Python

More important basics: using shell commands, write your own module, basic input and output, dealing with errors using try-except.

system stuff

- You can issue system commands from python programs, to do things like ***making directories***.
- Here is a function for making directories. We'll go over some details of the parts of this function later in the PPT.
- Note that there are two kinds of inputs, or "arguments", that we feed to the function. The first "dirname" is required. The second "clean" is optional, with a default value of False. It is called a keyword argument, or "kwarg". You could also call the function using clean=True. Keyword arguments always come after required arguments in the function definition.

```
def make_dir(dirname, clean=False):
    # Make a directory if it does not exist.
    # Use clean=True to clobber the existing directory.
    import os, shutil # separate modules with a comma
    if clean == True:
        # remove the directory and any sub-directories
        shutil.rmtree(dirname, ignore_errors=True)
        # then create the new empty directory
        os.mkdir(dirname)
    else:
        try:
            # try to create the directory if it does not exist
            os.mkdir(dirname)
        except OSError:
            pass # assume OSError was raised because directory already exists
```

Background on system-oriented modules: os, sys, and shutil

- https://docs.python.org/3/library/os.html#module-os
  - This module provides a portable way of using operating system dependent functionality. If you just want to read or write a file see open(), if you want to manipulate paths, see the os.path module, and if you want to read all the lines in all the files on the command line see the fileinput module. For creating temporary files and directories see the tempfile module, and for high-level file and directory handling see the shutil module.
- https://docs.python.org/3/library/sys.html
  - This module provides access to some variables used or maintained by the interpreter and to functions that interact strongly with the interpreter. It is always available.
- https://docs.python.org/3/library/shutil.html
  - The shutil module offers a number of high-level operations on files and collections of files. In particular, functions are provided which support file copying and removal. For operations on individual files, see also the os module.

Write and use your own module

- If you have a function that is used by more than one program, like the make_dir() function from slide 2, then you can put it in your own module instead of having to copy and paste it into each program.
- ***The module is just a python program***: a text file called [my_module].py, that contains the function definitions. You can do module imports at the top of the file, and they are available to all the functions in the module. You can also define variables, and they are available to all the functions in the module, and to any program that uses your module.

How to access a module you put in a different directory, and using "reload"

- You do this by adding that directory to your path in a python program, and then you can load it using the usual "import [module]" command.
- Use ***relative*** paths to point to where the module is. Using relative paths helps code to work on different machines, as long as the relative file structure is the same.
- In this example I have a module called **Lfun.py in a directory called alpha** that is at the same level as the current directory. The command below combines two steps. First is creates the full path name using the "abspath" command, and then it appends that path to the python search path using "path.append".
- `sys.path.append(os.path.abspath('../alpha'))`
- Then I can simply load my module with the command:
- `import Lfun # no .py needed`
- If I am in the process of editing the methods in Lfun.py, I need to make sure my python code gets those updates. To do this you "reload" using these commands:
- `from importlib import reload`
- `reload(Lfun)`

Module example

- Go to the class web site, go and download:
  - test_my_module.py
  - my_module.py
- and place them in [your folder] and [your folder]/shared/
- Then run test_my_module from the ipython command line.
- *Read the code to see what it is doing.*

Input and output

- To get python working for you, you need to be able to read in a variety of files: text files, Excel spreadsheets, .mat files, NetCDF files, XML data from remote servers, csv files, and pickle files saved by python.
- Each of these tasks uses somewhat different tools. Let's start simple with a text file.

Input-output example: text file

- Go to the class web site and download:
    - test_input_output.py
    - 2017-01-0118.ctd
- and place them in [your folder] and [your data folder]
- edit this line in the program:
- `myplace = 'pmec' # *** YOU NEED TO EDIT THIS ***`
- Then run test_input_output from the ipython command line.
- *Read the code to see what it is doing.*

try-except

- Sometimes python "throws" or "raises" errors and stops your program. By using the try-except structure you can handle specific errors.
- General advice: put a try-except trap around the ***smallest*** piece of code you need to, and be ***specific about what errors*** you are handling:
- Example from our make_dir() method:

```
try:
    # try to create the directory if it does not exist
    os.mkdir(dirname)
except OSError:
    pass # assume OSError was raised because directory already exists
```

- Here the error name we are trying to handle, "OSError", came from looking at the screen output in ipython when we tried to use os_mkdir() to create a directory that already existed. Since we don't want our code to stop there, we just tell it to "pass" (i.e. keep going) if OSError is raised. The directory already exists so our job is done.
- In general you will find that python errors are always typed with the same capitalization format: "NameError".

What was the error name?

- To catch unknown errors and figure out what they are in a try-except structure use code like:

```
try:
    ...
except Exception as e:
    print(e)
```