Herrera_Christian_Lab5

February 20, 2026

1 Lab 5.OOP-1 (Questions)

2 Assignment Submission Guidelines

2.0.1 Steps to Submit Your Lab Assignments Using Google Colab

1. **Submission Deadline:**
   - All assignments must be submitted **no later than 23:59 PM tonight**.
   - 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_Lab5.ipynb` and `Alex_John_Lab5.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

3.0.1 Question 1: (20 Points)

Create a Python class called Student with attributes name and age. Implement a method display to print the student's name and age.

[43]:
```
class Student:
    # Set up the student info
    def __init__(self, name, age):
        self.name = name
        self.age = age

    # Print the student info
    def display(self):
```

```
        print("Student name:", self.name)
        print("Student age:", self.age)


# Make a student object
student1 = Student("Christian", 78)

# Run the method
student1.display()
```

```
Student name: Christian
Student age: 78
```

3.0.2 Question 2: (20 Points)

Define a class `Car` with the following instance attributes initialized via the `__init__` method: `make`, `model`, and `year`.

Create a method within the `Car` class that prints a description of the car (including its make, model, and year).

Instantiate an object of the `Car` class with make "Toyota", model "Corolla", and year 2020, and call the method to print its description.

[44]:
```
class Car:
    # Set up the car info.
    def __init__(self, make, model, year):
        self.make = make
        self.model = model
        self.year = year

    # Print the car info.
    def print_description(self):
        print("Car description:")
        print("Make:", self.make)
        print("Model:", self.model)
        print("Year:", self.year)


# Make a car object.
my_car = Car("Toyota", "Corolla", 2020)

# Run the meethod.
my_car.print_description()
```

```
Car description:
Make: Toyota
Model: Corolla
Year: 2020
```

3.0.3 Question 3: (20 Points)

Implement a Python class called BankAccount with attributes account_number and balance. Implement methods deposit and withdraw to add and subtract money from the account.

[45]:
```python
class BankAccount:
    # Set up account info.
    def __init__(self, account_number, balance):
        self.account_number = account_number
        self.balance = balance

    # Add money to the account.
    def deposit(self, amount):
        self.balance = self.balance + amount
        print("You made a deposit of the amount:", amount)
        print("Your new balance is:", self.balance)

    # Cash out.
    def withdraw(self, amount):
        self.balance = self.balance - amount
        print("You made a withdrawl of the amount:", amount)
        print("Your new balance is:", self.balance)
```

[46]:
```python
# Example
account = BankAccount("12345", 1000)
account.deposit(500)
account.withdraw(300)
```

```
You made a deposit of the amount: 500
Your new balance is: 1500
You made a withdrawl of the amount: 300
Your new balance is: 1200
```

3.0.4 Question 4: (20 Points)

3.1 Banking System Simulation

Objective: Create a simple simulation of a banking system using Object-Oriented Programming (OOP) concepts. This task will help you understand how to define classes, create instances, and implement class methods to perform operations.

3.1.1 Part 1: Define the BankAccount Class

Class Definition: Start by defining a class named `BankAccount`. **Attributes:** - `account_number`: A unique identifier for the bank account. - `name`: The name of the account holder. - `balance`: The current balance in the account (initialized with the balance provided when the account is created).

Constructor: - Define an `__init__` method that initializes the `account_number`, `name`, and `balance` attributes.

3.1.2 Part 2: Implement BankAccount Methods

Deposit: - Define a method named `deposit` that takes an amount as an argument and adds it to the account's balance. After adding, print the new balance with a message.

Withdraw: - Create a `withdraw` method that takes an amount to be withdrawn. If there are sufficient funds (balance is greater than or equal to the amount), subtract the amount from the balance and print the new balance. If not, print a message indicating insufficient funds.

Get Balance: - Implement a method `get_balance` that prints the current balance of the account.

Transfer: - Add a method named `transfer` that takes two arguments: the amount to be transferred and the `recipient_account` (another instance of `BankAccount`). First, check if the current account has enough balance to cover the transfer. If yes, subtract the amount from the sender's account and add it to the recipient's account balance. Print a success message including the new balance. If not, print a message indicating insufficient funds.

3.1.3 After Creating the classes

- Create at least two instances (Objects) of BankAccount with different initial balances.
- Demonstrate depositing, withdrawing, and transferring funds between these accounts.

[47]:
```
class BankAccount:
    # Basic account stuff.
    def __init__(self, account_number, name, balance):
        self.account_number = account_number
        self.name = name
        self.balance = balance
    # Put money in.
    def deposit(self, amount):
        new_total = self.balance + amount    # could do it in one line, just␣
 ↪keeping it clear
        self.balance = new_total
        print(f"{self.name} deposited ${amount}")
```

```
        print(f"New balance is: ${self.balance}")
    # Take money out (if they actually have it).
    def withdraw(self, amount):
        if self.balance >= amount:
            self.balance = self.balance - amount
            print(f"{self.name} withdrew ${amount}.")
            print("Balance now:", self.balance)
        else:
            print(f"{self.name} has no money.")
            print(f"Current balance: ${self.balance}")
    # Check what is left.
    def get_balance(self):
        current_balance = self.balance
        print(f"{self.name}'s balance right now: ${current_balance}")

    # Send money to another account
    def transfer(self, amount, recipient_account):
        if self.balance >= amount:
            self.balance = self.balance - amount

            # add to the other account
            recipient_account.balance = recipient_account.balance + amount

            print(f"Transferred ${amount} to {recipient_account.name}.")
            print(f"{self.name}'s new balance: ${self.balance}")
        else:
            print(f"Transfer failed: {self.name} is broke.")
            print("Still got:", self.balance)
```

[48]:
```
# Creating BankAccount instances and demonstrating functionality
account1 = BankAccount("1001", "Alice", 500)
account2 = BankAccount("1002", "Bob", 300)

print("Accounts created successfully!")
print("Account 1 holder:", account1.name)
print("Account 2 holder:", account2.name)


# Demonstrating account operations
account1.get_balance()
account2.get_balance()

account1.deposit(200)
account1.withdraw(100)

account1.transfer(250, account2)
```

```
# Hopefully the calculations are correct.
# I am willing to accept any left over money.
account1.get_balance()
account2.get_balance()
```

```
Accounts created successfully!
Account 1 holder: Alice
Account 2 holder: Bob
Alice's balance right now: $500
Bob's balance right now: $300
Alice deposited $200
New balance is: $700
Alice withdrew $100.
Balance now: 600
Transferred $250 to Bob.
Alice's new balance: $350
Alice's balance right now: $350
Bob's balance right now: $550
```

3.1.4 Question 5: (20 Points)

4 Create a Pickaboo Class

In this exercise, you will create a class called `Pickaboo`, representing a magical, playful creature. Your task is to design the class with attributes and methods to manage Pickaboo's behavior and characteristics. Follow the instructions below to complete the task.

4.0.1 Instructions:

1. **Define a class called `Pickaboo`** with the following attributes:
   - `name` (string): The name of the Pickaboo.
   - `mood` (string): The current mood of Pickaboo, which can be `"happy"`, `"curious"`, `"sneaky"`, or `"angry"`. It should start as `"happy"`.
   - `orb_colors` (list): A list of colors that represent the glowing orbs on Pickaboo's tail. Initially, this should contain three colors: `"blue"`, `"pink"`, and `"purple"`.
   - `energy_level` (integer): A number representing Pickaboo's energy, which should start at 100.
2. **Create the following methods for the `Pickaboo` class**:

- `change_mood(self, new_mood)`: This method changes Pickaboo's mood. If the new mood is `"angry"`, reduce the energy level by 20. If the new mood is `"happy"`, increase the energy level by 10. Use a conditional statement to handle these changes.
  - `play_hide_and_seek(self)`: This method simulates Pickaboo playing hide-and-seek. It should reduce the energy level by 15 and change the mood to `"sneaky"`. If the energy

level is too low (less than 30), print that Pickaboo is too tired to play.Use a conditional statement to handle these changes.

- `recharge(self)`: This method increases Pickaboo's energy level by 30 and changes the mood to `"happy"`. Make sure the energy level does not go above 100.
- `glow_orbs(self)`: This method should loop through the `orb_colors` list and print out the colors of the glowing orbs. If Pickaboo's mood is `"angry"`, temporarily add `"red"` to the list of orb colors while glowing. Use a conditional statement to handle these changes.

3. **Additional Task:**
   - Create an instance of the `Pickaboo` class with a name of your choice.
   - Use the methods to:
     1. Change its mood to `"curious"`.
     2. Play hide-and-seek with Pickaboo.
     3. Recharge its energy.
     4. Make its orbs glow.

Sample Output:

[49]:
```
Picky's mood is now: curious
Picky is playing hide-and-seek!
Picky's energy level is now: 85
Picky is recharging...
Picky's energy level is now: 100
Picky's orbs are glowing in these colors: ['blue', 'pink', 'purple']
```

```
  Cell In[49], line 1
    Picky's mood is now: curious
                               ^
SyntaxError: EOL while scanning string literal
```

[ ]:
```python
# Pickaboo thing
class Pickaboo:
    mood_choices = ["happy", "curious", "sneaky", "angry"]

    # Set it up
    def __init__(self, name):
        self.name = name
        self.mood = "happy"
        self.orb_colors = ["blue", "pink", "purple"]
        self.energy_level = 100

    # Change mood
    def change_mood(self, new_mood):
        if new_mood not in self.mood_choices:
            print("Invalid mood.")
```

```
            return
        self.mood = new_mood
        # mood tax / bonus
        if new_mood == "angry":
            self.energy_level = self.energy_level - 20
        elif new_mood == "happy":
            self.energy_level = self.energy_level + 10

        # Stop it from exploding
        if self.energy_level > 100:
            self.energy_level = 100
        if self.energy_level < 0:
            self.energy_level = 0
        print(f"{self.name}'s mood is now: {self.mood}")
        print(f"{self.name}'s energy level is: {self.energy_level}")

    # Sneaky mode
    def play_hide_and_seek(self):
        if self.energy_level < 30:
            print(f"{self.name} is too tired to play.")
            return
        print(f"{self.name} is playing hide-and-seek!")
        self.energy_level = self.energy_level - 15
        self.mood = "sneaky"
        print(f"{self.name}'s energy level is now: {self.energy_level}")

    # Juice up
    def recharge(self):
        print(f"{self.name} is recharging...")
        self.energy_level = self.energy_level + 30
        # No red bull
        if self.energy_level > 100:
            self.energy_level = 100
        self.mood = "happy"
        print(f"{self.name}'s energy level is now: {self.energy_level}")

    # pretty lights
    def glow_orbs(self):
        colors_to_glow = self.orb_colors.copy()

        # angry DLC color
        if self.mood == "angry":
            colors_to_glow.append("red")

        print(f"{self.name}'s orbs are glowing in these colors:␣
↪{colors_to_glow}")
```

[ ]:
```
# Create him
pickaboo1 = Pickaboo("Picky")

# Create an instance of the Pickaboo class
pickaboo1.change_mood("curious")
pickaboo1.play_hide_and_seek()
pickaboo1.recharge()
pickaboo1.glow_orbs()
```

```
Picky's mood is now: curious
Picky's energy level is: 100
Picky is playing hide-and-seek!
Picky's energy level is now: 85
Picky is recharging…
Picky's energy level is now: 100
Picky's orbs are glowing in these colors: ['blue', 'pink', 'purple']
```

[ ]: