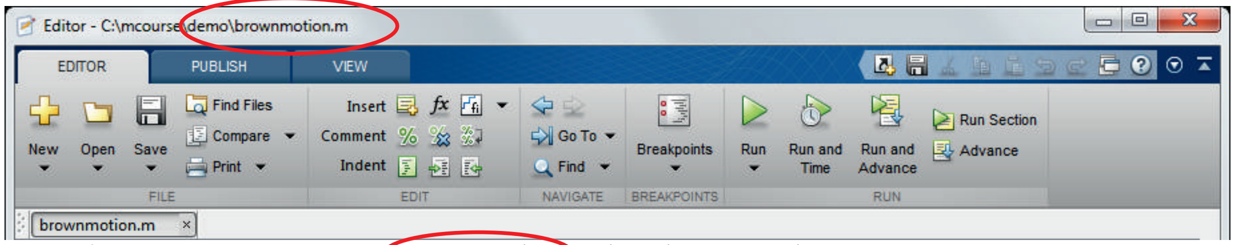
Example of well-structured, readable code

Function name matches file name, is meaningful and not identical to any of Matlab's functions

First line of help text ('H1 linie') repeats function definition, followed by a brief explanation of the function's job

Variable number of input and output arguments lends flexibility to the code



```
function [nJump,varargout]=brownmotion(nDim,lim,varargin)
% ** function [nJump,varargout]=brownmotion(nDim,lim,varargin)
% simulates Brownian motion of a particle in 1, 2 or 3 dimensions.
%
% >>> INPUT VARIABLES >>>
% NAME           TYPE          DESCRIPTION
% nDim           scalar        number of dimensions
% lim            scalar        limit of summed path of particle beyond
%                              which simulation will terminate
% varargin{1}   row array      start position of particle (pay attention to
%                              dimensionality!)
%
% <<< OUTPUT VARIABLES <<<
% NAME           TYPE          DESCRIPTION
% nJump          scalar        the number of jumps the particle made to
%                              fulfill the termination criterion
% varargout{1}  2d-array       the array holding the trajectory of the
%                              particle (nDim columns, time runs down the
%                              columns)

% ------- checks of input arguments ---------------------------------
if nDim<1 || nDim>3
  error('nDim must be between 1 and 3');
end
if nargin>2
  % the particle's position at time 0
  pos=varargin{1};
  if numel(pos)~=nDim
    error('dimension mismatch between starting position and input var ''nDim''');
  else
    % make sure pos is a row array (without actually checking this)
    pos=pos(:)';
  end
else
  % the particle's position at time 0 is by default the origin
  pos=zeros(1,nDim);
end

% ------- deal with output arguments ---------------------------------
% check how many additional output arguments were requested (up to 2)
% doKeepTraj is a variable determining whether trajectory shall be put out
doKeepTraj=false;
switch nargout
case {0,1}
```

Clear definition of input and output variables

Relatively thorough (though by no means exhaustive) check of input arguments

```
  % do nothing: either no output or just 'nJump' requested
case 2
  doKeepTraj=true;
otherwise
  error('only 1 additional output argument (trajectory) is legal')
end
```

Variable names are telling, at least to some degree

Ample comments throughout the file with proper line breaks and consistently placed above the line(s) they inform us about

```
% ------- initialzation of variables ---------------------------------
% the particle's trajectory
traj=pos;
% the particle's summed path at the beginning
sumPath=0;
% number of jumps the particle shall make en block (to speed up code)
nBlockJumps=1000;
% the (running) number of loop executions performed
nLoopRun=0;
% logical variable indicating whether the particle shall move or not
doMove=true;
```

Comments also used for visual segmentation of the code into functionally different parts

Proper indentation of code

```
% ------- move! -----------------------------------------------
while doMove
  nLoopRun=nLoopRun+1;
  % variables jump, pos and traj are column arrays with as many
  % columns as dimensions; time goes down the columns
  jump=randn(nBlockJumps,nDim);
  pos=cumsum([pos(end,:); jump]);
  % the Euclidian distance covered in each jump
  jumpDist=sqrt(sum(jump.^2,2));
  % summed path (= total travelled distance)
  sumPath=cumsum([sumPath(end,:); jumpDist]);
  % see whether summed path is already above limit
  r=min(find(sumPath>=lim));
  doMove=isempty(r);
  if doKeepTraj
    % if trajectory is requested as output argument, concatenate all
    % positions into variable traj (note that this is memory-consuming and
    % slow because preallocation is not really possible here!)
    traj=[traj;pos];
  end
end
```

Code with line breaks where appropriate

```
% ------- aftermath -------------------------------------------
% the number of elementary particle jumps
nJump=(nLoopRun-1)*nBlockJumps+r;
% additional output arg, if requested
if doKeepTraj
  varargout{1}=traj(1:nJump,:);
end
% put out a little information
disp(['condition met after ' int2str(nJump) ' elementary particle jumps (=' ...
  int2str(nLoopRun) ' loop runs)']);
```

Example of error-prone, impenetrable code

Function name does NOT match file name and is not particularly specific. Moreover, the file name is identical to Matlab's move function

The H1 line is above the function name - it will be displayed, but as it is disconnected from the rest of the comments it supposedly belongs to those comments will not be displayed in the command window upon 'help move'

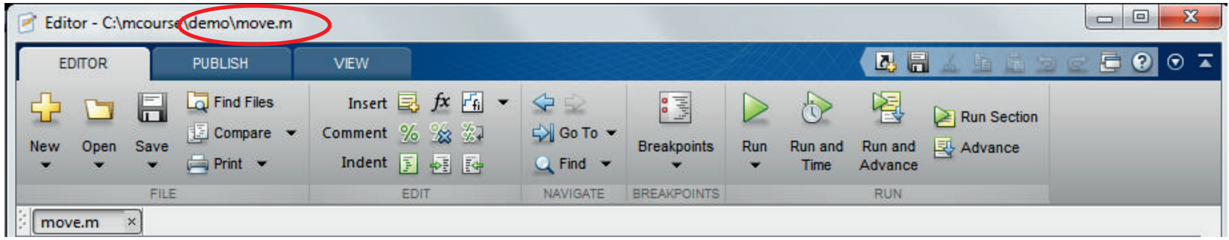
Variable names are ambiguous (loop), unrevealing (t) or outright dangerous (move, because that is also the file's name)

Input and output arguments are described in one paragraph without any visual distinction between them

Variable 'particlezeroposition' is
- too long of a variable name if there are no structuring elements in it (capital letters or underscores or so)
- ambiguous: is it the origin of the coordinate system or the particle's position at time zero?
- never used in the function (possibly it had been 'forgotten')

It can't get much worse than that:
- more than one command per line (one line seems to have suffered from an inadvertent deletion of a line break)
- the loop count variable has a potentially ambiguous name, and its value is incremented in the middle of the other code, as opposed to right at the beginning of the loop or at the end
- inconsistent or missing indentation
- empty lines impose visual structure on code that is not matched by its workings

varargout is used here but has not been declared in function definition



```
% simulates Brownian motion of a particle in 1, 2 or 3 dimensions.
function [nJump,t]=Motion(nDim,lim,pos)

% nDim is the number of dimensions
% startpos = start position of particle
% lim is the limit of summed path of particle beyond which simulation will
% terminate;  nJump = the number of jumps the particle made to fulfill the termination criterion, namely
% the summed path distance
% t is the array holding the trajectory of the particle

if nDim<1 || nDim>3
  error('nDim must be between 1 and 3');
end
particlezeroposition=[0]
t=pos;sumPath=0;nBlockJumps=1000;loop=0;

move=true;% to move or not to move, that is the question
doKeepTraj=1;
while move

  jump=randn(nBlockJumps,nDim);                              pos=cumsum([pos(end,:); jump]);
  % the Euclidian distance covered in each jump
  jumpDist=sqrt(sum(jump.^2,2));
  loop=loop+1;sumPath=cumsum([sumPath(end,:); jumpDist]);
  r=min(find(sumPath>=lim));
  % see whether summed path is already above limit

  move=isempty(r);
  if doKeepTraj
       t=[t;pos];
end
end


nJump=(loop-1)*nBlockJumps+r; % the number of elementary particle jumps
% additional output arg, if requested
if doKeepTraj
      varargout{1}=t(1:nJump,:);
end
% put out a little information
disp(['condition met after ' int2str(nJump) ' elementary particle jumps (=' int2str(loop) ' loop
```

Description of input arguments does not match actual input variable names (pos vs. startpos), is partly redundant and is not properly formatted (one line extends far beyond the rest of the comment and code)

Far too many commands in one line

There are not nearly enough comments to understand the code. The few comments that are there are inconsistently placed (mostly above code, sometimes to the right, sometimes below code they refer to)