OOPS

Practical/ Application based:

Write a Python program using OOP to model a bank account system with functionalities like deposit, withdraw, and check balance.

```
class BankAccount:
    def __init__(self, account_number, account_holder, balance=0):
        self.account_number = account_number
        self.account_holder = account_holder
        self.balance = balance
    def deposit(self, amount):
        if amount > 0:
            self.balance += amount
            print(f"Deposited ${amount}. New balance: ${self.balance}")
        else:
            print("Deposit amount must be positive.")
    def withdraw(self, amount):
        if amount > 0:
            if amount <= self.balance:
                self.balance -= amount
                print(f"Withdrew ${amount}. New balance: ${self.balance}")
            else:
                print("Insufficient funds.")
        else:
            print("Withdrawal amount must be positive.")
    def check_balance(self):
        print(f"Current balance: ${self.balance}")
# Example usage:
if __name__ == "__main__":
    # Create a new bank account
    account = BankAccount("12345678", "John Doe", 1000)
    # Check initial balance
    account.check_balance()  # Output: Current balance: $1000
    # Deposit money
    account.deposit(500)     # Output: Deposited $500. New balance: $1500
    # Withdraw money
    account.withdraw(200)    # Output: Withdrew $200. New balance: $1300
    # Check balance after transactions
    account.check_balance()  # Output: Current balance: $1300
```

Given a base class Shape with derived classes Circle and Rectangle, implement a method to calculate the area of these shapes using polymorphism.

```
import math

# Base class
class Shape:
    def area(self):
        raise NotImplementedError("Subclasses must implement this method")

# Derived class for Circle
class Circle(Shape):
    def __init__(self, radius):
        self.radius = radius

    def area(self):
        return math.pi * (self.radius ** 2)

# Derived class for Rectangle
class Rectangle(Shape):
    def __init__(self, width, height):
        self.width = width
        self.height = height

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

# Function to calculate area using polymorphism
def calculate_area(shape):
    return shape.area()

# Creating instances of Circle and Rectangle
circle = Circle(5)
rectangle = Rectangle(4, 6)

# Calculating areas using the calculate_area function
print(f"Area of the circle: {calculate_area(circle):.2f}")  # Output: Area of the circle: 78.54
print(f"Area of the rectangle: {calculate_area(rectangle):.2f}")  # Output: Area of the
rectangle: 24.00
```

Given the following class definition, how would you modify it to include encapsulation for its attributes?

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

self.year = year

```
class Car:
    def __init__(self, make, model, year):
        self.__make = make   # Private attribute
        self.__model = model # Private attribute
        self.__year = year   # Private attribute

    # Getter for make
    @property
    def make(self):
        return self.__make

    # Setter for make
    @make.setter
    def make(self, make):
        self.__make = make

    # Getter for model
    @property
    def model(self):
        return self.__model

    # Setter for model
    @model.setter
    def model(self, model):
        self.__model = model

    # Getter for year
    @property
    def year(self):
        return self.__year

    # Setter for year
    @year.setter
    def year(self, year):
        if year > 1885:  # The first car was invented around 1886
            self.__year = year
        else:
            raise ValueError("Year must be greater than 1885")

# Example usage:
my_car = Car("Toyota", "Corolla", 2020)
print(my_car.make)   # Output: Toyota
print(my_car.model)  # Output: Corolla
print(my_car.year)   # Output: 2020
```

```
my_car.make = "Honda"
my_car.model = "Civic"
my_car.year = 2021
print(my_car.make)   # Output: Honda
print(my_car.model)  # Output: Civic
print(my_car.year)   # Output: 2021

# Trying to set an invalid year
try:
    my_car.year = 1800
except ValueError as e:
    print(e)  # Output: Year must be greater than 1885
```

Extend the Library class to prevent a book from being added if it already exists in the collection.

class Library:
def __init__(self):
self.books = []
def add_book(self, book):
self.books.append(book)

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

    def add_book(self, book):
        if book in self.books:
            print(f"The book '{book}' already exists in the collection.")
        else:
            self.books.append(book)
            print(f"The book '{book}' has been added to the collection.")

    def display_books(self):
        if not self.books:
            print("The library has no books.")
        else:
            print("The library contains the following books:")
            for book in self.books:
                print(f"- {book}")

# Example usage:
library = Library()
library.add_book("The Great Gatsby")
library.add_book("1984")
```

```
library.add_book("The Great Gatsby")  # Should not add this book again
library.display_books()

# Output:
# The book 'The Great Gatsby' has been added to the collection.
# The book '1984' has been added to the collection.
# The book 'The Great Gatsby' already exists in the collection.
# The library contains the following books:
# - The Great Gatsby
# - 1984
```

Regular Expressions

Practical/ Application based:

Write a regular expression to match an email address.

```
r'[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}'
```

Write a Python function to extract all phone numbers from a given text.

```
import re
def extract_phone_numbers(text):
    pattern = r'\b\d{3}[-.]?\d{3}[-.]?\d{4}\b'
    return re.findall(pattern, text)
```

Given a log file, write a Python script to extract lines containing the word "ERROR".

```
import re
def extract_error_lines(logfile):
    with open(logfile, 'r') as file:
        lines = file.readlines()
    error_lines = [line for line in lines if re.search(r'ERROR', line)]
    return error_lines
```

How would you use regular expressions to validate a password with the following rules: at least 8 characters, at least one uppercase letter, one lowercase letter, one digit, and one special character?

```
import re
def validate_password(password):
    pattern =r'^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)(?=.*[@$!%*?&])[A-Za-z\d@$!%*?&]{8,}$'
    return bool(re.match(pattern, password))
```

How do you find the starting and ending positions of a match in a string?

```
import re
def find_match_positions(text, pattern):
    match = re.search(pattern, text)
    if match:
        return match.start(), match.end()
    return None
```

Write a regular expression to find dates in the format DD-MM-YYYY and DD/MM/YYYY.

```
pattern = r'\b\d{2}[-/]\d{2}[-/]\d{4}\b'
```

Write a function to extract all URLs from a given string using regular expressions.

```
import re
def extract_urls(text):
    pattern = r'https?://(?:[-\w.]|(?:%[\da-fA-F]{2}))+'
    return re.findall(pattern, text)
# Example
text = "Visit our website at https://www.example.com and follow us at
http://twitter.com/example."
urls = extract_urls(text)
print(urls)  # Output: ['https://www.example.com', 'http://twitter.com/example']
```

How do you remove all HTML tags from a string using regular expressions?

```
import re

def remove_html_tags(text):
    pattern = r'<.*?>'
    return re.sub(pattern, '', text)

# Example
html_text = "<div>Hello, <b>world</b>! This is <i>HTML</i> text.</div>"
clean_text = remove_html_tags(html_text)
print(clean_text)  # Output: "Hello, world! This is HTML text."
```

How do you extract all hashtags from a given string using regular expressions?

```
import re

def extract_hashtags(text):
    pattern = r'#\w+'
    return re.findall(pattern, text)
```

```
# Example
text = "Here are some hashtags: #Python, #regex, and #coding."
hashtags = extract_hashtags(text)
print(hashtags)  # Output: ['#Python', '#regex', '#coding']
```

Python Packages

Practical/Application based:

How do you perform element-wise addition of two NumPy arrays?

```
import numpy as np
arr1 = np.array([1, 2, 3])
arr2 = np.array([4, 5, 6])
result = arr1 + arr2
# result is array([5, 7, 9])
```

How do you create a pandas DataFrame?

You can create a DataFrame using the pd.DataFrame() function.

```
import pandas as pd
data = {'Name': ['Alice', 'Bob', 'Charlie'], 'Age': [25, 30, 35]}
df = pd.DataFrame(data)
```

Can you describe the process of merging two DataFrames in pandas?

```
merged_df = pd.merge(df1, df2, on='common_column')
```

How do you read an Excel file into a pandas DataFrame?

```
import pandas as pd
df = pd.read_excel('file.xlsx')
```

How do you read a CSV file into a pandas DataFrame and display the first 5 rows?

```
import pandas as pd
df = pd.read_csv('data.csv')
print(df.head())
```

How can you concatenate two DataFrames along rows?

```
df1 = pd.DataFrame({'A': [1, 2], 'B': [3, 4]})
```

```
df2 = pd.DataFrame({'A': [5, 6], 'B': [7, 8]})
concatenated_df = pd.concat([df1, df2], axis=0)
```

How do you set a column as the index of a DataFrame?

```
df.set_index('column_name', inplace=True)
```

How do you create a simple pivot table in pandas?

```
import pandas as pd
data = {
    'A': ['foo', 'foo', 'bar', 'bar'],
    'B': ['one', 'two', 'one', 'two'],
    'C': [1, 2, 3, 4]
}
df = pd.DataFrame(data)
pivot_table = df.pivot_table(values='C', index='A', columns='B', aggfunc='sum')
```

How do you find and remove duplicate rows from a DataFrame?

```
df.drop_duplicates(inplace=True)
```

Given a NumPy array, how do you normalise the data to have values between 0 and 1?

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5])
normalized_arr = (arr - arr.min()) / (arr.max() - arr.min())
```

How do you convert a column of date strings to datetime objects and extract the year?

```
df['date_column'] = pd.to_datetime(df['date_column'])
df['year'] = df['date_column'].dt.year
```