COMPUTER SKILLS

LESSON 12

Valeria Cardellini
cardellini@ing.uniroma2.it
A.Y. 2015/16

Objectives of this lesson

We'll discuss

- Functions that return more than one value
- Functions that accomplish a task without returning values
- Functions that return values versus printing
- Passing arguments to functions

Recall function definition (lesson 9)

Function header

```
function [output arguments] = functionname(input arg)
%
%comment describing the function
%
      statements here; these must include putting values
      in all the output arguments listed in the
      header
end
```

Comments area

Function body

`Function` and `end` are reserved words and are mandatory

Objectives of this lesson

We'll discuss

- Functions that return more than one value
- Functions that accomplish a task without returning values
- Functions that return values versus printing
- Passing arguments to functions

Functions that return more than one value

```
function [area, circum] = areacirc(rad)
    % areacirc returns the area and
    % the circumference of a circle
    % Format: areacirc(radius)

    area = pi * rad .* rad;
    circum = 2 * pi * rad;
    end
```

```
>> help areacirc
  areacirc returns the area and
  the circumference of a circle
  Format: areacirc(radius)
```

Functions that return more than one value (cont'd)

- `function [area, circum] = areacirc(rad)`

```
>> [a, c]=areacirc(4)
a =
   50.2655
c =
   25.1327
```

```
>> areacirc(4)
ans =
   50.2655
```

```
>> a=areacirc(4)
a =
   50.2655
```

```
>> [~, c]=areacirc(4)
c =
   25.1327
```

Passing a vector as input

- `function [area, circum] = areacirc(rad)`

```
>> [a, c]=areacirc(1:4)
a =
    3.1416   12.5664   28.2743   50.2655
c =
    6.2832   12.5664   18.8496   25.1327
```

```
>> [a, c]=areacirc(1:4)
```

is equivalent to

```
>> [a, c]=areacirc([1,2,3,4])
```

Alternative way to process an input vector

- Create a script with the following MATLAB code

```
for i=1:4
     [a(i), c(i)]=areacirc(i);
end
```

- After running the script, in the workspace there will be

```
>> a
a =
    3.1416   12.5664   28.2743   50.2655
>> c
c =
    6.2832   12.5664   18.8496   25.1327
```

Alternative way to process an input vector

- Create a script with the following MATLAB code

```
for i=1:4
     [a(i), c(i)]=areacirc(i);
end
```

- After running the script, in the wo

```
>> a
a =
    3.1416   12.5664   28.2743   50.2655
>> c
c =
    6.2832   12.5664   18.8496   25.1327
```

Easier and more efficient to write
>> [a, c]=areacirc(1:4)

How to call a function

- From the Command Windows
- From a script
  - Let's write a script to call `areacirc.m`

```
% This script prompts the user for the radius of a circle,
% calls a function to calculate and returns both the area
% and the circumference, and prints the results. To keep
% the script simple, we ignore units and error-checking

radius = input('Please enter the radius of the circle: ');
[area, circ] = areacirc(radius);
fprintf('For a circle with a radius of %.1f,\n', radius)
fprintf('the area is %.1f and the circumference is %.1f\n',...
    area, circ)
```

Example: `perimarea.m`

- Write a function that calculates and returns the perimeter and area of a rectangle
  - Input: length and width
  - Output: perimeter and area

- Call the function from a M-script (`calcareaperim.m`)

calcareaperim.m

```
% Prompt the user for the length and width of a rectangle,
% call a function to calculate and return the perimeter
% and area, and print the result.

length = input('Please enter the length of the rectangle: ');
width = input('Please enter the width of the rectangle: ');
[perim, area] = perimarea(length, width);
fprintf('For a rectangle with length of %.1f ', length)
fprintf(' and width of %.1f,\nthe perimeter is %.1f,', width,
perim)
fprintf(' and the area is %.1f\n', area)
```

```
>> calcareaperim
Please enter the length of the rectangle: 10
Please enter the width of the rectangle: 3
For a rectangle with length of 10.0 and a width of 3.0,
the perimeter is 26.0, and the area is 30.0
```

perimarea.m

```
function [ perim, area ] = perimarea( length, width )
%
%Compute the perimeter and area of a rectangle
%
    perim = 2*(length + width);
    area = length * width;
end
```

Let's call `perimarea.m` from cmd

```
>> [p, a]=perimarea(2,5)
p =
    14
a =
    10
```

```
>> [a, p]=perimarea([1, 5, 10],[2, 4, 6])
Error using  *
Inner matrix dimensions must agree.

Error in perimarea (line 6)
    area = length * width;
```

perimarea.m

```
function [ perim, area ] = perimarea( length, width )
%
%Compute the perimeter and area of a rectangle
%
    perim = 2*(length + width);
    area = length * width;
end
```

This is a matrix product

```
>> [a, p]=perimarea([1, 5, 10],[2, 4, 6])
Error using  *
Inner matrix dimensions must agree.

Error in perimarea (line 6)
    area = length * width;
```

perimarea.m

```
function [ perim, area ] = perimarea( length, width )
%
% Compute the perimeter and area of a rectangle
%
    perim=2.*(length + width);
    area = length .* width;
end
```

You must use the scalar product here

```
>> [p, a] = perimarea([1, 5, 10],[2, 4, 6])
p =
     6    18    32
a =
     2    20    60
```

Another example (3 output args)

- Write a function that converts from seconds to hours + minutes + seconds
  - E.g. 7515 seconds = 2 hours, 5 minutes and 15 seconds
- Solution:
  - Input: total seconds (ts)
  - Output: hh, mm, ss
  - Algorithm
    - hh = integer part of ts/3600; the reminder of ts/3600 is the remaining number of seconds
    - mm = integer part of the (reminder of ts/3600)/60; it is the remaining number of minutes
    - ss = the reminder of (reminder of ts/3600)/60
- For example: 7515 = 3600*2 + 60*5 + 15

breaktime.m

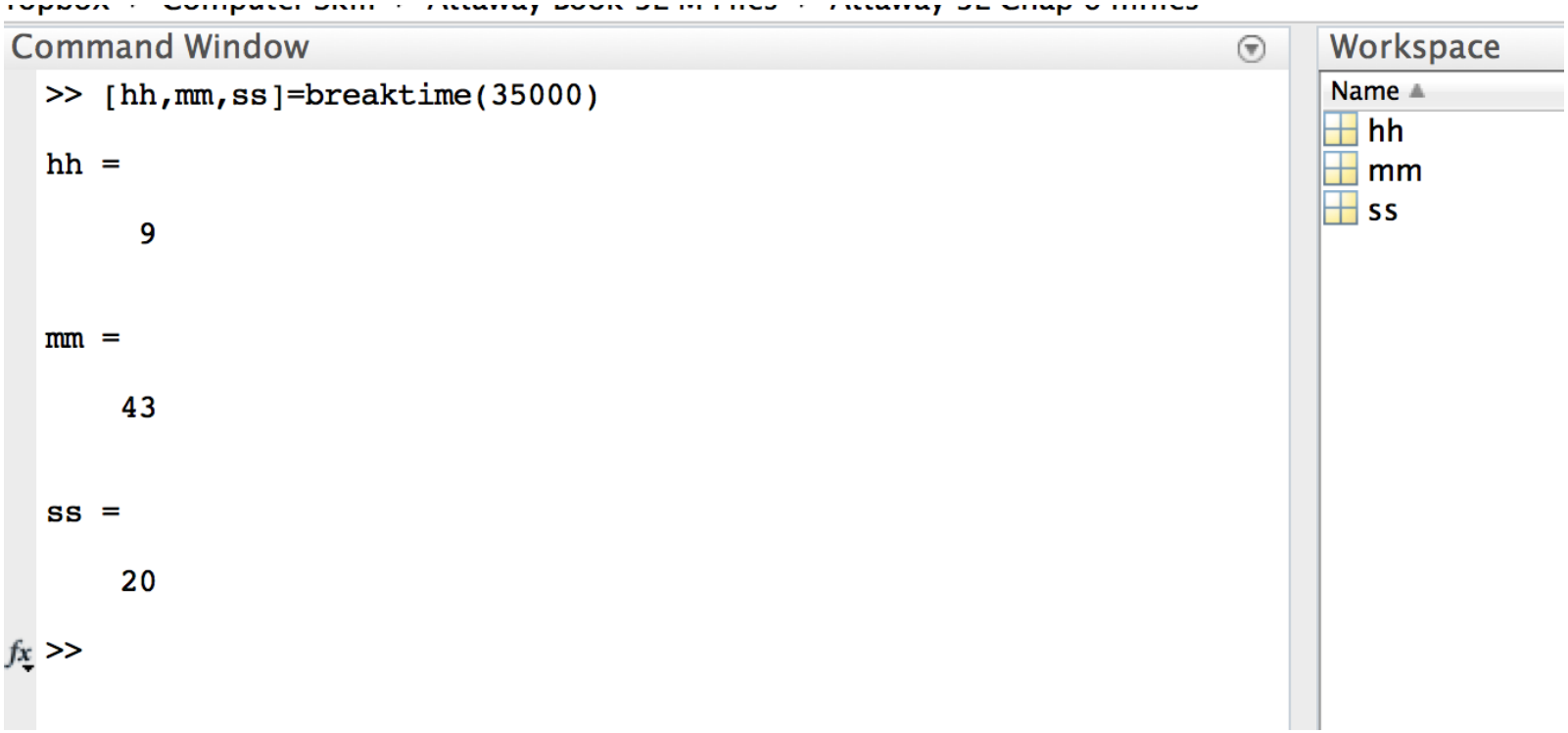
```
function [hours, minutes, secs] = breaktime(totseconds)
% breaktime breaks a total number of seconds into
% hours, minutes, and remaining seconds
% Format: breaktime(totalSeconds)

hours = floor(totseconds/3600);
remsecs = rem(totseconds, 3600);
minutes = floor(remsecs/60);
secs = rem(remsecs,60);
end
```

```
>> [hh,mm,ss]=breaktime(35000)
hh =
     9
mm =
    43
ss =
    20
```

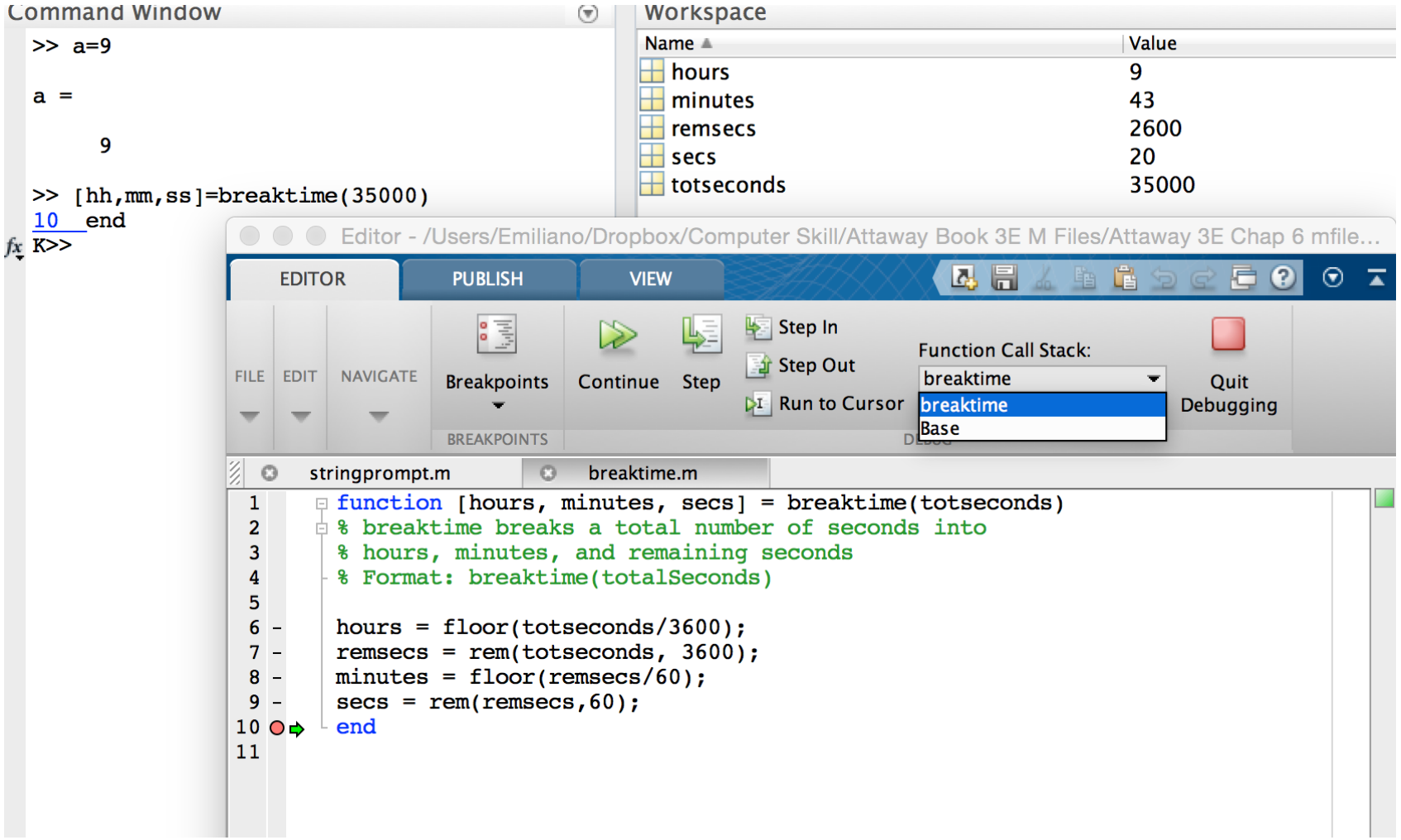
Variable scope: global variables

- `hh, mm, ss` are **global variables**, i.e. part of the base workspace
- `hours, minutes, secs` are **local variables** to the function `breaktime.m`



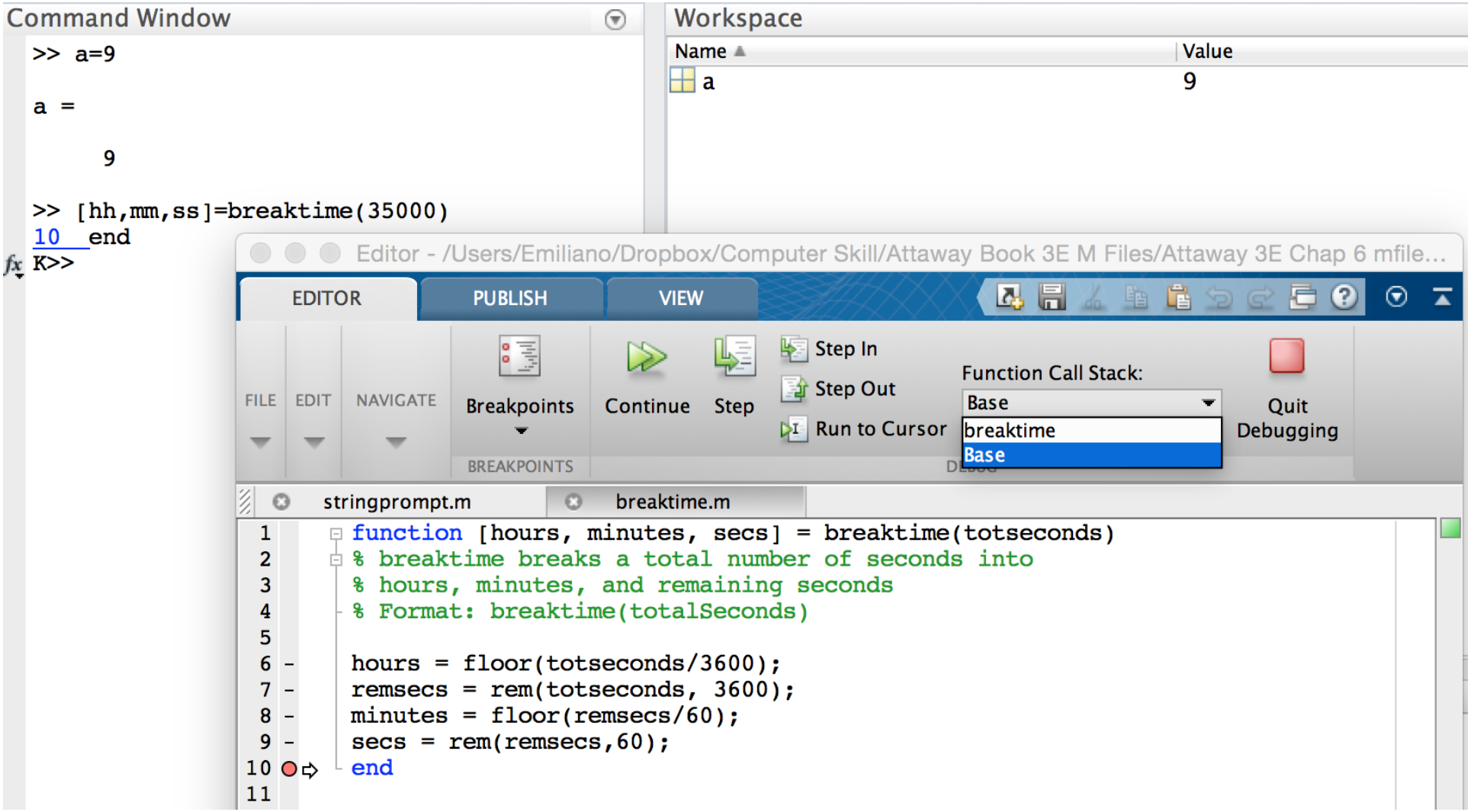
Variable scope: local variables

- `hours, minutes, secs` are **local variables** to the function `breaktime.m`



Variable scope: local variables

- `hours, minutes, secs` are **local variables** to the function `breaktime.m`



Objectives of this lesson

We'll discuss

- Functions that return more than one value
- Functions that accomplish a task without returning values
- Functions that return values versus printing
- Passing arguments to functions

Functions with no output

- **function functionname(input arg)**

```
function printem(a,b)
% printem prints two numbers in a sentence
format
% Format: printem(num1, num2)

fprintf('The first number is %.1f and the
second is %.1f\n',a,b)
end
```

```
>> printem(4,5)
The first number is 4.0 and the second is
5.0

>> x=printem(4,5)
Error using printem
Too many output arguments.
```

Objectives of this lesson

We'll discuss

- Functions that return more than one value
- Functions that accomplish a task without returning values
- Functions that return values versus printing
- Passing arguments to functions

Functions that return values versus printing

```
function calccircum1(radius)
% calccircum1 displays the circumference of a circle
%   but does not return the value
% Format: calccircum1(radius)

disp(2 * pi * radius)
end
```

In this version you cannot store the result of the function in a variable …

```
function circle_circum = calccircum2(radius)
% calccircum2 calculates and returns the
%   circumference of a circle
% Format: calccircum2(radius)

circle_circum = 2 * pi * radius;
end
```

… in this version you can

Functions that return values versus printing (cont'd)

```
>> calccircum1(10)
   62.8319

>> x=calccircum1(10)
Error using calccircum1
Too many output arguments.

>> calccircum2(10)
ans =
   62.8319

>> x=calccircum2(10)
x =
   62.8319
```

Good programming practice

- It is a good programming practice to keep separate the computation and the return of a value from printing that value. That is
  - In the function you should never print the result using **`disp`**, **`printf`**, **`plot`**, …
  - In the function you should never read input from the keyboard using **`input`**
  - In the function you should always assign the final result of any computation to an output variable
  - The printing of results must be left to a specific script/function or to the script that invokes the function

Objectives of this lesson

We'll discuss

- Functions that return more than one value
- Functions that accomplish a task without returning values
- Functions that return values versus printing
- Passing arguments to functions

Passing argument to variables

- Argument are passed to functions using the **call-by-value** method
- It is possible to declare functions without any input parameter
  - **`function [output args] = functionname()`**
  - **`function functionname()`**

```
function printrand()
% printrand prints one random
number
% Format: printrand or
printrand()

fprintf('The random # is %.2f
\n',rand)
end
```

```
function printrandnp
% printrandnp prints one
random number
% Format: printrandnp or
printrandnp()

fprintf('The random # is %.2f
\n',rand)
end
```

Another example

```
function outstr = stringprompt
% stringprompt prompts for a string and returns it
% Format stringprompt or stringprompt()

disp('When prompted, enter a string of any length.')
outstr = input('Enter the string here: ', 's');
end
```

This function returns the string entered by the user

Homework

- Write a version of `calcareaperim.m` that considers:
  - Unit measures and input error-checking
- Do Practice 6.2
- Do Practice 6.3
- Do exercises of Chapter 6

Assignment for Wednesday, Dec. 9th

- Study by yourself (see Chapter 6 of the book):
  - MATLAB program organization (sec. 6.2)
    - Modular programs (sec. 6.2.1)
    - Subfunctions (sec. 6.2.2)
  - Do Practice 6.4 and Practice 6.5
  - Solve the following exercises of Chapter 6
    - 2, 3, 6, 8, 10, 13, 18, 19, 24, 25