3.1 Numeric Data Types

- When it comes to working with data we should choose the most appropriate data types.

- Standard numeric data types are either fixed or floating point (more on this next we). Floating point numbers are preferred for values over a large range and but rounding errors may occur more easily than fixed types.

- Floating points represent variable precision numbers (decimal point can shift) along with a certain number of decimal places depending on whether they are **single** or **double** precision data type. Double is default in Matlab.

single	**double**
7 d.p.	15 d.p.

3.1 Numeric Data Types

- Fixed data types are used to minimise error propagation caused by rounding errors (again, more next week).

- They are also useful in embedded systems that require more precise memory management of data.

- Matlab has 4 signed, and 4 unsigned **integer** data types.

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`

3.1 Numeric Data Types

- You can create **complex** numbers by combining floating point numbers with multiplication by the imaginary unit, i.

```
>> z = 2 + 3i
```

- The value of infinity is assigned to division by 0. This can be useful for improper integrals and infinite series. Also if a number is too large for memory it returns **Inf**.

```
>> 1 / 0
ans
      Inf
```

```
>> exp(800)
ans
      Inf
```

- **NaN** (Not a Number) is assigned to other undefined quantities such as indeterminate forms (0 / 0 etc).

3.2 Character Data Types

- Characters input between single quotes are of **char** type.
- These are useful for data that is represented as single characters (multiple choice answers, DNA sequences, any alphabetical categorisation).

‘hello’ is type char and has length 5

- Characters input between double quotes are of **string** type.
- They are useful for text analysis and come with many functions to manipulate them.

“hello you” is type string and has length 1

EXAMPLE 1

Double vs. Single

```
Command Window
>> format long
>> single(pi)

ans =

  single

   3.1415927

>> double(pi)

ans =

   3.141592653589793
```

EXAMPLE 2

Char vs. String

```
Command Window
>> x = 'ADBBCD';
>> length(x)

ans =

     6

>> x(4)

ans =

    'B'
```

```
Command Window
>> y = "Lorem ipsum dolor sit amet";
>> length(y)

ans =

     1

>> strlength(y)

ans =

    26

>> length(strsplit(y))

ans =

     5

>> y(1)

ans =

    "Lorem ipsum dolor sit amet"

>> y{1}

ans =

    'Lorem ipsum dolor sit amet'

>> y{1}(2)

ans =

    'o'

>> y(1)(2)
Error: Invalid array indexing.
```

3.3 Mixed Variable Data Types

EXAMPLE 3

- The **cell** data type stores values of different type by **index**.
- They use curly braces **{}** to access their values.

```
Command Window
>>  cell_data = { 'Some words', [1,2,3], [1 3; 8 5] }

cell_data =

  1×3 cell array

    {'Some words'}    {[1 2 3]}    {2×2 double}

>> cell_data(2)

ans =

  1×1 cell array

    {[1 2 3]}

>> cell_data{2}

ans =

     1     2     3

>> cell_data{2}(3)

ans =

     3
```

3.3 Mixed Variable Data Types

EXAMPLE 4

- The **structure** type stores values of different type by **field-value** pairs.
- They use the dot operator (**.**) along with the index to access the value.

Command Window

```
>> structure_data = struct( 'age',23, 'height', 180, 'sex', 'M')

structure_data =

  struct with fields:

       age: 23
    height: 180
       sex: 'M'

>> structure_data.age

ans =

    23

>> structure_data(2).age = 27; structure_data(2).height = 173; structure_data(2).sex = 'F'

structure_data =

  1×2 struct array with fields:

    age
    height
    sex

>> structure_data(2).age

ans =

    27
```

3.3 Mixed Variable Data Types

- The **map container** type stores values of different type by **key-value** pairs.

- They use key-indexes to access the values.

EXAMPLE 5

Command Window

```
>> cmap_data = containers.Map({'Jerry', 'Jenny'}, {78, 86})

cmap_data =

  Map with properties:

        Count: 2
      KeyType: char
    ValueType: double

>> cmap_data('Jenny')

ans =

    86
```

Note: The keys can be any data type

3.4 Categorical Data Type

EXAMPLE 6

- Used for finite categories of data.
- Useful when you need to search for elements of an array which have a certain category value.

```
Command Window
>> A = {'small' 'medium'; 'medium' 'medium'; 'large' 'small'}
A =
  3×2 cell array
    {'small' }    {'medium'}
    {'medium'}    {'medium'}
    {'large' }    {'small' }
>> valueset = {'medium', 'small', 'large'}
valueset =
  1×3 cell array
    {'medium'}    {'small'}    {'large'}
>> B = categorical(A,valueset)
B =
  3×2 categorical array
     small        medium
     medium       medium
     large        small
>> categories(B)
ans =
  3×1 cell array
    {'medium'}
    {'small' }
    {'large' }
>> B(1,2)
ans =
  categorical
     medium
>> B=='small'
ans =
  3×2 logical array
   1   0
   0   0
   0   1
```

3.4 Categorical Data Type

EXAMPLE 7

- They can be ordinal (with a specific ordering)
- For this you must consider the mapping between the ordering values and the data, then input the data as the ordering values.

Command Window

```
>> valueset = 1:3
valueset =
     1     2     3
>> cats = {'small' 'medium' 'large'}
cats =
  1×3 cell array
    {'small'}    {'medium'}    {'large'}
>> A = [1 2; 2 2; 3 1]
A =
     1     2
     2     2
     3     1
>> B = categorical(A,valueset,cats, 'Ordinal', 1)
B =
  3×2 categorical array
     small       medium
     medium      medium
     large       small
>> B(1,:) < B(2,:)
ans =
  1×2 logical array
   1   0
```

3.4 Categorical Data Type

- Along with the **discretize()** function they can be useful for easily categorising numeric data.

EXAMPLE 8

Variables - y

x

100x150 double

	1	2	3	4	5	6	
1	0.2858	0.5801	0.5481	0.0399	0.6202	0.3601	0
2	0.9312	0.2475	0.7807	0.5000	0.4357	0.6265	0
3	0.7999	0.9359	0.5589	0.2103	0.4158	0.8529	0
4	0.3311	0.5646	0.3017	0.4760	0.2852	0.1200	0
5	0.7374	0.8796	0.4606	0.6540	0.9051	0.1743	0
6	0.5674	0.9927	0.9209	0.0490	0.6828	0.9506	0
7	0.0161	0.9463	0.8862	0.8583	0.1043	0.9775	0
8	0.4255	0.4784	0.8795	0.0395	0.0835	0.2520	0

y

100x150 categorical

	1	2	3	4	5	6	
1	medium	medium	medium	small	medium	medium	larg
2	large	small	large	medium	medium	medium	med
3	large	large	medium	small	medium	large	med
4	medium	medium	medium	medium	medium	small	med
5	medium	large	medium	medium	large	small	med
6	medium	large	large	small	medium	large	larg
7	small	large	large	large	small	large	larg
8	medium	medium	large	small	small	medium	med

Command Window

```
>> x = rand(100,150);
>> y = discretize(x,[0 .25 .75 1],'categorical',{'small','medium','large'});
>> summary(y(1,:))
     small       medium       large
     35          81           34
>> summary(y(:,2))
     small        28
     medium       45
     large        27
```

3.5 Table Data Type

- Used for data that is stored as rows and columns (relational databases).

- Spreadsheets and SQL-databases are examples of relational databases.

- We can access the values by indexing (see right and next slide) or by the names (see slide 14).

EXAMPLE 9

Command Window

```
>> T1 = table([1:3]', [1:3; 4:6; 7:9], {'M'; 'F'; 'F'})
T1 =
  3×3 table
    Var1        Var2           Var3

    ____    ___________        _____
     1      1     2     3      {'M'}
     2      4     5     6      {'F'}
     3      7     8     9      {'F'}
>> T1(2,2)
ans =
  table
        Var2

    ___________
    4     5     6
>> T1{2,2}
ans =
     4     5     6
>> T1{2,2}(2)
ans =
     5
```

3.5 Table Data Type

EXAMPLE 10

Command Window

```
>> load patients
>> T2 = table(Age, Gender, Weight, Height);
>> T2(1:4,:)
ans =
  4×4 table
    Age        Gender        Weight     Height
    ___     ___________    ______    ______
    38      {'Male'   }     176        71
    43      {'Male'   }     163        69
    38      {'Female'}     131        64
    40      {'Female'}     133        67
>> T2.BMI = (T2.Weight*0.453592)./(T2.Height*0.0254).^2;
>> T2(1:4,:)
ans =
  4×5 table
    Age        Gender        Weight     Height       BMI
    ___     ___________    ______    ______     ______
    38      {'Male'   }     176        71       24.547
    43      {'Male'   }     163        69       24.071
    38      {'Female'}     131        64       22.486
    40      {'Female'}     133        67       20.831
>> T2.Properties.VariableNames{'BMI'} = 'Body Mass Index';
>> T2(1:4,:)
ans =
  4×5 table
    Age        Gender        Weight     Height    Body Mass Index
    ___     ___________    ______    ______    _______________
    38      {'Male'   }     176        71           24.547
    43      {'Male'   }     163        69           24.071
    38      {'Female'}     131        64           22.486
    40      {'Female'}     133        67           20.831
```

Workspace

Name	Value	Class
Age	*100x1 double*	double
ans	*4x5 table*	table
Diastolic	*100x1 double*	double
Gender	*100x1 cell*	cell
Height	*100x1 double*	double
LastName	*100x1 cell*	cell
Location	*100x1 cell*	cell
SelfAssesse...	*100x1 cell*	cell
Smoker	*100x1 logical*	logical
Systolic	*100x1 double*	double
T2	*100x5 table*	table
Weight	*100x1 double*	double

3.5 Table Data Type

EXAMPLE 11

Command Window

```
>> T2(1:4,:)
ans =
  4×5 table
    Age       Gender       Weight    Height    Body Mass Index
    ___    __________    ______    ______    _______________
    38     {'Male'  }     176       71           24.547
    43     {'Male'  }     163       69           24.071
    38     {'Female'}     131       64           22.486
    40     {'Female'}     133       67           20.831
>> T2(2:4,{'Gender', 'Body Mass Index'})
ans =
  3×2 table
      Gender       Body Mass Index
    __________    _______________
    {'Male'  }        24.071
    {'Female'}        22.486
    {'Female'}        20.831
>> T2.Properties.RowNames = LastName;
>> T2(1:4,:)
ans =
  4×5 table
                  Age       Gender       Weight    Height    Body Mass Index
                  ___    __________    ______    ______    _______________
    Smith         38     {'Male'  }     176       71           24.547
    Johnson       43     {'Male'  }     163       69           24.071
    Williams      38     {'Female'}     131       64           22.486
    Jones         40     {'Female'}     133       67           20.831
>> T2({'Johnson', 'Jones'},{'Gender', 'Body Mass Index'})
ans =
  2×2 table
                    Gender       Body Mass Index
                  __________    _______________
    Johnson       {'Male'  }        24.071
    Jones         {'Female'}        20.831
```

3.6 Checking Data Types

EXAMPLE 12

- Since each data type has unique properties and operators it is important to be able to check them when writing code that does calculations with them.

- Here are some common cases of checking. For a full list see the documentation.

Command Window

```
>> isfloat( 44 )
ans =
  logical
   1
>> isfloat( int8(44) )
ans =
  logical
   0
>> ischar( 'a' )
ans =
  logical
   1
>> isstring( 'a' )
ans =
  logical
   0
>> iscell( {'a' 'b' 'c'} )
ans =
  logical
   1
>> isstruct(struct('Pim', 91, 'Gun', 88))
ans =
  logical
   1
>> istable(struct('Pim', 91, 'Gun', 88))
ans =
  logical
   0
>> isinf(exp(800))
ans =
  logical
   1
```

3.7 Converting Data Types **EXAMPLE 13**

- It's possible to convert between any data types using appropriate functions and syntax.
- Certain conversions may be invalid depending on size constraints of the converted vs. unconverted data.

Command Window

```
>> x = single(pi)
x =
  single
   3.1415927
>> double(x)
ans =
   3.141592741012573
>> pi_char = '3.1415927'
pi_char =
    '3.1415927'
>> str2double(pi_char)
ans =
   3.141592700000000
>> dec2bin(37)
ans =
    '100101'
>> cellstr( ["Lorem", "ipsum", "dolor"] )
ans =
  1×3 cell array
    {'Lorem'}    {'ipsum'}    {'dolor'}
>> mycell = {[1],    [2 3 4]; [5:7; 8:10], ones(3)}
mycell =
  2×2 cell array
    {[        1]}    {[   2 3 4]}
    {2×3 double}    {3×3 double}
>> cell2mat(mycell(1,:))
ans =
     1     2     3     4
>> cell2table(mycell)
ans =
  2×2 table
        mycell1         mycell2

    ____________    ____________
    {[        1]}    {[   2 3 4]}
    {2×3 double}    {3×3 double}
```

What's the output?

```
x = {2:2:10, [1 2;3 4], 'label'}

>> x{2}
```

- 2
- 4
- 2 4 6 8 10
-
- [1 x 5 double]
- [2 x 2 double]

Tc 0

Powered by Poll Everywhere

Start the presentation to see live content. For screen share software, share the entire screen. Get help at **pollev.com/app**

What's the output?

```
x = {2:2:10, [1 2;3 4], 'label'};

>> x(1)
```

- 1
- 2
- 2 4 6 8 10
- [1 x 5 double]
- [2 x 2 double]

Powered by Poll Everywhere

Start the presentation to see live content. For screen share software, share the entire screen. Get help at **pollev.com/app**

How to get Dog?

```
x = struct('Species',{'Bird','Dog','Cat'},'Sex',{'F','F','M'});
```

- x.Species(2)
- x(2).Species
- x{2}(2)
- x(2){2}

Powered by Poll Everywhere

Start the presentation to see live content. For screen share software, share the entire screen. Get help at **pollev.com/app**

How to add a female mongoose?

```
x = struct('Species',{'Bird','Dog','Cat'},'Sex',{'F','F','M'});
```

- x.Species(4)='Mongoose'; x.Sex(4)='F'
- x(4).Species='Mongoose'; x(4).Sex='F'
- x(1,4)='Mongoose'; x(2,4).Sex='F'
- x(4,1)='Mongoose'; x(4,2).Sex='F'

Tc 0

3.8 Getting User Input

EXAMPLE 14

Command Window

```
>> x = input('What is your name?')
fx What is your name?
```

Command Window

```
>> x = input('What is your name?')
What is your name? 'Jon'
x =
     'Jon'
```

Workspace

Name	Value
x	'Jon'

3.9 Formatting String Output

- Use the **sprintf** or **fprintf** function to format strings.

sprintf('format',x,...)

EXAMPLE 15

Command Window

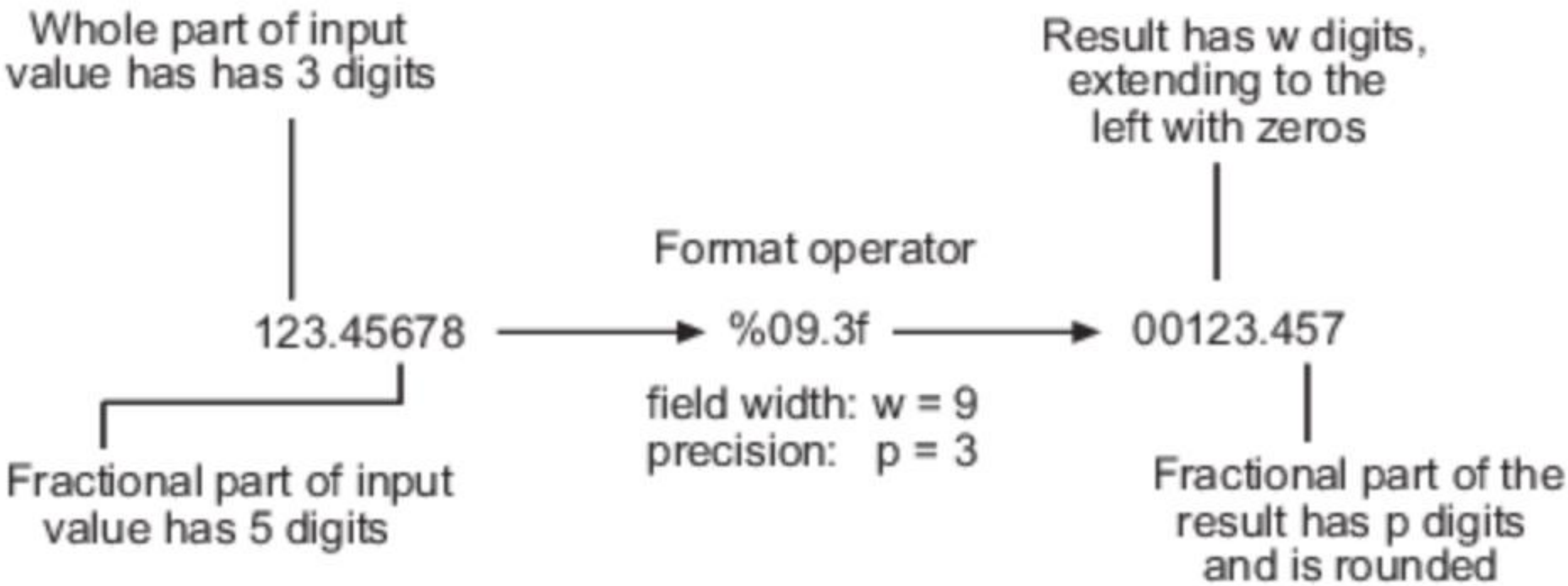
```
>> v = 50.6175;
>> fprintf('The velocity is %8.4f m/s\n', v)
The velocity is  50.6175 m/s
```

Workspace

Name	Value
v	50.6175

Format Code	Description
%d	Integer format
%e	Scientific format with lowercase e
%E	Scientific format with uppercase E
%f	Decimal format
%g	The more compact of %e or %f

Control Code	Description
\n	Start new line
\t	Tab


Whole part of input value has has 3 digits
Result has w digits, extending to the left with zeros
Format operator
123.45678
%09.3f
00123.457
field width: w = 9
precision: p = 3
Fractional part of input value has 5 digits
Fractional part of the result has p digits and is rounded

EXAMPLE 16

Command Window

```
>> fprintf('%5d %10.3f %8.5e\n',100,2*pi,pi);
  100      6.283 3.14159e+00
```

EXAMPLE 17

fprintfdemo.m

```
function fprintfdemo
x = [1 2 3 4 5];
y = [20.4 12.6 17.8 88.7 120.4];
z = [x;y];
fprintf('    x      y\n');
fprintf('%5d %10.3f\n',z);
```

Command Window

```
>> fprintfdemo
    x      y
    1     20.400
    2     12.600
    3     17.800
    4     88.700
    5    120.400
```

What the output of sprintf('pi to 7 d.p. is: %012.7f',pi)?

- 'pi to 7 d.p. is: 3.1415927'
- 'pi to 7 d.p. is: 03.1415927'
- 'pi to 7 d.p. is: 003.1415927'
- 'pi to 7 d.p. is: 0003.1415927'
- 'pi to 7 d.p. is: 00003.1415927'
- 'pi to 7 d.p. is: 000003.1415927'

Tc 0

Start the presentation to see live content. For screen share software, share the entire screen. Get help at **pollev.com/app**

3.10 Creating & Accessing Data Files

- You can **save** variables you have calculated and **load** them at a later date.

EXAMPLE 18

Command Window

```
>> g = 9.81;m = 80;t = 5;
>> cd = [.25 .267 .245 .28 .273]';
>> v = sqrt(g*m ./cd).*tanh(sqrt(g*cd/m)*t);
>> save veldrag v cd
>>
```

Workspace

Name	Value
cd	[0.2500;0....
g	9.8100
m	80
t	5
v	[39.4514;...

↓

Command Window

```
>> clear
>>
```

Workspace

Name	Value

↓

Command Window

```
>> clear
>> load veldrag.mat
>>
```

Workspace

Name	Value
cd	[0.2500;0....
v	[39.4514;...

3.11 Vectorisation of Code

- If possible, avoid loops and use **vectorised code** instead.

EXAMPLE 19

```
i = 0;
for t = 0:0.02:50
  i = i + 1;
  y(i) = cos(t);
end
```

→

```
t = 0:0.02:50;
y = cos(t);
```

3.12 Preallocation of Memory

- Since MATLAB increases the size of an array on every iteration of a loop it is computationally faster to **preallocate** the size of the vector/matrix if known.

EXAMPLE 20

```
t = 0:.01:5;
for i = 1:length(t)
  if t(i)>1
    y(i) = 1/t(i);
  else
    y(i) = 1;
  end
end
```

```
t = 0:.01:5;
y = ones(size(t));
for i = 1:length(t)
  if t(i)>1
    y(i) = 1/t(i);
  end
end
```

Which code gives the same output?

```
A=1;
for i=2:4
    A(i)=2*A(i-1);
end
```

- A=1; A(end)=[2 4 8]
- A=1; A(2:2:4)=[2 4 8]
- A=[1 1 1 1]; A(2:end)=[2 4 8]
- A=[1 1 1 1]; A(2:2:4)=[2 4 8]

Tc 0

Start the presentation to see live content. For screen share software, share the entire screen. Get help at **pollev.com/app**

3.13 Anonymous Functions

- We can create functions in the workspace instead of as a file. These are known as **anonymous functions**.

EXAMPLE 21

```
>> function_name = @(inputs) expression
```

Creates anonymous function

```
>> f1=@(x,y) x^2 + y^2;

>> f1(3,4)

ans =
    25
```

3.14 Function of a Function

- Instead of using a number or variable as input to a function, we can also pass anonymous functions.

EXAMPLE 22

```
>> f1 = @(x) x.^2;
>> f2 = @(f,a,b) f(b) - f(a);
>> f2(f1,2,3)
ans =
     5
```

First function is used as input to another function

Which command returns the value 18?

```
a = @(s,t) s*t
b = @(j,k) k(j(1),j(2))+k(j(2),j(3))
```

- a([2 3 4],b)
- b([2 3 4],a)
- a([2 9],b)
- b([2 9],a)

Start the presentation to see live content. For screen share software, share the entire screen. Get help at **pollev.com/app**

3.15 Variable Input Argument

- Sometimes we want the flexibility for a function to accept a different number of inputs depending on the user. We use **varargin**.

EXAMPLE 23

```
function y = f_varin(a,b,varargin)
if isempty(varargin)
    y = a+b;
elseif length(varargin) == 1
    y = a+b+varargin{1};
elseif length(varargin) == 2
    y = a+b+varargin{1}-varargin{2};
else
    disp('Too many input arguments.')
end
```

Tells MATLAB to expect a variable number of inputs (cell array)

Checks if variable is empty or not

```
>> f_varin(1,2)
ans =
     3

>> f_varin(1,2,3)
ans =
     6

>> f_varin(1,2,3,4)
ans =
     2

>> f_varin(1,2,3,4,5)
Too many input
arguments.
```

3.16 Variable Output Functions

- We can also output a variable number of arguments from functions.

```
function [y,varargout] = f_varout(a,b)
y = a+b;
if b > a
    varargout{1} = a*b;
    varargout{2} = a^b;
elseif a == b
    varargout{1} = a/b;
end
```

```
>> [y,t] = f_varout(2,3)
y =
     5
t =
     6

>> [y,t1,t2] = f_varout(2,3)
y =
     5
t1 =
     6
t2 =
     8

>> [y,t1,t2,t3] = f_varout(2,3)
Output argument "varargout{3}" (and maybe others)
not assigned during call to "f_varout".
```

Too many outputs requested

```
>> [y,t] = f_varout(2,2)
y =
        4
t =
        1
>> [y,t1,t2] = f_varout(2,2)
Output argument "varargout{2}" (and maybe others)
not assigned during call to "f_varout".
```

Too many outputs requested

Too many outputs requested

```
>> [y,t] = f_varout(3,2)
One or more output arguments not assigned during
call to "varargout".
```

- We can check that the user requests enough output arguments using **nargoutchk**.

```
function [y,varargout] = f_varout(a,b)
y = a+b;
if b > a
    disp('For these inputs we have 2 outputs.')
    nargoutchk(2,2)
    varargout{1} = a*b;
    varargout{2} = a^b;
elseif a == b
    disp('For these inputs we have 1 output.')
    nargoutchk(1,1)
    varargout{1} = a/b;
end
```

The 2 numbers represent minimum & maximum number of arguments

```
Command Window
>> y=f_varout(1,2)
For these inputs we have 2 outputs.
Error using f_varout (line 5)
Not enough output arguments.
```

```
Command Window
>> [y,z]=f_varout(1,2)
For these inputs we have 2 outputs.
y =
     3
z =
     2
```

```
Command Window
>> [y,z]=f_varout(1,1)
For these inputs we have 1 output.
Error using f_varout (line 10)
Too many output arguments.
```

```
Command Window
>> y=f_varout(1,1)
For these inputs we have 1 output.
y =
     2
```

Output of f1([2,3,4])

```
function y = f1(x,varargin)
z=cell2mat(varargin);
if isempty(z)
   y = x;
elseif length(z)==1
   y = x + 1;
else
   y = x*sum(z);
end
```

- 2
- 3
- 4
- 14
- 2 3 4
- 3 4 5

Start the presentation to see live content. For screen share software, share the entire screen. Get help at **pollev.com/app**

Output of f1(2,3,4)

```
function y = f1(x,varargin)
z=cell2mat(varargin);
if isempty(z)
   y = x;
elseif length(z)==1
   y = x + 1;
else
   y = x*sum(z);
end
```

- 2
- 3
- 4
- 14
- 2 3 4
- 3 4 5

To 0

Start the presentation to see live content. For screen share software, share the entire screen. Get help at **pollev.com/app**