

3. m-file

M-file이란, MATLAB 명령어로 작성된 파일을 의미한다. 즉, 지금까지처럼 command window에 한 줄씩 입력하는 즉시 실행되는 방식이 아니라, editor를 통해 code를 작성한 후 한꺼번에 실행하는 방식이다. 간단한 작업이 아니면 대부분의 MATLAB 작업은 m-file을 통해 실행한다. 모든 명령어는 m-file이나 command window 모두 동일하게 작동하며, 이 장에서는 m-file에서 사용하기 편리한 기능과 함수를 소개할 것이다.

M-file은 크게 function mode와 script mode로 나눌 수 있다. Script mode는 명령어가 command window에서 실행한 것처럼 작동하는 방식으로, 생성된 변수나 결과가 모두 workspace에 저장된다. 즉, script 내에서 사용한 변수가 모두 workspace 상에 드러나며, workspace에 미리 저장되어 있던 데이터도 임의로 접근할 수 있다. 반면에, function mode는 함수를 만드는 방식으로, 입력과 출력을 정의해주어야 하며 m-file 내부에서 생성된 변수는 함수가 끝나면 사라진다. (독립된 workspace를 지니며, 함수가 종료되면 workspace도 사라진다.) 그 외에는 script와 다른 점이 없다.

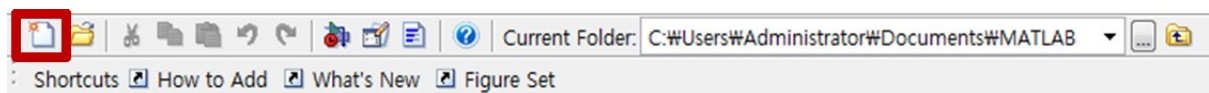
Script mode m-file을 작성할 때는 조건이 없다. 실행하고자 하는 명령어를 차례대로 작성하면 script mode로 자동 인식한다. Function mode m-file을 작성할 때는 첫 줄에 함수 정의가 있어야 한다. 함수 정의 양식은 다음과 같다.

□ `function [output1, output2, output3] = function_name(input1, input2, input3)`

맨 처음의 function은 반드시 써주어야 하며, 입력과 출력 인자인 output과 input은 개수에 제한이 없다. (없어도 된다.) 입력 인자는 함수가 시작할 때 이미 변수가 생성되어 있는 것처럼 사용할 수 있으며, 출력 인자는 동일한 이름의 변수에 값을 넣어두면 자동으로 반환된다.

M-file은 실행할 때 m-file의 파일 이름을 명령어로 사용한다. 즉, script mode m-file을 저장한 후 command window에서 파일 이름을 입력하면 해당하는 m-file이 실행되는 것이다. 마찬가지로, function mode m-file은 함수 정의 양식과 비슷하게 입력과 출력을 넣어주면 실행된다. 다만, function mode m-file의 함수 명과 m-file 명은 동일해야 한다.

3.1. 간단한 M-file 작성하기



위 그림은 MATLAB 메인 창의 툴바이다. 사각형으로 표시된 아이콘(New Script)을 누르면 새 창에 editor가 생성된다. 이 editor로 m-file을 작성할 수 있다.

간단한 예제로 팩토리얼을 구하는 function mode m-file을 작성해보자.

예제3.1. Factorial1

```

1  function y = my_factorial(n)
2
3  % my_factorial Factorial function.
4  % my_factorial(n)은 스칼라 n에 대하여 1부터 n까지의 모든 정수의 곱이다.
5
6  y = prod(1:n);

```

```

>> my_factorial(4)

ans =
    24

>> help my_factorial
my_factorial Factorial function.
    my_factorial(n)은 스칼라 n에 대하여 1부터
    n까지의 모든 정수의 곱이다.

>> lookfor factorial
my_factorial - Factorial function.
my_factorial - Factorial function.
factorial    - Factorial function.
ff2n         - Two-level full-factorial design.
fracfact     - Fractional factorial design for two-
level factors.
fracfactgen  - Fractional factorial design
generators.
fullfact     - Mixed-level full-factorial designs.

```

위의 m-file을 파일 명이 my_factorial.m이 되도록 저장한 후, command window에서 왼쪽과 같이 실행하면 결과가 나타난다. 참고로, 출력 변수가 하나일 때는 []를 사용하지 않아도 되며, prod는 벡터의 모든 요소의 곱을 구해주는 함수이다.

%는 m-file 내에서 한 줄 주석을 의미하는데, 함수 정의 바로 아래에 있는 주석은 help 명령을 실행했을 때 나타난다.

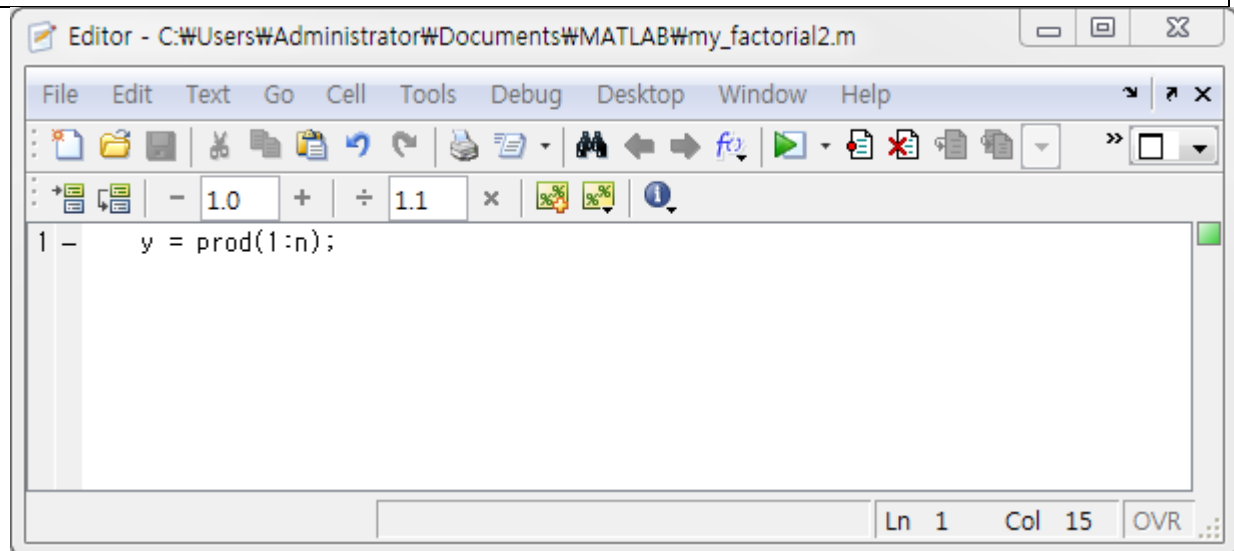
특히, 주석 첫 번째 줄은 lookfor를 통해 검색할 때 키가 되는 문장이다.

정리하면, 다음과 같은 과정으로 function mode m-file을 만들 수 있다.

- ① 함수 정의: function [output1, output2, ...] = function_name(input1, input2, ...)
- ② 함수 기능 구현 및 출력 정의
- ③ 주석 첨부: help와 lookfor 검색을 위한 주석은 함수 정의 바로 다음에 작성
- ④ 함수 명과 동일한 이름으로 m-file 저장

Script mode m-file은 function mode보다 훨씬 간단하다. 그냥 차례대로 실행시키고 싶은 명령어를 저장해두면 command window에서 m-file 이름만으로 실행시킬 수 있다. Command window에서 실행시킨 것과 동일한 효과이다.

예제3.2. Factorial2

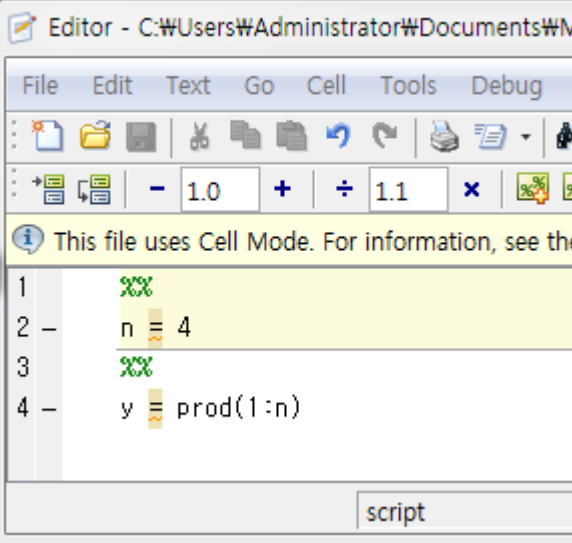
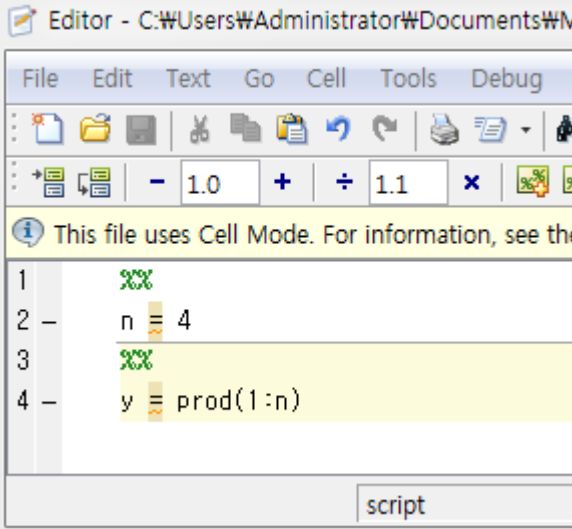


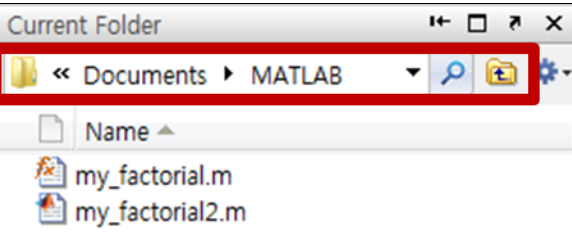
```
>> n = 4;
>> my_factorial2
>> y

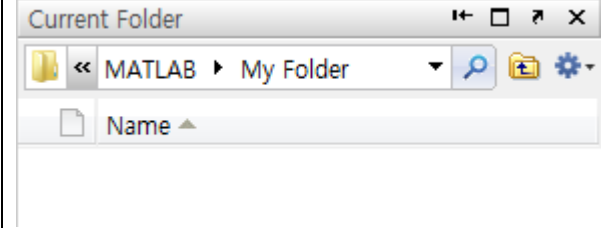
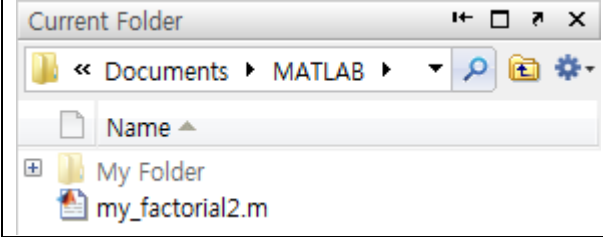
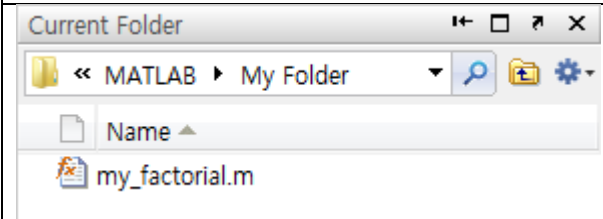
y =
    24
```

위의 m-file을 my_factorial2.m으로 저장한 뒤, 왼쪽과 같이 실행할 수 있다. 처음에 $n = 4$ 라고 정의해주지 않으면 m-file의 `prod(1:n)`의 n 이 없어 실행되지 않는다. 실행결과인 y 는 workspace에 저장되어 있다. 즉, command window에서 `y = prod(1:n);` 을 실행시킨 것과 완전히 동일하다.

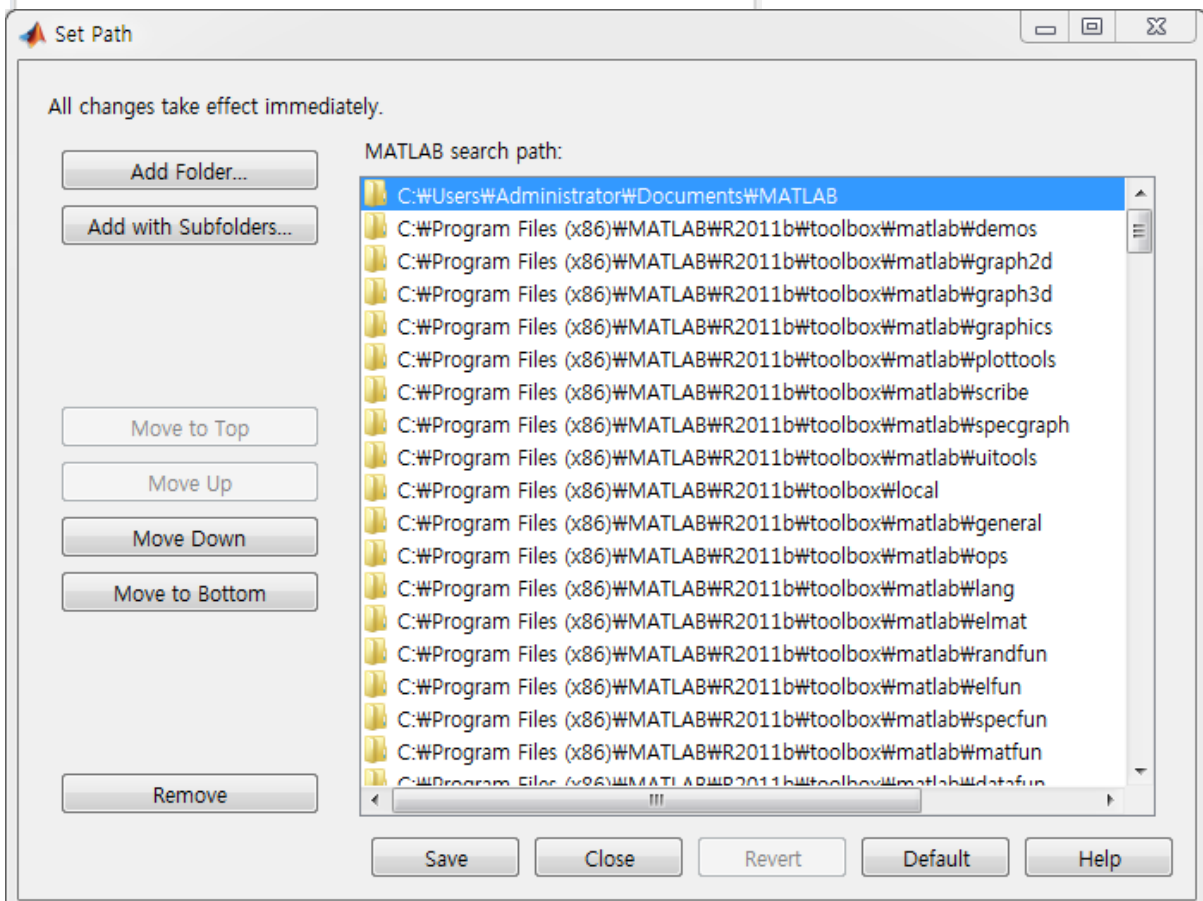
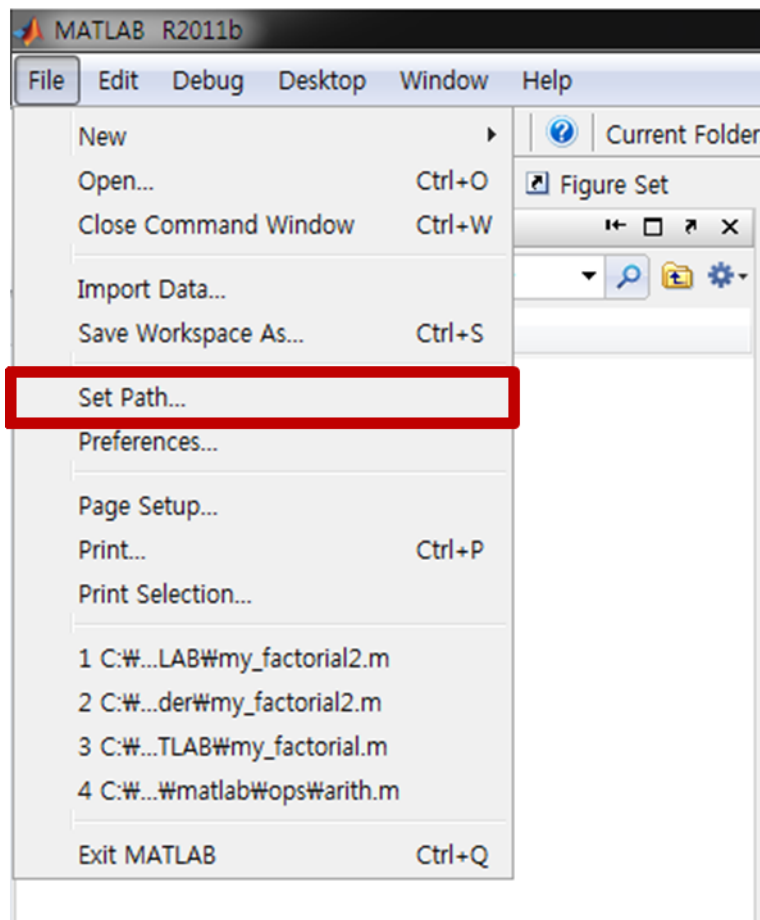
Script mode에서 더욱 간단하게 명령어를 실행시키는 방법이 있다. %를 하나만 사용했을 때는 주석인데, %%로 두 개를 이어서 쓰면 cell mode로 전환된다. Cell mode라는 것은 코드의 일부분을 간편하게 실행해보고 싶을 때 사용하는 기능으로, m-file로 저장하지 않아도 실행 가능하다. 커서가 있는 부분의 코드에 노란색 배경라운드가 생기는데, 이 때 `Ctrl + Enter`를 통해 그 부분만 실행할 수 있다.

	<p>%%를 쓰고 코드를 작성하니 코드 상에 구역이 나누어지며, 현재 커서가 있는 구역에 노란색 배경그라운드가 생긴다. Ctrl + Enter로 실행해보자.</p>
<p>n = 4</p>	<p>n = 4를 실행한 결과가 command window에 표시된다.</p>
	<p>커서를 이동하니 아래 구역에 백그라운드가 생긴다. 실행해보자.</p>
<p>y = 24</p>	<p>y = prod(1:n)을 실행한 결과가 command window에 표시된다.</p>

	<p>지금까지 두 개의 m-file을 작성하였다. 이 m-file은 current folder에 보면 나타나 있다. 사각형으로 표시된 부분은 현재 MATLAB이 가리키는 경로가 Documents > MATLAB이라는 뜻이다. 여기에 새 폴더를 만들어 들어가보자.</p>
---	---

	<p>My Folder라는 새 폴더를 만든 후, 들어가보았다. (더블클릭으로 경로 이동) 당연히 새 폴더이므로 아무 파일도 들어있지 않다. 이 상태에서 앞에서 만들었던 my_factorial을 실행해보자.</p>
<pre>>> my_factorial(4) ans = 24</pre>	<p>잘 실행된다. 그렇다면, 이번에는 상위 폴더인 MATLAB으로 되돌아가 my_factorial.m을 My Folder에 넣어보자. (드래그 앤 드롭으로 파일 이동)</p>
	<p>이제 MATLAB 폴더에는 my_factorial2.m만 있고, my_factorial.m은 My Folder에 들어있다. 이 상태에서 다시 my_factorial을 실행시켜보자.</p>
<pre>>> my_factorial(4) Undefined function 'my_factorial' for input arguments of type 'double'.</pre>	<p>이번에는 실행되지 않는다. 함수 my_factorial이 정의되어 있지 않다고 나온다. 다시 My Folder에 들어가보자.</p>
	<p>이 상태에서 my_factorial을 실행해보자. 잘 실행되는 것을 확인할 수 있다.</p>

위 과정을 통해 MATLAB이 인식하는 m-file의 범위를 알 수 있다. 먼저, current folder에 있는 m-file은 실행 가능하다. 그리고 MATLAB 폴더에 있는 m-file도 실행 가능하다. 그 외의 위치에 있는 m-file은 실행이 안 된다. 좀 더 자세히 설명하자면, current folder와 미리 지정된 path의 m-file 외에는 command window에서 실행할 수 없다. 지정된 path는 다음을 통해 확인할 수 있다.



Set path의 첫 번째 경로가 기본 위치인 MATLAB 폴더인 것을 알 수 있다. 그래서 MATLAB 폴더 내의 m-file은 실행이 되었던 것이다. 또한, 아래의 많은 경로는 MATLAB toolbox로, MATLAB에서 제공하는 함수가 current folder에 관계없이 실행되는 이유이다.

Set path는 필요할 때마다 폴더를 추가하기보다, 가능하면 건드리지 말고 current folder를 이동하여 작업하는 것이 좋다. 함수가 서로 겹쳐 프로그램이 꼬일 수 있기 때문이다. 각 프로젝트를 진행하면서 작성한 m-file은 따로 폴더로 정리하여 필요할 때마다 current folder를 이동하여 실행하는 것이 바람직하다고 생각한다.

3.2. 명령어의 흐름 제어

프로그래밍 언어라면 항상 등장하는 대표적인 흐름 제어 문법으로 if else, switch, for, while이 있다. MATLAB에서도 사용가능하며, command window에서도 사용할 수 있으나 대부분 m-file에서만 사용한다. 이 장에서는 각 문법의 사용법만 소개하며, 응용은 다루지 않는다.

① if else

if 논리식	대표적인 조건문 if else의 MATLAB 문법은 왼쪽과 같다. 참거짓으로 나타낼 수 있는 식이 논리식에 해당하며, 그 다음 줄부터 명령어를 이어나가면 된다. 논리식과 명령어 모두 괄호로 감싸지 않아도 잘 인식한다. 가운데 조건문인 elseif는 개수에 제한이 없으며, 조건문의 마지막에는 반드시 end가 들어가야 한다.
명령어	
elseif 논리식	
명령어	
else	
명령어	
end	

참(True)과 거짓(False)으로 나타나는 식이 논리식인데, MATLAB 상에서 거짓의 값은 0이며, 참은 0이 아닌 값이다. 즉, 반드시 AND나 OR로 이루어진 식이어야 하는 것은 아니다. 또한, 만약 논리식의 결과가 행렬이라면 행렬의 모든 요소가 0이 아니어야 참이다. 반대로, 하나라도 0인 요소가 있으면 거짓이다. 다음 표는 논리 연산자와 관계연산자를 정리한 것이다.

논리연산자		관계연산자	
&	Element-wise AND	<	~보다 작다
&&	Short-circuit AND	<=	~보다 작거나 같다
	Element-wise OR	>	~보다 크다
	Short-circuit OR	>=	~보다 크거나 같다
~	NOT	==	같다
xor(A, B)	Exclusive OR	~=	다르다

논리연산자에서 element-wise와 short-circuit의 차이는 행렬 연산과 스칼라 연산의 차이와 비슷하다. 앞에서 행렬과 스칼라 사이에 관계연산자를 사용했을 때($A > 3$), 참인 부분은 1, 거짓인

부분은 0인 행렬이 출력된 것을 보았을 것이다. 이와 마찬가지로, 행렬과 스칼라 사이에 element-wise 논리연산자를 사용하면 행렬의 각 요소마다 따로 연산하여 동일한 크기의 행렬이 출력된다. 같은 크기의 행렬 사이도 마찬가지이다. 각 위치마다 따로 연산하여 행렬을 출력한다. 그러나 서로 크기가 다른 행렬은 연산할 수 없다. 즉, 행렬 vs. 스칼라 또는 동일한 크기의 두 행렬 사이에서만 element-wise 논리연산자를 사용할 수 있다.

Short-circuit은 element-wise와 반대로 스칼라 vs. 스칼라만 가능한 연산자라고 생각하면 쉽다. 그러나 본디 short-circuit의 의미는 먼저 나온 연산자의 결과를 토대로 뒤의 연산을 생략하고 넘어가는 것을 말한다. 간단히 설명하면 $A \parallel B \&\& C \parallel D \dots$ 와 같이 길게 이어진 논리식이 있다고 가정할 때, 만약 A가 1이면 뒤의 B, C, D 등은 연산하지 않아도 전체가 1이라는 것을 알 수 있다. 왜냐하면 OR는 둘 중 하나만 1이면 전체가 1이기 때문이다. 비슷하게, A와 B가 동시에 0이면 C, D 등은 연산할 필요가 없다. 왜냐하면 AND는 둘 중 하나가 0이면 전체가 0이기 때문이다. 이와 같이, 먼저 나온 조건을 통해 끝까지 연산하지 않아도 최종결과를 알 수 있는 경우를 포착하여 뒤의 연산을 생략하는 기법을 short-circuit evaluation이라 하며, short-circuit 논리연산자는 그러한 알고리즘이 구현된 연산자이다.

② switch

<pre>switch 판별식 case 값1 명령어 case 값2 명령어 otherwise 명령어 end</pre>	<p>대표적인 분기문 switch의 MATLAB 문법은 왼쪽과 같다. 판별식의 결과 중 해당하는 값의 case가 있으면 그 아래의 명령어를 실행하며, C 언어와 달리 break는 쓰지 않는다. C 언어의 default에 해당하는 부분이 otherwise이며, if else문과 마찬가지로 end로 마친다.</p>
---	--

③ for

<pre>for 인덱스 = 벡터 명령어 end</pre>	<p>대표적인 반복문 for는 앞의 조건문, 분기문과 달리 사용법이 독특하다. 반복문이 실행될 때, 인덱스는 벡터의 값을 하나씩 취하며 명령어를 반복하는데, 벡터의 길이만큼 실행을 반복한다. 매 반복마다 인덱스는 벡터 내의 스칼라가 하나씩 대입된다. 물론 인덱스는 명령어 내에서 사용할 수 있다.</p>
-------------------------------------	---

예제3.3. 구구단 출력하기

1	<code>function</code> multiplication_table(k)
2	
3	<code>for</code> n = 1:10
4	disp([num2str(k) ' X ' num2str(n) ' = ' num2str(k*n)])

5	end	
<pre>>> multiplication_table(5) 5 X 1 = 5 5 X 2 = 10 5 X 3 = 15 5 X 4 = 20 5 X 5 = 25 5 X 6 = 30 5 X 7 = 35 5 X 8 = 40 5 X 9 = 45 5 X 10 = 50</pre>		<p>실행하면 command window 상에 구구단을 출력하는 함수이다. 입력 변수 k는 몇 단을 원하는지 입력 받고, k에 1부터 10까지 곱하여 차례대로 출력한다.</p> <p>3번째 줄의 for n = 1:10에서 1:10은 1 부터 10까지의 벡터를 나타내므로, n에 1, 2, 3, ..., 10까지 한 번씩 대입하여 for 루프 내부의 명령을 실행한다.</p> <p>4번째 줄의 함수 disp는 문자열을 command window에 출력하며, 함수 num2str은 숫자를 문자열로 변환한다. 문자열에 대한 자세한 사항은 부록에서 다룬다.</p>

④ while

while 논리식 명령어 end	반복문 while은 논리식이 참일 경우, 계속해서 명령어를 실행한다. C 언어와 큰 차이점은 없다.
-------------------------	---

3.3. Function Mode M-file의 특수한 기능

Script mode m-file은 사실상 command window에서 실행할 명령어를 모은 것과 동일하다. 그러나 function mode m-file은 MATLAB에서 입력과 출력이 있는 함수로 작동하기 때문에 sub-function, global variable, persistent variable, nargin, nargout과 같이 script mode에서는 잘 쓰이지 않지만 function mode에서는 매우 유용한 기능이 존재한다.

① Sub-function

각 m-file의 첫 줄에 정의된 함수를 primary-function이라고 하며, 이 함수의 이름이 m-file의 이름이 된다. 이 때, primary-function에서만 사용하기 위해 같은 m-file 내에 정의한 함수를 sub-function이라고 한다. 이 sub-function은 해당 m-file 내에서만 사용 가능하며, 다른 m-file에서 사용하려면 번거로운 작업을 거쳐야 한다. Sub-function을 정의하는 방법은 primary-function과 동일하며, 정의하는 부분 아래의 명령어는 sub-function에 대한 코드가 된다. (따로 함수의 범위를 지정해 줄 필요가 없다.)

예제3.4. Mean, Median, Mid-point		
1	function [avg, med, mid] = mean_median_midpoint(data)	Sub-function을 활용하여 작성한 m-file이다. 4번째 줄은 만약 data가 행렬이었을 때도 벡터일 때와
2		
3	% data가 행렬일 경우 벡터로 변환	
4	data = data(:);	

<pre> 5 num = length(data); 6 7 % sub-function 호출 8 avg = my_mean(data, num); 9 med = my_median(data, num); 10 mid = my_midpoint(data); 11 % ----- primary-function ----- 12 13 % ----- sub-function1 ----- 14 function m = my_mean(data, n) 15 m = sum(data)/n; 16 % ----- sub-function2 ----- 17 function m = my_median(data, n) 18 data_ascend = sort(data); 19 if rem(n, 2) % 요소 수가 홀수 20 m = data_ascend((n+1)/2); 21 else % 요소 수가 짝수 22 m = (data_ascend(n/2) + data_ascend(n/2+1))/2; 23 end 24 % ----- sub-function3 ----- 25 function m = my_midpoint(data) 26 m = (max(data) + min(data))/2; </pre>	<p>동일하게 전체의 평균값, 중간값, 중앙값을 계산하기 위해 벡터로 변환하는 명령이다. 변수가 벡터일 경우, 아무 변화가 없을 것이다. 다음 줄의 length는 벡터의 크기를 구하는 함수이다.</p> <p>전체 요소의 합을 구한 후, 요소의 개수로 나누어 평균을 계산한다.</p> <p>요소를 차례대로 나열하였을 때 가운데에 위치하는 값을 구하기 위해 먼저 정렬한다. (함수 sort) 그리고 만약 전체 개수가 홀수인 경우 한 가운데의 값을 출력하며, 짝수일 경우 가운데의 두 값의 평균을 출력한다. (함수 rem)</p> <p>최댓값과 최솟값의 평균을 구한다.</p>
<pre> >> [avg, med, mid] =mean_median_midpoint([9 1 4 2 6]) avg = 4.4000 med = 4 mid = 5 </pre>	<p>[9 1 4 2 6]의 5개 요소를 넣어 작동하는지 확인한다.</p> <p>평균은 22/5로, 4.4이다.</p> <p>차례대로 정렬하면 1 2 4 6 9이므로, 가운데에 위치한 요소는 4이다.</p> <p>최댓값 9와 최솟값 1의 평균은 5이다.</p>

② Global, Persistent Variables

MATLAB의 함수는 각기 독립적인 local workspace를 가지고 있다. Sub-function도 마찬가지이다. 이는 script mode m-file을 실행했을 때 내부에서 사용되는 모든 변수가 workspace 창에 나타나지만, function mode m-file은 출력 결과만 내보내는 이유이다. Command window에서 작동하는 workspace를 local workspace와 구별하여 base workspace라고 부른다. 그러나 서로 연관된 함수를 작성하다 보면 다른 workspace에서 변수를 공유해야 할 때가 있으며, 이 때

원하는 변수에 global 속성을 부여하면 해당 변수는 global로 선언되어 있는 모든 workspace에서 공유하게 된다.

예제3.5. Mean, Median, Mid-point Global		
1	<code>function [avg, med, mid] = mean_median_midpoint(data)</code>	예제3.4에서 벡터의 길이를 함수 인자 대신 global로 전달하도록 수정한 것이다.
2		
3	<code>% global 변수로 선언</code>	변수를 global로 사용할 때는 사용하기 전에 반드시 global이라고 선언해야 한다. 그래서 각 함수의 정의 다음에 global 선언을 하는 경우가 대부분이다.
4	<code>global n</code>	
5	<code>% data가 행렬일 경우 벡터로 변환</code>	
6	<code>data = data(:);</code>	
7	<code>n = length(data);</code>	
8		
9	<code>% sub-function 호출</code>	
10	<code>avg = my_mean(data);</code>	
11	<code>med = my_median(data);</code>	
12	<code>mid = my_midpoint(data);</code>	
13	<code>% ----- primary-function -----</code>	동일한 변수 명으로 global 선언을 한 workspace에서는 모두 해당 변수를 불러올 수 있다. 예를 들어, 이 예제의 m-file에서 선언된 global n은 기본적으로 base workspace에 나타나지 않지만, 함수를 실행하기 전에 command window에서 global n을 입력하면 함수가 실행된 후 workspace의 n이라는 변수에 data의 길이가 저장되어 있을 것이다.
14		
15	<code>% ----- sub-function1 -----</code>	
16	<code>function m = my_mean(data)</code>	
17	<code>global n</code>	
18	<code>m = sum(data)/n;</code>	
19	<code>% ----- sub-function2 -----</code>	
20	<code>function m = my_median(data)</code>	
21	<code>global n</code>	
22	<code>data_ascend = sort(data);</code>	
23	<code>if rem(n, 2) % 요소 수가 홀수</code>	함수의 실행 결과는 예제3.4와 동일하다.
24	<code> m = data_ascend((n+1)/2);</code>	
25	<code>else % 요소 수가 짝수</code>	
26	<code> m = (data_ascend(n/2) + data_ascend(n/2+1))/2;</code>	
27	<code>end</code>	
28	<code>% ----- sub-function3 -----</code>	
29	<code>function m = my_midpoint(data)</code>	
30	<code>m = (max(data) + min(data))/2;</code>	

C 언어의 static 변수와 비슷한 변수로, MATLAB에는 persistent라는 것이 있다. 이 변수는 function mode m-file에서만 사용가능하며, global과 같이 persistent 선언이 되어있는 모든 함수에서 접근 가능하다. 특징은 함수가 종료될 때도 변수가 사라지지 않으며, 다른 함수가 실행될 때 이전 값을 유지한다는 것이다.

예제3.6. 누적값		
1	<code>function s = accum(value)</code>	<p>실행할 때마다 지금까지 입력한 모든 값의 누적 합을 출력해주는 함수이다.</p> <p>처음에 persistent 선언을 하면 MATLAB은 해당 변수 명에 빈 행렬을 넣어둔다. 이는 global 선언도 마찬가지이다. 그래서 일반적으로 isempty를 이용하여 빈 행렬일 경우 초기화 해주는 방식이 사용된다.</p>
2		
3	<code>persistent sumValue</code>	
4	<code>if isempty(sumValue)</code>	
5	<code> sumValue = 0;</code>	
6	<code>end</code>	
7		
8	<code>sumValue = sumValue + value;</code>	
9	<code>s = sumValue;</code>	
<pre>>> accum(1) ans = 1 >> accum(5) ans = 6 >> accum(10) ans = 16 >> clear accum >> accum(5) ans = 5</pre>		<p>첫 번째 실행에서는 초기화가 일어나 입력한 값 그대로 출력한다.</p> <p>두 번째 실행부터, 지금까지 입력했던 모든 값을 누적하여 출력한다.</p> <p>Clear 명령을 이용하여 해당 함수의 workspace를 지워주면 다시 초기화가 일어난다. M-file을 다시 저장하거나 MATLAB을 종료하고 다시 실행해도 초기화가 일어난다.</p>

③ nargin, nargout

도움말 기능을 이용하여 MATLAB 함수를 살펴보면 대부분이 입력 변수나 출력 변수의 수에 따라 다르게 작동한다는 것을 알 수 있을 것이다. 간단한 sum만 해도, 입력 변수의 수에 따라 열 방향으로만 더하다가 행 방향으로 더할 수 있게 설정할 수 있다. 이와 같은 설정을 할 때

사용하는 것이 nargin과 nargs이다. 각각 'number of input arguments'와 'number of output arguments'의 약자이다. 말 그대로, 함수가 실행되었을 때 전달받은 입력 인자와 출력해야 할 인자의 수가 저장된 변수이다. 모든 함수에서 자동으로 생성되며, 조건문을 통해 인자의 수에 따라 다르게 실행되도록 m-file을 작성할 수 있다.

예제3.7. 조건에 맞는 요소의 개수		
1	<code>function [row, col, total] = threshold_counter(data, thres)</code>	<p>입력 변수 data의 요소 중 thres보다 큰 값의 수를 행과 열에 따라 몇 개인지 세는 함수이다. 행렬에 관계 연산자를 사용하면 참인 요소의 위치에 1, 거짓인 요소의 위치에 0을 대입하므로, 1인 위치를 더하면 개수를 셀 수 있을 것이다.</p> <p>3번째 줄에서 nargin == 1은 입력 인자가 하나로, 변수 thres를 전달받지 못했다는 의미이므로 0으로 default 값을 지정한다. (없으면 thres를 전달받지 못했을 때 에러 발생)</p>
2		
3	<code>if nargin == 1 % 입력 인자가 1개일 경우</code>	
4	<code>thres = 0;</code>	
5	<code>end</code>	
6		
7	<code>row = sum(data > thres, 2); % 행 방향으로 세기</code>	
8	<code>col = sum(data > thres, 1); % 열 방향으로 세기</code>	
9	<code>total = sum(col); % 전체 세기</code>	
10		
11	<code>if nargsout <= 1 % 출력 인자가 1개 또는 0개일 경우</code>	
12	<code>row = total;</code>	
13	<code>end</code>	
<pre>>> data = [9 -2 4; -1 -7 3]; >> threshold_counter(data) ans = 3 >> total = threshold_counter(data) total = 3 >> [row, col] = threshold_counter(data) row = 2 1 col =</pre>		<p>11번째 줄에서 nargsout <= 1은 출력 인자가 하나 또는 없다는 뜻인데, 이는 함수의 출력 정의에서 첫 번째인 row만 반환된다는 것을 의미한다. (출력인자가 없을 경우, 자동으로 ans에 저장) 따라서, row에 total을 대입하면 전체를 센 개수가 출력될 것이다. 만약 nargsout == 2라면 row와 col이 반환되고, nargsout == 3이라면 전부 다 반환되니 다른 값은 처리해주지 않아도 된다.</p> <p>아래는 m-file을 저장한 후 command window에서 실행한</p>

<pre> 1 0 2 >> [row, col, total] = threshold_counter(data) row = 2 1 col = 1 0 2 total = 3 </pre>	<p>예이다. 변수 data에 [9 -2 4; -1 -7 3]을 저장한 후 함수의 첫 번째 인자로 준다. 두 번째 인자는 없으므로, thres는 0으로 설정된다. 그러면 행렬에서 0보다 큰 요소의 수는 3개이므로, 출력 인자가 없을 때와 한 개일 때 모두 3이 출력된다.</p> <p>출력 인자가 두 개일 때는 행에 따라 더한 row와 열에 따라 더한 col이 출력되며, 세 개일 때는 total도 출력된다.</p>
---	--

입력 인자의 개수인 nargin은 전달받지 못한 인자에 대하여 default 값을 설정하는 용도로 자주 쓰이며, 출력 인자의 개수인 nargout은 출력 인자의 수에 따라 적절한 데이터를 출력해주는 용도로 쓰인다.

3.4. 기타 M-file 작성시 자주 쓰이는 함수

MATLAB m-file을 작성할 때 유용하게 쓸 수 있는 함수의 리스트이다. 기능은 간단하게만 설명하였고, 자세한 사용법은 직접 도움말 기능을 통해 검색하길 바란다.

함수 이름	기능 설명
rem	나눈 나머지
fix	정수 부분 추출 (소수 부분 지우기)
sort	정렬
input	문자열 입력 (scanf와 유사)
menu	버튼 선택 메뉴
msgbox	메시지 상자
disp	문자열 출력 (printf와 유사)
error	에러 메시지 출력 후 함수 종료
warning	경고 메시지 출력
isempty	빈 행렬 확인
waitbar	반복문 루프 진행상황 확인