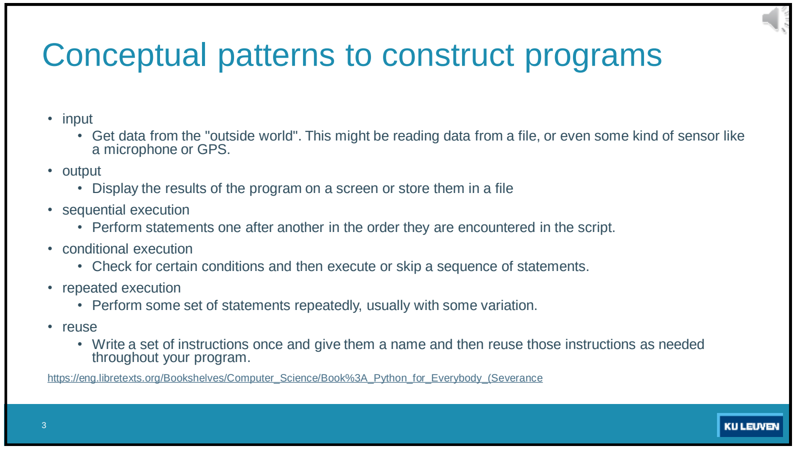
KU LEUVEN

MATLAB

flow control

1



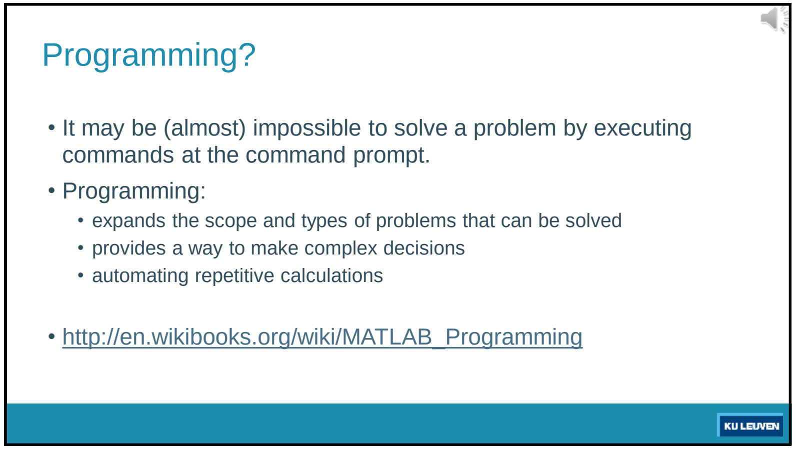
Conceptual patterns to construct programs

- input
  - Get data from the "outside world". This might be reading data from a file, or even some kind of sensor like a microphone or GPS.
- output
  - Display the results of the program on a screen or store them in a file
- sequential execution
  - Perform statements one after another in the order they are encountered in the script.
- conditional execution
  - Check for certain conditions and then execute or skip a sequence of statements.
- repeated execution
  - Perform some set of statements repeatedly, usually with some variation.
- reuse
  - Write a set of instructions once and give them a name and then reuse those instructions as needed throughout your program.

https://eng.libretexts.org/Bookshelves/Computer_Science/Book%3A_Python_for_Everybody_(Severance

3 KU LEUVEN

3

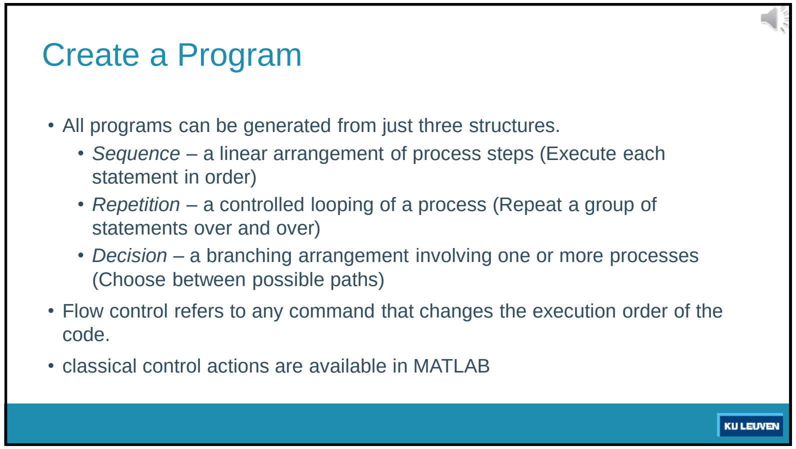


Programming?

- It may be (almost) impossible to solve a problem by executing commands at the command prompt.
- Programming:
  - expands the scope and types of problems that can be solved
  - provides a way to make complex decisions
  - automating repetitive calculations
- http://en.wikibooks.org/wiki/MATLAB_Programming

KU LEUVEN

5

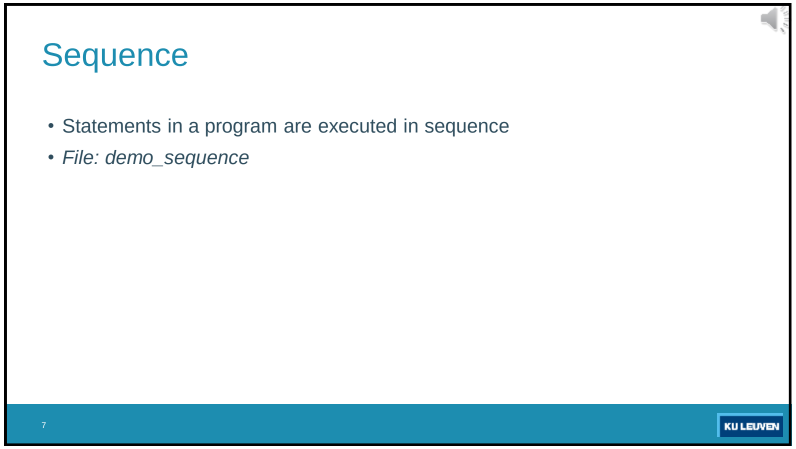


Create a Program

- All programs can be generated from just three structures.
  - *Sequence* – a linear arrangement of process steps (Execute each statement in order)
  - *Repetition* – a controlled looping of a process (Repeat a group of statements over and over)
  - *Decision* – a branching arrangement involving one or more processes (Choose between possible paths)
- Flow control refers to any command that changes the execution order of the code.
- classical control actions are available in MATLAB

KU LEUVEN

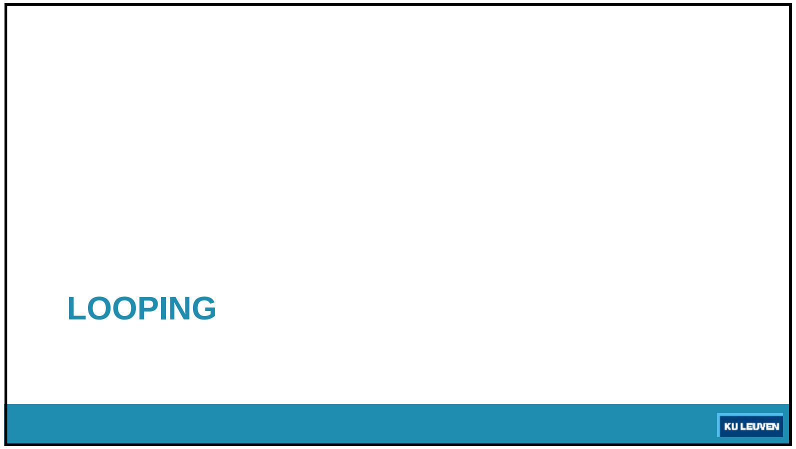
6



Sequence

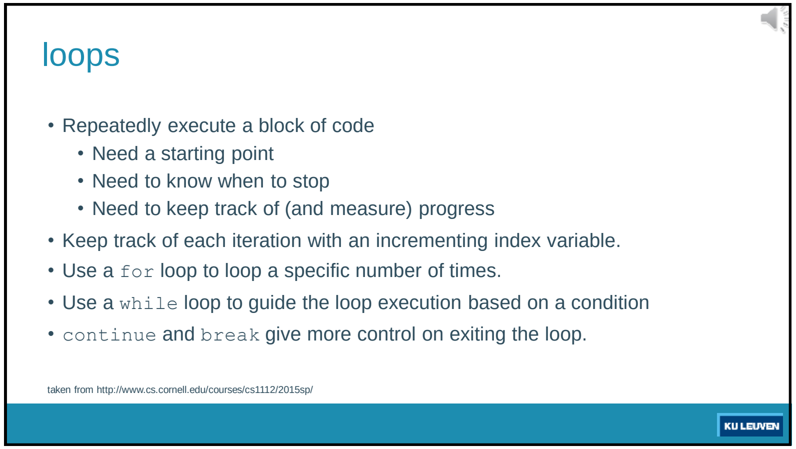
- Statements in a program are executed in sequence
- *File: demo_sequence*

7



LOOPING

8



loops

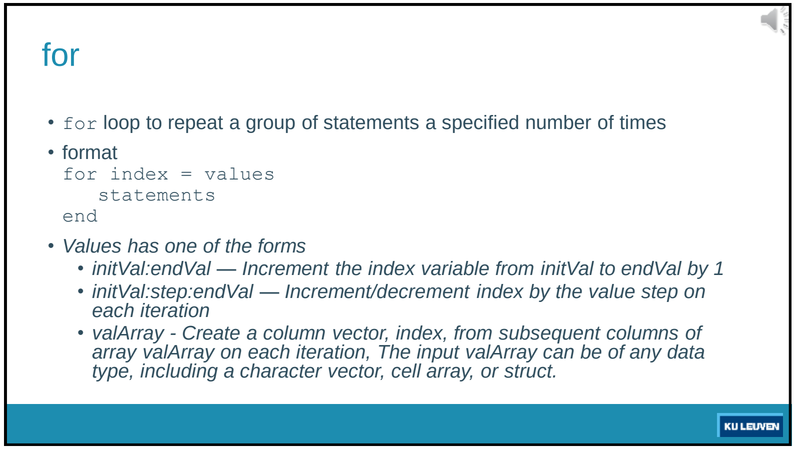
- Repeatedly execute a block of code
  - Need a starting point
  - Need to know when to stop
  - Need to keep track of (and measure) progress
- Keep track of each iteration with an incrementing index variable.
- Use a `for` loop to loop a specific number of times.
- Use a `while` loop to guide the loop execution based on a condition
- `continue` and `break` give more control on exiting the loop.

taken from http://www.cs.cornell.edu/courses/cs1112/2015sp/

9

- Compare:
  - sum_bruteforce.m
  - sum_for_loop.m
  - sum_while_loop.m

10



for

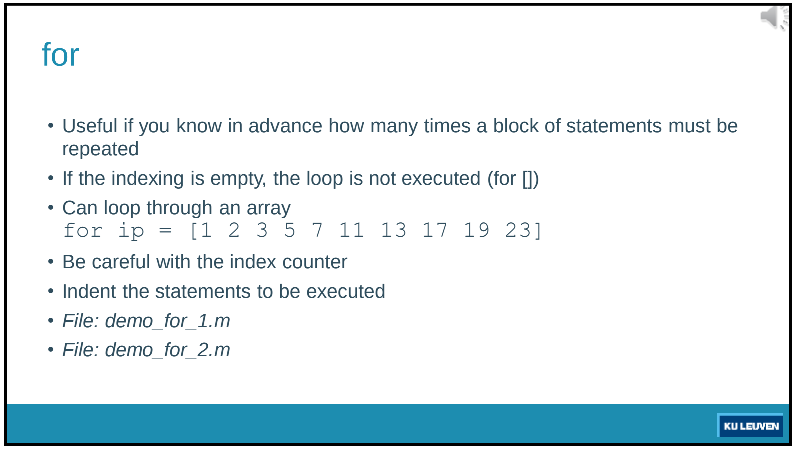
- `for` loop to repeat a group of statements a specified number of times
- format

```
for index = values
     statements
end
```

- *Values has one of the forms*
  - *initVal:endVal — Increment the index variable from initVal to endVal by 1*
  - *initVal:step:endVal — Increment/decrement index by the value step on each iteration*
  - *valArray - Create a column vector, index, from subsequent columns of array valArray on each iteration, The input valArray can be of any data type, including a character vector, cell array, or struct.*

KU LEUVEN

11



for

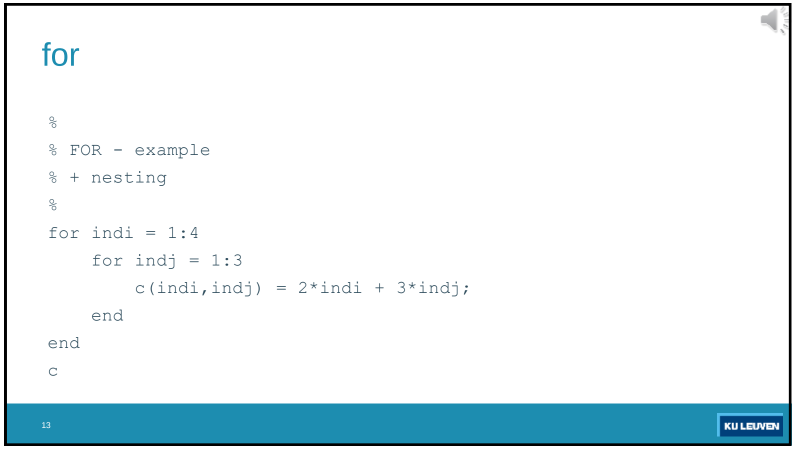
- Useful if you know in advance how many times a block of statements must be repeated
- If the indexing is empty, the loop is not executed (for [])
- Can loop through an array

```
for ip = [1 2 3 5 7 11 13 17 19 23]
```

- Be careful with the index counter
- Indent the statements to be executed
- *File: demo_for_1.m*
- *File: demo_for_2.m*

KU LEUVEN

12



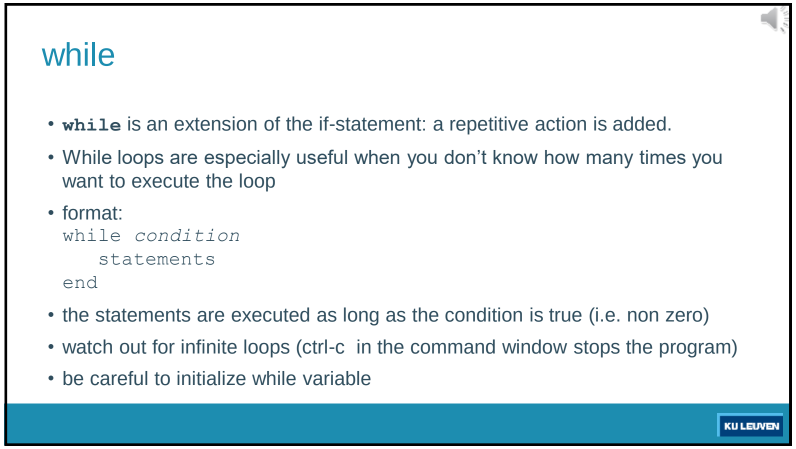
for

```
%
% FOR - example
% + nesting
%
for indi = 1:4
    for indj = 1:3
        c(indi,indj) = 2*indi + 3*indj;
    end
end
c
```

13

KU LEUVEN

13



while

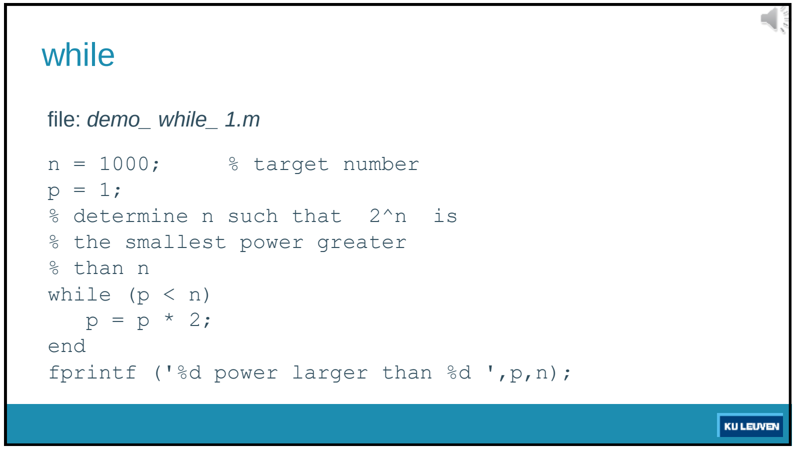
- **`while`** is an extension of the if-statement: a repetitive action is added.
- While loops are especially useful when you don’t know how many times you want to execute the loop
- format:

```
while condition
    statements
end
```

- the statements are executed as long as the condition is true (i.e. non zero)
- watch out for infinite loops (ctrl-c in the command window stops the program)
- be careful to initialize while variable

KU LEUVEN

14



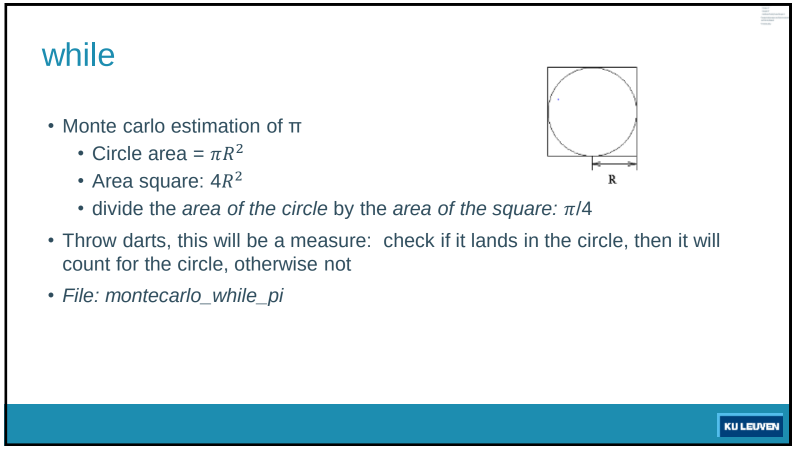
while

file: *demo_ while_ 1.m*

```
n = 1000;     % target number
p = 1;
% determine n such that  2^n  is
% the smallest power greater
% than n
while (p < n)
   p = p * 2;
end
fprintf ('%d power larger than %d ',p,n);
```

KU LEUVEN

15

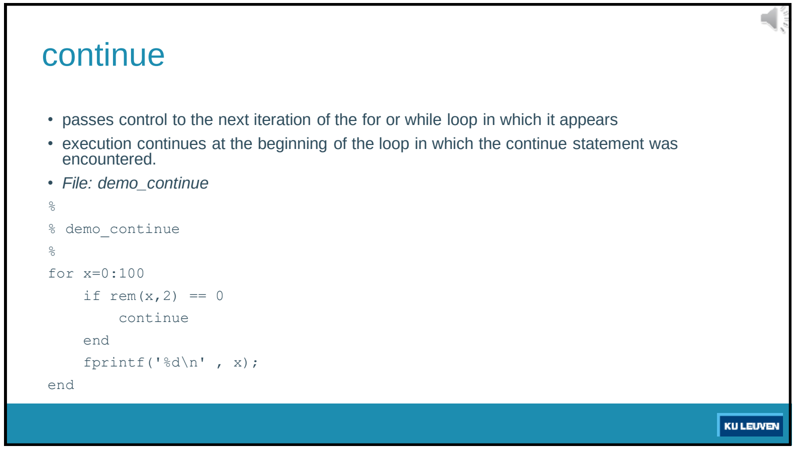


while

- Monte carlo estimation of π
  - Circle area = πR^2
  - Area square: $4R^2$
  - divide the *area of the circle* by the *area of the square:* $\pi/4$
- Throw darts, this will be a measure: check if it lands in the circle, then it will count for the circle, otherwise not
- *File: montecarlo_while_pi*

KU LEUVEN

16



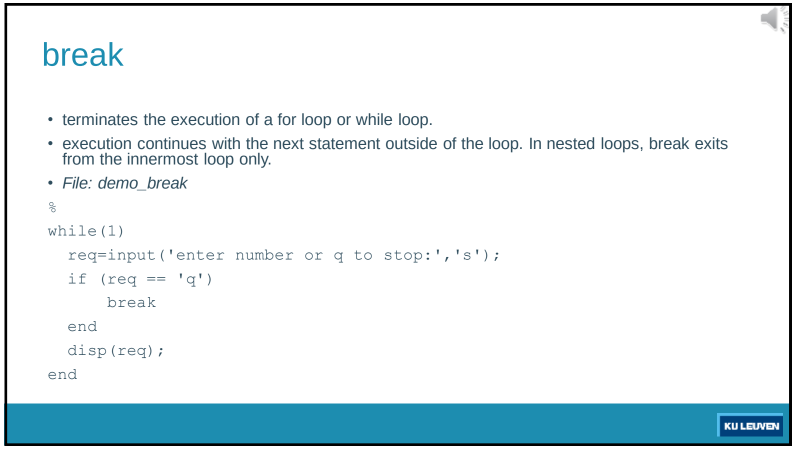
continue

- passes control to the next iteration of the for or while loop in which it appears
- execution continues at the beginning of the loop in which the continue statement was encountered.
- *File: demo_continue*

```
%
% demo_continue
%
for x=0:100
    if rem(x,2) == 0
        continue
    end
    fprintf('%d\n' , x);
end
```

KU LEUVEN

17



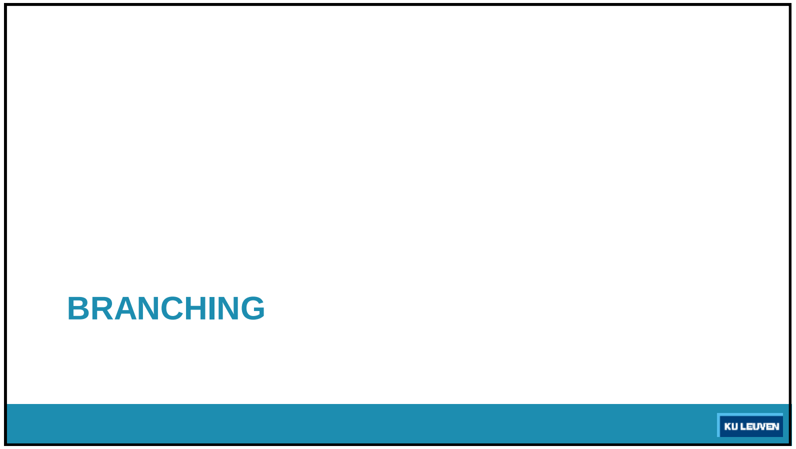
break

- terminates the execution of a for loop or while loop.
- execution continues with the next statement outside of the loop. In nested loops, break exits from the innermost loop only.
- *File: demo_break*

```
%
while(1)
  req=input('enter number or q to stop:','s');
  if (req == 'q')
      break
  end
  disp(req);
end
```

KU LEUVEN

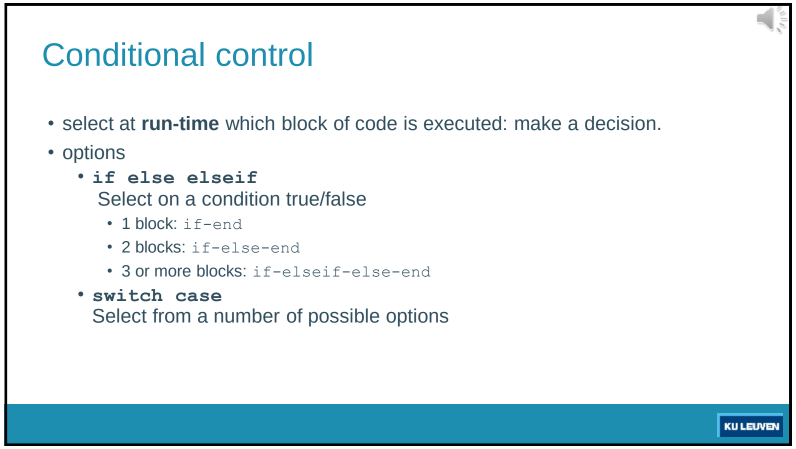
18



BRANCHING

KU LEUVEN

19

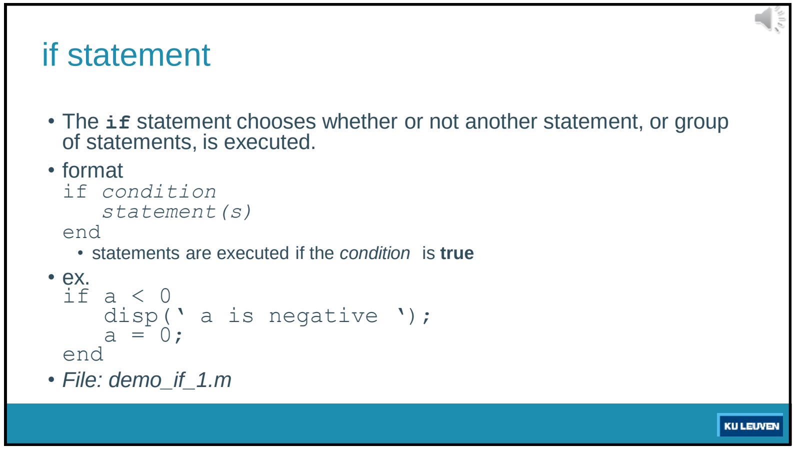


Conditional control

- select at **run-time** which block of code is executed: make a decision.
- options
  - **`if else elseif`**
    Select on a condition true/false
    - 1 block: `if-end`
    - 2 blocks: `if-else-end`
    - 3 or more blocks: `if-elseif-else-end`
  - **`switch case`**
    Select from a number of possible options

KU LEUVEN

20



if statement

- The **`if`** statement chooses whether or not another statement, or group of statements, is executed.
- format

```
if condition
   statement(s)
end
```

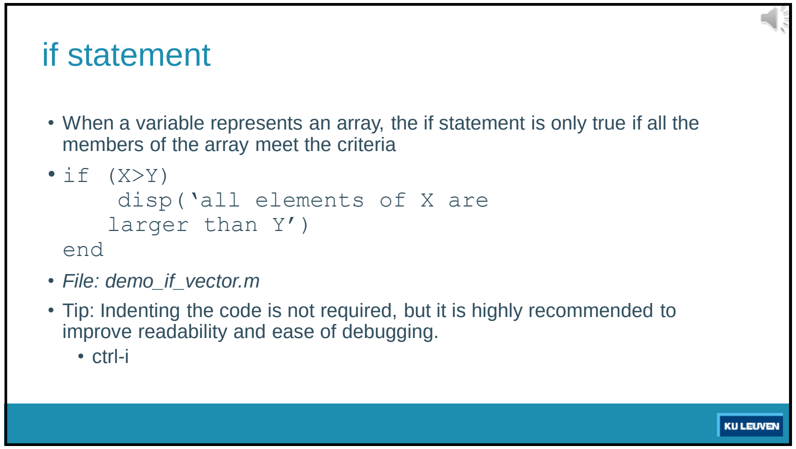
  - statements are executed if the *condition* is **true**
- ex.

```
if a < 0
   disp(‘ a is negative ’);
   a = 0;
end
```

- *File: demo_if_1.m*

KU LEUVEN

21



if statement

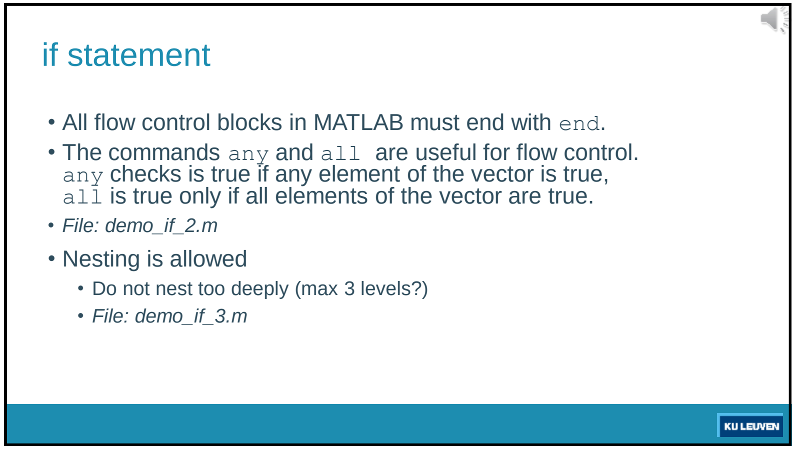
- When a variable represents an array, the if statement is only true if all the members of the array meet the criteria
-

```
if (X>Y)
     disp(‘all elements of X are
    larger than Y’)
end
```

- *File: demo_if_vector.m*
- Tip: Indenting the code is not required, but it is highly recommended to improve readability and ease of debugging.
  - ctrl-i

KU LEUVEN

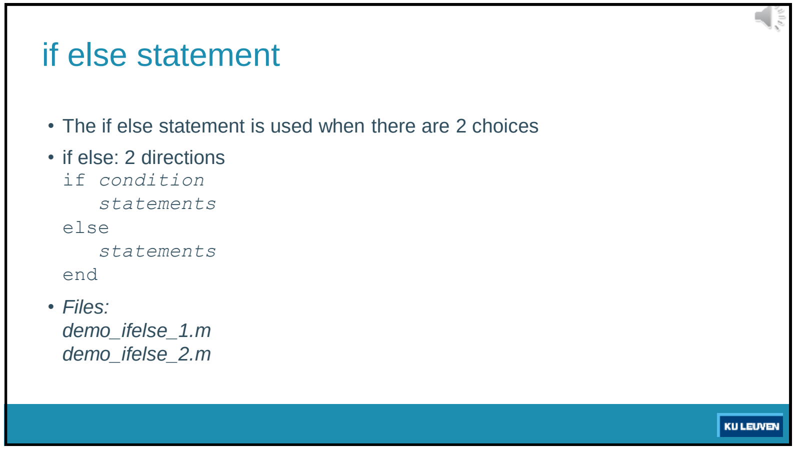
22



if statement

- All flow control blocks in MATLAB must end with `end`.
- The commands `any` and `all` are useful for flow control. `any` checks is true if any element of the vector is true, `all` is true only if all elements of the vector are true.
- *File: demo_if_2.m*
- Nesting is allowed
  - Do not nest too deeply (max 3 levels?)
  - *File: demo_if_3.m*

23



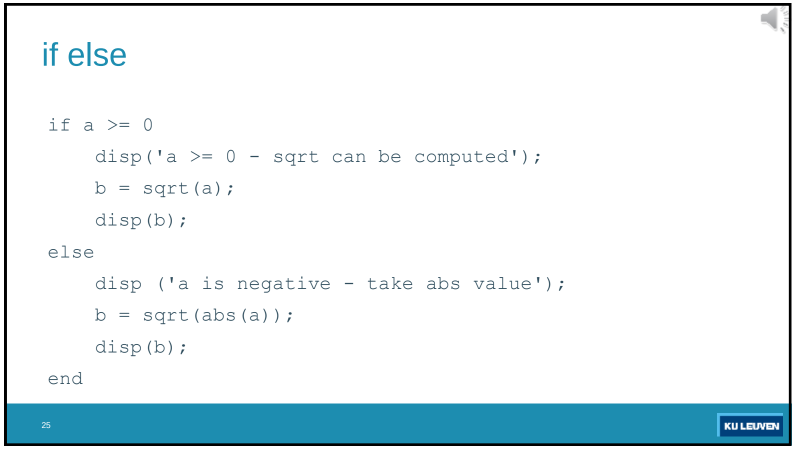
if else statement

- The if else statement is used when there are 2 choices
- if else: 2 directions

```
if condition
   statements
else
   statements
end
```

- *Files:*
  demo_ifelse_1.m
  demo_ifelse_2.m

24

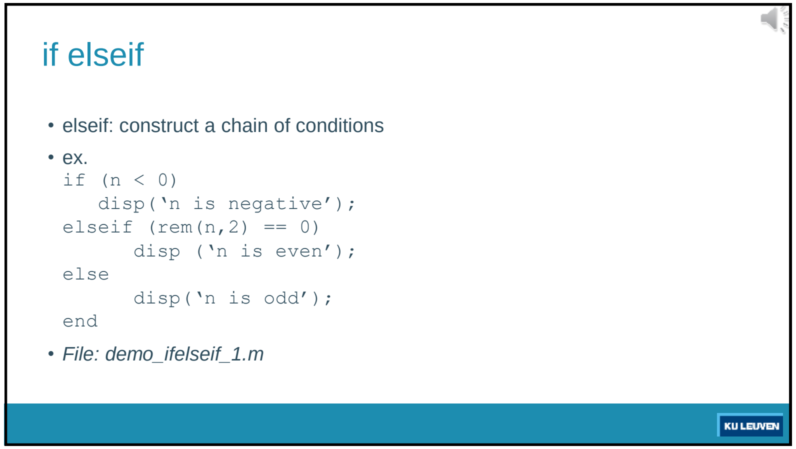


if else

```
if a >= 0
    disp('a >= 0 - sqrt can be computed');
    b = sqrt(a);
    disp(b);
else
    disp ('a is negative - take abs value');
    b = sqrt(abs(a));
    disp(b);
end
```

25

25



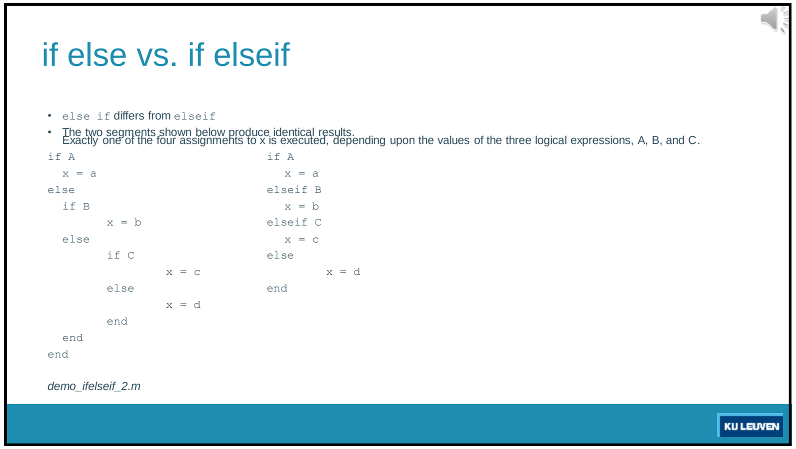
if elseif

- elseif: construct a chain of conditions
- ex.

```
if (n < 0)
   disp(‘n is negative’);
elseif (rem(n,2) == 0)
      disp (‘n is even’);
else
      disp(‘n is odd’);
end
```

- *File: demo_ifelseif_1.m*

27



if else vs. if elseif

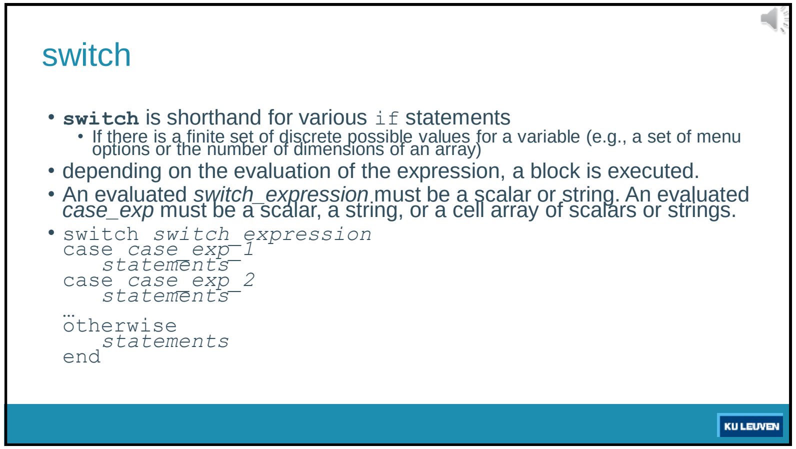
- `else if` differs from `elseif`
- The two segments shown below produce identical results. Exactly one of the four assignments to x is executed, depending upon the values of the three logical expressions, A, B, and C.

```
if A                           if A
  x = a                           x = a
else                           elseif B
  if B                            x = b
        x = b                  elseif C
  else                            x = c
        if C                   else
                 x = c                  x = d
        else                   end
                 x = d
        end
  end
end
```

demo_ifelseif_2.m

KU LEUVEN

28

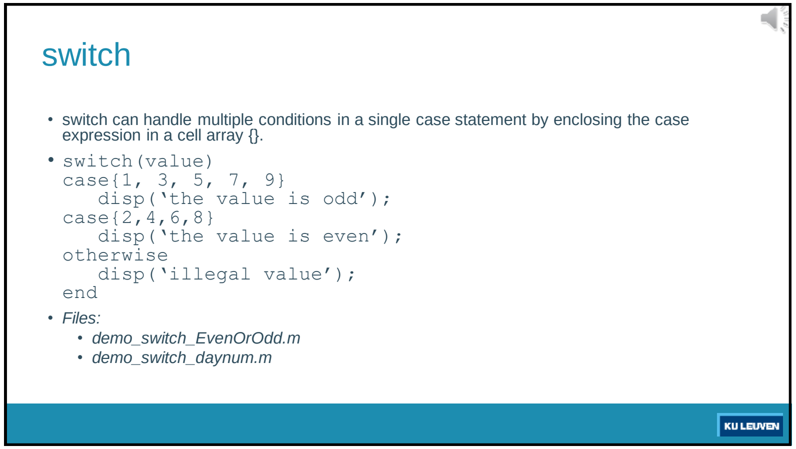


switch

- **`switch`** is shorthand for various `if` statements
  - If there is a finite set of discrete possible values for a variable (e.g., a set of menu options or the number of dimensions of an array)
- depending on the evaluation of the expression, a block is executed.
- An evaluated *switch_expression* must be a scalar or string. An evaluated *case_exp* must be a scalar, a string, or a cell array of scalars or strings.
-
```
switch switch_expression
case case_exp_1
    statements
case case_exp_2
    statements
…
otherwise
    statements
end
```

KU LEUVEN

29

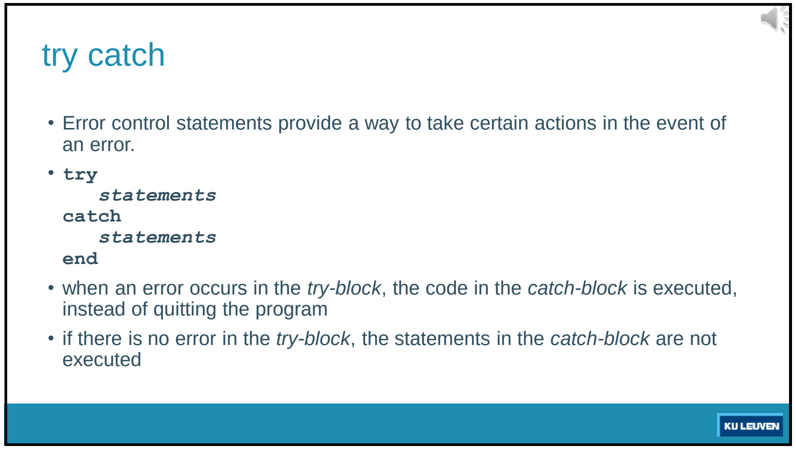


switch

- switch can handle multiple conditions in a single case statement by enclosing the case expression in a cell array {}.
-
```
switch(value)
case{1, 3, 5, 7, 9}
   disp(‘the value is odd’);
case{2,4,6,8}
   disp(‘the value is even’);
otherwise
   disp(‘illegal value’);
end
```
- *Files:*
  - *demo_switch_EvenOrOdd.m*
  - *demo_switch_daynum.m*

KU LEUVEN

31

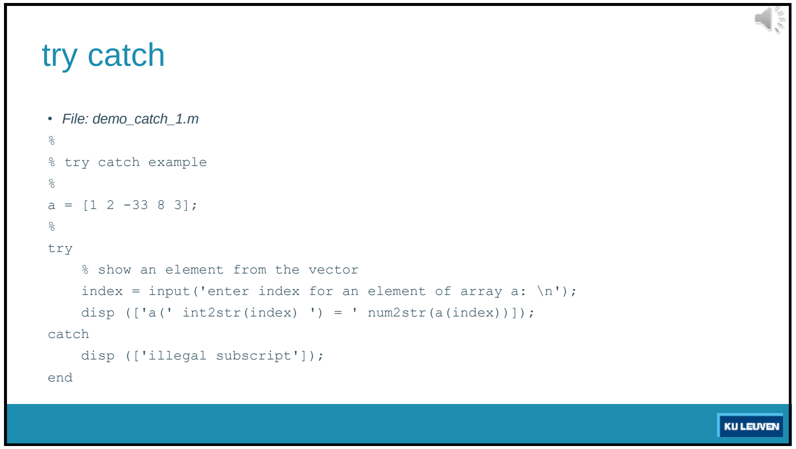


try catch

- Error control statements provide a way to take certain actions in the event of an error.
-
```
try
   statements
catch
   statements
end
```
- when an error occurs in the *try-block*, the code in the *catch-block* is executed, instead of quitting the program
- if there is no error in the *try-block*, the statements in the *catch-block* are not executed

KU LEUVEN

33

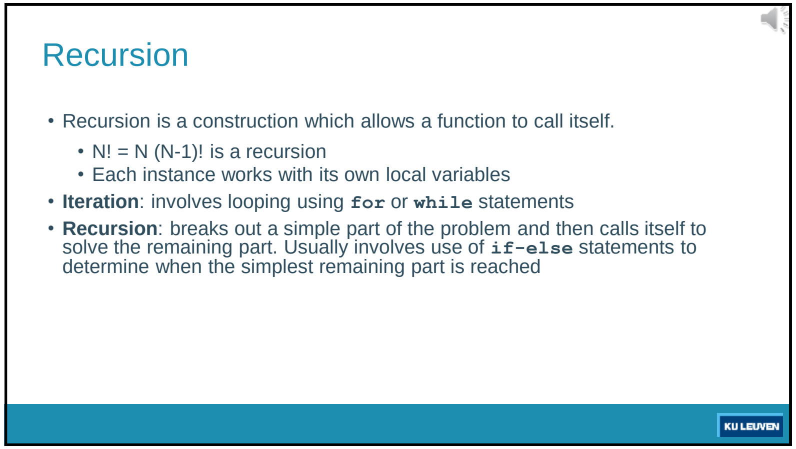


try catch

- *File: demo_catch_1.m*

```
%
% try catch example
%
a = [1 2 -33 8 3];
%
try
    % show an element from the vector
    index = input('enter index for an element of array a: \n');
    disp (['a(' int2str(index) ') = ' num2str(a(index))]);
catch
    disp (['illegal subscript']);
end
```

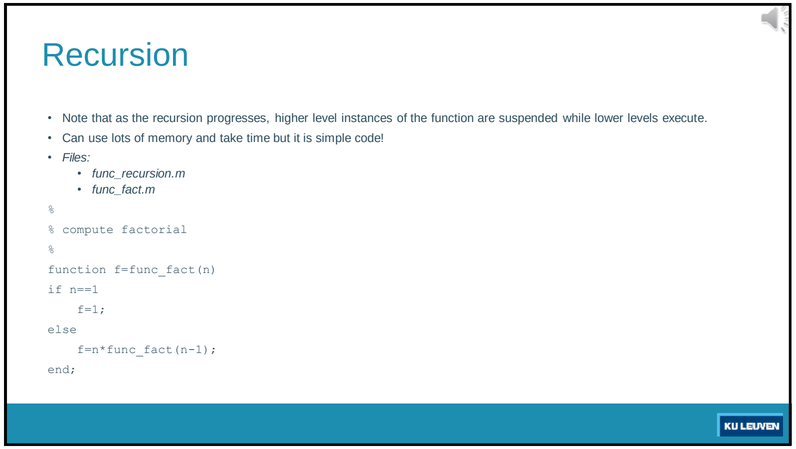
34



Recursion

- Recursion is a construction which allows a function to call itself.
  - N! = N (N-1)! is a recursion
  - Each instance works with its own local variables
- **Iteration**: involves looping using `for` or `while` statements
- **Recursion**: breaks out a simple part of the problem and then calls itself to solve the remaining part. Usually involves use of `if-else` statements to determine when the simplest remaining part is reached

36



Recursion

- Note that as the recursion progresses, higher level instances of the function are suspended while lower levels execute.
- Can use lots of memory and take time but it is simple code!
- *Files:*
  - *func_recursion.m*
  - *func_fact.m*

```
%
% compute factorial
%
function f=func_fact(n)
if n==1
    f=1;
else
    f=n*func_fact(n-1);
end;
```

37