Mathematical and computational methods for economists
PART II

Prof. Mauro Maria Baldi

Department of Economics and Law,
University of Macerata,
mauromaria.baldi@unimc.it

AGENDA

- The **linspace** command
- Punctual operations
- Functions applied to vectors
- for loops
- while loops
- selection
- infinite loops

ACKNOWLEDGEMENT

This material was prepared also taking inspiration from some slides by Professor Elisabetta Michetti, to whom my thanks go.

OPERATOR :

An **equally spaced vector** from i to j is a vector in which

i is the first element

j is the last element

The distance between each element and the previous one is constant and given by **p**

The command **i:p:j** is used to create an equally spaced vector from element i to element j with step p

Notice: if i>j a negative p must be used

Notice: if p is not specified then p=1 will be assumed

OPERATOR :

EX: save a row vector Y with equally spaced elements from 0 to 40 and step 4

```
>> Y=0:4:40;
```

EX: from vector Y, create a new vector given by the elements of Y from the second to the sixth

```
>> Y1=Y(2:6)
```

Notice: in this last case p is not specified since the required step is equal to 1

EXERCISE

I) Save the equally spaced column vector P having elements from -10 to 60 and step 2.

II) Save the equally spaced row vector Q having 11 elements from -1 to -6. Which step do you have to use?

III) Trasform Q into a column vector and then delete its elements from the 2nd to the 5th

SOLUTION - I

```
>> P = (-10:2:60)'

P =

-10
-8
-6
-4
-2
0
2
4
6
8

% etcetera

54
56
58
60
```

This time, we'll be working directly in the command window. It's a personal choice whether to write a script or work directly in the command window. Generally, complex programs require a script (and often more than one!), while instructions for simpler problems can be typed directly into the command line

The array has been cut for obvious reasons of space. If you try on your own, you will see all the elements in the array.

SOLUTION - II

To address this point, we need to find a formula for the step size. Consider, for example, the array [0, 2, 4]. There are three equally-spaced elements between 0 and 4 with a step size Δ of 2. Notice that

Last element in the array

First element in the array

$$\Delta = \frac{4 - 0}{3 - 1} = 2$$

Number of elements in the array

It is easy to write the general formula for the interval $[a, b]$ with N equally-spaced points:

$$\Delta = \frac{b - a}{N - 1}$$

SOLUTION – III

```
>> step = (-1 - (-6))/(11 - 1)

step =

0.5000

>> Q = -6:step:-1

Q =

Columns 1 through 4

-6.0000 -5.5000 -5.0000 -4.5000

Columns 5 through 8

-4.0000 -3.5000 -3.0000 -2.5000

Columns 9 through 11

-2.0000 -1.5000 -1.0000
```

As we will see shortly, there is a more efficient method for generating an evenly spaced array based on the interval's boundaries and the desired number of points.

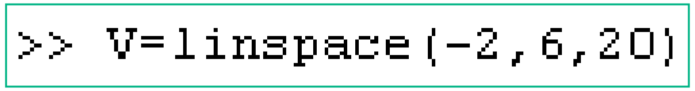
SOLUTION – IV

```
>> Q = Q';
>> Q(2:5) = []

Q =

-6.0000
-3.5000
-3.0000
-2.5000
-2.0000
-1.5000
-1.0000
```

OPERATOR linspace

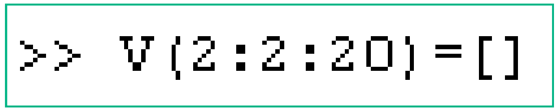
To create an equally speced vector from i to j composed by n elements use the command **linspace(i,j,n)**

EX: save the row vector V having 20 equally spaced elements from -2 to 6



```
>> V=linspace(-2,6,20)
```

EX: delete the elements of V having even indexes



```
>> V(2:2:20)=[]
```

EXERCISE

Save the row equally-spaced vector A with 60 elements from 3 to 20 and the equally spaced column vector B with 10 elements from 2 to -5

Which step has been used in the two cases?

OPERATION: sum

Consider two vectors X and Y of the same type (both row or column) and the same dimension (same number of elements)

Then it is possible the **sum (difference)** X+(-)Y between the two vectors, thus obtaining Z of the same type and dimension of the initial vectors. Each element of Z is given by the sum (difference) of the elements of X and Y having the same index

You have to use the operator **+ (-)**

EX: P row vector with equally spaced elements from 6 to 20 with step 2 and Q row vector with 8 equally spaced elements form 1 to 3. Compute R=P-Q

```
>> P=6:2:20;
>> Q=linspace(1,3,8);
>> R=P-Q
```

Notice: if it is computed X+1, the number 1 is added to each element of X

EXERCISE

Equally spaced vectors: save row A with 16 elements from 3 to 20 and column B with elements from 0 to 90 step 5

1. Can you sum the two vectors?
2. Trasform the two vector in order to make their sum possible.

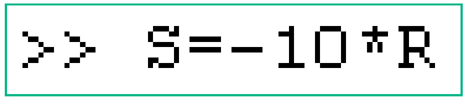
OPERATION: PRODUCT BETWEEN A SCALAR AND A VECTOR

Consider a vector X and a real number k.

The **scalar moltiplication** kX gives a vector Z (same type and dimension of X). Each element of Z is given by the product of number k with the correspondent element of X

The command to be used is *

EX: compute S=-10R



```
>> S=-10*R
```

EXERCISE

Save the row vector X with elements -1,0,5,7 and the equally spaced row vector Y with 4 elements form 10 to -8.

1. Compute Z=X+Y.
2. Compute V=0.5X.
3. Compute Z-2V.

OPERATION: PUNCTUAL PRODUCT

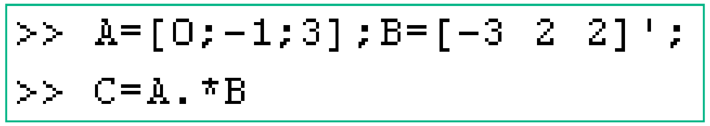
It is used to compute the products between two vectors, element by element.

Consider two vectors X and Y of the same type and same dimension.

It is possible to compute the **punctual product X.*Y** between the two vectors. The vector Z, of the same time and dimension of X, is obtained and each element of Z is given by the product of the elements of X and Y having the same position.

The command to be used is **.***

EX: consider the two column vectors A with elements 0,-1,3 and B with elements -3,2,2. Compute their punctual product.



```
>> A=[0;-1;3];B=[-3 2 2]';
>> C=A.*B
```

OPERATION: PUNCTUAL DIVISION

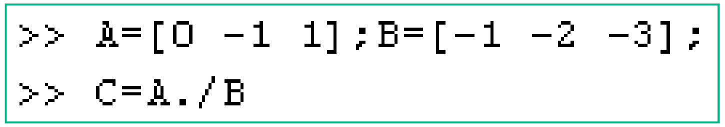
Consider two vectors X and Y of the same type and dimension

Then it is possible the **punctual division X./Y** thus obtaining Z: each element is given by the division between the corresponding element of X and Y.

The command to be used is **./**

Notice: if one lement of Y is zero, the vector Z is computed but Inf or NaN will be notified

EX: consider two row vectors A=(0,-1,1) and B=(-1,-2,-3). Compute the punctual division A./B



```
>> A=[0 -1 1];B=[-1 -2 -3];
>> C=A./B
```

EXERCISE

Save the row vector X with elements -1,0,1,2,3 and the row vector Y with equally spaced elements from -3 to 1 with step 1.

1. Calculate Z as the puntual product between X and Y. Does the commutative property hold?

2.Obtain V by dividing X with respect to Y. Is it possible on set $\mathbb{R}$ of real numbers? And with MatLab?

3. Substitute the null elements of Y with the unitary value and calculate V=X./Y+3X-1.

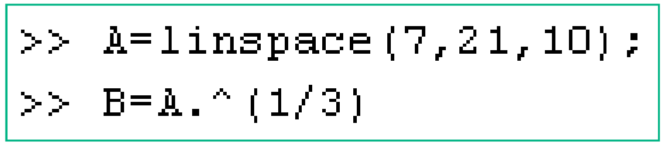
OPERATION: PUNCTUAL POWER

Consider a vector X and a real number k. Then it can be computed the **punctual power X.^k** thus obtaining a vector in which each element is obtained as the power-k of the correspondent element of X.

The command is **.^**

Notice: if for some elements of X the power-k cannot be computed, it will be notified. The other elements will be calculated. If the operation is defined only in complex set, it will be computed.

EX: row equally spaced vector A with 10 elements from 7 to 21 and k=1/3. Compute A.^k



```
>> A=linspace(7,21,10);
>> B=A.^(1/3)
```

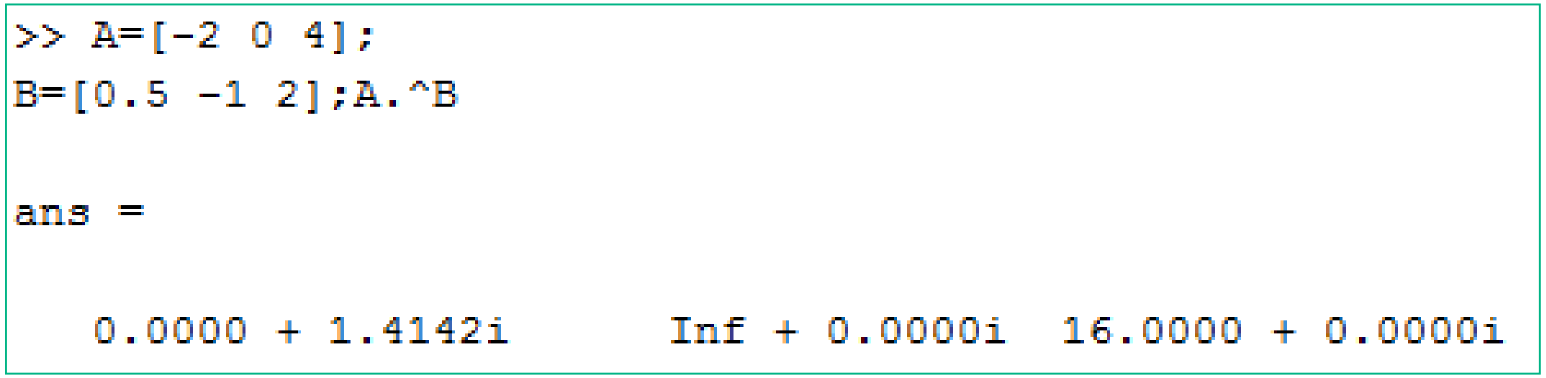
OPERATION: PUNCTUAL POWER

Consider two vectors X and Y of the same type and dimension. Then it can be computed the **punctual power X.^Y** thus obtaining a vector in which each element is obtained by raising each element of X to the correspondent element of Y.

The command is, again, **.^**

EX: A is the row with elements -2,0,4 while B is the row with elements

0.5,-1,2.



```
>> A=[-2 0 4];
B=[0.5 -1 2];A.^B

ans =

   0.0000 + 1.4142i        Inf + 0.0000i  16.0000 + 0.0000i
```

Notice: the first element of the obtained vector is a complex number while the second element cannot be computed (Inf is notified).

EXERCISE

Save the column vector X with elements -10,0,1,2,3 and the column vector Y with 6 equally spaced elements from -1 to 1.5.

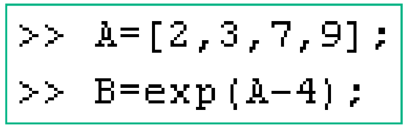
1. Calculate Z by elevating each element of X to the power -1. Is it possible in $\mathbb{R}$? With MatLab?
2. Calculate W=X.^Y and observe the result to understand its meaning.

OPERATION: A FUNCTION APPLIED TO A VECTOR

Consider a vector X (row or column with dimension n) and let **f be a function of one real variable**.

Then it is possible to calculate **f(X)** thus obtaining a vector Z (row or column with dimension n) such that each element of Z is given by the application of function f to the correspondent element of vector X.

EX: A=[2,3,7,9] and $f(x) = e^{x-4}$. Calculate $f(A)$.



```
>> A=[2,3,7,9];
>> B=exp(A-4);
```

EXERCISE

Save the following row vectors: X with equally spaced elements from 10 to 30 and step ¼ and Y with elements -1,3,7,0

1. Let $f(x)=\ln(x)-(x+1)^{2/3}$. Calculate Z=f(X) and observe the result to understand its meaning.
2. Let $f(x)=1/x+2x-\ln(2x)$. Calculate W=f(Y) and observe the result to understand its meaning.

EXERCISE

Save the row vector X with 24 equally speced elements form -5 to 7

A. Calculate $Y = \sqrt{2}X$

B. From Y delete the elements having indexes that are multiple of 3

C. Create Z with equally spaced elements form 1 to 16 step 1

D. Calculate, if possible, $V = Y + Z$

E. Let $f(x) = \dfrac{\sqrt[3]{2x^2 - 3}}{0.2x}$, calculate $W = f(Z)$

Notice: use the punctual operators when it is necessary!

EXERCISE

Save the column vector A with equally speced elements form 9 to -18 step-0,5 and the row vector B with elements (7,-1,3,5,8,e).

A. Which is the dimension of A?

B. Substitute the 10th element of A with -3

C. Transform the row vector B into the column vector $B1$

D. Let $f(x) = e^{\frac{|x+2|}{x^2}}$, calculate $C = f(\mathrm{B1})$

E. Let $f(x) = \sqrt[3]{x} - 3$, calculate $D = f(A)$

EXERCISE

Create the row vector X with 8 equally speced elements from 1 to 25, the row vector Y with equally speced elements from 10 to 90 step 10, the column vector Z with 8 elements all equal to 1

A. It it possible to compute $X+Y$? Why?

B. Delete from Y the 6th element

C. Convert Z into a row vector V

D. Calculate $W=(X./Y).\wedge V$

E. Let $f(x)=(x+30)\ln(x+30),$ calculate $f(W)$

EXERCISE

Save the row vectors x=(-1,1,-2,2,-3,3), y=(7,5,-1,3,2,0) and z=(e,e^2,e^3,e^4,e^5,e^6)

Notice: find a way to define z without listing all its elements

A. Save $a = e^3, b = \log_2 10, c = \sqrt[3]{5^2}$

B. Calculate $V = (ax - by)cz$

C. Let $f(x) = 10x^{0.5}$, calculate $f(V)$.

AGAIN OUR GUIDING EXAMPLE

The last variant of our guiding example was saved in **simple_interest_5.m**

```
w0 = 2;
ii = .1; % It's the same as 10/100 or 0.1
w3 = w0*(1 + ii*3);
fprintf("w3 = %.2f\n", w3)
w4 = w0*(1 + ii*4);
fprintf("w4 = %.2f\n", w4)
w5 = w0*(1 + ii*5);
fprintf("w5 = %.2f\n", w5)
```

The script yielded:

```
w3 = 2.60
w4 = 2.80
w5 = 3.00
```

CAN WE DO BETTER?

We can use what we have learnt so far with arrays to find the same results with less instructions! Check it out in script **simple_interest_6.m**

```
w0 = 2;
ii = .1;
t = 3:5;
w = w0*(1 + ii*t)
```

The result in the command window is:

```
w =


        2.6000        2.8000        3.0000
```

ANOTHER VARIANT OF OUR GUIDING EXAMPLE

To better appreciate the power of the punctual operator, let's consider our guiding example. This time, we will apply the compound interest rule and work directly in the command window.

```
>> w0 = 2; ii = .1; t = 3:5;
>> w = w0*(1 + ii).^t

w =

        2.6620        2.9282        3.2210
```

Again, you are free to choose whether to write a script or work directly in the command window.

CAN WE DO BETTER?

Let’s go back to the results of the last two examples. The results respectively were:

```
w =

         2.6000    2.8000    3.0000
```

```
w =

         2.6620    2.9282    3.2210
```

Now, we want to see something like this:

```
The amount at time 3 is 2.6
The amount at time 4 is 2.8
The amount at time 5 is 3.0
```

To accomplish this, we need to introduce a new tool: the iteration.

ITERATION

The word iteration comes from the Latin. In Latin, the verb «iterare» means to repeat. An alternative term in non-Latin Computer-Science is loop.

There are several kinds of loops in computer programming that are common to all the programming languages. Perhaps the most popular one is the for loop, which we will introduce first.

The syntax

```
for index = an_array
    % internal code
end
```

indicates that the variable «index» takes on, in turn, all the values stored in the array «an_array».

Let's examine some examples to solidify this concept.

DISPLAY THE CONTENT OF AN ARRAY

Suppose you have the array «cost» containing the cost of several products and you want to display them. The code is in script **loop1.m**

```
cost = [500 1000 70];
for c = cost
   disp(c)
end
```

The variable c takes on, in turn, all the values of the elements stored in the array «costs».

Each value is then displayed in the command line.

The result is:

```
500

1000

70
```

CAN WE DO BETTER?

We want to improve the code in order to display something like this:

```
The price of product 1 is 500 euro
The price of product 2 is 1000 euro
The price of product 3 is 70 euro
```

If we have the index ii of the array «cost», then the label of the product is the index ii itself and its price is given by cost(ii).
So, we need to print something like:

```
fprintf("The price of product %d is %.2f\n", ii, cost(ii))
```

where ii ranges from 1 to 3.
Thus, the array regulating the for loop is:

```
1:3
```

A FIRST SOLUTION (NOT RECOMMENDED)

Let’s save this code in script **loop2.m**

```
cost = [500 1000 70];
for ii = 1:3
    fprintf("The price of product %d " + ...
    "is %.2f\n", ii, cost(ii))
end
```

We get:

```
The price of product 1 is 500.00
The price of product 2 is 1000.00
The price of product 3 is 70.00
```

A FIRST ISSUE

The code is correct but there is a drawback. How about if we add a new product? For example, we can have something like this:

```
cost = [500 1000 70 50];
for ii = 1:3
    fprintf("The price of product %d " + ...
    "is %.2f\n", ii, cost(ii))
end
```

We still get:

```
The price of product 1 is 500.00
The price of product 2 is 1000.00
The price of product 3 is 70.00
```

But the price of the new product has not been displayed!

ANOTHER ISSUE

Suppose now we have only two products:

```
cost = [500 1000];
for ii = 1:3
    fprintf("The price of product %d " + ...
    "is %.2f\n", ii, cost(ii))
end
```

This time we even get an error!

```
The price of product 1 is 500.00
The price of product 2 is 1000.00
Index exceeds the number of array elements. Index must not
exceed 2.

Error in loop2 (line 5)
"is %.2f\n", ii, cost(ii))
```

HOW TO FIX THE PROBLEM - I

We could fix the problem in the two previous examples as follows:

```
cost = [500 1000 70 50];
for ii = 1:4
    fprintf("The price of product %d " + ...
"is %.2f\n", ii, cost(ii))
end
```

```
cost = [500 1000];
for ii = 1:2
    fprintf("The price of product %d " + ...
"is %.2f\n", ii, cost(ii))
end
```

The respective outputs are:

```
The price of product 1 is 500.00
The price of product 2 is 1000.00
The price of product 3 is 70.00
The price of product 4 is 50.00
```

```
The price of product 1 is 500.00
The price of product 2 is 1000.00
```

CAN WE DO BETTER?

The previous scripts are correct but are prone to errors

```
cost = [500 1000 70 50];
for ii = 1:4
    fprintf("The price of product %d " + ...
"is %.2f\n", ii, cost(ii))
end
```

```
cost = [500 1000];
for ii = 1:2
    fprintf("The price of product %d " + ...
"is %.2f\n", ii, cost(ii))
end
```

If the length of the array changes, we also need to update the upper bound of the array associated with the indexes. Forgetting to do this could lead to incorrect behavior in the program.

We need a function to determine the number of elements in an array. Both the length() and numel() functions can serve this purpose, and they behave similarly for one-dimensional arrays.

AN EXAMPLE

```
>> v1 = ["sono", "il", "vettore", '1'];
>> length(v1)

ans =

4

>> v2 = ["ed io sono", "il vettore", 2];
>> numel(v2)

ans =

3

>> % Continues in the next box
```

```
>> v3 = (5:2:11)‘

v3 =

5
7
9
11

>> length(v3), numel(v3)

ans =

4

ans =

4
```

HOW TO FIX THE PROBLEM - II

Let’s write the improved code in script **loop3.m**

```
cost = [500 1000 70];
for ii = 1:length(cost)
% or:
% for ii = 1:numel(cost)
    fprintf("The price of product %d " + ...
    "is %.2f\n", ii, cost(ii))
end
```

We have:

```
The price of product 1 is 500.00
The price of product 2 is 1000.00
The price of product 3 is 70.00
```

HOW TO FIX THE PROBLEM - III

This time, if we change the array, we don’t get any error.

```
cost = [500 1000 70 100];
for ii = 1:numel(cost)
    fprintf("The price of product %d " + ...
    "is %.2f\n", ii, cost(ii))
end
```

```
cost = [500 1000];
for ii = 1:numel(cost)
fprintf("The price of product %d " + ...
"is %.2f\n", ii, cost(ii))
end
```

```
The price of product 1 is 500.00
The price of product 2 is 1000.00
The price of product 3 is 70.00
The price of product 4 is 100.00
```

```
The price of product 1 is 500.00
The price of product 2 is 1000.00
```

AGAIN OUR FINANCIAL EXAMPLES!

Let’s go back to our financial examples reported below. This time we improve the way we print the arrays. We name the corresponding scripts **simple_interest_7.m** and **compund_interest_1.m**

```
w0 = 2;
ii = .1;
t = 3:5;
w = w0*(1 + ii*t)
```

```
>> w0 = 2; ii = .1; t = 3:5;
>> w = w0*(1 + ii).^t

w =


        2.6620  2.9282  3.2210
```

```
w =


        2.6000  2.8000  3.0000
```

simple_interest_7.m

```
w0 = 2;
ii = .1;
t = 3:5;
w = w0*(1 + ii*t);
for it = 1:numel(t)
    fprintf("In year %d you gain %.2f €\n", t(it), w(it))
end
```

```
In year 3 you gain 2.60 €
In year 4 you gain 2.80 €
In year 5 you gain 3.00 €
```

compound_interest_1.m

```
w0 = 2;
ii = .1;
t = 3:5;
w = w0*(1 + ii).^t;
for it = 1:numel(t)
    fprintf("In year %d you gain %.2f €\n", t(it), w(it))
end
```

```
In year 3 you gain 2.66 €
In year 4 you gain 2.93 €
In year 5 you gain 3.22 €
```

OTHER EXAMPLES

In order to become more familiar with the for loops, we will present the following examples in the next slides:

❖ Add the name of the products to the costs.

❖ A variant of the classical «Hello world!» script.

DON'T FORGET THE PRODUCTS!

In the script **loop4.m**, we include the names in the product costs. We store the product names in another array and then print the content of both arrays.

```
cost = [70 500 1000 18000];
product = ["shoes", "mobile phone", "computer", "car"];
for ii = 1:numel(cost)
   fprintf("Cost of %s: %.2f €\n", product(ii), cost(ii))
end
```

```
Cost of shoes: 70.00 €
Cost of mobile phone: 500.00 €
Cost of computer: 1000.00 €
Cost of car: 18000.00 €
```

HELLO WORLD!

It is common in every programming language course to begin with a very simple example by printing the sentence: 'Hello, world!' We already know how to do this, and we accomplish it with the script **hello_world_1.m**.

```
disp("Hello world!")
```

```
Hello world!
```

We modify this widely-adopted tradition by initially printing the string 'Hello, world!' ten times, and subsequently, by printing it a number of times defined by the user.

HELLO WORLD! – VARIANT 1

In script **hello_world_2.m**, we print ten times the sentence «Hello world!»

```
for ii = 1:10
    disp("Hello world!")
end
```

```
Hello world!
Hello world!
Hello world!
Hello world!
Hello world!
Hello world!
Hello world!
Hello world!
Hello world!
Hello world!
```

HELLO WORLD! – VARIANT 2

In the script **hello_world_3.m**, we showcase the sentence 'Hello, world!' as many times as decided by the user. To achieve this, we employ the input() function.

```
N = input("Choose a number:");
for ii = 1:N
    disp("Hello world!")
end
```

```
Choose a number:
4
Hello world!
Hello world!
Hello world!
Hello world!
```

NOT ONLY FOR

So far, we have worked with 'for' loops, where the index is automatically increased.

However, the same iterations can be achieved using another construct: the 'while' loop.

Nevertheless, in 'while' loops, we need to ensure the manual incrementation of the index.

Here are some revisited examples, this time implemented using 'while' loops.

COMPOUND INTEREST REVISITED

In the script **compound_interest_2.m** we use a while loop to display the elements in the array.

```
w0 = 2;
ii = .1;
t = 3:5;
w = w0*(1 + ii).^t;
N = length(w);
n = 1;
while n <= N
    fprintf("In year %d you gain %.2f €\n", t(n), w(n))
    n = n + 1;
end
```

```
In year 3 you gain 2.66 €
In year 4 you gain 2.93 €
In year 5 you gain 3.22 €
```

PRODUCTS AND COSTS REVISITED

In the script **loop5.m**, we revisit the task of displaying product names and their costs, this time employing a while loop.

```
cost = [70 500 1000 18000];
product = ["shoes", "mobile phone", "computer", "car"];
N = numel(cost);
it = 1;
while it <= N
   fprintf("Cost of %s: %.2f €\n", product(it), cost(it))
   it = it + 1;
end
```

```
Cost of shoes: 70.00 €
Cost of mobile phone: 500.00 €
Cost of computer: 1000.00 €
Cost of car: 18000.00 €
```

HELLO WORLD REVISITED

In the script **hello_world_4.m**, we revisit the task of displaying the sentence 'Hello world!' multiple times, allowing the user to decide the iteration count. This is achieved using a while loop.

```
N = input("Choose a number:");
ii = 1;
while ii <= N
disp("Hello world!")
   ii = ii + 1;
end
```

```
Choose a number:
2
Hello world!
Hello world!
```

SELECTION

Let's return to our list of products with their prices. This time, we want to determine whether a product costs less than a given price (let's say 700€) or more. The idea is to implement something like this:

```
if the product ii costs less than 700€ print 'Product(ii) costs less than 700€�because
else print 'Product(ii) costs more than 700€'
```

This opportunity is provided by the selection operator, whose syntax is:

```
if condition
    % instructions executed when the condition is true
else
    % instructions executed if the condition is false
end
```

MORE INFORMATION ABOUT OUR PRODUCTS - I

In the script **loop6.m**, we display information about our products, including their prices, and indicate whether each product costs less or more than 700€.

```
cost = [70 500 1000 18000];
product = ["shoes", "mobile phone", "computer", "car"];
N = numel(cost);
it = 1;
while it <= N
    auxString = sprintf("Cost of %s: %.2f€", product(it), cost(it));
    if cost(it) <= 700
        auxString = auxString + ' <= 700€';
    else
        auxString = auxString + ' > 700€';
    end
    disp(auxString)
    it = it + 1;
end
```

MORE INFORMATION ABOUT OUR PRODUCTS - II

The result is:

```
Cost of shoes: 70.00€ <= 700€
Cost of mobile phone: 500.00€ <= 700€
Cost of computer: 1000.00€ > 700€
Cost of car: 18000.00€ > 700€
```

An important remark: with the instruction

```
auxString = auxString + ' <= 700€'
```

we append the string ' <= 700€' to auxString and store the result back in the variable auxString. This operation is commonly referred to as concatenation, as it involves combining the existing string 'auxString' with the new string ' <= 700€'.

CAN WE DO BETTER?

Suppose now we also want to determine if a product costs exactly €1000. A potential solution is presented in the script **loop7.m**.

```
cost = [70 500 1000 18000];
product = ["shoes", "mobile phone", "computer", "car"];
for it = 1:length(product)
    auxString = sprintf("Cost of %s: %.2f€", product(it), cost(it));
    if cost(it) == 1000
        auxString = auxString + ' = 1000€';
    else
        if cost(it) <= 700
            auxString = auxString + ' <= 700€';
        else
            auxString = auxString + ' > 700€';
        end
    end
    disp(auxString)
end
```

THE OUTPUT

And the output is:

```
Cost of shoes: 70.00€ <= 700€
Cost of mobile phone: 500.00€ <= 700€
Cost of computer: 1000.00€ = 1000€
Cost of car: 18000.00€ > 700€
```

CAN WE DO BETTER?

We can enhance our code by consolidating the nested if-else block with the outer if-else block, utilizing a single if-elseif-else block. The suggested solution is outlined in the script **loop8.m**.

```
cost = [70 500 1000 18000];
product = ["shoes", "mobile phone", "computer", "car"];
for it = 1:length(product)
    auxString = sprintf("Cost of %s: %.2f€", product(it), cost(it));
    if cost(it) == 1000
        auxString = auxString + ' = 1000€';
    elseif cost(it) <= 700
        auxString = auxString + ' <= 700€';
    else
        auxString = auxString + ' > 700€';
    end
    disp(auxString)
end
```

THE OUTPUT

And the output is again:

```
Cost of shoes: 70.00€ <= 700€
Cost of mobile phone: 500.00€ <= 700€
Cost of computer: 1000.00€ = 1000€
Cost of car: 18000.00€ > 700€
```

ANOTHER EXAMPLE

In the script **logical_operators.m**, we play with the logical operators NOT, AND (&), and OR (|) in concert with the selection (if) block.

```
name = input("Enter the name of a product: ");
cost = input("Enter the cost of " + name + ": ");
if cost == 30
    fprintf("The cost of %s is 30 euros\n", name)
end
if cost ~= 40
    fprintf("The cost of %s is not 40 euros\n", name)
end
% The code continues in the next slide.
```

CONTINUATION

```
%{  Continuation of the code
from the previous slide  %}
if cost > 10 & cost < 20
    fprintf("The cost of %s is more than " + ...
    "10 euros and less than 20 euros\n", name)
elseif cost < 10 | cost > 20
    fprintf("The cost of %s is less than " + ...
    "10 euros or more than 20 euros\n", name)
else
    fprintf("The cost of %s is either 10 " + ...
    "euros or 20 euros\n", name)
end
```

%{ and %} allow us to write multiple-lines comments

The '+ ...' sequence allows us to break long strings or commands across multiple lines."

POSSIBLE OUTPUTS - I

Here and in the next slides, we offer examples of output based on different inputs.

```
Enter the name of a product:
"wine"
Enter the cost of wine:
30
The cost of wine is 30 euros
The cost of wine is not 40 euros
The cost of wine is less than 10 euros or more than 20 euros
```

POSSIBLE OUTPUTS - II

```
Enter the name of a product:
"pencil"
Enter the cost of pencil:
2
The cost of pencil is not 40 euros
The cost of pencil is less than 10 euros or more than 20 euros
```

```
Enter the name of a product:
"mug"
Enter the cost of mug:
15
The cost of mug is not 40 euros
The cost of mug is more than 10 euros and less than 20 euros
```

INFINITE LOOPS

Many programming languages, including C, C++, and Pascal, provide a variation of the while loop known as the do-while or repeat-until loop. The fundamental concept is as follows:

```
do

    % some instructions

while (a condition is true)
```

```
repeat

    % some instructions

until (a condition becomes true)
```

Matlab does not have these types of loops. However, as for other programming languages, we can use infinite loops.

HELLO WORLD…FOREVER!

The following piece of code creates an infinite loop, where all the instructions within the loop are repeated indefinitely.

```
while true

    % Some instructions

end
```

You can try with the following loop that eternally displays the sentence «Hello world!». Use CTR+C to force Matlab to stop the execution of the code.

```
while true
    disp("Hello world!")
end
```

A CURIOSITY

The term 'true' in the preceding code represents a data type known as boolean. You can gain an understanding of the boolean type by entering the following instructions:

```
>> a = true
a = logical 1

>> ~a % Negation
ans = logical 0

>> 1 > 2
ans = logical 0

>> p = 'a' > 'b'
p = logical 0

>> p = 'a' < 'b'
p = logical 1
```

Due to space constraints, certain white lines and newlines have been omitted.

ALPHABETICAL ORDER

The last two commands are very interesting because they allow us to determine the alphabetical order of two words or even sentences. The result of the comparison can be stored in a boolean variable.

```
>> product1 = "potatoes";
>> product2 = "tomatoes";
>> product3 = "peppers";
>> product1 > product2
ans = logical 0

>> product1 < product2
ans = logical 1

>> product1 < product3
ans = logical 0

>>product3 < product2
ans = logical 1
```

It is «true» that «peppers» come before (<) «tomatoes» in alphabetical order!

A VARIANT OF INFINITE LOOP

- ❖ For these reasons, the expression 'while true' signifies 'perform the following instruction as long as true remains true.' However, since true is always true, the instructions are executed indefinitely.

- ❖ A convention shared by all programming languages is that the value 0 corresponds to false, while all other values correspond to true. Therefore, instead of 'while true,' we can use, for example, 'while 1' to achieve the same result.

- ❖ But how can we escape from an infinite loop? We implement a stopping condition within an 'if' block, in which we enforce the exit from the infinite loop using the 'break' instruction.

AN EXAMPLE

In the script **loop9.m**, we address the same results achieved in the script **loop8.m**, this time using an infinite loop and forcing its end after the counter reaches the end of the array.

```
cost = [70 500 1000 18000];
product = ["shoes", "mobile phone", "computer", "car"];
it = 1;
N = length(cost);
while 2
    if it > N
        break;
    end
    auxString = sprintf("Cost of %s: %.2f€", product(it), cost(it));
    % The code continues in the next slide
```

CONTINUATION AND OUTPUT

```
    % The code originates from the preceding slide.
    if cost(it) == 1000
        auxString = auxString + ' = 1000€';
    elseif cost(it) <= 700
        auxString = auxString + ' <= 700€';
    else
        auxString = auxString + ' > 700€';
    end
    disp(auxString)
    it = it + 1;
end
```

```
Cost of shoes: 70.00€ <= 700€
Cost of mobile phone: 500.00€ <= 700€
Cost of computer: 1000.00€ = 1000€
Cost of car: 18000.00€ > 700€
```