Christian_Herrera_HW5

February 22, 2026

1 HW5. Basic OOP-1 (Questions)

2 Assignment Submission Guidelines

Please follow the guidelines below for submitting your assignment:

1. **Submission Deadline:**
   - All assignments must be submitted **no later than 23:59 PM next Tuesday (02/24)**.
   - Late submissions will not be accepted unless prior arrangements have been made by the TAs.
2. **Complete Your Work:**
   - Finish your work in Google Colab and ensure that your notebook is saved.
3. **Submit Your Assignment on Canvas:**
   - Log in to **Canvas** and navigate to the assignment submission page.
4. **File Naming Convention:**
   - Please name your files as follows: `Lastname_Firstname_AssignmentName`
   - Example: `Alex_John_HW5.ipynb` and `Alex_John_HW5.pdf`
5. **Technical Issues:**
   - If you encounter any technical issues with Canvas or your submission, please contact the TAs immediately **before the deadline** to avoid penalties.

3 Questions

---

4 Question 1: Create a Simple Calculator Class (10 Points)

Define a Python class named `Calculator` that can perform basic arithmetic operations: addition, subtraction, multiplication, and division. The class should include the following methods:

- `add(x, y)`: Returns the sum of `x` and `y`.
- `subtract(x, y)`: Returns the difference between `x` and `y`.
- `multiply(x, y)`: Returns the product of `x` and `y`.
- `divide(x, y)`: Returns the quotient of `x` divided by `y`. This method should handle division by zero by returning a suitable message.

[ ]:
```
class Calculator:
    def __init__(self) -> None:
```

```python
        # The calculator
        pass
    def add(self, x,y):
        return x + y
    def subtract(self, x,y):
        return x - y
    def multiply(self, x,y):
        return x * y
    def divide(self, x,y):
        if y == 0:
            return "Cannot divide by zero." # Illegal.
        else:
            return x / y

# Creating the instance.
calculator1 = Calculator()

# Testing all the functions.
print(calculator1.add(4,2))
print(calculator1.subtract(4,2))
print(calculator1.multiply(4,2))
print(calculator1.divide(4,2))
```

```
6
2
8
2.0
```

5 Question 2: Implement a Book Class (10 Points)

Create a class named `Book` that represents information about a book. The class should include the following attributes:

- `title` (string): The title of the book.
- `author` (string): The author of the book.
- `isbn` (string): The International Standard Book Number (ISBN) of the book.
- `total_pages` (integer): The total number of pages in the book.

5.1 Methods

- `__init__(self, title, author, isbn, total_pages)`: Initializes a new instance of the `Book` class with the specified title, author, ISBN, and total number of pages.
- `get_book_info(self)`: Returns a string containing the book's title, author, and ISBN.
- `is_long_book(self)`: Returns `True` if the book has more than 500 pages, otherwise `False`.

[2]:
```python
class Book:
    def __init__(self, title:str, author:str, isbn:str, total_pages:int):
        # Initializing the class.
        self.title = title
        self.author = author
        self.isbn = isbn
        self.total_pages = int(total_pages)

    def get_book_info(self):
        # Retunrning the book info.
        return f'Title: {self.title}, Author: {self.author}, ISBN: {self.isbn}'

    def is_long_book(self):
        # Is it too long?
        if self.total_pages > 500:
            return True
        else:
            return False

# Creating the class instance.
book1 = Book('How to pet Cats', 'Christian Herrera', '0118 999 881 999 119 725␣
 ↪3', 5020)

# Demonstrating.
print(book1.get_book_info())
print(book1.is_long_book())
```

```
Title: How to pet Cats, Author: Christian Herrera, ISBN: 0118 999 881 999 119
725 3
True
```

6 Question 3: Design a Temperature Converter Class (10 Points)

Design a class named `TemperatureConverter` that converts temperatures between Fahrenheit and Celsius. The class should provide the following methods:

- `fahrenheit_to_celsius(f)`: Converts a temperature from Fahrenheit to Celsius.
- `celsius_to_fahrenheit(c)`: Converts a temperature from Celsius to Fahrenheit.

[3]:
```python
class TemperatureConverter:
    def __init__(self) -> None:
        # Creating the class.
        pass
    def fahrenheit_to_celsius(self, f):
        return (f - 32) * 5 / 9 # f -> c

    def celsius_to_fahrenheit(self, c):
        return (c * 9 / 5) + 32 # c -> f

# Creating the class instance.
converter = TemperatureConverter()

# Demonstrating.
print(converter.fahrenheit_to_celsius(212))
print(converter.celsius_to_fahrenheit(100))
```

```
100.0
212.0
```

---

7 Question 4: Create a Rectangle Class (10 Points)

Define a class named `Rectangle` with two attributes: `length` and `width`. This class includes methods to calculate the rectangle's area and perimeter, and a method to check if the rectangle is a square.

7.1 Attributes

- `length` (float): The length of the rectangle.
- `width` (float): The width of the rectangle.

7.2 Methods

- `__init__(self, length, width)`: Initializes a new instance of the `Rectangle` class with the specified length and width.
- `area(self)`: Calculates and returns the rectangle's area.
- `perimeter(self)`: Calculates and returns the rectangle's perimeter.
- `is_square(self)`: Returns `True` if the rectangle is a square (length equals width), otherwise `False`.

[4]:
```
class Rectangle:
    def __init__(self, length, width):
        # Initializing the class.
        self.length = length
        self.width = width
    def area(self):
        return self.length * self.width # Returning the area.
    def perimeter(self):
        return 2 * (self.length) + 2 * (self.width) # Returning the Perimeter.
    def is_square(self):
        if self.length == self.width: # Square if true.
            return True
        else:
            return False

# Class instance.
rectangle1 = Rectangle(2,4)

# Demonstrating attributes/methods.
print('Length:', rectangle1.length)
print('Width:', rectangle1.width)
print('Area:', rectangle1.area())
print('Perimeter', rectangle1.perimeter())
print('Is a square:', rectangle1.is_square())
```

```
Length: 2
Width: 4
Area: 8
Perimeter 12
Is a square: False
```

8 Question 5: Develop a Student Class (10 Points)

Create a class named `Student` with the following attributes:

- `name` (string): The student's name.
- `age` (integer): The student's age.
- `grades` (list of integers): A list of the student's grades.

8.1 Methods

- `__init__(self, name, age, grades)`: Initializes a new instance of the `Student` class with the specified name, age, and list of grades.
- `average_grade(self)`: Calculates and returns the student's average grade.
- `add_grade(self, grade)`: Adds a new grade to the student's list of grades.
- `get_student_info(self)`: Returns a string with the student's name, age, and average grade.

[5]:
```
class Student:
    def __init__(self, name:str, age:int, grades:list):
        # Initializing the class.
        self.name = name
        self.age = int(age)
        self.grades = list(grades)

    def average_grade(self):
        return sum(self.grades)/len(self.grades) # Average grade.

    def add_grade(self, grade):
        print(f'Added {grade}')
        self.grades.append(grade) # Adding the grade.

    def get_student_info(self):
        return f'Name: {self.name} \nAge: {self.age} \nAverage Grade: {sum(self.
 ↪grades)/len(self.grades)}'

# List of sample grades for the student.
sample_grades = [90, 97, 99, 85, 92]

# Creating the student and passing the sample grades.
student1 = Student('Spongebob', 23, sample_grades)

# Demonstrating the attributes and methods.
print(student1.average_grade())
student1.add_grade(84)
print(student1.get_student_info())
```

```
92.6
Added 84
Name: Spongebob
Age: 23
Average Grade: 91.16666666666667
```

9 Question 6: Employee Management System (10 Points)

Create a class named `Employee` that represents an employee with the following attributes:

- `name` (string): The employee's name.
- `id_number` (string): The employee's identification number.
- `position` (string): The employee's position within the company.
- `salary` (float): The employee's current salary.

9.1 Methods

- `__init__(self, name, id_number, position, salary)`: Initializes a new instance of the `Employee` class with the specified name, identification number, position, and salary.
- `update_position(self, new_position)`: Updates the employee's position to `new_position`.
- `apply_raise(self, percentage)`: Increases the employee's salary by a given percentage if the salary is less than 50,000; otherwise, apply a fixed bonus of 5,000.

[6]:
```python
class Employee:
    def __init__(self, name, id_number:int, position, salary:int):
        # Creating the employee class.
        self.name = name
        self.id_number = int(id_number)
        self.position = position
        self.salary = int(salary)

    def update_position(self, new_position):
        # The promotion.
        print(f'Congratulations! You have been promoted from {self.position} to␣
 ↪{new_position}!')
        self.position = new_position

    def apply_raise(self, percentage):
        # Applying the raise accoring to the instructions.
        old_salary = self.salary

        if self.salary < 50000:
            increase_amount = self.salary * percentage
            self.salary = self.salary + increase_amount
            print(f'Congratulations! Your salary has been increased by␣
 ↪{percentage * 100}%!')
            print(f'Your salary went from {old_salary} to {self.salary}')
        else:
            self.salary = self.salary + 5000
            print('Congratulations! You received a fixed bonus of 5000!')
            print(f'Your salary went from {old_salary} to {self.salary}')

# The best employee.
employee1 = Employee('SpongeBob', 1, 'Frycook', 500)

# Giving him a raise.
employee1.update_position('Cashier')
employee1.apply_raise(.50)
```

```
Congratulations! You have been promoted from Frycook to Cashier!
Congratulations! Your salary has been increased by 50.0%!
Your salary went from 500 to 750.0
```

This cell kept breaking my PDF exporter for some reason, the issue seemed to be the "—-", so I removed them.

10 Question 7: Library Management System (20 Points)

Define a class named `Library` that manages a collection of books. The class should have the following attribute:

- `books` (list of dictionaries): A list where each dictionary represents a book and includes keys for `title`, `author`, and `status` (either `'available'` or `'checked out'`).

10.0.1 Methods:

- `__init__(self)`: Initializes the library's book collection as an empty list.
- `add_book(self, title, author)`: Adds a new book to the collection. Each book should be represented as a dictionary with keys for `title`, `author`, and `status`. The status of a new book should be set to `'available'`.
- `check_out_book(self, title)`: Changes the status of a book to `'checked out'` if it is available. If the book is not found or already checked out, print an appropriate message. If the book is found and its status is `'available'`, update its status to `'checked out'` and print a message confirming that the book has been checked out. If the book is already `'checked out'` or cannot be found in the list, print a message indicating that the book cannot be checked out. **Example**: If the book `'1984'` is available, calling `check_out_book('1984')` will change its status and print: `"Book '1984' has been checked out."` If the book is already checked out, it should print: `"Book '1984' is not available for checkout."`
- `return_book(self, title)`: Changes the status of a book to `'available'` if it is currently checked out. Print a message if the book is not found or is already available.If the book is found and its status is `'checked out'`, update its status to `'available'` and print a message confirming that the book has been returned. If the book is not found or is already `'available'`, print a message indicating that the book cannot be returned. **Example**: If the book `'1984'` is checked out, calling `return_book('1984')` will change its status and print:
- Add a method `display_books(self)` that prints all the books in the library along with their current status.

10.0.2 Example of the `books` list structure:

"'python [{'title': 'Book Title 1', 'author': 'Author Name', 'status': 'available'}, {'title': 'Book Title 2', 'author': 'Author Name', 'status': 'checked out'}]

[7]:
```python
class Library:
    def __init__(self):
        # Initializing.
        self.books = []

    def add_book(self, title, author):
        # Adding book method.
        new_book = {
            'title': title,
            'author': author,
            'status': 'available'
        }
        self.books.append(new_book)
        print(f"Added '{title}' by {author}")

    def check_out_book(self, title):
        # Checkout method.
        for book in self.books:
            if book['title'] == title:
                if book['status'] == 'available':
                    book['status'] = 'checked out'
                    print(f"Book '{title}' has been checked out.")
                    return
                else:
                    print(f"Book '{title}' is not available for checkout.")
                    return

        print(f"Book '{title}' is not available for checkout.") # Not available.
↪

    def return_book(self, title):
        # Giving back the book.
        for book in self.books:
            if book['title'] == title:
                if book['status'] == 'checked out':
                    book['status'] = 'available'
                    print(f"Book '{title}' has been returned.")
                    return
                else:
                    print(f"Book '{title}' cannot be returned.")
                    return

        print(f"Book '{title}' cannot be returned.")

    def display_books(self):
        # What do we have in the library?
        if len(self.books) == 0:
            print('No books in library.')
        else:
            for book in self.books:
                print(f"Title: {book['title']}, Author: {book['author']},␣
↪Status: {book['status']}")

# Creating class instance.
library1 = Library()

# Adding books.
library1.add_book('1984', 'George Orwell')
library1.add_book('The Hobbit', 'J.R.R. Tolkien')
library1.add_book('Dune', 'Frank Herbert')

# What do we have?
library1.display_books()

# Reading a lot of books.
library1.check_out_book('1984')
library1.check_out_book('1984')   # already checked out
library1.check_out_book('Dracula')  # not found

# Now what do we have?
library1.display_books()

library1.return_book('1984')
library1.return_book('1984')   # already available
library1.return_book('Dracula')  # not found

library1.display_books()
```

```
Added '1984' by George Orwell
Added 'The Hobbit' by J.R.R. Tolkien
Added 'Dune' by Frank Herbert
Title: 1984, Author: George Orwell, Status: available
Title: The Hobbit, Author: J.R.R. Tolkien, Status: available
Title: Dune, Author: Frank Herbert, Status: available
Book '1984' has been checked out.
Book '1984' is not available for checkout.
Book 'Dracula' is not available for checkout.
Title: 1984, Author: George Orwell, Status: checked out
Title: The Hobbit, Author: J.R.R. Tolkien, Status: available
Title: Dune, Author: Frank Herbert, Status: available
Book '1984' has been returned.
Book '1984' cannot be returned.
Book 'Dracula' cannot be returned.
Title: 1984, Author: George Orwell, Status: available
Title: The Hobbit, Author: J.R.R. Tolkien, Status: available
Title: Dune, Author: Frank Herbert, Status: available
```

```
[ ]:
```