Class Definition


```
classdef
  properties
  methods
```

- constructor

A constructor is a method
that creates an object and

Contact.m

```
classdef Contact
    properties
        FirstName
        LastName
        PhoneNumber
    end
    methods
        function obj = Contact(lname,fname,phone)
            obj.LastName = string(lname);
            obj.FirstName = string(fname);
            obj.PhoneNumber = string(phone);
        end
    end
end
```

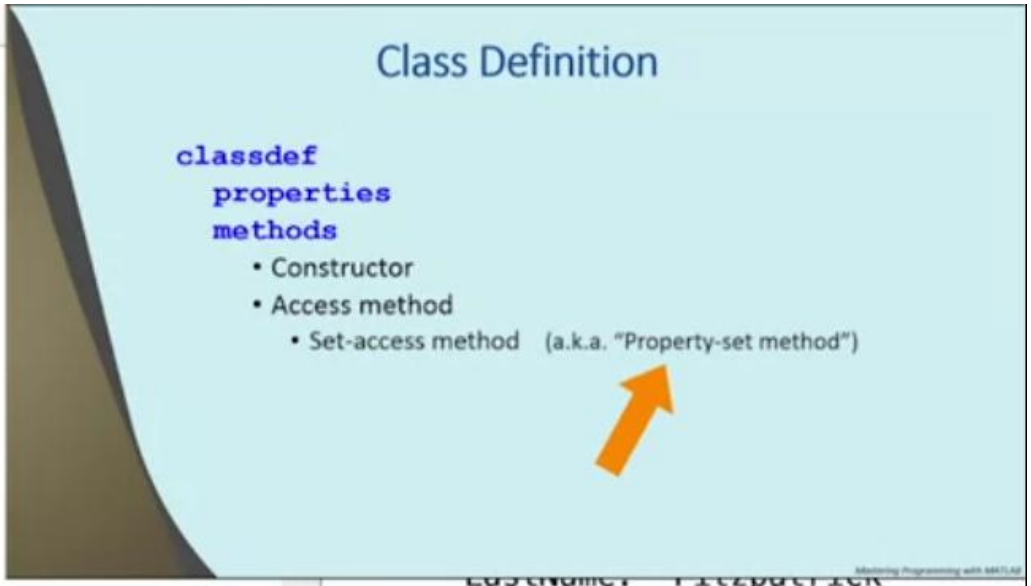
COMMAND WINDOW

```
>> person = Contact
person =
  Contact with properties:

      FirstName: []
       LastName: []
    PhoneNumber: []
>> person.FirstName = "Mike"
person =
  Contact with properties:

      FirstName: "Mike"
       LastName: []
    PhoneNumber: []
>> person.LastName = "Fitzpatrick"
person =
  Contact with properties:

      FirstName: "Mike"
       LastName: "Fitzpatrick"
    PhoneNumber: []
>> person.PhoneNumber = "(555) 123-4
person =
```

As promised, we've introduced the methods section, with the keyword methods.

Contact.m *

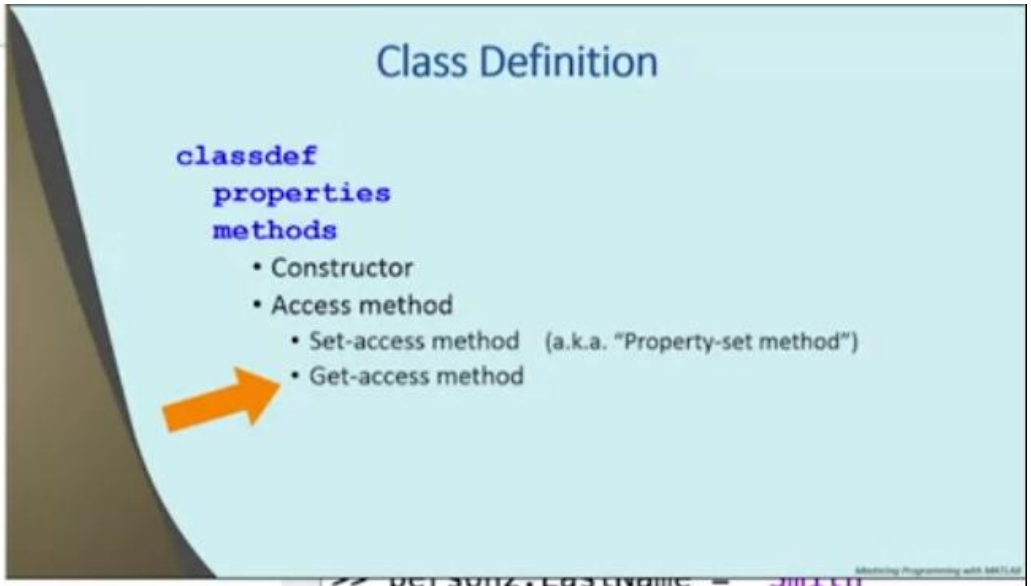
```
classdef Contact
    properties
        FirstName
        LastName
        PhoneNumber
    end
    methods
        function obj = Contact(lname,fname,phone)
            obj.LastName = string(lname);
            obj.FirstName = string(fname);
            obj.PhoneNumber = string(phone);
        end
        function obj = set.LastName(obj,lname)
            obj.LastName = string(lname);
        end
    end
end
```



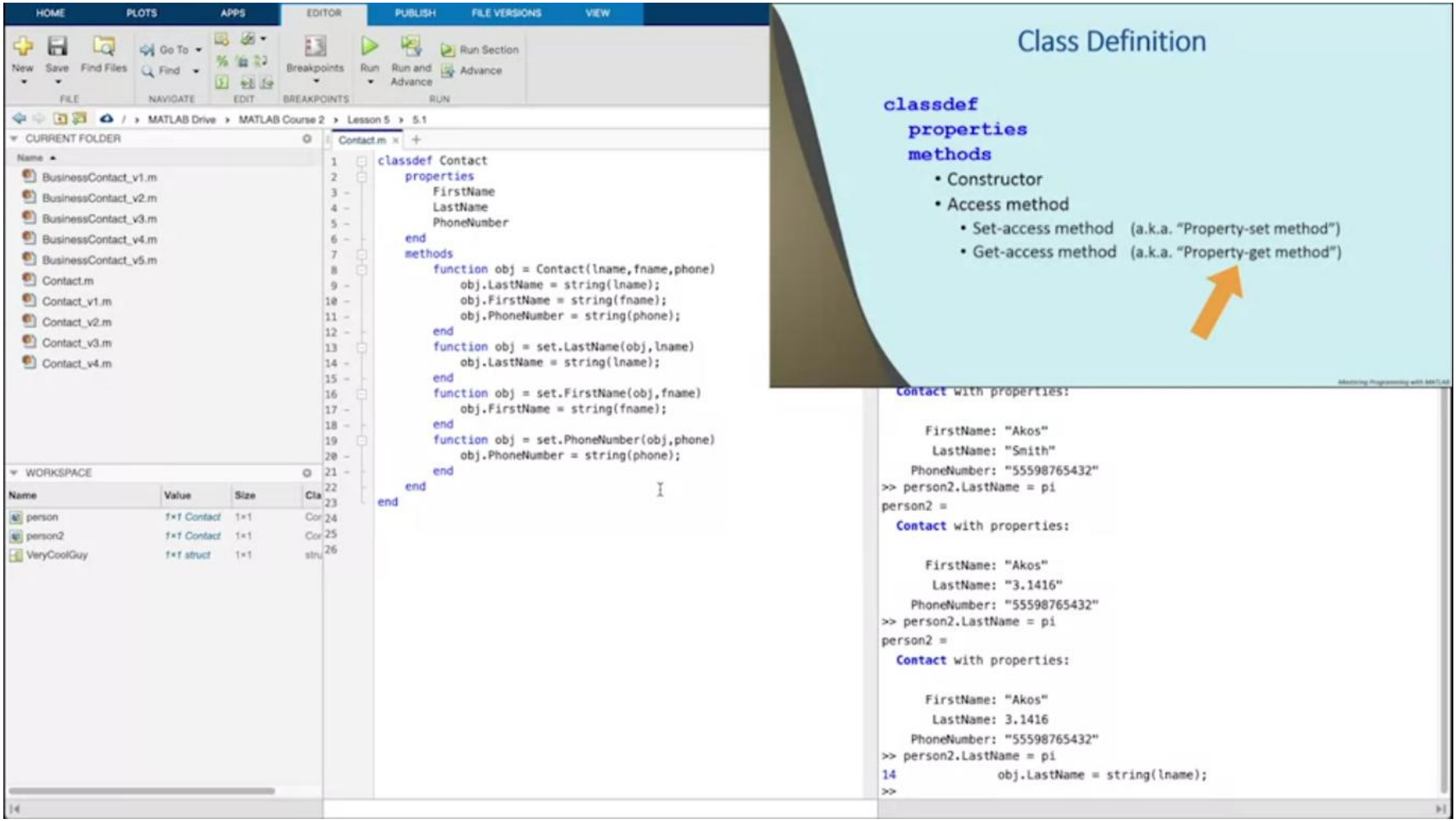

```
    PhoneNumber: []
>> person.PhoneNumber = "(555) 123-45
person =
  Contact with properties:

      FirstName: "Mike"
       LastName: "Fitzpatrick"
    PhoneNumber: "(555) 123-4567"
>> person2 = Contact("Ledeczi" , "Ako
person2 =
  Contact with properties:
```

Contact.m

```
classdef Contact
    properties
        FirstName
        LastName
        PhoneNumber
    end
    methods
        function obj = Contact(lname,fname,phone)
            obj.LastName = string(lname);
            obj.FirstName = string(fname);
            obj.PhoneNumber = string(phone);
        end
        function obj = set.LastName(obj,lname)
            obj.LastName = string(lname);
        end
        function obj = set.FirstName(obj,fname)
            obj.FirstName = string(fname);
        end
        function obj = set.PhoneNumber(obj,phone)
            obj.PhoneNumber = string(phone);
        end
    end
end
```




```
person2 =
  Contact with properties:

      FirstName: "Akos"
       LastName: "Smith"
    PhoneNumber: "55598765432"
>> person2.LastName = pi
person2 =
  Contact with properties:

      FirstName: "Akos"
       LastName: "3.1416"
```


HOME
PLOTS
APPS
EDITOR
PUBLISH
FILE VERSIONS
VIEW
New
Save
Find Files
Go To
Find
Breakpoints
Run
Run and Advance
Run Section
Advance
FILE
NAVIGATE
EDIT
BREAKPOINTS
RUN
MATLAB Drive › MATLAB Course 2 › Lesson 5 › 5.1
CURRENT FOLDER
Name
BusinessContact_v1.m
BusinessContact_v2.m
BusinessContact_v3.m
BusinessContact_v4.m
BusinessContact_v5.m
Contact.m
Contact_v1.m
Contact_v2.m
Contact_v3.m
Contact_v4.m
WORKSPACE
Name
Value
Size
person
person2
VeryCoolGuy
Contact.m
classdef Contact
properties
FirstName
LastName
PhoneNumber
end
methods
function obj = Contact(lname,fname,phone)
obj.LastName = string(lname);
obj.FirstName = string(fname);
obj.PhoneNumber = string(phone);
end
function obj = set.LastName(obj,lname)
obj.LastName = string(lname);
end
function obj = set.FirstName(obj,fname)
obj.FirstName = string(fname);
end
function obj = set.PhoneNumber(obj,phone)
obj.PhoneNumber = string(phone);
end
end
end
Class Definition
classdef
properties
methods
• Constructor
• Access method
• Set-access method (a.k.a. "Property-set method")
• Get-access method (a.k.a. "Property-get method")
Contact with properties:
FirstName: "Akos"
LastName: "Smith"
PhoneNumber: "55598765432"
>> person2.LastName = pi
person2 =
Contact with properties:
FirstName: "Akos"
LastName: "3.1416"
PhoneNumber: "55598765432"
>> person2.LastName = pi
person2 =
Contact with properties:
FirstName: "Akos"
LastName: 3.1416
PhoneNumber: "55598765432"
>> person2.LastName = pi
14 obj.LastName = string(lname);
>>

Inheritance

super class

```
Property_1
Property_2
Method_1
Method_2
```

subclass

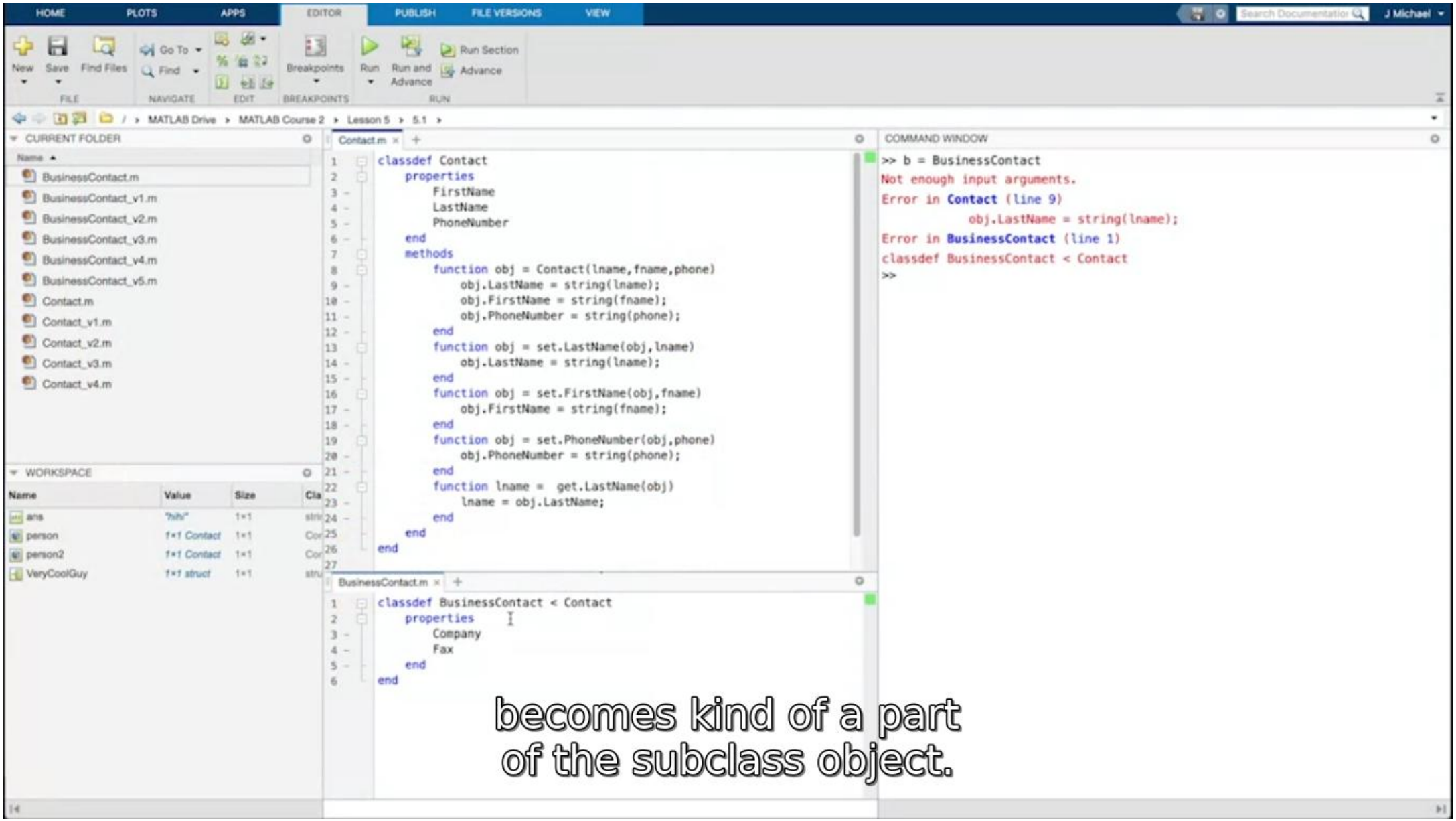
```
Property_1
Property_2
Method_1
Method_2
```

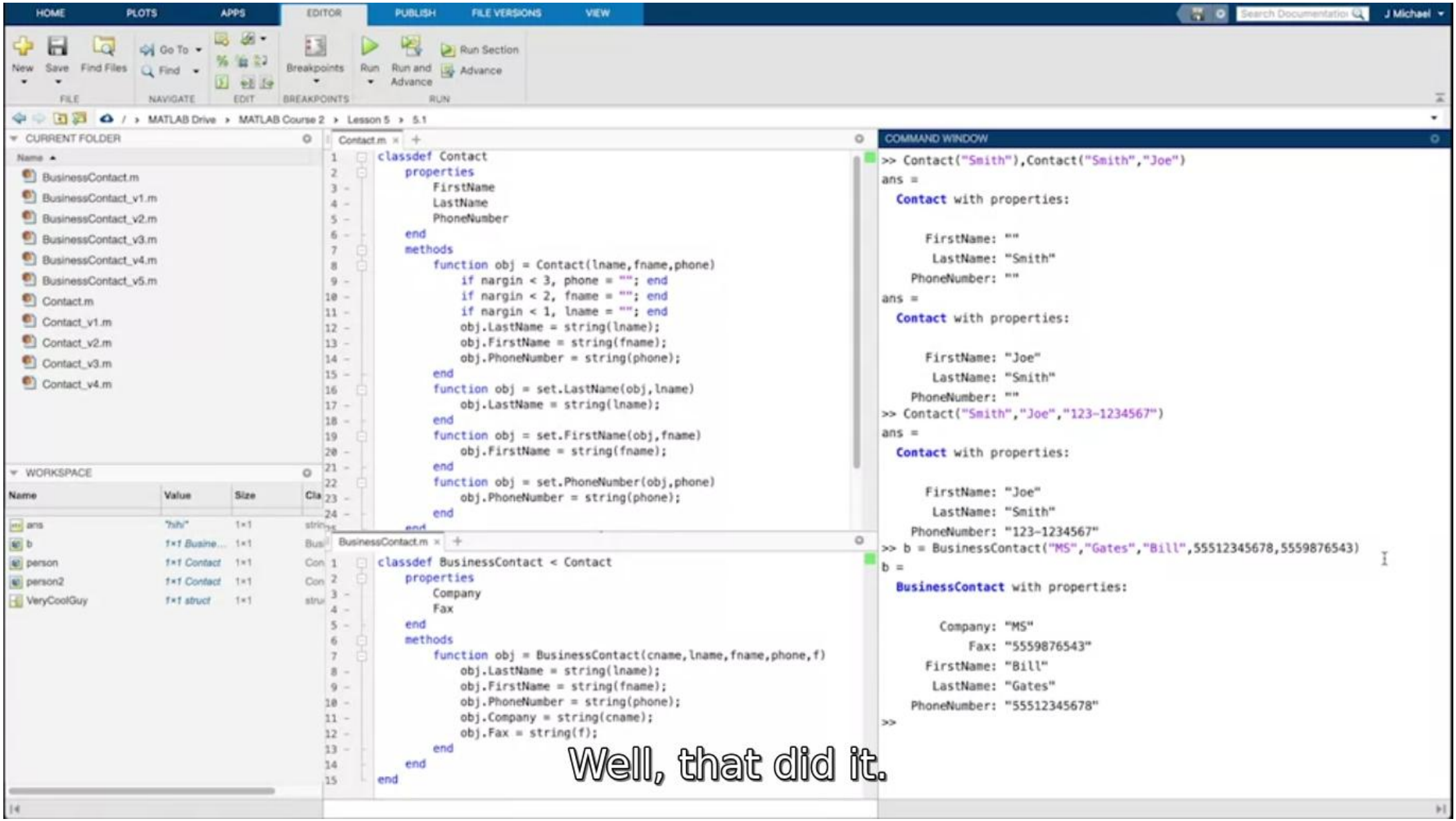
subclass

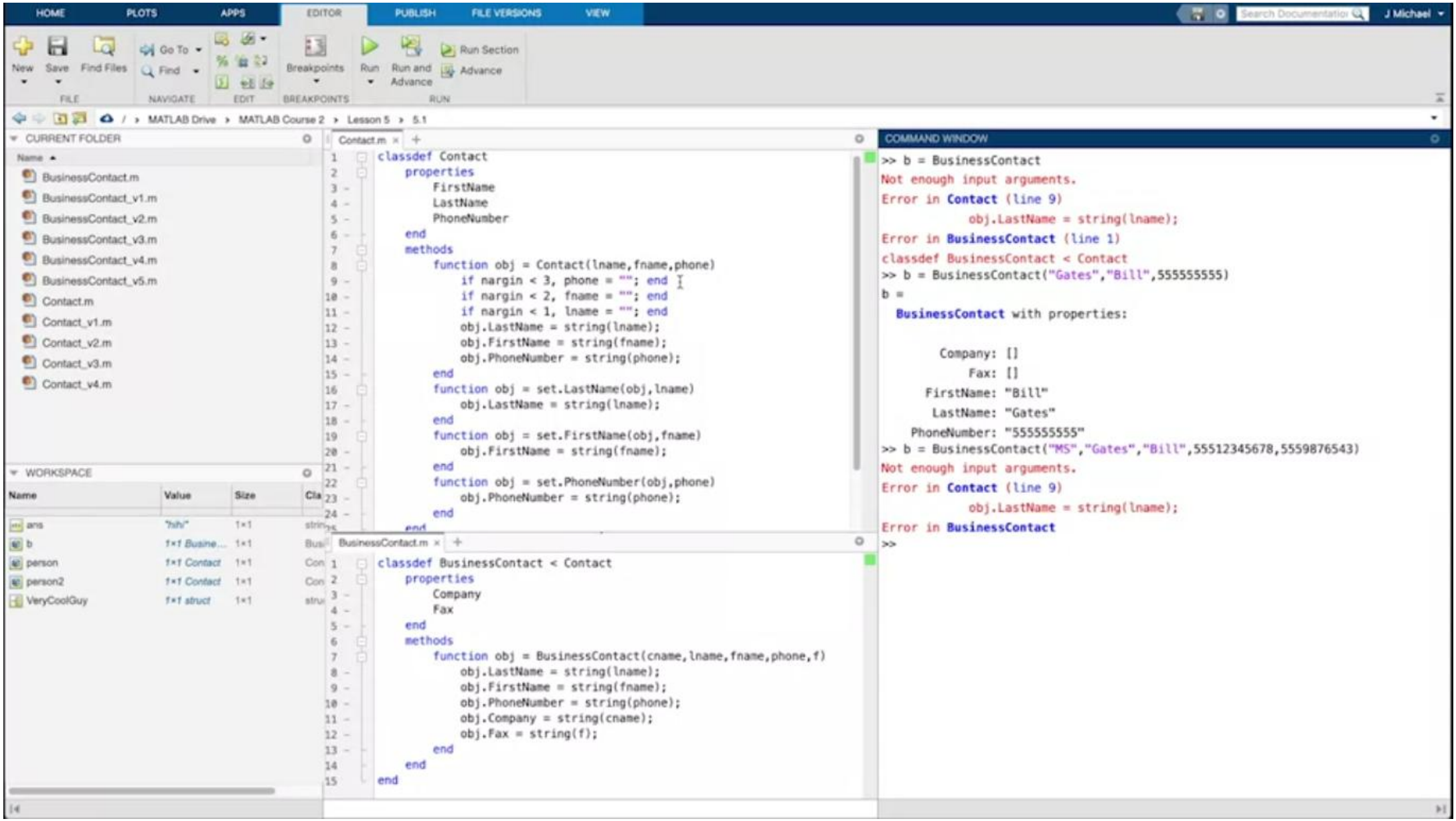
```
Property_1
Property_2
Method_1
Method_2
```

. . .

Which, somewhat confusingly,
is also called a base class.


HOME
PLOTS
APPS
EDITOR
PUBLISH
FILE VERSIONS
VIEW
J Michael
New
Save
Find Files
Go To
Find
Breakpoints
Run
Run and Advance
Run Section
Advance
FILE
NAVIGATE
EDIT
BREAKPOINTS
RUN
MATLAB Drive › MATLAB Course 2 › Lesson 5 › 5.1 ›
CURRENT FOLDER
Name
BusinessContact.m
BusinessContact_v1.m
BusinessContact_v2.m
BusinessContact_v3.m
BusinessContact_v4.m
BusinessContact_v5.m
Contact.m
Contact_v1.m
Contact_v2.m
Contact_v3.m
Contact_v4.m
WORKSPACE
Name
Value
Size
ans
"hihi"
1×1
person
1×1 Contact
1×1
person2
1×1 Contact
1×1
VeryCoolGuy
1×1 struct
1×1
Contact.m
classdef Contact
properties
FirstName
LastName
PhoneNumber
end
methods
function obj = Contact(lname,fname,phone)
obj.LastName = string(lname);
obj.FirstName = string(fname);
obj.PhoneNumber = string(phone);
end
function obj = set.LastName(obj,lname)
obj.LastName = string(lname);
end
function obj = set.FirstName(obj,fname)
obj.FirstName = string(fname);
end
function obj = set.PhoneNumber(obj,phone)
obj.PhoneNumber = string(phone);
end
function lname = get.LastName(obj)
lname = obj.LastName;
end
end
end
BusinessContact.m
classdef BusinessContact < Contact
properties
Company
Fax
end
end
COMMAND WINDOW
>> b = BusinessContact
Not enough input arguments.
Error in Contact (line 9)
obj.LastName = string(lname);
Error in BusinessContact (line 1)
classdef BusinessContact < Contact
>>
becomes kind of a part
of the subclass object.


HOME
PLOTS
APPS
EDITOR
PUBLISH
FILE VERSIONS
VIEW
New
Save
Find Files
Go To
Find
Breakpoints
Run
Run and Advance
Run Section
Advance
FILE
NAVIGATE
EDIT
BREAKPOINTS
RUN
MATLAB Drive › MATLAB Course 2 › Lesson 5 › 5.1
CURRENT FOLDER
BusinessContact.m
BusinessContact_v1.m
BusinessContact_v2.m
BusinessContact_v3.m
BusinessContact_v4.m
BusinessContact_v5.m
Contact.m
Contact_v1.m
Contact_v2.m
Contact_v3.m
Contact_v4.m
WORKSPACE
Name
Value
Size
ans
b
person
person2
VeryCoolGuy
Contact.m
classdef Contact
properties
FirstName
LastName
PhoneNumber
end
methods
function obj = Contact(lname,fname,phone)
if nargin < 3, phone = ""; end
if nargin < 2, fname = ""; end
if nargin < 1, lname = ""; end
obj.LastName = string(lname);
obj.FirstName = string(fname);
obj.PhoneNumber = string(phone);
end
function obj = set.LastName(obj,lname)
obj.LastName = string(lname);
end
function obj = set.FirstName(obj,fname)
obj.FirstName = string(fname);
end
function obj = set.PhoneNumber(obj,phone)
obj.PhoneNumber = string(phone);
end
BusinessContact.m
classdef BusinessContact < Contact
properties
Company
Fax
end
methods
function obj = BusinessContact(cname,lname,fname,phone,f)
obj.LastName = string(lname);
obj.FirstName = string(fname);
obj.PhoneNumber = string(phone);
obj.Company = string(cname);
obj.Fax = string(f);
end
end
end
COMMAND WINDOW
>> Contact("Smith"),Contact("Smith","Joe")
ans =
Contact with properties:
FirstName: ""
LastName: "Smith"
PhoneNumber: ""
ans =
Contact with properties:
FirstName: "Joe"
LastName: "Smith"
PhoneNumber: ""
>> Contact("Smith","Joe","123-1234567")
ans =
Contact with properties:
FirstName: "Joe"
LastName: "Smith"
PhoneNumber: "123-1234567"
>> b = BusinessContact("MS","Gates","Bill",55512345678,5559876543)
b =
BusinessContact with properties:
Company: "MS"
Fax: "5559876543"
FirstName: "Bill"
LastName: "Gates"
PhoneNumber: "55512345678"
>>
Well, that did it.


HOME
PLOTS
APPS
EDITOR
PUBLISH
FILE VERSIONS
VIEW
New
Save
Find Files
Go To
Find
Breakpoints
Run
Run and Advance
Run Section
Advance
FILE
NAVIGATE
EDIT
BREAKPOINTS
RUN
MATLAB Drive › MATLAB Course 2 › Lesson 5 › 5.1
CURRENT FOLDER
Name
BusinessContact.m
BusinessContact_v1.m
BusinessContact_v2.m
BusinessContact_v3.m
BusinessContact_v4.m
BusinessContact_v5.m
Contact.m
Contact_v1.m
Contact_v2.m
Contact_v3.m
Contact_v4.m
WORKSPACE
Name
Value
Size
ans
b
person
person2
VeryCoolGuy
Contact.m
classdef Contact
properties
FirstName
LastName
PhoneNumber
end
methods
function obj = Contact(lname,fname,phone)
if nargin < 3, phone = ""; end
if nargin < 2, fname = ""; end
if nargin < 1, lname = ""; end
obj.LastName = string(lname);
obj.FirstName = string(fname);
obj.PhoneNumber = string(phone);
end
function obj = set.LastName(obj,lname)
obj.LastName = string(lname);
end
function obj = set.FirstName(obj,fname)
obj.FirstName = string(fname);
end
function obj = set.PhoneNumber(obj,phone)
obj.PhoneNumber = string(phone);
end
end
BusinessContact.m
classdef BusinessContact < Contact
properties
Company
Fax
end
methods
function obj = BusinessContact(cname,lname,fname,phone,f)
obj.LastName = string(lname);
obj.FirstName = string(fname);
obj.PhoneNumber = string(phone);
obj.Company = string(cname);
obj.Fax = string(f);
end
end
end
COMMAND WINDOW
>> b = BusinessContact
Not enough input arguments.
Error in Contact (line 9)
obj.LastName = string(lname);
Error in BusinessContact (line 1)
classdef BusinessContact < Contact
>> b = BusinessContact("Gates","Bill",555555555)
b =
BusinessContact with properties:
Company: []
Fax: []
FirstName: "Bill"
LastName: "Gates"
PhoneNumber: "555555555"
>> b = BusinessContact("MS","Gates","Bill",55512345678,5559876543)
Not enough input arguments.
Error in Contact (line 9)
obj.LastName = string(lname);
Error in BusinessContact
>>

Object-Oriented Programming

- OOP is data-centered.
- Data is stored inside an **object**.
- The type of an object is called a **class**.
- A class must be defined by the user.

Class Definition


```
classdef ClassName
   properties
      Property_1
      Property_2
      ...
   end
end
```

```
>> object.Property_3 = "(555) 123-4567";
>> object.Property_3
ans =
    "(555) 123-4567"
```

Class Definition

```
classdef ClassName
   properties
      Property_1
      Property_2
      ...
   end
   methods
      function out = Method_1(inputs)
         <function body>
      end
      function out = Method_2(inputs)
         <function body>
      end
   end
end
```

Subclass Definition

```
classdef SubClass < SuperClass
   properties
      Property_3
      ...
   end
   methods
      function out = Method_3(inputs)
         <function body>
      end
      ...
   end
end
```

Inheritance

super class

- Property_1
- Property_2
- Method_1
- Method_2

subclass

- Property_1
- Property_2
- Method_1
- Method_2

subclass

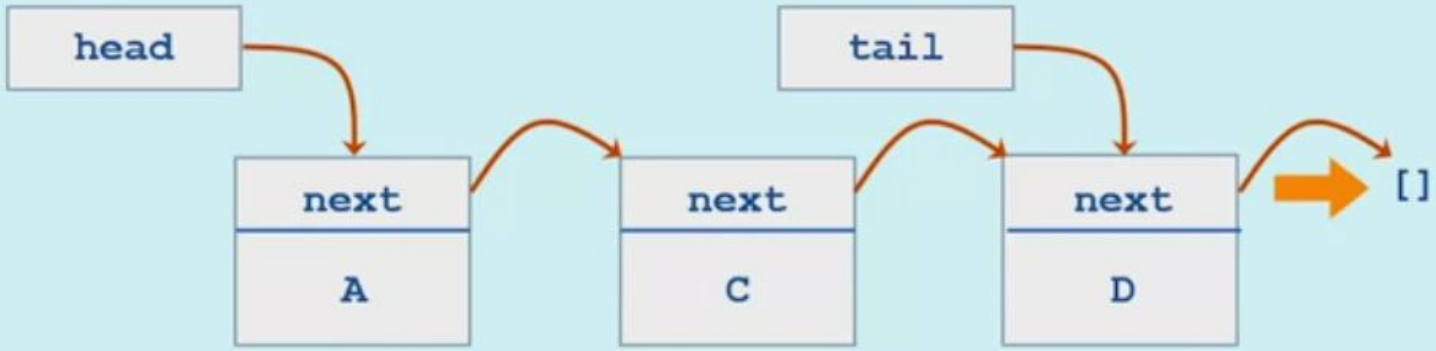
- Property_1
- Property_2
- Method_1
- Method_2

. . .

Class Definition

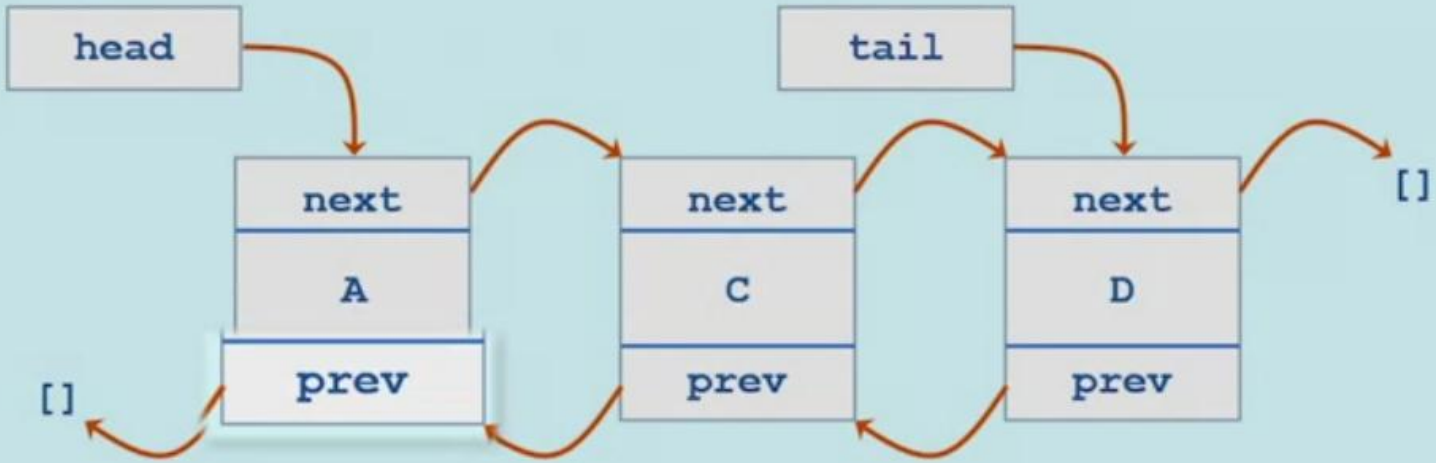
```
classdef ClassName
   properties
      ...
   end
   methods
      function obj = ClassName(input)
         <function body>
      end
      function obj = set.PropertyName(input)
         <function body>
      end
      function PropertyValue = get.PropertyName(input)
         <function body>
      end
   end
```

Linked List



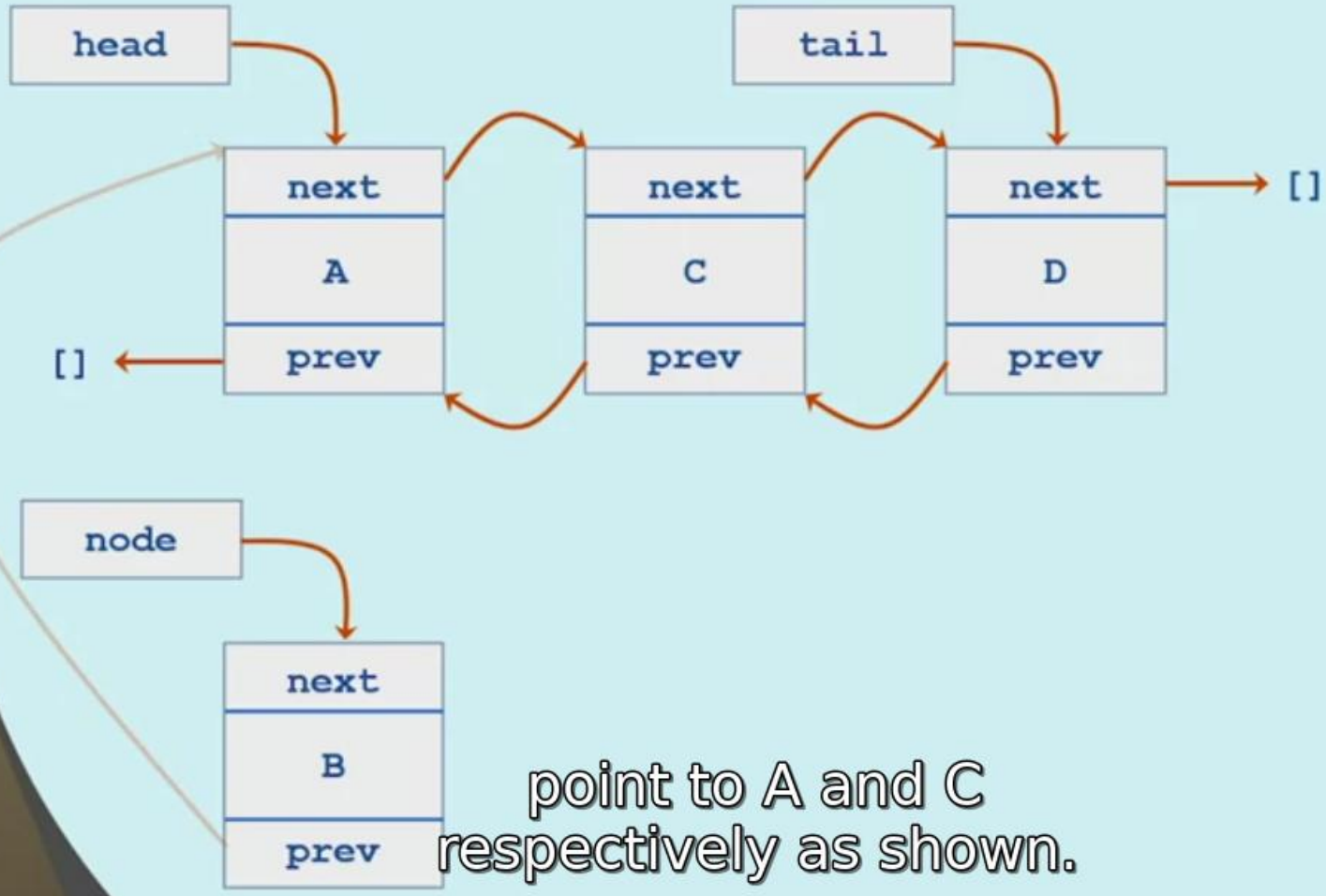
a next field that's equal
to the empty array.

Doubly Linked List



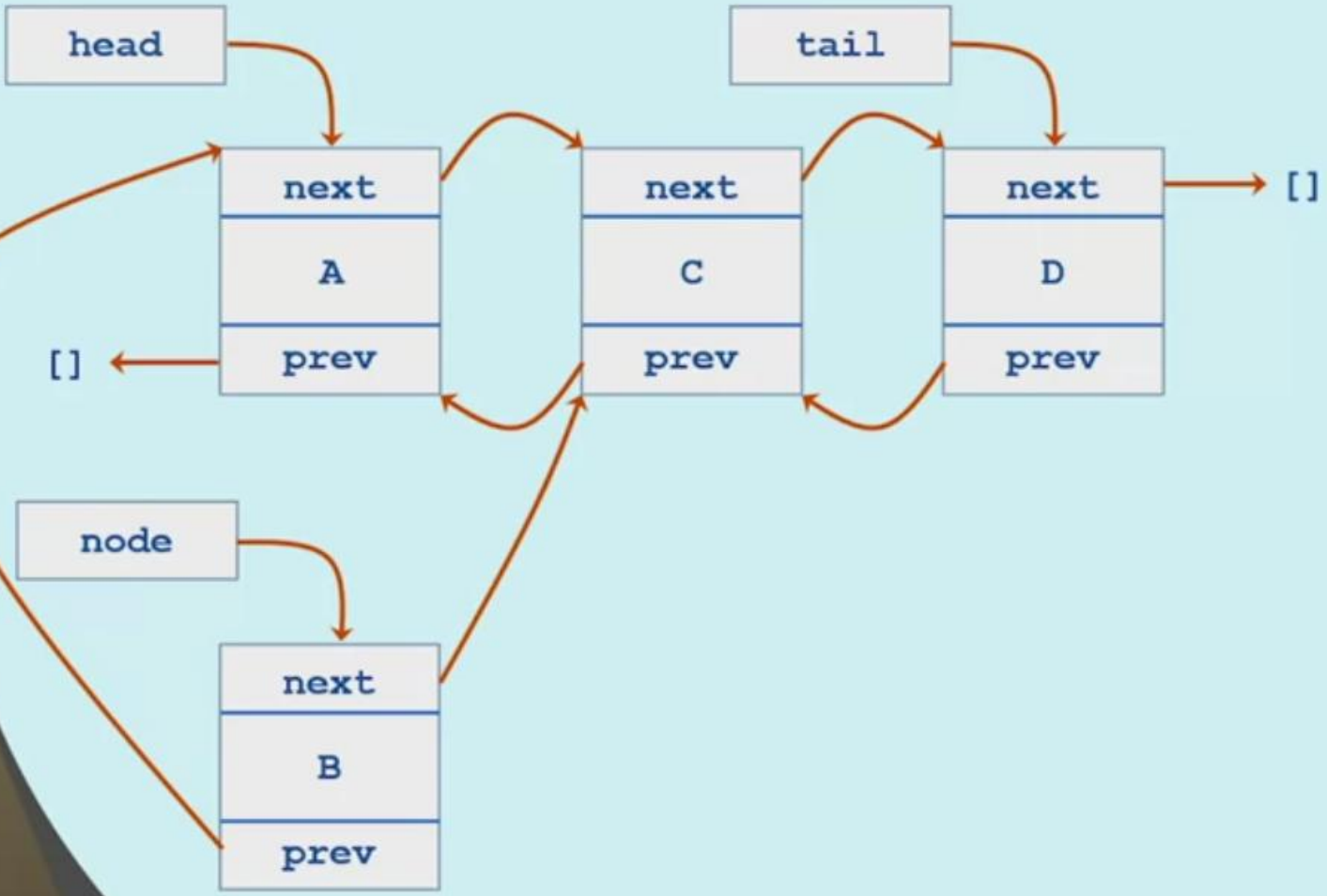
that points to the previous element of the list.

Update the Node's Pointers

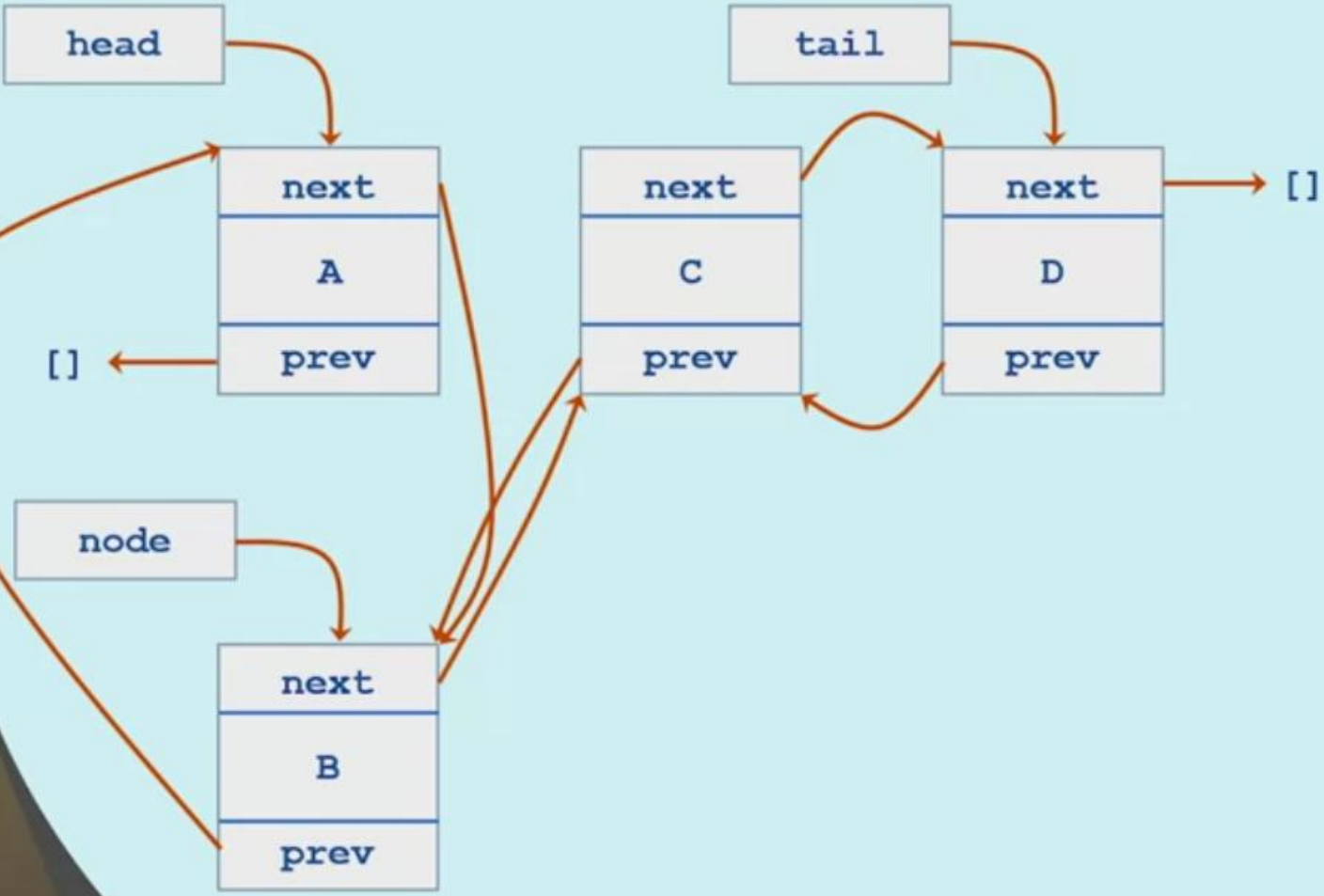


point to A and C
respectively as shown.

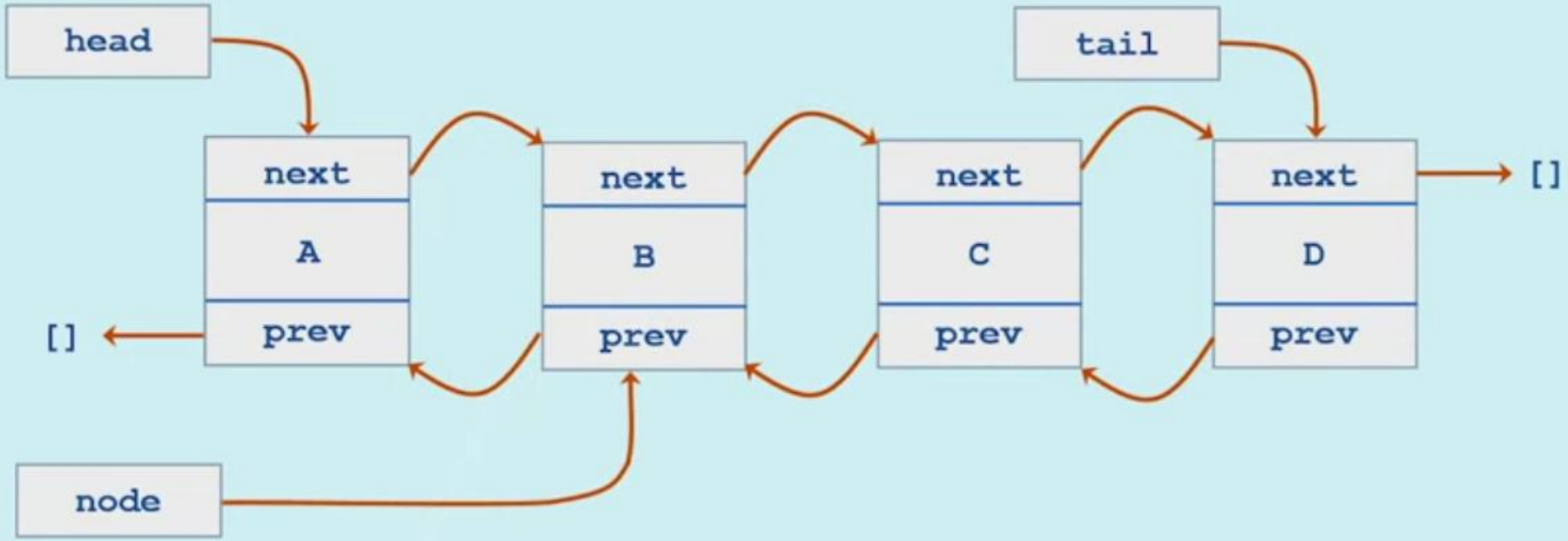
Update the Node's Pointers



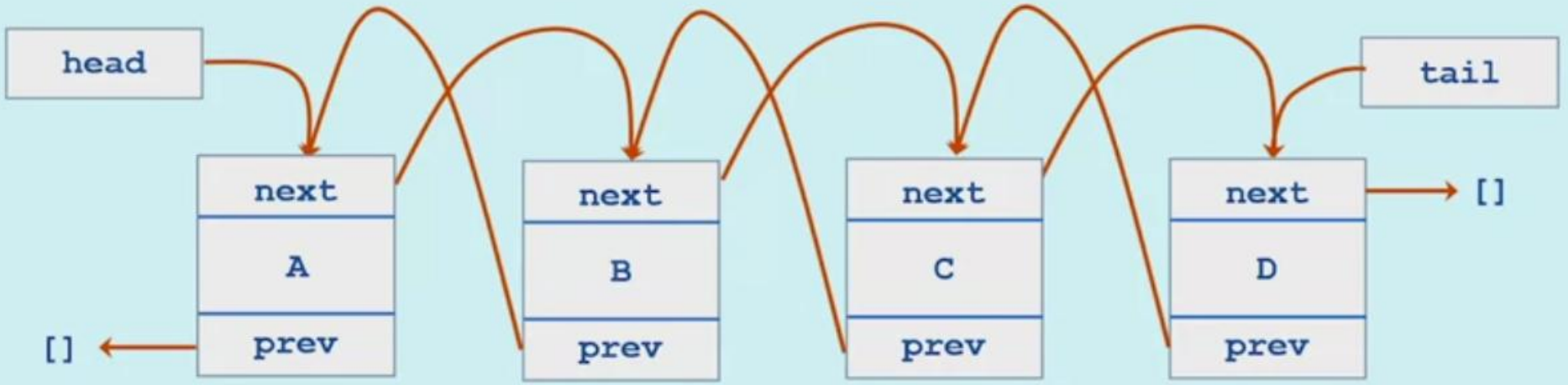
Update C's prev Pointer



The Updated List



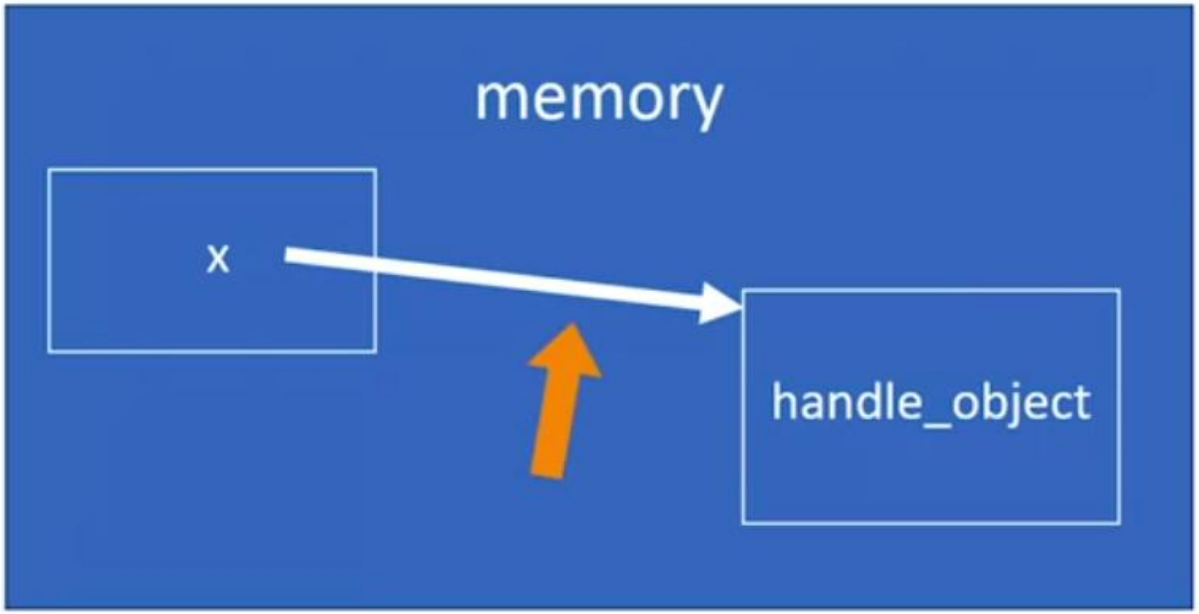
The Updated List



Hey, look, I got nothing against spaghetti.

Handle Objects

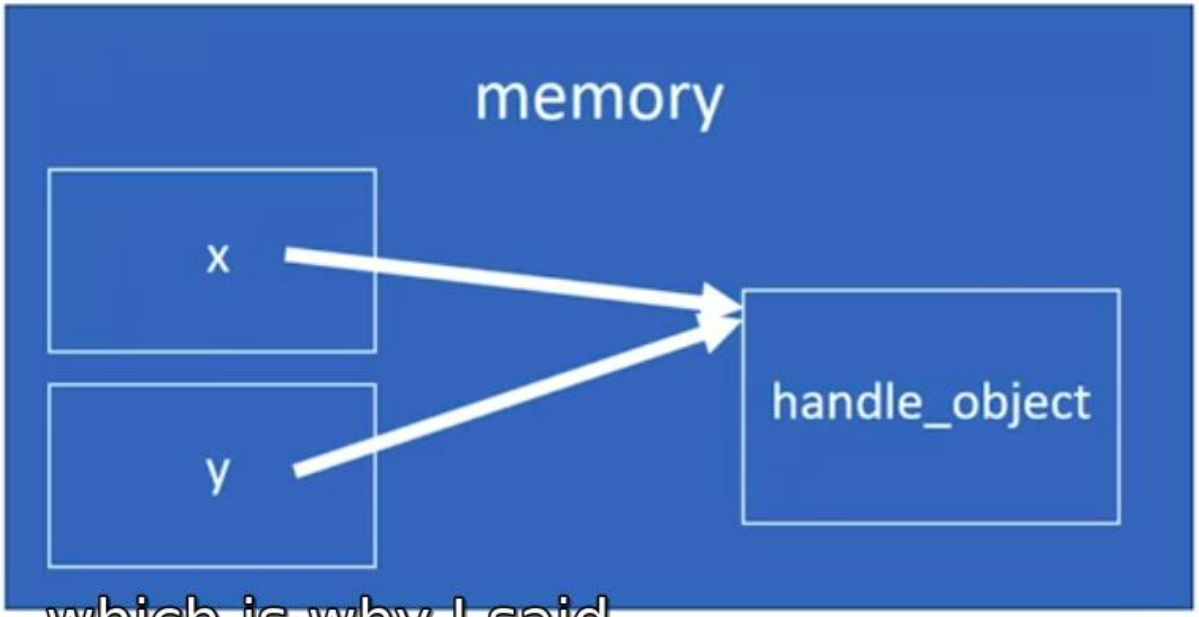
```
>> x = handle_object;
```



gets is called a handle,

Handle Objects

```
>> x = handle_object;
>> y = x;
```



which is why I said
that it's exactly what

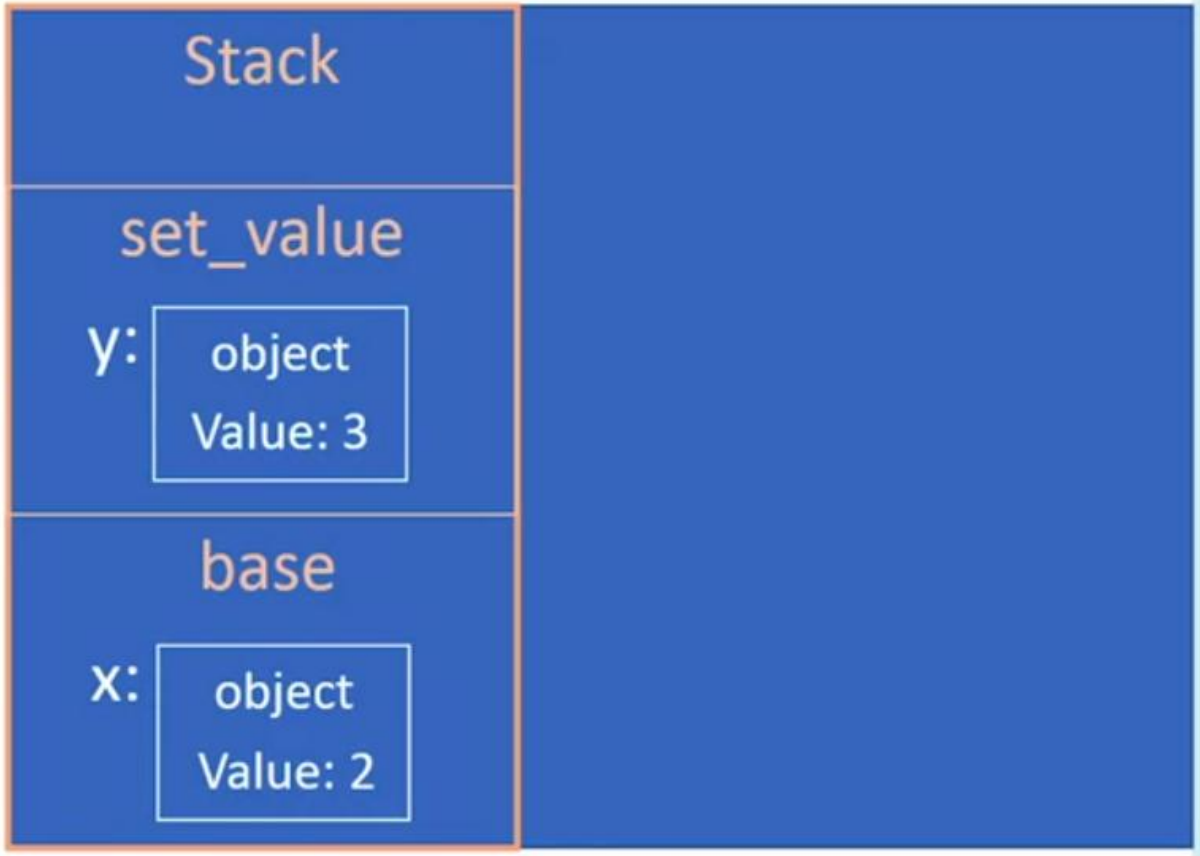
Handle Objects

set_value.m

```
1   function set_value(y)
2 -     y.Value = 3;
3 - end
```

COMMAND WINDOW

```
>> x = ValueClass;
>> x.Value = 2;
>> set_value(x);
```



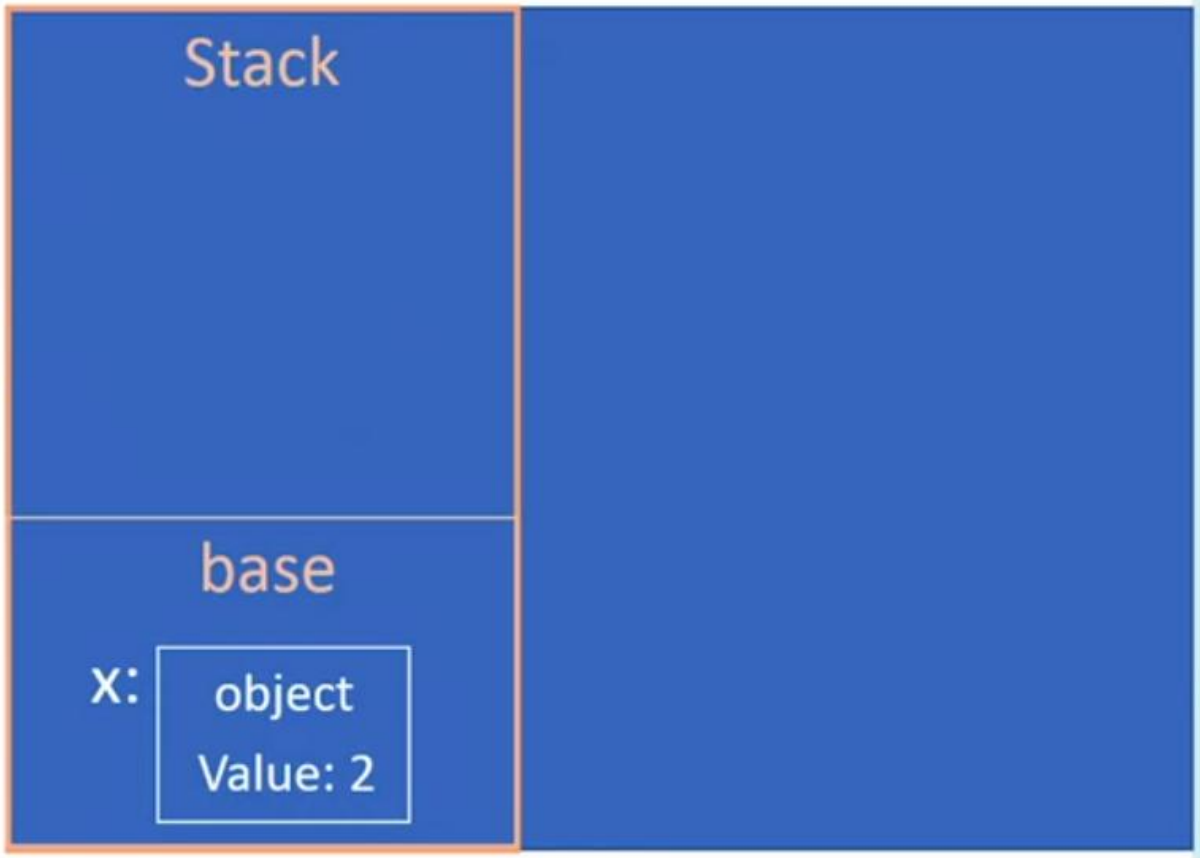
Handle Objects

set_value.m

```
1   function set_value(y)
2 -     y.Value = 3;
3 - end
```

COMMAND WINDOW

```
>> x = ValueClass;
>> x.Value = 2;
>> set_value(x);
>>
```



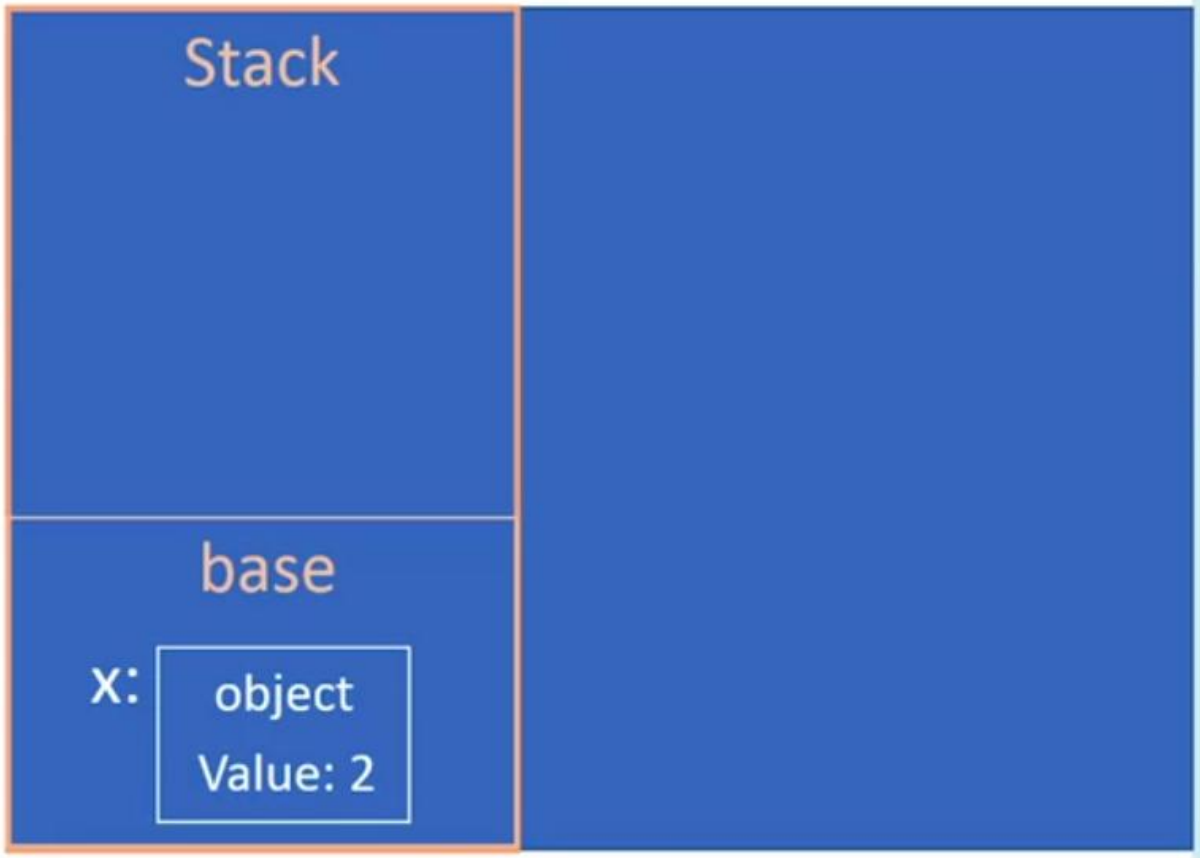
Handle Objects

set_value.m

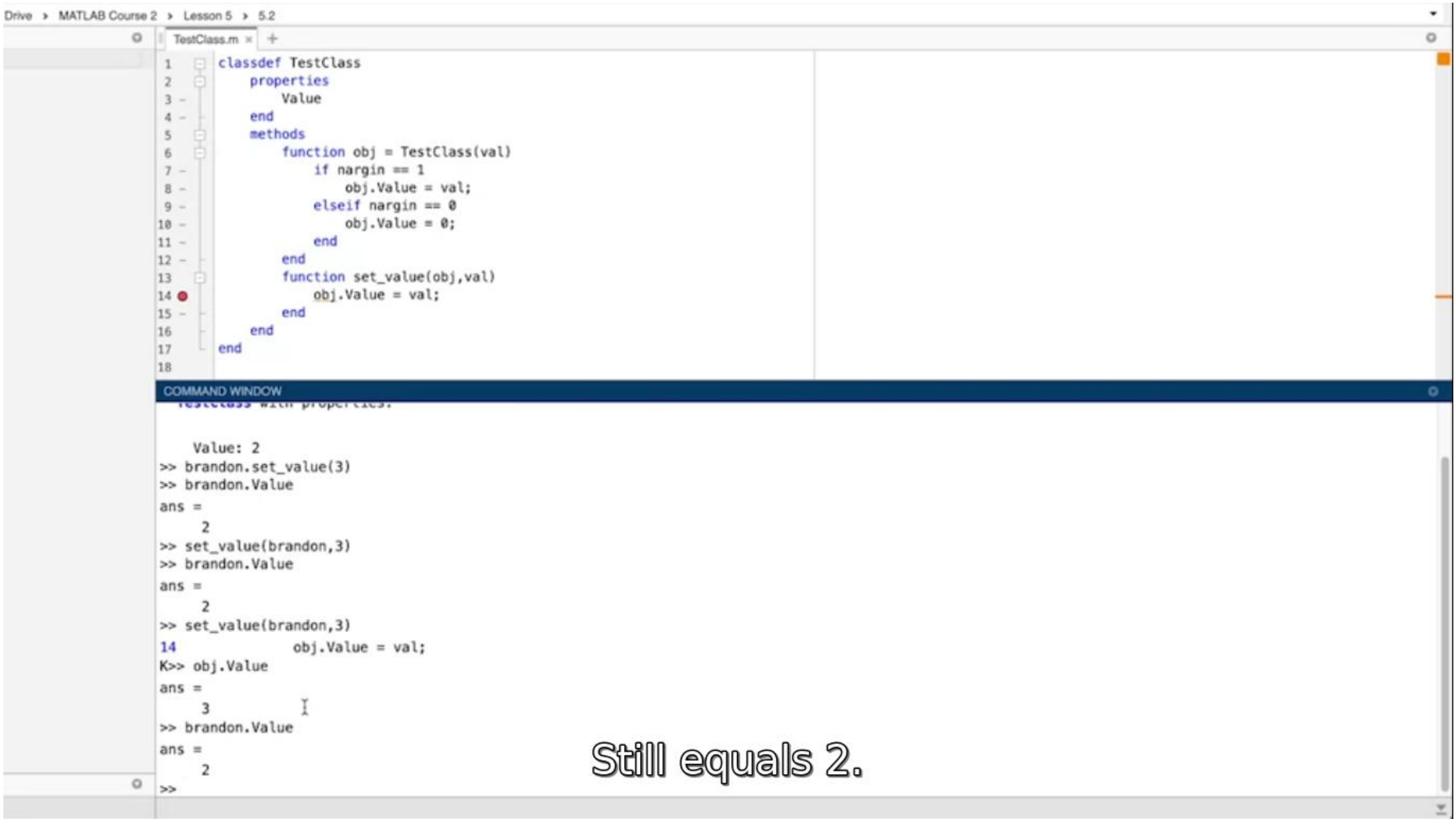
```
1   function set_value(y)
2 -     y.Value = 3;
3 - end
```

COMMAND WINDOW

```
>> x = ValueClass;
>> x.Value = 2;
>> set_value(x);
>> x.Value
ans =
     2
```

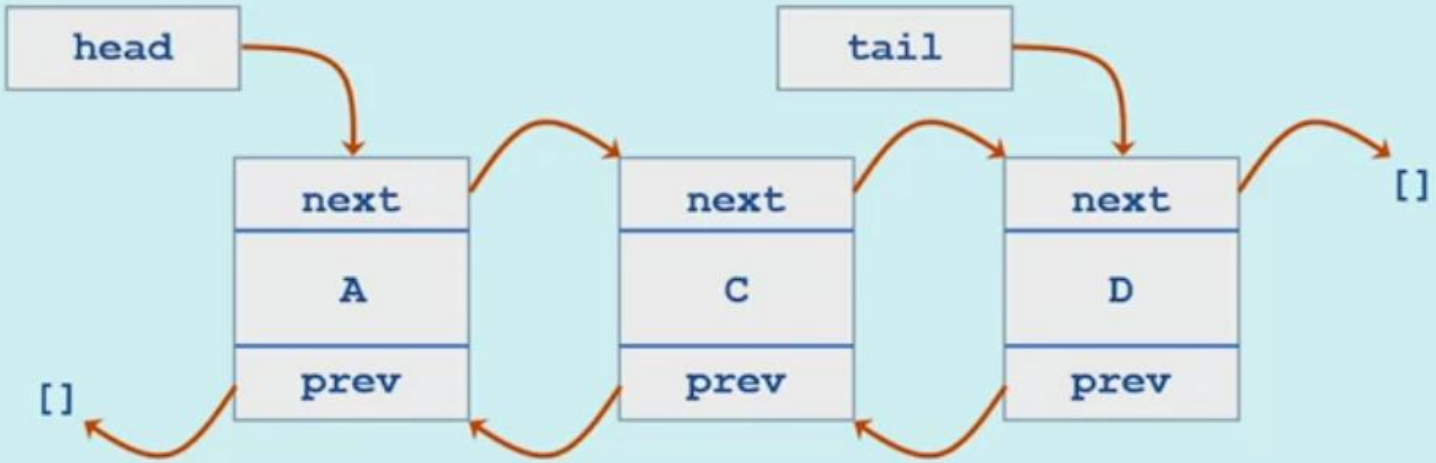



Drive › MATLAB Course 2 › Lesson 5 › 5.2
TestClass.m
classdef TestClass
properties
Value
end
methods
function obj = TestClass(val)
if nargin == 1
obj.Value = val;
elseif nargin == 0
obj.Value = 0;
end
end
function set_value(obj,val)
obj.Value = val;
end
end
end
COMMAND WINDOW
>> brandon = TestClass(2)
brandon =
TestClass with properties:
Value: 2
>> brandon.set_value(3)
>> brandon.Value
ans =
2
>>
We shouldn't be surprised,


Drive › MATLAB Course 2 › Lesson 5 › 5.2
TestClass.m
classdef TestClass
properties
Value
end
methods
function obj = TestClass(val)
if nargin == 1
obj.Value = val;
elseif nargin == 0
obj.Value = 0;
end
end
function set_value(obj,val)
obj.Value = val;
end
end
end
COMMAND WINDOW
Value: 2
>> brandon.set_value(3)
>> brandon.Value
ans =
2
>> set_value(brandon,3)
>> brandon.Value
ans =
2
>> set_value(brandon,3)
14 obj.Value = val;
K>> obj.Value
ans =
3
>> brandon.Value
ans =
2
>>
Still equals 2.

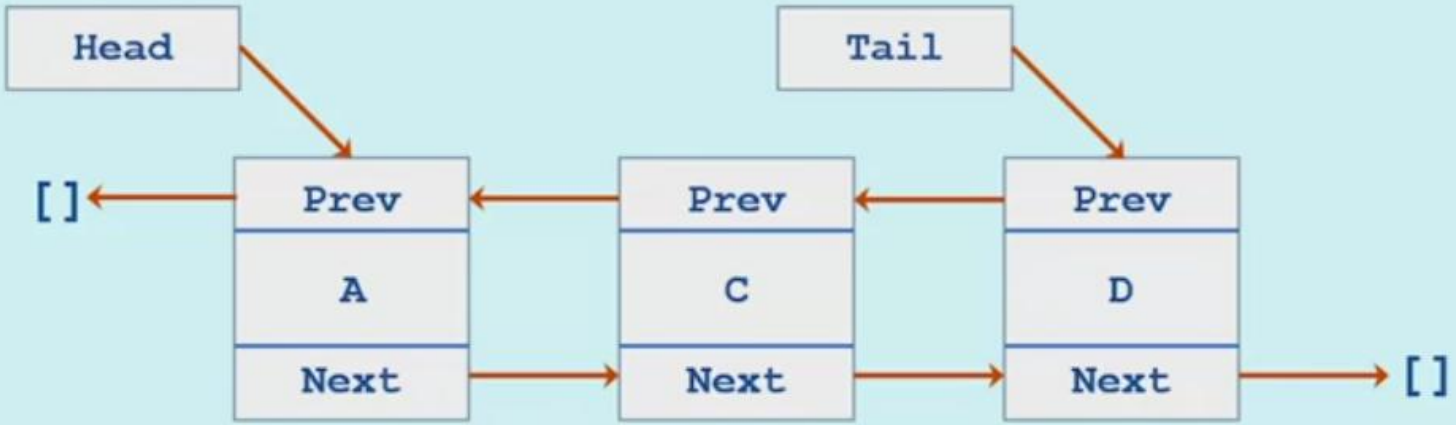

Drive › MATLAB Course 2 › Lesson 5 › 5.2
TestClass.m
classdef TestClass
properties
Value
end
methods
function obj = TestClass(val)
if nargin == 1
obj.Value = val;
elseif nargin == 0
obj.Value = 0;
end
end
function obj = set_value(obj,val)
obj.Value = val;
end
end
end
COMMAND WINDOW
brandon =
TestClass with properties:
Value: 3
>> brandon.set_value(4)
ans =
TestClass with properties:
Value: 4
>> brandon.Value
ans =
3
>> brandon = set_value(brandon,4)
brandon =
TestClass with properties:
Value: 4
>>
That's nothing new.

Doubly Linked List



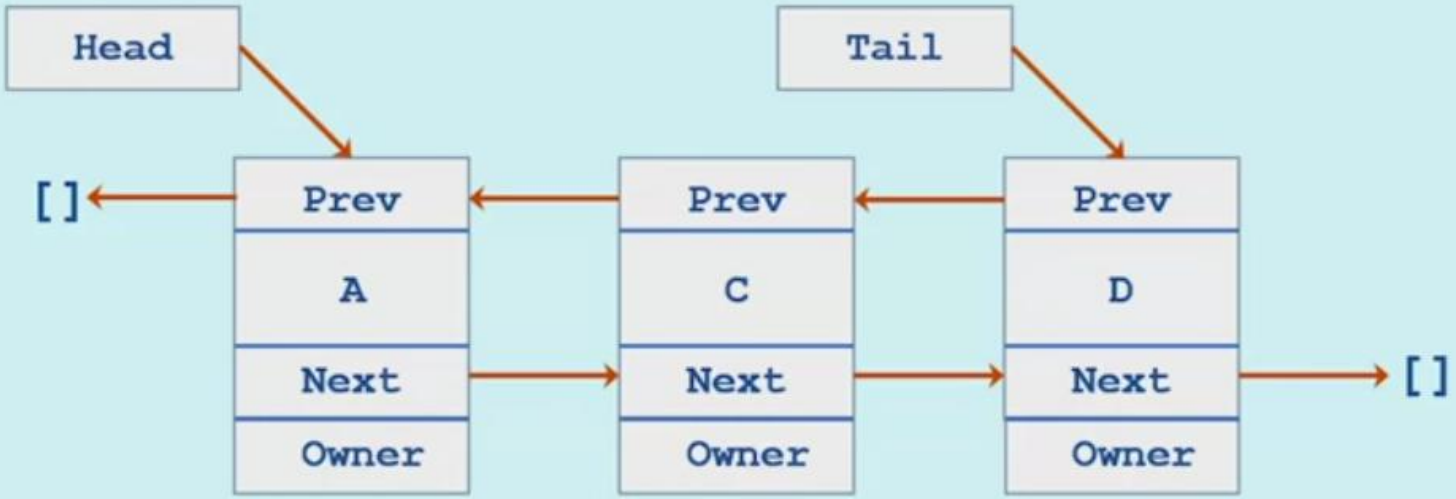
the next and previous
pointers can be changed.

Doubly Linked List



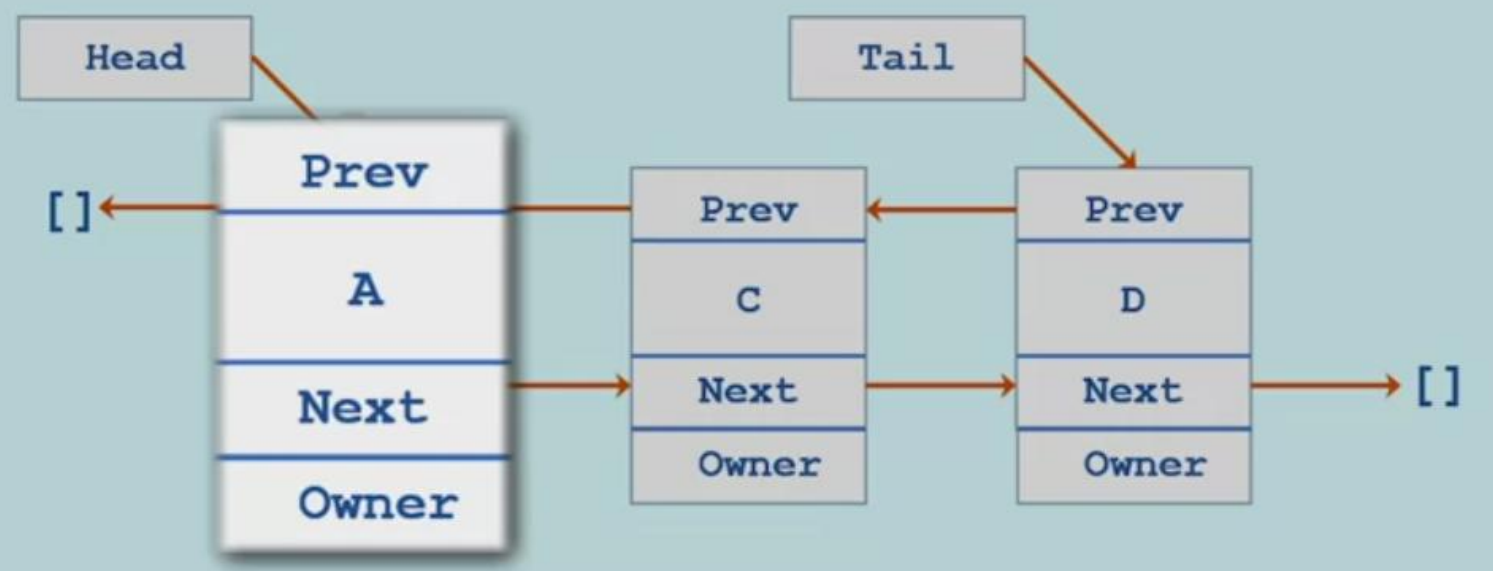
Their names can be capitalized

Doubly Linked List



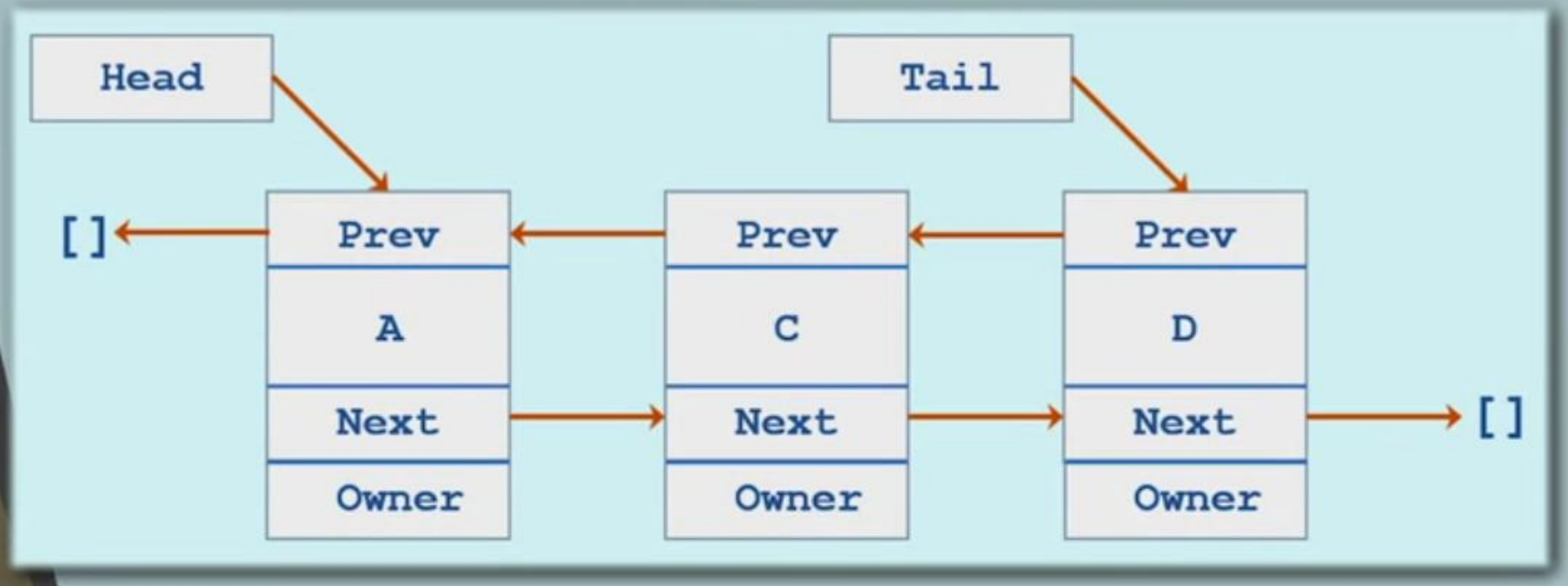
and other properties can
be added to the nodes.

Doubly Linked List



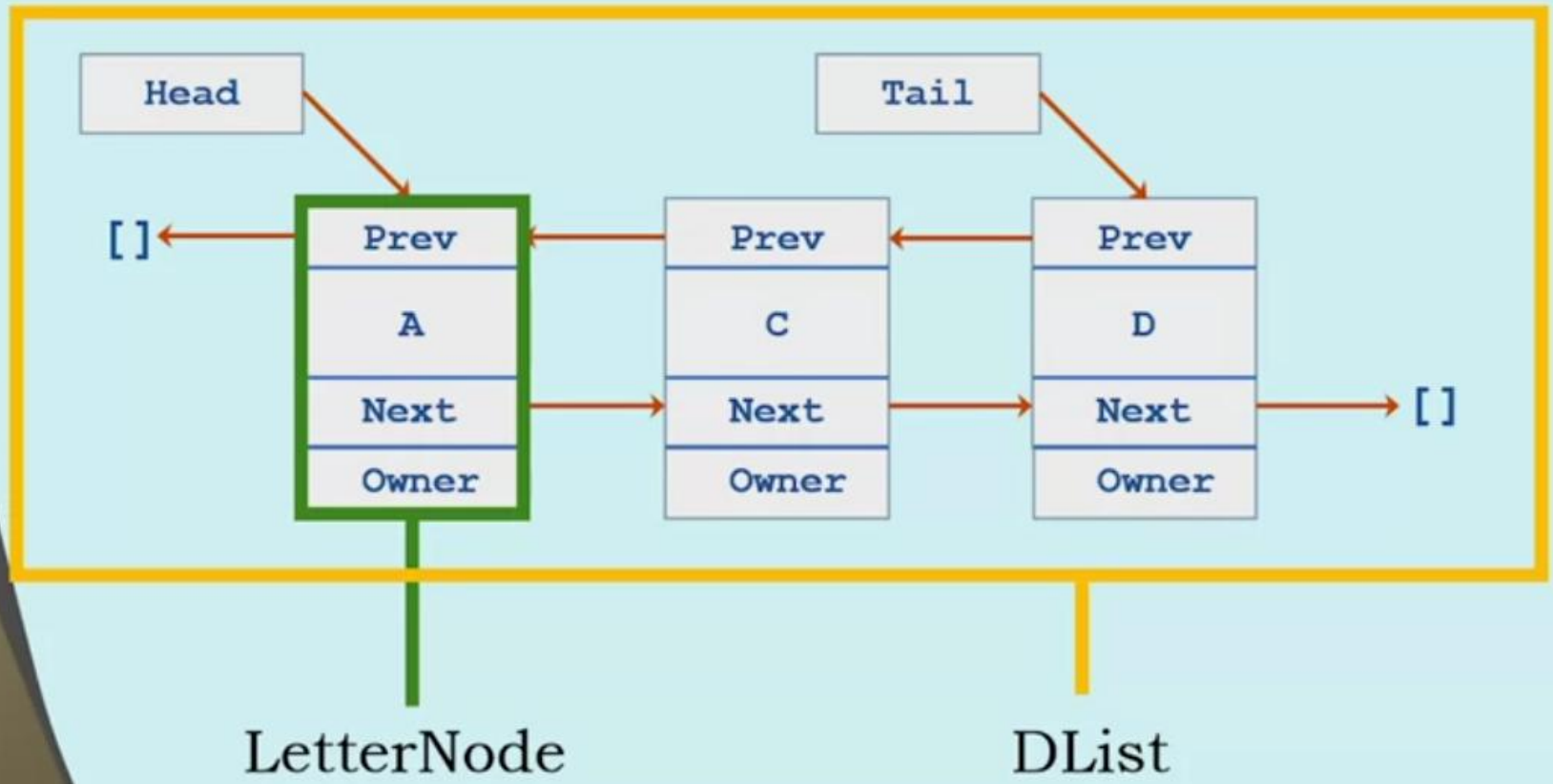
A node has some data and
a next and prev pointer.

Doubly Linked List



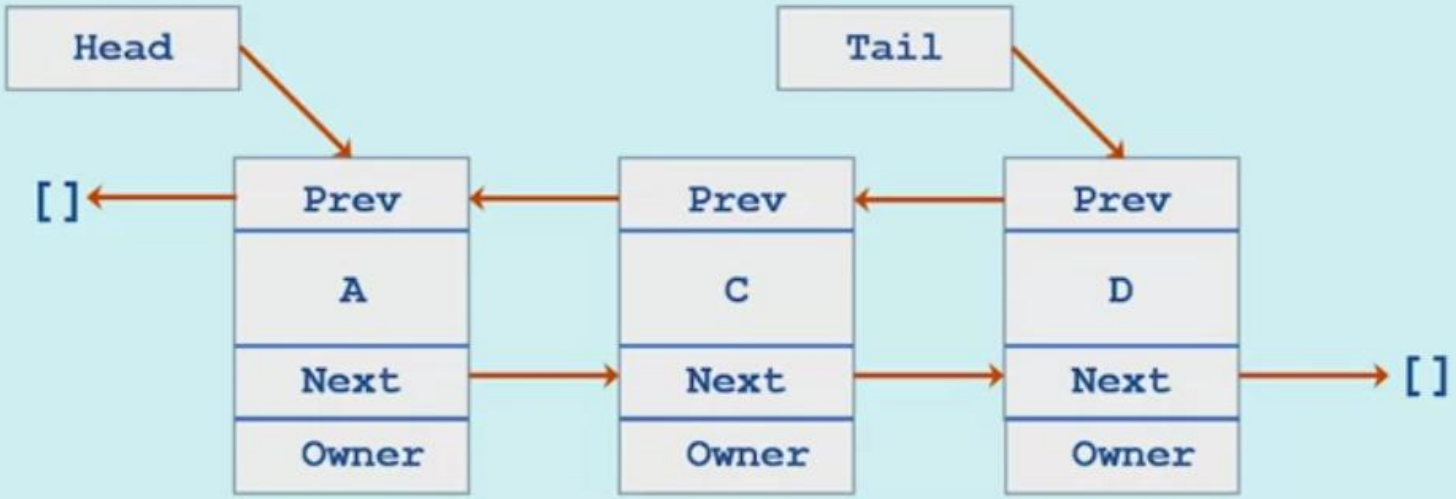
multiple nodes plus
a head and a tail.

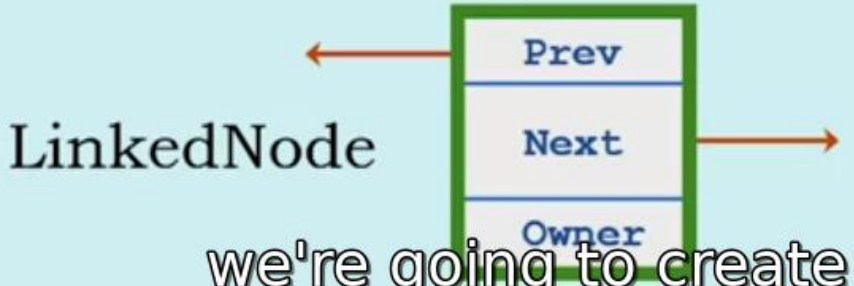
Doubly Linked List



But before we create
either of those classes,

Doubly Linked List





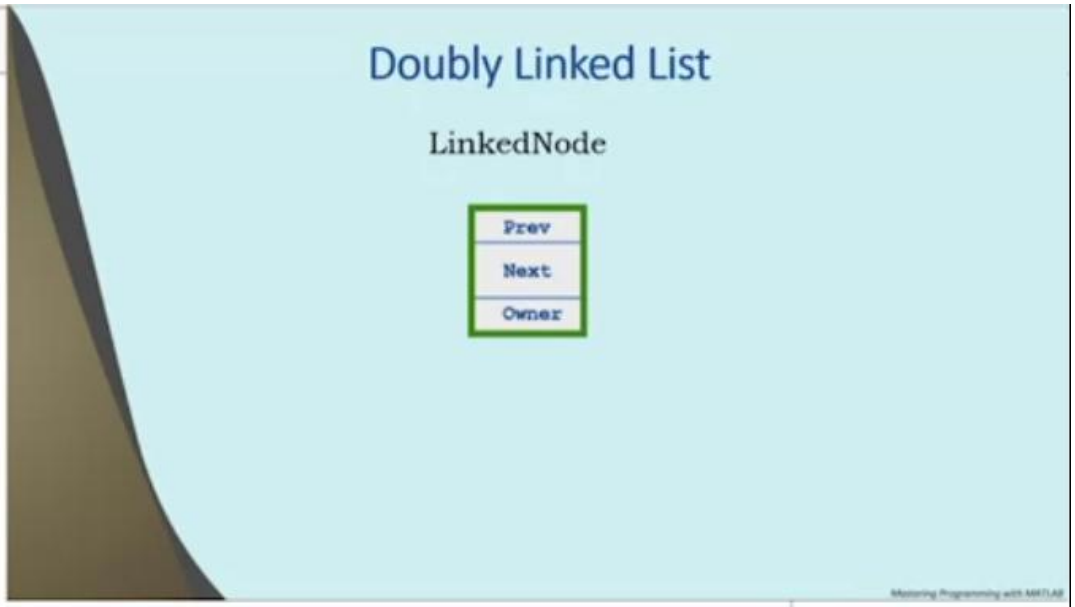
we're going to create a superclass called LinkedNode.

CURRENT FOLDER

WORKSPACE

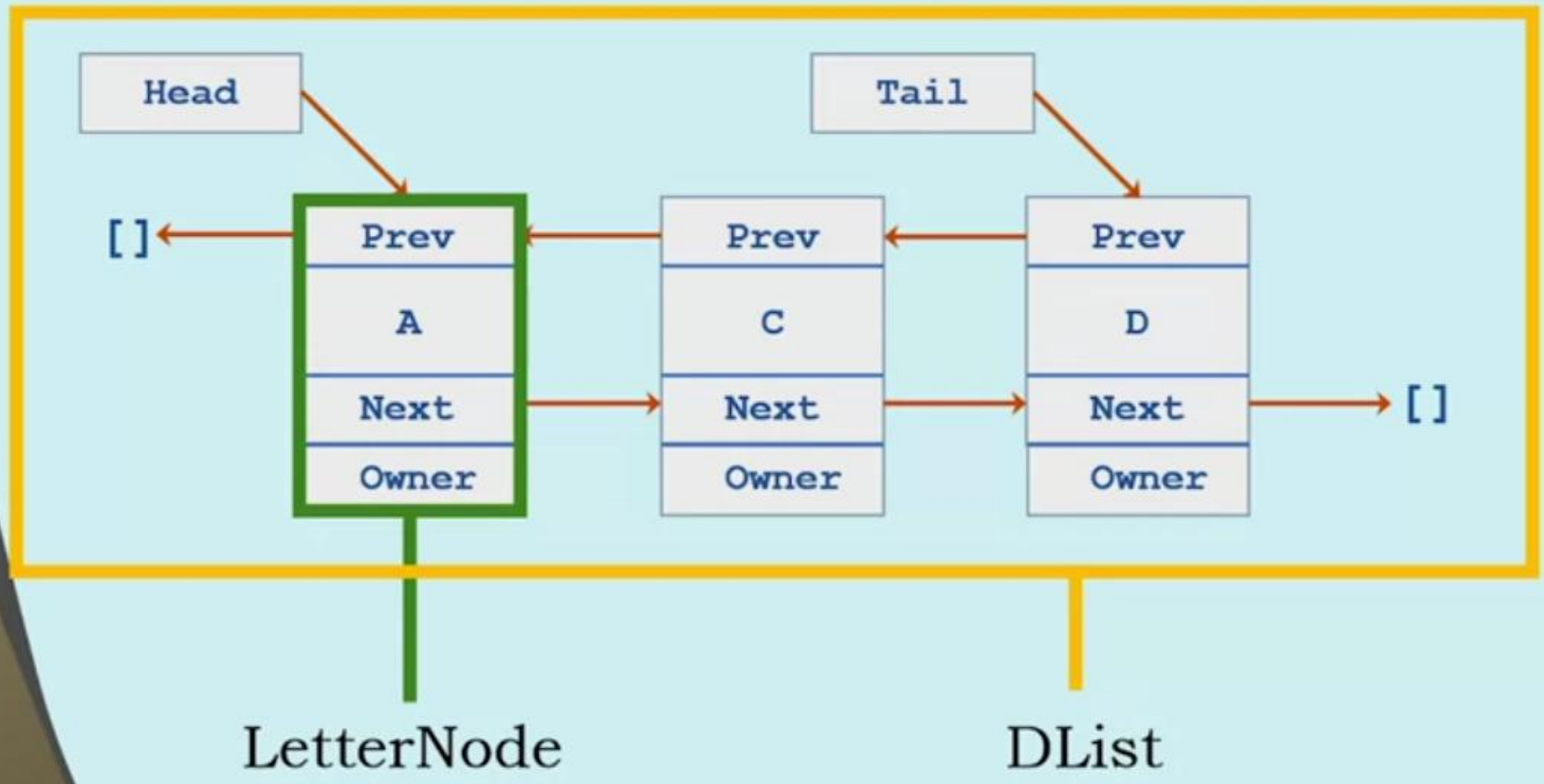
LinkedNode.m × | LetterNode.m × | DList.m × | +

```
1  classdef LinkedNode < handle
2      properties
3          Prev
4          Next
5          Owner
6      end
7      methods
8          function node = LinkedNode()
9              node.Prev = [];
10             node.Next = [];
11             node.Owner = [];
12         end
13     end
14 end
15
```

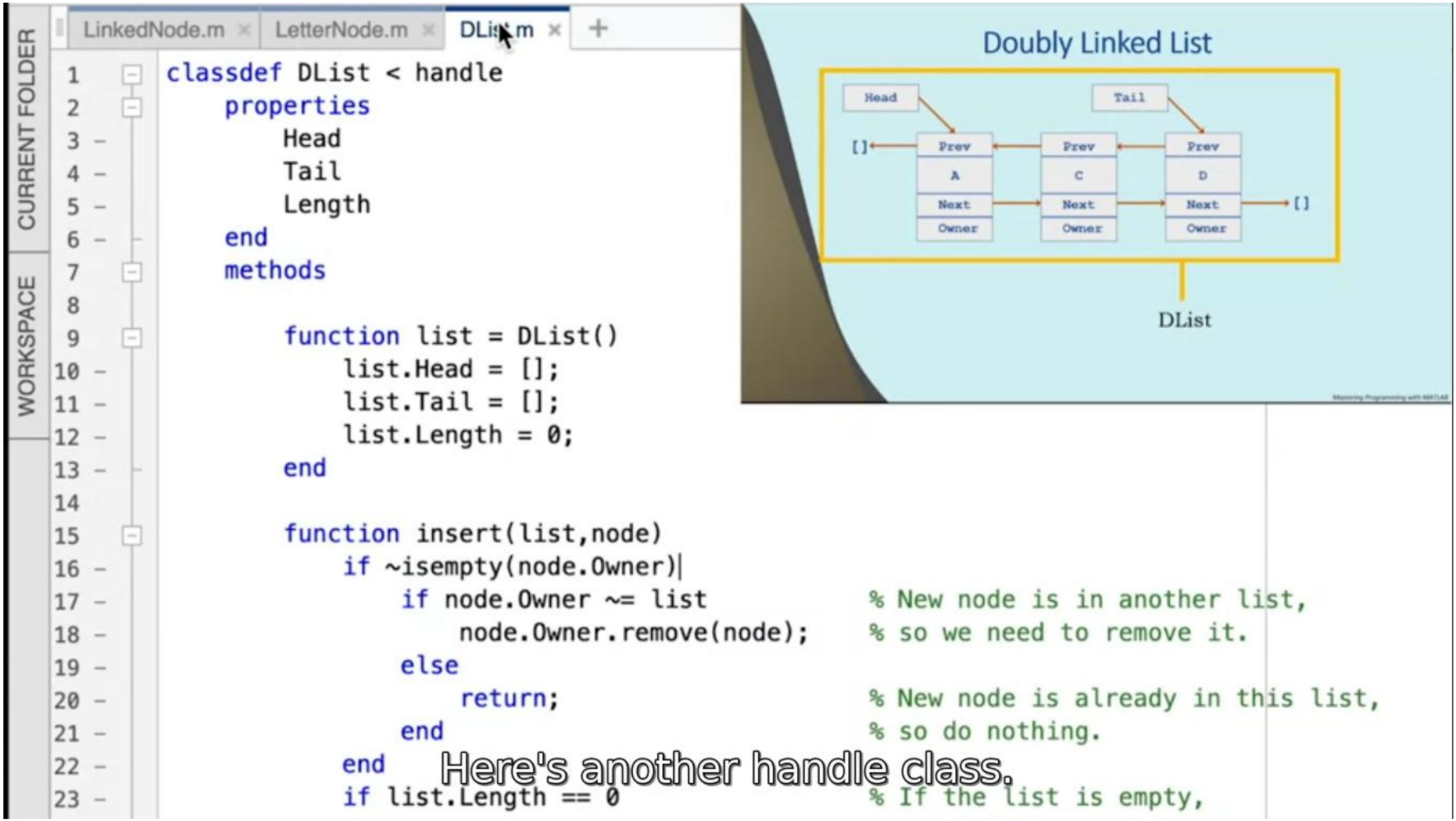


and it has three properties;

Doubly Linked List

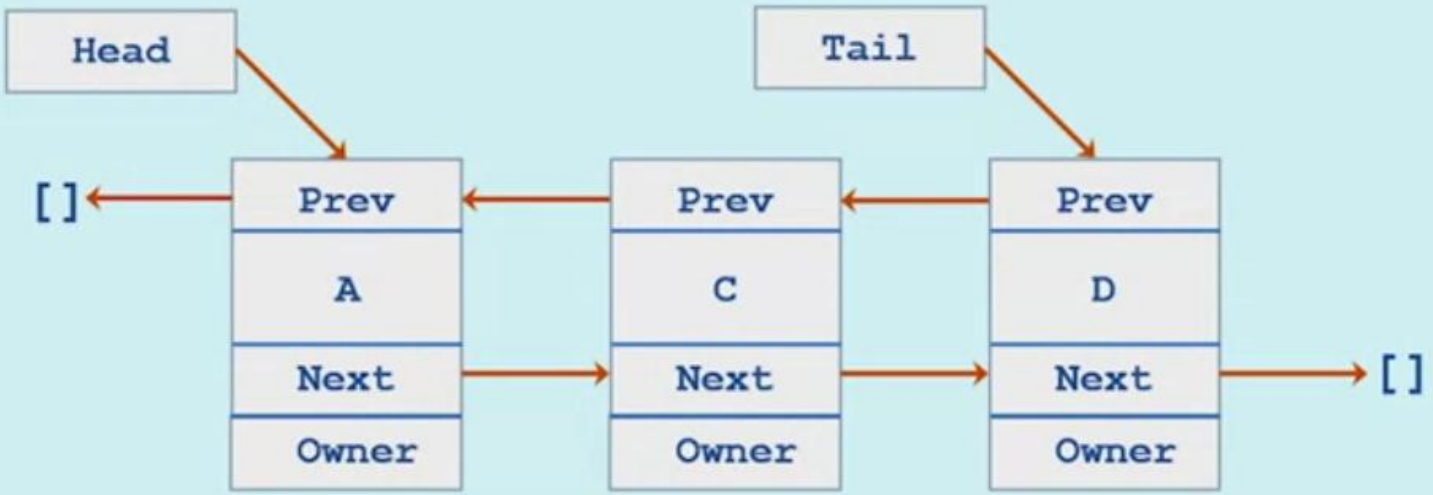


which we'll call LetterNode.


CURRENT FOLDER
WORKSPACE
LinkedNode.m
LetterNode.m
DList.m
Doubly Linked List
Head
Tail
[]
Prev
A
Next
Owner
Prev
C
Next
Owner
Prev
D
Next
Owner
[]
DList
classdef DList < handle
properties
Head
Tail
Length
end
methods
function list = DList()
list.Head = [];
list.Tail = [];
list.Length = 0;
end
function insert(list,node)
if ~isempty(node.Owner)
if node.Owner ~= list % New node is in another list,
node.Owner.remove(node); % so we need to remove it.
else
return; % New node is already in this list,
end % so do nothing.
end
if list.Length == 0 % If the list is empty,
Here's another handle class.

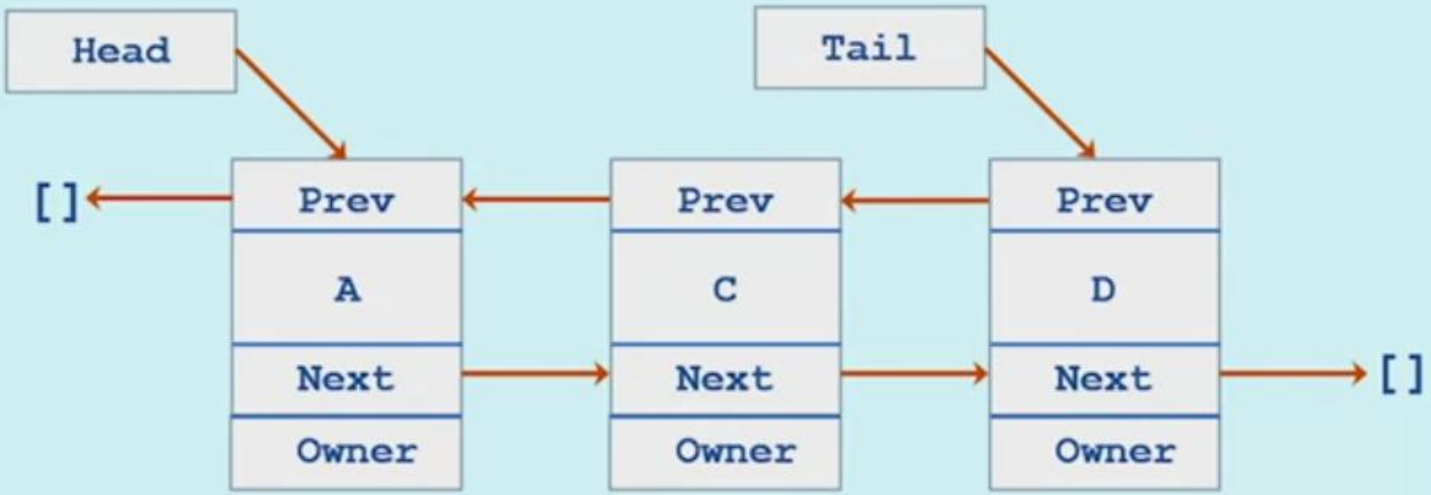
```
function insert(list,node)
    if ~isempty(node.Owner)
        if node.Owner ~= list           % New node is in another list,
            node.Owner.remove(node);    % so we need to remove it.
        else
            return;                     % New node is already in this list,
        end                             % so do nothing.
    end
    if list.Length == 0                 % If the list is empty,
        list.Head = node;               % put new node at the head,
    else
        list.Tail.Next = node;          % else, point tail node at it.
    end
    node.Next = [];                     % New node is at the end.
    node.Prev = list.Tail;              % Previous node is old tail node.
    list.Tail = node;                   % Make Tail node point at new node.
    list.Length = list.Length + 1;
    node.Owner = list;
end % insert
```

Doubly Linked List



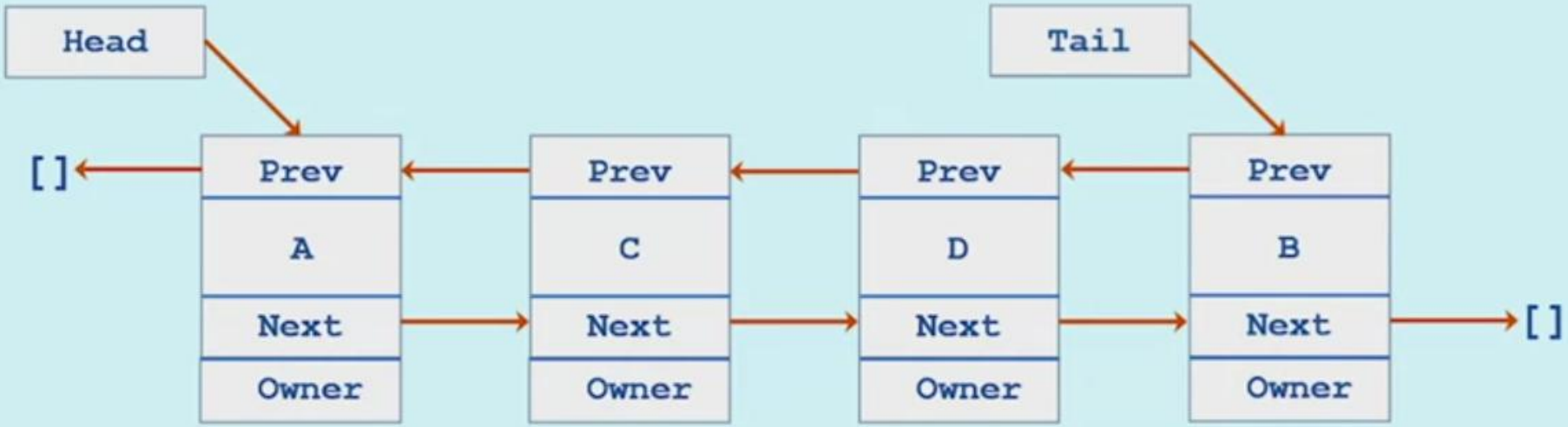
our doubly linked list.

DList



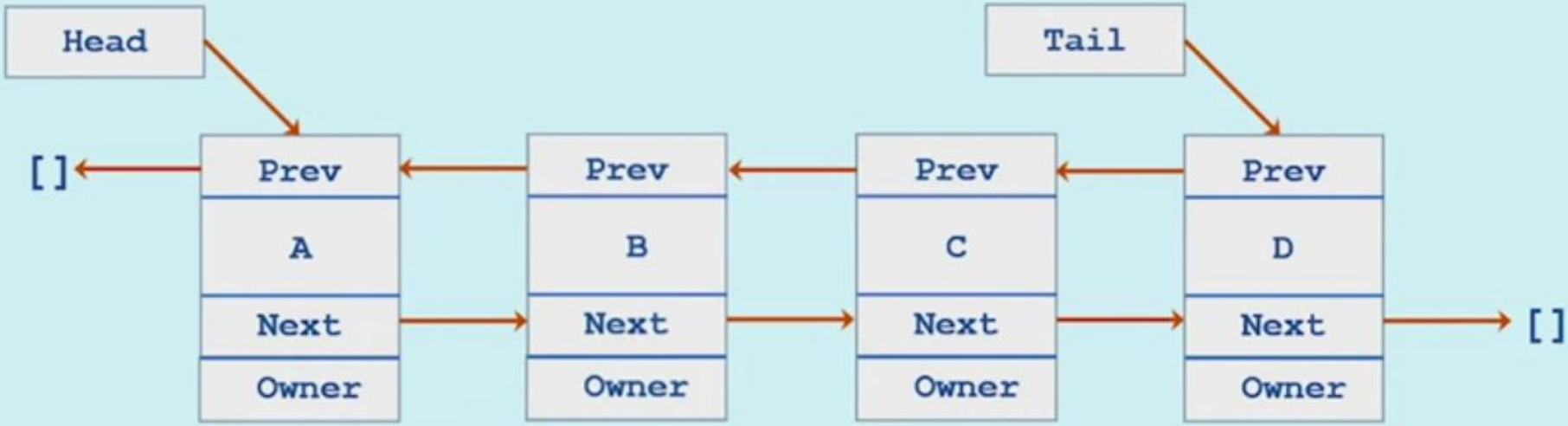
Idealist class and the
previous lecture stores data

DList

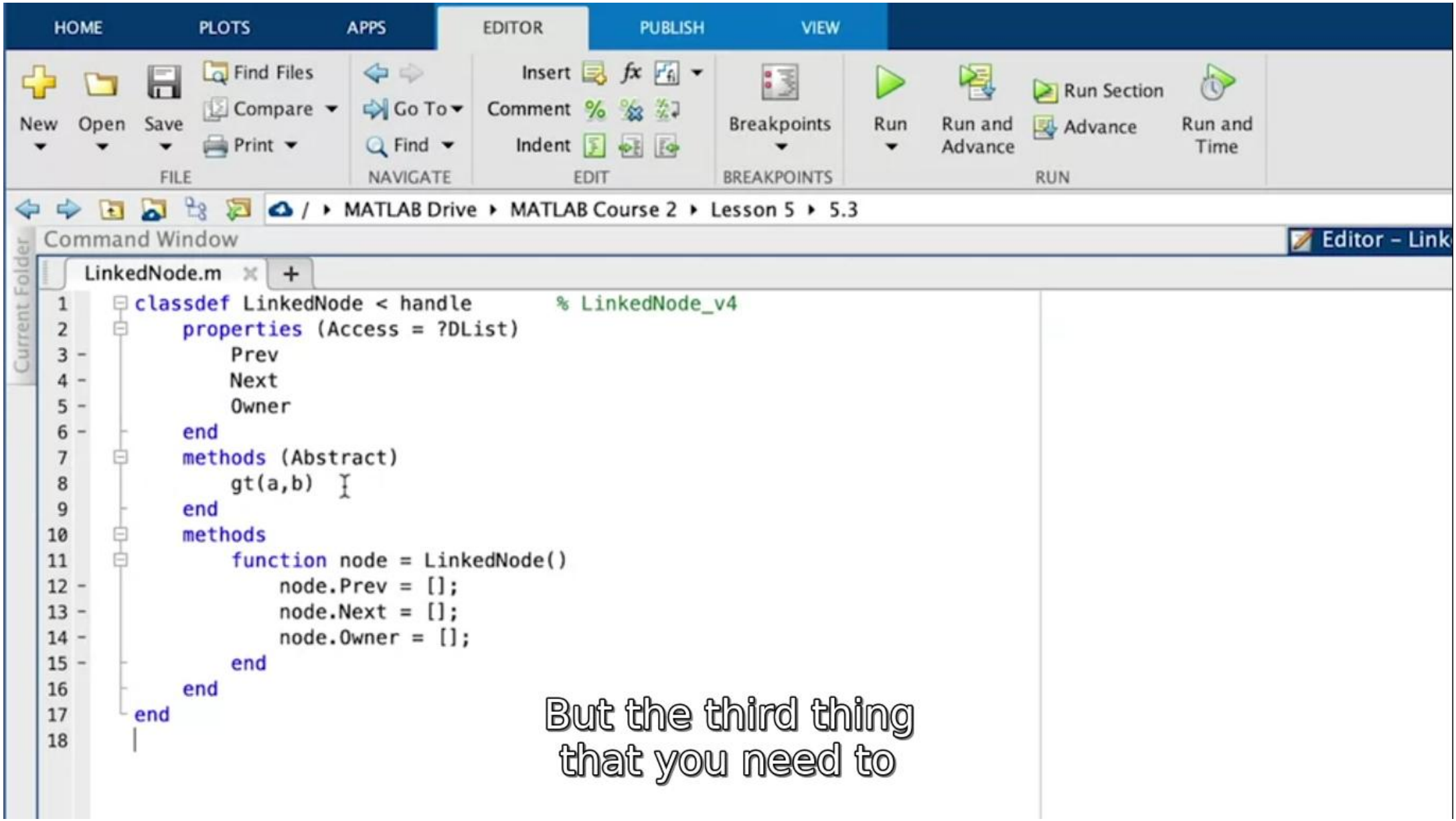


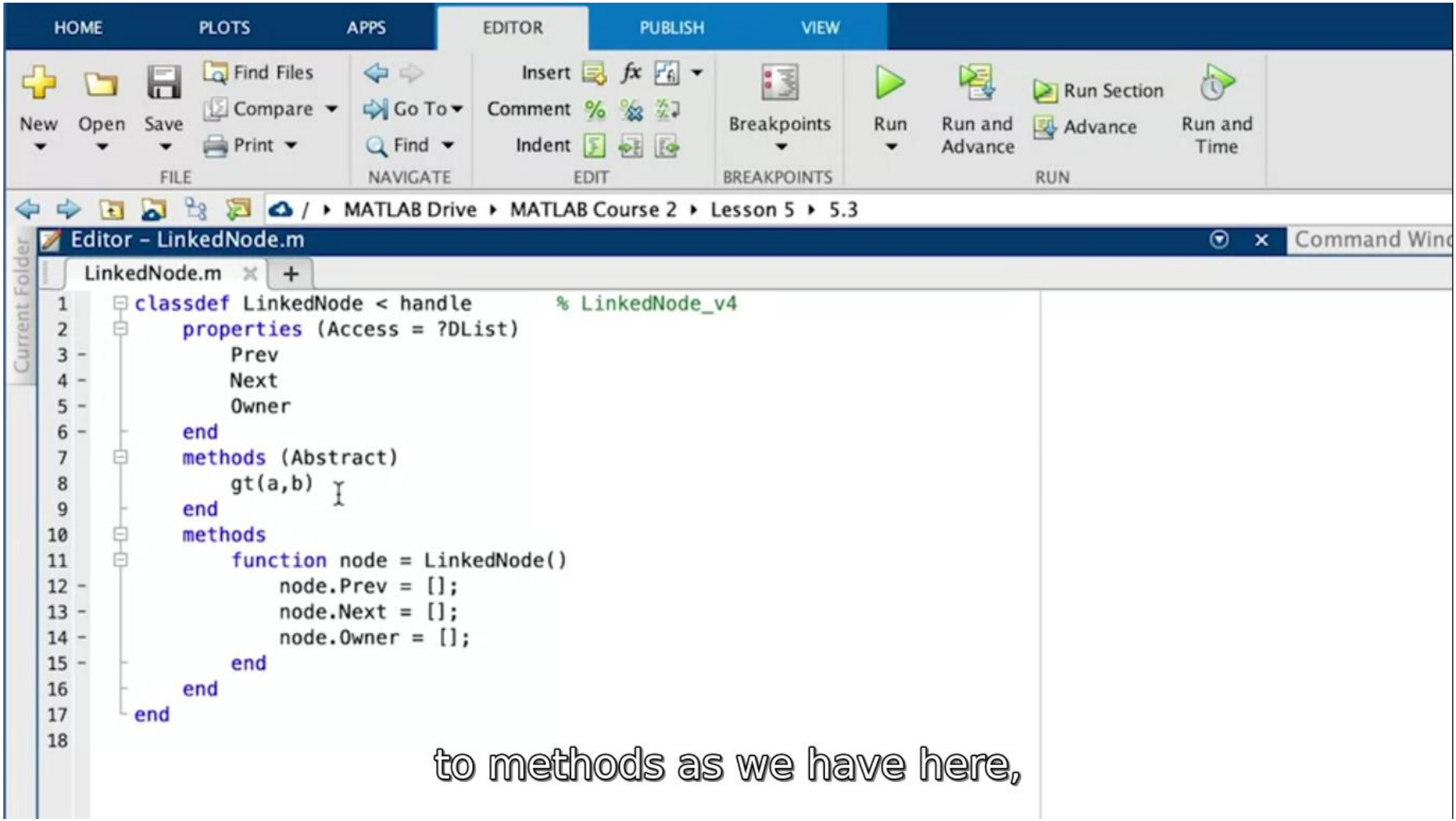
it goes at the end of the list.

OrderedList



OrderedList that will insert items in increasing order.


HOME
PLOTS
APPS
EDITOR
PUBLISH
VIEW
New
Open
Save
Find Files
Compare
Print
FILE
Go To
Find
NAVIGATE
Insert
Comment
Indent
EDIT
Breakpoints
BREAKPOINTS
Run
Run and Advance
Run Section
Advance
Run and Time
RUN
/ ▸ MATLAB Drive ▸ MATLAB Course 2 ▸ Lesson 5 ▸ 5.3
Command Window
Editor - Link
Current Folder
LinkedNode.m
1 classdef LinkedNode < handle % LinkedNode_v4
2 properties (Access = ?DList)
3 Prev
4 Next
5 Owner
6 end
7 methods (Abstract)
8 gt(a,b)
9 end
10 methods
11 function node = LinkedNode()
12 node.Prev = [];
13 node.Next = [];
14 node.Owner = [];
15 end
16 end
17 end
18
But the third thing
that you need to

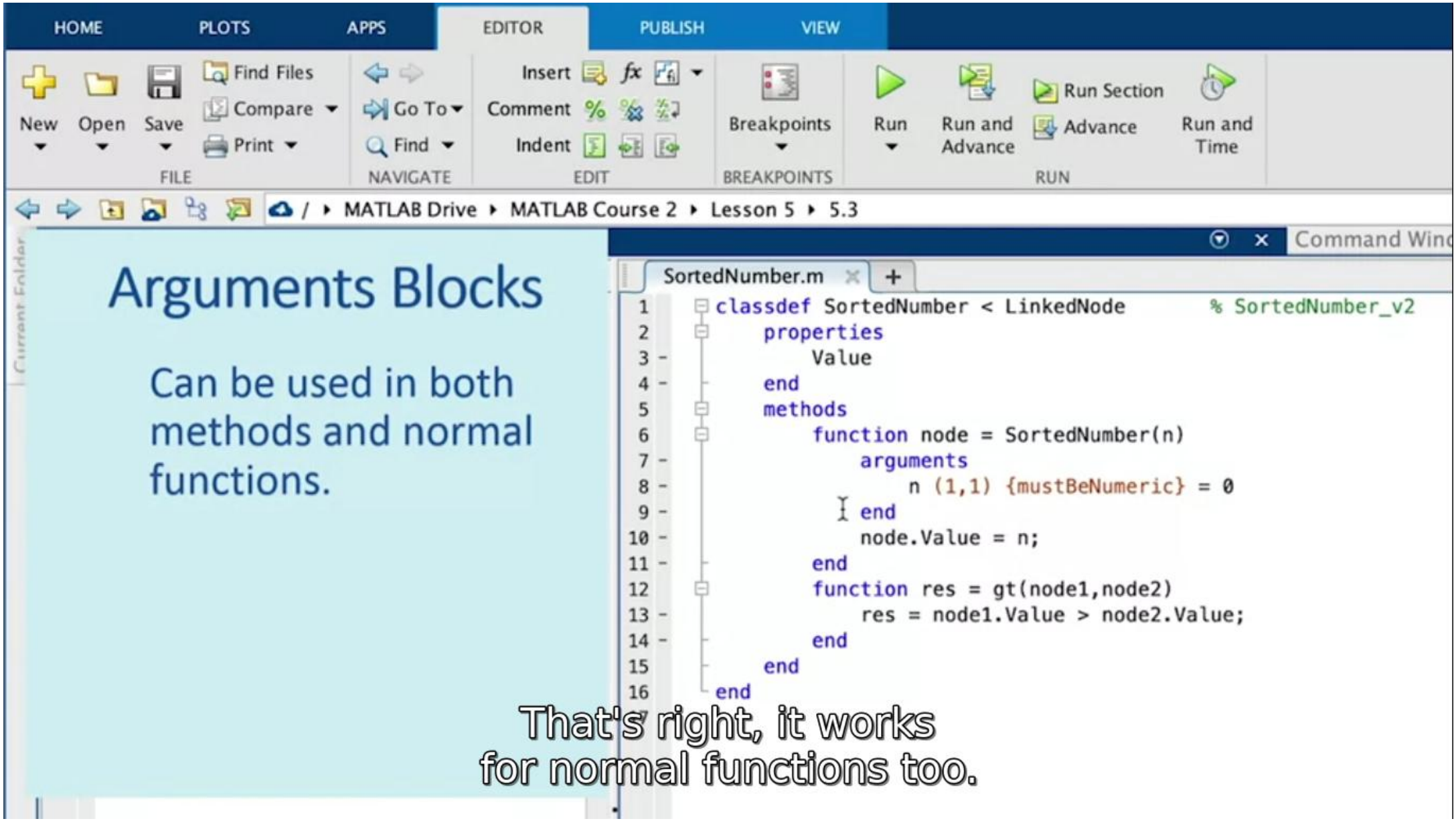


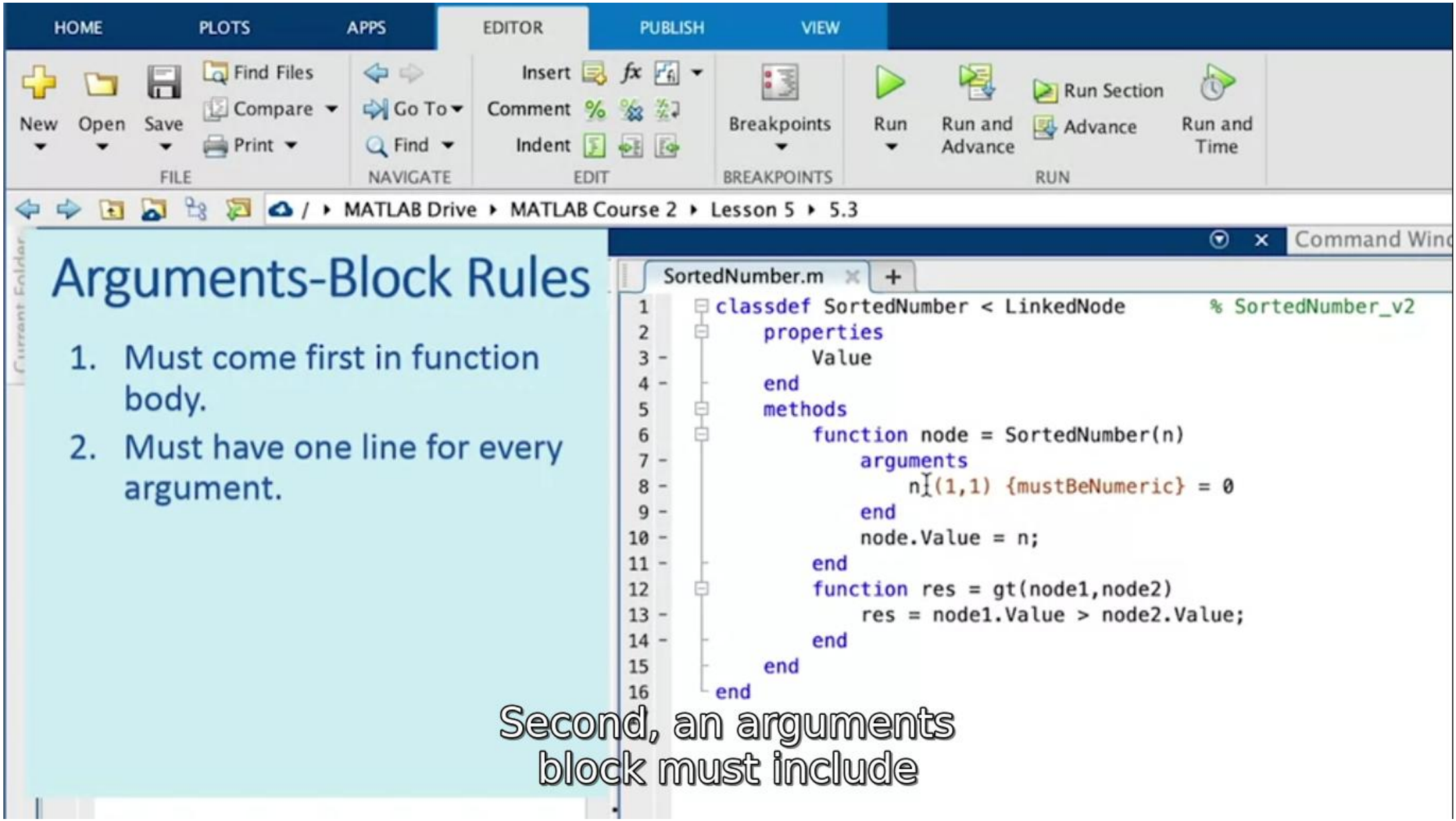
to methods as we have here,

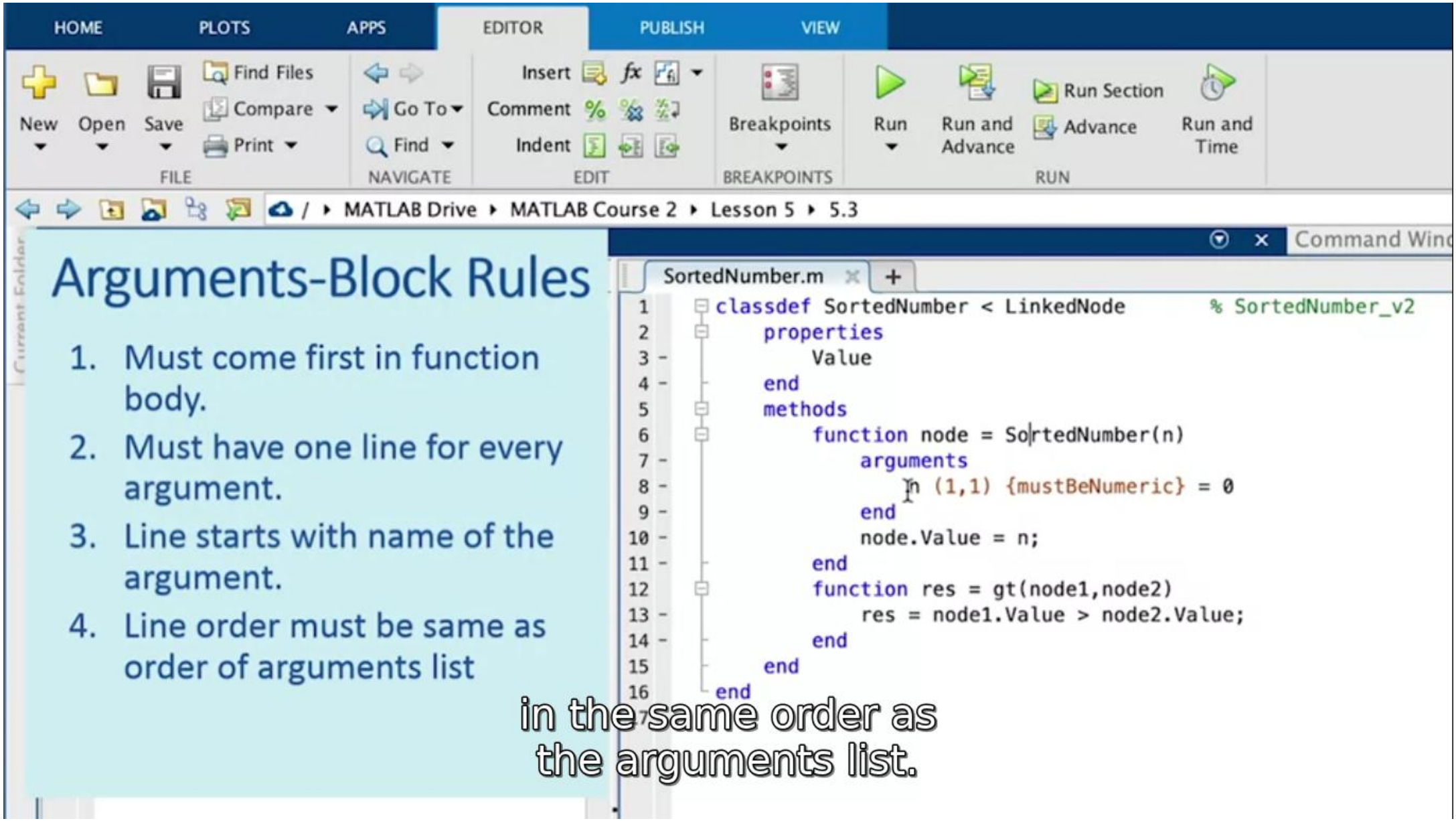
MATLAB Keywords

arguments	enumeration	persistent
break	events	properties
case	for	return
catch	function	spmd
classdef	global	switch
continue	if	try
else	methods	while
elseif	otherwise	
end	parfor	

in the entire vast
kingdom of MATLAB.


HOME
PLOTS
APPS
EDITOR
PUBLISH
VIEW
New
Open
Save
Find Files
Compare
Print
FILE
Go To
Find
NAVIGATE
Insert
Comment
Indent
EDIT
Breakpoints
BREAKPOINTS
Run
Run and Advance
Run Section
Advance
Run and Time
RUN
/ ▸ MATLAB Drive ▸ MATLAB Course 2 ▸ Lesson 5 ▸ 5.3
Arguments Blocks
Can be used in both methods and normal functions.
SortedNumber.m
1 classdef SortedNumber < LinkedNode % SortedNumber_v2
2 properties
3 Value
4 end
5 methods
6 function node = SortedNumber(n)
7 arguments
8 n (1,1) {mustBeNumeric} = 0
9 end
10 node.Value = n;
11 end
12 function res = gt(node1,node2)
13 res = node1.Value > node2.Value;
14 end
15 end
16 end
That's right, it works for normal functions too.


HOME
PLOTS
APPS
EDITOR
PUBLISH
VIEW
New
Open
Save
Find Files
Compare
Print
FILE
Go To
Find
NAVIGATE
Insert
Comment
Indent
EDIT
Breakpoints
BREAKPOINTS
Run
Run and Advance
Run Section
Advance
Run and Time
RUN
MATLAB Drive
MATLAB Course 2
Lesson 5
5.3
Arguments-Block Rules
1. Must come first in function body.
2. Must have one line for every argument.
SortedNumber.m
1 classdef SortedNumber < LinkedNode % SortedNumber_v2
2 properties
3 Value
4 end
5 methods
6 function node = SortedNumber(n)
7 arguments
8 n(1,1) {mustBeNumeric} = 0
9 end
10 node.Value = n;
11 end
12 function res = gt(node1,node2)
13 res = node1.Value > node2.Value;
14 end
15 end
16 end
Second, an arguments block must include


HOME
PLOTS
APPS
EDITOR
PUBLISH
VIEW
New
Open
Save
Find Files
Compare
Print
FILE
Go To
Find
NAVIGATE
Insert
Comment
Indent
EDIT
Breakpoints
BREAKPOINTS
Run
Run and Advance
Run Section
Advance
Run and Time
RUN
MATLAB Drive
MATLAB Course 2
Lesson 5
5.3
Arguments-Block Rules
1. Must come first in function body.
2. Must have one line for every argument.
3. Line starts with name of the argument.
4. Line order must be same as order of arguments list
SortedNumber.m
classdef SortedNumber < LinkedNode % SortedNumber_v2
properties
Value
end
methods
function node = SortedNumber(n)
arguments
n (1,1) {mustBeNumeric} = 0
end
node.Value = n;
end
function res = gt(node1,node2)
res = node1.Value > node2.Value;
end
end
end
in the same order as
the arguments list.

Requirements for Delete Method

- It can have only one input argument, which is the object that it's deleting.
- It can't have any output arguments.
- It can't include an arguments block for validating the input.
- It can't be declared abstract.

and it can't be
declared abstract.

Operator Overloading

<

==

>

First, we learned how to overload

Operator Overloading

<

==

>

```
function res = gt(node1,node2)
    res = node1.Value > node2.Value;
end
```

an operator using greater
than as our example,

Abstract Attribute

```
methods (Abstract)
    gt(a,b)
end
```

We learned how to declare
methods to be abstract,

Abstract Attribute

```
methods (Abstract)
    gt(a,b)
end
```

```
>> A = LinkedNode
Abstract classes cannot be instantiated.  Class
'LinkedNode' defines abstract methods and/or
properties.
```

cannot be used to define objects.

Abstract-method Implementation

```
function res = gt(node1,node2)
    res = node1.Value > node2.Value;
end
```

it to abstract methods and

Argument Validation

```
arguments
    n (1,1) {mustBeNumeric} = 0
end
```

We learned how to validate function arguments,

Access Extends to Subclasses of Listed Classes

```
properties (Access = ?DList)
```

We learned that an access list,

A Wild and Crazy Trick to Extend Access

```
properties (Access = {?DList, ?LinkedNode})
```

We learned a wild and crazy
trick that takes advantage of

Protected Attribute

```
properties (Access = protected)
```

We learned about the
protected access attribute,

Handle-Class Destructors

```
function delete(list)
    while ~isempty(list.Head)
        list.Head.delete();
    end
end
```

We learned about
handle-class destructors,

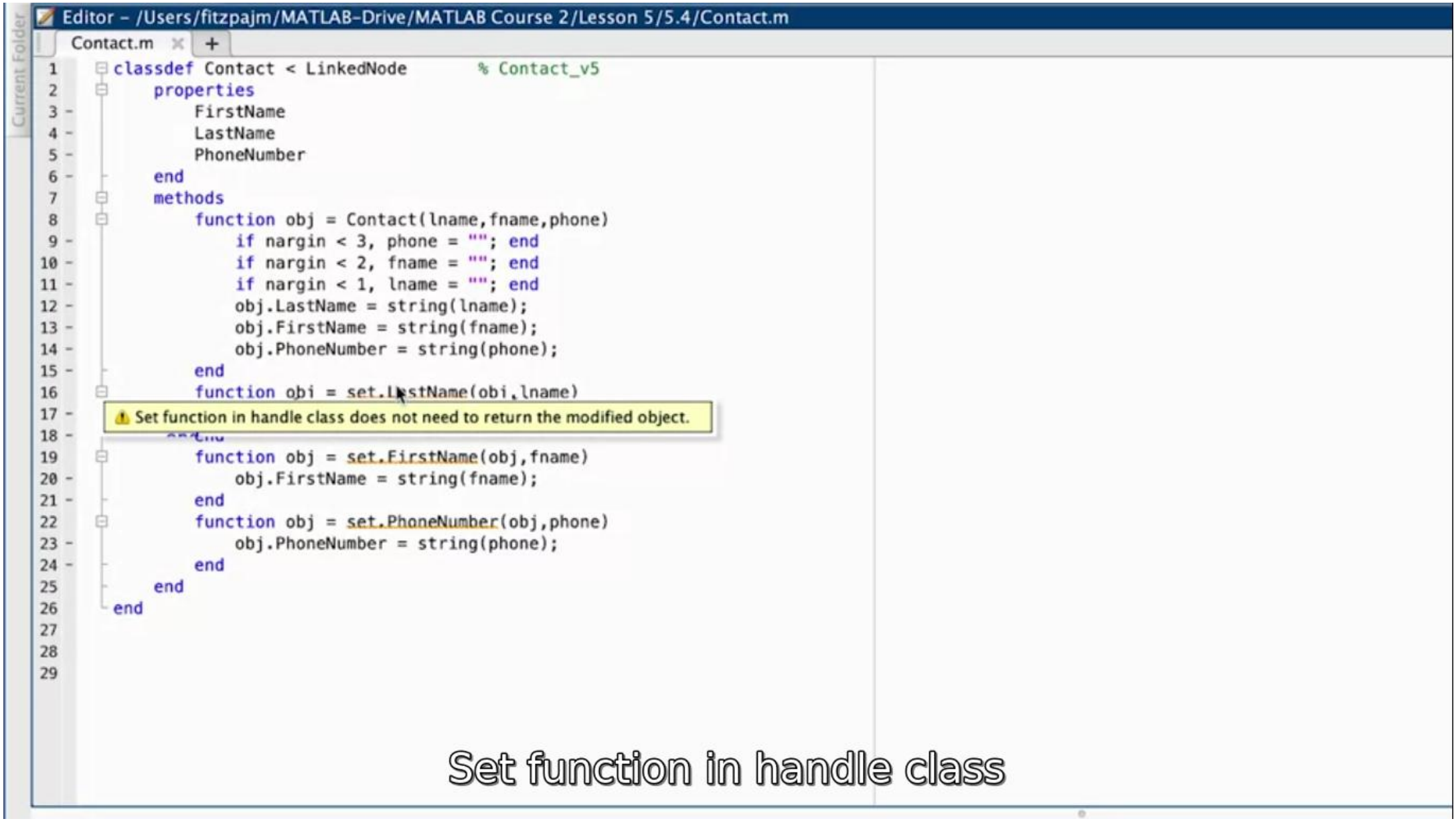
Handle-Class Destructors

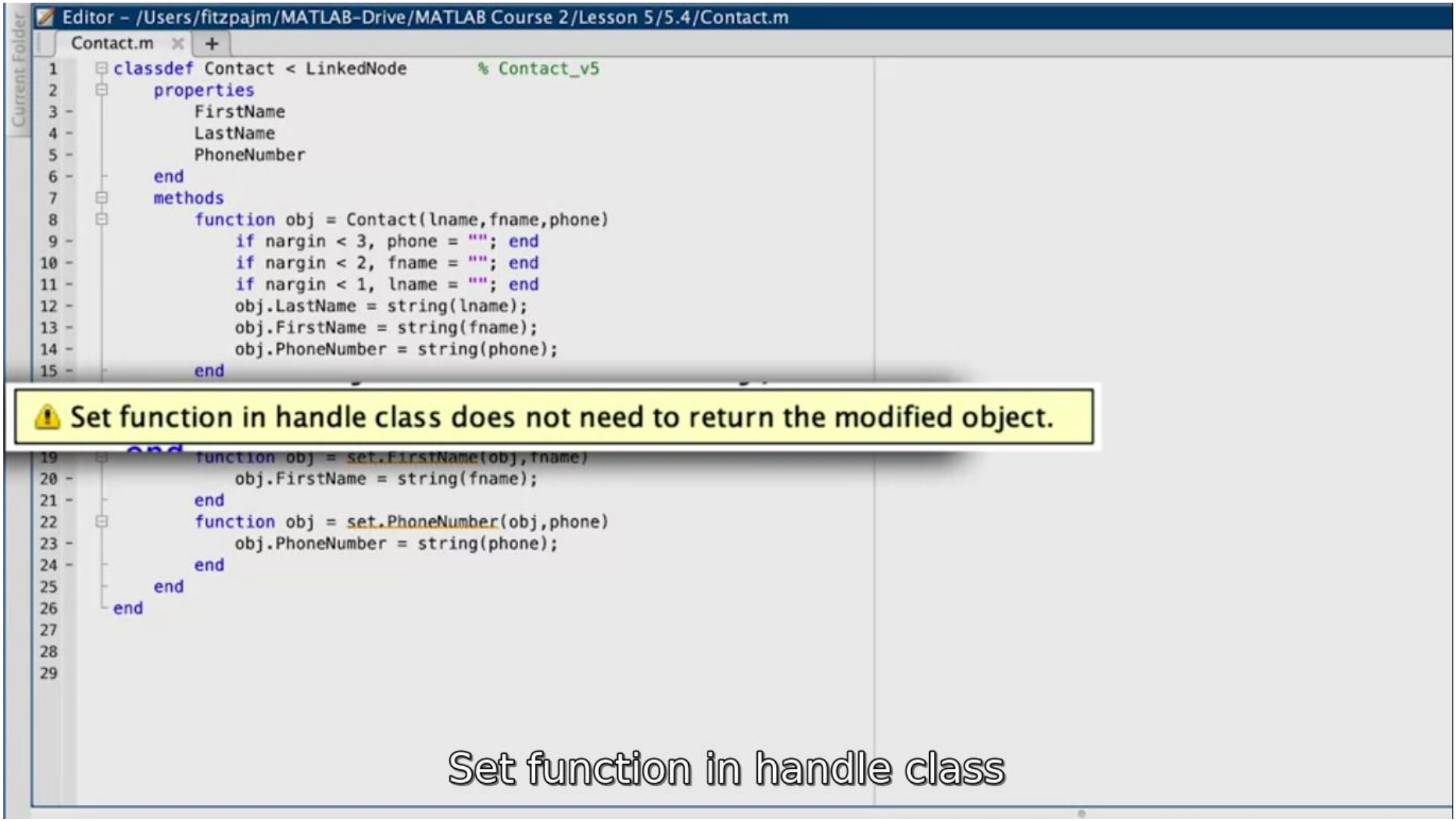
Memory Leaks

Its constant recycling avoids
so-called memory leaks,

Handle-Class Destructors

occupied by inaccessible objects.


Editor - /Users/fitzpajm/MATLAB-Drive/MATLAB Course 2/Lesson 5/5.4/Contact.m
Contact.m
Current Folder
1 classdef Contact < LinkedNode % Contact_v5
2 properties
3 FirstName
4 LastName
5 PhoneNumber
6 end
7 methods
8 function obj = Contact(lname,fname,phone)
9 if nargin < 3, phone = ""; end
10 if nargin < 2, fname = ""; end
11 if nargin < 1, lname = ""; end
12 obj.LastName = string(lname);
13 obj.FirstName = string(fname);
14 obj.PhoneNumber = string(phone);
15 end
16 function obj = set.LastName(obj,lname)
17 Set function in handle class does not need to return the modified object.
18 end
19 function obj = set.FirstName(obj,fname)
20 obj.FirstName = string(fname);
21 end
22 function obj = set.PhoneNumber(obj,phone)
23 obj.PhoneNumber = string(phone);
24 end
25 end
26 end
27
28
29
Set function in handle class

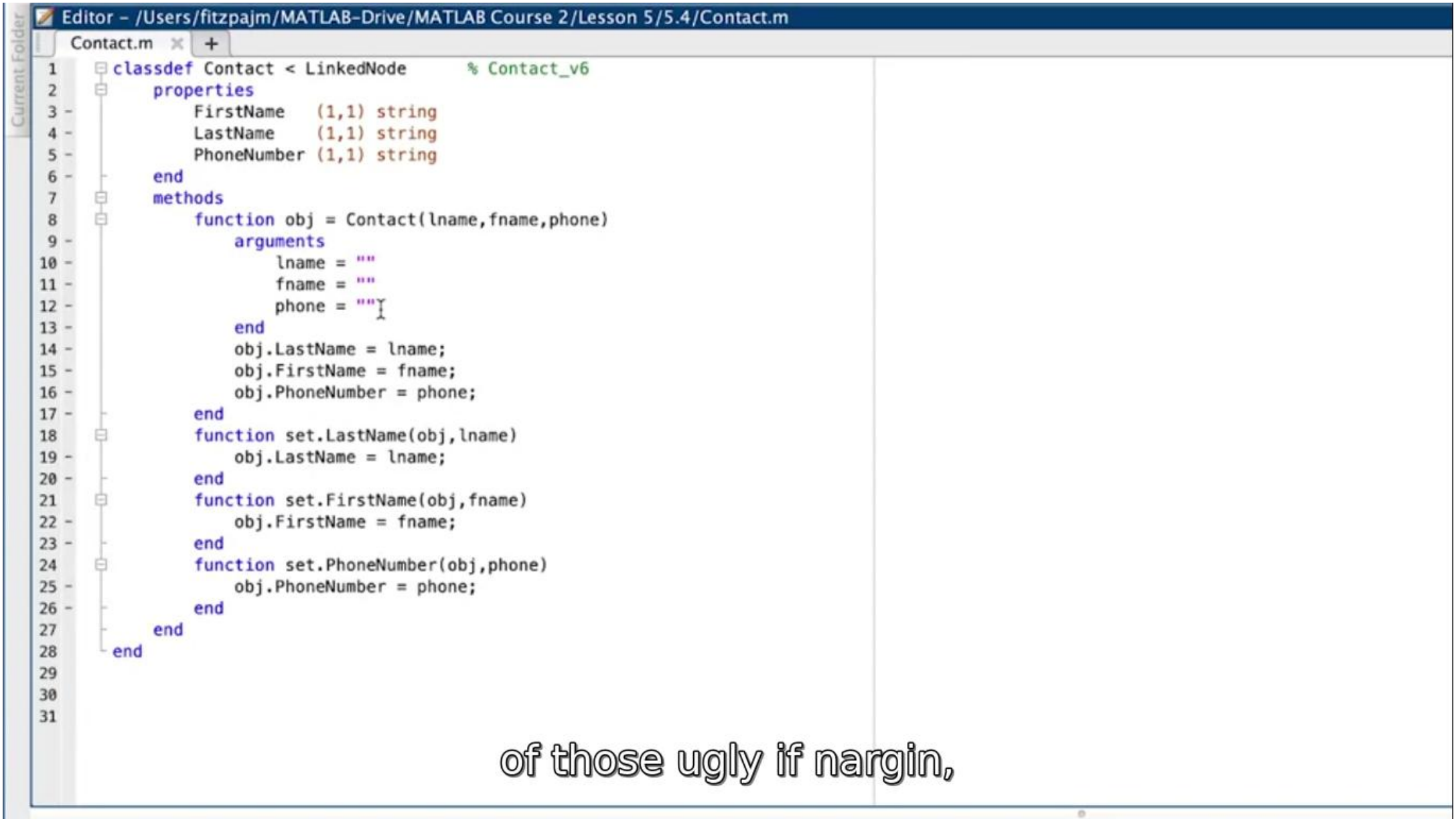

Editor - /Users/fitzpajm/MATLAB-Drive/MATLAB Course 2/Lesson 5/5.4/Contact.m
Contact.m
Current Folder
1 classdef Contact < LinkedNode % Contact_v5
2 properties
3 FirstName
4 LastName
5 PhoneNumber
6 end
7 methods
8 function obj = Contact(lname,fname,phone)
9 if nargin < 3, phone = ""; end
10 if nargin < 2, fname = ""; end
11 if nargin < 1, lname = ""; end
12 obj.LastName = string(lname);
13 obj.FirstName = string(fname);
14 obj.PhoneNumber = string(phone);
15 end
Set function in handle class does not need to return the modified object.
19 function obj = set.FirstName(obj,fname)
20 obj.FirstName = string(fname);
21 end
22 function obj = set.PhoneNumber(obj,phone)
23 obj.PhoneNumber = string(phone);
24 end
25 end
26 end
27
28
29
Set function in handle class

Editor - /Users/fitzpajm/MATLAB-Drive/MATLAB Course 2/Lesson 5/5.4/Contact.m*

Contact.m*

```
classdef Contact < LinkedNode       % Contact_v5
    properties
        FirstName
        LastName
        PhoneNumber
    end
    methods
        function obj = Contact(lname,fname,phone)
            if nargin < 3, phone = ""; end
            if nargin < 2, fname = ""; end
            if nargin < 1, lname = ""; end
            obj.LastName = string(lname);
            obj.FirstName = string(fname);
            obj.PhoneNumber = string(phone);
        end
        function  set.LastName(obj,lname)
            obj.LastName = string(lname);
        end
        function set.FirstName(obj,fname)
            obj.FirstName = string(fname);
        end
        function set.PhoneNumber(obj,phone)
            obj.PhoneNumber = string(phone);
        end
    end
end
```

Other than that one
optional detail,



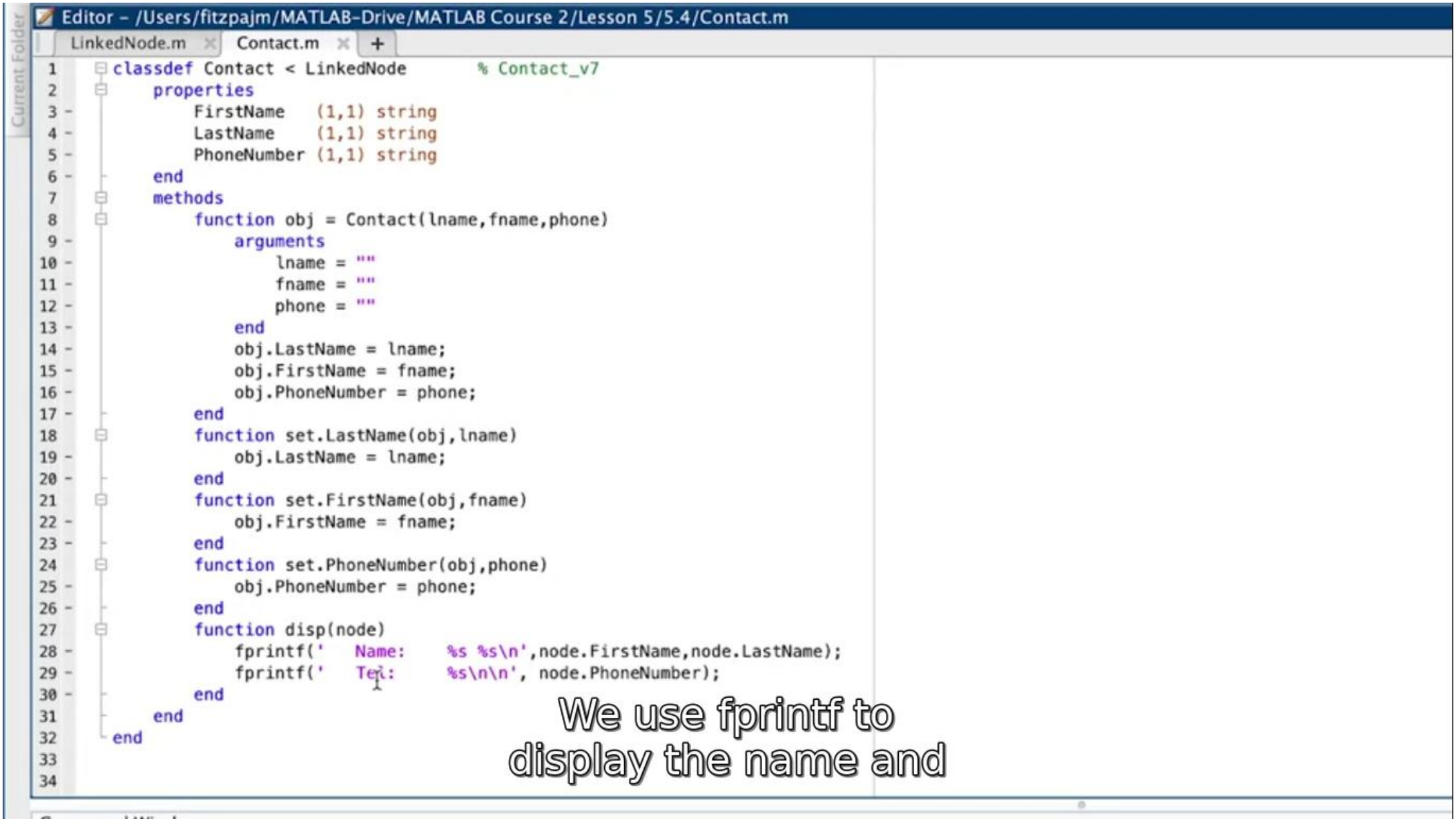
of those ugly if nargin,

Editor - /Users/fitzpajm/MATLAB-Drive/MATLAB Course 2/Lesson 5/5.4/LinkedNode.m

LinkedNode.m | Contact.m | +

```
1  classdef LinkedNode < handle       % LinkedNode_v8
2      properties (Access = {?DList ?LinkedNode})
3          Prev
4          Next
5          Owner
6      end
7      methods (Abstract)
8          gt(a,b)
9          ge(a,b)
10         lt(a,b)
11         le(a,b)
12         eq(a,b)
13         ne(a,b)
14         disp(a)
15     end
16     methods
17         function node = LinkedNode()
18             node.Prev = [];
19             node.Next = [];
20             node.Owner = [];
21         end
22         function delete(node)
23             if ~isempty(node.Owner)
24                 node.Owner.remove(node);
25             end
26         end
27     end
28 end
29
```

Remember those relational methods and the disk method?


Editor - /Users/fitzpajm/MATLAB-Drive/MATLAB Course 2/Lesson 5/5.4/Contact.m
LinkedNode.m
Contact.m
Current Folder
1 classdef Contact < LinkedNode % Contact_v7
2 properties
3 FirstName (1,1) string
4 LastName (1,1) string
5 PhoneNumber (1,1) string
6 end
7 methods
8 function obj = Contact(lname,fname,phone)
9 arguments
10 lname = ""
11 fname = ""
12 phone = ""
13 end
14 obj.LastName = lname;
15 obj.FirstName = fname;
16 obj.PhoneNumber = phone;
17 end
18 function set.LastName(obj,lname)
19 obj.LastName = lname;
20 end
21 function set.FirstName(obj,fname)
22 obj.FirstName = fname;
23 end
24 function set.PhoneNumber(obj,phone)
25 obj.PhoneNumber = phone;
26 end
27 function disp(node)
28 fprintf(' Name: %s %s\n',node.FirstName,node.LastName);
29 fprintf(' Tel: %s\n\n', node.PhoneNumber);
30 end
31 end
32 end
33
34
We use fprintf to
display the name and

Editor - /Users/fitzpajm/MATLAB-Drive/MATLAB Course 2/Lesson 5/5.4/LinkedNode.m

LinkedNode.m | Contact.m | +

```
1  classdef LinkedNode < handle       % LinkedNode_v8
2      properties (Access = {?DList ?LinkedNode})
3          Prev
4          Next
5          Owner
6      end
7      methods (Abstract)
8          gt(a,b)
9          ge(a,b)
10         lt(a,b)
11         le(a,b)
12         eq(a,b)
13         ne(a,b)
14         disp(a)
15     end
16     methods
17         function node = LinkedNode()
18             node.Prev = [];
19             node.Next = [];
20             node.Owner = [];
21         end
22         function delete(node)
23             if ~isempty(node.Owner)
24                 node.Owner.remove(node);
25             end
26         end
27     end
28 end
29
```

As we learned in the previous lecture,

Editor - /Users/fitzpajm/MATLAB-Drive/MATLAB Course 2/Lesson 5/5.4/BusinessContact.m

LinkedNode.m | Contact.m | BusinessContact.m | live_script_5_4_1.mlx | +

```
1   classdef BusinessContact < Contact       % BusinessContact_v5
2       properties
3           Company
4           Fax
5       end
6       methods
7           function obj = BusinessContact(cname,lname,fname,phone,f)
8               if nargin < 5 f = ""; end
9               if nargin < 4 phone = ""; end
10              if nargin < 3 fname = ""; end
11              if nargin < 2 lname = ""; end
12              if nargin < 1 cname = ""; end
13              obj@Contact(lname,fname,phone);
14              obj.Company = string(cname);
15              obj.Fax = string(f);
16          end
17          function obj = set.Company(obj,cname)
18              obj.Company = string(cname);
19          end
20          function obj = set.Fax(obj,f)
21              obj.Fax = string(f);
22          end
23      end
24  end
```

for these set access methods.

Editor - /Users/fitzpajm/MATLAB-Drive/MATLAB Course 2/Lesson 5/5.4/BusinessContact.m

LinkedNode.m | Contact.m | BusinessContact.m | live_script_5_4_1.mlx | +

```
classdef BusinessContact < Contact       % BusinessContact_v6
    properties
        Company (1,1) string
        Fax     (1,1) string
    end
    methods
        function obj = BusinessContact(cname,lname,fname,phone,f)
            arguments
                cname = ""
                lname = ""
                fname = ""
                phone = ""
                    f = ""
            end
            obj@Contact(lname,fname,phone);
            obj.Company = cname;
            obj.Fax = f;
        end
        function set.Company(obj,cname)
            obj.Company = cname;
        end
        function set.Fax(obj,f)
            obj.Fax = f;
        end
    end
end
```

Here we go, night and day.

OOP High-Level Concepts

Encapsulation

- Both the data and the methods that implement the functionality of an object are kept inside the object.
- The implementation, ie, how it works, is distinguished from the interface, ie, how it's used.

Information hiding

Portions of the implementation of an object are accessible only to a subset of other objects.

includes all of contacts,

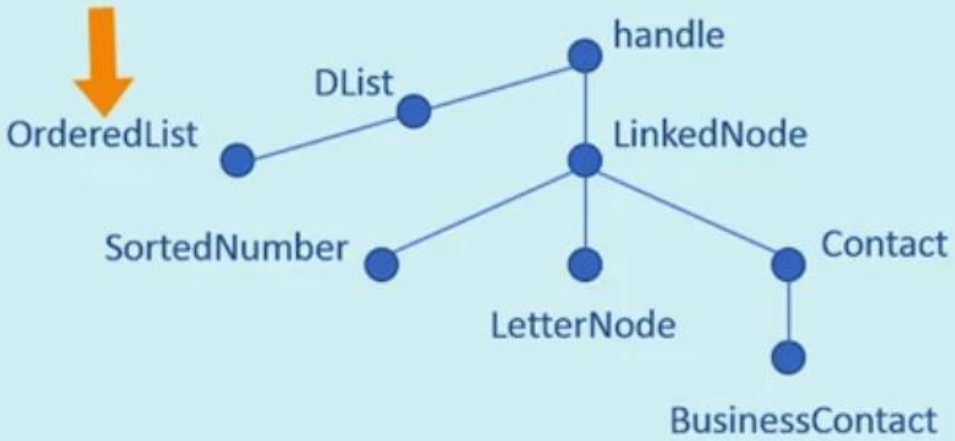
OOP High-Level Concepts

- Benefits of Encapsulation
  - The developer can maintain control over how an object can be used.
  - The developer can modify the implementation without affecting the use as long as the interface remains unchanged.
  - The user does not have to assume anything about how the class is implemented.

easier to debug, and
easier to maintain.

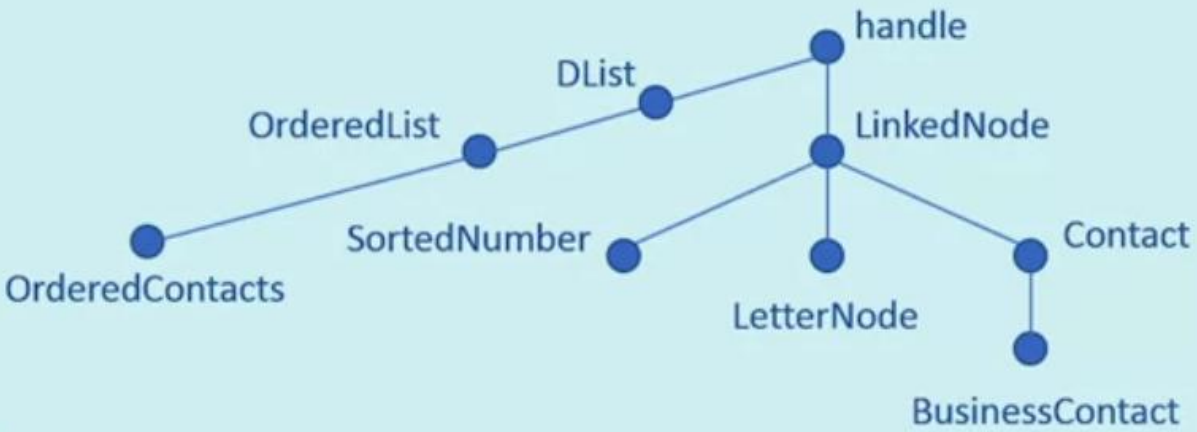
OOP High-Level Concepts

- Inheritance
  - Code reuse
  - Specialization
  - “is a” relationship



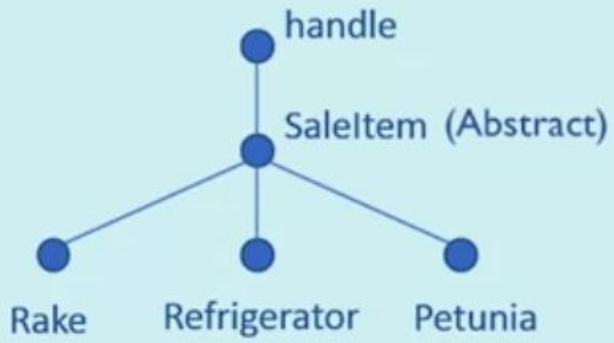
OOP High-Level Concepts

- Inheritance
  - Code reuse
  - Specialization
  - “is a” relationship



OOP High-Level Concepts

- Inheritance
  - Abstraction



OOP High-Level Concepts

- Polymorphism = variation in the behavior of a function of a given name, when the number or types of its actual arguments varies.
- The function is called “polymorphic”

OOP High-Level Concepts

- Most MATLAB built-in functions are polymorphic:

```
>> sqrt(4)
ans =
     2
```

OOP High-Level Concepts

- Most MATLAB built-in functions are polymorphic:

```
>> sqrt(4)
ans =
     2
>> sqrt([4,9,16,25])
```

OOP High-Level Concepts

```
>> help plot
 plot  Linear plot.
    plot(X,Y) plots vector Y versus vector X. If X or Y is a matrix,
    then the vector is plotted versus the rows or columns of the matrix,
    whichever line up. If X is a scalar and Y is a vector, disconnected
    line objects are created and plotted as discrete points vertically at
    X.

    plot(Y) plots the columns of Y versus their index.
    If Y is complex, plot(Y) is equivalent to plot(real(Y),imag(Y)).
    In all other uses of plot, the imaginary part is ignored.

    . . .
```

Writing a polymorphic function

```
function out = meld(in1,in2)
    % MELD(X,Y) combine inputs
    if nargin == 1
        out = in1;
    elseif isa(in1,"char") && isa(in2,"char")
        out = append(in1,in2);
    else
        if ~isa(in1,"numeric")
            in1 = string(in1);
        end
        if ~isnumeric(in2) % relative of isa()
            in2 = string(in2);
        end
        out = in1 + in2;
    end
end
```

OOP High-Level Concepts

- Polymorphism
  - Method overriding: subclasses inherit public and protected methods of their superclasses, but they can provide their own implementation specialized to their needs to override those methods.
  - Operator overloading: classes can provide their own implementations of built-in MATLAB operators

MATLAB's Secret

It's all Object Oriented!

Live Editor - /Users/fitzpajm/MATLAB-Drive/MATLAB Course 2/Lesson 5/5.4/live_script_5_4_4.mlx *

Current Folder

Contact.m | BusinessContact.m | DList.m | OrderedList.m | LinkedNode.m | live_script_5_4_4.mlx * | +

```
1  clear
2  x = -1.2
```

```
x = -1.2000
```

```
3  y = uint8(3)
```

```
y = uint8
    3
```

```
4  name = "Mike"
```

```
name = "Mike"
```

```
5  Hermione = Contact
```

```
Hermione =
   Name:
   Tel:
```

```
6  whos
```

```
  Name          Size            Bytes  Class     Attributes

  Hermione      1x1                 8  Contact
  name          1x1               148  string
  x             1x1                 8  double
  y             1x1                 1  uint8
```

Live Editor - /Users/fitzpajm/MATLAB-Drive/MATLAB Course 2/Lesson 5/5.4/live_script_5_4_4.mlx *

Contact.m | BusinessContact.m | DList.m | OrderedList.m | LinkedNode.m | live_script_5_4_4.mlx *

```
name          1x1          148  string
x             1x1            8  double
y             1x1            1  uint8
```

```
7  double_methods = methods("double")
```

double_methods = 210×1 cell

	1
18	loading...
19	loading...
20	loading...
21	loading...
22	loading...
23	loading...
24	loading...
25	loading...

```
8  string_methods = methods("string")
```

```
9  "Mike" > "Akos"
```

```
10 name
11 upper(name)
```

Live Editor - /Users/fitzpajm/MATLAB-Drive/MATLAB Course 2/Lesson 5/5.4/live_script_5_4_4.mlx *

Current Folder

Contact.m | BusinessContact.m | DList.m | OrderedList.m | LinkedNode.m | live_script_5_4_4.mlx * | +

```
7  double_methods = methods("double")
```

```
double_methods = 210×1 cell
    'tand'
    'tanh'
    'times'
    'transpose'
    'tril'
    'triu'
    'uminus'
    'uplus'
    'vecnorm'
    'xor'
```

```
8  string_methods = methods("string")
```

```
string_methods = 38×1 cell
    'append'
    'cellstr'
    'char'
    'compose'
    'contains'
    'count'
    'double'
    'endsWith'
    'eq'
    'erase'
        :
        :
```

/ ▸ Users ▸ fitzpajm ▸ MATLAB-Drive ▸ MATLAB Course 2 ▸ Lesson 5 ▸ 5.4

Live Editor - /Users/fitzpajm/MATLAB-Drive/MATLAB Course 2/Lesson 5/5.4/live_script_5_4_4.mlx *

Current Folder

Contact.m | BusinessContact.m | DList.m | OrderedList.m | LinkedNode.m | live_script_5_4_4.mlx * | +

```
8  string_methods = methods("string")

   string_methods = 38×1 cell
      'eraseBetw…
      'extractAf…
      'extractBe…
      'extractBe…
      'ge'
      'gt'
      'insertAft…
      'insertBef…
      'ismissing'
      'issorted'

9  "Mike" > "Akos"

   ans = logical
      1

10 name

   name = "Mike"

11 upper(name)

   ans = "MIKE"
```


/ ▸ Users ▸ fitzpajm ▸ MATLAB-Drive ▸ MATLAB Course 2 ▸ Lesson 5 ▸ 5.4
Live Editor - /Users/fitzpajm/MATLAB-Drive/MATLAB Course 2/Lesson 5/5.4/live_script_5_4_4.mlx *
Current Folder
Contact.m
BusinessContact.m
DList.m
OrderedList.m
LinkedNode.m
live_script_5_4_4.mlx *
name = "Mike"
11
upper(name)
ans = "MIKE"
12
name.upper
ans = "MIKE"
13
name
name = "Mike"
14
abs(x)
ans = 1.2000
15
x.abs
Dot indexing is not supported for variables of this type.

Live Editor - /Users/fitzpajm/MATLAB-Drive/MATLAB Course 2/Lesson 5/5.4/live_script_5_4_4.mlx *

Current Folder

Contact.m | BusinessContact.m | DList.m | OrderedList.m | LinkedNode.m | live_script_5_4_4.mlx * | +

```
15  x.abs
```

```
Dot indexing is not supported for variables of this type.
```

```
16  upper(name)
```

```
ans = "MIKE"
```

```
17  properties("BusinessContact")
```

```
Properties for class BusinessContact:
    Company
    Fax
    FirstName
    LastName
    PhoneNumber
```

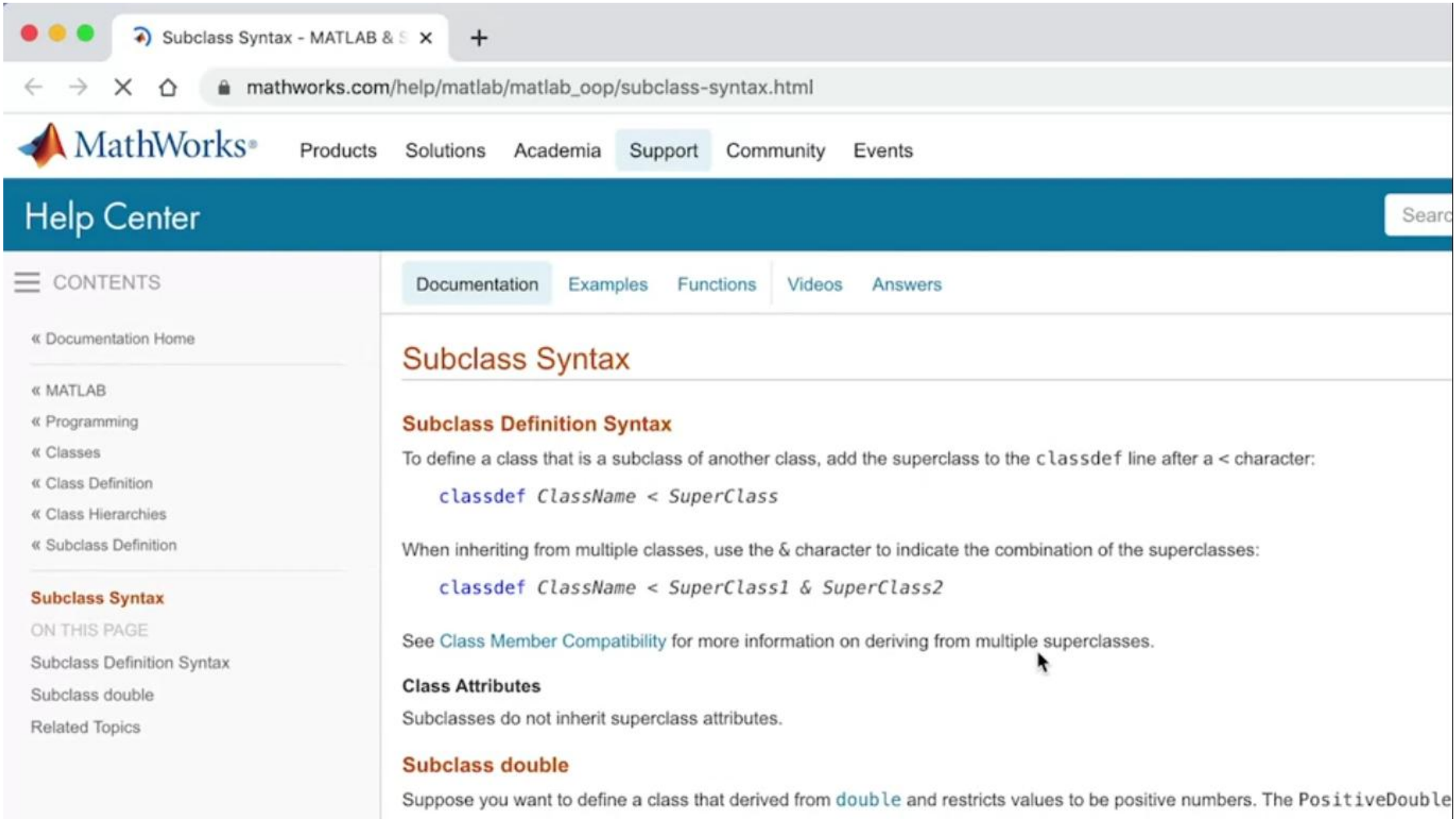
```
18  properties("string")
```

```
No properties for class string.
```

```
19  properties("double")
```

```
No properties for class double.
```

```
16  upper(name)

    ans = "MIKE"

17  properties("BusinessContact")

    Properties for class BusinessContact:
        Company
        Fax
        FirstName
        LastName
        PhoneNumber

18  properties("string")

    No properties for class string.

19  properties("double")

    No properties for class double.

20  properties("LinkedNode")

    No properties for class LinkedNode.

21  web('https://www.mathworks.com/help/matlab/matlab_oop/subclass-syntax.html')
```


Subclass Syntax - MATLAB &
mathworks.com/help/matlab/matlab_oop/subclass-syntax.html
MathWorks
Products
Solutions
Academia
Support
Community
Events
Help Center
CONTENTS
« Documentation Home
« MATLAB
« Programming
« Classes
« Class Definition
« Class Hierarchies
« Subclass Definition
Subclass Syntax
ON THIS PAGE
Subclass Definition Syntax
Subclass double
Related Topics
Documentation
Examples
Functions
Videos
Answers
Subclass Syntax
Subclass Definition Syntax
To define a class that is a subclass of another class, add the superclass to the classdef line after a < character:
classdef ClassName < SuperClass
When inheriting from multiple classes, use the & character to indicate the combination of the superclasses:
classdef ClassName < SuperClass1 & SuperClass2
See Class Member Compatibility for more information on deriving from multiple superclasses.
Class Attributes
Subclasses do not inherit superclass attributes.
Subclass double
Suppose you want to define a class that derived from double and restricts values to be positive numbers. The PositiveDouble

Subclass Syntax - MATLAB &

43:20 / 46:04

mathworks.com/help/matlab/matlab_oop/subclass-syntax.html

Help Center

Search

CONTENTS

« Documentation Home

« MATLAB
« Programming
« Classes
« Class Definition
« Class Hierarchies
« Subclass Definition

Subclass Syntax

ON THIS PAGE

Subclass Definition Syntax
Subclass double
Related Topics

Documentation | Examples | Functions | Videos | Answers

Subclass double

Suppose you want to define a class that derived from `double` and restricts values to be positive numbers. The `PositiveDouble`

- Supports a default constructor (no input arguments). See No Input Argument Constructor Requirement
- Restricts the inputs to positive values using `mustBePositive`.
- Calls the superclass constructor with the input value to create the double numeric value.

```
classdef PositiveDouble < double
   methods
      function obj = PositiveDouble(data)
         if nargin == 0
            data = 1;
         else
            mustBePositive(data)
         end
         obj = obj@double(data);
      end
   end
end
```

Create an object of the `PositiveDouble` class using a 1-by-5 array of numbers:

OOP High-Level Concepts

- Encapsulation
- Inheritance
- Polymorphism