KU LEUVEN

MATLAB

Fundamental Data Types: basics

Variables

KU LEUVEN

Variables

MATLAB variable is a tag that you assign to a value while that value remains in memory.

- Tag = reference to the value in memory
  Named container, held by the computer in a memory location.
  - programs can read it,
  - operate on it with other data,
  - save it back to memory.

Variables: why?

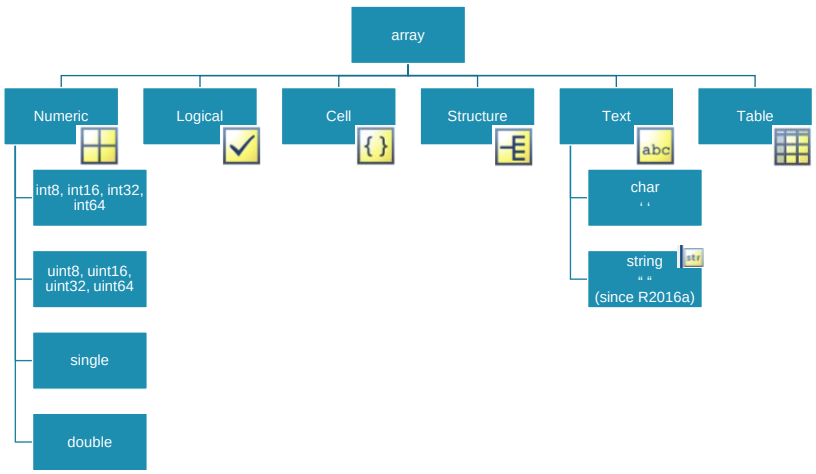
- Avoid recomputing a value that is used repeatedly.
  - Example: if you need e
  - compute it once and save the result.
  - `>> e = exp(1)`
- Make the connection between the code and the underlying mathematics more apparent. If you are computing the area of a circle, you might want to use a variable named r:
  - `>> r = 3`
  - `>> area = pi * r^2`
- Often it is better to break a long computation into a sequence of steps and assign intermediate results to variables.
- *Downey, Physical Modeling in Matlab*

Variables: naming

- **Naming Variables**
  - Must begin with a letter, followed by any combination of letters, digits, and underscores.
  - Case sensitive
    `A` is not the same variable as `a`
  - Names can be of any length
    MATLAB uses only the first N characters of the name, (where N is the number returned by the function `namelengthmax`)
  - Check with `isvarname`
- **Avoid Using Function Names for Variables**
  - Make sure you are not using a name that is already used as a function name, another variable.
  - Test it with
    - `which name`
    - `exist name`

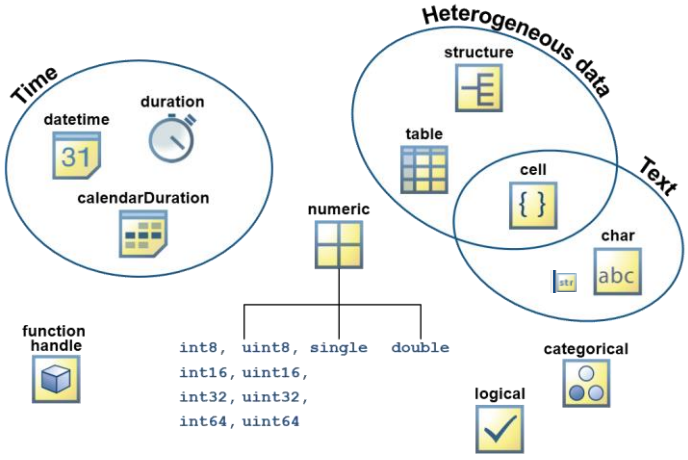
KU LEUVEN

Overview data types (classes)



https://nl.mathworks.com/help/matlab/matlab_prog/fundamental-matlab-classes.html

KU LEUVEN



Taken from MATLABacademy

Data types

- By default, MATLAB treats everything as a matrix of double-precision floating-point numbers.
- Other kinds of data:
  - character strings,
  - heterogeneous lists whose items are different kinds of data,
  - etc.
- MATLAB is a *loosely* or *weakly-typed* language, which has a number of implications: you do not have to explicitly declare the types of the variables you use.

Numeric data types

KU LEUVEN

9

Numeric data types

Class Name	Documentation	Intended Use
`double, single`	Floating-Point Numbers	•Required for fractional numeric data. •Double and Single precision. •Use realmin and realmax to show range of values. •Two-dimensional arrays can be sparse. •Default numeric type
`int8, uint8, int16, uint16, int32, uint32, int64, uint64`	Integers	•Use for signed and unsigned whole numbers. •More efficient use of memory / speed. •Use intmin and intmax to show range of values. •Choose from 4 sizes (8, 16, 32, and 64 bits).

KU LEUVEN

10

Default data type

- Default data type: **double precision array**
- Automatically allocates required memory
- Arrays are resized dynamically
- Reuse a variable name, assigning a new value in an assignment statement
- FORTRAN roots:
  - Base-1 indexing
  - Column wise storage
- MATLAB displays results in scientific notation
  - Use **Home>Environment>Preferences** and/or **format** function to change default
  - format short (.4 digits), format long (.15 digits)

floating point: double

- MATLAB represents floating-point numbers in:
  - double-precision (default)
  - single-precision format.
- Maximum and Minimum Double-Precision Values:
  - `realmax` and `realmin` return the maximum and minimum values that you can represent with the double data type
  - conversion function: `double`
- *File: demo_float*

floating point: single

- MATLAB constructs the single data type according to IEEE Standard 754 for single precision. Maximum and Minimum Double-Precision Values:
  - `realmax('single')` and `realmin('single')` return the maximum and minimum values that you can represent with the double data type
  - conversion function: `single`

```
>> as = single(10)

as =

  single

    10
```

KU LEUVEN

Advanced: double or single?

- Most applications use double precision math for the following reasons:
  - To minimize the accumulation of round-off error,
  - For ill-conditioned problems that require higher precision,
  - The 8 bit exponent defined by the IEEE floating point standard for 32-bit arithmetic will not accommodate the calculation, or
  - There are critical sections in the code which require higher precision.
- *File: demo_single_double.m*
- http://www.hpcwire.com/hpcwire/2006-06-16/less_is_more_exploiting_single_precision_math_in_hpc-1.html

KU LEUVEN

integer

- 4 signed and 4 unsigned integer data types.
- Signed types:
  - enable negative integers as well as positive.
- Unsigned types
  - wider range of numbers.
  - zero or positive.
- MATLAB supports 1-, 2-, 4-, and 8-byte storage for integer data.
  - save memory and execution time for your programs if you use the smallest integer type that accommodates your data.

KU LEUVEN

16

integer

Class	Range of Values	Conversion Function
Signed 8-bit integer	-2^7 to 2^7-1	`int8`
Signed 16-bit integer	-2^{15} to $2^{15}-1$	`int16`
Signed 32-bit integer	-2^{31} to $2^{31}-1$	`int32`
Signed 64-bit integer	-2^{63} to $2^{63}-1$	`int64`
Unsigned 8-bit integer	0 to 2^8-1	`uint8`
Unsigned 16-bit integer	0 to $2^{16}-1$	`uint16`
Unsigned 32-bit integer	0 to $2^{32}-1$	`uint32`
Unsigned 64-bit integer	0 to $2^{64}-1$	`uint64`

KU LEUVEN

17

integer

- MATLAB stores numeric data as double-precision floating point by default.
- Store data as an integer: use one of the conversion functions
  `x = int16(32501);`
- Use the `whos` function to show the dimensions, byte count, and data type of an array represented by a variable.
- ex. – *PAY ATTENTION!*
- *File: demo_integer*

```
>> x = int8(60);
>> y = int8(70);
>> z = x + y

z =

  int8

   127
```

KU LEUVEN

18

complex

- Complex numbers consist of two separate parts: a real part and an imaginary part.
- The basic imaginary unit is equal to the square root of -1.
  represented by: i or j
  `x = 2 + 3i;`
- Another way – use the `complex` function.

```
x = rand(3) * 5;
y = rand(3) * -8;
z = complex(x, y)
```

- Separate a complex number into its real and imaginary parts using the real and imag functions:

```
zr = real(z);
zi = imag(z);
```

KU LEUVEN

19

Inf and NaN

- **Infinity**
  - special value `Inf`.
  - results from operations like division by zero and overflow.
  - Use the `isinf` function to verify that x is positive or negative infinity!
- **NaN**
  - values that are not real or complex numbers
  - NaN: Not a Number.
  - 0/0 and inf/inf result in NaN
  - Use the `isnan` function to verify that x is NaN!
  - Logical Operations on NaN.
    Because two NaNs are not equal to each other, logical operations involving NaN always return false, except for a test for inequality

```
x = 1/0
x =
     Inf
x = 1.e1000
x =
     Inf
x = exp(1000)
x =
     Inf
x = log(0)
x =
    -Inf

x = log(0); isinf(x)
ans = 1

x = 7i/0
x = NaN + Infi

x = log(0); isnan(x) ans = 1
```


Logical data type

Logical data type

Class Name	Documentation	Intended Use
`logical`	Logical Operations	•Use in relational conditions or to test state. •Can have one of two values: true or false. •Also useful in array indexing. •Two-dimensional arrays can be sparse

Logical data

- Logical arrays are usually the result of logical operations

```
M = [true false true]
M =
     1     0     1
x = 1:5;
y = [2 0 1 9 4];
z = x>y
z =
     0     1     1     0     1
```

- conversion function: `logical`
- Logical values can be used to extract data
- *File: demo_logical*

Text data type

KU LEUVEN

24

text

Class Name	Documentation	Intended Use
`char`, `string`	Characters and Strings	• Data type for text. • Native or Unicode®. • Converts to/from numeric. • Use with regular expressions. • For multiple character arrays, use cell arrays. • Starting in R2017a, you also can store text in string arrays.

KU LEUVEN

25

Text

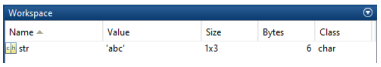
- **character array** is a sequence of characters, just as a numeric array is a sequence of numbers.
  - A typical use is to store **short** pieces of text as character vectors.
  - `c = 'Hello World'`
- **string array** is a container for pieces of text.
  - String arrays provide a set of functions for working with text as data.
  - Starting in R2016b
  - `str = "Greetings friend"`

Character arrays

```
>> str = 'abc'

str =

    'abc'
```



- surrounded by single quotes
  - `str = 'abc'`

    is equivalent to str = ['a' 'b' 'c']
- each character requires 2 bytes
- All operations that apply to vectors and arrays can be used together with strings as well
  - `» str(1) → 'a'`
  - `» str( [ 1 2 ] ) = 'XX' → s = 'XXc'`
  - `» str(end) → 'c'`

Character arrays

- Character arrays can be manipulated like numerical arrays.

```
» T = 'How about this character string?'
» u = T(16:24)
» u = T(24:-1:16)
» u = T(16:24)'
» v = 'I can''t find the manual!'  %  Note quote in string
» u ='If a woodchuck could chuck wood,';
» v = 'how much wood could a woodchuck chuck?';
» disp(u)   % works just like for arrays
```


Character array conversion

- Conversion to numerical arrays: `double / int16`

```
» double( 'abc xyz' )
» ans =
     97    98    99    32   120   121   122
» double( 'ABC XYZ' )
» ans =
     65    66    67    32    88    89    90
```

- Conversion character array: `char`

```
» char( [ 72 101 108 108 111 33 ] )
» ans =
  Hello!
```

Character Arrays

- Horizontal concatenation
  - `s1 = 'hello'`
  - `s2 = 'world'`
  - `s = [s1, ' ', s2]`
- Vertical concatenation (2-D character arrays)
  - `» s = [ 'my first string'; 'my second string' ]`
    `??? Error`
  - `» s = char( 'my first string', 'my second string' )`
    `» s =`
    `my first string`
    `my second string`
  - `» size(s)` → `[ 2 16 ]`
  - `» size( deblank( s(1,:) ) )` → `[ 1 15 ]`
  - char function automatically pads blancs

KU LEUVEN

Character Arrays

- You can use char to hold an m-by-n array of strings as long as each string in the array has the same length. (MATLAB arrays must be rectangular.)
- character arrays in multiple rows: must have the same number of columns as a matrix, use **`char`**

```
group = char('Jan', 'Arnoud',...
'Karel', 'Paul')
group1 = ['Jan    '; 'Arnoud';...
'Karel ', 'Paul  '];
```

- Advice: to hold an array of strings of unequal length, use a *cell array or string array*.
- *File: demo_chararray.m*

KU LEUVEN

Tests

- `ischar()` : returns 1 for a character array
  - `» ischar ( 'LU 1111' )`
    ```
    ans =
         1
    ```
- `isletter()` : returns 1 for letters of the alphabet
  - `» isletter( 'LU 1111' )`
    ```
    ans =
         1     1     0     0     0     0     0
    ```
- `isspace()` : returns 1 for whitespace (blank, tab, new line)
  - `» isspace( 'LU 1111' )`
    ```
    ans =
         0     0     1     0     0     0     0
    ```

KU LEUVEN

Comparison

- Comparing two characters
  - `» 'a' < 'e'`
    ```
    ans =
         1
    ```
- Comparing two arrays character by character
  - `» 'fate' == 'cake'`
    ```
    ans =
         0     1     0     1
    ```
  - `» 'fate' > 'cake'`
    ```
    ans =
         1     0     1     0
    ```

KU LEUVEN

String Comparison

- `strcmp()` : returns 1 if two strings are identical
  - `» a = 'ICTS';`
  - `» strcmp( a, 'ICTS' )`
    `ans =`
    `1`
  - `» strcmp( 'Hello', 'hello' )`
    `ans =`
    `0`
- `strcmpi()` : returns 1 if two strings are identical ignoring case
  - `» strcmpi( 'Hello', 'hello' )`
    `ans =`
    `1`

String Case Conversion

- Lowercase-to-uppercase
  - `» a = upper( 'This is test 1!' )`
    `a =`
    `THIS IS TEST 1!`
- Uppercase-to-lowercase
  - `» a = lower( 'This is test 1!' )`
    `a =`
    `this is test 1!`

Replacing in Strings

- `strrep()` : replaces one string with another
  - `s1 = 'This is a good example';`
  - `s2 = strrep( s1, 'good', 'great' )`

```
s2 =
This is a great example
```

KU LEUVEN

String Conversion

- `num2str()` for numeric-to-string conversion
  - `str = [ 'Plot for x = ' num2str( 10.3 ) ]`

```
str =
Plot for x = 10.3
```

- `str2num()` : converts strings containing numbers to numeric form
  - `x = str2num( '3.1415' )`

```
x =
    3.1415
```

KU LEUVEN

String arrays

- String arrays were introduced in MATLAB 2016b to enhance working with text data.
- Unlike character arrays, strings can be considered complete objects.
- You cannot directly access individual characters within a string using indexing.
- Strings provide a set of functions specifically designed for working with text data.
- Internally, strings are encoded using UTF-16.
- They are more efficient for storing text data compared to character arrays.

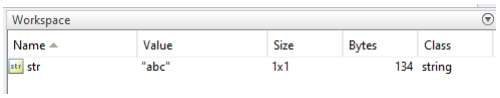
String arrays

```
>> str = "abc"

str =

    "abc"
```

- *File: demo_stringarrays.mlx*
- surrounded by double quotes
  - `str = "abc"`



- Concatenation

```
str1 = "hello"
str2 = "MATLAB"
sv = [str1 ; str2]
sh = [str1 , str2]
```

```
>> str0 = string([])

str0 =

  0×0 empty string array

>> whos
  Name        Size              Bytes  Class     Attributes

  str0        0x0                  96  string

>>
```

http://blogs.mathworks.com/loren/2016/12/22/singing-the-praises-of-strings/
http://blogs.mathworks.com/loren/2017/04/24/working-with-text-in-MATLAB/

String arrays

- No individual characters in string array, treatment similar to cell arrays

```
str_a = ["Mercury","Gemini","Apollo";...
       "Skylab","Skylab B","ISS"]
stra1 = str_a(2,2)
stra2 = str_a(2,:)
stra1_c = str_a{2,2}
```

- MATLAB provides a set of functions to work with string arrays. For example, you can use the `split`, `join`, and `sort` functions