MATLAB SESSION 2

INDEX

1) INTRODUCTION

INTRODUCTION

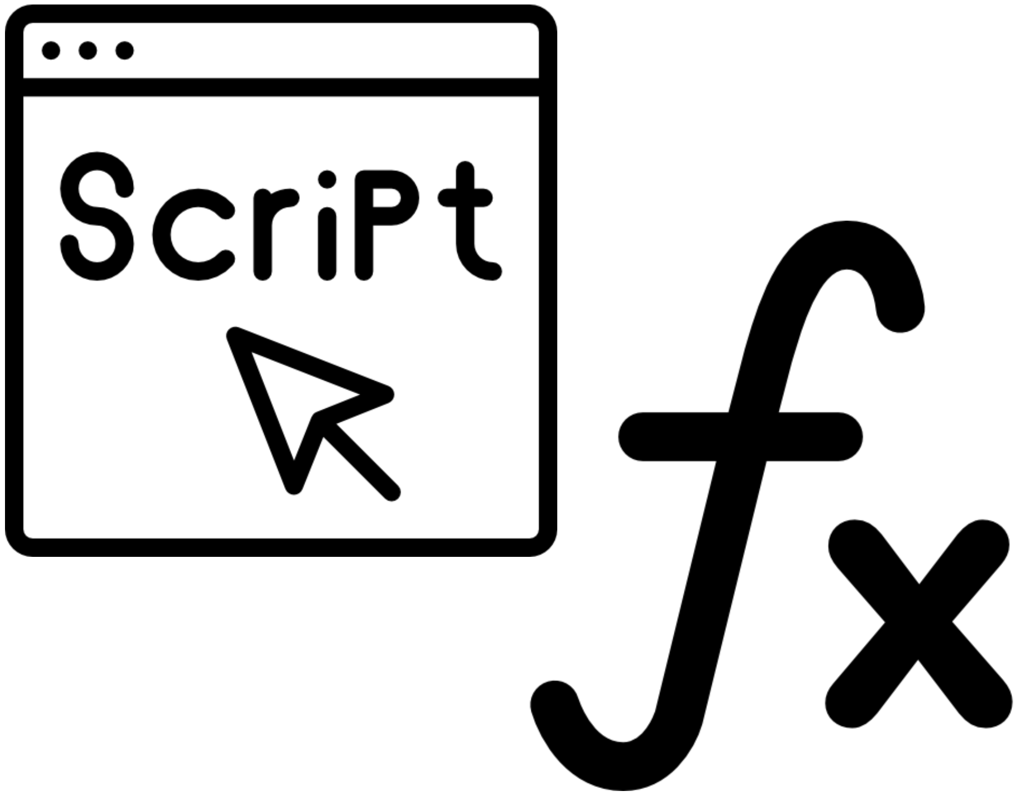
- MATLAB IS A **PROGRAMMING LANGUAGE**

- THERE ARE 2 TYPES OF **.m FILES**:

  - **SCRIPTS** = NO INPUT
  - **FUNCTIONS** = INPUT

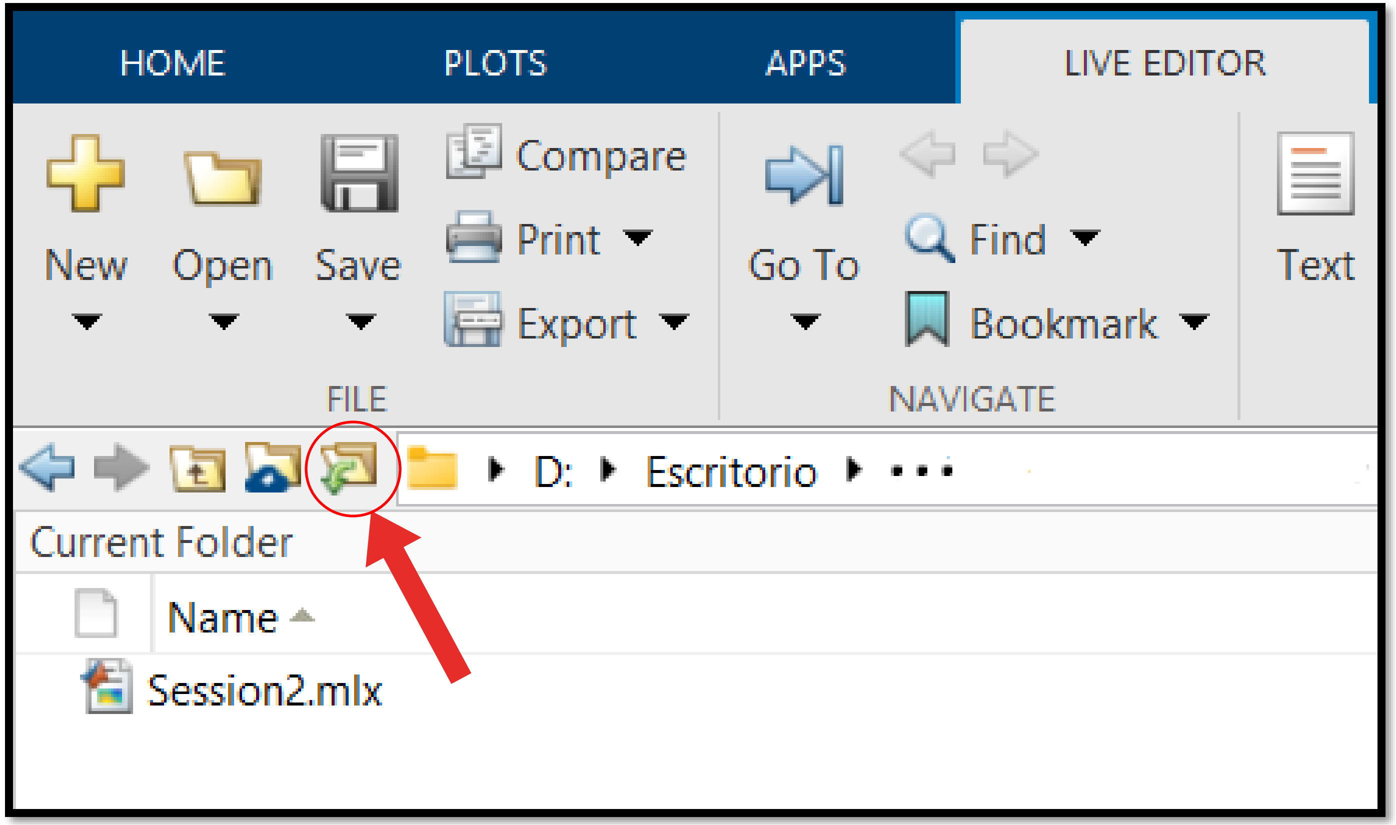
- **IMPORTANT**:

  - SAVE .M FILES IN THE **SAME FOLDER**
  - THE **NAME** OF THE FILE MUST **START WITH A LETTER**
  - MATLAB IS **CASE-SENSITIVE**



EXERCISE 1

- **CREATE A FOLDER** IN WHICH YOU WILL SAVE YOUR FILES.



2) SCRIPTS

SCRIPTS

- **CREATE A NEW SCRIPT**
- ASSIGN THE VALUES √**2/2** AND **-**√**2/2** TO VARIABLES:

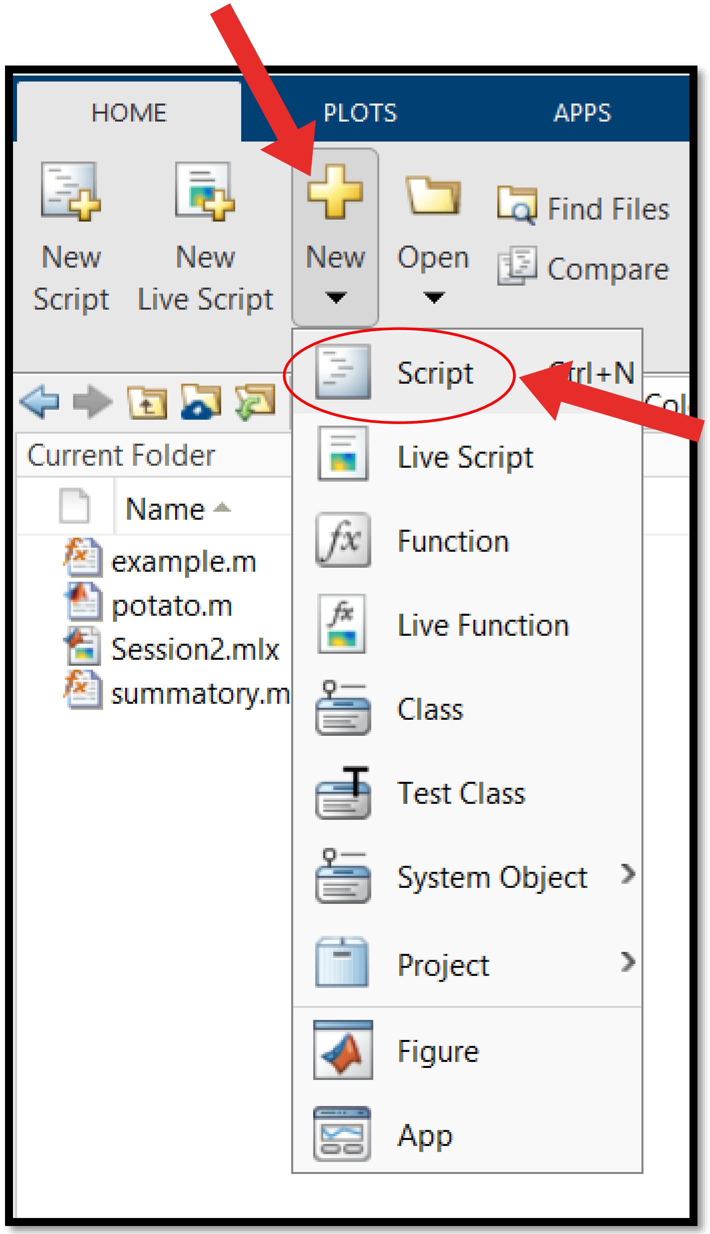
  - √2/2
  - \- √2/2

```
r1=sqrt(2)/2;
r2=-r1;
```

- THEN, **SAVE** IT AS "**potato.m**"

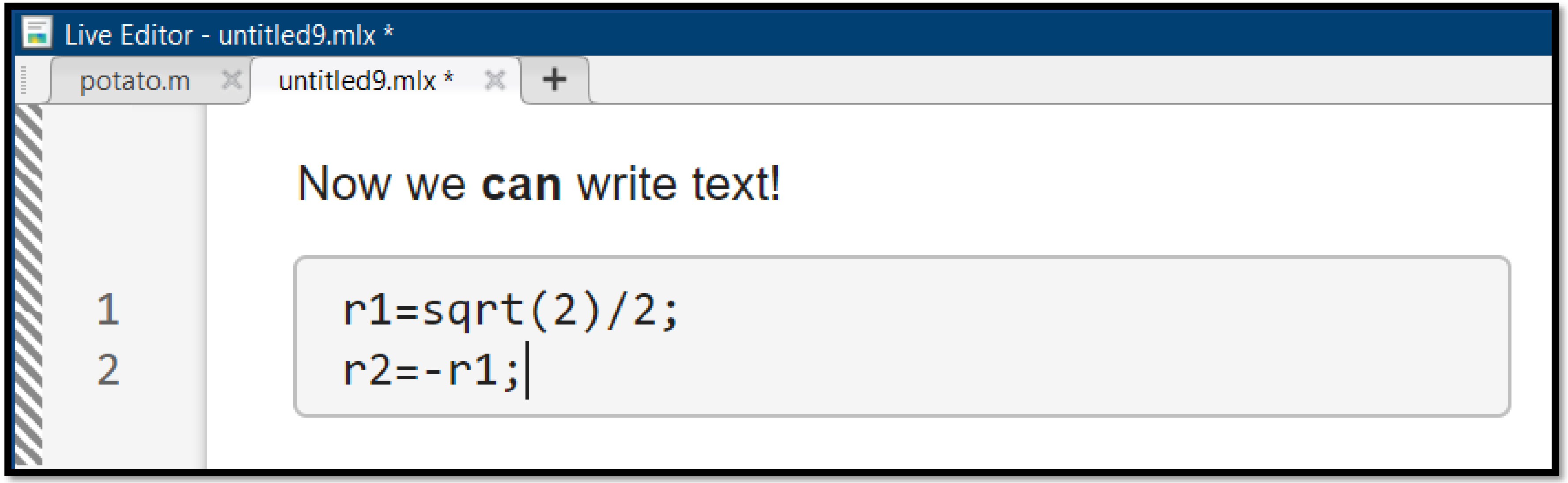
- NOW, **TO RUN THIS SCRIPT**, WE ONLY NEED TO WRITE "**potato**" IN THE COMMAND WINDOW

```
>> potato
```



SCRIPTS

- WE CAN USE THE **LIVE EDITOR** TO CREATE SCRIPTS:
    - THE **EXTENSION** WILL CHANGE TO "**.mlx**"
    - THIS NEW ENVIRONMENT IS MORE APPEALING



3) FOR

FOR

- USED TO **REPEAT** STATEMENTS
- VERY USEFUL FOR **SUMMATORIES**
- **SYNTAX OF A FOR:**

```
for i=1:n
sentences
end
```

Command Window

```
>> s=0;
>> for k=1:10
s=s+k;
end
>> s

s =

    55

>>
```

EXAMPLE:

CALCULATE THE SUM OF THE FIRST 10 NATURAL NUMBERS

EXERCISE 2

- **CREATE 2 SCRIPT** FILES:
  - ONE FOR COMPUTING:
    $$\boxed{\sum_{k=1}^{10} \frac{1}{2^k}}$$
  - ONE FOR COMPUTING:
    $$\boxed{\sum_{k=1}^{100} \frac{1}{2^k}}$$
- USE "**FORMAT LONG**" TO VISUALIZE THE RESULTS

EXERCISE 2 · SOLVED

$$\sum_{k=1}^{10} \frac{1}{2^k}$$

```
s=0

for k=1:10
    s = s + (1/(2^k));
end

format long
s
```

```
s =
   0.999023437500000
```

$$\sum_{k=1}^{100} \frac{1}{2^k}$$

```
s=0

for k=1:100
    s = s + (1/(2^k));
end

format long
s
```

```
s =
     1
```

4) FUNCTIONS

FUNCTIONS

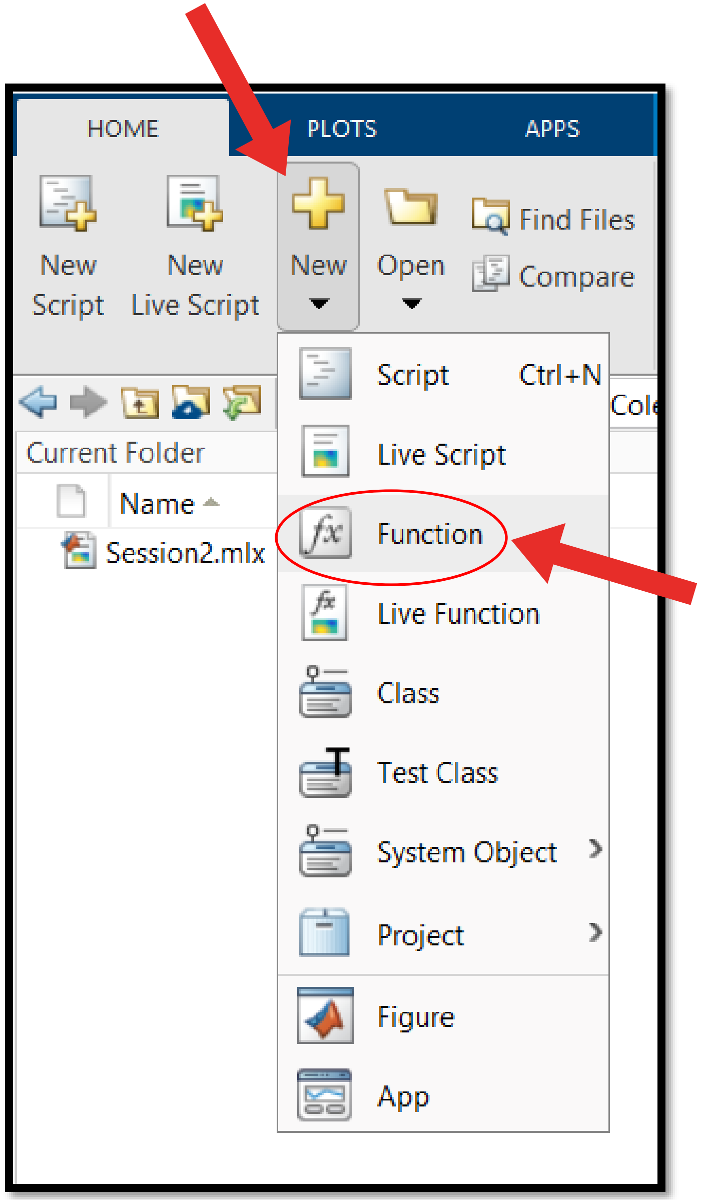
- ACCEPT **INPUT AND OUTPUT** ARGUMENTS

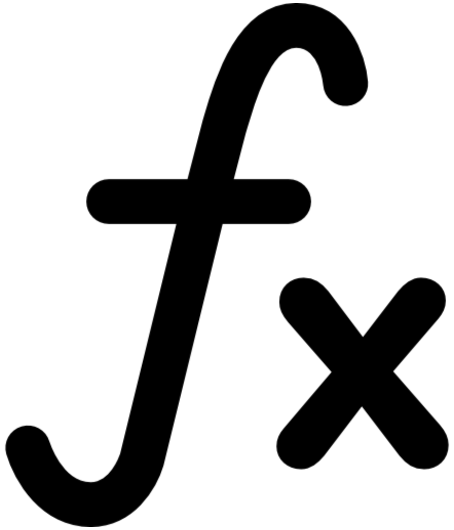
- **SYNTAX OF A FUNCTION**:

function [x,y] = name(a,b,c)

OUTPUT **INPUT**

- **TO CREATE A FUNCTION >>>**





FUNCTIONS

- WE CAN **WRITE COMMENTS** USING:

%

Editor - untitled4 *

Session2.mlx | untitled4 *

```
function [output] = untitled4(input)
%UNTITLED4 Summary of this function goes here
%   Detailed explanation goes here
outputArg1 = inputArg1;
outputArg2 = inputArg2;
end
```

- THESE COMMENTS WILL APPEAR WHEN YOU TYPE:

```
>> help *name_of_your_function*
```

- **IMPORTANT**: THE NAME OF THE FUNCTION MUST BE THE SAME AS THE .m FILE

FUNCTIONS · EXAMPLE

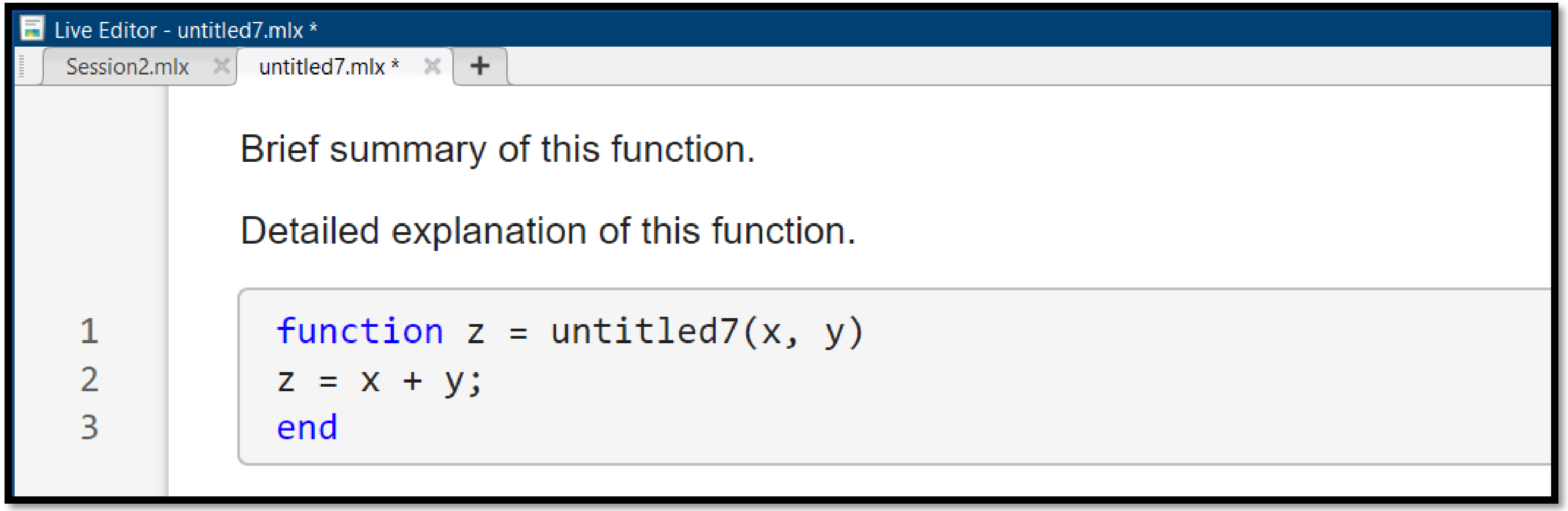
- THIS FUNCTION CALCULATES THE **SUM AND PRODUCT OF 3 GIVEN NUMBERS**:

```
function [x,y] = example(a,b,c)
% This function has 2 output arguments:
% x for the sum of a, b and c
% y for the product of a, b and c
x = a + b + c;
y = a * b * c;
end
```

- THEN, WE CAN SAVE IT AS "**example.m**" ("***example.mlx***" *IN THE LIVE EDITOR*)

FUNCTIONS

- WE CAN USE THE **LIVE EDITOR** TO CREATE FUNCTIONS:
  - THE **EXTENSION** WILL CHANGE TO "**.mlx**"
  - THIS NEW ENVIRONMENT IS MORE APPEALING



FUNCTIONS · EXAMPLE

- ONCE WE HAVE SAVED IT, WE CAN **TYPE IN THE COMMAND WINDOW**:

```
[x,y]=example(4,8,-3)
```

- AND WE WOULD OBTAIN:

```
x = 9
y = -96
```

EXERCISE 3

- CREATE A FUNCTION THAT COMPUTES THE FOLLOWING FOR A GIVEN “**n**”:

$$\sum_{k=1}^{n} \frac{1}{2^k}$$

  - THE **INPUT** IS “**n**”
  - THE **OUTPUT** IS THE **SUM**

- THEN, COMPUTE THE RESULT FOR:
  - **n = 10**
  - **n = 100**

EXERCISE 3 · SOLVED

- THESE ARE THE **FUNCTION** AND THE **RESULTS** FOR **n = 10** AND **n = 100**:

```
function [sum] = summatory(n)
% This function computes the summatory from 1 to n
% It has 1 input (n)
% It has 1 output (the result of the summatory)
sum = 0;
for k=1:n
    sum = sum + (1/(2^k));
end
```

```
[sum]=summatory(10)     sum = 0.9990
[sum]=summatory(100)    sum = 1
```

5) IF

IF

- USED TO **CHOOSE** BETWEEN STATEMENTS BASED ON **LOGICAL PROPOSITIONS**

- **SYNTAX OF AN IF:**

```
if (condition)
sentences
end
```

```
if (condition)
sentences 1
else
sentences 2
end
```

BRACKETS ARE NOT MANDATORY

- IN THE PROPOSITION, WE CAN USE:

  - RELATIONAL OPERATORS:

    < *(less than)*
    > *(greater than)*
    <= *(less or equal than)*
    >= *(greater or equal than)*
    == *(equal)*
    ~= *(not equal)*

  - LOGICAL OPERATORS:

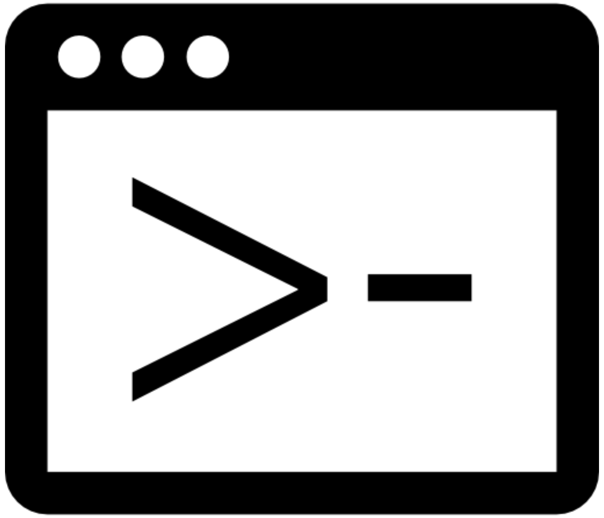
    & *(for "and")*
    | *(for "or")*
    ~ *(for "not")*

EXERCISE 4

- **FIND OUT** WHAT THESE **USEFUL** COMMANDS DO:

  (USE: "help name_of_the_command")

  - **disp**
  - **return**
  - **fprintf**



EXERCISE 4 · SOLUTION

disp:

```
x = 'Hello, world'
disp(x)
```

return:

```
y = 3
if (y < 5)
    disp(x)
    return
else
    ...
end
```

fprintf:

Writes formatted data to a text file

```
x = 'Hello, world'

Hello, world
```

```
y =

     3

Hello, world
```

EXERCISE 5

- **CREATE A FUNCTION** CALLED "**ecuagr2**" TO SOLVE QUADRATIC EQUATIONS.
- THIS ALGORITHM MIGHT BE **USEFUL**:
  - **INPUT**: COEFFICIENTS **a, b** AND **c** OF $ax^2 + bx + c = 0$
  - **OUTPUT**: A **VECTOR x** WITH **2 COORDINATES** (*THE REAL ROOTS*) **OR A MESSAGE** TELLING US THAT THE ROOTS ARE NOT REAL. STEPS:
    1. Calculate $d = b^2 - 4ac$
    2. If $d < 0$, print *The roots are complex* , do $x = []$ (empty vector) and end the program.
    3. If $d > 0$, do $d = \sqrt{d}$, $x_1 = \frac{-b+d}{2a}$ and $x_2 = \frac{-b-d}{2a}$.
- USE THE FUNCTION THAT YOU OBTAINED WITH THESE 3 **POLYNOMIALS:**

$\boxed{x^2 + x + 1}$ $\boxed{2x^2 - 8}$ $\boxed{x^2 - 16x + 64}$

EXERCISE 5 · SOLVED

- THESE ARE THE **FUNCTION** AND THE **ROOTS** OF THE POLYNOMIALS:

```
function [x] = ecuagr2(a,b,c)
% This function gets the real roots of a polynomial
% The 3 input variables must be written according to:
% ax^2+by+c=0
d = b^2-4*a*c;
if (d < 0)
    x = [];
    disp('The roots are complex')
else
    d = sqrt(d);
    x1 = (-b+d)/(2*a);
    x2 = (-b-d)/(2*a);
    x = [x1, x2];
end
end
```

x^2+x+1:

```
[x]=ecuagr2(1,1,1)
```

```
The roots are complex

x =

    []
```

2x^2-8:

```
[x]=ecuagr2(2,0,-8)
```

```
x = 1×2
     2    -2
```

x^2-16x+64:

```
[x]=ecuagr2(1,-16,64)
```

```
x = 1×2
     8     8
```

6) WHILE

WHILE

- USED TO **REPEAT** ACTIONS DEPENDING ON A LOGICAL EXPRESSION

- **SYNTAX OF A WHILE:**

```
while (logical expression)
sentences
end
```

- **EXAMPLE**: WE WANT TO CREATE A FUNCTION TO KNOW HOW MANY TERMS (k) ARE NEEDED FOR THIS SERIES

$$\sum_{n=1}^{\infty} \frac{1}{n}$$

TO BE GREATER THAN "B" (INPUT)

```
function k=numterseries(B)
% k=numterseries(bound) is the number of terms
% such that 1+ 1/2 + ... + 1/k is greater than B
k=0;
s=0;
while s<=B
k=k+1;
s=s+1/k;
end
```

EXERCISE 6

- COMPILE THE **LAST FUNCTION** WITH **B=10** AND **B=20:**
- ADD “**tic toc**” TO MEASURE THE TIME SPENT IN CALCULATIONS

```
function [k] = numterseries(B)
% k=numterseries(bound) is the number of terms
% such that 1 + 1/2 + ... + 1/k is greater than B
k=0;
s=0;
while (s<=B)
k=k+1;
s=s+1/k;
end
```

EXERCISE 6 · SOLUTION

```
tic
[k]=numterseries(10)
[k]=numterseries(20)
toc
```

```
k = 12367
k = 272400600
Elapsed time is 0.345131 seconds.
```

```
function [k] = numterseries(B)
% k=numterseries(bound) is the number of terms
% such that 1 + 1/2 + ... + 1/k is greater than B
k=0;
s=0;
while (s<=B)
k=k+1;
s=s+1/k;
end
```

EXERCISE 7

- WE HAVE A SHEET OF PAPER THAT IS **0.5mm IN THICKNESS**
- **EACH TIME** WE FOLD THE PAPER IN HALF, WE OBTAIN THE DOUBLE IN THICKNESS
    - **INPUT**: A LENGHT **x** (IN km)
    - **OUTPUT**: THE NUMBER OF TIMES (**n**) THAT THE SHEET OF PAPER HAS TO BE FOLDED TO BE GREATER IN THICKNESS THAN **x**
- USE THE FUNCTION FOR THE FOLLOWING VALUES OF **x**:
    - THE DISTANCE BETWEEN **ALBACETE AND MADRID (223 km)**
    - THE DISTANCE BETWEEN **THE EARTH AND THE MOON (384.400 km)**

EXERCISE 7 · SOLUTION

```
function [n] = fold(x)
% This function calculates how many times a paper sheet
% has to be folded for its thickness to be greater than
% a certain distance "x" that must be expressed in km
% (Each times it folds the thickness gets doubled)
n=0;
thickness = 0.5/1000000;
while (thickness < x)
    thickness = thickness*2;
    n = n+1;
end
```

Distance between **Albacete and Madrid** (223 km):

```
[n]=fold(223)
```

```
n = 29
```

Distance between the **Earth and the Moon** (384400km):

```
[n]=fold(384400)
```

```
n = 40
```