TEHNIČKI FAKULTET
Sveučilište u Rijeci

Medicinski fakultet
Rijeka

Modul Scripts
Matlab & Octave

Željko Jeričević, dr. sc.
Zavod za računarstvo, Tehnički fakultet &
Zavod za biologiju i medicinsku genetiku, Medicinski fakultet
51000 Rijeka, Croatia
Phone: (+385) 51-651 594 Office: 1-48B
E-mail: zeljko.jericevic@riteh.hr
http://www.riteh.uniri.hr/~zeljkoj/Zeljko_Jericevic.html

TEHNIČKI FAKULTET
Sveučilište u Rijeci

Medicinski fakultet
Rijeka

Octave scripts

- Often we need to execute a sequence of commands. Instead of typing these commands directly into Octave's command prompt, we can save them in a text file and then execute that file from the prompt.

TEHNIČKI FAKULTET
Sveučilište u Rijeci

Medicinski fakultet
Rijeka

Octave scripts

- A file composed of a single command or a sequence of commands is referred to as a script file or just a script.
- Sometimes it is useful to save your work, that is, to save the variables that contain the main results of your efforts. We will see how to load the variables back into Octave's workspace.

TEHNIČKI FAKULTET
Sveučilište u Rijeci

Medicinski fakultet
Rijeka

Octave scripts

- What a script is and how to execute it.
- How to control the execution of commands in a script using the if and switch statements.
- To use for, while, and do statements.
- Control exception handling.
- How to save and load your work.
- About printing text to the screen and how to retrieve inputs from the user.

Octave scripts

1. Start the Octave interactive environment and open the editor:
   octave:>> edit
2. Write the following commands in the editor, (hash marks **#number** are used only for reference:

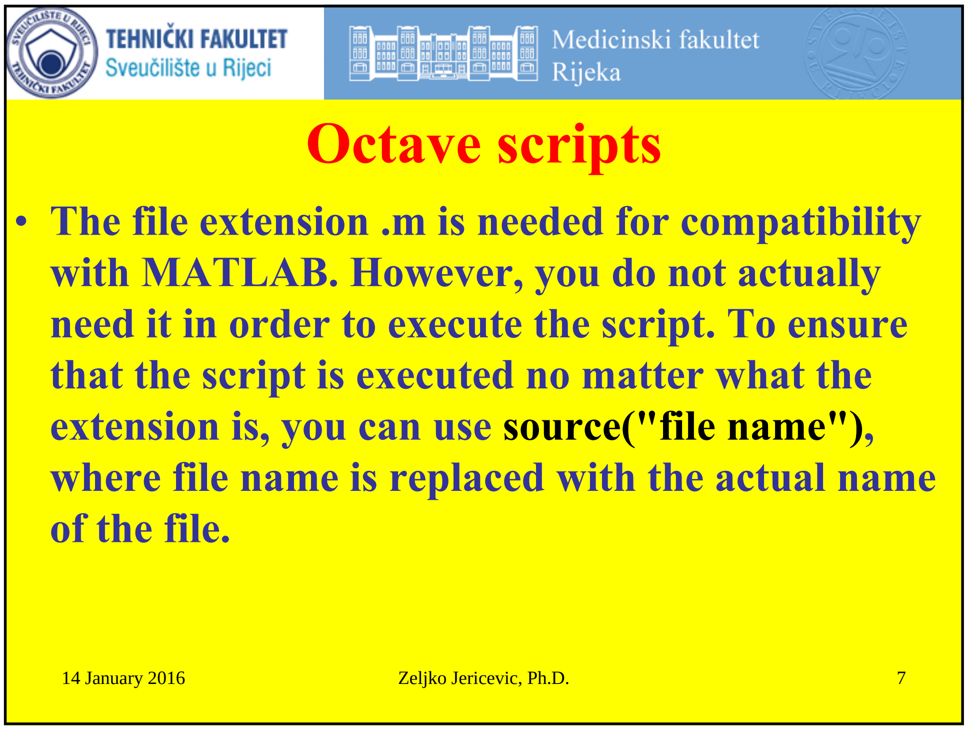
```
A = rand(3,5);          #1
min(min(A))             #2
```

3. Save the file as script41.m (notice the extension .m) under the current directory or anywhere in the Octave search path directory.
4. Now executing the commands in the script file is done by simply typing:

```
octave:>> script41
ans = 0.1201
```

Octave scripts

- Using the editor and two Octave commands were written in the file script41.m Asking the Octave to execute the file script41.m executes each command in the file.
- Since we did not add a semicolon at the end of line #2 in the file script41.m, the command returns the result in ans which is then displayed.

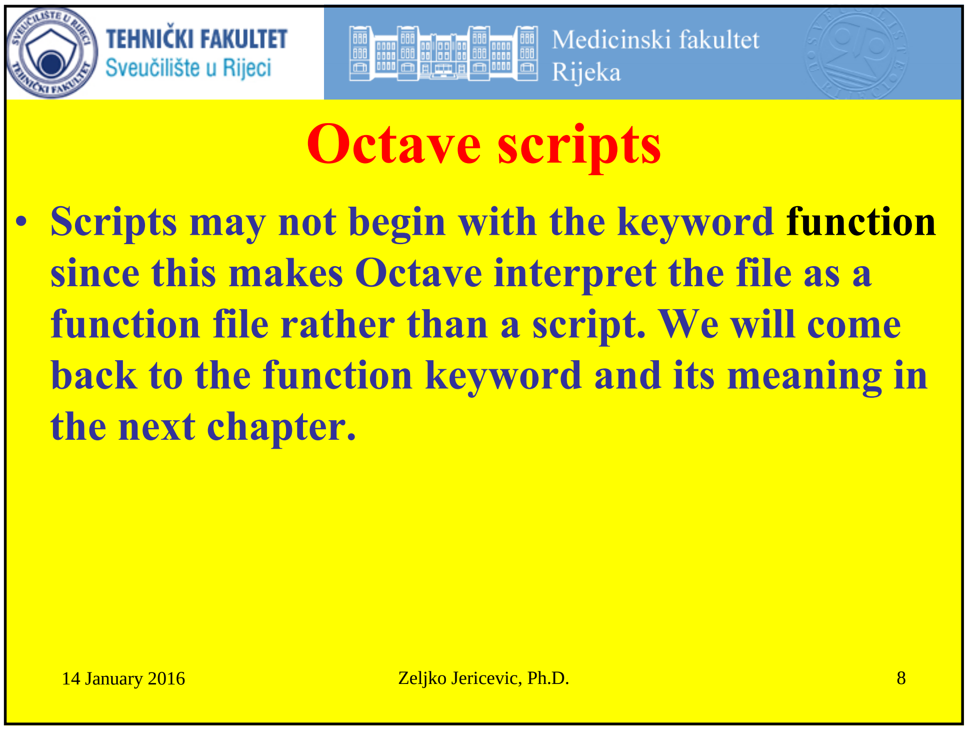

TEHNIČKI FAKULTET Sveučilište u Rijeci

Medicinski fakultet Rijeka

Octave scripts

- The file extension .m is needed for compatibility with MATLAB. However, you do not actually need it in order to execute the script. To ensure that the script is executed no matter what the extension is, you can use **source("file name")**, where file name is replaced with the actual name of the file.

14 January 2016 Zeljko Jericevic, Ph.D. 7


TEHNIČKI FAKULTET Sveučilište u Rijeci

Medicinski fakultet Rijeka

Octave scripts

- Scripts may not begin with the keyword **function** since this makes Octave interpret the file as a function file rather than a script. We will come back to the function keyword and its meaning in the next chapter.

14 January 2016 Zeljko Jericevic, Ph.D. 8

Octave scripts

- Improving the scripts: **input** and **disp**
- It is possible for script to interact with a user using the **input** and **disp** functions.

  Input is in general called using:

  a = input(prompt string, "s")

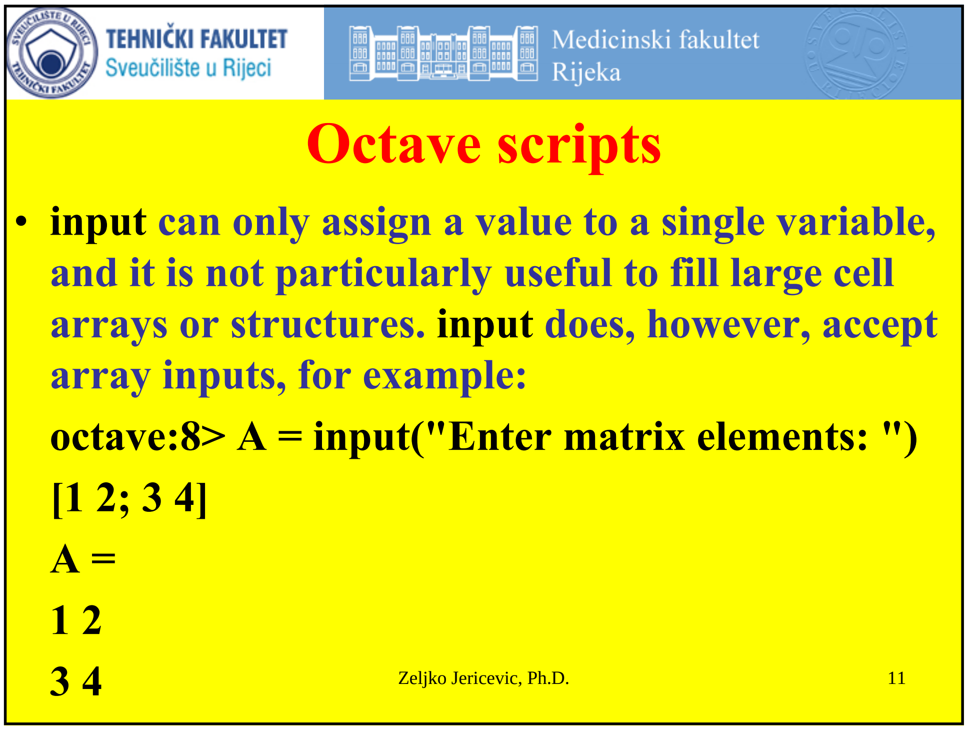
  where prompt string is a text string and **"s"** is an optional argument that must be included if the input is a string.

Octave scripts

- Few examples using the Octave command **input:**

```
octave:>> a = input("Enter a number: ");
Enter a number: 42
octave:>> a
a = 42
octave:>> s = input("Enter a string: " );
Enter a string: Hello World
error: 'Hello' undefined near line 1 column 1
octave:>> s = input("Enter a string: " , "s");
Enter a string: Hello World
octave:>> ischar(s)
ans = 1
```

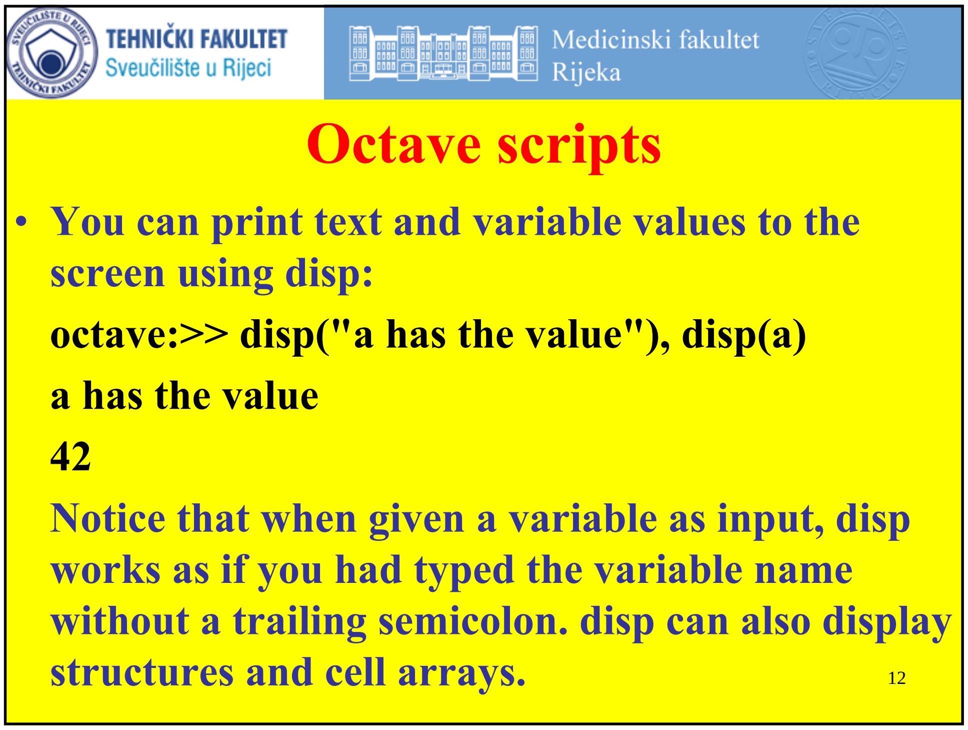
"s" must be in **input** call if argument is a string!



Octave scripts

- **input** can only assign a value to a single variable, and it is not particularly useful to fill large cell arrays or structures. **input** does, however, accept array inputs, for example:

```
octave:8> A = input("Enter matrix elements: ")
[1 2; 3 4]
A =
1 2
3 4
```



Octave scripts

- You can print text and variable values to the screen using disp:

```
octave:>> disp("a has the value"), disp(a)
a has the value
42
```

Notice that when given a variable as input, disp works as if you had typed the variable name without a trailing semicolon. disp can also display structures and cell arrays.

Octave scripts

- Script file script42.m

```
nr = input("Enter number of rows in matrix: ");        #1
nc = input("Enter number of columns in matrix: ");   #2
A = rand(nr,nc);                                      #3
minA = min(min(A));                                   #4
disp("The minimum of A is");                          #5
disp(minA);                                           #6
```

Octave scripts

- Executing script42.m:

```
octave:>>script42
Enter the number of rows in the matrix: 12
Enter the number of columns in the matrix: 20
The minimum of A is
0.00511
```

- The result is printed using the disp function.

Octave scripts

- Flush the buffer

  On some systems, the text that you want to print to the screen may be buffered, i.e. text can sit in a queue and wait to be displayed and can potentially be an annoying problem. Flush the buffer before you prompt the user for input using the command **fflush(stdout)**, where **stdout** is the output buffer (or stream). For example:

```
octave:>> fflush(stdout);
octave:>> a = input("Enter a number: ");
Enter a number: 42
```

Octave scripts

- Comments

  When script becomes larger and more complicated, it is useful to have comments explaining it. Any line beginning with a hash mark # or percentage sign % (MATLAB compatible!) will be ignored by the interpreter :

```
octave:>> a=42; a
a = 42
octave:>> # a
octave:>> % a
```

Octave scripts

- Script file script43.m

```
% flush the output stream
fflush(stdout);
% Get the number of rows and columns from the user
nr = input("Enter number of rows in matrix: ");        #1
nc = input("Enter number of columns in matrix: ");   #2
% Instantiate A and assign elements random numbers
A = rand(nr,nc);                                      #3
% Evaluate the minimum value
minA = min(min(A));                                   #4
% Print the result to the screen
disp("The minimum of A is");                          #5
disp(minA);                                           #6
```

Octave scripts

- Very long commands

  Sometimes you need to write a very long command. To break a command into several lines, use three full stops (periods) ... or back slash \ . For example:

```
nr = input("Enter the number of rows ...
in the matrix: ");
```

or:

```
nr = input("Enter the number of rows \
in the matrix: ");
```

The backslash is used in Unix to indicate that the line continues.

The three full stops are used in MATLAB.

TEHNIČKI FAKULTET
Sveučilište u Rijeci

Medicinski fakultet
Rijeka

Octave scripts

- Workspace

  After executing an Octave script, the variable instantiated in the script is stored in the current workspace and is accessible after the script has finished executing.

```
octave:>> clear
octave:>> who
octave:>> script41
ans = 0.62299
octave:>> who
Variables in the current scope:
A ans
```

19

TEHNIČKI FAKULTET
Sveučilište u Rijeci

Medicinski fakultet
Rijeka

Octave scripts

- Workspace

  It is important to keep track of what variables you have instantiated, including variables instantiated in the scripts. Assume that we have happily forgotten that A was instantiated through **script41** and we now type the following command:

```
octave:20> A(:,1) = [0:10]
error: A(I,J,...) = X: dimension mismatch.
```

20

Octave scripts

- For GNU/Linux and MacOS X users

  Call an Octave script directly from the shell:

1. Find out where Octave is installed. Typically the executable will be at: /usr/bin/octave.
2. Add the following line at the very start of your script file

   #! /usr/bin/octave –qf
3. Save the file and make it executable.
4. At the shell prompt, run the file as any other executable program

Octave scripts

- Statements

  A simple script is just a sequence of commands. Real programs can be constructed in the same way using , loops, flow control and logical statements.

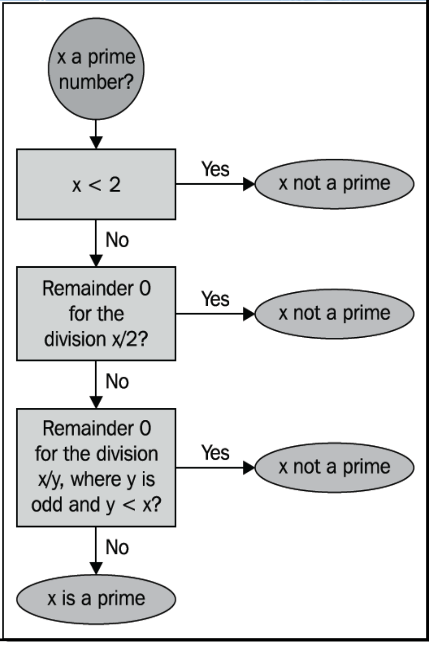
TEHNIČKI FAKULTET
Sveučilište u Rijeci

Medicinski fakultet
Rijeka

Octave sc

- **Prime numbers (naïve program)**

  if, for, while, and so on by evaluating whether a number is a prime number or not. A prime number *x* is a natural positive number that has exactly two divisors—1 and itself. A divisor *y* is a natural positive number larger than 1 such that the division *x*/*y* has no remainder.


x a prime number?
x < 2
Yes
x not a prime
No
Remainder 0 for the division x/2?
Yes
x not a prime
No
Remainder 0 for the division x/y, where y is odd and y < x?
Yes
x not a prime
No
x is a prime

TEHNIČKI FAKULTET
Sveučilište u Rijeci

Medicinski fakultet
Rijeka

Octave scripts

- **Decision making – the if statement**

  From the program flow chart, it is seen that we can decide if a number is not a prime by evaluating whether the number is smaller than 2 or if the remainder of the division *x*/2 is zero using if and elseif statements:

```
if condition 1
    do something (body)
elseif condition 2
    do something else (body)
...
else
    do something if no conditions are met (body)
endif
```

24

TEHNIČKI FAKULTET
Sveučilište u Rijeci

Medicinski fakultet
Rijeka

Octave scripts

- The code snippet that checks if the two first conditions in the flow chart are met:

```
if ( x<2 )
     disp("x not a prime");
elseif ( rem(x,2)==0 )
     disp("x not a prime");
else
     disp("x could be a prime number");
endif
```

TEHNIČKI FAKULTET
Sveučilište u Rijeci

Medicinski fakultet
Rijeka

Octave scripts

- The **rem** function returns the remainder of ***x*/2**. It is important to underline that if the first if statement body is executed, meaning that if the comparison operation ***x*<2** evaluates to true, the **elseif** and **else** statements are not evaluated. The Octave interpreter simply jumps to the line after the endif statement. Likewise, if **rem(x,2)==0** is true, the else statement is not executed. This happens only if both the conditions to **if** and **elseif** are false.

Octave scripts

- **rem and mod functions**
- **Mapping Function: rem (*x, y*)**
- **Mapping Function: fmod (*x, y*)**
  - **Return the remainder of the division *x* / *y*, computed using the expression**
  - ***x* - *y* .* fix (*x* ./ *y*)**
  - **An error message is printed if the dimensions of the arguments do not agree, or if either of the arguments is complex.**

Octave scripts

- **rem and mod functions**
- **Mapping Function: mod (*x, y*)**
  - **Compute the modulo of *x* and *y*. Conceptually this is given by:**
  - ***x* - *y* .* floor (*x* ./ *y*)**
  - **and is written such that the correct modulus is returned for integer types. This function handles negative values correctly. That is, mod (-1, 3) is 2, not -1, as rem (-1, 3) returns. mod (*x*, 0) returns *x*.**
  - **An error results if the dimensions of the arguments do not agree, or if either of the arguments is complex.**

Octave scripts

- Boolean operators

  Octave has a set of so-called Boolean operators. They enable to include several comparisons within a single statement in order to avoid repeating code.
- Octave's Boolean operators are divided into element-wise and short-circuit operators.

Octave scripts

- Boolean operators

  Element-wise Boolean operators

  There are three element-wise Boolean operators, namely, &, |, and !. A couple of examples how to use them:

```
octave:>> A=eye(2); B=[1 2;3 4];
octave:>> A==eye(2) & B==eye(2)
ans =
1 0
0 0
```

Octave scripts

- **Boolean operators**

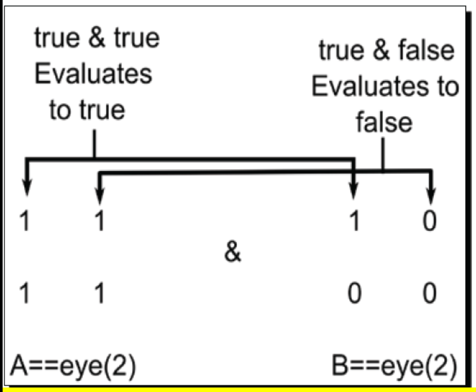
The | operator evaluates to true if the left Boolean is true or if the right Boolean is true. For example:

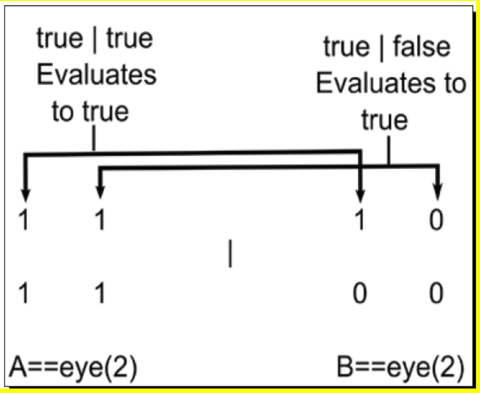
```
octave:>> A==eye(2) | B==eye(2)
ans =
1 1
1 1
```

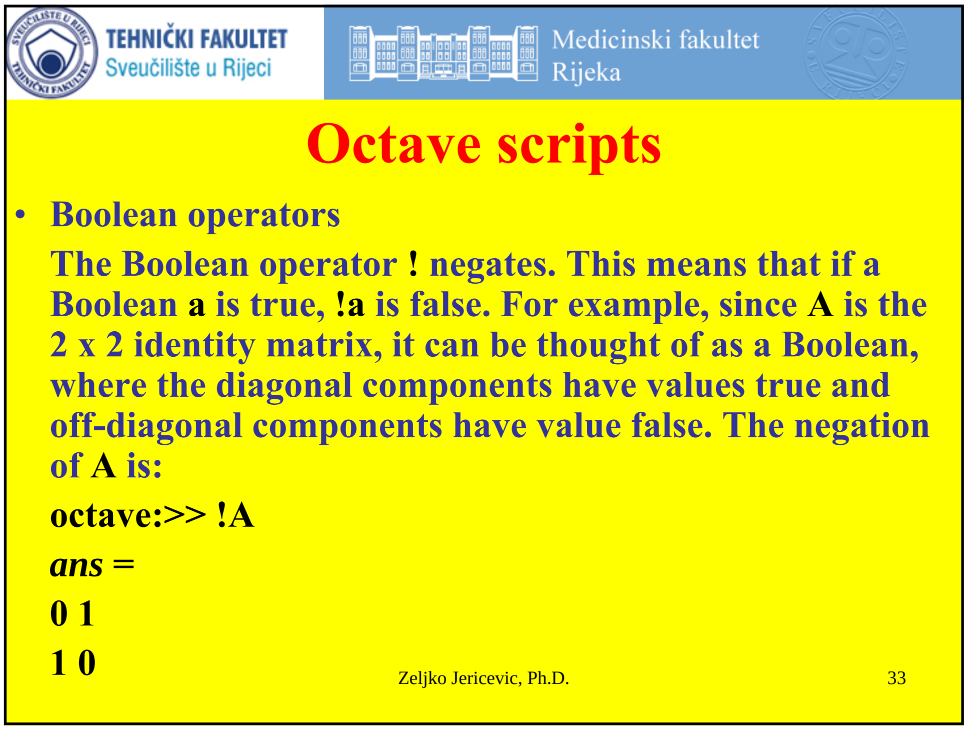
since A is simply equal to eye(2).

Octave scripts

A=eye(2); B=[1 2;3 4];







TEHNIČKI FAKULTET
Sveučilište u Rijeci

Medicinski fakultet
Rijeka

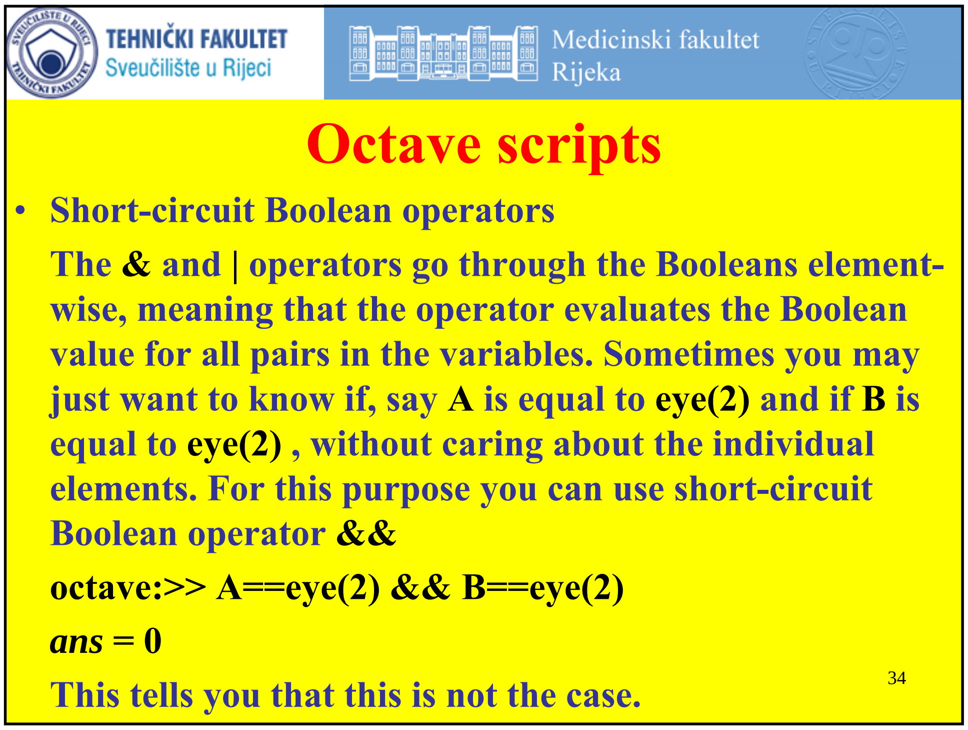
Octave scripts

- **Boolean operators**

  The Boolean operator ! negates. This means that if a Boolean a is true, !a is false. For example, since A is the 2 x 2 identity matrix, it can be thought of as a Boolean, where the diagonal components have values true and off-diagonal components have value false. The negation of A is:

```
octave:>> !A
ans =
0 1
1 0
```

Zeljko Jericevic, Ph.D. 33



TEHNIČKI FAKULTET
Sveučilište u Rijeci

Medicinski fakultet
Rijeka

Octave scripts

- **Short-circuit Boolean operators**

  The & and | operators go through the Booleans element-wise, meaning that the operator evaluates the Boolean value for all pairs in the variables. Sometimes you may just want to know if, say A is equal to eye(2) and if B is equal to eye(2) , without caring about the individual elements. For this purpose you can use short-circuit Boolean operator &&

```
octave:>> A==eye(2) && B==eye(2)
ans = 0
```

This tells you that this is not the case.

34

Octave scripts

- Boolean operators have lower precedence than comparison operators.

  Short-circuit Boolean operators

  Likewise we can use the the short-circuit operator || to check if **A** or **B** is equal to eye(2):

  octave:>> A==eye(2) || B==eye(2)

  ans **= 1**

  This is because when A was instantiated, it was set to eye(2).

Octave scripts

- The table below summarizes the output from the Boolean operators **&, &&, |** and **||**

Operators	Boolean 1	Boolean 2	Result
& and &&	T	T	T
	T	F	F
	F	F	F
| and ||	T	T	T
	T	F	T
	F	F	F

Octave scripts

- Using Boolean operators with an **if** statement

  Instead of using the **if** and **elseif** statement construction in the previous code snippet, we can now apply both conditions to the same **if** statement:

```
if ( x<2 | rem(x,2)==0 )
    disp("x is not a prime");
else
    disp("x could be a prime");
endif
```

Octave scripts

- If *x* is, for example, 2, then the comparison operation **x<2** evaluates to **false** and **rem(x,2)==0** evaluates to **true**, so the **|** Boolean operator evaluates to **true** according to the table and the condition in the **if** statement is met. On the other hand, if *x* is 9, then the comparison operations **x<2** and **rem(x,2)==0** evaluate to **false** and the **if** statement body is not executed. In this example, you could also use the short-circuit Boolean operator **||** because *x* is a simple scalar.

TEHNIČKI FAKULTET
Sveučilište u Rijeci

Medicinski fakultet
Rijeka

Octave scripts

- Like other programming languages, you can have if statement constructions within an **if**, **elseif**, or **else** statement. For example:

```
if ( x<2 | rem(x,2)==0 )
    disp("x is not a prime");
else
    if ( x>3 & rem(x,3)==0 )
        disp("x not a prime");
    else
        disp("x could be a prime");
    endif
endif
```

TEHNIČKI FAKULTET
Sveučilište u Rijeci

Medicinski fakultet
Rijeka

Octave scripts

- The switch statement syntax

```
switch option
case option
    do something (body)
case option
    do something else (body)
...
otherwise
    do something default (body)
endswitch
```

TEHNIČKI FAKULTET Sveučilište u Rijeci — Medicinski fakultet Rijeka

Octave scripts

- The use of the switch statement can be illustrated by rewriting the previous code snippet:

```
switch ( x<2 | rem(x,2)== 0 )
case 1
     disp("x not a prime");
otherwise
     disp("x could be a prime");
endswitch
```

Unlike in C, switch is not falling through statement!

TEHNIČKI FAKULTET Sveučilište u Rijeci — Medicinski fakultet Rijeka

Octave scripts

- Loops

  From the program flow chart shown above, it can be seen that we need to calculate the remainder of the division *x*/*y* for all values of *y* that are smaller than *x*, so if *x* is large, we need to call **rem** many times. In Octave, we can do so using the **for**, **while**, or **do** statements.
- The for statement syntax:

```
for condition
     do something (body)
endfor
```

TEHNIČKI FAKULTET
Sveučilište u Rijeci

Medicinski fakultet
Rijeka

Octave scripts

- Loops

The for statement and the corresponding endfor construct a so-called for-loop. A for-loop is executed as long as condition is true. Let us see a code snippet:

```
for y=3:x-1
    if ( rem(x, y)==0 )
            disp("x not a prime");
    endif
endfor
```

TEHNIČKI FAKULTET
Sveučilište u Rijeci

Medicinski fakultet
Rijeka

Octave scripts

- Loops

From definition of prime numbers, there is no need to calculate the remainder of all the even numbers. We can skip the even numbers by letting y have the values 3, 5, 7, and so on. This is done by starting at *y*=3 and then incrementing *y* by 2, which is done by replacing line 1 in the code snippet with the following:

```
for y=3:2:x-1
```

In this way, we carry out only half the computations

Octave scripts

- In the previous code, the remainder is calculated for all *y*<*x*, even though we know that if it equals 0 for just one single *y*, *x* is not a prime. The **break** command (or keyword) will make Octave to break out of the loop, but continue executing any code after the loop, like this:

```
for y=3:2:x-1
    if ( rem(x,y)==0 )
            disp("x is not a prime");
            break;
    endif
endfor
```

Octave scripts

- Instead of breaking out of a loop if some condition is met, you may want the loop to continue. You can tell Octave to continue looping via the **continue** keyword. This keyword should be used with some care.

Octave scripts

- The while and do statements

  As an alternative to for loops, you can use while and do statements. The syntax for the while construction is:

```
while condition
    do something (body)
endwhile
```

Octave scripts

- The reimplementation of the code snippet above is straight forward:

```
y=3;
while y < x
    if ( rem(x,y)==0 )
            disp("x is not a prime");
            break;
    endif
    y = y + 2;
endwhile
```

Octave scripts

- Instead of ending the blocks with **endif**, **endfor**, and so forth, you can simply use **end**. This is compatible with MATLAB. Trying to use the **continue** keyword instead, leads to infinite loop:

```
y=3;
while y<x
      if ( rem(x,y)!=0 )
               continue;
      else
               disp("x is not a prime");
      endif
      y = y + 2;
endwhile
```

Octave scripts

- This leads to an infinite loop because continue will make the interpreter skip all the commands inside the loop body including the line *y*=*y*+2, meaning that y is never incremented and the comparison operation *y*<*x* is always true.

TEHNIČKI FAKULTET
Sveučilište u Rijeci

Medicinski fakultet
Rijeka

Octave scripts

- You can force Octave to execute a loop once and then continue that loop if a certain condition is met. The syntax is:

```
do
    something (body)
until condition
```

TEHNIČKI FAKULTET
Sveučilište u Rijeci

Medicinski fakultet
Rijeka

Octave scripts

- The code example will now be:

```
y=3;
do
    if ( rem(x,y)==0 )
            disp("x is not a prime");
            break;
    endif
    y = y + 2;
until y>=x
```

Note that we have to use the comparison *y*>=*x* (and not *y*<*x*) here.

TEHNIČKI FAKULTET
Sveučilište u Rijeci

Medicinski fakultet
Rijeka

Octave scripts

- Incremental operators

  In the previous examples, the variable *y* is incremented. Octave supports the C style incremental and decremental operators. For example, to increment a variable *y* with 1, we can use *y*+=1 or *y*++ which are both equivalent to *y*=*y*+1. In general you can increment *y* by any number, *x*, using *y*+=*x*. To decrement, we simply use *y*-- or in general *y*-=*x*.

  Since Octave is a vectorized programming language, the incremental operators also work on multidimensional arrays, incrementing element-wise.

TEHNIČKI FAKULTET
Sveučilište u Rijeci

Medicinski fakultet
Rijeka

Octave scripts

- Incremental operators

  Strictly speaking, the incremental operation *y*++ will return the old value of *y* before incrementing it. For example:

```
octave:>> y=0;  y++
ans = 0
octave:>> y
ans = 1
```

Octave scripts

- Nested loops

  Like nested **if**, nested loops are possible. For example:

```
sum=0;
for n=1:nr
     for m=1:nc
          sum += A(n,m);
     endfor
endfor
```

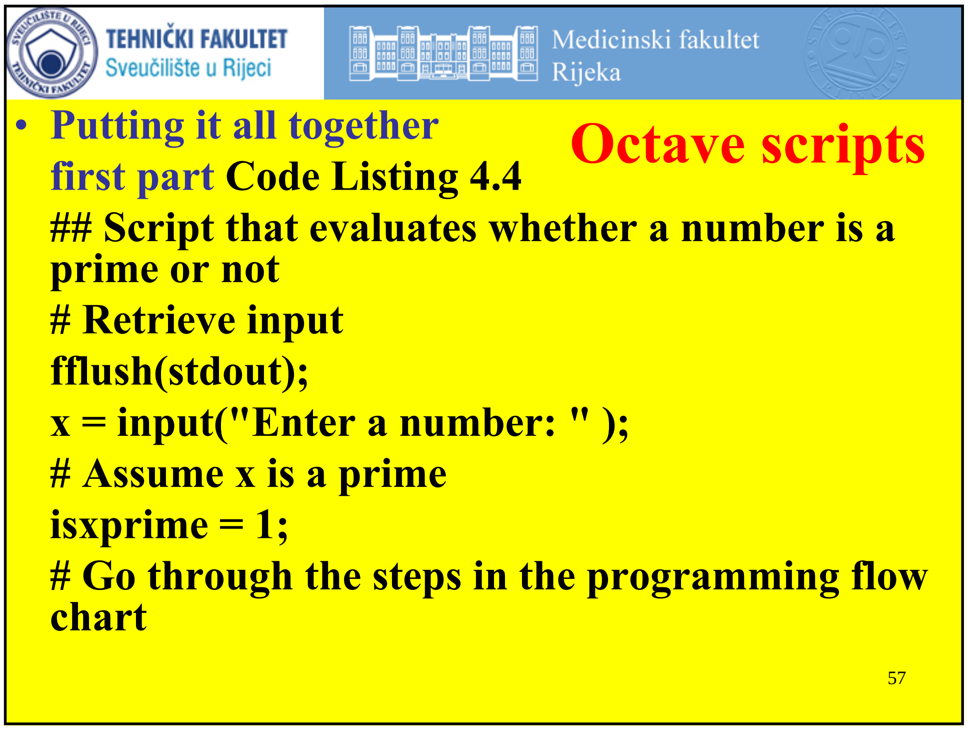
- Note: function **sum** computes the sum of an array much quicker than the nested loop construction above.
- If you use nested loops, **break** will break out of the inner-most loop.

Octave scripts

- Putting it all together

  Code Listing 4.4

```
## Script that evaluates whether a number is a prime or not
# Retrieve input
fflush(stdout);
x = input("Enter a number: " );
# Assume x is a prime
isxprime = 1;
# Go through the steps in the programming flow chart
if ( x!=2 & (x<2 | rem(x,2)==0) )
     isxprime = 0;
else
     for y=3:2:x-1
          if ( rem(x,y)==0 )
               isxprime = 0;
               break;
          endif
     endfor
endif
if ( isxprime )
     disp("x is a prime");
else
     disp("x is not a prime");
endif
```



TEHNIČKI FAKULTET
Sveučilište u Rijeci

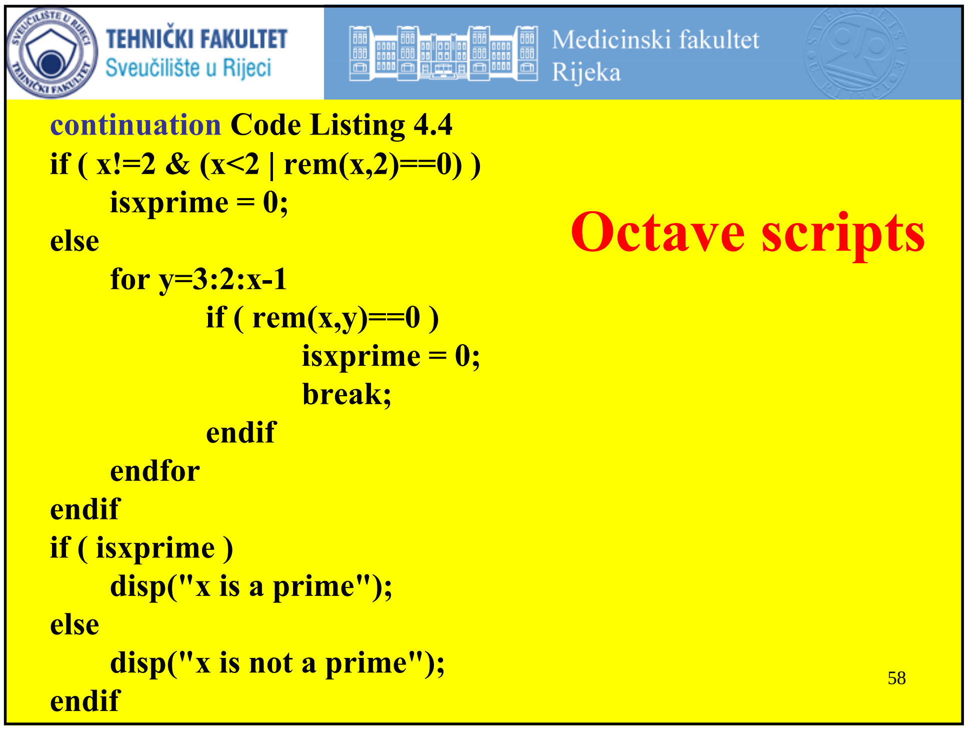
Medicinski fakultet
Rijeka

Octave scripts

- **Putting it all together**
  first part Code Listing 4.4

```
## Script that evaluates whether a number is a
prime or not
# Retrieve input
fflush(stdout);
x = input("Enter a number: " );
# Assume x is a prime
isxprime = 1;
# Go through the steps in the programming flow
chart
```

57



TEHNIČKI FAKULTET
Sveučilište u Rijeci

Medicinski fakultet
Rijeka

Octave scripts

continuation Code Listing 4.4

```
if ( x!=2 & (x<2 | rem(x,2)==0) )
    isxprime = 0;
else
    for y=3:2:x-1
        if ( rem(x,y)==0 )
            isxprime = 0;
            break;
        endif
    endfor
endif
if ( isxprime )
    disp("x is a prime");
else
    disp("x is not a prime");
endif
```

58

Octave scripts

- Testing the script. Save the file as script44.m, for example, and execute it at the Octave prompt:

```
octave:>> script44
Enter a number: 2
x is a prime
octave:>> script44
Enter a number: 109221
x is not a prime
octave:>> script44
Enter a number: 109211
x is a prime
```

Octave scripts

- Octave has its own built-in function, **isprime**, that can tell you if a number is a prime or not. Check if your script agrees with Octave's function and notice the difference in execution speed for large input values.

TEHNIČKI FAKULTET
Sveučilište u Rijeci

Medicinski fakultet
Rijeka

Octave scripts

- If you want to calculate a whole sequence of primes, then it is not practical to use the script from Code Listing 4.4. However, we can easily modify it to meet our needs. The strategy is the same, but instead of prompting the user for a specific number, she will have to enter the number of primes she wants to retrieve starting from 2. The primes are stored in the array prime_sequence:

TEHNIČKI FAKULTET
Sveučilište u Rijeci

Medicinski fakultet
Rijeka

Octave scripts

Code Listing 4.5

```
## Script that calculates a sequence of primes
# Clear the prime array
clear prime_sequence;
# Retrieve user input
fflush(stdout);
nprimes = input("Enter number of primes (>0): ");
# Initializing x to 2 - gets rid of a comparison
# operation inside the loop
x = 2;
# Initialize counter to 1 since 2 is a prime
prime_counter = 1;
prime_sequence(prime_counter) = x;
while prime_counter<nprimes
        # Assume x is a prime number
        is_x_prime = 1;
        # if the remainder of x/2 or x/y for y<x is zero then
        #x is not a prime.
        if ( rem(x,2)==0 )
                is_x_prime = 0;
        else
                for y=3:2:x-1
                        if ( rem(x,y)==0 )
                        is_x_prime = 0;
                        break;
                        endif
                endfor
        endif
        # if is_x_prime is true (1) then save the value of
        # x in an array
        if ( is_x_prime )
                prime_counter++;
                prime_sequence(prime_counter) = x;
        endif
        x++;
endwhile
```

Octave scripts

- Save the script as script45.m and execute it:

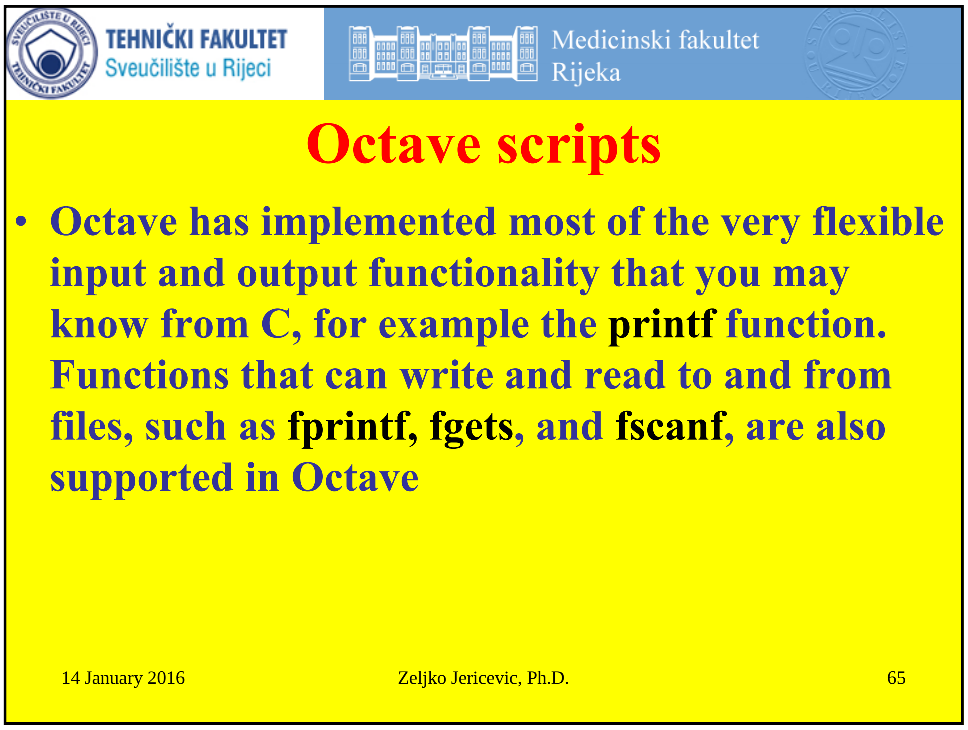
```
octave:>> script45
Enter number of primes (>0): 10
octave:>> prime_sequence
prime_sequence
2 3 5 7 11 13 17 19 23 29
```

Try to increase the number of primes in the sequence, say enter 20, 100, and 1000. Notice how long it takes to compute the primes as you increase the length of the sequence.

Octave scripts

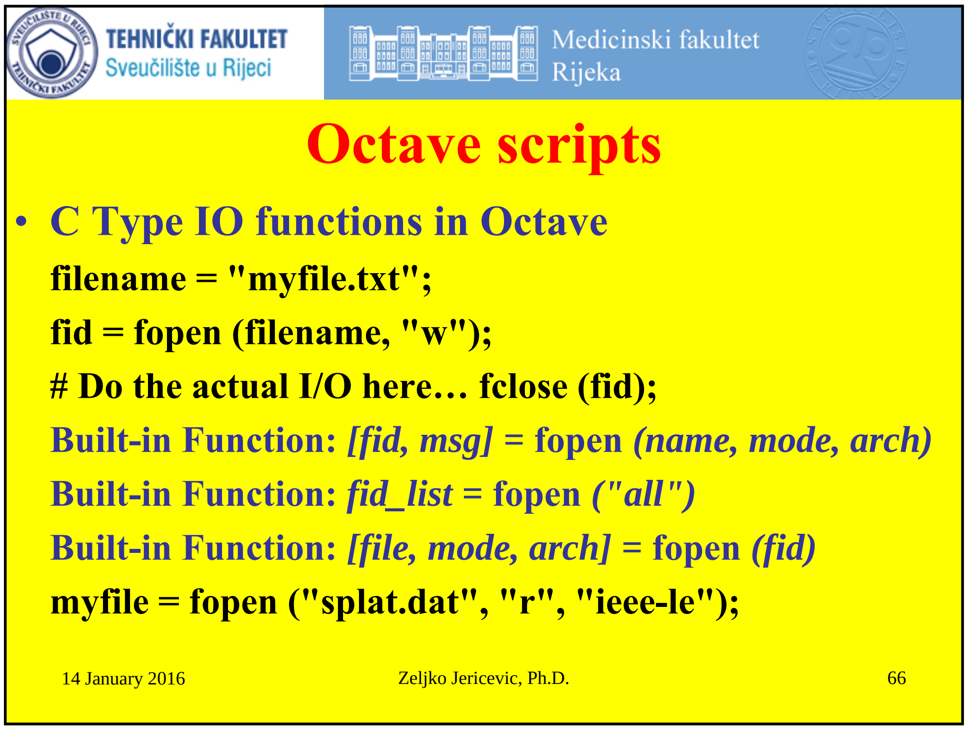
- Added flexibility – C style input and output functions

The function **disp** is easy to use. However, it has limitations. For example, we can display only a single variable with **disp** and it always prints a newline character after displaying the variable value.



Octave scripts

- Octave has implemented most of the very flexible input and output functionality that you may know from C, for example the **printf** function. Functions that can write and read to and from files, such as **fprintf**, **fgets**, and **fscanf**, are also supported in Octave



Octave scripts

- C Type IO functions in Octave

filename = "myfile.txt";

fid = fopen (filename, "w");

# Do the actual I/O here… fclose (fid);

Built-in Function: *[fid, msg]* = fopen *(name, mode, arch)*

Built-in Function: *fid_list* = fopen *("all")*

Built-in Function: *[file, mode, arch]* = fopen *(fid)*

myfile = fopen ("splat.dat", "r", "ieee-le");

TEHNIČKI FAKULTET
Sveučilište u Rijeci

Medicinski fakultet
Rijeka

Octave scripts

- C Type IO functions in Octave
  fid = fopen *(name, mode, arch)*
- The possible values 'mode' may have are
- 'r'
  - Open a file for reading.
- 'w'
  - Open a file for writing. The previous contents are discarded.
- 'a'
  - Open or create a file for writing at the end of the file.
- 'r+'
  - Open an existing file for reading and writing.
- 'w+'
  - Open a file for reading or writing. The previous contents are discarded.
- 'a+'

TEHNIČKI FAKULTET
Sveučilište u Rijeci

Medicinski fakultet
Rijeka

Octave scripts

- C Type IO functions in Octave
  fid = fopen *(name, mode, arch)*
- The parameter *arch* is a string specifying the default data format for the file. Valid values for *arch* are:
- 'native'
  - The format of the current machine (this is the default).
- 'ieee-be'
  - IEEE big endian format.
- 'ieee-le'
  - IEEE little endian format.

Octave scripts

- C Type IO functions in Octave
- Built-in Function: fclose *(fid)*
- Built-in Function: fclose *("all")*

Octave scripts

- C Type IO functions in Octave

  Once a file has been opened for writing a string can be written to the file using the **fputs** function. The following example shows how to write the string ‘Free Software is needed for Free Science’ to the file ‘free.txt’.

```
filename = "free.txt";
fid = fopen (filename, "w");
fputs (fid, "Free Software is needed for Free Science");
fclose (fid);
```

- Built-in Function: fputs *(fid, string)*
- Built-in Function: puts *(string)*

Octave scripts

- C Type IO functions in Octave

  Built-in Function: *str* = fgets *(fid)*

  Built-in Function: *str* = fgets *(fid, len)*

  - Read characters from a file, stopping after a newline, or EOF, or *len* characters have been read. The characters read, including the possible trailing newline, are returned as a string.
  - If *len* is omitted, fgets reads until the next newline character.
  - If there are no more characters to read, fgets returns -1.

Octave scripts

- C Type IO functions in Octave
- Built-in Function: printf *(template, …)*

  Print optional arguments under the control of the template string *template* to the stream stdout and return the number of characters printed.
- Built-in Function: fprintf *(fid, template, …)*

  This function is just like printf, except that the output is written to the stream *fid* instead of stdout. If *fid* is omitted, the output is written to stdout.

TEHNIČKI FAKULTET
Sveučilište u Rijeci

Medicinski fakultet
Rijeka

Octave scripts

- **printf** is an acronym for print formatted text.

  General syntax is: **printf(template, ...)**

  Template is a text string and can also include text format specifiers and/or escape sequences. The **...** indicates optional arguments. For example:

```
octave:>>for n=1:4
>printf("n is %d\n", n);
>endfor
n is 1
n is 2
n is 3
n is 4
```

TEHNIČKI FAKULTET
Sveučilište u Rijeci

Medicinski fakultet
Rijeka

Octave scripts

- **printf("n is %d\n", n);**

  Here the template includes a text **n is**, a format specifier **%d**, and the escape sequence **\n**. **%d** instructs printf to print the value of **n** as an integer and the sequence **\n** means new line. Since we specify a format, there must be an argument with a value to print. This is given by the value of **n.**

TEHNIČKI FAKULTET
Sveučilište u Rijeci

Medicinski fakultet
Rijeka

Octave scripts

- Format specifiers and Escape sequence

%d Integer format \n Newline

%f Floating point format \t Horizontal tab

%e or %E Scientific floating point format \b Backspace

%c Character format \r Carriage return

%s String format

TEHNIČKI FAKULTET
Sveučilište u Rijeci

Medicinski fakultet
Rijeka

Octave scripts

- Try to change the format specifier and escape sequence characters. For example:

```
octave:>> for n=1:4
>printf("n is %f \t", n);
>endfor
n is 1.0000 n is 2.0000 n is 3.0000 n is 4.0000
```

Octave scripts

- C Type IO functions in Octave
- Built-in Function: sprintf *(template, …)*

  This is like printf, except that the output is returned as a string. Unlike the C library function, which requires you to provide a suitably sized string as an argument, Octave's sprintf function returns the string, automatically sized to hold all of the items converted.

Octave scripts

- C Type IO functions in Octave

```
pct = 37; filename = "foo.txt";
printf ("Processed %d%% of '%s'.\nPlease be patient.\n“
                ,pct, filename);
```

produces output like:

```
Processed 37% of 'foo.txt'. Please be patient.
```

Octave scripts

- **C Type IO functions in Octave**

  Built-in Function:

  [val, count, errmsg]* = fscanf *(fid, template, size)

  Built-in Function:

  [v1, v2, …, count, errmsg]* = fscanf *(fid, template, "C")

  In the first form, read from *fid* according to *template*, returning the result in the matrix *val*.

Octave scripts

- **C Type IO functions in Octave**

  The optional argument *size* for fscanf specifies the amount of data to read and may be one of

  – **Inf** Read as much as possible, returning a column vector.

  – ***Nr*** Read up to *nr* elements, returning a column vector.

  – **[*nr*, Inf]** Read as much as possible, returning a matrix with *nr* rows. If the number of elements read is not an exact multiple of *nr*, the last column is padded with zeros.

  – **[*nr*, *nc*]** Read up to *nr* * *nc* elements, returning a matrix with *nr* rows. If the number of elements read is not an exact multiple of *nr*, the last column is padded with zeros.

TEHNIČKI FAKULTET
Sveučilište u Rijeci

Medicinski fakultet
Rijeka

Octave scripts

- C Type IO functions in Octave

Built-in Function:

[val, count]* = fread *(fid, size, precision, skip, arch)

- **Read binary data of type *precision* from the specified file ID *fid*.**
- **The optional argument *size* specifies the amount of data to read and may be one of**
- **Inf** Read as much as possible, returning a column vector.
- ***Nr*** Read up to *nr* elements, returning a column vector.
- **[*nr*, Inf]** Read as much as possible, returning a matrix with *nr* rows. If the number of elements read is not an exact multiple of *nr*, the last column is padded with zeros.
- **[*nr*, *nc*]** Read up to *nr* * *nc* elements, returning a matrix with *nr* rows. If the number of elements read is not an exact multiple of *nr*, the last column is padded with zeros.
- **If *size* is omitted, a value of Inf is assumed.**

TEHNIČKI FAKULTET
Sveučilište u Rijeci

Medicinski fakultet
Rijeka

Octave scripts

- C Type IO functions in Octave

Built-in Function:

count* = fwrite *(fid, data, precision, skip, arch)

- **Write data in binary form of type *precision* to the specified file ID *fid*, returning the number of values successfully written to the file.**
- **The argument *data* is a matrix of values that are to be written to the file. The values are extracted in column-major order.**
- **The remaining arguments *precision*, *skip*, and *arch* are optional, and are interpreted as described for fread.**
- **The behavior of fwrite is undefined if the values in *data* are too large to fit in the specified precision.**

TEHNIČKI FAKULTET
Sveučilište u Rijeci

Medicinski fakultet
Rijeka

Octave scripts

- Saving your work

  Sometimes it is a good idea to save the variable prime_sequence to avoid calculating the sequence again.

  Saving variables in Octave is easy. Simply use:

  octave:>> save primes.mat prime_sequence

  to save prime_sequence in a file called primes.mat. If you want to save more than one variable, you just write all the variable names after the file name.

83

TEHNIČKI FAKULTET
Sveučilište u Rijeci

Medicinski fakultet
Rijeka

Octave scripts

- Saving your work

  The general syntax is:

  save –option1 –option2 filename variable1 variable2 ...

  where **–option1 –option2** specifies the file format, **filename** is the name of the file (for example, **primes.mat**) and **variable1 variable2 ...** is the list of variables that you wish to save. You can use wildcards to save all variables with a specific pattern, for example, if **variable1** is given as primes*, all variables with prefix primes will be saved. If you do not specify any variables, all variables in the current workspace are saved.

84

Octave scripts

- Saving your work

  From your editor, try to open the file **primes.mat**. You will see something like this:

```
# Created by Octave 3.2.4, Sun Aug 29 12:30:20 2010 ...
# name: prime_sequence
# type: matrix
# rows: 1
# columns: 1385
2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73
79 83 89
```

Octave scripts

- Octave has added five lines beginning with a hash mark, #.This is referred to as the heading. Here various information is stored, like who created the file (in the case Octave, it could also be a username) and the name of the variable. If you load the file into Octave's workspace, the interpreter will go through the heading and create a variable called prime_sequence (overwriting any existing one) with the rows, columns, and values listed. Of course, the heading and therefore the file cannot be read by other programs unless they are specially designed to do so. If you save more than one variable, a header will be written for each one.

Octave scripts

- You can tell Octave to change the output format such that it can be read by other programs. This is specified via the options listed below
- Option Description

  -text Saves the variables in readable text format with information about the variables (names, dimensions, and so on.) Also, Octave prints a small file header about who created the file and when. This option is set as default.

Octave scripts

- Option Description

  -ascii Saves the variables in ASCII format. This format will not include variable information. This is not recommended if you save more than one variable and wish to load them into Octave at a later stage. This is useful when data is read by other programs.

  -binary Saves the variables in Octave's own binary format. This could speed up things.

TEHNIČKI FAKULTET
Sveučilište u Rijeci

Medicinski fakultet
Rijeka

Octave scripts

- Option Description

  -hdf5 Portable binary format.

  -vx or –Vx Saves the variables in MATLAB format. Currently, x can have values 4, 6, or 7 and indicates the MATLAB version number.

  -zip or -z Compressed output format (for saving hard disk space). This option can be used together with any format option above.

TEHNIČKI FAKULTET
Sveučilište u Rijeci

Medicinski fakultet
Rijeka

Octave scripts

- For example, to save primes_sequence in simple ascii format, use:

```
octave:>> save –ascii primes.dat prime_sequence
```

  Take a look at the output file. You can, of course, type help save to see all the available options.

Octave scripts

- Loading your work

To load the variable(s) stored in a file, first clear the workspace to be sure to actually load the variable prime_sequence stored in the file primes.mat:

```
octave:>> clear; whos
octave:>> load primes.mat
octave:>> whos
Variables in the current scope:
Attr Name size Bytes Class
==== ==== ==== ===== =====
prime_sequence 1x1385 11080 double
```

Notice that Octave treats the numbers as doubles, since we have not explicitly told it otherwise.

Octave scripts

- Loading your work.
- The general syntax for load is:

```
load –option1 –option2 filename
```

where the options are the same as for the save command. For example, to load the data stored in the ascii file primes.dat, we can use:

```
octave:49> load –ascii primes.dat
octave:50> whos
Variables in the current scope:
Attr Name size Bytes Class
==== ==== ==== ===== =====
prime_sequence 1x1385 11080 double
primes 1x1385 11080 double
```

Octave scripts

- Loading your work.

  Notice that when loading an ascii file like we did above, Octave will create a variable called primes that contains the prime sequence. It is infortunate that we chose to load data into a variable called prime in this example, since this stops you from usin the built-in function of the same name.

  In general, if the file does not contain a heading, Octave will create a variable having the name of the data file, excluding the extension, overwriting any existing variable or function with that name.

Octave scripts

- Functional forms

  To avoid the problem of overwriting existing variables and function names, you can use the functional form of load. To load data stored in an ASCII file named **primes.**

  dat into a variable, say prime_sequence you can use:

  ```
  octave:>>prime_sequence = load("primes.dat", "ascii");
  ```

TEHNIČKI FAKULTET
Sveučilište u Rijeci

Medicinski fakultet
Rijeka

Octave scripts

- Functional forms

  octave:>>prime_sequence = load("primes.dat", "ascii");

  This will also work if you have saved the data in other formats, if and only if the data file contains a single simple variable and not a structure or cell array. It is recommend to use the default text format when you save and load your data files, unless there is a specific reason not to.

  save also has a corresponding functional form, for example, command above could be replaced with:

  octave:40> save("prime.mat", "prime_sequence");

TEHNIČKI FAKULTET
Sveučilište u Rijeci

Medicinski fakultet
Rijeka

Octave scripts

- Summary
- Write a simple script and execute it.
- Use the control statements if and switch.
- Use for, while, and dostatements.
- Put everything together in order to write a script with complicated program flow.
- Save and load our work using the save and load commands.
- Use the printf function.