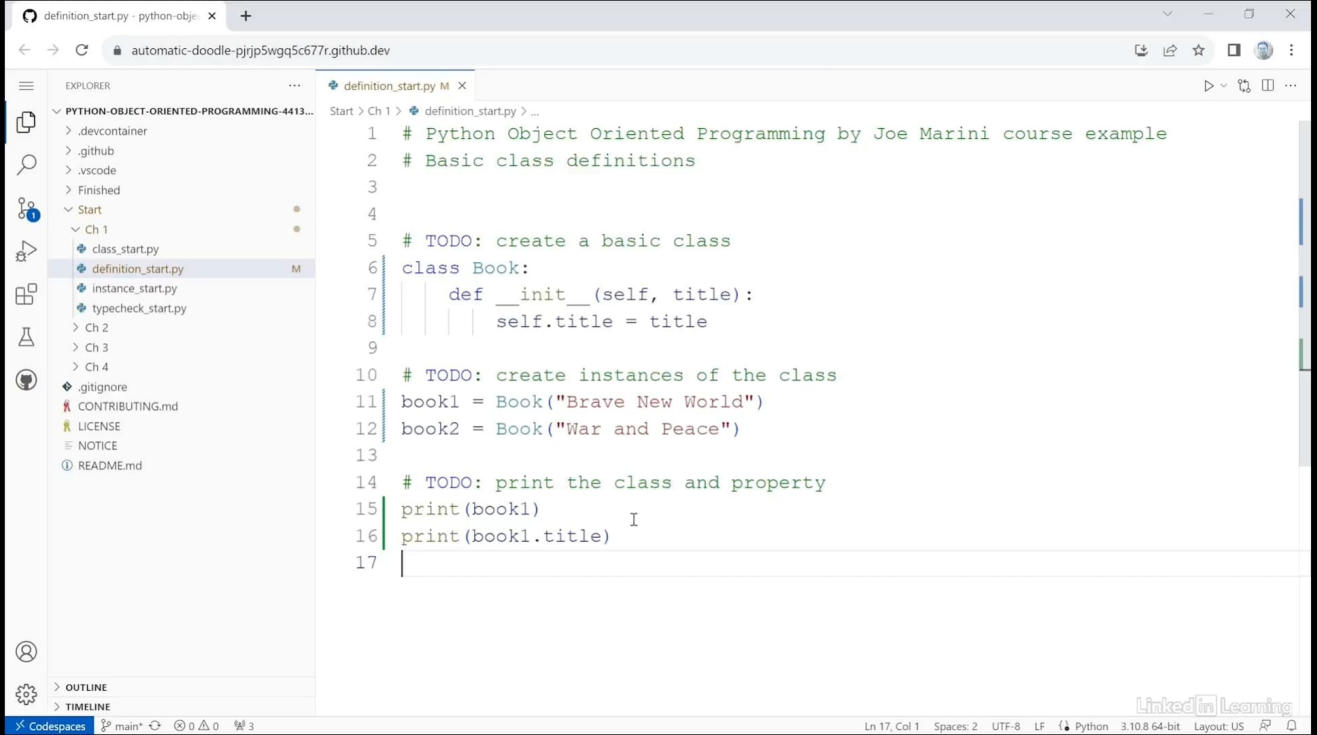
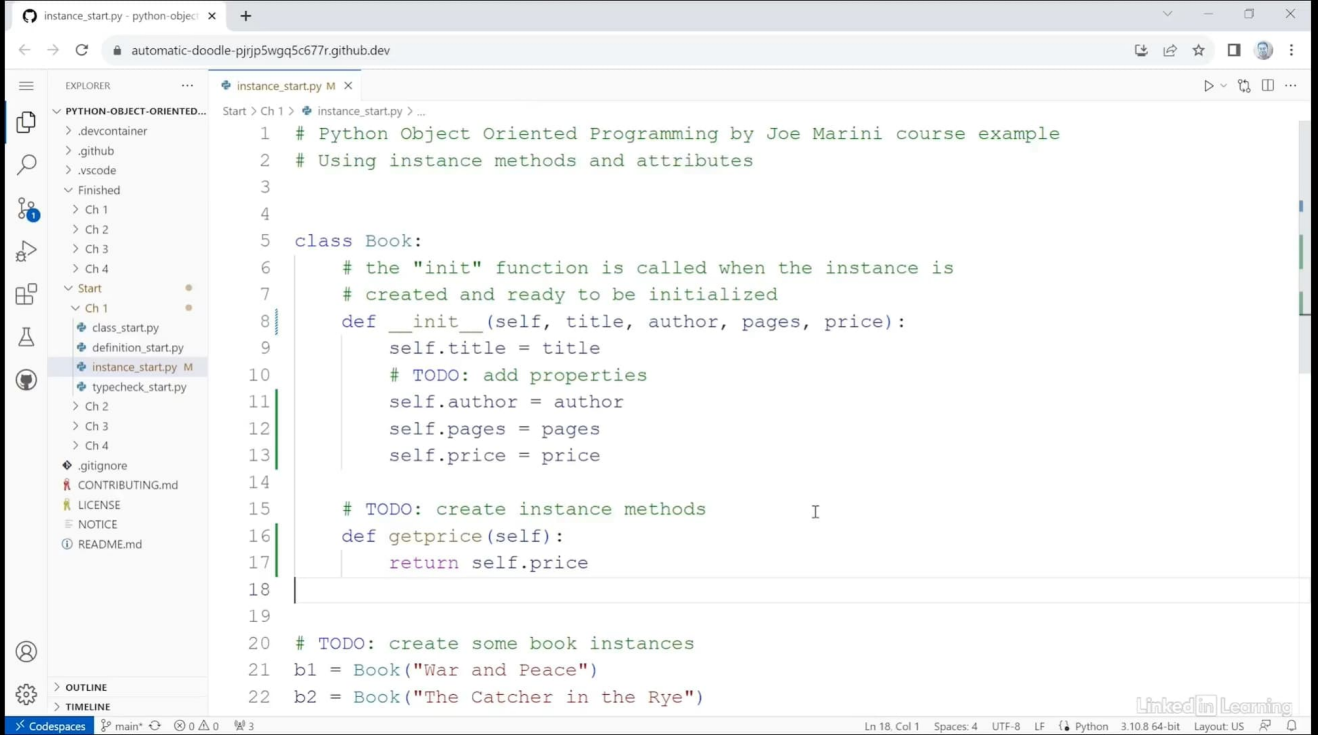
Object-Oriented Programming Refresher

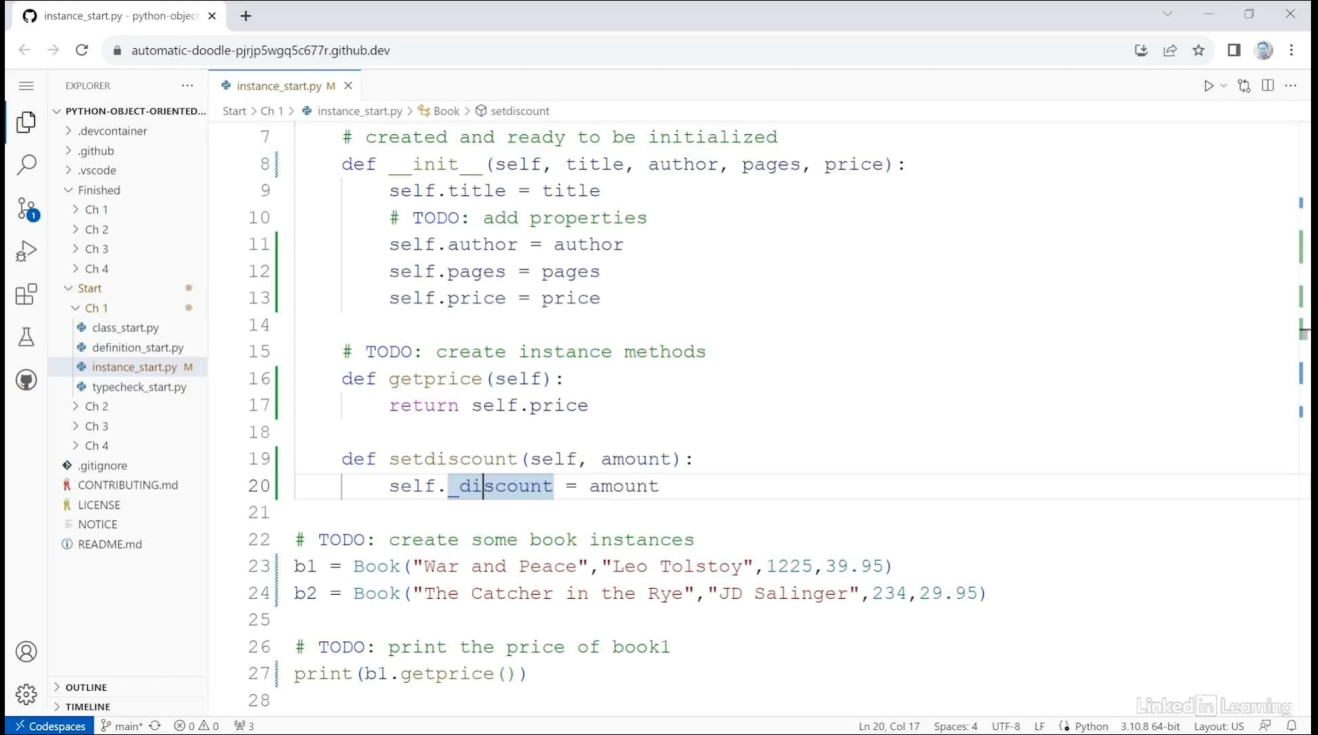
+ **Python does not require you to use objects or classes**
+ **Complex programs are hard to keep organized**
+ **Object-oriented programming organizes and structures code**
  - Groups together data and behavior into one place
  - Promotes modularization of programs
  - Isolates different parts of the program from each other

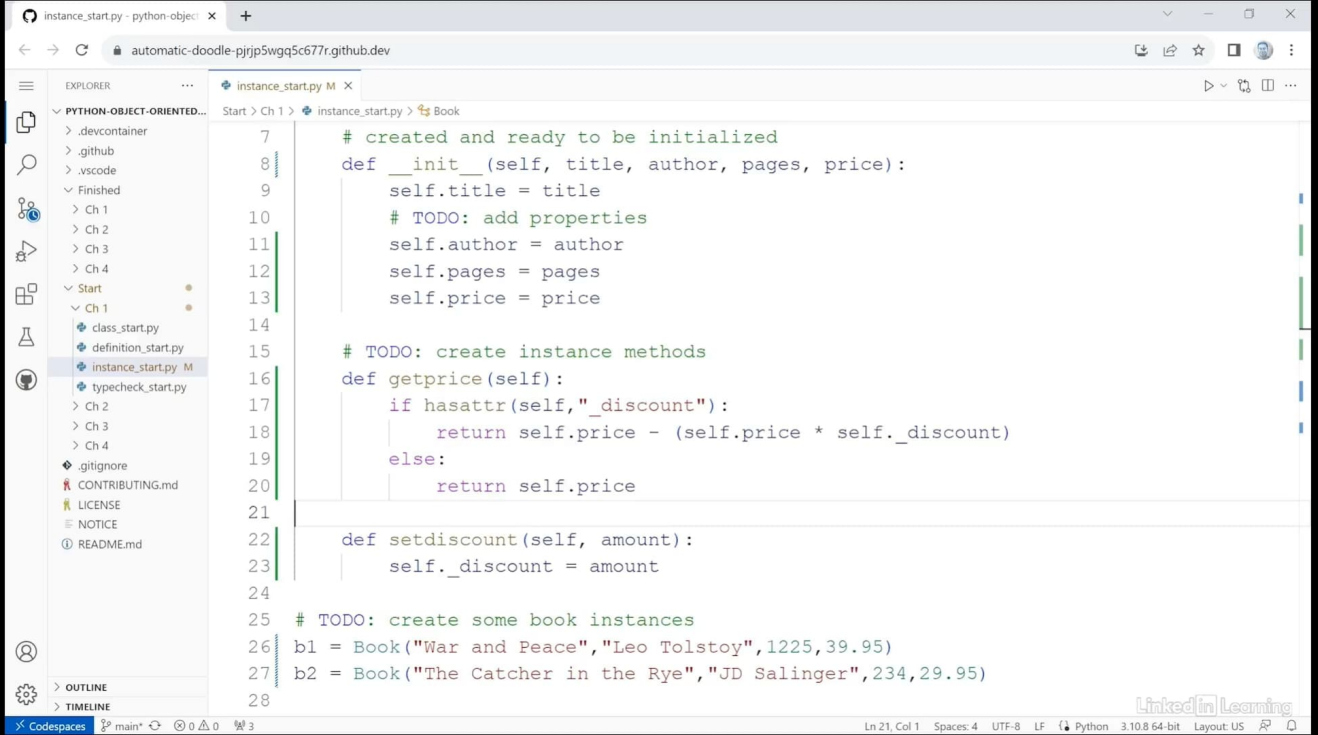
Object-Oriented Programming Terms

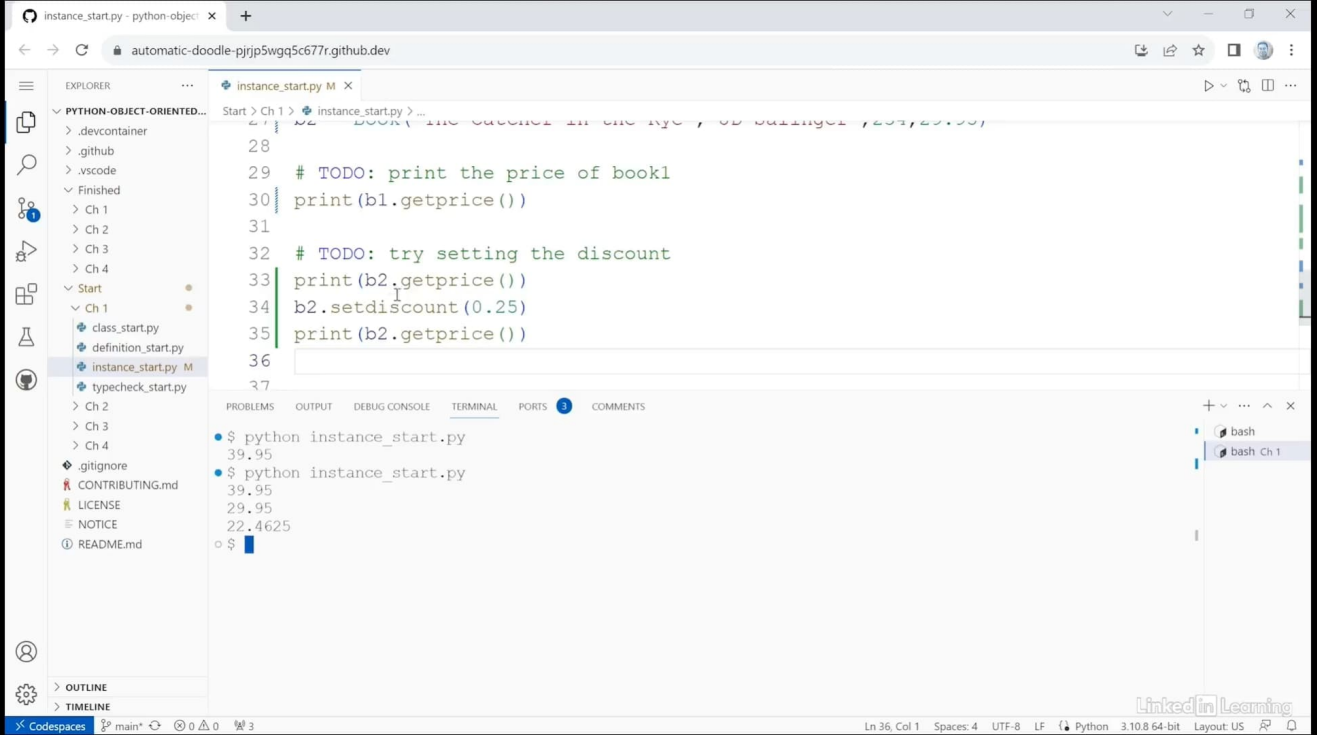
Class	A blueprint for creating objects of a particular type
Methods	Regular functions that are part of a class
Attributes	Variables that hold data that are part of a class
Object	A specific instance of a class
Inheritance	Means by which a class can inherit capabilities from another
Composition	Means of building complex objects out of other objects

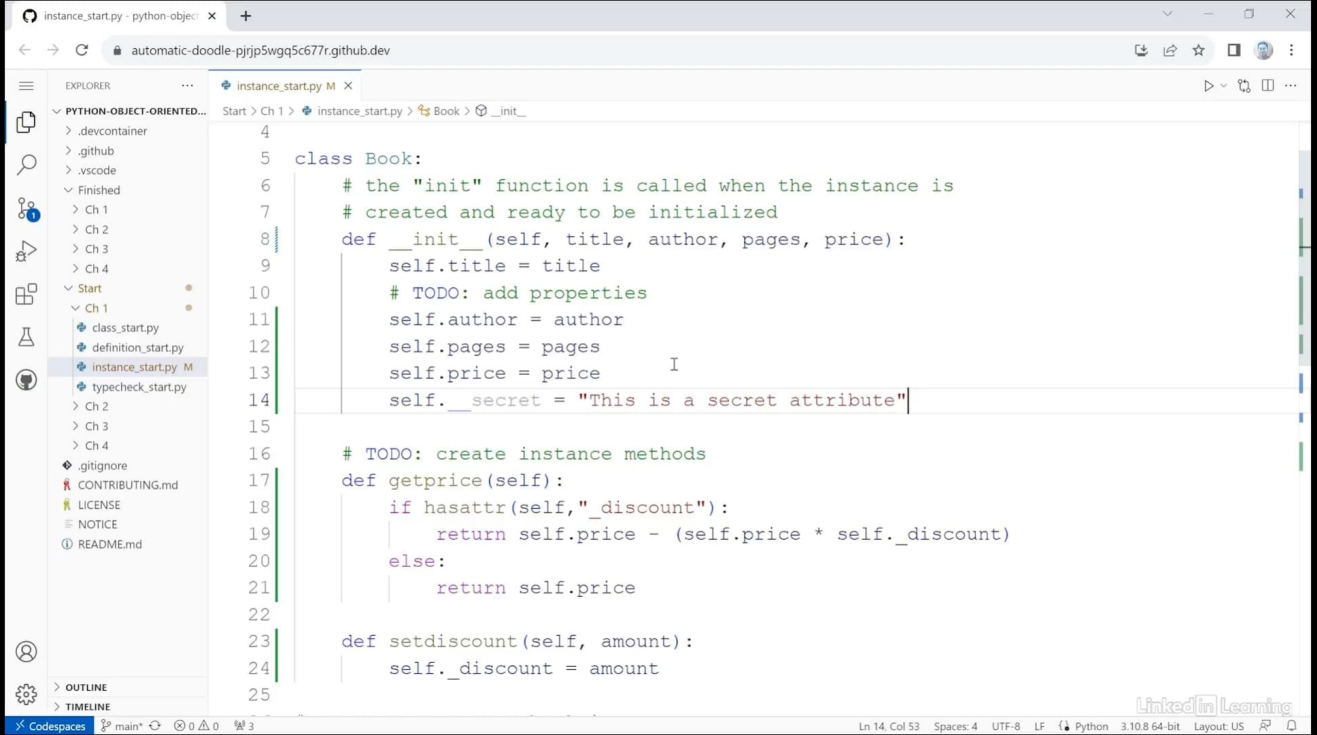

definition_start.py - python-obje
automatic-doodle-pjrjp5wgq5c677r.github.dev
EXPLORER
definition_start.py M
PYTHON-OBJECT-ORIENTED-PROGRAMMING-4413...
.devcontainer
.github
.vscode
Finished
Start
Ch 1
class_start.py
definition_start.py M
instance_start.py
typecheck_start.py
Ch 2
Ch 3
Ch 4
.gitignore
CONTRIBUTING.md
LICENSE
NOTICE
README.md
Start > Ch 1 > definition_start.py > ...
1 # Python Object Oriented Programming by Joe Marini course example
2 # Basic class definitions
3
4
5 # TODO: create a basic class
6 class Book:
7 def __init__(self, title):
8 self.title = title
9
10 # TODO: create instances of the class
11 book1 = Book("Brave New World")
12 book2 = Book("War and Peace")
13
14 # TODO: print the class and property
15 print(book1)
16 print(book1.title)
17
OUTLINE
TIMELINE
Codespaces
main*
Ln 17, Col 1
Spaces: 2
UTF-8
LF
Python
3.10.8 64-bit
Layout: US
LinkedIn Learning

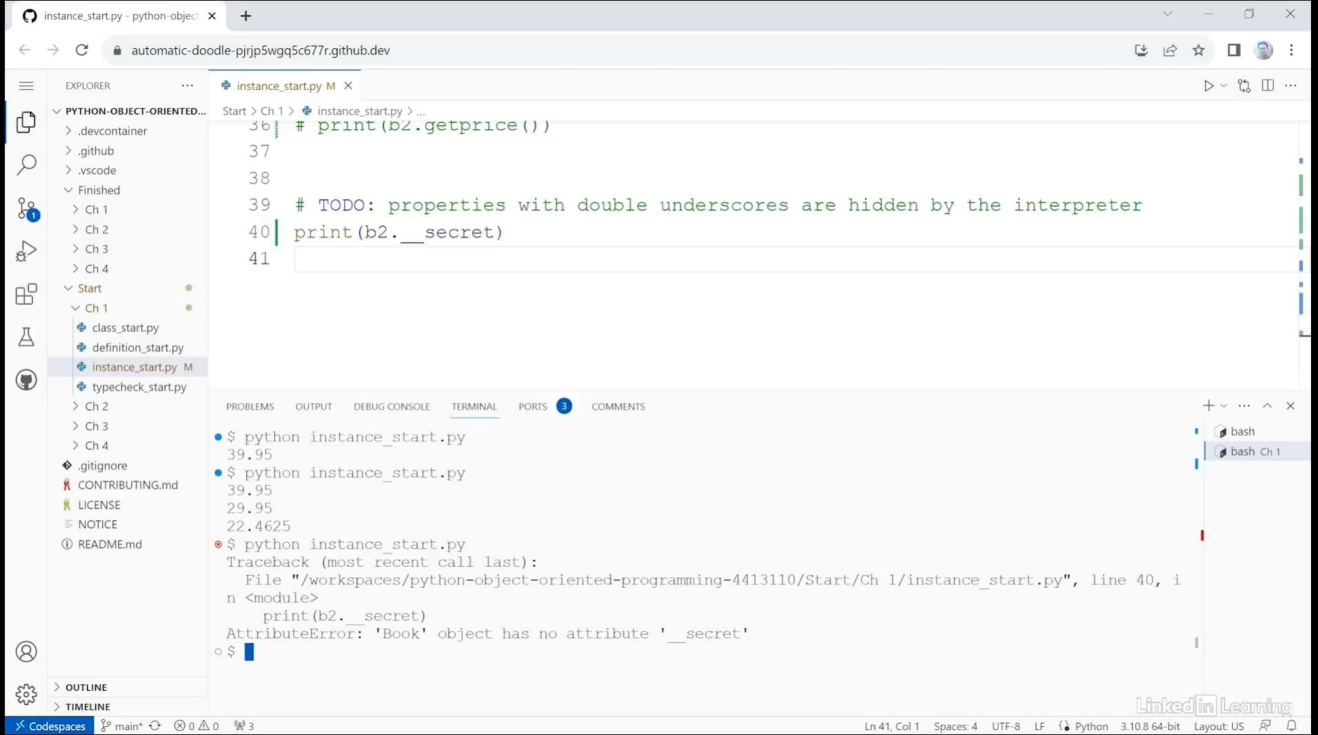

instance_start.py - python-object
automatic-doodle-pjrjp5wgq5c677r.github.dev
EXPLORER
PYTHON-OBJECT-ORIENTED...
.devcontainer
.github
.vscode
Finished
Ch 1
Ch 2
Ch 3
Ch 4
Start
Ch 1
class_start.py
definition_start.py
instance_start.py M
typecheck_start.py
Ch 2
Ch 3
Ch 4
.gitignore
CONTRIBUTING.md
LICENSE
NOTICE
README.md
OUTLINE
TIMELINE
instance_start.py M
Start > Ch 1 > instance_start.py > ...
1 # Python Object Oriented Programming by Joe Marini course example
2 # Using instance methods and attributes
3
4
5 class Book:
6 # the "init" function is called when the instance is
7 # created and ready to be initialized
8 def __init__(self, title, author, pages, price):
9 self.title = title
10 # TODO: add properties
11 self.author = author
12 self.pages = pages
13 self.price = price
14
15 # TODO: create instance methods
16 def getprice(self):
17 return self.price
18
19
20 # TODO: create some book instances
21 b1 = Book("War and Peace")
22 b2 = Book("The Catcher in the Rye")
Codespaces
main*
Ln 18, Col 1
Spaces: 4
UTF-8
LF
Python
3.10.8 64-bit
Layout: US
LinkedIn Learning


instance_start.py - python-object
automatic-doodle-pjrjp5wgq5c677r.github.dev
EXPLORER
instance_start.py M
PYTHON-OBJECT-ORIENTED...
.devcontainer
.github
.vscode
Finished
Ch 1
Ch 2
Ch 3
Ch 4
Start
Ch 1
class_start.py
definition_start.py
instance_start.py M
typecheck_start.py
Ch 2
Ch 3
Ch 4
.gitignore
CONTRIBUTING.md
LICENSE
NOTICE
README.md
OUTLINE
TIMELINE
Start > Ch 1 > instance_start.py > Book > setdiscount
7 # created and ready to be initialized
8 def __init__(self, title, author, pages, price):
9 self.title = title
10 # TODO: add properties
11 self.author = author
12 self.pages = pages
13 self.price = price
14
15 # TODO: create instance methods
16 def getprice(self):
17 return self.price
18
19 def setdiscount(self, amount):
20 self._discount = amount
21
22 # TODO: create some book instances
23 b1 = Book("War and Peace","Leo Tolstoy",1225,39.95)
24 b2 = Book("The Catcher in the Rye","JD Salinger",234,29.95)
25
26 # TODO: print the price of book1
27 print(b1.getprice())
28
Codespaces
main*
Ln 20, Col 17
Spaces: 4
UTF-8
LF
Python
3.10.8 64-bit
Layout: US
LinkedIn Learning


instance_start.py - python-object
automatic-doodle-pjrjp5wgq5c677r.github.dev
EXPLORER
instance_start.py M
Start > Ch 1 > instance_start.py > Book
PYTHON-OBJECT-ORIENTED...
.devcontainer
.github
.vscode
Finished
Ch 1
Ch 2
Ch 3
Ch 4
Start
Ch 1
class_start.py
definition_start.py
instance_start.py M
typecheck_start.py
Ch 2
Ch 3
Ch 4
.gitignore
CONTRIBUTING.md
LICENSE
NOTICE
README.md
OUTLINE
TIMELINE
# created and ready to be initialized
def __init__(self, title, author, pages, price):
self.title = title
# TODO: add properties
self.author = author
self.pages = pages
self.price = price
# TODO: create instance methods
def getprice(self):
if hasattr(self,"_discount"):
return self.price - (self.price * self._discount)
else:
return self.price
def setdiscount(self, amount):
self._discount = amount
# TODO: create some book instances
b1 = Book("War and Peace","Leo Tolstoy",1225,39.95)
b2 = Book("The Catcher in the Rye","JD Salinger",234,29.95)
Codespaces
main*
Ln 21, Col 1
Spaces: 4
UTF-8
LF
Python
3.10.8 64-bit
Layout: US
LinkedIn Learning


instance_start.py - python-object
automatic-doodle-pjrjp5wgq5c677r.github.dev
EXPLORER
instance_start.py M
PYTHON-OBJECT-ORIENTED...
.devcontainer
.github
.vscode
Finished
Ch 1
Ch 2
Ch 3
Ch 4
Start
Ch 1
class_start.py
definition_start.py
instance_start.py M
typecheck_start.py
Ch 2
Ch 3
Ch 4
.gitignore
CONTRIBUTING.md
LICENSE
NOTICE
README.md
Start > Ch 1 > instance_start.py > ...
28
29 # TODO: print the price of book1
30 print(b1.getprice())
31
32 # TODO: try setting the discount
33 print(b2.getprice())
34 b2.setdiscount(0.25)
35 print(b2.getprice())
36
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS 3
COMMENTS
$ python instance_start.py
39.95
$ python instance_start.py
39.95
29.95
22.4625
$
bash
bash Ch 1
OUTLINE
TIMELINE
Codespaces
main*
Ln 36, Col 1
Spaces: 4
UTF-8
LF
Python
3.10.8 64-bit
Layout: US
LinkedIn Learning


instance_start.py - python-object
automatic-doodle-pjrjp5wgq5c677r.github.dev
EXPLORER
instance_start.py M
PYTHON-OBJECT-ORIENTED...
.devcontainer
.github
.vscode
Finished
Ch 1
Ch 2
Ch 3
Ch 4
Start
Ch 1
class_start.py
definition_start.py
instance_start.py M
typecheck_start.py
Ch 2
Ch 3
Ch 4
.gitignore
CONTRIBUTING.md
LICENSE
NOTICE
README.md
OUTLINE
TIMELINE
Start > Ch 1 > instance_start.py > Book > __init__
4
5 class Book:
6 # the "init" function is called when the instance is
7 # created and ready to be initialized
8 def __init__(self, title, author, pages, price):
9 self.title = title
10 # TODO: add properties
11 self.author = author
12 self.pages = pages
13 self.price = price
14 self.__secret = "This is a secret attribute"
15
16 # TODO: create instance methods
17 def getprice(self):
18 if hasattr(self,"_discount"):
19 return self.price - (self.price * self._discount)
20 else:
21 return self.price
22
23 def setdiscount(self, amount):
24 self._discount = amount
25
Codespaces
main*
Ln 14, Col 53
Spaces: 4
UTF-8
LF
Python
3.10.8 64-bit
Layout: US
LinkedIn Learning


instance_start.py - python-object
automatic-doodle-pjrjp5wgq5c677r.github.dev
EXPLORER
PYTHON-OBJECT-ORIENTED...
.devcontainer
.github
.vscode
Finished
Ch 1
Ch 2
Ch 3
Ch 4
Start
Ch 1
class_start.py
definition_start.py
instance_start.py M
typecheck_start.py
Ch 2
Ch 3
Ch 4
.gitignore
CONTRIBUTING.md
LICENSE
NOTICE
README.md
OUTLINE
TIMELINE
instance_start.py M
Start > Ch 1 > instance_start.py > ...
36 # print(b2.getprice())
37
38
39 # TODO: properties with double underscores are hidden by the interpreter
40 print(b2.__secret)
41
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS 3
COMMENTS
bash
bash Ch 1
$ python instance_start.py
39.95
$ python instance_start.py
39.95
29.95
22.4625
$ python instance_start.py
Traceback (most recent call last):
File "/workspaces/python-object-oriented-programming-4413110/Start/Ch 1/instance_start.py", line 40, in <module>
print(b2.__secret)
AttributeError: 'Book' object has no attribute '__secret'
$
Codespaces
main*
Ln 41, Col 1
Spaces: 4
UTF-8
LF
Python
3.10.8 64-bit
Layout: US

IP KT Interview

Case study instructions

This exercise is designed to be similar to the kinds of requests/projects you might receive in the IP Knowledge Team role (however, this exercise is on a slightly tighter timeframe)

You have a maximum of 60 minutes to review the materials and complete any answers. If you are unable to document your thoughts, you can elaborate the same in the follow-on interview discussion.

Your final output should be in either Excel, Word or PowerPoint – we recommend you use the format with which you are most familiar.

Your case study output should not be similar to any other outputs submitted by your peers. If the similarity was found in the output your candidature would not be considered for the further process of recruitment.

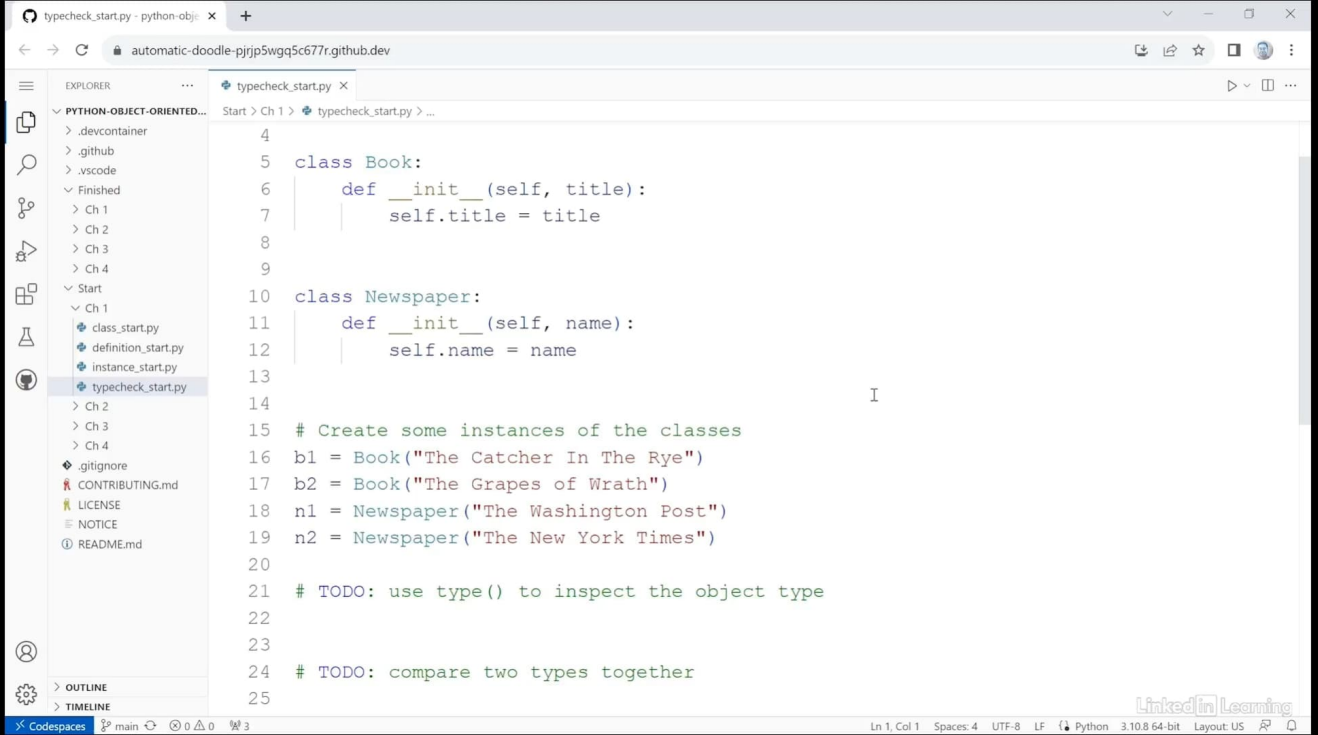
Your internet connection and video should be working pretty well during the follow-on discussion.

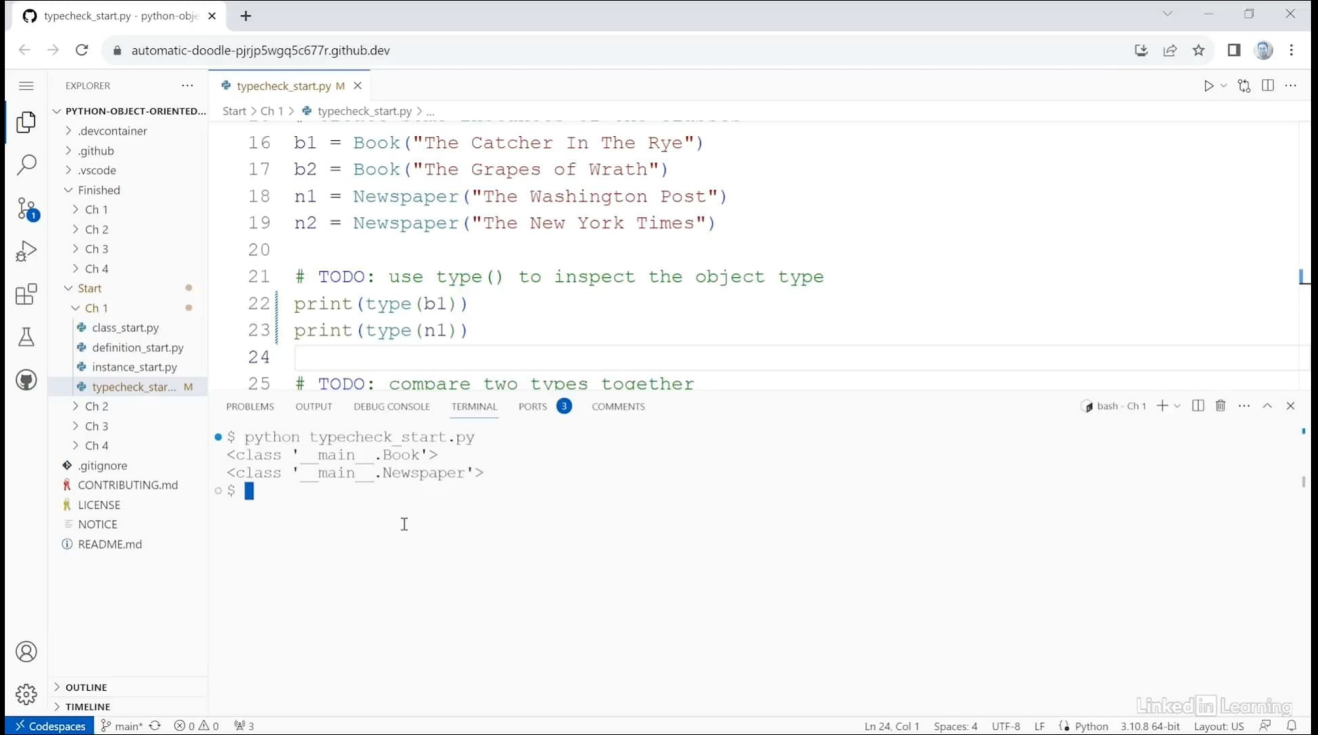
Start Time – 10:30 AM
End Time – 11:30 AM

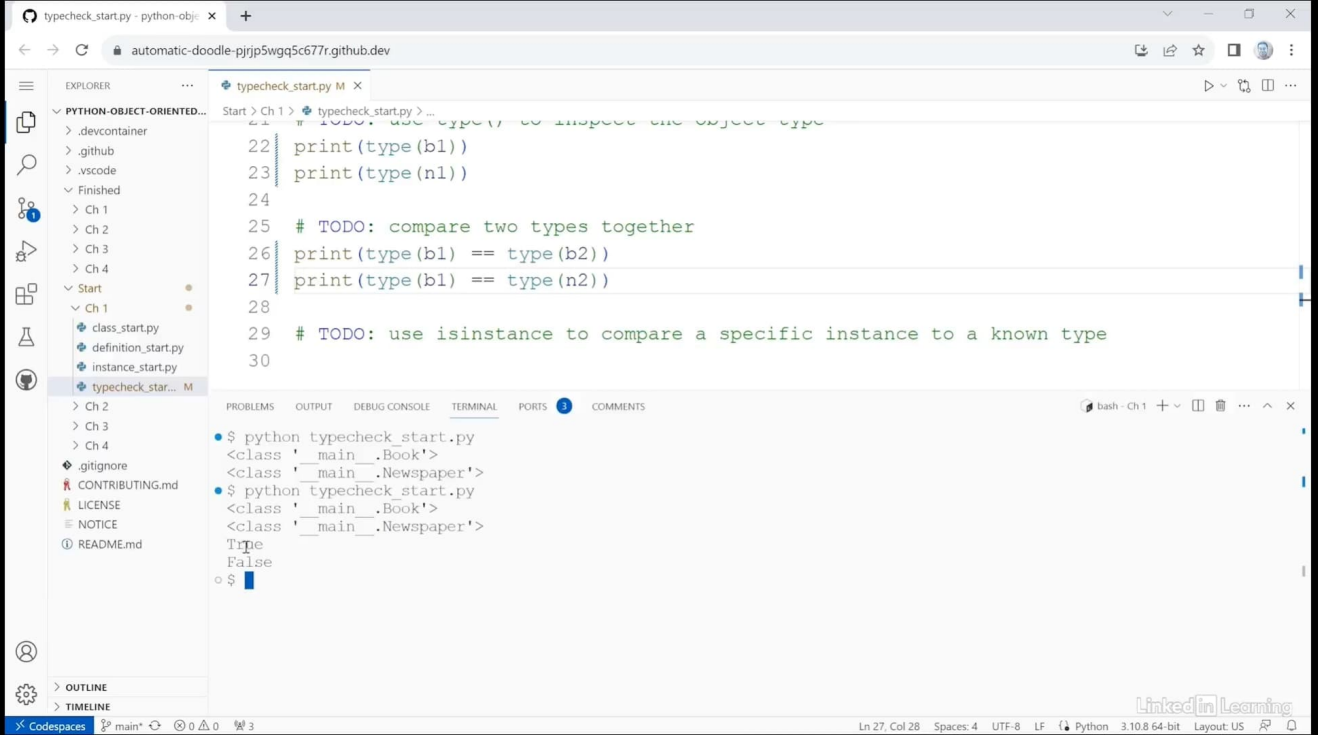
When you finish, please

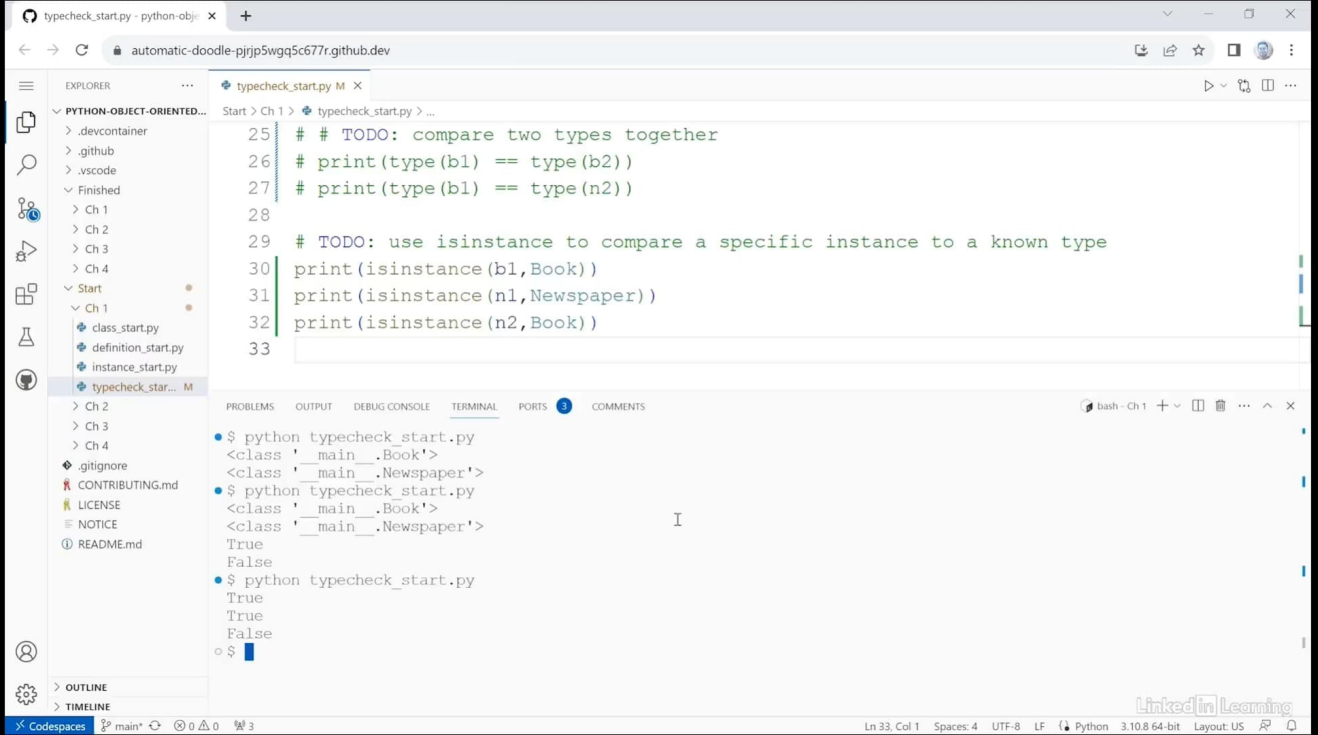
- Save your file to the desktop
- **Email file to Nayanika Dhamija to** hr@intellect-partners.com after one hour.

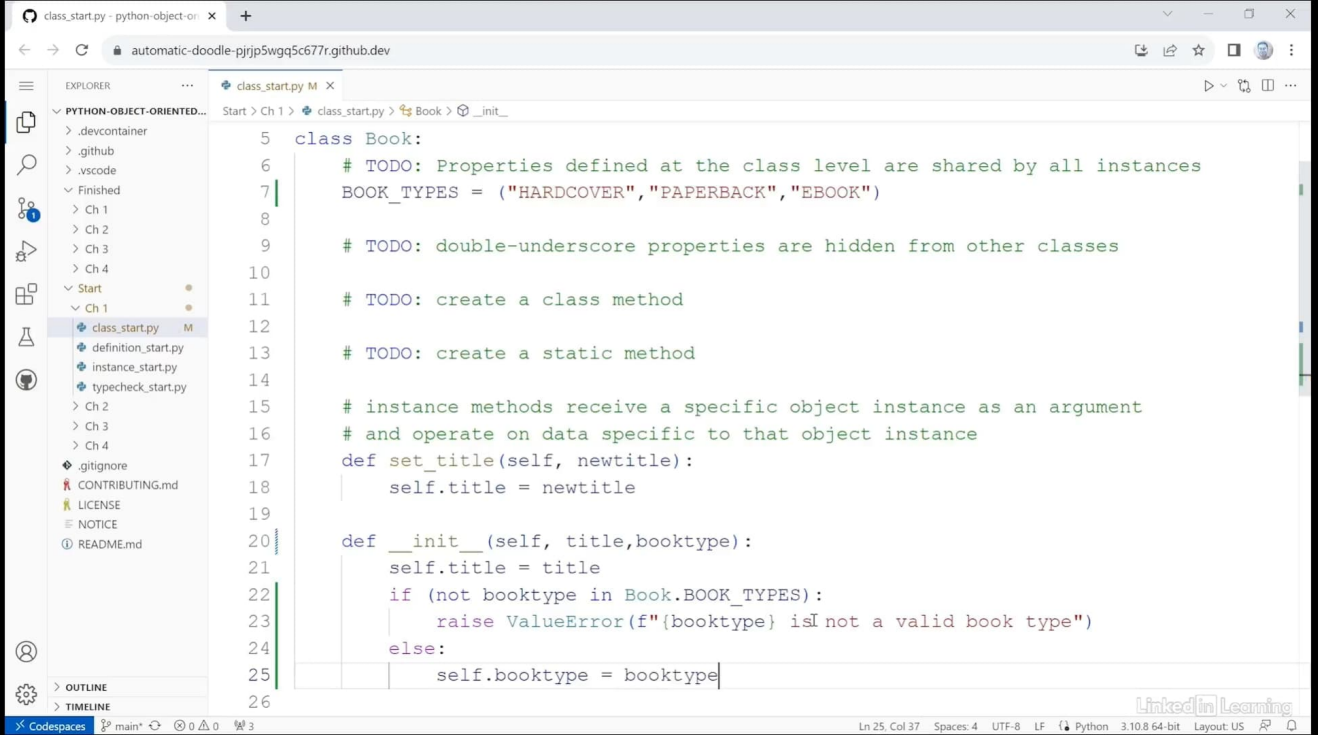
Note: In-case you face any difficulty, please connect with Ms. Nayanika @ 8595626117.

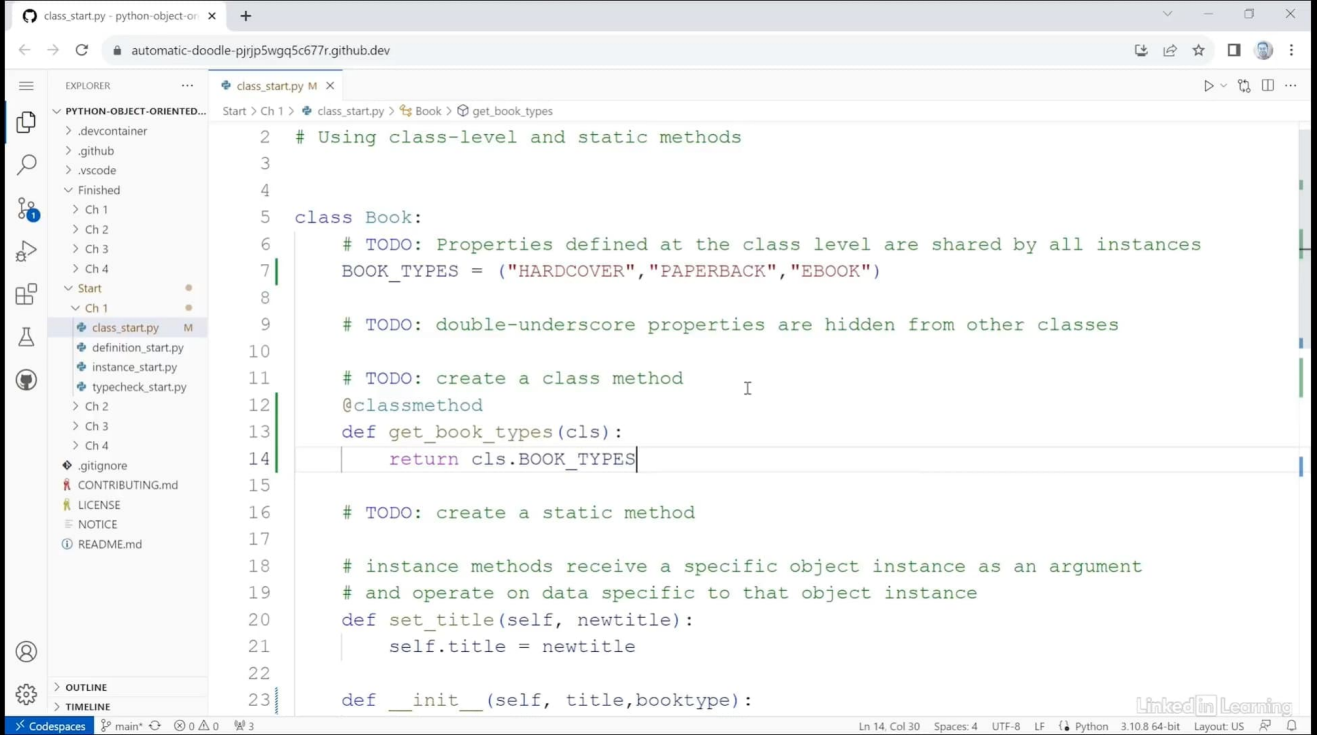

typecheck_start.py - python-obje
automatic-doodle-pjrjp5wgq5c677r.github.dev
EXPLORER
typecheck_start.py
PYTHON-OBJECT-ORIENTED...
.devcontainer
.github
.vscode
Finished
Ch 1
Ch 2
Ch 3
Ch 4
Start
Ch 1
class_start.py
definition_start.py
instance_start.py
typecheck_start.py
Ch 2
Ch 3
Ch 4
.gitignore
CONTRIBUTING.md
LICENSE
NOTICE
README.md
OUTLINE
TIMELINE
Start > Ch 1 > typecheck_start.py > ...
4
5 class Book:
6 def __init__(self, title):
7 self.title = title
8
9
10 class Newspaper:
11 def __init__(self, name):
12 self.name = name
13
14
15 # Create some instances of the classes
16 b1 = Book("The Catcher In The Rye")
17 b2 = Book("The Grapes of Wrath")
18 n1 = Newspaper("The Washington Post")
19 n2 = Newspaper("The New York Times")
20
21 # TODO: use type() to inspect the object type
22
23
24 # TODO: compare two types together
25
Codespaces
main
Ln 1, Col 1
Spaces: 4
UTF-8
LF
Python
3.10.8 64-bit
Layout: US

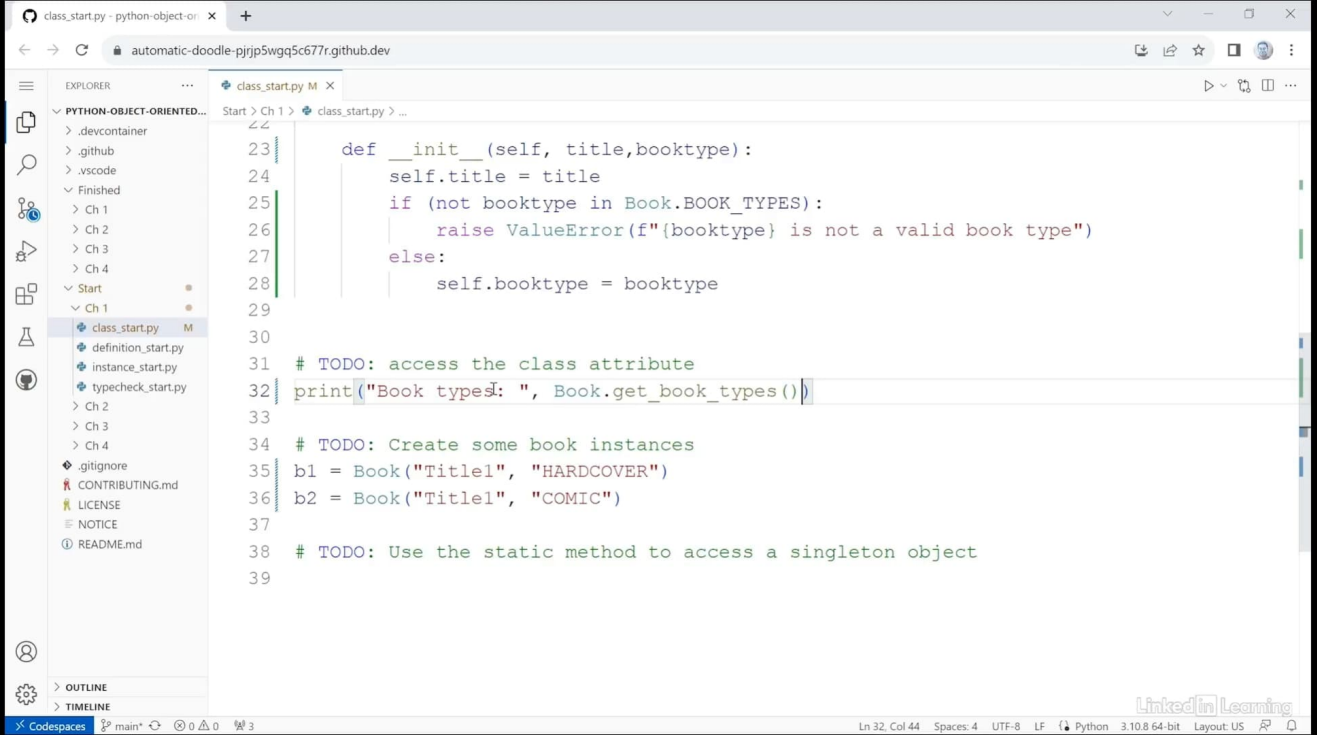

typecheck_start.py - python-obje
automatic-doodle-pjrjp5wgq5c677r.github.dev
EXPLORER
typecheck_start.py M
PYTHON-OBJECT-ORIENTED...
.devcontainer
.github
.vscode
Finished
Ch 1
Ch 2
Ch 3
Ch 4
Start
Ch 1
class_start.py
definition_start.py
instance_start.py
typecheck_star... M
Ch 2
Ch 3
Ch 4
.gitignore
CONTRIBUTING.md
LICENSE
NOTICE
README.md
Start > Ch 1 > typecheck_start.py > ...
16 b1 = Book("The Catcher In The Rye")
17 b2 = Book("The Grapes of Wrath")
18 n1 = Newspaper("The Washington Post")
19 n2 = Newspaper("The New York Times")
20
21 # TODO: use type() to inspect the object type
22 print(type(b1))
23 print(type(n1))
24
25 # TODO: compare two types together
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS 3
COMMENTS
bash - Ch 1
$ python typecheck_start.py
<class '__main__.Book'>
<class '__main__.Newspaper'>
$
OUTLINE
TIMELINE
Codespaces
main*
Ln 24, Col 1
Spaces: 4
UTF-8
LF
Python
3.10.8 64-bit
Layout: US
LinkedIn Learning

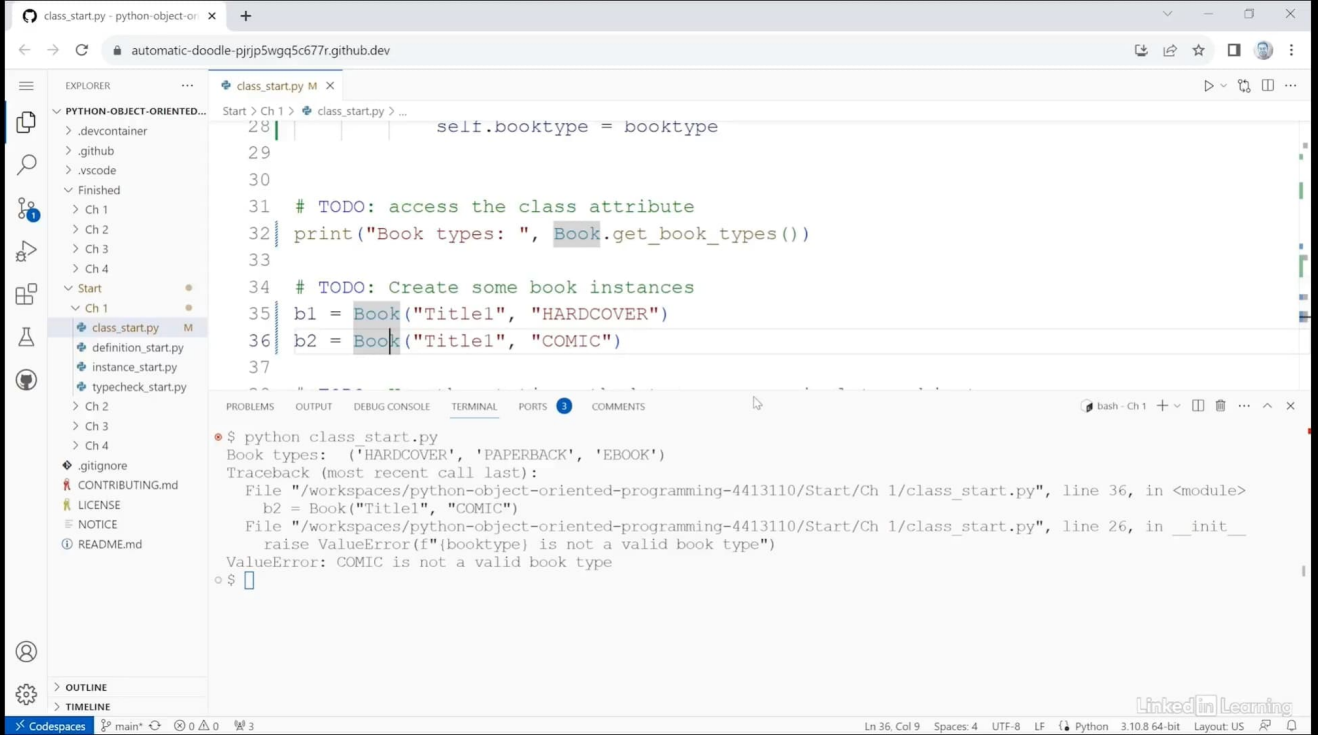

typecheck_start.py - python-obje
automatic-doodle-pjrjp5wgq5c677r.github.dev
EXPLORER
typecheck_start.py M
Start > Ch 1 > typecheck_start.py > ...
PYTHON-OBJECT-ORIENTED...
.devcontainer
.github
.vscode
Finished
Ch 1
Ch 2
Ch 3
Ch 4
Start
Ch 1
class_start.py
definition_start.py
instance_start.py
typecheck_star... M
Ch 2
Ch 3
Ch 4
.gitignore
CONTRIBUTING.md
LICENSE
NOTICE
README.md
22 print(type(b1))
23 print(type(n1))
24
25 # TODO: compare two types together
26 print(type(b1) == type(b2))
27 print(type(b1) == type(n2))
28
29 # TODO: use isinstance to compare a specific instance to a known type
30
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS 3
COMMENTS
bash - Ch 1
$ python typecheck_start.py
<class '__main__.Book'>
<class '__main__.Newspaper'>
$ python typecheck_start.py
<class '__main__.Book'>
<class '__main__.Newspaper'>
True
False
$
OUTLINE
TIMELINE
Codespaces
main*
Ln 27, Col 28
Spaces: 4
UTF-8
LF
Python
3.10.8 64-bit
Layout: US
LinkedIn Learning

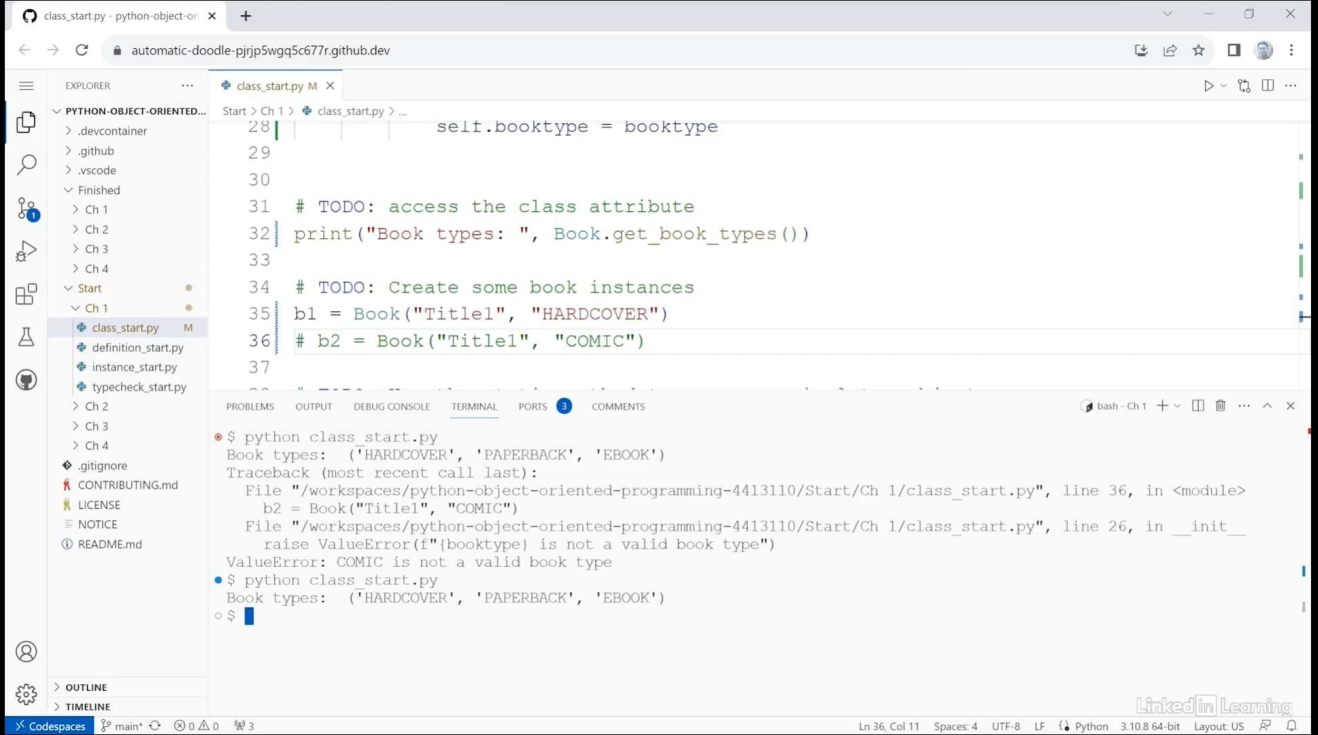

typecheck_start.py - python-obje
automatic-doodle-pjrjp5wgq5c677r.github.dev
EXPLORER
PYTHON-OBJECT-ORIENTED...
.devcontainer
.github
.vscode
Finished
Ch 1
Ch 2
Ch 3
Ch 4
Start
Ch 1
class_start.py
definition_start.py
instance_start.py
typecheck_star... M
Ch 2
Ch 3
Ch 4
.gitignore
CONTRIBUTING.md
LICENSE
NOTICE
README.md
typecheck_start.py M
Start > Ch 1 > typecheck_start.py > ...
25 # # TODO: compare two types together
26 # print(type(b1) == type(b2))
27 # print(type(b1) == type(n2))
28
29 # TODO: use isinstance to compare a specific instance to a known type
30 print(isinstance(b1,Book))
31 print(isinstance(n1,Newspaper))
32 print(isinstance(n2,Book))
33
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS 3
COMMENTS
bash - Ch 1
$ python typecheck_start.py
<class '__main__.Book'>
<class '__main__.Newspaper'>
$ python typecheck_start.py
<class '__main__.Book'>
<class '__main__.Newspaper'>
True
False
$ python typecheck_start.py
True
True
False
$
OUTLINE
TIMELINE
Codespaces
main*
Ln 33, Col 1
Spaces: 4
UTF-8
LF
Python
3.10.8 64-bit
Layout: US
LinkedIn Learning

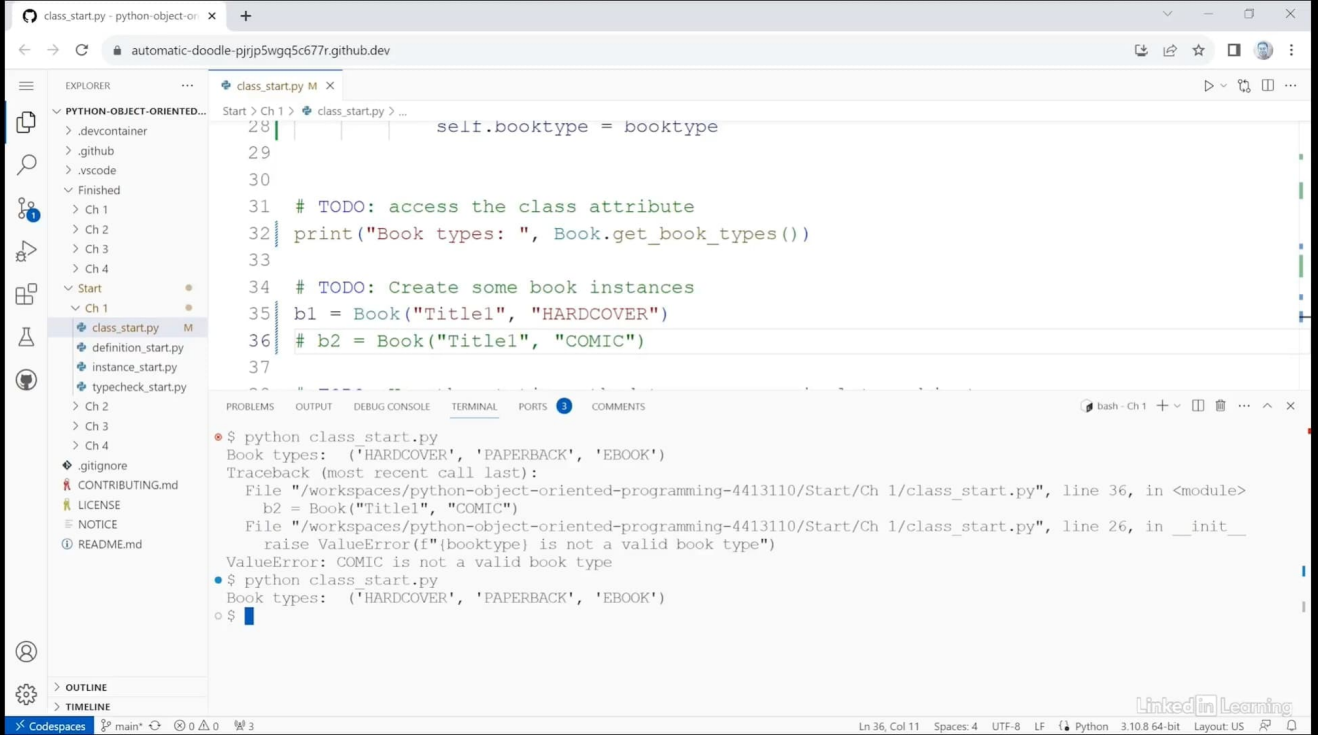

class_start.py - python-object-or
automatic-doodle-pjrjp5wgq5c677r.github.dev
EXPLORER
class_start.py M
Start > Ch 1 > class_start.py > Book > __init__
PYTHON-OBJECT-ORIENTED...
.devcontainer
.github
.vscode
Finished
Ch 1
Ch 2
Ch 3
Ch 4
Start
Ch 1
class_start.py M
definition_start.py
instance_start.py
typecheck_start.py
Ch 2
Ch 3
Ch 4
.gitignore
CONTRIBUTING.md
LICENSE
NOTICE
README.md
OUTLINE
TIMELINE
5 class Book:
6 # TODO: Properties defined at the class level are shared by all instances
7 BOOK_TYPES = ("HARDCOVER","PAPERBACK","EBOOK")
8
9 # TODO: double-underscore properties are hidden from other classes
10
11 # TODO: create a class method
12
13 # TODO: create a static method
14
15 # instance methods receive a specific object instance as an argument
16 # and operate on data specific to that object instance
17 def set_title(self, newtitle):
18 self.title = newtitle
19
20 def __init__(self, title,booktype):
21 self.title = title
22 if (not booktype in Book.BOOK_TYPES):
23 raise ValueError(f"{booktype} is not a valid book type")
24 else:
25 self.booktype = booktype
26
Codespaces
main*
Ln 25, Col 37
Spaces: 4
UTF-8
LF
Python
3.10.8 64-bit
Layout: US
LinkedIn Learning

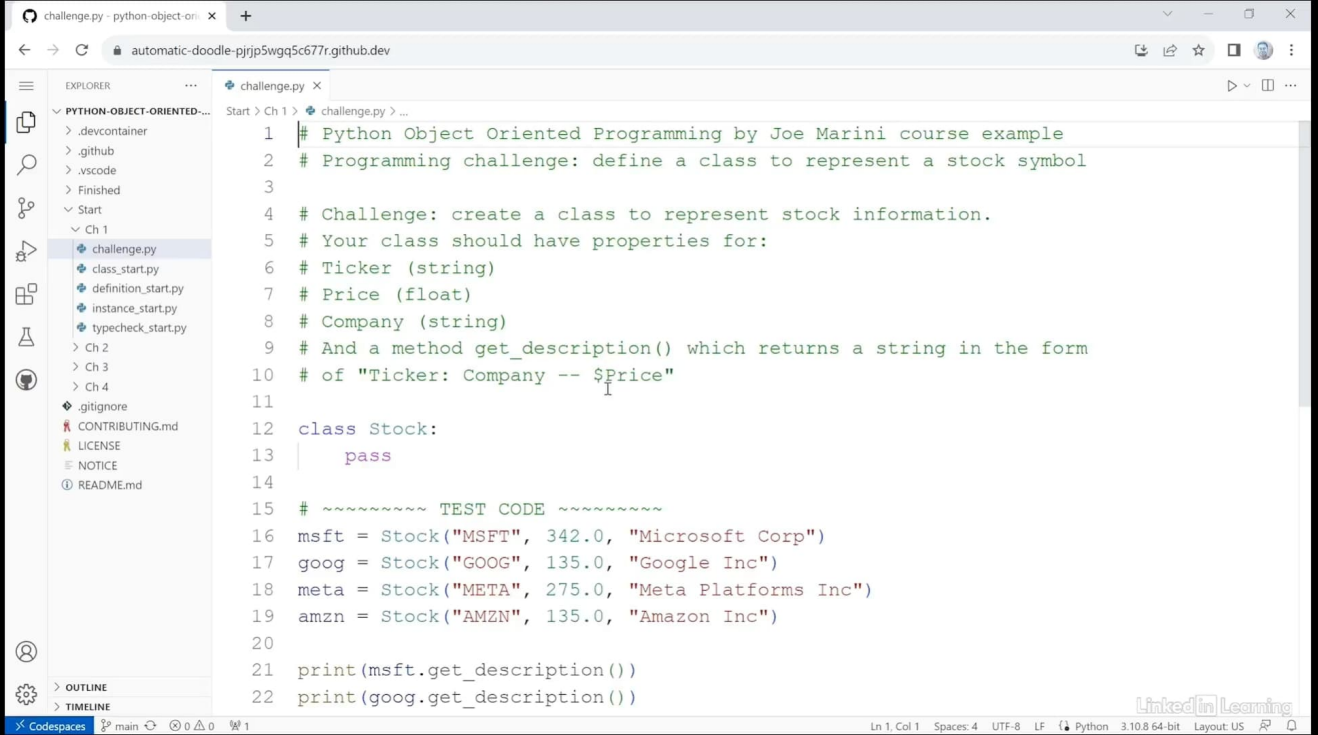

class_start.py - python-object-or
automatic-doodle-pjrjp5wgq5c677r.github.dev
EXPLORER
class_start.py M
Start > Ch 1 > class_start.py > Book > get_book_types
PYTHON-OBJECT-ORIENTED...
.devcontainer
.github
.vscode
Finished
Ch 1
Ch 2
Ch 3
Ch 4
Start
Ch 1
class_start.py M
definition_start.py
instance_start.py
typecheck_start.py
Ch 2
Ch 3
Ch 4
.gitignore
CONTRIBUTING.md
LICENSE
NOTICE
README.md
OUTLINE
TIMELINE
# Using class-level and static methods
class Book:
# TODO: Properties defined at the class level are shared by all instances
BOOK_TYPES = ("HARDCOVER","PAPERBACK","EBOOK")
# TODO: double-underscore properties are hidden from other classes
# TODO: create a class method
@classmethod
def get_book_types(cls):
return cls.BOOK_TYPES
# TODO: create a static method
# instance methods receive a specific object instance as an argument
# and operate on data specific to that object instance
def set_title(self, newtitle):
self.title = newtitle
def __init__(self, title,booktype):
Codespaces
main*
Ln 14, Col 30
Spaces: 4
UTF-8
LF
Python
3.10.8 64-bit
Layout: US
LinkedIn Learning

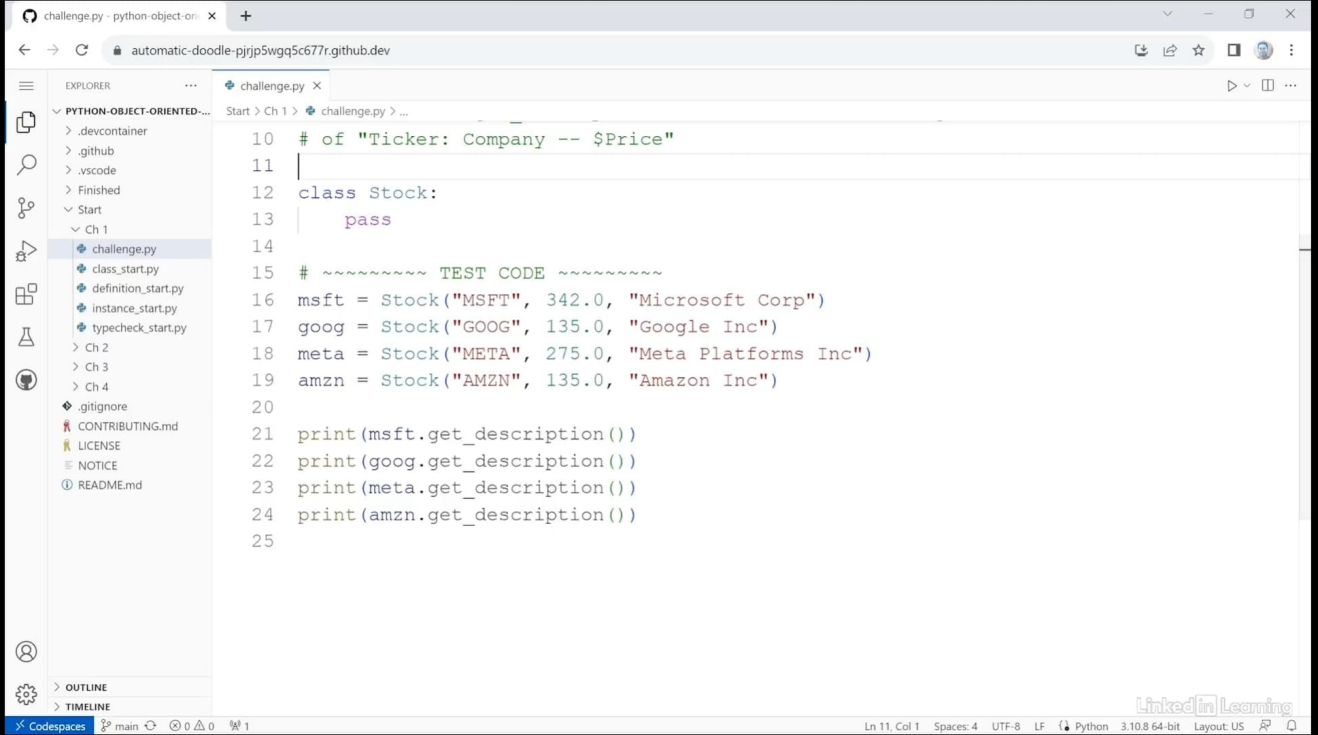

class_start.py - python-object-or
automatic-doodle-pjrjp5wgq5c677r.github.dev
EXPLORER
class_start.py M
Start > Ch 1 > class_start.py > ...
PYTHON-OBJECT-ORIENTED...
.devcontainer
.github
.vscode
Finished
Ch 1
Ch 2
Ch 3
Ch 4
Start
Ch 1
class_start.py M
definition_start.py
instance_start.py
typecheck_start.py
Ch 2
Ch 3
Ch 4
.gitignore
CONTRIBUTING.md
LICENSE
NOTICE
README.md
OUTLINE
TIMELINE
def __init__(self, title,booktype):
self.title = title
if (not booktype in Book.BOOK_TYPES):
raise ValueError(f"{booktype} is not a valid book type")
else:
self.booktype = booktype
# TODO: access the class attribute
print("Book types: ", Book.get_book_types())
# TODO: Create some book instances
b1 = Book("Title1", "HARDCOVER")
b2 = Book("Title1", "COMIC")
# TODO: Use the static method to access a singleton object
Codespaces
main*
Ln 32, Col 44
Spaces: 4
UTF-8
LF
Python
3.10.8 64-bit
Layout: US
LinkedIn Learning

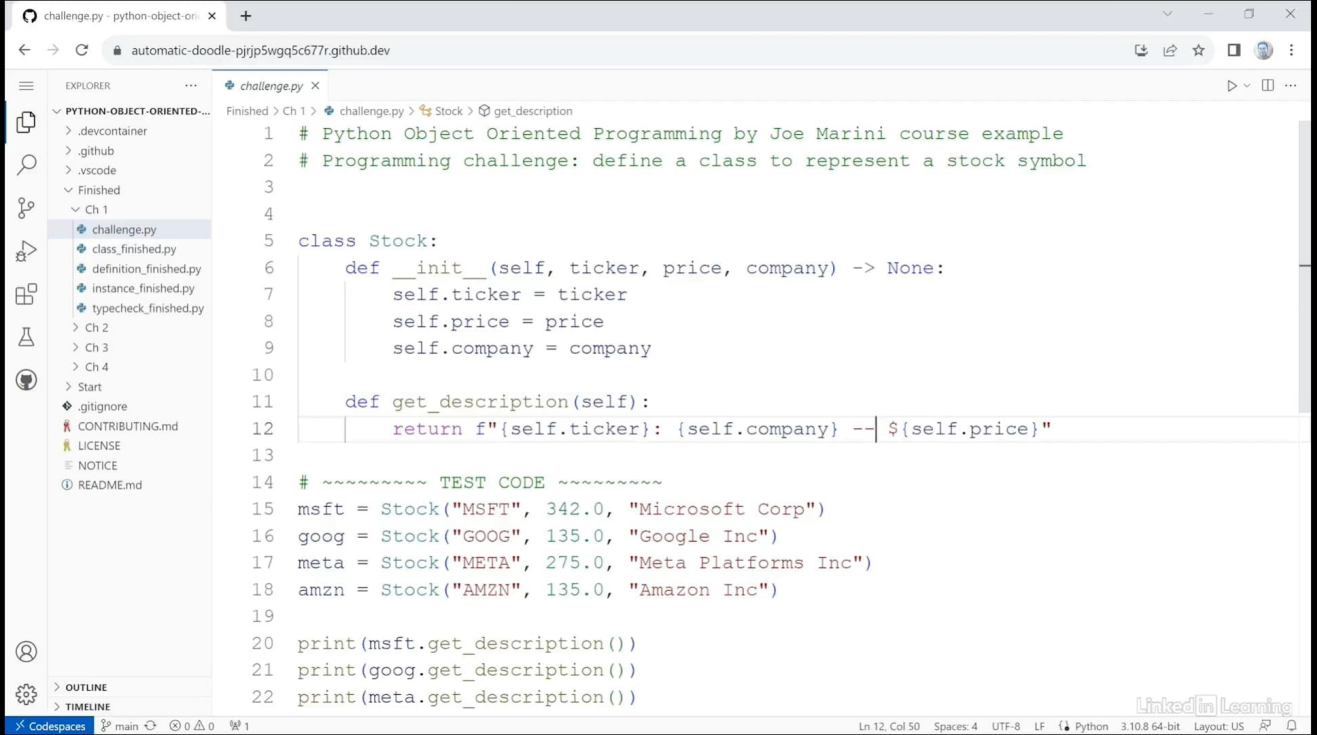

class_start.py - python-object-or
automatic-doodle-pjrjp5wgq5c677r.github.dev
EXPLORER
PYTHON-OBJECT-ORIENTED...
.devcontainer
.github
.vscode
Finished
Ch 1
Ch 2
Ch 3
Ch 4
Start
Ch 1
class_start.py M
definition_start.py
instance_start.py
typecheck_start.py
Ch 2
Ch 3
Ch 4
.gitignore
CONTRIBUTING.md
LICENSE
NOTICE
README.md
OUTLINE
TIMELINE
class_start.py M
Start > Ch 1 > class_start.py > ...
28 self.booktype = booktype
29
30
31 # TODO: access the class attribute
32 print("Book types: ", Book.get_book_types())
33
34 # TODO: Create some book instances
35 b1 = Book("Title1", "HARDCOVER")
36 b2 = Book("Title1", "COMIC")
37
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS 3
COMMENTS
bash - Ch 1
$ python class_start.py
Book types: ('HARDCOVER', 'PAPERBACK', 'EBOOK')
Traceback (most recent call last):
File "/workspaces/python-object-oriented-programming-4413110/Start/Ch 1/class_start.py", line 36, in <module>
b2 = Book("Title1", "COMIC")
File "/workspaces/python-object-oriented-programming-4413110/Start/Ch 1/class_start.py", line 26, in __init__
raise ValueError(f"{booktype} is not a valid book type")
ValueError: COMIC is not a valid book type
$
Codespaces
main*
Ln 36, Col 9
Spaces: 4
UTF-8
LF
Python
3.10.8 64-bit
Layout: US
LinkedIn Learning


class_start.py - python-object-or
automatic-doodle-pjrjp5wgq5c677r.github.dev
EXPLORER
class_start.py M
Start > Ch 1 > class_start.py > ...
28 self.booktype = booktype
29
30
31 # TODO: access the class attribute
32 print("Book types: ", Book.get_book_types())
33
34 # TODO: Create some book instances
35 b1 = Book("Title1", "HARDCOVER")
36 # b2 = Book("Title1", "COMIC")
37
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS 3 COMMENTS
bash - Ch 1
$ python class_start.py
Book types: ('HARDCOVER', 'PAPERBACK', 'EBOOK')
Traceback (most recent call last):
File "/workspaces/python-object-oriented-programming-4413110/Start/Ch 1/class_start.py", line 36, in <module>
b2 = Book("Title1", "COMIC")
File "/workspaces/python-object-oriented-programming-4413110/Start/Ch 1/class_start.py", line 26, in __init__
raise ValueError(f"{booktype} is not a valid book type")
ValueError: COMIC is not a valid book type
$ python class_start.py
Book types: ('HARDCOVER', 'PAPERBACK', 'EBOOK')
$
PYTHON-OBJECT-ORIENTED...
.devcontainer
.github
.vscode
Finished
Ch 1
Ch 2
Ch 3
Ch 4
Start
Ch 1
class_start.py M
definition_start.py
instance_start.py
typecheck_start.py
Ch 2
Ch 3
Ch 4
.gitignore
CONTRIBUTING.md
LICENSE
NOTICE
README.md
OUTLINE
TIMELINE
Codespaces
main*
Ln 36, Col 11
Spaces: 4
UTF-8
LF
Python
3.10.8 64-bit
Layout: US
LinkedIn Learning


class_start.py - python-object-or
automatic-doodle-pjrjp5wgq5c677r.github.dev
EXPLORER
PYTHON-OBJECT-ORIENTED...
.devcontainer
.github
.vscode
Finished
Ch 1
Ch 2
Ch 3
Ch 4
Start
Ch 1
class_start.py M
definition_start.py
instance_start.py
typecheck_start.py
Ch 2
Ch 3
Ch 4
.gitignore
CONTRIBUTING.md
LICENSE
NOTICE
README.md
OUTLINE
TIMELINE
class_start.py M
Start > Ch 1 > class_start.py > ...
28 self.booktype = booktype
29
30
31 # TODO: access the class attribute
32 print("Book types: ", Book.get_book_types())
33
34 # TODO: Create some book instances
35 b1 = Book("Title1", "HARDCOVER")
36 # b2 = Book("Title1", "COMIC")
37
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS 3
COMMENTS
bash - Ch 1
$ python class_start.py
Book types: ('HARDCOVER', 'PAPERBACK', 'EBOOK')
Traceback (most recent call last):
File "/workspaces/python-object-oriented-programming-4413110/Start/Ch 1/class_start.py", line 36, in <module>
b2 = Book("Title1", "COMIC")
File "/workspaces/python-object-oriented-programming-4413110/Start/Ch 1/class_start.py", line 26, in __init__
raise ValueError(f"{booktype} is not a valid book type")
ValueError: COMIC is not a valid book type
$ python class_start.py
Book types: ('HARDCOVER', 'PAPERBACK', 'EBOOK')
$
Codespaces
main*
Ln 36, Col 11
Spaces: 4
UTF-8
LF
Python
3.10.8 64-bit
Layout: US
LinkedIn Learning


challenge.py - python-object-ori
automatic-doodle-pjrjp5wgq5c677r.github.dev
EXPLORER
challenge.py
Start > Ch 1 > challenge.py > ...
PYTHON-OBJECT-ORIENTED-...
.devcontainer
.github
.vscode
Finished
Start
Ch 1
challenge.py
class_start.py
definition_start.py
instance_start.py
typecheck_start.py
Ch 2
Ch 3
Ch 4
.gitignore
CONTRIBUTING.md
LICENSE
NOTICE
README.md
OUTLINE
TIMELINE
1 # Python Object Oriented Programming by Joe Marini course example
2 # Programming challenge: define a class to represent a stock symbol
3
4 # Challenge: create a class to represent stock information.
5 # Your class should have properties for:
6 # Ticker (string)
7 # Price (float)
8 # Company (string)
9 # And a method get_description() which returns a string in the form
10 # of "Ticker: Company -- $Price"
11
12 class Stock:
13 pass
14
15 # ~~~~~~~~~ TEST CODE ~~~~~~~~~
16 msft = Stock("MSFT", 342.0, "Microsoft Corp")
17 goog = Stock("GOOG", 135.0, "Google Inc")
18 meta = Stock("META", 275.0, "Meta Platforms Inc")
19 amzn = Stock("AMZN", 135.0, "Amazon Inc")
20
21 print(msft.get_description())
22 print(goog.get_description())
Codespaces
main
Ln 1, Col 1
Spaces: 4
UTF-8
LF
Python
3.10.8 64-bit
Layout: US
LinkedIn Learning


challenge.py - python-object-ori
automatic-doodle-pjrjp5wgq5c677r.github.dev
EXPLORER
challenge.py
PYTHON-OBJECT-ORIENTED-...
.devcontainer
.github
.vscode
Finished
Start
Ch 1
challenge.py
class_start.py
definition_start.py
instance_start.py
typecheck_start.py
Ch 2
Ch 3
Ch 4
.gitignore
CONTRIBUTING.md
LICENSE
NOTICE
README.md
Start > Ch 1 > challenge.py > ...
10 # of "Ticker: Company -- $Price"
11
12 class Stock:
13 pass
14
15 # ~~~~~~~~~ TEST CODE ~~~~~~~~~
16 msft = Stock("MSFT", 342.0, "Microsoft Corp")
17 goog = Stock("GOOG", 135.0, "Google Inc")
18 meta = Stock("META", 275.0, "Meta Platforms Inc")
19 amzn = Stock("AMZN", 135.0, "Amazon Inc")
20
21 print(msft.get_description())
22 print(goog.get_description())
23 print(meta.get_description())
24 print(amzn.get_description())
25
OUTLINE
TIMELINE
Codespaces
main
Ln 11, Col 1
Spaces: 4
UTF-8
LF
Python
3.10.8 64-bit
Layout: US
LinkedIn Learning


challenge.py - python-object-ori
automatic-doodle-pjrjp5wgq5c677r.github.dev
EXPLORER
challenge.py
Finished > Ch 1 > challenge.py > Stock > get_description
PYTHON-OBJECT-ORIENTED-...
.devcontainer
.github
.vscode
Finished
Ch 1
challenge.py
class_finished.py
definition_finished.py
instance_finished.py
typecheck_finished.py
Ch 2
Ch 3
Ch 4
Start
.gitignore
CONTRIBUTING.md
LICENSE
NOTICE
README.md
OUTLINE
TIMELINE
1 # Python Object Oriented Programming by Joe Marini course example
2 # Programming challenge: define a class to represent a stock symbol
3
4
5 class Stock:
6 def __init__(self, ticker, price, company) -> None:
7 self.ticker = ticker
8 self.price = price
9 self.company = company
10
11 def get_description(self):
12 return f"{self.ticker}: {self.company} -- ${self.price}"
13
14 # ~~~~~~~~~ TEST CODE ~~~~~~~~~
15 msft = Stock("MSFT", 342.0, "Microsoft Corp")
16 goog = Stock("GOOG", 135.0, "Google Inc")
17 meta = Stock("META", 275.0, "Meta Platforms Inc")
18 amzn = Stock("AMZN", 135.0, "Amazon Inc")
19
20 print(msft.get_description())
21 print(goog.get_description())
22 print(meta.get_description())
Codespaces
main
Ln 12, Col 50
Spaces: 4
UTF-8
LF
Python
3.10.8 64-bit
Layout: US
LinkedIn Learning

Your Answers

Question **1** of **7**

What is the purpose of the `self` parameter within instance methods of a class?

You are correct

to refer to the instance of the class on which the method is called

Next chapter

Your Answers

Question **2** of **7**

What is the primary purpose of the `init` method?

You are correct

to initialize the instance content such as properties and variables

Next chapter

Your Answers

Question **3** of **7**

What is the primary difference between the `type()` function and the `isinstance()` function when checking the type of an object?

You are correct

type() returns the specific class or type of an object, while isinstance() checks if an object is an instance of a particular class or its subclasses

Next chapter

Your Answers

Question **4** of **7**

What is the correct code to determine if an instance of a class is either a `MyClass` object or a descendant of one?

You are correct

```
result = isinstance(my_obj, MyClass)
```

Your Answers

Question **5** of **7**

What is an instance method?

You are correct

a method that can be called on a specific instance of a class

Next chapter

Your Answers

Question **6** of **7**

What is an instance attribute in Python?

You are correct

a variable that holds data specific to an instance of a class

Next chapter

Your Answers

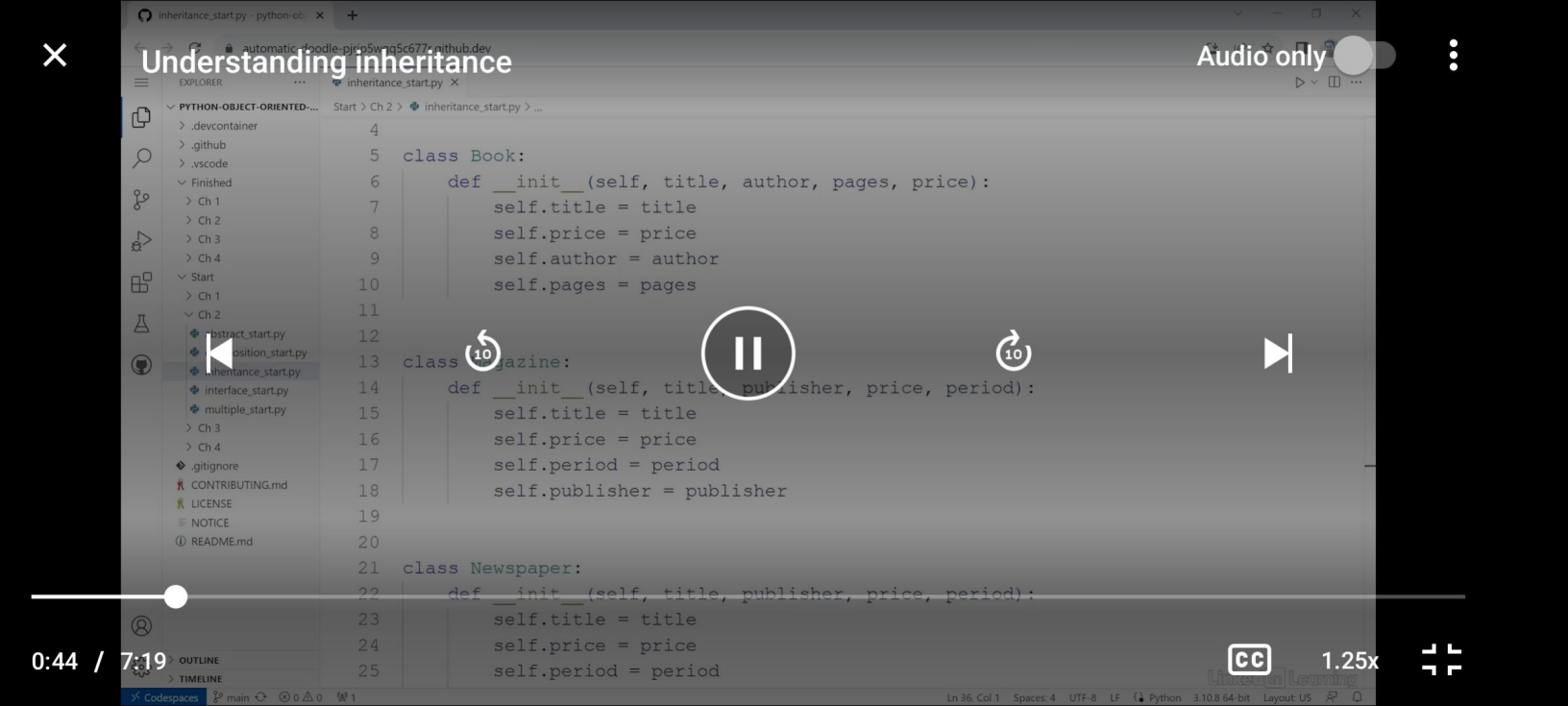
Question **7** of **7**

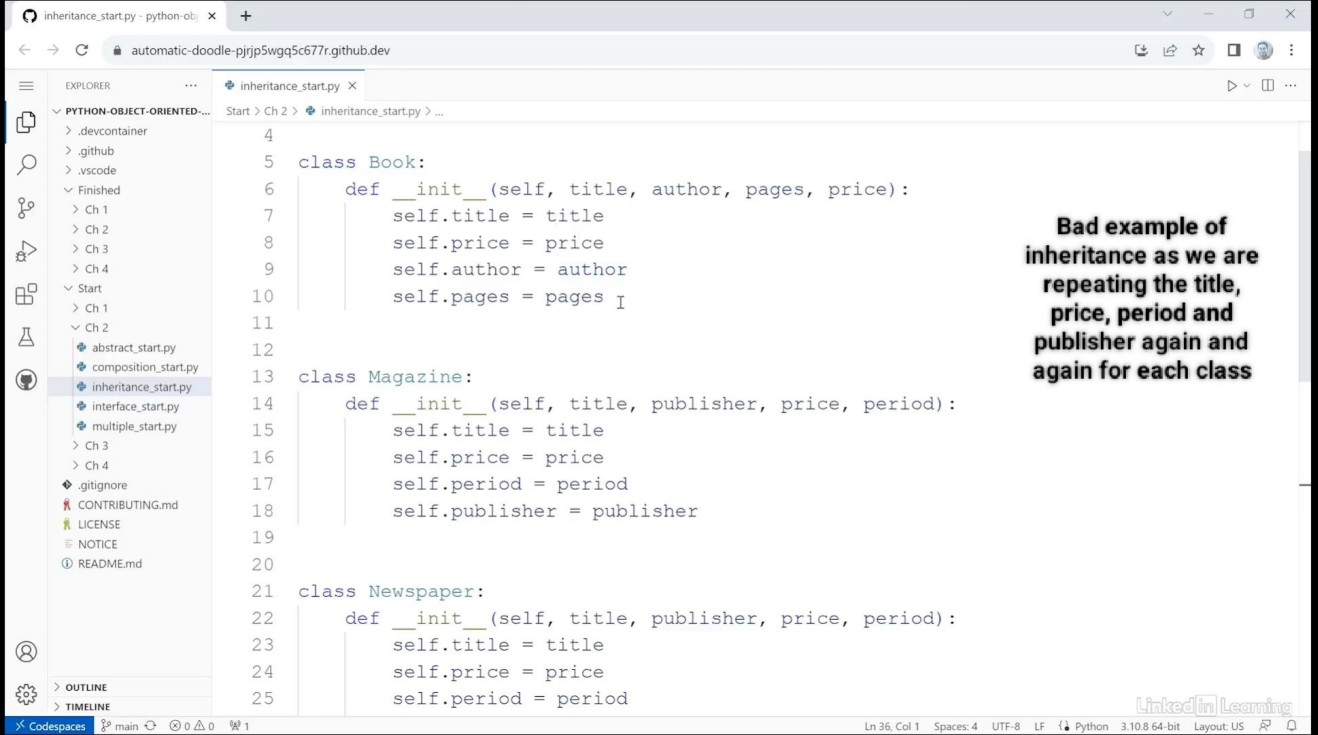
What is the key difference between class variables and instance variables?

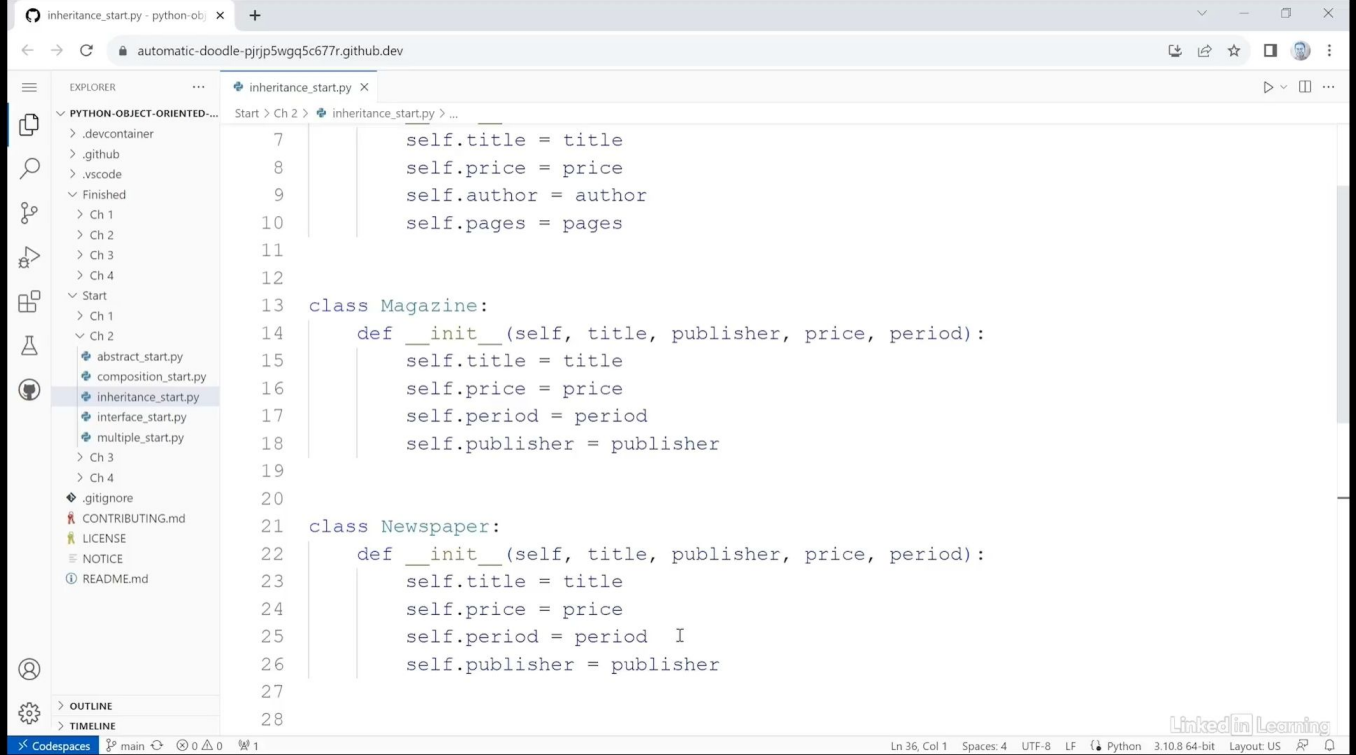
You are correct

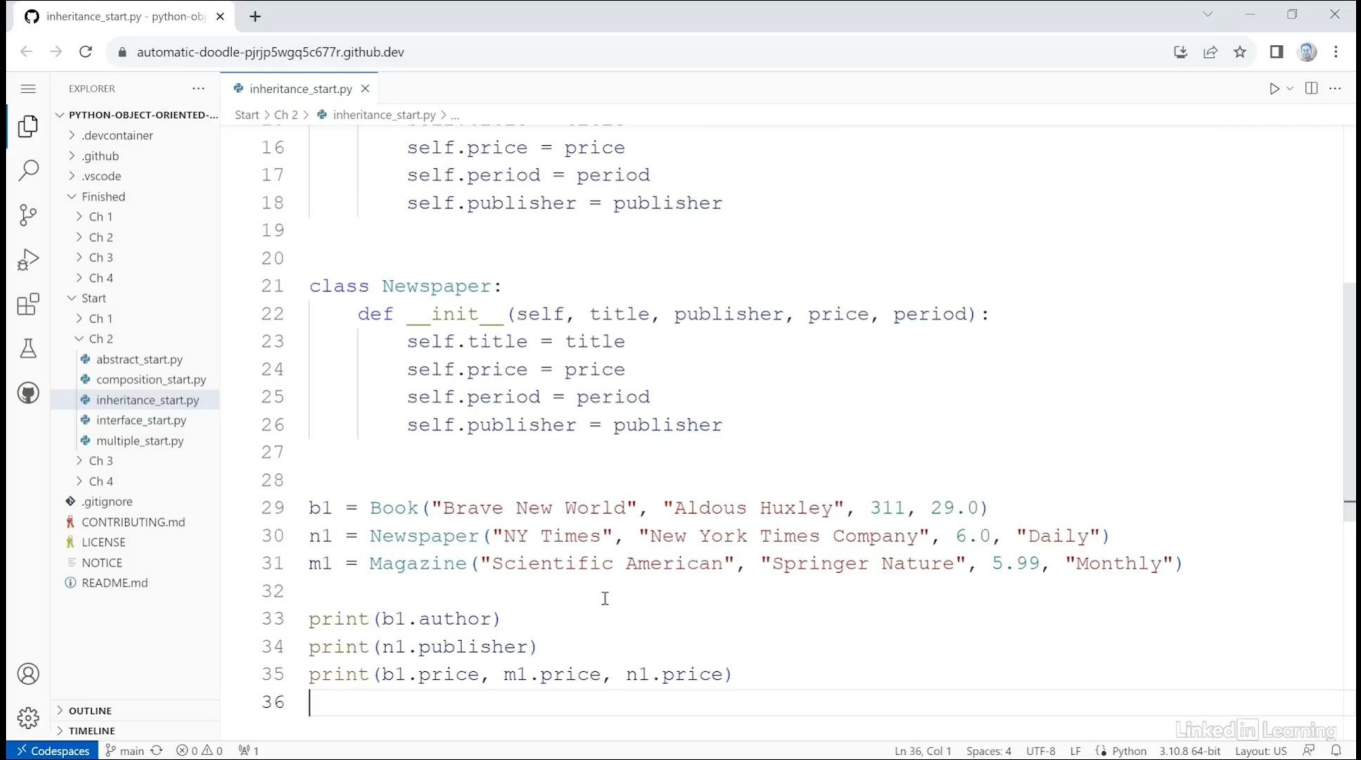
Class variables are shared among all instances of the class, while instance variables are unique to each instance.

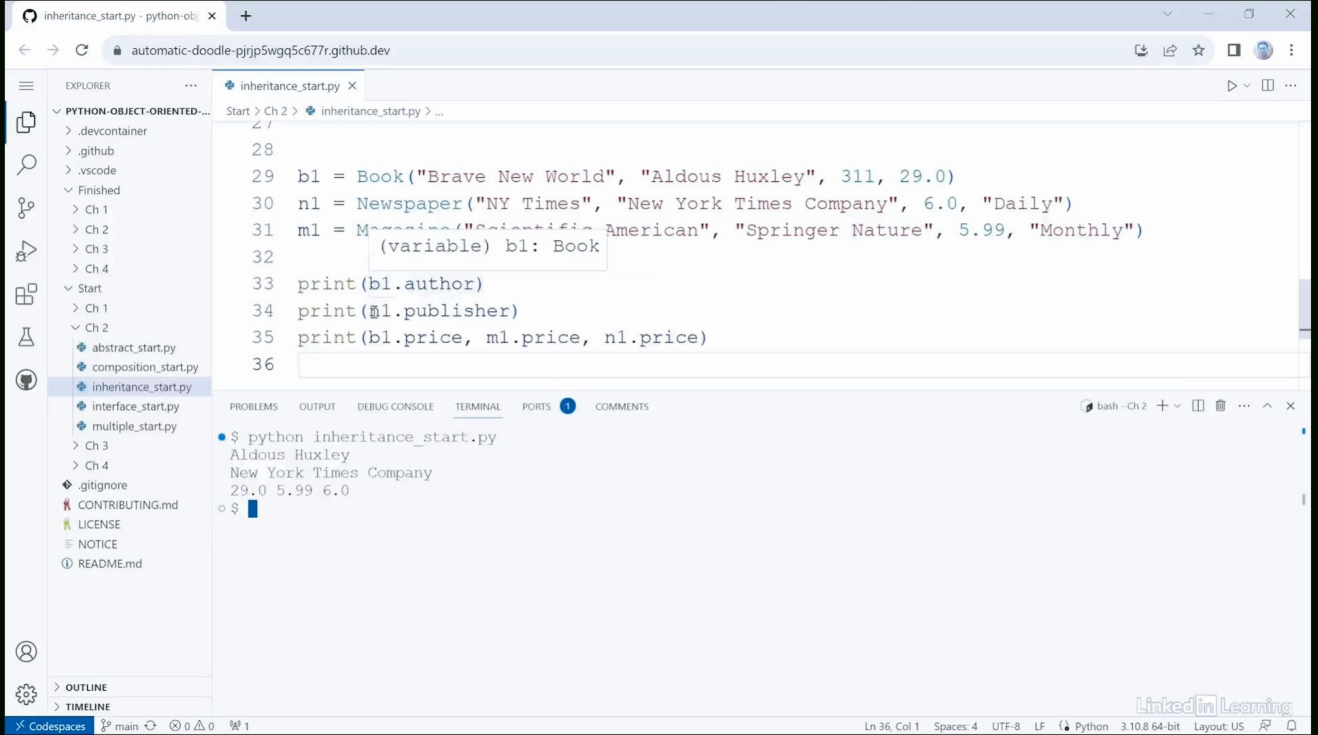
Next chapter

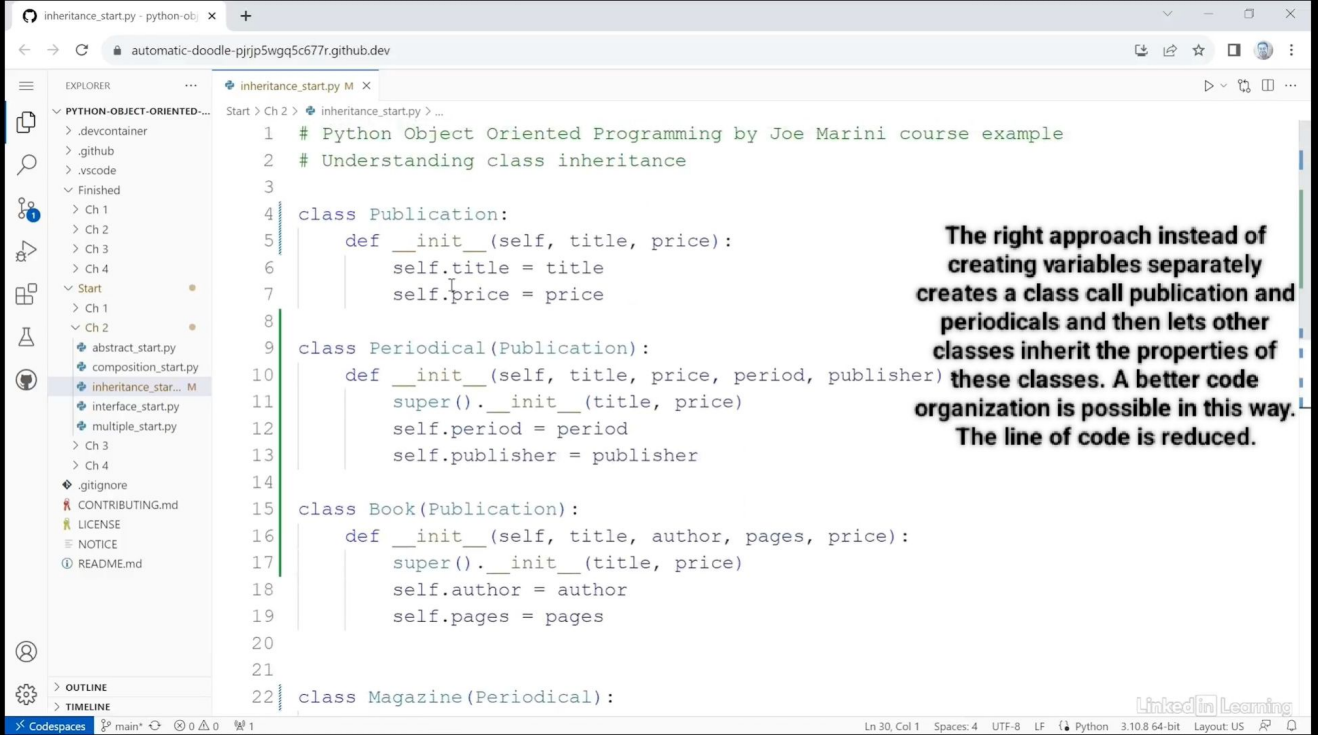

Understanding inheritance
Audio only
0:44 / 7:19
CC
1.25x

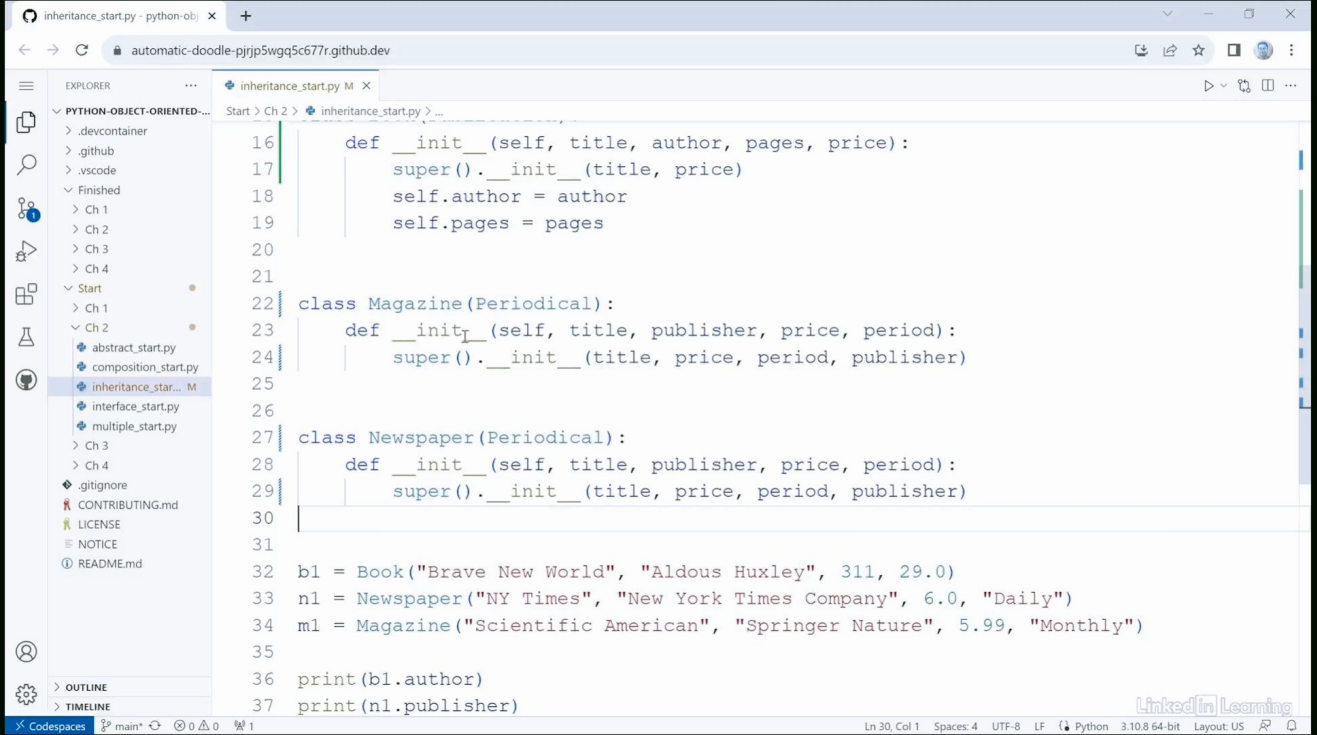

inheritance_start.py - python-obj
automatic-doodle-pjrjp5wgq5c677r.github.dev
EXPLORER
PYTHON-OBJECT-ORIENTED-...
.devcontainer
.github
.vscode
Finished
Ch 1
Ch 2
Ch 3
Ch 4
Start
Ch 1
Ch 2
abstract_start.py
composition_start.py
inheritance_start.py
interface_start.py
multiple_start.py
Ch 3
Ch 4
.gitignore
CONTRIBUTING.md
LICENSE
NOTICE
README.md
OUTLINE
TIMELINE
inheritance_start.py
Start > Ch 2 > inheritance_start.py > ...
4
5 class Book:
6 def __init__(self, title, author, pages, price):
7 self.title = title
8 self.price = price
9 self.author = author
10 self.pages = pages
11
12
13 class Magazine:
14 def __init__(self, title, publisher, price, period):
15 self.title = title
16 self.price = price
17 self.period = period
18 self.publisher = publisher
19
20
21 class Newspaper:
22 def __init__(self, title, publisher, price, period):
23 self.title = title
24 self.price = price
25 self.period = period
Bad example of inheritance as we are repeating the title, price, period and publisher again and again for each class
Codespaces
main
Ln 36, Col 1
Spaces: 4
UTF-8
LF
Python
3.10.8 64-bit
Layout: US
LinkedIn Learning

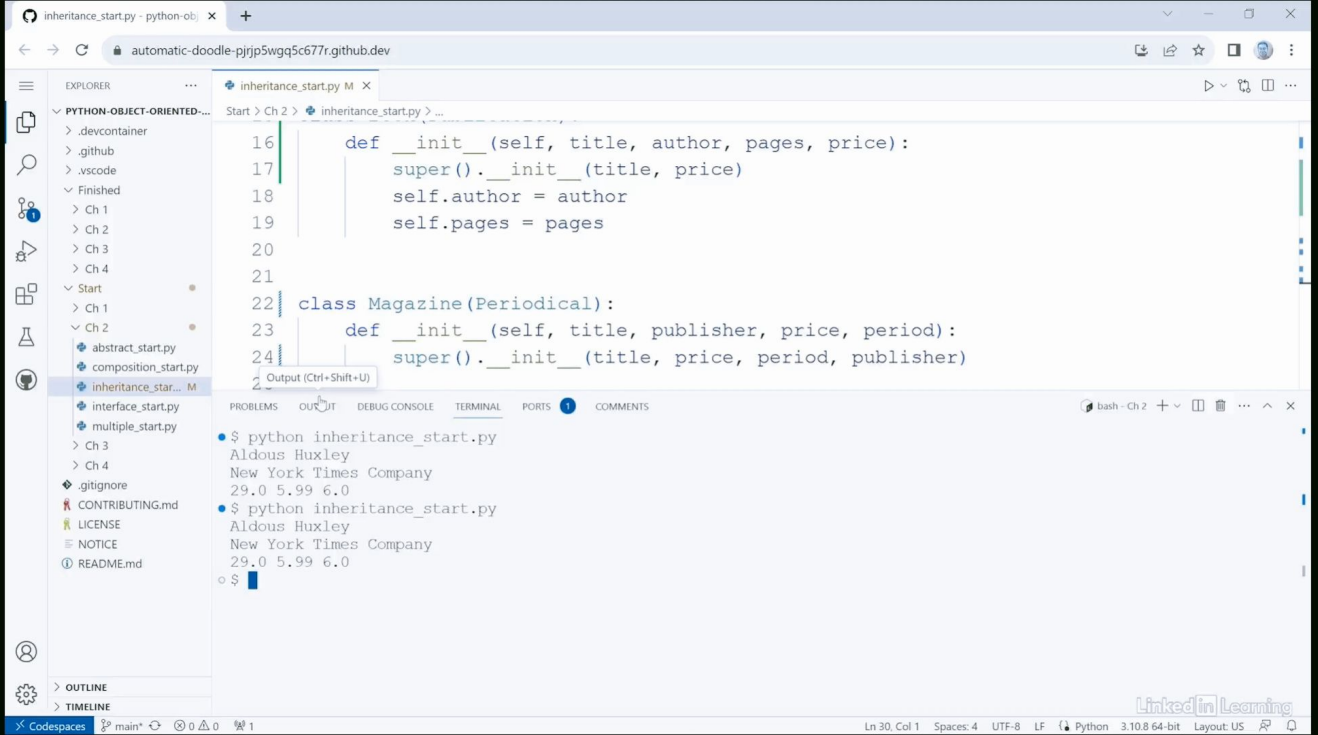

inheritance_start.py - python-obj
automatic-doodle-pjrjp5wgq5c677r.github.dev
EXPLORER
PYTHON-OBJECT-ORIENTED-...
.devcontainer
.github
.vscode
Finished
Ch 1
Ch 2
Ch 3
Ch 4
Start
Ch 1
Ch 2
abstract_start.py
composition_start.py
inheritance_start.py
interface_start.py
multiple_start.py
Ch 3
Ch 4
.gitignore
CONTRIBUTING.md
LICENSE
NOTICE
README.md
OUTLINE
TIMELINE
inheritance_start.py
Start > Ch 2 > inheritance_start.py > ...
7 self.title = title
8 self.price = price
9 self.author = author
10 self.pages = pages
11
12
13 class Magazine:
14 def __init__(self, title, publisher, price, period):
15 self.title = title
16 self.price = price
17 self.period = period
18 self.publisher = publisher
19
20
21 class Newspaper:
22 def __init__(self, title, publisher, price, period):
23 self.title = title
24 self.price = price
25 self.period = period
26 self.publisher = publisher
27
28
Codespaces
main
Ln 36, Col 1
Spaces: 4
UTF-8
LF
Python
3.10.8 64-bit
Layout: US
LinkedIn Learning

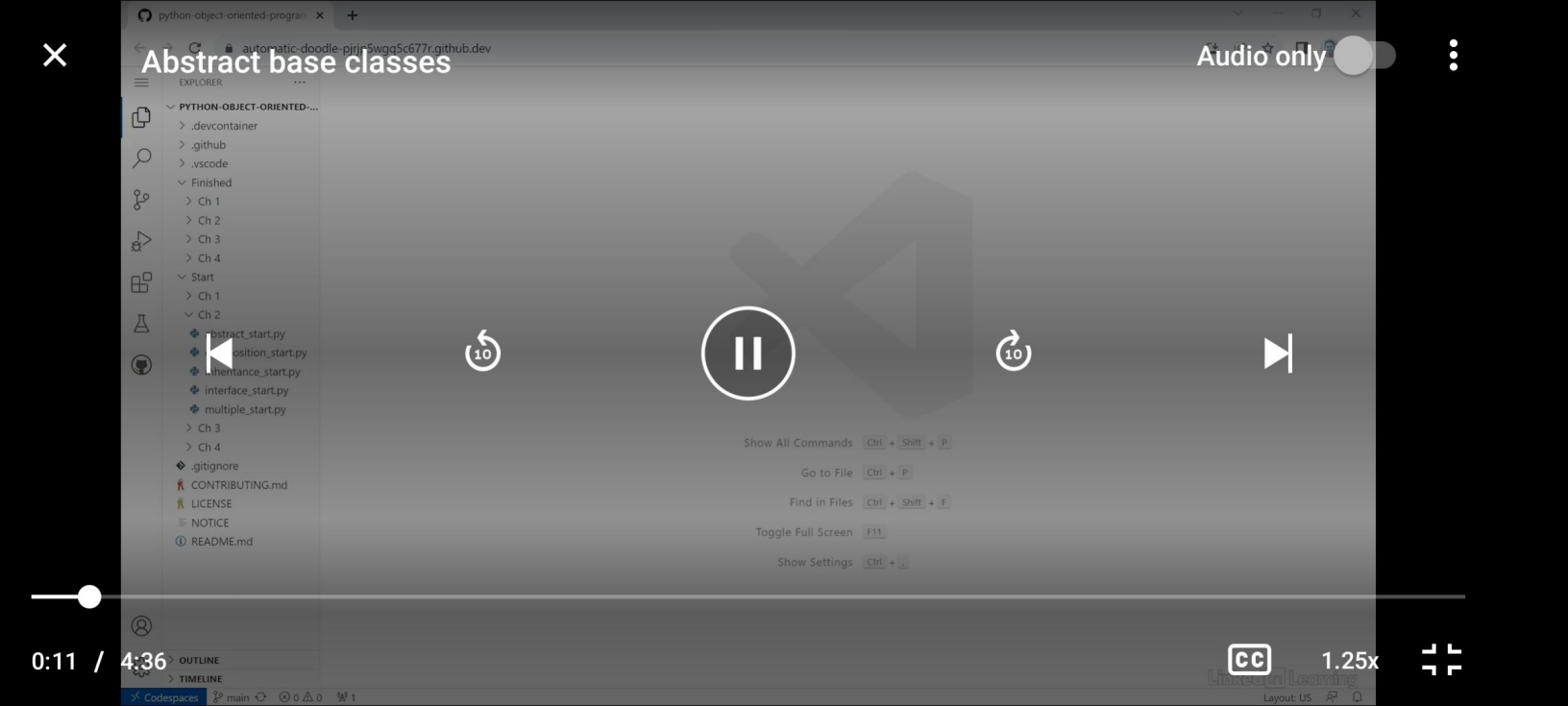

inheritance_start.py - python-obj
automatic-doodle-pjrjp5wgq5c677r.github.dev
EXPLORER
PYTHON-OBJECT-ORIENTED-...
.devcontainer
.github
.vscode
Finished
Ch 1
Ch 2
Ch 3
Ch 4
Start
Ch 1
Ch 2
abstract_start.py
composition_start.py
inheritance_start.py
interface_start.py
multiple_start.py
Ch 3
Ch 4
.gitignore
CONTRIBUTING.md
LICENSE
NOTICE
README.md
OUTLINE
TIMELINE
inheritance_start.py
Start > Ch 2 > inheritance_start.py > ...
16 self.price = price
17 self.period = period
18 self.publisher = publisher
19
20
21 class Newspaper:
22 def __init__(self, title, publisher, price, period):
23 self.title = title
24 self.price = price
25 self.period = period
26 self.publisher = publisher
27
28
29 b1 = Book("Brave New World", "Aldous Huxley", 311, 29.0)
30 n1 = Newspaper("NY Times", "New York Times Company", 6.0, "Daily")
31 m1 = Magazine("Scientific American", "Springer Nature", 5.99, "Monthly")
32
33 print(b1.author)
34 print(n1.publisher)
35 print(b1.price, m1.price, n1.price)
36
Codespaces
main
Ln 36, Col 1
Spaces: 4
UTF-8
LF
Python
3.10.8 64-bit
Layout: US
LinkedIn Learning


inheritance_start.py - python-obj
automatic-doodle-pjrjp5wgq5c677r.github.dev
EXPLORER
PYTHON-OBJECT-ORIENTED-...
.devcontainer
.github
.vscode
Finished
Ch 1
Ch 2
Ch 3
Ch 4
Start
Ch 1
Ch 2
abstract_start.py
composition_start.py
inheritance_start.py
interface_start.py
multiple_start.py
Ch 3
Ch 4
.gitignore
CONTRIBUTING.md
LICENSE
NOTICE
README.md
OUTLINE
TIMELINE
inheritance_start.py
Start > Ch 2 > inheritance_start.py > ...
b1 = Book("Brave New World", "Aldous Huxley", 311, 29.0)
n1 = Newspaper("NY Times", "New York Times Company", 6.0, "Daily")
m1 = Magazine("Scientific American", "Springer Nature", 5.99, "Monthly")
(variable) b1: Book
print(b1.author)
print(m1.publisher)
print(b1.price, m1.price, n1.price)
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS 1
COMMENTS
bash - Ch 2
$ python inheritance_start.py
Aldous Huxley
New York Times Company
29.0 5.99 6.0
$
Codespaces
main
Ln 36, Col 1
Spaces: 4
UTF-8
LF
Python
3.10.8 64-bit
Layout: US
LinkedIn Learning


inheritance_start.py - python-obj
automatic-doodle-pjrjp5wgq5c677r.github.dev
EXPLORER
PYTHON-OBJECT-ORIENTED-...
.devcontainer
.github
.vscode
Finished
Ch 1
Ch 2
Ch 3
Ch 4
Start
Ch 1
Ch 2
abstract_start.py
composition_start.py
inheritance_star... M
interface_start.py
multiple_start.py
Ch 3
Ch 4
.gitignore
CONTRIBUTING.md
LICENSE
NOTICE
README.md
OUTLINE
TIMELINE
inheritance_start.py M
Start > Ch 2 > inheritance_start.py > ...
1 # Python Object Oriented Programming by Joe Marini course example
2 # Understanding class inheritance
3
4 class Publication:
5 def __init__(self, title, price):
6 self.title = title
7 self.price = price
8
9 class Periodical(Publication):
10 def __init__(self, title, price, period, publisher):
11 super().__init__(title, price)
12 self.period = period
13 self.publisher = publisher
14
15 class Book(Publication):
16 def __init__(self, title, author, pages, price):
17 super().__init__(title, price)
18 self.author = author
19 self.pages = pages
20
21
22 class Magazine(Periodical):
The right approach instead of creating variables separately creates a class call publication and periodicals and then lets other classes inherit the properties of these classes. A better code organization is possible in this way. The line of code is reduced.
Codespaces
main*
Ln 30, Col 1
Spaces: 4
UTF-8
LF
Python
3.10.8 64-bit
Layout: US
LinkedIn Learning


inheritance_start.py - python-obj
automatic-doodle-pjrjp5wgq5c677r.github.dev
EXPLORER
inheritance_start.py M
Start > Ch 2 > inheritance_start.py > ...
PYTHON-OBJECT-ORIENTED-...
.devcontainer
.github
.vscode
Finished
Ch 1
Ch 2
Ch 3
Ch 4
Start
Ch 1
Ch 2
abstract_start.py
composition_start.py
inheritance_star... M
interface_start.py
multiple_start.py
Ch 3
Ch 4
.gitignore
CONTRIBUTING.md
LICENSE
NOTICE
README.md
OUTLINE
TIMELINE
16 def __init__(self, title, author, pages, price):
17 super().__init__(title, price)
18 self.author = author
19 self.pages = pages
20
21
22 class Magazine(Periodical):
23 def __init__(self, title, publisher, price, period):
24 super().__init__(title, price, period, publisher)
25
26
27 class Newspaper(Periodical):
28 def __init__(self, title, publisher, price, period):
29 super().__init__(title, price, period, publisher)
30
31
32 b1 = Book("Brave New World", "Aldous Huxley", 311, 29.0)
33 n1 = Newspaper("NY Times", "New York Times Company", 6.0, "Daily")
34 m1 = Magazine("Scientific American", "Springer Nature", 5.99, "Monthly")
35
36 print(b1.author)
37 print(n1.publisher)
Codespaces
main*
Ln 30, Col 1
Spaces: 4
UTF-8
LF
Python
3.10.8 64-bit
Layout: US
LinkedIn Learning


inheritance_start.py - python-obj
automatic-doodle-pjrjp5wgq5c677r.github.dev
EXPLORER
PYTHON-OBJECT-ORIENTED-...
.devcontainer
.github
.vscode
Finished
Ch 1
Ch 2
Ch 3
Ch 4
Start
Ch 1
Ch 2
abstract_start.py
composition_start.py
inheritance_star... M
interface_start.py
multiple_start.py
Ch 3
Ch 4
.gitignore
CONTRIBUTING.md
LICENSE
NOTICE
README.md
OUTLINE
TIMELINE
inheritance_start.py M
Start > Ch 2 > inheritance_start.py > ...
16 def __init__(self, title, author, pages, price):
17 super().__init__(title, price)
18 self.author = author
19 self.pages = pages
20
21
22 class Magazine(Periodical):
23 def __init__(self, title, publisher, price, period):
24 super().__init__(title, price, period, publisher)
Output (Ctrl+Shift+U)
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS 1
COMMENTS
bash - Ch 2
$ python inheritance_start.py
Aldous Huxley
New York Times Company
29.0 5.99 6.0
$ python inheritance_start.py
Aldous Huxley
New York Times Company
29.0 5.99 6.0
$
Codespaces
main*
Ln 30, Col 1
Spaces: 4
UTF-8
LF
Python
3.10.8 64-bit
Layout: US
LinkedIn Learning


Abstract base classes
Audio only
0:11 / 4:36
CC
1.25x

Abstract Classes in Python

An abstract class can be considered a blueprint for other classes. It allows you to create a set of methods that must be created within any child classes built from the abstract class.

A class that contains one or more abstract methods is called an **abstract class**. An **abstract method** is a method that has a declaration but does not have an implementation.

We use an abstract class while we are designing large functional units or when we want to provide a common interface for different implementations of a component.

Abstract Base Classes in Python

By defining an abstract base class, you can define a common **Application Program Interface(API)** for a set of subclasses. This capability is especially useful in situations where a third party is going to provide implementations, such as with plugins, but can also help you when working in a large team or with a large code base where keeping all classes in your mind is difficult or not possible.

Working on Python Abstract classes

By default, Python does not provide **abstract classes**. Python comes with a module that provides the base for defining A**bstract Base classes(ABC)** and that module name is **ABC**.

ABC works by decorating methods of the base class as an abstract and then registering concrete classes as implementations of the abstract base. A method becomes abstract when decorated with the keyword @abstractmethod.

Example 1:

This code defines an abstract base class called "**Polygon**" using the ABC (Abstract Base Class) module in Python. The "Polygon" class has an abstract method called "**noofsides**" that needs to be implemented by its subclasses.

There are four subclasses of "Polygon" defined: "Triangle," "Pentagon," "Hexagon," and "Quadrilateral." Each of these subclasses overrides the "noofsides" method and provides its own implementation by printing the number of sides it has.

In the driver code, instances of each subclass are created, and the "noofsides" method is called on each instance to display the number of sides specific to that shape.

Python3

```
# Python program showing
# abstract base class work
from abc import ABC, abstractmethod


class Polygon(ABC):

    @abstractmethod
    def noofsides(self):
        pass


class Triangle(Polygon):

    # overriding abstract method
    def noofsides(self):
        print("I have 3 sides")


class Pentagon(Polygon):

    # overriding abstract method
    def noofsides(self):
        print("I have 5 sides")


class Hexagon(Polygon):

    # overriding abstract method
    def noofsides(self):
        print("I have 6 sides")


class Quadrilateral(Polygon):

    # overriding abstract method
    def noofsides(self):
```

implementations, such as with plugins, but can also help you when working in a large team or with a large code base where keeping all classes in your mind is difficult or not possible.

Working on Python Abstract classes

By default, Python does not provide **abstract classes**. Python comes with a module that provides the base for defining A**bstract Base classes(ABC)** and that module name is **ABC**.

ABC works by decorating methods of the base class as an abstract and then registering concrete classes as implementations of the abstract base. A method becomes abstract when decorated with the keyword @abstractmethod.

Example 1:

This code defines an abstract base class called "**Polygon**" using the ABC (Abstract Base Class) module in Python. The "Polygon" class has an abstract method called "**noofsides**" that needs to be implemented by its subclasses.

There are four subclasses of "Polygon" defined: "Triangle," "Pentagon," "Hexagon," and "Quadrilateral." Each of these subclasses overrides the "noofsides" method and provides its own implementation by printing the number of sides it has.

In the driver code, instances of each subclass are created, and the "noofsides" method is called on each instance to display the number of sides specific to that shape.

Python3

```
# Python program showing
# abstract base class work
from abc import ABC, abstractmethod


class Polygon(ABC):

    @abstractmethod
    def noofsides(self):
        pass


class Triangle(Polygon):

    # overriding abstract method
    def noofsides(self):
        print("I have 3 sides")


class Pentagon(Polygon):

    # overriding abstract method
    def noofsides(self):
        print("I have 5 sides")


class Hexagon(Polygon):

    # overriding abstract method
    def noofsides(self):
        print("I have 6 sides")


class Quadrilateral(Polygon):

    # overriding abstract method
    def noofsides(self):
        print("I have 4 sides")


# Driver code
R = Triangle()
R.noofsides()

K = Quadrilateral()
K.noofsides()

R = Pentagon()
R.noofsides()

K = Hexagon()
K.noofsides()
```

Output

```
I have 3 sides
I have 4 sides
I have 5 sides
I have 6 sides
```

Example 2:

```
K = Quadrilateral()
K.noofsides()

R = Pentagon()
R.noofsides()

K = Hexagon()
K.noofsides()
```

Output

```
I have 3 sides
I have 4 sides
I have 5 sides
I have 6 sides
```

Example 2:

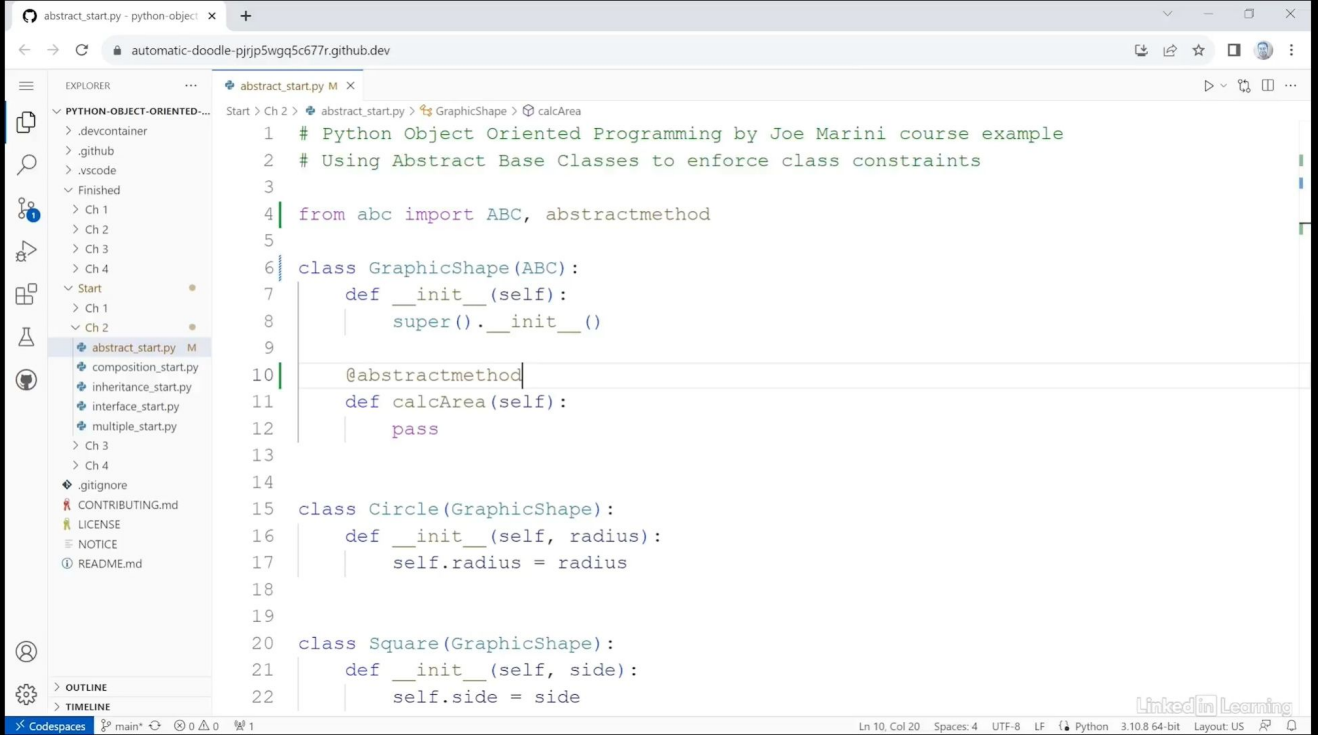
Here, This code defines an abstract base class called "Animal" using the ABC (Abstract Base Class) module in Python. The "Animal" class has a non-abstract method called "move" that does not have any implementation. There are four subclasses of "Animal" defined: "Human," "Snake," "Dog," and "Lion." Each of these subclasses overrides the "move" method and provides its own implementation by printing a specific movement characteristic.

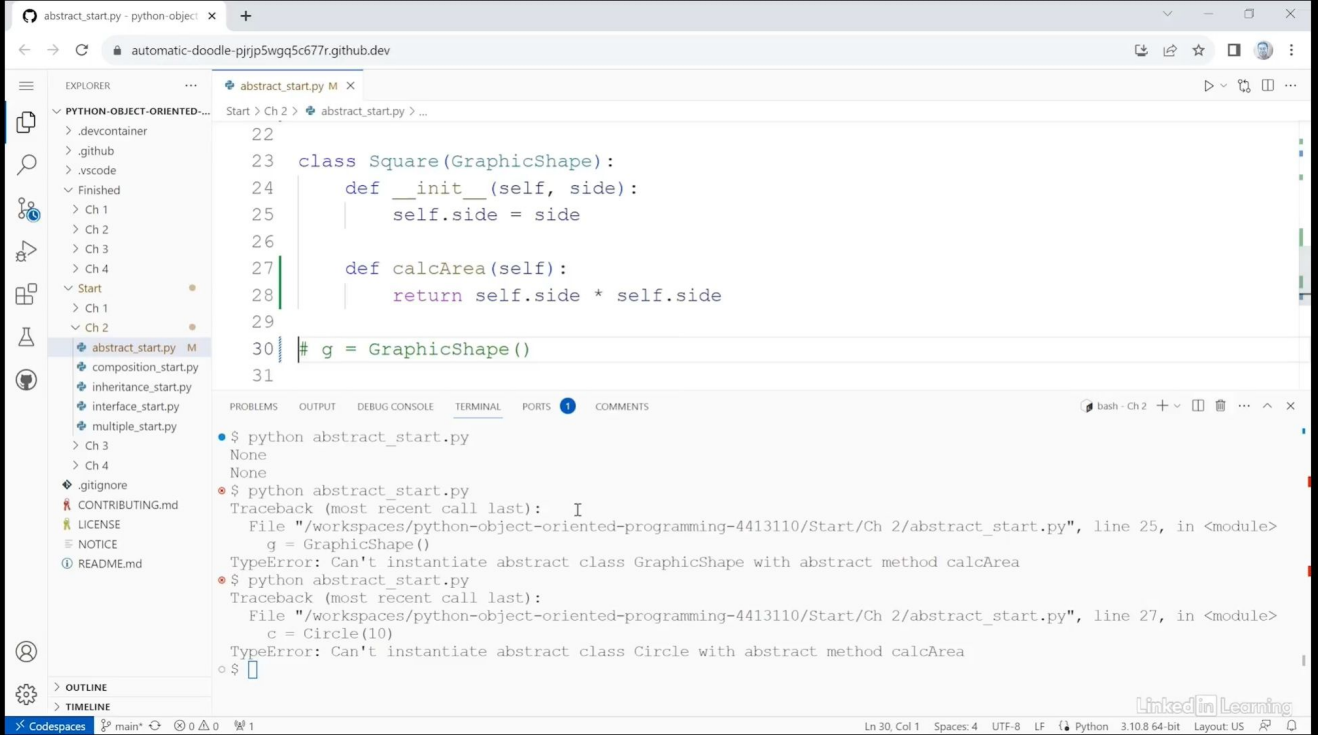
Python3

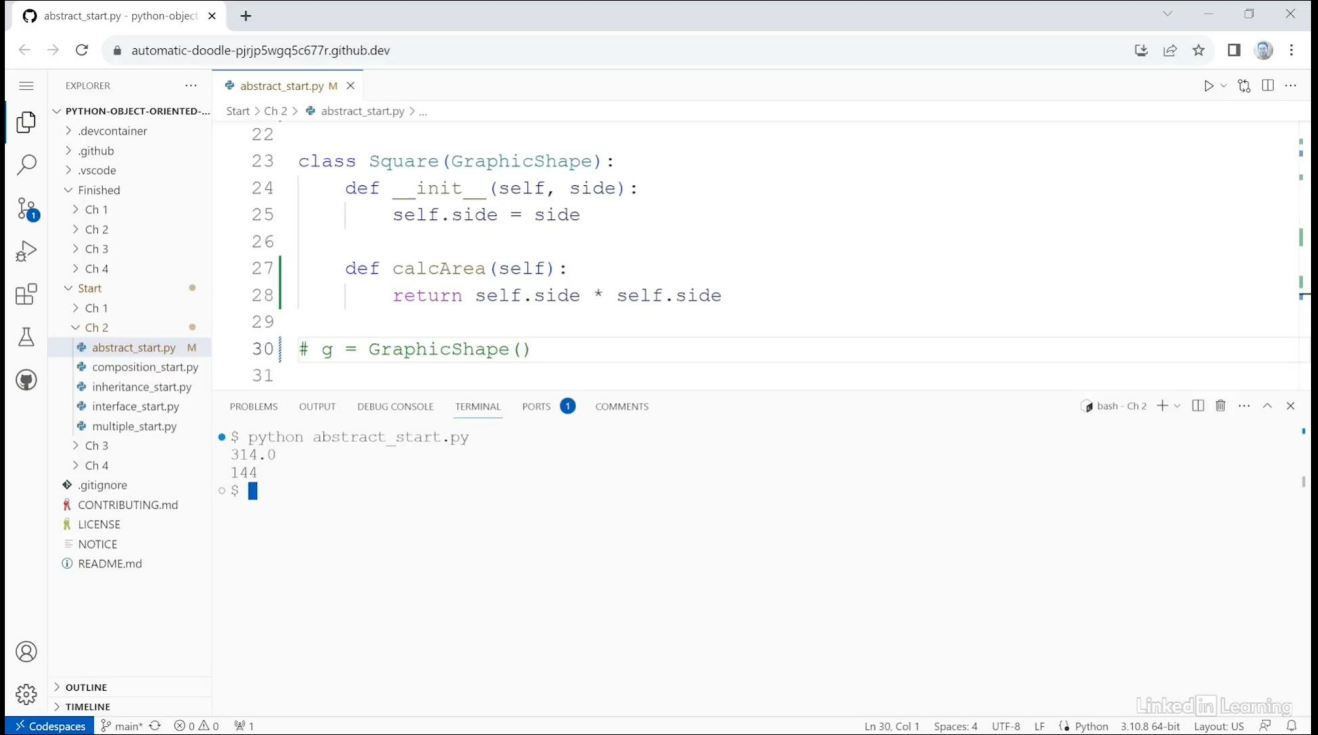
```
# Python program showing
# abstract base class work
from abc import ABC, abstractmethod


class Animal(ABC):

    def move(self):
        pass

class Human(Animal):

    def move(self):
        print("I can walk and run")

class Snake(Animal):

    def move(self):
        print("I can crawl")

class Dog(Animal):

    def move(self):
        print("I can bark")

class Lion(Animal):

    def move(self):
        print("I can roar")

# Driver code
R = Human()
R.move()

K = Snake()
K.move()

R = Dog()
R.move()

K = Lion()
K.move()
```

Output

```
I can walk and run
I can crawl
I can bark
I can roar
```


abstract_start.py - python-object
automatic-doodle-pjrjp5wgq5c677r.github.dev
EXPLORER
abstract_start.py M
PYTHON-OBJECT-ORIENTED-...
.devcontainer
.github
.vscode
Finished
Ch 1
Ch 2
Ch 3
Ch 4
Start
Ch 1
Ch 2
abstract_start.py M
composition_start.py
inheritance_start.py
interface_start.py
multiple_start.py
Ch 3
Ch 4
.gitignore
CONTRIBUTING.md
LICENSE
NOTICE
README.md
OUTLINE
TIMELINE
Start > Ch 2 > abstract_start.py > GraphicShape > calcArea
1 # Python Object Oriented Programming by Joe Marini course example
2 # Using Abstract Base Classes to enforce class constraints
3
4 from abc import ABC, abstractmethod
5
6 class GraphicShape(ABC):
7 def __init__(self):
8 super().__init__()
9
10 @abstractmethod
11 def calcArea(self):
12 pass
13
14
15 class Circle(GraphicShape):
16 def __init__(self, radius):
17 self.radius = radius
18
19
20 class Square(GraphicShape):
21 def __init__(self, side):
22 self.side = side
Codespaces
main*
Ln 10, Col 20
Spaces: 4
UTF-8
LF
Python
3.10.8 64-bit
Layout: US
LinkedIn Learning


abstract_start.py - python-object
automatic-doodle-pjrjp5wgq5c677r.github.dev
EXPLORER
PYTHON-OBJECT-ORIENTED-...
.devcontainer
.github
.vscode
Finished
Ch 1
Ch 2
Ch 3
Ch 4
Start
Ch 1
Ch 2
abstract_start.py M
composition_start.py
inheritance_start.py
interface_start.py
multiple_start.py
Ch 3
Ch 4
.gitignore
CONTRIBUTING.md
LICENSE
NOTICE
README.md
OUTLINE
TIMELINE
abstract_start.py M
Start > Ch 2 > abstract_start.py > ...
22
23 class Square(GraphicShape):
24 def __init__(self, side):
25 self.side = side
26
27 def calcArea(self):
28 return self.side * self.side
29
30 # g = GraphicShape()
31
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS 1
COMMENTS
bash - Ch 2
$ python abstract_start.py
None
None
$ python abstract_start.py
Traceback (most recent call last):
File "/workspaces/python-object-oriented-programming-4413110/Start/Ch 2/abstract_start.py", line 25, in <module>
g = GraphicShape()
TypeError: Can't instantiate abstract class GraphicShape with abstract method calcArea
$ python abstract_start.py
Traceback (most recent call last):
File "/workspaces/python-object-oriented-programming-4413110/Start/Ch 2/abstract_start.py", line 27, in <module>
c = Circle(10)
TypeError: Can't instantiate abstract class Circle with abstract method calcArea
$
Codespaces
main*
Ln 30, Col 1
Spaces: 4
UTF-8
LF
Python
3.10.8 64-bit
Layout: US
LinkedIn Learning

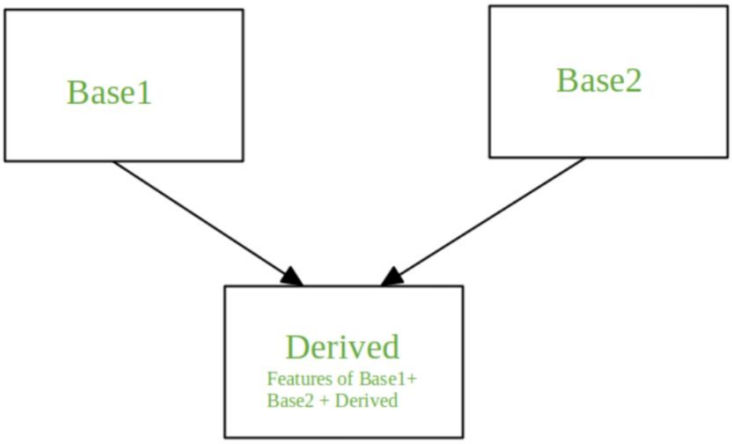

abstract_start.py - python-object
automatic-doodle-pjrjp5wgq5c677r.github.dev
EXPLORER
abstract_start.py M
PYTHON-OBJECT-ORIENTED-...
.devcontainer
.github
.vscode
Finished
Ch 1
Ch 2
Ch 3
Ch 4
Start
Ch 1
Ch 2
abstract_start.py M
composition_start.py
inheritance_start.py
interface_start.py
multiple_start.py
Ch 3
Ch 4
.gitignore
CONTRIBUTING.md
LICENSE
NOTICE
README.md
Start > Ch 2 > abstract_start.py > ...
22
23 class Square(GraphicShape):
24 def __init__(self, side):
25 self.side = side
26
27 def calcArea(self):
28 return self.side * self.side
29
30 # g = GraphicShape()
31
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS 1
COMMENTS
bash - Ch 2
$ python abstract_start.py
314.0
144
$
OUTLINE
TIMELINE
Codespaces
main*
Ln 30, Col 1
Spaces: 4
UTF-8
LF
Python
3.10.8 64-bit
Layout: US
LinkedIn Learning

Multiple Inheritance in Python

Inheritance is the mechanism to achieve the re-usability of code as one class(child class) can derive the properties of another class(parent class). It also provides transitivity ie. if class C inherits from P then all the sub-classes of C would also inherit from P.

Multiple Inheritance

When a class is derived from more than one base class it is called multiple Inheritance. The derived class inherits all the features of the base case.

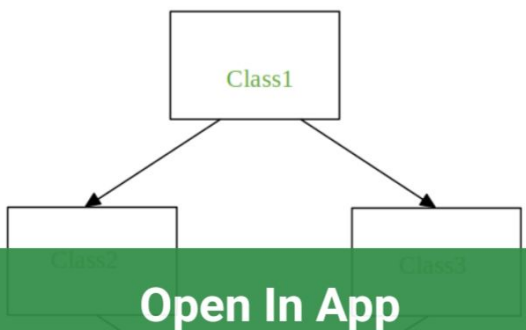


Syntax:

```
Class Base1:
        Body of the class

Class Base2:
     Body of the class

Class Derived(Base1, Base2):
     Body of the class
```

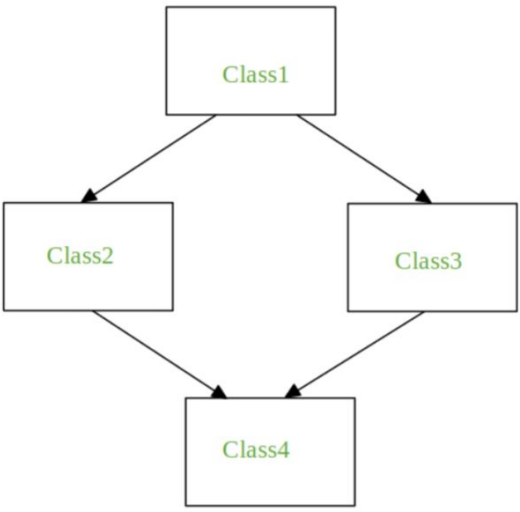
In the coming section, we will see the problem faced during multiple inheritance and how to tackle it with the help of examples.

The Diamond Problem



faced during multiple inheritance and how to tackle it with the help of examples.

The Diamond Problem



It refers to an ambiguity that arises when two classes Class2 and Class3 inherit from a superclass Class1 and class Class4 inherits from both Class2 and Class3. If there is a method **“m”** which is an overridden method in one of Class2 and Class3 or both then the ambiguity arises which of the method “m” Class4 should inherit.

It refers to an ambiguity that arises when two classes Class2 and Class3 inherit from a superclass Class1 and class Class4 inherits from both Class2 and Class3. If there is a method **"m"** which is an overridden method in one of Class2 and Class3 or both then the ambiguity arises which of the method "m" Class4 should inherit.

When the method is overridden in both classes

Python3

```
# Python Program to depict multiple inheritance
# when method is overridden in both classes
class Class1:
    def m(self):
        print("In Class1")

class Class2(Class1):
    def m(self):
        print("In Class2")
class Class3(Class1):
    def m(self):
        print("In Class3")

class Class4(Class2, Class3):
    pass

obj = Class4()
obj.m()
```

Output:

```
In Class2
```

Note: When you call obj.m() (m on the instance of Class4) the output is In Class2. If Class4 is declared as Class4(Class3, Class2) then the output of obj.m() will be In Class3.

When the method is overridden in one of the classes

Python3

```
# Python Program to depict multiple inheritance
# when method is overridden in one of the classes
class Class1:
    def m(self):
        print("In Class1")

class Class2(Class1):
    pass
class Class3(Class1):
    def m(self):
        print("In Class3")

class Class4(Class2, Class3):
    pass
obj = Class4()
obj.m()
```

Output:

```
In Class3
```

When every class defines the same method

When every class defines the same method

Python3

```
# Python Program to depict multiple inheritance
# when every class defines the same method

class Class1:
    def m(self):
        print("In Class1")

class Class2(Class1):
    def m(self):
        print("In Class2")

class Class3(Class1):
    def m(self):
         print("In Class3")

class Class4(Class2, Class3):
    def m(self):
        print("In Class4")

obj = Class4()
obj.m()
Class2.m(obj)
Class3.m(obj)
Class1.m(obj)
```

Output:

```
In Class4
In Class2
In Class3
In Class1
```

The output of the method obj.m() in the above code is **In Class4**. The method "m" of Class4 is executed. To execute the method "m" of the other classes it can be done using the class names.
Now, to call the method m for Class1, Class2, Class3 directly from the method "m" of the Class4 see the below example

Python3

```
# Python Program to depict multiple inheritance
# when we try to call the method m for Class1,
# Class2, Class3 from the method m of Class4

class Class1:
    def m(self):
        print("In Class1")

class Class2(Class1):
    def m(self):
        print("In Class2")

class Class3(Class1):
    def m(self):
        print("In Class3")

class Class4(Class2, Class3):
    def m(self):
        print("In Class4")
        Class2.m(self)
        Class3.m(self)
        Class1.m(self)

obj = Class4()
obj.m()
```

Output:

```
In Class4
In Class2
In Class3
In Class1
```

To call "m" of Class1 from both "m" of Class2 and "m" of Class3 instead of Class4 is shown below:

```
    def m(self):
        print("In Class3")
        Class1.m(self)

class Class4(Class2, Class3):
    def m(self):
        print("In Class4")
        Class2.m(self)
        Class3.m(self)

obj = Class4()
obj.m()
```

Output:

```
In Class4
In Class2
In Class1
In Class3
In Class1
```

The output of the above code has one problem associated with it, the method m of Class1 is called twice. Python provides a solution to the above problem with the help of the super() function. Let's see how it works.

The super Function

Python3

```
# Python program to demonstrate
# super()

class Class1:
    def m(self):
        print("In Class1")

class Class2(Class1):
    def m(self):
        print("In Class2")
        super().m()

class Class3(Class1):
    def m(self):
        print("In Class3")
        super().m()

class Class4(Class2, Class3):
    def m(self):
        print("In Class4")
        super().m()

obj = Class4()
obj.m()
```

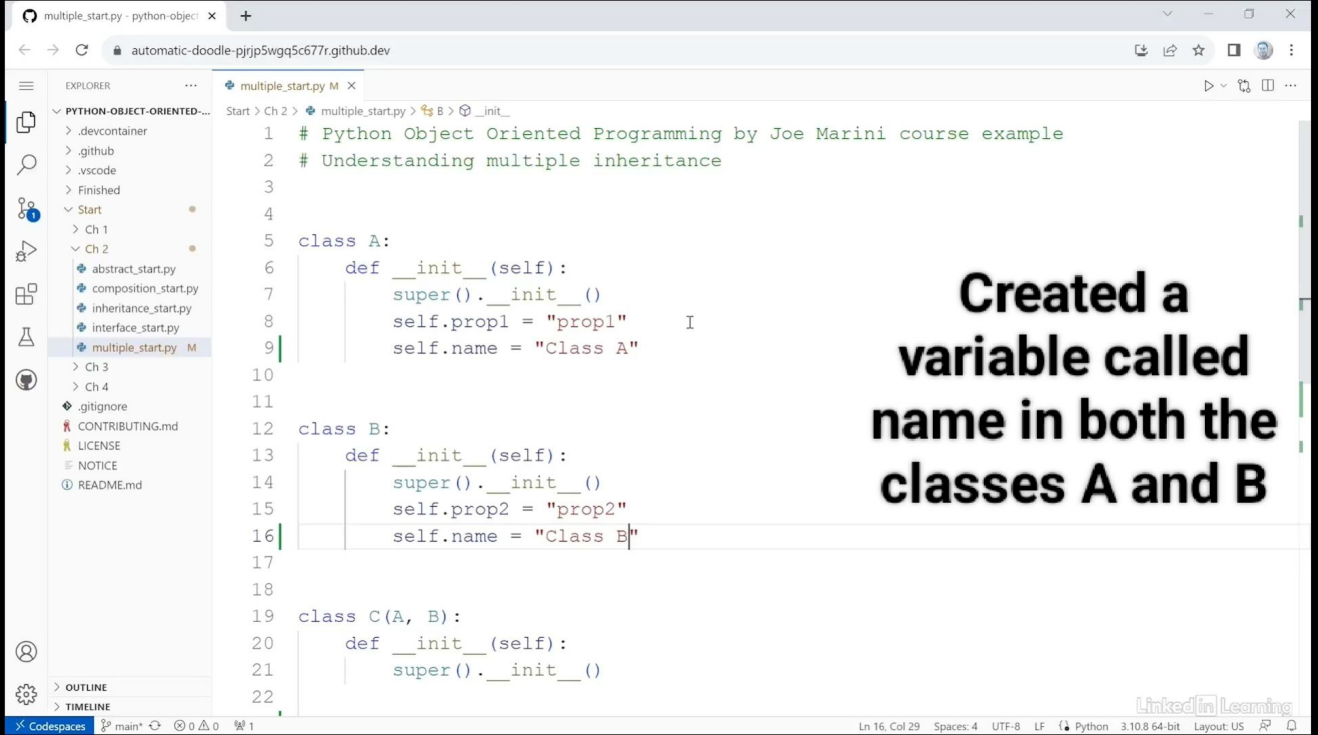
Output:

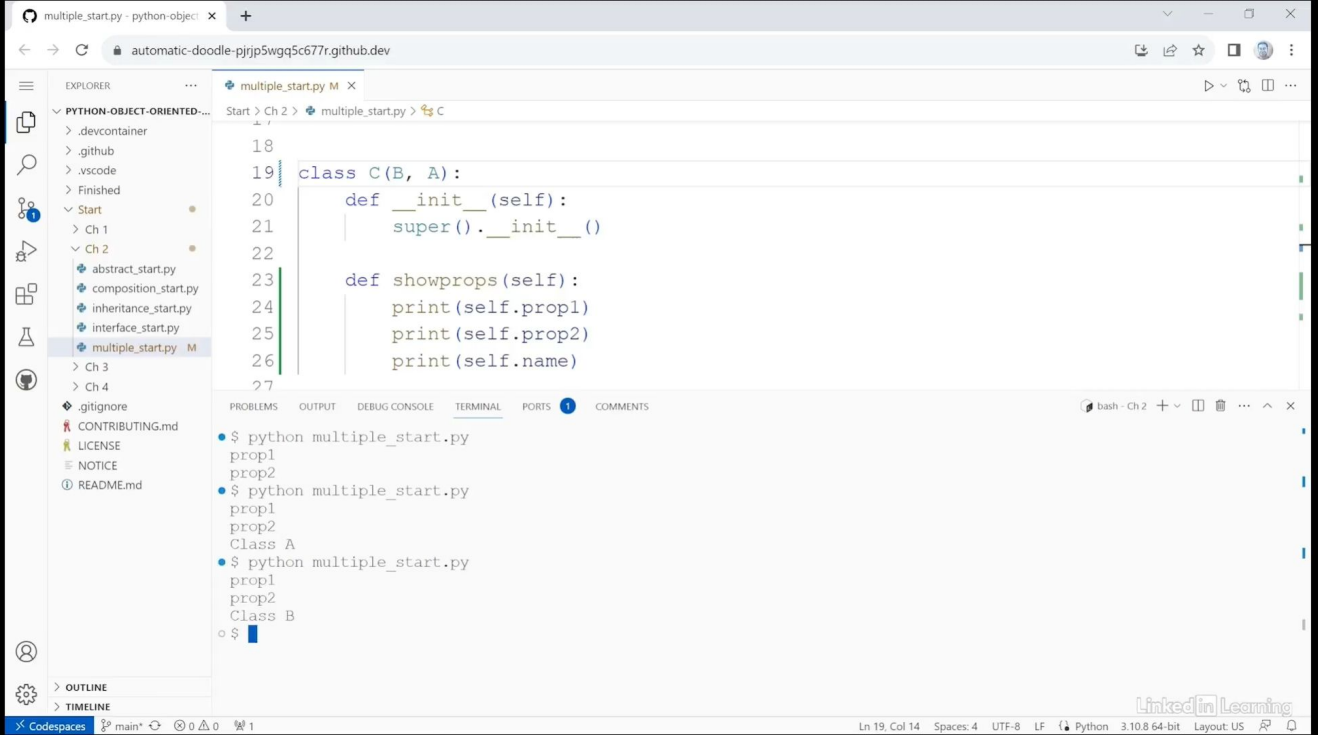
```
In Class4
In Class2
In Class3
In Class1
```

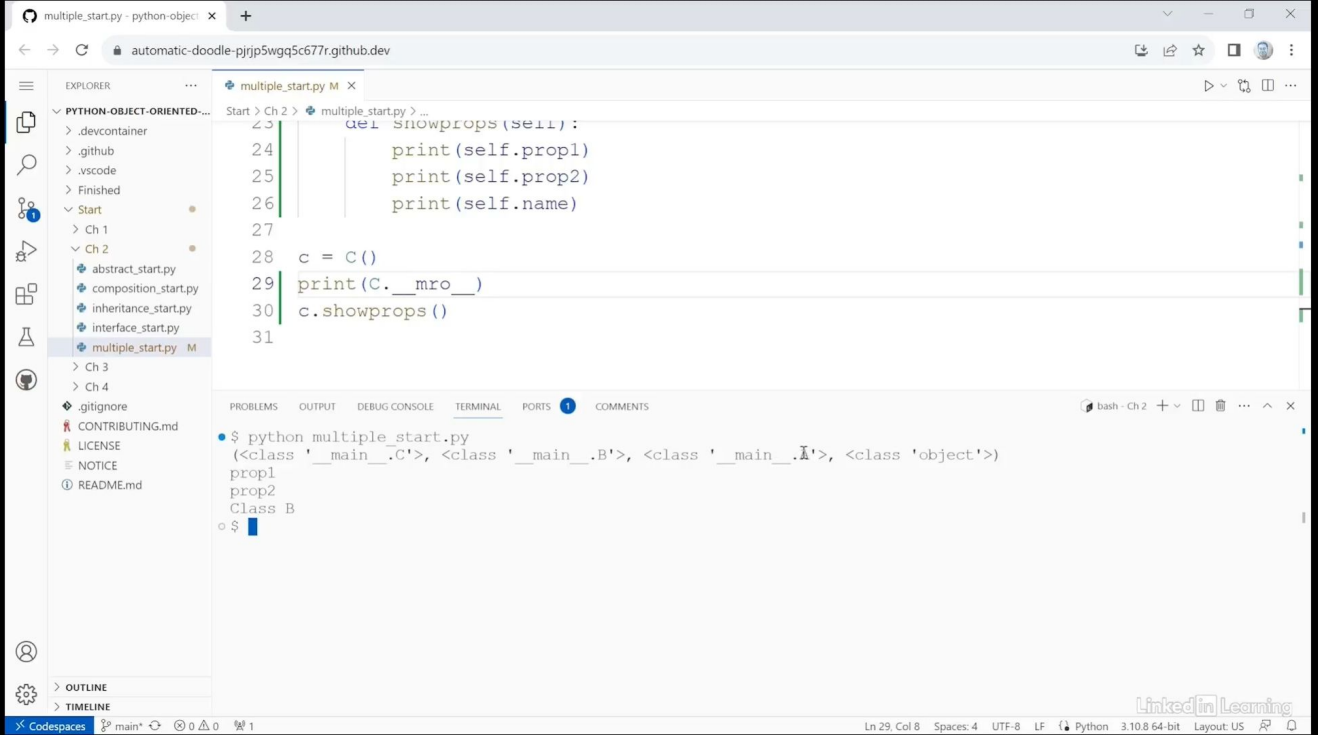
Super() is generally used with the __init__ function when the instances are initialized. The super function comes to a conclusion, on which method to call with the help of the **method resolution order (MRO)**.

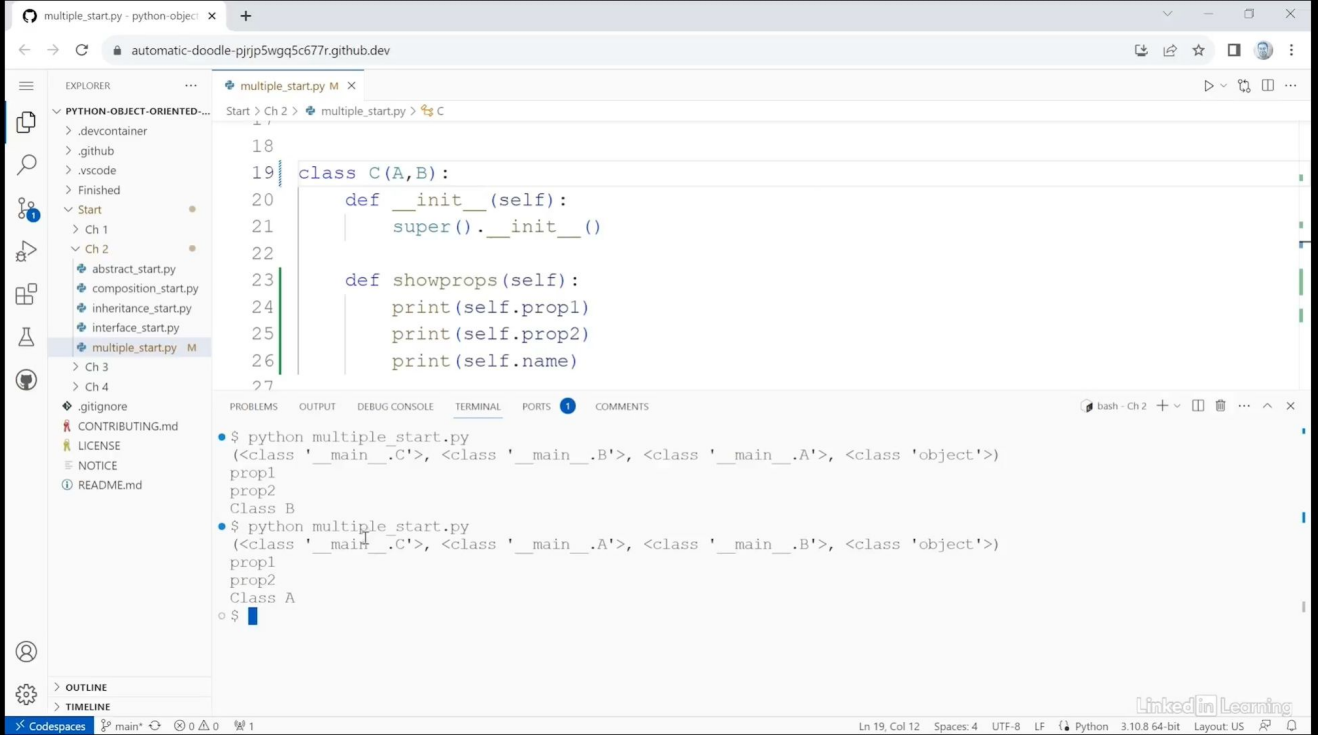
Method resolution order:

In Python, every class whether built-in or user-defined is derived from the object class and all the objects are instances of the class object. Hence, the object class is the base class for all the other classes.
In the case of multiple inheritance, a given attribute is first searched in the current class if it's not found then it's searched in the parent classes. The parent classes are searched in a left-right fashion and each class is searched once.


multiple_start.py - python-object
automatic-doodle-pjrjp5wgq5c677r.github.dev
EXPLORER
multiple_start.py M
Start > Ch 2 > multiple_start.py > B > __init__
PYTHON-OBJECT-ORIENTED-...
.devcontainer
.github
.vscode
Finished
Start
Ch 1
Ch 2
abstract_start.py
composition_start.py
inheritance_start.py
interface_start.py
multiple_start.py M
Ch 3
Ch 4
.gitignore
CONTRIBUTING.md
LICENSE
NOTICE
README.md
OUTLINE
TIMELINE
1 # Python Object Oriented Programming by Joe Marini course example
2 # Understanding multiple inheritance
3
4
5 class A:
6 def __init__(self):
7 super().__init__()
8 self.prop1 = "prop1"
9 self.name = "Class A"
10
11
12 class B:
13 def __init__(self):
14 super().__init__()
15 self.prop2 = "prop2"
16 self.name = "Class B"
17
18
19 class C(A, B):
20 def __init__(self):
21 super().__init__()
22
Created a variable called name in both the classes A and B
Codespaces
main*
Ln 16, Col 29
Spaces: 4
UTF-8
LF
Python
3.10.8 64-bit
Layout: US
LinkedIn Learning


multiple_start.py - python-object
automatic-doodle-pjrjp5wgq5c677r.github.dev
EXPLORER
PYTHON-OBJECT-ORIENTED-...
.devcontainer
.github
.vscode
Finished
Start
Ch 1
Ch 2
abstract_start.py
composition_start.py
inheritance_start.py
interface_start.py
multiple_start.py M
Ch 3
Ch 4
.gitignore
CONTRIBUTING.md
LICENSE
NOTICE
README.md
multiple_start.py M
Start > Ch 2 > multiple_start.py > C
18
19 class C(B, A):
20 def __init__(self):
21 super().__init__()
22
23 def showprops(self):
24 print(self.prop1)
25 print(self.prop2)
26 print(self.name)
27
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS 1
COMMENTS
bash - Ch 2
$ python multiple_start.py
prop1
prop2
$ python multiple_start.py
prop1
prop2
Class A
$ python multiple_start.py
prop1
prop2
Class B
$
OUTLINE
TIMELINE
Codespaces
main*
Ln 19, Col 14
Spaces: 4
UTF-8
LF
Python
3.10.8 64-bit
Layout: US
LinkedIn Learning


multiple_start.py - python-object
automatic-doodle-pjrjp5wgq5c677r.github.dev
EXPLORER
PYTHON-OBJECT-ORIENTED-...
.devcontainer
.github
.vscode
Finished
Start
Ch 1
Ch 2
abstract_start.py
composition_start.py
inheritance_start.py
interface_start.py
multiple_start.py M
Ch 3
Ch 4
.gitignore
CONTRIBUTING.md
LICENSE
NOTICE
README.md
multiple_start.py M
Start > Ch 2 > multiple_start.py > ...
def showprops(self):
print(self.prop1)
print(self.prop2)
print(self.name)
c = C()
print(C.__mro__)
c.showprops()
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS 1
COMMENTS
bash - Ch 2
$ python multiple_start.py
(<class '__main__.C'>, <class '__main__.B'>, <class '__main__.A'>, <class 'object'>)
prop1
prop2
Class B
$
OUTLINE
TIMELINE
Codespaces
main*
Ln 29, Col 8
Spaces: 4
UTF-8
LF
Python
3.10.8 64-bit
Layout: US
LinkedIn Learning


multiple_start.py - python-object
automatic-doodle-pjrjp5wgq5c677r.github.dev
EXPLORER
PYTHON-OBJECT-ORIENTED-...
.devcontainer
.github
.vscode
Finished
Start
Ch 1
Ch 2
abstract_start.py
composition_start.py
inheritance_start.py
interface_start.py
multiple_start.py M
Ch 3
Ch 4
.gitignore
CONTRIBUTING.md
LICENSE
NOTICE
README.md
multiple_start.py M
Start > Ch 2 > multiple_start.py > C
18
19 class C(A,B):
20 def __init__(self):
21 super().__init__()
22
23 def showprops(self):
24 print(self.prop1)
25 print(self.prop2)
26 print(self.name)
27
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS 1
COMMENTS
bash - Ch 2
$ python multiple_start.py
(<class '__main__.C'>, <class '__main__.B'>, <class '__main__.A'>, <class 'object'>)
prop1
prop2
Class B
$ python multiple_start.py
(<class '__main__.C'>, <class '__main__.A'>, <class '__main__.B'>, <class 'object'>)
prop1
prop2
Class A
$
OUTLINE
TIMELINE
Codespaces
main*
Ln 19, Col 12
Spaces: 4
UTF-8
LF
Python
3.10.8 64-bit
Layout: US
LinkedIn Learning

In software engineering, an interface is a software architectural pattern. An interface is like a class but its methods just have prototype signature definition without any body to implement. The recommended functionality needs to be implemented by a concrete class.

In languages like Java, there is interface keyword which makes it easy to define an interface. Python doesn't have it or any similar keyword. Hence the same ABC class and @abstractmethod decorator is used as done in an abstract class.

An abstract class and interface appear similar in Python. The only difference in two is that the abstract class may have some non-abstract methods, while all methods in interface must be abstract, and the implementing class must override all the abstract methods.

Example

```
from abc import ABC, abstractmeth
class demoInterface(ABC):
    @abstractmethod
    def method1(self):
```

Example

```
from abc import ABC, abstractmeth
class demoInterface(ABC):
   @abstractmethod
   def method1(self):
      print ("Abstract method1")
      return

   @abstractmethod
   def method2(self):
      print ("Abstract method1")
      return
```

The above interface has two abstract methods. As in abstract class, we cannot instantiate an interface.

```
  obj = demoInterface()
      ^^^^^^^^^^^^^^^^
TypeError: Can't instantiate abstract class demo
```

Let us provide a class that implements both the abstract methods. If doesn't contain implementations of all abstract methods, Python shows following error −

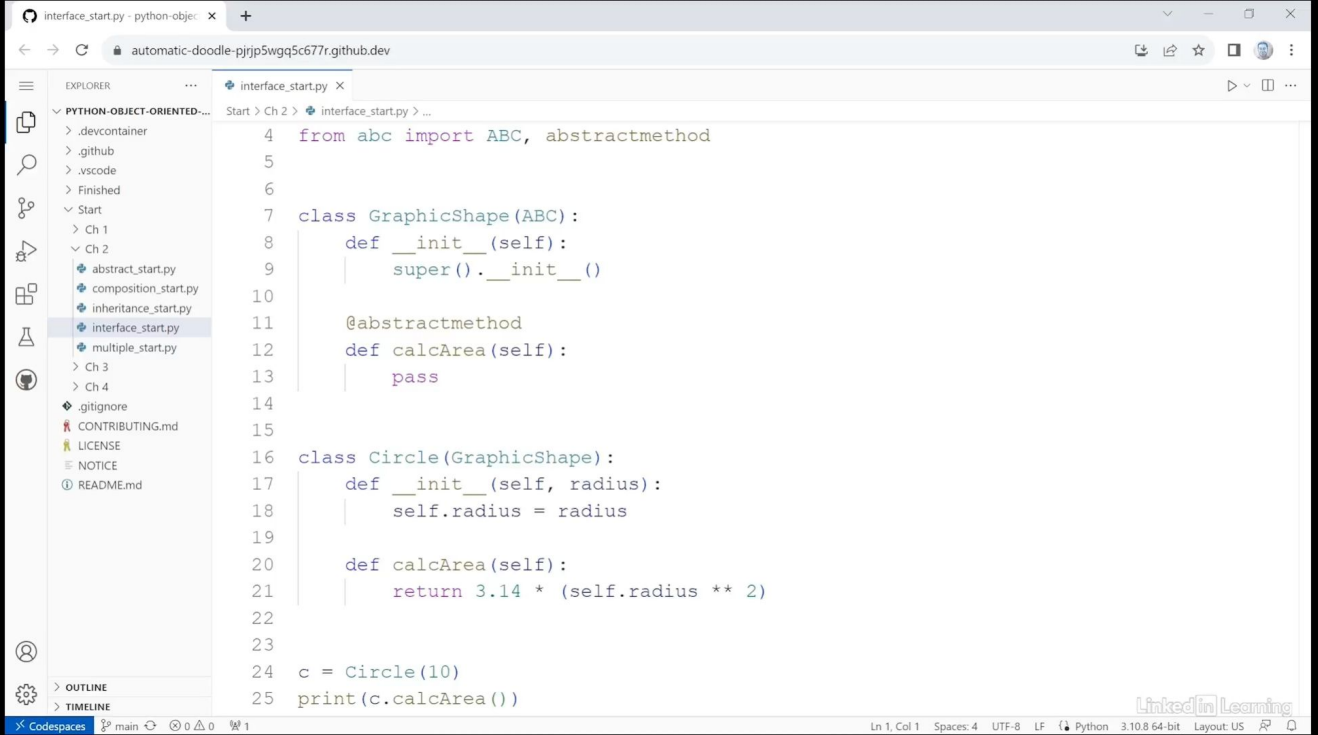
The following class implements both methods –

```
class concreteclass(demoInterface
   def method1(self):
      print ("This is method1")
      return

   def method2(self):
      print ("This is method2")
      return

obj = concreteclass()
obj.method1()
obj.method2()
```

Output

When you execute this code, it will produce the following output –

```
This is method1
This is method2
```


interface_start.py - python-objec
automatic-doodle-pjrjp5wgq5c677r.github.dev
EXPLORER
interface_start.py
PYTHON-OBJECT-ORIENTED-...
.devcontainer
.github
.vscode
Finished
Start
Ch 1
Ch 2
abstract_start.py
composition_start.py
inheritance_start.py
interface_start.py
multiple_start.py
Ch 3
Ch 4
.gitignore
CONTRIBUTING.md
LICENSE
NOTICE
README.md
OUTLINE
TIMELINE
Start > Ch 2 > interface_start.py > ...
4 from abc import ABC, abstractmethod
5
6
7 class GraphicShape(ABC):
8 def __init__(self):
9 super().__init__()
10
11 @abstractmethod
12 def calcArea(self):
13 pass
14
15
16 class Circle(GraphicShape):
17 def __init__(self, radius):
18 self.radius = radius
19
20 def calcArea(self):
21 return 3.14 * (self.radius ** 2)
22
23
24 c = Circle(10)
25 print(c.calcArea())
Codespaces
main
Ln 1, Col 1
Spaces: 4
UTF-8
LF
Python
3.10.8 64-bit
Layout: US
LinkedIn Learning

Class method vs Static method in Python

Last Updated : 30 Dec, 2022

In this article, we will cover the basic **difference between the class method vs Static method** in Python and **when to use the class method and static method** in python.

What is Class Method in Python?

The @classmethod decorator is a built-in function decorator that is an expression that gets evaluated after your function is defined. The result of that evaluation shadows your function definition. A class method receives the class as an implicit first argument, just like an instance method receives the instance

Syntax Python Class Method:

```
class C(object):
    @classmethod
    def fun(cls, arg1, arg2, ...):
        ....
fun: function that needs to be converted into a class method
returns: a class method for function.
```

- A class method is a method that is bound to the class and not the object of the class.
- They have the access to the state of the class as it takes a class parameter that points to the class and not the object instance.
- It can modify a class state that would apply across all the instances of the class. For example, it can modify a class variable that will be applicable to all the instances.

What is the Static Method in Python?

A static method does not receive an implicit first argument. A static method is also a method that is bound to the class and not the object of the class. This method can't access or modify the class state. It is present in a class because it makes sense for the method to be present in class.

Syntax Python Static Method:

```
class C(object):
    @staticmethod
    def fun(arg1, arg2, ...):
        ...
returns: a static method for function fun.
```

Class method vs Static Method

The difference between the Class method and the static method is:

- A class method takes cls as the first parameter while a static method needs no specific parameters.
- A class method can access or modify the class state while a static method can't access or modify it.
- In general, static methods know nothing about the class state. They are utility-type methods that take some parameters and work upon those parameters. On the other hand class methods must have class as a parameter.

What is the Static Method in Python?

A static method does not receive an implicit first argument. A static method is also a method that is bound to the class and not the object of the class. This method can't access or modify the class state. It is present in a class because it makes sense for the method to be present in class.

Syntax Python Static Method:

```
class C(object):
    @staticmethod
    def fun(arg1, arg2, ...):
        ...
returns: a static method for function fun.
```

Class method vs Static Method

The difference between the Class method and the static method is:

- A class method takes cls as the first parameter while a static method needs no specific parameters.
- A class method can access or modify the class state while a static method can't access or modify it.
- In general, static methods know nothing about the class state. They are utility-type methods that take some parameters and work upon those parameters. On the other hand class methods must have class as a parameter.
- We use @classmethod decorator in python to create a class method and we use @staticmethod decorator to create a static method in python.

When to use the class or static method?

- We generally use the class method to create factory methods. Factory methods return class objects (similar to a constructor) for different use cases.
- We generally use static methods to create utility functions.

How to define a class method and a static method?

To define a class method in python, we use @classmethod decorator, and to define a static method we use @staticmethod decorator.
Let us look at an example to understand the difference between both of them. Let us say we want to create a class Person. Now, python doesn't support method overloading like C++ or Java so we use class methods to create factory methods. In the below example we use a class method to create a person object from birth year.

As explained above we use static methods to create utility functions. In the below example we use a static method to check if a person is an adult or not.

One simple Example :

class method:

Python3

```
class MyClass:
    def __init__(self, value):
        self.value = value

    def get_value(self):
        return self.value

# Create an instance of MyClass
obj = MyClass(10)

# Call the get_value method on the instance
print(obj.get_value())  # Output: 10
```

Class Polymorphism

Polymorphism is often used in Class methods, where we can have multiple classes with the same method name.

For example, say we have three classes: `Car`, `Boat`, and `Plane`, and they all have a method called `move()`:

Example

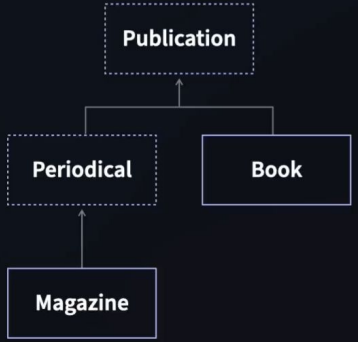
Different classes with the same method:

```
class Car:
  def __init__(self, brand, model):
    self.brand = brand
    self.model = model

  def move(self):
    print("Drive!")

class Boat:
  def __init__(self, brand, model):
    self.brand = brand
    self.model = model

  def move(self):
    print("Sail!")

class Plane:
  def __init__(self, brand, model):
    self.brand = brand
    self.model = model

  def move(self):
    print("Fly!")

car1 = Car("Ford", "Mustang")       #Create a Car class
boat1 = Boat("Ibiza", "Touring 20") #Create a Boat class
plane1 = Plane("Boeing", "747")     #Create a Plane class

for x in (car1, boat1, plane1):
  x.move()
```

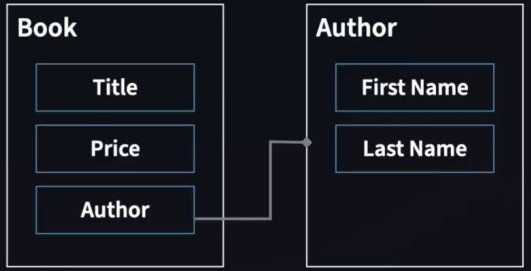
Try it Yourself »

Using Composition

Inheritance



Composition



Your Answers

Question **1** of **6**

In the case of multiple inheritance, if two base classes have methods with the same name, and a derived class calls this method, which method will be executed?

You are correct

The method of the first base class listed in the inheritance chain will be executed.

Next chapter

Chapter Quiz

Your Answers

Question **2** of **6**

What is multiple inheritance in Python?

You are correct

It is the ability to inherit from multiple classes simultaneously.

Next chapter

Your Answers

Question **3** of **6**

In Python, how do you define an Abstract Base Class using the `abc` module?

You are correct

by creating a class that inherits from the abc.ABC base class

Next chapter

Your Answers

Question **4** of **6**

What is object composition in Python, and how does it differ from inheritance?

You are correct

Object composition is a way to create new objects by combining them using references to other objects, while inheritance is a way to create new classes by inheriting attributes and methods from other classes.

Next chapter

Your Answers

Question **5** of **6**

What is the primary purpose of creating and using an interface?

You are correct

to enforce that classes inheriting from the interface must implement certain methods

Next chapter

Chapter Quiz

Your Answers

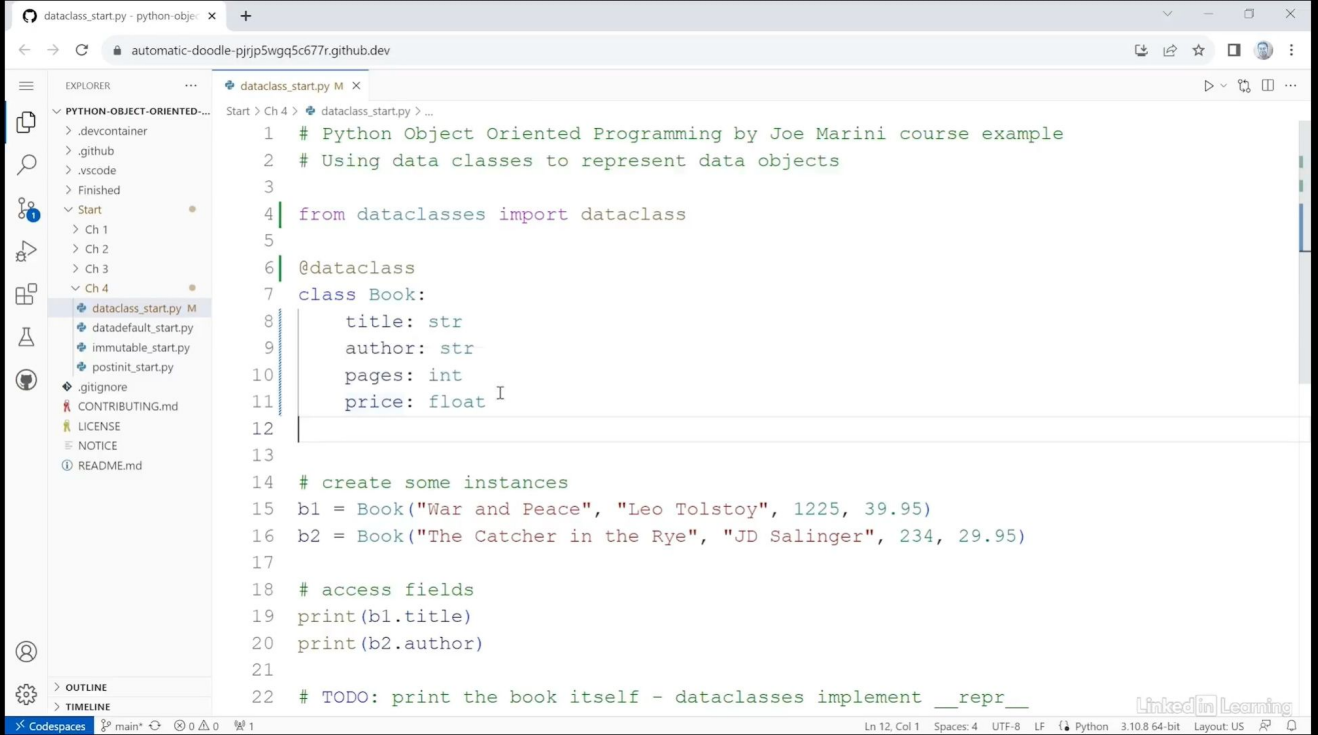
Question **6** of **6**

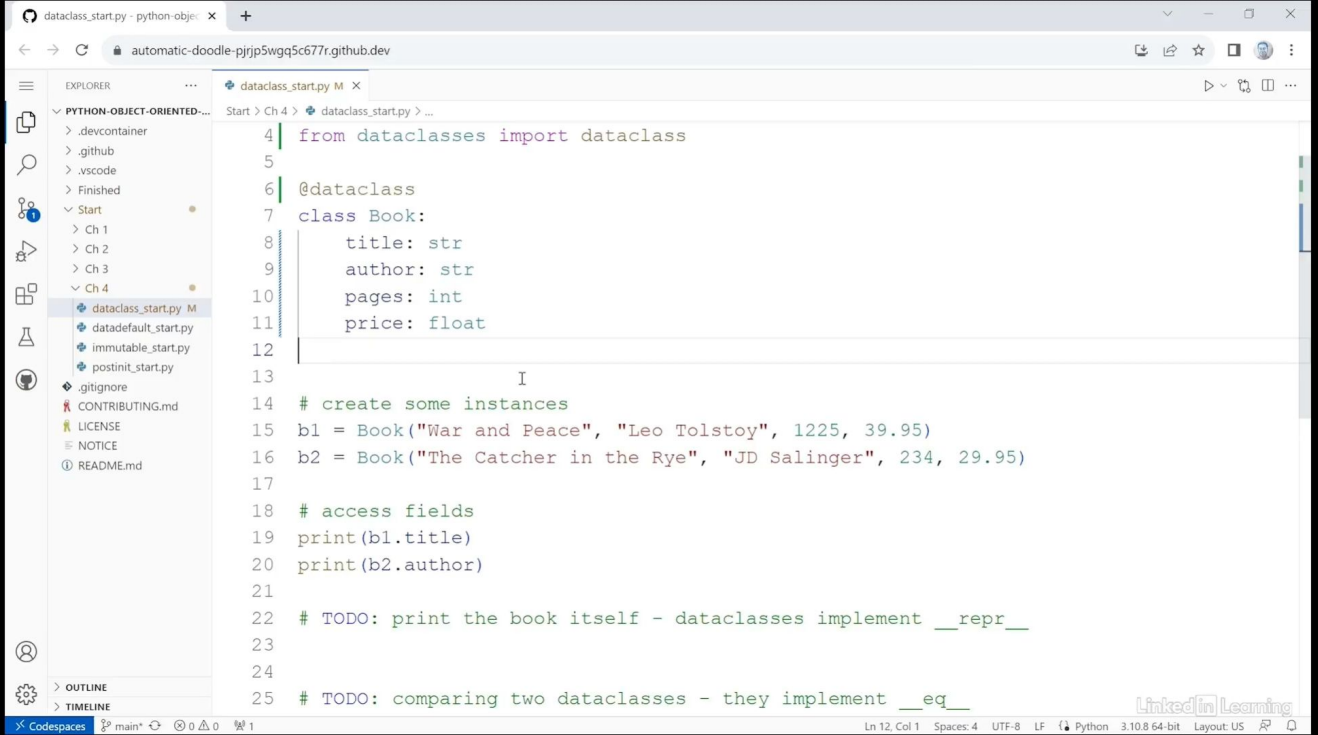
What is the correct syntax for calling a base class method from another Python method in a subclass?

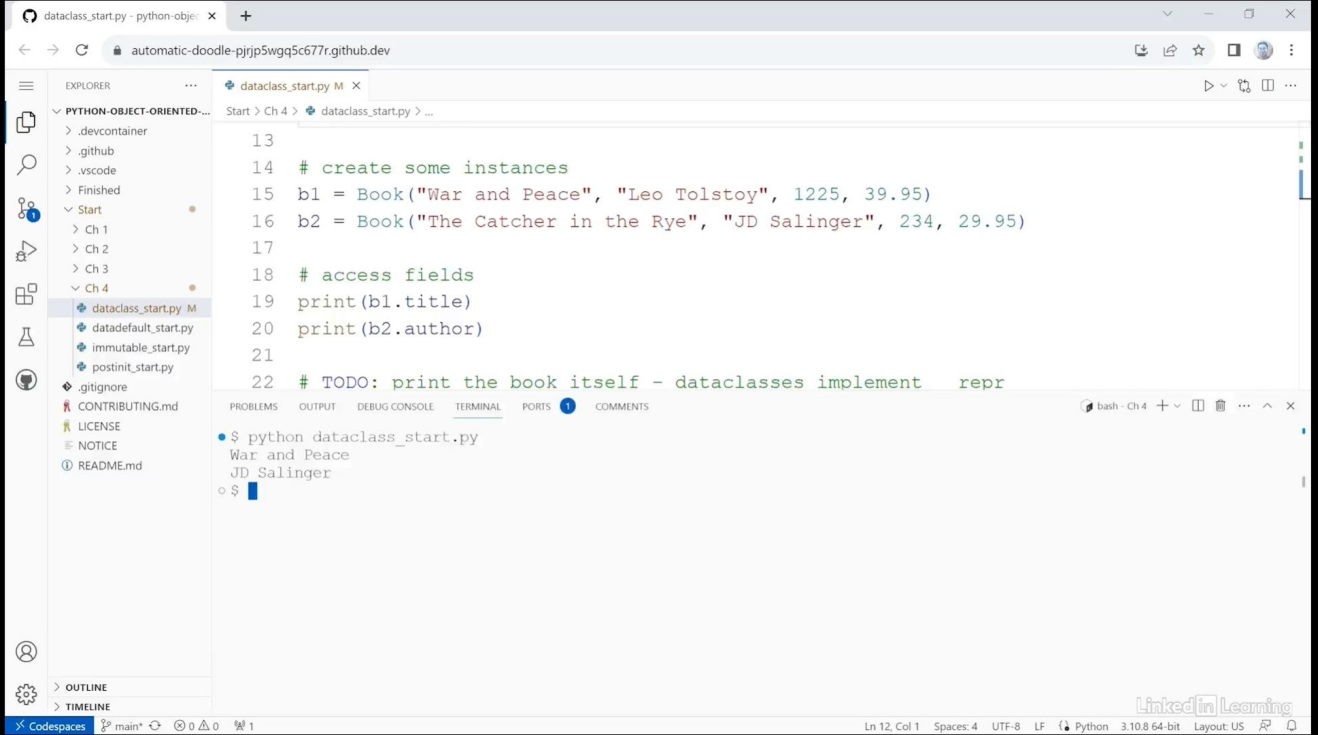
You are correct

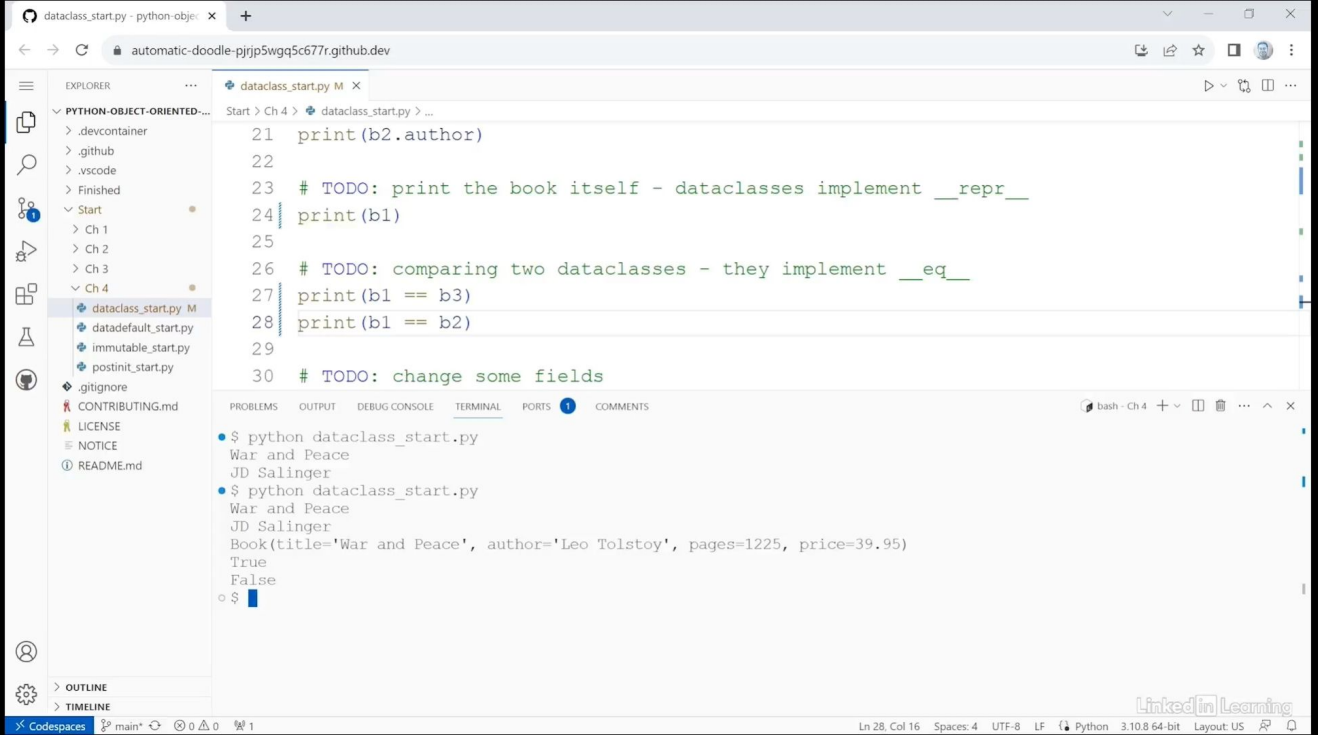
super().method_name()

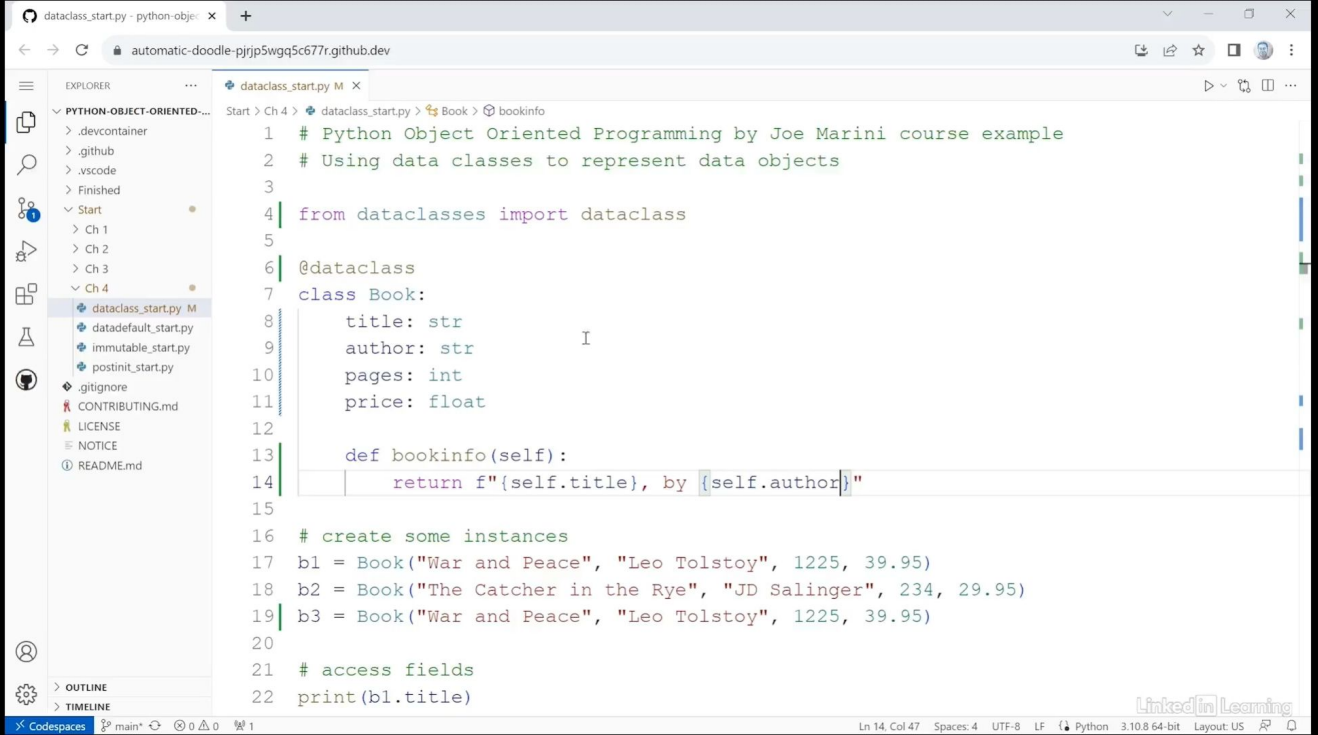
Next chapter

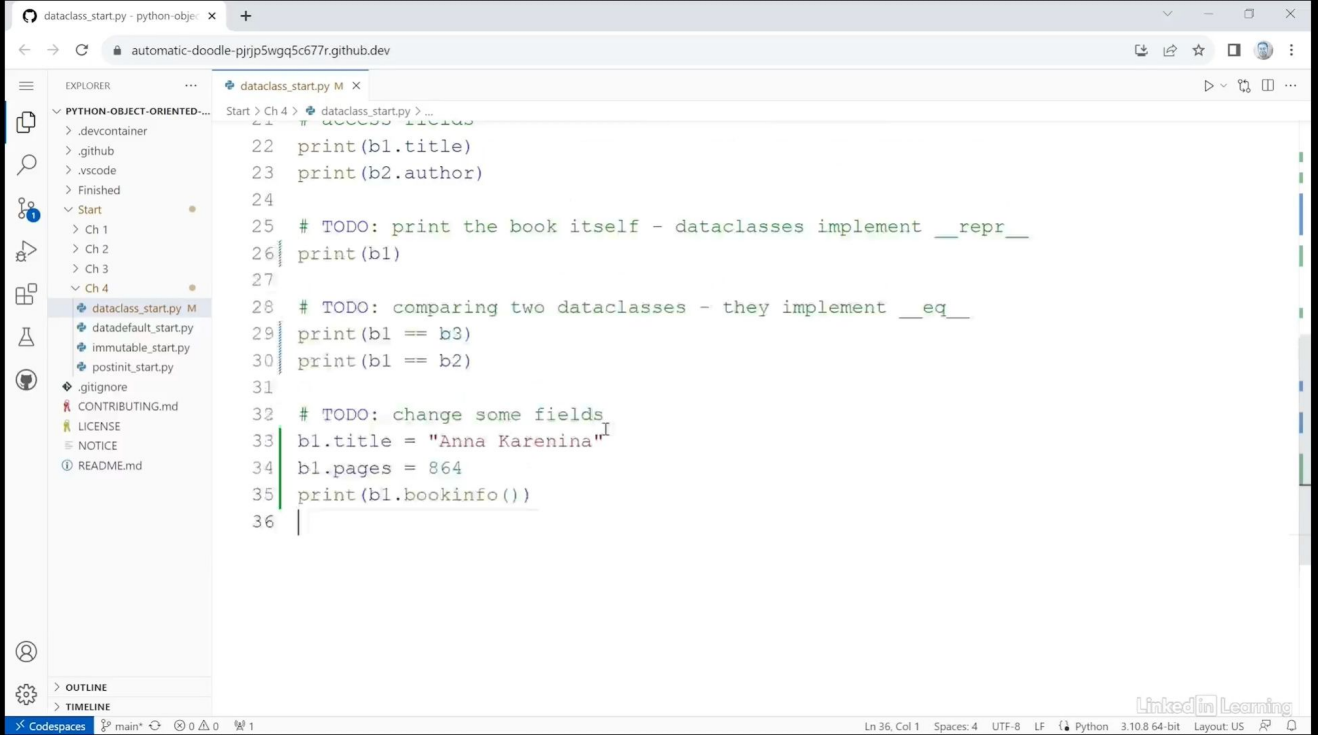

dataclass_start.py - python-objec
automatic-doodle-pjrjp5wgq5c677r.github.dev
EXPLORER
dataclass_start.py M
PYTHON-OBJECT-ORIENTED-...
.devcontainer
.github
.vscode
Finished
Start
Ch 1
Ch 2
Ch 3
Ch 4
dataclass_start.py M
datadefault_start.py
immutable_start.py
postinit_start.py
.gitignore
CONTRIBUTING.md
LICENSE
NOTICE
README.md
Start > Ch 4 > dataclass_start.py > ...
1 # Python Object Oriented Programming by Joe Marini course example
2 # Using data classes to represent data objects
3
4 from dataclasses import dataclass
5
6 @dataclass
7 class Book:
8 title: str
9 author: str
10 pages: int
11 price: float
12
13
14 # create some instances
15 b1 = Book("War and Peace", "Leo Tolstoy", 1225, 39.95)
16 b2 = Book("The Catcher in the Rye", "JD Salinger", 234, 29.95)
17
18 # access fields
19 print(b1.title)
20 print(b2.author)
21
22 # TODO: print the book itself - dataclasses implement __repr__
OUTLINE
TIMELINE
Codespaces
main*
Ln 12, Col 1
Spaces: 4
UTF-8
LF
Python
3.10.8 64-bit
Layout: US
LinkedIn Learning

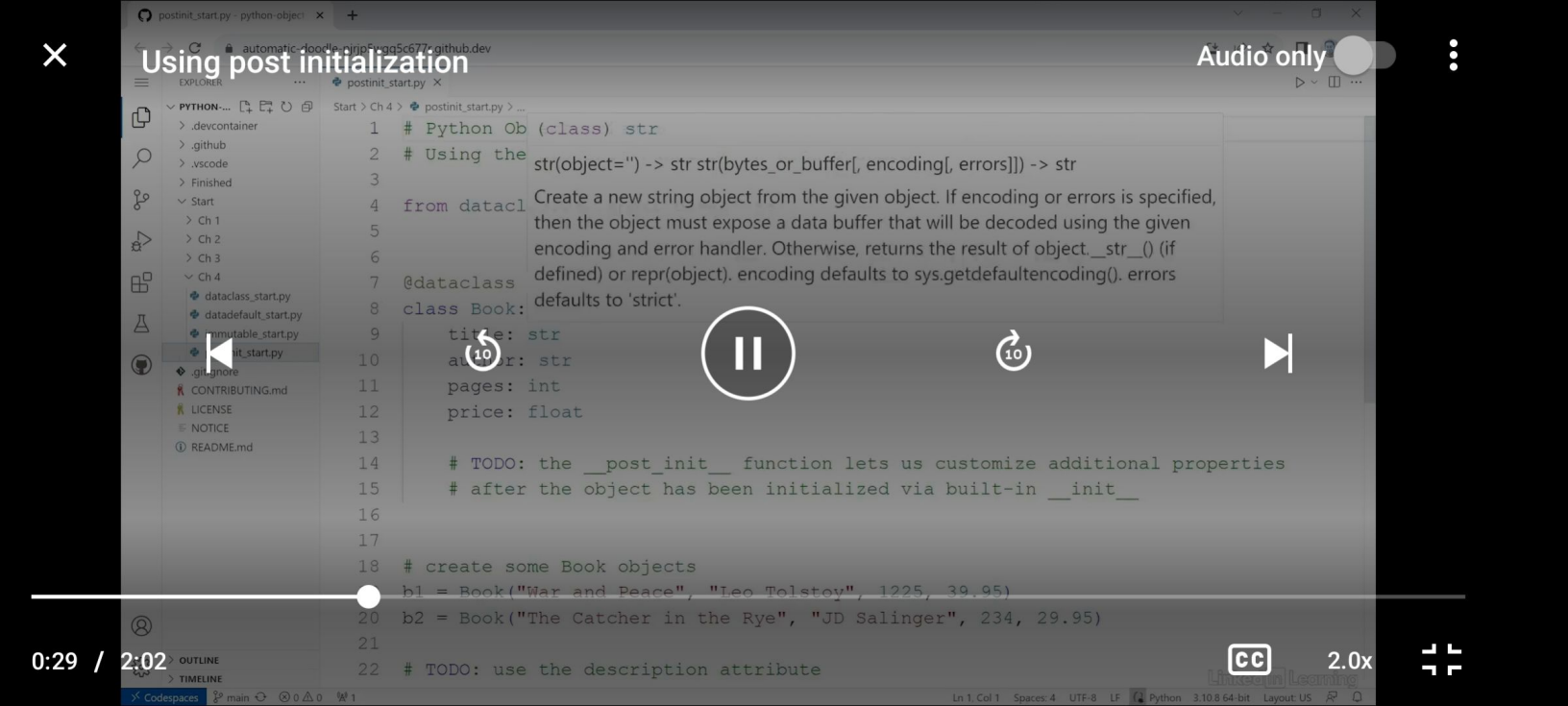

dataclass_start.py - python-objec
automatic-doodle-pjrjp5wgq5c677r.github.dev
EXPLORER
dataclass_start.py M
PYTHON-OBJECT-ORIENTED-...
.devcontainer
.github
.vscode
Finished
Start
Ch 1
Ch 2
Ch 3
Ch 4
dataclass_start.py M
datadefault_start.py
immutable_start.py
postinit_start.py
.gitignore
CONTRIBUTING.md
LICENSE
NOTICE
README.md
Start > Ch 4 > dataclass_start.py > ...
4 from dataclasses import dataclass
5
6 @dataclass
7 class Book:
8 title: str
9 author: str
10 pages: int
11 price: float
12
13
14 # create some instances
15 b1 = Book("War and Peace", "Leo Tolstoy", 1225, 39.95)
16 b2 = Book("The Catcher in the Rye", "JD Salinger", 234, 29.95)
17
18 # access fields
19 print(b1.title)
20 print(b2.author)
21
22 # TODO: print the book itself - dataclasses implement __repr__
23
24
25 # TODO: comparing two dataclasses - they implement __eq__
OUTLINE
TIMELINE
Codespaces
main*
Ln 12, Col 1
Spaces: 4
UTF-8
LF
Python
3.10.8 64-bit
Layout: US
LinkedIn Learning

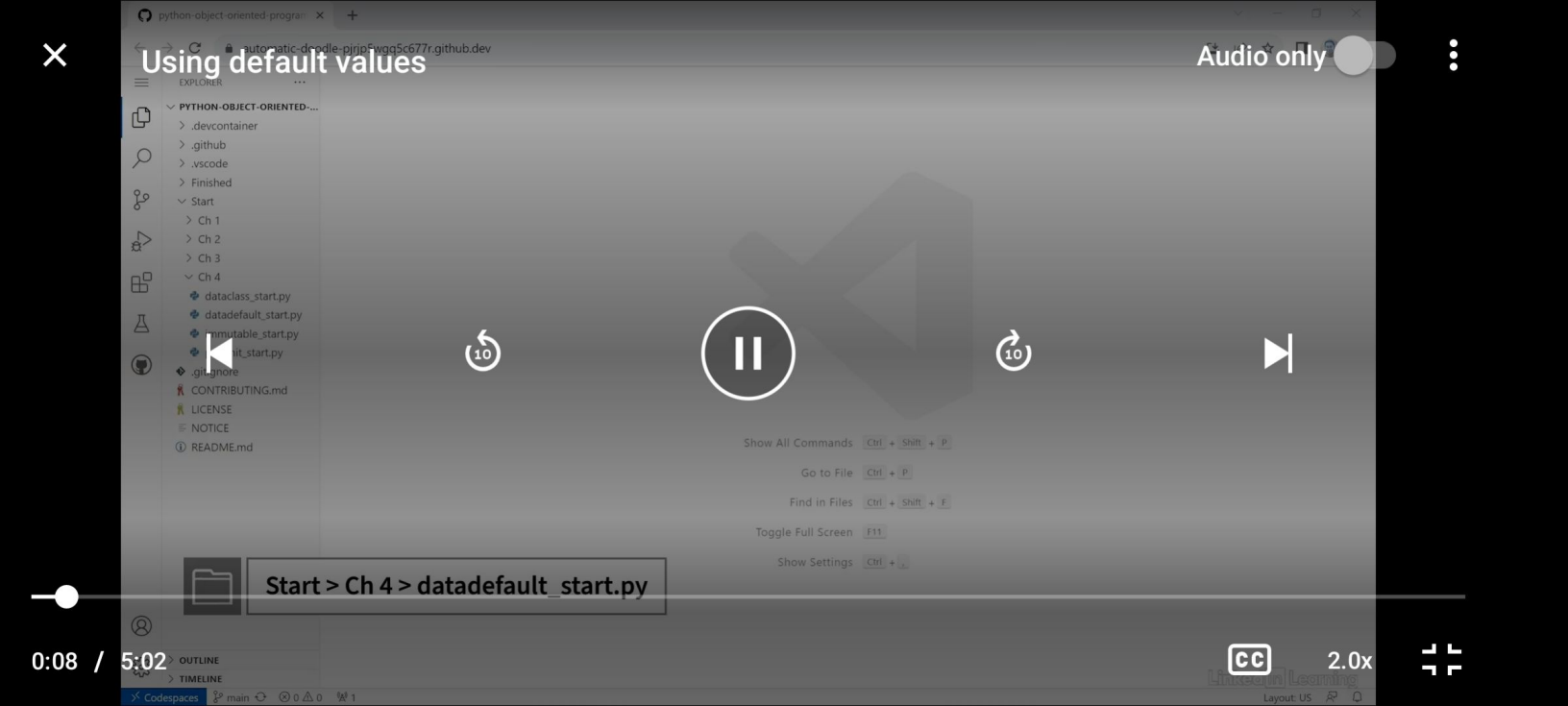

dataclass_start.py - python-objec
automatic-doodle-pjrjp5wgq5c677r.github.dev
EXPLORER
PYTHON-OBJECT-ORIENTED-...
.devcontainer
.github
.vscode
Finished
Start
Ch 1
Ch 2
Ch 3
Ch 4
dataclass_start.py M
datadefault_start.py
immutable_start.py
postinit_start.py
.gitignore
CONTRIBUTING.md
LICENSE
NOTICE
README.md
dataclass_start.py M
Start > Ch 4 > dataclass_start.py > ...
13
14 # create some instances
15 b1 = Book("War and Peace", "Leo Tolstoy", 1225, 39.95)
16 b2 = Book("The Catcher in the Rye", "JD Salinger", 234, 29.95)
17
18 # access fields
19 print(b1.title)
20 print(b2.author)
21
22 # TODO: print the book itself - dataclasses implement repr
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS 1
COMMENTS
bash - Ch 4
$ python dataclass_start.py
War and Peace
JD Salinger
$
OUTLINE
TIMELINE
Codespaces
main*
Ln 12, Col 1
Spaces: 4
UTF-8
LF
Python
3.10.8 64-bit
Layout: US
LinkedIn Learning

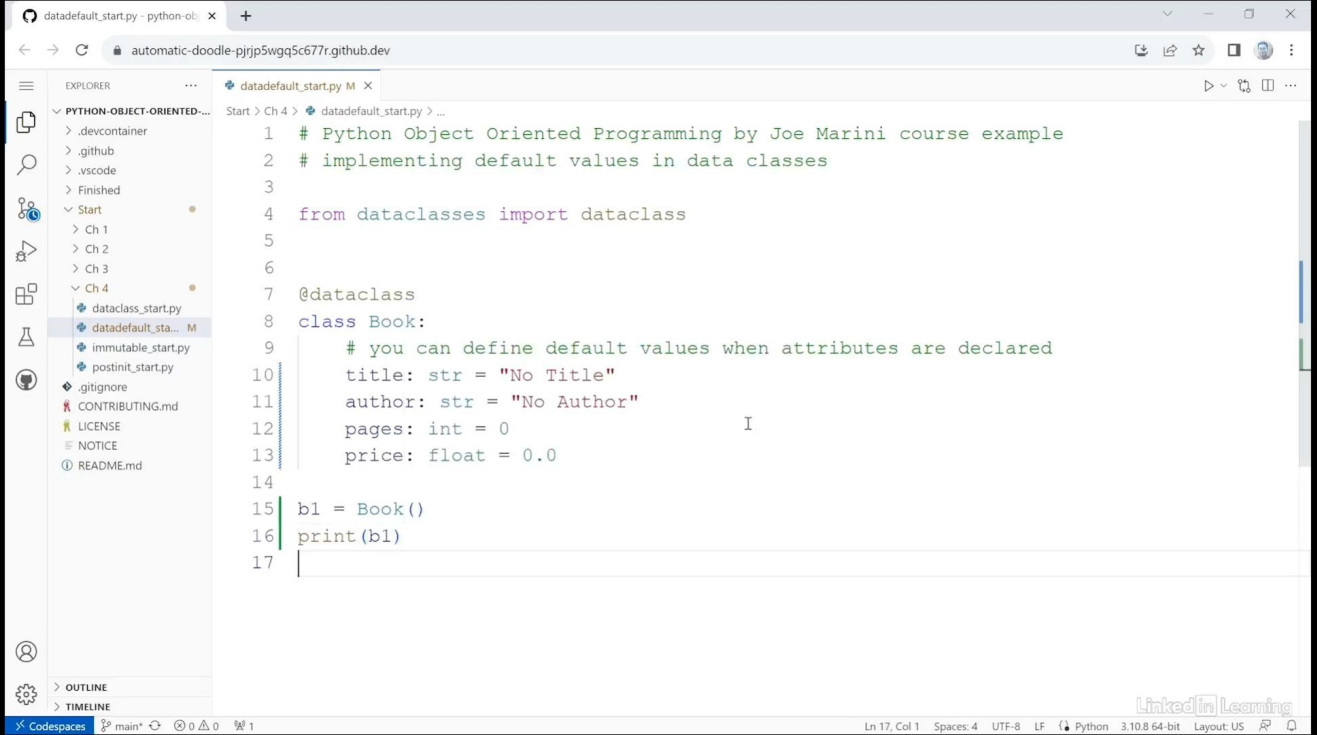

dataclass_start.py - python-objec
automatic-doodle-pjrjp5wgq5c677r.github.dev
EXPLORER
PYTHON-OBJECT-ORIENTED-...
.devcontainer
.github
.vscode
Finished
Start
Ch 1
Ch 2
Ch 3
Ch 4
dataclass_start.py M
datadefault_start.py
immutable_start.py
postinit_start.py
.gitignore
CONTRIBUTING.md
LICENSE
NOTICE
README.md
OUTLINE
TIMELINE
dataclass_start.py M
Start > Ch 4 > dataclass_start.py > ...
21 print(b2.author)
22
23 # TODO: print the book itself - dataclasses implement __repr__
24 print(b1)
25
26 # TODO: comparing two dataclasses - they implement __eq__
27 print(b1 == b3)
28 print(b1 == b2)
29
30 # TODO: change some fields
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS 1
COMMENTS
bash - Ch 4
$ python dataclass_start.py
War and Peace
JD Salinger
$ python dataclass_start.py
War and Peace
JD Salinger
Book(title='War and Peace', author='Leo Tolstoy', pages=1225, price=39.95)
True
False
$
Codespaces
main*
Ln 28, Col 16
Spaces: 4
UTF-8
LF
Python
3.10.8 64-bit
Layout: US
LinkedIn Learning

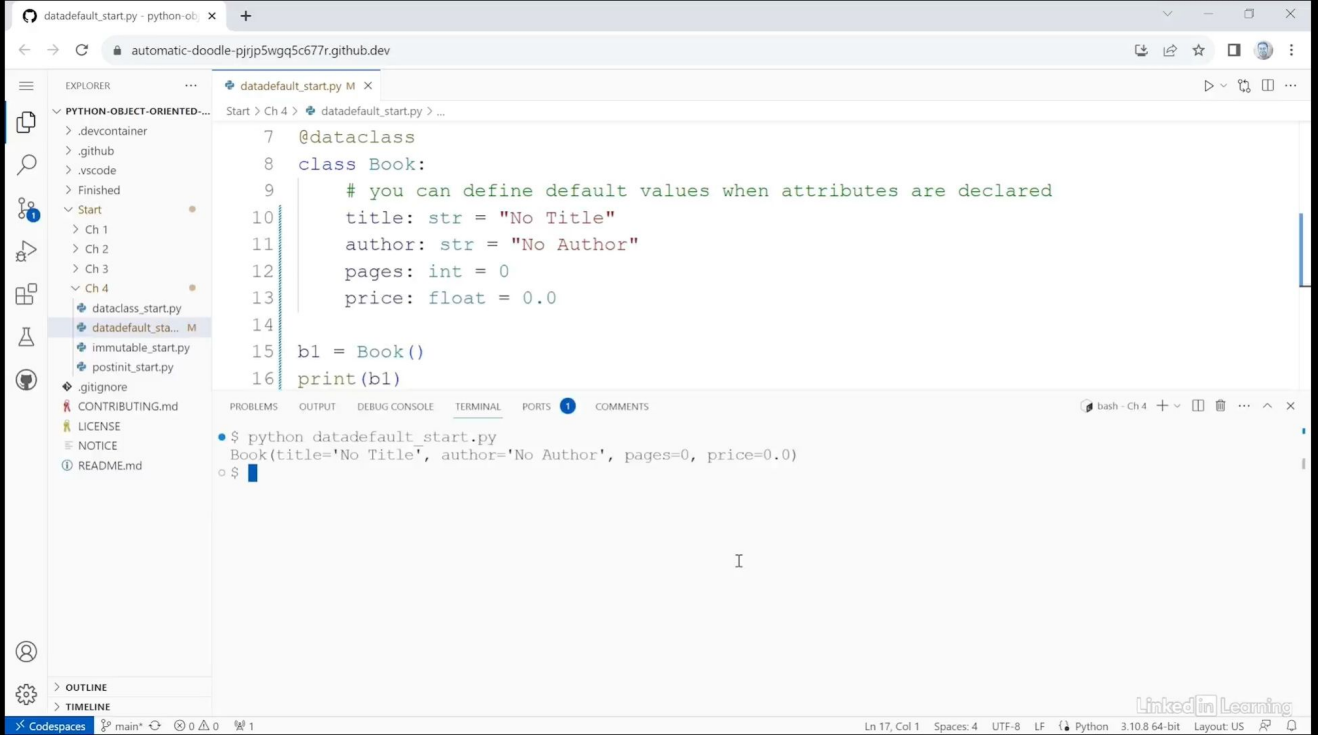

dataclass_start.py - python-objec
automatic-doodle-pjrjp5wgq5c677r.github.dev
EXPLORER
dataclass_start.py M
PYTHON-OBJECT-ORIENTED-...
.devcontainer
.github
.vscode
Finished
Start
Ch 1
Ch 2
Ch 3
Ch 4
dataclass_start.py M
datadefault_start.py
immutable_start.py
postinit_start.py
.gitignore
CONTRIBUTING.md
LICENSE
NOTICE
README.md
Start > Ch 4 > dataclass_start.py > Book > bookinfo
1 # Python Object Oriented Programming by Joe Marini course example
2 # Using data classes to represent data objects
3
4 from dataclasses import dataclass
5
6 @dataclass
7 class Book:
8 title: str
9 author: str
10 pages: int
11 price: float
12
13 def bookinfo(self):
14 return f"{self.title}, by {self.author}"
15
16 # create some instances
17 b1 = Book("War and Peace", "Leo Tolstoy", 1225, 39.95)
18 b2 = Book("The Catcher in the Rye", "JD Salinger", 234, 29.95)
19 b3 = Book("War and Peace", "Leo Tolstoy", 1225, 39.95)
20
21 # access fields
22 print(b1.title)
OUTLINE
TIMELINE
Codespaces
main*
Ln 14, Col 47
Spaces: 4
UTF-8
LF
Python
3.10.8 64-bit
Layout: US
LinkedIn Learning

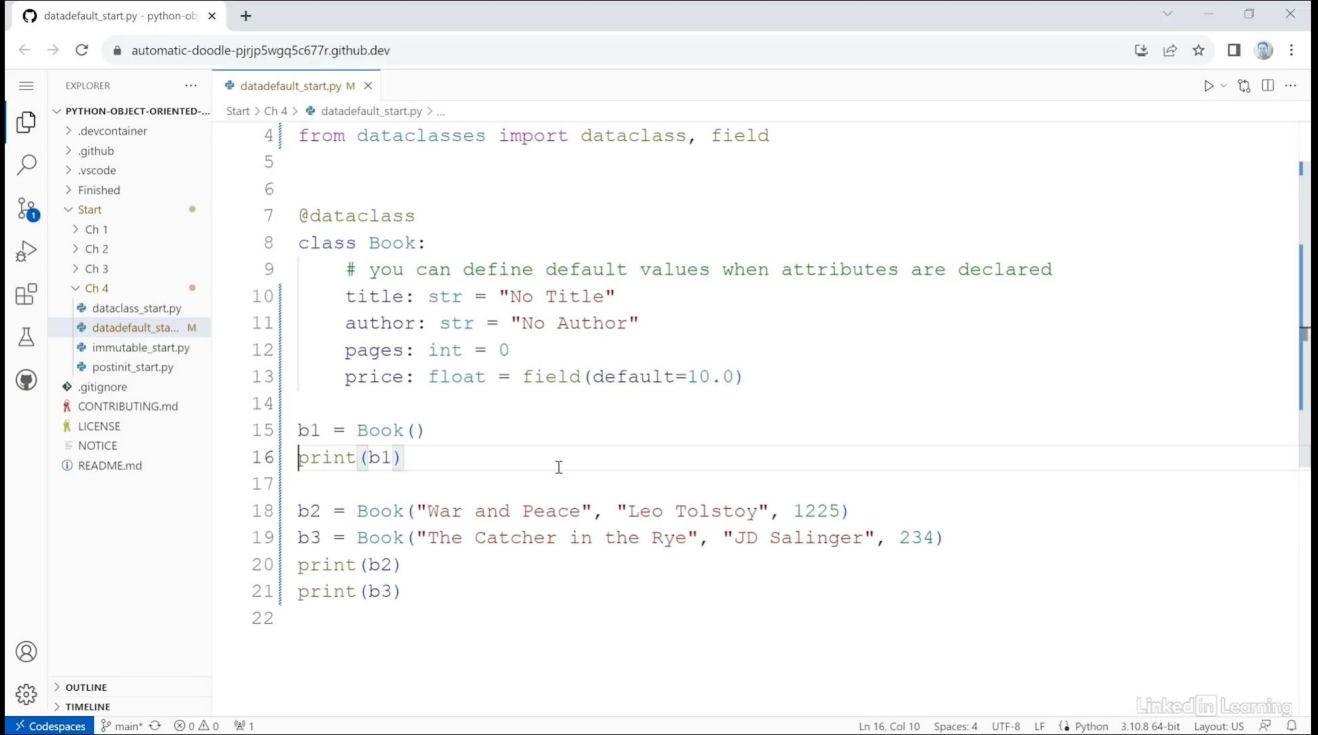

dataclass_start.py - python-objec
automatic-doodle-pjrjp5wgq5c677r.github.dev
EXPLORER
dataclass_start.py M
PYTHON-OBJECT-ORIENTED-...
.devcontainer
.github
.vscode
Finished
Start
Ch 1
Ch 2
Ch 3
Ch 4
dataclass_start.py M
datadefault_start.py
immutable_start.py
postinit_start.py
.gitignore
CONTRIBUTING.md
LICENSE
NOTICE
README.md
Start > Ch 4 > dataclass_start.py > ...
print(b1.title)
print(b2.author)
# TODO: print the book itself - dataclasses implement __repr__
print(b1)
# TODO: comparing two dataclasses - they implement __eq__
print(b1 == b3)
print(b1 == b2)
# TODO: change some fields
b1.title = "Anna Karenina"
b1.pages = 864
print(b1.bookinfo())
OUTLINE
TIMELINE
Codespaces
main*
Ln 36, Col 1
Spaces: 4
UTF-8
LF
Python
3.10.8 64-bit
Layout: US
LinkedIn Learning

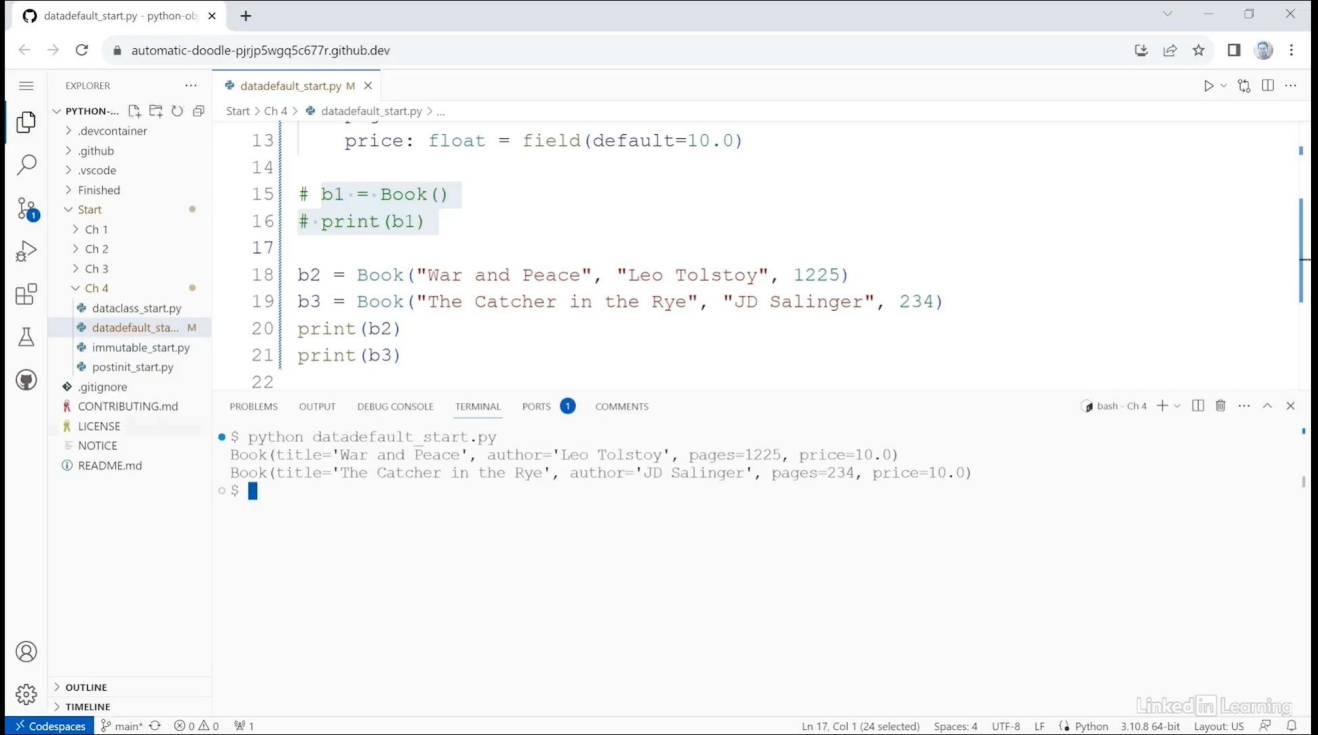

Using post initialization
Audio only
0:29 / 2:02
CC
2.0x

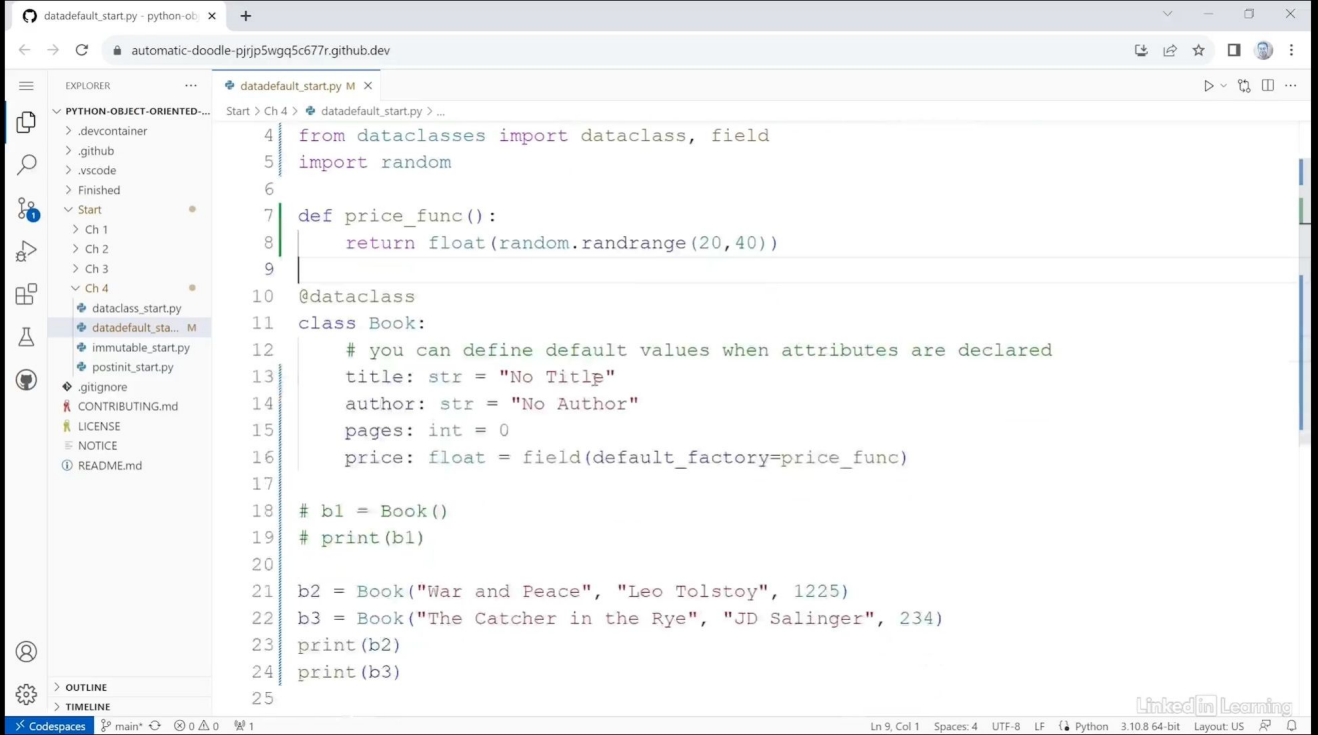

Using default values
Audio only
Start > Ch 4 > datadefault_start.py
0:08 / 5:02
CC
2.0x

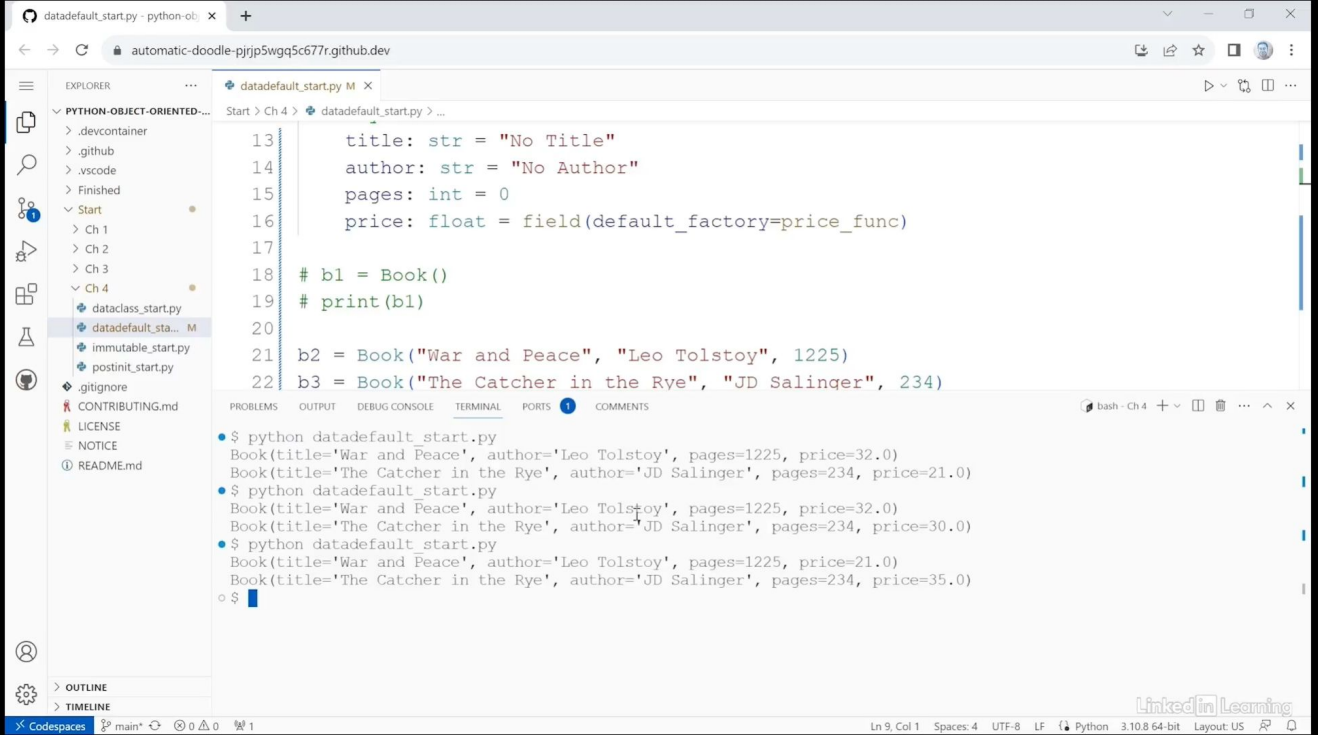

datadefault_start.py - python-ob
automatic-doodle-pjrjp5wgq5c677r.github.dev
EXPLORER
datadefault_start.py M
Start > Ch 4 > datadefault_start.py > ...
PYTHON-OBJECT-ORIENTED-...
.devcontainer
.github
.vscode
Finished
Start
Ch 1
Ch 2
Ch 3
Ch 4
dataclass_start.py
datadefault_sta... M
immutable_start.py
postinit_start.py
.gitignore
CONTRIBUTING.md
LICENSE
NOTICE
README.md
1 # Python Object Oriented Programming by Joe Marini course example
2 # implementing default values in data classes
3
4 from dataclasses import dataclass
5
6
7 @dataclass
8 class Book:
9 # you can define default values when attributes are declared
10 title: str = "No Title"
11 author: str = "No Author"
12 pages: int = 0
13 price: float = 0.0
14
15 b1 = Book()
16 print(b1)
17
OUTLINE
TIMELINE
Codespaces
main*
Ln 17, Col 1
Spaces: 4
UTF-8
LF
Python
3.10.8 64-bit
Layout: US
LinkedIn Learning

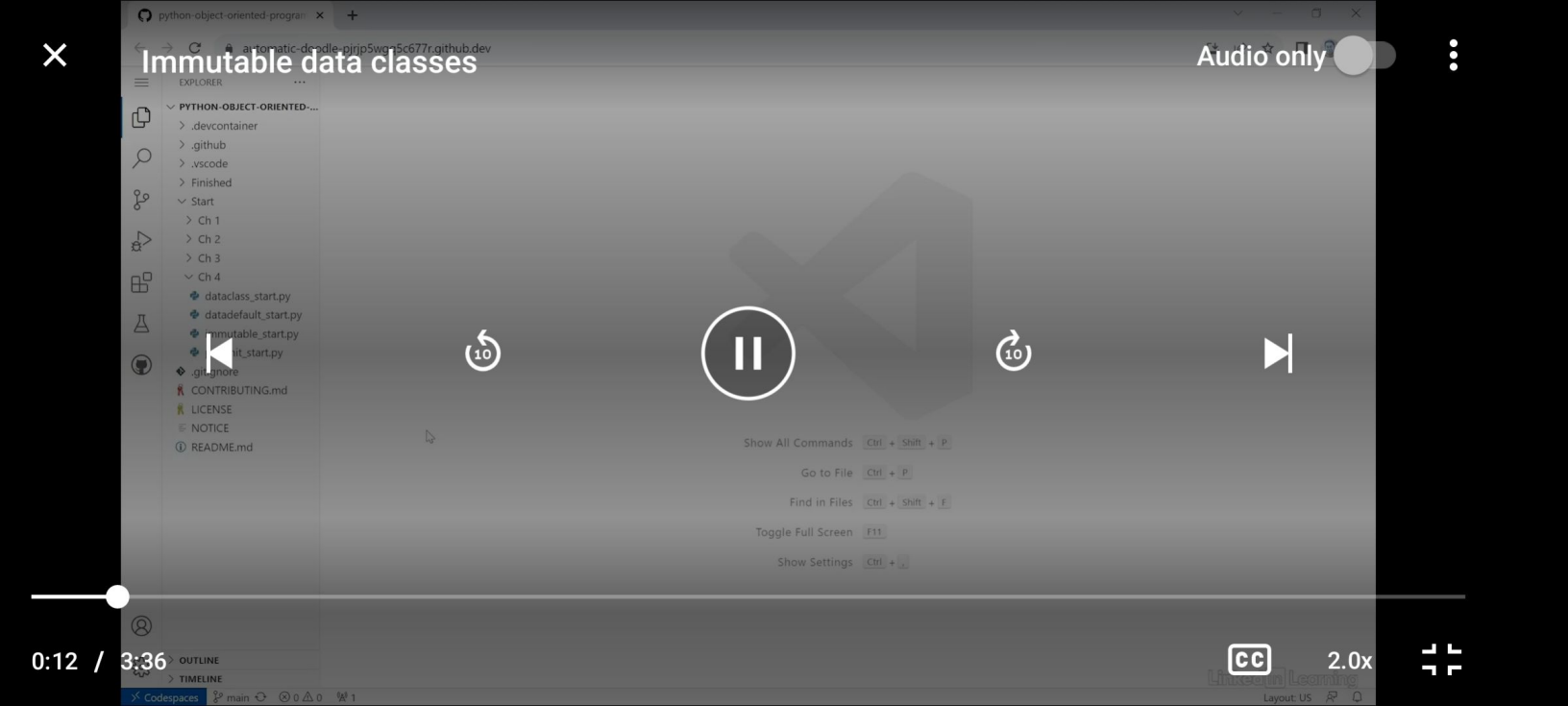

datadefault_start.py - python-ob
automatic-doodle-pjrjp5wgq5c677r.github.dev
EXPLORER
PYTHON-OBJECT-ORIENTED-...
.devcontainer
.github
.vscode
Finished
Start
Ch 1
Ch 2
Ch 3
Ch 4
dataclass_start.py
datadefault_sta... M
immutable_start.py
postinit_start.py
.gitignore
CONTRIBUTING.md
LICENSE
NOTICE
README.md
OUTLINE
TIMELINE
datadefault_start.py M
Start > Ch 4 > datadefault_start.py > ...
7 @dataclass
8 class Book:
9 # you can define default values when attributes are declared
10 title: str = "No Title"
11 author: str = "No Author"
12 pages: int = 0
13 price: float = 0.0
14
15 b1 = Book()
16 print(b1)
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS 1
COMMENTS
bash - Ch 4
$ python datadefault_start.py
Book(title='No Title', author='No Author', pages=0, price=0.0)
$
Codespaces
main*
Ln 17, Col 1
Spaces: 4
UTF-8
LF
Python
3.10.8 64-bit
Layout: US
LinkedIn Learning

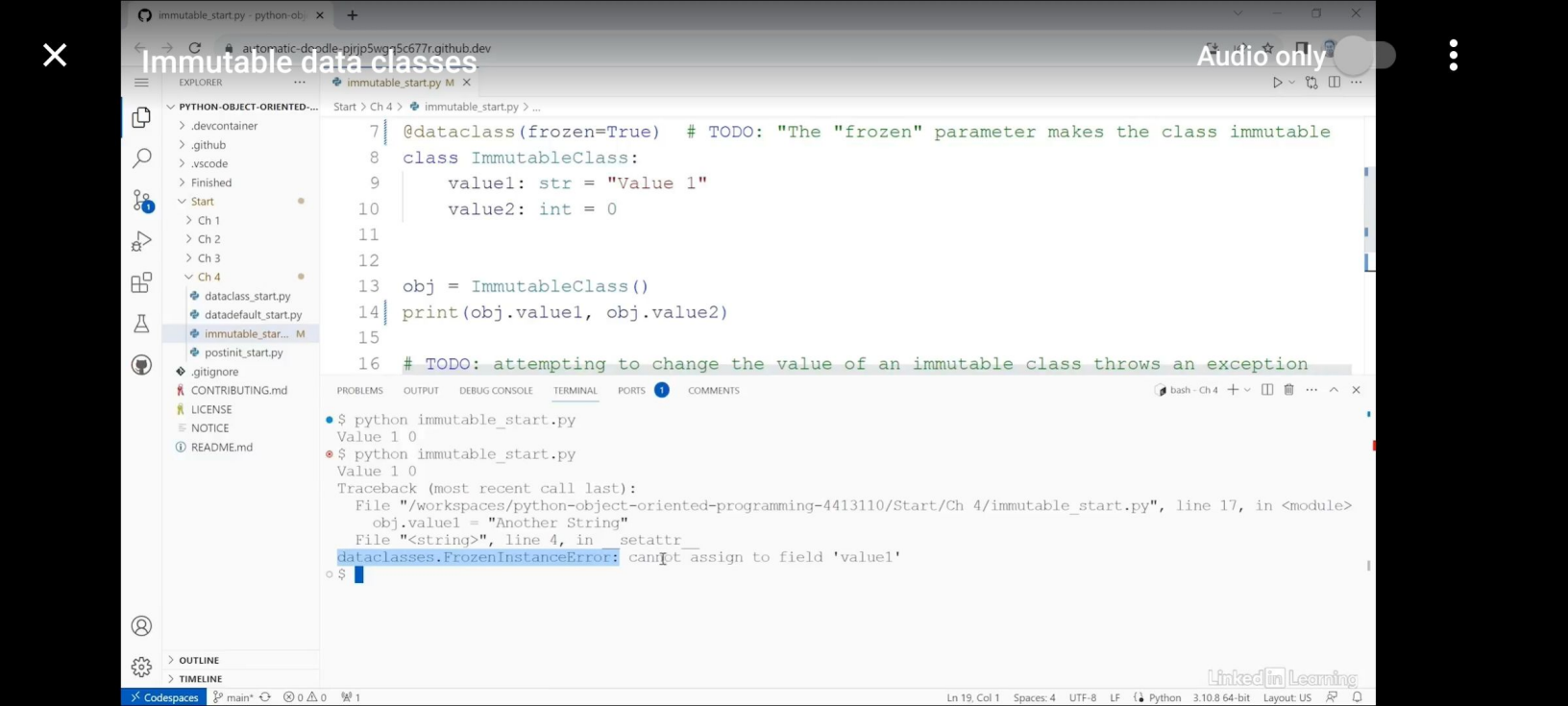

datadefault_start.py - python-ob
automatic-doodle-pjrjp5wgq5c677r.github.dev
EXPLORER
datadefault_start.py M
Start > Ch 4 > datadefault_start.py > ...
PYTHON-OBJECT-ORIENTED-...
.devcontainer
.github
.vscode
Finished
Start
Ch 1
Ch 2
Ch 3
Ch 4
dataclass_start.py
datadefault_sta... M
immutable_start.py
postinit_start.py
.gitignore
CONTRIBUTING.md
LICENSE
NOTICE
README.md
OUTLINE
TIMELINE
4 from dataclasses import dataclass, field
5
6
7 @dataclass
8 class Book:
9 # you can define default values when attributes are declared
10 title: str = "No Title"
11 author: str = "No Author"
12 pages: int = 0
13 price: float = field(default=10.0)
14
15 b1 = Book()
16 print(b1)
17
18 b2 = Book("War and Peace", "Leo Tolstoy", 1225)
19 b3 = Book("The Catcher in the Rye", "JD Salinger", 234)
20 print(b2)
21 print(b3)
22
Codespaces
main*
Ln 16, Col 10
Spaces: 4
UTF-8
LF
Python
3.10.8 64-bit
Layout: US
LinkedIn Learning


datadefault_start.py - python-ob
automatic-doodle-pjrjp5wgq5c677r.github.dev
EXPLORER
PYTHON-...
.devcontainer
.github
.vscode
Finished
Start
Ch 1
Ch 2
Ch 3
Ch 4
dataclass_start.py
datadefault_sta... M
immutable_start.py
postinit_start.py
.gitignore
CONTRIBUTING.md
LICENSE
NOTICE
README.md
OUTLINE
TIMELINE
datadefault_start.py M
Start > Ch 4 > datadefault_start.py > ...
13 price: float = field(default=10.0)
14
15 # b1 = Book()
16 # print(b1)
17
18 b2 = Book("War and Peace", "Leo Tolstoy", 1225)
19 b3 = Book("The Catcher in the Rye", "JD Salinger", 234)
20 print(b2)
21 print(b3)
22
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS 1
COMMENTS
bash - Ch 4
$ python datadefault_start.py
Book(title='War and Peace', author='Leo Tolstoy', pages=1225, price=10.0)
Book(title='The Catcher in the Rye', author='JD Salinger', pages=234, price=10.0)
$
Codespaces
main*
Ln 17, Col 1 (24 selected)
Spaces: 4
UTF-8
LF
Python
3.10.8 64-bit
Layout: US
LinkedIn Learning


datadefault_start.py - python-ob
automatic-doodle-pjrjp5wgq5c677r.github.dev
EXPLORER
datadefault_start.py M
Start > Ch 4 > datadefault_start.py > ...
PYTHON-OBJECT-ORIENTED-...
.devcontainer
.github
.vscode
Finished
Start
Ch 1
Ch 2
Ch 3
Ch 4
dataclass_start.py
datadefault_sta... M
immutable_start.py
postinit_start.py
.gitignore
CONTRIBUTING.md
LICENSE
NOTICE
README.md
OUTLINE
TIMELINE
4 from dataclasses import dataclass, field
5 import random
6
7 def price_func():
8 return float(random.randrange(20,40))
9
10 @dataclass
11 class Book:
12 # you can define default values when attributes are declared
13 title: str = "No Title"
14 author: str = "No Author"
15 pages: int = 0
16 price: float = field(default_factory=price_func)
17
18 # b1 = Book()
19 # print(b1)
20
21 b2 = Book("War and Peace", "Leo Tolstoy", 1225)
22 b3 = Book("The Catcher in the Rye", "JD Salinger", 234)
23 print(b2)
24 print(b3)
25
Codespaces
main*
Ln 9, Col 1
Spaces: 4
UTF-8
LF
Python
3.10.8 64-bit
Layout: US
LinkedIn Learning


datadefault_start.py - python-ob
automatic-doodle-pjrjp5wgq5c677r.github.dev
EXPLORER
PYTHON-OBJECT-ORIENTED-...
.devcontainer
.github
.vscode
Finished
Start
Ch 1
Ch 2
Ch 3
Ch 4
dataclass_start.py
datadefault_sta... M
immutable_start.py
postinit_start.py
.gitignore
CONTRIBUTING.md
LICENSE
NOTICE
README.md
datadefault_start.py M
Start > Ch 4 > datadefault_start.py > ...
13 title: str = "No Title"
14 author: str = "No Author"
15 pages: int = 0
16 price: float = field(default_factory=price_func)
17
18 # b1 = Book()
19 # print(b1)
20
21 b2 = Book("War and Peace", "Leo Tolstoy", 1225)
22 b3 = Book("The Catcher in the Rye", "JD Salinger", 234)
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS 1
COMMENTS
bash - Ch 4
$ python datadefault_start.py
Book(title='War and Peace', author='Leo Tolstoy', pages=1225, price=32.0)
Book(title='The Catcher in the Rye', author='JD Salinger', pages=234, price=21.0)
$ python datadefault_start.py
Book(title='War and Peace', author='Leo Tolstoy', pages=1225, price=32.0)
Book(title='The Catcher in the Rye', author='JD Salinger', pages=234, price=30.0)
$ python datadefault_start.py
Book(title='War and Peace', author='Leo Tolstoy', pages=1225, price=21.0)
Book(title='The Catcher in the Rye', author='JD Salinger', pages=234, price=35.0)
$
OUTLINE
TIMELINE
Codespaces
main*
Ln 9, Col 1
Spaces: 4
UTF-8
LF
Python
3.10.8 64-bit
Layout: US
LinkedIn Learning


Immutable data classes
Audio only
0:12 / 3:36
CC
2.0x


Immutable data classes
Audio only
automatic-doodle-pjrjp5wq-5c677r.github.dev
immutable_start.py - python-obj
EXPLORER
PYTHON-OBJECT-ORIENTED-...
.devcontainer
.github
.vscode
Finished
Start
Ch 1
Ch 2
Ch 3
Ch 4
dataclass_start.py
datadefault_start.py
immutable_star... M
postinit_start.py
.gitignore
CONTRIBUTING.md
LICENSE
NOTICE
README.md
OUTLINE
TIMELINE
immutable_start.py M
Start > Ch 4 > immutable_start.py > ...
7 @dataclass(frozen=True) # TODO: "The "frozen" parameter makes the class immutable
8 class ImmutableClass:
9 value1: str = "Value 1"
10 value2: int = 0
11
12
13 obj = ImmutableClass()
14 print(obj.value1, obj.value2)
15
16 # TODO: attempting to change the value of an immutable class throws an exception
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS 1 COMMENTS
bash - Ch 4
$ python immutable_start.py
Value 1 0
$ python immutable_start.py
Value 1 0
Traceback (most recent call last):
File "/workspaces/python-object-oriented-programming-4413110/Start/Ch 4/immutable_start.py", line 17, in <module>
obj.value1 = "Another String"
File "<string>", line 4, in __setattr__
dataclasses.FrozenInstanceError: cannot assign to field 'value1'
$
Codespaces
main*
Ln 19, Col 1
Spaces: 4
UTF-8
LF
Python
3.10.8 64-bit
Layout: US
LinkedIn Learning

Your Answers

Question **1** of **4**

What is the name of the method to define if you want to perform additional initialization on a data class?

You are correct

post_init

Next chapter

Your Answers

Question **2** of **4**

Which of these is the primary advantage of using the `@dataclass` decorator in Python?

You are correct

It automatically generates special methods like init and repr for the class.

Next chapter

Your Answers

Question **3** of **4**

What is the correct way of declaring a data class that is immutable?

You are correct

@dataclass(frozen=True)

Next chapter

Your Answers

Question **4** of **4**

Which of these is the correct way to define a data class that has default values?

You are correct

```
@dataclass
class Person:
    occupation : str
    name : str = "John Doe"
    age : int = 35
    city : str = "New York"
```

65) WAP to implement inheritance using shape class and also make rectangle as a derived class. declare a function area to calculate the area of rectangle and also show the colour of the rectangle. Also make required constructor

```
class shape:
    def __init__(self, color):
        self.color = color

class Rectangle(shape):
    def __init__(self, color, length, width):
        super().__init__(color)
        self.length = length
        self.width = width

    def area(self):
        return self.length * self.width

    def display_info(self):
        print(f"I am a {self.color} rectangle with area {self.area()}. square units.")

my_rectangle = Rectangle(color="Blue", length=5.0, width=3.0)
my_rectangle.display_info()
```

```
I am a Blue rectangle with area 15.0. square units.
```

66) WAP to implement Multiple Inheritance using the example of employee class, waged employee and salaried employee. Write Appropriate constructors and functions to calculate pay.

```
class Employee:
    def __init__(self, emp_id, name):
        self.emp_id = emp_id
        self.name = name

class WagedEmployee(Employee):
    def __init__(self, emp_id, name, hourly_rate, hours_worked):
        super().__init__(emp_id, name)
        self.hourly_rate = hourly_rate
        self.hours_worked = hours_worked

    def calculate_pay(self):
        return self.hourly_rate * self.hours_worked

class SalariedEmployee(Employee):
    def __init__(self, emp_id, name, monthly_salary):
        super().__init__(emp_id, name)
        self.monthly_salary = monthly_salary

    def calculate_pay(self):
        return self.monthly_salary

waged_emp = WagedEmployee(emp_id=101, name="John", hourly_rate=20, hours_worked=40)
print(f"Waged Employee Pay: ${waged_emp.calculate_pay()}")

salaried_emp = SalariedEmployee(emp_id=102, name="Alice", monthly_salary=5000)
print(f"Salaried Employee Pay: ${salaried_emp.calculate_pay()}")
```

```
Waged Employee Pay: $800
Salaried Employee Pay: $5000
```

67) Implement operator overloading for + and less than equal to operator for user defined class.

```
class MyNumber:
    def __init__(self, value):
        self.value = value

    def __add__(self, other):
        return MyNumber(self.value + other.value)

    def __le__(self, other):
        return self.value <= other.value
num1 = MyNumber(10)
num2 = MyNumber(20)
result = num1 + num2
print(f"Sum: {result.value}")
print(f"Is num1 <= num2? {num1 <= num2}")
```

```
Sum: 30
Is num1 <= num2? True
```

68) Implement operator overloading for less than equal to operator for user defined class.

```
class MyNumber:
    def __init__(self, value):
        self.value = value

    def __le__(self, other):
        return self.value <= other.value
num1 = MyNumber(10)
num2 = MyNumber(20)

print(f"Is num1 <= num2? {num1 <= num2}")
```

```
Is num1 <= num2? True
```

69) Implement operator overloading for -(minus operator).

```
class MyNumber:
    def __init__(self, value):
        self.value = value

    def __sub__(self, other):
        return MyNumber(self.value - other.value)

num1 = MyNumber(20)
num2 = MyNumber(10)

result = num1 - num2
print(f"Result of subtraction: {result.value}")
```

```
Result of subtraction: 10
```

70) Implement operator overloading for unary operator++, and - -.

```
class MyNumber:
    def __init__(self, value):
        self.value = value

    def __pos__(self):
        return MyNumber(self.value)

    def __neg__(self):
        return MyNumber(-self.value)

    def __str__(self):
        return str(self.value)

num = MyNumber(10)

# Unary +
positive_num = +num
print(f"Unary +: {positive_num}")

# Unary -
negative_num = -num
print(f"Unary -: {negative_num}")
```

```
Unary +: 10
Unary -: -10
```