Making maps for science in python

We are all working in a field science, and displying our data in geographic context is essential. No matter what the specific nature of your research, **you will need to make some maps!**

Python has some nice packages for mapmaking. The one we will learn today is Cartopy. If you poke around you will see that many people use a package called Basemap. This works well, but is not going to continue to be developed, and will be replaced by Cartopy.

What is a map?

Dumb question? We all know what maps are, but there are some important features of a map that make them different from the standard figures we make.

Maps are (almost always) a two-dimensional representation of our 3-dimensional word. We need to take data that is spread around on a roundish ball, and show it accurately on something flat, usually a computer screen or a piece of paper.

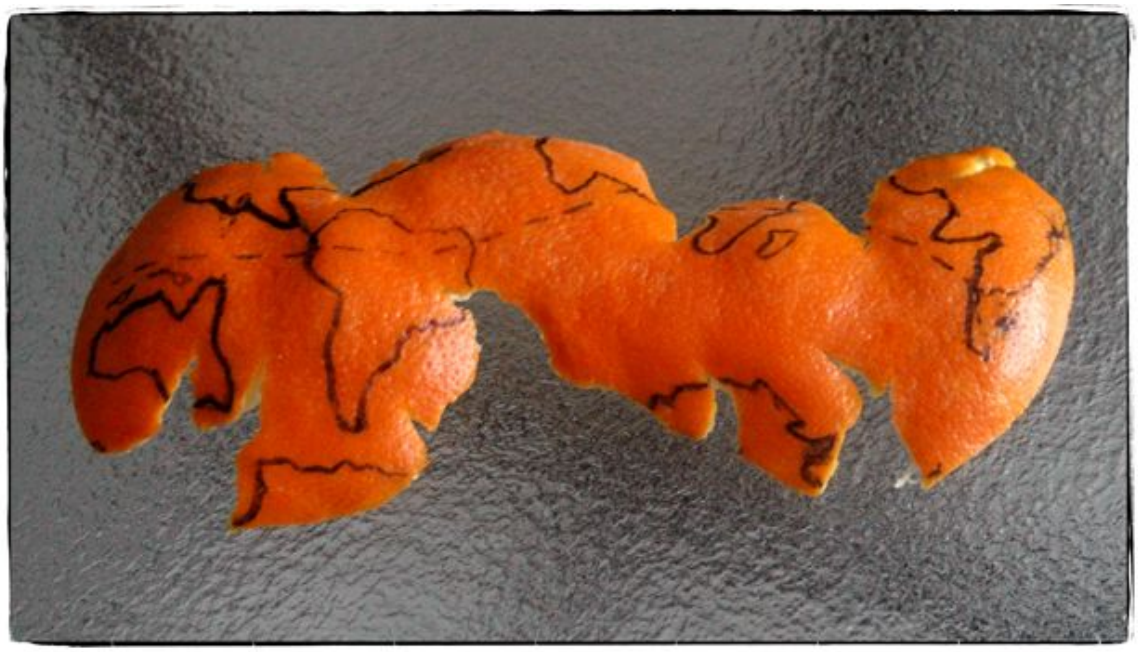
It turns out that representing a 3D object in 2D space is complicated. Making accurate maps that show what you want can be very complicated. We are going to go through some of the basics here and try to understand enough to make some standard maps, and understand enough to know what else we might need to know to dive deeper.

credit

As always, lots of this lesson is based on Ryan Abernathy's course: https://rabernat.github.io/research_computing_2018/maps-with-cartopy.html. Parts too are from the person who wrote the Cartopy package, Phil Elson , tutorial here: https://github.com/SciTools/cartopy-tutorial/tree/42cb77062a08063a53e7a511a9681bdb15e70fe7.

Map Projections: transforming from spherical to flat

- The surface of a sphere is fundementally different from a 2D surface, therefore we have to cut the sphere somewhere to 'make it flat' (see: orange)
- a map projection is the method we use to 'flatten' the sphere. There are lots of choices, but
- A sphere's surface cannot be represented on a plane without distortion.

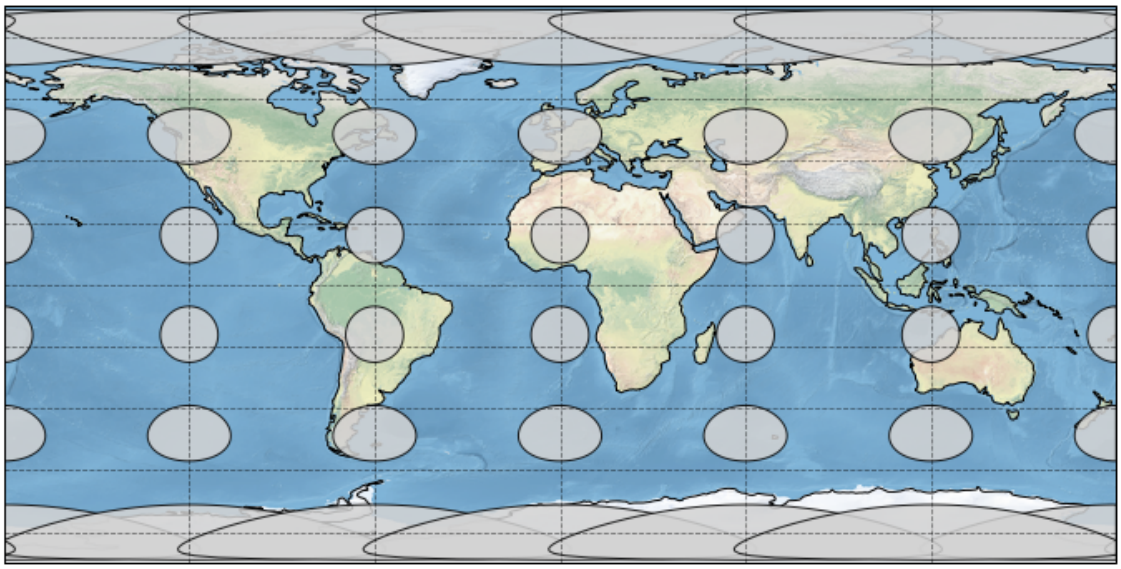


Common distortions of map projections

Metric properties of maps that are often not preserved:

- Area
- Shape
- Direction
- Distance
- Scale

It's important to remember that all maps have some sort of distortion! The figure below show a map where we have drawn equal-area circles at a bunch of diffent points on the earth.



A detailed discussion of the basic different types of map projections is included in Phil Elisons tutorial. It's a good resource if you ever find yourself with a headache thinking about map projections.

Cartopy

https://scitools.org.uk/cartopy/docs/latest/

Cartopy is a Python package designed for geospatial data processing in order to produce maps and other geospatial data analyses.

We will install Cartopy from the terminal they same way we did for xarray:

```
$ conda install -c conda-forge cartopy
```

Cartopy makes use of the powerful PROJ.4, NumPy and Shapely libraries and includes a programmatic interface built on top of Matplotlib for the creation of publication quality maps.

Cartopy Projection sytems

The key ingredient of making a map is defining the projection, ie the instructions to 'flatten' the world.

Cartopy is going to give us the projection, so we need to load the library that contains it.

We need cartopy's crs module. This is typically imported as ccrs (Cartopy Coordinate Reference Systems).

In [1]:
```
import cartopy.crs as ccrs
import cartopy
import matplotlib.pyplot as plt

%matplotlib inline
```

Drawing a map

Like many other things in Python, Cartopy is built to work well with another library, namely Matplotlib for plotting.

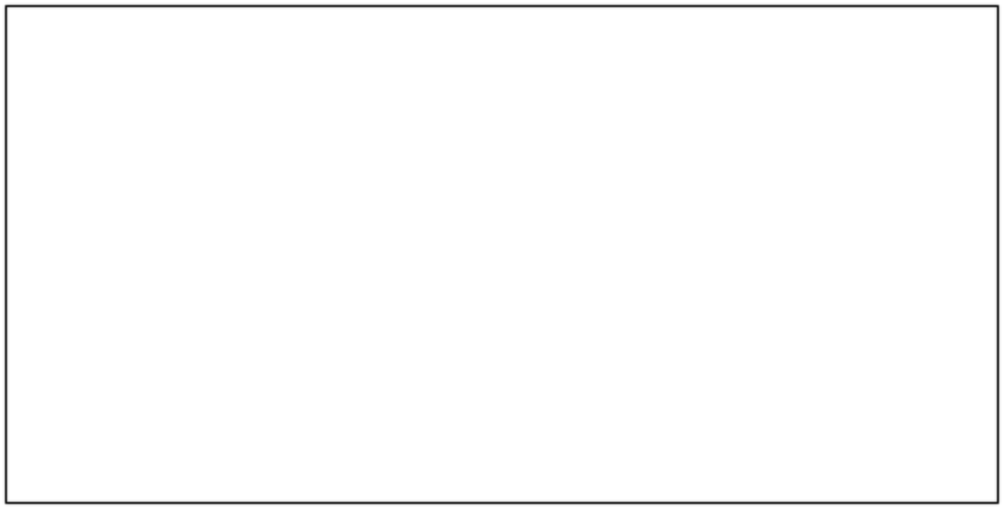
Let's check this out. I'm going to create a figure with matplotlib, then I'm going to tell the axes that we want to use a cartopy projection. In this case I'm going to use the Plate Carree projection (also known as Equirectangular projection)

The basic steps here are the same as making any matplotlib figure, but the special

sauce is that we need to tell matplotlib that we want the figure axes to have a projection, or instructions to represent the sphere in 2D. To do that we use the `projection=ccrs.PlateCarree()` option for argument in `plt.axes()`

In [2]:
```
plt.figure()
ax = plt.axes(projection=ccrs.PlateCarree())
ax

# compare with
# ax = plt.axes()
# ax
```

Out[2]:
```
<GeoAxes: >
```



ok, that didn't do very much that was interesting, but it did create a figure. And if you look the text that got spit out we see that the axes of figure is one of those GeoAxes[Subplot] instances - this is telling us that rather than a regular matplotlib figure, this is a map that knows something about the projection.

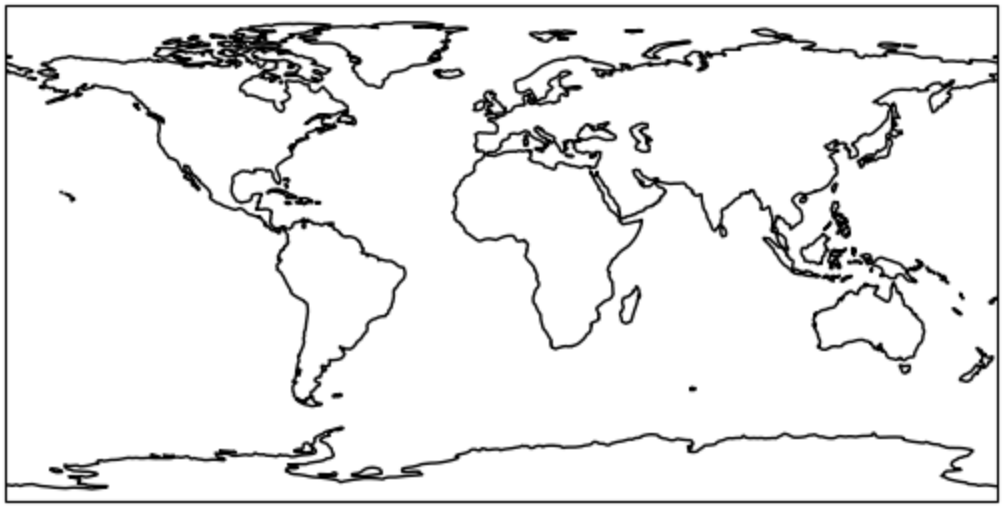
Now this figure axes (`ax`) is a Cartopy thing (object).

And because of that `ax` has all the features of the Cartopy library. Now we can start doing some things to make the map look like a real map.

A simple cartopy feature is called `coastlines` which, you guessed it, adds coastlines to whatever map axes we made:

In [3]:
```
plt.figure()
ax = plt.axes(projection=ccrs.PlateCarree())
ax.coastlines()
```

Out[3]:
```
<cartopy.mpl.feature_artist.FeatureArtist at 0x111182110>
```



Projection classes have options we can use to customize the map. We can read about them by googling the cartopy docs, or by running the projection with a `?` or SHIFT+TAB

In [4]:
```
ccrs.PlateCarree?
```

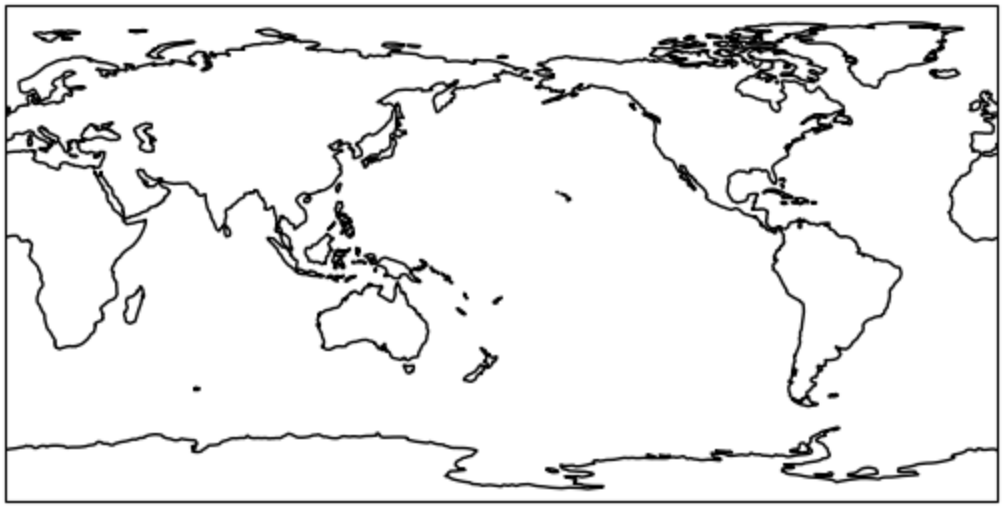
```
Init signature: ccrs.PlateCarree(central_longitude=0.0, globe=None)
Docstring:
The abstract class which denotes cylindrical projections where we
want to allow x values to wrap around.
Init docstring:
Parameters
----------
proj4_params: iterable of key-value pairs
    The proj4 parameters required to define the
    desired CRS.  The parameters should not describe
    the desired elliptic model, instead create an
    appropriate Globe instance. The ``proj4_params``
    parameters will override any parameters that the
    Globe defines.
globe: :class:`~cartopy.crs.Globe` instance, optional
    If omitted, the default Globe instance will be created.
    See :class:`~cartopy.crs.Globe` for details.
File:           ~/miniconda3/envs/mac_swbc2023/lib/python3.11/site-packages/
cartopy/crs.py
Type:           ABCMeta
Subclasses:
```

Looks like we can change the central longitude of the map. and add coastlines:

```
plt.figure()
ax = plt.axes(projection=ccrs.PlateCarree(____=180))
ax.____()
```

In [5]:
```
plt.figure()
ax = plt.axes(projection=ccrs.PlateCarree(central_longitude=180))
ax.coastlines()
```

Out[5]:
```
<cartopy.mpl.feature_artist.FeatureArtist at 0x110dea190>
```



Useful methods of a GeoAxes

The cartopy.mpl.geoaxes.GeoAxes class adds a number of useful methods.

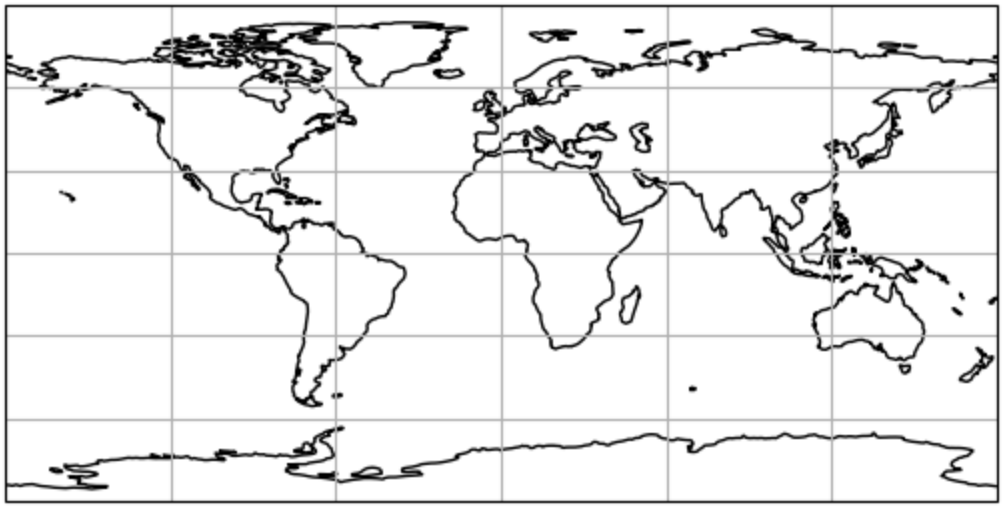
Let's take a look at:

- `set_global` - zoom the map out as much as possible
- `set_extent` - zoom the map to the given bounding box
- `gridlines` - add a graticule (and optionally labels) to the axes
- `coastlines` - add Natural Earth coastlines to the axes
- `stock_img` - add a low-resolution Natural Earth background image to the axes
- `imshow` - add an image (numpy array) to the axes
- `add_geometries` - add a collection of geometries (Shapely) to the axes

Let's try using `ax.gridlines()` here:

In [6]:
```
plt.figure()
ax = plt.axes(projection=ccrs.PlateCarree())
ax.coastlines()
ax.gridlines()
```

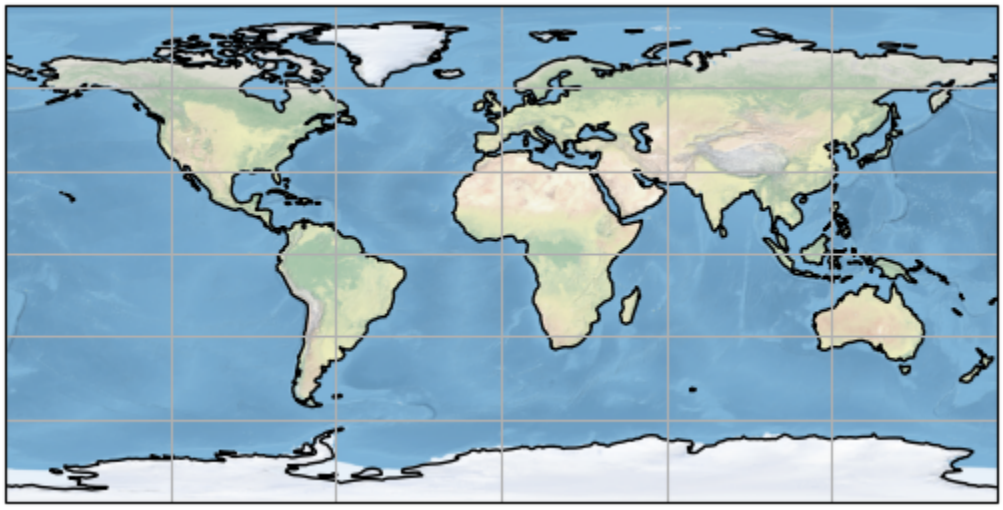
Out[6]:
```
<cartopy.mpl.gridliner.Gridliner at 0x1112ae710>
```



What about `ax.stock_img()`?

In [7]:
```
plt.figure()
ax = plt.axes(projection=ccrs.PlateCarree())
ax.coastlines()
ax.gridlines()
ax.stock_img()
```

Out[7]:
```
<matplotlib.image.AxesImage at 0x111079bd0>
```



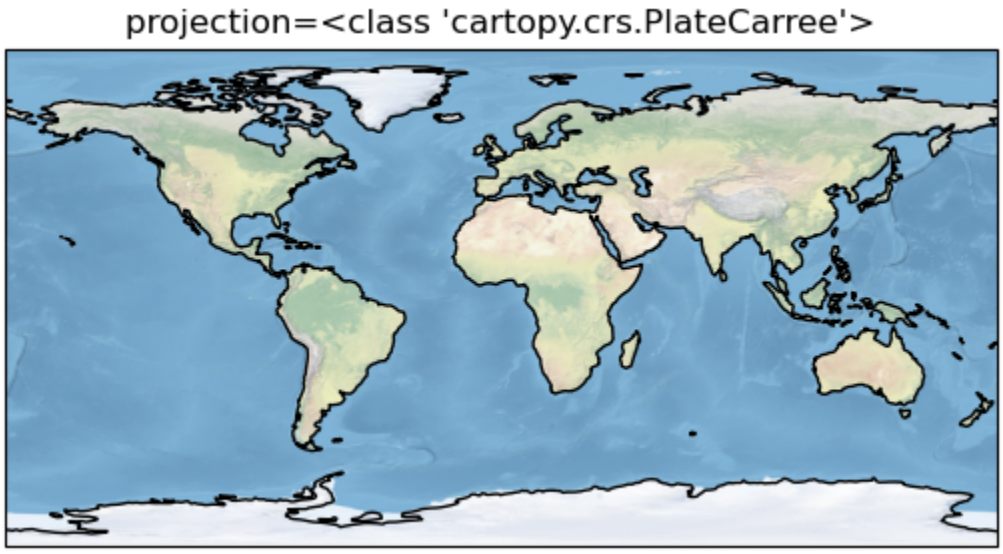
More examples of Cartopy projections

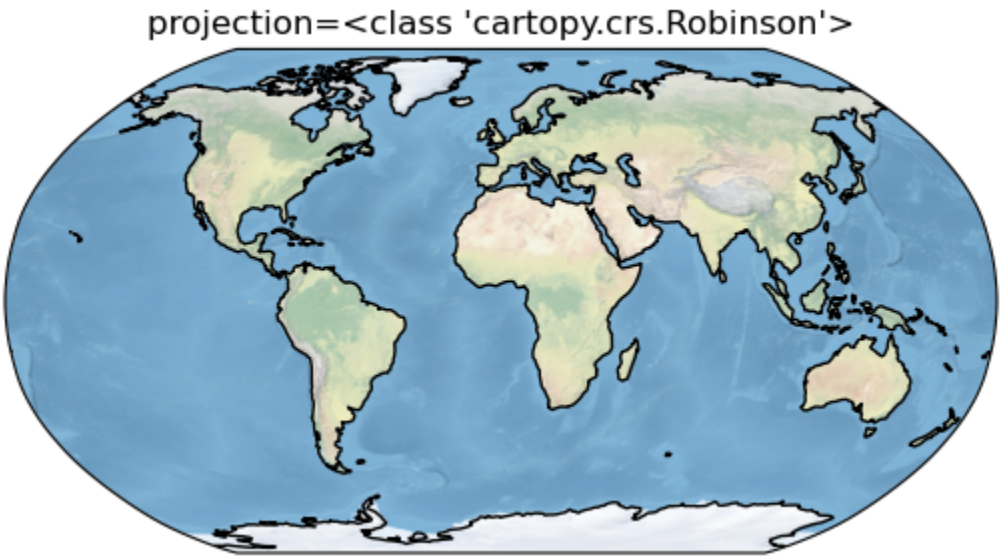
let's loop through a few projections and look at different maps. We will make a list of projections, then fill in the blanks below to loop through it and create figures:

```
for proj in ____:
    plt.figure()
    ax = plt.axes(projection=___)
    ax.stock_img()
    ax.coastlines()
```

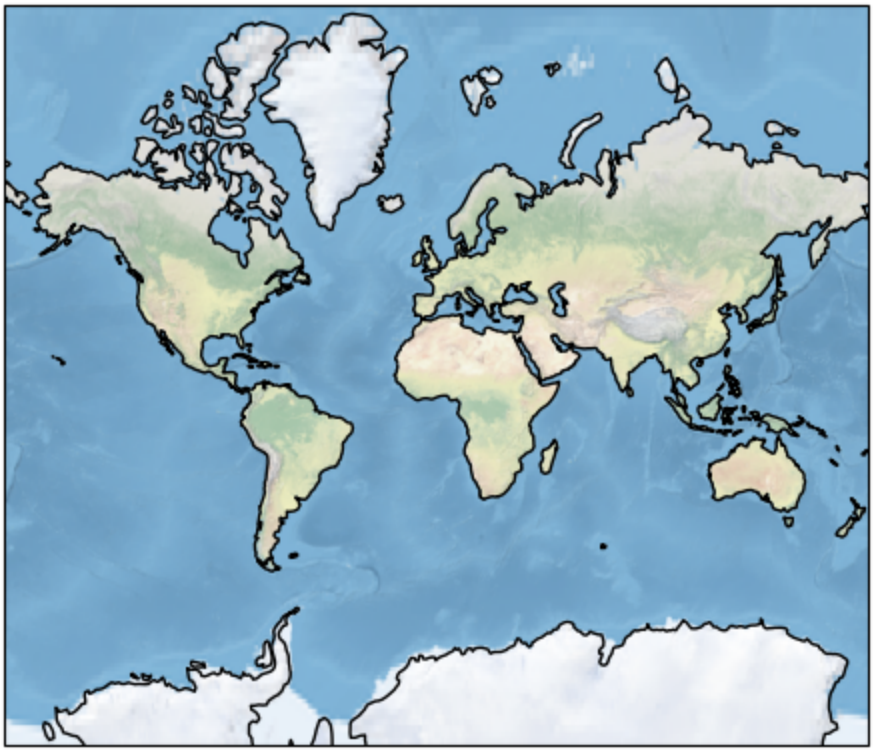
```
    ax.set_title(f'projection={type(proj)}')
```

In [8]:
```
projections5 = [ccrs.PlateCarree(),
                ccrs.Robinson(),
                ccrs.Mercator(),
                ccrs.Orthographic(),
                ccrs.InterruptedGoodeHomolosine()
               ]

for proj in projections5:
    plt.figure()
    ax = plt.axes(projection=proj)
    ax.stock_img()
    ax.coastlines()
    ax.set_title(f'projection={type(proj)}')
```

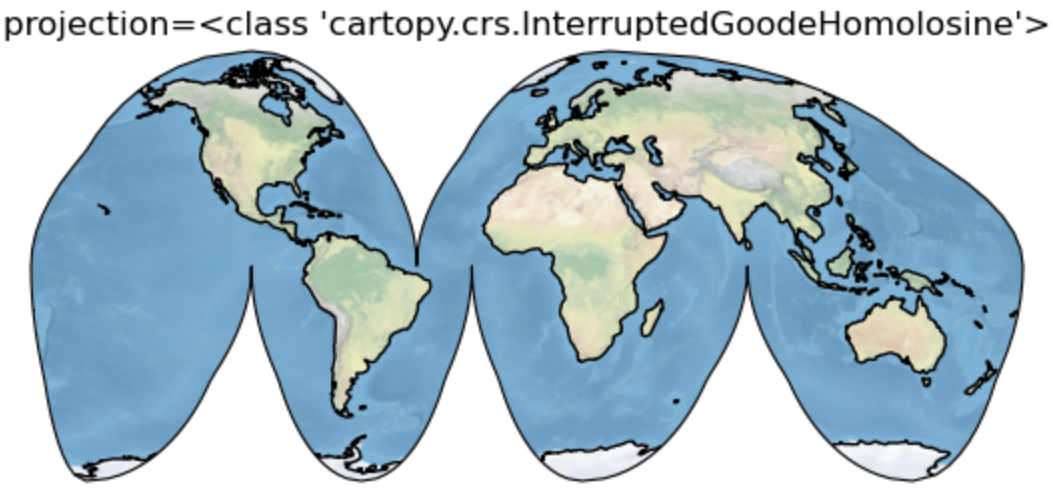




projection=<class 'cartopy.crs.Mercator'>



projection=<class 'cartopy.crs.Orthographic'>



In [ ]:

Customizing maps

We've seen how to create simple global maps, and taken a look at some of the different projections Cartopy has to offer. But we might want to customize these maps for our purposes. Let's start looking at ways to customize.

credit

As always, lots of this lesson is based on Ryan Abernathy's course: https://rabernat.github.io/research_computing_2018/maps-with-cartopy.html. Parts too are from the person who wrote the Cartopy package, Phil Elson , tutorial here: https://github.com/SciTools/cartopy-tutorial /tree/42cb77062a08063a53e7a511a9681bdb15e70fe7.

In [9]:
```
# import statements

import cartopy.crs as ccrs
# import cartopy

import matplotlib.pyplot as plt

%matplotlib inline
```

zooming in on a region

We are each probably interested in a particular part of the world.

To customize our map we will want to use the `set_extent` method/function of Cartopy to do this.

How does it work? first create a geoaxis

In [10]:
```
plt.figure()
ax = plt.axes(projection=ccrs.PlateCarree())
```

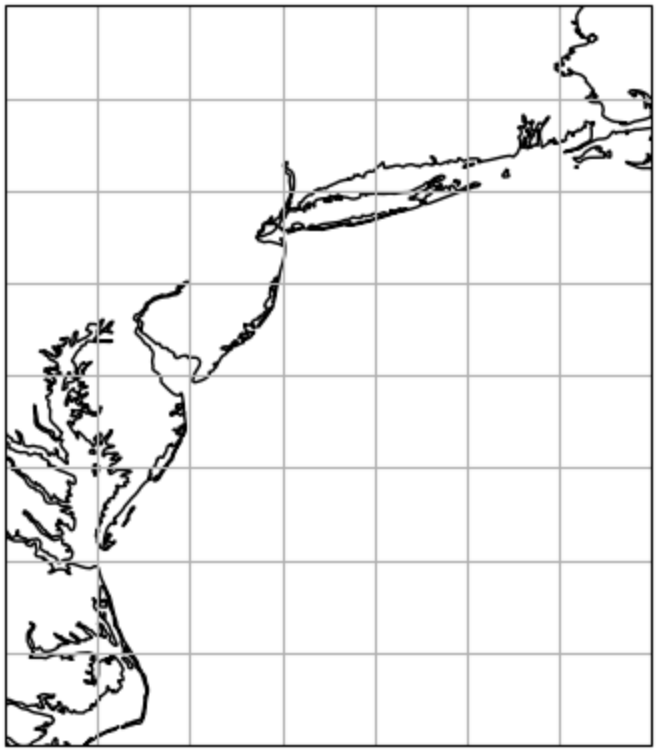
Use the `?` (or SHIFT+TAB) to figure out what `ax.set_extent()` does:

In [11]:
```
ax.set_extent?
```

```
Signature: ax.set_extent(extents, crs=None)
Docstring:
Set the extent (x0, x1, y0, y1) of the map in the given
coordinate system.

If no crs is given, the extents' coordinate system will be assumed
to be the Geodetic version of this axes' projection.

Parameters
----------
extents
    Tuple of floats representing the required extent (x0, x1, y0, y1).
File:      ~/miniconda3/envs/mac_swbc2023/lib/python3.11/site-packages/carto
py/mpl/geoaxes.py
Type:      method
```

Let's make a map of our area, use `extent = [-77, -70, 35, 43]` with `ax.set_extent()`

In [12]:
```
extent = [-77, -70, 35, 43]
ax = plt.axes(projection=ccrs.PlateCarree())
ax.gridlines()
ax.coastlines()
ax.set_extent(extent)
```



This defaults to an appropriate resolution, but you can check out the documentation in `coastlines` to see how to specify higher/lower resolution in the coastline

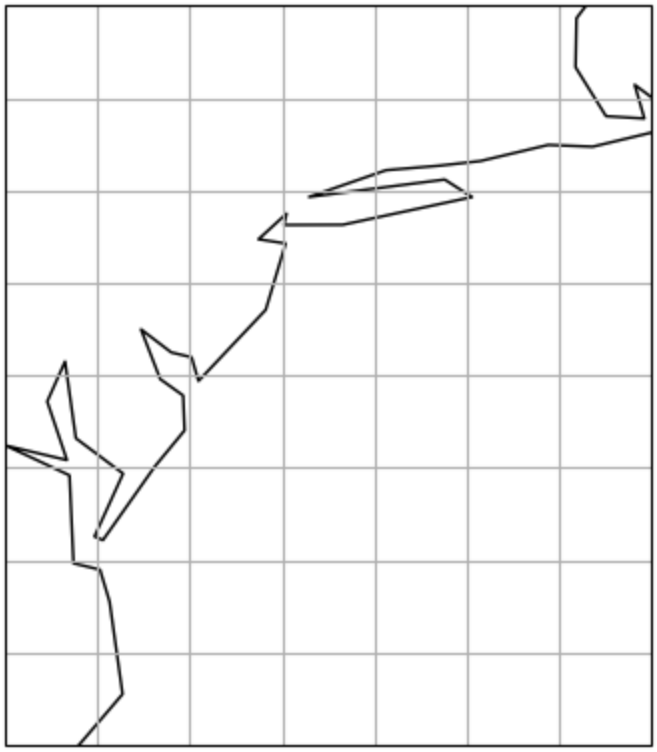
In [13]:
```
ax.coastlines?
```

```
Signature: ax.coastlines(resolution='auto', color='black', **kwargs)
Docstring:
Add coastal **outlines** to the current axes from the Natural Earth
"coastline" shapefile collection.

Parameters
----------
resolution : str or :class:`cartopy.feature.Scaler`, optional
    A named resolution to use from the Natural Earth
    dataset. Currently can be one of "auto" (default), "110m", "50m",
    and "10m", or a Scaler object.  If "auto" is selected, the
    resolution is defined by `~cartopy.feature.auto_scaler`.
File:      ~/miniconda3/envs/mac_swbc2023/lib/python3.11/site-packages/carto
py/mpl/geoaxes.py
Type:      method
```

remake the same plot but with the resolution set to 50m:

In [14]:
```
# Redo this, with low resolution
extent = [-77, -70, 35, 43]
ax = plt.axes(projection=ccrs.PlateCarree())
ax.set_extent(extent)
ax.gridlines()
ax.coastlines(resolution='110m')
```

Out[14]:
```
<cartopy.mpl.feature_artist.FeatureArtist at 0x112ebf6d0>
```



Adding Features to the Map

To give our map more styles and details, we add cartopy.feature objects. Many useful features are built in. These "default features" are at coarse (110m) resolution.

- cartopy.feature.BORDERS Country boundaries
- cartopy.feature.COASTLINE Coastline, including major islands
- cartopy.feature.LAKES Natural and artificial lakes
- cartopy.feature.LAND Land polygons, including major islands
- cartopy.feature.OCEAN Ocean polygons
- cartopy.feature.RIVERS Single-line drainages, including lake centerlines
- cartopy.feature.STATES (limited to the United States at this scale)

to do this, we need to import the `cartopy.feature` part of cartopy. We ususally do this as

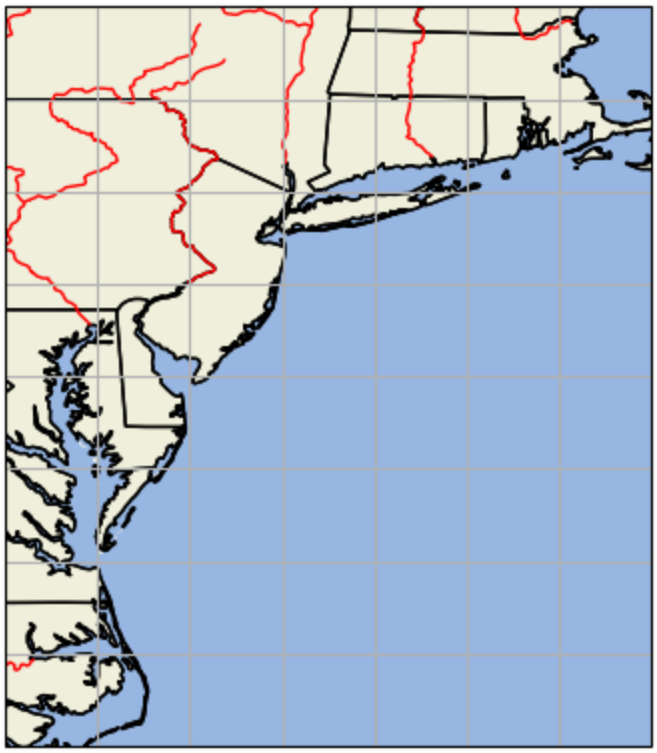
```
import cartopy.feature as cfeature
```

Lets make the same map again, but now use the `ax.add_feature()` function to add ocean, states , and land:

In [15]:
```
import cartopy.feature as cfeature

extent = [-77, -70, 35, 43]
ax = plt.axes(projection=ccrs.PlateCarree())
ax.set_extent(extent)
ax.gridlines()
```

```
ax.coastlines()

ax.add_feature(cfeature.OCEAN)
ax.add_feature(cfeature.LAND)
ax.add_feature(cfeature.STATES)
ax.add_feature(cfeature.RIVERS,edgecolor='red')
```

Out[15]: <cartopy.mpl.feature_artist.FeatureArtist at 0x11131e150>



Exercise 01

Make a map of Africa. Include the borders between countries and major rivers. Play with other features, resolution, background images, etc.

In [16]:
```
import cartopy.crs as ccrs
import cartopy.feature as cfeature

import matplotlib.pyplot as plt

%matplotlib inline
```

In [17]:
```
fig = plt.figure()
ax = fig.add_subplot(1, 1, 1, projection=ccrs.PlateCarree())
ax.set_extent([-20, 60, -40, 45], crs=ccrs.PlateCarree())

ax.add_feature(cfeature.LAND)
ax.add_feature(cfeature.OCEAN)
ax.add_feature(cfeature.COASTLINE)
ax.add_feature(cfeature.BORDERS, linestyle=':')
ax.add_feature(cfeature.LAKES, alpha=0.5)
```

```
ax.add_feature(cfeature.RIVERS)

plt.show()
```

Adding data to the map

Now we know how to make and customize basic maps. Great. But we want to use these maps to convey information or results, ie to display data.

So how do we add data to our maps?

credit

As always, lots of this lesson is based on Ryan Abernathy's course: https://rabernat.github.io/research_computing_2018/maps-with-cartopy.html. Parts too are from the person who wrote the Cartopy package, Phil Elson , tutorial here: https://github.com/SciTools/cartopy-tutorial/tree/42cb77062a08063a53e7a511a9681bdb15e70fe7.

In [18]:
```
# import statements

import cartopy.crs as ccrs
import cartopy

import matplotlib.pyplot as plt

%matplotlib inline
```

Cartopy plays well with matplotlib

Because our map *is* a matplotlib axis, we can use all the familiar maptplotlib plotting tools and commands to make plots. By default, the map extent will be adjusted to match the data. We can override this with the `.set_global` or `.set_extent` commands.

In [19]:
```
# create some test data
new_york_lon = -74.0060
new_york_lat = 40.7128
honolulu_lon = -157.8583
honolulu_lat = 21.3069

lons = [new_york_lon, honolulu_lon]
lats = [new_york_lat, honolulu_lat]
```

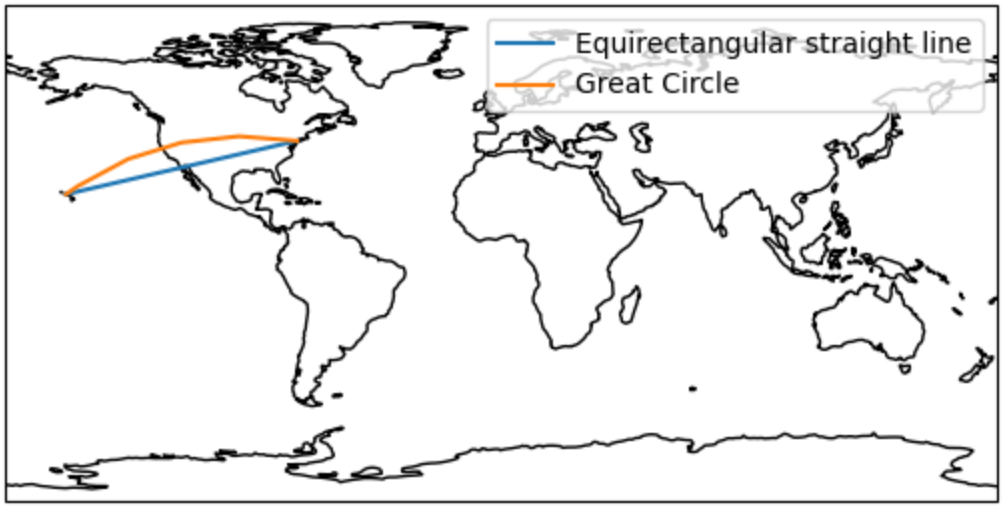
Key point:

The data *ALSO* have to be transformed to the projection space. This is done via the `transform=` keyword in the plotting method. The argument is another `cartopy.crs` object. If you don't specify a transform, Cartopy assume that the data is using the same projection as the underlying GeoAxis.

> ## From the Cartopy Documentation:
>
> > The core concept is that the projection of your axes is independent of the coordinate system your data is defined in. The `projection` argument is used when creating plots and determines the projection of the resulting plot (i.e. what the plot looks like). The `transform` argument to plotting functions tells Cartopy what coordinate system your data are defined in.

```
ax = plt.axes(___=ccrs.PlateCarree())
ax.plot(lons, lats, label='Equirectangular straight line')
ax.plot(lons, lats, label='Great Circle', ___=ccrs.Geodetic())
ax.coastlines()
ax.legend()
ax.set_global()
```

In [20]:
```
ax = plt.axes(projection=ccrs.PlateCarree())
ax.plot(lons, lats, label='Equirectangular straight line')
ax.plot(lons, lats, label='Great Circle', transform=ccrs.Geodetic())
ax.coastlines()
ax.legend()
ax.set_global()
```



deconstruct the plot above

First, we just used a normal matplotlib `.plot()` command to plot our lat an lon data. This is actually just the normal command, its nothing special from Cartopy. In the first case we didn't add the `transform=` argument, so the map assumed that the data are already in the map projection (what is the map projection here?).

In the second case we told the plotting function that the original data we want to plot (`lon` , `lat`) are in `ccrs.Geodetic()` projection, so it then transformed them into the defined map projection.

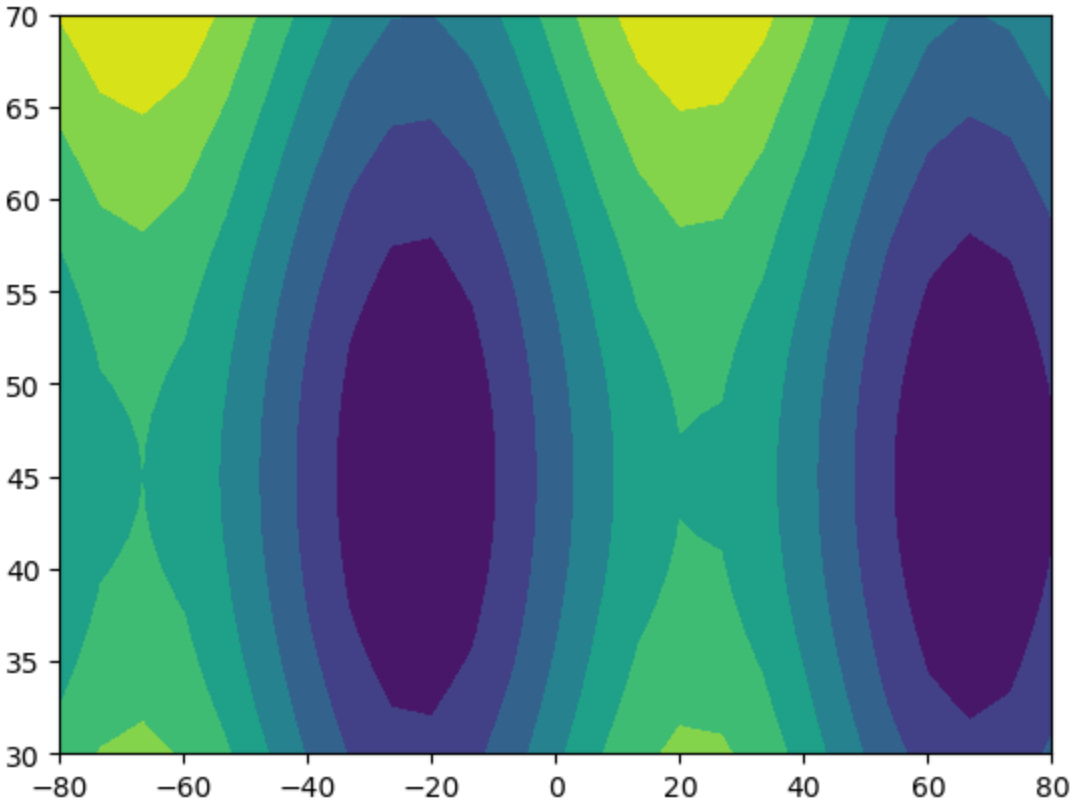
Plotting 2D (Raster) Data

The same principles apply to 2D data. Below we create some fake example data defined in regular lat / lon coordinates.

When we plot this there is no projection, just good old standard matplotlib.

In [21]:
```
import numpy as np

lon = np.linspace(-80, 80, 25)
lat = np.linspace(30, 70, 25)
lon2d, lat2d = np.meshgrid(lon, lat)

data = np.cos(np.deg2rad(lat2d) * 4) + np.sin(np.deg2rad(lon2d) * 4)

plt.contourf(lon2d, lat2d, data)
```

Out[21]:
```
<matplotlib.contour.QuadContourSet at 0x1809f2cd0>
```

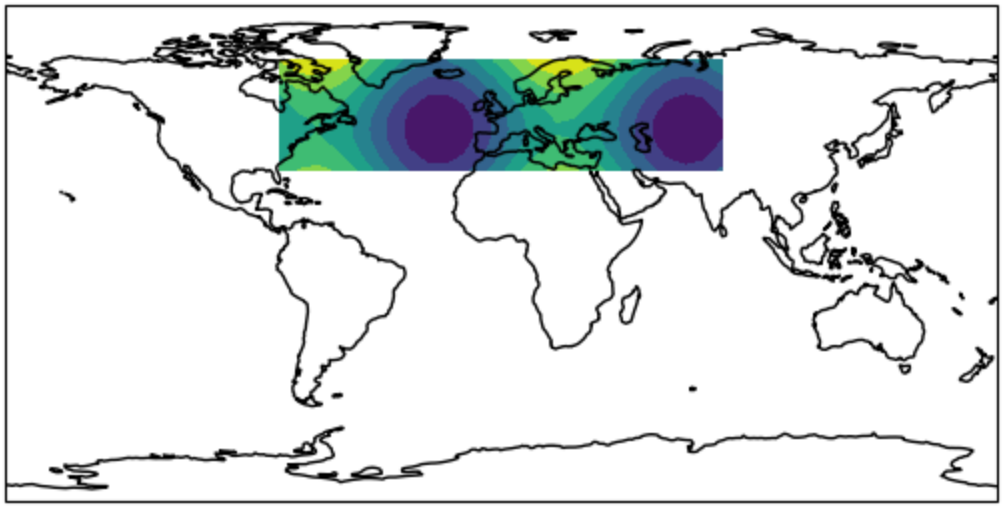


Now we create a `PlateCarree` projection and plot the data on it without any `transform` keyword. This happens to work because `PlateCarree` is the simplest projection of lat / lon data.

```
ax = plt.axes(___=ccrs.PlateCarree())
ax.set_global()
ax.coastlines()
ax.contourf(lon, lat, data)
```

In [22]:
```
ax = plt.axes(projection=ccrs.PlateCarree())
ax.set_global()
ax.contourf(lon, lat, data)
ax.coastlines()
```

Out[22]:
```
<cartopy.mpl.feature_artist.FeatureArtist at 0x112c68690>
```

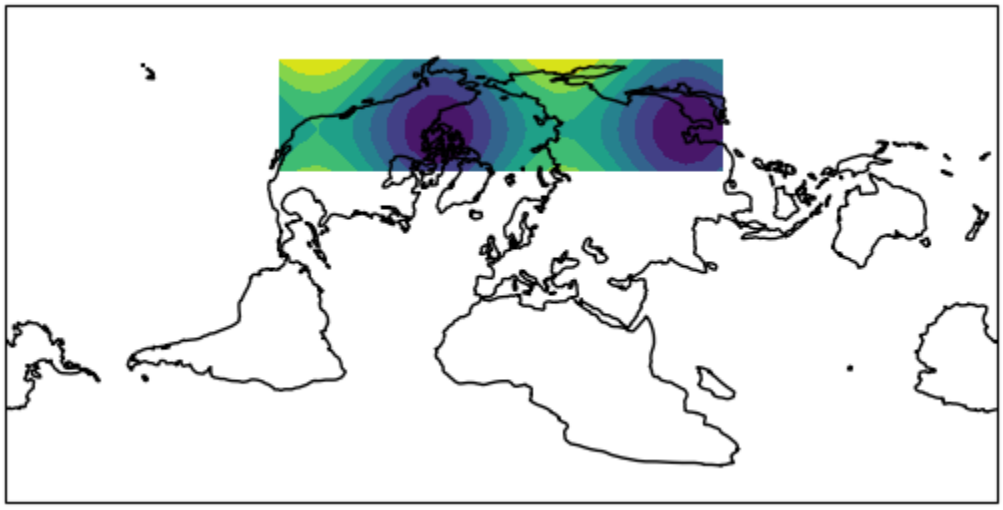


However, if we try the same thing with a different projection, we get the wrong result.

```
projct = ccrs.RotatedPole(pole_longitude=-177.5, pole_latitude=37.5)
ax = plt.axes(projection=projct)
ax.set_global()
ax.coastlines()
ax.contourf(lon, lat, data)
```

In [23]:
```
projct = ccrs.RotatedPole(pole_longitude=-177.5, pole_latitude=37.5)
ax = plt.axes(projection=projct)
ax.set_global()
ax.contourf(lon, lat, data)
ax.coastlines()
```

Out[23]:
```
<cartopy.mpl.feature_artist.FeatureArtist at 0x1809e2690>
```



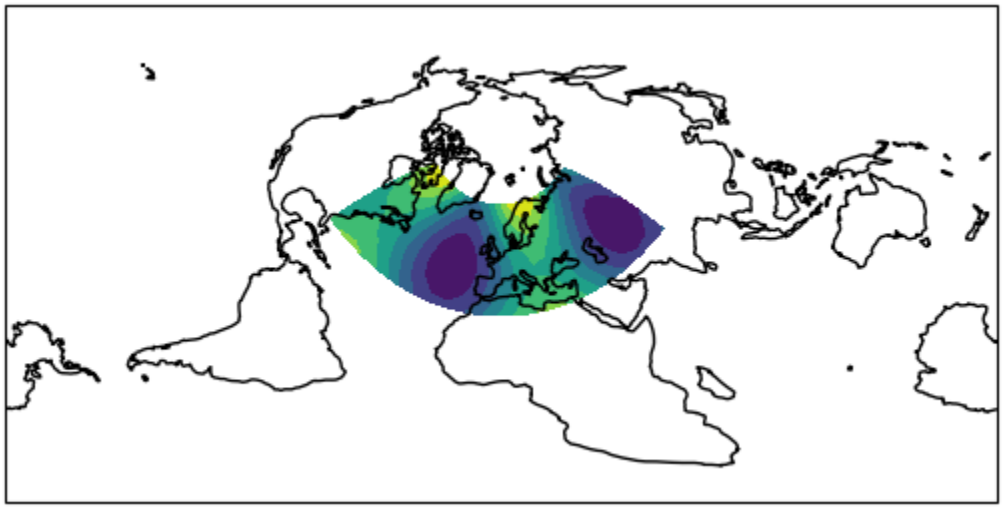
To make this work, we need to make sure to give the `transform=` argument in `contourf`. In particular we need to tell contourf that the data is in the standard `PlateCarree` projection. This is almost always going to be the original data projection (as far as I know, unless you know for sure otherwise). It's generally a good bet to pass the `transform=ccrs.PlateCarree()` in whatever matplotlib plotting function you

are using in a map.

```
projection = ccrs.RotatedPole(pole_longitude=-177.5,
pole_latitude=37.5)
ax = plt.axes(projection=projection)
ax.set_global()
ax.coastlines()
ax.contourf(lon, lat, data, ___=ccrs.PlateCarree())
```

In [24]:
```
projection = ccrs.RotatedPole(pole_longitude=-177.5, pole_latitude=37.5)

ax = plt.axes(projection=projection)
ax.set_global()
ax.contourf(lon, lat, data, transform=ccrs.PlateCarree())
ax.coastlines()
```

Out[24]:
```
<cartopy.mpl.feature_artist.FeatureArtist at 0x180a26bd0>
```



Showing images

We can plot a satellite image easily on a map if we know its lat/lon extent. There are lots more ways to plot satellite data we can dig into if you want

make a figure with the Satellite data

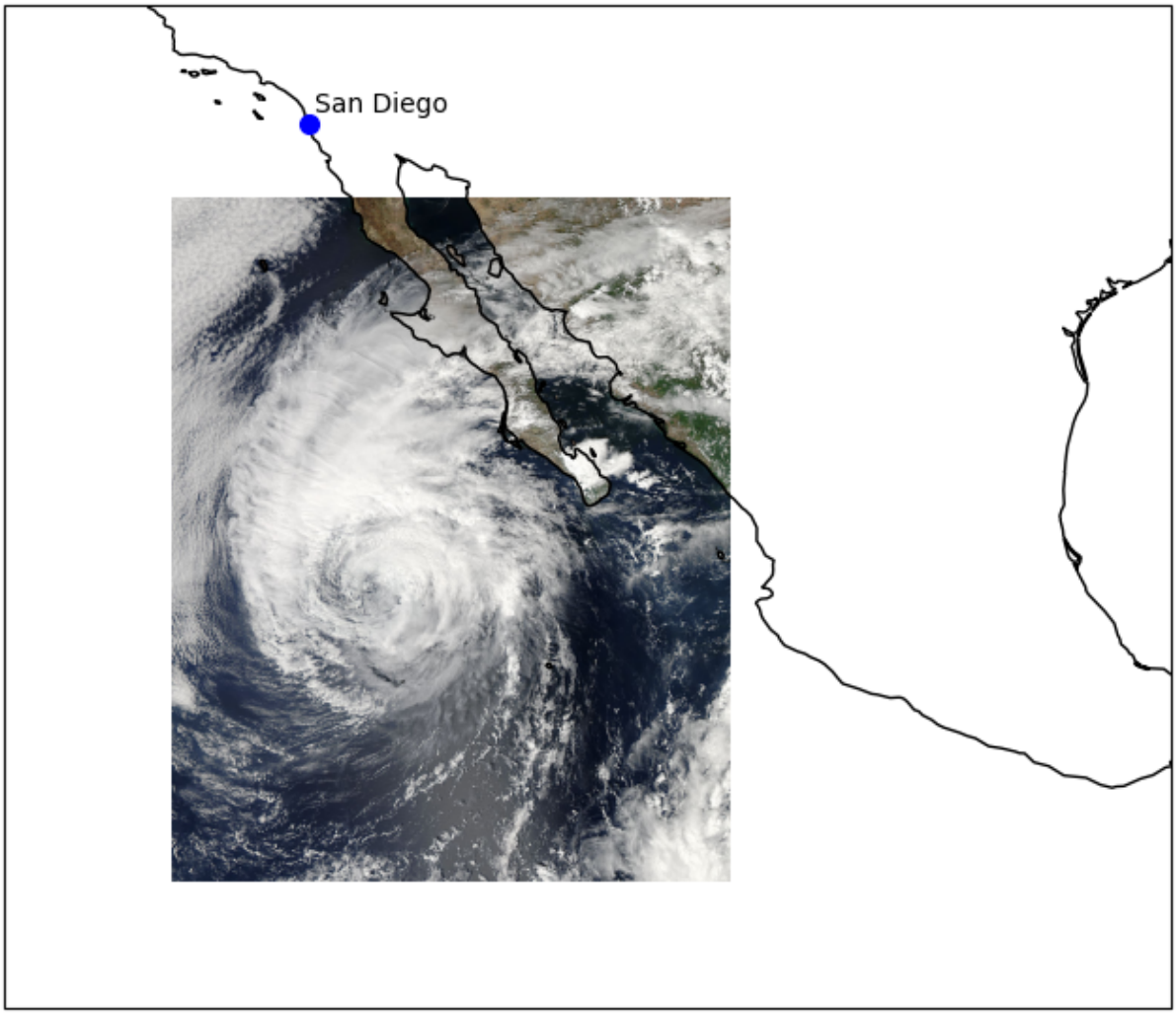
We are going to use a matplotlib function called `imshow()`. What do we need to add to it?

```
fig = plt.figure(figsize=(8, 10))

# load image & load coordinates of the corners
fname = '../data/Miriam.A2012270.2050.2km.jpg'
img_extent = (-120.67660000000001, -106.32104523100001,
```

```
13.2301484511245, 30.766899999999502)
img = plt.imread(fname)

# define projection for axes
_____

# add the image. Because this image was a tif, the "origin" of the
image is in the
# upper left corner
ax.imshow(____, origin='upper', extent=_____, transform=_______)
ax.coastlines(resolution='50m', color='black', linewidth=1)

# mark a known place to help us geo-locate ourselves
ax.plot(-117.1625, 32.715, 'bo', markersize=7)
ax.text(-117, 33, 'San Diego')

# set map extent to larger area
_____
```

In [25]:

```
fig = plt.figure(figsize=(8, 10))

# load image & load coordinates of the corners
fname = '../data/Miriam.A2012270.2050.2km.jpg'
img_extent = (-120.67660000000001, -106.32104523100001, 13.2301484511245, 30
img = plt.imread(fname)

# define projection for axes
ax = plt.axes(projection=ccrs.PlateCarree())

# add the image. Because this image was a tif, the "origin" of the image is
# upper left corner
ax.imshow(img, origin='upper', extent=img_extent, transform=ccrs.PlateCarree
ax.coastlines(resolution='50m', color='black', linewidth=1)

# mark a known place to help us geo-locate ourselves
ax.plot(-117.1625, 32.715, 'bo', markersize=7)
ax.text(-117, 33, 'San Diego')

# set map extent to larger area
ax.set_extent([-125, -95,10,35])
```



Xarray Integration

Cartopy transforms can be passed to xarray!

This creates a very quick path for creating professional looking maps from netCDF data.

Remember we've looked at this SST dataset before in the xarray class. We know xarray can make nice default plots, but now we can pass the `transform=` argument and get proper maps!

In [26]:
```
# import statements: xarray, cartopy.crs, cartopy, matplotlib
import xarray as xr

import cartopy.crs as ccrs

import matplotlib.pyplot as plt

%matplotlib inline
```

load in our SST dataset

```
url = 'http://www.esrl.noaa.gov/psd/thredds/dodsC/Datasets
/noaa.ersst.v5/sst.mnmean.nc'

ds = xr.___(url, _____)
```

In [27]:
```
url = 'http://www.esrl.noaa.gov/psd/thredds/dodsC/Datasets/noaa.ersst.v5/sst
ds = xr.open_dataset(url, drop_variables=['time_bnds'])

ds
```

Out[27]: xarray.Dataset

▸ Dimensions: (**lat**: 89, **lon**: 180, **time**: 2035)

▾ Coordinates:

lat	(lat)	float32	88.0 86.0 84.0 ... -86.0 ...
lon	(lon)	float32	0.0 2.0 4.0 ... 354.0 356...
time	(time)	datetime64[ns]	1854-01-01 ... 2023-07...

▾ Data variables:

sst	(time, lat, lon)	float32	...

▸ Indexes: (3)

▸ Attributes: (39)

make a map of SST

Pick a particular day and create a dataset for it:

In [28]:
```
sst = ds.sst.sel(time='2000-01-01', method='nearest')
```

then create a figure with Robinson projection, coastlines & gridlines:

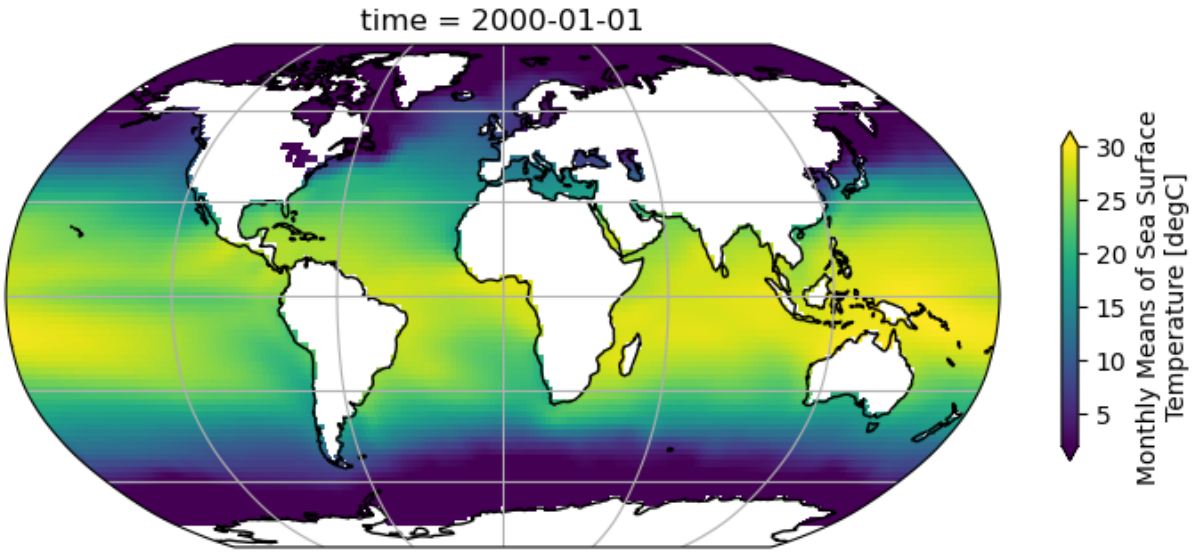
```
fig = plt.figure(figsize=(9,6))
ax = plt.axes(____=ccrs.Robinson())
ax.coastlines()
ax.gridlines()
```

and plot our xarray dataset using the built in xarray plotting, plus that new special thing we need to do for maps:

```
sst.___(ax=ax, ___=ccrs.PlateCarree(), vmin=2, vmax=30, cbar_kwargs=
{'shrink': 0.4})
```

In [29]:
```
fig = plt.figure(figsize=(9,6))
ax2 = plt.axes(projection=ccrs.Robinson())
ax2.gridlines()

# could do sst.plot(ax=ax2,...) but not needed since only one axis
sst.plot(transform=ccrs.PlateCarree(),vmin=2, vmax=30, cbar_kwargs={'shrink'
ax2.coastlines()
```

Out[29]:
```
<cartopy.mpl.feature_artist.FeatureArtist at 0x186e56cd0>
```



Exercise 02

1. Southern Ocean

Remake the map we just made but instead focus on the Southern Ocean.

Change the map projection to `ccrs.Orthographic(__,__)` and make plot so that the South Pole is in the center of the map.

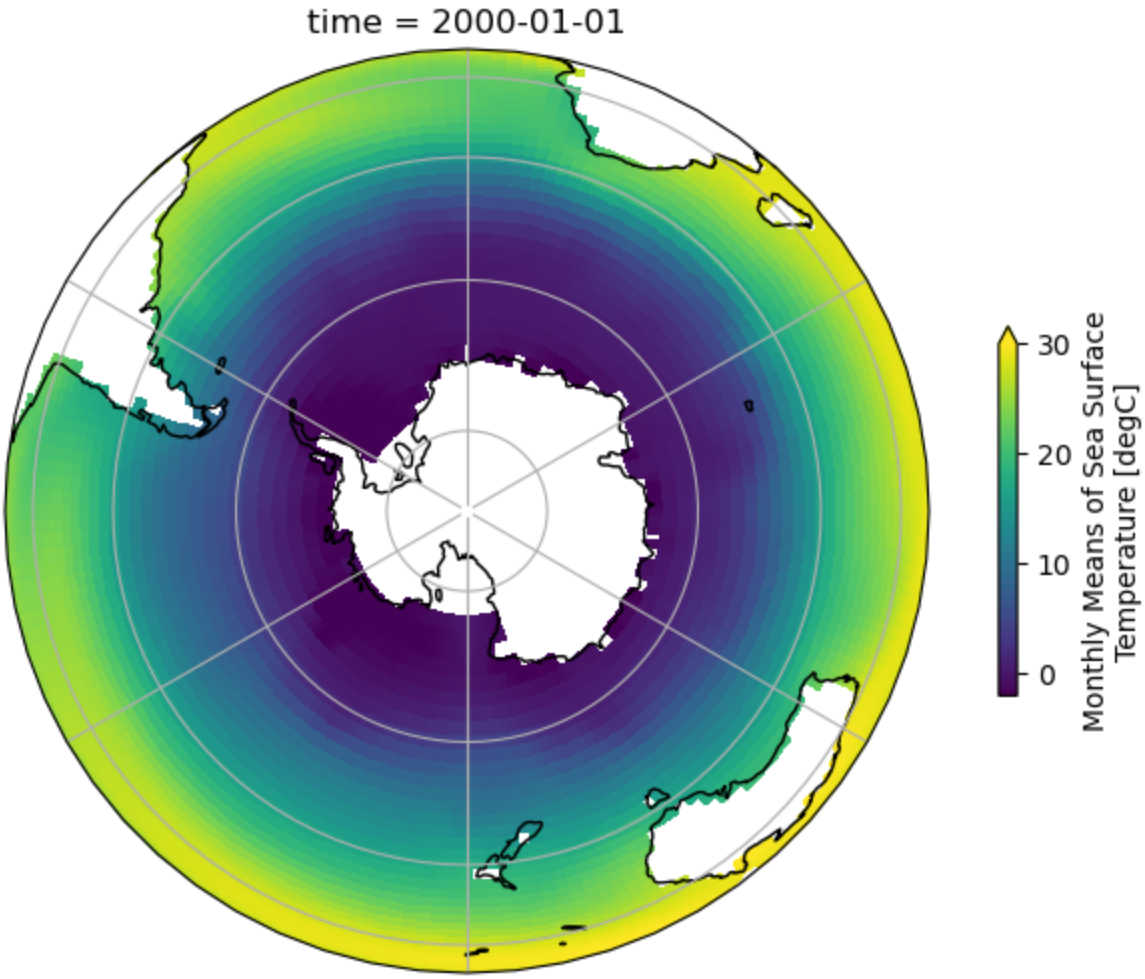
In [30]:
```
import xarray as xr
import cartopy.crs as ccrs
import matplotlib.pyplot as plt

%matplotlib inline
```

In [31]:
```
url = 'http://www.esrl.noaa.gov/psd/thredds/dodsC/Datasets/noaa.ersst.v5/sst
ds = xr.open_dataset(url, drop_variables=['time_bnds'])

sst = ds.sst.sel(time='2000-01-01', method='nearest')
```

In [32]:
```
fig = plt.figure(figsize=(9,6))
ax = plt.axes(projection=ccrs.Orthographic(0,-90))
ax.coastlines()
ax.gridlines()
```

```
sst.plot(ax=ax, transform=ccrs.PlateCarree(),
         vmin=-2, vmax=30, cbar_kwargs={'shrink': 0.4})
```

Out[32]: <cartopy.mpl.geocollection.GeoQuadMesh at 0x1877867d0>



2. Chlorophyll off MAB

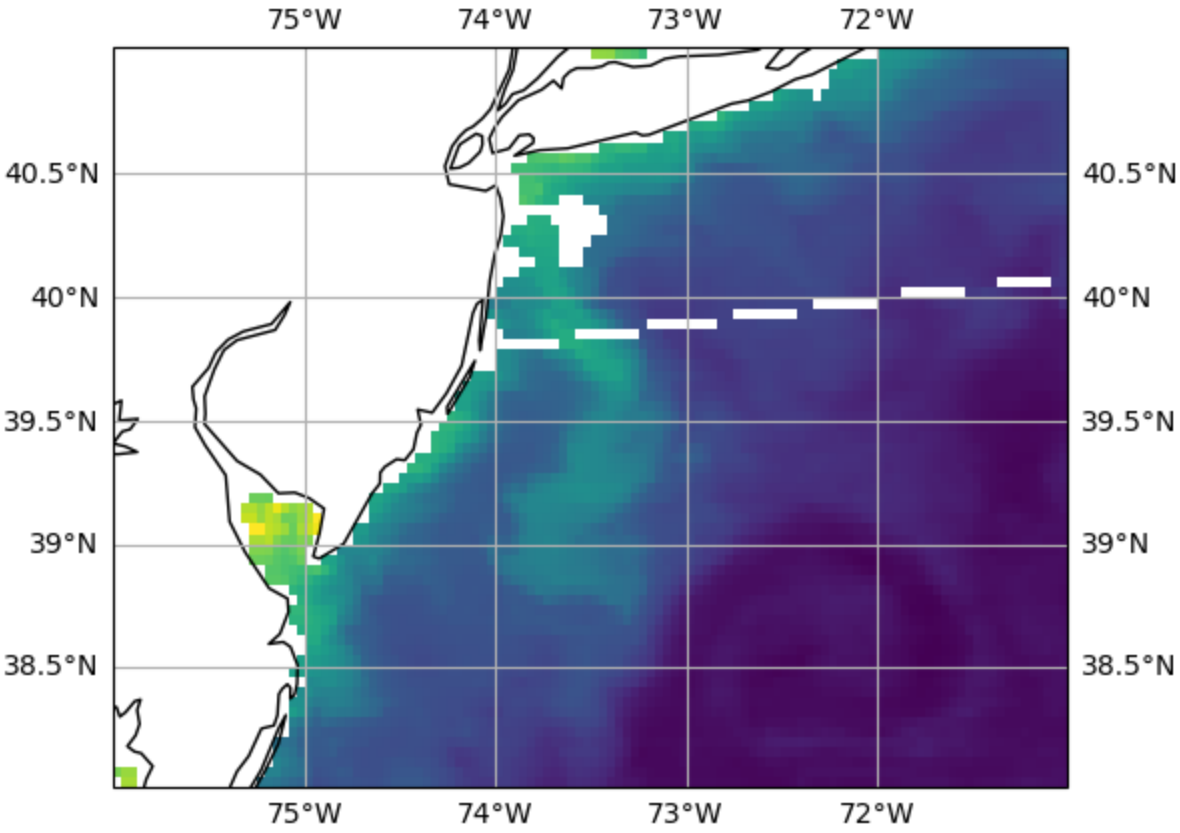
Remake the map of chlorophyll that we looked at before using Mercator projection. Make sure to plot the log of chlorophyll as `np.log10(___)` and this means we need to load numpy, too.

In [33]:
```
import numpy as np

# https://polarwatch.noaa.gov/erddap/griddap/erdMH1chla1day.html
url = '../data/A2019210.L3m_DAY_CHL_chlor_a_4km.nc'
data = xr.open_dataset(url)

data_mab_nj = data.sel( lat=slice(41, 38), lon=slice(-76,-71))
```

In [34]:
```
fig = plt.figure()
ax = plt.axes(projection=ccrs.Mercator())
ax.gridlines(draw_labels=True)
ax.coastlines(resolution='50m')
```

```
plt.pcolormesh(data_mab_nj.lon, data_mab_nj.lat, np.log10(data_mab_nj.chlor_
```

Out[34]: <cartopy.mpl.geocollection.GeoQuadMesh at 0x1877ff990>



Bathymetry

We've just seen that xarray can 'play nice' with cartopy. Guess what format most bathymetry files are given in? Netcdf. That is nice, so we know how to make maps with bathmetry already

ETOPO:

Etopo is a global dataset with both ocean depth and land elevations. It comes in two different resolutions, and is generally a workhorse for ocean bathymetry maps. If you are working in a special region you may need to find you own special bathymetry data, which someone you work with probably has.

2 minute data: https://www.ngdc.noaa.gov/mgg/global/etopo2.html

download it for yourself: https://www.ngdc.noaa.gov/mgg/global/relief/ETOPO2/ETOPO2v2-2006/ETOPO2v2c/netCDF/ETOPO2v2c_f4_netCDF.zip

1 minute data: https://www.ngdc.noaa.gov/mgg/global/global.html

fname = https://gamone.whoi.edu/thredds/dodsC/usgs/data0/bathy/ETOPO2v2c_f4.nc.html

In [35]:
```
# import statements first

import xarray as xr
import matplotlib.pyplot as plt
import cartopy.crs as ccrs
import cartopy.feature as cfeature

%matplotlib inline
```

load the bathymetry data:

you can do this using the weblink I found for someone at whoi, or you can just get the file onto your computer somewhere (I recommend getting it yourself so you can use it anywhere). There's also a version in the data folder that we'll use.

option1: load from website:

```
fname = 'https://gamone.whoi.edu/thredds/dodsC/usgs/data0
/bathy/ETOPO2v2c_f4.nc'
ds_etpo = xr.open_dataset( fname)
ds_etpo
```

option 2: load from local file:

In [36]:
```
# option2: load from local file

fname = '../data/ETOPO2v2c_f4.nc'

ds_etpo = xr.open_dataset(fname)

ds_etpo
```

Out[36]: xarray.Dataset

▸ Dimensions: (**x**: 10800, **y**: 5400)

▾ Coordinates:

x	(x)	float32	-180.0 -179.9 ... 179.9 180.0	
y	(y)	float32	-89.98 -89.95 ... 89.95 89.98	

▾ Data variables:

z	(y, x)	float32	...	

▸ Indexes: (2)

▾ Attributes:

Conventions :	COARDS
title :	
source :	-Rd -I2m -ZTLf
node_offset :	1

select a subset, a region of interest, and plot the bathymetry

Lets make a simple plot of the Mid Atlantic Bight

Because the global bathymetry dataset can be really huge, we want to make sure we take a subset before we try to plot:

```
region = ds_etpo.sel(x=slice(-77, -70), y=slice(35, 43))
```

define the map (figure) projection:

```
proj = ___.Mercator()
```

and the data projection for the transformation:

```
data_crs = ___.PlateCarree()
```

set up your figure:

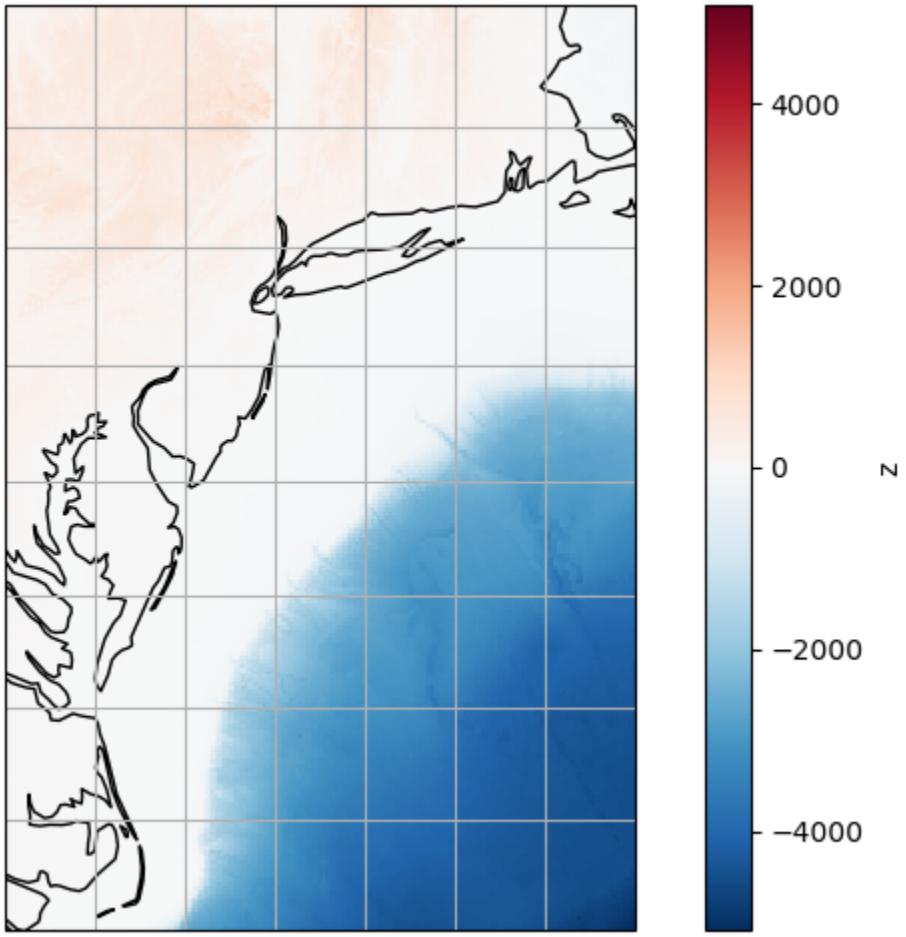
```
fig = plt.figure(figsize=(9,6))

ax = plt.axes(projection=___)

___.coastlines(resolution='50m')

ax.gridlines(draw_labels=True)
```

use xarray to make a simple plot

```
region._.plot( ___ =data_crs )
```

In [37]:
```
region = ds_etpo.sel(x=slice(-77, -70),y=slice(35,43))
proj = ccrs.Mercator()
data_crs = ccrs.PlateCarree()

fig = plt.figure(figsize=(9,6))
ax = plt.axes(projection=proj)
ax.gridlines()

region.z.plot(transform=data_crs)
# region.z.plot(transform=data_crs, vmin=-3000, vmax =0)
ax.coastlines(resolution='50m')
```

Out[37]:
```
<cartopy.mpl.feature_artist.FeatureArtist at 0x187966a10>
```



change the plot to be the southern coast of greenland:

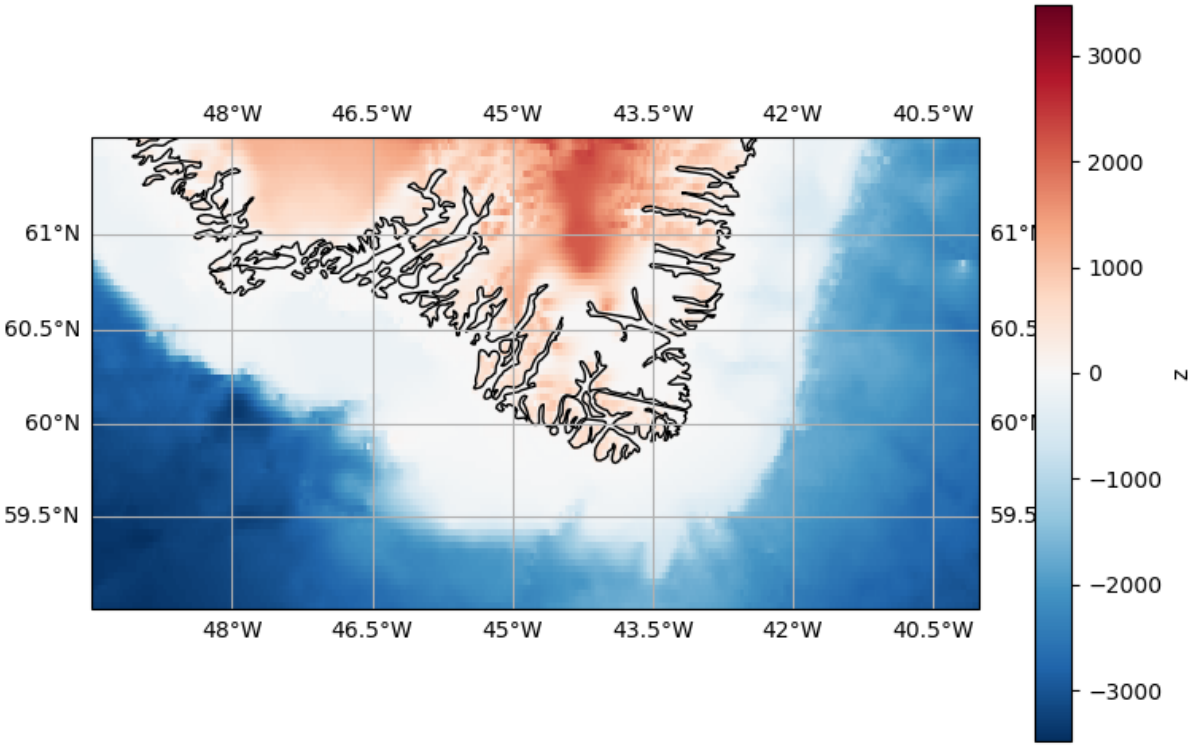
```
region = ds_etpo.sel(x=slice(-48.75, -40), y=slice(59, 61.5))
```

make the same map:

In [38]:
```
region = ds_etpo.sel(x=slice(-49.5, -40), y=slice(59, 61.5))

proj = ccrs.Mercator()
data_crs = ccrs.PlateCarree()
```

```
fig = plt.figure(figsize=(9,6))
ax = plt.axes(projection=proj)
ax.gridlines(draw_labels=True)

region.z.plot( transform =data_crs )
ax.coastlines(resolution='10m')
```

Out[38]: <cartopy.mpl.feature_artist.FeatureArtist at 0x187894350>



use matplotlib to make a map, not xarray default.

Our bathymetry maps could use a little help. Lets make a map that has depth contours, which is the way we typically see maps. To do this we will use a standard matplotlib function called `plt.contourf()` that makes filled contours. We just need to give it the x, y, z data, and a list of contours we want to draw (i.e. isobaths). Then we need to pass it the `transform=` command since we are working with a cartopy map

Define the map and data projections, get the same region of the etopo bathymetry and create the figure.

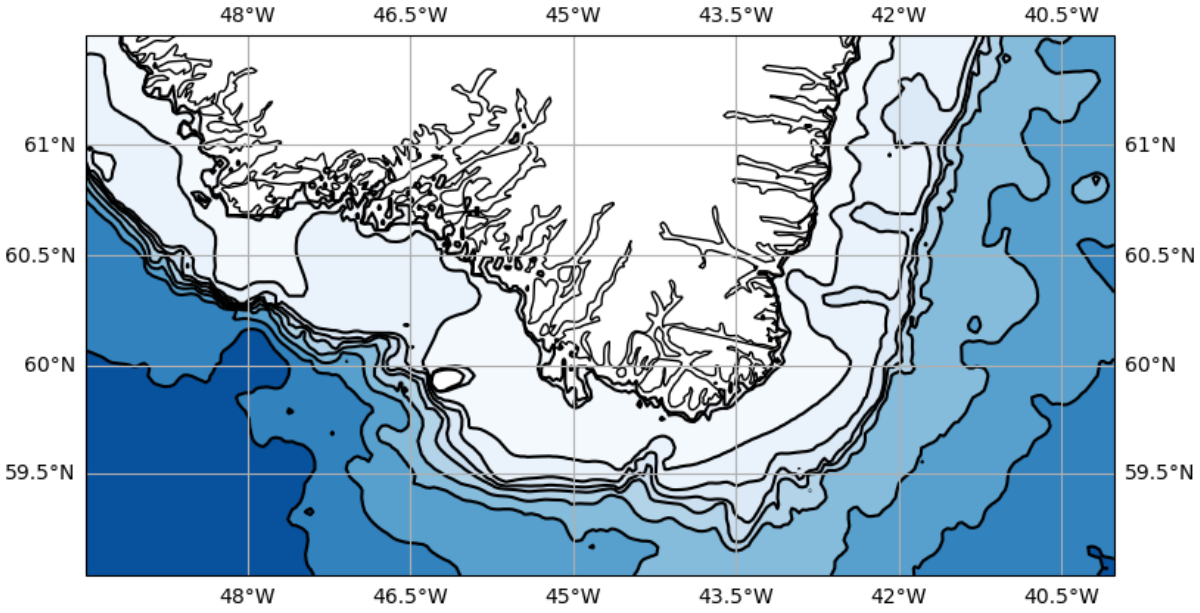
Now through we are going to use matplotlib:

```
# define the isobaths I want to plot:
lvls = [-4000, -3000, -2500, -2000, -1500, -1000, -700, -400, -145,
-10, 0]
```

```
plt.contourf( ___.x, ___.y, ___.z, levels=lvls, ___=data_crs ,
cmap='Blues_r')
```

In [39]:
```
proj = ccrs.Mercator()
data_crs = ccrs.PlateCarree()

# define the isobaths I want to plot:
lvls = [-4000, -3000, -2500, -2000, -1500, -1000, -700, -400, -145, -10, 0]

fig = plt.figure(figsize=(9,6))
ax = plt.axes(projection=proj)
ax.coastlines(resolution='10m')
ax.gridlines(draw_labels=True)

plt.contourf( region.x, region.y, region.z, levels=lvls, transform=data_crs
plt.contour( region.x, region.y, region.z, levels=lvls, transform=data_crs ,
```

Out[39]:
```
<cartopy.mpl.contour.GeoContourSet at 0x187986e90>
```



finally, plot some data on a map:

Note that we are still just using matplotlib to plot things. The only extra thing we are doing is passing the `transform=` argument that tells the figure how to place our data onto a map. So we should just be able to make any normal plots we want and add data to the map.

I've put a csv file in the data folder that has lat, lon, and average 0-500m helium observations from a cruise a few years ago. You can think of this like the meltwater content in the water column. Let's plot these data on the map. We will use pandas to read the csv file, then use `plt.scatter()` to plot them

```
df = pd.read_csv('../data/OSNAP_helium.csv')
```

```
df.head()
```

In [40]:
```
# import pandas and use pd.read_csv()
import pandas as pd

df = pd.read_csv('../data/OSNAP_helium.csv')
df.head()
```

Out[40]:

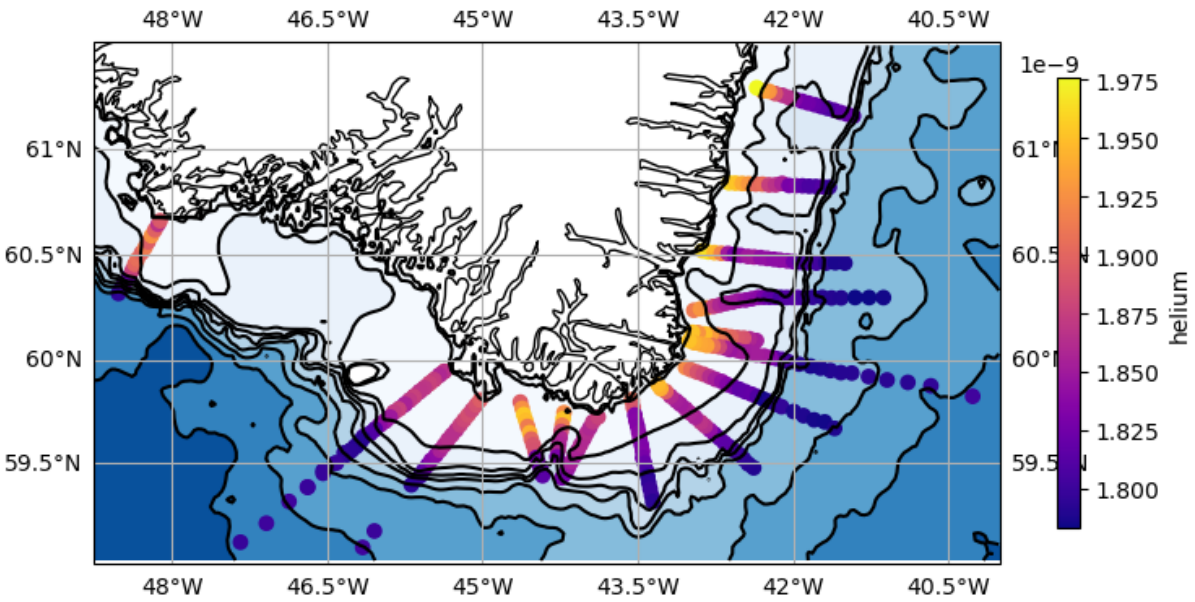
	Unnamed: 0	lon	lat	meanHE
0	0	-41.420667	61.156500	1.810445e-09
1	1	-41.481667	61.165167	1.818041e-09
2	2	-41.545333	61.173500	1.819294e-09
3	3	-41.606667	61.183500	1.815365e-09
4	4	-41.670667	61.192500	1.808523e-09

now set up the same figure again, and fill in the blanks below to plot the meltwater data

```
plt.scatter(___, ___,  c=___, ___=data_crs, cmap='plasma')
ax.set_extent([-48.75, -40, 59, 61.5])
```

In [41]:
```
proj = ccrs.Mercator()
data_crs = ccrs.PlateCarree()

# define the isobaths I want to plot:
lvls = [-4000, -3000, -2500, -2000, -1500, -1000, -700, -400, -145, -10, 0]

fig = plt.figure(figsize=(9,6))
ax = plt.axes(projection=proj)
ax.coastlines(resolution='10m')
ax.gridlines(draw_labels=True)

plt.contourf( region.x, region.y, region.z, levels=lvls, transform=data_crs
plt.contour( region.x, region.y, region.z, levels=lvls, transform=data_crs ,

plt.scatter(df.lon, df.lat,  c=df.meanHE, transform=data_crs, cmap='plasma')
ax.set_extent([-48.75, -40, 59, 61.5])

plt.colorbar(shrink=0.6, label='helium')
```

Out[41]:
```
<matplotlib.colorbar.Colorbar at 0x187a6d110>
```



slightly nicer version

Now you should have the tools to make a simple map of your data anywhere in the world. You can make these maps infinitly complex and nice looking, but by analogy with what we have done so far you should be able to get started.

Just for fun the code below is almost like what we have done, but is more how I would make the map above for a paper or a talk, it's just a little bit nicer:

In [42]:
```
fig, ax = plt.subplots( figsize=(7, 7), subplot_kw=dict(projection=ccrs.Merc

ax.set_extent([-49.5, -40, 59, 61.5],crs=ccrs.PlateCarree())

lvl = [-4000, -3000, -2500, -2000, -1500, -1000, -700, -400, -145, -15]

feature = ax.add_feature(cfeature.NaturalEarthFeature('physical', 'land', '1

ax.contourf(region.x, region.y, region.z,levels=50,  cmap='Blues_r' ,  trans
ax.contour(region.x, region.y, region.z, levels= lvl, colors='k', linestyles

glb = ax.gridlines(draw_labels=True, alpha=0.5, linewidth=.5)
# glb.xlabels_top = glb.ylabels_left = False
glb.top_labels = glb.left_labels = False

plt.scatter(df.lon, df.lat,  c=df.meanHE, transform=data_crs, cmap='plasma')

# add an inset figure for context
sub_ax = fig.add_axes([0.07, 0.28, 0.2, 0.2],
                      projection=ccrs.NearsidePerspective(central_longitude=

sub_ax.plot([-49.5, -49.5,  -40, -40, -49.5] ,
                        [ 59, 61.5, 61.5, 59, 59 ], color='tab:red',linewidt
sub_ax.set_global()
```

```
LAND = cfeature.NaturalEarthFeature(
    'physical', 'land', '50m',
    edgecolor='face',
    facecolor='black')
sub_ax.add_feature(LAND, zorder=0)
sub_ax.gridlines(linewidths=.5)
```

Out[42]: <cartopy.mpl.gridliner.Gridliner at 0x187ce6010>