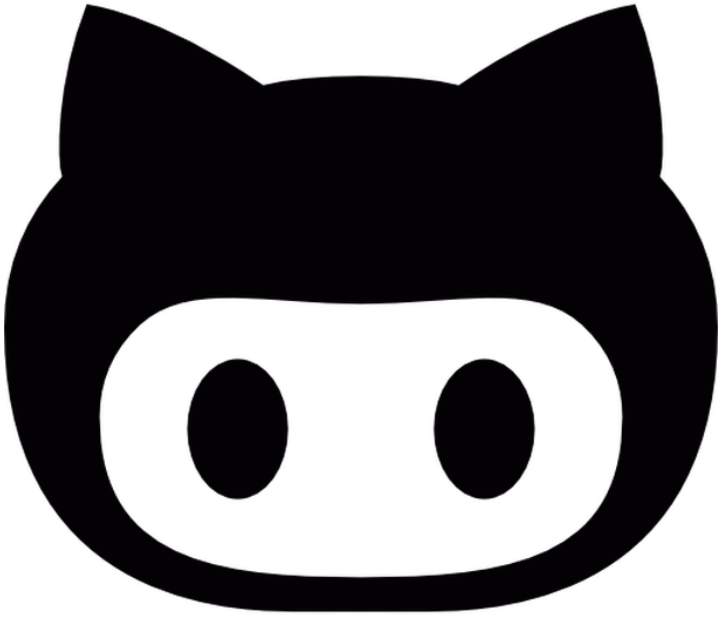
Level 1 FEAT setup!

Open up a fresh blank script in your Spyder IDE and save it as make_level1.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

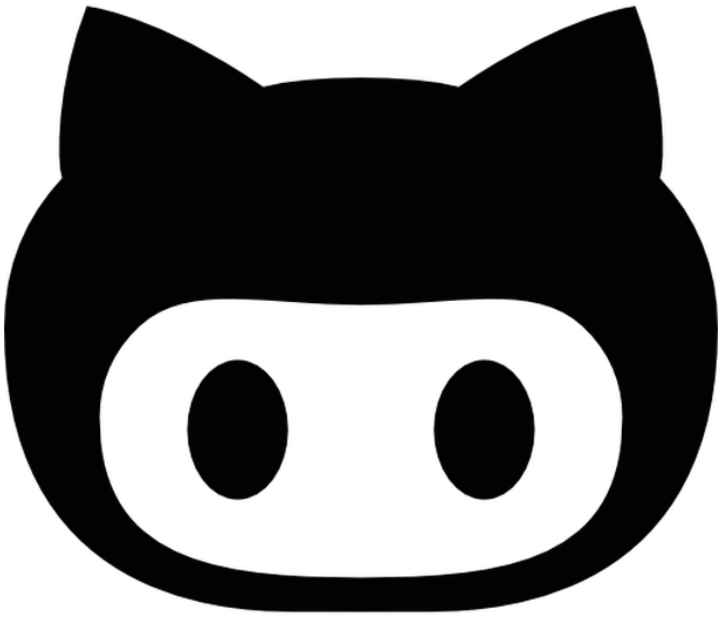


As always we are going to start by importing our modules

In [1]:
```
import glob
import os
from subprocess import check_output
#import pdb
import argparse
from IPython.display import Image
from IPython.core.display import HTML
```

Lets start by building our two functions. This time we will call the first create_fsf( ) and as always main( )

```
def create_fsf():


def main ():


main()
```



Strategy:

We are going to use the .fsf file to automate our feat analysis. To do this we are going to need to do the following:

1. Decide on our analysis plan (what variables are we using? This should be done before we even try to automate)
2. Get a .fsf and find all the variables we are going to automate and which will be hard coded
3. Use a dictonary in python to fill in our fsf file
4. Make a script to run our new fsf files in parallel

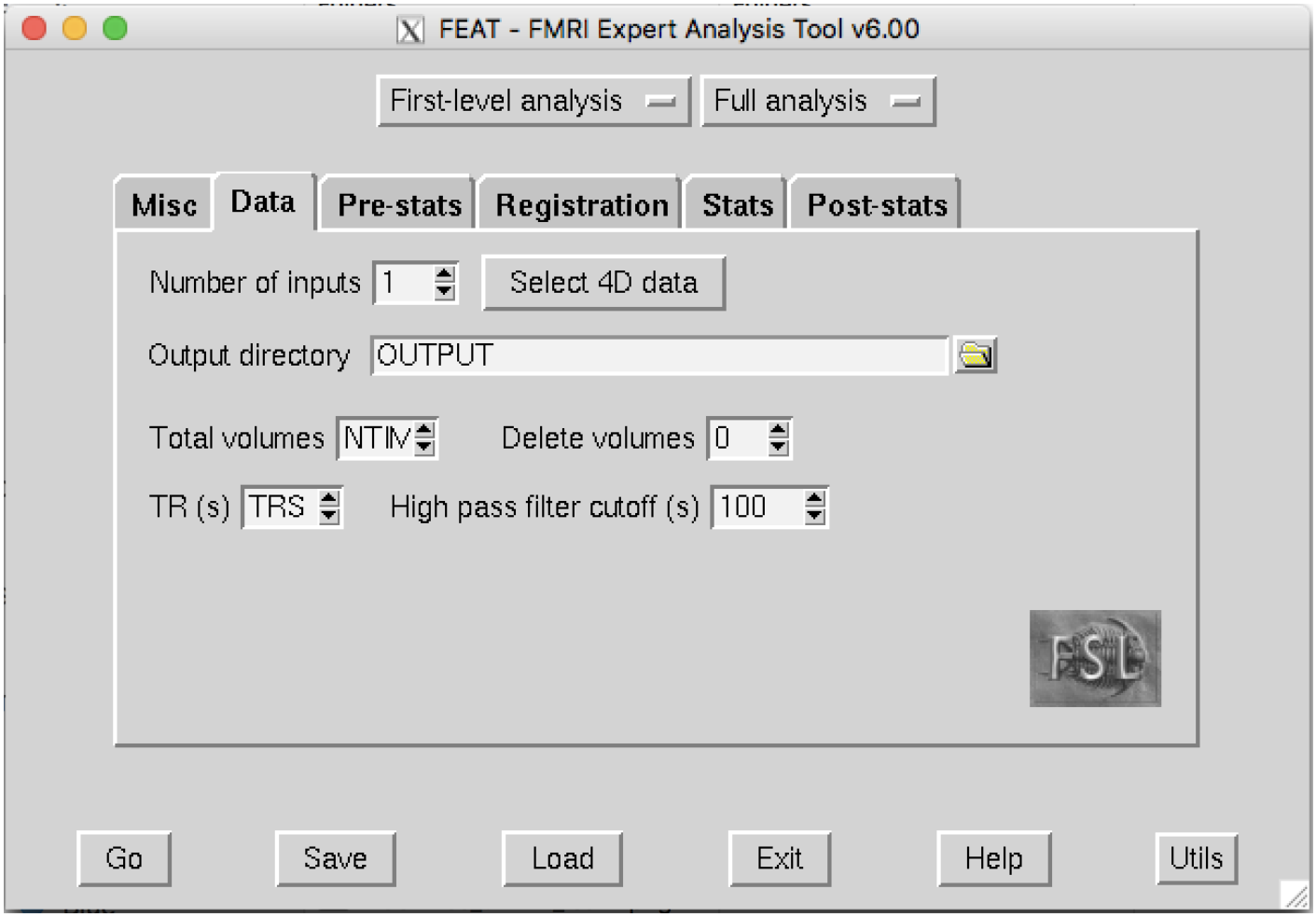
1. Decide on our analysis plan (what variables are we using? This should be done before we even try to automate)

Normally this would be set out before you even gathered your data, or if you are doing secondary data analysis, before you dive into the data. You could even do a spiffy pre-reg https://osf.io/ (https://osf.io/)

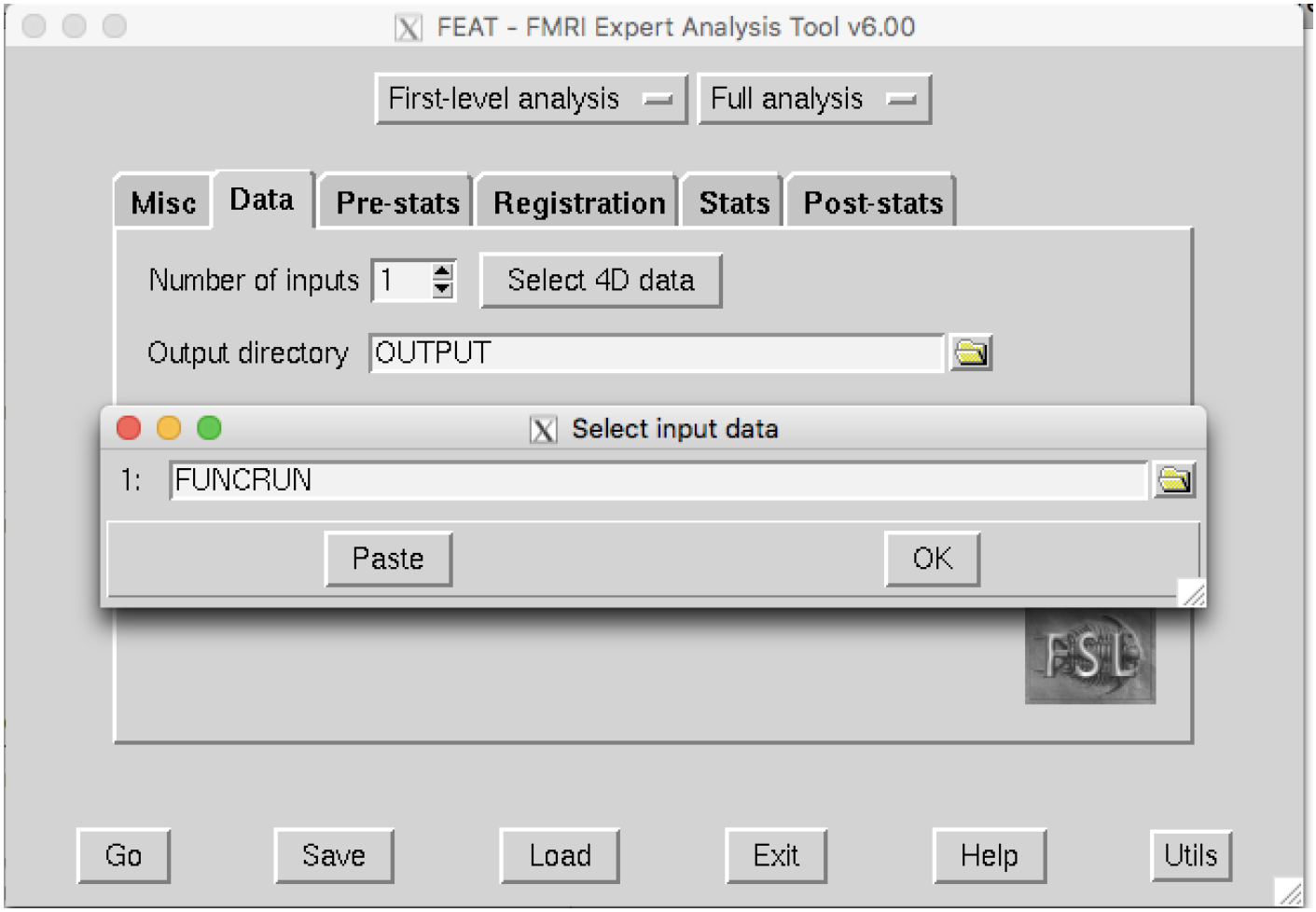
Since this is a workshop we are going to pretend we did all that leg work. We are going to use the bart task and look at the difference between CASH trials and the ACCEPT trials and EXPLODE trials. Does this analysis make sense in real life? I don't know, but that doesn't really matter here.

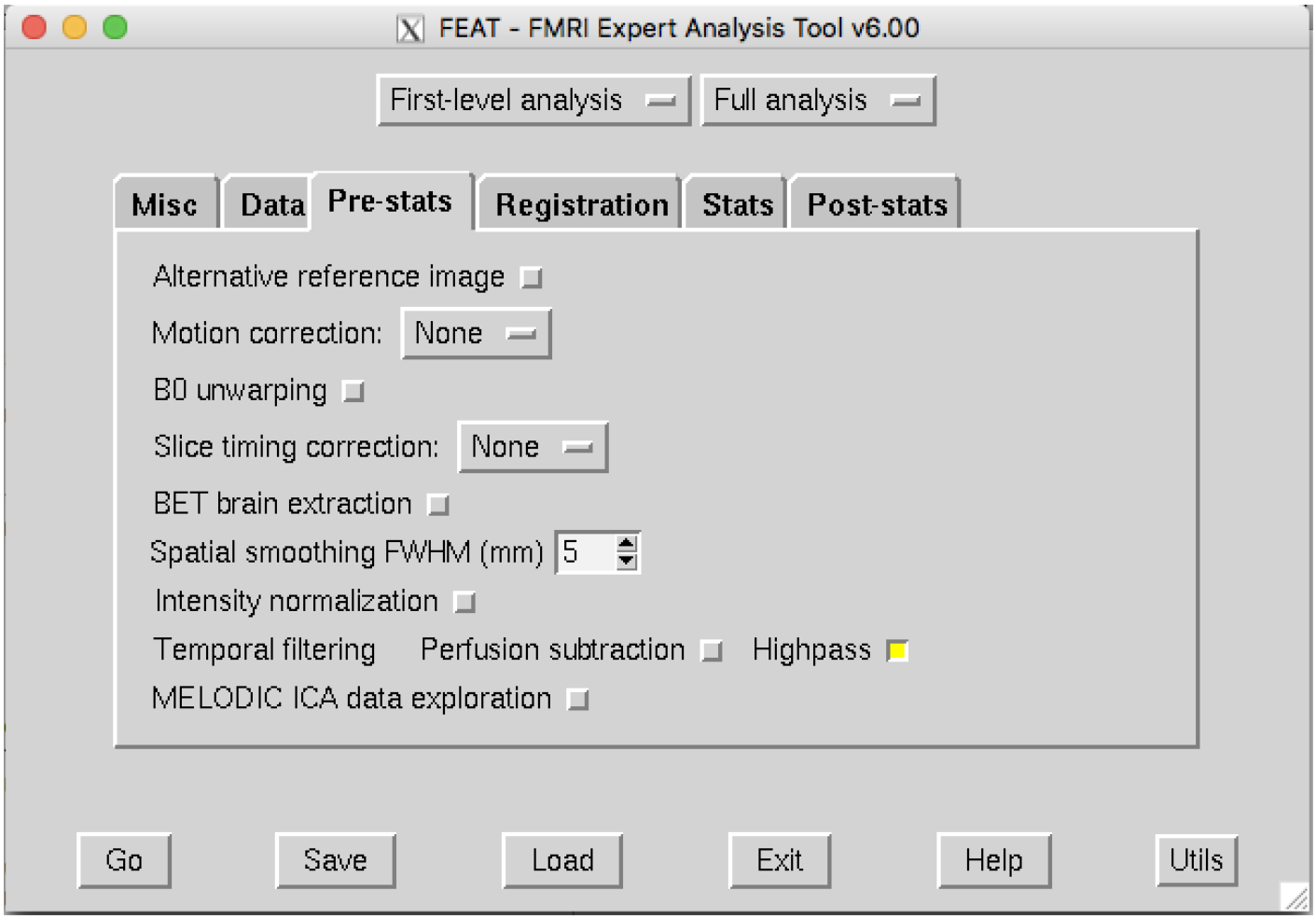
2. Get a .fsf and find all the variables we are going to automate and which will be hard coded

The easiest way to do this is to fire up the Feat gui and to start a list of variables we are going to hard code and those we are going to fill in through python.



We can do a lot of heavy lifting here. Notice my output directory is now just OUTPUT I have variables for the total volumes and the TRs I try to populate as many fields as possible with info from the image

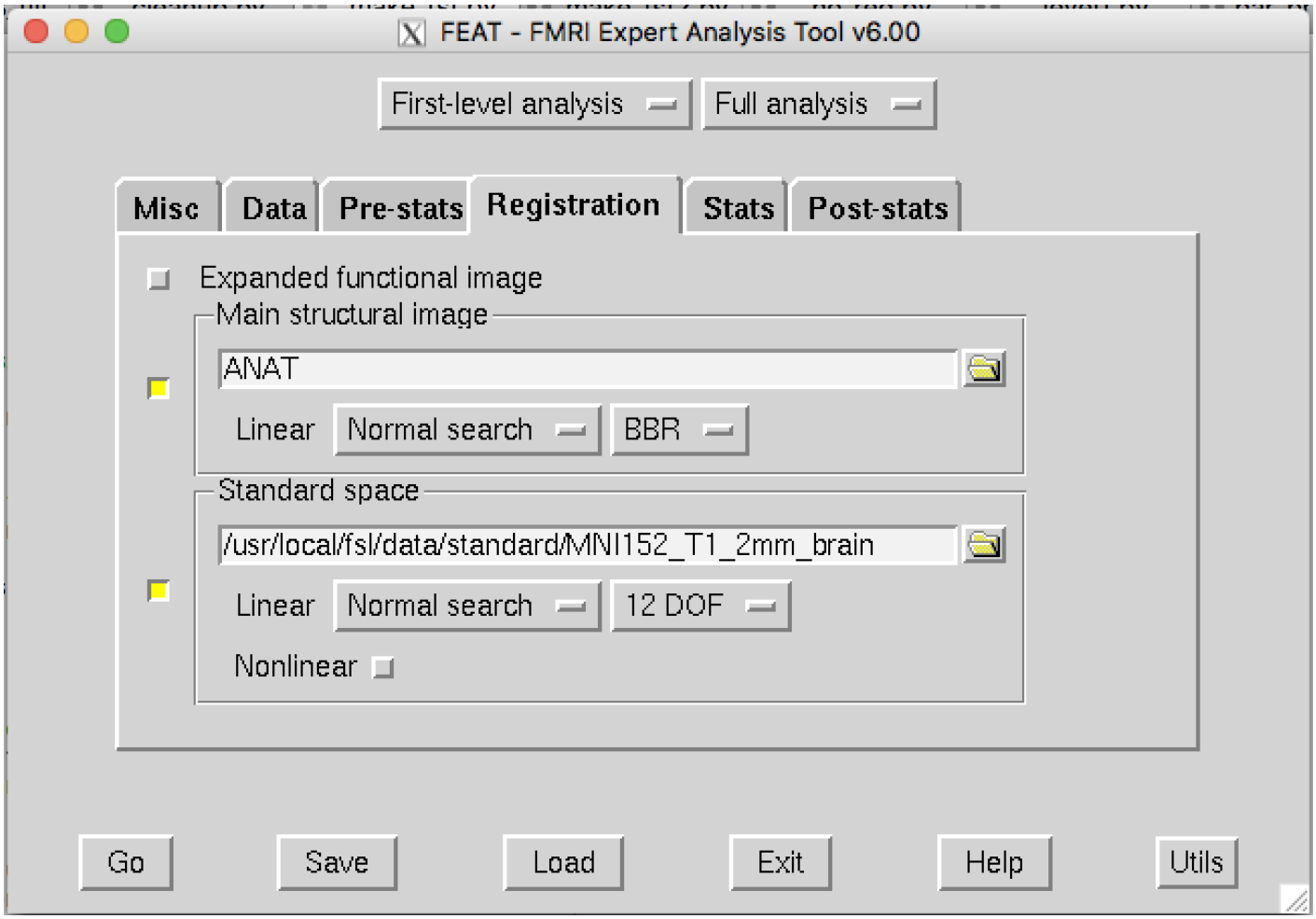




Alternatively, the pre-stats can almost be completely filled into the GUI once Why would we ever change this between runs/subjects? I can't think of a reason

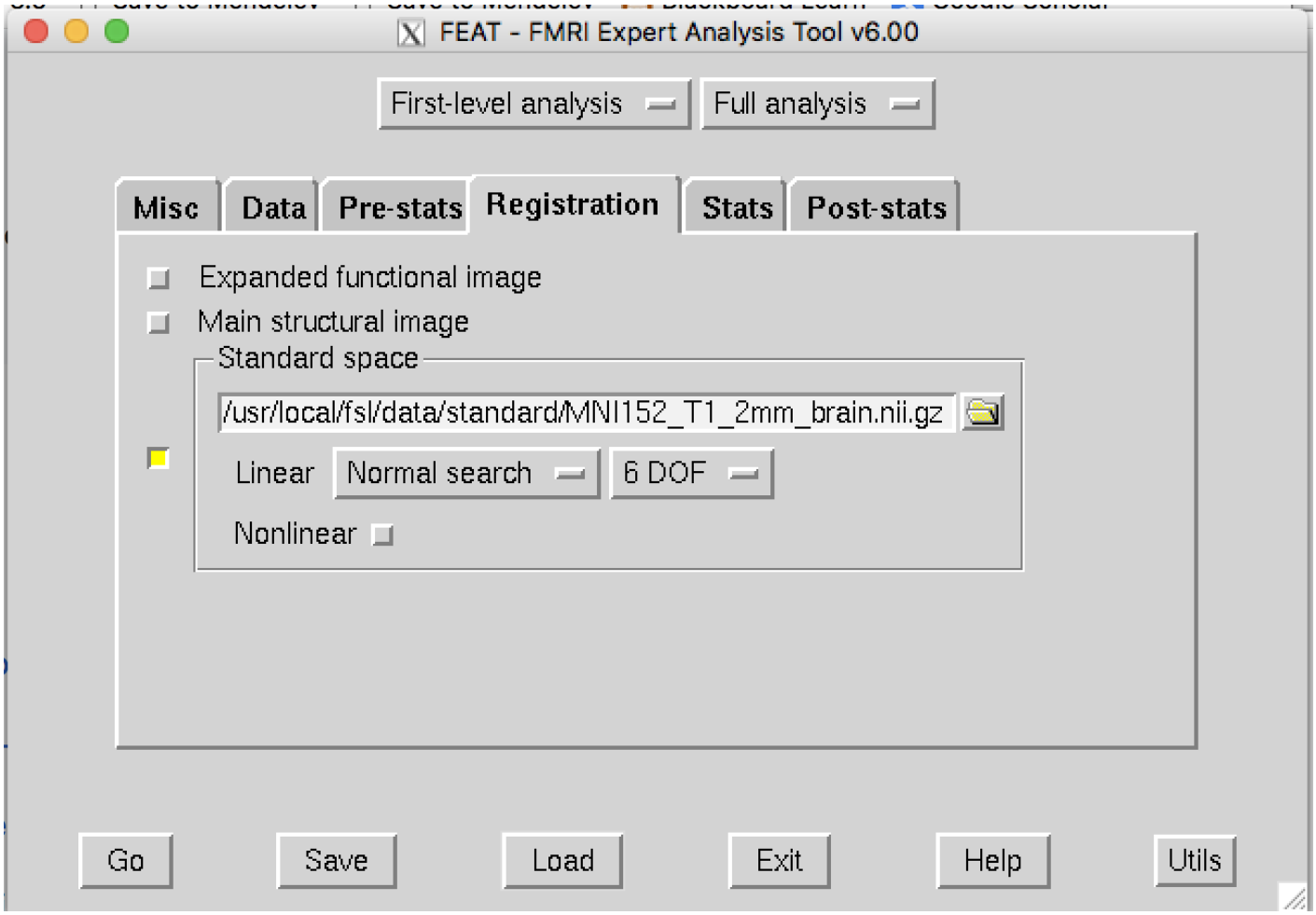
Now to the registration tab, this is big choice point! Are we planning on using data registered elsewhere?

Nope gonna register it right here



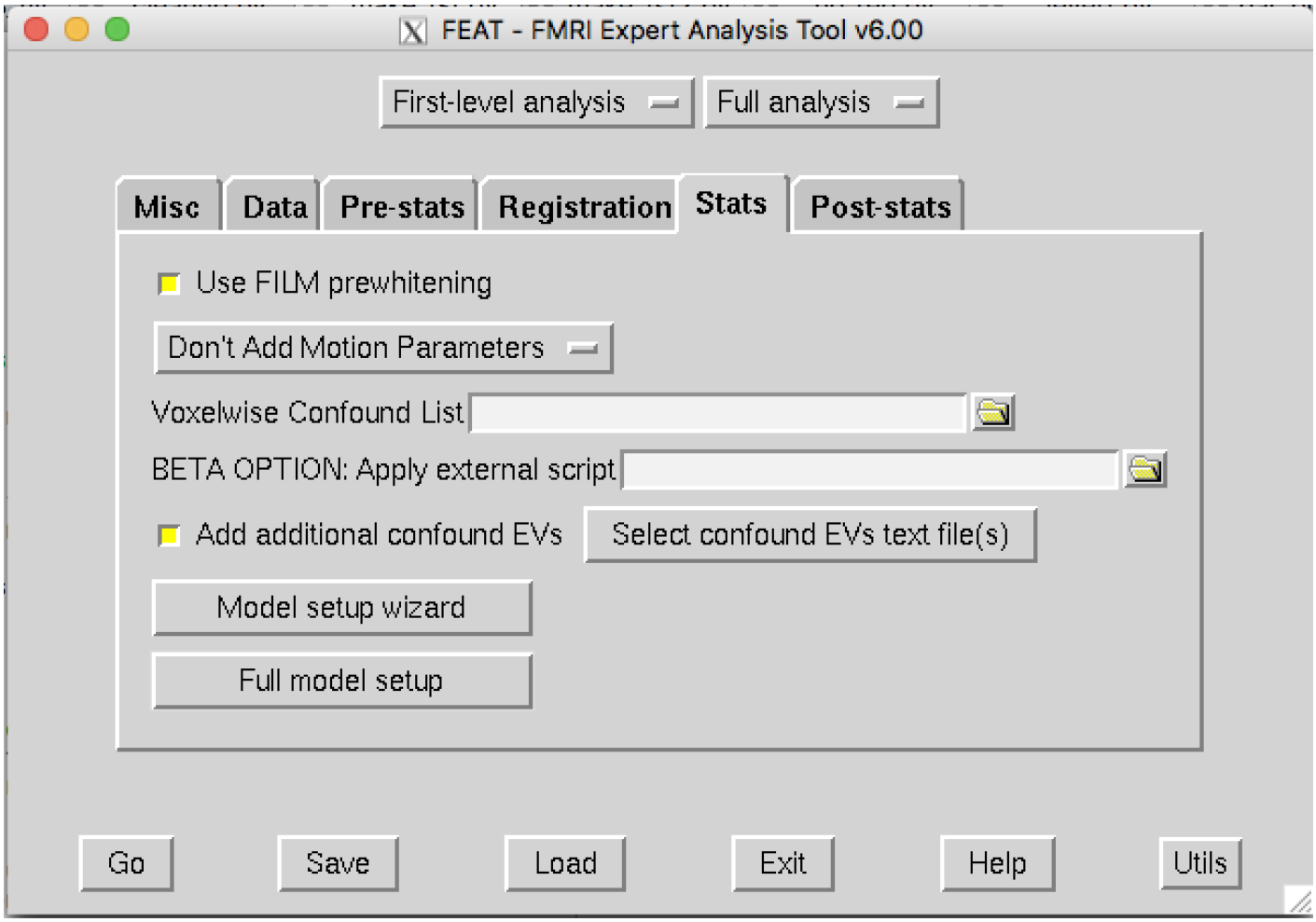
Since we are loading in our anatomical image we are going to create a variable for it. Make sure you have the correct path to your standard space!

I'll hand my own registration elsewhere thank you

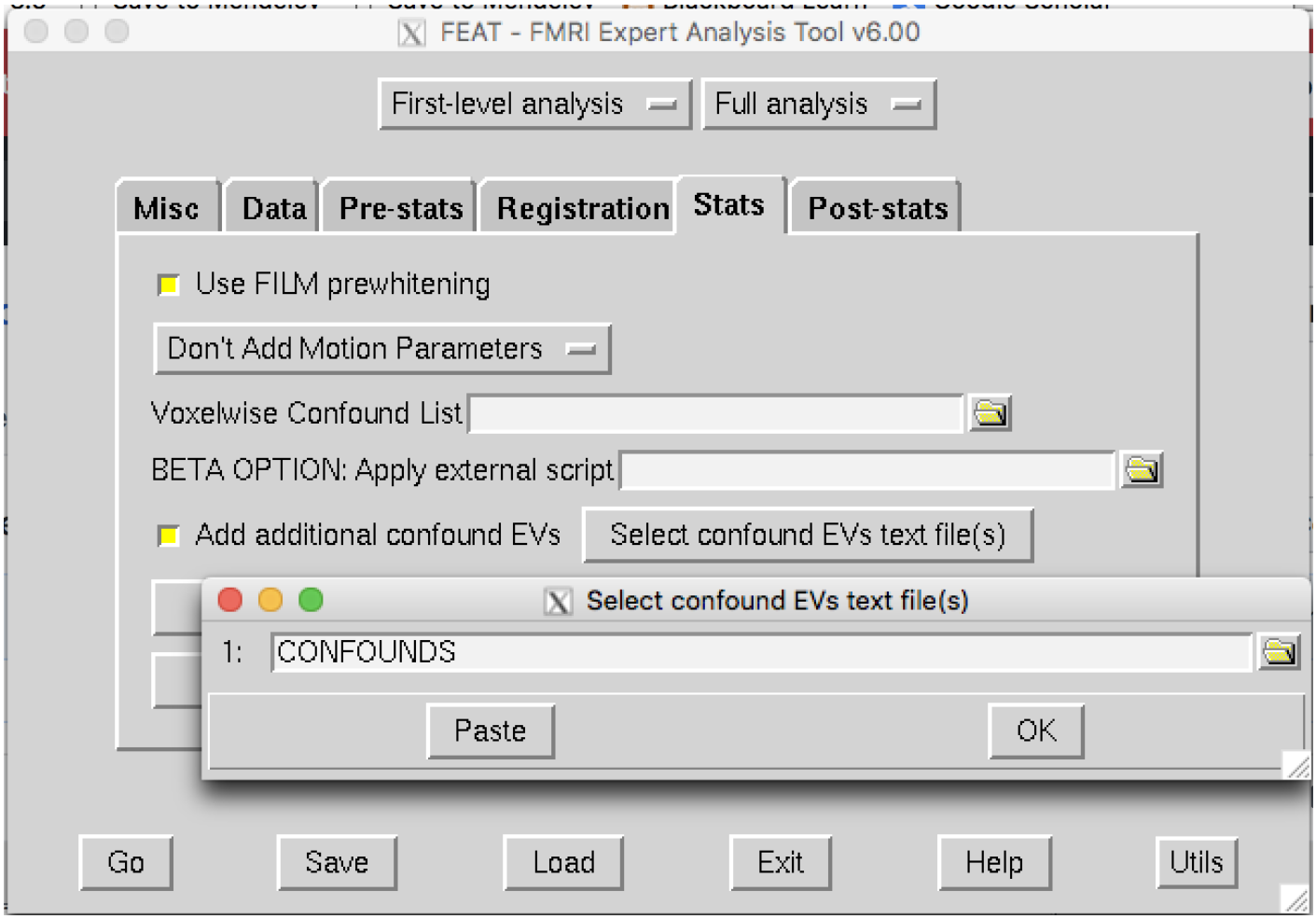


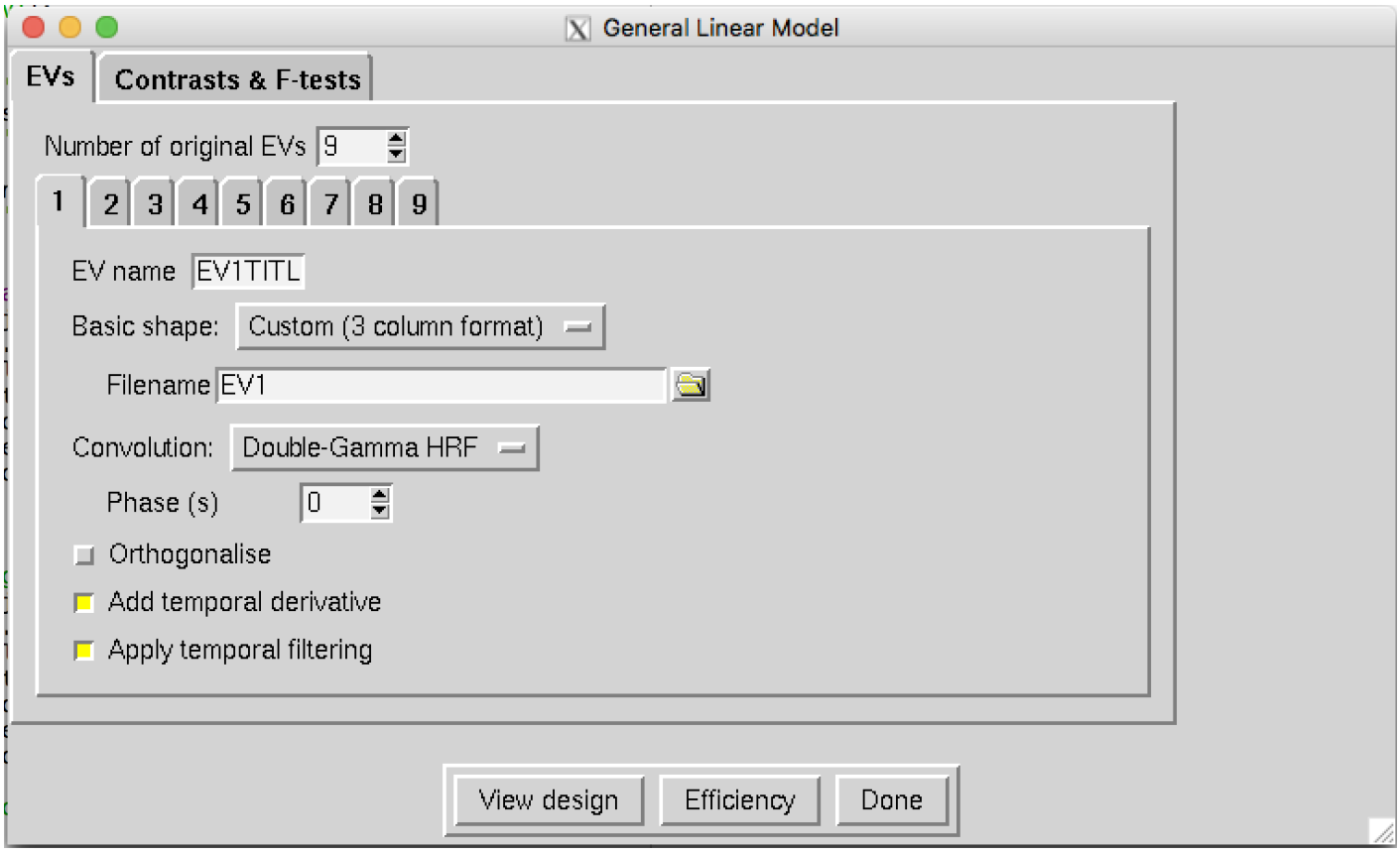
We probably spent a long time (or computing power) on our custom registration, so let's save time here and use only 6 DOF ;)

The heavy lifting: The Stats Tab!

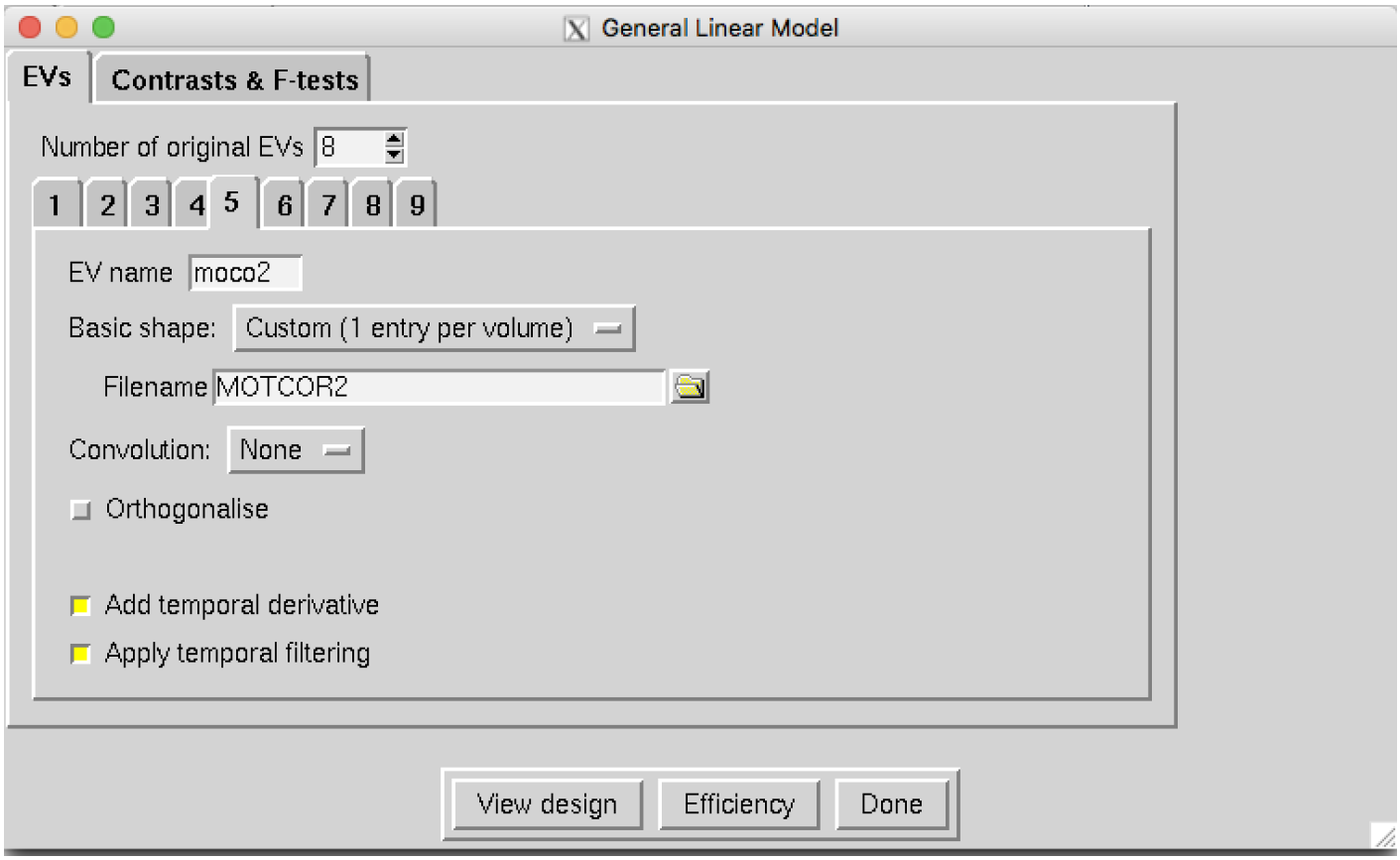


Notice we have both the FILM prewhiting tab and the Add additional confounds checked.


FEAT - FMRI Expert Analysis Tool v6.00
First-level analysis
Full analysis
Misc
Data
Pre-stats
Registration
Stats
Post-stats
Use FILM prewhitening
Don't Add Motion Parameters
Voxelwise Confound List
BETA OPTION: Apply external script
Add additional confound EVs
Select confound EVs text file(s)
Select confound EVs text file(s)
1:
CONFOUNDS
Paste
OK
Go
Save
Load
Exit
Help
Utils

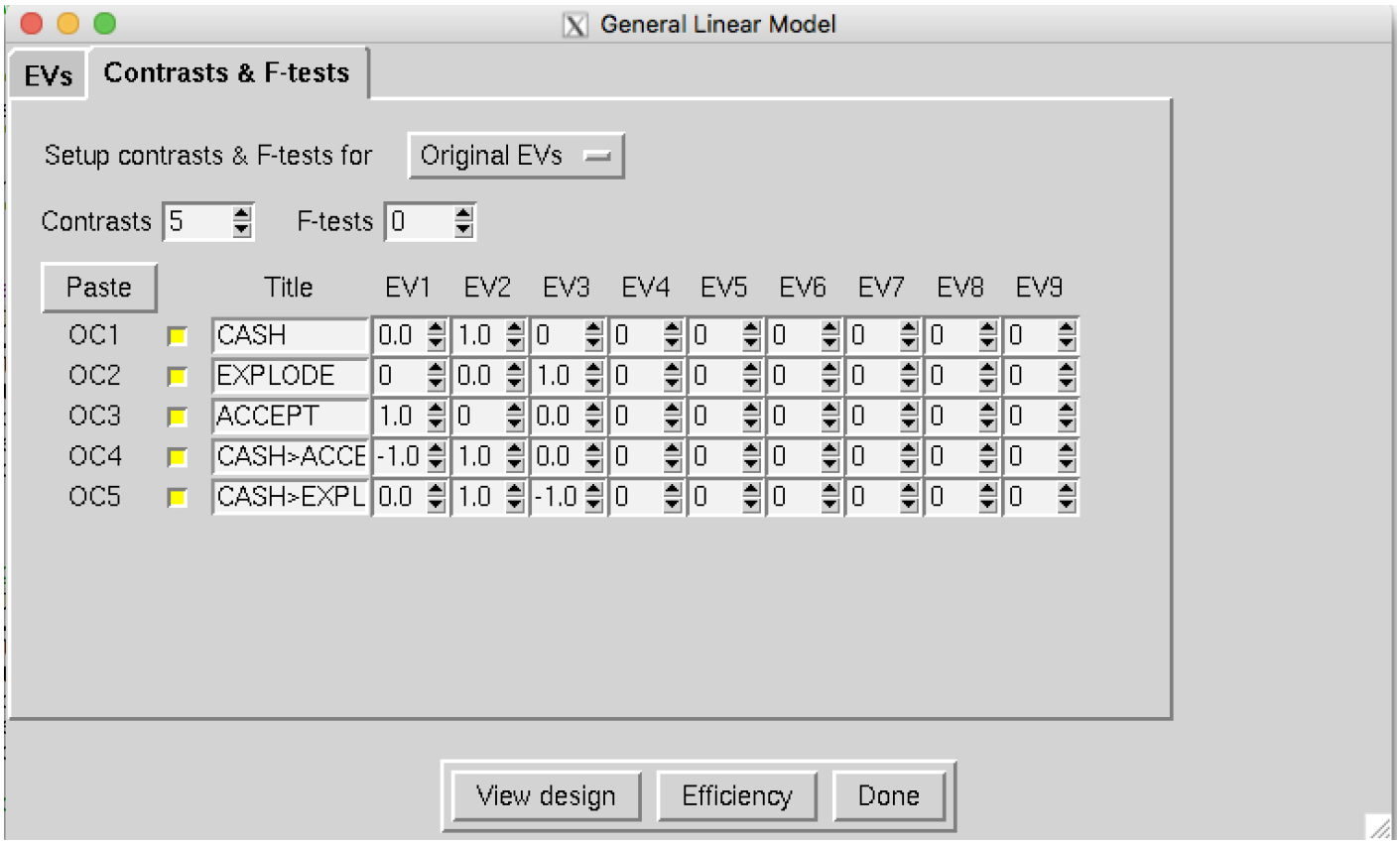


Here we can set a variable for each EVTITLE and the EV itself We just need to make sure they match We can also do this for the motion correction parameters (if we chose not to do it through FEAT)



Notice for the motion correction I have set the shape to a single column and the convolution to none.

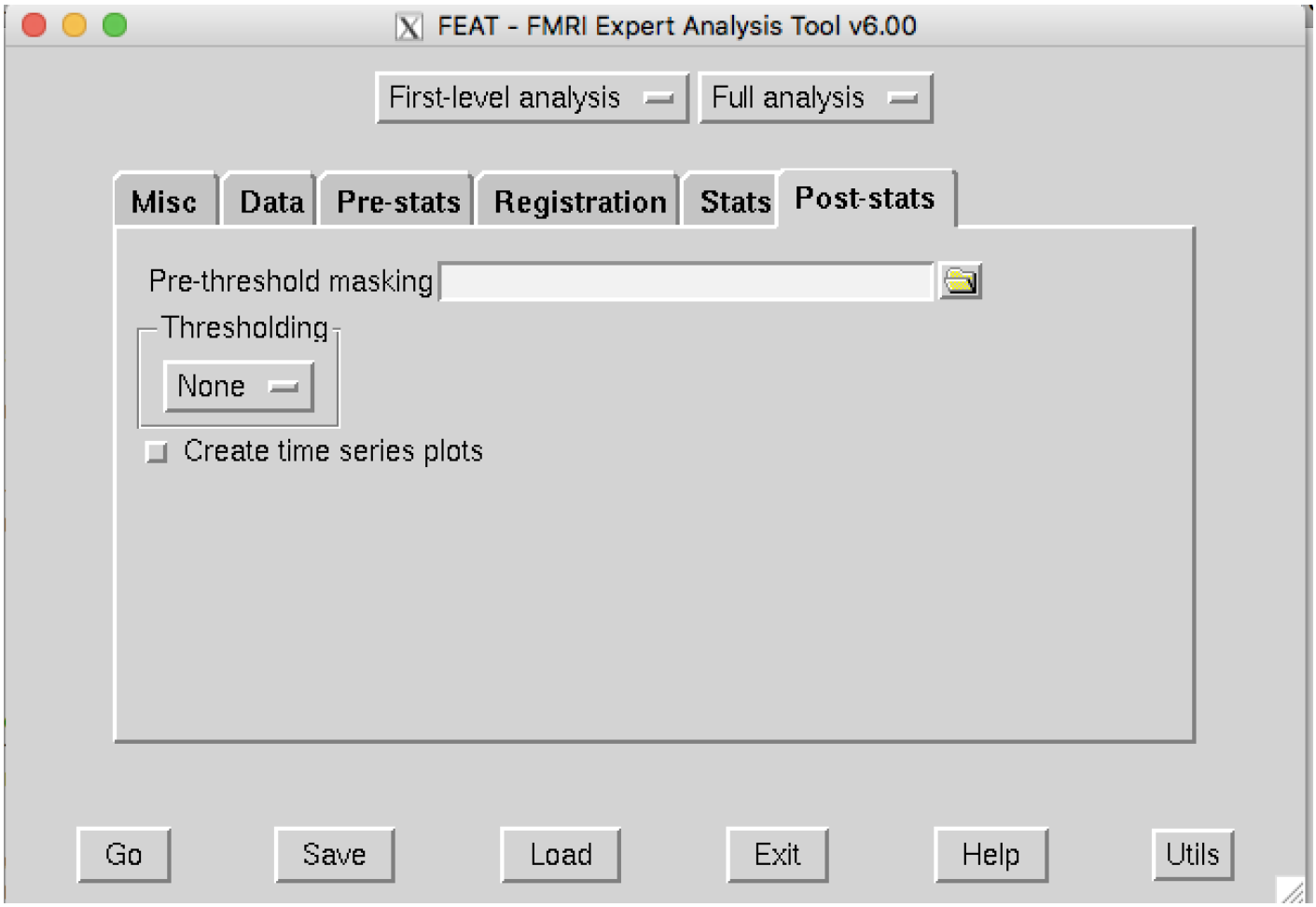
Concept Check: Why did set the convolution to none?



Here are the fake contrasts we will look at I like to take a screenshot of this as a reference since after this fsl will refer to these by number only

Future Knowledge: What does fsl call the higher level directories based on our contrasts?

Last but not least (ok kinda least): POST STATS



My personal favorite, turn everything off we will come back to this when we do our group level model

Great we are done with our one GUI, and if you tend to do similar analyeses maybe the last ever! Click save and save the file as:

Level1_design.fsf

The GUI will get mad at you and tell you you are missing a lot of stuff. Just click through. Make sure the .fsf file was generated. We can delete all the other files.

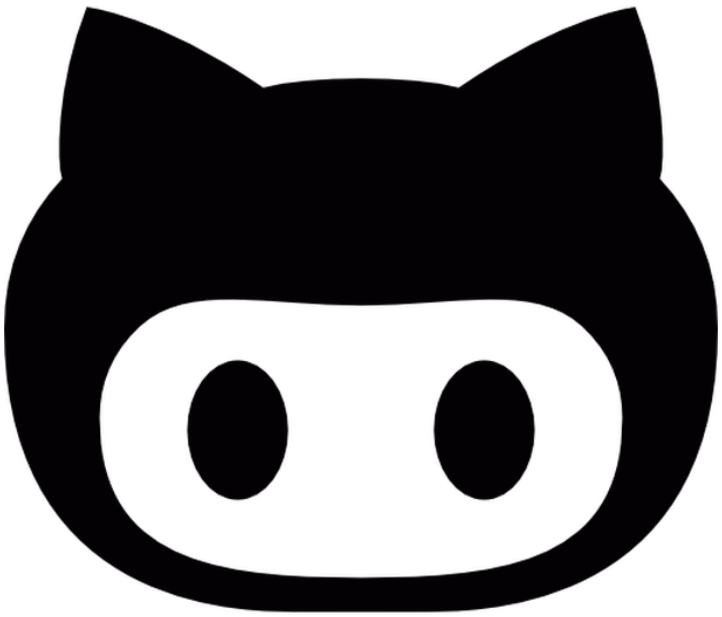
Let's remind ourselves what we variables we will need to generate

1. FUNCRUN = this will be our input image
2. NTIMEPOINTS = this is the number of timepoints per image
3. TRS = this is the TRs in the nifti
4. OUTPUT = what we want to call our output directory, it will be something .feat
5. ANAT = this is our T1W image, if we are using FNIRT it needs have skull, if we are using FLIRT we want the defaced (betted)
6. CONFOUNDS = this is the confound.txt file generated from fsl_motion_outliers that we should have in our motion assessment directory if our prepro script worked
7. EV = this is the path to the explanatory variable text file we want to use (we will have more than 1), each needs to be in the 3 column format
8. EVTITLE = this is a useful name for each EV
9. MOTCOR = this is the path to the motion parameters we generated in with the preprocessing script

Let's also remember a major choice point

Are we using feat to register or no?

At this time let's open that .fsf file we just created with our text editor and take a look at our handy work!



Okay now for the actual python part!

right now our script looks a little bare

```
def create_fsf():

def main():

main()
```

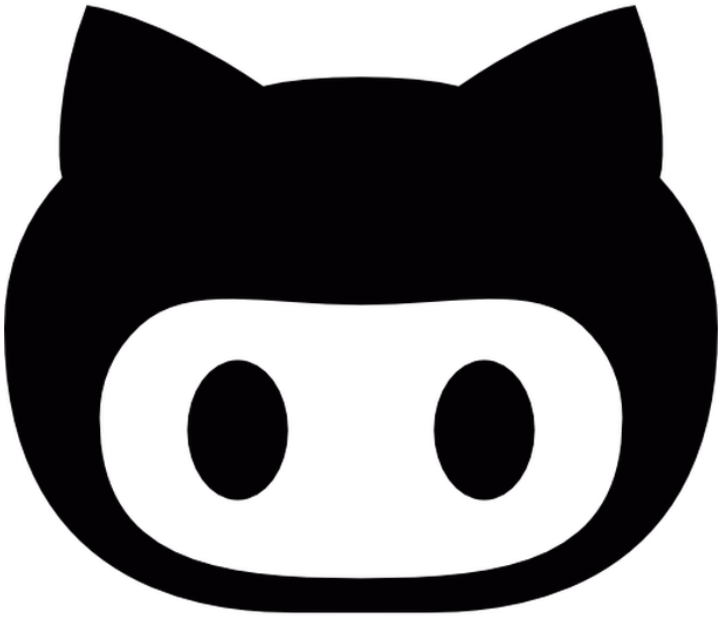
Let's set up some globals

What are we sure we are going to need?

1. basedir = this is going to be like second nature, where is our data?
2. outdir = where is our output data located, if we are using a bids like structure probably some where with derivative in the directory tree
3. Do we want to pass arguments via argparse? YES
   - We have multiple tasks so we will want something argument to differentiate task
   - We have a lot of potential EVs so we will want something that will take multiple arguments
   - We have a serious choice point of to reg or not to reg (shakespeare's got nothing on us) ### Give it a try and write your main( ) function

In [ ]:
```
def main ():
  basedir='/Users/gracer/Desktop/data'
  outdir=os.path.join(basedir,'derivatives','task')
  parser=argparse.ArgumentParser(description='making fsf files')
  parser.add_argument('-noreg',dest='NOREG', action='store_true',
                      default=False, help='Did you already register yo
ur data (using ANTZ maybe)?')
  parser.add_argument('-task',dest='TASK',
                      default=False, help='which task are we using?')
  parser.add_argument('-evs',dest='EV',nargs='+',
                      default=False, help='which evs are we using?')

  repl_dict={}
  args = parser.parse_args()
  arglist={}
  for a in args._get_kwargs():
      arglist[a[0]]=a[1]
      print(arglist)
  create_fsf(basedir,repl_dict, outdir, arglist)
```

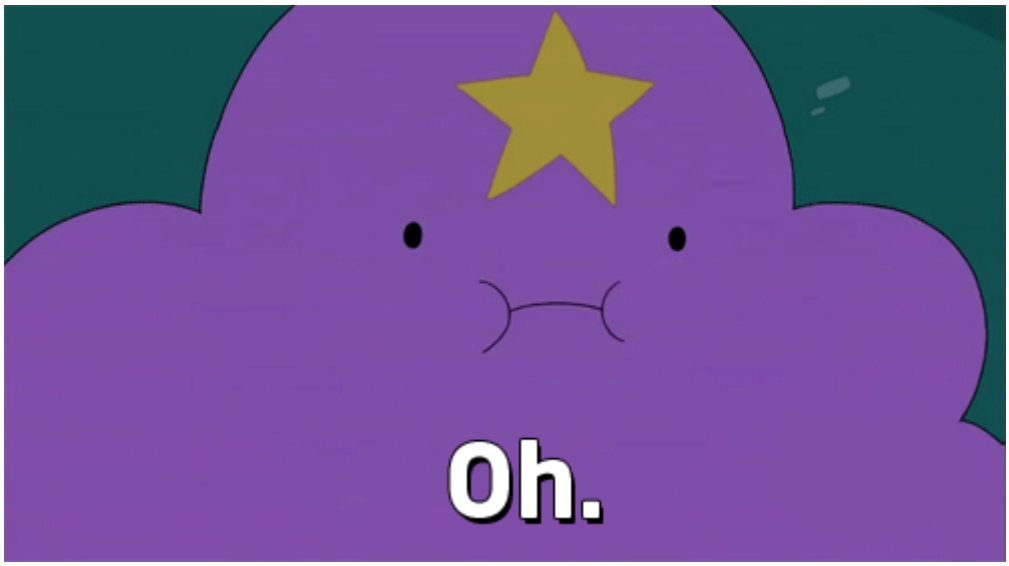
This is what I came up with does anyone have anything different? There are a lot of ways we could accomplish this



On to the meaty bits! The create_fsf( ) function!

We need to get all the subjects.

Concept Check: What can we use to get all the subjects?


Oh.

In [3]:
```
basedir='/Users/gracer/Desktop/data'
listy=glob.glob(os.path.join(basedir,'sub-*','func','*.nii.gz'))
print(listy[1].split('/'))
repl_dict={}
```

```
['', 'Users', 'gracer', 'Desktop', 'data', 'sub-10159', 'func', 'sub-10
159_task-bart_bold_brain.nii.gz']
['/Users/gracer/Desktop/data/sub-10159/func/sub-10159_task-bart_bold.ni
i.gz', '/Users/gracer/Desktop/data/sub-10159/func/sub-10159_task-bart_b
old_brain.nii.gz', '/Users/gracer/Desktop/data/sub-10159/func/sub-10159
_task-bart_bold_brain_mask.nii.gz', '/Users/gracer/Desktop/data/sub-101
59/func/sub-10159_task-bart_bold_brain_mcf.nii.gz', '/Users/gracer/Desk
top/data/sub-10159/func/sub-10159_task-rest_bold.nii.gz', '/Users/grace
r/Desktop/data/sub-10159/func/sub-10159_task-scap_bold.nii.gz', '/User
s/gracer/Desktop/data/sub-10159/func/sub-10159_task-stopsignal_bold.ni
i.gz', '/Users/gracer/Desktop/data/sub-10159/func/sub-10159_task-tasksw
itch_bold.nii.gz', '/Users/gracer/Desktop/data/sub-10171/func/sub-10171
_task-bart_bold.nii.gz', '/Users/gracer/Desktop/data/sub-10171/func/sub
-10171_task-bart_bold_brain.nii.gz', '/Users/gracer/Desktop/data/sub-10
171/func/sub-10171_task-bart_bold_brain_mask.nii.gz', '/Users/gracer/De
sktop/data/sub-10171/func/sub-10171_task-bart_bold_brain_mcf.nii.gz',
'/Users/gracer/Desktop/data/sub-10171/func/sub-10171_task-bht_bold.nii.
gz', '/Users/gracer/Desktop/data/sub-10171/func/sub-10171_task-rest_bol
d.nii.gz', '/Users/gracer/Desktop/data/sub-10171/func/sub-10171_task-sc
ap_bold.nii.gz', '/Users/gracer/Desktop/data/sub-10171/func/sub-10171_t
ask-stopsignal_bold.nii.gz', '/Users/gracer/Desktop/data/sub-10171/fun
c/sub-10171_task-taskswitch_bold.nii.gz', '/Users/gracer/Desktop/data/s
ub-10189/func/sub-10189_task-bart_bold.nii.gz', '/Users/gracer/Desktop/
data/sub-10189/func/sub-10189_task-bart_bold_brain.nii.gz', '/Users/gra
cer/Desktop/data/sub-10189/func/sub-10189_task-bart_bold_brain_mask.ni
i.gz', '/Users/gracer/Desktop/data/sub-10189/func/sub-10189_task-bart_b
old_brain_mcf.nii.gz', '/Users/gracer/Desktop/data/sub-10189/func/sub-1
0189_task-rest_bold.nii.gz', '/Users/gracer/Desktop/data/sub-10189/fun
c/sub-10189_task-scap_bold.nii.gz', '/Users/gracer/Desktop/data/sub-101
89/func/sub-10189_task-stopsignal_bold.nii.gz', '/Users/gracer/Desktop/
data/sub-10189/func/sub-10189_task-taskswitch_bold.nii.gz', '/Users/gra
cer/Desktop/data/sub-10193/func/sub-10193_task-bart_bold.nii.gz', '/Use
rs/gracer/Desktop/data/sub-10193/func/sub-10193_task-bart_bold_brain.ni
i.gz', '/Users/gracer/Desktop/data/sub-10193/func/sub-10193_task-bart_b
old_brain_mask.nii.gz', '/Users/gracer/Desktop/data/sub-10193/func/sub-
10193_task-bart_bold_brain_mcf.nii.gz', '/Users/gracer/Desktop/data/sub
-10206/func/sub-10206_task-bart_bold.nii.gz', '/Users/gracer/Desktop/da
ta/sub-10206/func/sub-10206_task-bart_bold_brain.nii.gz', '/Users/grace
r/Desktop/data/sub-10206/func/sub-10206_task-bart_bold_brain_mask.nii.g
z', '/Users/gracer/Desktop/data/sub-10206/func/sub-10206_task-bart_bold
_brain_mcf.nii.gz', '/Users/gracer/Desktop/data/sub-10206/func/sub-1020
6_task-rest_bold.nii.gz', '/Users/gracer/Desktop/data/sub-10206/func/su
b-10206_task-scap_bold.nii.gz', '/Users/gracer/Desktop/data/sub-10206/f
unc/sub-10206_task-stopsignal_bold.nii.gz', '/Users/gracer/Desktop/dat
a/sub-10206/func/sub-10206_task-taskswitch_bold.nii.gz', '/Users/grace
r/Desktop/data/sub-10217/func/sub-10217_task-bart_bold.nii.gz', '/User
s/gracer/Desktop/data/sub-10217/func/sub-10217_task-bart_bold_brain.ni
i.gz', '/Users/gracer/Desktop/data/sub-10217/func/sub-10217_task-bart_b
old_brain_mask.nii.gz', '/Users/gracer/Desktop/data/sub-10217/func/sub-
10217_task-bart_bold_brain_mcf.nii.gz', '/Users/gracer/Desktop/data/sub
-10217/func/sub-10217_task-rest_bold.nii.gz', '/Users/gracer/Desktop/da
ta/sub-10217/func/sub-10217_task-scap_bold.nii.gz', '/Users/gracer/Desk
top/data/sub-10217/func/sub-10217_task-stopsignal_bold.nii.gz', '/User
s/gracer/Desktop/data/sub-10217/func/sub-10217_task-taskswitch_bold.ni
i.gz', '/Users/gracer/Desktop/data/sub-10225/func/sub-10225_task-bart_b
old.nii.gz', '/Users/gracer/Desktop/data/sub-10225/func/sub-10225_task-
```

bart_bold_brain.nii.gz', '/Users/gracer/Desktop/data/sub-10225/func/sub
-10225_task-bart_bold_brain_mask.nii.gz', '/Users/gracer/Desktop/data/s
ub-10225/func/sub-10225_task-bart_bold_brain_mcf.nii.gz', '/Users/grace
r/Desktop/data/sub-10225/func/sub-10225_task-bht_bold.nii.gz', '/Users/
gracer/Desktop/data/sub-10225/func/sub-10225_task-rest_bold.nii.gz', '/
Users/gracer/Desktop/data/sub-10225/func/sub-10225_task-scap_bold.nii.g
z', '/Users/gracer/Desktop/data/sub-10225/func/sub-10225_task-stopsigna
l_bold.nii.gz', '/Users/gracer/Desktop/data/sub-10225/func/sub-10225_ta
sk-taskswitch_bold.nii.gz', '/Users/gracer/Desktop/data/sub-10227/func/
sub-10227_task-bart_bold.nii.gz', '/Users/gracer/Desktop/data/sub-1022
7/func/sub-10227_task-bart_bold_brain.nii.gz', '/Users/gracer/Desktop/d
ata/sub-10227/func/sub-10227_task-bart_bold_brain_mask.nii.gz', '/User
s/gracer/Desktop/data/sub-10227/func/sub-10227_task-bart_bold_brain_mc
f.nii.gz', '/Users/gracer/Desktop/data/sub-10227/func/sub-10227_task-re
st_bold.nii.gz', '/Users/gracer/Desktop/data/sub-10227/func/sub-10227_t
ask-scap_bold.nii.gz', '/Users/gracer/Desktop/data/sub-10227/func/sub-1
0227_task-stopsignal_bold.nii.gz', '/Users/gracer/Desktop/data/sub-1022
7/func/sub-10227_task-taskswitch_bold.nii.gz', '/Users/gracer/Desktop/d
ata/sub-10228/func/sub-10228_task-bart_bold.nii.gz', '/Users/gracer/Des
ktop/data/sub-10228/func/sub-10228_task-bart_bold_brain.nii.gz', '/User
s/gracer/Desktop/data/sub-10228/func/sub-10228_task-bart_bold_brain_mas
k.nii.gz', '/Users/gracer/Desktop/data/sub-10228/func/sub-10228_task-ba
rt_bold_brain_mcf.nii.gz', '/Users/gracer/Desktop/data/sub-10228/func/s
ub-10228_task-bht_bold.nii.gz', '/Users/gracer/Desktop/data/sub-10228/f
unc/sub-10228_task-rest_bold.nii.gz', '/Users/gracer/Desktop/data/sub-1
0228/func/sub-10228_task-scap_bold.nii.gz', '/Users/gracer/Desktop/dat
a/sub-10228/func/sub-10228_task-stopsignal_bold.nii.gz', '/Users/grace
r/Desktop/data/sub-10228/func/sub-10228_task-taskswitch_bold.nii.gz',
'/Users/gracer/Desktop/data/sub-10235/func/sub-10235_task-bart_bold.ni
i.gz', '/Users/gracer/Desktop/data/sub-10235/func/sub-10235_task-bart_b
old_brain.nii.gz', '/Users/gracer/Desktop/data/sub-10235/func/sub-10235
_task-bart_bold_brain_mask.nii.gz', '/Users/gracer/Desktop/data/sub-102
35/func/sub-10235_task-bart_bold_brain_mcf.nii.gz', '/Users/gracer/Desk
top/data/sub-10235/func/sub-10235_task-rest_bold.nii.gz', '/Users/grace
r/Desktop/data/sub-10235/func/sub-10235_task-scap_bold.nii.gz', '/User
s/gracer/Desktop/data/sub-10235/func/sub-10235_task-stopsignal_bold.ni
i.gz', '/Users/gracer/Desktop/data/sub-10235/func/sub-10235_task-tasksw
itch_bold.nii.gz', '/Users/gracer/Desktop/data/sub-10249/func/sub-10249
_task-bart_bold.nii.gz', '/Users/gracer/Desktop/data/sub-10249/func/sub
-10249_task-bart_bold_brain.nii.gz', '/Users/gracer/Desktop/data/sub-10
249/func/sub-10249_task-bart_bold_brain_mask.nii.gz', '/Users/gracer/De
sktop/data/sub-10249/func/sub-10249_task-bart_bold_brain_mcf.nii.gz',
'/Users/gracer/Desktop/data/sub-10249/func/sub-10249_task-bht_bold.nii.
gz', '/Users/gracer/Desktop/data/sub-10249/func/sub-10249_task-rest_bol
d.nii.gz', '/Users/gracer/Desktop/data/sub-10249/func/sub-10249_task-sc
ap_bold.nii.gz', '/Users/gracer/Desktop/data/sub-10249/func/sub-10249_t
ask-stopsignal_bold.nii.gz', '/Users/gracer/Desktop/data/sub-10249/fun
c/sub-10249_task-taskswitch_bold.nii.gz', '/Users/gracer/Desktop/data/s
ub-10269/func/sub-10269_task-bart_bold.nii.gz', '/Users/gracer/Desktop/
data/sub-10269/func/sub-10269_task-bart_bold_brain.nii.gz', '/Users/gra
cer/Desktop/data/sub-10269/func/sub-10269_task-bart_bold_brain_mask.ni
i.gz', '/Users/gracer/Desktop/data/sub-10269/func/sub-10269_task-bart_b
old_brain_mcf.nii.gz', '/Users/gracer/Desktop/data/sub-10269/func/sub-1
0269_task-rest_bold.nii.gz', '/Users/gracer/Desktop/data/sub-10269/fun
c/sub-10269_task-scap_bold.nii.gz', '/Users/gracer/Desktop/data/sub-102
69/func/sub-10269_task-stopsignal_bold.nii.gz', '/Users/gracer/Desktop/
data/sub-10269/func/sub-10269_task-taskswitch_bold.nii.gz', '/Users/gra

cer/Desktop/data/sub-10271/func/sub-10271_task-bart_bold.nii.gz', '/Use
rs/gracer/Desktop/data/sub-10271/func/sub-10271_task-bart_bold_brain.ni
i.gz', '/Users/gracer/Desktop/data/sub-10271/func/sub-10271_task-bart_b
old_brain_mask.nii.gz', '/Users/gracer/Desktop/data/sub-10271/func/sub-
10271_task-bart_bold_brain_mcf.nii.gz', '/Users/gracer/Desktop/data/sub
-10271/func/sub-10271_task-bht_bold.nii.gz', '/Users/gracer/Desktop/dat
a/sub-10271/func/sub-10271_task-rest_bold.nii.gz', '/Users/gracer/Deskt
op/data/sub-10271/func/sub-10271_task-scap_bold.nii.gz', '/Users/grace
r/Desktop/data/sub-10271/func/sub-10271_task-stopsignal_bold.nii.gz',
'/Users/gracer/Desktop/data/sub-10271/func/sub-10271_task-taskswitch_bo
ld.nii.gz', '/Users/gracer/Desktop/data/sub-10273/func/sub-10273_task-b
art_bold.nii.gz', '/Users/gracer/Desktop/data/sub-10273/func/sub-10273_
task-bart_bold_brain.nii.gz', '/Users/gracer/Desktop/data/sub-10273/fun
c/sub-10273_task-bart_bold_brain_mask.nii.gz', '/Users/gracer/Desktop/d
ata/sub-10273/func/sub-10273_task-bart_bold_brain_mcf.nii.gz', '/Users/
gracer/Desktop/data/sub-10273/func/sub-10273_task-bht_bold.nii.gz', '/U
sers/gracer/Desktop/data/sub-10273/func/sub-10273_task-rest_bold.nii.g
z', '/Users/gracer/Desktop/data/sub-10273/func/sub-10273_task-scap_bol
d.nii.gz', '/Users/gracer/Desktop/data/sub-10273/func/sub-10273_task-st
opsignal_bold.nii.gz', '/Users/gracer/Desktop/data/sub-10273/func/sub-1
0273_task-taskswitch_bold.nii.gz', '/Users/gracer/Desktop/data/sub-1027
4/func/sub-10274_task-bart_bold.nii.gz', '/Users/gracer/Desktop/data/su
b-10274/func/sub-10274_task-bart_bold_brain.nii.gz', '/Users/gracer/Des
ktop/data/sub-10274/func/sub-10274_task-bart_bold_brain_mask.nii.gz',
'/Users/gracer/Desktop/data/sub-10274/func/sub-10274_task-bart_bold_bra
in_mcf.nii.gz', '/Users/gracer/Desktop/data/sub-10274/func/sub-10274_ta
sk-rest_bold.nii.gz', '/Users/gracer/Desktop/data/sub-10274/func/sub-10
274_task-scap_bold.nii.gz', '/Users/gracer/Desktop/data/sub-10274/func/
sub-10274_task-stopsignal_bold.nii.gz', '/Users/gracer/Desktop/data/sub
-10274/func/sub-10274_task-taskswitch_bold.nii.gz', '/Users/gracer/Desk
top/data/sub-10280/func/sub-10280_task-bart_bold.nii.gz', '/Users/grace
r/Desktop/data/sub-10280/func/sub-10280_task-bart_bold_brain.nii.gz',
'/Users/gracer/Desktop/data/sub-10280/func/sub-10280_task-bart_bold_bra
in_mask.nii.gz', '/Users/gracer/Desktop/data/sub-10280/func/sub-10280_t
ask-bart_bold_brain_mcf.nii.gz', '/Users/gracer/Desktop/data/sub-10280/
func/sub-10280_task-bht_bold.nii.gz', '/Users/gracer/Desktop/data/sub-1
0280/func/sub-10280_task-rest_bold.nii.gz', '/Users/gracer/Desktop/dat
a/sub-10280/func/sub-10280_task-scap_bold.nii.gz', '/Users/gracer/Deskt
op/data/sub-10280/func/sub-10280_task-stopsignal_bold.nii.gz', '/Users/
gracer/Desktop/data/sub-10280/func/sub-10280_task-taskswitch_bold.nii.g
z', '/Users/gracer/Desktop/data/sub-10290/func/sub-10290_task-bart_bol
d.nii.gz', '/Users/gracer/Desktop/data/sub-10290/func/sub-10290_task-ba
rt_bold_brain.nii.gz', '/Users/gracer/Desktop/data/sub-10290/func/sub-1
0290_task-bart_bold_brain_mask.nii.gz', '/Users/gracer/Desktop/data/sub
-10290/func/sub-10290_task-bart_bold_brain_mcf.nii.gz', '/Users/gracer/
Desktop/data/sub-10290/func/sub-10290_task-bht_bold.nii.gz', '/Users/gr
acer/Desktop/data/sub-10290/func/sub-10290_task-rest_bold.nii.gz', '/Us
ers/gracer/Desktop/data/sub-10290/func/sub-10290_task-scap_bold.nii.g
z', '/Users/gracer/Desktop/data/sub-10290/func/sub-10290_task-stopsigna
l_bold.nii.gz', '/Users/gracer/Desktop/data/sub-10290/func/sub-10290_ta
sk-taskswitch_bold.nii.gz', '/Users/gracer/Desktop/data/sub-10292/func/
sub-10292_task-bart_bold.nii.gz', '/Users/gracer/Desktop/data/sub-1029
2/func/sub-10292_task-bart_bold_brain.nii.gz', '/Users/gracer/Desktop/d
ata/sub-10292/func/sub-10292_task-bart_bold_brain_mask.nii.gz', '/User
s/gracer/Desktop/data/sub-10292/func/sub-10292_task-bart_bold_brain_mc
f.nii.gz', '/Users/gracer/Desktop/data/sub-10292/func/sub-10292_task-bh
t_bold.nii.gz', '/Users/gracer/Desktop/data/sub-10292/func/sub-10292_ta

```
sk-rest_bold.nii.gz', '/Users/gracer/Desktop/data/sub-10292/func/sub-10
292_task-scap_bold.nii.gz', '/Users/gracer/Desktop/data/sub-10292/func/
sub-10292_task-stopsignal_bold.nii.gz', '/Users/gracer/Desktop/data/sub
-10292/func/sub-10292_task-taskswitch_bold.nii.gz']
```

1. funcrun

In [ ]:
```
def create_fsf():
    for item in glob.glob(os.path.join(basedir,'sub-*')):
        sub=item.split('/')[5]
        funcrun=(os.path.join(item,'func','%s_task-%s_bold_brain_mcf.ni
i.gz')%(sub,arglist['TASK']))
        repl_dict.update({'FUNCRUN':funcrun})
```

2. NTIMEPOINTS = this is the number of timepoints per image

3. TRS = this is the TRs in the nifti

In [ ]:
```
def create_fsf():
    for item in glob.glob(os.path.join(basedir,'sub-*')):
        sub=item.split('/')[5]
        funcrun=(os.path.join(item,'func','%s_task-%s_bold_brain_mcf.ni
i.gz')%(sub,arglist['TASK']))
        repl_dict.update({'FUNCRUN':funcrun})

        ntmpts=check_output(['fslnvols',funcrun])
        repl_dict.update({'NTIMEPOINTS':ntmpts})

        trs=check_output(['fslval','%s'%(funcrun),'pixdim4',scan])
        print(trs)
        repl_dict.update({'TRS':trs})
```

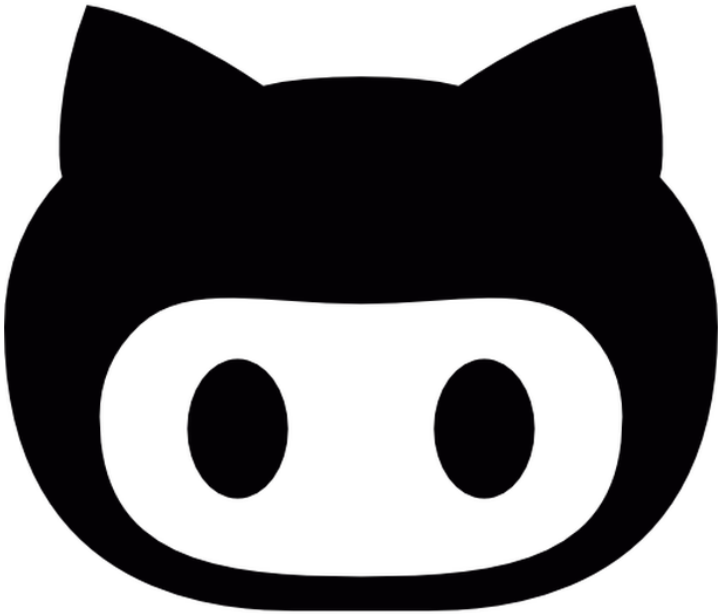
4. OUTPUT = what we want to call our output directory, it will be something .feat

5. ANAT = this is our T1W image, if we are using FNIRT it needs have skull, if we are using FLIRT we want the defaced (betted)

In [ ]:
```
        output=os.path.join(outdir,sub,'grace_edit',arglist['TASK'])
        repl_dict.update({'OUTPUT':output})
        anat=os.path.join(basedir,sub,'anat','%s_T1w_brain.nii.gz'%(sub
))
        repl_dict.update({'ANAT':anat})
```

6. CONFOUNDS = this is the confound.txt file generated from fsl_motion_outliers that we should have in our motion assessment directory if our prepro script worked

In [ ]:
```
        confounds=os.path.join(basedir,sub,'func','motion_assessment','%
s_task-%s_bold_brain_confound.txt'%(sub,arglist['TASK']))
        repl_dict.update({'CONFOUNDS':confounds})
```



7. EV = this is the path to the explanatory variable text file we want to use (we will have more than 1), each needs to be in the 3 column format

8. EVTITLE = this is a useful name for each EV

This is going to be a tough one. We have a certain number of EVs we have defined using our argparse. How are we going to match it up with the EVs and EV titles? Let's take a look at what the EV files look like.

sub-10171_bart_action_EXPLODE_output.txt

What parts of this do we need to be able to swap out?

1. subject
2. task
3. action type (EXPLODE, CASHOUT, ACCEPT)

Concept Check: Where have we already defined these? Or do we need to define them now?

1. subject = we have defined this as sub
2. task = we have this in our arglist dictionary
3. action type (EXPLODE, CASHOUT, ACCEPT) = we have this in our arglist dictionary

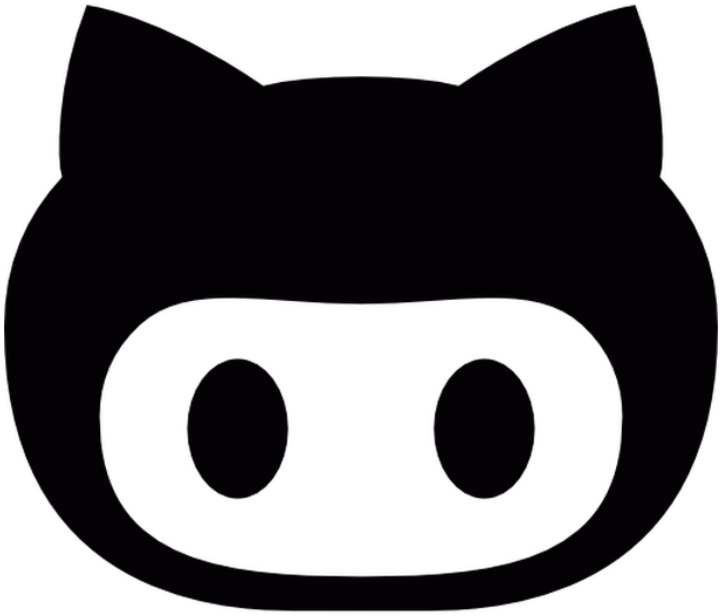
We can use a counter to loop through our EVs. This gives us the numbers we will need in the EV title.

In [ ]:
```
        ctr=0
        for item in arglist['EV']:
            print(item)
            ctr=ctr+1
            repl_dict.update({'EV%iTITLE'%ctr:item})
            ev=os.path.join(basedir,sub,'func','onsets','%s_%s_%s_outpu
t.txt'%(sub,arglist['TASK'],item))
            repl_dict.update({'EV%i'%ctr:ev})
```

9. MOTCOR = this is the path to the motion parameters we generated in with the preprocessing script

Similar to the EVs we will need to use a counter for the motion parameters. Unlike the EVs though, we will always only have 6 motion parameters. We can use the range function as a counter!

In [ ]:
```
    for i in range(6):
        motcor=os.path.join(basedir,sub,'func','motion_assessment','%s_t
ask-%s_bold_brain_motcor%i.txt' %(sub,arglist['TASK'],i))
        repl_dict.update({'MOTCOR%i'%i:motcor})
```



Let's add a print statement incase we need to debug

In [ ]:
```
print(repl_dict)
```

We should now have something that looks like this

In [ ]:
```
def create_fsf(basedir,repl_dict,outdir, arglist):
    os.chdir(basedir)
    for sub in glob.glob('sub-*'):
        repl_dict.update({'SUB':sub})

        scan=(os.path.join(sub,'func','%s_task-%s_bold_brain_mcf.nii.gz'
)%(sub,arglist['TASK']))

        funcrun=os.path.join(basedir,scan)
        repl_dict.update({'FUNCRUN':funcrun})

        ntmpts=check_output(['fslnvols',funcrun])
        repl_dict.update({'NTIMEPOINTS':ntmpts})

        trs=check_output(['fslval','%s'%(funcrun),'pixdim4',scan])
        print(trs)
        repl_dict.update({'TRS':trs})


        output=os.path.join(outdir,sub,'grace_edit',arglist['TASK'])
        repl_dict.update({'OUTPUT':output})
        anat=os.path.join(basedir,sub,'anat','%s_T1w_brain.nii.gz'%(sub
))
        repl_dict.update({'ANAT':anat})

        confounds=os.path.join(basedir,sub,'func','motion_assessment','%
s_task-%s_bold_brain_confound.txt'%(sub,arglist['TASK']))
        repl_dict.update({'CONFOUNDS':confounds})
        ctr=0
        for item in arglist['EV']:
            print(item)
            ctr=ctr+1
            repl_dict.update({'EV%iTITLE'%ctr:item})
            ev=os.path.join(basedir,sub,'func','onsets','%s_%s_%s_outpu
t.txt'%(sub,arglist['TASK'],item))
            repl_dict.update({'EV%i'%ctr:ev})
        for i in range(6):
            motcor=os.path.join(basedir,sub,'func','motion_assessment','
%s_task-%s_bold_brain_motcor%i.txt' %(sub,arglist['TASK'],i))
            repl_dict.update({'MOTCOR%i'%i:motcor})

        print(repl_dict)
```

We have all our variables now we need to swap out the place holder variables in our .fsf files

First we need to open a file to read

```
with open(os.path.join(basedir,'design.fsf'),'r') as infile:
```

Second we need to read the file.

You may recall we previously have used the function readlines(), this time we don't want to read in the file as a list. We want to read in the file as a string.

```
tempfsf=infile.read()
```

Third we need to loop through our dictionary using the key

```
for key in repl_dict:
    tempfsf = tempfsf.replace(key, repl_dict[key])
```

We can use the replace function to replace the variables in the fsf file with the values in the dictionary.

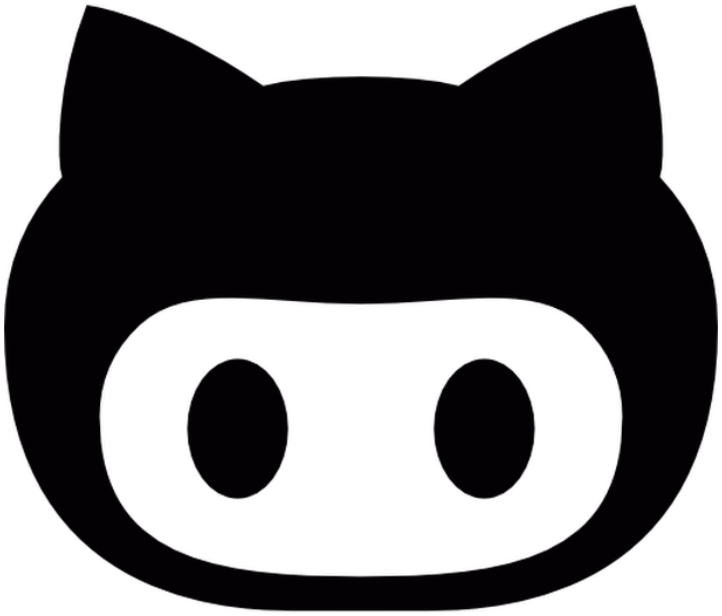
Concept Check: the replace function requires a string argument. If we decided to use the readlines function instead of the read function, would replace still work?

Now that we have replaced the variables in the fsf with the key values we can write the strings to a new file

```
with open(os.path.join(outdir,sub,'%s_%s.fsf'%(sub,arglist['TASK'])),'w') as
 outfile:
    outfile.write(tempfsf)
outfile.close()
```

All Together Now

In [ ]:
```
def create_fsf(basedir,repl_dict,outdir, arglist):
    os.chdir(basedir)
    for sub in glob.glob('sub-*'):
        repl_dict.update({'SUB':sub})

        scan=(os.path.join(sub,'func','%s_task-%s_bold_brain_mcf.nii.gz'
)%(sub,arglist['TASK']))

        funcrun=os.path.join(basedir,scan)
        repl_dict.update({'FUNCRUN':funcrun})

        ntmpts=check_output(['fslnvols',funcrun])
        repl_dict.update({'NTIMEPOINTS':ntmpts})

        trs=check_output(['fslval','%s'%(funcrun),'pixdim4',scan])
        print(trs)
        repl_dict.update({'TRS':trs})


        output=os.path.join(outdir,sub,'grace_edit',arglist['TASK'])
        repl_dict.update({'OUTPUT':output})
        anat=os.path.join(basedir,sub,'anat','%s_T1w_brain.nii.gz'%(sub
))
        repl_dict.update({'ANAT':anat})

        confounds=os.path.join(basedir,sub,'func','motion_assessment','%
s_task-%s_bold_brain_confound.txt'%(sub,arglist['TASK']))
        repl_dict.update({'CONFOUNDS':confounds})
        ctr=0
        for item in arglist['EV']:
            print(item)
            ctr=ctr+1
            repl_dict.update({'EV%iTITLE'%ctr:item})
            ev=os.path.join(basedir,sub,'func','onsets','%s_%s_%s_outpu
t.txt'%(sub,arglist['TASK'],item))
            repl_dict.update({'EV%i'%ctr:ev})
        for i in range(6):
            motcor=os.path.join(basedir,sub,'func','motion_assessment','
%s_task-%s_bold_brain_motcor%i.txt' %(sub,arglist['TASK'],i))
            repl_dict.update({'MOTCOR%i'%i:motcor})

        print(repl_dict)
        if arglist['NOREG']==False:
            with open(os.path.join(basedir,'design.fsf'),'r') as infile:
                tempfsf=infile.read()
                for key in repl_dict:
                    tempfsf = tempfsf.replace(key, repl_dict[key])
                    with open(os.path.join(outdir,sub,'%s_%s.fsf'%(sub,a
rglist['TASK'])),'w') as outfile:
                        outfile.write(tempfsf)
                    outfile.close()
            infile.close()
```



But wait... what about the registration issue???

To address this we can use an if/else statement with our arglist

Give it a try first

In [ ]:
```
if arglist['NOREG']==False:
            with open(os.path.join(basedir,'design.fsf'),'r') as infile:
                tempfsf=infile.read()
                for key in repl_dict:
                    tempfsf = tempfsf.replace(key, repl_dict[key])
                    with open(os.path.join(outdir,sub,'%s_%s.fsf'%(sub,a
rglist['TASK'])),'w') as outfile:
                        outfile.write(tempfsf)
                    outfile.close()
                infile.close()

        else:
            print("skipping registration")
            with open(os.path.join(basedir,'no_reg_design.fsf'),'r') as
infile:
                tempfsf=infile.read()
                for key in repl_dict:
                    tempfsf = tempfsf.replace(key, repl_dict[key])
                    with open(os.path.join(outdir,sub,'%s_%s_no_reg.fsf'
%(sub,arglist['TASK'])),'w') as outfile:
                        outfile.write(tempfsf)
                    outfile.close()
                infile.close()
```

All Together Now!

In [ ]:
```
#!/usr/bin/env python

import glob
import os
from subprocess import check_output
#import pdb
import argparse


def create_fsf(basedir,repl_dict,outdir, arglist):
    os.chdir(basedir)
```

```
    for sub in glob.glob('sub-*'):
        repl_dict.update({'SUB':sub})

        scan=(os.path.join(sub,'func','%s_task-%s_bold_brain_mcf.nii.gz'
)%(sub,arglist['TASK']))

        funcrun=os.path.join(basedir,scan)
        repl_dict.update({'FUNCRUN':funcrun})

        ntmpts=check_output(['fslnvols',funcrun])
        repl_dict.update({'NTIMEPOINTS':ntmpts})

        trs=check_output(['fslval','%s'%(funcrun),'pixdim4',scan])
        print(trs)
        repl_dict.update({'TRS':trs})


        output=os.path.join(outdir,sub,'grace_edit',arglist['TASK'])
        repl_dict.update({'OUTPUT':output})
        anat=os.path.join(basedir,sub,'anat','%s_T1w_brain.nii.gz'%(sub
))
        repl_dict.update({'ANAT':anat})

        confounds=os.path.join(basedir,sub,'func','motion_assessment','%
s_task-%s_bold_brain_confound.txt'%(sub,arglist['TASK']))
        repl_dict.update({'CONFOUNDS':confounds})
        ctr=0
        for item in arglist['EV']:
            print(item)
            ctr=ctr+1
            repl_dict.update({'EV%iTITLE'%ctr:item})
            ev=os.path.join(basedir,sub,'func','onsets','%s_%s_%s_outpu
t.txt'%(sub,arglist['TASK'],item))
            repl_dict.update({'EV%i'%ctr:ev})
        for i in range(6):
            motcor=os.path.join(basedir,sub,'func','motion_assessment','
%s_task-%s_bold_brain_motcor%i.txt' %(sub,arglist['TASK'],i))
            repl_dict.update({'MOTCOR%i'%i:motcor})

        print(repl_dict)
        if arglist['NOREG']==False:
            with open(os.path.join(basedir,'design.fsf'),'r') as infile:
                tempfsf=infile.read()
                for key in repl_dict:
                    tempfsf = tempfsf.replace(key, repl_dict[key])
                    with open(os.path.join(outdir,sub,'%s_%s.fsf'%(sub,a
rglist['TASK'])),'w') as outfile:
                        outfile.write(tempfsf)
                    outfile.close()
            infile.close()

        else:
            print("skipping registration")
            with open(os.path.join(basedir,'no_reg_design.fsf'),'r') as
infile:
                tempfsf=infile.read()
                for key in repl_dict:
```

```
                    tempfsf = tempfsf.replace(key, repl_dict[key])
                    with open(os.path.join(outdir,sub,'%s_%s_no_reg.fsf'
%(sub,arglist['TASK'])),'w') as outfile:
                        outfile.write(tempfsf)
                    outfile.close()
            infile.close()
    os.chdir('/Users/gracer/Google Drive/fMRI_workshop/scripts/feat_scri
pts')


def main ():
  basedir='/Users/gracer/Desktop/data'
  outdir=os.path.join(basedir,'derivatives','task')
  parser=argparse.ArgumentParser(description='making fsf files')
  parser.add_argument('-noreg',dest='NOREG', action='store_true',
                        default=False, help='Did you already register yo
ur data (using ANTZ maybe)?')
  parser.add_argument('-task',dest='TASK',
                        default=False, help='which task are we using?')
  parser.add_argument('-evs',dest='EV',nargs='+',
                        default=False, help='which evs are we using?')

  repl_dict={}
  args = parser.parse_args()
  arglist={}
  for a in args._get_kwargs():
      arglist[a[0]]=a[1]
      print(arglist)
  create_fsf(basedir,repl_dict, outdir, arglist)
main()
os.chdir('/Users/gracer/Google Drive/fMRI_workshop/scripts/feat_scripts'
)
```

Give it a run and then open a random one in Feat and we will check that it populated correctly

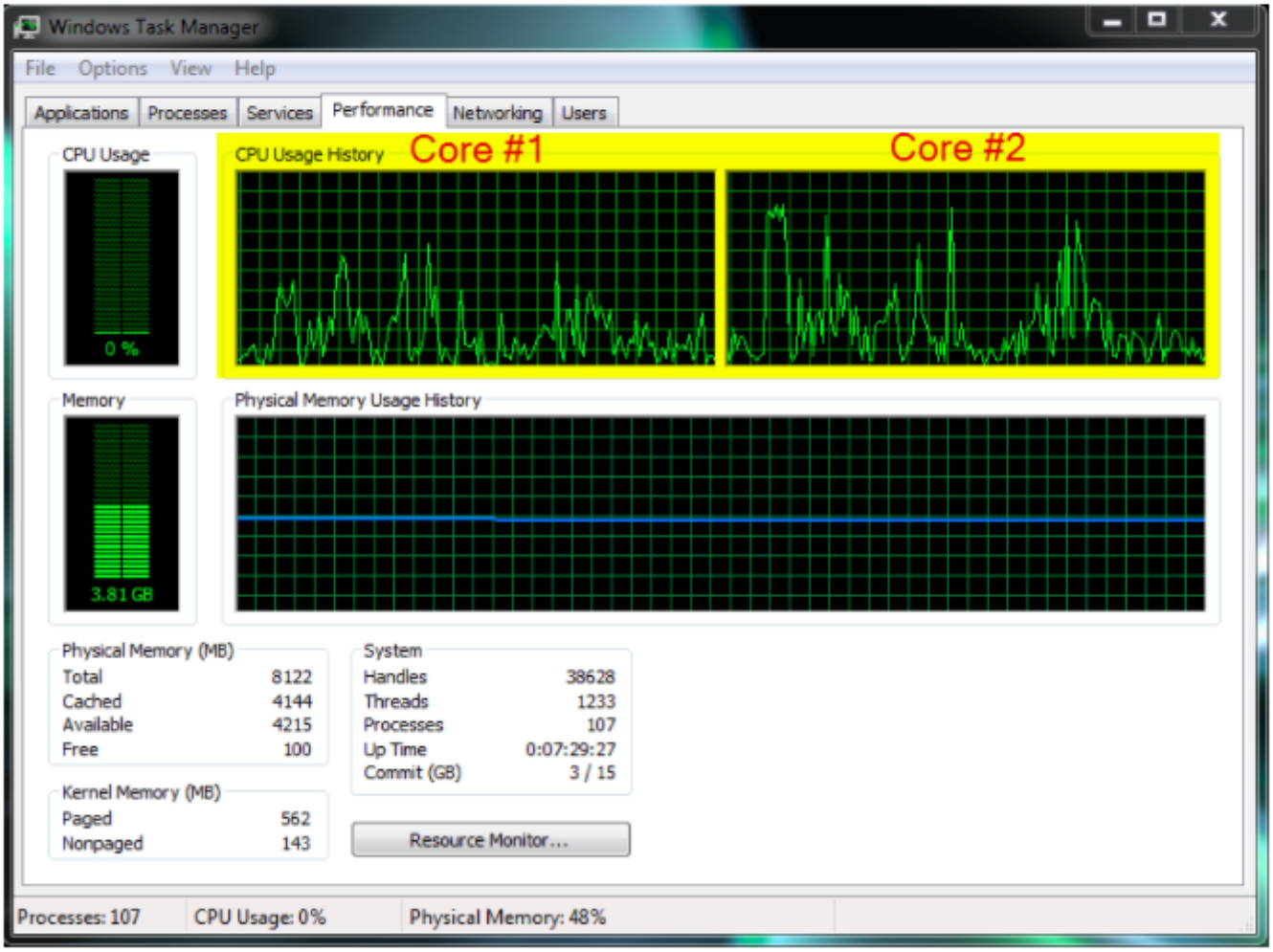
If everything looks good we will move on to creating a parallel feat launcher!

First lets see how many cores we have available

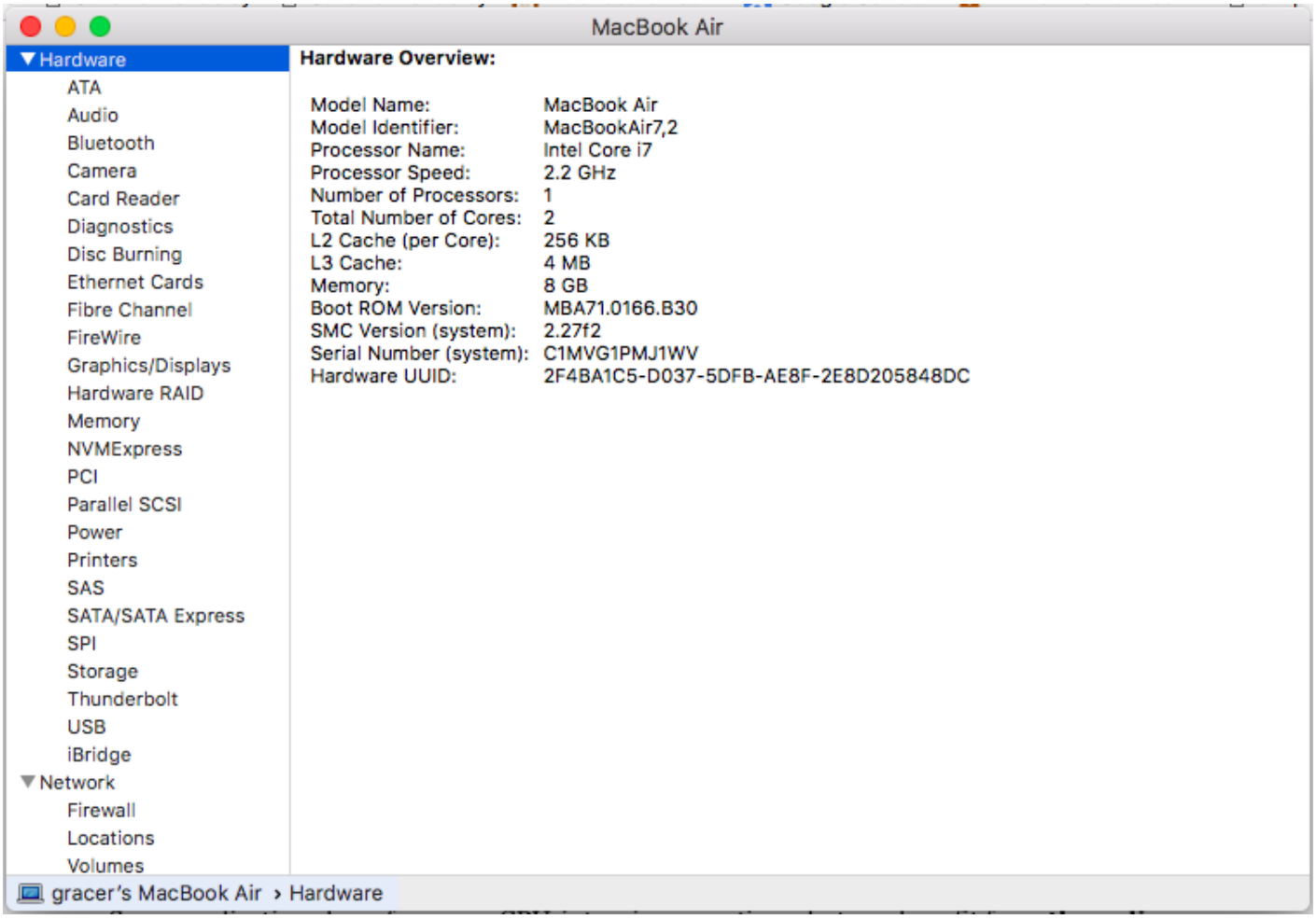
If you are running on pc

Do a Ctrl + Shift + Esc. This will open the **Windows Task Manager**. Once you are here, go to **Performance**. Now you should see many boxes in the **CPU Usage History** section which will identify how many cores you have. This will include hyper threaded cores also.

-Hope this helps.



If you are running on mac



Looks like I have 2 cores available, which means I have 4 cores including virtual cores

You don't want to max out and use all 4 cores though... You need some of those cores to do normal computer stuff For this tutorial we will assume we only need to increase our efficiency by 2, but if you were say on a High Performance Computing Cluster or a super beefy machine you could theoretically increase the number

Open a new blank script in Spyder and call this level1.py

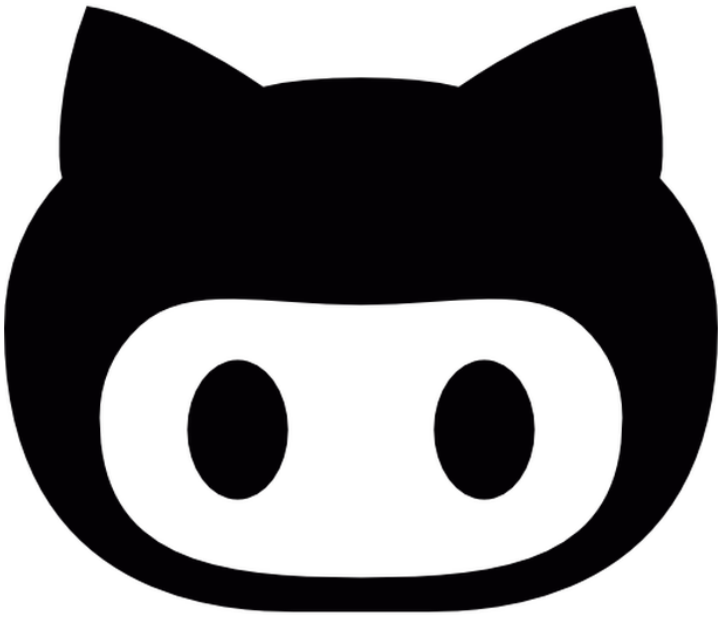
In [ ]:
```
import os
import glob
from multiprocessing import Pool
```

The os and glob should be old friends by now, but we are also going to use multiprocessing which is going to allow us to split our analysis over multiple cores

This is going to be one of the few scripts that does follow our predictable pattern

1. We are going to get all our fsf files in to one large list

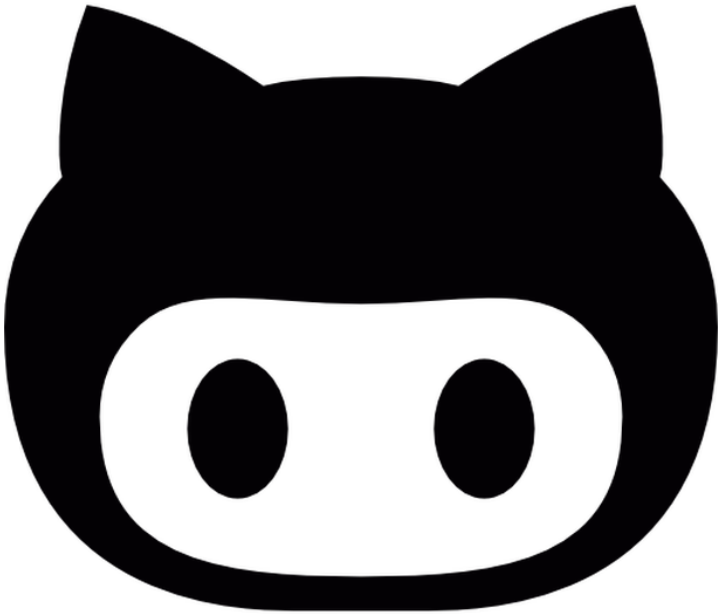
Concept check: Create a variable called all_data which is a list of all the .fsf files we created previously



You should have something that looks like this:

In [ ]:
```
basedir='/Users/gracer/Desktop/data/derivatives/task'
all_data=glob.glob(os.path.join(basedir,'sub*','sub*bart.fsf'))
```

Concept Challenge: Write a function called split_list. It should take any list of any length and create two lists, one with the top half of the list and one with the bottom



There are a lot of ways you could accomplish this, but here is what I found on stackover flow that I really liked

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

Concept check: supposed my computer was tricked out and I was able to run 16 different lists at once. How would I change the above code to make 16 lists?

If this tutorial is too slow, try this and we will check back in

Next let's write a function that will loop through a list and run the feat command. Remember feat is a linux command so we will need to use a wrapper function.

Concept Check: Write the above function call it run_level1

Make sure you don't capitalize feat, else it will launch a GUI

Here is what I came up with

In [ ]:
```
def run_level1(DATA):
    for item in DATA:
        print('starting to run on %s'%item)
        os.system("feat %s"%item)
```

Up to this point we haven't done anything new.

Now we are going to introduce a special if statement to execute our functions in parallel

In [ ]:
```
if __name__ == '__main__':
    pool = Pool(processes=2)
    pool.map(run_level1, [B,C])
    pool.close()
    pool.join()
```

What is this nonsense???

```
if __name__ == '__main__':
```

So far we have kept it simple describing how python executes scripts (top to bottom, left to right, etc)... To get a little more into it, when python reads a file it sets up a variety of special variables. One is called:

```
__name__
```

This is telling python if the script is begin executed as a moduled being imported (like we have done with glob or os so far) or if it is being run as a standalone. When

```
__name__== main
```

then it is a standalone. Alternatively, when we import glob

```
__name__  == glob
```

and python recognizes it as an import.

So essentially we are telling python, if we execute this program as a standalone script start the following actions. If you want to know more about this, there is a good summary here https://stackoverflow.com/questions/419163/what-does-if-name-main-do (https://stackoverflow.com/questions/419163/what-does-if-name-main-do).

```
pool=Pool(processes=2)
```

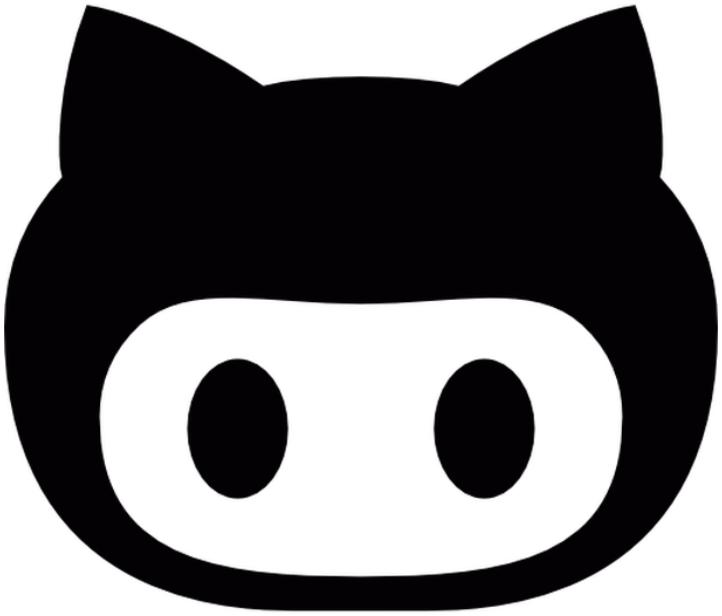
Is opening a pool of worker processess in which data can be run in parallel (which is why we are splitting our dataset)

```
pool.map(run_level1, [B,C])
```

Is mapping a function (in our case run_level1) over both lists B and C. If you want to know more: http://chriskiehl.com/article/parallelism-in-one-line/ (http://chriskiehl.com/article/parallelism-in-one-line/)

In total we should have something that looks like this:

In [ ]:
```
#!/usr/bin/env python2
# -*- coding: utf-8 -*-
"""
Created on Wed Nov 29 16:11:03 2017

@author: gracer
"""
import os
import glob
from multiprocessing import Pool



basedir='/Users/gracer/Desktop/data/derivatives/task'
all_data=glob.glob(os.path.join(basedir,'sub*','sub*bart.fsf'))
def split_list(a_list):
        half = len(a_list)/2
        return a_list[:half], a_list[half:]

B, C = split_list(all_data)

def run_level1(DATA):
    for item in DATA:
        print('starting to run on %s'%item)
        os.system("feat %s"%item)

if __name__ == '__main__':
   pool = Pool(processes=2)
   pool.map(run_level1, [B,C])
   pool.close()
   pool.join()
```



Go ahead and run that

Make sure you don't have any debug statements though... we won't have a way to get out of them

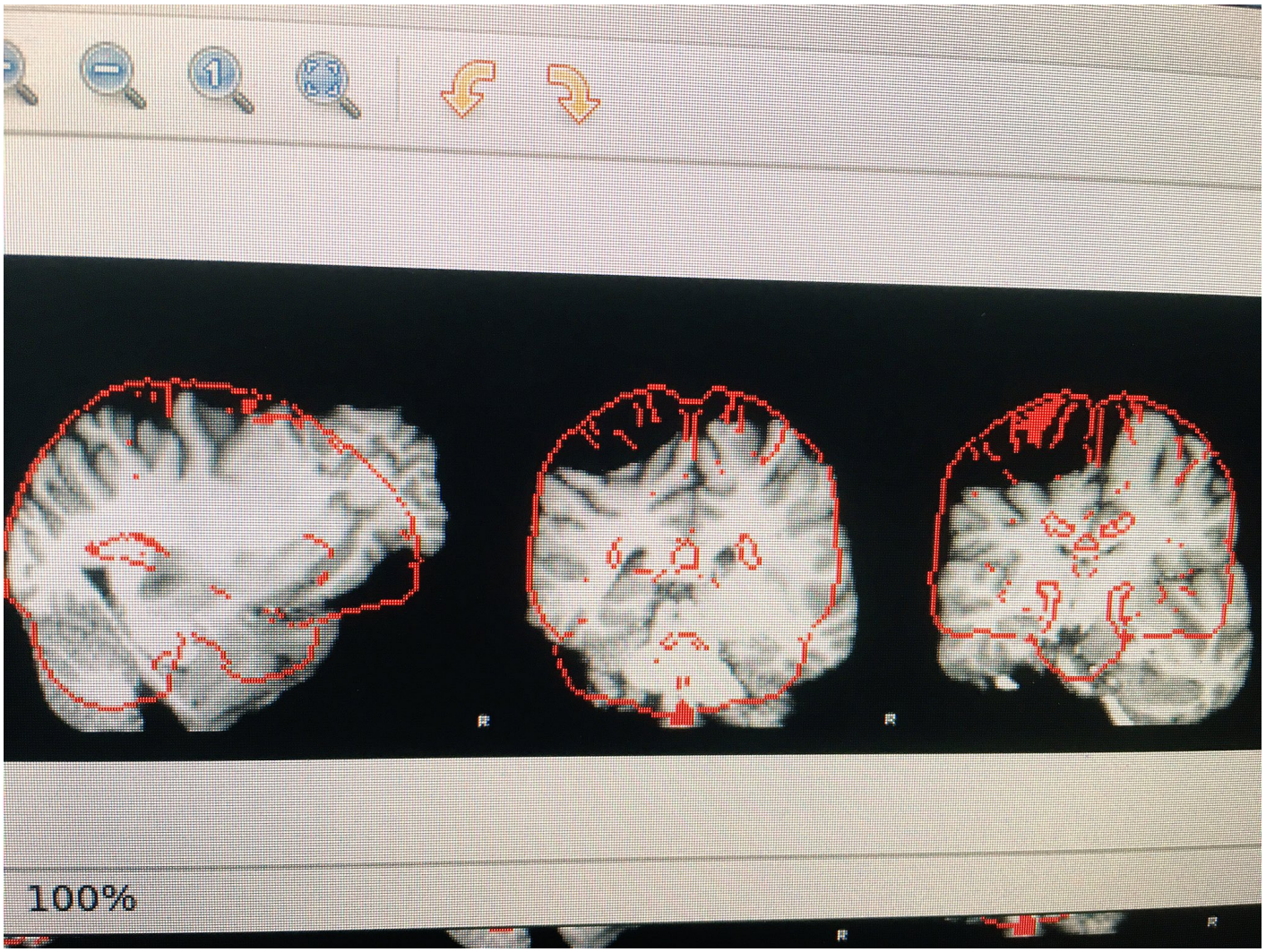
While we are waiting for those to run, lets make a first level QA check

Heads up this is going to be a lot of concept checks! You are pretty much gonna write this on your own!

Like always lets make two functions: QA_writer( ) and main ( )

Next we are going to need think about what indicates a complete feat analysis

Personally, I like to check the number of stats files produced. We are also going to want to check the registration and the stats.



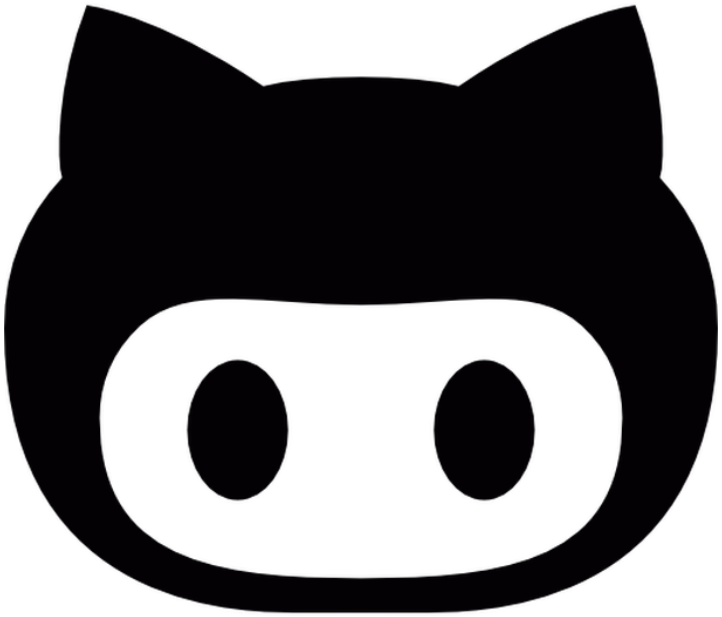
fail registration credit: https://twitter.com/DanielaJPalombo (https://twitter.com/DanielaJPalombo)

The registration and the registration and stats are all contained in png images. This is a great oppertunity to create another html file like we did for the motion correction!

First lets consider the main() function. What globals should be define? Remember we want to make an html

1. basedir
2. writedir
3. datestamp
4. outfile

Concept Check: Fill in your main( ) so that you define the above variables.



In [ ]:
```
def main():
    basedir='/Users/gracer/Desktop/data/derivatives/task'
    writedir='/Users/gracer/Desktop/data/'
    datestamp=datetime.datetime.now().strftime("%Y-%m-%d-%H_%M_%S")
    outfile = os.path.join(writedir,'lev1_QA_%s.html'%datestamp)
    QA_writer(basedir,outfile,writedir)
```

Of course your paths will be different!

The main( ) is looking good! Lets move on to the QA_writer( ).

Concept Check: We are going to need something that gets all the file paths to the first level analyses and also that generates a subject number variable. Give it a shot!

I came up with the following:

In [ ]:
```
def QA_writer(basedir,outfile,writedir):

    for file in glob.glob(os.path.join(basedir,'sub*','grace_edit','*.fe
at')):
        os.chdir(file)
        print(file)
        sub=file.split('/')[7]
```

Since we are looking for png files in multiple subdirectories we could write multiple loops to look for each.... Or we can use the walk function from os to search each subdirectory for us!

In [ ]:
```
        dict_of_files = {}
        for (dirpath, dirnames, filenames) in os.walk(file):
            for filename in filenames:
                if filename.startswith('fsl') or filename.startswith('ve
rt'):
                    print('skipping')
                elif filename.endswith('.png'):
                    list_of_files[filename] = os.sep.join([dirpath, file
name])
```

os.walk is a little intimidating at first. It generates the following:

- dirpath= the path to the directory
- dirnames= a list of subdirectories
- filenames= list of non-directory files by "walking" through the directory tree either up or down. So we can give it a file path and ask it to generate a the dirpath, names of all the inner directories, and the filenames. With in those, we can pick out which we want to print to the html. All this and we don't need to know the specific names of all the subdirectories!

So what is going on with this if/elif statement?

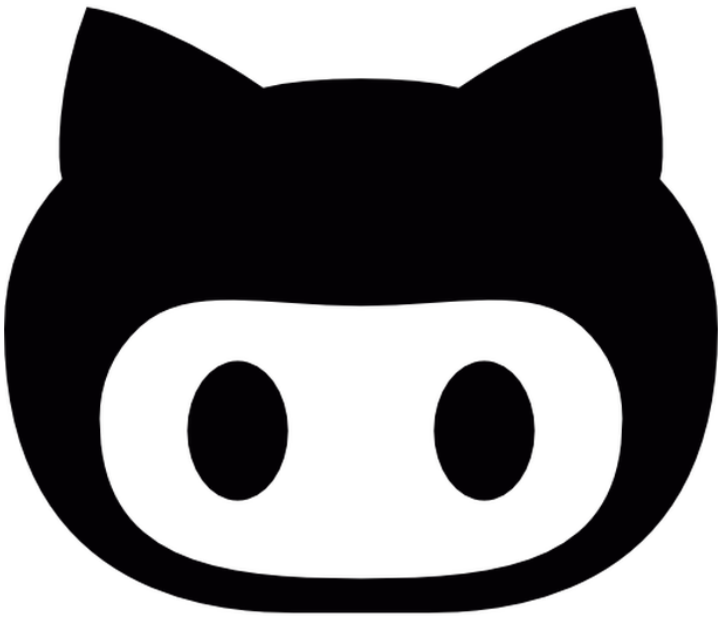
- We want to find all the png files associated with registration and the statistical analysis. So we could just glob all png files. BUT.... fsl generates a bunch of extra png files (logos, etc), and they clutter the html. We can skip file names that match "clutter" and keep the rest.
- elif is a combination of else if. It allows us to pass an else statement with a condition after an if.
- Finally we are going to population our dictionary dict_of_files with the filename as the key and the path to the file as the variable.

Concept check: Now that we have a dictionary of our png files, create a loop that will get the file and input it into html formatting. If you are stuck look how we did this in the preprocessing script

In [ ]:
```
   for key in dict_of_files:
            os.system("echo '<p>=============<p> %s %s <br><IMG BORDER=0
 SRC=%s WIDTH=%s></BODY></HTML>' >> %s"%(sub,key,list_of_files[key],'10
0%', outfile))
#            shutil.copy(dict_of_files[key],writedir)
```

Concept check: Finally, finish off the script with something that checks the number of cope files in each feat. If a feat directory is miss copes move it to a folder called fail

In [ ]:
```
        if os.path.exists(os.path.join(basedir,'fails'))==False:
            os.makedirs(os.path.join(basedir,'fails'))

        if len(glob.glob(os.path.join(file,'stats','cope*.nii.gz')))==5:
            print(file+' has 5 cope files :D')
        else:
            print(file+' is missing copes, need to rerun')
            name=file.split('/')[9].split('.')[0]
            shutil.move(file,os.path.join(basedir,'fails',sub,name))
```



All together now!

In [ ]:
```
import os
import glob
import shutil
#import pdb
import datetime

def QA_writer(basedir,outfile,writedir):

    for file in glob.glob(os.path.join(basedir,'sub*','grace_edit','*.fe
at')):
        os.chdir(file)
        print(file)
        sub=file.split('/')[7]
        dict_of_files = {}
        for (dirpath, dirnames, filenames) in os.walk(file):
            for filename in filenames:
                if filename.startswith('fsl') or filename.startswith('ve
rt'):
                    print('skipping')
                elif filename.endswith('.png'):
                    dict_of_files[filename] = os.sep.join([dirpath, file
name])

        for key in dict_of_files:
            os.system("echo '<p>=============<p> %s %s <br><IMG BORDER=0
 SRC=%s WIDTH=%s></BODY></HTML>' >> %s"%(sub,key,dict_of_files[key],'10
0%', outfile))
            #shutil.copy(dict_of_files[key],writedir)
        if os.path.exists(os.path.join(basedir,'fails'))==False:
            os.makedirs(os.path.join(basedir,'fails'))

        if len(glob.glob(os.path.join(file,'stats','cope*.nii.gz')))==4:
            print(file+' has 4 cope files :D')
        else:
            print(file+' is missing copes, need to rerun')
            name=file.split('/')[9].split('.')[0]
            shutil.copytree(file,os.path.join(basedir,'fail',sub,name))

def main():
    basedir='/Users/gracer/Desktop/data/derivatives/task'
    writedir='/Users/gracer/Desktop/data/'
    datestamp=datetime.datetime.now().strftime("%Y-%m-%d-%H_%M_%S")
    outfile = os.path.join(writedir,'lev1_QA_%s.html'%datestamp)
    QA_writer(basedir,outfile,writedir)
main()

os.chdir('/Users/gracer/Google Drive/fMRI_workshop/scripts')
```

Lets see how our feat analysis coming ...

Hey wait.... what about the no registration ones...? Seems like that is going to be an issue....

You're right we need a to do a couple extra steps!

- check that reg_standard exists in the first level
  - if it exists delete
- delete all .mat files in the reg folder
  - Replace with the identity matrix from fsl
- copy the mean_func.nii.gz to reg/standard.nii.gz
- check the voxel intensities in the stats/cope.nii.gz and reg_standard/stats/cope.nii.gz are EXACTLY the same
  - data dimension and pixel size should be the same
  - will have to do this after higher level analysis (reg_standard doesn't exist until then)

This seems like a good place to try writing a python module!

So far we have imported a lot of modules and they seem pretty awesome. Wouldn't it be cool to make one?

We briefly talked about how to create a module when we made our level1.py script. We needed that strange

```
if __name__ == '__main__':
```

We are going to use that again, because maybe sometimes we are going to want to run this script as is

Concept check: Write a python script called no_reg.py. This need to do the following:

- Check if each feat directory has a subdirectory called 'reg_standard'
  - If it does, delete it.
- Then we need to delete all the .mat files in the reg directory for each feat analysis
- Move the identity matrix (standard with fsl) to the reg directory for each feat analysis
- **Copy** the mean_func.nii.gz for each feat analysis to the reg directorys

A couple hints:

- The path to the identity matrix should look something like, '/usr/local/fsl/etc/flirtsch/ident.mat'
- We want to copy the mean_func.nii.gz, NOT move what module have we used to move and copy things?
- shutil has a function called rmtree, this will remove an entire directory tree

You should have something like this. Just as a reminder there are a lot of different ways to accomplish this task. If you have another way let me know, this is part of the process!

In [1]:
```
import os
import glob
import shutil

def reg_check(basedir,IDmat):
    for sub in glob.glob(os.path.join(basedir,'sub-*','grace_edit','*.fe
at')):
        if os.path.exists(os.path.join(sub,'reg_standard')):
            shutil.rmtree(os.path.join(sub,'reg_standard'))
            print('%s has the reg standard'%sub)

        for files in glob.glob(os.path.join(sub,'reg','*.mat')):
            print(files)
            os.remove(files)

        shutil.copy2(IDmat,os.path.join(sub,'reg'))
        meanFUNC=os.path.join(sub,'mean_func.nii.gz')
        regDIR=os.path.join(sub,'reg','standard.nii.gz')
        shutil.copy2(meanFUNC,regDIR)



def main():
    basedir='/Users/gracer/Desktop/data/derivatives/task'
    IDmat='/usr/local/fsl/etc/flirtsch/ident.mat'
    reg_check(basedir,IDmat)
```

Ok, lets make a couple tweeks so this will work as a module!

1. We are going to want to un-hard code our basedir and add BASEDIR as a parameter in main
2. We need to add that name check

In [2]:
```
import os
import glob
import shutil

def reg_check(basedir,IDmat):
    for sub in glob.glob(os.path.join(basedir,'sub-*','grace_edit','*.fe
at')):
        if os.path.exists(os.path.join(sub,'reg_standard')):
            shutil.rmtree(os.path.join(sub,'reg_standard'))
            print('%s has the reg standard'%sub)

        for files in glob.glob(os.path.join(sub,'reg','*.mat')):
            print(files)
            os.remove(files)

        shutil.copy2(IDmat,os.path.join(sub,'reg'))
        meanFUNC=os.path.join(sub,'mean_func.nii.gz')
        regDIR=os.path.join(sub,'reg','standard.nii.gz')
        shutil.copy2(meanFUNC,regDIR)




def main(BASEDIR):
    basedir=BASEDIR
    IDmat='/usr/local/fsl/etc/flirtsch/ident.mat'
    reg_check(basedir,IDmat)

if __name__ == "__main__":
    BASEDIR='/Users/gracer/Desktop/data/derivatives/task'
    main(BASEDIR)
```

This reminds us why keeping global variables in the main function is so useful. With a couple minor tweeks we were able to change a script into a module! Notice I kept the IDmat variable hardcoded. Since this is a native fsl file it **should** be the same for every installation of fsl!

Alright! Lets add this to our check_level1.py!

Concept check: Incorportate your new module into the check_level1.py! Create functionality to allow for either registration or no registration feat analysis

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

```
def QA_writer(basedir,outfile,writedir,arglist):

    if os.path.exists(os.path.join(basedir,'fails'))==False:
                os.makedirs(os.path.join(basedir,'fails'))

    for file in glob.glob(os.path.join(basedir,'sub*','grace_edit','*.fe
at')):
        sub=file.split('/')[7]
        if len(glob.glob(os.path.join(file,'stats','cope*.nii.gz')))==2:
            print(file+' has 2 cope files :D')
        else:
            print(file+' is missing copes, need to rerun')
            name=file.split('/')[9].split('.')[0]
            shutil.copytree(file,os.path.join(basedir,'fail',sub, name))

    if arglist['NOREG']== False:
        for file in glob.glob(os.path.join(basedir,'sub*','grace_edit',
'*.feat')):
            sub=file.split('/')[7]
            dict_of_files = {}
            for (dirpath, dirnames, filenames) in os.walk(file):
                for filename in filenames:
                    if filename.startswith('fsl') or filename.startswith
('vert'):
                        print('skipping')
                    elif filename.endswith('.png'):
                        dict_of_files[filename] = os.sep.join([dirpath,
filename])

            for key in dict_of_files:
                os.system("echo '<p>==============<p> %s %s <br><IMG BORD
ER=0 SRC=%s WIDTH=%s></BODY></HTML>' >> %s"%(sub,key,dict_of_files[key],
'100%', outfile))
                #shutil.copy(dict_of_files[key],writedir)
    else:
        print("no need to look at fake registration, but do need to clea
n up this reg dirctory!")
        no_reg.main(basedir)
        for file in glob.glob(os.path.join(basedir,'sub*','grace_edit',
'*.feat')):
            sub=file.split('/')[7]
            pdb.set_trace()
            for design_file in glob.glob(os.path.join(file,'design*.png'
)):
                pdb.set_trace()
                os.system("echo '<p>==============<p> %s <br><IMG BORDER=
0 SRC=%s WIDTH=%s></BODY></HTML>' >> %s"%(sub,design_file,'100%', outfil
e))


def main():
    basedir='/Users/gracer/Desktop/data/derivatives/task'
    writedir='/Users/gracer/Desktop/data/'
    datestamp=datetime.datetime.now().strftime("%Y-%m-%d-%H_%M_%S")
    outfile = os.path.join(writedir,'lev1_QA_%s.html'%datestamp)
    parser=argparse.ArgumentParser(description='checking first level fea
t analysis')
```