MATLAB Basics
As a Programming Language

Variables / Arrays

- First of all: **Every variable in MATLAB is an array!**
  - An array has two dimensions by default (so it's like a matrix), but it is easy to create arrays of more dimensions.
- The first two dimensions represent **rows** and **columns**.
- A **scalar** is just an array with a single element.
- A **vector** is an array with at most one **non-singleton** dimension.
- An array element can be
  - A single value ("normal" arrays)
  - Another array (more complicated; to be discussed later)

Variables / Arrays

- MATLAB variables are created and deleted dynamically.
- Variable names are case-sensitive.
- To create a variable: Just assign something to it.
- List initialization / specification of an array:
  - Enclose the element values in `[...]`, using semicolon (`;`) to separate rows.
- Specify an empty array using `[]`.
- Ways to free the memory used by an array:
  - Setting it to an empty array; the variable name remains valid.
  - Statement **`clear`**; the variable name is deleted.

Data Types

- Basic numerical data types: **`double`** (the default), **`single`**, **`int8`**, **`uint8`**, **`int16`**, **`uint16`**, **`int32`**, **`uint32`**
- Complex numbers (**`complex`**)
- Logical data type (**`logical`**) with values **`true`** and **`false`**
- Character data type (**`char`**; to be discussed later)
- Note: All the data types are actually **classes** (like in OOP).
- Advanced and user-defined data types (to be discussed later)
- Functions related to data types: **`class`**, **`isa`**, **`logical`**, **`islogical`**
- Type casting (conversion between data types)

Allocation and Initialization

- Some functions for allocating arrays: **`zeros`**, **`ones`**, **`eye`**, **`true`**, **`false`**, **`rand`**
  - Specification of sizes / dimensions / data type (when applicable)
- Functions for getting array dimensions and sizes: **`size`**, **`length`**, **`numel`**, **`isempty`**
- Initializing vectors with evenly-spaced values:
  - **`a:b`** and **`a:c:b`** expressions
  - Function **`linspace`**

Array Element Indexing

- **Important: Array indices are 1-based in MATLAB.**
- Access and assignment of a sub-array based on element locations:
  - Format: `A(v1,v2,v3,...)`
  - Each dimension is specified by a vector (which can be a scalar) that represents the **subscripts** for that dimension.
  - The use of a single colon (`:`) to index a dimension:
  - The use of keyword `end` in array indices:

Array Element Indexing

```
A = [ 15 23 8; 7 11 14 ]
```

A MATLAB array in the memory:

Header	15	7	23	11	8	14
Subscript #1 (row):	1	2	1	2	1	2
Subscript #2 (column):	1	1	2	2	3	3
Linear index:	1	2	3	4	5	6

- **Linear index** corresponds to the arrangement / ordering of array elements in the memory.
- For ordering elements of multi-dimensional arrays, MATLAB uses "**column-major**", while C uses "**row-major**".

Array Element Indexing

- Use **`A(:)`** to convert an array to a **column vector** (along the first dimension).
- Conversion between subscripts and linear indices: Functions **`sub2ind`**, **`ind2sub`**
- Related functions that change "array shapes" while preserving the array data arrangement:
  - Function **`squeeze`**
  - Function **`reshape`**
  - Function **`permute`** (not preserving memory ordering)
- **`A(B)`** type array access (both **`A`** and **`B`** are arrays), with **`B`** containing positive integers that are **linear indices** into **`A`**.

Adding and Deleting Rows/Columns

- Adding rows/columns:
  - Just assign elements at indices beyond the current size. (New memory block has to be allocated and data copied, so this can affect efficiency.)
  - When possible, use pre-allocation
- Deleting rows/columns:
  - Assign the deleted rows/columns to empty array
- Concatenation
  - Along columns and rows
  - Beyond the first two dimensions: Function **`cat`**
- Repeating an array: Function **`repmat`**

Text Output

- Just type a variable name or an expression:
  - Or the variable name followed by assignment to it.
  - Just give an expression without a variable name: Assignment is made to variable **`ans`**
  - End the statement with a semicolon (`;`) to suppress such text outputs.
- Function **`disp`** shows the value of an expression without the variable name, and is therefore more compact.
- C-style formatted output: function **`fprintf`**
  - **`fprintf('Size of A is %dx%d\n', size(A));`**
  - C-style escape sequences ok, (**`'\n'`**, **`'\t'`**, etc.)
  - The format string is repeated if there are more values to print out.

Basic Operators

scalar operator scalar

- Arithmetic operations: `+`, `-`, `*`, `/`, `^`; function: `mod`
- Relational operators: `>`, `>=`, `<`, `<=`, `==`, `~=`
- Logical operators: `~`(unary), `&`, `&&`, `|`, `||`
  - `&&` and `||` are preferred over `&` and `|` for efficiency when the conditions are scalars (skipping unnecessary evaluations as in C, often called short-circuit operators).

Basic Operators

array operator scalar

- Result is an array.
- Operation is between each array element and the scalar.
- Arithmetic operations: `+`, `-`, `*`, `/`, `.^`; function: `mod`
  - `^`: This is a matrix operation (only valid if the array is a square matrix).
  - `.^`: This is for element-wise power computation.
- Relational operators: `>`, `>=`, `<`, `<=`, `==`, `~=`
- Logical operators: `~`(unary), `&`, `|`

Basic Operators

scalar operator array

- Result is an array.
- Operation is between each array element and the scalar.
- Arithmetic operations: `+`, `-`, `*`, `^`, `./`; function: `mod`
  - `/`: Cannot be used in this case. (Use `./` instead.)
- Relational operators: `>`, `>=`, `<`, `<=`, `==`, `~=`
- Logical operators: `~`(unary), `&`, `|`

Basic Operators

array operator array

- Element-wise operations: The two arrays must have the same size, or can be broadcasted to the same size. Result is an array.
  - Numerical operations require that the two arrays to be of the same type (unless one is a scalar double).
  - Arithmetic operations: `+`, `-`, `.*`, `./`, `.^`; function: `mod`
  - Relational operators: `>`, `>=`, `<`, `<=`, `==`, `~=`
  - Logical operators: `~`(unary), `&`, `|`
- Array size broadcasting: Enlarge the two arrays to the same size (using `repmat` internally), if applicable.
  - Use with care; unexpected things can happen if this happens automatically without you knowing it.