Manipulating data and metadata in cf-python

Homepage [https://ncas-cms.github.io/cf-python (https://ncas-cms.github.io/cf-python)](https://ncas-cms.github.io/cf-python) for background, tutorial, reference, and installation

Contents:

1. Read, inspect, write netCDF files

2. Subspace

3. Data

4. Calculate statistics

5. PP and UM datasets

6. What this course doesn't cover

1. Read, inspect and write files

[https://ncas-cms.github.io/cf-python/function/cf.read.html (https://ncas-cms.github.io/cf-python/function/cf.read.html)](https://ncas-cms.github.io/cf-python/function/cf.read.html)

In [1]:

```
import cf
cf.__version__
```

Out[1]:

```
'3.1.0'
```

In [2]:

```
cf.CF()
```

Out[2]:

```
'1.7'
```

In [3]:

```
f = cf.read('ncas_data/IPSL-CM5A-LR_r1i1p1_tas_n96_rcp45_mnth.nc')[0]
```

In [4]:

```
f
```

Out[4]:

```
<CF Field: air_temperature(time(120), latitude(145), longitude(192)) K>
```

In [5]:

```
print(f)
```

```
Field: air_temperature (ncvar%tas)
----------------------------------
Data            : air_temperature(time(120), latitude(145), longitude(192)) K
Cell methods    : time(120): mean (interval: 30 minutes)
Dimension coords: time(120) = [1959-12-16 12:00:00, ..., 1969-11-16 00:00:00] 365_day
                : latitude(145) = [-90.0, ..., 90.0] degrees_north
                : longitude(192) = [0.0, ..., 358.125] degrees_east
                : height(1) = [2.0] m
```

[https://ncas-cms.github.io/cf-python/method/cf.Field.dump.html (https://ncas-cms.github.io/cf-python/method/cf.Field.dump.html)](https://ncas-cms.github.io/cf-python/method/cf.Field.dump.html)

In [6]:

```
f.dump()
```

```
----------------------------------
Field: air_temperature (ncvar%tas)
----------------------------------
CDI = 'Climate Data Interface version 1.7.0 (http://mpimet.mpg.de/cdi)'
CDO = 'Climate Data Operators version 1.7.0 (http://mpimet.mpg.de/cdo)'
Conventions = 'CF-1.5'
_FillValue = 1.0000000200408773e+20
associated_files = 'baseURL: http://cmip-pcmdi.llnl.gov/CMIP5/dataLocation
                    gridspecFile: gridspec_atmos_fx_IPSL-
                    CM5A-LR_historical_r0i0p0.nc areacella: areacella_fx_IPSL-
                    CM5A-LR_historical_r0i0p0.nc'
branch_time = 1850.0
cmor_version = '2.5.1'
comment = 'This 20th century simulation include natural and anthropogenic
           forcings.'
contact = 'ipsl-cmip5 _at_ ipsl.jussieu.fr Data manager : Sebastien Denvil'
creation_date = '2011-02-23T17:52:35Z'
experiment = 'historical'
experiment_id = 'historical'
forcing = 'Nat,Ant,GHG,SA,Oz,LU,SS,Ds,BC,MD,OC,AA'
frequency = 'mon'
history = "Thu May 26 15:47:13 2016: cdo mergetime /data/cr1/hadlg/helix/IPSL-
           CM5A-LR_rcp45_tmp_output_1_hist.nc /data/cr1/hadlg/helix/IPSL-
           CM5A-LR_rcp45_tmp_output_1_fut.nc /data/cr1/hadlg/helix/IPSL-
           CM5A-LR_r1i1p1_tas_merged_rcp45.nc\n2011-06-24T02:32:44Z altered by
           CMOR: Treated scalar dimension: 'height'. 2011-06-24T02:32:44Z
           altered by CMOR: replaced missing value flag (9.96921e+36) with
           standard missing value (1e+20). 2011-06-24T02:32:45Z altered by
           CMOR: Inverted axis: lat."
initialization_method = 1
institute_id = 'IPSL'
institution = 'IPSL (Institut Pierre Simon Laplace, Paris, France)'
long_name = 'Near-Surface Air Temperature'
missing_value = 1e+20
model_id = 'IPSL-CM5A-LR'
modeling_realm = 'atmos'
original_name = 't2m'
parent_experiment = 'pre-industrial control'
parent_experiment_id = 'piControl'
parent_experiment_rip = 'r1i1p1'
physics_version = 1
product = 'output'
project_id = 'CMIP5'
realization = 1
references = 'Model documentation and further reference available here :
              http://icmc.ipsl.fr'
source = 'IPSL-CM5A-LR (2010) : atmos : LMDZ4 (LMDZ4_v5, 96x95x39); ocean :
          ORCA2 (NEMOV2_3, 2x2L31); seaIce : LIM2 (NEMOV2_3); ocnBgchem :
          PISCES (NEMOV2_3); land : ORCHIDEE (orchidee_1_9_4_AR5)'
standard_name = 'air_temperature'
table_id = 'Table Amon (31 January 2011) 53b766a395ac41696af40aab76a49ae5'
title = 'IPSL-CM5A-LR model output prepared for CMIP5 historical'
tracking_id = '826ee5e9-3cc9-40a6-a42b-d84c6b4aad97'
units = 'K'

Data(time(120), latitude(145), longitude(192)) = [[[244.82579040527344, ..., 244.52688598632812
]]] K

Cell Method: time(120): mean (interval: 30 minutes)
```

```
Domain Axis: height(1)
Domain Axis: latitude(145)
Domain Axis: longitude(192)
Domain Axis: time(120)

Dimension coordinate: time
    axis = 'T'
    calendar = '365_day'
    long_name = 'time'
    standard_name = 'time'
    units = 'days since 1850-1-1 00:00:00'
    Data(time(120)) = [1959-12-16 12:00:00, ..., 1969-11-16 00:00:00] 365_day
    Bounds:calendar = '365_day'
    Bounds:units = 'days since 1850-1-1 00:00:00'
    Bounds:Data(time(120), 2) = [[1959-12-01 00:00:00, ..., 1969-12-01 00:00:00]] 365_day

Dimension coordinate: latitude
    axis = 'Y'
    long_name = 'latitude'
    standard_name = 'latitude'
    units = 'degrees_north'
    Data(latitude(145)) = [-90.0, ..., 90.0] degrees_north
    Bounds:units = 'degrees_north'
    Bounds:Data(latitude(145), 2) = [[-90.0, ..., 90.0]] degrees_north

Dimension coordinate: longitude
    axis = 'X'
    long_name = 'longitude'
    standard_name = 'longitude'
    units = 'degrees_east'
    Data(longitude(192)) = [0.0, ..., 358.125] degrees_east
    Bounds:units = 'degrees_east'
    Bounds:Data(longitude(192), 2) = [[-0.9375, ..., 359.0625]] degrees_east

Dimension coordinate: height
    axis = 'Z'
    long_name = 'height'
    positive = 'up'
    standard_name = 'height'
    units = 'm'
    Data(height(1)) = [2.0] m
```

Properties

https://ncas-cms.github.io/cf-python/method/cf.Field.properties.html (https://ncas-cms.github.io/cf-python/method/cf.Field.properties.html)

In [7]:

```
f.properties()
```

Out[7]:

```
{'Conventions': 'CF-1.5',
 'comment': 'This 20th century simulation include natural and anthropogenic forcings.',
 'model_id': 'IPSL-CM5A-LR',
 'CDI': 'Climate Data Interface version 1.7.0 (http://mpimet.mpg.de/cdi)',
 'parent_experiment_id': 'piControl',
 'creation_date': '2011-02-23T17:52:35Z',
 'frequency': 'mon',
 'references': 'Model documentation and further reference available here : http://icmc.ipsl.fr'
,
 'title': 'IPSL-CM5A-LR model output prepared for CMIP5 historical',
 'original_name': 't2m',
 'contact': 'ipsl-cmip5 _at_ ipsl.jussieu.fr Data manager : Sebastien Denvil',
 'source': 'IPSL-CM5A-LR (2010) : atmos : LMDZ4 (LMDZ4_v5, 96x95x39); ocean : ORCA2 (NEMOV2_3,
2x2L31); seaIce : LIM2 (NEMOV2_3); ocnBgchem : PISCES (NEMOV2_3); land : ORCHIDEE (orchidee_1_9
_4_AR5)',
 'experiment': 'historical',
 'realization': 1,
 'project_id': 'CMIP5',
 'institute_id': 'IPSL',
 'initialization_method': 1,
 'product': 'output',
 'tracking_id': '826ee5e9-3cc9-40a6-a42b-d84c6b4aad97',
 'cmor_version': '2.5.1',
 'parent_experiment': 'pre-industrial control',
 'branch_time': 1850.0,
 'institution': 'IPSL (Institut Pierre Simon Laplace, Paris, France)',
 'forcing': 'Nat,Ant,GHG,SA,Oz,LU,SS,Ds,BC,MD,OC,AA',
 'CDO': 'Climate Data Operators version 1.7.0 (http://mpimet.mpg.de/cdo)',
 'physics_version': 1,
 'associated_files': 'baseURL: http://cmip-pcmdi.llnl.gov/CMIP5/dataLocation gridspecFile: grid
spec_atmos_fx_IPSL-CM5A-LR_historical_r0i0p0.nc areacella: areacella_fx_IPSL-CM5A-LR_historical
_r0i0p0.nc',
 'modeling_realm': 'atmos',
 'table_id': 'Table Amon (31 January 2011) 53b766a395ac41696af40aab76a49ae5',
 'experiment_id': 'historical',
 'history': "Thu May 26 15:47:13 2016: cdo mergetime /data/cr1/hadlg/helix/IPSL-CM5A-LR_rcp45_t
mp_output_1_hist.nc /data/cr1/hadlg/helix/IPSL-CM5A-LR_rcp45_tmp_output_1_fut.nc /data/cr1/hadl
g/helix/IPSL-CM5A-LR_r1i1p1_tas_merged_rcp45.nc\n2011-06-24T02:32:44Z altered by CMOR: Treated
scalar dimension: 'height'. 2011-06-24T02:32:44Z altered by CMOR: replaced missing value flag (
9.96921e+36) with standard missing value (1e+20). 2011-06-24T02:32:45Z altered by CMOR: Inverte
d axis: lat.",
 'parent_experiment_rip': 'r1i1p1',
 '_FillValue': 1.0000000200408773e+20,
 'long_name': 'Near-Surface Air Temperature',
 'standard_name': 'air_temperature',
 'missing_value': 1e+20,
 'units': 'K'}
```

In [8]:

```
f.get_property('project_id')
```

Out[8]:

```
'CMIP5'
```

In [9]:

```
f.set_property('project_id', 'banana')
f.get_property('project_id')
```

Out[9]:

```
'banana'
```

In [10]:

```
f.del_property('project_id')
f.get_property('project_id')  # This should fail!
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
/share/apps/NCAS/training/lib/python3.7/site-packages/cfdm/core/abstract/properties.py in get_p
roperty(self, prop, default)
    190         try:
--> 191             return self._get_component('properties')[prop]
    192         except KeyError:

KeyError: 'project_id'

During handling of the above exception, another exception occurred:

ValueError                                Traceback (most recent call last)
<ipython-input-10-cb231cbd51a2> in <module>
      1 f.del_property('project_id')
----> 2 f.get_property('project_id')  # This should fail!

/share/apps/NCAS/training/lib/python3.7/site-packages/cf/mixin/properties.py in get_property(se
lf, prop, default)
    557
    558         # Still here? Then get a non-special property
--> 559         return super().get_property(prop, default=default)
    560
    561

/share/apps/NCAS/training/lib/python3.7/site-packages/cfdm/core/abstract/properties.py in get_p
roperty(self, prop, default)
    193             return self._default(default,
    194                                  "{!r} has no {!r} property".format(
--> 195                                      self.__class__.__name__, prop))
    196
    197

/share/apps/NCAS/training/lib/python3.7/site-packages/cfdm/core/abstract/container.py in _defau
lt(self, default, message)
     87                 default.args = (message,)
     88
---> 89             raise default
     90
     91         return default

ValueError: 'Field' has no 'project_id' property
```

In [11]:

```
f.get_property('project_id', 'no project')
```

Out[11]:

```
'no project'
```

[https://ncas-cms.github.io/cf-python/method/cf.Field.get_property.html (https://ncas-cms.github.io/cf-python/method/cf.Field.get_property.html)](https://ncas-cms.github.io/cf-python/method/cf.Field.get_property.html)

In [12]:

```
help(f.get_property)
```

```
Help on method get_property in module cf.mixin.properties:

get_property(prop, default=ValueError()) method of cf.field.Field instance
    Get a CF property.

    .. versionadded:: 3.0.0

    .. seealso:: `clear_properties`, `del_property`, `has_property`,
                 `properties`, `set_property`

    :Parameters:

        prop: `str`
            The name of the CF property.

            *Parameter example:*
              ``prop='long_name'``

        default: optional
            Return the value of the *default* parameter if the
            property does not exist. If set to an `Exception` instance
            then it will be raised instead.

    :Returns:

            The value of the named property or the default value, if
            set.

    **Examples:**


    >>> f.set_property('project', 'CMIP7')
    >>> f.has_property('project')
    True
    >>> f.get_property('project')
    'CMIP7'
    >>> f.del_property('project')
    'CMIP7'
    >>> f.has_property('project')
    False
    >>> print(f.del_property('project', None))
    None
    >>> print(f.get_property('project', None))
    None
```

Shorthand for named CF properties

http://cfconventions.org/Data/cf-conventions/cf-conventions-1.7/cf-conventions.html#attribute-appendix
(http://cfconventions.org/Data/cf-conventions/cf-conventions-1.7/cf-conventions.html#attribute-appendix)

In [13]:

```
print(f.standard_name)
f.standard_name = 'banana'
print(f.standard_name)
del(f.standard_name)
f.standard_name = 'air_temperature'
print(f.standard_name)
```

```
air_temperature
banana
air_temperature
```

Reading many files

In [14]:

```
fl = cf.read('$PWD/ncas_data/data[2-7].nc')
print(type(fl))
fl
```

```
<class 'cf.fieldlist.FieldList'>
```

Out[14]:

```
[<CF Field: air_temperature(long_name=t(1), long_name=p(1), long_name=latitude(256), long_name=
longitude(512)) K>,
 <CF Field: air_temperature(long_name=t(1), long_name=p(1), latitude(160), longitude(320)) K>,
 <CF Field: eastward_wind(time(1), pressure(23), latitude(160), longitude(320)) m s**-1>]
```

A `FieldList` object inherits all of the usual Python `list` functionality

In [15]:

```
for x in fl:
    print('IDENTITY:', x.identity(), 'SHAPE:', x.shape, 'UNITS:', x.units)
```

```
IDENTITY: air_temperature SHAPE: (1, 1, 256, 512) UNITS: K
IDENTITY: air_temperature SHAPE: (1, 1, 160, 320) UNITS: K
IDENTITY: eastward_wind SHAPE: (1, 23, 160, 320) UNITS: m s**-1
```

Select by list position with the usual list indices

In [16]:

```
g = fl[0]
g
```

Out[16]:

```
<CF Field: air_temperature(long_name=t(1), long_name=p(1), long_name=latitude(256), long_name=l
ongitude(512)) K>
```

In [17]:

```
fl[1:]
```

Out[17]:

```
[<CF Field: air_temperature(long_name=t(1), long_name=p(1), latitude(160), longitude(320)) K>,
 <CF Field: eastward_wind(time(1), pressure(23), latitude(160), longitude(320)) m s**-1>]
```

In [18]:

```
fl[4:]
```

Out[18]:

```
[]
```

Select by metadata

[https://ncas-cms.github.io/cf-python/tutorial.html#sorting-and-selecting-from-field-lists (https://ncas-cms.github.io/cf-python/tutorial.html#sorting-and-selecting-from-field-lists)](https://ncas-cms.github.io/cf-python/tutorial.html#sorting-and-selecting-from-field-lists)

In [19]:

```
fl.select('air_temperature')
```

Out[19]:

```
[<CF Field: air_temperature(long_name=t(1), long_name=p(1), long_name=latitude(256), long_name=
longitude(512)) K>,
 <CF Field: air_temperature(long_name=t(1), long_name=p(1), latitude(160), longitude(320)) K>]
```

In [20]:

```
fl.select('northward_wind')
```

Out[20]:

```
[]
```

In [21]:

```
fl.select_by_units('km h-1')
```

Out[21]:

```
[]
```

In [22]:

```
fl.select_by_units('km h-1', exact=False)
```

Out[22]:

```
[<CF Field: eastward_wind(time(1), pressure(23), latitude(160), longitude(320)) m s**-1>]
```

In [23]:

```
import re
fl.select(re.compile('(east|north)ward_wind'))
```

Out[23]:

```
[<CF Field: eastward_wind(time(1), pressure(23), latitude(160), longitude(320)) m s**-1>]
```

Write fields to a netCDF file

[https://ncas-cms.github.io/cf-python/function/cf.write.html (https://ncas-cms.github.io/cf-python/function/cf.write.html)](https://ncas-cms.github.io/cf-python/function/cf.write.html)

In [24]:

```
cf.write(f, 'new_file.nc')
```

[https://ncas-cms.github.io/cf-python/method/cf.Field.equals.html (https://ncas-cms.github.io/cf-python/method/cf.Field.equals.html)](https://ncas-cms.github.io/cf-python/method/cf.Field.equals.html)

In [25]:

```
g = cf.read('new_file.nc')[0]
f.equals(g)
```

Out[25]:

```
True
```

2. Subspace a field

Index-space: [square brackets]

[https://ncas-cms.github.io/cf-python/tutorial.html#subspacing-by-index (https://ncas-cms.github.io/cf-python/tutorial.html#subspacing-by-index)](https://ncas-cms.github.io/cf-python/tutorial.html#subspacing-by-index)

In [26]:

```
print(f)
```

```
Field: air_temperature (ncvar%tas)
----------------------------------
Data            : air_temperature(time(120), latitude(145), longitude(192)) K
Cell methods    : time(120): mean (interval: 30 minutes)
Dimension coords: time(120) = [1959-12-16 12:00:00, ..., 1969-11-16 00:00:00] 365_day
                : latitude(145) = [-90.0, ..., 90.0] degrees_north
                : longitude(192) = [0.0, ..., 358.125] degrees_east
                : height(1) = [2.0] m
```

In [27]:

```
print(f[0, 0, 0])
```

```
Field: air_temperature (ncvar%tas)
----------------------------------
Data            : air_temperature(time(1), latitude(1), longitude(1)) K
Cell methods    : time(1): mean (interval: 30 minutes)
Dimension coords: time(1) = [1959-12-16 12:00:00] 365_day
                : latitude(1) = [-90.0] degrees_north
                : longitude(1) = [0.0] degrees_east
                : height(1) = [2.0] m
```

In [28]:

```
print(f[0:6, :, :])
```

```
Field: air_temperature (ncvar%tas)
----------------------------------
Data            : air_temperature(time(6), latitude(145), longitude(192)) K
Cell methods    : time(6): mean (interval: 30 minutes)
Dimension coords: time(6) = [1959-12-16 12:00:00, ..., 1960-05-16 12:00:00] 365_day
                : latitude(145) = [-90.0, ..., 90.0] degrees_north
                : longitude(192) = [0.0, ..., 358.125] degrees_east
                : height(1) = [2.0] m
```

Metadata-space: (`subspace` method)

[https://ncas-cms.github.io/cf-python/tutorial.html#subspacing-by-metadata (https://ncas-cms.github.io/cf-python/tutorial.html#subspacing-by-metadata)](https://ncas-cms.github.io/cf-python/tutorial.html#subspacing-by-metadata)

In [29]:

```
print(f)
```

```
Field: air_temperature (ncvar%tas)
----------------------------------
Data            : air_temperature(time(120), latitude(145), longitude(192)) K
Cell methods    : time(120): mean (interval: 30 minutes)
Dimension coords: time(120) = [1959-12-16 12:00:00, ..., 1969-11-16 00:00:00] 365_day
                : latitude(145) = [-90.0, ..., 90.0] degrees_north
                : longitude(192) = [0.0, ..., 358.125] degrees_east
                : height(1) = [2.0] m
```

In [30]:

```
print(f.subspace(longitude=180))
```

```
Field: air_temperature (ncvar%tas)
----------------------------------
Data            : air_temperature(time(120), latitude(145), longitude(1)) K
Cell methods    : time(120): mean (interval: 30 minutes)
Dimension coords: time(120) = [1959-12-16 12:00:00, ..., 1969-11-16 00:00:00] 365_day
                : latitude(145) = [-90.0, ..., 90.0] degrees_north
                : longitude(1) = [180.0] degrees_east
                : height(1) = [2.0] m
```

`cf.lt(30)` is a "query" that means *less than 30*

[https://ncas-cms.github.io/cf-python/tutorial.html#encapsulating-conditions (https://ncas-cms.github.io/cf-python/tutorial.html#encapsulating-conditions)](https://ncas-cms.github.io/cf-python/tutorial.html#encapsulating-conditions)

In [31]:

```
print(f.subspace(latitude=cf.lt(30)))
```

```
Field: air_temperature (ncvar%tas)
----------------------------------
Data            : air_temperature(time(120), latitude(96), longitude(192)) K
Cell methods    : time(120): mean (interval: 30 minutes)
Dimension coords: time(120) = [1959-12-16 12:00:00, ..., 1969-11-16 00:00:00] 365_day
                : latitude(96) = [-90.0, ..., 28.75] degrees_north
                : longitude(192) = [0.0, ..., 358.125] degrees_east
                : height(1) = [2.0] m
```

cf.wi(90, 270) is a query that means ***within the range [90, 270]***

In [32]:

```
print(f.subspace(longitude=cf.wi(90, 270)))
```

```
Field: air_temperature (ncvar%tas)
----------------------------------
Data            : air_temperature(time(120), latitude(145), longitude(97)) K
Cell methods    : time(120): mean (interval: 30 minutes)
Dimension coords: time(120) = [1959-12-16 12:00:00, ..., 1969-11-16 00:00:00] 365_day
                : latitude(145) = [-90.0, ..., 90.0] degrees_north
                : longitude(97) = [90.0, ..., 270.0] degrees_east
                : height(1) = [2.0] m
```

In [33]:

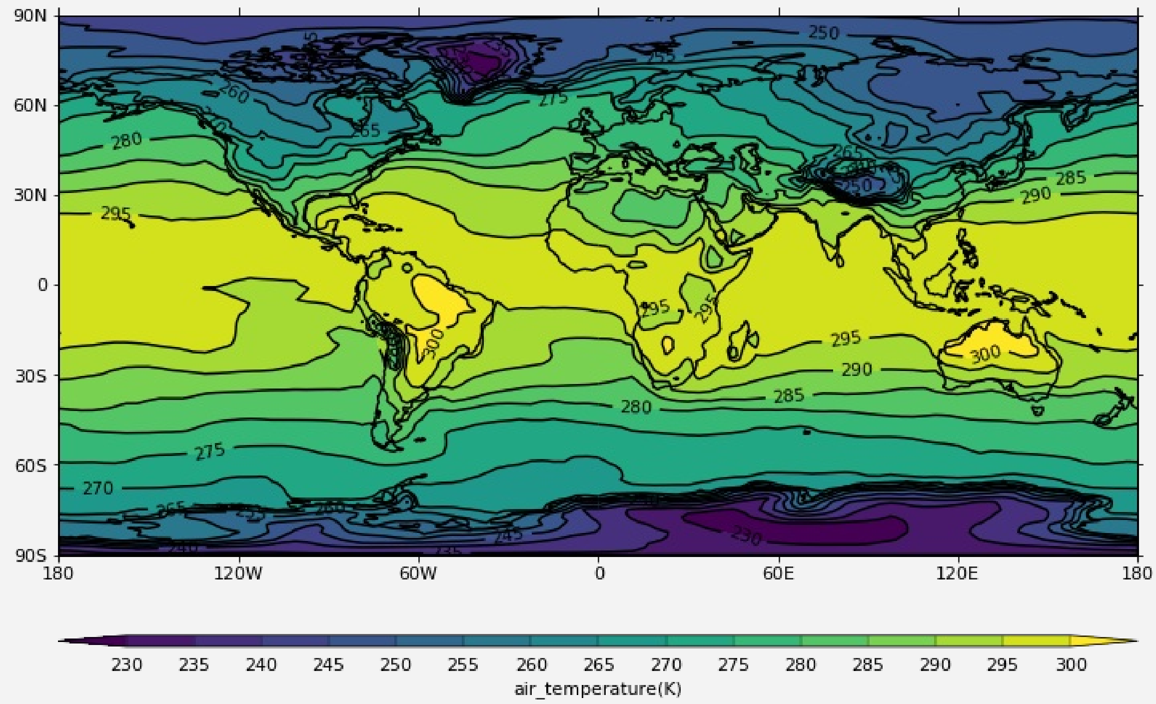
```
g = f.subspace(time=cf.dt('1965-11-16'))
print(g)
```

```
Field: air_temperature (ncvar%tas)
----------------------------------
Data            : air_temperature(time(1), latitude(145), longitude(192)) K
Cell methods    : time(1): mean (interval: 30 minutes)
Dimension coords: time(1) = [1965-11-16 00:00:00] 365_day
                : latitude(145) = [-90.0, ..., 90.0] degrees_north
                : longitude(192) = [0.0, ..., 358.125] degrees_east
                : height(1) = [2.0] m
```

In [34]:

```
# In-line images
%matplotlib inline

import cfplot as cfp
cfp.con(g)
```



T is shorthand for ***time***

In [35]:

```
print(f.subspace(T=cf.ge(cf.dt('1967-2-18'))))
```

```
Field: air_temperature (ncvar%tas)
----------------------------------
Data            : air_temperature(time(33), latitude(145), longitude(192)) K
Cell methods    : time(33): mean (interval: 30 minutes)
Dimension coords: time(33) = [1967-03-16 12:00:00, ..., 1969-11-16 00:00:00] 365_day
                : latitude(145) = [-90.0, ..., 90.0] degrees_north
                : longitude(192) = [0.0, ..., 358.125] degrees_east
                : height(1) = [2.0] m
```

In [36]:

```
print(f.subspace(T=cf.month(4)))
```

```
Field: air_temperature (ncvar%tas)
----------------------------------
Data            : air_temperature(time(10), latitude(145), longitude(192)) K
Cell methods    : time(10): mean (interval: 30 minutes)
Dimension coords: time(10) = [1960-04-16 00:00:00, ..., 1969-04-16 00:00:00] 365_day
                : latitude(145) = [-90.0, ..., 90.0] degrees_north
                : longitude(192) = [0.0, ..., 358.125] degrees_east
                : height(1) = [2.0] m
```

In [37]:

```
print(f.subspace(time=cf.dt('1965-11-16'), Y=cf.gt(30)))
```

```
Field: air_temperature (ncvar%tas)
----------------------------------
Data            : air_temperature(time(1), latitude(48), longitude(192)) K
Cell methods    : time(1): mean (interval: 30 minutes)
Dimension coords: time(1) = [1965-11-16 00:00:00] 365_day
                : latitude(48) = [31.25, ..., 90.0] degrees_north
                : longitude(192) = [0.0, ..., 358.125] degrees_east
                : height(1) = [2.0] m
```

3. The field's data

In [38]:

```
f.data
```

Out[38]:

```
<CF Data(120, 145, 192): [[[244.82579040527344, ..., 244.52688598632812]]] K>
```

Get the data as a `numpy` array

In [39]:

```
print(type(f.array))
```

```
<class 'numpy.ndarray'>
```

In [40]:

```
f.array
```

Out[40]:

```
array([[[244.82579041, 244.82579041, 244.82579041, ..., 244.82579041,
         244.82579041, 244.82579041],
        [245.76259871, 245.64571488, 245.52913189, ..., 246.10911099,
         246.01224121, 245.88722523],
        [246.0103291 , 245.86191647, 245.71379773, ..., 246.47294155,
         246.33730691, 246.17361118],
        ...,
        [246.92743832, 246.92339943, 246.91909425, ..., 246.96167234,
         246.95784869, 246.94273518],
        [246.83550681, 246.83572591, 246.8362101 , ..., 246.84365246,
         246.84140166, 246.83832122],
        [246.11326599, 246.11326599, 246.11326599, ..., 246.11326599,
         246.11326599, 246.11326599]],

       [[246.98564148, 246.98564148, 246.98564148, ..., 246.98564148,
```

```
         246.98564148, 246.98564148],

        [248.46694996, 248.35942057, 248.25239525, ..., 248.76876914,
         248.68722049, 248.57679331],
        [248.94832661, 248.81420465, 248.68104777, ..., 249.34295834,
         249.23124955, 249.08926564],
        ...,
        [244.75140282, 244.79450904, 244.83979468, ..., 244.6257257 ,
         244.63474002, 244.69223537],
        [244.26617971, 244.26601925, 244.26605184, ..., 244.25610468,
         244.26110323, 244.26354541],
        [243.73991394, 243.73991394, 243.73991394, ..., 243.73991394,
         243.73991394, 243.73991394]],

       [[238.64672852, 238.64672852, 238.64672852, ..., 238.64672852,
         238.64672852, 238.64672852],
        [240.90044159, 240.79150484, 240.68335504, ..., 241.2274457 ,
         241.13952195, 241.01954646],
        [241.5250896 , 241.37456484, 241.22489827, ..., 241.95320127,
         241.8409942 , 241.68252141],
        ...,
        [248.00269058, 248.04255923, 248.08195776, ..., 247.92383843,
         247.93659988, 247.96997174],
        [248.07975765, 248.09382317, 248.10796599, ..., 248.03324901,
         248.0463194 , 248.06300552],
        [247.88311768, 247.88311768, 247.88311768, ..., 247.88311768,
         247.88311768, 247.88311768]],

       ...,

       [[218.3809967 , 218.3809967 , 218.3809967 , ..., 218.3809967 ,
         218.3809967 , 218.3809967 ],
        [222.51105005, 222.37146927, 222.23335266, ..., 222.92622521,
         222.80920679, 222.65934352],
        [223.39134329, 223.18564669, 222.98016442, ..., 224.08781177,
         223.9088083 , 223.64966105],
        ...,
        [263.3827055 , 263.41147103, 263.44018716, ..., 263.32895833,
         263.32190459, 263.35243633],
        [263.36784083, 263.3719884 , 263.37622917, ..., 263.35296469,
         263.34722803, 263.35749049],
        [263.11798096, 263.11798096, 263.11798096, ..., 263.11798096,
         263.11798096, 263.11798096]],

       [[224.6340332 , 224.6340332 , 224.6340332 , ..., 224.6340332 ,
         224.6340332 , 224.6340332 ],
        [228.74383178, 228.61669429, 228.49101809, ..., 229.09874974,
         228.99826944, 228.87027344],
        [229.74313793, 229.55913786, 229.37601502, ..., 230.33489247,
         230.19045123, 229.96608959],
        ...,
        [256.26714909, 256.28421392, 256.30294055, ..., 256.20159163,
         256.20182277, 256.23381276],
        [255.77698867, 255.77362267, 255.77060791, ..., 255.79799041,
         255.7896479 , 255.78314092],
        [254.81634521, 254.81634521, 254.81634521, ..., 254.81634521,
         254.81634521, 254.81634521]],

       [[233.46508789, 233.46508789, 233.46508789, ..., 233.46508789,
         233.46508789, 233.46508789],
        [235.90397092, 235.80538782, 235.7076634 , ..., 236.19998278,
         236.11504753, 236.00904252],
        [236.57557625, 236.44044317, 236.30624061, ..., 237.00251856,
         236.88045565, 236.72743849],
        ...,
        [243.62023857, 243.68235199, 243.74373306, ..., 243.47283281,
         243.50055213, 243.56089229],
        [243.91050955, 243.93941386, 243.96810263, ..., 243.8179779 ,
         243.85034625, 243.88054648],
        [244.52688599, 244.52688599, 244.52688599, ..., 244.52688599,
         244.52688599, 244.52688599]]])
```

In [41]:

```
print(type(f.array))
f.array[-1, 3, -2]
```

```
<class 'numpy.ndarray'>
```

Out[41]:

```
237.56118774414062
```

In [42]:

```
g = f.subspace[-1, 3, -2]
print(g)
```

```
Field: air_temperature (ncvar%tas)
----------------------------------
Data            : air_temperature(time(1), latitude(1), longitude(1)) K
Cell methods    : time(1): mean (interval: 30 minutes)
Dimension coords: time(1) = [1969-11-16 00:00:00] 365_day
                : latitude(1) = [-86.25] degrees_north
                : longitude(1) = [356.25] degrees_east
                : height(1) = [2.0] m
```

In [43]:

```
g.array
```

Out[43]:

```
array([[[237.56118774]]])
```

In [44]:

```
x = f.copy()
x[0, 0, 0] = -999
x[0, 0, :10].array
```

Out[44]:

```
array([[[-999.        ,  244.82579041,  244.82579041,  244.82579041,
          244.82579041,  244.82579041,  244.82579041,  244.82579041,
          244.82579041,  244.82579041]]])
```

In [45]:

```
x.subspace[0, 0, :3] = 888
x.subspace[0, 0, :10].array
```

Out[45]:

```
array([[[888.        , 888.        , 888.        , 244.82579041,
         244.82579041, 244.82579041, 244.82579041, 244.82579041,
         244.82579041, 244.82579041]]])
```

In [46]:

```
import numpy
y = -numpy.arange(145*192).reshape(145, 192)
print('Field shape:', x.shape)
print('Array shape:', y.shape)
```

```
Field shape: (120, 145, 192)
Array shape: (145, 192)
```

In [47]:

```
x[0, ...] = y
print(x[0, ...].array)
```

```
[[[ 0.0000e+00 -1.0000e+00 -2.0000e+00 ... -1.8900e+02 -1.9000e+02
   -1.9100e+02]
  [-1.9200e+02 -1.9300e+02 -1.9400e+02 ... -3.8100e+02 -3.8200e+02
   -3.8300e+02]
  [-3.8400e+02 -3.8500e+02 -3.8600e+02 ... -5.7300e+02 -5.7400e+02
   -5.7500e+02]
  ...
  [-2.7264e+04 -2.7265e+04 -2.7266e+04 ... -2.7453e+04 -2.7454e+04
   -2.7455e+04]
  [-2.7456e+04 -2.7457e+04 -2.7458e+04 ... -2.7645e+04 -2.7646e+04
   -2.7647e+04]
  [-2.7648e+04 -2.7649e+04 -2.7650e+04 ... -2.7837e+04 -2.7838e+04
   -2.7839e+04]]]
```

In [48]:

```
print(x[1].array)
```

```
[[[246.98564148 246.98564148 246.98564148 ... 246.98564148 246.98564148
   246.98564148]
  [248.46694996 248.35942057 248.25239525 ... 248.76876914 248.68722049
   248.57679331]
  [248.94832661 248.81420465 248.68104777 ... 249.34295834 249.23124955
   249.08926564]
  ...
  [244.75140282 244.79450904 244.83979468 ... 244.6257257  244.63474002
   244.69223537]
  [244.26617971 244.26601925 244.26605184 ... 244.25610468 244.26110323
   244.26354541]
  [243.73991394 243.73991394 243.73991394 ... 243.73991394 243.73991394
   243.73991394]]]
```

Modify the data where a condition is met

https://ncas-cms.github.io/cf-python/tutorial.html#assignment-by-condition (https://ncas-cms.github.io/cf-python/tutorial.html#assignment-by-condition)

In [49]:

```
print(f)
```

```
Field: air_temperature (ncvar%tas)
----------------------------------
Data            : air_temperature(time(120), latitude(145), longitude(192)) K
Cell methods    : time(120): mean (interval: 30 minutes)
Dimension coords: time(120) = [1959-12-16 12:00:00, ..., 1969-11-16 00:00:00] 365_day
                : latitude(145) = [-90.0, ..., 90.0] degrees_north
                : longitude(192) = [0.0, ..., 358.125] degrees_east
                : height(1) = [2.0] m
```

In [50]:

```
f.data.stats()
```

Out[50]:

```
{'minimum': <CF Data(): 203.62451171875 K>,
 'mean': <CF Data(): 276.5847382914912 K>,
 'median': <CF Data(): 280.7393942529291 K>,
 'maximum': <CF Data(): 311.89597497768546 K>,
 'range': <CF Data(): 108.27146325893546 K>,
 'mid_range': <CF Data(): 257.7602433482177 K>,
 'standard_deviation': <CF Data(): 20.816570165513593 K>,
 'root_mean_square': <CF Data(): 277.3669898333767 K>,
 'sample_size': 3340800}
```

Set values below 290 to missing data

In [51]:

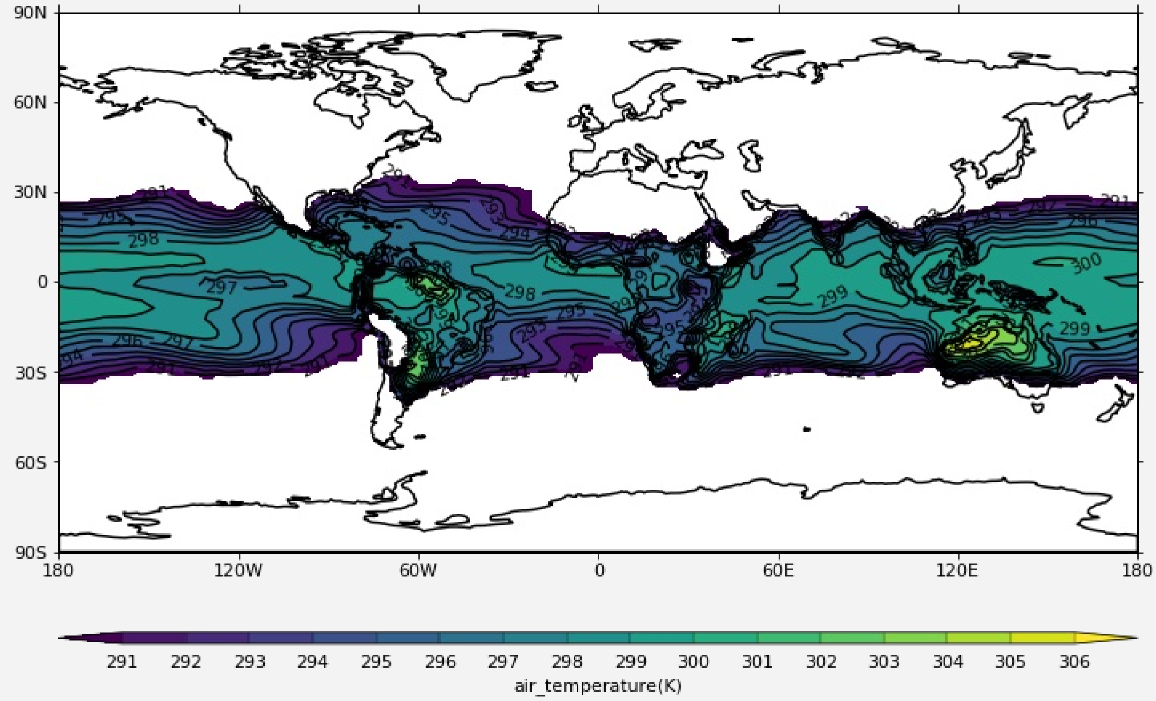
```
x = f.where(cf.lt(290), cf.masked)
x.data.stats()
```

Out[51]:

```
{'minimum': <CF Data(): 290.00001682247773 K>,
 'mean': <CF Data(): 296.502288030716 K>,
 'median': <CF Data(): 297.0859381465523 K>,
 'maximum': <CF Data(): 311.89597497768546 K>,
 'range': <CF Data(): 21.895958155207722 K>,
 'mid_range': <CF Data(): 300.9479959000816 K>,
 'standard_deviation': <CF Data(): 3.0873025594916057 K>,
 'root_mean_square': <CF Data(): 296.51836072078834 K>,
 'sample_size': 1139992}
```

In [52]:

```
cfp.con(x.subspace[0])
```



Manipulate the axes

In [53]:

```
f.transpose(['X', 'T', 'Y'])
```

Out[53]:

```
<CF Field: air_temperature(longitude(192), time(120), latitude(145)) K>
```

Modifying the units

In [54]:

```
f = cf.read('ncas_data/IPSL-CM5A-LR_r1i1p1_tas_n96_rcp45_mnth.nc')[0]
f.units, f.mean()
```

Out[54]:

```
('K', <CF Data(): 276.5847382914912 K>)
```

In [55]:

```
f.units = 'degC'
f.units, f.mean()
```

Out[55]:

```
('degC', <CF Data(): 3.434738291491425 degC>)
```

In [56]:

```
f.Units # Upper case "U" gives a units object that we can manipulate
```

Out[56]:

```
<Units: degC>
```

In [57]:

```
f.Units += 273.15
f.Units, f.units, f.mean()
```

Out[57]:

```
(<Units: K>, 'K', <CF Data(): 276.5847382914912 K>)
```

Field arithmetic

In [58]:

```
f
```

Out[58]:

```
<CF Field: air_temperature(time(120), latitude(145), longitude(192)) K>
```

In [59]:

```
f.data.stats()
```

Out[59]:

```
{'minimum': <CF Data(): 203.62451171875 K>,
 'mean': <CF Data(): 276.5847382914912 K>,
 'median': <CF Data(): 280.7393942529291 K>,
 'maximum': <CF Data(): 311.89597497768546 K>,
 'range': <CF Data(): 108.27146325893546 K>,
 'mid_range': <CF Data(): 257.7602433482177 K>,
 'standard_deviation': <CF Data(): 20.816570165513593 K>,
 'root_mean_square': <CF Data(): 277.3669898333767 K>,
 'sample_size': 3340800}
```

In [60]:

```
g = f + 2
g
```

Out[60]:

```
<CF Field: air_temperature(time(120), latitude(145), longitude(192)) K>
```

In [61]:

```
g.data.stats() #min(), g.mean(), g.max()
```

Out[61]:

```
{'minimum': <CF Data(): 205.62451171875 K>,
 'mean': <CF Data(): 278.5847382914912 K>,
 'median': <CF Data(): 282.7393942529291 K>,
 'maximum': <CF Data(): 313.89597497768546 K>,
 'range': <CF Data(): 108.27146325893546 K>,
 'mid_range': <CF Data(): 259.7602433482177 K>,
 'standard_deviation': <CF Data(): 20.816570165513593 K>,
 'root_mean_square': <CF Data(): 279.3613896056407 K>,
 'sample_size': 3340800}
```

In [62]:

```
g = f - f
g
```

Out[62]:

```
<CF Field: air_temperature(time(120), latitude(145), longitude(192)) K>
```

In [63]:

```
g.data.stats()
```

Out[63]:

```
{'minimum': <CF Data(): 0.0 K>,
 'mean': <CF Data(): 0.0 K>,
 'median': <CF Data(): 0.0 K>,
 'maximum': <CF Data(): 0.0 K>,
 'range': <CF Data(): 0.0 K>,
 'mid_range': <CF Data(): 0.0 K>,
 'standard_deviation': <CF Data(): 0.0 K>,
 'root_mean_square': <CF Data(): 0.0 K>,
 'sample_size': 3340800}
```

In [64]:

```
x = f.copy()
x.units = 'degC'
x.data
```

Out[64]:

```
<CF Data(120, 145, 192): [[[-28.32420959472654, ..., -28.623114013671852]]] degC>
```

Subtract the celcius field from the Kelvin field and check that the result is zero

In [65]:

```
(f - x).data.stats()
```

Out[65]:

```
{'minimum': <CF Data(): 0.0 K>,
 'mean': <CF Data(): 0.0 K>,
 'median': <CF Data(): 0.0 K>,
 'maximum': <CF Data(): 0.0 K>,
 'range': <CF Data(): 0.0 K>,
 'mid_range': <CF Data(): 0.0 K>,
 'standard_deviation': <CF Data(): 0.0 K>,
 'root_mean_square': <CF Data(): 0.0 K>,
 'sample_size': 3340800}
```

In [66]:

```
g = f * x
g
```

Out[66]:

```
<CF Field: ncvar%tas(time(120), latitude(145), longitude(192)) K2>
```

Find the anomalies relaitive to the first time (broadcasting)

In [67]:

```
first_time = f.subspace[0]
first_time = first_time.transpose(['Y', 'T', 'X'])
first_time
```

Out[67]:

```
<CF Field: air_temperature(latitude(145), time(1), longitude(192)) K>
```

In [68]:

```
anomaly  = f - first_time
anomaly
```

Out[68]:

```
<CF Field: air_temperature(time(120), latitude(145), longitude(192)) K>
```

In [69]:

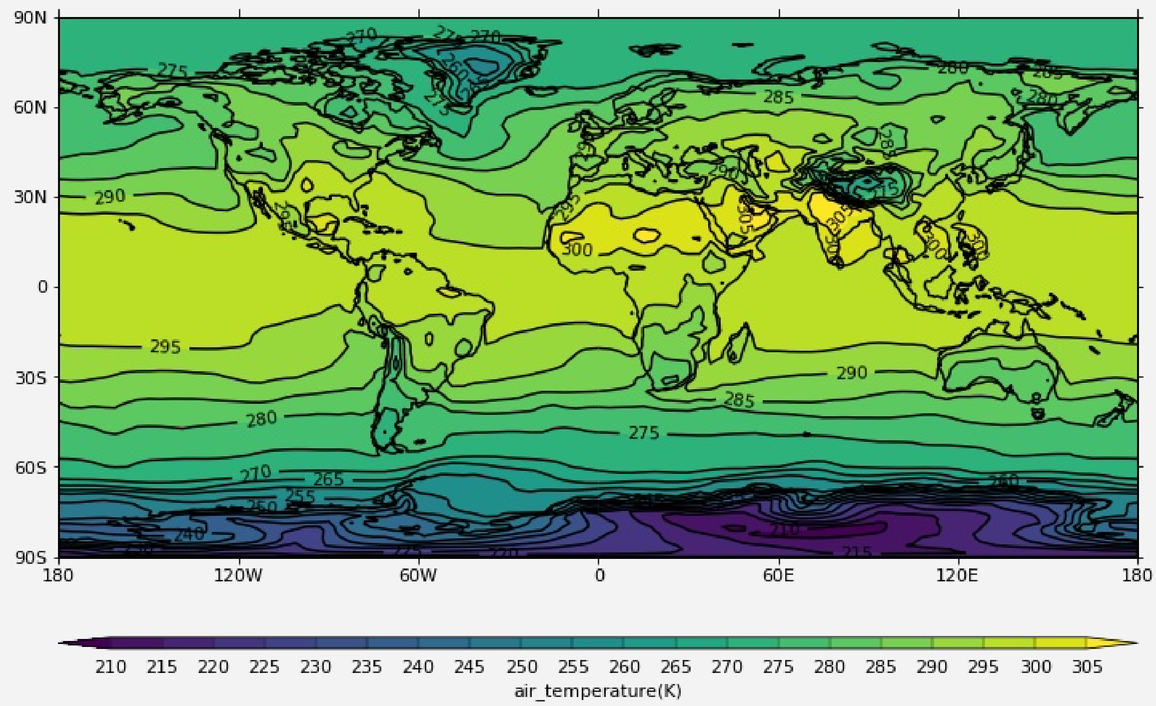
```
anomaly.data.stats()
```

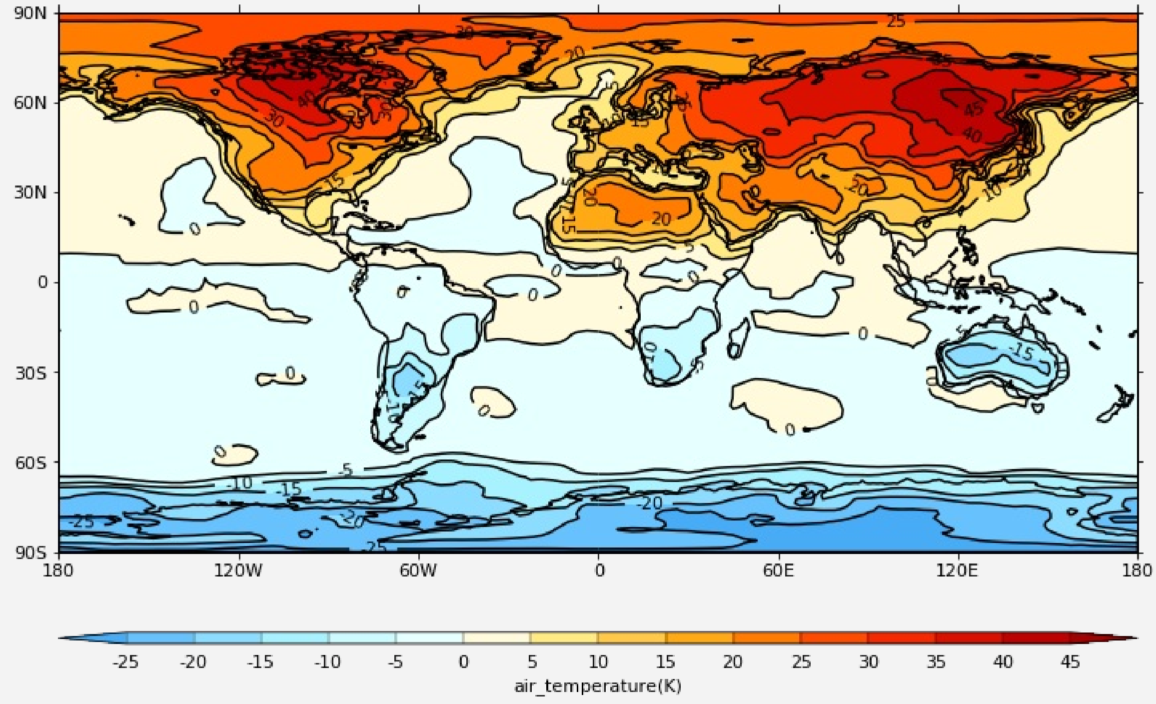
Out[69]:

```
{'minimum': <CF Data(): -32.62007141113281 K>,
 'mean': <CF Data(): 1.441687174432139 K>,
 'median': <CF Data(): 0.0 K>,
 'maximum': <CF Data(): 53.50559997558594 K>,
 'range': <CF Data(): 86.12567138671875 K>,
 'mid_range': <CF Data(): 10.442764282226562 K>,
 'standard_deviation': <CF Data(): 10.874943957481905 K>,
 'root_mean_square': <CF Data(): 10.970089698233751 K>,
 'sample_size': 3340800}
```

In [70]:

```
cfp.con(f.subspace(T=cf.contains(cf.dt('1962-06-04'))))
cfp.con(anomaly.subspace(T=cf.contains(cf.dt('1962-06-04'))))
```





4. Statistical operations

[https://ncas-cms.github.io/cf-python/analysis.html#statistical-collapses (https://ncas-cms.github.io/cf-python/analysis.html#statistical-collapses)](https://ncas-cms.github.io/cf-python/analysis.html#statistical-collapses)

In [71]:

```
g = f.collapse('max')
g
```

Out[71]:

```
<CF Field: air_temperature(time(1), latitude(1), longitude(1)) K>
```

In [72]:

```
g.data
```

Out[72]:

```
<CF Data(1, 1, 1): [[[311.89597497768546]]] K>
```

In [73]:

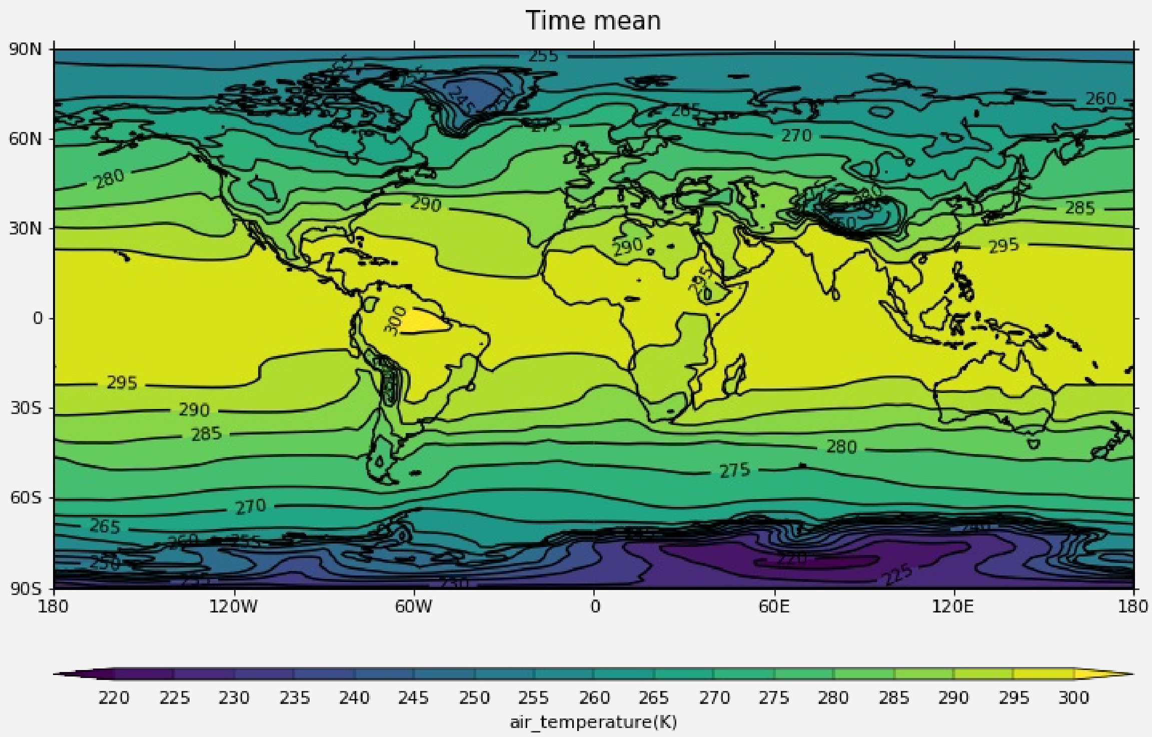
```
g = f.collapse('T: mean')
print(g)
print('data values:\n', g.data, '\n')
print('time bounds:\n', g.coord('T').bounds.dtarray)
```

```
Field: air_temperature (ncvar%tas)
----------------------------------
Data            : air_temperature(time(1), latitude(145), longitude(192)) K
Cell methods    : time(1): mean (interval: 30 minutes)
Dimension coords: time(1) = [1964-12-01 00:00:00] 365_day
                : latitude(145) = [-90.0, ..., 90.0] degrees_north
                : longitude(192) = [0.0, ..., 358.125] degrees_east
                : height(1) = [2.0] m
data values:
 [[[227.6330727895101, ..., 254.5096071879069]]] K

time bounds:
 [[cftime.DatetimeNoLeap(1959-12-01 00:00:00)
  cftime.DatetimeNoLeap(1969-12-01 00:00:00)]]
```

In [74]:

```python
cfp.con(g, title='Time mean')
```



Collapse multiple axes simultaneously

In [75]:

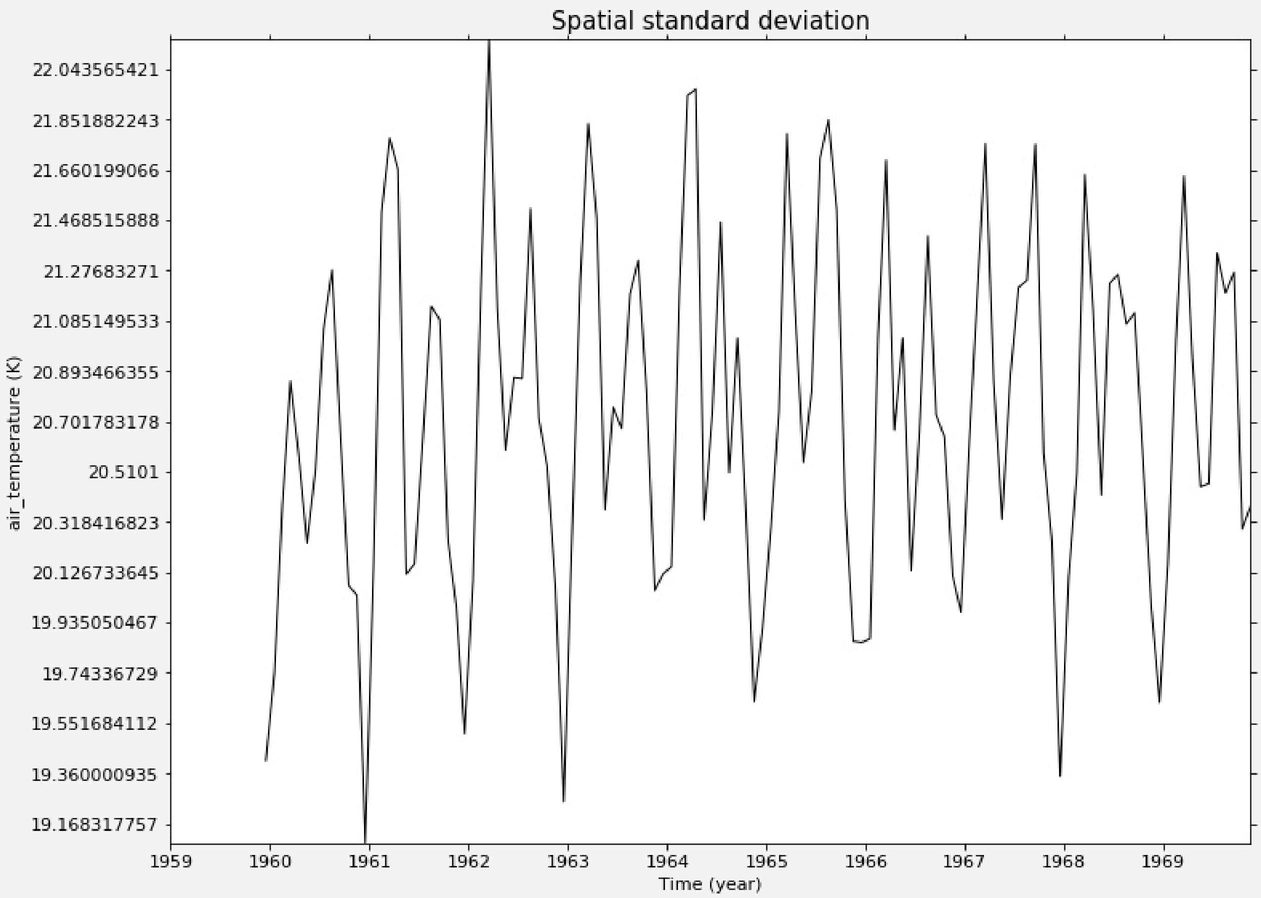
```python
g = f.collapse('X: Y: sd')
g
```

Out[75]:

```
<CF Field: air_temperature(time(120), latitude(1), longitude(1)) K>
```

In [76]:

```python
cfp.lineplot(g, title='Spatial standard deviation')
```



Collapse an axis into groups, rather than a single value

In [77]:

```python
g = f.collapse('T: mean', group=cf.seasons())
print(g)
```

```
Field: air_temperature (ncvar%tas)
----------------------------------
Data            : air_temperature(time(40), latitude(145), longitude(192)) K
Cell methods    : time(40): mean (interval: 30 minutes) time(40): mean
Dimension coords: time(40) = [1960-01-15 00:00:00, ..., 1969-10-16 12:00:00] 365_day
                : latitude(145) = [-90.0, ..., 90.0] degrees_north
                : longitude(192) = [0.0, ..., 358.125] degrees_east
                : height(1) = [2.0] m
```

`cf.seasons()` returns a (customizable) list of queries, each of which defines a range of months

In [78]:

```python
cf.seasons()
```

Out[78]:

```
[<CF Query: month[(ge 12) | (le 2)]>,
 <CF Query: month(wi (3, 5))>,
 <CF Query: month(wi (6, 8))>,
 <CF Query: month(wi (9, 11))>]
```

By default, collapses are not weighted

In [ ]:

In [79]:

```
g = f.collapse('area: mean', weights='area')  # Area mean for each time
g = g.collapse('T: max')                      # Time maxiumum of the area means
g.data
print(g)
```

```
Field: air_temperature (ncvar%tas)
----------------------------------
Data            : air_temperature(time(1), latitude(1), longitude(1)) K
Cell methods    : time(1): mean (interval: 30 minutes) area: mean time(1): maximum
Dimension coords: time(1) = [1964-12-01 00:00:00] 365_day
                : latitude(1) = [0.0] degrees_north
                : longitude(1) = [179.0625] degrees_east
                : height(1) = [2.0] m
```

File aggregation

Create a sequence of files on disk, each of which contains one year

In [80]:

```
f = cf.read('ncas_data/IPSL-CM5A-LR_r1i1p1_tas_n96_rcp45_mnth.nc')[0]
print(f)
for i in range(10):
    g = f.subspace[12*i:12*(i+1)]
    year = g.coord('T').year.array[0]
    new_file = 'air_temperature_'+str(year)+'.nc'
    cf.write(g, new_file)
    print('    creating new file:',new_file)
```

```
Field: air_temperature (ncvar%tas)
----------------------------------
Data            : air_temperature(time(120), latitude(145), longitude(192)) K
Cell methods    : time(120): mean (interval: 30 minutes)
Dimension coords: time(120) = [1959-12-16 12:00:00, ..., 1969-11-16 00:00:00] 365_day
                : latitude(145) = [-90.0, ..., 90.0] degrees_north
                : longitude(192) = [0.0, ..., 358.125] degrees_east
                : height(1) = [2.0] m
    creating new file: air_temperature_1959.nc
    creating new file: air_temperature_1960.nc
    creating new file: air_temperature_1961.nc
    creating new file: air_temperature_1962.nc
    creating new file: air_temperature_1963.nc
    creating new file: air_temperature_1964.nc
    creating new file: air_temperature_1965.nc
    creating new file: air_temperature_1966.nc
    creating new file: air_temperature_1967.nc
    creating new file: air_temperature_1968.nc
```

In ipython `!` preceeeds a shell command

In [81]:

```
!ls -o air_temperature_*.nc
```

```
-rw-r--r-- 1 swsheaps 2709551 Feb 14 10:24 air_temperature_1959.nc
-rw-r--r-- 1 swsheaps 2709551 Feb 14 10:24 air_temperature_1960.nc
-rw-r--r-- 1 swsheaps 2709551 Feb 14 10:24 air_temperature_1961.nc
-rw-r--r-- 1 swsheaps 2709551 Feb 14 10:24 air_temperature_1962.nc
-rw-r--r-- 1 swsheaps 2709551 Feb 14 10:24 air_temperature_1963.nc
-rw-r--r-- 1 swsheaps 2709551 Feb 14 10:24 air_temperature_1964.nc
-rw-r--r-- 1 swsheaps 2709551 Feb 14 10:24 air_temperature_1965.nc
-rw-r--r-- 1 swsheaps 2709551 Feb 14 10:24 air_temperature_1966.nc
-rw-r--r-- 1 swsheaps 2709551 Feb 14 10:24 air_temperature_1967.nc
-rw-r--r-- 1 swsheaps 2709551 Feb 14 10:24 air_temperature_1968.nc
```

In [82]:

```
f2 = cf.read('air_temperature_*.nc')
print(f2)
```

```
[<CF Field: air_temperature(time(120), latitude(145), longitude(192)) K>]
```

In [83]:

```
f.equals(f2[0])
```

Out[83]:

```
True
```

In [84]:

```
f3 = cf.read('air_temperature_*.nc', aggregate=False)
f3
```

Out[84]:

```
[<CF Field: air_temperature(time(12), latitude(145), longitude(192)) K>,
 <CF Field: air_temperature(time(12), latitude(145), longitude(192)) K>,
 <CF Field: air_temperature(time(12), latitude(145), longitude(192)) K>,
 <CF Field: air_temperature(time(12), latitude(145), longitude(192)) K>,
 <CF Field: air_temperature(time(12), latitude(145), longitude(192)) K>,
 <CF Field: air_temperature(time(12), latitude(145), longitude(192)) K>,
 <CF Field: air_temperature(time(12), latitude(145), longitude(192)) K>,
 <CF Field: air_temperature(time(12), latitude(145), longitude(192)) K>,
 <CF Field: air_temperature(time(12), latitude(145), longitude(192)) K>,
 <CF Field: air_temperature(time(12), latitude(145), longitude(192)) K>]
```

5. PP and UM fields files

In [85]:

```
x = cf.read('ncas_data/aaaaoa.pmh8dec.pp')
x
```

Out[85]:

```
[<CF Field: relative_humidity(grid_latitude(30), grid_longitude(24)) %>,
 <CF Field: id%UM_m01s08i233_vn405(grid_latitude(30), grid_longitude(24)) kg m-2 s-1>,
 <CF Field: relative_humidity(air_pressure(17), grid_latitude(30), grid_longitude(24)) %>]
```

In [86]:

```
print(x[1])
```

```
Field: id%UM_m01s08i233_vn405 (ncvar%UM_m01s08i233_vn405)
---------------------------------------------------------
Data            : id%UM_m01s08i233_vn405(grid_latitude(30), grid_longitude(24)) kg m-2 s-1
Cell methods    : time(1): mean
Dimension coords: time(1) = [1978-12-16 12:00:00] gregorian
                : grid_latitude(30) = [7.480000078678131, ..., -5.279999852180481] degrees
                : grid_longitude(24) = [-5.720003664493561, ..., 4.399996280670166] degrees
Auxiliary coords: latitude(grid_latitude(30), grid_longitude(24)) = [[61.004354306111864, ...,
48.51422609871432]] degrees_north
                : longitude(grid_latitude(30), grid_longitude(24)) = [[-13.762685427418687, ...
, 4.622216504491947]] degrees_east
Coord references: grid_mapping_name:rotated_latitude_longitude
```

In [87]:

```
cf.write(x, 'aaaaoa.pmh8dec.nc')
y = cf.read('aaaaoa.pmh8dec.nc')
y
```

Out[87]:

```
[<CF Field: relative_humidity(grid_latitude(30), grid_longitude(24)) %>,
 <CF Field: long_name=CANOPY THROUGHFALL RATE      KG/M2/S(grid_latitude(30), grid_longitude(24
)) kg m-2 s-1>,
 <CF Field: relative_humidity(air_pressure(17), grid_latitude(30), grid_longitude(24)) %>]
```

6. What this course doesn't cover

Create new field constructs in memory

Incorporate, and create, metadata stored in external files

Read, write, and create data that have been compressed by convention (i.e. ragged or gathered arrays), whilst presenting a view of the data in its uncompressed form

Perform histogram, percentile and binning operations on field constructs

Apply convolution filters to field constructs

Calculate derivatives of field constructs

Create field constructs to create derived quantities (such as vorticity)

... however, regridding is covered later