# MA2252 Introduction to Computing

## Lecture 8

## Part 1: Iteration

Sharad Kumar Keshari

School of Computing and Mathematical Sciences

University of Leicester

# Learning outcomes

At the end of lecture, students will be able to understand and create

- For-Loop
- While-Loop

# Iteration

Iteration means to perform a task repeatedly.

**Example:**

To find the sum of first n natural numbers, addition of two numbers taken at a time should be done repeatedly.

# Iteration (contd.)

In MATLAB, iteration can be done via

1. For-loop
2. While-loop

# For-Loop

**Construction:**

```
for looping variable = looping array
   code block
end
```

**Function:** For-loop executes the code block for values of looping variable from first to last element of looping array.

# For-Loop (contd.)

**Example 1:**

Find the sum of first n natural numbers.

```
clc
clear all
sum=1;
N=10;
for n=2:N
   sum=sum+n;
end
```

# For-Loop (contd.)

**Example 2:**

Find the following sum:

$$1x2 + 2x3 + 3x4 + \cdots + 10x11$$

```
clc
clear all
product=0;
for n=1:10
   product = product + n*(n+1);
end
```

# For-Loop (contd.)

Demo

# For-Loop (contd.)

## Nested For-Loops

A for-loop entirely contained in other for-loop is called a nested for-loop.

# For-Loop (contd.)

**Example:** Create a 4x4 identity matrix using nested for-loops.

```
N=4;
A=zeros(N);
for i=1:N
   for j=1:N
      if i==j
         A(i,j)=1;
      else
         A(i,j)=0;
      end
   end
end

disp(A)
```

# For-Loop (contd.)

Demo

# For-Loop (contd.)

**Using break and continue keywords**

- break is used to exit the for-loop.
- continue skips the rest of for-loop's code and begins the next iteration.

**Note:** In nested for-loops, break only exits the inner-most for-loop in which it is contained.

# For-Loop (contd.)

**Example:**

Write a function to test if a given number (greater than 2) is prime or not.

```
function s = test_prime(x)
for i = 2:x-1
   if mod(x,i)==0
      s=sprintf('%d is not prime',x);
      break
   elseif i==x-1
      s=sprintf('%d is prime',x);
   else
      continue
   end
end
```

# For-Loop (contd.)

Demo

# While-Loop

A while loop executes the code as long as a given logical expression is true.

**Construction:**

```
while logical expression
code block
end
```

# While-Loop (contd.)

**Example:**

Find all square numbers less than 50.

```
clc
clear all
x=1;
while x^2<50
   disp(x^2)
   x=x+1;
end
```

# While-Loop (contd.)

Demo

# Activity

A student wrote an alternative code to find all square numbers less than 50. What is happening with this code?

```
clc
clear all
x=1;
y=x^2;
while y<50
   disp(y)
   x=x+1;
end
```

Open your MATLAB, write this code and interpret the output.

# Infinite Loop

An infinite loop runs forever.

**Note:** In this scenario, use ctrl+c to stop the code execution by MATLAB.

# End of Part 1

Please provide your feedback here

# MA2252 Introduction to Computing

## Lecture 8

## Part 2: Recursion

Sharad Kumar Keshari

School of Computing and Mathematical Sciences

University of Leicester

# Learning outcomes

At the end of lecture, students will be able to

- understand recursion
- create recursive functions
- understand the difference between recursion and iteration
- solve recursion problems e.g. games

# Recursion

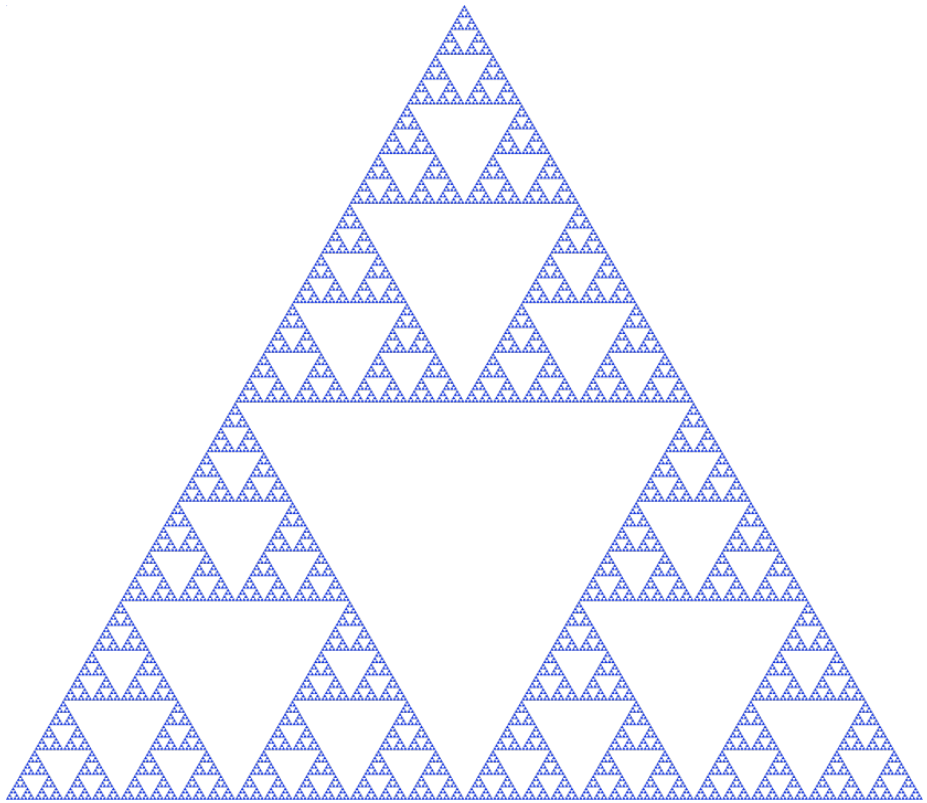
Recursion occurs when something is defined in terms of itself.

**Examples:**

- Romanesco broccoli[1]

# Recursion (contd.)

- Sierpiński triangle [2]



[1] Picture credit: Ivar Leidus
[2] Picture credit: Beojan Stanislaus

# Recursive functions

A recursive function is defined in terms of itself.

**Examples:**

- Factorial function

$$n! = \begin{cases} 1, & n = 0 \\ n(n-1)!, & n \in \mathbb{N}. \end{cases}$$

- 

$$f(n) = \begin{cases} \sqrt{2}, & n = 1 \\ \sqrt{2 + f(n-1)}, & n > 1. \end{cases}$$

# Recursive functions (contd.)

The definition of a recursive function includes

- **Base case:** Function's value is given or can be calculated without using recursion.
- **Recursive step:** Function's value is calculated by calling to itself.

For example, in factorial function's definition, the cases n=0 and n>0 are base case and recursive step respectively.

# Recursive functions (contd.)

**Example:** Write a recursive function in MATLAB to find n!

```
function out = myfactorial(n)
if n==0
out=1;
else
out=n*myfactorial(n-1);
end
end
```

# Recursive functions (contd.)

Demo

# Recursive functions (contd.)

The factorial function can also be calculated using while loop. Observe how tricky it is to code now!

```
function out = myfactorial_while(n)
if n==0||n==1
   out=1;
else
   fact=n;
   while n>1
      fact=fact*(n-1);
      n=n-1;
   end
   out=fact;
end
end
```

# Recursion vs Iteration

| **Recursion** | **Iteration** |
|---|---|
| Easier to code recursive functions | Harder to code recursive functions |
| creates extra workspace | uses only one workspace |
| consumes more memory | consumes less memory |
| code runs rather slow | code runs faster |

# Tower of Hanoi

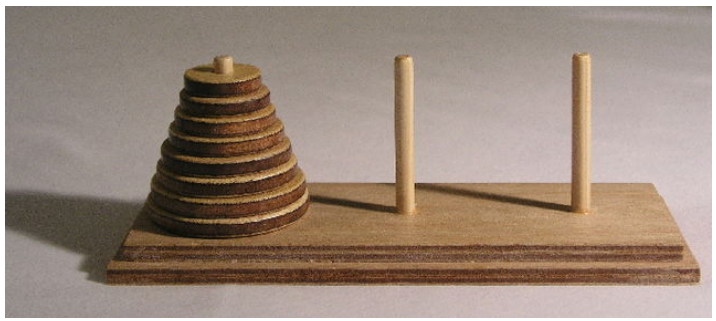


Figure: Tower of Hanoi game with 3 pegs [3]

**Goal:** To move an entire stack of disks from one peg to another.

**Rules:**

- Only one disk can be used at a time.
- Only top disk of a stack can be moved to an empty peg or top of other stack.
- Larger diameter disk cannot go on top of smaller diameter disk.

# Recursion in Games (contd.)

**Question:**
What is the minimum number of moves required to move a stack of n disks to other peg?

**Possible strategy:**
Exploit the recursive property of Tower of Hanoi game.

# Recursion in Games (contd.)

The recursive definition of minimum number of moves function $f(n)$ is given as

$$f(n) = \begin{cases} 1, & n = 1 \\ 2 * f(n-1) + 1, & n > 1. \end{cases}$$

# Recursion in Games (contd.)

Write a recursive MATLAB function to find the minimum number of moves needed to play Tower of Hanoi game with n disks and 3 pegs.

```
function moves = tower_of_hanoi(n)
if n==1
   moves=1;
else
   moves=2*tower_of_hanoi(n-1)+1;
end
end
```

# Recursion in Games (contd.)

Demo

## Beyond 3 pegs?

**Frame-Stewart** algorithm can be used to find minimum number of moves. The proof of optimality of this algorithm is still an unsolved problem in Mathematics.

[3] Picture credit: Evanherk

# End of Part 2

Please provide your feedback here