MEA 462 - Observational Methods and Data Analysis in Marine Physics

Introduction to Python Programming

Joseph B. Zambon

Session 3: 30 March 2022
Introduction to Channels, Data Access Protocol, Datasets, Plots, Maps, and GitHub

In the previous 2 sessions, we've completed very simple calculations using Python. For this session we're going to apply this knowledge to developing high-resolution, publication-quality images with actual oceanographic data. At the end we'll figure out how to share our awesome code with the world using one of the biggest code-sharing communities in the world, GitHub.

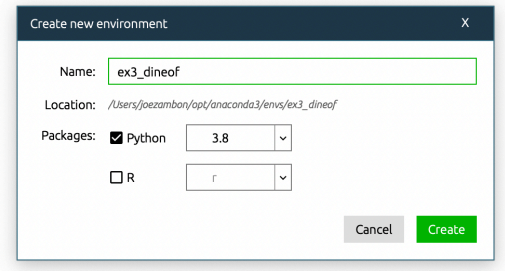
Channels

Anaconda comes preloaded with a default channel. This channel is useful to find a lot of modules or routines, but occasionally you need more specialized software. For example, the Integrated Ocean Observing System (IOOS) group developed their own software routines that are separate from the Anaconda defaults. Similarly, another open-source channel that is widely used is conda-forge.

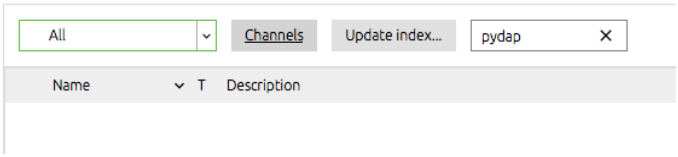
To start this exercise, we are going to need the pydap package, specifically the subroutines available under pydap.client.

pydap – Python utilities designed for the Data Access Protocol (DAP) method of retrieving data. A number of servers exist that use DAP to transfer data efficiently, we will be using our in-house Thematic Real-time Environmental Distributed Data Services (THREDDS) DAP server to provide data. (Package info: http://anaconda.org/conda-forge/pydap and information about OPeNDAP http://www.opendap.org/about)

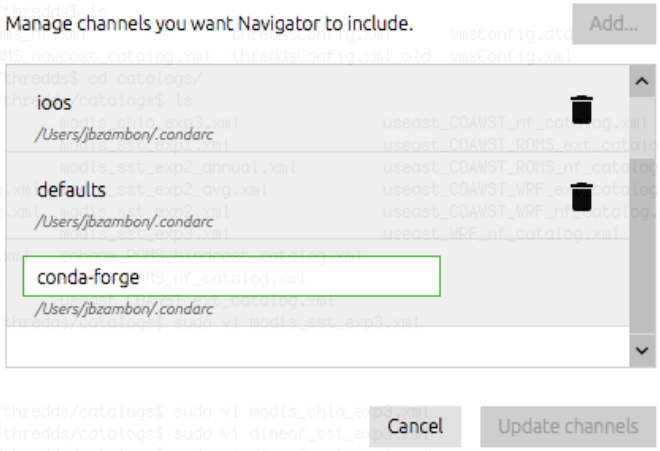
Create a new environment, I'm calling this one ex3_dineof. Again, make sure you're running in Python 3.x.



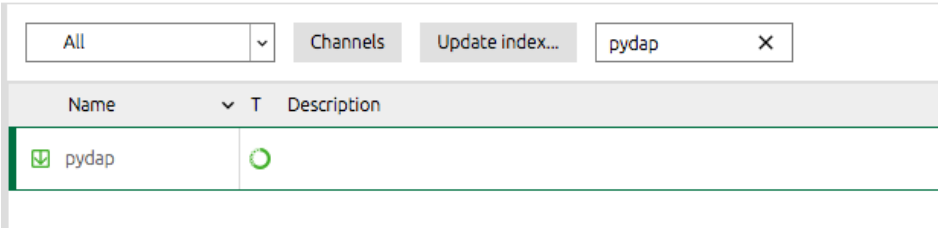
Change your filters from Installed to All and search for "pydap". Nothing comes up, because it is not available in your default channels and separate search channels need to be added.



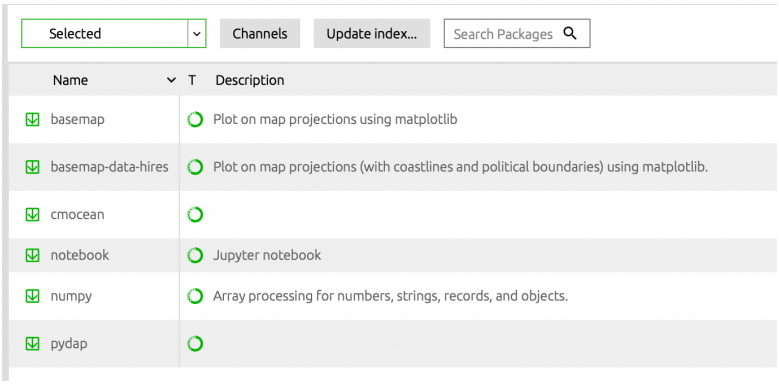
Click “Channels” then “Add…” and type in the channel name, “conda-forge”. We’re also going to be playing with colormaps from the ioos channel, so grab that too.



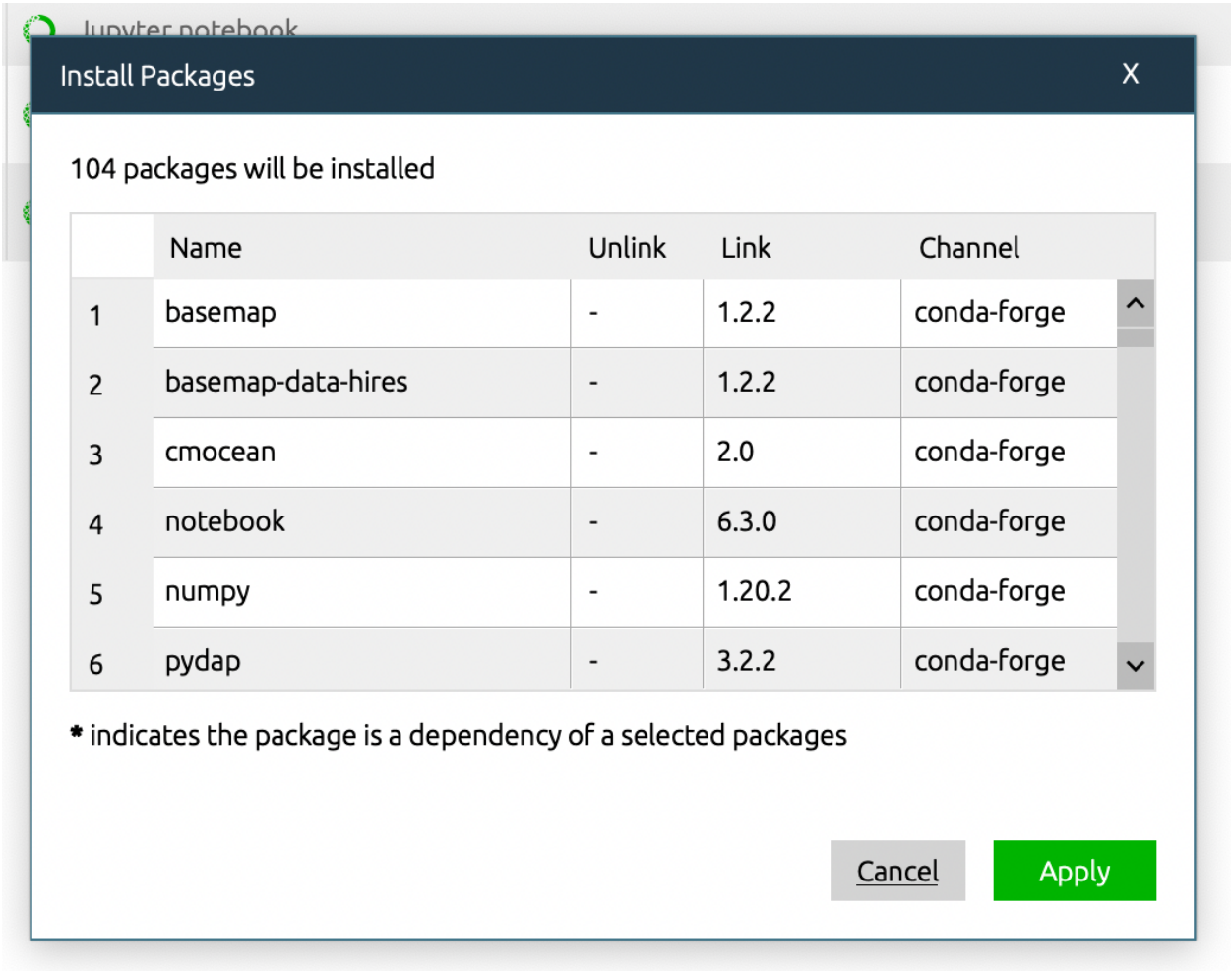
Click “Update channels”, then at the main navigator screen click “Update index…”. This may take a few moments. Once updated, search for pydap and you should be able to find it from the conda-forge channel, select it for download.



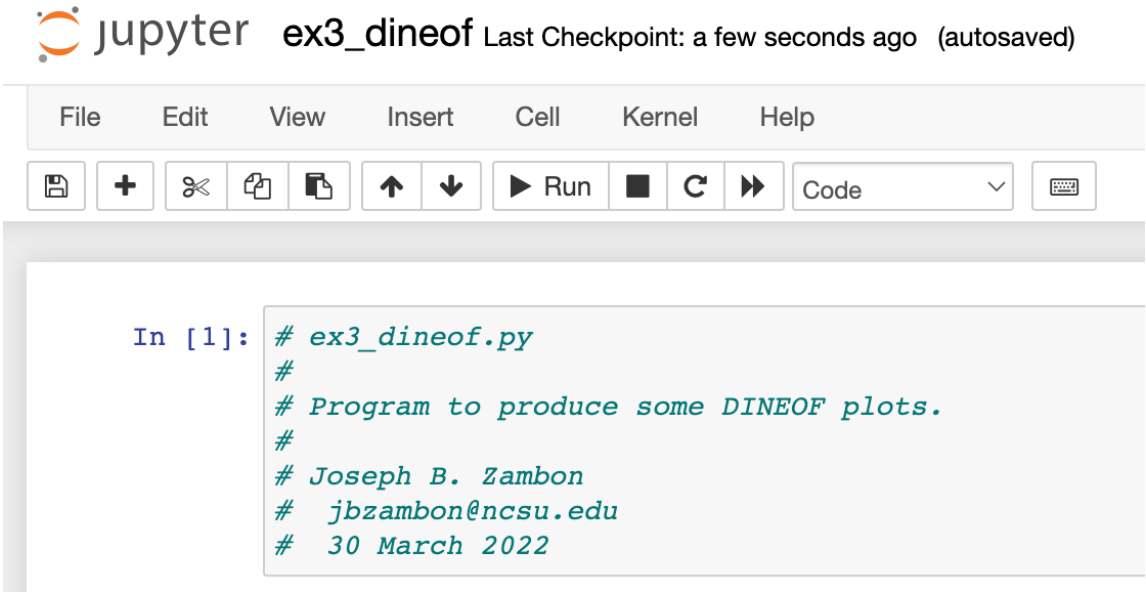
To produce publication-quality images, we will also need to download another package called matplotlib. As you can tell from the name, it is heavily derived from Matlab and you’ll hopefully notice several parallels that make going from Matlab to Python not very difficult. Also download basemap so that we can draw some maps, download basemap-data-hires so we can draw some detailed maps (as we will see, the defaults are crudely drawn). The cmocean package has wonderful perceptually uniform colormaps we’re going to use (Reference: http://matplotlib.org/cmocean/). Download notebook and numpy as we have been doing. Hit Apply and Apply again to load the packages and requisite dependencies. All told, you should have installed: pydap, matplotlib, basemap, basemap-data-hires, notebook, cmocean, and numpy.



In my case, it's going to install/update 104 packages.



Click the Play button, the Open with Jupyter Notebook, and create a new Python 3 notebook called ex3_dineof.



Here are all of the modules we're going to be using in this exercise.

```
In [2]: # For inline plotting
        %pylab inline

        # Module declarations
        import numpy as np                              # conda install numpy
                                                          # Imported to express various datatypes
        from pydap.client import open_url               # conda install -c conda-forge pydap
                                                          # Imported for DAP server access
        import matplotlib.pyplot as plt                 # conda install matplotlib
                                                          # Imported for drawing figures
        from mpl_toolkits.basemap import Basemap        # conda install basemap
                                                          # Imported to draw publication-quality figures
        from datetime import datetime                   # conda install datetime
                                                          # Imported to produce datetime string objects
        import cmocean                                  # conda install -c ioos cmocean
                                                          # Imported for its perceptually-uniform colormaps
        from matplotlib.colors import LogNorm           # conda install matplotlib
                                                          # Imported for chlorophyll-a plotting

        Populating the interactive namespace from numpy and matplotlib
```

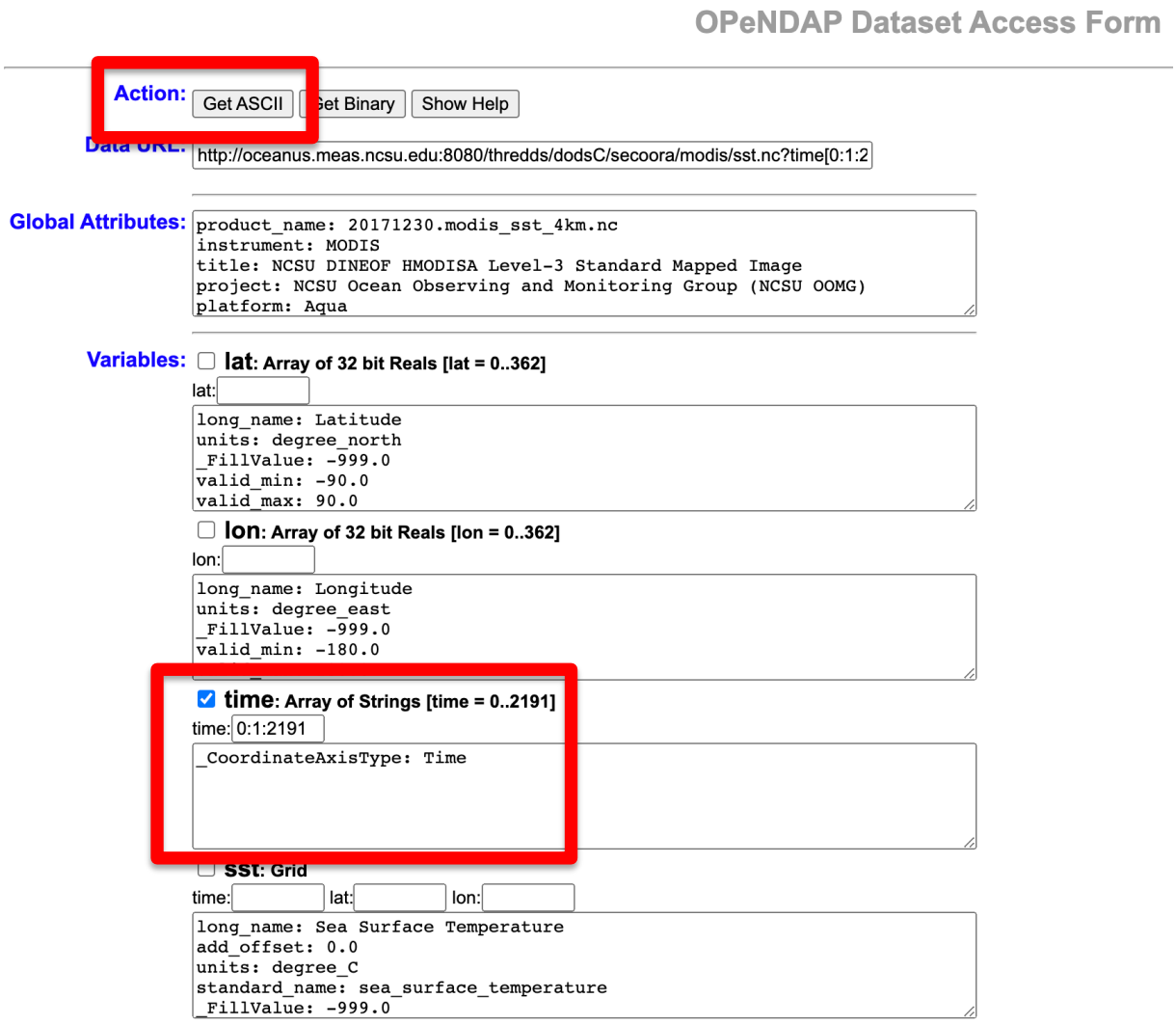
We're going to use the datetime function to make a dynamic program that allows you to select any date and time that's available from the DAP server. In this case, we'll start with 17 April 2017 (the day we arrived on station for a NSF-PEACH cruise). After you work through feel free to use any date from 2003-2017. We'll show how to look at available dates next.

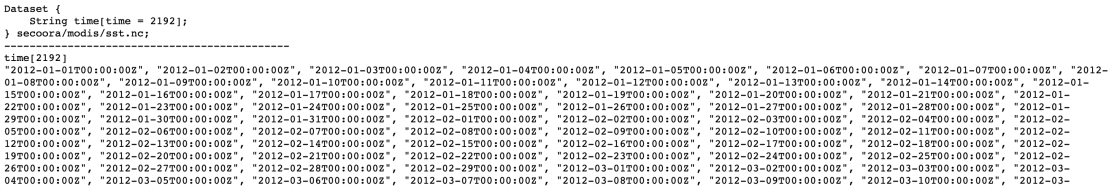
```
In [5]: # Put your desired plot date in here
        #                       YYYY,MM,DD
        plot_date = datetime(2017, 4, 17)
        print(plot_date)

        2017-04-17 00:00:00
```

OPeNDAP Access

The datasets we're going to be using are on a 180TB server that sits in Jordan Hall. The catalog of all available data products can be found at http://oceanus.meas.ncsu.edu:8080/thredds/catalog.html. The individual products we're going to be using are "SECOORA 4km Satellite DINEOF Data Products (Multivariate: >3 sigma vs 10yr average removed)". To see what products are available, click on one of the variables (i.e. "MODIS SST"), then the dataset (i.e. "USEast MODIS SST Experiment 3 (Multivariate: >3 sigma vs 10yr average removed)"), then the OPENDAP dataset under "Access" ("OPENDAP: /thredds/dodsC/useast/modis_exp3/sst.nc").

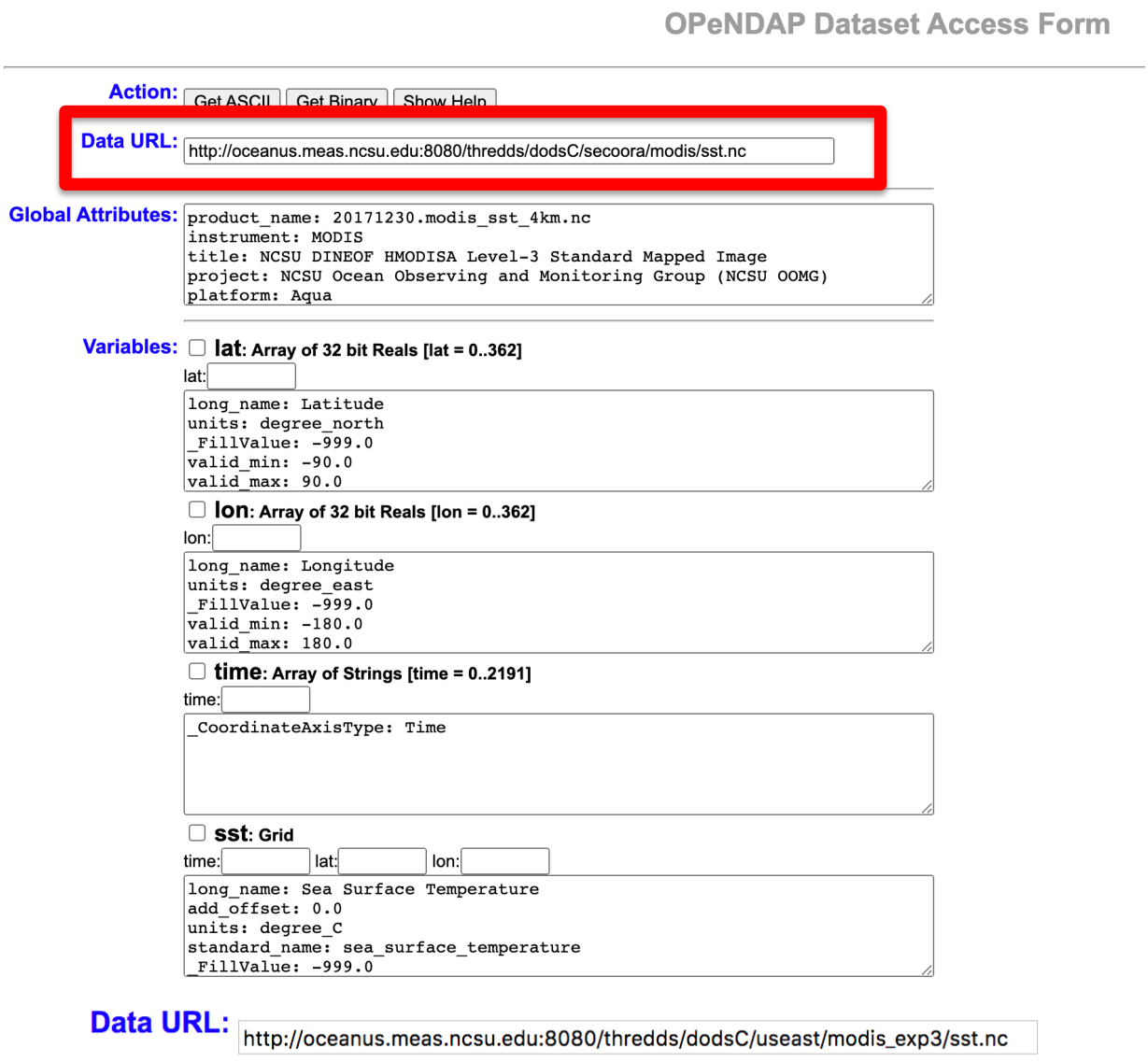
Here you can easily browse the available data. If you click on time, you will be able to view the array of strings for currently available dates (at the time of this writing it was 2192 frames). Click Get ASCII to view the available times in a simple text format.





```
Dataset {
    String time[time = 2192];
} secoora/modis/sst.nc;
---------------------------------------------
time[2192]
"2012-01-01T00:00:00Z", "2012-01-02T00:00:00Z", "2012-01-03T00:00:00Z", "2012-01-04T00:00:00Z", "2012-01-05T00:00:00Z", "2012-01-06T00:00:00Z", "2012-01-07T00:00:00Z", "2012-
01-08T00:00:00Z", "2012-01-09T00:00:00Z", "2012-01-10T00:00:00Z", "2012-01-11T00:00:00Z", "2012-01-12T00:00:00Z", "2012-01-13T00:00:00Z", "2012-01-14T00:00:00Z", "2012-01-
15T00:00:00Z", "2012-01-16T00:00:00Z", "2012-01-17T00:00:00Z", "2012-01-18T00:00:00Z", "2012-01-19T00:00:00Z", "2012-01-20T00:00:00Z", "2012-01-21T00:00:00Z", "2012-01-
22T00:00:00Z", "2012-01-23T00:00:00Z", "2012-01-24T00:00:00Z", "2012-01-25T00:00:00Z", "2012-01-26T00:00:00Z", "2012-01-27T00:00:00Z", "2012-01-28T00:00:00Z", "2012-01-
29T00:00:00Z", "2012-01-30T00:00:00Z", "2012-01-31T00:00:00Z", "2012-02-01T00:00:00Z", "2012-02-02T00:00:00Z", "2012-02-03T00:00:00Z", "2012-02-04T00:00:00Z", "2012-02-
05T00:00:00Z", "2012-02-06T00:00:00Z", "2012-02-07T00:00:00Z", "2012-02-08T00:00:00Z", "2012-02-09T00:00:00Z", "2012-02-10T00:00:00Z", "2012-02-11T00:00:00Z", "2012-02-
12T00:00:00Z", "2012-02-13T00:00:00Z", "2012-02-14T00:00:00Z", "2012-02-15T00:00:00Z", "2012-02-16T00:00:00Z", "2012-02-17T00:00:00Z", "2012-02-18T00:00:00Z", "2012-02-
19T00:00:00Z", "2012-02-20T00:00:00Z", "2012-02-21T00:00:00Z", "2012-02-22T00:00:00Z", "2012-02-23T00:00:00Z", "2012-02-24T00:00:00Z", "2012-02-25T00:00:00Z", "2012-02-
26T00:00:00Z", "2012-02-27T00:00:00Z", "2012-02-28T00:00:00Z", "2012-02-29T00:00:00Z", "2012-03-01T00:00:00Z", "2012-03-02T00:00:00Z", "2012-03-03T00:00:00Z", "2012-03-
04T00:00:00Z", "2012-03-05T00:00:00Z", "2012-03-06T00:00:00Z", "2012-03-07T00:00:00Z", "2012-03-08T00:00:00Z", "2012-03-09T00:00:00Z", "2012-03-10T00:00:00Z", "2012-03-
```

The last important bit on this page is the Data URL. This is what you will be using in your python script to reference the data. Unselect "time" so that you can gain access to all of the data.



Referencing OPeNDAP through Python

So now we have a Data Access URL. This will allow us to download subsets of the data, which is extremely useful to efficiently get data. For example, we can extract a small part of the spatial domain (i.e. just the NC coast) or a specific point in the temporal domain (i.e. 17 April 2017).

Let's open a connection (open_url) to all of the datasets we're going to be using (MODIS and DINEOF SST and Chlorophyll-a). Then lets use the .keys function to see what data variables are available in python (we already did this using your web browser).

In [6]:
```
# Link OPeNDAP datasets
modis_sst_url = 'http://oceanus.meas.ncsu.edu:8080/thredds/dodsC/secoora/modis/sst.nc'
modis_chla_url = 'http://oceanus.meas.ncsu.edu:8080/thredds/dodsC/secoora/modis/chla.nc'
dineof_sst_url = \
          'http://oceanus.meas.ncsu.edu:8080/thredds/dodsC/secoora/dineof/sst.nc'
dineof_chla_url = \
          'http://oceanus.meas.ncsu.edu:8080/thredds/dodsC/secoora/dineof/chla.nc'

modis_sst_dataset = open_url(modis_sst_url,output_grid=False)
modis_chla_dataset = open_url(modis_chla_url,output_grid=False)
dineof_sst_dataset = open_url(dineof_sst_url,output_grid=False)
dineof_chla_dataset = open_url(dineof_chla_url,output_grid=False)
print('Available data:')
print('MODIS SST:', modis_sst_dataset.keys)
print('MODIS Chla:', modis_chla_dataset.keys)
print('DINEOF SST:', dineof_sst_dataset.keys)
print('DINEOF Chla:', dineof_chla_dataset.keys)
```

```
Available data:
MODIS SST: <bound method Mapping.keys of <DatasetType with children 'lat', 'lon', 'time', 'sst'>>
MODIS Chla: <bound method Mapping.keys of <DatasetType with children 'lat', 'lon', 'time', 'chlor_a'>>
DINEOF SST: <bound method Mapping.keys of <DatasetType with children 'lat', 'lon', 'time', 'sst'>>
DINEOF Chla: <bound method Mapping.keys of <DatasetType with children 'lat', 'lon', 'time', 'chlor_a'>>
```

Next, we use the datetime function to convert the desired date to an ordinal (numerical) value. Then load the available list of times from one of the datasets, and convert those to ordinal values. They are all temporally identical, so it doesn't matter if you choose modis_sst_dataset or dineof_chla_dataset, etc. With an apples-to-apples (ordinals-to-ordinals) comparison, we can find the matching value. This will give you the time index (t_index) we will reference for the remainder of the exercise. The value "1933" represents the 1934th frame (counting from 0) and is the day we are interested in (17 April 2017).

In [5]:
```
# Find time index from available times

plot_date = datetime.toordinal(plot_date)
times = np.array(modis_sst_dataset['time'])
for t in range(0,np.size(times)):
    times[t] = datetime.toordinal(datetime.strptime(times[t], "%Y-%m-%dT%H:%M:%SZ"))
float_times = np.float32(times)
t_index = int(np.where(float_times==plot_date)[0])
print(t_index)
```

```
1933
```

Now that we know the correct time index, we can download that specific time subset of data. This allows us to only download a megabyte or so of data, rather than the several gigabytes of the entire dataset. You could further spatially subset if you're only interested in a specific region, but we'll leave that out for now.

We will also download the latitude (lat) and longitude (lon) information for this data. Just like the time data, the space (lat, lon) data is congruent among all 4 data products so we only need to load it from one dataset (i.e. modis_sst_dataset).

```
# Load data from found time index
lat = np.array(modis_sst_dataset['lat'][:])    #Constant among datasets
lon = np.array(modis_sst_dataset['lon'][:])    #Constant among datasets

modis_sst = np.array(modis_sst_dataset['sst'][t_index][:][:])         #MODIS SST
modis_chla = np.array(modis_chla_dataset['chlor_a'][t_index][:][:])   #MODIS Chlor-a
dineof_sst = np.array(dineof_sst_dataset['sst'][t_index][:][:])       #DINEOF Analyzed SST
dineof_chla = np.array(dineof_chla_dataset['chlor_a'][t_index][:][:]) #DINEOF Analyzed Chlor-a
```

You now have loaded in 4 arrays for the same time and space. Let's take a look at the shape of the data. Notice that there is an extra dimension in the loaded data. You can get rid of it using the "squeeze" function (you may recall this function from Matlab). Now the data represents a 2-D, 363x363, shape that we can plot on a map. We also use the meshgrid function to transfer the 1-D lat and lon information into a 2-D, 363x363, shape.

In [12]:
```
print('Original MODIS shape: ', np.shape(modis_sst))
modis_sst = np.squeeze(modis_sst)     #Get rid of singleton time dimension
print('Squeezed MODIS shape: ', np.shape(modis_sst))
modis_chla = np.squeeze(modis_chla)     #Get rid of singleton time dimension
dineof_sst = np.squeeze(dineof_sst)     #Get rid of singleton time dimension
dineof_chla = np.squeeze(dineof_chla)     #Get rid of singleton time dimension

print('Original Lat shape:   ', np.shape(lat))
print('Original Lon shape:   ', np.shape(lon))
[xlon,xlat] = np.meshgrid(lon,lat)    # Create lon/lat grid
print('Meshgrided Lat shape: ', np.shape(xlon))
print('Meshgrided Lon shape: ', np.shape(xlat))
```

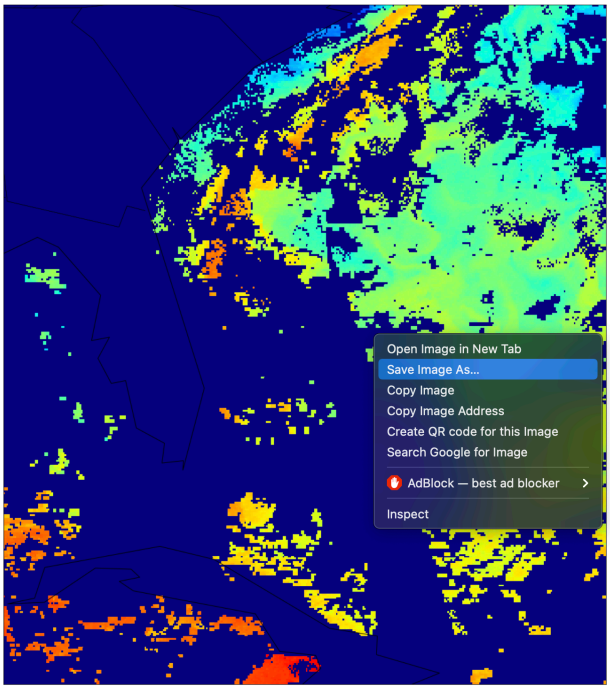
```
Original MODIS shape:  (363, 363)
Squeezed MODIS shape:  (363, 363)
Original Lat shape:    (363,)
Original Lon shape:    (363,)
Meshgrided Lat shape:  (363, 363)
Meshgrided Lon shape:  (363, 363)
```

First, we create a 25x20 figure and initialize the plot [plt.clf()]. We start by creating a basemap that we will draw the rest of the figure onto, the variable is simply "map". The basemap is on a Mercator ("merc") projection, has coarse ('c') resolution, and the defined bounds and projection information shown. We then draw coastlines, countries, and states. Finally, we can put the data on the map using the pcolormesh() function to plot the 2-D data with defined min/max of 15º/30ºC. We start with the "jet" colormap, which probably looks familiar to you.

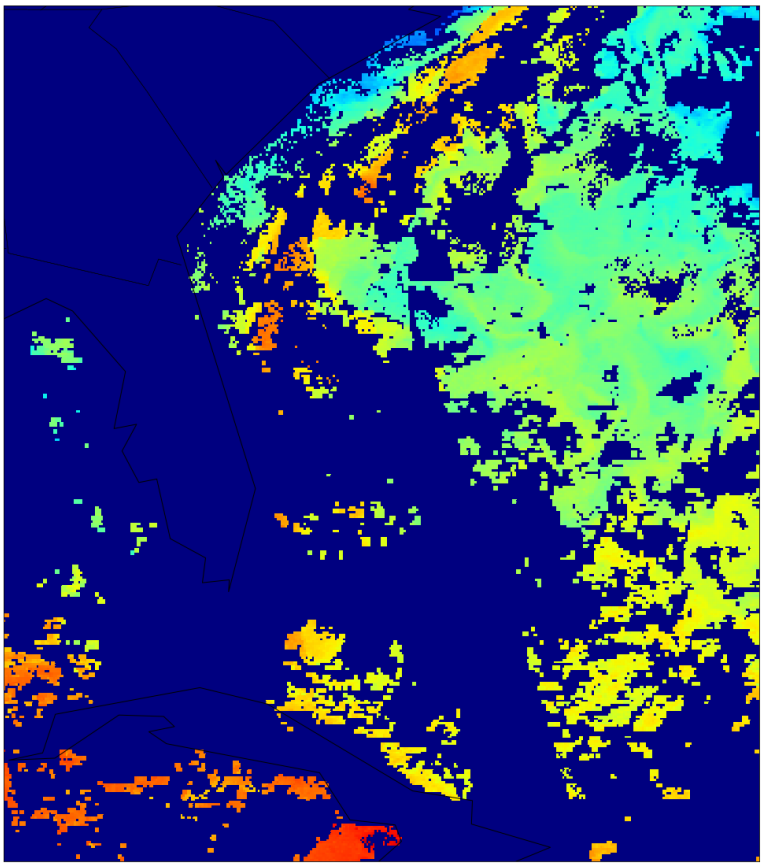
In [19]:
```
figsize(25,20)
plt.clf()
map = Basemap(projection='merc',
    resolution='c',lat_0=((np.max(lat)-np.min(lat))/2),
    lon_0=((np.max(lon)-np.min(lon))/2),
    llcrnrlon=np.min(lon),llcrnrlat=np.min(lat),
    urcrnrlon=np.max(lon),urcrnrlat=np.max(lat))
map.drawcoastlines()
map.drawcountries()
map.drawstates()
map.pcolormesh(xlon,xlat,modis_sst[:,:],cmap='jet',vmin=15,vmax=30,latlon='true')
```

You can save the figure to full resolution simply by right-clicking the image and then “Save Image As…”. There is a way to save images directly but I won’t get into that right now.

```
Out[19]: <matplotlib.collections.QuadMesh at 0x7f9778844df0>
```



Finally, here is the full-resolution image.

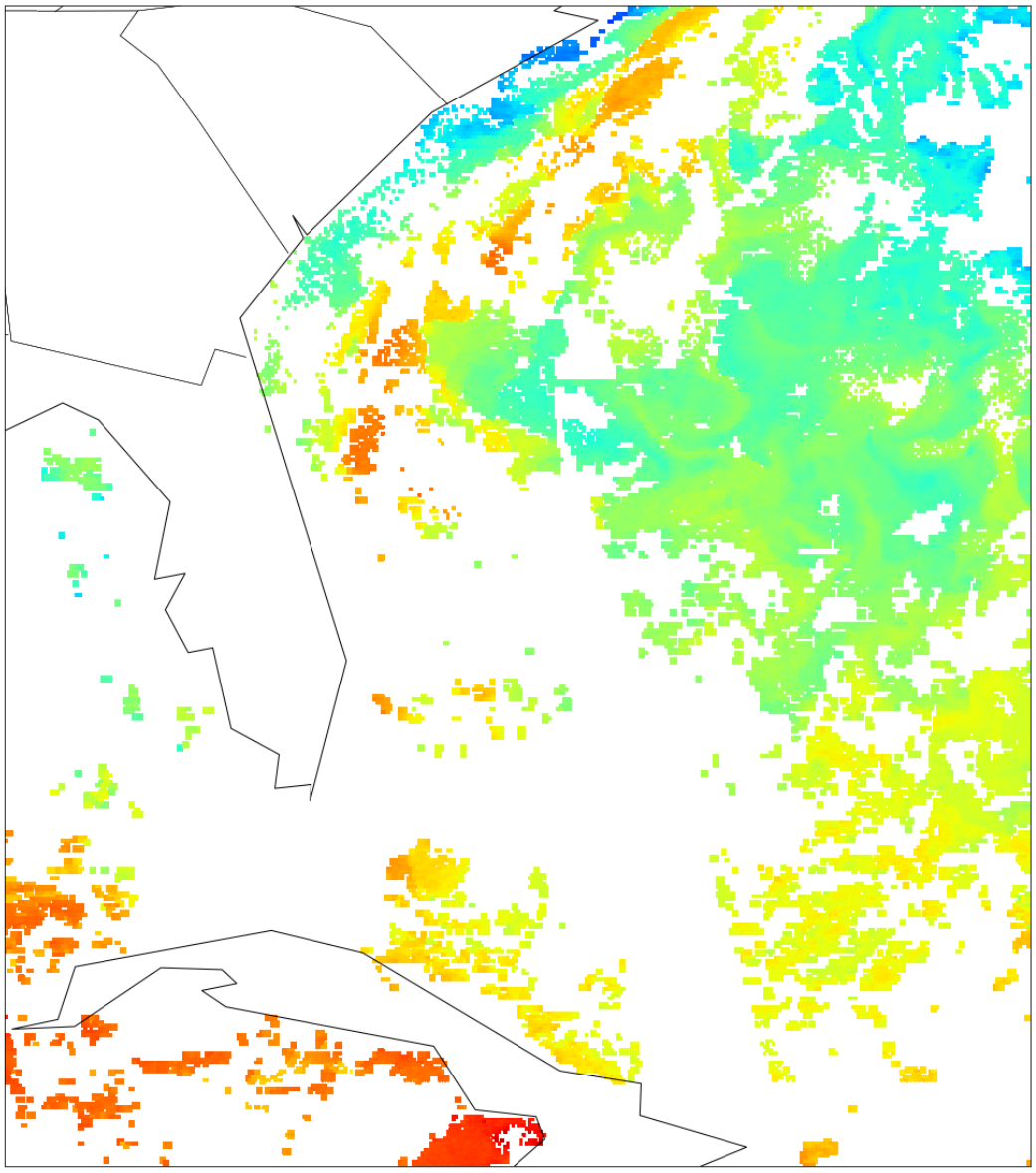


Wow, that's a lot of dark blue. What's going on?

In creating the dataset, I defined the masked area with a special value (-999). This works because it is well outside the bounds of an acceptable temperature (>725ºC below *absolute zero*). We can easily mask out values=-999 so that it doesn't plot by defining those values as "Not-a-Number" or NaN by using the numpy nan (np.nan) value. This masking value also works for Chlorophyll-a since its impossible to have Chlorophyll < 0.

In [15]:
```
modis_sst[modis_sst==-999]=np.nan

figsize(25,20)
plt.clf()
map = Basemap(projection='merc',
    resolution='c',lat_0=((np.max(lat)-np.min(lat))/2),
    lon_0=((np.max(lon)-np.min(lon))/2),
    llcrnrlon=np.min(lon),llcrnrlat=np.min(lat),
    urcrnrlon=np.max(lon),urcrnrlat=np.max(lat))
map.drawcoastlines()
map.drawcountries()
map.drawstates()
map.pcolormesh(xlon,xlat,modis_sst[:,:],cmap='jet',vmin=15,vmax=30,latlon='true')
```

With the correct masking included, we get a better-looking map.



So now we have the colors on the map, but the casual observer has no idea what the colors mean, or when the map is drawn for. The jet colormap (blue < green < yellow < red) is terrible because it is not perceptually uniform and people that are red-green colorblind (~8% of men, 0.6% of women) literally will only see gray.

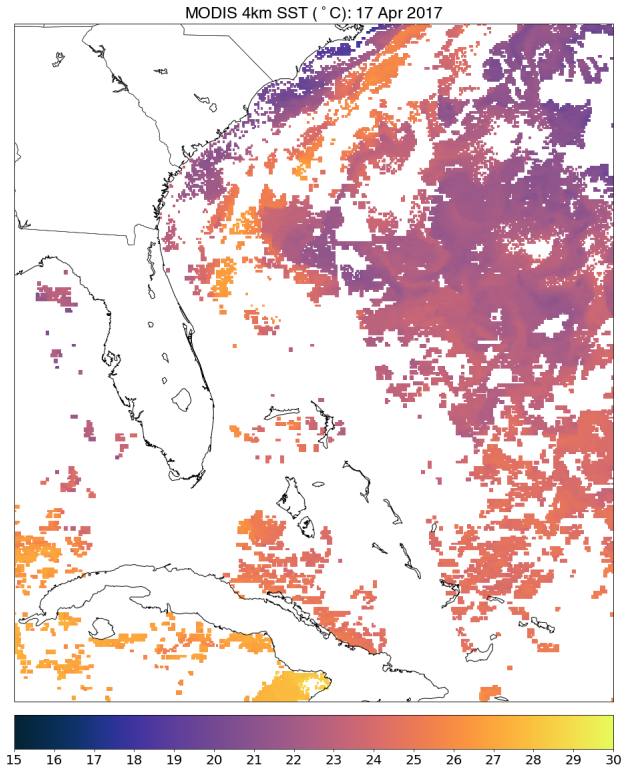
Perceptually Uniform Colormapping - A "perceptually uniform" colormap is one for which the "perceptual deltas" plot makes a simple horizontal line. (This is essentially the derivative of the colormap in perceptual space with respect to the data. We want our colormap to have the property that if your data goes from 0.1 to 0.2, this should create about the same perceptual change as if your data goes from 0.8 to 0.9.) (Reference: http://bids.github.io/colormap/)

Now, we change the colormap to the cmocean module's "thermal" colormap, add a colorbar, and use the datetime function to print a title as a concatenated string with the date converted back

from ordinal. Lastly, we represent the coastlines, national and state boundaries with higher, "intermediate" ("i") resolution.

```
figsize(25,20)
plt.clf()
map = Basemap(projection='merc',
    resolution='i',lat_0=((np.max(lat)-np.min(lat))/2),
    lon_0=((np.max(lon)-np.min(lon))/2),
    llcrnrlon=np.min(lon),llcrnrlat=np.min(lat),
    urcrnrlon=np.max(lon),urcrnrlat=np.max(lat))
map.drawcoastlines()
map.drawcountries()
map.drawstates()
map.pcolormesh(xlon,xlat,modis_sst[:,:],cmap=cmocean.cm.thermal,vmin=15,vmax=30,latlon='true')
cbar=map.colorbar(location='bottom',ticks=np.arange(0,30+1,1))
cbar.ax.tick_params(labelsize=20)
plt.title(('MODIS 4km SST ($^\circ$C): ' + \
          datetime.fromordinal(plot_date).strftime("%d %b %Y"))\
          ,fontsize=24,family='Helvetica')
```

Now we get the following map, it's certainly coming along!

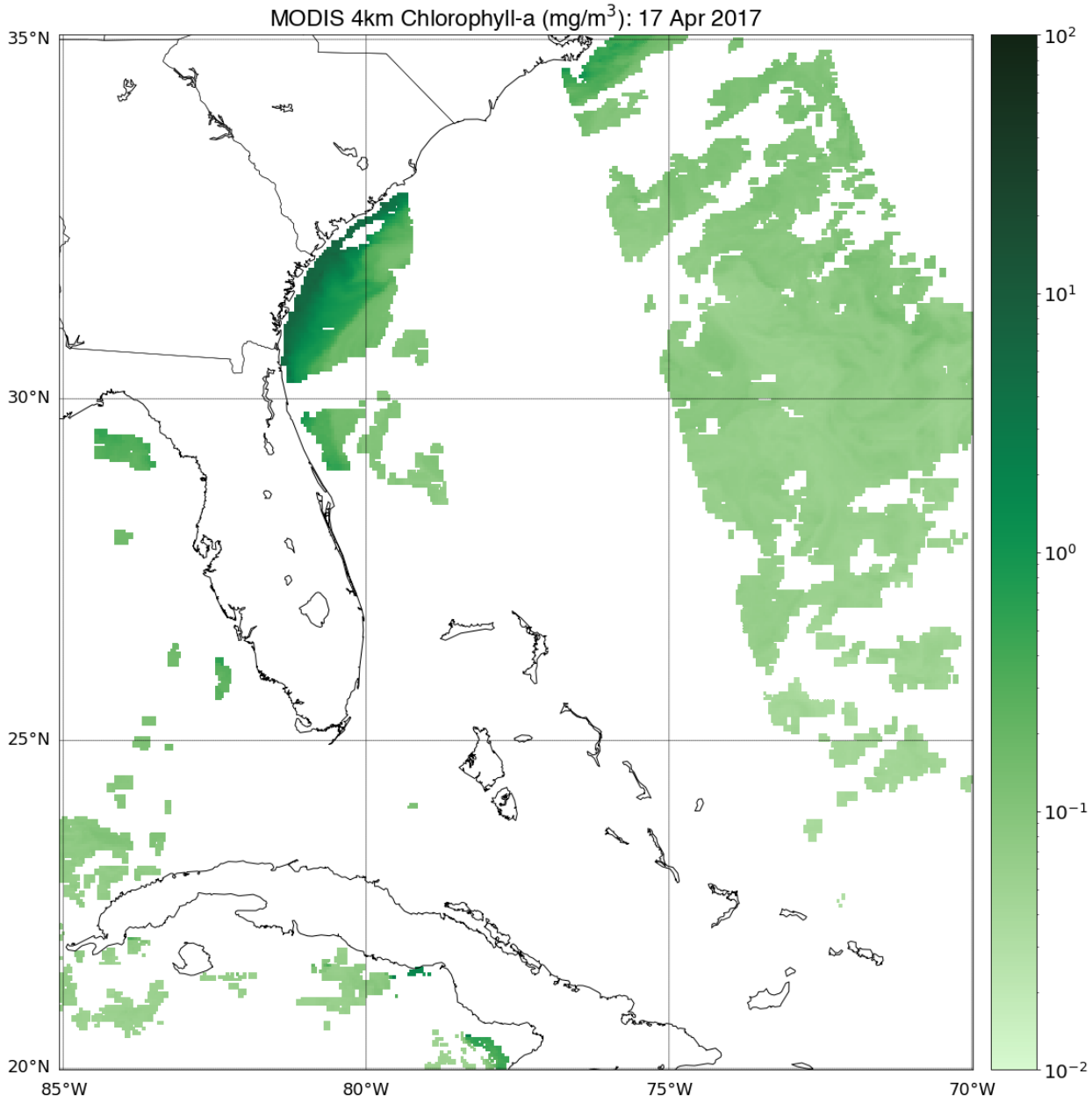


The SST is looking great, but how about the Chlorophyll-a? We can pretty much copy this same exact map with a few exceptions. Obviously the range has to be changed (Chlorophyll-a concentrations in mm/m3 are ~10x the ºC SST values. We also use a logarithmic scale (LogNorm) to better represent Chlorophyll-a concentrations. We also draw parallels (lines of latitude) and meridians (lines of longitude), spaced every 5º. Also to prevent the longitude labels

from overlapping with the colorbar at the bottom, we move the colorbar to the right side of the figure.

```
parallels = np.arange(0.,90,5.)
meridians = np.arange(180.,360.,5.)

figsize(25,20)
plt.clf()
map = Basemap(projection='merc',
    resolution='i',lat_0=((np.max(lat)-np.min(lat))/2),
    lon_0=((np.max(lon)-np.min(lon))/2),
    llcrnrlon=np.min(lon),llcrnrlat=np.min(lat),
    urcrnrlon=np.max(lon),urcrnrlat=np.max(lat))
map.drawcoastlines()
map.drawcountries()
map.drawstates()
map.pcolormesh(xlon,xlat,modis_chla[:,:],norm=LogNorm(vmin=0.01, vmax=100),cmap=cmocean.cm.alga
cbar=map.colorbar(location='right',norm=LogNorm(vmin=0, vmax=100),ticks=[0,0.01,0.1,1,10,100])
map.drawparallels(parallels,labels=[1,0,0,0],fontsize=18)
map.drawmeridians(meridians,labels=[0,0,0,1],fontsize=18)
cbar.ax.tick_params(labelsize=20)
plt.title(('MODIS 4km Chlorophyll-a (mg/m$^3$): ' + \
          datetime.fromordinal(plot_date).strftime("%d %b %Y"))\
          ,fontsize=24,family='Helvetica')
```

This gives us…



That is a pretty good-looking map, however it would be great to look at SST and Chlorophyll-a concentrations side by side. In addition, we will now finally plot the DINEOF analyzed fields. This gives us 4 separate images, we'll plot them in a 2x2 grid using the subplot() function. This is a very large cell, but there are lots of similarities between the different subplots. While it looks like a lot, we're really copy & pasting the code from our earlier plots and making subtle changes.

Firstly, we need to make sure all of the data is correctly masked to NaN -999 from the dataset. Then we define our parallels/meridians (spaced out a bit more this time to prevent cluttering; 10º). We make this figure more square-like (20x20) since it's a 2x2 figure. Finally, we create one "super-title" (suptitle) to define the date and resolution of the image. We'll also give titles for the individual subplots, but they have the information in the super-title in common.

```
modis_sst[modis_sst==-999]=np.nan
modis_chla[modis_chla==-999]=np.nan
dineof_sst[dineof_sst==-999]=np.nan
dineof_chla[dineof_chla==-999]=np.nan

parallels = np.arange(0.,90,5.)
meridians = np.arange(180.,360.,5.)

figsize(20,20)
plt.clf()

plt.suptitle('4km MODIS/DINEOF: ' + \
             datetime.fromordinal(plot_date).strftime("%d %b %Y"),\
             fontsize=36,family='Helvetica',position=(0.5,0.95))
```

In the first (1) subplot, the original MODIS SST data sits in the 2x2 subplot grid at location 1 (upper-left). We comment out the colorbar since we want it to sit to the right, the upper-right subplot will display it instead, eliminating redundancy. The title is simplified to only reflect that it's the original MODIS SST data since the date is in the super-title.

```
# Original MODIS SST
plt.subplot(2,2,1)
map = Basemap(projection='merc',
    resolution='i',lat_0=((np.max(lat)-np.min(lat))/2),
    lon_0=((np.max(lon)-np.min(lon))/2),
    llcrnrlon=np.min(lon),llcrnrlat=np.min(lat),
    urcrnrlon=np.max(lon),urcrnrlat=np.max(lat))
map.drawcoastlines()
map.drawcountries()
map.drawstates()
map.pcolormesh(xlon,xlat,modis_sst[:,:],cmap=cmocean.cm.thermal,vmin=15,vmax=30,latlon='true')
map.drawparallels(parallels,labels=[1,0,0,0],fontsize=18)
map.drawmeridians(meridians,labels=[0,0,0,1],fontsize=18)
#cbar=map.colorbar(location='right',ticks=[arange(0,30+1,5)])
cbar.ax.tick_params(labelsize=20)
plt.title('Original SST ($^\circ$C)',fontsize=24,family='Helvetica')
```

The second (2) subplot, the DINEOF MODIS SST data sits in the 2x2 subplot grid at location 2 (upper-right). We keep the colorbar here. The title is simplified to only reflect that it's the DINEOF MODIS SST data.

```
# Analyzed DINEOF SST
plt.subplot(2,2,2)
map = Basemap(projection='merc',
    resolution='i',lat_0=((np.max(lat)-np.min(lat))/2),
    lon_0=((np.max(lon)-np.min(lon))/2),
    llcrnrlon=np.min(lon),llcrnrlat=np.min(lat),
    urcrnrlon=np.max(lon),urcrnrlat=np.max(lat))
map.drawcoastlines()
map.drawcountries()
map.drawstates()
map.pcolormesh(xlon,xlat,dineof_sst[:,:],cmap=cmocean.cm.thermal,vmin=15,vmax=30,latlon='true')
cbar=map.colorbar(location='right',ticks=np.arange(0,100+1,1))
map.drawparallels(parallels,labels=[1,0,0,0],fontsize=18)
map.drawmeridians(meridians,labels=[0,0,0,1],fontsize=18)
cbar.ax.tick_params(labelsize=20)
plt.title('DINEOF SST ($^\circ$C)',fontsize=24,family='Helvetica')
```

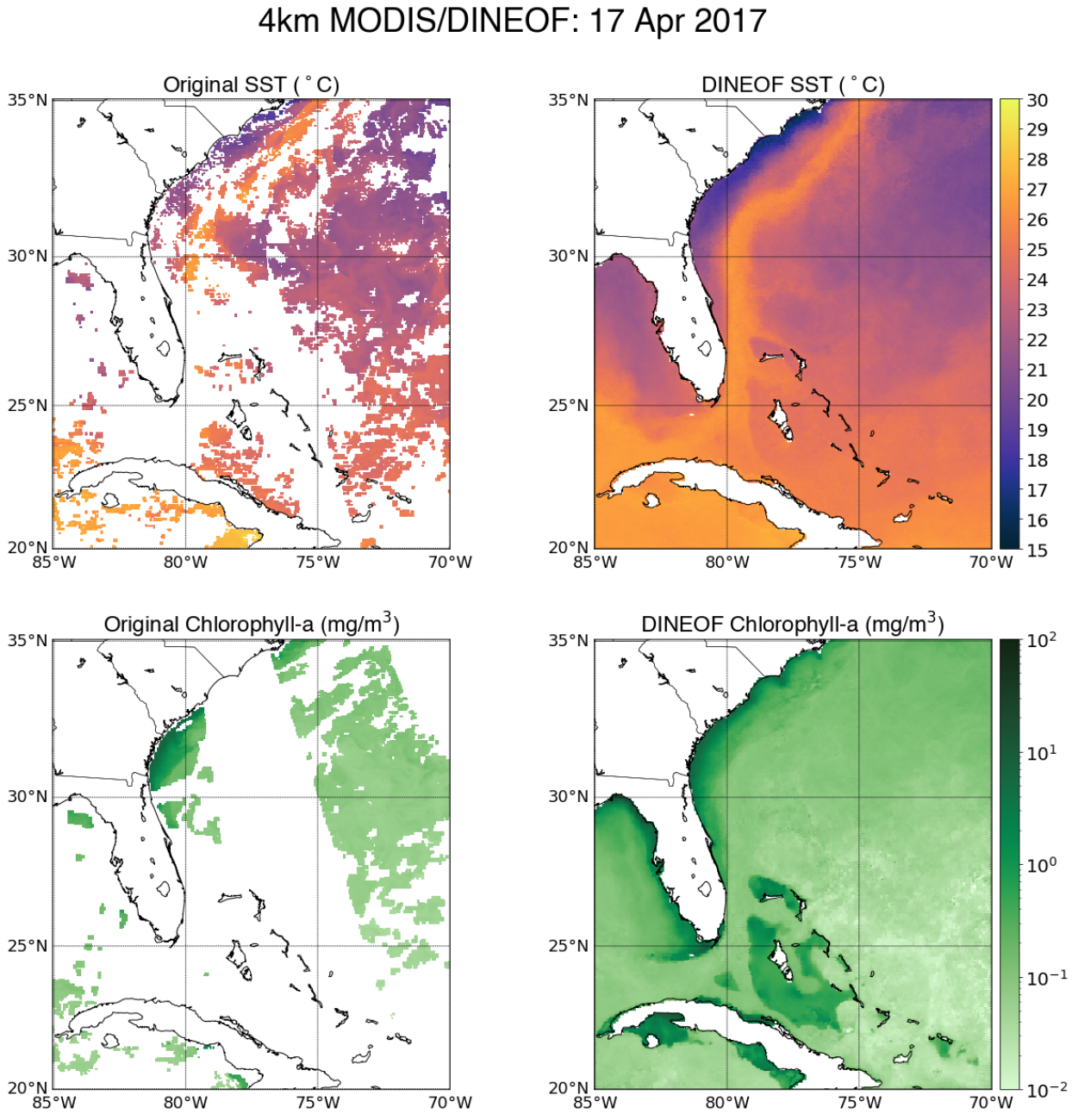
The remaining subplots should be fairly self-explanatory. The third (3) subplot, the original MODIS Chlorophyll-a data sits in the 2x2 subplot grid at location 3 (lower-left). Again, there's no colorbar for the left column of plots (subplots 1 and 3).

```
# Original MODIS Chlorophyll-a
plt.subplot(2,2,3)
map = Basemap(projection='merc',
    resolution='i',lat_0=((np.max(lat)-np.min(lat))/2),
    lon_0=((np.max(lon)-np.min(lon))/2),
    llcrnrlon=np.min(lon),llcrnrlat=np.min(lat),
    urcrnrlon=np.max(lon),urcrnrlat=np.max(lat))
map.drawcoastlines()
map.drawcountries()
map.drawstates()
map.pcolormesh(xlon,xlat,modis_chla[:,:],norm=LogNorm(vmin=0.01, vmax=100),\
               cmap=cmocean.cm.algae,latlon='true')
#cbar=map.colorbar(location='right',norm=LogNorm(vmin=0, vmax=100),ticks=[0,0.01,0.1,1,10,100])
map.drawparallels(parallels,labels=[1,0,0,0],fontsize=18)
map.drawmeridians(meridians,labels=[0,0,0,1],fontsize=18)
cbar.ax.tick_params(labelsize=20)
plt.title('Original Chlorophyll-a (mg/m$^3$)',fontsize=24,family='Helvetica')
```

The fourth (4) subplot, the DINEOF MODIS Chlorophyll-a data sits in the 2x2 subplot grid at location 4 (lower-right). Again, there's a colorbar for the right column of plots.

```
# Analyzed DINEOF Chlorophyll-a
plt.subplot(2,2,4)
map = Basemap(projection='merc',
    resolution='i',lat_0=((np.max(lat)-np.min(lat))/2),
    lon_0=((np.max(lon)-np.min(lon))/2),
    llcrnrlon=np.min(lon),llcrnrlat=np.min(lat),
    urcrnrlon=np.max(lon),urcrnrlat=np.max(lat))
map.drawcoastlines()
map.drawcountries()
map.drawstates()
map.pcolormesh(xlon,xlat,dineof_chla[:,:],norm=LogNorm(vmin=0.01, vmax=100),\
               cmap=cmocean.cm.algae,latlon='true')
cbar=map.colorbar(location='right',norm=LogNorm(vmin=0, vmax=100),ticks=[0,0.01,0.1,1,10,100])
map.drawparallels(parallels,labels=[1,0,0,0],fontsize=18)
map.drawmeridians(meridians,labels=[0,0,0,1],fontsize=18)
cbar.ax.tick_params(labelsize=20)
plt.title('DINEOF Chlorophyll-a (mg/m$^3$)',fontsize=24,family='Helvetica')
```

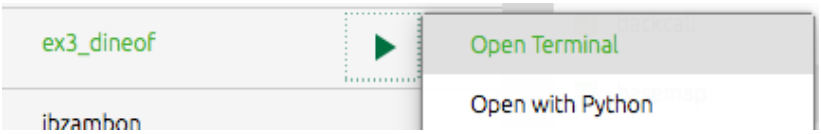
Your resultant plot should look like this. After a few more meticulous and/or pedantic changes, you'll have a publication-quality graphic that was produced in Python! Feel free to play around with the date to print the South Atlantic Bight SST and Chlorphyll from MODIS and my DINEOF reconstructions. Note that if you change seasons, the SSTs may be different, so you may need to update vmin and vmax to prevent the image from being washed out (e.g. in summer months vmin and vmax may be too low and vice versa for the winter months).



Sharing your Work!

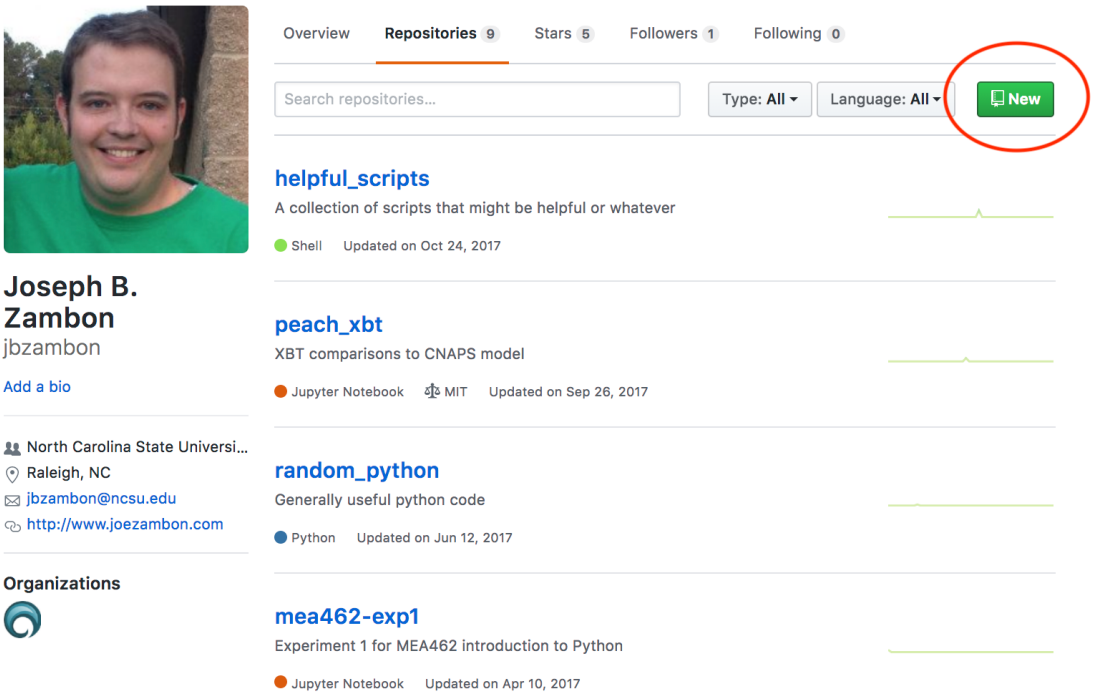
One of the most important aspects of academia is sharing your work with the world. Just like the publically available datasets we used, I can make the plotting code publically available. There are a number of great tools for global code-sharing available, I prefer GitHub. It has a learning curve and is outside the scope of this course, but there are lots of great online tutorials if you are interested (i.e. http://www.youtube.com/watch?v=0fKg7e37bQE). I'll quickly go through how to use GitHub to upload your code and download/run others' codes.

To share on GitHub, you must create an account, create a repository, and upload your code and Anaconda environment. To clone your environment, open your environment in Terminal as we did in Lesson 1, do "conda env export > environment.yml". The environment.yml file will allow future users of your code to import your environment and run your code, independent of their machine or OS!

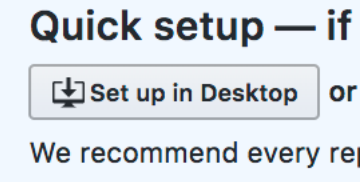


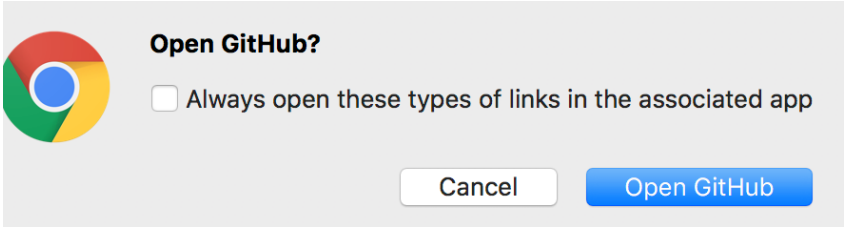

```
omgmac01:~ jbzambon$ /Users/jbzambon/.anaconda/navigator/a.tool ; exit;
(ex3_dineof) bash-3.2$ conda env export > environment.yml
```

Next save your .ipynb file, upload it and the environment.yml file to your Git repository. For simplicity, I'll be using Git Desktop for this demonstration. First, create a new Repository on GitHub.

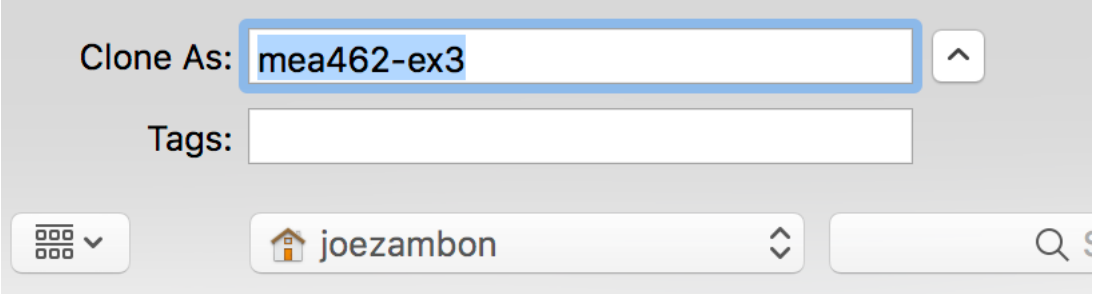


Name it whatever is appropriate, then click “Set up in Desktop” and “Open GitHub”.

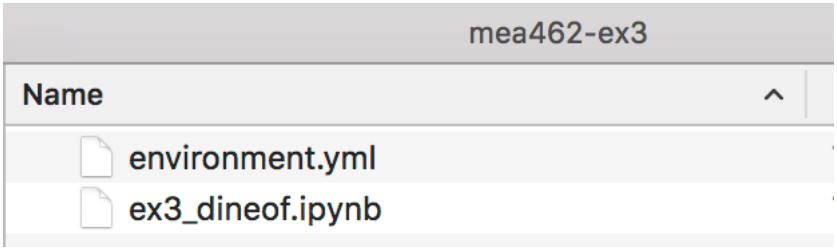


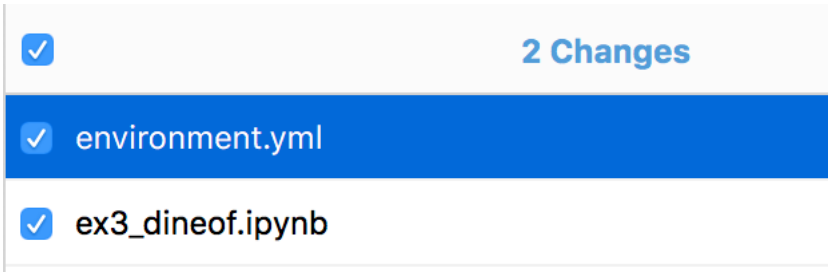


Save the directory to wherever is convenient for you to copy files over.

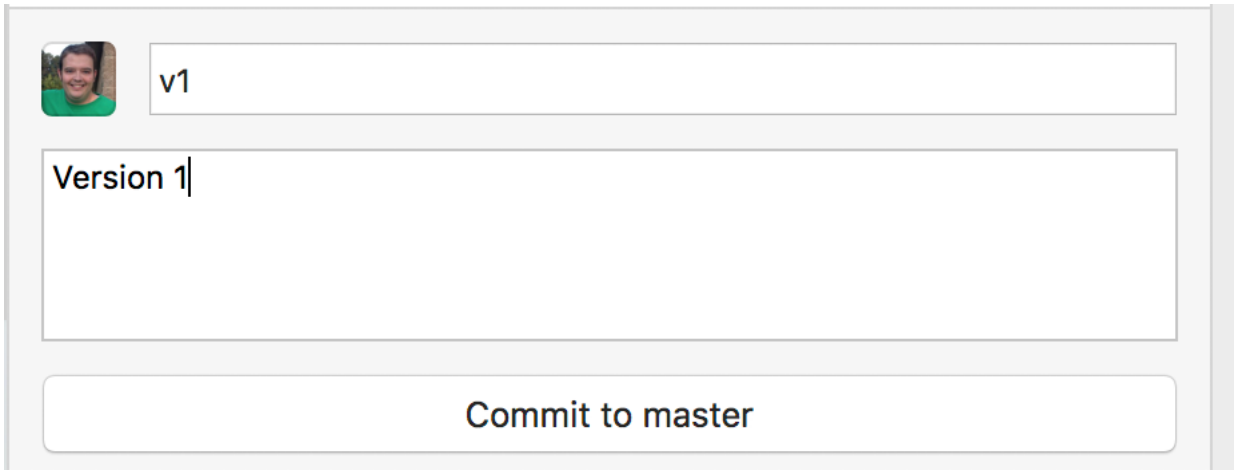


Now you have a folder you can simply drag and drop files into. Copy the files you want to be a part of the branch into this folder, then go back to the GitHub application. Notice these files are now able to be merged into the master branch.

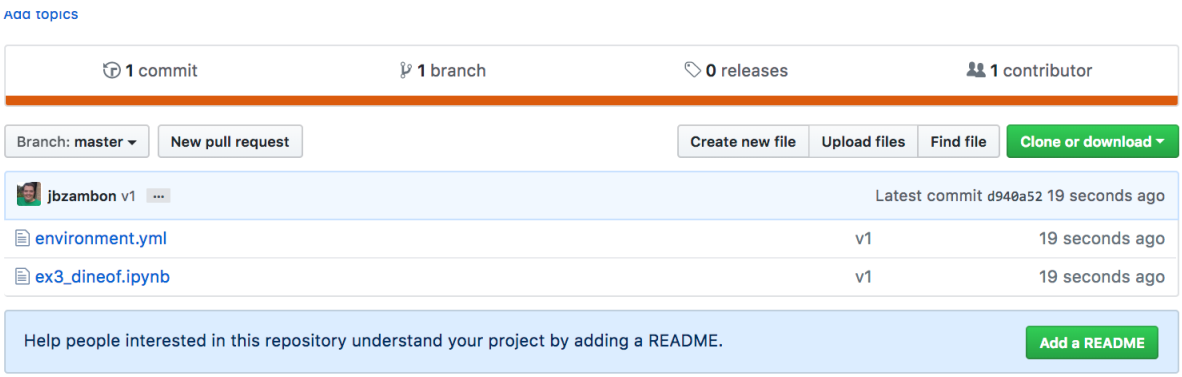




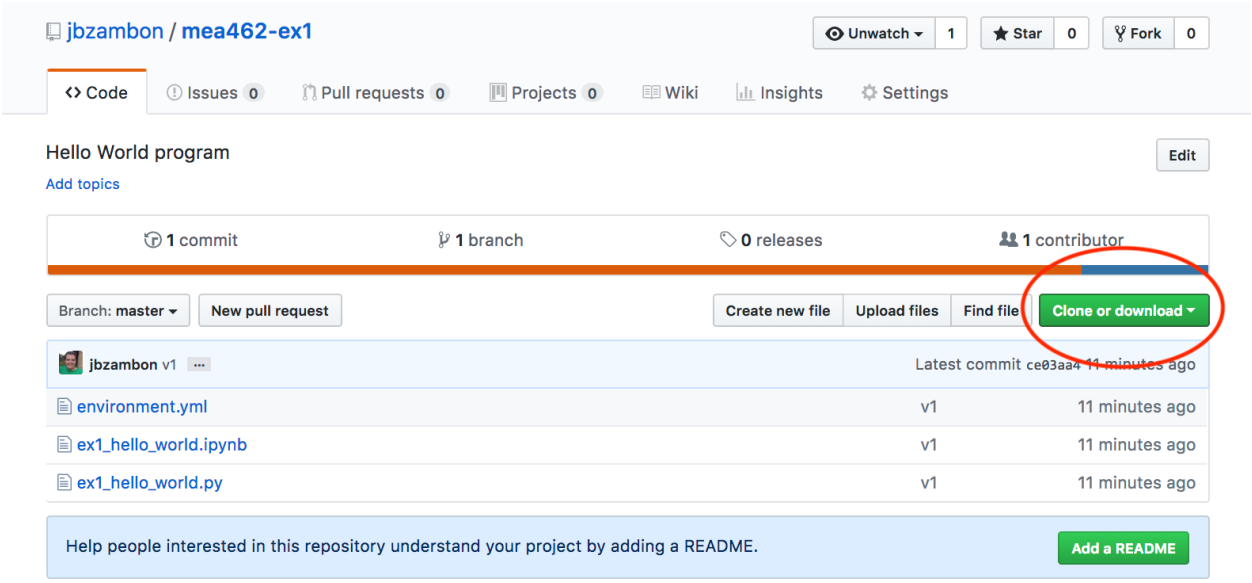
Commit the selected files to the master branch and then Publish.



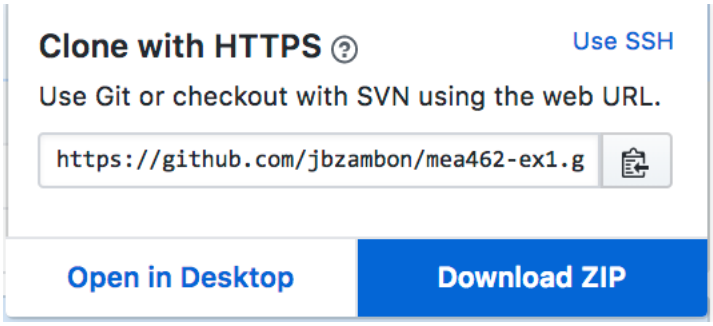
Now if you go back to your GitHub Repository page, you can Clone/Download the files included in your branch.

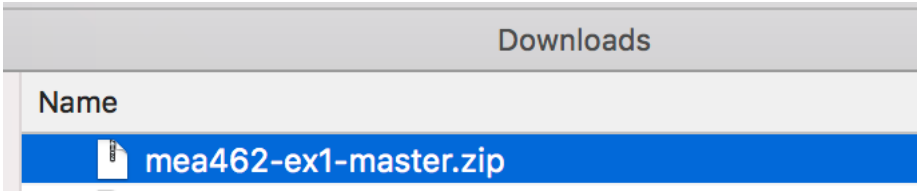


Once the files are downloaded, you can import the cloned environment, and run the iPython Notebook and/or Python code. Using our “Hello World” code as an example...

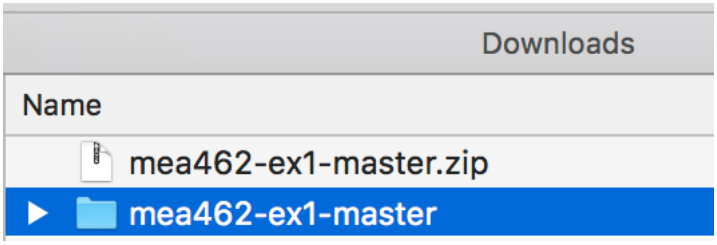


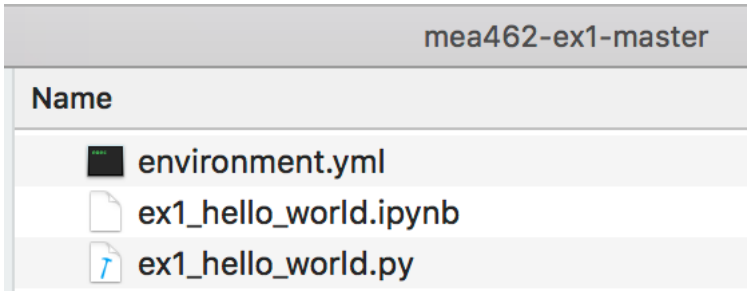
Download the branch as a zip file.



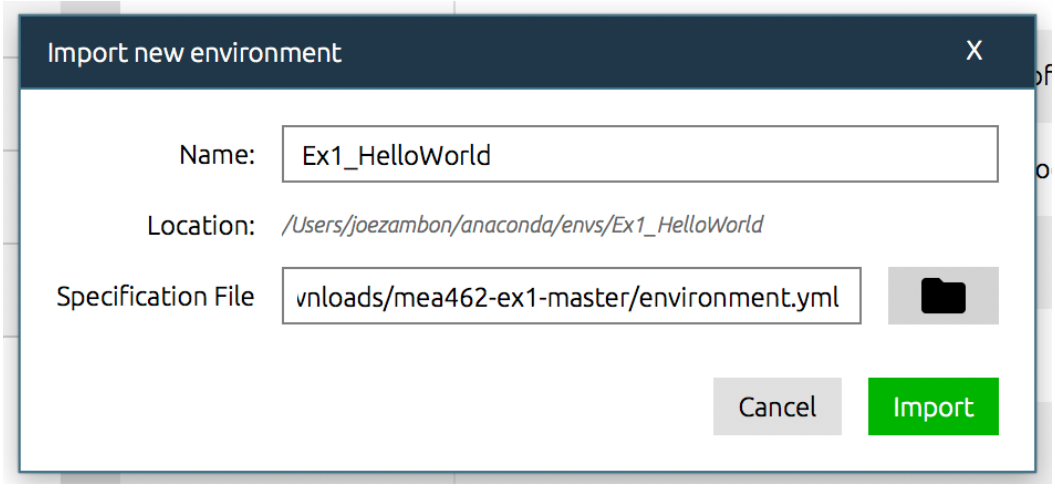


Unzip the file, open the master branch folder.





You now have access to the original environment that we wrote the code in, the iPython Notebook, and the Python program we worked on in Lesson 1. To run this code, first Import the Environment, then run the code from either iPython Notebook or Terminal as we did in Lesson 1.



That's it! Now you know how to export and import Python code using one of the world's most powerful code-sharing networks!

Oh, now that you've completed all lesson tutorials, I'll let you in on a secret. They've been posted to GitHub this entire time. Feel free to share! As a bonus there is some code in Lesson 3's py script to further your plotting enjoyment!

Feel free to check out some awesome GitHub pages to run oceanographic Python code.
https://github.com/jbzambon
https://github.com/rsignell-usgs
https://github.com/ocefpaf

Congrats on making it through these lessons in Python! You now are able to program and run a very powerful programming language. You also have access to shared codes and environments that allow you to access and manipulate oceanographic data.

If you keep this up, you'll be able to produce fantastic images in no time!