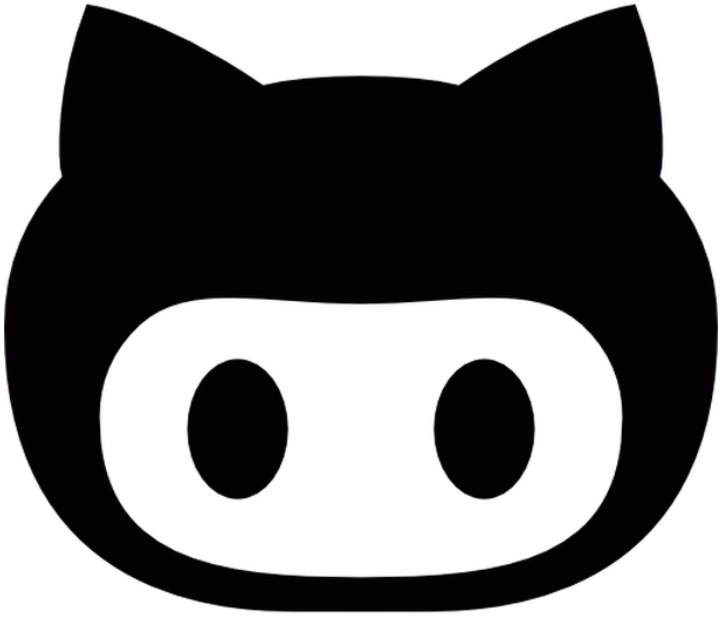
Writing a preprocessing script in python

Open up a fresh blank script in your Spyder IDE and save it as prepro.py

For this tutorial you will input all code into this script and periodically commit your changes and push them to github when you see the logo



First thing we will always do is load our modules

In [ ]:
```
import glob
import os
import pdb
import subprocess
import argparse
import datetime
import shutil
```

Often times we won't know all the modules we want to import right off the bat but I like to make sure that as I am scripting I always put my modules at the top.

This allows others who may use my script to make sure they have all the necessary tools

Now lets start by building a function that will hold all the commands we want to execute

```
def prepro():
    #do something cool
```

We will make a function that will hold all of our global variables and our above function

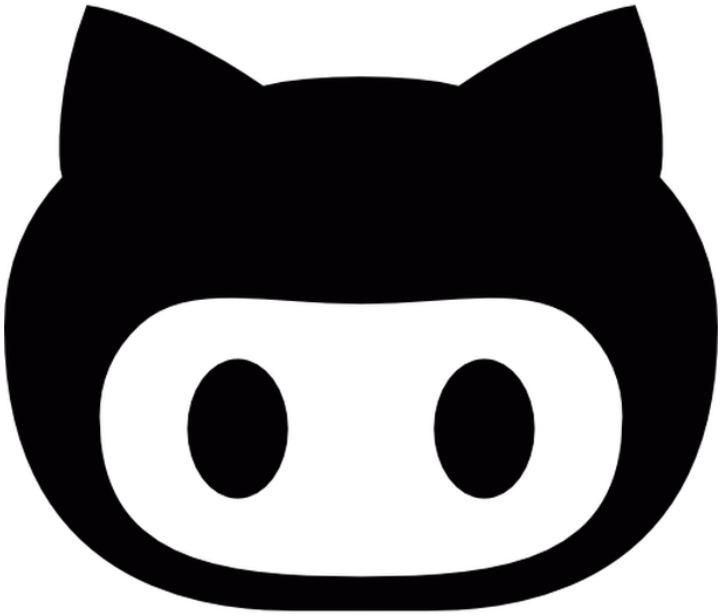
I personally like to call this main

```
def main():
    prepro()
```

Finally we have our two functions and lastly we will call our main( ) which will execute both our global variables and our command function

In [ ]:
```
def prepro(basedir):
    #Do something cool
    print('Hello data in the path '+basedir)
def main():
    #load in all the global variables prepro needs, right now this is ju
st the path to the data
    basedir='/Users/gracer/Desktop/data'
    prepro(basedir) #call prepro to do cool things
```

In [ ]:
```
main()#call main to execute all our globals then run our prepro function
```



What do we want the function to accomplish:

1. skull stripping
2. motion correction
    - creating motion regressors
    - creating framewise displacement regressor
    - a nice easy to read PDF/html?
3. re-orient?
4. trim extra TRs?

Let's fill in our main( ) function first with the global variables we will need.

In [ ]:
```
def main():
    basedir='/Users/gracer/Desktop/data'
    prepro()
```

Anything you define in the main( ) function has to become an argument in the prepro( ) function.

In [ ]:
```
def prepro(basedir):
    print('Hello data in the path '+basedir)
def main():
    basedir='/Users/gracer/Desktop/data'
    prepro(basedir)
```

Let's start with skull stripping using fsl's BET function.

This is a linux based command so we are going to need to use a module to python to understand it.

Normally at the command line we would run something like this:

```
bet input output [options]
```

In python we can use the os module to run linux commands

```
os.system(bet input output -F)
```

In [ ]:
```
#try running this

print(os.system('echo $FSLDIR'))


#now look at the terminal you launched your jupyter notebook from
```

next lets take a close look at the input and output we need. What will the input look like? What do we want the output to look like?

In [ ]:
```
input='/Users/gracer/Desktop/data/<subject number>/func/<nifiti_file>'
```

Each time we run this command the only things we really need to change are the subject number and the name of the nifti file

Our subject numbers and nifti files use a predictable pattern!

We can use the glob module to find everything with a similar pattern.

Here we are going to use a wildcard character (*) to represent the portions of the subject number that differ.

In [ ]:
```
input=glob.glob('/Users/gracer/Desktop/data/sub-*/func/sub-*.nii.gz')
print(input[0:10])
```

glob has created a list with everything matching our pattern criteria. We can use any of python's list comprehension tools to further explore the list

In [ ]:
```
len(input)
```

In [ ]:
```
input[-1]
```

We can also take any element from the list and make it a string. By making a string we can grab IDs or other parts of interest

In [ ]:
```
#x=input[1]
#print('my path is '+x)
#y=x.split('/')
#print (y)
#whatcomp=y[2]
#sub=y[5]
#print sub

sub=input[1].split('/')[5]
print(sub)
```

Let's make this look a little nicer

In [ ]:
```
sub=input[1].split('/')[5]
print(sub)
```

Now we have the subject number but it looks like we have multiple tasks. How can we split an element from the list to get the task information and the subject information?

In [ ]:
```
subtask=input[1].split('/')[7].split('.')[0]
#subtask=subtask.strip('.nii.gz')
print(subtask)
```

In [ ]:
```
output=subtask+'_brain'
print(output)
```

Lets go back to our bet command in the os wrapper. We now have all the elements we need to execute it.

In [ ]:
```
os.system('bet' x output '-F')
```

This is a problem, we have our input defined, but it looks like os.system is expecting a string argument.

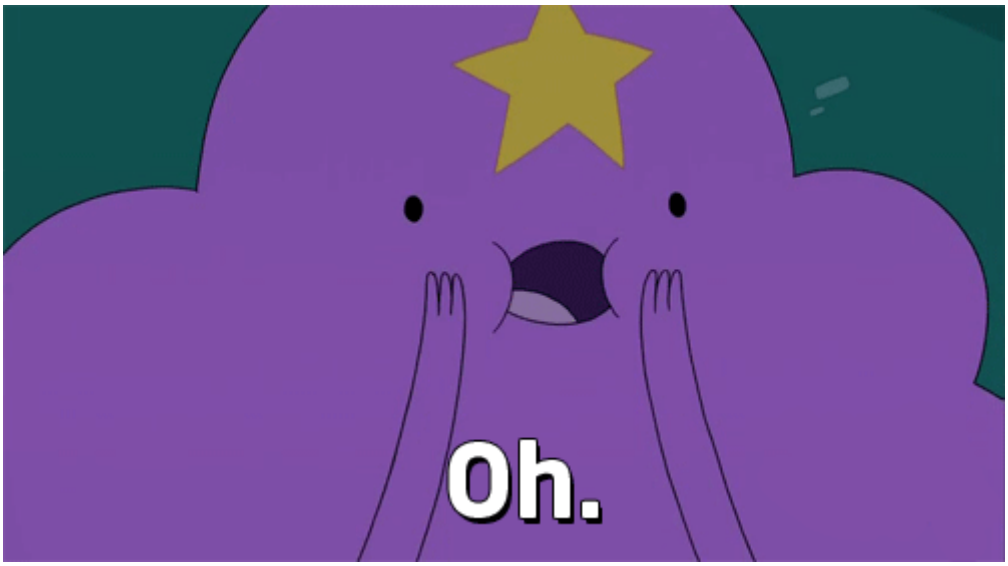
We need to use another wildcare to pass our variables as strings!

In [ ]:
```
#os.system('bet' x output '-F')
#print(x)
#print(output)
os.system('bet %s %s -F'%(x, output))
```

The %s is a placeholder for string variable

The **%** lets python know to look to the % sign outside the string for the variable of interest. We could also use this to pass **integers and floats using %i and %f** respectively.

Now we have the ability to run bet through python on one subject.... but what about all the other scans.... ?



In [ ]:
```
input=glob.glob('/Users/gracer/Desktop/data/sub-*/func/sub-*.nii.gz')
#print input
```

this is a little long to type each time,

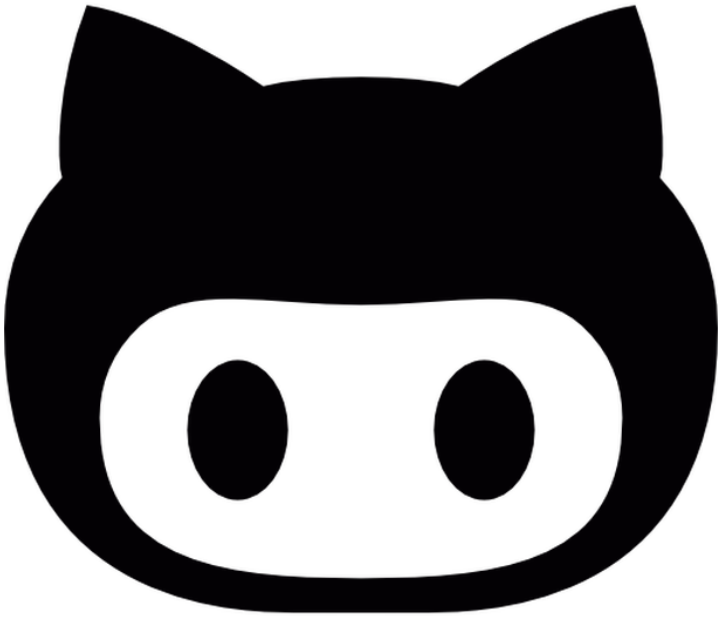
and it is really easy to mess up the / formating

os.path.join( ) is super useful to quickly define paths. It will format strings into paths and allows us to use the %s

In [ ]:
```
#input=glob.glob('/Users/gracer/Desktop/data/sub-*/func/sub-*.nii.gz')
basedir='/Users/gracer/Desktop/data'
path=os.path.join(basedir,'sub-*','func','sub-*.nii.gz')
#print(path)
#input=glob.glob(path)
#len(input[1:5])
os.path.exists(basedir)
```

Let's put this altogether into our function prepro( ) with a loop

In [ ]:
```
def prepro(basedir):
    for item in glob.glob(os.path.join(basedir,'sub-*','func','sub-*.ni
i.gz')):
        input=item
        output_path=item.strip('.nii.gz')
        output=output_path+('_brain')
        os.system("/usr/local/fsl/bin/bet %s %s -F"%(input, output))
        pdb.set_trace()
def main():
    basedir='/Users/gracer/Desktop/data'
    prepro(basedir)
```



In [ ]:
```
main()
#prepro(os.path.join('Users','gracer','Desktop','data'))
```

Concept Check: Why do we have to type main( ) after def prepro( ) and def main( )? Why do we have 2 functions?

Ta Da!!! You have your first preprocessing script!

But wait... how do you make sure you don't end up running the same function on the same data over and over?

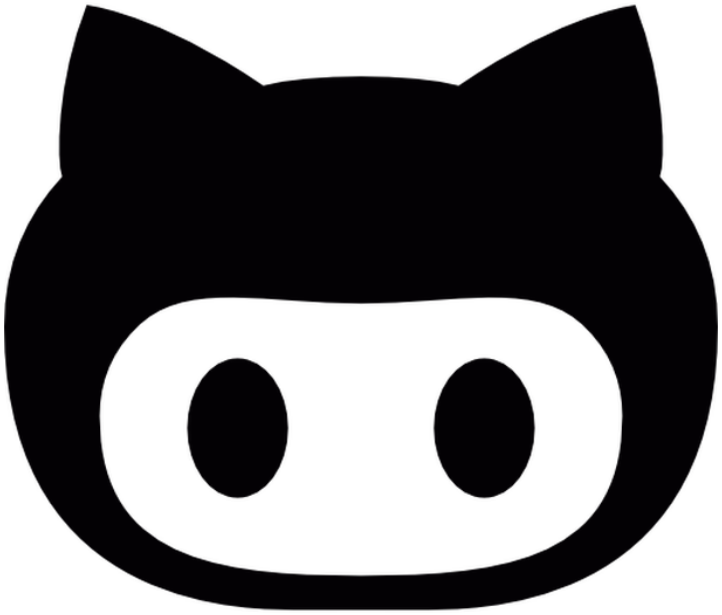
Let's write in a check statement

We can use os.path.exists( ) to check if we have already run BET, and tell our function to skip that subject

This is useful if you have two people preprocessing data, or if something happens (aka your computer runs out of power)

In [ ]:
```
def prepro(basedir):
    for item in glob.glob(os.path.join(basedir,'sub-*','func','sub-*.ni
i.gz')):
        input=item
        output_path=item.strip('.nii.gz')
        output=output_path+'_brain.nii.gz'
        print(output)
        pdb.set_trace()
        if os.path.exists(output):
            print(output_path+' is already stripped')
        else:
            os.system("/usr/local/fsl/bin/bet %s %s -F"%(input, output))
        #pdb.set_trace()
def main():
    basedir='/Users/gracer/Desktop/data'
    prepro(basedir)
```

In [ ]:
```
main()
#%clear
```



Now that we know how:

1. To make a set of functions
2. Set our global variables
3. Wrap our linux commands
4. Use glob to get all our subjects through wildcard matching
5. Loop through our list of subjects (from glob)
6. Use string comprehension to format file names
7. Use if/else loops to check for existing data

Try writing a function to skull strip a T1w scan

In [ ]:
```
def prepro(basedir, args, arglist, DATA):
#bet
    if args.STRIP==True:
        for sub in DATA:
            for nifti in glob.glob(os.path.join(sub,'func','sub-*_task-%
s_bold.nii.gz')%(arglist['TASK'])):
                input=item
                output_path=item.strip('.nii.gz')
                output=output_path+'_brain.nii.gz'
                print(output)

                if os.path.exists(output):
                    print(output_path+' is already stripped')
                else:
                    os.system("/usr/local/fsl/bin/bet %s %s -F"%(input,
output))
#bet rage
    if args.RAGE==True:
        print("starting bet rage")
        for sub in DATA:
            for input in glob.glob(os.path.join(sub,'anat','*T1w.nii.gz'
)):
                output=input.strip('.nii.gz')
                print(output)
                if os.path.exists(output+'_brain.nii.gz'):
                    print(output+' exists, skipping')
                else:
                    BET_OUTPUT=output+'_defaced'
                    x=("/usr/local/fsl/bin/bet %s %s -R"%(input, BET_OUT
PUT))
                    print(x)
                    os.system(x)

def main(DATA):
    basedir='/Users/gracer/Desktop/data'
    parser=argparse.ArgumentParser(description='preprocessing')
    parser.add_argument('-task',dest='TASK',
                        default=False, help='which task are we running o
n?')
    parser.add_argument('-bet',dest='STRIP',action='store_true',
                        default=False, help='bet via fsl using defaults
 for functional images')
    parser.add_argument('-betrage',dest='RAGE',action='store_true',
                        default=False, help='bet via fsl using robust es
timation for anatomical images')
    args = parser.parse_args()
    arglist={}
    for a in args._get_kwargs():
        arglist[a[0]]=a[1]
    print(arglist)
    prepro(basedir, args, arglist, DATA)
```

Whoa what is all this extra stuff???

Let's take a look at all the new functionality that is possible

argparse

This will allow us to give user defined arguments

Why would I make my life more complicated?

- This is a good option for if you have multiple people working on preprocessing
- You want to make one giant preprocessing script and have control over which commands are run

  Instead of a bunch of smaller functions that infividually have to be called

In [ ]:
```
def main(DATA):
    basedir='/Users/gracer/Desktop/data'
    parser=argparse.ArgumentParser(description='preprocessing')
    parser.add_argument('-task',dest='TASK',
                        default=False, help='which task are we running o
n?')
    parser.add_argument('-bet',dest='STRIP',action='store_true',
                        default=False, help='bet via fsl using defaults
 for functional images')
    parser.add_argument('-betrage',dest='RAGE',action='store_true',
                        default=False, help='bet via fsl using robust es
timation for anatomical images')
    args = parser.parse_args()
    arglist={}
    for a in args._get_kwargs():
        arglist[a[0]]=a[1]
    print(arglist)
    prepro(basedir, args, arglist, DATA)
```

- Define the parser, and give it a nice description

In [ ]:
```
parser=argparse.ArgumentParser(description='preprocessing')
```

- Start thinking about arguments, these should be things that either trigger functionality or are global variables

In [ ]:
```
parser.add_argument('-task',dest='TASK',
                        default=False, help='which task are we running o
n?')
parser.add_argument('-betrage',dest='RAGE',action='store_true',
                        default=False, help='bet via fsl using robust es
timation for anatomical images')
```

1. The first part is the flag, it will be entered with our script at the commandline
2. Next is the destination, that is what the argparser is saving our input as
3. We set the default to false, meaning you have to enter something for it to be true
4. It is always nice to have a help especially if you are sharing your code!
5. Action will store your variable, this is nice for instances where the presences of the flag indicates true

- Put all our arguments in an variable called args

In [ ]:
```
args = parser.parse_args()
```

- We could keep our arguments here and call them like this:

In [ ]:
```
args.RAGE==True
```

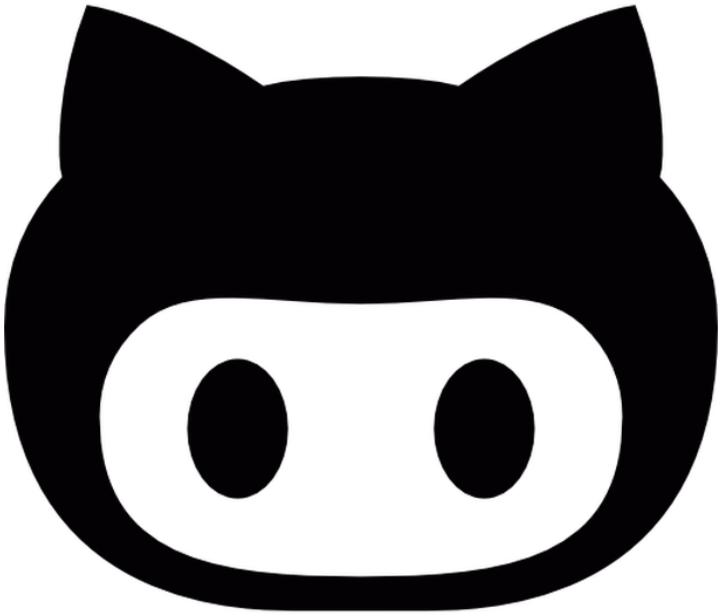
- Personally I like putting them in a dictionary!

In [ ]:
```
arglist={}
    for a in args._get_kwargs():
        arglist[a[0]]=a[1]
```

- Then you can call them in a dictionary form like this:

In [ ]:
```
for nifti in glob.glob(os.path.join(sub,'func','sub-*_task-%s_bold.nii.g
z')%(arglist['TASK'])):
```

- The key is 'TASK' and python will understand that to be what ever you entered into your argument list



What about motion correction and motion QA???

In [ ]:
```
def prepro(basedir, args, arglist, outhtml, out_bad_bold_list,DATA):
#def better(args,arglist,basedir):
    #bet
    if args.STRIP==True:
        print("starting bet")

        for sub in DATA:
            for nifti in glob.glob(os.path.join(sub,'func','sub-*_task-%
s_bold.nii.gz')%(arglist['TASK'])):
                input=nifti
                output=nifti.strip('.nii.gz')
                if os.path.exists(output+'_brain.nii.gz'):
                    print(output+' exists, skipping')
                else:
                    BET_OUTPUT=output+'_brain'
                    x=("/usr/local/fsl/bin/bet %s %s -F"%(input, BET_OUT
PUT))
                    os.system(x)


#bet rage
    if args.RAGE==True:
        print("starting bet rage")
        for sub in DATA:
            for input in glob.glob(os.path.join(sub,'anat','*T1w.nii.gz'
)):
                output=input.strip('.nii.gz')
                print(output)
```

```
                if os.path.exists(output+'_brain.nii.gz'):
                    print(output+' exists, skipping')
                else:
                    BET_OUTPUT=output+'_defaced'
                    x=("/usr/local/fsl/bin/bet %s %s -R"%(input, BET_OUT
PUT))
                    print(x)
                    os.system(x)



#motion correction
    if args.MOCO==False:
        print("No motion correction performed")
    else:
        print("starting motion correction")
        for sub in DATA:
            for dir in glob.glob(os.path.join(sub,'func')):
                if not os.path.exists(os.path.join(basedir,dir,'motion_a
ssessment')):
                    os.makedirs(os.path.join(basedir,dir,'motion_assessm
ent'))
                os.chdir(os.path.join(basedir, dir))
                for input in glob.glob('*brain.nii.gz'):
                    output=input.split('.')[0]
                    print(output)
                    if output.endswith('mcf'):
                        print(output+' exists, skipping')
                    else:
                        os.system("mcflirt -in %s -plots"%(output))
                        os.system("fsl_motion_outliers -i %s -o motion_a
ssessment/%s_confound.txt --fd --thresh=%s -p motion_assessment/fd_plot
 -v > motion_assessment/%s_outlier_output.txt"%(output,output,arglist['M
OCO'],output))
                        os.system("cat motion_assessment/%s_outlier_outp
ut.txt >> %s"%(output,outhtml))
                        plotz=os.path.join(basedir,dir,'motion_assessmen
t','fd_plot.png')
                        os.system("echo '<p>=============<p>FD plot %s <
br><IMG BORDER=0 SRC=%s WIDTH=%s></BODY></HTML>' >> %s"%(output,plotz,'1
00%', outhtml))

                        if os.path.isfile("motion_assessment/%s_confoun
d.txt"%(output))==False:
                            os.system("touch motion_assessment/%s_confou
nd.txt"%(output))

                        check = subprocess.check_output("grep -o 1 motio
n_assessment/%s_confound.txt | wc -l"%(output), shell=True)
                        num_scrub = [int(s) for s in check.split() if s.
isdigit()]

                        if num_scrub[0]>45:
                            with open(out_bad_bold_list, "a") as myfile:
                                myfile.write("%s\n"%(output))
                            myfile.close()
```

```
                        if os.path.exists("%s_mcf.par"%(output)):
                            if os.path.exists(os.path.join(basedir,dir,
'motion_assessment',"%s_mcf.par"%(output))):
                                os.remove(os.path.join(basedir,dir,
'motion_assessment',"%s_mcf.par"%(output)))

                        shutil.move("%s_mcf.par"%(output),os.path.join(b
asedir,dir,'motion_assessment'))
                        rawfile = open(os.path.join(os.path.join(basedir
,dir,'motion_assessment','%s_mcf.par'%(output))), 'r')
                        table = [line.rstrip().split() for line in rawfi
le.readlines()]
                        for i in range(6):
                            newtable = ([[line[i]] for line in table])
                            f=open(os.path.join(basedir,dir,'motion_asse
ssment','%s_motcor%i.txt'%(output,i)),'w')
                            for item in newtable:
                                neat=item[0]
                                f.write(str(neat)+'\n')
                            f.close()
```

In [ ]:
```
def main(DATA):
    basedir='/Users/gracer/Desktop/data'
    writedir='/Users/gracer/Desktop/data'

    datestamp=datetime.datetime.now().strftime("%Y-%m-%d-%H_%M_%S")
    outhtml = os.path.join(writedir,'bold_motion_QA_%s.html'%(datestamp
))
    out_bad_bold_list = os.path.join(writedir,'lose_gt_45_vol_scrub_%s.t
xt'%(datestamp))

    parser=argparse.ArgumentParser(description='preprocessing')
    parser.add_argument('-task',dest='TASK',
                        default=False, help='which task are we running o
n?')
    parser.add_argument('-bet',dest='STRIP',action='store_true',
                        default=False, help='bet via fsl using defaults
 for functional images')
    parser.add_argument('-betrage',dest='RAGE',action='store_true',
                        default=False, help='bet via fsl using robust es
timation for anatomical images')
    parser.add_argument('-moco',dest='MOCO',
                        default=False, help='this is using fsl_motion_ou
tliers to preform motion correction and generate a confounds.txt as well
 as DVARS')
    args = parser.parse_args()
    arglist={}
    for a in args._get_kwargs():
        arglist[a[0]]=a[1]
    print(arglist)
    prepro(basedir, args, arglist, outhtml, out_bad_bold_list,DATA)
```

Yikes! This looks like a ton of new information!

We actually are only adding a couple new elements

We are going to:

1. Make a directory (folder)
2. Save text output to a useful location
3. Save our output plots to a single html
4. Check our text output files for subjects for excessive motion
5. Create our motion regressors
6. Move everything into our directory we made (#1)

1. Make a directory (folder)

In [ ]:
```
            for dir in glob.glob(os.path.join(sub,'func')):
                if not os.path.exists(os.path.join(basedir,dir,'motion_a
ssessment')):
                    os.makedirs(os.path.join(basedir,dir,'motion_assessm
ent'))
                os.chdir(os.path.join(basedir, dir))
```

Again we are using glob to get all our subjects. Next we are checking that a path exists (and therefore our directory) with the command

In [ ]:
```
os.path.exists()
```

If it doesn't exist that means we need to make it. There are a lot of ways to make directories, but I like:

In [ ]:
```
os.makedirs()
```

Finally, we are going to change directory into the subject we are on using:

In [ ]:
```
os.chdir(os.path.join(basedir, dir))
```

Notice the indentation of the code above. The last line is not in the if statement loop.

This is important because we want to change directory regardless whether the if statement is true or not.

This should look familiar

In [ ]:
```
for input in glob.glob('*brain.nii.gz'):
    output=input.split('.')[0]
    print(output)
    if os.path.exists(os.path.join(dir,output+'_mcf.nii.gz'))==True:
        print(output+' exists, skipping')
    else:
        os.system("mcflirt -in %s -plots"%(output))
        os.system("fsl_motion_outliers -i %s -o motion_assessment/%s_con
found.txt --fd --thresh=%s -p motion_assessment/fd_plot -v > motion_asse
ssment/%s_outlier_output.txt"%(output,output,arglist['MOCO'],output))
```

- Here we are getting all the files we previously skull stripped and using our familiar friend...
- os.system() to run a linux command! This time mcflirt and fsl_motion_outliers
- We have included a if/else statement to check if we have already done this (why waste our own time?)
- We also are piping the output of the fsl_motion_outliers command to a text file

2. Save text output to a useful location

In [ ]:
```
os.system("cat motion_assessment/%s_outlier_output.txt >> %s"%(output,ou
thtml))
```

In [ ]:
```
* This is very similar to above, again we are using os.system to execute
 a linux redirect
* This time we are redirecting the text from the output file we made to
a html
* Why put this all in an html?
   * html is useful for sharing the output
   * We could embed it into a fancy shmancy jupyter notebook
   * We could make a pdf and send it to our PI
   * We could print it and wall paper our walls with QA reports!
#### Concept check: where is the variable outhtml defined?
```

3. Save our output plots to a single html

In [ ]:
```
plotz=os.path.join(basedir,dir,'motion_assessment','fd_plot.png')
os.system("echo '<p>==============<p>FD plot %s <br><IMG BORDER=0 SRC=%s
 WIDTH=%s></BODY></HTML>' >> %s"%(output,plotz,'100%', outhtml))
```

Here we are doing two things

1. We are creating a variable called plotz, which is the path to our fd_plot.png
2. We are formatting and entering the fd_plot.png into our html using os.system and linux commands to make sure it looks correct in the html

Concept check: Think about the structure of the loop we are building, what is directly above the plot in our html?

4. Check our text output files for subjects for excessive motion

In [ ]:
```
if os.path.isfile("motion_assessment/%s_confound.txt"%(output))==False:
    os.system("touch motion_assessment/%s_confound.txt"%(output))

check = subprocess.check_output("grep -o 1 motion_assessment/%s_confoun
d.txt | wc -l"%(output), shell=True)
num_scrub = [int(s) for s in check.split() if s.isdigit()]

if num_scrub[0]>45:
    with open(out_bad_bold_list, "a") as myfile:
        myfile.write("%s\n"%(output))
    myfile.close()
```

We are using our favorite if/else statement to check that the confound.txt exists.

- But what about perfect subjects? They won't have a confound.txt??? HALP
- This is kinda great problem, and easy to fix, we are going to use

  ```
  os.system("touch motion_assessment/%s_confound.txt"%(output))
  ```

  to create a blank text file with the correct name for our 1st level feats to find

Most of our particpants aren't perfectly still, so we want to create a check variable:

- We are using a new module called subprocess
- This operates very similar to os.system, but we can define the type of shell and use the additional check_output functionality to save our output to a variable
- So here we are searching for our confound.txt using grep, then we are counting the lines in the file (wc -l)

In [ ]:
```
check = subprocess.check_output("grep -o 1 motion_assessment/%s_confoun
d.txt | wc -l"%(output), shell=True)
num_scrub = [int(s) for s in check.split() if s.isdigit()]
```

Next we are creating a num_scrub variable:

- This is a weird looking loop!
- We could have also written it as:

In [ ]:
```
for s in check.split():
    if s.isdigit:
        num_scrub=int(s)
```

- Basically we want to remove white space with split, check if it is a digit, and make it an integer if it is

Great we have an integer now we can check if that integer is bigger or smaller than our "bad" criteria

- Our final loop compares our num_scrub to a criteria we set (45)
- If num_scrub is bigger we want to note it so we are going to open a file

In [ ]:
```
with open(out_bad_bold_list, "a") as myfile:
```

- We are going to write the subject information to the file

In [ ]:
```
  myfile.write("%s\n"%(output))
```

- Then we are going to close the file

In [ ]:
```
  myfile.close()
```

Important note: make sure you close the file outside the loop! Otherwise on the second loop it will try to write to closed file! Or it will try to open an already open file. Either way you won't be happy with the error.

Bonus points: what is the \n when we are writing the output to file? Why include this?

5. Move everything into our directory we made (#1)

In [ ]:
```
if os.path.exists("%s_mcf.par"%(output)):
    if os.path.exists(os.path.join(basedir,dir,'motion_assessment',"%s_m
cf.par"%(output))):
        os.remove(os.path.join(basedir,dir,'motion_assessment',"%s_mcf.p
ar"%(output)))
shutil.move("%s_mcf.par"%(output),os.path.join(basedir,dir,'motion_asses
sment'))
```

Since we ran MCFLIRT using the -plot flag we should have a .par file

- We are checking that file exists, seeing if we already moved, if we already did we are deleting the old one

In [ ]:
```
if os.path.exists("%s_mcf.par"%(output)):
    if os.path.exists(os.path.join(basedir,dir,'motion_assessment',"%s_m
cf.par"%(output))):
        os.remove(os.path.join(basedir,dir,'motion_assessment',"%s_mcf.p
ar"%(output)))
```

We are moving the .par to our motion_assessment directory using the shutil command

- It is always a good idea to check that the file doesn't already exist
- Shutil won't overwrite a file instead it will error out
- Shutil uses the following formula

In [ ]:
```
shutil.move(path_file_to_move, path_to_new_file_location)
```

Bonus points: How could we use shutil.move to rename a file?

6. Create our motion regressors

Note: If you are planning on using MCFLIRT in FEAT you don't need this, but it is fun to learn right???

In [ ]:
```
rawfile = open(os.path.join(os.path.join(basedir,dir,'motion_assessment'
,'%s_mcf.par'%(output))), 'r')
table = [line.rstrip().split() for line in rawfile.readlines()]
for i in range(6):
    newtable = ([[line[i]] for line in table])
    f=open(os.path.join(basedir,dir,'motion_assessment','%s_motcor%i.tx
t'%(output,i)),'w')
    for item in newtable:
        neat=item[0]
        f.write(str(neat)+'\n')
    f.close()
```

We are going to open our .par file as rawfile

Gah! Another weird looking loop!

In [ ]:
```
for line in rawfile.readlines():
    table=line.rstrip().split()
```

- We are reading in the rawfile line by line (readlines)
- We are removing the whitespace characters on the right (rstrip) and splitting it by white space (this makes the column)

We always are going to have 6 motion parameters from MCFLIRT so we are setting a counter called i

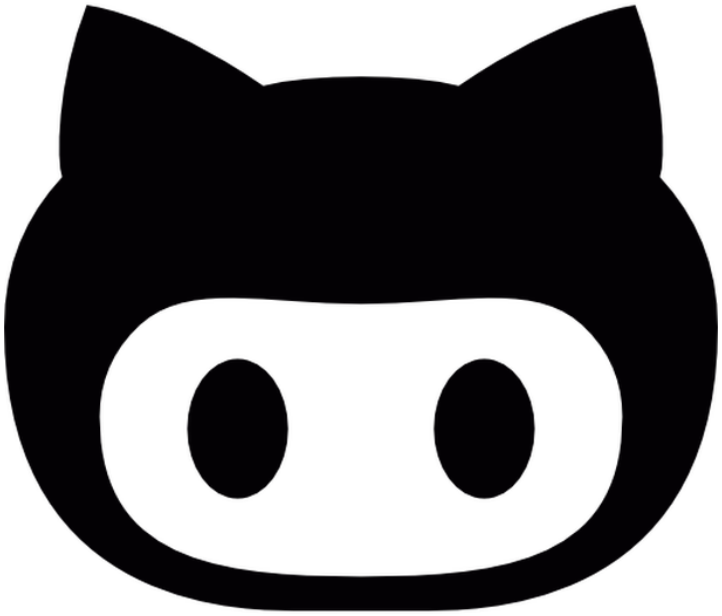
- This counter is going to count up to 6
- Python likes to be a little difficult, and starts counting at 0, so our counter will count (0,1,2,3,4,5) ## Newtable is taking our inital table and raw data and making a new table in a list format

In [ ]:
```
newtable = ([[line[i]] for line in table])
```

- It is using our counter (i) to grab each line

With our nice list we can now open a file to write to and each list item

- We are using the same counter so we don't mix up the parameters



In [ ]:
```
#!/usr/bin/env python
import glob
import os
#import pdb
import datetime
import argparse
import shutil
import subprocess
from multiprocessing import Pool

def prepro(basedir, args, arglist, outhtml, out_bad_bold_list,DATA):
#def better(args,arglist,basedir):
    #bet
    if args.STRIP==True:
        print("starting bet")

        for sub in DATA:
            for nifti in glob.glob(os.path.join(sub,'func','sub-*_task-%
s_bold.nii.gz')%(arglist['TASK'])):
                output=nifti.strip('.nii.gz')
                if os.path.exists(output+'_brain.nii.gz'):
                    print(output+' exists, skipping')
                else:
                    BET_OUTPUT=output+'_brain'
                    x=("/usr/local/fsl/bin/bet %s %s -F"%(input, BET_OUT
PUT))
                    os.system(x)


#bet rage
    if args.RAGE==True:
        print("starting bet rage")
```

```
        for sub in DATA:
            for input in glob.glob(os.path.join(sub,'anat','*T1w.nii.gz'
)):
                output=input.strip('.nii.gz')
                print(output)
                if os.path.exists(output+'_brain.nii.gz'):
                    print(output+' exists, skipping')
                else:
                    BET_OUTPUT=output+'_defaced'
                    x=("/usr/local/fsl/bin/bet %s %s -R"%(input, BET_OUT
PUT))
                    print(x)
                    os.system(x)



#motion correction
    if args.MOCO==False:
        print(" ")
    else:
        print("starting motion correction")
        for sub in DATA:
            for dir in glob.glob(os.path.join(sub,'func')):
                if not os.path.exists(os.path.join(basedir,dir,'motion_a
ssessment')):
                    os.makedirs(os.path.join(basedir,dir,'motion_assessm
ent'))
                os.chdir(os.path.join(basedir, dir))
                for input in glob.glob('*brain.nii.gz'):
                    output=input.split('.')[0]
                    if os.path.exists(os.path.join(dir,output+'_mcf.nii.
gz'))==True:
                        print(output+' exists, skipping')
                    else:
                        os.system("mcflirt -in %s -plots"%(output))
                        os.system("fsl_motion_outliers -i %s -o motion_a
ssessment/%s_confound.txt --fd --thresh=%s -p motion_assessment/fd_plot
 -v > motion_assessment/%s_outlier_output.txt"%(output,output,arglist['M
OCO'],output))
                        os.system("cat motion_assessment/%s_outlier_outp
ut.txt >> %s"%(output,outhtml))
                        plotz=os.path.join(basedir,dir,'motion_assessmen
t','fd_plot.png')
                        os.system("echo '<p>=============<p>FD plot %s <
br><IMG BORDER=0 SRC=%s WIDTH=%s></BODY></HTML>' >> %s"%(output,plotz,'1
00%', outhtml))

                        if os.path.isfile("motion_assessment/%s_confoun
d.txt"%(output))==False:
                            os.system("touch motion_assessment/%s_confou
nd.txt"%(output))

                        check = subprocess.check_output("grep -o 1 motio
n_assessment/%s_confound.txt | wc -l"%(output), shell=True)
                        num_scrub = [int(s) for s in check.split() if s.
isdigit()]
```

```
                        if num_scrub[0]>45:
                            with open(out_bad_bold_list, "a") as myfile:
                                myfile.write("%s\n"%(output))
                            myfile.close()

                        if os.path.exists("%s_mcf.par"%(output)):
                            if os.path.exists(os.path.join(basedir,dir,
'motion_assessment',"%s_mcf.par"%(output))):
                                    os.remove(os.path.join(basedir,dir,
'motion_assessment',"%s_mcf.par"%(output)))

                        shutil.move("%s_mcf.par"%(output),os.path.join(b
asedir,dir,'motion_assessment'))
                        rawfile = open(os.path.join(os.path.join(basedir
,dir,'motion_assessment','%s_mcf.par'%(output))), 'r')
                        table = [line.rstrip().split() for line in rawfi
le.readlines()]
                        for i in range(6):
                            newtable = ([[line[i]] for line in table])
                            f=open(os.path.join(basedir,dir,'motion_asse
ssment','%s_motcor%i.txt'%(output,i)),'w')
                            for item in newtable:
                                neat=item[0]
                                f.write(str(neat)+'\n')
                            f.close()




def split_list(a_list):
        half = len(a_list)/2
        return a_list[:half], a_list[half:]


def main(DATA):
    basedir='/Users/gracer/Desktop/data'
    writedir='/Users/gracer/Desktop/data'

    datestamp=datetime.datetime.now().strftime("%Y-%m-%d-%H_%M_%S")
    outhtml = os.path.join(writedir,'bold_motion_QA_%s.html'%(datestamp
))
    out_bad_bold_list = os.path.join(writedir,'lose_gt_45_vol_scrub_%s.t
xt'%(datestamp))

    parser=argparse.ArgumentParser(description='preprocessing')
    parser.add_argument('-task',dest='TASK',
                        default=False, help='which task are we running o
n?')
    parser.add_argument('-bet',dest='STRIP',action='store_true',
                        default=False, help='bet via fsl using defaults
 for functional images')
    parser.add_argument('-betrage',dest='RAGE',action='store_true',
                        default=False, help='bet via fsl using robust es
timation for anatomical images')
    parser.add_argument('-reorient',dest='REOR',action='store_true',
```