

# 2강 - 프로그래밍 & MATLAB 기초

# 목차

- MATLAB의 목적 + 기능
- Primitive data type
- Vector / Matrix
- Object-oriented programming / Procedural programming

# MATLAB의 기능

- Engineering 분야에서 가장 많이 사용되는 언어 중 하나
  - C / C++ / MATLAB
- Why?
  - Matrix calculation
  - Data visualization
  - Easy to use
  - Popular algorithms are already implemented
- i.e. 빠른 공학 계산 + 정확한 공학 계산

# MATLAB의 성능

- C / C++ 보다 느림
- Python 보다 느림
- 임베디드 컴퓨터에서 실행 불가능
- 엄청 비쌈
- ... Then why?
  - Easy to use
  - Built-in algorithms
  - 이 두개가 있기 때문에 MATLAB을 실행시킨 순간부터 실험 가능
    - C/C++, Python은 불가능

# MATLAB + 태완's project

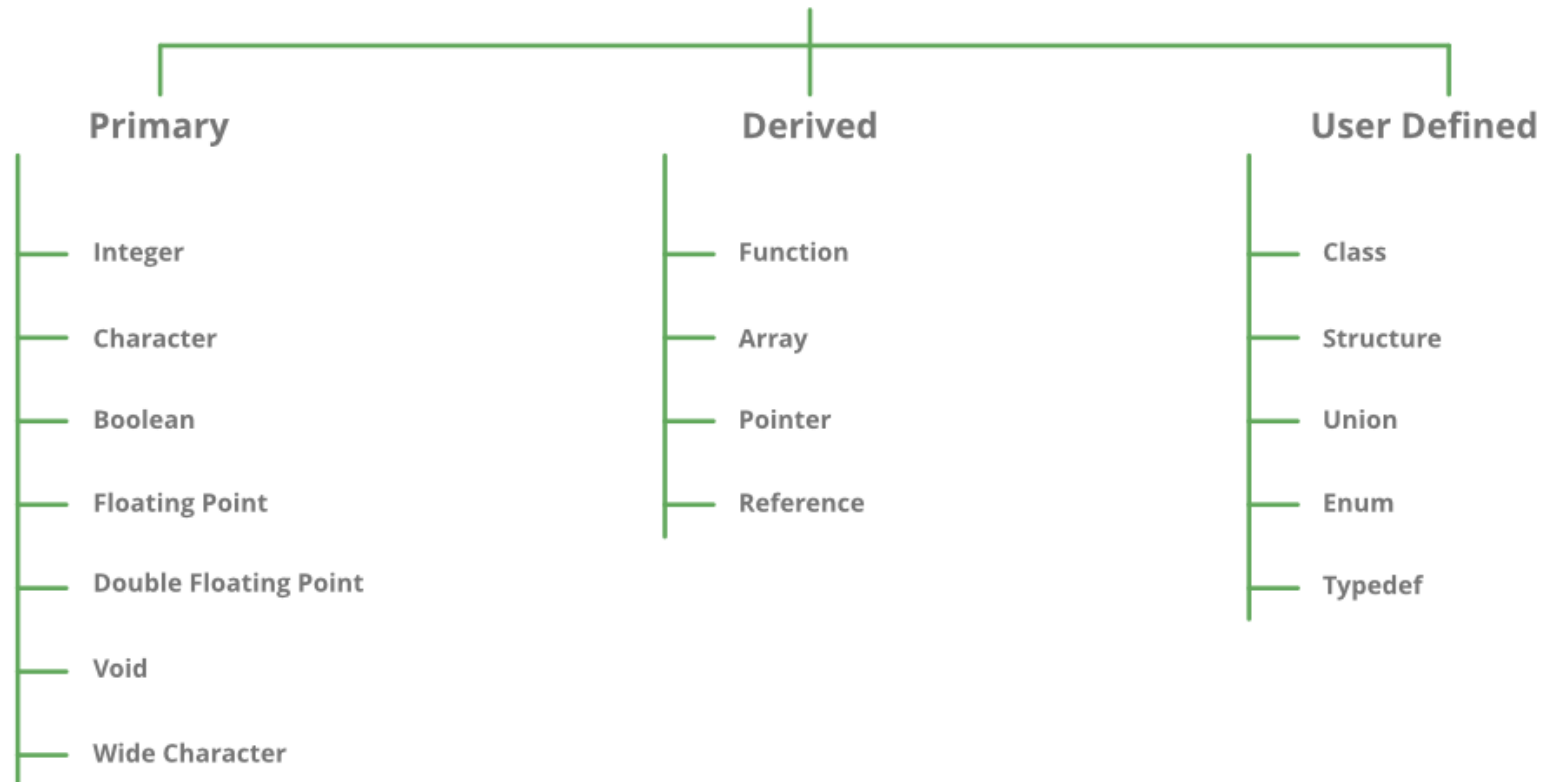
- MATLAB으로 Chip과의 인터페이스가 있다는 것
- == MATLAB에서 계산한 정보를 Chip으로 넘길 수 있다는 것
- == Chip을 내가 원하는대로 통제할 수 있다는 것
- == 3D Printer as I please

# MATLAB 공부 과정

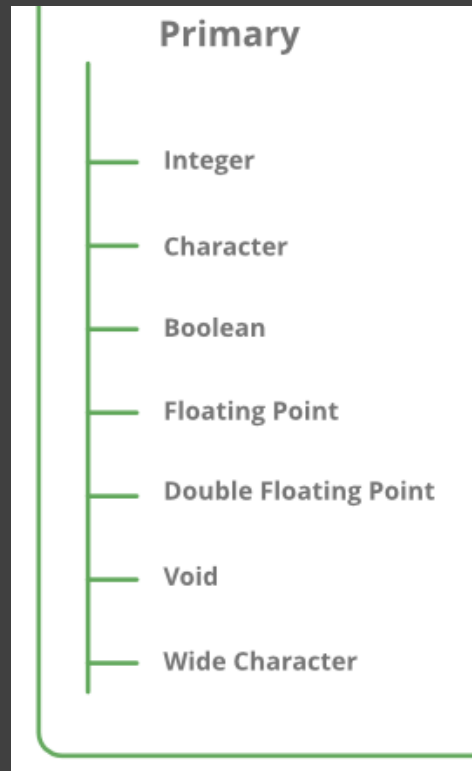
- Primitive data type handling
- Matrix data handling
- Boolean and conditional operations
- For loops
- While loops
- Plotting and data visualization

# Primitive data types

## DataTypes in C / C++



# Primitive data types



구분	자료형	범위	바이트
정수형	char	-128 ~ 127	1(8)
	unsigned char	0 ~ 255	1(8)
	short	-32768 ~ 32767	2(16)
	int	-2,147,483,648 ~ 2,147,483,647	4(32)
	long	-2,147,483,648 ~ 2,147,483,647	4(32)
	unsigned short	0~65535	2(16)
	unsigned int	0~4,294,967,295	4(32)
	unsigned long	0~4,294,967,295	4(32)
실수형	float	$8.4 \times 10^{-37} \sim 3.4 \times 10^{38}$	4(32)
	double	$2.2 \times 10^{-308} \sim 1.8 \times 10^{308}$	8(64)



# Primitive data types

- 이렇게 왜 중요한가?
  - MATLAB에서는 안중요함 (ㅋㅋ)
  - C/C++ 에서는 중요함

구분	자료형	범위	바이트
정수형	char	-128 ~ 127	1(8)
	unsigned char	0 ~ 255	1(8)
	short	-32768 ~ 32767	2(16)
	int	-2,147,483,648 ~ 2,147,483,647	4(32)
	long	-2,147,483,648 ~ 2,147,483,647	4(32)
	unsigned short	0~65535	2(16)
	unsigned int	0~4,294,967,295	4(32)
	unsigned long	0~4,294,967,295	4(32)
실수형	float	$8.4 \times 10^{-37} \sim 3.4 \times 10^{38}$	4(32)
	double	$2.2 \times 10^{-308} \sim 1.8 \times 10^{308}$	8(64)

# Primitive data types

- 메모리 공간은 단순히 0/1 을 저장할 수 있는 컨테이너임
  - 근데 그게 엄청나게 많이 이어짐
- 우리가 데이터를 저장할때는
  - 데이터의 값 (i.e. 숫자 값)을 메모리에 저장하고
  - 동시에 데이터의 타입과 위치를 저장함.
    - 메모리 위치의 개념은 나중에 pointer라는 개념으로 배울거임



0	1	0	0	0	1	0	0	0	0	0	0	1	1	1	1	1	0	0	0	0	0	1	1	1	1	1	0	0	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

# Primitive data types

- C/C++에서는 매 데이터마다 type을 정확하게 적어줘야함.

```
#include <iostream>
#include <string>

using namespace std;

int main()
{
    int a = 2;
    float b = 1.5f;
    double c = 3.1234;
    char d = 'f';
    string e = "string";

    std::cout << a << std::endl;
    std::cout << b << std::endl;
    std::cout << c << std::endl;
    std::cout << d << std::endl;
    std::cout << e << std::endl;
}
```

```
2
1.5
3.1234
f
string
```

# Primitive data types

- 데이터 타입을 안적으면 에러가 남.





```
1 // Example program
2 #include <iostream>
3 #include <string>
4
5 using namespace std;
6
7 int main()
8 {
9     a = 2;
10    b = 1.5f;
11    c = 3.1234;
12    d = 'f';
13    e = "string";
14
15    std::cout << a << std::endl;
16    std::cout << b << std::endl;
17    std::cout << c << std::endl;
18    std::cout << d << std::endl;
19    std::cout << e << std::endl;
20 }
21
```

abc	E0020	identifier "a" is undefined
abc	E0020	identifier "b" is undefined
abc	E0020	identifier "c" is undefined
abc	E0020	identifier "d" is undefined
abc	E0020	identifier "e" is undefined

# Primitive data types

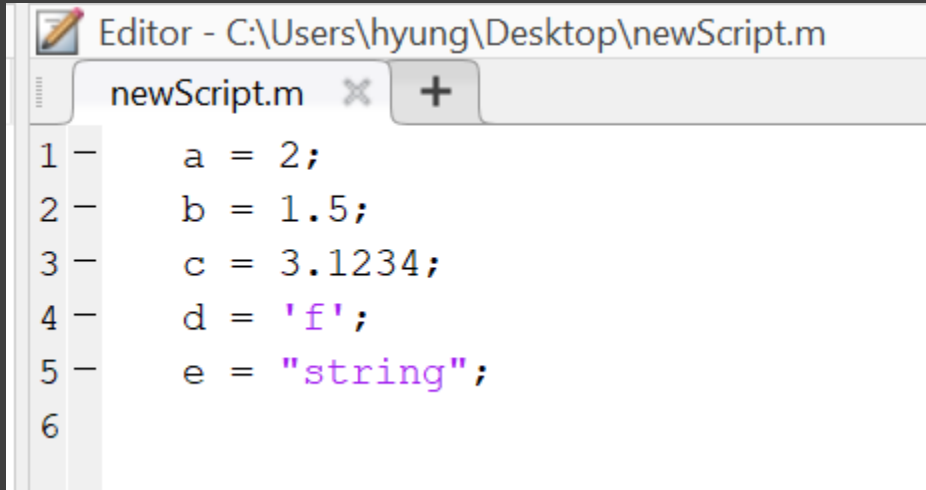
- 데이터 타입을 잘못 적어도 에러가 남.

```
1 // Example program
2 #include <iostream>
3 #include <string>
4
5 using namespace std;
6
7 int main()
8 {
9     char a = 2;
10    string b = 1.5f;
11    int c = 3.1234;
12    float d = 'f';
13    int e = "string";
14
15    std::cout << a << std::endl;
16    std::cout << b << std::endl;
17    std::cout << c << std::endl;
18    std::cout << d << std::endl;
19    std::cout << e << std::endl;
20 }
21
```






 E0415	no suitable constructor exists to convert from "float" to "std::basic_string<char, std::char_traits<char>, std::allocator<char>>"
 E0144	a value of type "const char *" cannot be used to initialize an entity of type "double"
 C2440	'initializing': cannot convert from 'float' to 'std::basic_string<char, std::char_traits<char>, std::allocator<char>>'
 C4244	'initializing': conversion from 'double' to 'int', possible loss of data

# Primitive data types

- MATLAB에서는 안적어도 에러가 안남 ㅎㅎ
  - 근데 int, float를 안 만들고 전부 double로 만듦



```
Editor - C:\Users\hyung\Desktop\newScript.m
newScript.m
1 a = 2;
2 b = 1.5;
3 c = 3.1234;
4 d = 'f';
5 e = 'string';
6
```

Name ^	Value	Class
 a	2	double
 b	1.5000	double
 c	3.1234	double
 d	'f'	char
 e	"string"	string

# Primitive data types

- 왜 MATLAB에서는 double이 기본적으로 쓰는걸까?
  - int 자료형은 double 자료형보다 적은 메모리를 차지하고 더 빠르게 읽을 수 있지만, 소숫점을 표현하지 못한다.
  - 공학 계산에서는 소숫점 계산이 많음.
  - 혹시나 모를 계산 실수를 대비해서 MATLAB에서는 double이 기본적으로 사용됨.
    - 하지만 그래서 속도도 느림.

```
double a = 2.5;  
double b = 1.0;  
  
double c = a / b;  
  
std::cout << c << std::endl;
```

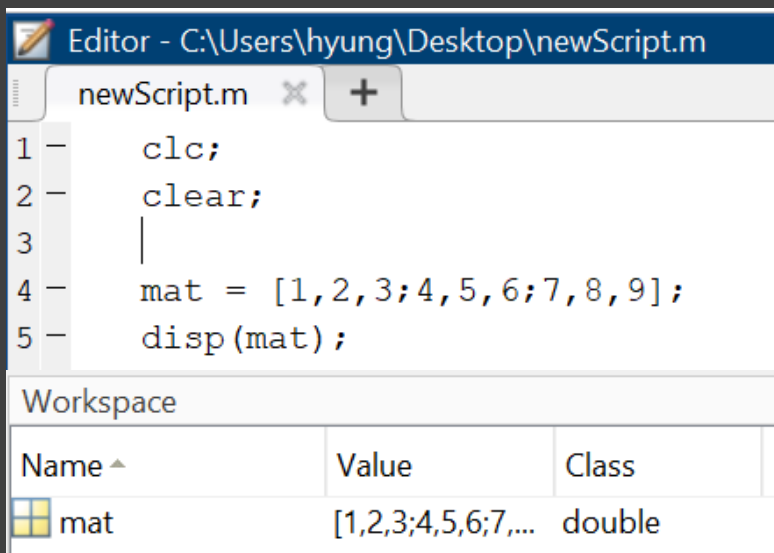
2.5

```
int a = 2.5;  
int b = 1.0;  
  
int c = a / b;  
  
std::cout << c << std::endl;
```

2

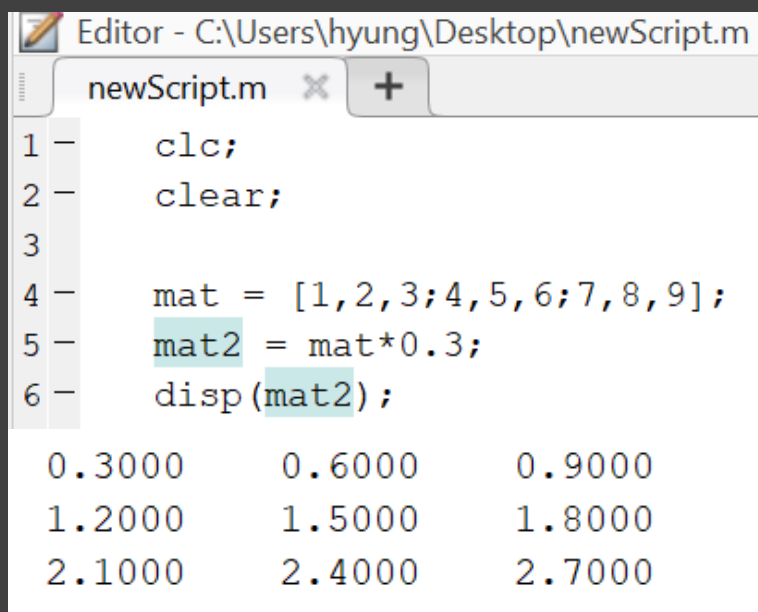
# Primitive data types

- 심지어 정수(int)만 있는 매트릭스를 만들어도 다 double로 채워넣음
  - 이라서 계속 매트릭스 메모리를 할당할 때 속도가 느림
  - 하지만 아래처럼 매트릭스 계산을 쉽게 할 수 있음



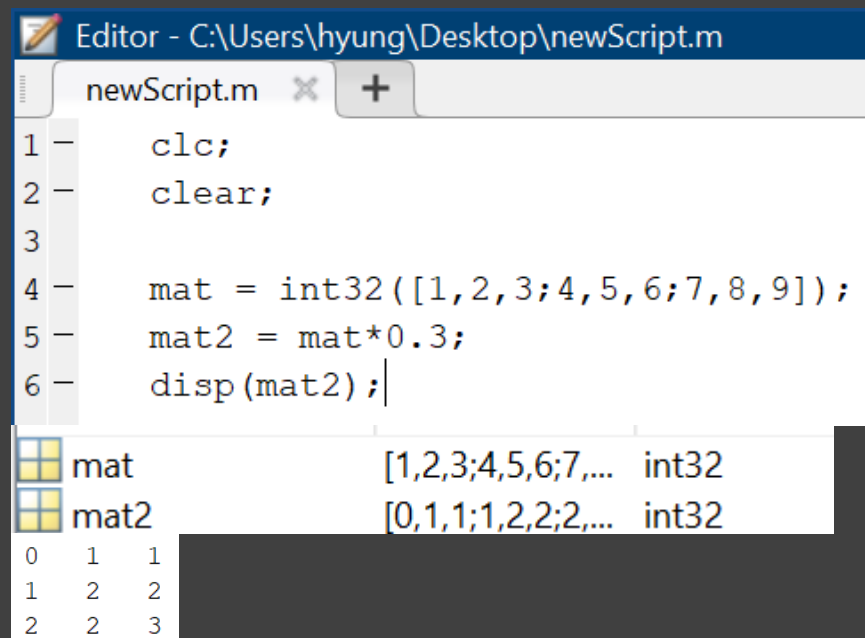
```
Editor - C:\Users\hyung\Desktop\newScript.m
newScript.m
1 - clc;
2 - clear;
3 -
4 - mat = [1,2,3;4,5,6;7,8,9];
5 - disp(mat);
```

Name ^	Value	Class
mat	[1,2,3;4,5,6;7,8,9]	double



```
Editor - C:\Users\hyung\Desktop\newScript.m
newScript.m
1 - clc;
2 - clear;
3 -
4 - mat = [1,2,3;4,5,6;7,8,9];
5 - mat2 = mat*0.3;
6 - disp(mat2);
```

0.3000	0.6000	0.9000
1.2000	1.5000	1.8000
2.1000	2.4000	2.7000



```
Editor - C:\Users\hyung\Desktop\newScript.m
newScript.m
1 - clc;
2 - clear;
3 -
4 - mat = int32([1,2,3;4,5,6;7,8,9]);
5 - mat2 = mat*0.3;
6 - disp(mat2);
```

mat	[1,2,3;4,5,6;7,8,9]	int32
mat2	[0,1,1;1,2,2;2,2,3]	int32



# Vector / Matrix?

- In physics, a **vector** holds magnitude + direction...
- In mathematics, **vectors** form a **vector space**...
- In computer science, a **vector** is a container that holds **values of the same data type**

# Vector / Matrix?

- There are a few different containers in C++
  - Vectors
  - Arrays
  - Maps
  - Lists
  - Deque...
- We'll learn these later (when we actually do C++)

# Vector / Matrix?

- In maths, a **matrix** is a linear transform of a **vector** space to another (?)
- In computer science, a **matrix** is a **vector** of a **vector**
  - In C++, we write `std::vector<std::vector<int>>`
  - In MATLAB, we write `mat[x,y];`
- For now, let's ignore the difficult way of C++.

# Vector / Matrix?

- MATLAB lets you see a matrix as how we learnt in maths.

```
mat = zeros(10,10);  
  
for r = 1:10  
    for c = 1:10  
        mat(r,c) = randi([1 100]);  
    end  
end
```

	1	2	3	4	5	6	7	8	9	10
1	47	90	41	66	96	31	73	76	19	57
2	7	69	67	14	90	17	55	4	82	60
3	41	65	37	100	71	20	83	31	53	63
4	1	78	22	21	80	27	84	100	40	90
5	54	9	42	2	91	38	2	51	45	86
6	51	12	80	94	86	16	9	63	10	4
7	33	65	59	68	80	39	8	44	79	31
8	74	11	18	93	79	78	88	91	94	41
9	29	77	95	28	38	28	48	97	11	55
10	31	16	43	33	56	19	22	57	91	70

# Object-oriented programming / Procedural programming

- OOP - C++ / Python
- Procedural - C / MATLAB

# Object-oriented programming / Procedural programming

PROCEDURAL ORIENTED PROGRAMMING	OBJECT ORIENTED PROGRAMMING
In procedural programming, program is divided into small parts called <b>functions</b> .	In object oriented programming, program is divided into small parts called <b>objects</b> .
Procedural programming follows <b>top down approach</b> .	Object oriented programming follows <b>bottom up approach</b> .
There is no access specifier in procedural programming.	Object oriented programming have access specifiers like private, public, protected etc.

Adding new data and function is not easy.	Adding new data and function is easy.
Procedural programming does not have any proper way for hiding data so it is <b>less secure</b> .	Object oriented programming provides data hiding so it is <b>more secure</b> .
In procedural programming, overloading is not possible.	Overloading is possible in object oriented programming.
In procedural programming, function is more important than data.	In object oriented programming, data is more important than function.
Procedural programming is based on <b>unreal world</b> .	Object oriented programming is based on <b>real world</b> .

# Object-oriented programming / Procedural programming

PROCEDURAL ORIENTED PROGRAMMING	OBJECT ORIENTED PROGRAMMING
In procedural programming, program is divided into small parts called <b>functions</b> .	In object oriented programming, program is divided into small parts called <b>objects</b> .
Procedural programming follows <b>top down approach</b> .	Object oriented programming follows <b>bottom up approach</b> .
There is no access specifier in procedural programming.	Object oriented programming have access specifiers like private, public, protected etc.

Adding new data and function is not easy.	Adding new data and function is easy.
Procedural programming does not have any proper way for hiding data so it is <b>less secure</b> .	Object oriented programming provides data hiding so it is <b>more secure</b> .
In procedural programming, overloading is not possible.	Overloading is possible in object oriented programming.
In procedural programming, function is more important than data.	In object oriented programming, data is more important than function.
Procedural programming is based on <b>unreal world</b> .	Object oriented programming is based on <b>real world</b> .