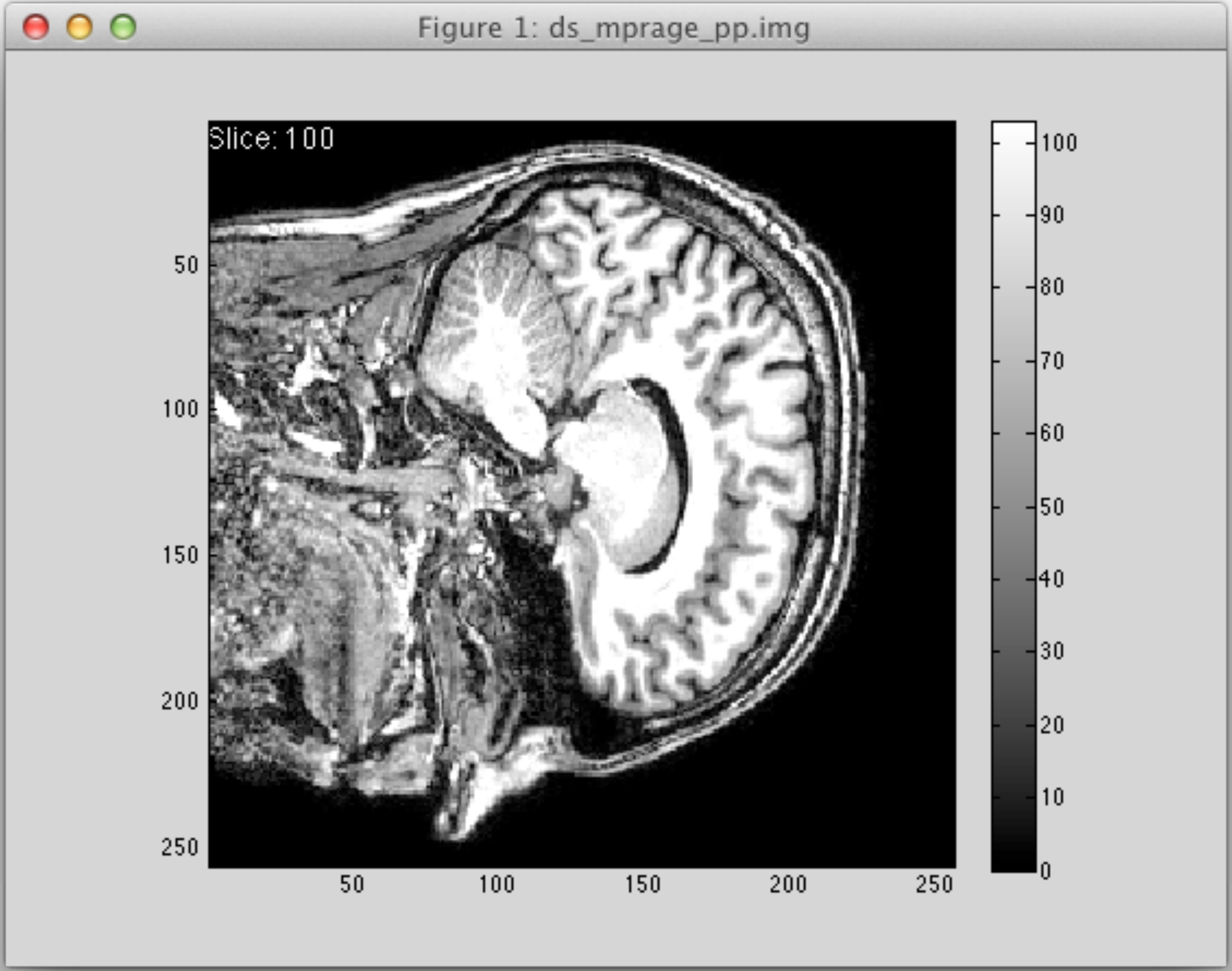
sliceview()

Denis Schluppeck
UoN, Psychology

1 – DIY image viewer



1 – specification

a main **function** sliceview() in sliceview.m

- **load**s anatomy.mat from current folder
- displays data in a figure window using **imagesc**
- displays a **colorbar** and **text** showing current slice
- allows you to scroll through slices (up, down)
- change “orientation” (o)
- ninja skills: mouse-click, scroll wheel, ...

returnSlice( )

- `s =` **`returnSlice(`**`array, sliceNum, orientation`**`)`**

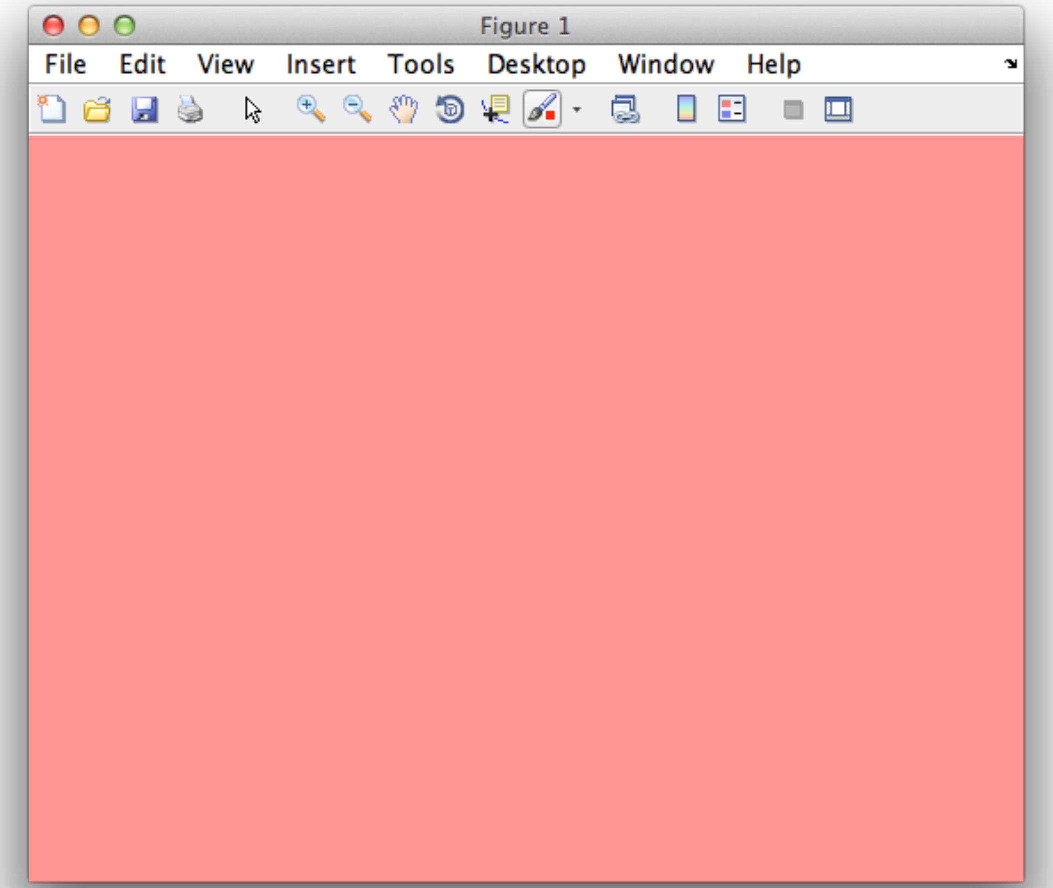
- **`s`** `should be a 2d array (a slice)`
- **`sliceNum`** `is the slice we want to get out in`
- **`orientation`** `(1, 2, or 3 for now)`

- `(( things to worry about / check ? ))`

interact.m

- to make things **interactive** -> callBack functions

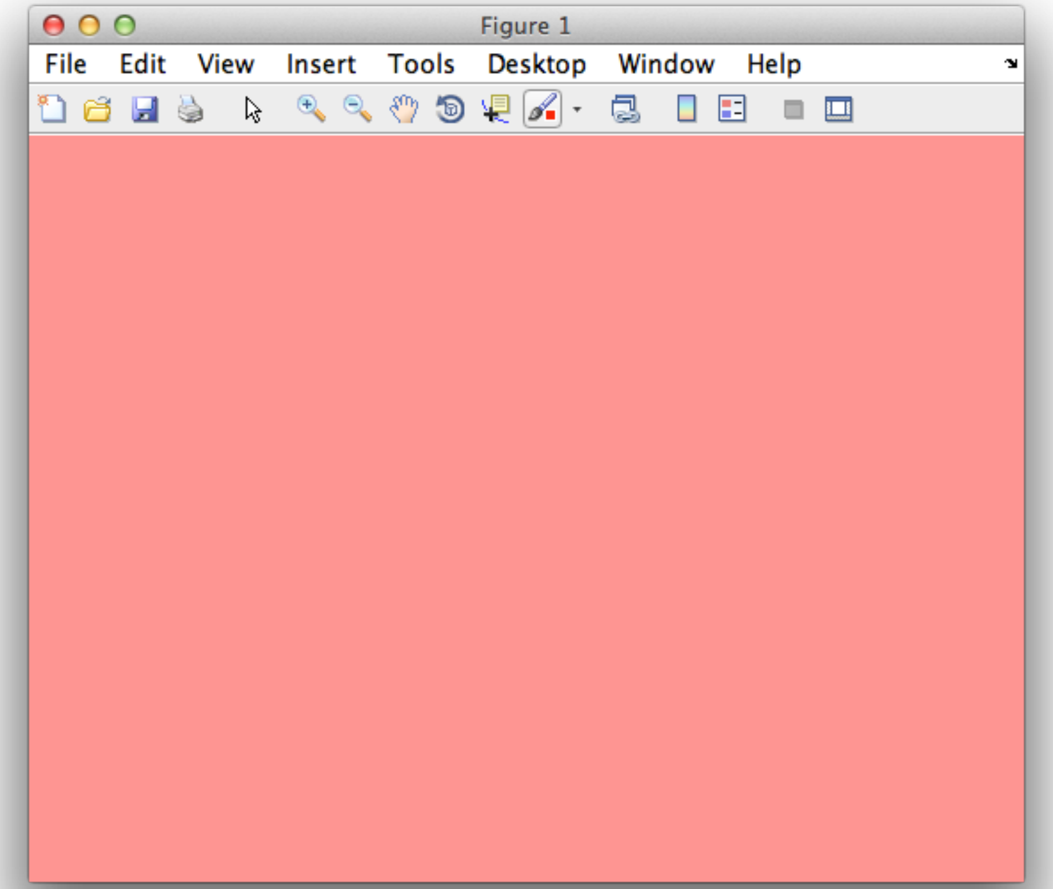
- run **interact()** from the command line or editor
- inspect the code and comments



interact.m

- to make things **interactive** -> callBack functions

- run **interact()** from the command line or editor
- inspect the code and comments



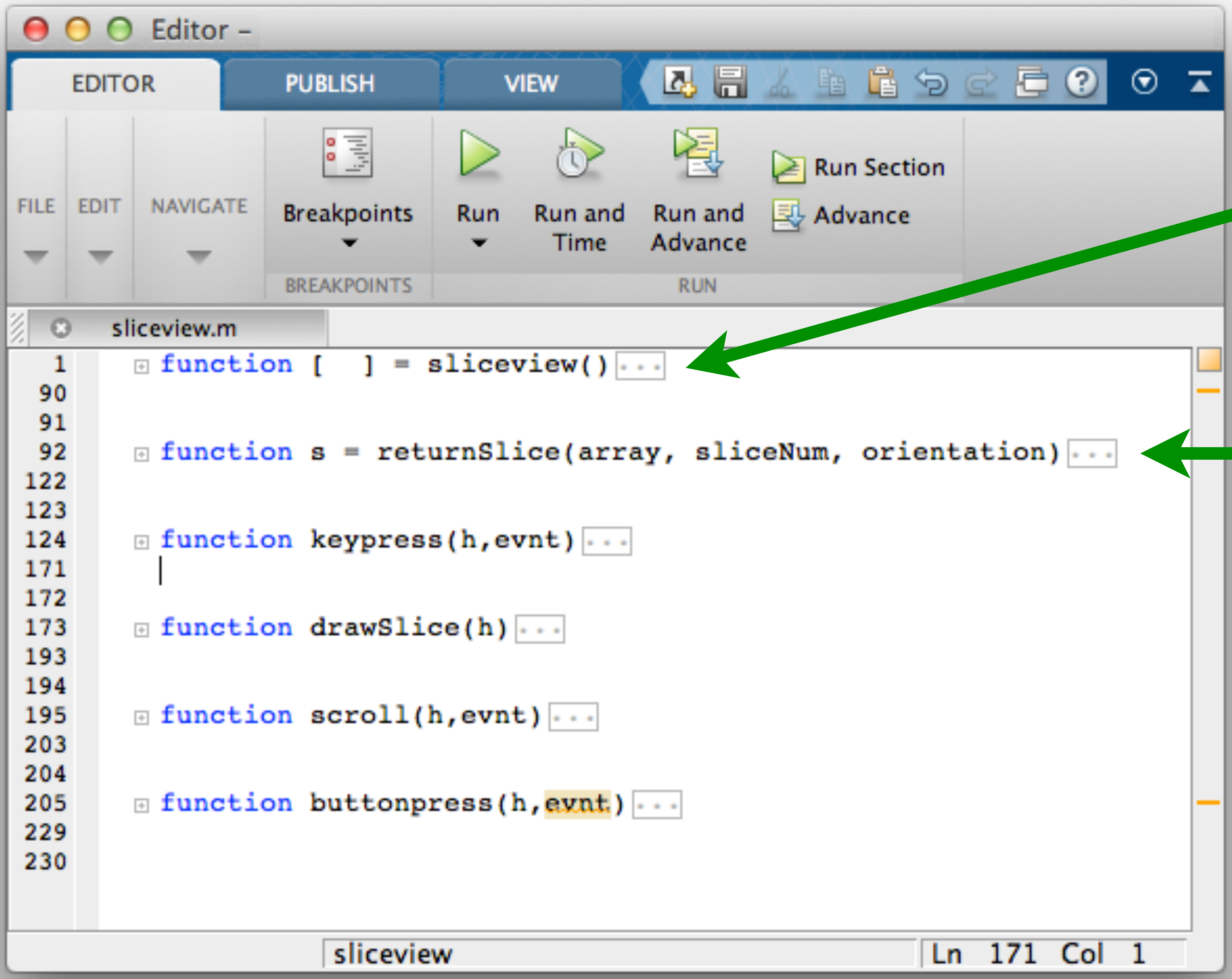
get / set

set(h, ‘WindowKeyPressFcn’, ...)

@keypress

get/set(h, ‘UserData’, ...)

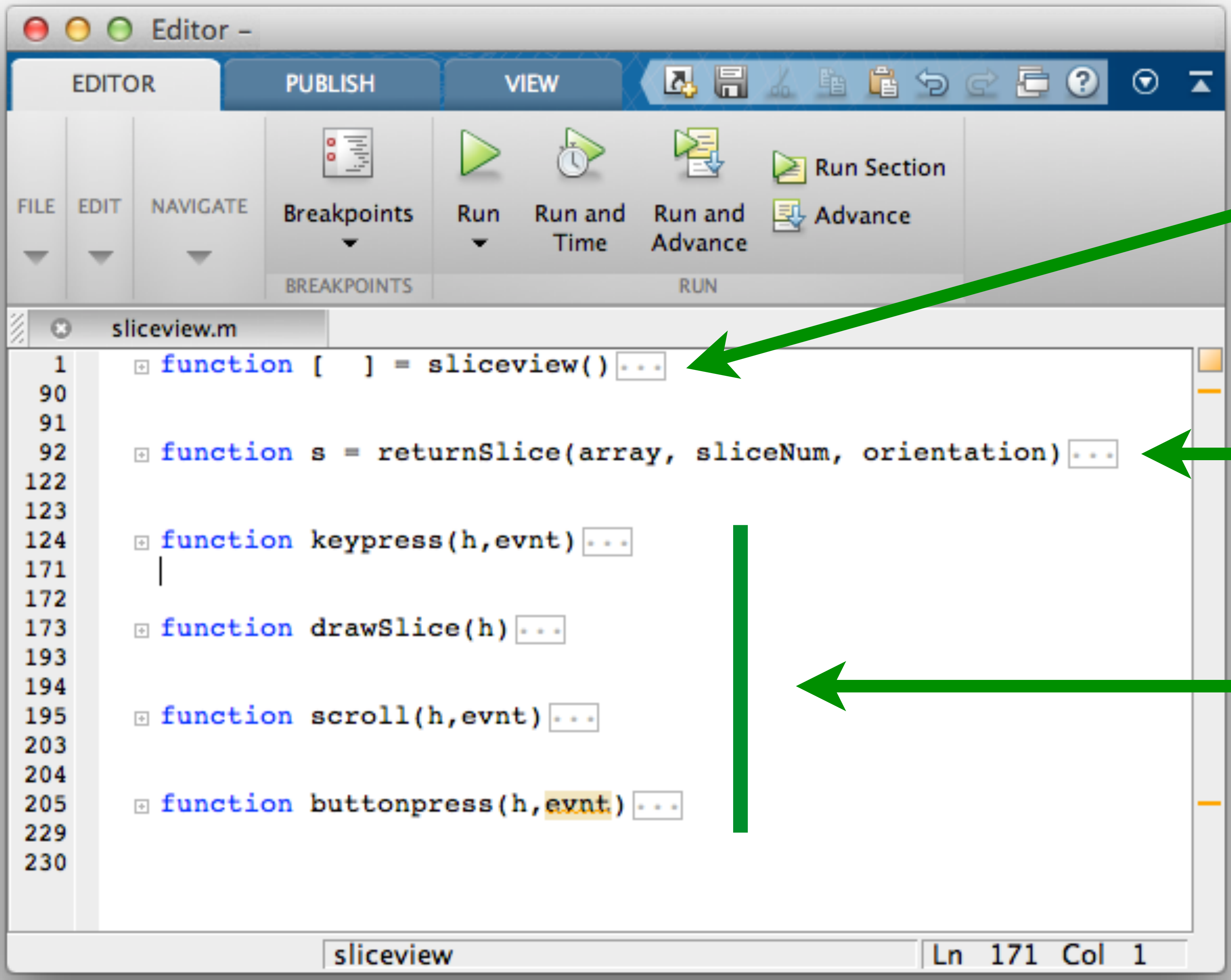
*use a **struct** to pass data around*



```
function [   ] = sliceview()...

function s = returnSlice(array, sliceNum, orientation)...

function keypress(h,evnt)...


function drawSlice(h)...

function scroll(h,evnt)...

function buttonpress(h,evnt)...
```

the ***main function***

helper function
takes 3d data and returns a slice in a particular orientation



```
function [   ] = sliceview()...

function s = returnSlice(array, sliceNum, orientation)...

function keypress(h,evnt)...


function drawSlice(h)...

function scroll(h,evnt)...

function buttonpress(h,evnt)...
```

the ***main function***

helper function
takes 3d data and returns a slice in a particular orientation

called every time a key is pressed

... every time we want to refresh the image

... every time the mouse wheel moves

... every time a mouse button is clicked

function refresher

`functions` allow you to reuse solutions to smaller sub-problems. make your code easier to follow.

```
function [ out ] = nameOfYourFunction(in1, in2)
[illegible]







end
```

function refresher

functions allow you to reuse solutions to smaller sub-problems. make your code easier to follow.

```
function [ out ] = nameOfYourFunction(in1, in2)








end
```

function refresher

functions allow you to reuse solutions to smaller sub-problems. make your code easier to follow.

```
function [ out ] = nameOfYourFunction(in1, in2)
% one line description
%
%   help text
%

- your code, using inputs in1 and in2 -
- packaging up everything into an output argument out -

end
```

function refresher

functions allow you to reuse solutions to smaller sub-problems. make your code easier to follow.

```
function [ d ] = distanceBetweenPoints(p, q)
%distanceBetweenPoints - calculates euclidian distance between two points
%
%    purpose: function calculates distance between two points, which can be
%    defined in n-dimensions. E.g. on a number line, or in 2d (on a plane),
%    or 3d, ...
%
%        e.g:  d = distanceBetweenPoints([0 0], [1 1])
```

function refresher

functions allow you to reuse solutions to smaller sub-problems. make your code easier to follow.

```matlab
function [ d ] = distanceBetweenPoints(p, q)
%distanceBetweenPoints - calculates euclidian distance between two points
%
%    purpose: function calculates distance between two points, which can be
%    defined in n-dimensions. E.g. on a number line, or in 2d (on a plane),
%    or 3d, ...
%
%        e.g:  d = distanceBetweenPoints([0 0], [1 1])

% check that inputs are the same size
if any( size(p) ~= size(q) )
    error('p and q need to have same number of coordinates')
end

% using pythagoras
% http://en.wikipedia.org/wiki/Euclidian_distance
% equation 1 will work for any number of dimensions ;]
d = sqrt( sum( (p - q).^2 ) );

end
```

struct refresher

keep disparate information organized in one convenient variable, e.g. **instead of having the clutter of different 5 variables**

struct refresher

keep disparate information organized in one convenient variable, e.g. **instead of having the clutter of different 5 variables**

```
currentPoint = [24, 10, 3];
currentOrientation = 1;
cmap = gray(256);
dataLimits = [0 256];
filename = ‘ds_mprage_pp.img’;
```

put them into a single “container”

struct refresher

keep disparate information organized in one convenient variable, e.g. **instead of having the clutter of different 5 variables**

```
data.currentPoint = [24, 10, 3];
data.currentOrientation = 1;
data.cmap = gray(256);
data.dataLimits = [0 256];
data.filename = ‘ds_mprage_pp.img’;
```

… so you can pass around **data** and be done

struct refresher

```
data.currentPoint = [24, 10, 3];
data.currentOrientation = 1;
data.cmap = gray(256);
data.dataLimits = [0 256];
data.filename = ‘ds_mprage_pp.img’;
```

struct refresher

```
data.currentPoint = [24, 10, 3];
data.currentOrientation = 1;
data.cmap = gray(256);
data.dataLimits = [0 256];
data.filename = ‘ds_mprage_pp.img’;
```

or

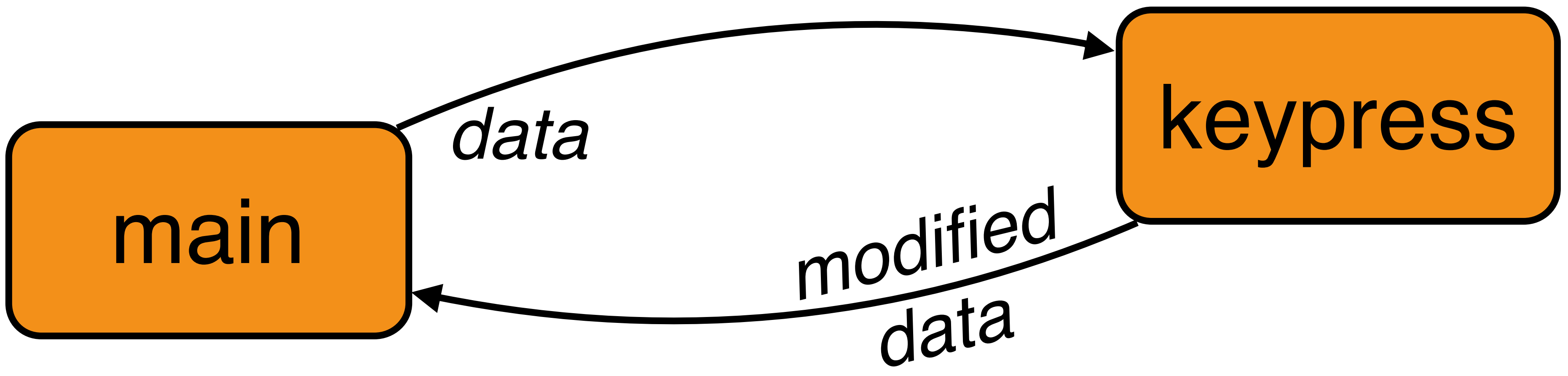
```
data = struct(‘currentPoint’, [24, 10, 3], …
              ‘currentOrientation’, 1, …
              ‘cmap’, gray(256), …
              ‘dataLimits’, [0 256], …
              ‘filename’, ‘ds_mprage_pp.img’);
```

passing data around

```
% keep everything that we want to pass round
data = struct(   'array', array, …
                 ’hdr’, hdr, …
                 ’currentSliceNum’, sliceNum, …
                 'currentOrientation', orientation, …
                 ’currentSlice’, s);

data.cmap =         ? ;  % fix the colormap and the range of values
data.dataLimits =   ? ;

etc… ?
```

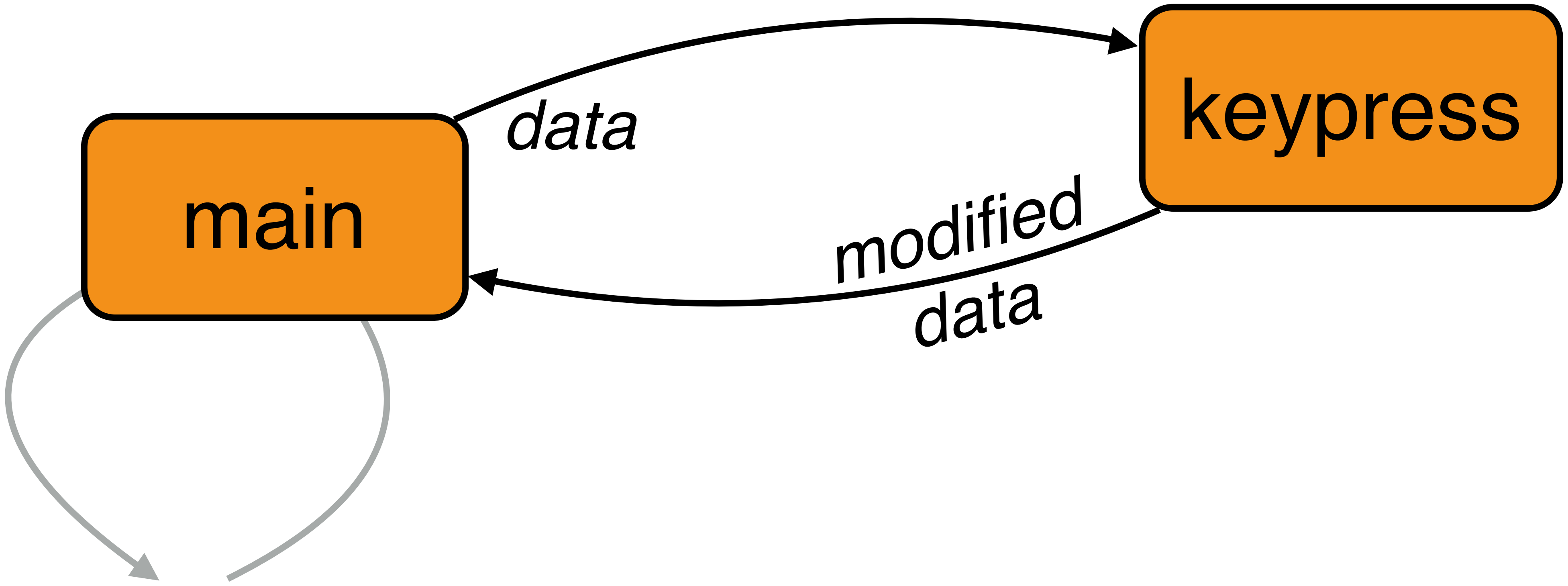
passing data around

```
% keep everything that we want to pass round
data = struct(   'array', array, …
                 'hdr', hdr, …
                 'currentSliceNum', sliceNum, …
                 'currentOrientation', orientation, …
                 'currentSlice', s);

data.cmap =         ? ;  % fix the colormap and the range of values
data.dataLimits =   ? ;

etc… ?
```



passing data around

```
% keep everything that we want to pass round
data = struct(   'array', array, …
                 'hdr', hdr, …
                 'currentSliceNum', sliceNum, …
                 'currentOrientation', orientation, …
                 'currentSlice', s);

data.cmap =         ? ;  % fix the colormap and the range of values
data.dataLimits =   ? ;

etc… ?
```



example solution

returnSlice()

```
function s = returnSlice(array, sliceNum, orientation)
% returnSlice - return a single slice from a 3d image

% if orientation is not given, keep the last (3rd?) index
fixed
if nargin < 3, orientation = 3; end

% pick data, keeping dimension=“orientation” fixed
switch orientation
    case 1
        s = array(sliceNum,:,:);
    case 2
        s = array(:,sliceNum,:);
    case 3
        s = array(:,:,sliceNum);
end

% now also make sure that s doesn't have
% some weird extraneous dimensions - GOTCHA
s = squeeze(s);

end
```

sliceview()

```
function [  ] = sliceview()
%sliceview - simple slice viewer for 3d data

% load a data file, 'array' and 'hdr'
load('anatomy.mat')

% make a figure
h = figure(); % h is the figure handle

% change the name of the figure
set(h,'Name', hdr.img_name);
```

sliceview()

```
function [  ] = sliceview()
%sliceview - simple slice viewer for 3d data

% load a data file, 'array' and 'hdr'
load('anatomy.mat')

% make a figure
h = figure(); % h is the figure handle

% change the name of the figure
set(h,'Name', hdr.img_name);

% hooks up “callback” function
set(h,'KeyPressFcn',@keypress);

% "orientation" of the image.
orientation = 1; % could be 1, 2, or 3
sliceNum = round(size(array, orientation)./2); % half way through
s = returnSlice(array, sliceNum, orientation); % now grab a slice
```

sliceview()

```
function [  ] = sliceview()
%sliceview - simple slice viewer for 3d data

% load a data file, 'array' and 'hdr'
load('anatomy.mat')

% make a figure
h = figure(); % h is the figure handle

% change the name of the figure
set(h,'Name', hdr.img_name);

% hooks up “callback” function
set(h,'KeyPressFcn',@keypress);

% "orientation" of the image.
orientation = 1; % could be 1, 2, or 3
sliceNum = round(size(array, orientation)./2); % half way through
s = returnSlice(array, sliceNum, orientation); % now grab a slice

% keep everything that we want to pass round
data = struct('array', array, 'hdr', hdr, 'currentSliceNum', sliceNum, ...
    'currentOrientation', orientation, 'currentSlice', s);

data.cmap = gray(256);  % fix the colormap and the range of values
data.dataLimits = prctile(array(:),[5 95]);

% attach the wrapped up "data" to the window (handle)
set(h,'UserData',data);
```

sliceview()

```matlab
function [  ] = sliceview()
%sliceview - simple slice viewer for 3d data

% load a data file, 'array' and 'hdr'
load('anatomy.mat')

% make a figure
h = figure(); % h is the figure handle

% change the name of the figure
set(h,'Name', hdr.img_name);

% hooks up “callback” function
set(h,'KeyPressFcn',@keypress);

% "orientation" of the image.
orientation = 1; % could be 1, 2, or 3
sliceNum = round(size(array, orientation)./2); % half way through
s = returnSlice(array, sliceNum, orientation); % now grab a slice

% keep everything that we want to pass round
data = struct('array', array, 'hdr', hdr, 'currentSliceNum', sliceNum, ...
    'currentOrientation', orientation, 'currentSlice', s);

data.cmap = gray(256);  % fix the colormap and the range of values
data.dataLimits = prctile(array(:),[5 95]);

% attach the wrapped up "data" to the window (handle)
set(h,'UserData',data);

% now for the first time, draw the slice now:
drawSlice(h);

end
```

sliceview()

```
function [  ] = sliceview()
%sliceview - simple slice viewer for 3d data

% load a data file, 'array' and 'hdr'
load('anatomy.mat')

% make a figure
h = figure(); % h is the figure handle

% change the name of the figure
set(h,'Name', hdr.img_name);

% hooks up “callback” function
set(h,'KeyPressFcn',@keypress);

% "orientation" of the image.
orientation = 1; % could be 1, 2, or 3
sliceNum = round(size(array, orientation)./2); % half way through
s = returnSlice(array, sliceNum, orientation); % now grab a slice

% keep everything that we want to pass round
data = struct('array', array, 'hdr', hdr, 'currentSliceNum', sliceNum, ...
    'currentOrientation', orientation, 'currentSlice', s);

data.cmap = gray(256);  % fix the colormap and the range of values
data.dataLimits = prctile(array(:),[5 95]);

% attach the wrapped up "data" to the window (handle)
set(h,'UserData',data);

% now for the first time, draw the slice now:
drawSlice(h);

end
```

drawslice()

```
function drawSlice(h)
% drawSlice - draws the current slice in the window

figure(h) % make sure we draw into the right figure
data = get(h,'UserData'); % get a local copy of the data

img = data.currentSlice;

% draw image, keeping colormap fixed, add colorbar
imagesc(img, data.dataLimits);
colormap(data.cmap)
colorbar
axis image

% bonus - add a text label:
t_ = text(0,0,['Slice: ' num2str(data.currentSliceNum, '%d') ] );

% and change default color, size, ...:
set(t_, 'color','w','fontsize',14, 'verticalalignment','top');

end
```

keypress()

```matlab
function keypress(h,evnt)
% keypress - called every time a key is pressed

% get hold of the data for use in this function...
data=get(h,'UserData');

switch evnt.Key
    case 'uparrow'
        data.currentSliceNum = data.currentSliceNum + 1;
    case 'downarrow'
        data.currentSliceNum = data.currentSliceNum - 1;
    case {'o','O'}
        % change orientation
        data.currentOrientation = mod(data.currentOrientation + 1,3) + 1;
    case {'Q','q','Escape'}
        disp('Byebye!'), close(h); return;
end

% check that we don't go under 0 or over the max
if data.currentSliceNum < 1
    disp('(keypress) UHOH! trying to go below 0!')
    data.currentSliceNum = 1;
end

% now also need to put the new slice image into its place
data.currentSlice = returnSlice(data.array, ...
    data.currentSliceNum, ...
    data.currentOrientation);

set(h,'UserData',data); % stuff changed data back
drawSlice(h); % things just changed! REDRAW...

end
```