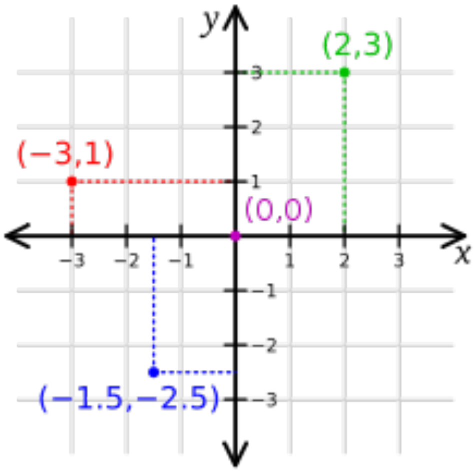
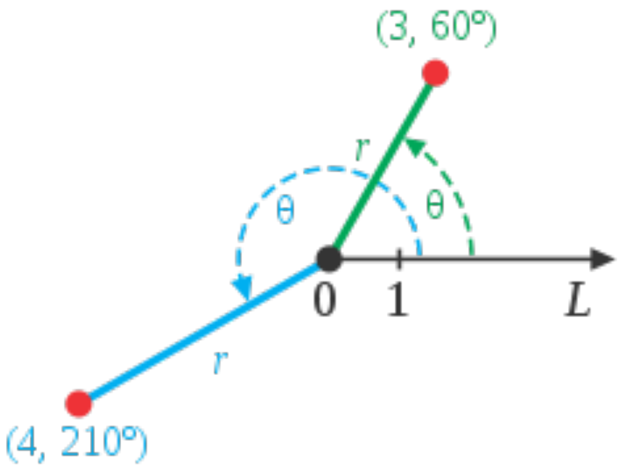
Programming and debugging

Class exercise

Write a function .m file that transfer the a point in the Cartesian coordinate system to the polar coordinate system.





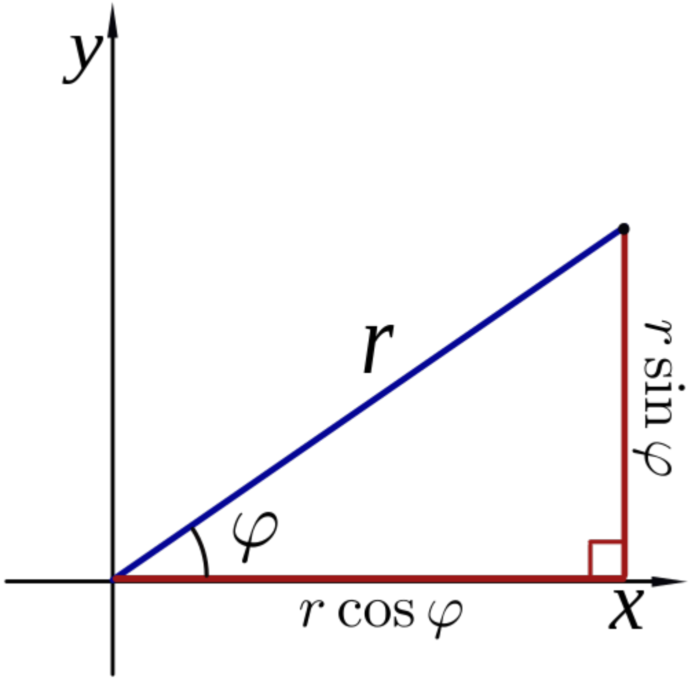
- Run the command “polarcoordinates(3,4)”.
- Run the command “theta=polarcoordinates(3,4)”.
- How to have the both outputs of r and theta?

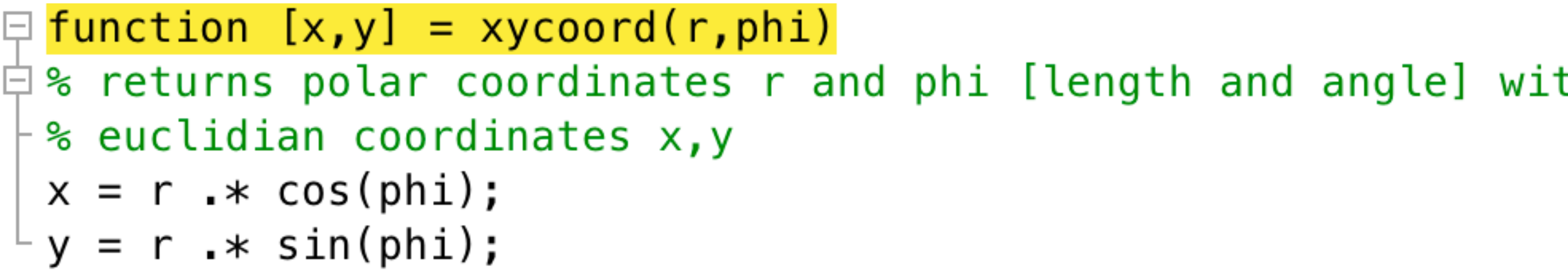
The polar coordinates *r* and *ϕ* can be converted to the Cartesian coordinates *x* and *y* by using the trigonometric functions sine and cosine:

$$
\begin{aligned}
x &= r\cos\varphi \\
y &= r\sin\varphi
\end{aligned}
$$

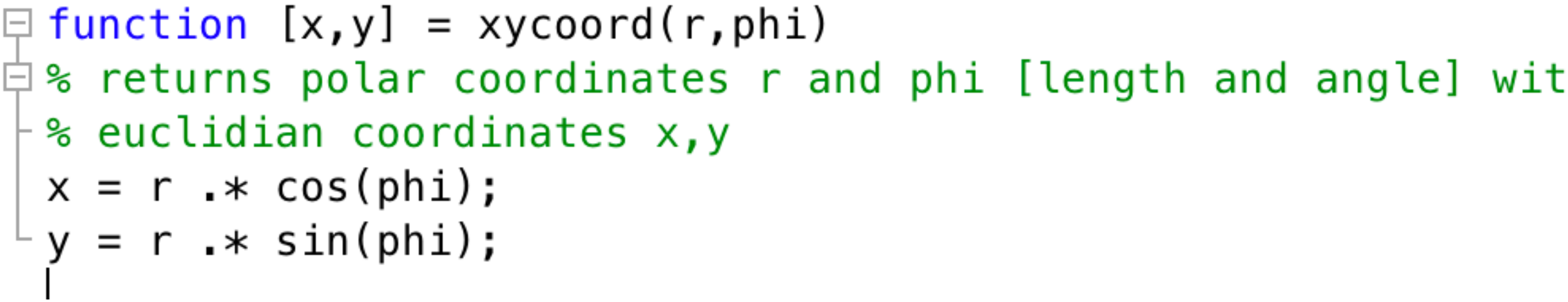
The Cartesian coordinates *x* and *y* can be converted to polar coordinates *r* and *ϕ* with *r* ≥ 0 and *ϕ* in the interval (−π, π] by:

$$
\begin{aligned}
r &= \sqrt{x^2 + y^2} \\
\varphi &= \operatorname{atan2}(y, x)
\end{aligned}
$$

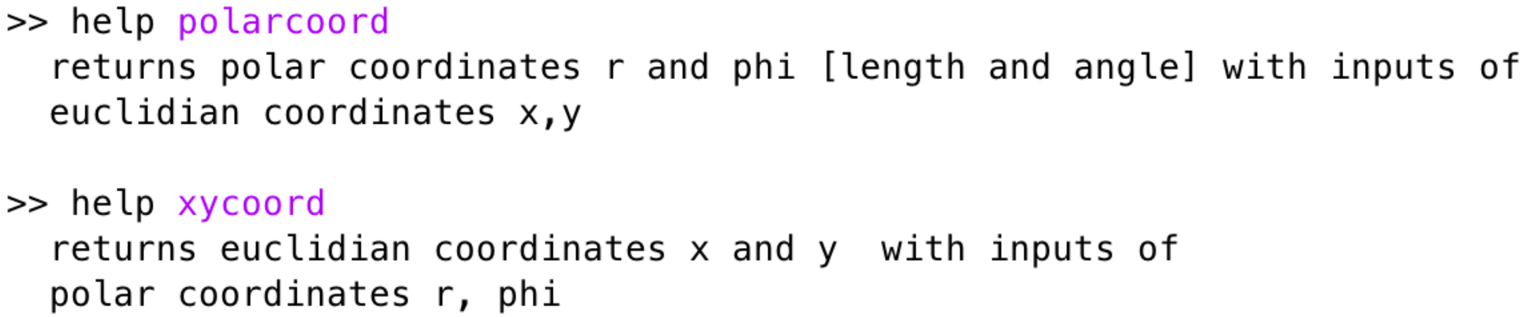




```
function [x,y] = xycoord(r,phi)
% returns polar coordinates r and phi [length and angle] wit
% euclidian coordinates x,y
x = r .* cos(phi);
y = r .* sin(phi);
```



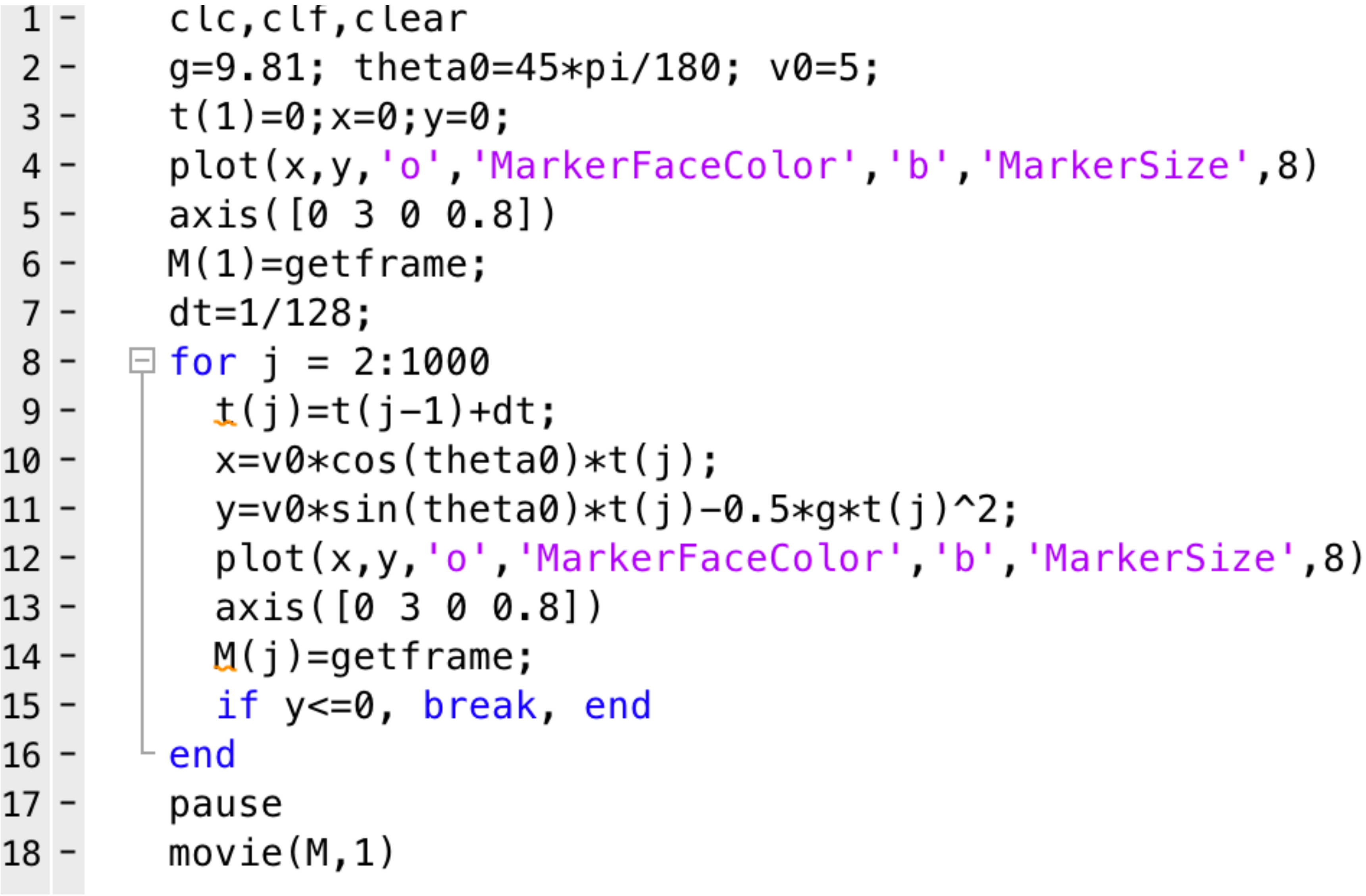
```
function [x,y] = xycoord(r,phi)
% returns polar coordinates r and phi [length and angle] wit
% euclidian coordinates x,y
x = r .* cos(phi);
y = r .* sin(phi);
```



```
>> help polarcoord
  returns polar coordinates r and phi [length and angle] with inputs of
  euclidian coordinates x,y

>> help xycoord
  returns euclidian coordinates x and y  with inputs of
  polar coordinates r, phi
```

https://en.wikipedia.org/wiki/Projectile_motion



```
clc,clf,clear
g=9.81; theta0=45*pi/180; v0=5;
t(1)=0;x=0;y=0;
plot(x,y,'o','MarkerFaceColor','b','MarkerSize',8)
axis([0 3 0 0.8])
M(1)=getframe;
dt=1/128;
for j = 2:1000
  t(j)=t(j-1)+dt;
  x=v0*cos(theta0)*t(j);
  y=v0*sin(theta0)*t(j)-0.5*g*t(j)^2;
  plot(x,y,'o','MarkerFaceColor','b','MarkerSize',8)
  axis([0 3 0 0.8])
  M(j)=getframe;
  if y<=0, break, end
end
pause
movie(M,1)
```

Debugging

- Use “pause” to stop execution at various points
  - After critical places where your script generates numerical outputs
  - After each graph is produced
  - After important comments
- Each time MATLAB reaches a “pause” command, it wait until the user press a key before proceeding.
- Insert the command “keyboard” into an M-file, for instance right before the line where an error may occur, so that you can examine the Workspace of the M-file at that point in its execution.
- Type “return” or “dbcont” to execution of the M-file.

Breakpoints

- Insert breakpoints in the M-file where errors may occur
- Once a breakpoint is inserted in the M-file, you will see a little red dot next to the appropriate line in the Editor/ Debugger.
- When the M-file is executed at the breakpoint (before the line is executed), the execution will stop and control will return to the Command Window.
- Type "dbcont" to continue execution
- Type "dbquit" to exit debugging AND stop execution.
- An article for more debugging commands

  http://blogs.mathworks.com/loren/2007/12/07/useful-debugging-commands-and-tips/

Debug Using Cell Features

- As you develop a MATLAB file, you can use the Editor cell features to evaluate the file cell-by-cell.
- This method helps you to experiment with, debug, and fine-tune your code. You can navigate from cell to cell, and evaluate each cell individually.
- A video of operating with cells

  https://blogs.mathworks.com/videos/2011/07/26/starting-in-matlab-cell-mode-scripts/
- A help document of working with cells

  https://www.mathworks.com/help/matlab/matlab_prog/run-sections-of-programs.html

The Find Function

- The build-in find function is useful for many logical and array indexing applications.
- The function takes a logical matrix expression and return a set of one-dimensional array indices for the elements in the input argument that satisfy the condition.
- Try the following commands and explain what you observe:

```
>>A=rand(3,3)
>>A>0.5
>>find(A>0.5)
```