MATH 3341: Introduction to Scientific Computing Lab

Libao Jin

University of Wyoming

January 28, 2019

Lab 03: Functions, Control Flows and LaTeX

Anonymous Functions

An *anonymous function* is a function that does not have a function name, but is associated with a variable whose data type is `function_handle`. Anonymous functions can accept inputs and return outputs, just as standard functions do. To define the function $f(x) = x^2 + 1$ we use `f = @(x) x.^2 + 1`, where the inputs (parameters) are defined by the `@` symbol in front of the list of variables in parenthesis.

Examples

- $f(y) = \sin(y)$: `f = @(y) sin(y)`.
- $g(x, y) = x^2 + y^2 - 1$: `g = @(x, y) x .^ 2 + y .^ 2 - 1`.
- $h(z) = e^{\sin z} = e^{f(z)}$: `h = @(z) exp(f(z))`.

Functions

“Fun is where you find it. Look closely, and you can find it in functions.” Defining functions can save you from writing the same code over and over again. Here is the syntax to define a function:

```
function [output_args] = functionName(input_args)
% FUNCTIONNAME Summary of the function
% Details of the function goes here such as syntax, etc.

% function body goes here
end
```

Example: sumProd

```
function [summation, product] = sumProd(x)
% SUMPROD Calculate the summation and product of
% all elements in x
% Syntax:
%   [summation, product] = sumProd(x)
%   summation = sumProd(x)

% Initialize variables summation and product
summation = 0;
product = 1;
for i = 1:length(x)
    summation = summation + x(i);
    product = product * x(i);
end
```

Which do I use?

- Anonymous functions are helpful when you are using functions with a simple definition.
- Otherwise, writing a function file is recommended.

Branching

One of the keys to designing intelligent programs is to give them the ability to make decision. MATLAB provides the `if` and `switch` statements to implement decisions. The `if` comes in two forms: `if` and `if else`. The `if` statement directs a program to execute a statement or statement block if a test condition is true and to skip that statement or block if the condition is false.

Syntax

Run `help if` in the Command Window:

```
 if Conditionally execute statements.
    The general form of the if statement is

       if expression
         statements
       elseif expression
         statements
       else
         statements
       end
```

Examples

```
% Example 1
n = 5;
if mod(n, 2) == 0
    disp('n = 5 is an even number');
else
    disp('n = 5 is an odd number');
end
```

Examples

```
function ret = isLeapYear(year)

if mod(year, 400) == 0
    ret = true;
elseif mod(year, 4) == 0 && mod(year, 100) ~= 0
    ret = true;
else
    ret = false;
end

end
```

switch statement

Run `help switch` in the Command Window:

```
 switch Switch among several cases based on expression.
    The general form of the switch statement is:

        switch switch_expr
          case case_expr,
            statement, ..., statement
          case {case_expr1, case_expr2, case_expr3,...}
            statement, ..., statement
         ...
          otherwise,
            statement, ..., statement
        end
```

Examples

```
function dayOfWeek1(d)

switch d
    case {'Monday', 'Tuesday', 'Wednesday', ...
            'Thursday', 'Friday'}
        fprintf('%s is weekday.\n', d)
    otherwise
        fprintf('%s is weekend.\n', d)
end

end
```

Examples

```
function dayOfWeek2(d)

switch d
    case {'Monday', 'Tuesday', 'Wednesday', ...
            'Thursday', 'Friday'}
        fprintf('%s is weekday.\n', d)
    case {'Saturday', 'Sunday'}
        fprintf('%s is weekend.\n', d)
    otherwise
        fprintf('Error!\n')
end

end
```

Relational Operators

Symbol	Meaning
==	equal to
~=	not equal
>	greater than
<	less than
>=	greater than or equal to
<=	less than or equal to

Logical Operators

Symbol	Meaning
&	element-wise logical AND
|	element-wise logical OR
&&	short-circuit logical AND
||	short-circuit logical OR
~	logical NOT

Repeating Tasks

Question: What should we do if we want to `disp('I am AWESOME!')` for 100 times?

```
disp('I am AWESOME!')
disp('I am AWESOME!')
disp('I am AWESOME!')
...
disp('I am AWESOME!')
```

Better Approach: Using `for` or `while` Loop

Run `help for` in the Command Window:

```
for    Repeat statements a specific number of times.
   The general form of a for statement is:

      for variable = expr, statement, ..., statement end
```

Better Approach: Using for or while Loop

Run help while in the Command Window:

```
 while  Repeat statements an indefinite number of times.
    The general form of a while statement is:

       while expression
         statements
       end
```

Problem solved!

Using `for` loop:

```
for i = 1:100
    disp('I am AWESOME!')
end
```

Using `while` loop:

```
i = 1;
while i <= 100
    disp('I am AWESOME!')
    i = i + 1;
end
```

LATEX Primer

Basic structure

```
\documentclass{article}
\usepackage{amssmb, amsmath}
\author{firstName lastName}
\title{The Title}
\date{\today}
\begin{document}
\maketitle
\section{Demo of Section}
\subsection{Demo of Subsection}
Here is the body.
\end{document}
```

Math Environment/Mode

```
\begin{equation}
% Put equation here
\end{equation}

$$
% Put equation here
$$

$Put inline equation here$
```

Multi-line equations

```
\begin{align}
% Put multiline equation here
\end{align}
```

Examples

```
\begin{equation*}
E = mc^2.
\end{equation*}
```

or

```
$$
E = mc^2.
$$
```

generates

$$E = mc^2.$$

Examples

```
\begin{align}
    \frac{d}{dx} f(g(x))
       & = \frac{d f(g(x))}{d g(x)} \frac{d g(x)}{dx}
    \\ & = f'(g(x)) g'(x).
\end{align}
```

generates

$$
\begin{align}
\frac{d}{dx} f(g(x)) &= \frac{df(g(x))}{dg(x)} \frac{dg(x)}{dx} \tag{1} \\
&= f'(g(x))g'(x). \tag{2}
\end{align}
$$

Subscripts and Supscripts

- a_{1}: `$a_{1}$`
- a^{2}: `$a^{2}$`
- a_{3}^{4}: `$a_{3}^{4}$`
- $a_{\text{sub}}^{\text{sup}}$: `$a_{\text{sub}}^{\text{sup}}$`

Fractions

- $\frac{\text{numerator}}{\text{denominator}}$: `$\frac{numerator}{denominator}$`
- $\frac{3}{5}$: `$\frac{3}{5}$`

Matrices

```
$$
\begin{matrix}
a_{11} & a_{12} \\
a_{21} & a_{22} \\
\end{matrix}
$$
```

Replace matrix with bmatrix, pmatrix, vmatrix, Vmatrix, repectively.

matrix environment

```
$$
\begin{matrix}
a_{11} & a_{12} \\
a_{21} & a_{22} \\
\end{matrix}
$$
```

generates

$$
\begin{matrix}
a_{11} & a_{12} \\
a_{21} & a_{22} \\
\end{matrix}
$$

bmatrix environment

```
$$
\begin{bmatrix}
a_{11} & a_{12} \\
a_{21} & a_{22} \\
\end{bmatrix}
$$
```

generates

$$\begin{bmatrix}
a_{11} & a_{12} \\
a_{21} & a_{22} \\
\end{bmatrix}$$

pmatrix environment

```
$$
\begin{pmatrix}
a_{11} & a_{12} \\
a_{21} & a_{22} \\
\end{pmatrix}
$$
```

generates

$$\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}$$

vmatrix environment

```
$$
\begin{vmatrix}
a_{11} & a_{12} \\
a_{21} & a_{22} \\
\end{vmatrix}
$$
```

generates

$$\begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix}$$

Vmatrix environment

```
$$
\begin{Vmatrix}
a_{11} & a_{12} \\
a_{21} & a_{22} \\
\end{Vmatrix}
$$
```

generates

$$\begin{Vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{Vmatrix}$$