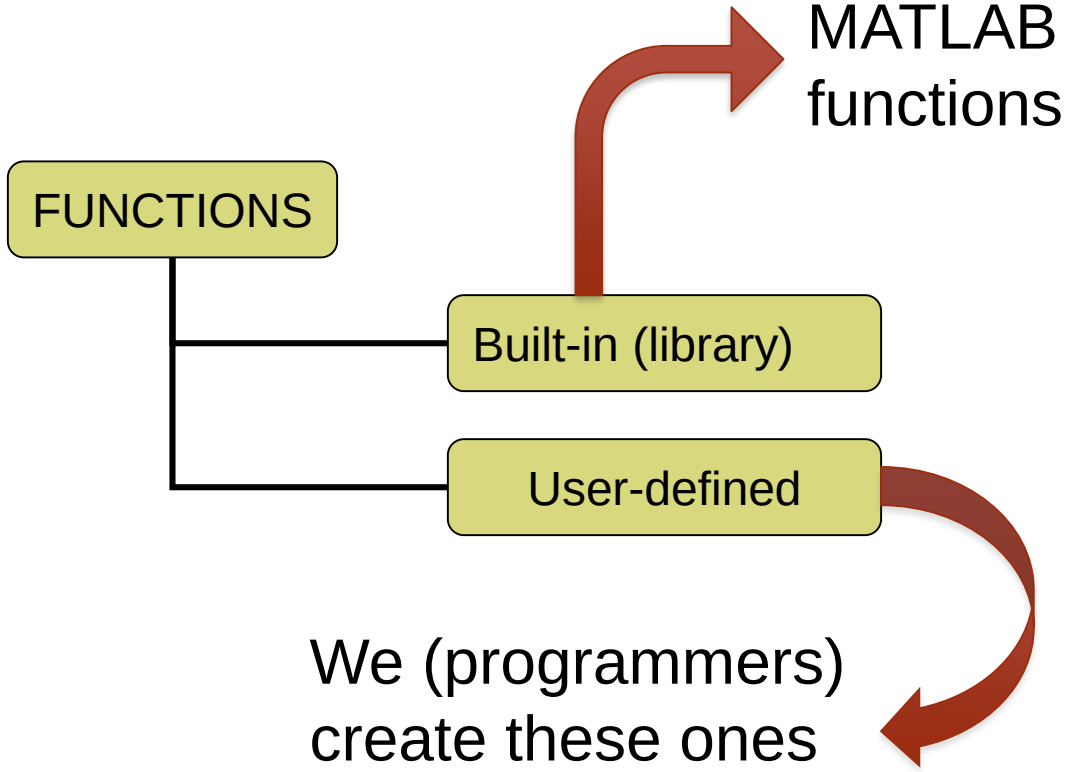
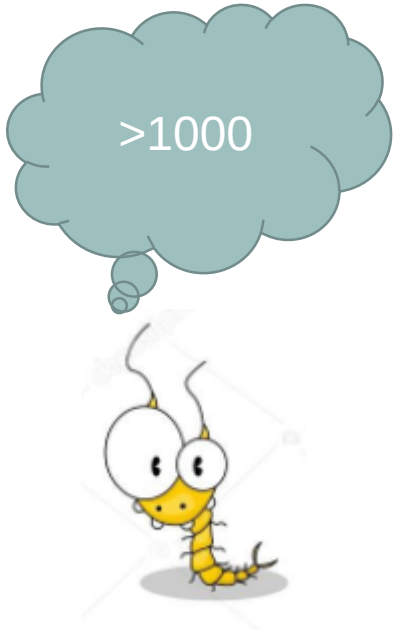
Introduction to FUNCTIONS

Small programming tasks can be implemented with the function structure



- sqrt(x)
- exp(x)
- log(x)
- log10(x)
- abs(x)
- min(x)
- max(x)
- sin(x)
- cos(x)
- tan(x)
- asin(x)
- acos(x)
- atan(x)
- pi, π

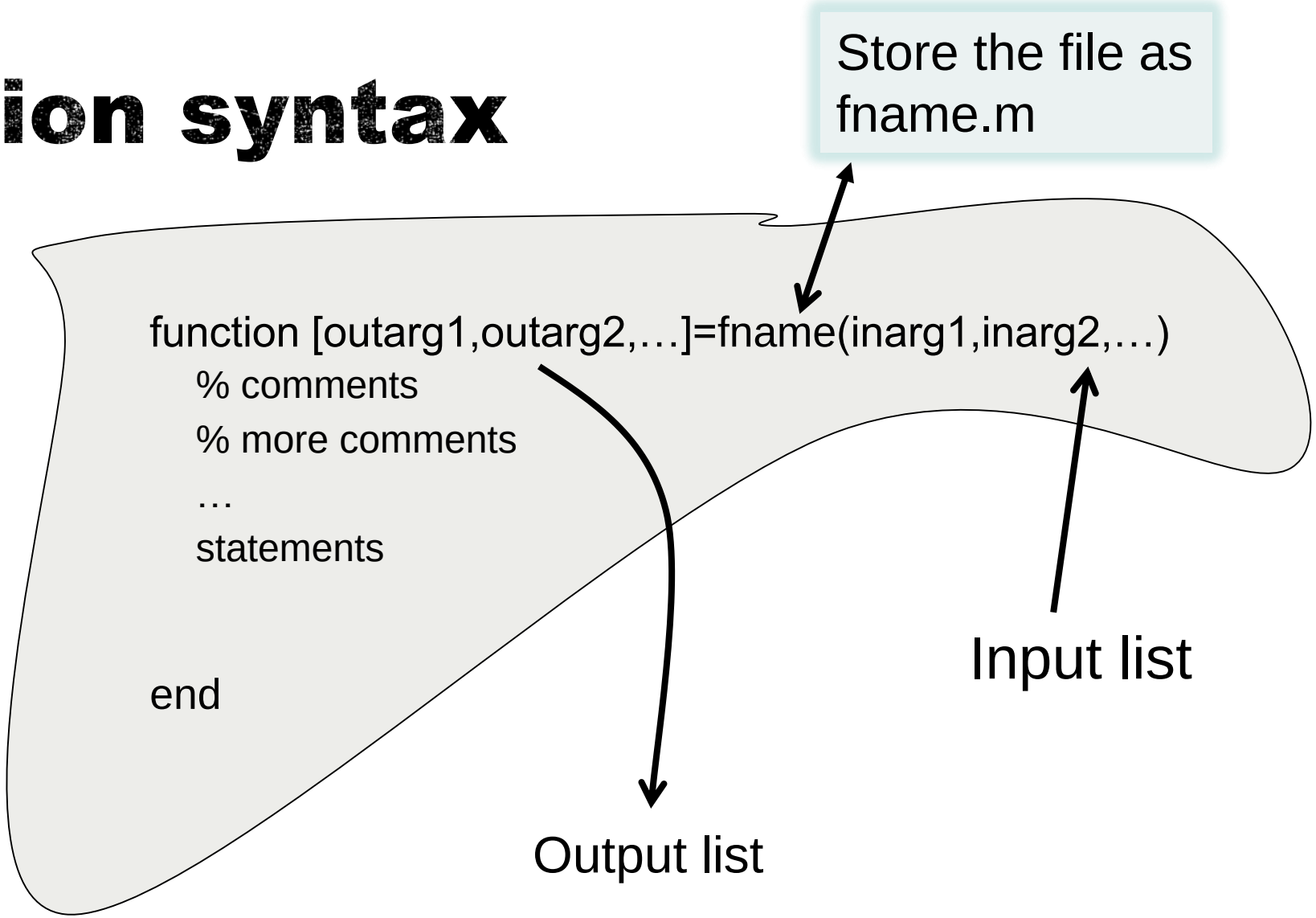


Benefits of Writing Functions

- Test and debug independently
- Reduce the efforts on large projects
- Reusable code
- Isolation from unintended side effects

It's time to take a look to the file: Simple Functions Exercises

Function syntax



~ user-defined functions are created within the Editor
~ save the file with the same name as the function

A simple Example

- Develop a function to multiply two numbers.
- The function receives two number as input arguments
- The function calculates the product of these numbers and returns the result
- Functions are developed similarly as any other program (script)
- The IPO diagram:

input	processing	output
A, B	p= A * B	p

Similarly to a computer program (script), you must be able to identify the input and output for the function, i.e., regarding number, order, and type of them.

A simple example

functions run in the Command Window or within a new program in the Editor

New M-file written in the Editor:

```
function [p] = AtimesB(A,B)
% Multiplies two numbers
    p=A * B;
end
```

Save it as **AtimesB.m**

Observe the variable p storing the result

Command Window

```
>>A=1 <E>
A =
        1
>>B=2<E>
B =
        2
>>AtimesB(A,B) <E>
ans=
      2
```

Editor

```
% Main Multiplicator
% more comments
clc, clear
A=1; B=2;
C=AtimesB(A,B);
fprintf('The results is %.1f \n', C);
```

OUTPUT

% Run it by yourself

Function call:
every time you use the function, the process is called a ' function call'

A simple example, Upgraded to handle arrays

Same M-file:

```
function [p] = AtimesB(A,B)
% Multiplies two arrays
    p=A.*B;
end
```

arrays

array operator

array

Command Window

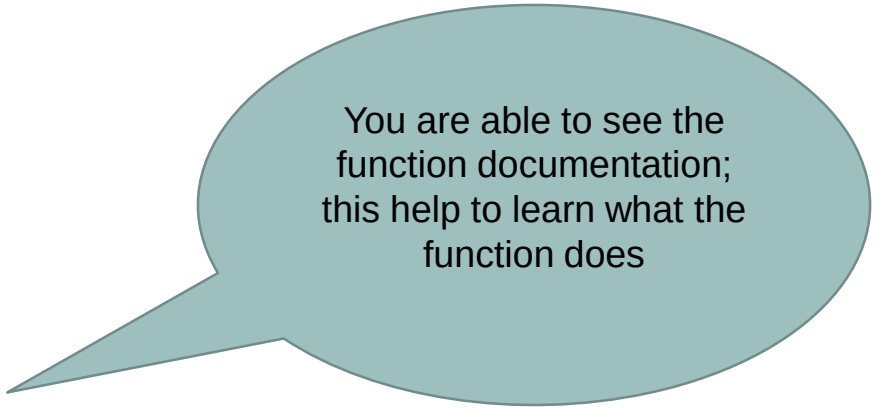
```
>>A=[1,2,3] <E>
A =
        1  2  3
>>B=[3,2,1]<E>
B =
        3  2  1
>>AtimesB(A,B) <E>
ans=
      3  4  3
```

Notice that AtimesB returns three results

Supporting On-line Help: Comments

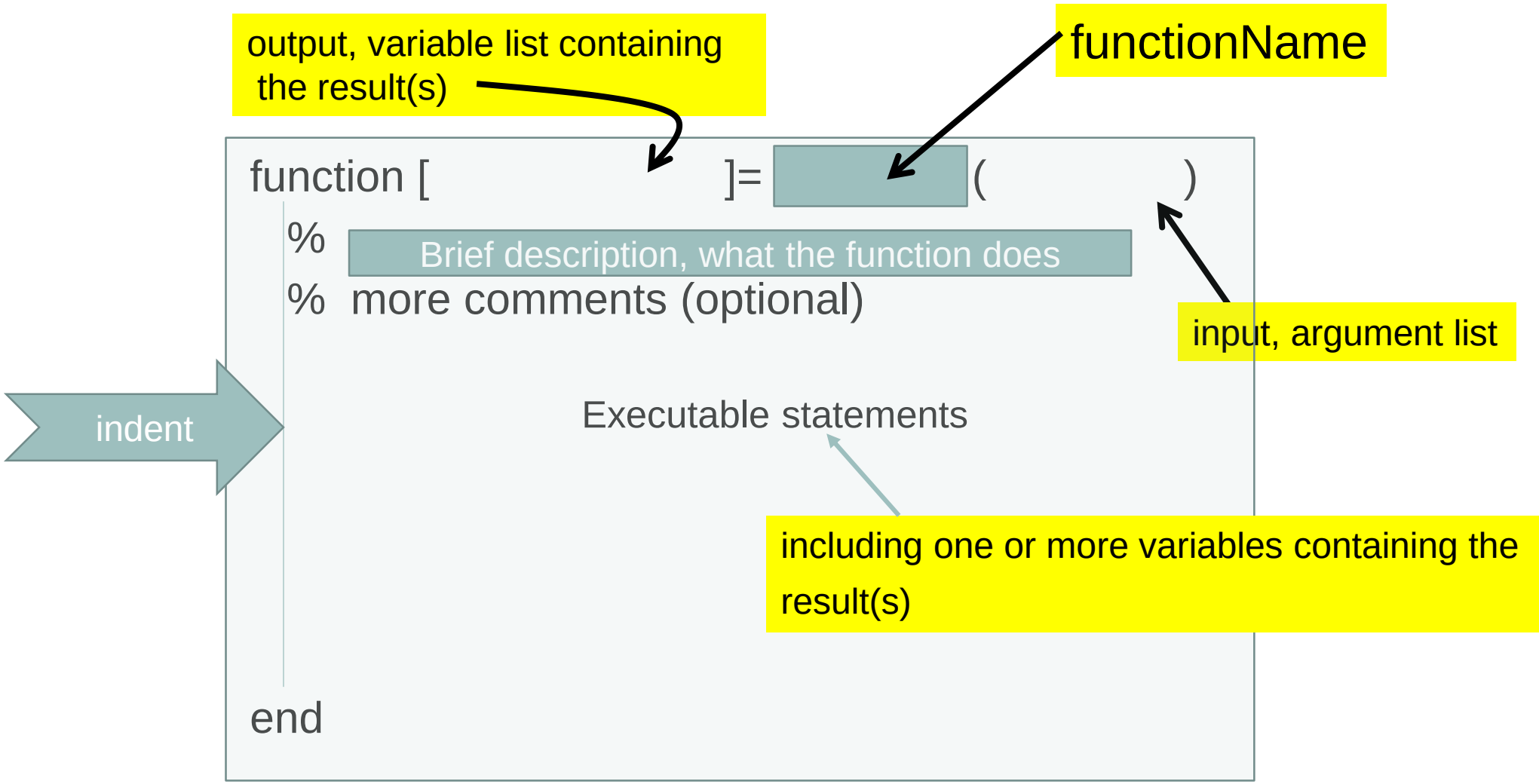
- First line of a function is the Function Header
- Second line must be a comment statement explaining what the function does
- All text from the second line up to the first non-comment is printed in response to
- `>> help functionName`

- Example with user-defined function:
- `>> help AtimesB`
- `  Computes A times B two numbers`



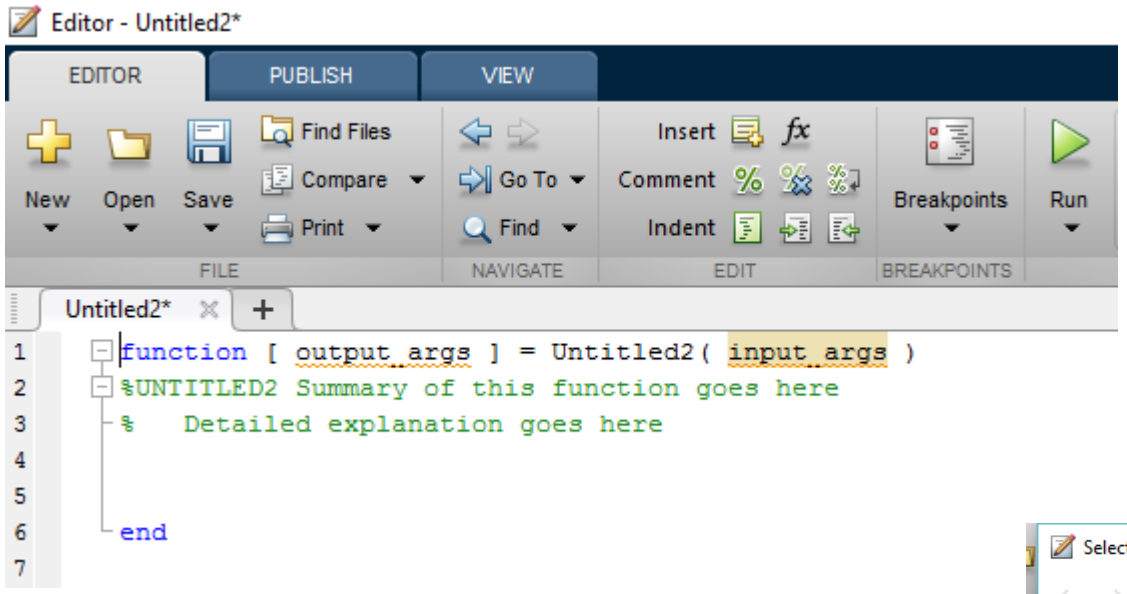
function template

Store the file under fuctionName.m

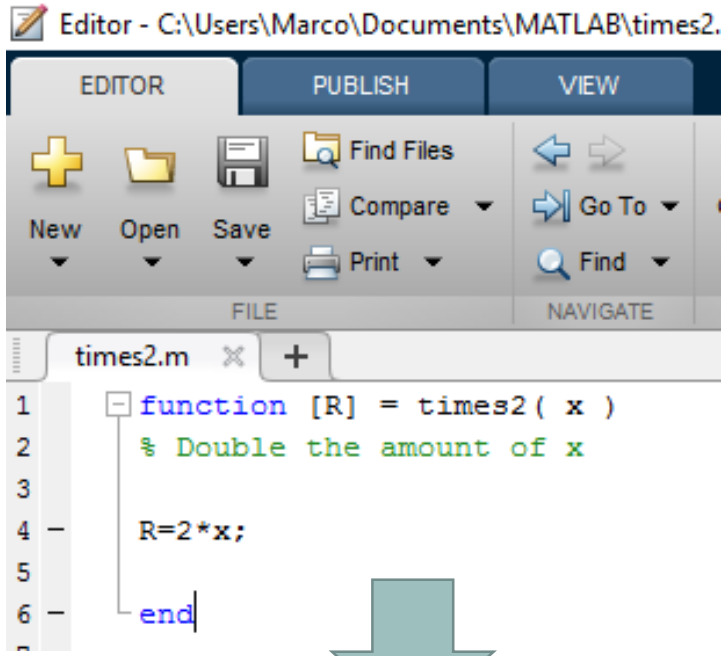


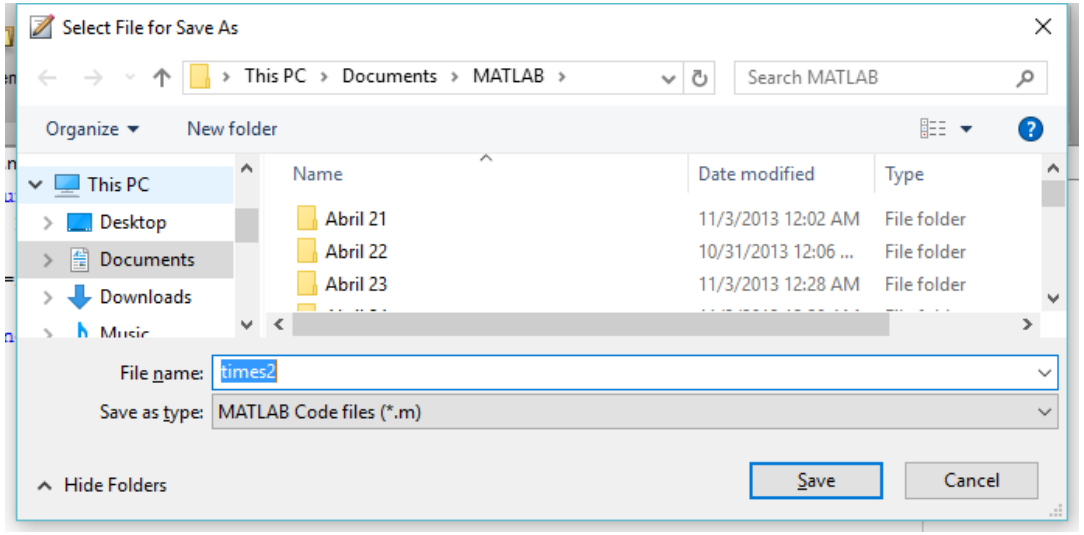
Create a Function

Editor➔New➔Function
A function template opens with some keywords already in



Write down the function:





- Save the function
in a folder in the MATLAB path.
- ML appends .m extension.

Make sure MATLAB knows the path to your function

- MATLAB cannot execute a function unless it knows where to find its m-file.
- This requires that the function be in the internal MATLAB path.
- Refer to “Setting the MATLAB Path” for more information.

- To change the path for a new folder go to the main menu: HOME/Set Path

Example: Perfect Number

(1) Write a function to check if the given number is "perfect“

A "perfect" number is an integer whose factors (excluding itself) add up to it.

- Example:
  - 6 is a perfect number because their sum of their factors gives 6
  - 1 + 2 + 3 = 6
  - 6 = 6
  - 8 is not perfect
  - 1 + 2 + 4 ~= 8
  - 7~= 8

(2) Write a program (Driver Program; Main Program) that calls the function checks if the numbers from 1 to 100 are perfect numbers. Then adds up all of them.

Perfect Number, solution (A)

```
function perfect = isPerfect(N)
    % Checks to see if the given integer number is "perfect"

    suma = 0;
    for ii = 1 : 1: (N - 1) % Exclude N
        if  mod(N,ii)==0
            suma = suma + ii;
        end
    end

    if (suma == N)  % Does the sum equal N inputtted?
            perfect = 1;
    else
            perfect = 0;
    end

end
```

Write a program that checks if the numbers from 1 to 100 are perfect numbers. Then adds up all of them.

Perfect Number, solution (B)

```
% sumPerfect.m
% Sum of perfect numbers

clc, clear
N=100;  % Better ask to user
s=0;
for ii=1:N
    if isPerfect(ii)==1
        s=s+ii;
    end
end
disp(['The sum is ', num2str(s)]);
```

Remarks

- Functions are subprograms
- Specific tasks can be encapsulated into functions. This modular approach enables development of structured solutions to complex problems.
- Functions use input and output parameters to communicate with programs, other functions, and the command window
- Functions use **local variables** that exist only while the function is executing. Local variables are distinct from variables of the same name in the workspace or in other functions.
- Input arguments allow the same calculation procedure (same algorithm) to be applied to different data. Thus, function m-files are **reusable**.
- Functions can call other functions.
- While writing functions you invest a little more time, later you would save it when the function is used over and over in different programs.