

```
print("Variable Name: ",pr_data_xr.name)
print("Dimensions: ",pr_data_xr.dims)
print("Coordinates: ",pr_data_xr.coords)
print("Sizes: ",pr_data_xr.sizes)
print("Attribute: ",pr_data_xr.attrs)
```

```
Variable Name:  Precipitation
Dimensions:  ('Station',)
Coordinates:  Coordinates:
    lon      (Station) float64 -97.25 -97.69 -96.81 ... -98.02 -97.48 -99.22
    lat      (Station) float64 36.75 36.41 36.63 34.11 ... 35.56 35.26 36.07
Sizes:  Frozen({'Station': 32})
Attribute:  {'units': 'mm', 'coords': 'lon lat'}
```

```
pr_data_xr
```

```
<xarray.DataArray 'Precipitation' (Station: 32)>
array([48. , 46.3, 49.8, 45. , 38.2, 46.6, 34.3, 50.2, 39.8, 47.1, 40.4,
       46.7, 39.9, 48.8, 46.5, 50.4, 26.4, 22.4, 49. , 43.2, 41.4, 33.6,
       33.9, 36.3, 40.2, 31.7, 42.8, 45.9, 45.3, 47.9, 46.8, 38.6])
Coordinates:
    lon      (Station) float64 -97.25 -97.69 -96.81 ... -98.02 -97.48 -99.22
    lat      (Station) float64 36.75 36.41 36.63 34.11 ... 35.56 35.26 36.07
Dimensions without coordinates: Station
```

```
Attributes:
    units:    mm
    coords:   lon lat
```

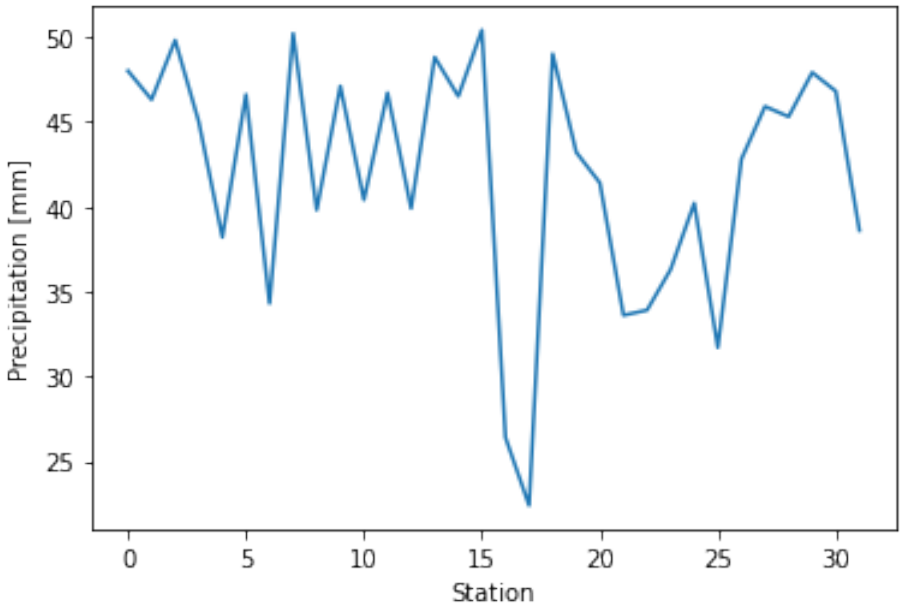
Dimensions

Dimensions are **indices** covering an interval of the length of the dimension.

In our example, we only have one dimension where each index refers to one **station**. However, if create a quick plot of the data with the DataArray variable's `.plot()` function, we only get a one dimensional view:

```
pr_data_xr.plot()
```

```
[<matplotlib.lines.Line2D at 0x7fff85097760>]
```




```
pr_data_xr.variable
```

```
<xarray.Variable (Station: 32)>
array([48. , 46.3, 49.8, 45. , 38.2, 46.6, 34.3, 50.2, 39.8, 47.1, 40.4,
       46.7, 39.9, 48.8, 46.5, 50.4, 26.4, 22.4, 49. , 43.2, 41.4, 33.6,
       33.9, 36.3, 40.2, 31.7, 42.8, 45.9, 45.3, 47.9, 46.8, 38.6])
Attributes:
    units:    mm
    coords:   lon lat
```

Create a two dimensional georeferenced plot Our goal for this session now is to reorganize the data so that `.plot()` returns a meshed grid plot. For that, we create a less condensed **two-dimensional** DataArray (with a lot of `NaN` values).

1. Create a two dimensional numpy with the size `len(pr_data)` x `len(pr_data)`
2. Assign `NaN` values to the entire array
3. On the diagonal of the quadratic array, insert the values of `pr_data`
4. Show the new data frame

You will need:

- `np.empty()`
- `np.Nan`
- `for` loop

```
pr_data_2d = np.zeros((len(pr_data), len(pr_data)))
pr_data_2d
```

```
array([[0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       ...,
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.]])
```

```
pr_data_2d[:] = np.nan
# or 1. and 2. step at once
#pr_data_2d = np.empty([len(pr_data),len(pr_data)]) * np.nan
#pr_data_2d = np.full([len(pr_data),len(pr_data)], np.nan)
pr_data_2d
```

```
array([[nan, nan, nan, ..., nan, nan, nan],
       [nan, nan, nan, ..., nan, nan, nan],
       [nan, nan, nan, ..., nan, nan, nan],
       ...,
       [nan, nan, nan, ..., nan, nan, nan],
       [nan, nan, nan, ..., nan, nan, nan],
       [nan, nan, nan, ..., nan, nan, nan]])
```

```
for x in range(len(pr_data)):
    pr_data_2d[x,x] = pr_data[x,2]

pr_data_2d
```

```
array([[48. ,  nan,  nan, ...,  nan,  nan,  nan],
       [ nan, 46.3,  nan, ...,  nan,  nan,  nan],
       [ nan,  nan, 49.8, ...,  nan,  nan,  nan],
       ...,
       [ nan,  nan,  nan, ..., 47.9,  nan,  nan],
       [ nan,  nan,  nan, ...,  nan, 46.8,  nan],
       [ nan,  nan,  nan, ...,  nan,  nan, 38.6]])
```

Let's pass this DataArray to **Xarray**.

1. Reset the variable `pr_data_xr` with a `xr.DataArray()` but use `pr_data_2d` as input.

    1.1. Set a correct configuration for the parameters of the function.

2. Plot again

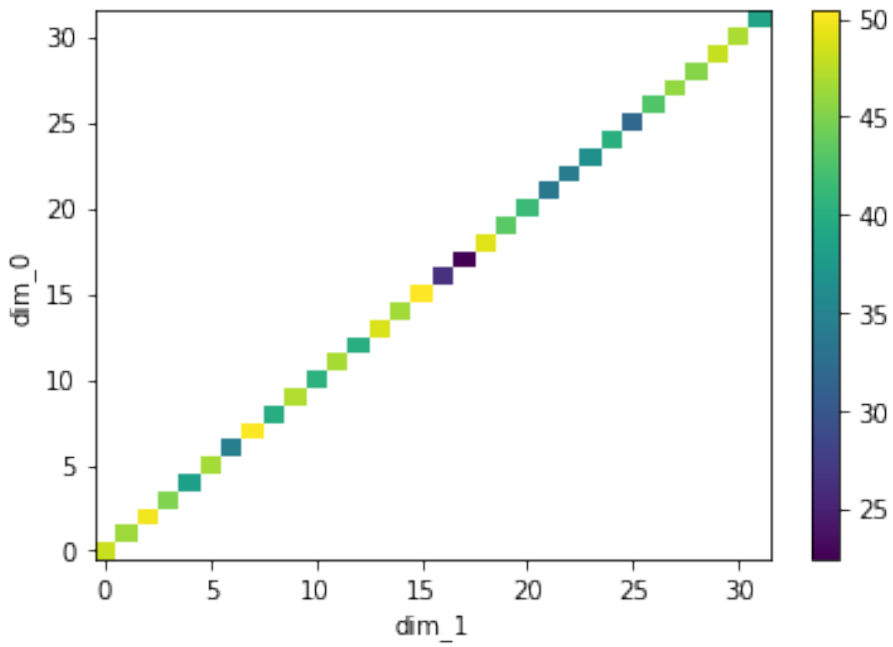
```
pr_data_xr = xr.DataArray(pr_data_2d)
pr_data_xr
```

```
<xarray.DataArray (dim_0: 32, dim_1: 32)>
array([[48. ,  nan,  nan, ...,  nan,  nan,  nan],
       [ nan, 46.3,  nan, ...,  nan,  nan,  nan],
       [ nan,  nan, 49.8, ...,  nan,  nan,  nan],
       ...,
       [ nan,  nan,  nan, ..., 47.9,  nan,  nan],
       [ nan,  nan,  nan, ...,  nan, 46.8,  nan],
       [ nan,  nan,  nan, ...,  nan,  nan, 38.6]])
Dimensions without coordinates: dim_0, dim_1
```

We plot the two dimension xr:

```
pr_data_xr.plot()
```

```
<matplotlib.collections.QuadMesh at 0x7fff7cf60730>
```



Coordinates

The plot only uses the indices of the dimensions for the x and y axes of the plot. This is because the **coordinates** `lat` and `lon` are not interpreted as **index coordinates**. `Xarray` will interpete coordinates as **index coordinates** only if the name of the coordinate is the same as the name of the dimension.

1. Reset the variable `pr_data_xr` with a `xr.DataArray()` but rename `coords` or `dims` so that they are equal.
2. Plot again

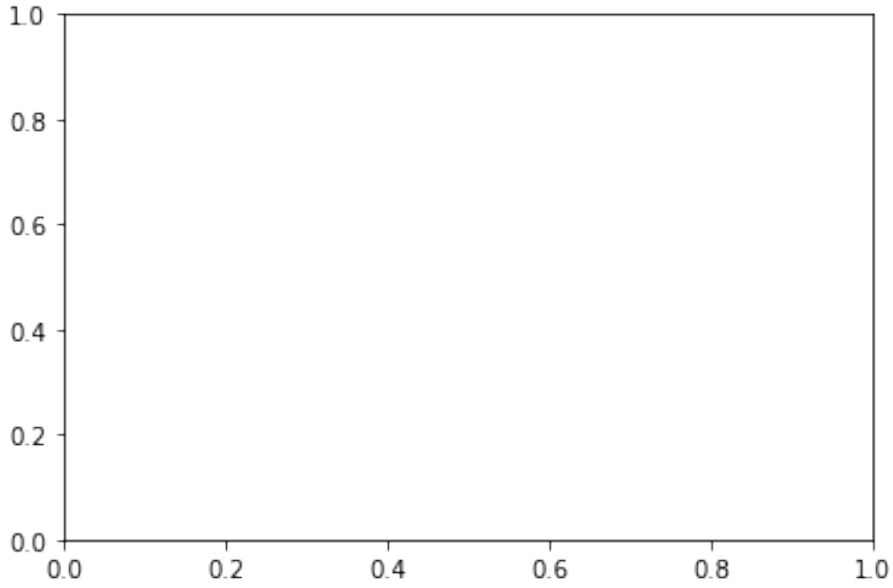
```
pr_data_xr = xr.DataArray(pr_data_2d,
                          coords={"lon":pr_data[:,1],
                                  "lat":pr_data[:,2]},
                          dims=["lon", "lat"])

pr_data_xr.plot()
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Input In [29], in <cell line: 1>()
----> 1 pr_data_xr.plot()
```

```
File /sw/spack-levante/mambaforge-4.11.0-0-Linux-x86_64-sobz6z/lib/python3.9/site-packages/x
    865 def __call__(self, **kwargs):
--> 866     return plot(self._da, **kwargs)

File /sw/spack-levante/mambaforge-4.11.0-0-Linux-x86_64-sobz6z/lib/python3.9/site-packages/x
    328     plotfunc = hist
    330 kwargs["ax"] = ax
--> 332 return plotfunc(darray, **kwargs)

File /sw/spack-levante/mambaforge-4.11.0-0-Linux-x86_64-sobz6z/lib/python3.9/site-packages/x
   1206     raise ValueError("plt.imshow's `aspect` kwarg is not available in xarray")
   1208 ax = get_axis(figsize, size, aspect, ax, **subplot_kws)
-> 1210 primitive = plotfunc(
   1211     xplt,
   1212     yplt,
   1213     zval,
   1214     ax=ax,
   1215     cmap=cmap_params["cmap"],
   1216     vmin=cmap_params["vmin"],
   1217     vmax=cmap_params["vmax"],
   1218     norm=cmap_params["norm"],
   1219     **kwargs,
   1220 )
   1222 # Label the plot with metadata
   1223 if add_labels:

File /sw/spack-levante/mambaforge-4.11.0-0-Linux-x86_64-sobz6z/lib/python3.9/site-packages/x
   1449 if (
   1450     infer_intervals
   1451     and not np.issubdtype(x.dtype, str)
   (...)
   1455     )
   1456 ):
   1457     if len(x.shape) == 1:
-> 1458         x = _infer_interval_breaks(x, check_monotonic=True, scale=xscale)
   1459     else:
   1460         # we have to infer the intervals on both axes
   1461         x = _infer_interval_breaks(x, axis=1, scale=xscale)

File /sw/spack-levante/mambaforge-4.11.0-0-Linux-x86_64-sobz6z/lib/python3.9/site-packages/x
    795 coord = np.asarray(coord)
    797 if check_monotonic and not _is_monotonic(coord, axis=axis):
--> 798     raise ValueError(
    799         "The input coordinate is not sorted in increasing "
    800         "order along axis %d. This can lead to unexpected "
    801         "results. Consider calling the `sortby` method on "
```

```
    802         "the input DataArray. To plot data with categorical "
    803         "axes, consider using the `heatmap` function from "
    804         "the `seaborn` statistical plotting library." % axis
    805     )
    807 # If logscale, compute the intervals in the logarithmic space
    808 if scale == "log":
```

```
ValueError: The input coordinate is not sorted in increasing order along axis 0. This can le
```



You will receive a

```
ValueError: The input coordinate is not sorted in increasing order along axis 0. Consider ca
```

So let's use `sortby`:

```
pr_data_xr.sortby(["lon","lat"])
```

```
pr_data_xr.sortby(["lon","lat"])
```

```
<xarray.DataArray (lon: 32, lat: 32)>
array([[ nan,  nan,  nan, ...,  nan,  nan,  nan],
       [ nan, 26.4,  nan, ...,  nan,  nan,  nan],
       [22.4,  nan,  nan, ...,  nan,  nan,  nan],
       ...,
       [ nan,  nan, 31.7, ...,  nan,  nan,  nan],
       [ nan,  nan,  nan, ...,  nan,  nan,  nan],
       [ nan,  nan,  nan, ...,  nan,  nan,  nan]])
Coordinates:
  * lon      (lon) float64 -101.0 -100.7 -100.6 -99.32 ... -95.59 -94.29 -94.17
```

```
  * lat      (lat) float64 22.4 26.4 31.7 33.6 33.9 ... 48.8 49.0 49.8 50.2 50.4
```

Wrong dimension size? There are sevaral ways to repair this. One is to use `xarray`'s transpose function:

```
pr_data_xr.transpose("lat","lon")
```

```
pr_data_xr.transpose("lat","lon")
```

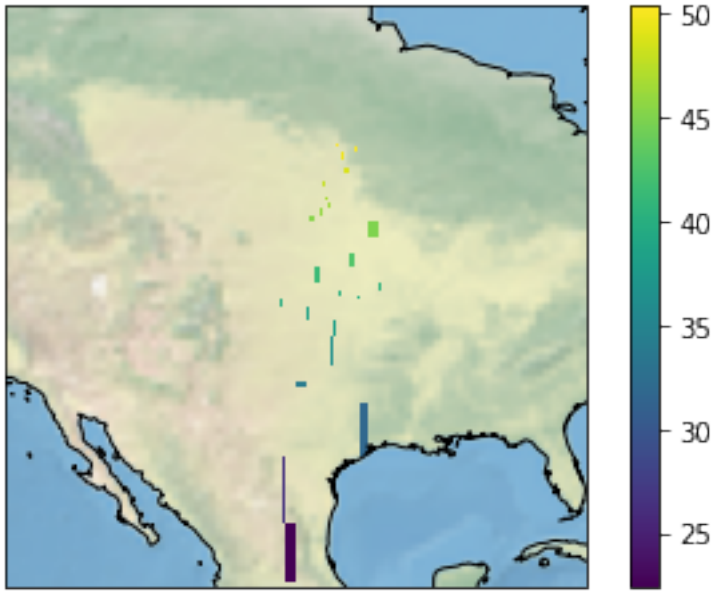
```
<xarray.DataArray (lat: 32, lon: 32)>
array([[48. ,  nan,  nan, ...,  nan,  nan,  nan],
       [ nan, 46.3,  nan, ...,  nan,  nan,  nan],
       [ nan,  nan, 49.8, ...,  nan,  nan,  nan],
       ...,
       [ nan,  nan,  nan, ..., 47.9,  nan,  nan],
       [ nan,  nan,  nan, ...,  nan, 46.8,  nan],
       [ nan,  nan,  nan, ...,  nan,  nan, 38.6]])
Coordinates:
  * lon      (lon) float64 -97.25 -97.69 -96.81 -94.29 ... -98.02 -97.48 -99.22
  * lat      (lat) float64 48.0 46.3 49.8 45.0 38.2 ... 45.9 45.3 47.9 46.8 38.6
```

We created a plot which gives us an idea of for which places the data is valid with only few commands based on `xarray`.

- The *boundaries* of the grid points are artificial. They are not specified but only rendered by the plot function.
- In the next sessions we will learn a more sophisticated plotting including e.g. *coastlines*.

```
import cartopy.crs as ccrs
import matplotlib.pyplot as plt

proj=ccrs.PlateCarree()
ax = plt.axes(projection=proj)
ax.set_extent([-120, -80, 20, 60], proj)
ax.stock_img()
ax.coastlines()

pr_data_xr.sortby(["lon","lat"]).transpose("lat","lon").plot()
```

```
<cartopy.mpl.geocollection.GeoQuadMesh at 0x7fff75809cd0>
```



Variable attributes

You can easily set an attribute, for instance the attribute *name* :

```
pr_data_xr.name = 'precip'
```

Variables in Earth Science commonly have attributes like **standard_name**, **long_name** or **units** which can be added via the *attrs* attribute to the DataArray.

Add the units attribute to the DataArray *da* :

```
pr_data_xr.attrs['units'] = 'mm'
```

```
# Change the name of the variable
pr_data_xr.name = 'pw'
```

```
pr_data_xr.attrs
```

```
{'units': 'mm', 'long_name': 'precipitable water'}
```

```
pr_data_xr
```

```
<xarray.DataArray 'pw' (lon: 32, lat: 32)>
array([[48. ,  nan,  nan, ...,  nan,  nan,  nan],
       [ nan, 46.3,  nan, ...,  nan,  nan,  nan],
       [ nan,  nan, 49.8, ...,  nan,  nan,  nan],
```

```
       ...,
       [ nan,  nan,  nan, ..., 47.9,  nan,  nan],
       [ nan,  nan,  nan, ...,  nan, 46.8,  nan],
       [ nan,  nan,  nan, ...,  nan,  nan, 38.6]])
Coordinates:
  * lon      (lon) float64 -97.25 -97.69 -96.81 -94.29 ... -98.02 -97.48 -99.22
  * lat      (lat) float64 48.0 46.3 49.8 45.0 38.2 ... 45.9 45.3 47.9 46.8 38.6
Attributes:
    units:      mm
    long_name:  precipitable water
```

```
pr_data_xr.attrs['units'] = 'mm'
pr_data_xr.attrs
```

```
{'units': 'mm'}
```

1. Add the variable attribute units as shown above
2. Add the variable long_name (as you like ;))
3. Change the long_name
4. Print all attributes

```
pr_data_xr.attrs['units'] = 'mm'
```

```
pr_data_xr.attrs['long_name'] = 'as you like'
```

```
pr_data_xr.attrs['long_name'] = 'precipitable water'
```

```
pr_data_xr.attrs
```

```
{'units': 'mm', 'long_name': 'precipitable water'}
```

Datasets

Xarray's function `open_dataset` can be used to open and read the content of a file. It supports various formats, such as **netcdf, grib, zarr**, etc. (default: netcdf4). The file content will be stored in the Xarray Dataset structure.

Example:

In the data directory of the course material, we use the file *tsurf.nc* to demonstrate Xarray's file handling.

```
ds = xr.open_dataset('../data/tsurf.nc')
```

```
ds.info()
```

Result:

```
xarray.Dataset {
dimensions:
```

```
        lat = 96 ;
        lon = 192 ;
        time = 40 ;

variables:
        datetime64[ns] time(time) ;
                time:standard_name = time ;
                time:axis = T ;
        float64 lon(lon) ;
                lon:standard_name = longitude ;
                lon:long_name = longitude ;
                lon:units = degrees_east ;
                lon:axis = X ;
        float64 lat(lat) ;
                lat:standard_name = latitude ;
                lat:long_name = latitude ;
                lat:units = degrees_north ;
                lat:axis = Y ;
        float32 tsurf(time, lat, lon) ;
                tsurf:long_name = surface temperature ;
                tsurf:units = K ;
                tsurf:code = 169 ;
                tsurf:table = 128 ;

// global attributes:
        :CDI = Climate Data Interface version 1.9.6 (http://mpimet.mpg.de/cdi) ;
        :Conventions = CF-1.6 ;
        :history = Thu Oct 10 16:08:50 2019: cdo selname,tsurf rectilinear_grid_2D.nc tsurf.
        :CDO = Climate Data Operators version 1.9.6 (http://mpimet.mpg.de/cdo) ;
}
```

Show variable names and coordinates

It is always good to have a closer look at the data, and this can be done very easily using the attributes explained above.

Show the coordinates stored in file:

```
coords = ds.coords
```

Result:

```
 Coordinates:
  * time     (time) datetime64[ns] 2001-01-01 ... 2001-01-10T18:00:00
```

```
  * lon      (lon) float64 -180.0 -178.1 -176.2 -174.4 ... 174.4 176.2 178.1
  * lat      (lat) float64 88.57 86.72 84.86 83.0 ... -83.0 -84.86 -86.72 -88.57
```

List the variables stored in the file:

```
variables = ds.variables
```

Here we can see the time displayed in a readable way, because Xarray use the datetime64 module under the hood. Also the variable and coordinate attributes are displayed.

Read the file and try the above commands.

```
ds = xr.open_dataset('../data/tsurf.nc')

ds.info()
```

```
xarray.Dataset {
dimensions:
time = 40 ;
lon = 192 ;
lat = 96 ;

variables:
datetime64[ns] time(time) ;
time:standard_name = time ;
time:axis = T ;
float64 lon(lon) ;
lon:standard_name = longitude ;
lon:long_name = longitude ;
lon:units = degrees_east ;
lon:axis = X ;
float64 lat(lat) ;
lat:standard_name = latitude ;
lat:long_name = latitude ;
lat:units = degrees_north ;
lat:axis = Y ;
float32 tsurf(time, lat, lon) ;
tsurf:long_name = surface temperature ;
tsurf:units = K ;
tsurf:code = 169 ;
tsurf:table = 128 ;

// global attributes:
:CDI = Climate Data Interface version 1.9.6 (http://mpimet.mpg.de/cdi) ;
:Conventions = CF-1.6 ;
:history = Thu Oct 10 16:08:50 2019: cdo selname,tsurf rectilinear_grid_2D.nc tsurf.nc ;
:CDO = Climate Data Operators version 1.9.6 (http://mpimet.mpg.de/cdo) ;
}
```

```
coords = ds.coords
coords
```

```
Coordinates:
  * time     (time) datetime64[ns] 2001-01-01 ... 2001-01-10T18:00:00
  * lon      (lon) float64 -180.0 -178.1 -176.2 -174.4 ... 174.4 176.2 178.1
  * lat      (lat) float64 88.57 86.72 84.86 83.0 ... -83.0 -84.86 -86.72 -88.57
```

```
variables = ds.variables
variables
```

```
Frozen({'time': <xarray.IndexVariable 'time' (time: 40)>
array(['2001-01-01T00:00:00.000000000', '2001-01-01T06:00:00.000000000',
       '2001-01-01T12:00:00.000000000', '2001-01-01T18:00:00.000000000',
       '2001-01-02T00:00:00.000000000', '2001-01-02T06:00:00.000000000',
       '2001-01-02T12:00:00.000000000', '2001-01-02T18:00:00.000000000',
       '2001-01-03T00:00:00.000000000', '2001-01-03T06:00:00.000000000',
       '2001-01-03T12:00:00.000000000', '2001-01-03T18:00:00.000000000',
       '2001-01-04T00:00:00.000000000', '2001-01-04T06:00:00.000000000',
       '2001-01-04T12:00:00.000000000', '2001-01-04T18:00:00.000000000',
       '2001-01-05T00:00:00.000000000', '2001-01-05T06:00:00.000000000',
       '2001-01-05T12:00:00.000000000', '2001-01-05T18:00:00.000000000',
       '2001-01-06T00:00:00.000000000', '2001-01-06T06:00:00.000000000',
       '2001-01-06T12:00:00.000000000', '2001-01-06T18:00:00.000000000',
       '2001-01-07T00:00:00.000000000', '2001-01-07T06:00:00.000000000',
       '2001-01-07T12:00:00.000000000', '2001-01-07T18:00:00.000000000',
       '2001-01-08T00:00:00.000000000', '2001-01-08T06:00:00.000000000',
       '2001-01-08T12:00:00.000000000', '2001-01-08T18:00:00.000000000',
       '2001-01-09T00:00:00.000000000', '2001-01-09T06:00:00.000000000',
       '2001-01-09T12:00:00.000000000', '2001-01-09T18:00:00.000000000',
       '2001-01-10T00:00:00.000000000', '2001-01-10T06:00:00.000000000',
       '2001-01-10T12:00:00.000000000', '2001-01-10T18:00:00.000000000'],
      dtype='datetime64[ns]')
Attributes:
    standard_name:  time
    axis:           T, 'lon': <xarray.IndexVariable 'lon' (lon: 192)>
array([-180.   , -178.125, -176.25 , -174.375, -172.5  , -170.625, -168.75 ,
       -166.875, -165.   , -163.125, -161.25 , -159.375, -157.5  , -155.625,
       -153.75 , -151.875, -150.   , -148.125, -146.25 , -144.375, -142.5  ,
       -140.625, -138.75 , -136.875, -135.   , -133.125, -131.25 , -129.375,
       -127.5  , -125.625, -123.75 , -121.875, -120.   , -118.125, -116.25 ,
       -114.375, -112.5  , -110.625, -108.75 , -106.875, -105.   , -103.125,
       -101.25 ,  -99.375,  -97.5  ,  -95.625,  -93.75 ,  -91.875,  -90.   ,
        -88.125,  -86.25 ,  -84.375,  -82.5  ,  -80.625,  -78.75 ,  -76.875,
        -75.   ,  -73.125,  -71.25 ,  -69.375,  -67.5  ,  -65.625,  -63.75 ,
        -61.875,  -60.   ,  -58.125,  -56.25 ,  -54.375,  -52.5  ,  -50.625,
        -48.75 ,  -46.875,  -45.   ,  -43.125,  -41.25 ,  -39.375,  -37.5  ,
```

```
       -35.625,  -33.75 ,  -31.875,  -30.   ,  -28.125,  -26.25 ,  -24.375,
       -22.5  ,  -20.625,  -18.75 ,  -16.875,  -15.   ,  -13.125,  -11.25 ,
        -9.375,   -7.5  ,   -5.625,   -3.75 ,   -1.875,    0.   ,    1.875,
         3.75 ,    5.625,    7.5  ,    9.375,   11.25 ,   13.125,   15.   ,
        16.875,   18.75 ,   20.625,   22.5  ,   24.375,   26.25 ,   28.125,
        30.   ,   31.875,   33.75 ,   35.625,   37.5  ,   39.375,   41.25 ,
        43.125,   45.   ,   46.875,   48.75 ,   50.625,   52.5  ,   54.375,
        56.25 ,   58.125,   60.   ,   61.875,   63.75 ,   65.625,   67.5  ,
        69.375,   71.25 ,   73.125,   75.   ,   76.875,   78.75 ,   80.625,
        82.5  ,   84.375,   86.25 ,   88.125,   90.   ,   91.875,   93.75 ,
        95.625,   97.5  ,   99.375,  101.25 ,  103.125,  105.   ,  106.875,
       108.75 ,  110.625,  112.5  ,  114.375,  116.25 ,  118.125,  120.   ,
       121.875,  123.75 ,  125.625,  127.5  ,  129.375,  131.25 ,  133.125,
       135.   ,  136.875,  138.75 ,  140.625,  142.5  ,  144.375,  146.25 ,
       148.125,  150.   ,  151.875,  153.75 ,  155.625,  157.5  ,  159.375,
       161.25 ,  163.125,  165.   ,  166.875,  168.75 ,  170.625,  172.5  ,
       174.375,  176.25 ,  178.125])
Attributes:
    standard_name:  longitude
    long_name:      longitude
    units:          degrees_east
    axis:           X, 'lat': <xarray.IndexVariable 'lat' (lat: 96)>
array([ 88.572169,  86.722531,  84.86197 ,  82.998942,  81.134977,  79.270559,
        77.405888,  75.541061,  73.676132,  71.811132,  69.946081,  68.080991,
        66.215872,  64.35073 ,  62.485571,  60.620396,  58.755209,  56.890013,
        55.024808,  53.159595,  51.294377,  49.429154,  47.563926,  45.698694,
        43.833459,  41.96822 ,  40.102979,  38.237736,  36.372491,  34.507243,
        32.641994,  30.776744,  28.911492,  27.046239,  25.180986,  23.315731,
        21.450475,  19.585219,  17.719962,  15.854704,  13.989446,  12.124187,
        10.258928,   8.393669,   6.528409,   4.66315 ,   2.79789 ,   0.93263 ,
        -0.93263 ,  -2.79789 ,  -4.66315 ,  -6.528409,  -8.393669, -10.258928,
       -12.124187, -13.989446, -15.854704, -17.719962, -19.585219, -21.450475,
       -23.315731, -25.180986, -27.046239, -28.911492, -30.776744, -32.641994,
       -34.507243, -36.372491, -38.237736, -40.102979, -41.96822 , -43.833459,
       -45.698694, -47.563926, -49.429154, -51.294377, -53.159595, -55.024808,
       -56.890013, -58.755209, -60.620396, -62.485571, -64.35073 , -66.215872,
       -68.080991, -69.946081, -71.811132, -73.676132, -75.541061, -77.405888,
       -79.270559, -81.134977, -82.998942, -84.86197 , -86.722531, -88.572169])
Attributes:
    standard_name:  latitude
    long_name:      latitude
    units:          degrees_north
    axis:           Y, 'tsurf': <xarray.Variable (time: 40, lat: 96, lon: 192)>
[737280 values with dtype=float32]
Attributes:
    long_name:  surface temperature
```

```
    units:      K
    code:       169
    table:      128})
```

Dimensions, shape and size

To get more informations about the dimension, shape and size of a **Dataset**, we can use the appropriate attributes.

```
dims  = ds.dims
shape = tsurf.shape
size  = tsurf.size
rank  = len(shape)

print('dimensions: ', dims)
print('shape:      ', shape)
print('size:       ', size)
print('rank:       ', rank)
```

```
tsurf = ds.tsurf
```

```
dims  = ds.dims
shape = tsurf.shape
size  = tsurf.size
rank  = len(shape)
```

Read another file format

Xarray needs an *engine* to read another file format. Here, we demonstrate how to read a GRIB file using the **cfgrib** *engine* from the additional library **cfgrib** (don't forget to import it).

```
import cfgrib

ds2 = xr.open_dataset('../data/MET9_IR108_cosmode_0909210000.grb2',
                      engine='cfgrib')

variables2 = ds2.variables
```

Read the GRIB file yourself.

```
import cfgrib

ds2 = xr.open_dataset('../data/MET9_IR108_cosmode_0909210000.grb2',
                      engine='cfgrib')
```

```
variables2 = ds2.variables
variables2
```

```
Frozen({'time': <xarray.Variable ()>
array('2009-09-21T00:00:00.000000000', dtype='datetime64[ns]')
Attributes:
    long_name:      initial time of forecast
    standard_name:  forecast_reference_time, 'latitude': <xarray.Variable (y: 461, x: 421)>
[194081 values with dtype=float64]
Attributes:
    units:          degrees_north
    standard_name:  latitude
    long_name:      latitude, 'longitude': <xarray.Variable (y: 461, x: 421)>
[194081 values with dtype=float64]
Attributes:
    units:          degrees_east
    standard_name:  longitude
    long_name:      longitude, 'p260532': <xarray.Variable (y: 461, x: 421)>
[194081 values with dtype=float32]
Attributes:
    GRIB_paramId:                             500393
    GRIB_shortName:                           OBSMSG_BT_IR10.8
    GRIB_units:                               Numeric
    GRIB_name:                                Obser. Sat. Meteosat sec. gener...
    GRIB_cfVarName:                           p260532
    GRIB_dataType:                            sa
    GRIB_missingValue:                        9999
    GRIB_numberOfPoints:                      194081
    GRIB_NV:                                  0
    GRIB_stepType:                            instant
    GRIB_gridType:                            rotated_ll
    GRIB_gridDefinitionDescription:           Rotated latitude/longitude
    GRIB_Nx:                                  421
    GRIB_Ny:                                  461
    GRIB_angleOfRotationInDegrees:            0.0
    GRIB_iDirectionIncrementInDegrees:        0.024994
    GRIB_iScansNegatively:                    0
    GRIB_jDirectionIncrementInDegrees:        0.024994
    GRIB_jPointsAreConsecutive:               0
    GRIB_jScansPositively:                    0
    GRIB_latitudeOfFirstGridPointInDegrees:   6.499786
    GRIB_latitudeOfLastGridPointInDegrees:    -4.996185
    GRIB_latitudeOfSouthernPoleInDegrees:     -40.0
    GRIB_longitudeOfFirstGridPointInDegrees:  -5.002594
    GRIB_longitudeOfLastGridPointInDegrees:   5.498184
    GRIB_longitudeOfSouthernPoleInDegrees:    10.0
```

```
    long_name:                                  Obser. Sat. Meteosat sec. gener...
    units:                                      Numeric})
```

Open multiple files

In the course directory **data** there are 3 files *precip_day01.nc, precip_day02.nc, and precip_day03.nc*, each containing the data of one day in 6 hour intervals.

Xarray provides the function `open_mfdataset` to read multiple files in one step as a single dataset. Before you can use `open_mfdataset` make sure that the Python module **dask** is installed in your environment.

```
dsm = xr.open_mfdataset('../data/precip_day*.nc')
dsm
```

```
<xarray.Dataset>
Dimensions:  (time: 12, lon: 192, lat: 96)
Coordinates:
  * time     (time) datetime64[ns] 2001-01-01 ... 2001-01-03T18:00:00
  * lon      (lon) float64 -180.0 -178.1 -176.2 -174.4 ... 174.4 176.2 178.1
  * lat      (lat) float64 88.57 86.72 84.86 83.0 ... -83.0 -84.86 -86.72 -88.57
Data variables:
    precip   (time, lat, lon) float32 dask.array<chunksize=(4, 96, 192), meta=np.ndarray>
Attributes:
    CDI:          Climate Data Interface version 1.9.9 (https://mpimet.mpg.de...
    Conventions:  CF-1.6
    history:      Wed Jul 14 15:47:55 2021: cdo -splitday -selname,precip rec...
    CDO:          Climate Data Operators version 1.9.9 (https://mpimet.mpg.de...
```

One reason why `xarray` is very fast with multiple files is that it does not **load** the data when the files are opened. This is possible by using an underlying library named `dask`. You can recognize that by checking for the `precip` variable in `dsm`.

```
dsm.precip[1,4,5]
```

will not show you an exact value but only a description of what this output will be. You would have to load the data into memory first for accessing one specific point of the array. This is most often not necessary for your workflow.

The entire array can be loaded into memory by `dsm.precip.load()`. You can also do:

```
dsm.precip.values[1,4,5]
```

While data is not in loaded, you can work on files that are *larger than memory.*

```
dsm.precip[1,4,5]
```

```
<xarray.DataArray 'precip' ()>
dask.array<getitem, shape=(), dtype=float32, chunksize=(), chunktype=numpy.ndarray>
Coordinates:
    time     datetime64[ns] 2001-01-01T06:00:00
    lon      float64 -170.6
    lat      float64 81.13
Attributes:
    long_name:      total precipitation
    units:          kg/m^2s
    code:           4
    table:          128
    CDI_grid_type:  gaussian
```

```
dsm.precip.load()
dsm.precip[1,4,5]
# is the same as
dsm.precip.values[1,4,5]
```

```
2.7939677e-08
```

The open_mfdataset function is very powerful. It contains over **10 arguments** which allow users to configure how the files are combined:

- On what dimension should the data be concatted
- How strict should tests ensure that the data can be concatted
- What are coordinates, what are data variables

Dataset attributes

Dataset attributes and variable attributes are important for understanding what the data represents not only for human but also the machine. Therefore, it is important that they are available and have a standard format. In addition to the attributes of a `DataArray`, there also **global** or dataset attributes.

```
tas_hr=xr.open_dataset("/work/ik1017/CMIP6/data/CMIP6/ScenarioMIP/DKRZ/MPI-ESM1-2-HR/ssp370/
tas_hr_atts = list(tas_hr.attrs)
global_atts = list(tas_hr.attrs)
print(global_atts)
```

Assumed that we know the variable and attribute names, we can get their content immediately.

```
units = tas_hr.tas.units

print('units:     ', units)
```

```
tas_hr=xr.open_dataset("/work/ik1017/CMIP6/data/CMIP6/ScenarioMIP/DKRZ/MPI-ESM1-2-HR/ssp370/
tas_hr_atts = list(tas_hr.attrs)
global_atts = list(tas_hr.attrs)
print(global_atts)
```

```
['Conventions', 'activity_id', 'branch_method', 'branch_time_in_child', 'branch_time_in_pare
```

List the attributes of the variable *tas* and print their content.

```
tas_hr.tas.attrs
```

```
{'standard_name': 'air_temperature',
 'long_name': 'Near-Surface Air Temperature',
 'comment': 'near-surface (usually, 2 meter) air temperature',
 'units': 'K',
 'cell_methods': 'area: time: mean',
 'cell_measures': 'area: areacella',
 'history': "2019-07-20T13:41:51Z altered by CMOR: Treated scalar dimension: 'height'. 2019-
```