Lecture 2: Variables, and 1D arrays
תזכורת מהשיעור הקודם:

קוד פתרון במטלאב:

```
i = 0;
s = 0;
while i < 10
    value = input('x =');
    s = s + value;
    i = i + 1;
end
disp(s);
```

הגדרת בעיה:

קלט: סדרת 10 מספרים.
פלט: סכום 10 מספרים.
Identifiers

- Identifiers are all the words that build up the program.
- An identifier is a sequence of letters, digits and underscores “_”.
- Maximal length of identifiers is 63 characters.
- Can’t start with a digit.
- Can’t be a reserved word.

Examples of Legal identifiers:
- time
- day_of_the_week
- bond007
- findWord

Examples of illegal identifiers:
- 007bond
- #time
- ba-baluba
- if
- while
An overview of the main players in a program

- **Reserved words**: if, while, end
- **Library functions**: Disp, input
- **Variables**: s, x
- **Constants**: 0, ‘Go home’
- **User defined functions**: 44
Reserved words (keywords)

- Words that are part of the Matlab language
  - There are 17 reserved words:
    - for
    - function
    - otherwise
    - try
    - break
    - end
    - return
    - switch
    - catch
    - if
    - elseif
    - continue
    - global
    - while
    - case
    - else
    - persistent

Do NOT try to redefine their meaning!
## Constants

- The value of a constant is fixed and does not change throughout the program

<table>
<thead>
<tr>
<th>Numbers</th>
<th>Arrays</th>
<th>Chars</th>
<th>Matrices</th>
<th>Strings</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>[ 1 2 3 4 5 ]</td>
<td>'c'</td>
<td>[5 3; 4 2]</td>
<td>'I like to eat sushi'</td>
</tr>
<tr>
<td>0.3</td>
<td></td>
<td></td>
<td></td>
<td>'1 + 2'</td>
</tr>
</tbody>
</table>
Variables

- Why do we need variables?

- Example:

  ```matlab
  >> salary = 9000;
  >> new_salary = salary * 3;
  >> disp(new_salary);
  27000
  ```
Variables

- Another example:

```matlab
price_bamba = 3
```

What happens if you omit the `;`?
Variables

- Another example:

```matlab
price_bamba = 3
n_bamba = 2;
```

The Matlab Console

```
price_bamba = 3
```

What happens when we add the `;`?
Variables

Another example:

price_bamba = 3
n_bamba = 2;
price_bisly = 5
n_bisly = 3;

total_price = price_bamba * n_bamba + price_bisly * n_bisly
n_bamba = 5
total_price

The Matlab Console

price_bamba = \frac{3}{3}
price_bisly = \frac{5}{5}
total_price = 21
n_bamba = 5
total_price = 21
Tip #1: Give your variable meaningful names.

a = 9000
b = 100

*are a bad choice for naming variables that store your working hours and salary!*

*A more meaningful choice of names would be:*

salary = 9000;
hours = 5;
Variables

- **Tip #2**: Don’t make variable names too long
  
  salary_I_got_for_my_work_at_the_gasoline_station = 9000;
  salary_I_got_for_my_work_in_the_bakery =
  salary_I_got_for_my_work_at_the_gasoline_station * 3;
  
  disp(salary_I_got_for_my_work_in_the_bakery);

  Very bad choice of variable name!!!

- **Tip #3**: Whatever you do - be consistent.
Variables Types

- Each variable has a *type*

- Why do we need variable types?

- Different types of variable store different types of data
Variables Types

Different types of variable store different types of data

```
>> a = 10
a =
 10

>> class(a)
ans =
double

>> b = 10.56
b =
 10.5600

>> class(b)
ans =
double

>> c = 'Bush'
c =
 Bush

>> class(c)
ans =
char

>> d = true
d =
 1

>> class(d)
ans =
logical
```

Returns the type of a variable

The default variable type in Matlab is double
Variables Types

- Different variable types require different memory allocations

```matlab
>> a = 10.4 %double requires 8 bytes
a =
    10.4
    1

1 0 0 0 1 1 0 0

>> b = 'B' %char requires 2 bytes
b =
    B
    1

1 0 0 0 1 1 0 0
```

Memory allocation and release is done automatically in Matlab.
וד על תיפוס

טייפוס שאינו נומרי

キーימות טבלאות סטנדרטיות המשויכות לקל関わר שלם. הנפוצה ביניהם היא טבלת American Standard Code for Information Interchange (ASCII), והעלו-피יה:

ה المتوים יז,י, ... ,א,ו מתאימים לערכיהם 97,98, ..., 122

ה المتوים 6,7, ..., 1,0 מתאימים לערכיהם 48,49, ..., 57

יש גים شيئיות נספונות,_ENTRIES

EBCDIC

_UNICODE

 mamma
עוד על תווים

יש גם תווים מיוחדים:

- תווים שלא נראים:
- ואותיות יוניות,
- עוד ועוד...

לכן ב-Matlab כל תווים מייצגים "16 סיביות (ולא 8)

אותיות יוניות, נוספים עוד...

&  38
*  42
+  43
backslash(\\)  92
tab(\t)  9
newline(\n)  10
back space(\b)  8
alarm bell(\a)  7
From character to number

- Convert character to number:
  ```python
  >> num = double('a');
  ```
  Set 97 to the numeric variable `num`.

- Matlab converts 'a' to 97 and performs the addition.
  ```python
  >> sum = 'a' + 3;
  ```
  The result is 'd', which is between 'a' and '97', as expected in Matlab.
Computer precision limitations

- How much is:
  \[ \gg 0.42 + 0.08 - 0.5 \]
  \[ \text{ans} = 0 \]

- How much is:
  \[ \gg 0.42 - 0.5 + 0.08 \]
  \[ \text{ans} = -1.3878e-017 \]
Special variables

- **ans**

\[
\begin{align*}
\text{>> } & 4 \ast 5 \\
\text{ans } & = \\
& 20 \\
\text{>> } & \text{ans } + 1 \\
\text{ans } & = \\
& 21
\end{align*}
\]
Special variables

- ans
- pi
- inf

```matlab
>> 2 * inf
ans =
    Inf

>> 1 / 0
Warning: Divide by zero.
ans =
    Inf
```
Special variables

- ans
- pi
- inf
- NaN

```plaintext
>> 0 / 0
Warning: Divide by zero.
ans =
    NaN

>> NaN + 1
ans =
    NaN
```
Basic build in functions

- **Round**
  ```matlab
  >> x = 2.6;
  >> round(x)
  ans =
   3
  >> x = 2.2;
  >> round(x)
  ans =
   2
  >> x = 1.5;
  >> round(x)
  ans =
   2
  ```

- **Round up**
  ```matlab
  >> x = 2.2;
  >> ceil(x)
  ans =
   3
  >> x = 2.6;
  >> ceil(x)
  ans =
   3
  ```

- **Round down**
  ```matlab
  >> x = 2.6;
  >> floor(x)
  ans =
   2
  >> x = 2.2;
  >> floor(x)
  ans =
   2
  ```
Basic build in functions

- I want to get the absolute value of a variable but don’t know the command… what should I do?

... *Google it!*

```
>> x = -2
>> abs(x)
ans = 2
```
Help!!!

- help
- doc
  - Example: doc disp
- Google
Data is stored in data structures

- A **Data Structure** is a set of values that are organized in a predefined manner

- Examples:
  - Tree
  - Stack
  - Queue
  - Array
  - And more...
Array is the main data structure used in Matlab

- Almost everything we build in Matlab - we build from arrays.

- The variables that contain one element that we learned so far can be thought of as a degenerate case of a $1 \times 1$ array.
Examples of 1D and 2D arrays

- An 1D array containing student grades
  \[
  \begin{array}{ccccccc}
    100 & 100 & 100 & 98 & 100 & 50 & 99 \\
  \end{array}
  \]

- An 1D array containing protein levels
  \[
  \begin{array}{cccc}
    0.54 & 0.01 & 0.0 & 1.2 & 1.24 \\
  \end{array}
  \]

- An 1D array of characters
  \[
  \text{Don't worry, be happy}
  \]

- A 2D array, also called matrix, of distances between cities
  \[
  \begin{array}{cccc}
    0 & 23 & 120 & 21 \\
    23 & 0 & 56 & 10 \\
    120 & 56 & 0 & 8 \\
  \end{array}
  \]
Creating 1D Arrays

- The simplest way to create an array

\[
\begin{align*}
\text{>> } x &= [1 \ 2 \ 3 \ 8 \ 12 \ 40]; \\
\text{>> } \text{disp}(x) \\
&\begin{array}{cccccc}
1 & 2 & 3 & 8 & 12 & 40 \\
\end{array}
\end{align*}
\]

- Adding commas to separate the numbers gives the same result:

\[
\begin{align*}
\text{>> } x &= [1, 2, 3, 8, 12, 40]; \\
\text{>> } \text{disp}(x) \\
&\begin{array}{cccccc}
1 & 2 & 3 & 8 & 12 & 40 \\
\end{array}
\end{align*}
\]
Creating 1D Arrays

- How do we create an array containing all numbers between 10 and 100?

No problem! This is how you do it:

```plaintext
>>> x = [10, 11, 12, 13, 14, 15, 16, ...
      17, 18, 19, 20, 21, 22, 23, 24, 25, ...
      26, 27, 28, 29, 30, 31, 32, 33, 34, ...
      35, 36, 37, 38, 39, 40, 41, 42, 43, ...
      44, 45, 46, 47, 48, 49, 50, 51, 52, ...
      53, 54, 55, 56, 57, 58, 59, 60, 61, ...
      62, 63, 64, 65, 66, 67, 68, 69, 70, ...
      71, 72, 73, 74, 75, 76, 77, 78, 79, ...
      80, 81, 82, 83, 84, 85, 86, 87, 88, ...
      89, 90, 91, 92, 93, 94, 95, 96, 97, ...
       98, 99, 100];
```
Creating 1D Arrays

- Alternatively we can do this:

```matlab
>> x = 10 : 1 : 100;
>> disp(x)
Columns 1 through 19
  10   11   12   13   14   15   16
  17   18   19   20   21   22   23
  24   25   26   27   28
...
```

Start at 10, count up by 1, stop when you get to 100
Creating 1D Arrays

- What is the value of x?

```matlab
>> x = 0 : 0.1 : 2;
>> disp(x)
0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9
1 1.1 1.2 1.3 1.4 1.5 1.6 1.7 1.8 1.9 2
```

- How can you create an array with numbers from 100 to 1 in decreasing order?
Creating 1D Arrays

- Initializing a vector to zeros
  \[ x = \text{zeros}(1, \ 5) \]
  \[ x = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \]

- Initializing a vector to ones
  \[ x = \text{ones}(1, \ 6) \]
  \[ x = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \]

- Initializing a vector to random numbers
  \[ x = \text{rand}(1, \ 4) \]
  \[ x = \begin{bmatrix} 0.9501 & 0.2311 & 0.6068 & 0.4860 \end{bmatrix} \]

Generates a Uniformly distributed pseudo-random number between 0 and 1
Concatenating 1D Arrays

- Example:

```matlab
x = 1 : 5;
y = 100 : 105;
z = [x, y, x];
disp(z);
```

x, y and x are concatenated into one array
Indexing 1D Arrays

- **Question:** How do we retrieve/manipulate the content of a specific element inside an array?

- **Answer:** We use a mailbox like system called indexing

- \( x(i) \rightarrow \) is the \( i \)'th element of \( x \)
We will study different types of indexing through examples:

What is the output of the following program?

```matlab
x = 10 : -1 : 1;
y = x(3);
disp(y);
8
```

What is the output of the following program?

```matlab
x = 10 : -2 : 1;
y = x(5);
disp(y);
```
Indexing 1D Arrays

- What is the output of the following program?
  
  ```
  x = 10 : -1 : 1;
  y = x(12);
  disp(y);
  ```

  ```
  ??? Index exceeds matrix dimensions.
  ```

- What is the output of the following program?
  
  ```
  x = 10 : -1 : 1;
  y = x(3.2);
  disp(y);
  ```

  ```
  ??? Subscript indices must either be real positive integers or logicals.
  ```
Indexing 1D Arrays

- We can retrieve more than one element at a time:

- What is the output of the following program?

```matlab
x = 10 : -1 : 1;
y = x([3, 9, 4]);
disp(y);
```

- What is the output of the following program?

```matlab
x = 10 : -1 : 1;
y = x(2 : 2 : 8);
disp(y);
```
Indexing 1D Arrays

- Indexing the last element in a vector...

- What is the output of the following program?
  \[ x = 10 : -1 : 1; \]
  \[ y = x(\text{end}) \]
  \[ \text{disp}(y); \]
  \[ 1 \]

- What is the output of the following program?
  \[ x = 10 : -1 : 1; \]
  \[ z = x(\text{end} - 1) \]
  \[ \text{disp}(z); \]
  \[ 2 \]
The same indexing can be applied to arrays of chars.

What is the output of the following program?

```matlab
str = 'If I was a rich man, Yaba dibi ...
    dibi dibi dibi di';
str2 = str([1 : 10, 21: 25]);
disp(str2);
If I was a Yaba
Using indexing to edit 1D arrays

- We can manipulate the content of an array

- What is the output of the following program?
  \[ x = 10 : -1 : 1; \]
  \[ x(2) = 10; \]
  \[ \text{disp}(x); \]

- What is the output of the following program?
  \[ x = 10 : -1 : 1; \]
  \[ x(14) = 99; \]
  \[ \text{disp}(x); \]

Remark: Notice that adding an element beyond the array boundary is NOT an error!
Using indexing to edit 1D arrays

- We can erase parts of an array

- What is the output of the following program?
  \[ x = 10 : -1 : 1; \]
  \[ x(2) = []; \]
  \[ disp(x); \]

- What is the output of the following program?
  \[ x = 10 : -1 : 1; \]
  \[ x(2 : 4) = []; \]
  \[ disp(x); \]
1D Array Orientation

- So far we’ve seen row vectors (one row and multiple columns),
  How can one create a column vector (one column and multiple rows)?

- Use **semicolon** separator:
  \[
  x = [1; 2; 3; 4; 5];
  \]
  \[
  \text{disp}(x);
  \]
  1
  2
  3
  4
  5

- Use **transpose** operator
  \[
  x = [1, 2, 3, 4, 5];
  y = x';
  \]
  \[
  \text{disp}(y)
  \]
  1
  2
  3
  4
  5
Simple operations on 1D arrays

All the operations that were applied on variables containing one element can be applied to 1D arrays

- What is the output of the following program?
  ```
  >> c = 1 : 0.1 : 2
  >> round(c)
  ```

- What is the output of the following program?
  ```
  >> c = 1 : 0.1 : 2
  >> ceil(c)
  ```

- What is the output of the following program?
  ```
  >> abs(floor(-0.3 : 0.1 : 0.3))
  ```
Simple operations on 1D arrays

- Finding the **maximal** number in a vector
  >> \( x = 1 : 50; \)
  >> \texttt{max}(x)
  50

- Finding the **minimal** number in a vector
  >> \texttt{min}(x)
  1

- Finding the **mean** of a vector
  >> \texttt{mean}(x)
  25.500
Simple operations on 1D arrays

- Finding the **size** of a vector

```matlab
>> x = 1 : 50;
>> size(x)
ans =
   1   50
```

```matlab
>> y = x'
>> size(y)
ans =
   50   1
```

- Finding the **length** of a matrix

```matlab
>> x = 1 : 50;
>> length(x)
ans =
    50
```

```matlab
>> y = x'
>> length(y)
ans =
    50
```

Returns the maximal dimension
Simple operations on 1D arrays

```
x = 1:50
```

What is the difference between:

- \( y = x(\text{length}(x)) \);
- \( y = x(\text{end}) \);

"Same same..."