75 Python Object Oriented Programming Exercises Volume 1

EDCORNER LEARNING

75 Python Object Oriented Programming Exercises Volume 1

75 Python Object Oriented Programming Exercises Volume 1

Edcorner Learning

Table of Contents

Introduction

Python is a general-purpose interpreted, interactive, object- oriented, and a powerful programming language with dynamic semantics. It is an easy language to learn and become expert. Python is one among those rare languages that would claim to be both easy and powerful. Python's elegant syntax and dynamic typing alongside its interpreted nature makes it an ideal language for scripting and robust application development in many areas on giant platforms.

Python helps with the modules and packages, which inspires program modularity and code reuse. The Python interpreter and thus the extensive standard library are all available in source or binary form for free of charge for all critical platforms and can be freely distributed. Learning Python doesn't require any pre- requisites. However, one should have the elemental understanding of programming languages.

This Book consist of 75 python Object Oriented Programming coding exercises to practice different topics.

In each exercise we have given the exercise coding statement you need to complete and verify your answers. We also attached our own input output screen of each exercise and their solutions.

Learners can use their own python compiler in their system or can use any online compilers available.

We have covered all level of exercises in this book to give all the learners a good and efficient Learning method to do hands on python different scenarios.

Module 1 Local Enclosed Global Built-In Rules

1. The stock_info( ) function is defined. Using the appropriate attribute of the stock_info( ) function, display the names of all arguments to this function to the console.

An example of calling the function:

print(stock_info('ABC', 'USA', 115, '$'))

Company: ABC

Country: USA

Price: $ 115

Tip: Use the code attribute of the function.

Expected result:

('company', 'country', 'price', 'currency')

def stock_info(company, country, price, currency):

return f'Company: {company}\nCountry: {country}\nPrice: {currency} {price}'

```
#Edcorner Learning Pythons OOPS Exercises


def stock_info(company, country, price,
currency):
    return f'Company: {company}\nCountry:
{country}\nPrice: {currency} {price}'


print(stock_info.__code__.co_varnames)
```

```
('company', 'country', 'price', 'currency')
```

2. Using the built-ins module import the sum( ) function. Then display its documentation of this function. Call the function on the list below and print the result to the console.

[-4, 3, 2]

Expected result:

Help on built-in function sun in nodule built-ins:

sum(iterable, /, start=0)

Return the sun of a 'start' value (default: 0) plus an iterable of numbers

When the Iterable is empty, return the start value.

This function is intended specifically for use with numeric values and may reject non-numeric types.

1

Solution:

```
import builtins


help(builtins.sum)

print(builtins.sum([-4, 3, 2]))
```

3. A global variable counter is given with an incorrectly implemented update_counter() function. Correct the implementation of the update_counter ( ) function so that you can modify the counter variable from this function. Then call the update_counter ( ) function.

Tip: Use the global statement.

Expected result:

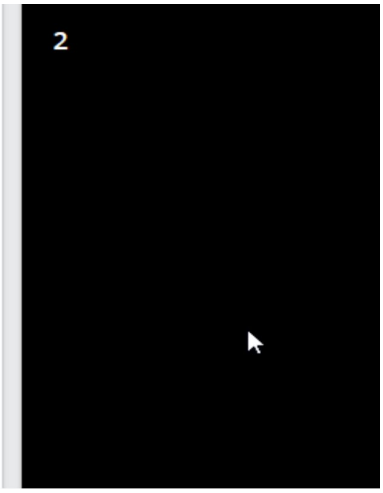
2

counter = 1

def update_counter():

counter += 1

```
#Edcorner Learning Pythons OOPS Exercises


counter = 1

def update_counter():
    global counter
    counter += 1
    print(counter)

update_counter()
```



print(counter)

4. The following global variables are given:

• counter

• dot_counter

and incorrectly implemented update_counters ( ) function. Correct the implementation of the update_counters ( ) function so that you can modify the values of the given global variables from this function. Then call update_counters() 40 times.

In response, print the value of the counter and dot_counter global variables to the console as shown below.

Tip: Use the global statement.

Expected result:

40

.....

counter = 0

dot_counter = ''

```
def update_counter():
    counter += 1
    dot_counter += '.'
```

Solution:

```
#Edcorner Learning Pythons OOPS Exercises


counter = 0
dot_counter = ''

def update_counter():
    global counter, dot_counter
    counter += 1
    dot_counter += '.'

[update_counter() for _ in range(40)]
print(counter)
print(dot_counter)
```

```
40
........................................
```

5. A display_info() function was implemented. This function has an incorrectly implemented internal update_counter( ) function. Correct the implementation of this function so that you can modify non-local variables: counter and dot_counter from the internal function

update_counter() .

In response, call dispiay_info( ) with the number_of_updates argument set to 10.

Tip: Use the nonlocal statement.

Expected result:

110

.....

```
def display_info(number_of_updates=1):
    counter = 100
    dot_counter = ''
    def update_counter():
        counter += 1
        dot_counter += '.'
    [update_counter() for _ in range(number_of_updates)]
    print(counter)
    print(dot_counter)
```

Solution:

```
#Edcorner Learning Pythons OOPS Exercises


def display_info(number_of_updates=1):
    counter = 100
    dot_counter = ''

    def update_counter():
        nonlocal counter, dot_counter
        counter += 1
        dot_counter += '.'

    [update_counter() for _ in
range(number_of_updates)]

    print(counter)
    print(dot_counter)

display_info(10)
```

```
110
..........
```

Module 2 Namespaces and Scopes

6. Import the built-in datetime module and display the namespace of this module (sorted alphabetically) as given below.

Tip: Use the _dict_ attribute of the datetime module.

Expected result:

MAXYEAR

MINYEAR

builtlns

cached

doc

file

loader

name

package

spec

date

datetime

datetime_CAPI

sys

time

timedelta

timezone

tzinfo

Solution:

```python
#Edcorner Learning Pythons OOPS Exercises

import datetime

for name in sorted(datetime.__dict__):
    print(name)
```

```
MAXYEAR
MINYEAR
__builtins__
__cached__
__doc__
__file__
__loader__
__name__
__package__
__spec__
date
datetime
datetime_CAPI
sys
time
timedelta
timezone
tzinfo
```

7. The Product class is given below. Display the namespace (value of the _dict_ attribute) of this class as shown below.

Expected result:

__module__

__init__

__repr__

get_id

__dict__

__weakref__

__doc__

```
import uuid
class Product:
    def __init__(self, product_name, price):
```

```
    self.product_id = self.get_id()
    self.product_name = product_name
    self.price = price
  def __repr__(self):
    return f"Product(product_name='{self.product_name}', price={self.price})"

      @staticmethod
        def get_id():
          return str(uuid.uuid4().fields[-1])[:6]
```

```
#Edcorner Learning Pythons OOPS Exercises

import uuid

class Product:

    def __init__(self, product_name,
price):
        self.product_id = self.get_id()
        self.product_name = product_name
        self.price = price

    def __repr__(self):
        return
f"Product(product_name='{self.product_nam
e}', price={self.price})"

    @staticmethod
    def get_id():
        return
str(uuid.uuid4().fields[-1])[:6]


for name in Product.__dict__:
    print(name)
```

```
__module__
__init__
__repr__
get_id
__dict__
__weakref__
__doc__
```

8. The Product class is specified. An instance of this class named product was created. Display the namespace (value of the _dict_ attribute) of this instance as shown below.

Expected result:

{'product_name': 'Mobile Phone1, 'product_id': '54274', 'price': 2900}

```
import uuid
class Product:
    def __init__(self, product_name, product_id, price):
        self.product_name = product_name
        self.product_id = product_id
        self.price = price
    def __repr__(self):
        return f"Product(product_name='{self.product_name}', price={self.price})"
product = Product('Mobile Phone', '54274', 2900)
```

Solution:

```
#Edcorner Learning Pythons OOPS Exercises

import uuid

class Product:

    def __init__(self, product_name,
product_id, price):
        self.product_name = product_name
        self.product_id = product_id
        self.price = price

    def __repr__(self):
        return
f"Product(product_name='{self.product_nam
e}', price={self.price})"

product = Product('Mobile Phone',
'54274', 2900)
print(product.__dict__)
```

Module 3 Args and Kwargs

9. Implement a function called stick( ) that takes any number of bare arguments and return an object of type str being a concatenation of all arguments of type str passed to the function with the '#' sign (see below).

Example:

[IN]: stick('sport', 'summer', 4, True)

[OUT]: 'sport#summer'

As an answer call the stick( ) function in the following ways (print the result to the console):

• stick('sport', 'summer')

• stick(3, 5, 7)

stick(False, 'time'. True, 'workout', [], 'gym')

Expected result:

Sport#sumer

time#workout#gym

Solution:

```python
#Edcorner Learning Pythons OOPS Exercises


def stick(*args):
    args = [arg for arg in args if
isinstance(arg, str)]
    result = '#'.join(args)
    return result

print(stick('sport', 'summer'))
print(stick(3, 5, 7))
print(stick(False, 'time', True,
'workout', [], 'gym'))
```

```
sport#summer

time#workout#gym
```

10. Implement a function called dispiay_info() which prints the name of the company (as shown below) and if the user also passes an argument named price , it prints the price (as shown below).

Example I:

[IN]: dlsplay_info(company='Amazon')

Company name: Apple

Example II:

[IN]: display_info(company='Amazon', price=1140)

Company name: Amazon Price: $ 1140

In response, call display_info() as shown below:

display_info(company='CD Projekt', price=100)

Expected result:

Company name: CD Projekt Price: $ 100

```
def display_info(company, **kwargs):
    pass
```

Solution:

```
#Edcorner Learning Pythons OOPS Exercises


def display_info(company, **kwargs):
    print(f'Company name: {company}')
    if 'price' in kwargs:
        print(f"Price: $
{kwargs['price']}")

display_info(company='CD Projekt',
price=100)
```

```
Company name: CD Projekt
Price: $ 100
```

Module 4 Classes

11. Create the simplest class in Python and name it Vehicle.

Tip: Use the pass statement.

Solution:

```
class Vehicle:
   pass
```

12. Create the simplest Python class named Phone and display its type to the console.

Expected result:

<class 'type'>

Solution:

```
class Phone:
   pass
print(type(Phone))
```

13. Create a class named Vehicle and add the following documentation:

"""This is a Vehicle class."""

Solution

```
class Vehicle:
   """This is a Vehicle class."""
```

14. The implementation of the Vehicle class is given:

class Vehicle:

This is a Vehicle class.

Display the value of the name attribute of the Vehicle class to the console.

Expected result:

Vehicle

class Vehicle:

"""This is a Vehicle class."""

```
#Edcorner Learning Pythons OOPS Exercises

class Vehicle:
    """This is a Vehicle class."""

print(Vehicle.__name__)
```

```
Vehicle
```

15. The implementation of the Container class is given:

class Container:

"""This is a Container class."""

Display all _dict_ attribute keys of the Container class to the console.

Expected result:

dict_keys(['__module__', '__dict__', '__weakref__', '__doc__'])

class Container:

"""This is a Container class."""

16. The implementation of the Container class is given:

class Container:

"""This Is a Container class.

Display the value of the module attribute of the Container class to the console.

Note: The solution that the user provides is in a file named exercise.py, while the checking code (which is invisible to the user) is executed from a file named evaluate.py from the level where the

Container class is imported. Therefore, instead of the name of the module _main_ , the

response will be the name of the module in which this class is implemented, exercise in this case.

Expected result:

exercise

class Container:

"""This is a Container class."""

Solution:

class Container:

"""This is a Container class."""

print(Container.__module__)

17. The implementation of the Container class is given:

class Container:

"""This Is a Container class.

Create an instance of the Container class and assign it to the container variable. Print the type of container variable to the console.

Note: The solution that the user provides is in a file named exercise.py, while the checking code (which is invisible to the user) is executed from a file named evaluate.py from the level where the Container class is imported. Therefore, instead of the name of the module _main_ , the response will be the name of the module in which this class is implemented, exercise in this case.

Expected result:

<class 'exercise.Container'>

class Container:

"""This is a Container class."""

Solution:

class Container:

"""This is a Container class."""

container = Container()

print(type(container))

18. The implementation of the Container class is given:

class Container:

"""This Is a Container class.

Create an instance of the Container class and assign to the container variable. Then print the _class _ attribute value of the container instance.

Note: The solution that the user provides is in a file named exercise.py, while the checking code (which is invisible to the user) is executed from a file named evaluate.py from the level where the Container class is imported. Therefore, instead of the name of the module _main_ , the response will be the name of the module in which this class is implemented, exercise in this case.

Expected result:

<class 'exercise.Container'>

class Container:

"""This is a Container class."""

Solution:

```
class Container:
    """This is a Container class."""
container = Container()
print(container.__class__)
```

19. Define a simple class named *Model*. Then create an instance of this class named *model*.

Using the built-in function isinstance( ) check if the *model* is an instance of the *Model* class. Print the result to the console.

Expected result:

True

```
#Edcorner Learning Pythons OOPS Exercises


class Model:
    pass


model = Model()
print(isinstance(model, Model))
```

```
True
```

20. Define two empty classes named:

• Model

• View

Then create two instances (one for each class):

• model for the Model class

• view for the View class

Using the built-in function isinstance( ) check whether the model and view objects are instances of the Model class. Print the result to the console.

Expected result:

True

False

```python
#Edcorner Learning Pythons OOPS Exercises


class Model:
    pass


class View:
    pass


model = Model()
view = View()

print(isinstance(model, Model))
print(isinstance(view, Model))
```

```
True
False
```

21. Two empty classes are defined:

• Model

• View

Three objects were created (object1, object2, object3).

Using the built-in function is instance ( ) check whether object1, object2 and object3 are instances of the Model class or of the View class. Print the result to the console.

Expected result:

True

False

False

```
class Model:
   pass
class View:
   pass
object1 = Model()
object2 = [Model(), Model()]
object3 = {}
```

```
#Edcorner Learning Pythons OOPS Exercises


class Model:
    pass


class View:
    pass


object1 = Model()
object2 = [Model(), Model()]
object3 = {}

print(isinstance(object1, (Model, View)))
print(isinstance(object2, (Model, View)))
print(isinstance(object3, (Model, View)))
```

```
True
False
False
```

22. Implement an empty class named Container. Then create an instance of this class named container. In response, display the type of dictionary attribute _dict_ for the Container class and for the container instance.

Expected result:

<class ‘mappingproxy’>

<class 'dict’>

```
#Edcorner Learning Pythons OOPS Exercises


class Container:
    pass


container = Container()

print(type(Container.__dict__))
print(type(container.__dict__))
```

```
<class 'mappingproxy'>
<class 'dict'>
```

23. Two empty classes are defined:

- Model
- View

Using the built-in function issubclassQ check if the classes Model and View are derived classes (subclasses) of the built-in object class.

Expected result:

True

True

class Model:

pass

class View:

pass

```
#Edcorner Learning Pythons OOPS Exercises


class Model:
    pass


class View:
    pass


print(issubclass(Model, object))
print(issubclass(View, object))
```

```
True
True
```

Module 5 Classes Attributes

24. Implement a class named Brandname. In the Phone class, define a class attribute named brand and set its value to ‘Amazon’. Then, using dot notation and print ( ) function, display the value of the brand attribute of the Phone class to the console.

Expected result:

Amazon

```
#Edcorner Learning Pythons OOPS Exercises


class Brandname:
    brand = 'Amazon'


print(Brandname.brand)
```

```
Amazon
```

25. Implement a class named Phone. In the Phone class, define two class attributes with names:

• brand

• model

and set their values to:

• 'Apple'

• ‘iPhone X'

Then use the built-in functions getattr() and print () to display the values of the given attributes of the Phone class to the console as shown below.

Expected result:

Apple

iPhone X

```
#Edcorner Learning Pythons OOPS Exercises


class Phone:
    brand = 'Apple'
    model = 'iPhone X'


print(getattr(Phone, 'brand'))
print(getattr(Phone, 'model'))
```

```
Apple
iPhone X
```

26. A class named Phone is defined below. Using dot notation, modify the value of the attributes:

• brand to 'Samsung'

• model to 'Galaxy'

In response, print the values for the brand and model attributes to the console as shown below.

Expected result:

brand: Samsung

model: Galaxy

class Phone:

brand = 'Apple'

```
#Edcorner Learning Pythons OOPS Exercises


class Phone:
    brand = 'Apple'
    model = 'iPhone X'


Phone.brand = 'Samsung'
Phone.model = 'Galaxy'

print(f'brand: {Phone.brand}')
print(f'model: {Phone.model}')
```

```
brand: Samsung
model: Galaxy
```

model = 'iPhone X'

27. A class named Laptop is defined below. Using the setattr( ) built-in function modify the value of attributes:

• brand to 'Acer'

• model to 'Predator'

In response, using the built-in function getattr() and print() .print the values of the brand and model attributes to the console as shown below.

Expected result:

brand: Acer

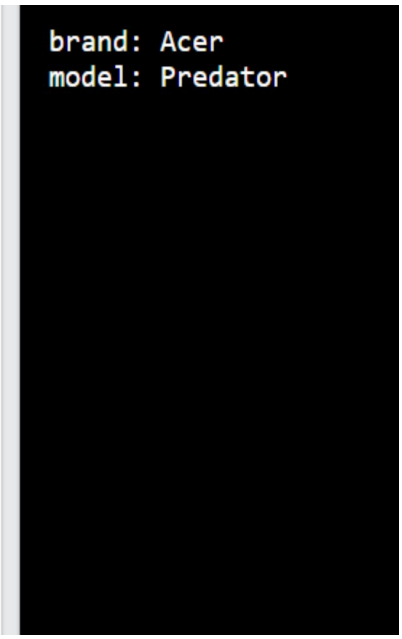
model: Predator

class Laptop:

brand = 'Lenovo'

model = 'ThinkPad'

```python
#Edcorner Learning Pythons OOPS Exercises


class Laptop:
    brand = 'Lenovo'
    model = 'ThinkPad'


setattr(Laptop, 'brand', 'Acer')
setattr(Laptop, 'model', 'Predator')

print(f"brand: {getattr(Laptop,
'brand')}")
print(f"model: {getattr(Laptop,
'model')}")
```



28. Implement a class named OnlineShop with the class attributes set appropriately:

• Sector to the Value 'electronics'

• sector_code to the value 'ele'

• is_public_company to the value False

Then, using dot notation, add a class attribute called country and set its value to 'usa' . In response, print the user-defined OnlineShop class attribute names as shown below.

Expected result:

['sector', 'sector_code', 1is_public_company','country']

```
#Edcorner Learning Pythons OOPS Exercises


class OnlineShop:
    sector = 'electronics'
    sector_code = 'ELE'
    is_public_company = False


OnlineShop.country = 'USA'
attrs = [attr for attr in
OnlineShop.__dict__.keys() if not
attr.startswith('_')]
print(attrs)
```

```
['sector', 'sector_code', 'is_public_company', 'country']
```

29. A class named OnlineShop was defined with the class attributes set accordingly:

• Sector to the Value 'electronics'

• sector_code to the value 'ele'

• is_public_company to the value False

Using the del statement remove the class attribute named sector_code. In response, print the rest of the user-defined OnlineShop class attribute names as a list as shown below.

Expected result:

[' sector',‘is_public_company’]

class OnlineShop:

sector = 'electronics'

sector_code = 'ELE'

is_public_company = False

```
#Edcorner Learning Pythons OOPS Exercises


class OnlineShop:
    sector = 'electronics'
    sector_code = 'ELE'
    is_public_company = False


del OnlineShop.sector_code
attrs = [attr for attr in
OnlineShop.__dict__.keys() if not
attr.startswith('_')]
print(attrs)
```

```
['sector', 'is_public_company']
```

30. A class named OnlineShop was defined with the class attributes set accordingly:

• Sector to the Value 'electronics'

• sector_code to the value 'ele'

• is_public_company to the value False

Using the builtin deiattrQ function remove the class attribute sector_code. In response, print the rest of the user-defined OnlineShop class attribute names as a list as shown below.

Expected result:

[‘sector','is_public_company’]

class OnlineShop:

    sector = 'electronics'

    sector_code = 'ELE'

    is_public_company = False

```
#Edcorner Learning Pythons OOPS Exercises


class OnlineShop:
    sector = 'electronics'
    sector_code = 'ELE'
    is_public_company = False


delattr(OnlineShop, 'sector_code')
attrs = [attr for attr in
OnlineShop.__dict__.keys() if not
attr.startswith('_')]
print(attrs)
```

```
['sector', 'is_public_company']
```

31. A class named OnlineShop was defined with the class attributes set accordingly:

• Sector to the Value 'electronics'

• sector_code to the value 'ele'

• is_public_company to the value False

Display all user-defined OnlineShop class attribute names with their values as shown below.

Expected result:

sector -> electronics

sector_code -> ELE

is_public_company -> False

```
class OnlineShop:
    sector = 'electronics'
    sector_code = 'ELE'
    is_public_company = False
```

```python
#Edcorner Learning Pythons OOPS Exercises


class OnlineShop:
    sector = 'electronics'
    sector_code = 'ELE'
    is_public_company = False


for attr, value in
OnlineShop.__dict__.items():
    if not attr.startswith('_'):
        print(f'{attr} -> {value}')
```

```
sector -> electronics
sector_code -> ELE
is_public_company -> False
```

32. A class named OnlineShop was defined with the class attributes set accordingly:

• Sector to the Value 'electronics'

• sector_code to the value 'ele'

• is_public_company to the value False

Outside of the class, implement a function called describe_attrs() that displays the names of all user-defined class attributes and their values as shown below. In response, call the

describe_attrs() function.

Expected result:

sector -> electronics

sector_code -> ELE

is_public_conpany -> False

class OnlineShop:

sector = 'electronics'

sector_code = 'ELE'

is_public_company = False

```python
#Edcorner Learning Pythons OOPS Exercises


class OnlineShop:
    sector = 'electronics'
    sector_code = 'ELE'
    is_public_company = False


def describe_attrs():
    for attr, value in
OnlineShop.__dict__.items():
        if not attr.startswith('__'):
            print(f'{attr} -> {value}')

describe_attrs()
```

```
sector -> electronics
sector_code -> ELE
is_public_company -> False
```

33. A class named OnlineShop was defined with the class attributes set accordingly:

• Sector to the Value 'electronics'

• sector_code to the value 'ele'

• is_public_company to the value False

Implement a function (class callable attribute) named get_sector() that returns the value of the sector attribute of OnlineShop class. You only need to implement this function.

class OnlineShop:

sector = 'electronics'

sector_code = 'ELE'

is_public_company = False

Solution:

```
class OnlineShop:
    sector = 'electronics'
    sector_code = 'ELE'
    is_public_company = False
    def get_sector():
        return OnlineShop.sector
```

34. Implement the HouseProject class with class attributes respectively:

- number_of_floors = 3
- area = 100

Then, in the HouseProject class implement a function (class callable attribute) called describe_project( ), which displays basic information about the project as follows:

Floor number: 3

Area: 100

In response, call describe_project() function.

Expected result:

Floor number: 3

Area: 100

```python
#Edcorner Learning Pythons OOPS Exercises


class HouseProject:
    number_of_floors = 3
    area = 100

    def describe_project():
        print(f'Floor number:
{HouseProject.number_of_floors}\nArea:
{HouseProject.area}')


HouseProject.describe_project()
```

```
Floor number: 3
Area: 100
```

Module 6 Instance Attributes

35. The Book class is defined. Create an instance of the Book class named book. Display the value of the _dict_ attribute for the book instance. Then assign two attributes to the book instance:

• author to the Value 'Dan Brown'

• title to the value ' Inferno '

In response, display the _dict_ attribute for the book instance again.

Expected result:

{}

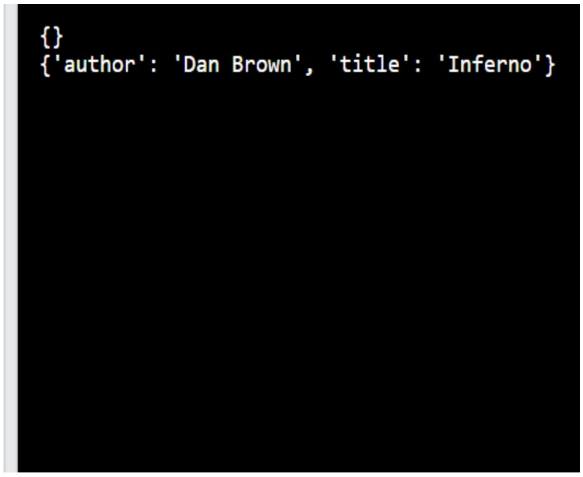
{'author': 'Dan Brown','title','Inferno'}

class Book:

language = 'ENG'

is_ebook = True

```
#Edcorner Learning Pythons OOPS Exercises


class Book:
    language = 'ENG'
    is_ebook = True


book = Book()
print(book.__dict__)
book.author = 'Dan Brown'
book.title = 'Inferno'
print(book.__dict__)
```



36. The Book class is defined. Create two instances of the Book class named book_1 and book_2. Then assign instance attributes to these objects (using dot notation) as follows:

• to object book_1:

• author = 'Edcorner Learning'

• title = 'Python Programming Exercises'

• to object book_2:

• author = 'Edcorner Learning'
• title = 'Python OOPS Exercises'

• year_of_publishment = 2021

In response, print the value of the _dict_ attribute of book_1 and book_2.

Expected result:

{'author': 'Edcorner Learning', 'title': 'Python Programming Exercises'}

{'author': 'Edcorner Learning', 'title': 'Python OOPS Exercises', 'year_of_publishment': 2021}

```
class Book:
    language = 'ENG'
    is_ebook = True
```

```
#Edcorner Learning Pythons OOPS Exercises


class Book:
    language = 'ENG'
    is_ebook = True


book_1 = Book()
book_2 = Book()

book_1.author = 'Edcorner Learning'
book_1.title = 'Python Programming
Exercises'

book_2.author = 'Edcorner Learning'
book_2.title = 'Python OOPS Exercises'
book_2.year_of_publishment = 2021

print(book_1.__dict__)
print(book_2.__dict__)
```

```
{'author': 'Edcorner Learning', 'title': 'Python Programming Exercises'}
{'author': 'Edcorner Learning', 'title': 'Python OOPS Exercises', 'year_of_publishment': 2021}
```

37. The Book class is defined. Two instances of the Book class named book_1 and book_2 was created. Then the instance attributes were assigned to these objects (using the dot notation), respectively:

• to object book_1:

• author = 'Dan Brown'

• title = 'Inferno'

to object book_2:

- title = 'The Da Vinci Code'

• year_of_publishment = 2003

Then a books list was created. Create a loop to list all the attributes of the book_l and book_2 instances with their values as shown below (separate each instance with a line of 30 1 - ' characters as shown below).

Expected result:

author -> Dan Brown

title -> Inferno

----

author -> Dan Brown

title -> The Da Vinci Code

year_of_publishment -> 2003

```
class Book:
    language = 'ENG'
    is_ebook = True



book_1 = Book()
book_2 = Book()

book_1.author = 'Dan Brown'
book_1.title = 'Inferno'

book_2.author = 'Dan Brown'
book_2.title = 'The Da Vinci Code'
book_2.year_of_publishment = 2003

books = [book_1, book_2]
```

```python
#Edcorner Learning Pythons OOPS Exercises


class Book:
    language = 'ENG'
    is_ebook = True


book_1 = Book()
book_2 = Book()

book_1.author = 'Dan Brown'
book_1.title = 'Inferno'

book_2.author = 'Dan Brown'
book_2.title = 'The Da Vinci Code'
book_2.year_of_publishment = 2003

books = [book_1, book_2]
for book in books:
    for attr in book.__dict__:
        print(f'{attr} -> {getattr(book,
attr)}')
    print('-' * 30)
```

```
author -> Dan Brown
title -> Inferno
------------------------------
author -> Dan Brown
title -> The Da Vinci Code
year_of_publishment -> 2003
------------------------------
```

38. The Book class is defined. A list books_data is also given.

books_data = [

{'author': 'Dan Brown', 'title': 'Inferno'},

{'author': 'Dan Brown', 'title': 'The Da Vinci Code', 'year_of_publishnent': 2003}

]

Based on this data, create two instances of the Book class, where the instance attributes will be the keys from the given dictionaries (books_data list) with their corresponding values.

In response, print the _dict_ attributes of the objects to the console as shown below.

Expected result:

{'author': 'Dan Brown', 'title': 'Inferno'}

{'author': 'Dan Brown', 'title': 'The Da Vinci Code', 'year_of_publishment': 2003}

class Book:

language = 'ENG'

```
    is_ebook = True
books_data = [
    {'author': 'Dan Brown', 'title': 'Inferno'},
    {'author': 'Dan Brown', 'title': 'The Da Vinci Code', 'year_of_publishment': 2003}
]
```

```
#Edcorner Learning Pythons OOPS Exercises


class Book:
    language = 'ENG'
    is_ebook = True


books_data = [
    {'author': 'Dan Brown', 'title':
'Inferno'},
    {'author': 'Dan Brown', 'title': 'The
Da Vinci Code', 'year_of_publishment':
2003}
]

books = []
for book_data in books_data:
    book = Book()
    for attr, value in book_data.items():
        setattr(book, attr, value)
    books.append(book)

for book in books:
    print(book.__dict__)
```

```
{'author': 'Dan Brown', 'title': 'Inferno'}
{'author': 'Dan Brown', 'title': 'The Da Vinci Code', 'year_of_publishment': 2003}
```

39. The Book class is defined. Implement a method called set_title() that allows you to set an instance attribute called title (without validation). Then create an instance of the Book class named book and set the title attribute to ' inferno' using the set_title( ) method.

In response, print the value of the title attribute of the book instance.

Expected result:

Python OOPS Exercises

class Book:

    language = 'ENG'

    is_ebook = True

```
#Edcorner Learning Pythons OOPS Exercises


class Book:
    language = 'ENG'
    is_ebook = True

    def set_title(self, value):
        self.title = value


book = Book()
book.set_title('Python OOPS Exercises')
print(book.title)
```

```
Python OOPS Exercises
```

40. The Book class is defined. Implement a method named set_title() that sets an instance attribute named title. Before setting the value, check if it's an object of str type, if not raise a TypeError with the following message:

'The value of the title attribute must be of str type.'

Then create an instance of the Book class named book and set the title attribute to ' inferno ' using the set_title() method.

In response, print the value of the title attribute of the book instance.

Expected result:

Python OOPS Exercises

```
class Book:
    language = 'ENG'
    is_ebook = True
```

```python
#Edcorner Learning Pythons OOPS Exercises


class Book:
    language = 'ENG'
    is_ebook = True

    def set_title(self, value):
        if not isinstance(value, str):
            raise TypeError('The value of
the title attribute must be of str '
                'type.')
        self.title = value


book = Book()
book.set_title('Python OOPS Exercises')
print(book.title)
```

```
Python OOPS Exercises
```

41. The Book class is defined. A method called set_title( ) was implemented that allows you to set an instance attribute called title. Create an instance of the Book class named book. Then, using the try ... except ... clause, try using the set_title()method to set the value of the title attribute to False . In case of a TypeError , print the error message to the console.

Expected result:

The value of the title attribute must be of str type.

```
class Book:
    language = 'ENG'
    is_ebook = True


    def set_title(self, value):
        if not isinstance(value, str):
```

raise TypeError('The value of the title attribute must be of str '

'type.')

self.title = value

```
#Edcorner Learning Pythons OOPS Exercises


class Book:
    language = 'ENG'
    is_ebook = True

    def set_title(self, value):
        if not isinstance(value, str):
            raise TypeError('The value of
the title attribute must be of str '
                'type.')
        self.title = value


book = Book()

try:
    book.set_title(False)
except TypeError as error:
    print(error)
```

```
The value of the title attribute must be of str type.
```

Module 7 The _Init_() Method

42. Implement a class called Laptop that sets the following instance attributes when creating an instance:

• brand

• model

• price

Then create an instance named laptop with the following attribute values:

• brand = 'Acer'

model = 'Predator'

price = 5490

Tip: Use the special method init

In response, print the value of the _dict_ attribute of the laptop instance.

Expected result:

{'brand ' :'Acer', 'model': 'Predator1, 'price': 5490}

```
#Edcorner Learning Pythons OOPS Exercises


class Laptop:

    def __init__(self, brand, model,
price):
        self.brand = brand
        self.model = model
        self.price = price


laptop = Laptop('Acer', 'Predator', 5490)
print(laptop.__dict__)
```

```
{'brand': 'Acer', 'model': 'Predator', 'price': 5490}
```

43. A class called Laptop was implemented.

Implement a method in the Laptop class called display_instance_attrs() that displays the names of all the attributes of the Laptop instance.

Then create an instance named laptop with the given attribute values:

• brand = 'Dell'

• model = 'Inspiron'

• price = 3699

In response, call display_instance_attrs() method on the laptop instance.

Expected result:

brand

model

price

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

```python
#Edcorner Learning Pythons OOPS Exercises


class Laptop:

    def __init__(self, brand, model,
price):
        self.brand = brand
        self.model = model
        self.price = price

    def display_instance_attrs(self):
        for attr in self.__dict__.keys():
            print(attr)


laptop = Laptop('Dell', 'Inspiron', 3699)
laptop.display_instance_attrs()
```

```
brand
model
price
```

44. A class called Laptop was implemented.

Implement a method in the Laptop class called display_attrs_with_values() , which displays the names of all the attributes of the Laptop class with their values as shown below (attribute name -> attribute value).

Then create an instance named laptop with the following values:

- brand = 'Dell'
- model = 'Inspiron'
- price = 3699

In response, call display_attrs_with_values( ) method on the laptop instance.

Expected result:

brand - Dell

model - Inspiron

price -3699

class Laptop:

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

```python
#Edcorner Learning Pythons OOPS Exercises


class Laptop:

    def __init__(self, brand, model,
price):
        self.brand = brand
        self.model = model
        self.price = price

    def display_attrs_with_values(self):
        for attr in self.__dict__.keys():
            print(f'{attr} ->
{getattr(self, attr)}')


laptop = Laptop('Dell', 'Inspiron', 3699)
laptop.display_attrs_with_values()
```

```
brand -> Dell
model -> Inspiron
price -> 3699
```

45. Implement a class named Vector that takes any number of n-dimensional vector coordinates as arguments when creating an instance (without any validation) and assign to instance attribute named components. Then create two instances with following coordinates:

• (1, 2)

• (4, 5, 2)

and assign to variables v7 and v2 respectively.

In response, print the value of the components attribute for v1 and v2 instance as shown below.

Expected result:

V1 -> (1, 2)

v2 -> (4, 5, 2)

```python
#Edcorner Learning Pythons OOPS Exercises


class Vector:

    def __init__(self, *components):
        self.components = components


v1 = Vector(1, 2)
v2 = Vector(4, 5, 2)

print(f'v1 -> {v1.components}')
print(f'v2 -> {v2.components}')
```

```
v1 -> (1, 2)
v2 -> (4, 5, 2)
```

46. Implement a class called Bucket that takes any number of named arguments (keyword arguments - use **kwargs) when creating an instance. The name of the argument is the name of the instance attribute, and the value for the argument is the value for the instance attribute.
Example:
[IN]: bucket = Bucket(apple=3.5)
[IN]: print(bucket. _dict_)

[OUT]: {'apple': 3.5}

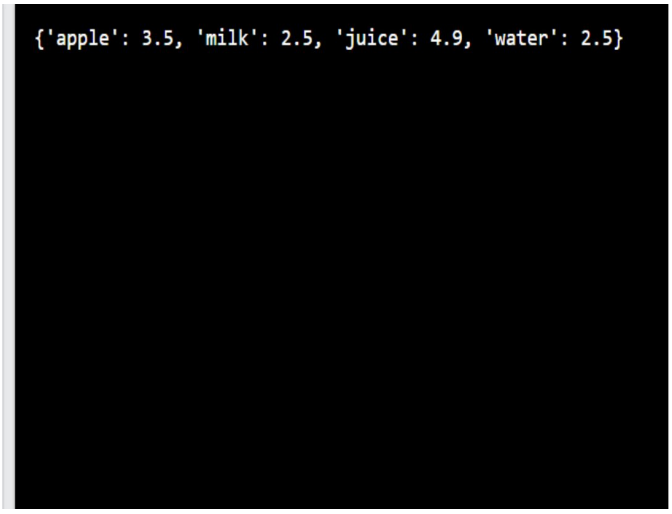
Then create instance named bucket by adding the following attributes with their values:
• apple = 3.5

- milk = 2.5
- juice = 4.9
- water = 2.5

In response, print the value of _dict_ attribute for the bucket instance.

Expected result:
{'apple': 3.5, 'milk': 2.5, 'juice': 4.9, 'water1: 2.5}

```
#Edcorner Learning Pythons OOPS Exercises


class Bucket:

    def __init__(self, **kwargs):
        for attr_name, attr_value in
kwargs.items():
            setattr(self, attr_name,
attr_value)


bucket = Bucket(apple=3.5, milk=2.5,
juice=4.9, water=2.5)
print(bucket.__dict__)
```



47. Implement a class called Car that sets the following instance attributes when creating an instance:
• brand
• model
• price
• type_of_car, by default ‘sedan '
Then create an instance named car with the given values:
• brand = 'Opel'

• model = 'Insignia'
• price = 115000
In response, print the value of the _dict_ attribute of the car instance.

Expected result:

{'brand': 'Opel', 'model': 'Insignia', 'price': 115000, 'type_of_car': 'sedan'}

```python
#Edcorner Learning Pythons OOPS Exercises


class Car:

    def __init__(self, brand, model,
price, type_of_car=None):
        self.brand = brand
        self.model = model
        self.price = price
        self.type_of_car = type_of_car if
type_of_car else 'sedan'


car = Car('Opel', 'Insignia', 115000)
print(car.__dict__)
```

```
{'brand': 'Opel', 'model': 'Insignia', 'price': 115000, 'type_of_car': 'sedan'}
```

48. Implement a class called Car that sets the following instance attributes when creating an instance:
• brand
• model
• price
• type_of_car, by default ‘sedan '
Then create an instance named car with the given values:

brand = 'BMW
• model = 'X3'
• price = 200000
• type_of_car = 'SUV'
In response, print the value of the diet attribute of the car instance.

Expected result:

{'brand': 'BMW, 'model': 'X3', 'price': 200000, 'type_of_car': 'SUV'}

```
class Car:

    def __init__(self, brand, model, price, type_of_car=None):
        self.brand = brand
        self.model = model
        self.price = price
self.type_of_car = type_of_car if type_of_car else 'sedan'
```

```python
#Edcorner Learning Pythons OOPS Exercises


class Car:

    def __init__(self, brand, model,
price, type_of_car=None):
        self.brand = brand
        self.model = model
        self.price = price
        self.type_of_car = type_of_car if
type_of_car else 'sedan'


car = Car('BMW', 'X3', 200000, 'SUV')
print(car.__dict__)
```

```
{'brand': 'BMW', 'model': 'X3', 'price': 200000, 'type_of_car': 'SUV'}
```

49. Implement a class called Laptop that sets the following instance attributes when creating an instance:

- brand
- model
- price

When creating an instance, add validation for the price attribute. The value of the price attribute must be an int or float type greater than zero. If it is not, raise the TypeError with the following message:

The price attribute must be a positive int or float.'

Then create an instance called laptop with the given attributes:

• brand = 'Acer'

• model = 'Predator'

• price = 5490

In response, print the value of the _dict_ attribute of the laptop instance.

Expected result:

{'brand ' :'Acer', 'model': 'Predator', 'price': 5490}

```python
#Edcorner Learning Pythons OOPS Exercises


class Laptop:

    def __init__(self, brand, model,
price):
        self.brand = brand
        self.model = model
        if isinstance(price, (int,
float)) and price > 0:
            self.price = price
        else:
            raise TypeError('The price
attribute must be a positive int or
float.')


laptop = Laptop('Acer', 'Predator', 5490)
print(laptop.__dict__)
```

```
{'brand': 'Acer', 'model': 'Predator', 'price': 5490}
```

50. A class called Laptop was implemented.

Try to create an instance named laptop with the given attribute values:

• brand = 'Acer'

• model = 'Predator'

• price = '5900'

Note that in this case the value of the price attribute is passed as a str type. In case of error, print the error message to the console (use the try ... except ... clause).

Expected result:

The price attribute must be a positive int or float.

```
class Laptop:


    def __init__(self, brand, model, price):
        self.brand = brand
        self.model = model
        if isinstance(price, (int, float)) and price > 0:
            self.price = price
        else:
            raise TypeError('The price attribute must be a positive int or float.')
```

```python
#Edcorner Learning Pythons OOPS Exercises


class Laptop:

    def __init__(self, brand, model,
price):
        self.brand = brand
        self.model = model
        if isinstance(price, (int,
float)) and price > 0:
            self.price = price
        else:
            raise TypeError('The price
attribute must be a positive int or '
                'float.')


try:
    laptop = Laptop('Acer', 'Predator',
'5900')
except TypeError as error:
    print(error)
```

```
The price attribute must be a positive int or float.
```

Module 8 Visibility of Variables

51. Implement a class called Laptop that sets the following instance attributes when creating an instance:

• brand as a bare instance attribute

• model as a protected attribute

• price as a private attribute

Then create an instance named laptop with the following arguments:

- 'Acer'
- 'Predator'

• 5490

In response, print the value of the _dict_ attribute of the laptop instance.

Expected result:

{'brand': 'Acer', '_model': 'Predator', '_Laptop__price': 5490}

```python
#Edcorner Learning Pythons OOPS Exercises


class Laptop:

    def __init__(self, brand, model,
price):
        self.brand = brand
        self._model = model
        self.__price = price


laptop = Laptop('Acer', 'Predator', 5490)
print(laptop.__dict__)
```

```
{'brand': 'Acer', '_model': 'Predator', '_Laptop__price': 5490}
```

52. A class called Laptop was implemented. Then, an instance of the Laptop class named laptop was created with the following arguments:

• 'Acer'

• 'Predator'

• 5490

In response, print the value for each instance attribute (on a separate line) of the laptop instance as shown below.

Expected result:

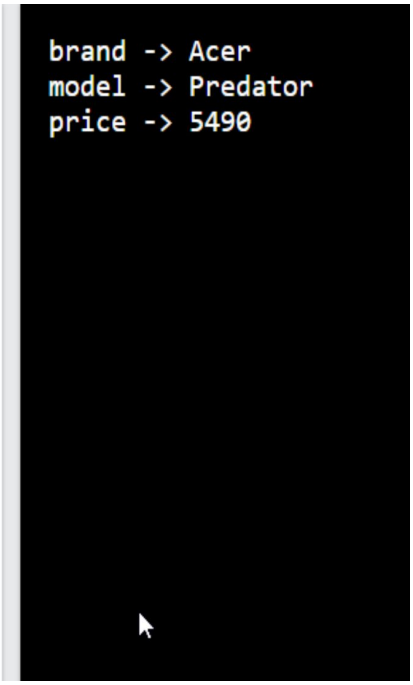
brand -> Acer

model -> Predator

price -> 5490

```
class Laptop:

   def __init__(self, brand, model, price):
      self.brand = brand
      self._model = model
      self.__price = price

laptop = Laptop('Acer', 'Predator', 5490)
```

```
#Edcorner Learning Pythons OOPS Exercises


class Laptop:

    def __init__(self, brand, model,
price):
        self.brand = brand
        self._model = model
        self.__price = price


laptop = Laptop('Acer', 'Predator', 5490)
print(f'brand -> {laptop.brand}')
print(f'model -> {laptop._model}')
print(f'price ->
{laptop._Laptop__price}')
```



53. An implementation of the Laptop class is given. Implement a method in the Laptop class called dispiay_private_attrs( ) that displays the names of all private attributes of the instance. Then create an instance with the given arguments:

• 'Acer'

• 'Predator'

• 'AC-100'

• 5490

- 0.2

and assign it to the variable laptop. In response, call dispiay_private_attrs() on the laptop instance.

Expected result:

_Laptop__price

_Laptop__margin

```
    def __init__(self, brand, model, code, price, margin):
        self.brand = brand
        self._model = model
        self._code = code
        self.__price = price
        self.__margin = margin
```

```python
#Edcorner Learning Pythons OOPS Exercises


class Laptop:

    def __init__(self, brand, model,
code, price, margin):
        self.brand = brand
        self._model = model
        self._code = code
        self.__price = price
        self.__margin = margin

    def display_private_attrs(self):
        for attr in self.__dict__:
            if
attr.startswith(f'_{self.__class__.__name
__}__'):
                print(attr)


laptop = Laptop('Acer', 'Predator', 'AC-
100', 5490, 0.2)
laptop.display_private_attrs()
```

```
_Laptop__price
_Laptop__margin
```

54. An implementation of the Laptop class is given. Implement a method in the Laptop class called dispay_private_attrs( ) that displays the names of all private attributes of the instance. Then create an instance with the given arguments:

• 'Acer'

• 'Predator'

• 'AC-100'

• 5490

- 0.2

and assign it to the variable laptop. In response, call display_protected_attrs( ) on the laptop instance.

Expected result:

_model

_code

```
class Laptop:
    def __init__(self, brand, model, code, price, margin):
        self.brand = brand
        self._model = model
        self._code = code
        self.__price = price
        self.__margin = margin
```

```python
#Edcorner Learning Pythons OOPS Exercises


class Laptop:

    def __init__(self, brand, model,
code, price, margin):
        self.brand = brand
        self._model = model
        self._code = code
        self.__price = price
        self.__margin = margin

    def display_protected_attrs(self):
        for attr in self.__dict__:
            if attr.startswith('_') and \
                not
attr.startswith(f'_{self.__class__.__name
__}'):
                print(attr)


laptop = Laptop('Acer', 'Predator', 'AC-
100', 5490, 0.2)
laptop.display_protected_attrs()
```

```
_model
_code
```

Module 9 Encapsulation

55. Implement a class called Laptop which in the init ( ) method sets the value of the price

protected attribute that stores the price of the laptop (without any validation). Then implement a method to read that attribute named get_price() and a method to modify that attribute named set_price( ) without validation as well.

Then create an instance of the Laptop class with a price of 3499 and follow these steps:

• using the get_price( ) method print the value of the price protected attribute to the console

• using the set_price( ) method, set the value of the price protected attribute to 3999

• using the get_price( ) method print the value of the price protected attribute to th e console

Expected result:

3499

3999

```python
#Edcorner Learning Pythons OOPS Exercises


class Laptop:

    def __init__(self, price):
        self._price = price

    def get_price(self):
        return self._price

    def set_price(self, value):
        self._price = value


laptop = Laptop(3499)
print(laptop.get_price())
laptop.set_price(3999)
print(laptop.get_price())
```

```
3499
3999
```

56. A class called Laptop was implemented. Implement a method named set_price() to modify price attribute that validates the value. Validation checks:

• whether the value is an int or float type, if it is not raise a TypeError with the following message:

'The price attribute must be an int or float type.'

whether the value is positive, if it is not raise vaiueError

with the following message:

The price attribute must be a positive int or float value.'

Then create an instance of the Laptop class with a price of 3499 and try to set 1 - 3000 ■ to the price using set_price( ) method. If an error is raised, print the error message to the console. Use a try ... except ... clause in your solution.

Expected result:

The price attribute must be an int or float type.

```
class Laptop:


    def __init__(self, price):
        self._price = price


    def get_price(self):
        return self._price
```

```python
#Edcorner Learning Pythons OOPS Exercises


class Laptop:

    def __init__(self, price):
        self._price = price

    def get_price(self):
        return self._price

    def set_price(self, value):

        if not isinstance(value, (int,
float)):
            raise TypeError('The price
attribute must be an int or float type.')

        if not value > 0:
            raise ValueError('The price
attribute must be a positive int or '
                'float value.')

        self._price = value


laptop = Laptop(3499)
try:
    laptop.set_price('-3000')
except TypeError as error:
    print(error)
```

```
The price attribute must be an int or float type.
```

57. A class called Laptop was implemented. The _init_( ) method sets the value of the price

protected attribute that stores the price of the laptop (without any validation).

Create an instance of the Laptop class with a price of 3499 and try to set the price to -3000 using the set_price( ) method. If an error is raised, print the error message to the console. Use a try ... except ... clause in your solution.

Expected result:

The price attribute must be a positive int or float value.

```
class Laptop:

    def __init__(self, price):
        self._price = price

    def get_price(self):
        return self._price

    def set_price(self, value):
        if not isinstance(value, (int, float)):
            raise TypeError('The price attribute must be an int or float type.')




        if not value > 0:
            raise ValueError('The price attribute must be a positive int or '
                'float value.')
        self._price = value
```

```
#Edcorner Learning Pythons OOPS Exercises


class Laptop:

    def __init__(self, price):
        self._price = price

    def get_price(self):
        return self._price

    def set_price(self, value):
        if not isinstance(value, (int,
float)):
            raise TypeError('The price
attribute must be an int or float type.')

        if not value > 0:
            raise ValueError('The price
attribute must be a positive int or '
                 'float value.')

        self._price = value


laptop = Laptop(3499)
try:
    laptop.set_price(-3000)
except ValueError as error:
    print(error)
```

```
The price attribute must be a positive int or float value.
```

58. A class called Laptop was implemented.

Add validation of the price attribute also at the stage of creating the instance (in _init_( )

method).

Then try to create an instance of the Laptop class with a price of -3499. If an error is raised, print the error message to the console. Use a try ... except ... clause in your solution.

Expected result:

The price attribute must be a positive int or float value.

class Laptop:

def __init__(self, price):

```
    self._price = price

def get_price(self):
    return self._price

def set_price(self, value):

    if not isinstance(value, (int, float)):
        raise TypeError('The price attribute must be an int or float type.')

            if not value > 0:
                raise ValueError('The price attribute must be a positive int or '
                    'float value.')

            self._price = value
```

```python
#Edcorner Learning Pythons OOPS Exercises


class Laptop:

    def __init__(self, price):
        self.set_price(price)

    def get_price(self):
        return self._price

    def set_price(self, value):

        if not isinstance(value, (int,
float)):
            raise TypeError('The price
attribute must be an int or float type.')

        if not value > 0:
            raise ValueError('The price
attribute must be a positive int or '
                 'float value.')

        self._price = value


try:
    laptop = Laptop(-3499)
except ValueError as error:
    print(error)
```

```
The price attribute must be a positive int or float value.
```

59. Implement a class named Person that has one protected instance attribute named first_name. Next, implement a method get_first_name( ) which reads the value of the firstname protected attribute. Then, using the get_first_name( ) method and the property class (do it in the standard way) create a property named firstname (read-only property).

Create an instance of the Person class and set the firstname attribute to ' john ' . Print the value of the firstname attribute of this instance to the console.

Expected result:

John

```python
#Edcorner Learning Pythons OOPS Exercises


class Person:

    def __init__(self, first_name):
        self._first_name = first_name

    def get_first_name(self):
        return self._first_name

    first_name =
property(fget=get_first_name)


person = Person('John')
print(person.first_name)
```

```
John
```

60 . Implement a class named Person that has two instance protected attributes named first_name and lastname, respectively. Then implement methods named get_first_name() and get_last_name( ) , which reads the protected attributes: firstname and lasLname.

Then, using the get_first_name() and get_last_name() methods and the property class (do it in the standard way) create two properties named firstname and lastname (read-only properties).

Create an instance of the Person class and set the following attributes:

• firstname to the value 'John'

• lastname to the value 'Dow'

Print the value of the firstname and last_name attribute of this instance to the console.

Expected result:

John

Dow

```python
#Edcorner Learning Pythons OOPS Exercises


class Person:

    def __init__(self, first_name,
last_name):
        self._first_name = first_name
        self._last_name = last_name

    def get_first_name(self):
        return self._first_name

    def get_last_name(self):
        return self._last_name

    first_name =
property(fget=get_first_name)
    last_name =
property(fget=get_last_name)


person = Person('John', 'Dow')
print(person.first_name)
print(person.last_name)
```

```
John
Dow
```

61 . Implement a class named Person that has two instance protected attributes named first_name and lastname, respectively. Then implement methods named get_first_name() and get_last_name( ) , which reads the protected attributes: firstname and lasLname.

Then, using the get_first_name() and get_last_name() methods and the property class (do it in the standard way) create two properties named firstname and lastname (read-only properties).

Create an instance of the Person class and set the first_name attribute to ■ Dohn ' . Then, using the set_first_name() method, Set new value 'Mike' .

In response, print the value of the first_name attribute to the console.

Expected result:

Mike

```python
#Edcorner Learning Pythons OOPS Exercises


class Person:

    def __init__(self, first_name):
        self._first_name = first_name

    def get_first_name(self):
        return self._first_name

    def set_first_name(self, value):
        self._first_name = value

    first_name =
property(fget=get_first_name,
fset=set_first_name)


person = Person('John')
person.set_first_name('Mike')
print(person.first_name)
```

```
Mike
```

62. Implement a class named Person that has two instance protected attributes named first_name and lastname, respectively. Then implement methods named get_first_name() and get_last_name( ) , which reads the protected attributes: firstname and lastname.

Then, using the get_first_name() and get_last_name() methods and the property class (do it in the standard way) create two properties named firstname and lastname (read-only properties).

Create an instance of the Person class with the following values:

• first_name = 'John'

• last_name = 'Dow'

Then print the values of these attributes to the console as shown below.

Using the dot notation, modify the attribute values for this instance, respectively:

• firstname to the value 'Tom'

• lastname to the value 'Smith'

In response, print the _dict_ attribute of the created instance to the console.

Expected result:

John

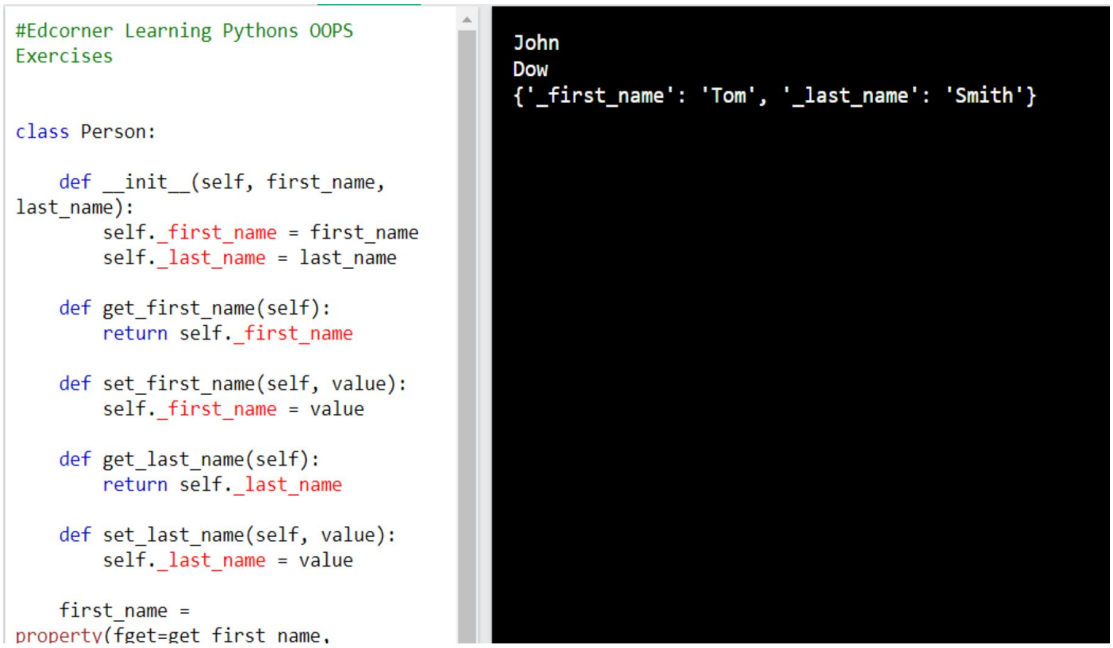
Dow

{'_first_name': 'Tom', '_last_name': 'Smith'}

Solution:

```
class Person:

    def __init__(self, first_name, last_name):
        self._first_name = first_name
        self._last_name = last_name

    def get_first_name(self):
        return self._first_name

    def set_first_name(self, value):
        self._first_name = value
```

```
def get_last_name(self):
    return self._last_name

def set_last_name(self, value):
    self._last_name = value




first_name = property(fget=get_first_name, fset=set_first_name)


     last_name = property(fget=get_last_name, fset=set_last_name)


    person = Person('John', 'Dow')
    print(person.first_name)
    print(person.last_name)

    person.first_name = 'Tom'
    person.last_name = 'Smith'
    print(person.__dict__)
```



```
#Edcorner Learning Pythons OOPS
Exercises


class Person:

    def __init__(self, first_name,
last_name):
        self._first_name = first_name
        self._last_name = last_name

    def get_first_name(self):
        return self._first_name

    def set_first_name(self, value):
        self._first_name = value

    def get_last_name(self):
        return self._last_name

    def set_last_name(self, value):
        self._last_name = value

    first_name =
property(fget=get first name,
```

```
John
Dow
{'_first_name': 'Tom', '_last_name': 'Smith'}
```

63. A class named Person was implemented.

Implement the del_first_name() method to remove the firstname protected attribute.

Then, Using the methods get_first_name() , set_first_name() , del_first_name() and the property class (do this in the standard way) create property named firstname (properties to read, modify and delete).

Create an instance of the Person class named person and assign the value ' Tom1 to firstname. Use the del_first_name() method to delete the firstname attribute of the person instance. Display the _dict_ attribute of the person instance to the console.

Expected result:

{}

``class Person:

```
    def __init__(self, first_name):
        self._first_name = first_name
    def get_first_name(self):
        return self._first_name
    def set_first_name(self, value):
        self._first_name = value
```

```
#Edcorner Learning Pythons OOPS Exercises


class Person:

    def __init__(self, first_name):
        self._first_name = first_name

    def get_first_name(self):
        return self._first_name

    def set_first_name(self, value):
        self._first_name = value

    def del_first_name(self):
        del self._first_name

    first_name =
property(fget=get_first_name,

 fset=set_first_name,

 fdel=del_first_name)


person = Person('John')
person.del_first_name()
print(person.__dict__)
```

```
{}
```

64. Implement a class named Pet that has one protected instance attribute name. Then implement a method name( ) which reads the value of the protected name attribute.

Create a property name (read-only) using the @property decorator.

Create an instance of the Pet class named pet and set name attribute to 'Max' . In response, print the contents of the _dict_ attribute of this instance.

Expected result:

{'_name': 'Max'}

```python
#Edcorner Learning Pythons OOPS Exercises


class Pet:

    def __init__(self, name):
        self._name = name

    @property
    def name(self):
        return self._name


pet = Pet('Max')
print(pet.__dict__)
```

```
{'_name': 'Max'}
```

65. Implement a class named Pet that has two protected instance attributes: name and age, respectively. Next implement the methods: name() and age(), which reads the value of the protected attributes: name and age.

Using the @property decorator, create properties: name and age, respectively (read-only properties).

Create an instance of the Pet class named pet and set the name attribute to ' Max ' and age to

In response, print the contents of the _dict_ attribute of pet instance to the console.

Expected result:

{'_name': 'Max', '_age': 5}

```python
#Edcorner Learning Pythons OOPS Exercises


class Pet:

    def __init__(self, name, age):
        self._name = name
        self._age = age

    @property
    def name(self):
        return self._name

    @property
    def age(self):
        return self._age


pet = Pet('Max', 5)
print(pet.__dict__)
```

```
{'_name': 'Max', '_age': 5}
```

66. Implement a class named Pet that has one protected instance attribute name. Then, using the @property decorator, create a property name (property to read and modify, without validation).

Create an instance of the Pet class named pet and set the name attribute to 'Max'. Then, using dot notation, modify the value of the name attribute to 'Oscar'.

In response, print the contents of the _dict_ attribute of this instance to the console.

Expected result:

{'_name': 'Oscar'}

```python
#Edcorner Learning Pythons OOPS Exercises


class Pet:

    def __init__(self, name):
        self._name = name

    @property
    def name(self):
        return self._name

    @name.setter
    def name(self, value):
        self._name = value


pet = Pet('Max')
pet.name = 'Oscar'
print(pet.__dict__)
```

```
{'_name': 'Oscar'}
```

67. Implement a class named Pet that has one protected instance attribute name. Then, using the @property decorator, create a property name (property to read and modify, without validation).

Create an instance of the Pet class with the name pet and attributes:

• name = 'Max'

• age = 5

Print the _dict_ attribute of the pet instance to the console. Then modify the attributes using the dot notation:

• name to the value 'Tom'

• age to the value 8

Again, print the _dict_ attribute of the pet instance to the console again.

Expected result:

{'_name': 'Max', '_age1 : 5}

{'_name': 'Ton', '_age1 : 8}

Exercises

```python
class Pet:

    def __init__(self, name, age):
        self._name = name
        self._age = age

    @property
    def name(self):
        return self._name

    @name.setter
    def name(self, value):
        self._name = value

    @property
    def age(self):
        return self._age

    @age.setter
    def age(self, value):
        self._age = value


pet = Pet('Max', 5)
print(pet.__dict__)

pet.name = 'Tom'
pet.age = 8
print(pet.__dict__)
```

```
{'_name': 'Max', '_age': 5}
{'_name': 'Tom', '_age': 8}
```

68. A class called Pet is implemented that has two properties: name and age (see below). Add validation to the age property at the stage of object creation and attribute modification:

• the value of the age attribute must be an int type, otherwise raise a TypeEmor with the following message:

'The value of age must be of type int.'

the value of the age attribute must be positive, otherwise raise valueEmor with the

following message:

'The value of age must be a positive integer.'

Then try to create an instance of the Pet class named pet and set the following values:

• 'Max'

• 'seven'

If there is an error, print an error message to the console. Use a try ... except ... clause your solution.

If there is an error, print an error message to the console. Use a try ... except . . clause in your solution.

Expected result:

The value of age must be of type Int.

class Pet:

```
def __init__(self, name, age):
    self._name = name
    self._age = age
@property
def name(self):
    return self._name

@name.setter
def name(self, value):
    self._name = value
@property
def age(self):
    return self._age
@age.setter
def age(self, value):
    self._age = value
```

Solution:

```
class Pet:

    def __init__(self, name, age):
        self._name = name
        self.age = age
```

```
    @property
    def name(self):
        return self._name

    @name.setter
    def name(self, value):
        self._name = value

    @property
    def age(self):
        return self._age



    @age.setter
    def age(self, value):
        if not isinstance(value, int):
            raise TypeError('The value of age must be of type int.')
        if not value > 0:
            raise ValueError('The value of age must be a positive integer.')
        self._age = value


try:
    pet = Pet('Max', 'seven')
except TypeError as error:
```

```
    print(error)
except ValueError as error:
    print(error)
```

```python
#Edcorner Learning Pythons OOPS
Exercises


class Pet:

    def __init__(self, name, age):
        self._name = name
        self.age = age

    @property
    def name(self):
        return self._name

    @name.setter
    def name(self, value):
        self._name = value

    @property
    def age(self):
        return self._age

    @age.setter
    def age(self, value):
        if not isinstance(value, int):
            raise TypeError('The value
of age must be of type int.')
        if not value > 0:
            raise ValueError('The value
of age must be a positive integer.')
```

```
The value of age must be of type int.
```

69. A class called Pet is implemented that has two properties: name and age (see below). Create an instance of the Pet class with the name pet and attribute values respectively:

• 'Max'

• 7

Then try to modify the value of the age attribute to -10. If there is an error, print this error message to the console. Use a try ... except ... clause in your solution.

Expected result:

The value of age must be a positive integer.

```
class Pet:


    def __init__(self, name, age):
        self._name = name
        self.age = age


    @property
    def name(self):
        return self._name


    @name.setter
            def name(self, value):
                self._name = value
```

```
    @property
    def age(self):
        return self._age

    @age.setter
    def age(self, value):
        if not isinstance(value, int):
            raise TypeError('The value of age must be of type int.')
        if not value > 0:
            raise ValueError('The value of age must be a positive integer.')
        self._age = value
```

Solution:

```
class Pet:

    def __init__(self, name, age):
        self._name = name
        self.age = age
```

```
    @property
    def name(self):
        return self._name

    @name.setter
    def name(self, value):
        self._name = value

    @property
    def age(self):
        return self._age

    @age.setter
    def age(self, value):
        if not isinstance(value, int):
            raise TypeError('The value of age must be of type int.')
        if not value > 0:
            raise ValueError('The value of age must be a positive integer.')
        self._age = value


pet = Pet('Max', 7)

try:
    pet.age = -10
except TypeError as error:
```

```
        print(error)
    except ValueError as error:
        print(error)
```

```
#Edcorner Learning Pythons OOPS Exercises


class Pet:

    def __init__(self, name, age):
        self._name = name
        self.age = age

    @property
    def name(self):
        return self._name

    @name.setter
    def name(self, value):
        self._name = value

    @property
    def age(self):
        return self._age

    @age.setter
    def age(self, value):
        if not isinstance(value, int):
            raise TypeError('The value of age
must be of type int.')
        if not value > 0:
            raise ValueError('The value of
age must be a positive integer.')
        self._age = value
```

```
The value of age must be a positive integer.
```

70. Implement a class named TechStack that has one protected instance attribute named tech_names. Then, using the @property decorator, create a property named tech_names (read, modify, and delete property, without validation).

Create an instance of the class named tech_stack and the tech_names attribute value:

• 'python,java,sql'

Print the content of the tech_names attribute. Then, modify this attribute to value:

• ‘python,sql'

Also print the contents of the tech_names attribute to the console.

Remove the tech_names attribute of the tech_stack instance.

Print the contents of the diet attribute of the tech_stack instance to the console.

Expected result:

python,java,sql

python,sql

{}

```python
#Edcorner Learning Pythons OOPS Exercises


class TechStack:

    def __init__(self, tech_names):
        self._tech_names = tech_names

    @property
    def tech_names(self):
        return self._tech_names

    @tech_names.setter
    def tech_names(self, value):
        self._tech_names = value

    @tech_names.deleter
    def tech_names(self):
        del self._tech_names


tech_stack = TechStack('python,java,sql')
print(tech_stack.tech_names)

tech_stack.tech_names = 'python,sql'
print(tech_stack.tech_names)

del tech_stack.tech_names
print(tech_stack.__dict__)
```

```
python,java,sql
python,sql
{}
```

71. Implement a class Game that has a property named level (read and modify property, defaults to 0). The value of the level attribute should be an integer in the range [0, 100] . Add validation at the instance creation and attribute modification stage. If the value is not of the int type, raise a TypeError with the following message:

'The value of level must be of type int.'

If the value is outside the range [0, 100] , set the exceeded boundary value (0 or 100 respectively). Then create a list called games consisting of four instances of the Game class:

games = [GameQ, Game(10), Game(-lO), Game(120)]

Iterate through the games list and print the value of the level attribute for each instance.

Expected result:

0

10

0

100

```python
#Edcorner Learning Pythons OOPS Exercises


class Game:

    def __init__(self, level=None):
        self.level = level if level else 0

    @property
    def level(self):
        return self._level

    @level.setter
    def level(self, value):
        if not isinstance(value, int):
            raise TypeError('The value of level
must be of type int.')
        if value < 0:
            self._level = 0
        elif value > 100:
            self._level = 100
        else:
            self._level = value


games = [Game(), Game(10), Game(-10),
Game(120)]
for game in games:
    print(game.level)
```

```
0
10
0
100
```

Module 10 Computed Attributes

72. Implement a class named Circle that will have the protected instance attribute radius - the radius of the circle (readable and modifiable property). Use the @property decorator.

Then create an instance named circle with radius=3.

In response, display the _dict_ attribute of circle instance.

Expected result:

{'_radius': 3}

```python
#Edcorner Learning Pythons OOPS Exercises


class Circle:

    def __init__(self, radius):
        self.radius = radius

    @property
    def radius(self):
        return self._radius

    @radius.setter
    def radius(self, value):
        self._radius = value


circle = Circle(3)
print(circle.__dict__)
```

```
{'_radius': 3}
```

73. A class named Circle is given. Add a property called area (read-only) to the class that calculates the area of a circle with a given radius. This property should only be computed at first reading or after modifying the radius attribute. To do this, also modify the way of setting the value of the

radius attribute in the _init_( ) method. Make sure that the value of the area attribute is

recalculated after changing the radius attribute.

Then create an instance named circle with radius=3 .

In response, display the value of the area attribute to the console (round the result to four decimal places).

Expected result:

28.2743

```
import math


class Circle:


    def __init__(self, radius):
        self.radius = radius


    @property
    def radius(self):
        return self._radius


    @radius.setter
    def radius(self, value):
        self._radius = value
```

Solution:

```python
#Edcorner Learning Pythons OOPS Exercises


import math


class Circle:
    def __init__(self, radius):
        self.radius = radius
        self._area = None

    @property
    def radius(self):
        return self._radius

    @radius.setter
    def radius(self, value):
        self._radius = value
        self._area = None

    @property
    def area(self):
        if self._area is None:
            self._area = math.pi * self._radius
* self._radius
        return self._area

circle = Circle(3)
print(f'{circle.area:.4f}')
```

```
28.2743
```

74. A class named Circle is given. Add a property called area (read-only) to the class that calculates the area of a circle with a given radius. This property should only be computed at first reading or after modifying the radius attribute. To do this, also modify the way of setting the value of the

radius attribute in the _init_( ) method. Make sure that the value of the area attribute is

recalculated after changing the radius attribute.

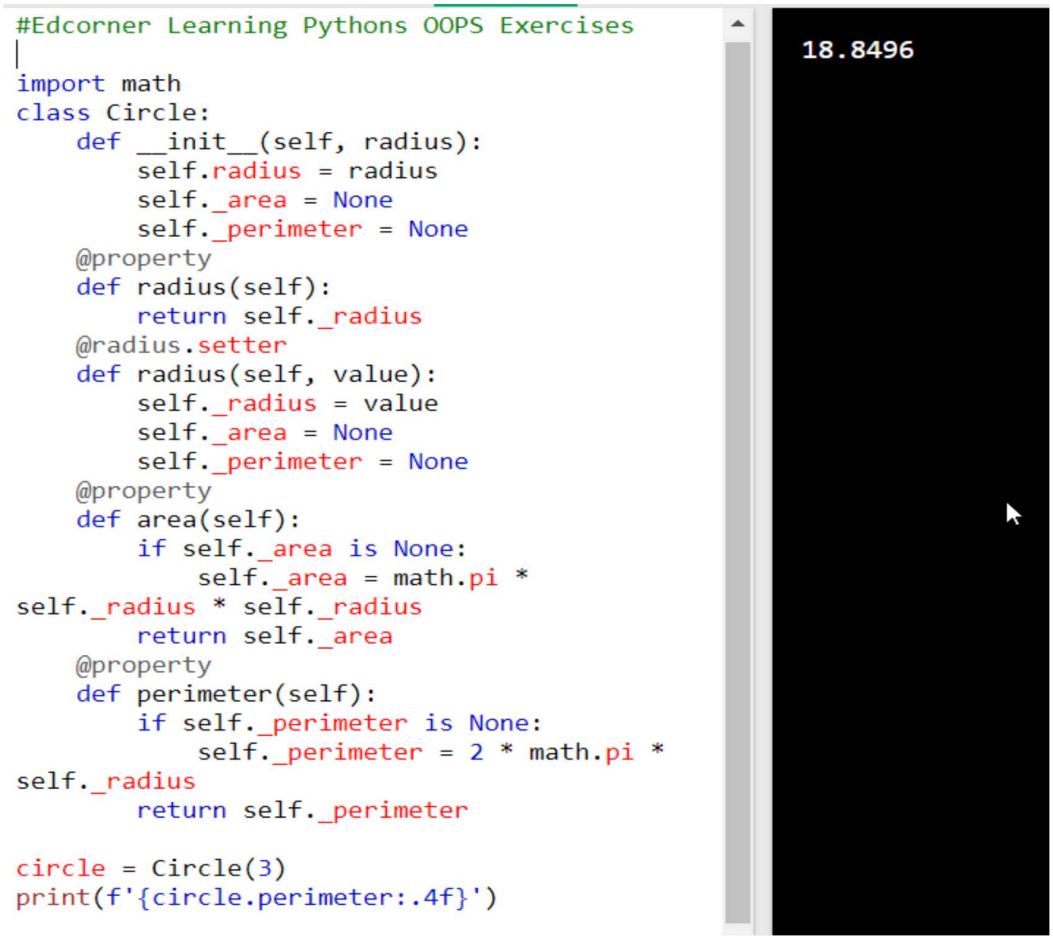
Then create an instance named circle with radius=3 .

In response, display the value of the perimeter attribute to the console (round the result to four decimal places).

Expected result:

18.8496

```
import math
class Circle:
    def __init__(self, radius):
        self.radius = radius
        self._area = None

    @property
    def radius(self):
            return self._radius

        @radius.setter
        def radius(self, value):
            self._radius = value
            self._area = None

        @property
        def area(self):
            if self._area is None:
                self._area = math.pi * self._radius * self._radius
            return self._area
```



```python
#Edcorner Learning Pythons OOPS Exercises

import math
class Circle:
    def __init__(self, radius):
        self.radius = radius
        self._area = None
        self._perimeter = None
    @property
    def radius(self):
        return self._radius
    @radius.setter
    def radius(self, value):
        self._radius = value
        self._area = None
        self._perimeter = None
    @property
    def area(self):
        if self._area is None:
            self._area = math.pi *
self._radius * self._radius
        return self._area
    @property
    def perimeter(self):
        if self._perimeter is None:
            self._perimeter = 2 * math.pi *
self._radius
        return self._perimeter

circle = Circle(3)
print(f'{circle.perimeter:.4f}')
```

```
18.8496
```

75. Implement a class named Rectangle which will have the following properties:

- width
- height

The width and height of the rectangle, respectively (for reading and for modification). Also add a property named area that stores the area of the rectangle (read-only). This property should be computed only at the first

reading or after modifying any of the rectangle sides. Skip attribute validation.

Then create an instance named rectangle with a width = 3 and a height = 4 and print the information about the rectangle instance to the console as shown below.

Expected result:

width: 3, height: 4 -> area: 12

```python
#Edcorner Learning Pythons OOPS Exercises

class Rectangle:
    def __init__(self, width, height):
        self.width = width
        self.height = height
        self._area = None
    @property
    def width(self):
        return self._width
    @width.setter
    def width(self, value):
        self._width = value
        self._area = None
    @property
    def height(self):
        return self._height
    @height.setter
    def height(self, value):
        self._height = value
        self._area = None
    @property
    def area(self):
        if self._area is None:
            self._area = self._width *
self._height
        return self._area
rectangle = Rectangle(3, 4)
print(f'width: {rectangle.width}, height:
{rectangle.height} -> area:
{rectangle.area}')
```

```
width: 3, height: 4 -> area: 12
```

ABOUT THE AUTHOR

“Edcorner Learning” and have a significant number of students on **Udemy** with more than **90000+ Student and Rating of 4.1 or above.**

Edcorner Learning is Part of Edcredibly.

Edcredibly is an online eLearning platform provides Courses on all trending technologies that maximizes learning outcomes and career opportunity for professionals and as well as students. Edcredibly have a significant number of 100000+ students on their own platform and have a **Rating of 4.9 on Google Play Store – Edcredibly App** .

Feel Free to check or join our courses on:

Edcredibly Website - https://www.edcredibly.com/

Edcredibly App – https://play.google.com/store/apps/details?id=com.edcredibly.courses

Edcorner Learning Udemy - https://www.udemy.com/user/edcorner/

Do check our other eBooks available on Kindle Store.