**Task No. 1:** Design a class Song that internally stores all the metadata of a song.

Implement an abstraction function in the Song class that returns a dictionary with only the song title, artist, and duration.

Demonstrate the functionality by creating a few Song objects and displaying their simplified views.

**Solution:**

class Song:

def \_\_init\_\_(self, title, artist, duration, album=None, release\_year=None):

self.title = title

self.artist = artist

self.duration = duration

self.album = album

self.release\_year = release\_year

def get\_simplified\_view(self):

return {

"Title": self.title,

"Artist": self.artist,

"Duration": self.duration

}

def \_\_str\_\_(self):

return f"{self.title} by {self.artist}"

# Create a few Song objects

song1 = Song("Shape of You", "Ed Sheeran", "3:53", "Divide", 2017)

song2 = Song("Bohemian Rhapsody", "Queen", "5:55", "A Night at the Opera", 1975)

song3 = Song("Imagine", "John Lennon", "3:02", "Imagine", 1971)

# Display the simplified views of the songs

for song in [song1, song2, song3]:

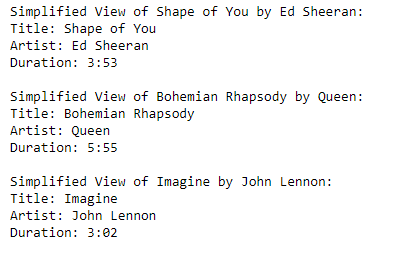
simplified\_view = song.get\_simplified\_view()

print(f"Simplified View of {song}:")

for key, value in simplified\_view.items():

print(f"{key}: {value}")

print()

**Output:**

**Task No. 2:** Design a class Vehicle that internally stores all the specifications of a vehicle.

Implement an abstraction function in the Vehicle class that returns a dictionary with only the vehicle's make, model, and year.

Demonstrate the functionality by creating a few Vehicle objects and displaying their quick overviews.

**Solution:**

class Vehicle:

def \_\_init\_\_(self, make, model, year, color, fuel\_type, engine\_size):

self.make = make

self.model = model

self.year = year

self.color = color

self.fuel\_type = fuel\_type

self.engine\_size = engine\_size

def get\_quick\_overview(self):

return {

"Make": self.make,

"Model": self.model,

"Year": self.year

}

def \_\_str\_\_(self):

return f"{self.year} {self.make} {self.model} ({self.color})"

# Create a few Vehicle objects

vehicle1 = Vehicle("Toyota", "Camry", 2020, "Silver", "Gasoline", "2.5L")

vehicle2 = Vehicle("Ford", "F-150", 2022, "Blue", "Diesel", "3.5L")

vehicle3 = Vehicle("Honda", "Civic", 2019, "Red", "Gasoline", "1.8L")

# Display quick overviews of the vehicles

for vehicle in [vehicle1, vehicle2, vehicle3]:

quick\_overview = vehicle.get\_quick\_overview()

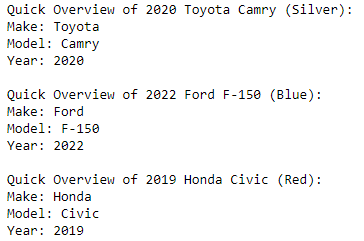
print(f"Quick Overview of {vehicle}:")

for key, value in quick\_overview.items():

print(f"{key}: {value}")

print()

**Output:**



**Task No. 3:** Design a class Product that internally stores all the attributes of a product.

Implement an abstraction function in the Product class that returns a dictionary with only the product name, price, and image.

Demonstrate the functionality by creating a few Product objects and displaying their listings.

**Solution:**

class Product:

def \_\_init\_\_(self, name, price, description, image\_url, category):

self.name = name

self.price = price

self.description = description

self.image\_url = image\_url

self.category = category

def get\_product\_listing(self):

return {

"Product Name": self.name,

"Price": self.price,

"Image": self.image\_url

}

def \_\_str\_\_(self):

return f"{self.name} - ${self.price}"

# Create a few Product objects

product1 = Product("Laptop", 899.99, "15-inch laptop with high-performance specs.", "laptop\_image.jpg", "Electronics")

product2 = Product("Coffee Maker", 49.99, "Programmable coffee maker for your morning brew.", "coffee\_maker\_image.jpg", "Appliances")

product3 = Product("Sneakers", 79.99, "Running shoes for active lifestyles.", "sneakers\_image.jpg", "Apparel")

# Display product listings

for product in [product1, product2, product3]:

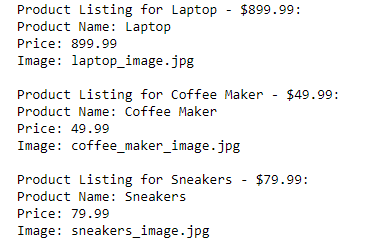
product\_listing = product.get\_product\_listing()

print(f"Product Listing for {product}:")

for key, value in product\_listing.items():

print(f"{key}: {value}")

print()

**Output:**

**Task No. 4:** Design a class Book that stores the ID, title, author, and number of copies available.

Implement rep invariants to ensure that:

The book ID is unique.

The number of copies available is never negative.

Demonstrate the functionality by creating a few Book objects and trying to set negative copies or duplicate IDs.

**Solution:**

class Book:

def \_\_init\_\_(self, book\_id, title, author, copies\_available):

self.book\_id = book\_id

self.title = title

self.author = author

self.copies\_available = copies\_available

def \_\_str\_\_(self):

return f"{self.title} by {self.author} (ID: {self.book\_id}) - Copies Available: {self.copies\_available}"

def set\_copies\_available(self, new\_copies):

if new\_copies >= 0:

self.copies\_available = new\_copies

else:

print("Error: Number of copies available cannot be negative.")

# Create Book objects

book1 = Book(1, "To Kill a Mockingbird", "Harper Lee", 5)

book2 = Book(2, "1984", "George Orwell", 10)

book3 = Book(3, "Pride and Prejudice", "Jane Austen", 3)

# Attempt to set negative copies available

book1.set\_copies\_available(-1) # This should produce an error message.

# Attempt to create a book with a duplicate ID

book4 = Book(1, "The Great Gatsby", "F. Scott Fitzgerald", 2) # This should produce an error message.

# Display book information

for book in [book1, book2, book3]:

print(book)

**Output:**

A close up of a white background

Description automatically generated

**Task No. 5:** Design a class Student that stores the student number, name, and a list of enrolled courses.

Implement rep invariants to ensure that:

The student numbers are unique.

A student is not enrolled in the same course more than once.

Demonstrate the functionality by creating a few Student objects and trying to enroll them in the same course multiple times or assign duplicate student numbers.

**Solution:**

class Student:

def \_\_init\_\_(self, student\_number, name):

self.student\_number = student\_number

self.name = name

self.enrolled\_courses = set()

def \_\_str\_\_(self):

return f"Student {self.student\_number}: {self.name} - Enrolled Courses: {', '.join(self.enrolled\_courses)}"

def enroll(self, course):

if course not in self.enrolled\_courses:

self.enrolled\_courses.add(course)

else:

print(f"Error: {self.name} is already enrolled in {course}.")

# Create Student objects

student1 = Student(1, "Alice")

student2 = Student(2, "Bob")

student3 = Student(3, "Charlie")

# Enroll students in courses

student1.enroll("Math 101")

student1.enroll("English 101")

student2.enroll("Math 101")

student3.enroll("Science 101")

# Attempt to enroll students in the same course multiple times

student1.enroll("Math 101") # This should produce an error message.

# Attempt to assign duplicate student numbers

student4 = Student(1, "David") # This should produce an error message.

# Display student information

for student in [student1, student2, student3]:

print(student)

A black text on a white background

Description automatically generated**Output:**