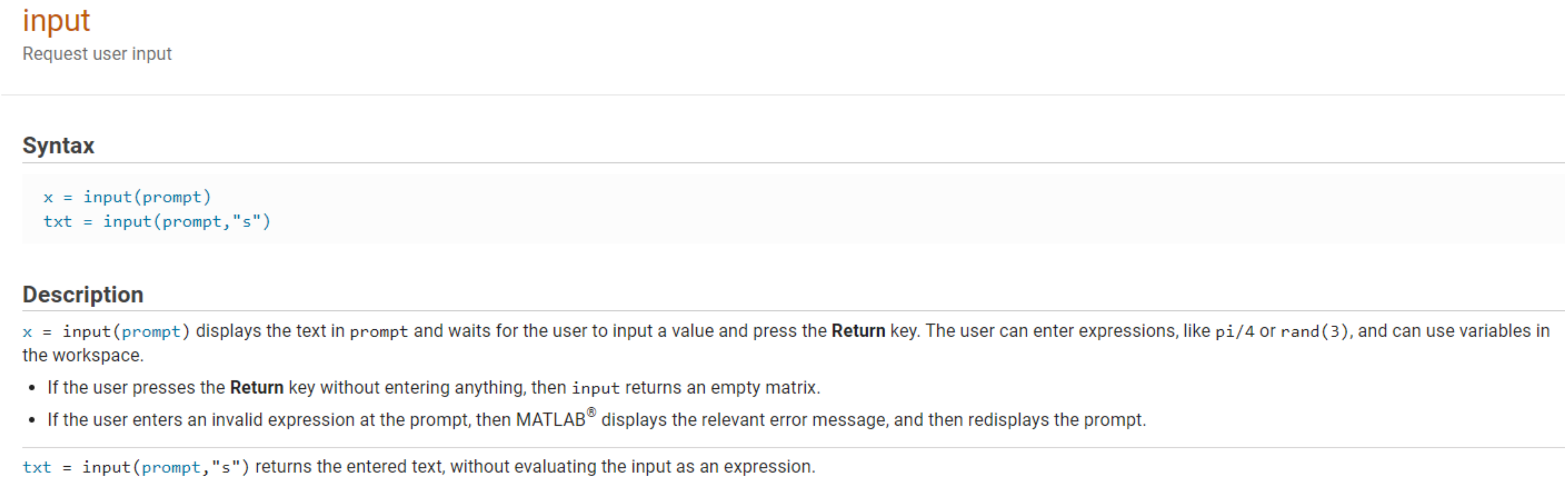
ESERCITAZIONI
MATLAB – Nozioni

User Input



input

Request user input

Syntax

```
x = input(prompt)
txt = input(prompt,"s")
```

Description

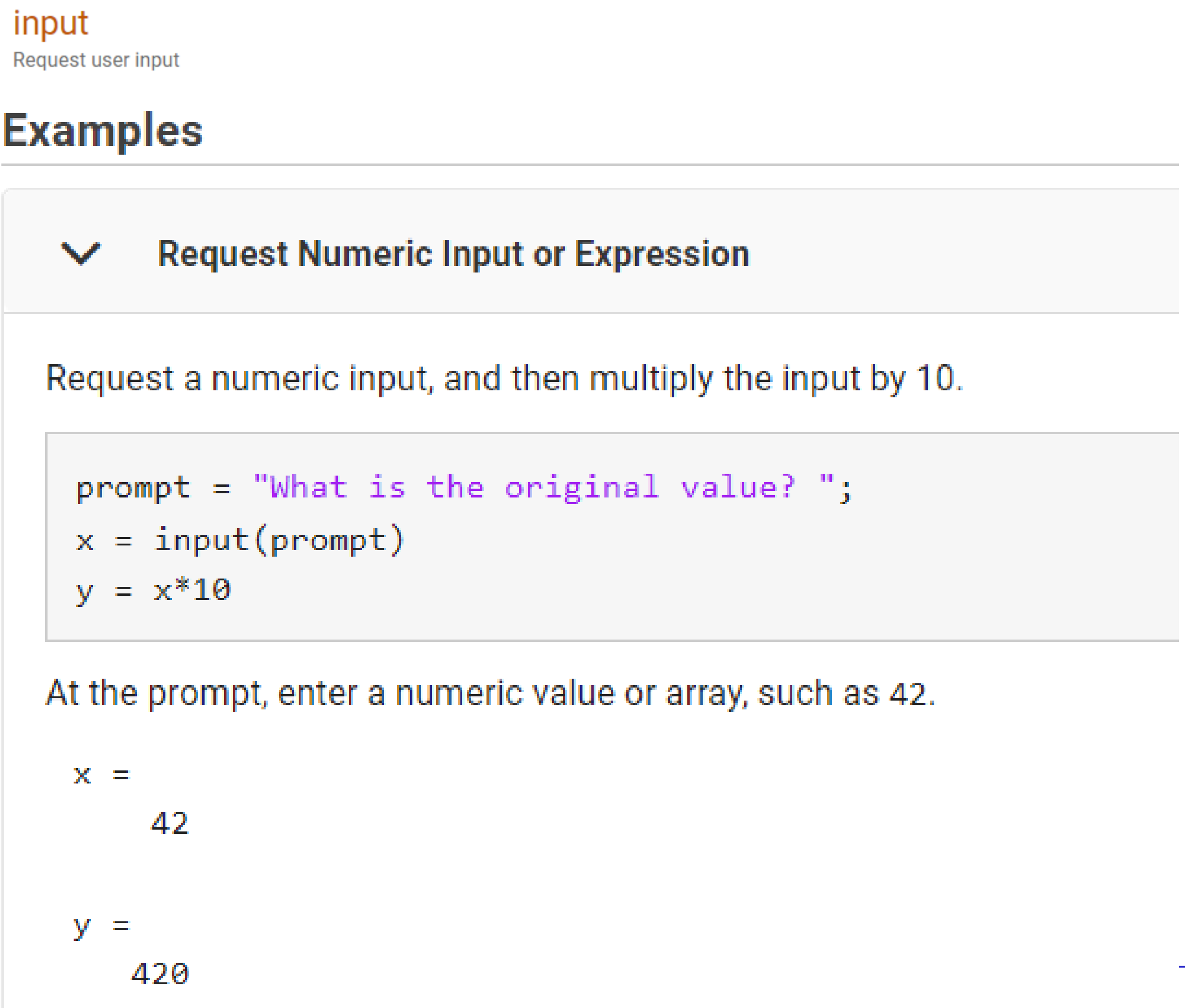
`x = input(prompt)` displays the text in `prompt` and waits for the user to input a value and press the **Return** key. The user can enter expressions, like `pi/4` or `rand(3)`, and can use variables in the workspace.

- If the user presses the **Return** key without entering anything, then `input` returns an empty matrix.
- If the user enters an invalid expression at the prompt, then MATLAB® displays the relevant error message, and then redisplays the prompt.

`txt = input(prompt,"s")` returns the entered text, without evaluating the input as an expression.

ESERCITAZIONI
MATLAB – Nozioni

User Input



input

Request user input

Examples

Request Numeric Input or Expression

Request a numeric input, and then multiply the input by 10.

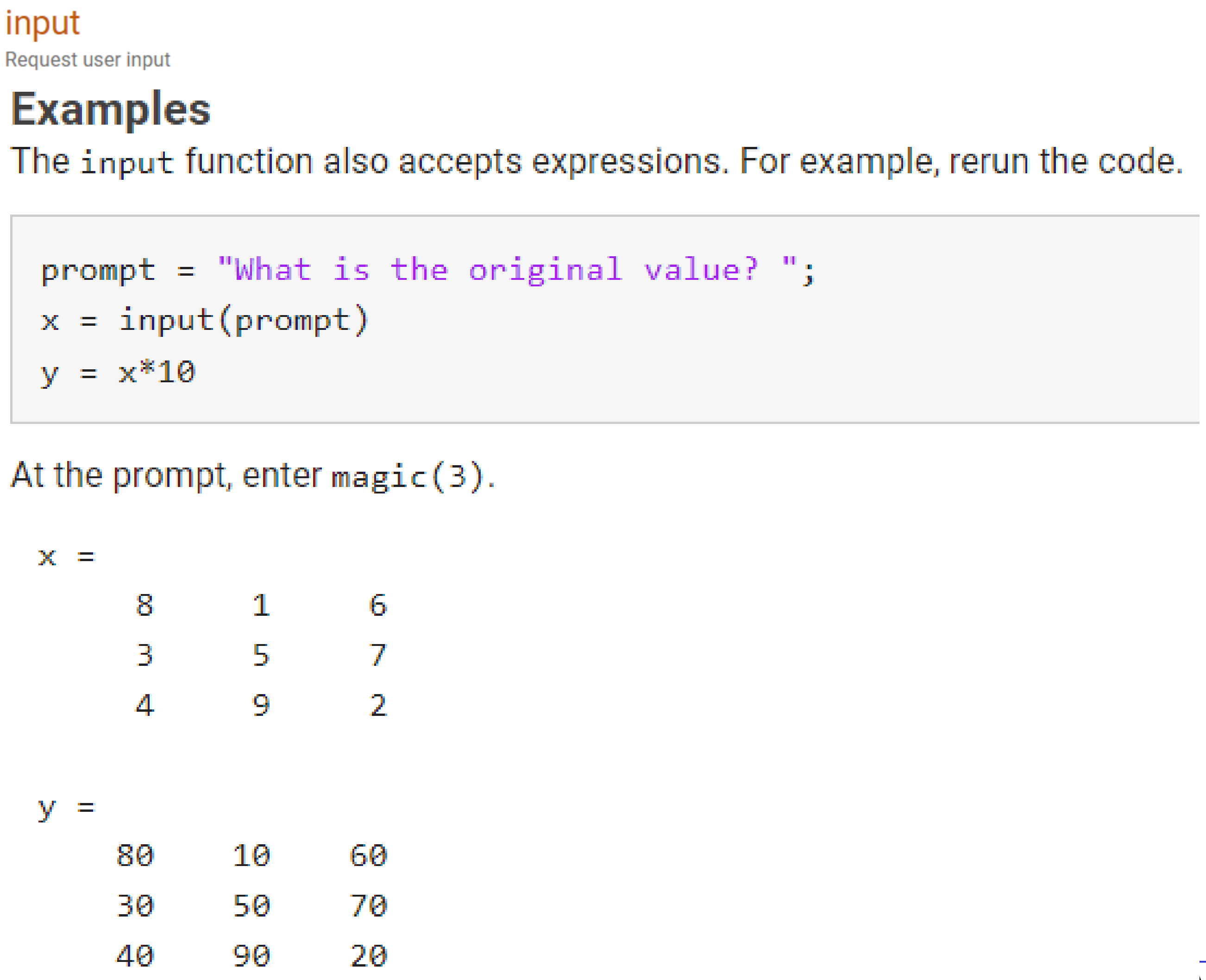
```
prompt = "What is the original value? ";
x = input(prompt)
y = x*10
```

At the prompt, enter a numeric value or array, such as 42.

```
x =

    42


y =

   420
```

ESERCITAZIONI
MATLAB – Nozioni

User Input



input

Request user input

Examples

The `input` function also accepts expressions. For example, rerun the code.

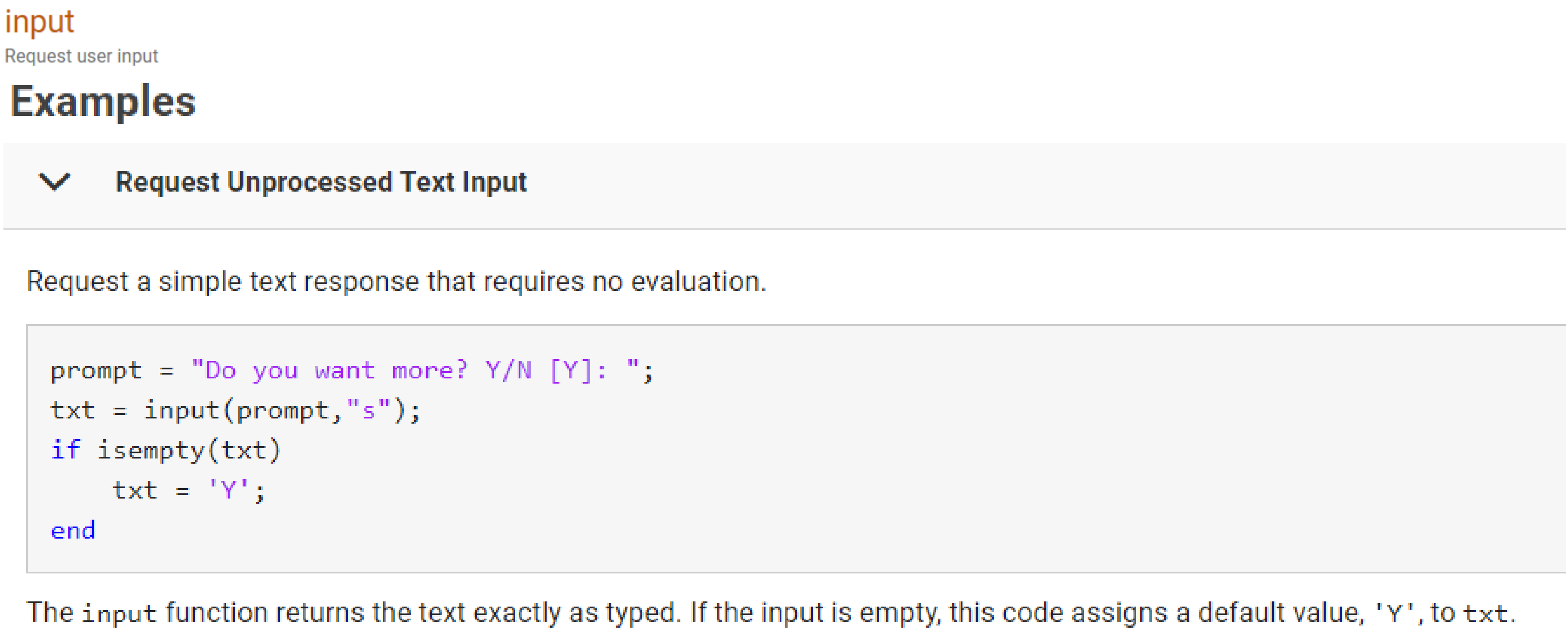
```
prompt = "What is the original value? ";
x = input(prompt)
y = x*10
```

At the prompt, enter `magic(3)`.

```
x =

     8     1     6
     3     5     7
     4     9     2


y =

    80    10    60
    30    50    70
    40    90    20
```

ESERCITAZIONI
MATLAB – Nozioni

User Input



input

Request user input

Examples

Request Unprocessed Text Input

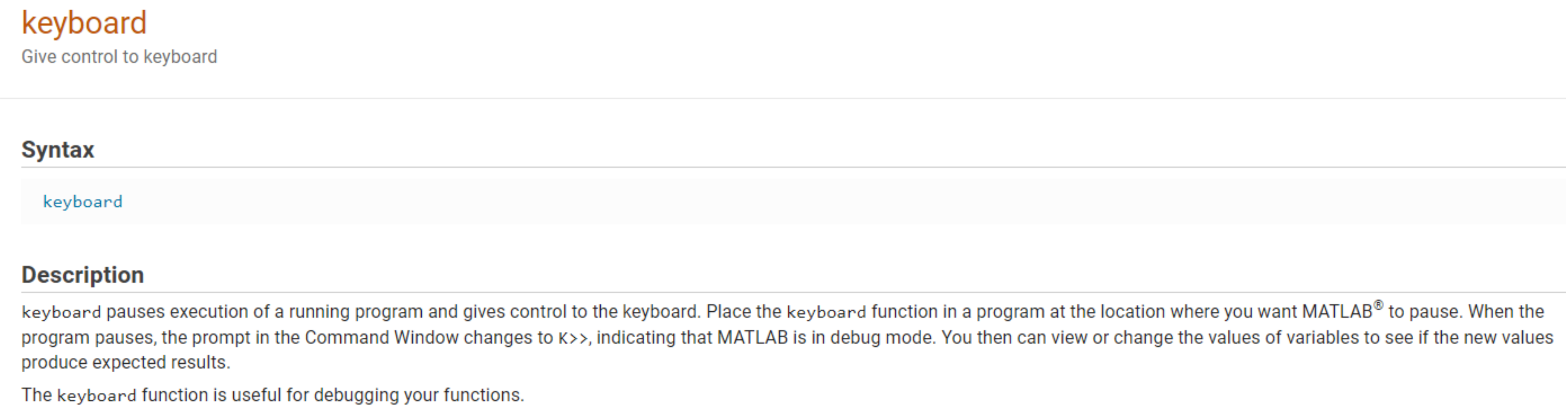
Request a simple text response that requires no evaluation.

```
prompt = "Do you want more? Y/N [Y]: ";
txt = input(prompt,"s");
if isempty(txt)
    txt = 'Y';
end
```

The `input` function returns the text exactly as typed. If the input is empty, this code assigns a default value, `'Y'`, to `txt`.

ESERCITAZIONI
MATLAB – Nozioni

Control to Keyboard



keyboard

Give control to keyboard

Syntax

```
keyboard
```

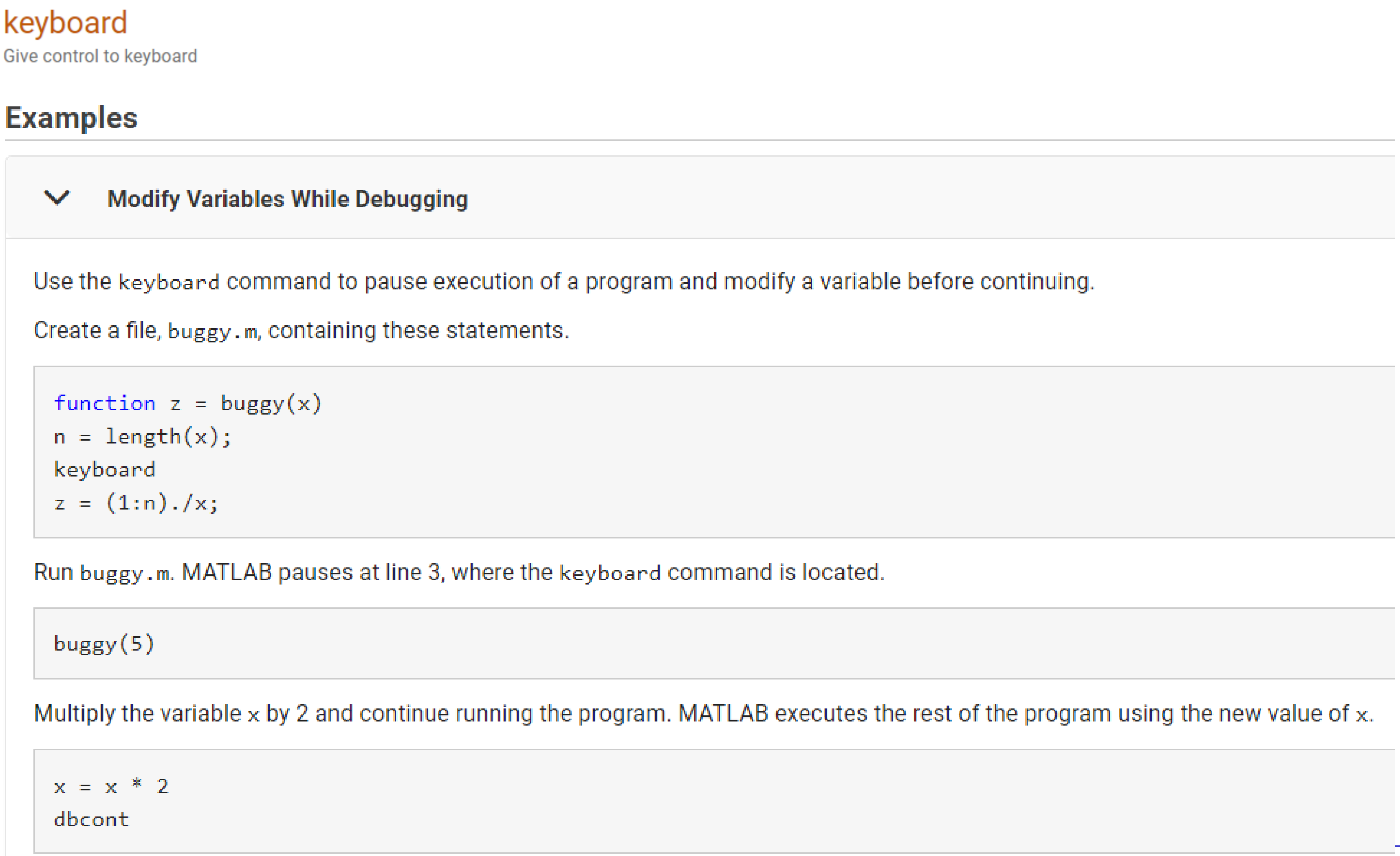
Description

`keyboard` pauses execution of a running program and gives control to the keyboard. Place the `keyboard` function in a program at the location where you want MATLAB® to pause. When the program pauses, the prompt in the Command Window changes to `K>>`, indicating that MATLAB is in debug mode. You then can view or change the values of variables to see if the new values produce expected results.

The `keyboard` function is useful for debugging your functions.

ESERCITAZIONI
MATLAB – Nozioni

Control to Keyboard



keyboard

Give control to keyboard

Examples

Modify Variables While Debugging

Use the `keyboard` command to pause execution of a program and modify a variable before continuing.

Create a file, `buggy.m`, containing these statements.

```
function z = buggy(x)
n = length(x);
keyboard
z = (1:n)./x;
```

Run `buggy.m`. MATLAB pauses at line 3, where the `keyboard` command is located.

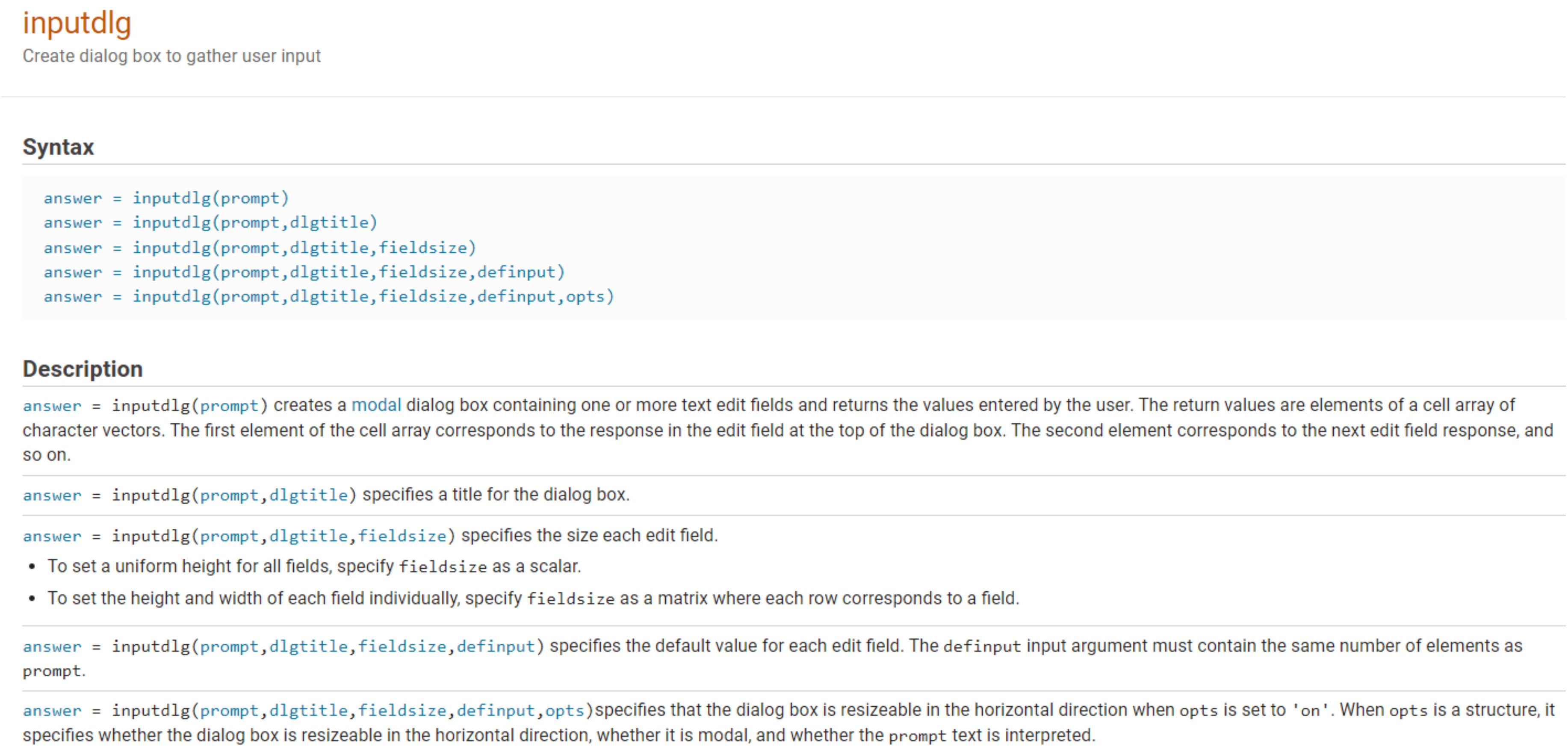
```
buggy(5)
```

Multiply the variable `x` by 2 and continue running the program. MATLAB executes the rest of the program using the new value of `x`.

```
x = x * 2
dbcont
```

ESERCITAZIONI
MATLAB – Nozioni

Dialog Box



inputdlg

Create dialog box to gather user input

Syntax

```
answer = inputdlg(prompt)
answer = inputdlg(prompt,dlgtitle)
answer = inputdlg(prompt,dlgtitle,fieldsize)
answer = inputdlg(prompt,dlgtitle,fieldsize,definput)
answer = inputdlg(prompt,dlgtitle,fieldsize,definput,opts)
```

Description

`answer = inputdlg(prompt)` creates a modal dialog box containing one or more text edit fields and returns the values entered by the user. The return values are elements of a cell array of character vectors. The first element of the cell array corresponds to the response in the edit field at the top of the dialog box. The second element corresponds to the next edit field response, and so on.

`answer = inputdlg(prompt,dlgtitle)` specifies a title for the dialog box.

`answer = inputdlg(prompt,dlgtitle,fieldsize)` specifies the size each edit field.

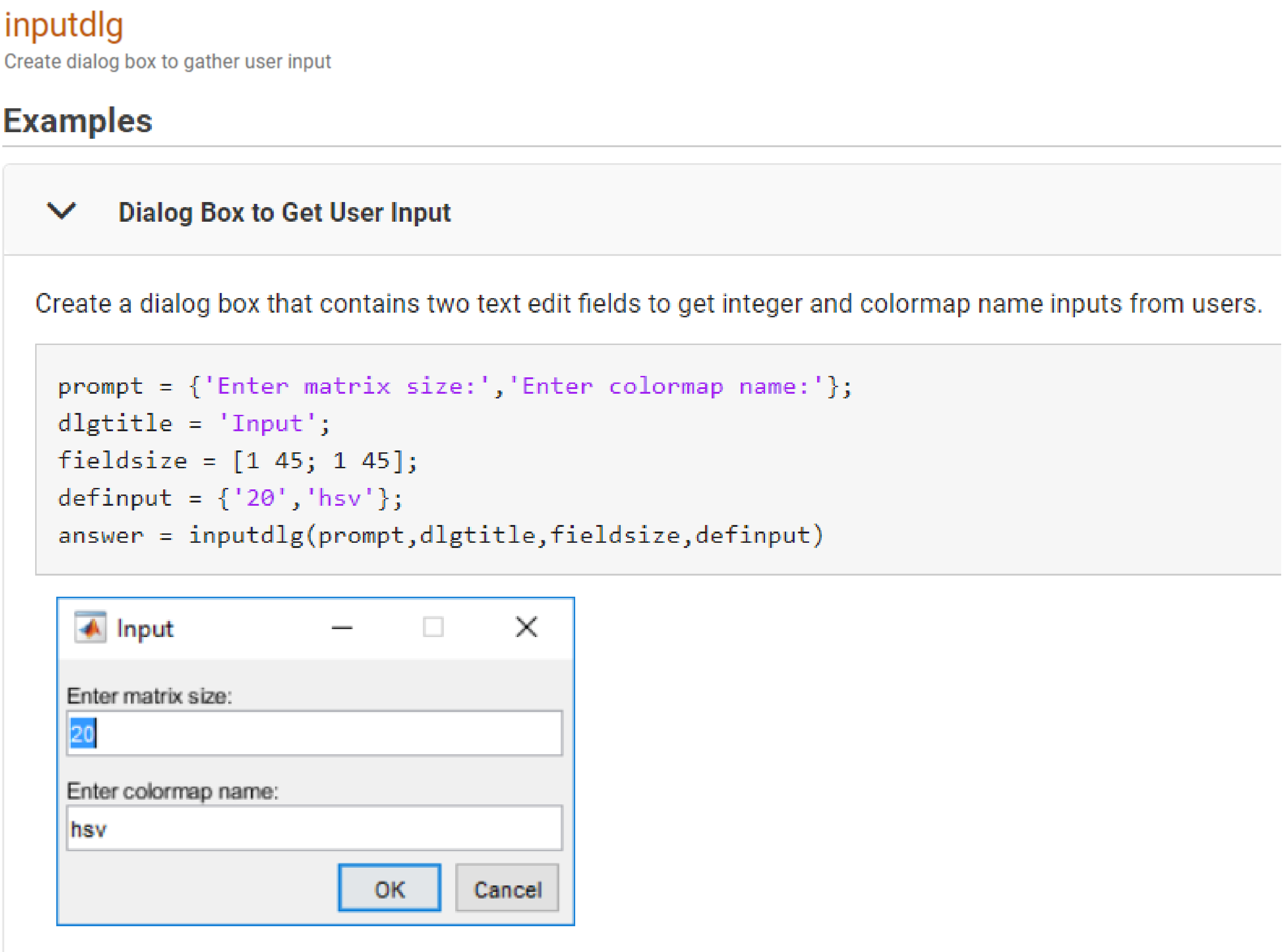
- To set a uniform height for all fields, specify `fieldsize` as a scalar.
- To set the height and width of each field individually, specify `fieldsize` as a matrix where each row corresponds to a field.

`answer = inputdlg(prompt,dlgtitle,fieldsize,definput)` specifies the default value for each edit field. The `definput` input argument must contain the same number of elements as `prompt`.

`answer = inputdlg(prompt,dlgtitle,fieldsize,definput,opts)`specifies that the dialog box is resizeable in the horizontal direction when `opts` is set to `'on'`. When `opts` is a structure, it specifies whether the dialog box is resizeable in the horizontal direction, whether it is modal, and whether the `prompt` text is interpreted.

ESERCITAZIONI
MATLAB – Nozioni

Dialog Box



inputdlg

Create dialog box to gather user input

Examples

Dialog Box to Get User Input

Create a dialog box that contains two text edit fields to get integer and colormap name inputs from users.

```
prompt = {'Enter matrix size:','Enter colormap name:'};
dlgtitle = 'Input';
fieldsize = [1 45; 1 45];
definput = {'20','hsv'};
answer = inputdlg(prompt,dlgtitle,fieldsize,definput)
```

Input

Enter matrix size:

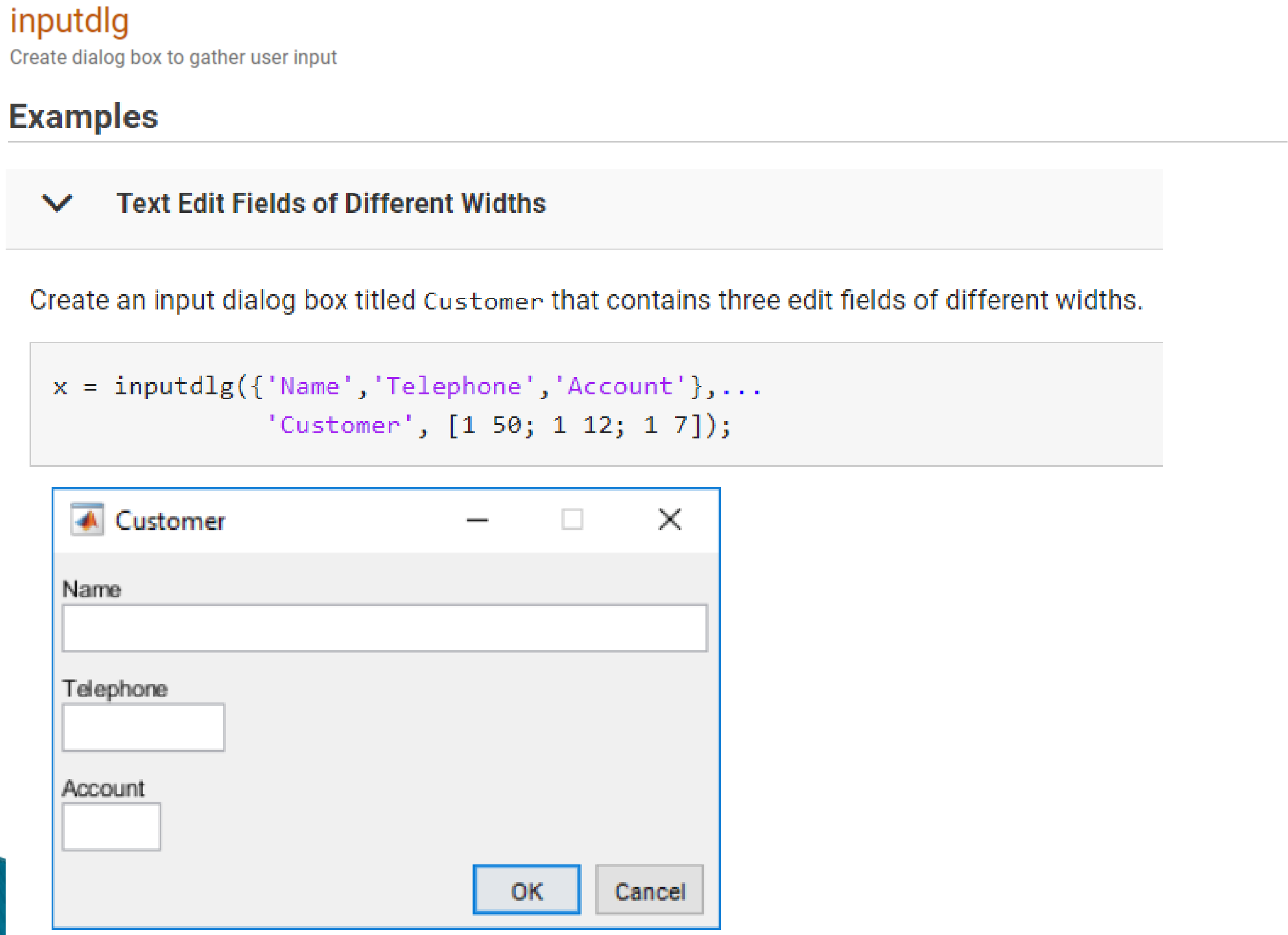
20

Enter colormap name:

hsv

ESERCITAZIONI
MATLAB – Nozioni

Dialog Box



inputdlg

Create dialog box to gather user input

Examples

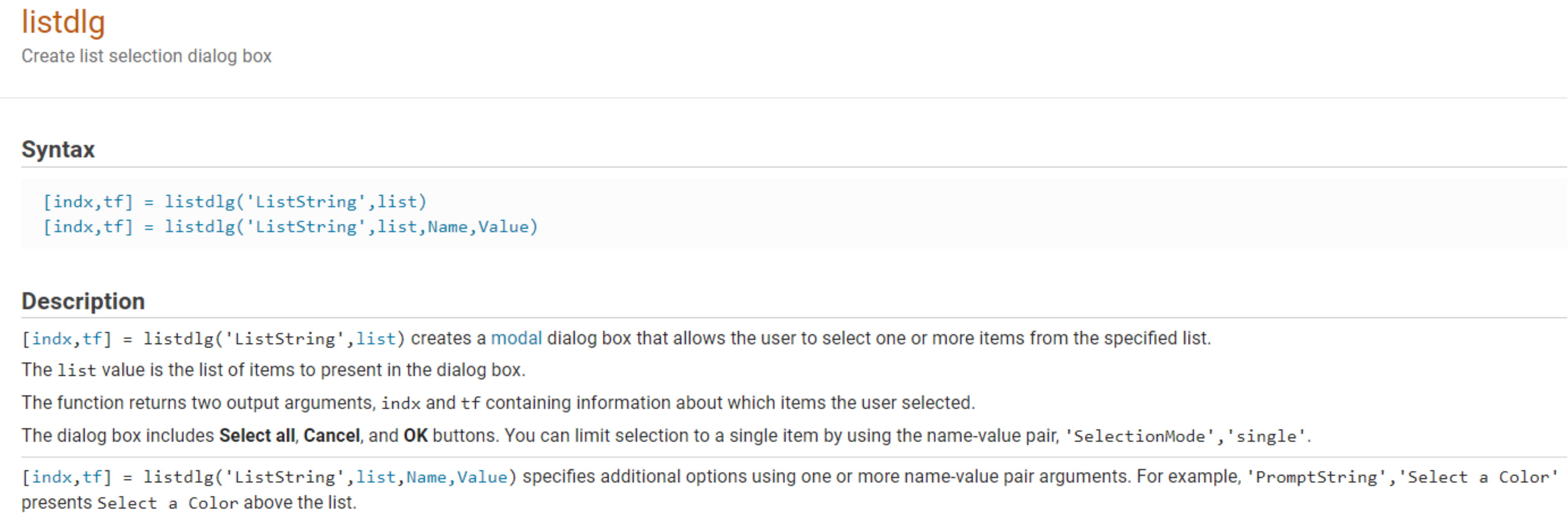
Text Edit Fields of Different Widths

Create an input dialog box titled `Customer` that contains three edit fields of different widths.

```
x = inputdlg({'Name','Telephone','Account'},...
              'Customer', [1 50; 1 12; 1 7]);
```

ESERCITAZIONI MATLAB – Nozioni

Dialog Box



listdlg

Create list selection dialog box

Syntax

```
[indx,tf] = listdlg('ListString',list)
[indx,tf] = listdlg('ListString',list,Name,Value)
```

Description

`[indx,tf] = listdlg('ListString',list)` creates a modal dialog box that allows the user to select one or more items from the specified list.

The `list` value is the list of items to present in the dialog box.

The function returns two output arguments, `indx` and `tf` containing information about which items the user selected.

The dialog box includes **Select all**, **Cancel**, and **OK** buttons. You can limit selection to a single item by using the name-value pair, `'SelectionMode','single'`.

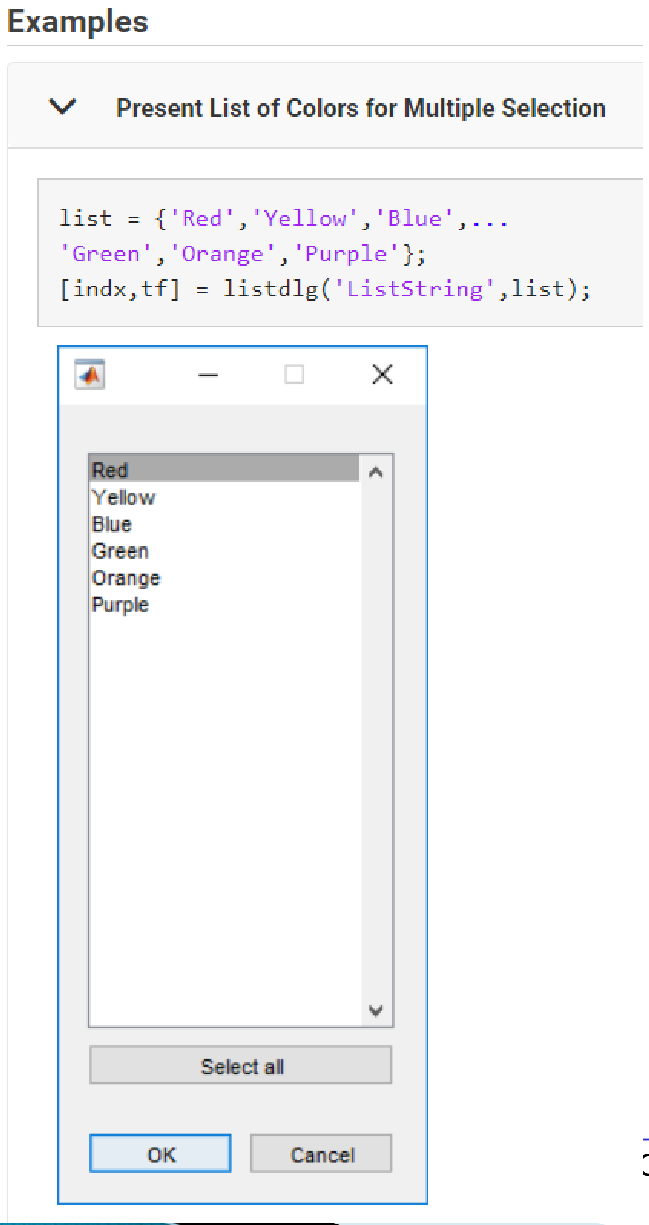
`[indx,tf] = listdlg('ListString',list,Name,Value)` specifies additional options using one or more name-value pair arguments. For example, `'PromptString','Select a Color'` presents `Select a Color` above the list.

ESERCITAZIONI
MATLAB – Nozioni

Dialog Box

listdlg

Create list selection dialog box



Examples

Present List of Colors for Multiple Selection

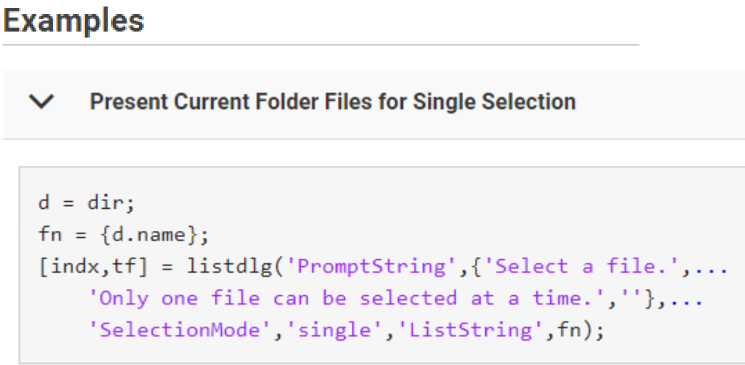
```
list = {'Red','Yellow','Blue',...
'Green','Orange','Purple'};
[indx,tf] = listdlg('ListString',list);
```

ESERCITAZIONI MATLAB – Nozioni

Dialog Box

listdlg

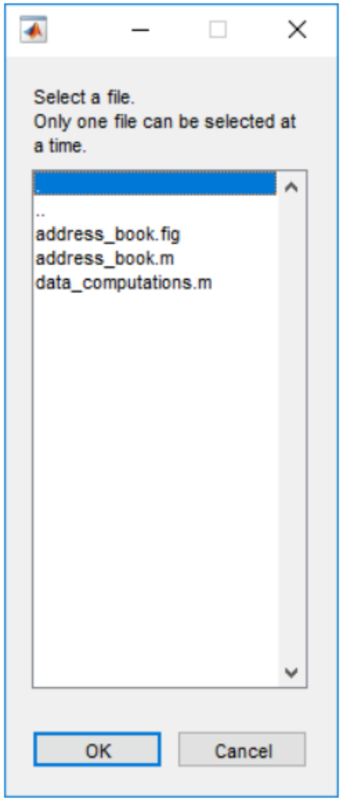
Create list selection dialog box



Examples

Present Current Folder Files for Single Selection

```
d = dir;
fn = {d.name};
[indx,tf] = listdlg('PromptString',{'Select a file.',...
    'Only one file can be selected at a time.',''},...
    'SelectionMode','single','ListString',fn);
```

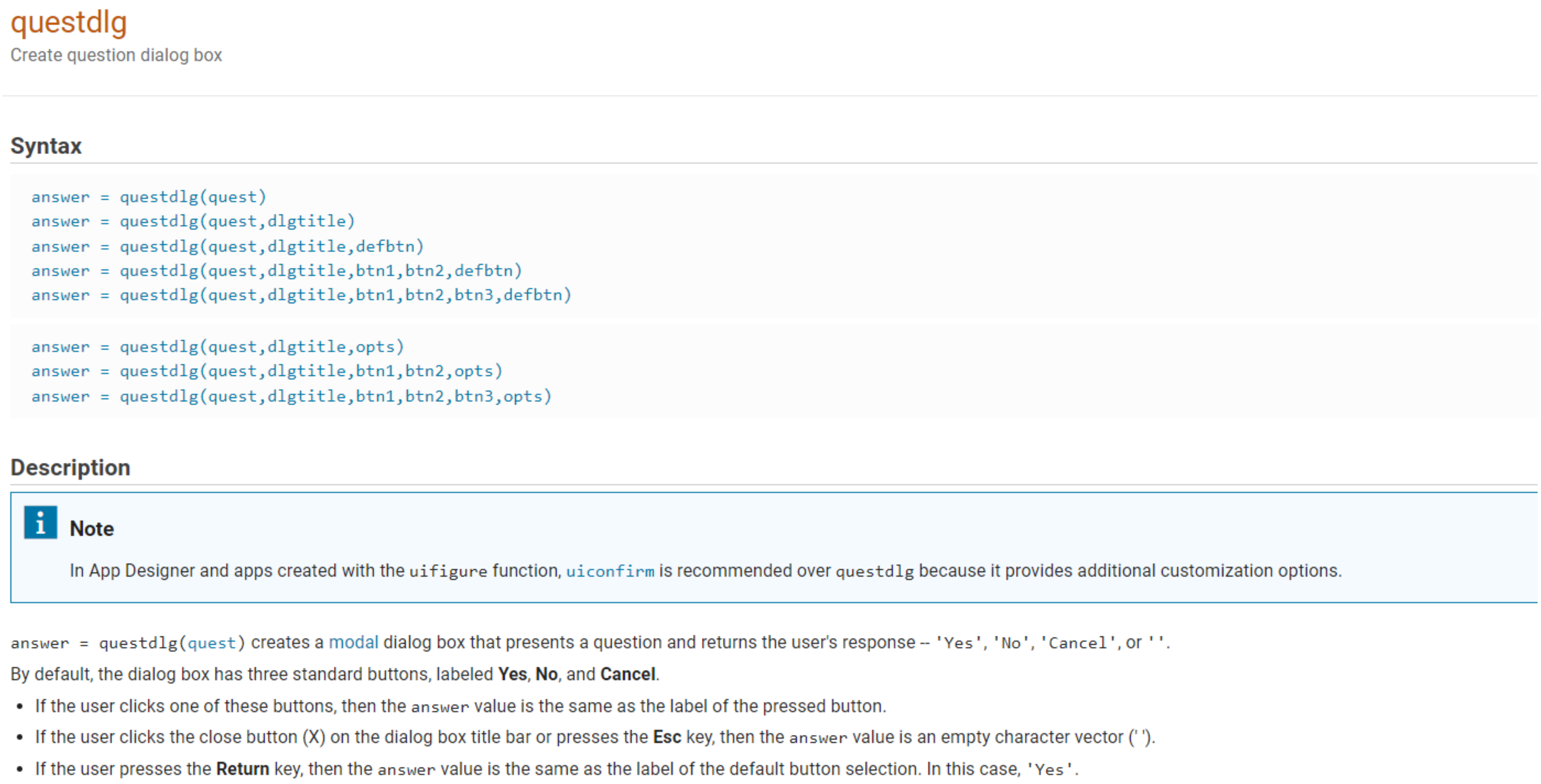


Select a file.
Only one file can be selected at a time.

.
..
address_book.fig
address_book.m
data_computations.m

ESERCITAZIONI
MATLAB – Nozioni

Dialog Box



questdlg

Create question dialog box

Syntax

```
answer = questdlg(quest)
answer = questdlg(quest,dlgtitle)
answer = questdlg(quest,dlgtitle,defbtn)
answer = questdlg(quest,dlgtitle,btn1,btn2,defbtn)
answer = questdlg(quest,dlgtitle,btn1,btn2,btn3,defbtn)
```

```
answer = questdlg(quest,dlgtitle,opts)
answer = questdlg(quest,dlgtitle,btn1,btn2,opts)
answer = questdlg(quest,dlgtitle,btn1,btn2,btn3,opts)
```

Description

i **Note**

In App Designer and apps created with the `uifigure` function, `uiconfirm` is recommended over `questdlg` because it provides additional customization options.

`answer = questdlg(quest)` creates a modal dialog box that presents a question and returns the user's response -- `'Yes'`, `'No'`, `'Cancel'`, or `''`.

By default, the dialog box has three standard buttons, labeled **Yes**, **No**, and **Cancel**.

- If the user clicks one of these buttons, then the `answer` value is the same as the label of the pressed button.
- If the user clicks the close button (X) on the dialog box title bar or presses the **Esc** key, then the `answer` value is an empty character vector (' ').
- If the user presses the **Return** key, then the `answer` value is the same as the label of the default button selection. In this case, `'Yes'`.

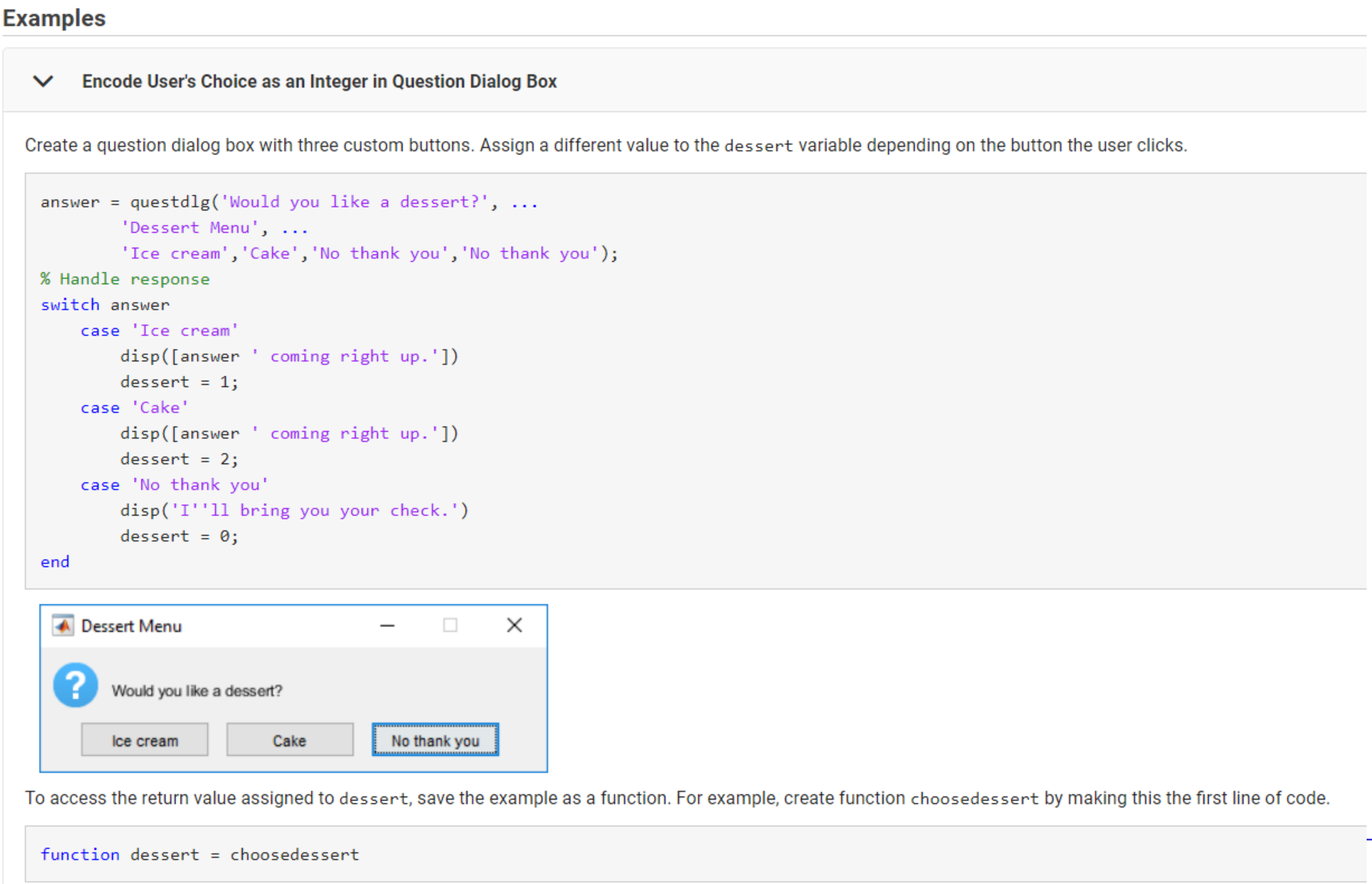
ESERCITAZIONI
MATLAB – Nozioni

MathWorks®

Dialog Box

questdlg

Create question dialog box



Examples

Encode User's Choice as an Integer in Question Dialog Box

Create a question dialog box with three custom buttons. Assign a different value to the dessert variable depending on the button the user clicks.

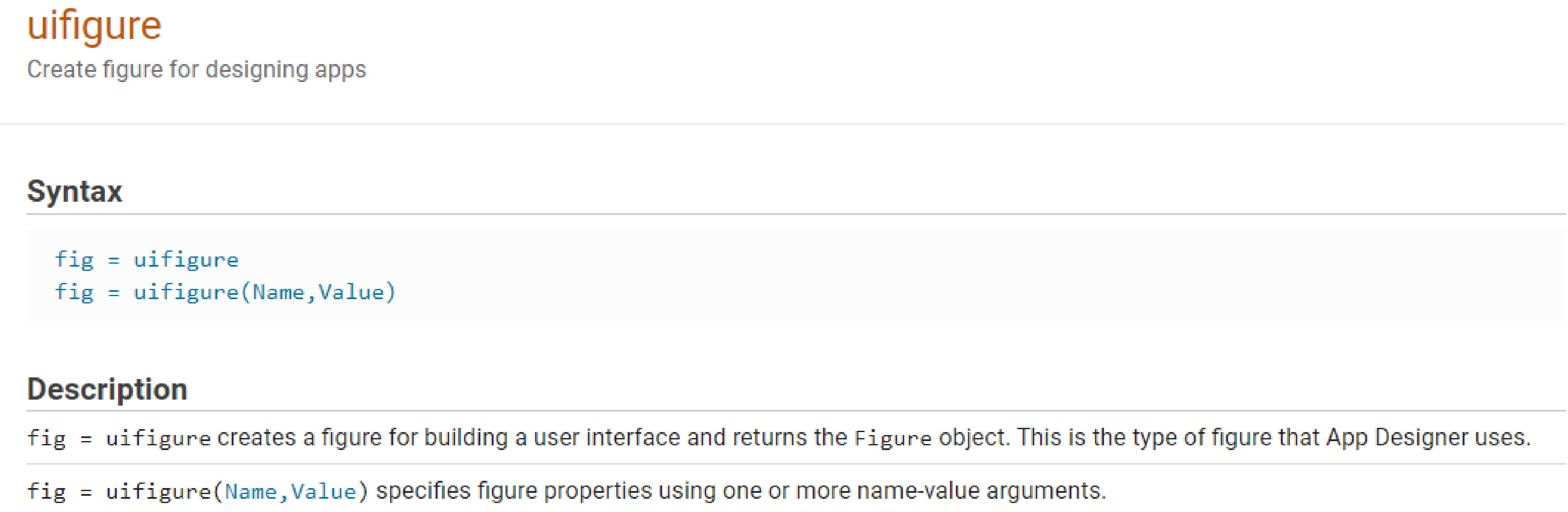
```
answer = questdlg('Would you like a dessert?', ...
        'Dessert Menu', ...
        'Ice cream','Cake','No thank you','No thank you');
% Handle response
switch answer
    case 'Ice cream'
        disp([answer ' coming right up.'])
        dessert = 1;
    case 'Cake'
        disp([answer ' coming right up.'])
        dessert = 2;
    case 'No thank you'
        disp('I''ll bring you your check.')
        dessert = 0;
end
```

To access the return value assigned to dessert, save the example as a function. For example, create function choosedessert by making this the first line of code.

```
function dessert = choosedessert
```

ESERCITAZIONI
MATLAB – Nozioni

User Interface



uifigure

Create figure for designing apps

Syntax

```
fig = uifigure
fig = uifigure(Name,Value)
```

Description

`fig = uifigure` creates a figure for building a user interface and returns the `Figure` object. This is the type of figure that App Designer uses.

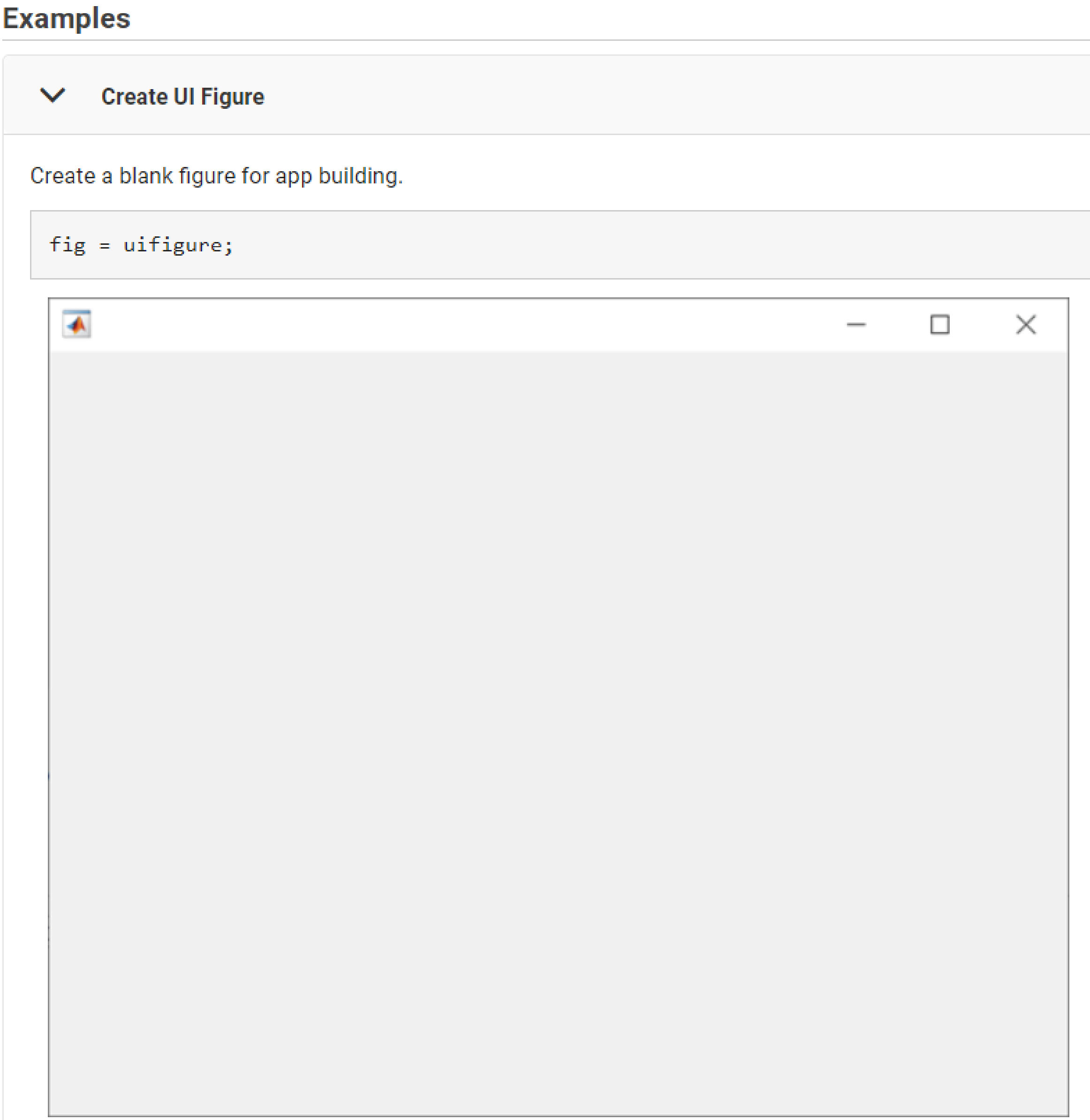
`fig = uifigure(Name,Value)` specifies figure properties using one or more name-value arguments.

ESERCITAZIONI
MATLAB – Nozioni

User Interface

uifigure

Create figure for designing apps



Examples

Create UI Figure

Create a blank figure for app building.

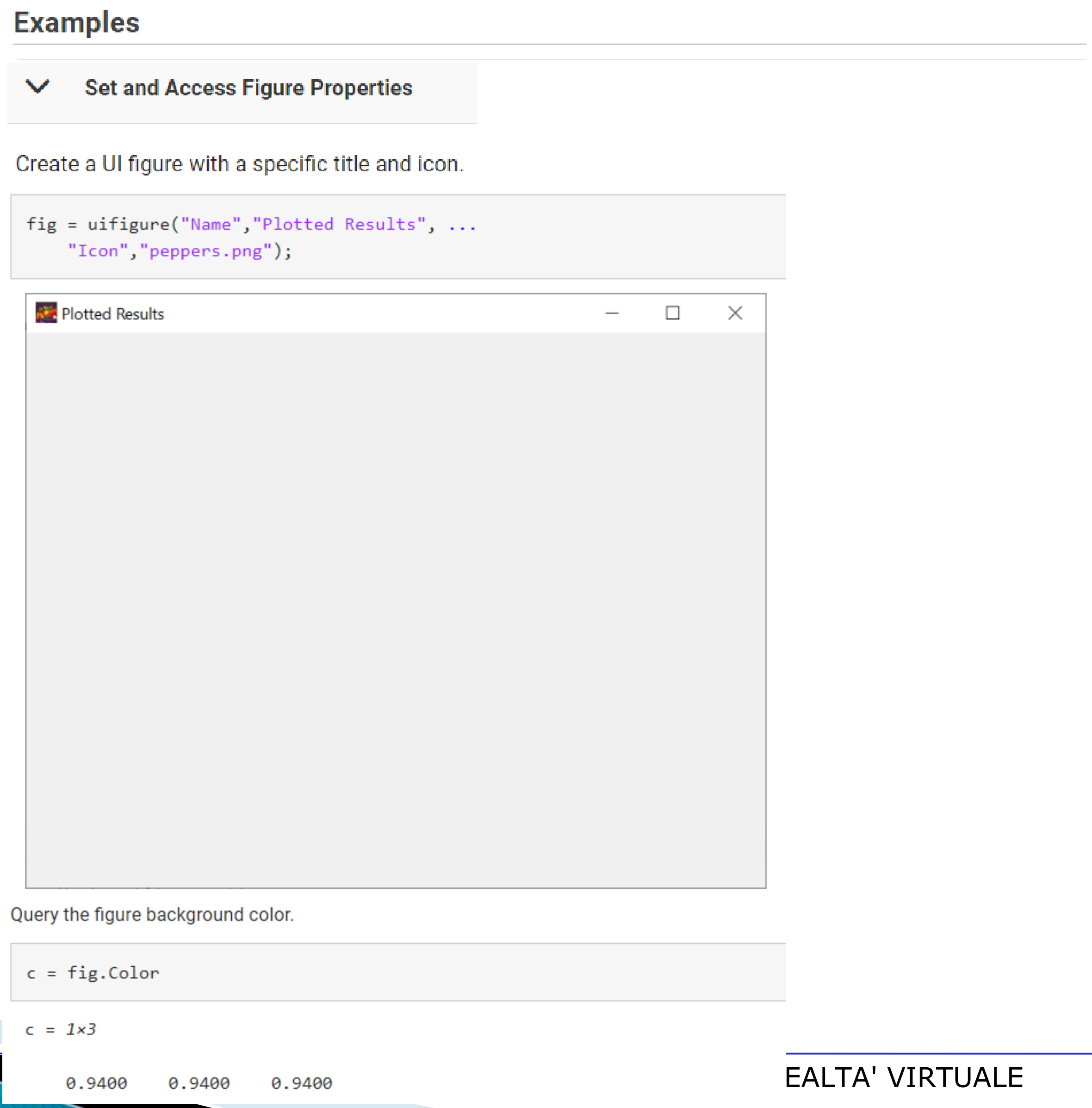
```
fig = uifigure;
```

ESERCITAZIONI
MATLAB – Nozioni

User Interface

uifigure

Create figure for designing apps



Examples

Set and Access Figure Properties

Create a UI figure with a specific title and icon.

```
fig = uifigure("Name","Plotted Results", ...
    "Icon","peppers.png");
```

Query the figure background color.

```
c = fig.Color
```

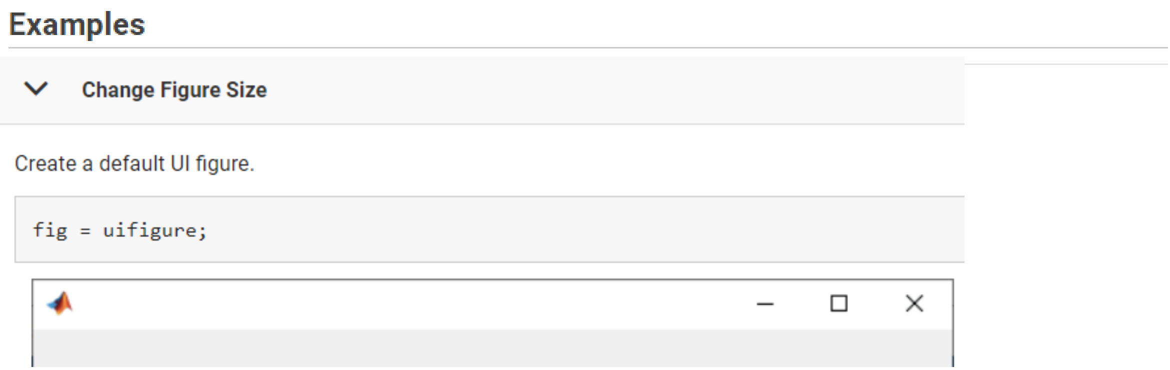
```
c = 1×3

    0.9400    0.9400    0.9400
```

ESERCITAZIONI
MATLAB – Nozioni

User Interface

uifigure

Create figure for designing apps

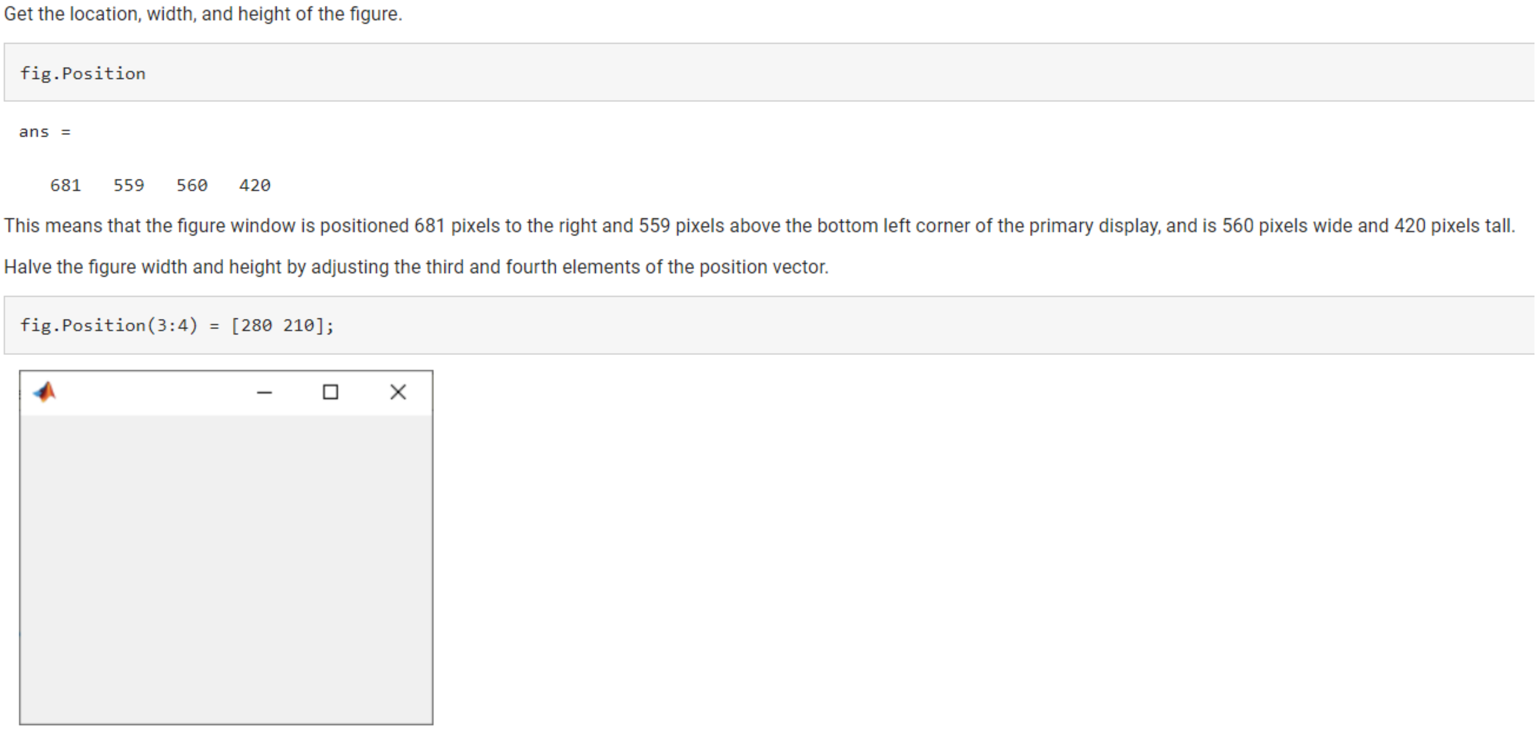


Examples

Change Figure Size

Create a default UI figure.

```
fig = uifigure;
```



Get the location, width, and height of the figure.

```
fig.Position
```

```
ans =

   681   559   560   420
```

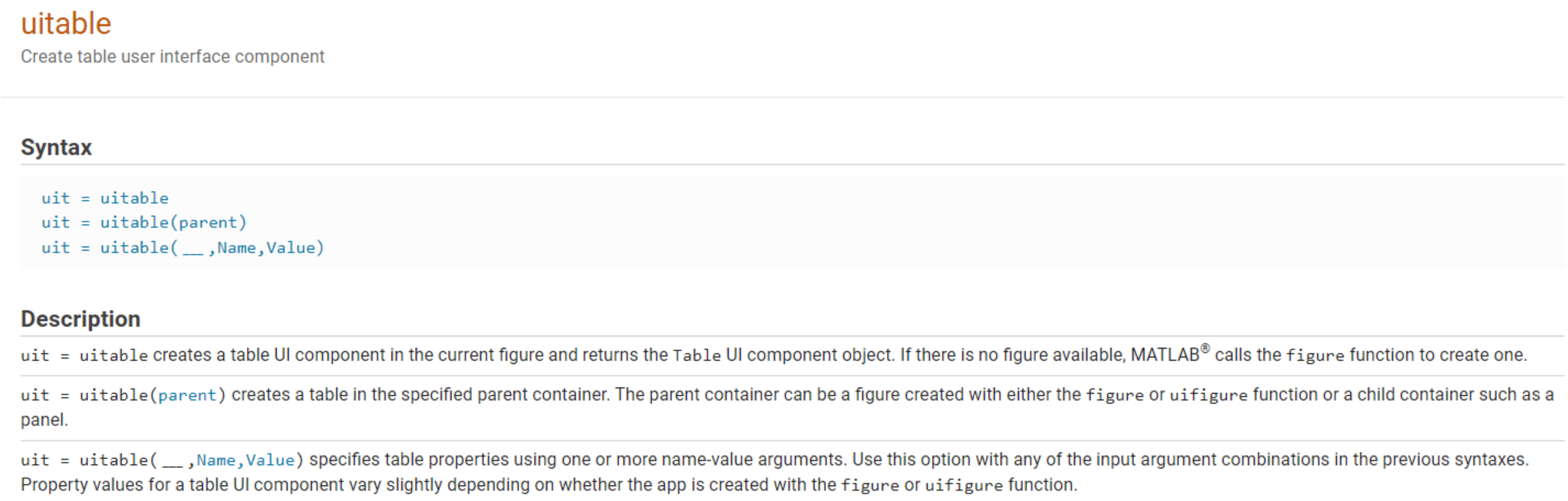
This means that the figure window is positioned 681 pixels to the right and 559 pixels above the bottom left corner of the primary display, and is 560 pixels wide and 420 pixels tall.

Halve the figure width and height by adjusting the third and fourth elements of the position vector.

```
fig.Position(3:4) = [280 210];
```

ESERCITAZIONI
MATLAB – Nozioni

User Interface



uitable

Create table user interface component

Syntax

```
uit = uitable
uit = uitable(parent)
uit = uitable(___,Name,Value)
```

Description

`uit = uitable` creates a table UI component in the current figure and returns the `Table` UI component object. If there is no figure available, MATLAB® calls the `figure` function to create one.

`uit = uitable(parent)` creates a table in the specified parent container. The parent container can be a figure created with either the `figure` or `uifigure` function or a child container such as a panel.

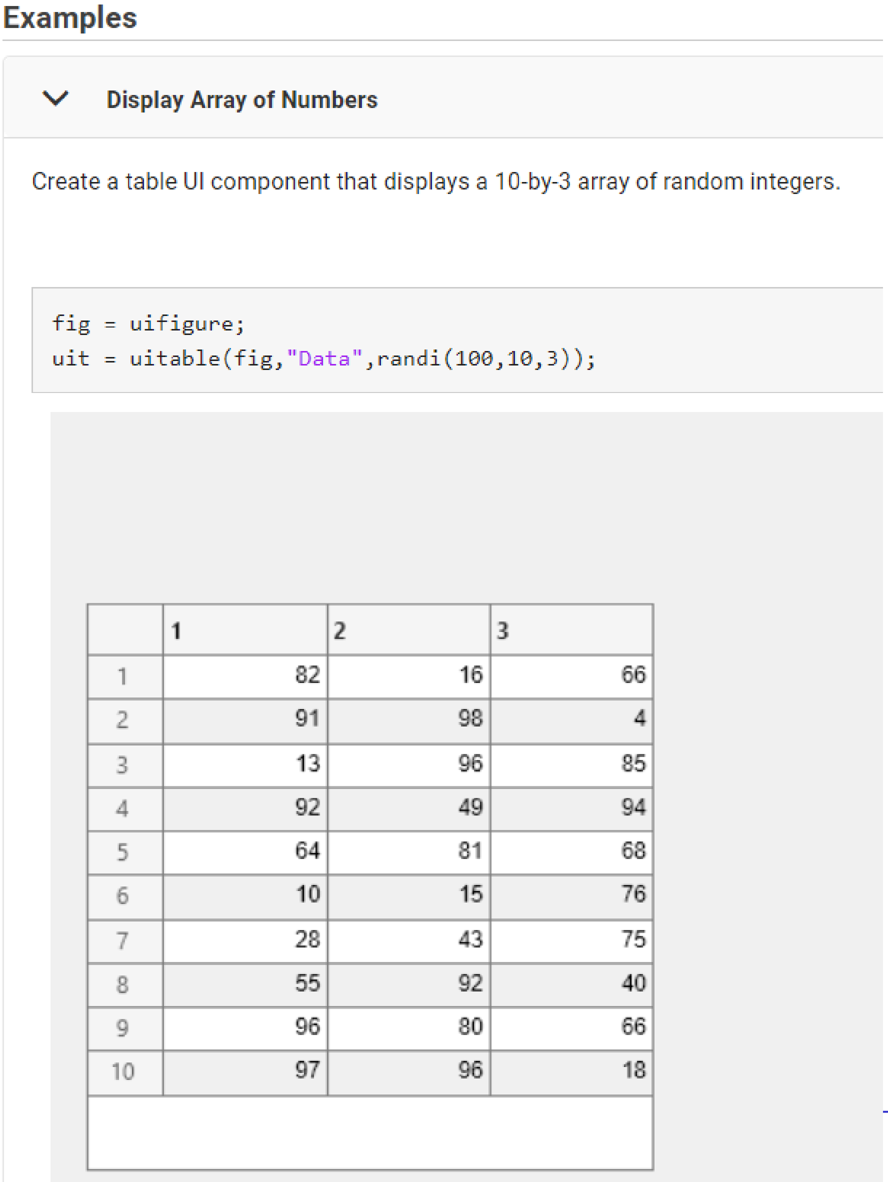
`uit = uitable(___,Name,Value)` specifies table properties using one or more name-value arguments. Use this option with any of the input argument combinations in the previous syntaxes. Property values for a table UI component vary slightly depending on whether the app is created with the `figure` or `uifigure` function.

ESERCITAZIONI MATLAB – Nozioni

User Interface

uitable

Create table user interface component



Examples

Display Array of Numbers

Create a table UI component that displays a 10-by-3 array of random integers.

```
fig = uifigure;
uit = uitable(fig,"Data",randi(100,10,3));
```

	1	2	3
1	82	16	66
2	91	98	4
3	13	96	85
4	92	49	94
5	64	81	68
6	10	15	76
7	28	43	75
8	55	92	40
9	96	80	66
10	97	96	18

ESERCITAZIONI
MATLAB – Nozioni

MathWorks®

User Interface

uitable

Create table user interface component

Examples

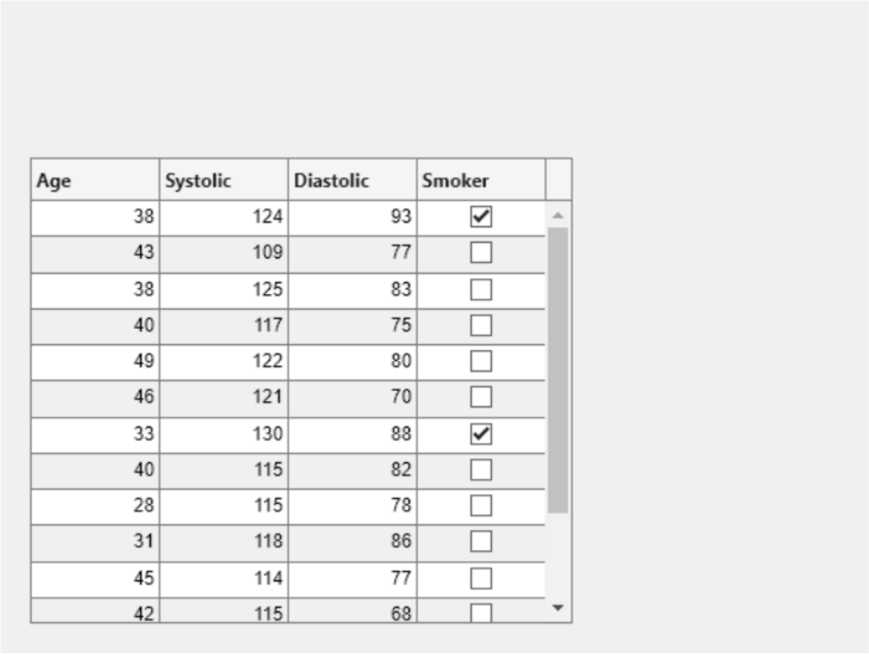
Display Table Data

Create a table array `t` with different data types by reading data from a file. Select the first 15 rows of four variables from `t`.

```
t = readtable("patients.xls");
vars = ["Age","Systolic","Diastolic","Smoker"];
t = t(1:15,vars);
```

Create a table UI component to display the tabular data. The data type determines how the data appears in the component. For example, logical data displays as a check box. For more information, see Format Tabular Data in Apps.

```
fig = uifigure;
uit = uitable(fig,"Data",t,"Position",[20 20 350 300]);
```

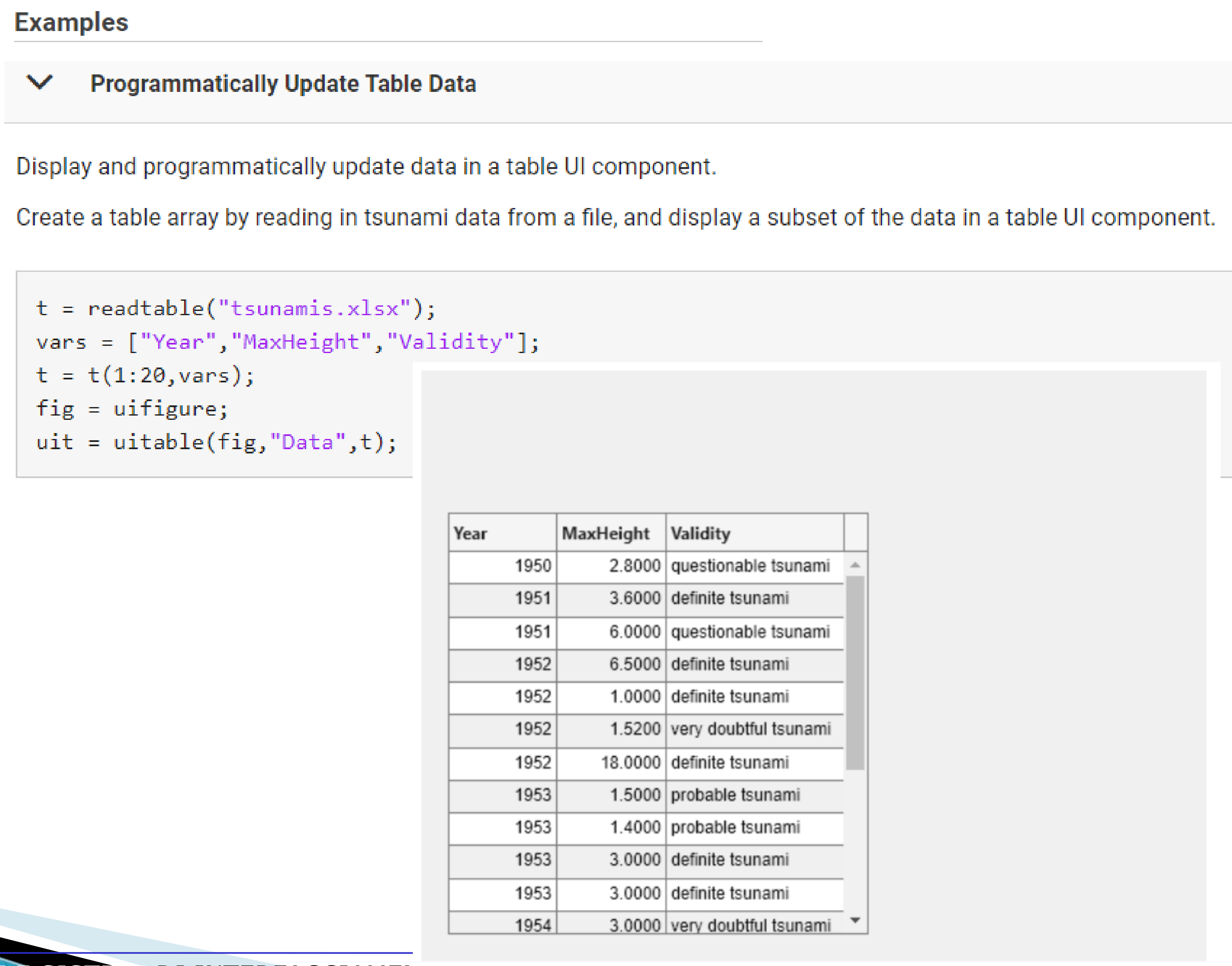


Age	Systolic	Diastolic	Smoker
38	124	93	☑
43	109	77	☐
38	125	83	☐
40	117	75	☐
49	122	80	☐
46	121	70	☐
33	130	88	☑
40	115	82	☐
28	115	78	☐
31	118	86	☐
45	114	77	☐
42	115	68	☐

User Interface

uitable

Create table user interface component



Examples

Programmatically Update Table Data

Display and programmatically update data in a table UI component.

Create a table array by reading in tsunami data from a file, and display a subset of the data in a table UI component.

```
t = readtable("tsunamis.xlsx");
vars = ["Year","MaxHeight","Validity"];
t = t(1:20,vars);
fig = uifigure;
uit = uitable(fig,"Data",t);
```

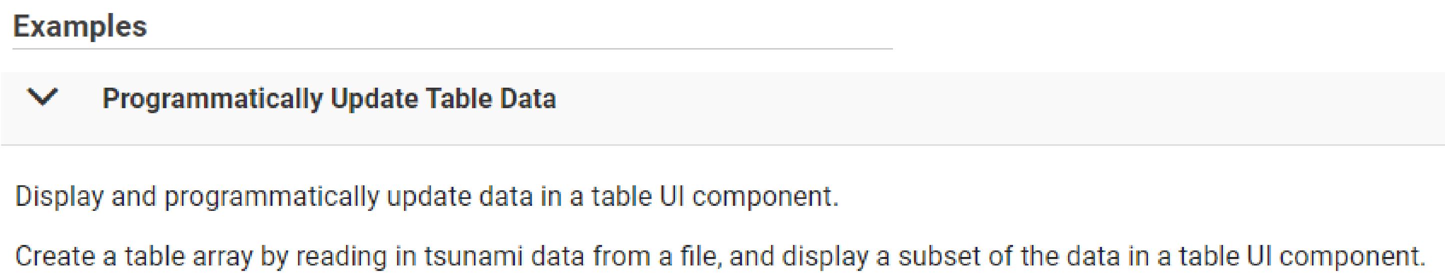
Year	MaxHeight	Validity
1950	2.8000	questionable tsunami
1951	3.6000	definite tsunami
1951	6.0000	questionable tsunami
1952	6.5000	definite tsunami
1952	1.0000	definite tsunami
1952	1.5200	very doubtful tsunami
1952	18.0000	definite tsunami
1953	1.5000	probable tsunami
1953	1.4000	probable tsunami
1953	3.0000	definite tsunami
1953	3.0000	definite tsunami
1954	3.0000	very doubtful tsunami

ESERCITAZIONI
MATLAB – Nozioni

User Interface

uitable

Create table user interface component

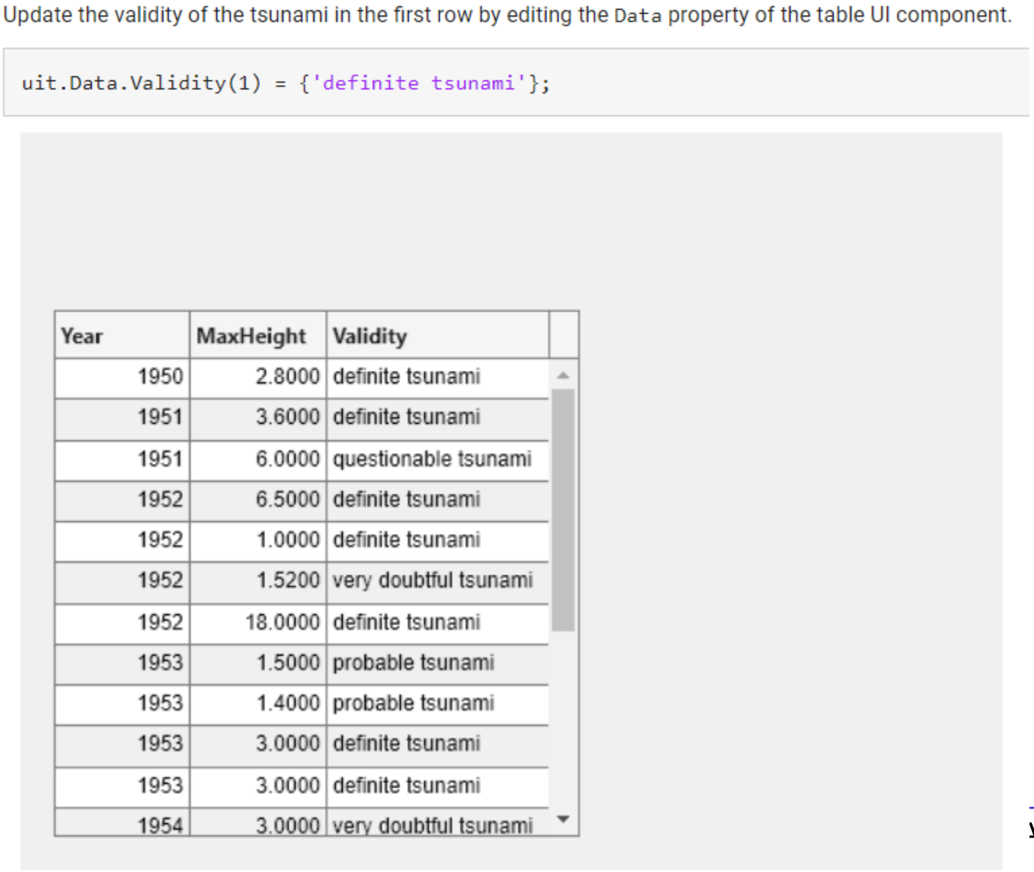


Examples

Programmatically Update Table Data

Display and programmatically update data in a table UI component.

Create a table array by reading in tsunami data from a file, and display a subset of the data in a table UI component.



Update the validity of the tsunami in the first row by editing the `Data` property of the table UI component.

```
uit.Data.Validity(1) = {'definite tsunami'};
```

Year	MaxHeight	Validity
1950	2.8000	definite tsunami
1951	3.6000	definite tsunami
1951	6.0000	questionable tsunami
1952	6.5000	definite tsunami
1952	1.0000	definite tsunami
1952	1.5200	very doubtful tsunami
1952	18.0000	definite tsunami
1953	1.5000	probable tsunami
1953	1.4000	probable tsunami
1953	3.0000	definite tsunami
1953	3.0000	definite tsunami
1954	3.0000	very doubtful tsunami

ESERCITAZIONI
MATLAB – Nozioni

User Interface

uitable

Create table user interface component

Examples

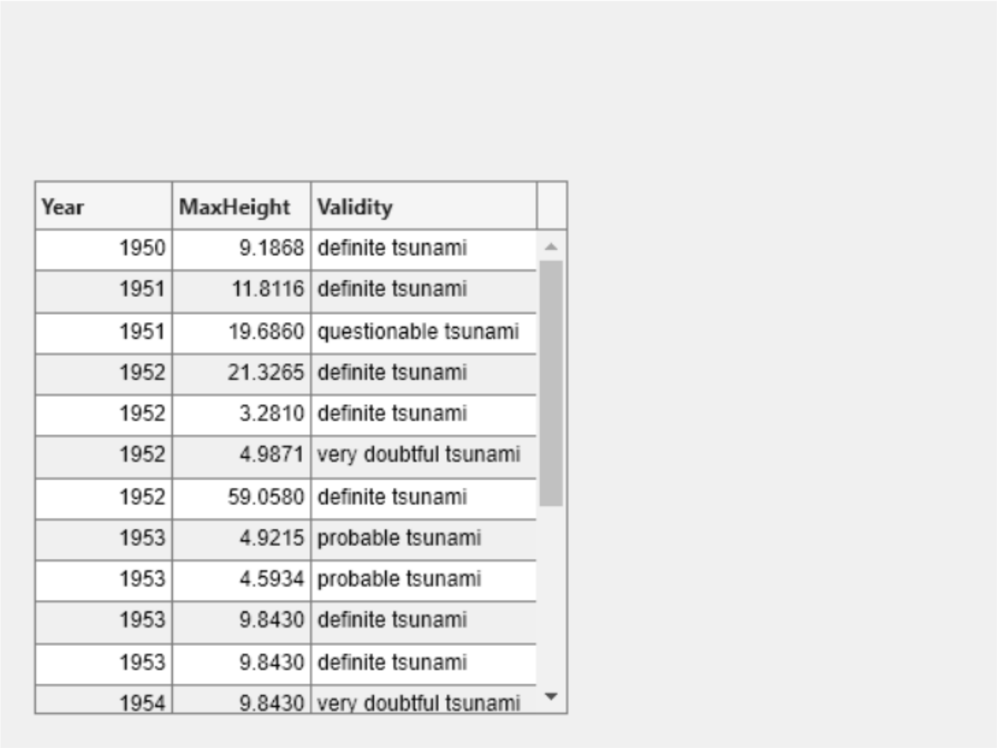
Programmatically Update Table Data

Display and programmatically update data in a table UI component.

Create a table array by reading in tsunami data from a file, and display a subset of the data in a table UI component.

Convert the maximum height data from meters to feet by accessing and modifying the data in the `MaxHeight` variable.

```
uit.Data.MaxHeight = uit.Data.MaxHeight*3.281;
```



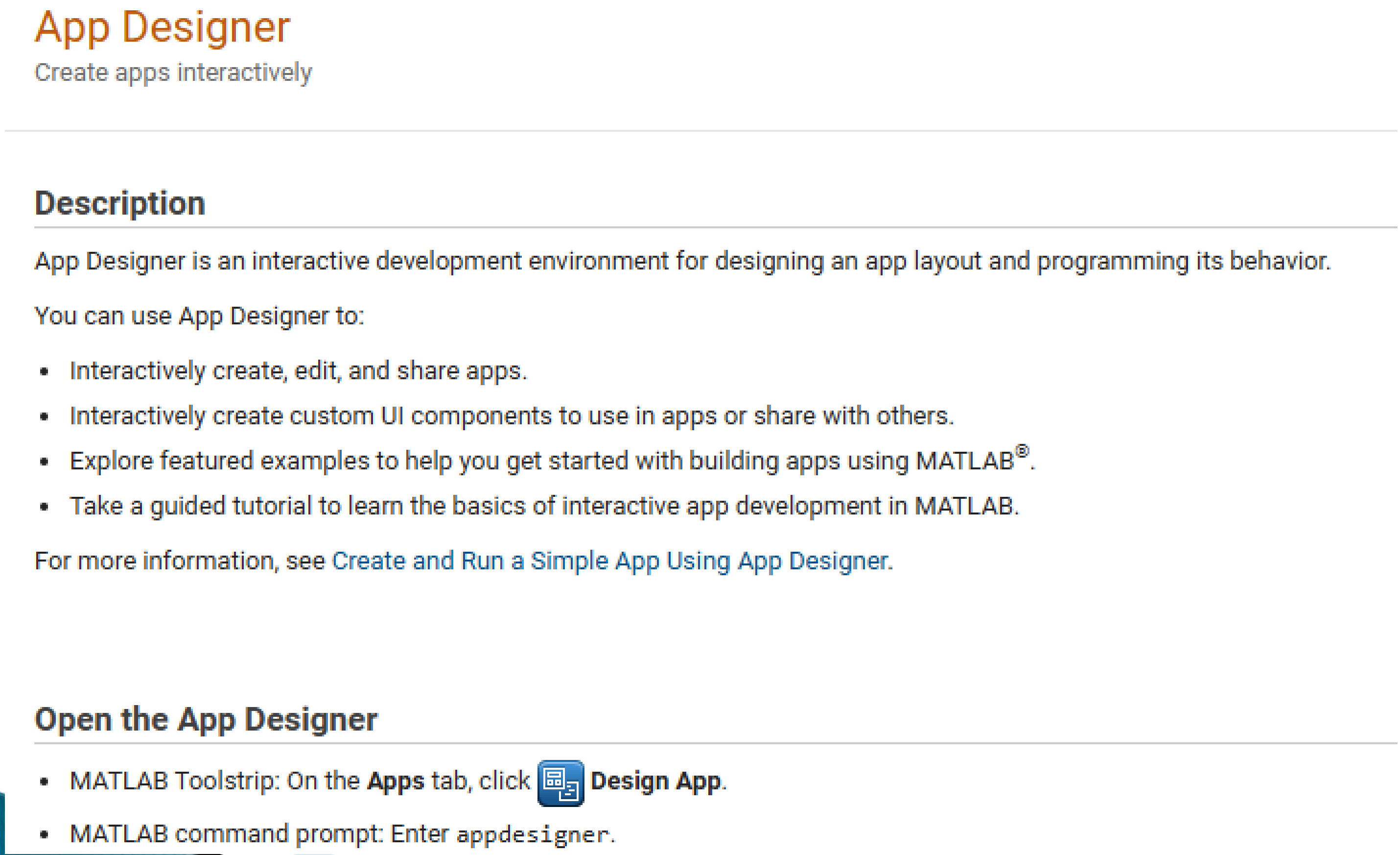
Year	MaxHeight	Validity
1950	9.1868	definite tsunami
1951	11.8116	definite tsunami
1951	19.6860	questionable tsunami
1952	21.3265	definite tsunami
1952	3.2810	definite tsunami
1952	4.9871	very doubtful tsunami
1952	59.0580	definite tsunami
1953	4.9215	probable tsunami
1953	4.5934	probable tsunami
1953	9.8430	definite tsunami
1953	9.8430	definite tsunami
1954	9.8430	very doubtful tsunami

Altre funzioni consigliate:
uibutton
uicontrol
uilabel

ESERCITAZIONI
MATLAB – Nozioni

App Designer

https://it.mathworks.com/help/matlab/ref/appdesigner.html



App Designer

Create apps interactively

Description

App Designer is an interactive development environment for designing an app layout and programming its behavior.

You can use App Designer to:

- Interactively create, edit, and share apps.
- Interactively create custom UI components to use in apps or share with others.
- Explore featured examples to help you get started with building apps using MATLAB®.
- Take a guided tutorial to learn the basics of interactive app development in MATLAB.

For more information, see Create and Run a Simple App Using App Designer.

Open the App Designer

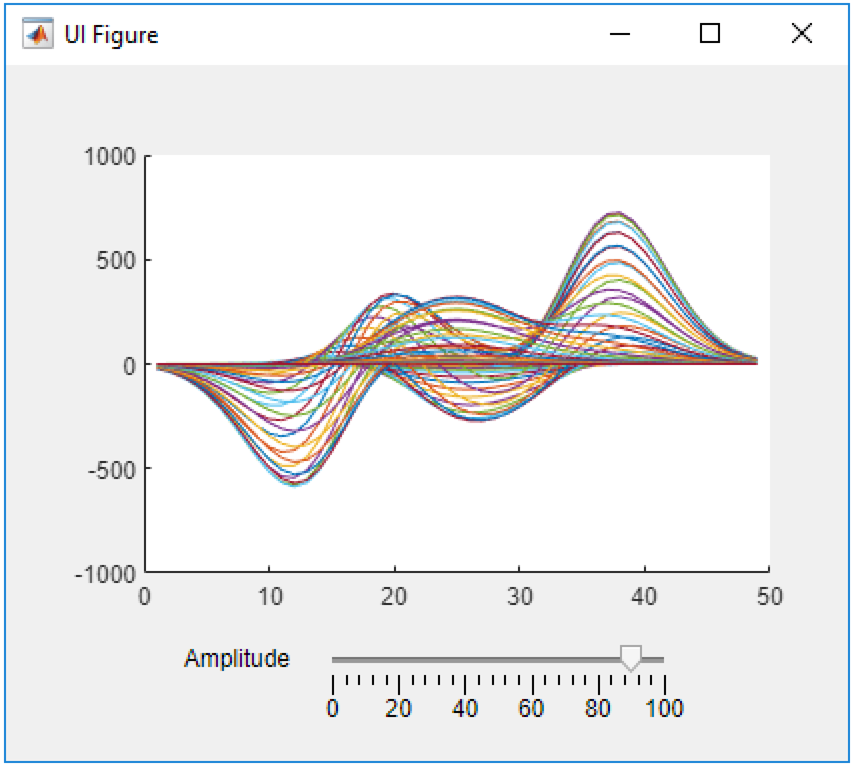
- MATLAB Toolstrip: On the **Apps** tab, click **Design App**.
- MATLAB command prompt: Enter `appdesigner`.

App Designer

Create and Run a Simple App Using App Designer

App Designer provides a tutorial that guides you through the process of creating a simple app containing a plot and a slider.

The slider controls the amplitude of the plotted function. You can create this app by running the tutorial, or you can follow the tutorial steps listed here.



Run the Tutorial

To run the tutorial in App Designer, open the App Designer Start Page and click **Show examples** in the **Apps** section. Then, select **Interactive Tutorial**.

ESERCITAZIONI
MATLAB – Nozioni

App Designer

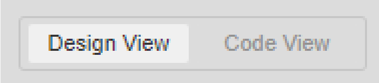
Create and Run a Simple App Using App Designer

Tutorial Steps for Creating the App

App Designer has two views for creating an app: **Design View** and **Code View**.

Use **Design View** to create UI components and interactively lay out your app.

Use **Code View** to program your app behavior. You can switch between the two views using the toggle buttons in the upper right-corner of App Designer.



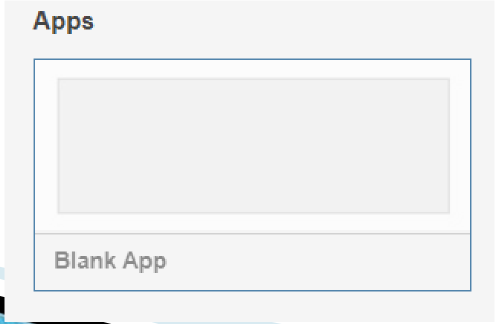
To create the simple plotting app, open a new app in App Designer and follow these steps.

Step 1: Create an Axes Component

In **Design View**, create UI components and modify their appearance interactively. The **Component Library** contains all components, containers, and tools that you can add to your app interactively. Add a component by dragging it from the **Component Library** onto the app canvas.
You can then change the appearance of the component by setting properties in the **Component Browser**,
or by editing certain aspects of the component, such as size and label text, directly on the canvas.

In your plotting app, create an axes component to display plotted data. Drag an **Axes** component from the **Component Library** onto the canvas.

```
>> appdesigner
```



App Designer

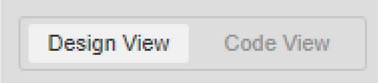
Create and Run a Simple App Using App Designer

Tutorial Steps for Creating the App

App Designer has two views for creating an app: **Design View** and **Code View**.

Use **Design View** to create UI components and interactively lay out your app.

Use **Code View** to program your app behavior. You can switch between the two views using the toggle buttons in the upper right-corner of App Designer.

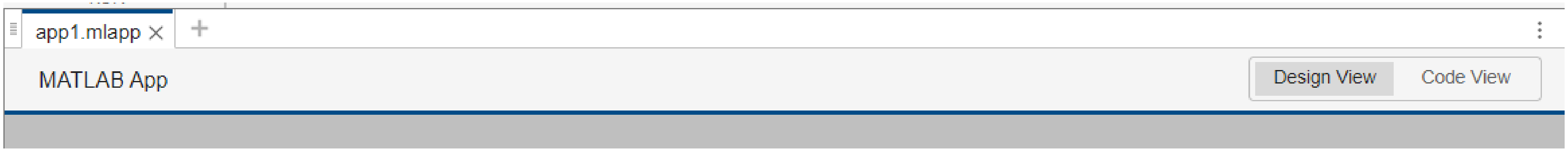


To create the simple plotting app, open a new app in App Designer and follow these steps.

Step 1: Create an Axes Component

In **Design View**, create UI components and modify their appearance interactively. The **Component Library** contains all components, containers, and tools that you can add to your app interactively. Add a component by dragging it from the **Component Library** onto the app canvas.
You can then change the appearance of the component by setting properties in the **Component Browser**,
or by editing certain aspects of the component, such as size and label text, directly on the canvas.

In your plotting app, create an axes component to display plotted data. Drag an **Axes** component from the **Component Library** onto the canvas.

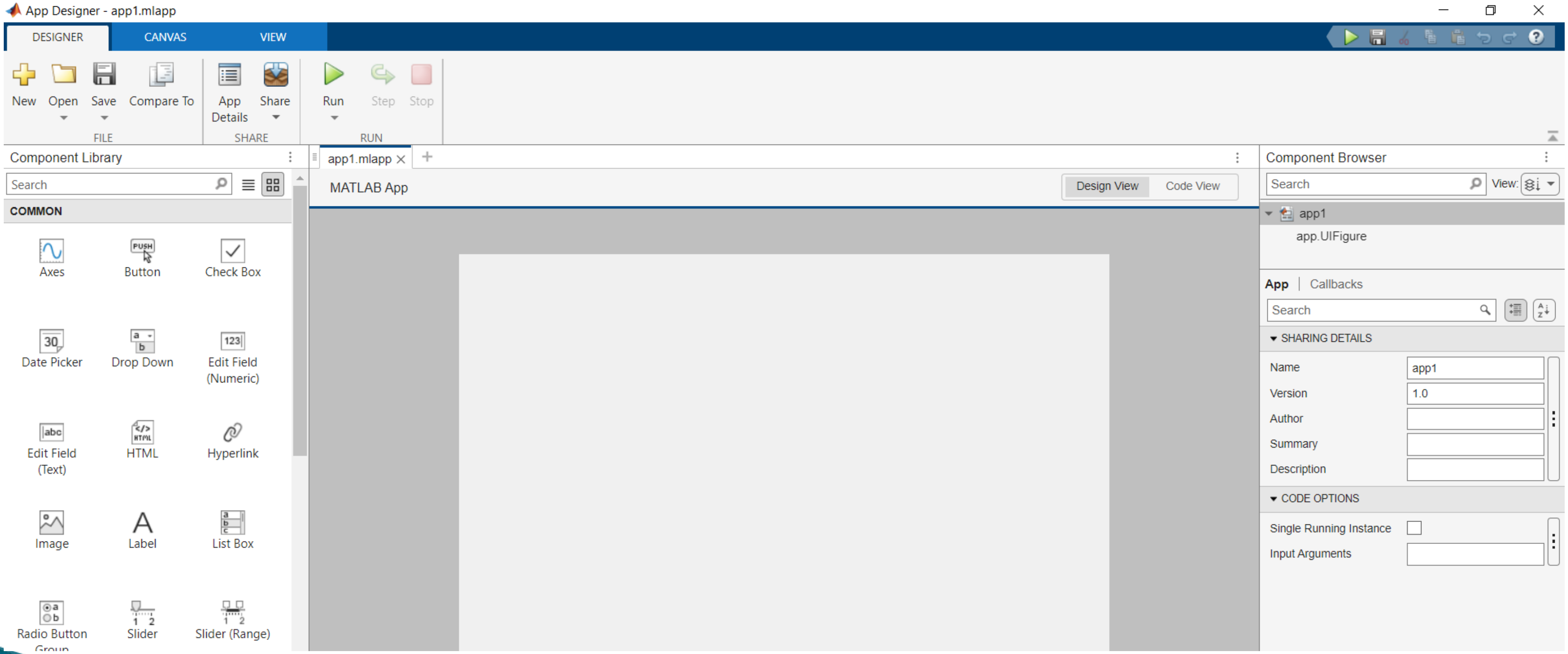


ESERCITAZIONI
MATLAB – Nozioni

App Designer

Create and Run a Simple App Using App Designer

Tutorial Steps for Creating the App



Name: app1

Version: 1.0

Author:

Summary:

Description:

Single Running Instance □

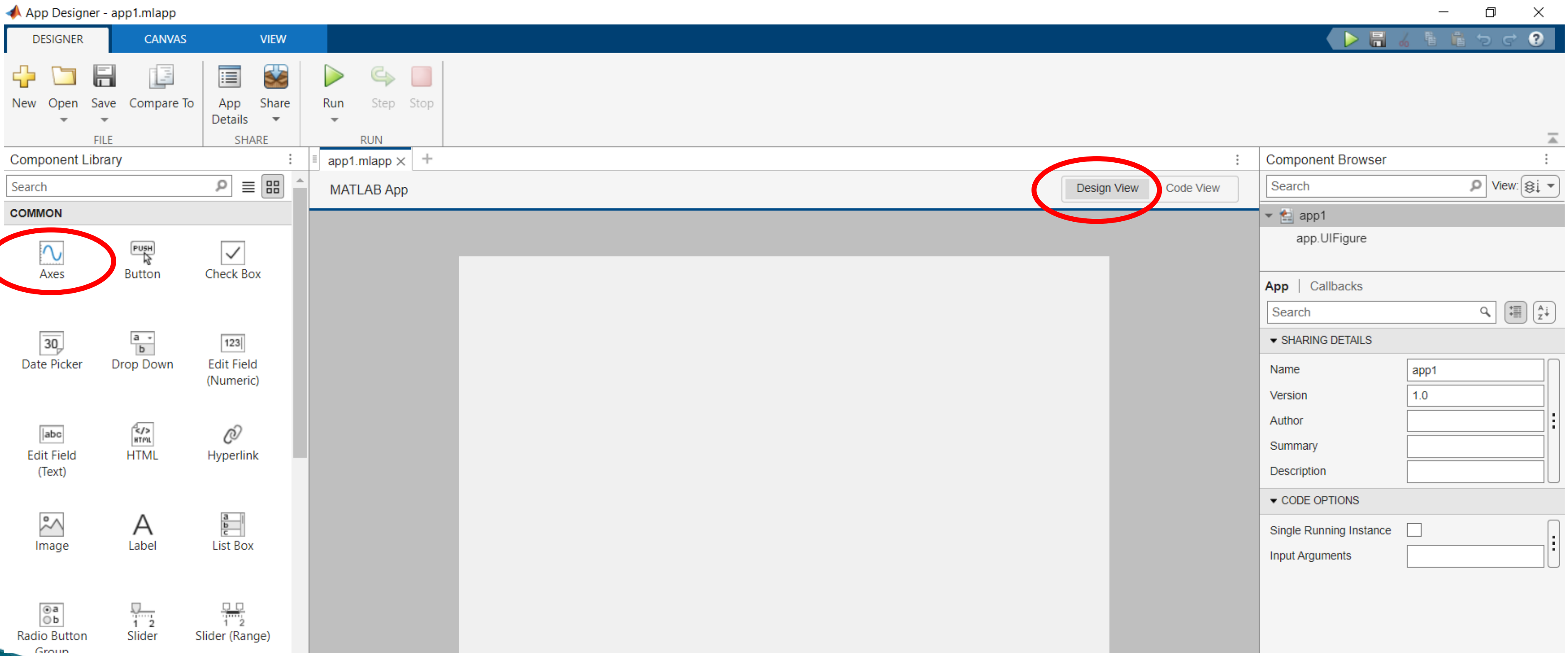
Input Arguments:

ESERCITAZIONI
MATLAB – Nozioni

App Designer

Create and Run a Simple App Using App Designer

Tutorial Steps for Creating the App



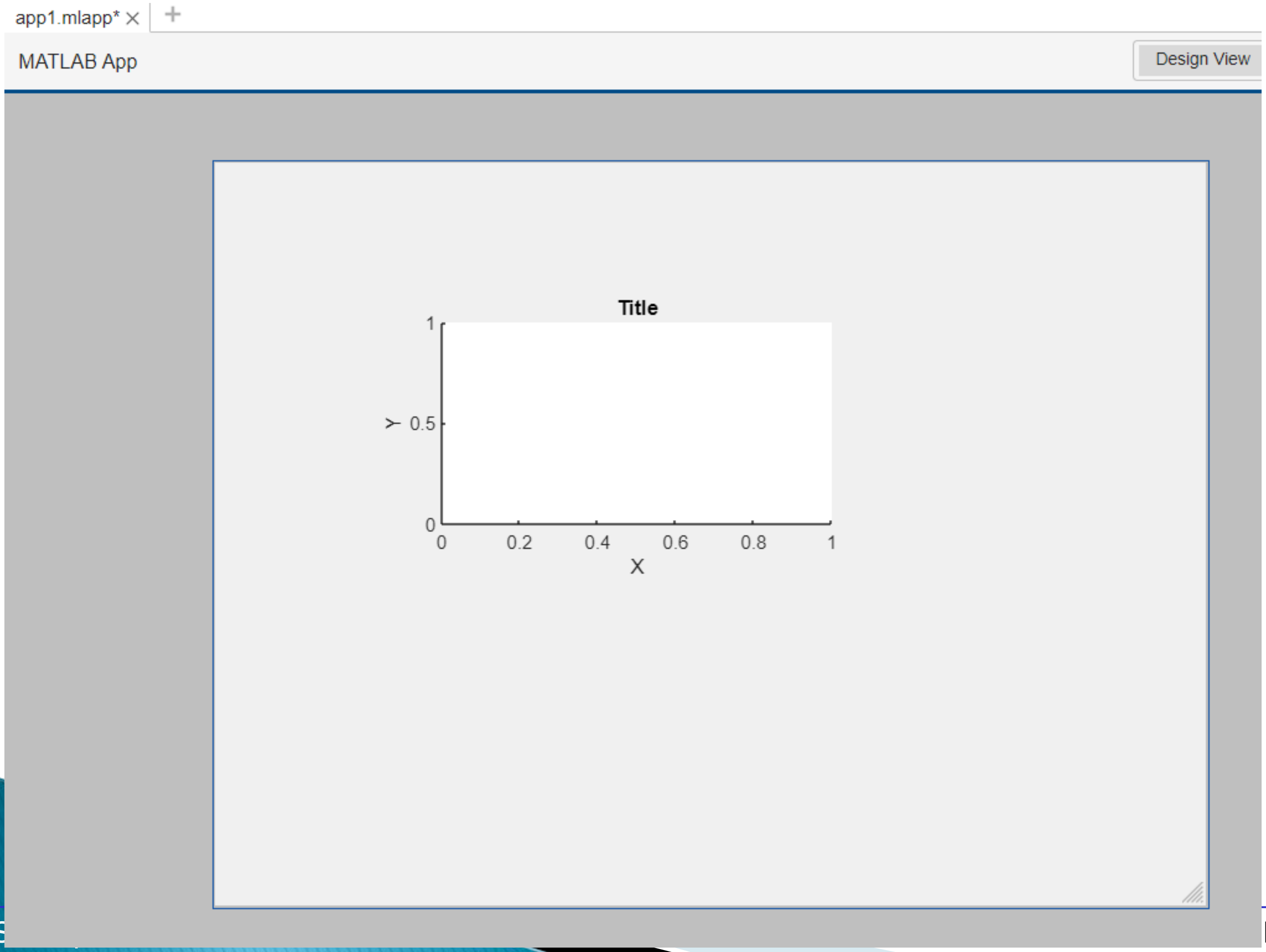
Name: app1
Version: 1.0
Author:
Summary:
Description:

ESERCITAZIONI
MATLAB – Nozioni

App Designer

Create and Run a Simple App Using App Designer

Tutorial Steps for Creating the App



ESERCITAZIONI MATLAB – Nozioni

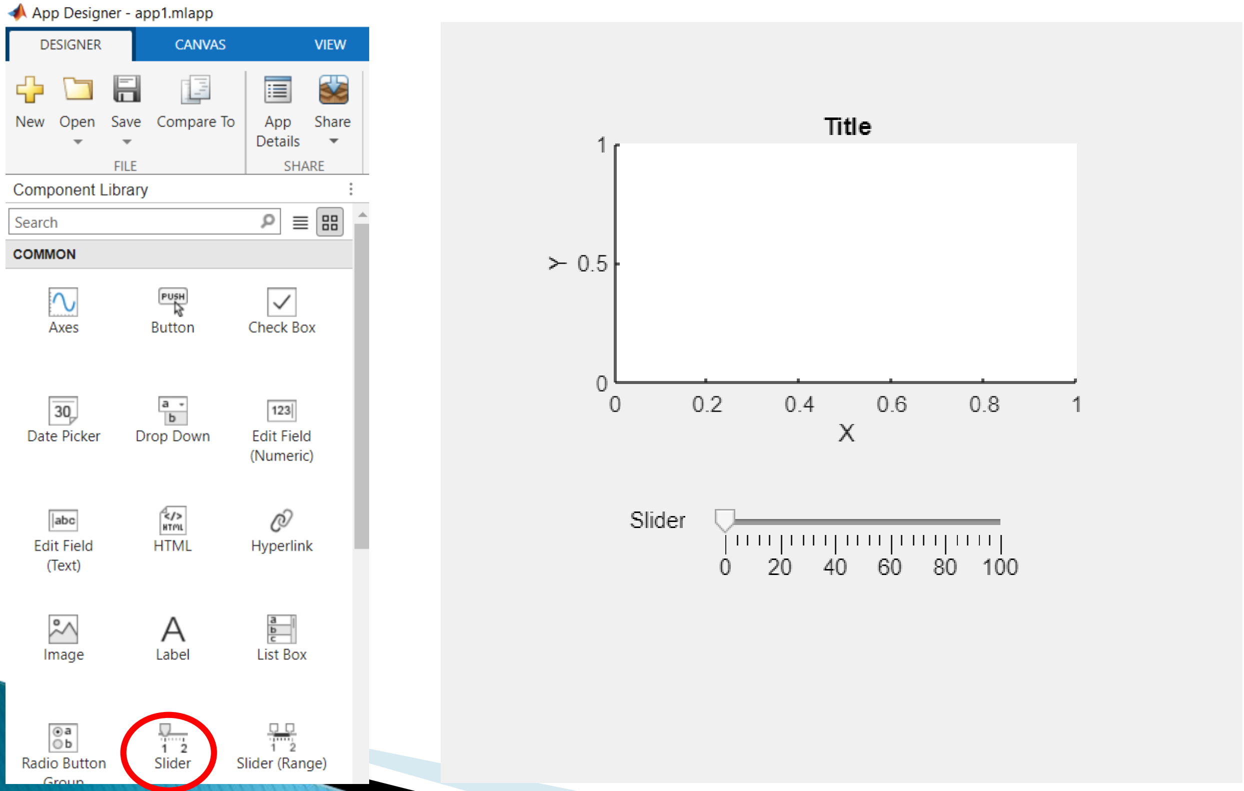
App Designer

Create and Run a Simple App Using App Designer

Tutorial Steps for Creating the App

Step 2: Create a Slider Component

Drag a **Slider** component from the **Component Library** onto the canvas. Place it below the axes component.



ESERCITAZIONI
MATLAB – Nozioni

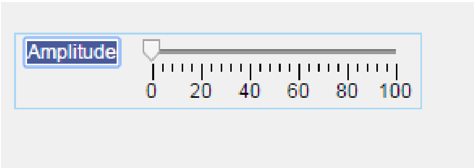
App Designer

Create and Run a Simple App Using App Designer

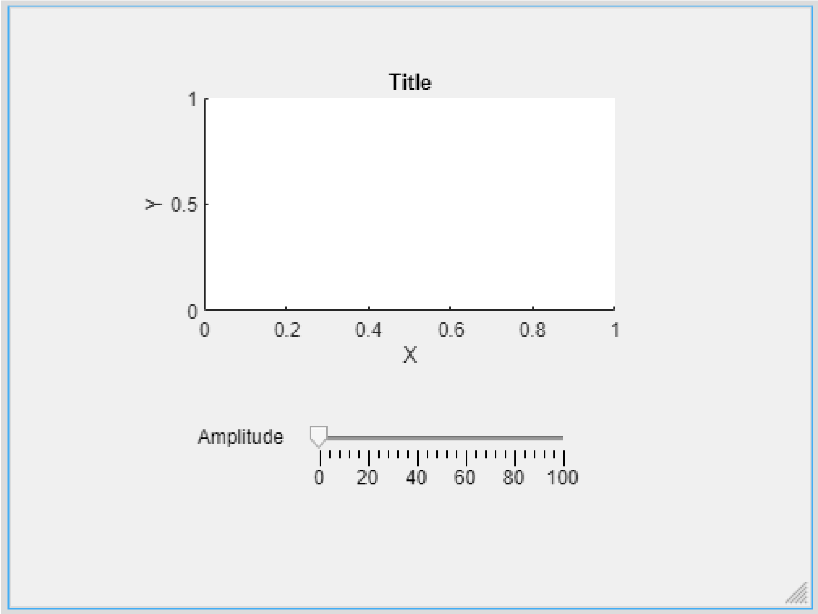
Tutorial Steps for Creating the App

Step 3: Update the Slider Label

Replace the slider label text. Double-click the label and replace the word `Slider` with `Amplitude`.



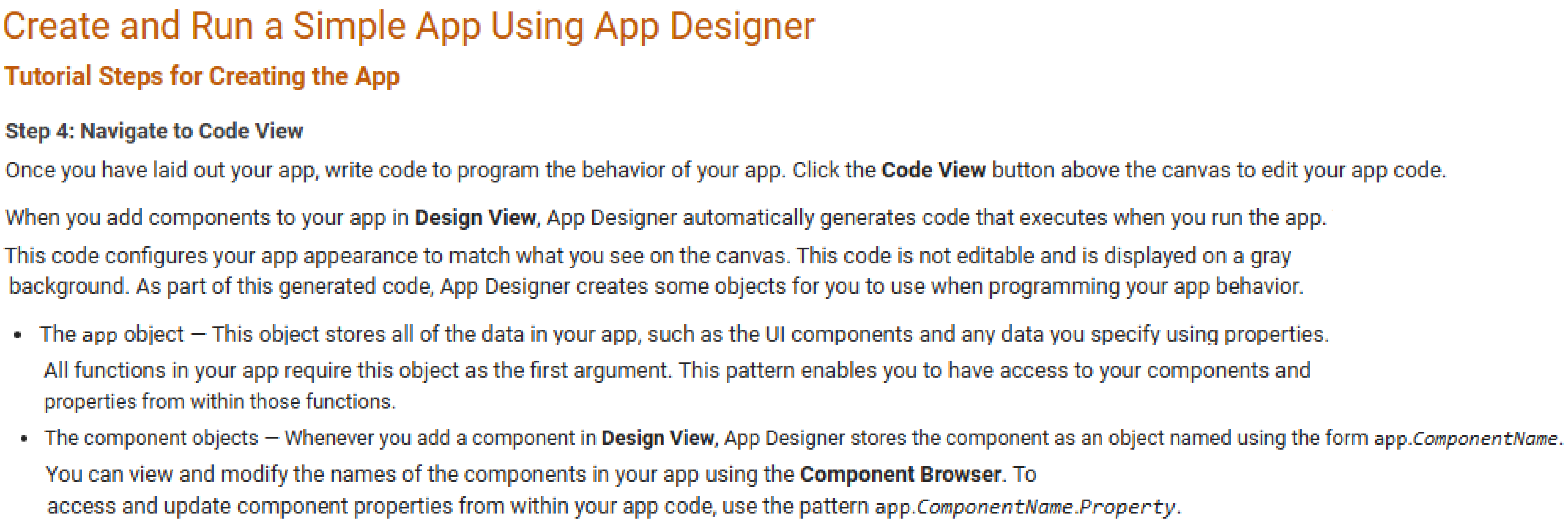
When you have finished laying out your app, the canvas in **Design View** should look like this:



For more information about laying out apps, see Lay Out Apps in App Designer Design View.

ESERCITAZIONI
MATLAB – Nozioni

App Designer



Create and Run a Simple App Using App Designer

Tutorial Steps for Creating the App

Step 4: Navigate to Code View

Once you have laid out your app, write code to program the behavior of your app. Click the **Code View** button above the canvas to edit your app code.

When you add components to your app in **Design View**, App Designer automatically generates code that executes when you run the app.

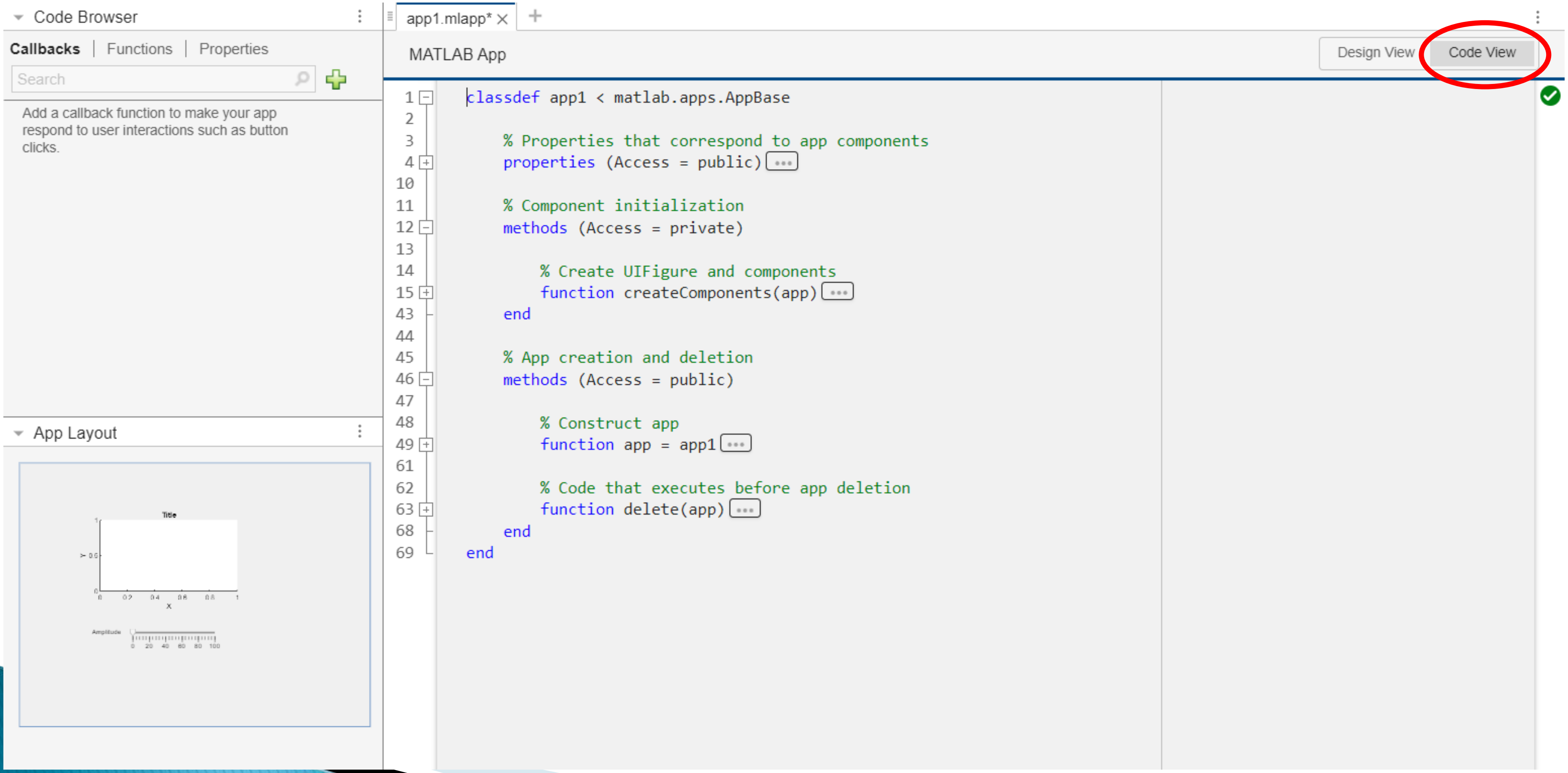
This code configures your app appearance to match what you see on the canvas. This code is not editable and is displayed on a gray background. As part of this generated code, App Designer creates some objects for you to use when programming your app behavior.

- The `app` object — This object stores all of the data in your app, such as the UI components and any data you specify using properties. All functions in your app require this object as the first argument. This pattern enables you to have access to your components and properties from within those functions.
- The component objects — Whenever you add a component in **Design View**, App Designer stores the component as an object named using the form `app.`*`ComponentName`*. You can view and modify the names of the components in your app using the **Component Browser**. To access and update component properties from within your app code, use the pattern `app.`*`ComponentName`*`.`*`Property`*.

App Designer

Create and Run a Simple App Using App Designer

Tutorial Steps for Creating the App


Code Browser
Callbacks | Functions | Properties
Search
Add a callback function to make your app respond to user interactions such as button clicks.
App Layout
Title
Amplitude
app1.mlapp*
MATLAB App
Design View
Code View

```
classdef app1 < matlab.apps.AppBase

    % Properties that correspond to app components
    properties (Access = public) ...

    % Component initialization
    methods (Access = private)

        % Create UIFigure and components
        function createComponents(app) ...
    end

    % App creation and deletion
    methods (Access = public)

        % Construct app
        function app = app1 ...

        % Code that executes before app deletion
        function delete(app) ...
    end
end
```

ESERCITAZIONI
MATLAB – Nozioni

App Designer

Create and Run a Simple App Using App Designer

Tutorial Steps for Creating the App

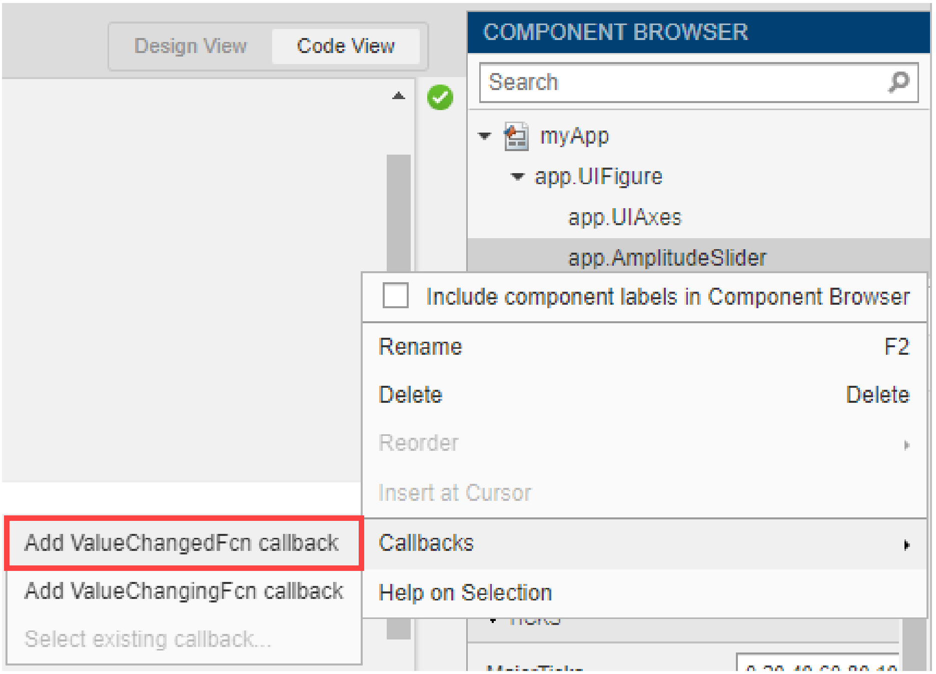
Step 5: Add a Slider Callback Function

Program your app behavior using *callback functions*.

A callback function is a function that executes when the app user performs a specific interaction, such as adjusting the value of a slider.

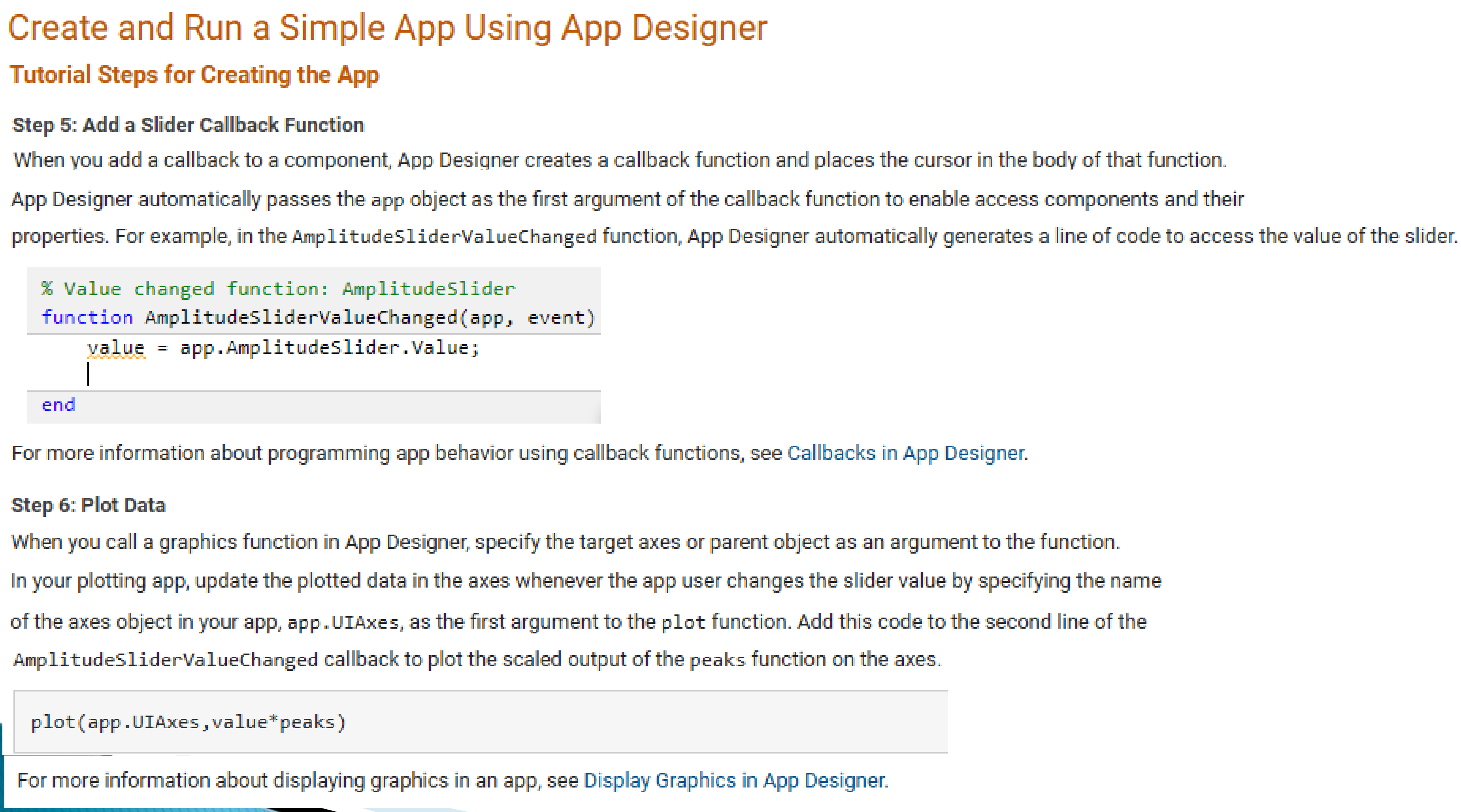
In your plotting app, add a callback function that executes whenever the user adjusts the slider value.

Right-click `app.AmplitudeSlider` in the **Component Browser**. Then select **Callbacks** > **Add ValueChangedFcn callback** in the context menu.



ESERCITAZIONI
MATLAB – Nozioni

App Designer



Create and Run a Simple App Using App Designer

Tutorial Steps for Creating the App

Step 5: Add a Slider Callback Function

When you add a callback to a component, App Designer creates a callback function and places the cursor in the body of that function.

App Designer automatically passes the `app` object as the first argument of the callback function to enable access components and their properties. For example, in the `AmplitudeSliderValueChanged` function, App Designer automatically generates a line of code to access the value of the slider.

```
% Value changed function: AmplitudeSlider
function AmplitudeSliderValueChanged(app, event)
    value = app.AmplitudeSlider.Value;
    
end
```

For more information about programming app behavior using callback functions, see Callbacks in App Designer.

Step 6: Plot Data

When you call a graphics function in App Designer, specify the target axes or parent object as an argument to the function.

In your plotting app, update the plotted data in the axes whenever the app user changes the slider value by specifying the name of the axes object in your app, `app.UIAxes`, as the first argument to the `plot` function. Add this code to the second line of the `AmplitudeSliderValueChanged` callback to plot the scaled output of the `peaks` function on the axes.

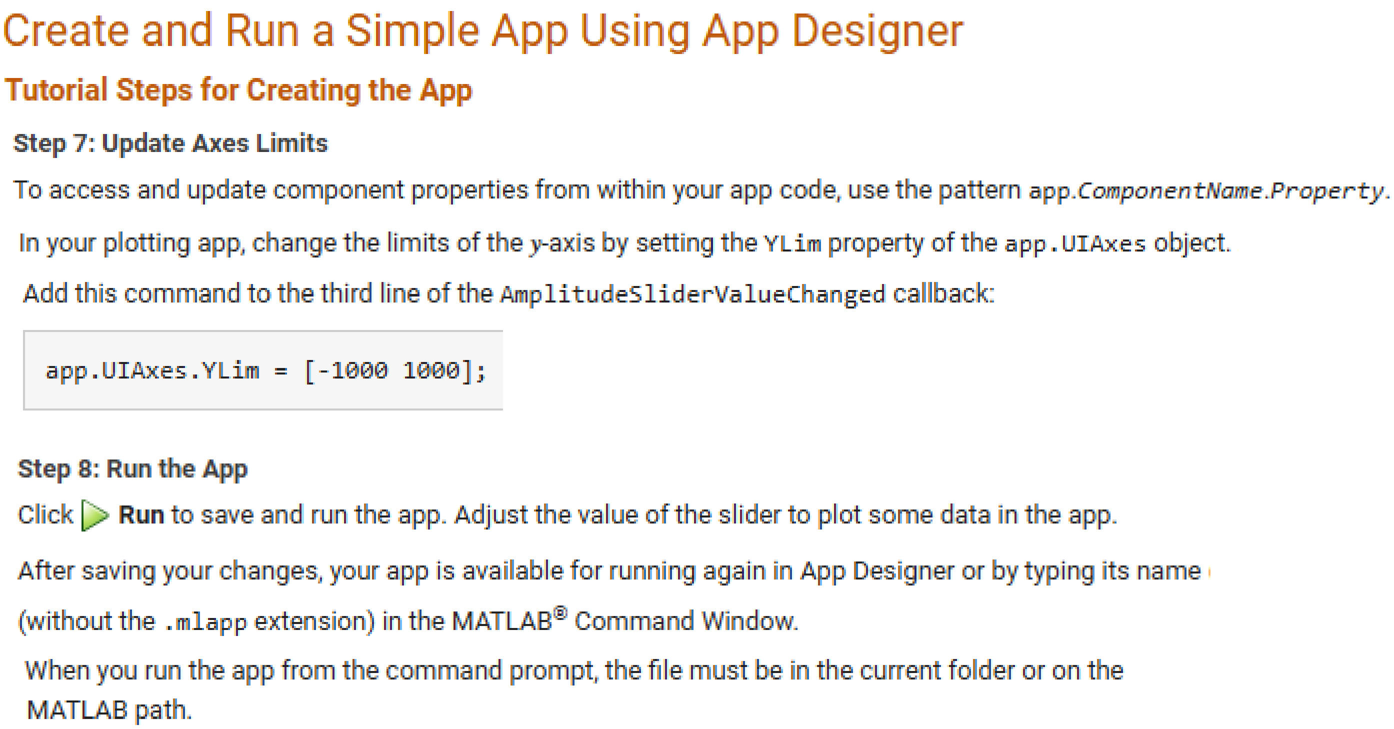
```
plot(app.UIAxes,value*peaks)
```

For more information about displaying graphics in an app, see Display Graphics in App Designer.

ESERCITAZIONI
MATLAB – Nozioni

MathWorks®

App Designer



Create and Run a Simple App Using App Designer

Tutorial Steps for Creating the App

Step 7: Update Axes Limits

To access and update component properties from within your app code, use the pattern `app.ComponentName.Property`.

In your plotting app, change the limits of the *y*-axis by setting the `YLim` property of the `app.UIAxes` object.

Add this command to the third line of the `AmplitudeSliderValueChanged` callback:

```
app.UIAxes.YLim = [-1000 1000];
```

Step 8: Run the App

Click ▷ **Run** to save and run the app. Adjust the value of the slider to plot some data in the app.

After saving your changes, your app is available for running again in App Designer or by typing its name

(without the `.mlapp` extension) in the MATLAB® Command Window.

When you run the app from the command prompt, the file must be in the current folder or on the MATLAB path.

Riferimenti Bibliografici

[1] https://it.mathworks.com