# Scalar vs Array Programming

This is a simple programming exercise. The goal is to contrast programming with scalars and arrays.

## BACKGROUND

MATLAB programs can be implemented with variables representing scalars or arrays. Working with scalars helps us to understand the simplest programming structures. Working with arrays will allow us to handle programs requiring larger amount of data and results.

**PROBLEM #1. The simplest example**: For instance, we would like to print a list of x values in the range [1,10] in steps of 1. Review the short programs below, find the differences and pros and cons of both methods.

```
% Scalar Implementation

clc, clear

for x=1:1:10
    fprintf('%d \n', x);
end
```

```
% Array Implementation

clc, clear

for ii=1:1:10
  if ii==1
     x(ii)=1;
  else
     x(ii)=x(ii-1)+1;
  end
  fprintf('%d \n', x(ii));
end
```

In the scalar method, in spite of its simplicity, x values are not stored, except for the last one. Each new computation of x, erases the value of the previous one. The main difference in the array implementation is that all elements of the computed x are stored in memory, as an array. Why do we want to do so? Answers could be, (A) we need the values of x for further computations, (B) we need to output values acquired by x, and then do further computation, (C) We prefer to use the Ferrari instead of the bicycle.

Exercise-1: Run both programs independently and afterwards write in on the Command Window

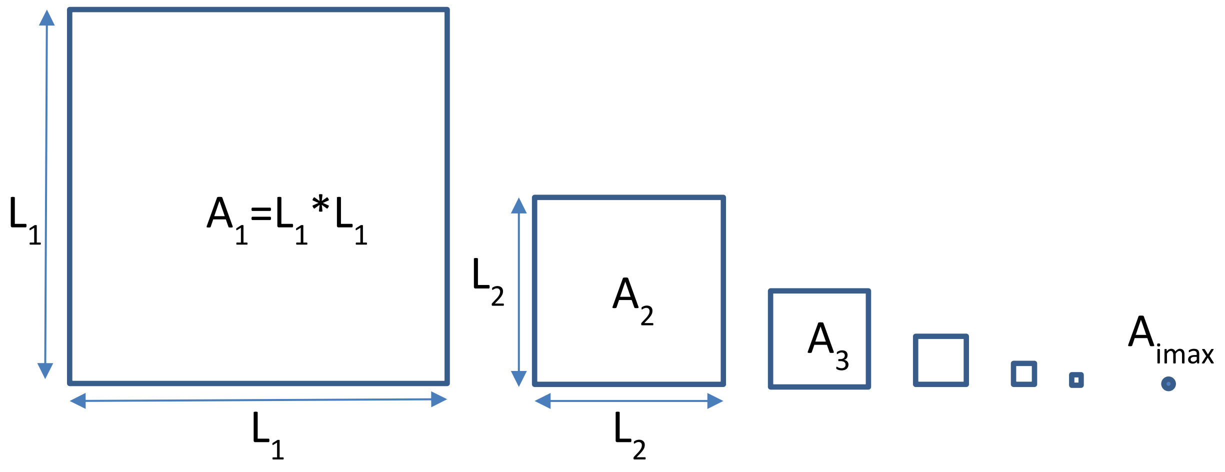
```
>> x  <Enter>
```

Evaluate the output of this statement after running each program.

## PROBLEM #2

You are required to apply the above ideas. Write a MATLAB program to sum the areas of squares decreasing in size as shown below using the array structure. First, develop a scalar solution and then take it as a basis for the array implementation.

Addition will occur as long the sum of areas is less than 13,333.33 unit$^2$:



This idea can be expressed as

$$S = \sum_{i=1}^{imax} A_i \leq 13{,}333.33\, unit^2$$

Also,

$$S = A_1 + A_2 + A_3 + \cdots + A_{imax} \leq 13{,}333.33\ \ unit^2$$

To compute each area:

$A_i = L_i * L_i$ , the area of the i-square

$L_{i+1} = \frac{L_i}{2}$, each new square side length is half the previous one

Specifications

- See figure above, excel sample calculation and MATLAB code below for a better understanding.
- L and A variables must be arrays
- Start with $L_1 = 100\ units$
- Initialize input variable(s)
- Decide appropriate loop syntax form and develop a stop criterion
- Compute the area of the current square.
- Accumulate the current square area to the subtotal
- Add a counter to find out how many areas you should add up to reach the target of 13333.33 unit$^2$
- Print the result: the number of squares added and the magnitude of the sum of areas.
- Although you are able to see the solution, assume you don't know *a priori* the number of iterations needed.

## Excel, Sample, Calculation:

| ii | L | A | SUM |
|---|---|---|---|
| 1 | 100 | 10000 | 10000 |
| 2 | 50 | 2500 | 12500 |
| 3 | 25 | 625 | 13125 |
| 4 | 12.5 | 156.25 | 13281.25 |
| 5 | 6.25 | 39.0625 | 13320.3125 |
| 6 | 3.125 | 9.765625 | 13330.07813 |
| 7 | 1.5625 | 2.441406 | 13332.51953 |
| 8 | 0.78125 | 0.610352 | 13333.12988 |
| 9 | 0.390625 | 0.152588 | 13333.28247 |
| 10 | 0.195313 | 0.038147 | 13333.32062 |
| 11 | 0.097656 | 0.009537 | 13333.33015 |
| 12 | 0.048828 | 0.002384 | 13333.33254 |
| 13 | 0.024414 | 0.000596 | 13333.33313 |

## Code with scalars

Consider the scalar solution below:

```
% Sum of Square Areas: Scalar Solution
% documentation

clc, clear, close

L=100;
A=L*L;
S=A;

iter=1;
while S<=13333.33
    L=L/2;
    A=L*L;
    S=S+A;
    iter=iter+1;
end

fprintf('The number of squares added to the series is %d \n',iter-1);
fprintf('which yields a total area of %f \n',S-A);
```

Run the above program so you can use it as a reference for the new array approach.

## Code with Arrays

```
% Sum of Square Areas
% Array Solution
clc, clear, close
Amax=13333.33

ii=1;        % work as array index and as iteration counter
L(ii)=100;
A(ii)=L(ii)*L(ii);
S=A(ii);      % There is no need the accumulator "S" to be an array

ii=2;
while S<=Amax
    L(ii)=L(ii-1)/2;     % The new L is half the old L
    A(ii)=L(ii)*L(ii);
    S=S+A(ii);
    ii=ii+1;             % Update index
end

fprintf('The number of squares added to the series is %d \n',ii-2);
fprintf('and it yields a total area of %f \n', S-A(ii-1));
```

NOTES:

- Accumulator "S" is a scalar, we are interested only on the final result
- Amax is a parameter, not need to be an array
- L and A are arrays, each indexed element stores a value
- Codes with multiple array variables are better manipulated via their indices, which many times are the same.
- In the instruction `[L(ii)=L(ii-1)/2;]`, the variable on the RHS `L(ii-1)` refers to previous square side length, while L(ii) refers to the new calculated square side length. Indices are a key for this calculation.
- The instruction ii = ii + 1; is one of the main statements. This allow us to update the indices and also to use it as counter variable.
- Could you explain why the number of squares added are "ii-2"
- Could you explain why the total area is "S-A(ii-1)"
- Instructor question: Why so many students, in their solutions, changed the needed "while" by a "for" loop . We didn't know how many iterations are needed, the while loop is essential.

**PROBLEM #3**

A MATLAB program to compute N factorial (i.e., N!) is shown below. Write another MATLAB program with the same objective but using array variables.

```
% Factorial Program: SCALAR SOLUTION
% This program computes the factorial of k

clc, clear, close

N=input('Enter factorial base N to compute N! \n');

    fac=1;
    if N==0
        fac=1;
    else
        for ii=1:1:N
            fac=fac*ii;
        end
    end

fprintf('The factorial of %d is %d \n', N, fac);
```

SOLUTION

```
% Factorial Program: ARRAY SOLUTION
% This program computes the factorial of k

clc, clear, close

N=input('Enter factorial base N to compute N!\n');

if N==0
   fac =1;
   fprintf('The factorial of %d is %d \n', N, fac);

elseif N==1
   fact=1;
   fprintf('The factorial of %d is %d \n', N, fac);
else
   fac(1)=1;
   for ii=2:1:N
      fac(ii)=fac(ii-1)*ii;
   end
end
fprintf('The factorial of %d is %d \n', N, fac(N));
```

The above program stores the 1!, 2!, in fact. And 3! …N! in fac(3),…,fac(N). It does not store 0! in fac(0), (sorry zero is not allowed as index in matlab arrays), but does 1! In fact(1). To avoid this index problem you could roll over indices: store 0! in fact(1), 1! in fact(2), 2! In fact(3) and so on. Then above solution must be slightly modified.