Python Object Oriented Programming Exercises

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

Python object oriented programming exercises are an essential part of mastering the Python programming language. Object-oriented programming (OOP) is a paradigm that uses "objects" to design applications and computer programs. It utilizes several key concepts such as classes, objects, inheritance, encapsulation, and polymorphism, which help structure the code in a more manageable way. In this article, we will explore various exercises designed to strengthen your grasp of OOP in Python, providing a comprehensive guide filled with examples and explanations.

Understanding Object-Oriented Programming in

Python

Before diving into exercises, it's crucial to have a solid understanding of the fundamental concepts of OOP. Here are the core principles:

1. Classes and Objects

- Classes: A class is a blueprint for creating objects. It defines properties (attributes) and behaviors (methods) that the created objects will have.
- Objects: An object is an instance of a class. It encapsulates data and can perform actions defined by its class.

2. Inheritance

- Inheritance allows one class (child class) to inherit the attributes and methods of another (parent class), promoting code reusability and organization.

3. Encapsulation

- Encapsulation involves bundling data and methods that operate on that data within one unit (a class). It restricts direct access to some of the object’s components, which can help prevent accidental interference and misuse.

4. Polymorphism

- Polymorphism allows methods to do different things based on the object it is acting upon, enabling one interface to be used for a general class of actions.

Exercises to Practice OOP in Python

Now that you have a clear understanding of the core concepts, let’s dive into practical exercises to enhance your skills in Python OOP.

Exercise 1: Creating a Basic Class

Objective: Create a simple class called `Car` that has attributes like `make`, `model`, and `year`, along with a method to display the car information.

Steps:
1. Define the `Car` class.
2. Initialize the attributes in the constructor.
3. Create a method to display the car's details.

```python
class Car:
def __init__(self, make, model, year):
self.make = make
self.model = model
self.year = year

def display_info(self):
print(f"{self.year} {self.make} {self.model}")

Create an object of the Car class
my_car = Car("Toyota", "Corolla", 2020)
my_car.display_info()
```

Exercise 2: Implementing Inheritance

Objective: Create a base class called `Animal` and subclasses `Dog` and `Cat` that inherit from it.

Steps:
1. Define the `Animal` class with a method `speak`.
2. Define `Dog` and `Cat` classes that override the `speak` method.

```python
class Animal:
def speak(self):
raise NotImplementedError("Subclasses must implement this method")

class Dog(Animal):
def speak(self):
return "Woof!"

class Cat(Animal):
def speak(self):
return "Meow!"

Create objects of Dog and Cat
dog = Dog()
cat = Cat()
print(dog.speak()) Output: Woof!
print(cat.speak()) Output: Meow!
```

Exercise 3: Encapsulation

Objective: Create a class called `BankAccount` that encapsulates account balance and provides methods to deposit and withdraw funds.

Steps:
1. Define the `BankAccount` class with a private attribute for balance.
2. Provide methods for depositing and withdrawing.

```python
class BankAccount:
def __init__(self, initial_balance=0):
self.__balance = initial_balance

def deposit(self, amount):
if amount > 0:
self.__balance += amount
print(f"Deposited: {amount}")
else:
print("Deposit amount must be positive.")

def withdraw(self, amount):
if 0 < amount <= self.__balance:
self.__balance -= amount
print(f"Withdrew: {amount}")
else:
print("Invalid withdrawal amount.")

def get_balance(self):
return self.__balance

Create an object of BankAccount
account = BankAccount(100)
account.deposit(50)
account.withdraw(30)
print(f"Current Balance: {account.get_balance()}")
```

Exercise 4: Polymorphism with Method Overriding

Objective: Expand on the `Animal` class to demonstrate polymorphism through method overriding.

Steps:
1. Create a method in `Animal` called `info` that provides a description.
2. Override this method in `Dog` and `Cat` classes.

```python
class Animal:
def info(self):
return "I am an animal."

class Dog(Animal):
def info(self):
return "I am a dog."
```

```
class Cat(Animal):
def info(self):
return "I am a cat."

Create a list of animals
animals = [Dog(), Cat()]

for animal in animals:
print(animal.info())
```

Exercise 5: Composition

Objective: Create a `Person` class that uses composition to include a `Car` object.

Steps:
1. Define the `Person` class that has an attribute for a `Car` object.
2. Provide a method to display the person's details including their car.

```python
class Person:
def __init__(self, name, car):
self.name = name
self.car = car

def display_info(self):
print(f"Name: {self.name}, Car: {self.car.make} {self.car.model}")

Create a Car object
my_car = Car("Honda", "Civic", 2018)

Create a Person object
person = Person("Alice", my_car)
person.display_info()
```

Exercise 6: Creating a Simple Library System

Objective: Develop a simple library system using classes for `Book` and `Library`.

Steps:
1. Create a `Book` class with attributes like `title`, `author`, and `isbn`.
2. Create a `Library` class that can add books and display all books.

```python
class Book:
def __init__(self, title, author, isbn):
```

```
self.title = title
self.author = author
self.isbn = isbn

class Library:
def __init__(self):
self.books = []

def add_book(self, book):
self.books.append(book)

def display_books(self):
for book in self.books:
print(f"Title: {book.title}, Author: {book.author}, ISBN: {book.isbn}")

Create Library object
library = Library()

Add books to the library
library.add_book(Book("1984", "George Orwell", "1234567890"))
library.add_book(Book("Brave New World", "Aldous Huxley", "0987654321"))

Display all books
library.display_books()
```

Conclusion

In this article, we explored various Python object oriented programming exercises that cover fundamental OOP concepts such as classes, inheritance, encapsulation, and polymorphism. Each exercise is designed to provide hands-on experience and reinforce your understanding of these principles. By practicing these exercises, you will be well-equipped to implement OOP in Python and tackle more complex programming tasks. Remember that consistent practice is key to mastering object-oriented programming, so keep coding!

Frequently Asked Questions

What is the purpose of using classes in Python object-oriented programming?

Classes in Python provide a means of bundling data and functionality together. Creating a new class creates a new user-defined data structure that can contain its own attributes and methods, allowing for better organization and modularity in code.

How do you create a class in Python?

You can create a class in Python using the 'class' keyword followed by the class name and a colon. For example: 'class MyClass:'. Inside the class, you can define methods and attributes.

What are instance variables and class variables in Python?

Instance variables are attributes that are specific to an instance of a class, defined within methods using 'self'. Class variables are shared across all instances of a class and are defined directly within the class body.

What is inheritance in Python, and how is it implemented?

Inheritance allows a class to inherit attributes and methods from another class. It is implemented by defining a new class that takes the parent class as an argument, like this: 'class ChildClass(ParentClass):'.

Can you explain polymorphism in Python with an example?

Polymorphism allows methods to do different things based on the object calling them. For example, if you have a method 'speak' in both 'Dog' and 'Cat' classes, calling 'speak' on an instance of either class will invoke the respective method implementation.

What is encapsulation, and how can it be achieved in Python?

Encapsulation is the concept of restricting access to certain components of an object. In Python, this can be achieved using private variables by prefixing the variable name with double underscores (e.g., '__private_var').

How do you implement operator overloading in Python?

Operator overloading in Python is implemented by defining special methods in your class that correspond to the operators you want to overload. For example, to overload the '+' operator, you define the '__add__' method in your class.

What are abstract classes in Python, and when would you use them?

Abstract classes in Python are defined using the 'abc' module and cannot be instantiated directly. They are used as blueprints for other classes. You would use them when you have a common interface for a group of related classes but want to enforce certain methods to be implemented in the derived classes.

Python Object Oriented Programming Exercises

What does colon equal (:=) in Python mean? - Stack Overflow
Mar 21, 2023 · In Python this is simply =. To translate this pseudocode into Python you would need to know the data structures being referenced, and a bit more of the algorithm …

What does asterisk * mean in Python? - Stack Overflow
What does asterisk * mean in Python? [duplicate] Asked 16 years, 7 months ago Modified 1 year, 6 months ago Viewed 319k times

What does the "at" (@) symbol do in Python? - Stack Overflow
Jun 17, 2011 · 96 What does the “at” (@) symbol do in Python? @ symbol is a syntactic sugar python provides to utilize decorator, to paraphrase the question, It's exactly about what does …

Is there a "not equal" operator in Python? - Stack Overflow
Jun 16, 2012 · 1 You can use the != operator to check for inequality. Moreover in Python 2 there was <> operator which used to do the same thing, but it has been deprecated in Python 3.

Using or in if statement (Python) - Stack Overflow
Using or in if statement (Python) [duplicate] Asked 7 years, 6 months ago Modified 8 months ago Viewed 149k times

python - What is the purpose of the -m switch? - Stack Overflow
Python 2.4 adds the command line switch -m to allow modules to be located using the Python module namespace for execution as scripts. The motivating examples were standard library …

What is Python's equivalent of && (logical-and) in an if-statement?
Mar 21, 2010 · There is no bitwise negation in Python (just the bitwise inverse operator ~ - but that is not equivalent to not). See also 6.6. Unary arithmetic and bitwise/binary operations and …

syntax - What do >> and <
Apr 3, 2014 · 15 The other case involving print >>obj, "Hello World" is the "print chevron" syntax for the print statement in Python 2 (removed in Python 3, replaced by the file argument of the …

python - Is there a difference between "==" and "is"? - Stack …
Since is for comparing objects and since in Python 3+ every variable such as string interpret as an object, let's see what happened in above paragraphs. In python there is id function that shows …

python - What does ** (double star/asterisk) and * (star/asterisk) …
Aug 31, 2008 · A Python dict, semantically used for keyword argument passing, is arbitrarily ordered. However, in Python 3.6+, keyword arguments are guaranteed to remember insertion …

What does colon equal (:=) in Python mean? - Stack Overflow
Mar 21, 2023 · In Python this is simply =. To translate this pseudocode into Python you would need to know the data structures being referenced, and a bit more of the algorithm …

What does asterisk * mean in Python? - Stack Overflow
What does asterisk * mean in Python? [duplicate] Asked 16 years, 7 months ago Modified 1 year, 6 months ago Viewed 319k times

What does the "at" (@) symbol do in Python? - Stack Overflow
Jun 17, 2011 · 96 What does the “at” (@) symbol do in Python? @ symbol is a syntactic sugar python provides to utilize decorator, to paraphrase the question, It's exactly about what does ...

Is there a "not equal" operator in Python? - Stack Overflow
Jun 16, 2012 · 1 You can use the != operator to check for inequality. Moreover in Python 2 there was <> operator which used to do the same thing, but it has been deprecated in Python 3.

Using or in if statement (Python) - Stack Overflow
Using or in if statement (Python) [duplicate] Asked 7 years, 6 months ago Modified 8 months ago Viewed 149k times

python - What is the purpose of the -m switch? - Stack Overflow
Python 2.4 adds the command line switch -m to allow modules to be located using the Python module namespace for execution as scripts. The motivating examples were standard library ...

What is Python's equivalent of && (logical-and) in an if-statement?
Mar 21, 2010 · There is no bitwise negation in Python (just the bitwise inverse operator ~ - but that is not equivalent to not). See also 6.6. Unary arithmetic and bitwise/binary operations and ...

syntax - What do >> and <
Apr 3, 2014 · 15 The other case involving print >>obj, "Hello World" is the "print chevron" syntax for the print statement in Python 2 (removed in Python 3, replaced by the file argument of the ...

python - Is there a difference between "==" and "is"? - Stack ...
Since is for comparing objects and since in Python 3+ every variable such as string interpret as an object, let's see what happened in above paragraphs. In python there is id function that shows ...

python - What does ** (double star/asterisk) and * (star/asterisk) ...
Aug 31, 2008 · A Python dict, semantically used for keyword argument passing, is arbitrarily ordered. However, in Python 3.6+, keyword arguments are guaranteed to remember insertion ...

Enhance your coding skills with our engaging Python object oriented programming exercises. Learn more and master OOP concepts effectively today!

Back to Home