In [43]:
```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

import nibabel
```

In [ ]:
```
img = nibabel.load("/data/cogneuro/fMRI/motor/s01_motorloc.nii.gz")
data = img.get_data().T
ni_array_1 = data.transpose(1, 2, 3, 0)[:, 20, 20, :].T
ni_array_2 = data.transpose(2, 3, 0, 1).mean(2)[:, 50, :]
ni_array_3 = data.transpose(3, 0, 1, 2).mean(1)[:, :, 25]
ni_array_4 = data.transpose(0, 1, 2, 3).mean(0)[20, :, :]
```

1. Working with 1D and 2D arrays.

The goal of this exercise is to become more familiar with how `np.reshape` works and how sometimes different arrays use the same data. You will end up creating two time series, which you will then stack together. After that you will compute both of their means using the `np.mean` command with the appropriate `axis`.

(a) [0.5pts] Create a 1-D array of zeros of length 100 and name it `timeseries1`. Reshape `timeseries1` into a 2-D array of shape `(10, 10)`. Call the result `square1`.

In [4]:
```
## (a)

timeseries1 = np.zeros(100)
square1 = timeseries1.reshape(10, 10)
```

In [5]:
```
# This check is to ensure that you have correctly filled the variables required for autogr
ok.grade('q3_1a')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    Passed: 2
    Failed: 0
[oooooooooook] 100.0% passed
```

Out[5]:
```
{'passed': 2, 'failed': 0, 'locked': 0}
```

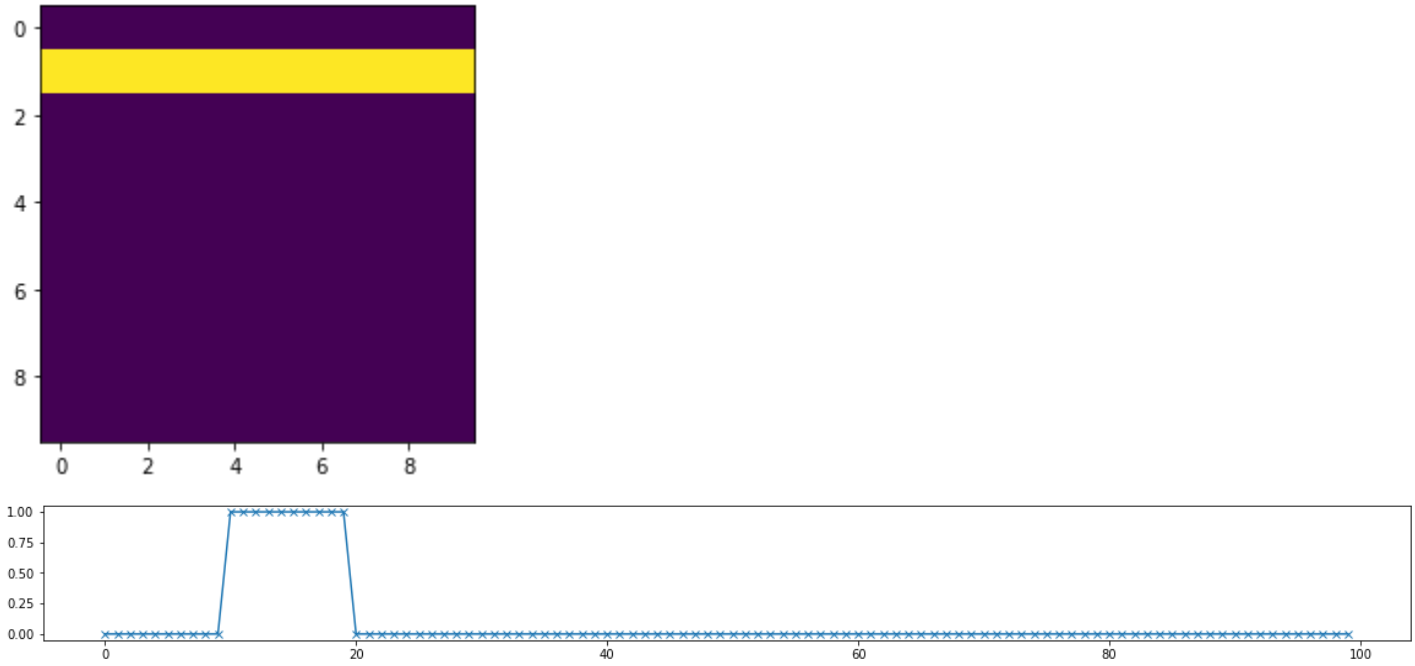
In [6]:
```
## This is for after the homework has been graded.
## When the homework is graded, you will be able to uncomment this cell and run it to see
## more details of the grading rubric.
# ok.grade('q3_1a_full')
```

(b) [1pt] Set all the values of the second row of `square1` to 1. Create a figure named `fig_sq1_1` and use `plt.imshow` to visualize `square1`. Create a figure named `fig_ts1_1` of `figsize 20, 2`. Plot `timeseries1` into it using `plt.plot`. Make sure to use the `'x-'` formatting as in the third lecture. Observe that the values of `timeseries1` have also changed.

In [7]:
```
## (b)

square1[1, :] = 1.
```

```
fig_sq1_1 = plt.figure()
plt.imshow(square1)

fig_ts1_1 = plt.figure(figsize=(20, 2))
plt.plot(timeseries1, 'x-')
```

Out[7]:
```
[<matplotlib.lines.Line2D at 0x7f650084bee0>]
```



In [8]:
```
# This check is to ensure that you have correctly filled the variables required for autogr
ok.grade('q3_1b')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    Passed: 2
    Failed: 0
[oooooooooook] 100.0% passed
```

Out[8]:
```
{'passed': 2, 'failed': 0, 'locked': 0}
```

In [9]:
```
## This is for after the homework has been graded.
# ok.grade('q3_1b_full')
```

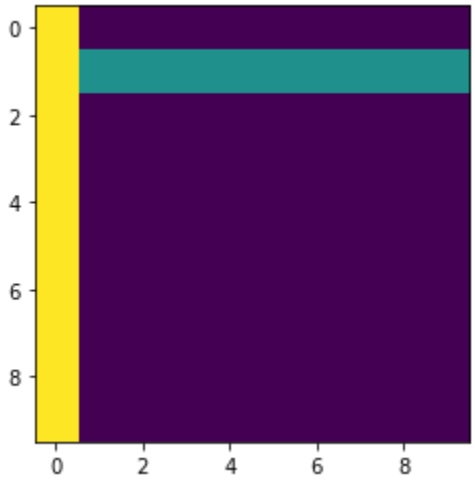
(c) [1pt] Now set all the values of the first column of `square1` to 2, and visualize both `square1` and `timeseries1` in figures `fig_sq1_2` and `fig_ts1_2`, exactly the same way as you did in part **(b)**. Next, set the bottom right quarter of `square1` to `-1` and visualize `square1` and `timeseries1` in figures `fig_sq1_3` and `fig_ts1_3` in the same way.

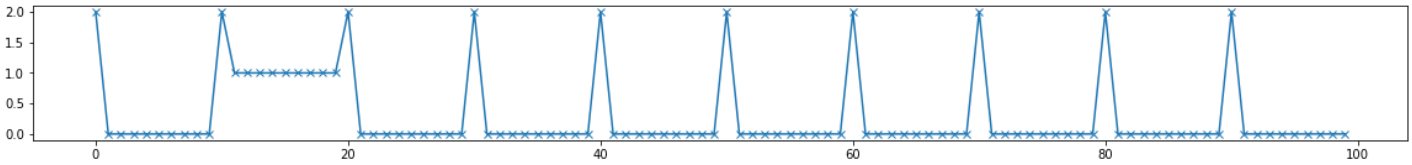
In [10]:
```
## (c)

square1[:, 0] = 2.

fig_sq1_2 = plt.figure()
plt.imshow(square1)

fig_ts1_2 = plt.figure(figsize=(20, 2))
plt.plot(timeseries1, 'x-')
```

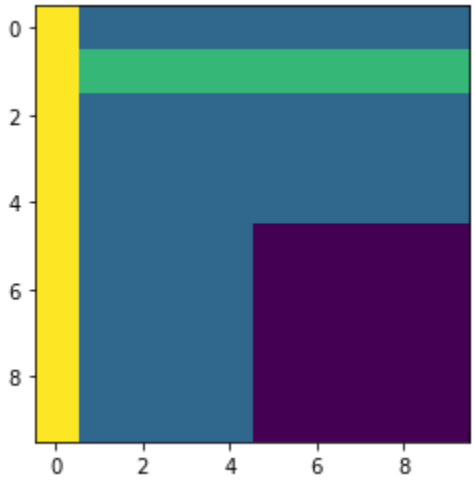
```
square1[5:, 5:] = -1.

fig_sq1_3 = plt.figure()
plt.imshow(square1)

fig_ts1_3 = plt.figure(figsize=(20, 2))
plt.plot(timeseries1, 'x-')
```

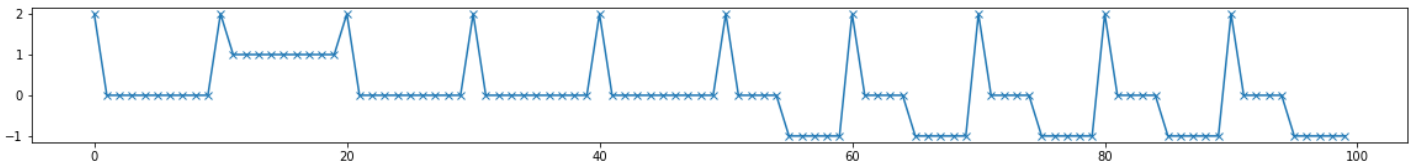
Out[10]:

```
[<matplotlib.lines.Line2D at 0x7f65006afb80>]
```









In [11]:

```
# This check is to ensure that you have correctly filled the variables required for autogr
ok.grade('q3_1c')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    Passed: 4
    Failed: 0
```

```
[ooooooooook] 100.0% passed
```

Out[11]:
```
{'passed': 4, 'failed': 0, 'locked': 0}
```

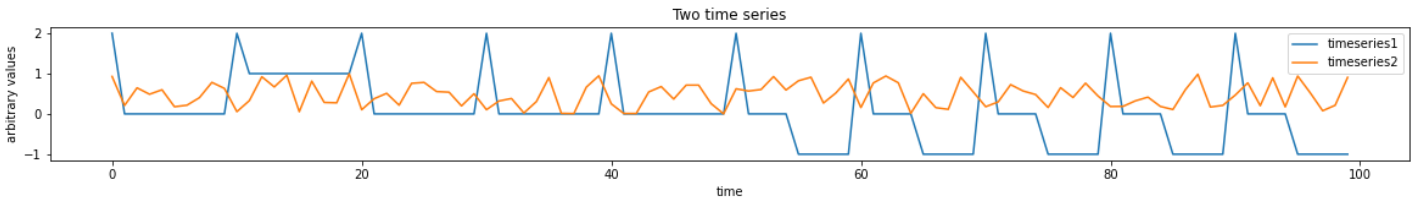
In [12]:
```
## This is for after the homework has been graded.
# ok.grade('q3_1c_full')
```

(d) [1pt] Now let's add a second time series to obtain a collection of "voxel" time series, which we can then visualize together with the first one. Create a random array using `np.random.rand` of the same size as `timeseries1`. Call it `timeseries2`. Now use `np.stack` to stack `timeseries1` and `timeseries2` on as columns and call the results `both_timeseries`. Plot them together using `plt.plot` into a figure named `fig_both_ts`.

Add an `xlabel` 'time', a `ylabel` 'arbitrary values', a title `'Two time series'` and a legend specifying which is `timeseries1` and which is `timeseries2`.

In [13]:
```
## (d)

timeseries2 = np.random.rand(len(timeseries1))

both_timeseries = np.stack((timeseries1, timeseries2), axis=1)

fig_both_ts = plt.figure(figsize=(20, 2))
plt.plot(both_timeseries)

plt.xlabel('time')
plt.ylabel('arbitrary values')
plt.title('Two time series')

plt.legend(('timeseries1', 'timeseries2'))
```

Out[13]:
```
<matplotlib.legend.Legend at 0x7f65005c4dc0>
```



In [14]:
```
# This check is to ensure that you have correctly filled the variables required for autogr
ok.grade('q3_1d')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    Passed: 3
    Failed: 0
[ooooooooook] 100.0% passed
```

Out[14]:
```
{'passed': 3, 'failed': 0, 'locked': 0}
```

In [15]:
```
## This is for after the homework has been graded.
# ok.grade('q3_1d_full')
```

(e) [0.5pts] Use `np.mean` with the correct `axis` to obtain the mean of both time series contained in

`both_timeseries`, and store it in `ts_means`. Print `ts_means`.

In [16]:
```
## (e)

ts_means = np.mean(both_timeseries, axis=0)

print(ts_means)
```

```
[0.04       0.46986706]
```

In [17]:
```
# This check is to ensure that you have correctly filled the variables required for autogr
ok.grade('q3_1e')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    Passed: 1
    Failed: 0
[oooooooooook] 100.0% passed
```

Out[17]:
```
{'passed': 1, 'failed': 0, 'locked': 0}
```

In [18]:
```
## This is for after the homework has been graded.
# ok.grade('q3_1e_full')
```

2. **Creating an RGB image.**

Color images are 3D arrays with two spatial axes and one categorical axis encoding three distinct channels for red, green and blue values.

In this exercise you will create an RGB image by filling its color channels one by one, with the goal of gaining an intuition of how they work. This is also a preview of next week's assignment: We are going to ask you to extend your understanding of `np.stack` from stacking 1-D arrays to create a 2-D array, and stack 2-D arrays to create a 3-D array.

For `plt.imshow` to display RGB images properly, they either need to be made of byte-sized integers (values ranging from 0 to 255) or floating point values between 0 and 1. We choose the first option in order to learn a new `dtype`.

(a) [0.5pts] Create three arrays full of zeros, of shape `(480, 640)` and of `dtype uint8`. Name them `R`, `G`, and `B`. We will call each of these a *color channel*.

In [19]:
```
## (a)

R, G, B = np.zeros((3, 640, 640), dtype='uint8')

# or

R = np.zeros((640, 640), dtype='uint8')
G = np.zeros((640, 640), dtype='uint8')
B = np.zeros((640, 640), dtype='uint8')
```

In [20]:
```
# This check is to ensure that you have correctly filled the variables required for autogr
ok.grade('q3_2a')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    Passed: 3
    Failed: 0
[oooooooooook] 100.0% passed
```

Out[20]:
```
{'passed': 3, 'failed': 0, 'locked': 0}
```

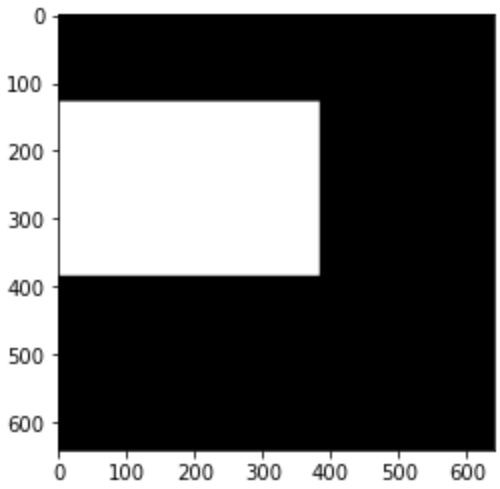
In [21]:
```
## This is for after the homework has been graded.
# ok.grade('q3_2a_full')
```

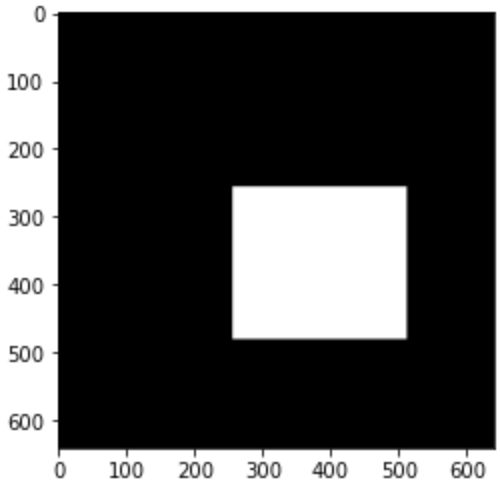
(b) [1pts] We will now draw some rectangles into these color channels: In the `R` channel, set the slice `128:384, 0:384` to the maximum color value, 255. This will result in a red rectangle later. In the `G` channel, set the slice `256:480, 256:512` to 255, resulting in a green rectangle later.

Use `plt.imshow` to visualize `R` and `G` separately, using the colormap `gray`. Create a figure for each and name them `fig_R` and `fig_G`. These are going to be black and white images showing where each of the channels have more or less high values (here they are either 255, white, or 0, black).

In [22]:
```
## (b)

R[128:384, 0:384] = 255
G[256:480, 256:512] = 255

fig_R = plt.figure()
plt.imshow(R, cmap='gray')

fig_G = plt.figure()
plt.imshow(G, cmap='gray')
```

Out[22]:
```
<matplotlib.image.AxesImage at 0x7f6500531c70>
```





In [23]:
```
# This check is to ensure that you have correctly filled the variables required for autogr
ok.grade('q3_2b')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    Passed: 4
    Failed: 0
[oooooooooook] 100.0% passed
```

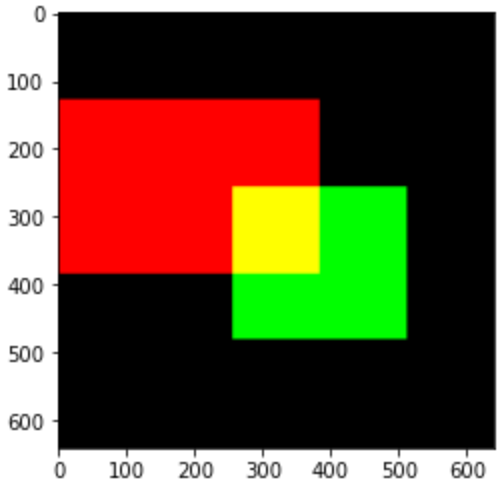
Out[23]:
```
{'passed': 4, 'failed': 0, 'locked': 0}
```

In [24]:
```
## This is for after the homework has been graded.
# ok.grade('q3_2b_full')
```

(c) [1pts] Use `np.stack` to stack `R`, `G`, and `B` together on `axis` 2 and call the result `RGB`. Then display `RGB` using `plt.imshow` in a figure named `fig_RGB`.

In [25]:
```
## (c)

RGB = np.stack((R, G, B), axis=2)

fig_RGB = plt.figure()
plt.imshow(RGB)
```

Out[25]:
```
<matplotlib.image.AxesImage at 0x7f65004469d0>
```



In [26]:
```
# This check is to ensure that you have correctly filled the variables required for autogr
ok.grade('q3_2c')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    Passed: 2
    Failed: 0
[oooooooooook] 100.0% passed
```

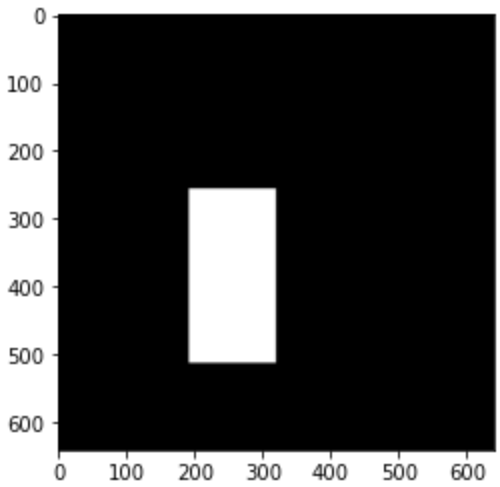
Out[26]:
```
{'passed': 2, 'failed': 0, 'locked': 0}
```

In [27]:
```
## This is for after the homework has been graded.
# ok.grade('q3_2c_full')
```

(d) [1pt] Now modify the blue channel `B` in whichever way you wish (except for leaving it at exactly 0!). Feel free to modify the `R` and `G` channels if you like. Display your modified `B` in gray scale in a figure named `fig_B`.

In [28]:
```
## (d)

B[256:512, 192:320] = 255

fig_B = plt.figure()
plt.imshow(B, cmap='gray')
```

Out[28]:
```
<matplotlib.image.AxesImage at 0x7f65004327f0>
```



In [29]:
```
# This check is to ensure that you have correctly filled the variables required for autogr
ok.grade('q3_2d')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    Passed: 2
    Failed: 0
[oooooooooook] 100.0% passed
```

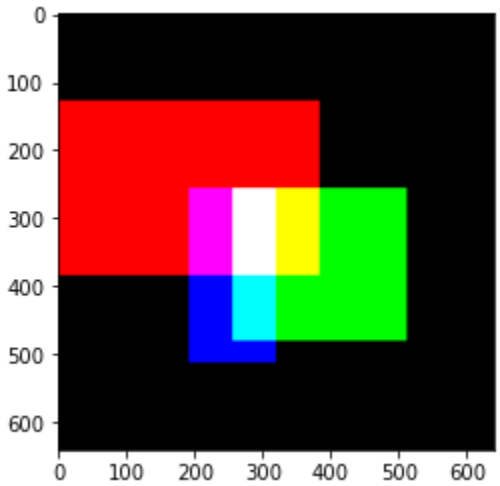
Out[29]:
```
{'passed': 2, 'failed': 0, 'locked': 0}
```

In [30]:
```
## This is for after the homework has been graded.
# ok.grade('q3_2d_full')
```

(e) [0.5pts] Now that you have modified at least the `B` channel, it is time to create a new RGB image and take a look at the effects. Create a new image `RGB2` by stacking `R, G, B` again, and display it in a figure `fig_RGB2`.

In [31]:
```
## (e)

RGB2 = np.stack((R, G, B), axis=2)
fig_RGB2 = plt.figure()
plt.imshow(RGB2)
```

Out[31]:
```
<matplotlib.image.AxesImage at 0x7f650039d6d0>
```



In [32]:
```
# This check is to ensure that you have correctly filled the variables required for autogr
ok.grade('q3_2e')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    Passed: 2
    Failed: 0
[ooooooooook] 100.0% passed
```

Out[32]:
```
{'passed': 2, 'failed': 0, 'locked': 0}
```

In [33]:
```
## This is for after the homework has been graded.
# ok.grade('q3_2e_full')
```

3. [2pts] **Visualizing 2-D neuroimaging data.**

In this exercise you will use what you have learned in the last two lectures visualize neuroimaging data. You will be presented with four 2-D arrays of brain data of unknown meta-types. By looking at their shapes and visualizing them, you need to figure out what meta-type of 2-D array you are looking at.

For each of the following arrays `ni_array_1`, `ni_array_2`, `ni_array_3`, `ni_array_4`, perform the following steps:

- using the visualization functions `plt.plot` and `plt.imshow`, and by inspecting the shape, figure out what meta-type of 2D array you are dealing with (one of `'coronal slice'`, `'axial slice'`, `'sagittal slice'`, `'time series'`, `'unidentified'`). If the array appears to be none of the four mentioned types, then declare them as `'unidentified'`. For each of the four arrays, store your answer in a name called `ni_array_?_type` (where `?` is to be replaced by one of `1, 2, 3, 4`).
  Note: Put the `plt.plot`s in figures that you name `fig_plot_ni_array_?` and the `plt.imshow`s in figures named `fig_imshow_ni_array_?`.

In [34]:
```
# 3 -

ni_array_1_type = 'time series'

ni_array_2_type = 'sagittal slice'

ni_array_3_type = 'coronal slice'

ni_array_4_type = 'axial slice'
```

In [ ]:
```
fig_plot_ni_array_1 = plt.figure()
plt.plot(ni_array_1)
fig_imshow_ni_array_1 = plt.figure()
plt.imshow(ni_array_1)
```

In [ ]:
```
fig_plot_ni_array_2 = plt.figure()
plt.plot(ni_array_2)
fig_imshow_ni_array_2 = plt.figure()
plt.imshow(ni_array_2)
```

In [ ]:
```
fig_plot_ni_array_3 = plt.figure()
plt.plot(ni_array_3)
fig_imshow_ni_array_3 = plt.figure()
plt.imshow(ni_array_3)
```

In [39]:
```
# This check is to ensure that you have correctly filled the variables required for autogr
ok.grade('q3_3')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
3 > Suite 1 > Case 10

>>> # It seems like you may have not yet created a figure named 'fig_imshow_ni_array_1'
>>> # Please create a figure named 'fig_imshow_ni_array_1' for this cell
>>> 'fig_imshow_ni_array_1' in vars()
False

# Error: expected
#     True
# but got
#     False

Run only this test case with "python3 ok -q q3_3 --suite 1 --case 10"
---------------------------------------------------------------------
Test summary
    Passed: 9
    Failed: 1
```

```
[ooooooooook.] 90.0% passed
```

Out[39]:
```
{'passed': 9, 'failed': 1, 'locked': 0}
```