CS-UY 1114: Lab 12

Object-Oriented Programming and Recursion

You must get checked out by your lab CA prior to leaving early. **If you leave without being checked out, you will receive 0 credits for the lab.**

Restrictions

The Python structures that you use in this lab should be restricted to those you have **learned in lecture so far (topics learned in Rephactor may only be used if they have also been taught in lecture)**. Please check with your teaching assistants in case you are unsure whether something is or is not allowed!

Create a new python file for each of the following problems.

Your files should be named *lab[num]_q[num].py* similar to homework naming conventions.

Problem 1: *Off to Bobst*

Given the following Library class and test code, write the values that would be output for each print statement.

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

    def __str__(self):
        return str(self.books)

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

    def borrow_book(self, book):
        ind = self.available(book)
        if ind >= 0:
            self.books.pop(ind)
            return True
        else:
            return False

    def available(self, book):
        for i in range(len(self.books)):
            if self.books[i] == book:
                return i
        return -1

def main():
    lib = Library()
    lib.add_book("A Game of Thrones")
    lib.add_book("1984")
    lib.add_book("Moby Dick")
    print(lib.available("1984"))
    lib.borrow_book("Moby Dick")
    print(lib)
```

```
    print(lib.borrow_book("Becoming"))

main()
```

Problem 2: *Mysterious Mystery Function*

Trace the following code and write the final output of a call to `main()`. Give the mystery function a new name that better fits its behavior.

```
def mystery(n):
    if n == 0:
        return 1
    else:
        return n * mystery(n - 1)

def main():
    print(mystery(4))
```

Problem 3: *More Recursion, More Fun!*

The following code has a function called `sum_double_recursive(n)`.

Trace and describe the behavior of the code for when a function call of `sum_double_recursive(5)` is made.

```
def sum_double_recursive(n):
    if n <= 0:
        return 0
    else:
        return n + sum_double_recursive(n - 1) + sum_double_recursive(n - 2)
```

Problem 4: *Rock, Paper, Scissors!*

This problem will be a simple model of instances of a student class playing rock, paper, scissors.

Let's start by setting up the `Student` class. The student class should have the following attributes:

```
selection
choices
name
wins
```

Note: selection is an empty string (upon initialization), choices is a list of the three options to choose from (rock, paper, or scissors). Name is taken from a parameter passed in the object's creation, and wins is an integer that starts at 0.

```
def __init__(self, name):
```

```
def __str__(self):
    """
        Returns a string representation of the student.
        """
```

The format of the student string should be:

```
<name> has won <wins> round(s) of rock paper scissors!
```

```
def pick_hand(self):
        """
        Selects a random hand to throw from choices
```

```
        """
```

```
def win_match(self):
        """
        Increments wins by one.
        """
```

Initialize a function `play_game` that will take two Student objects and make them play a game of rock, paper, scissors. Upon getting their selection, write conditions to properly increment the winning student's `win` attribute.

Try using the following to test your code. The program will stop when one student wins 2 rounds. Feel free to make your own tests!

```
def main():
   bob = Student("Bob")
   carl = Student("Carl")

   while bob.wins != 2 and carl.wins != 2:
       play_game(bob, carl)
   print(bob)
   print(carl)
```

A possible output is:

```
Bob has won 2 round(s) of rock paper scissors!
Carl has won 1 round(s) of rock paper scissors!
```

Problem 5: *I Want A Pet*

Lets make a program that lets you adopt and manage pets!

Write a class called `Pet`. It should have:

- A constructor that takes in a name, a type, and an age as parameters. A `Pet` should have 4 attributes: `name`, `type`, `age`, and `fav_treats`. `fav_treats` should be a list, and may be initialized to the empty list.
- A `rename` method, which takes in a new name as a parameter, and sets the pet's name to the new name.
- A `birthday` method, which just increments the pet's age by one.
- An `add_treat` method, which takes in a treat as a parameter and adds it to the pet's `fav_treats` list.
- A `__str__` that returns a string in this format:

```
<name> is a <type> that is <age> years old.
Favorite treats:
    <fav_treats[0]>
    <fav_treats[1]>
    ...
```

Next, write a `main()` function that creates a list of pets, then will repeatedly ask the user for a command and read user input until the user enters `Q` or `q`. Your program will do different things depending on what command the user entered. The valid commands are:

- `adopt`
    - Ask the user for the name of the pet, the type of pet (i.e., Cat, Dog, etc.), and the age of the pet.
    - Creates a new pet using these information and add it to the pets list
    - You may assume the user will always input an integer for the age of the pet
- `rename`
    - Ask the user for the old name of the pet and a new name

    - Change the name of the pet with the old name to the new name (must call the pet's `rename` method).
- `birthday`
    - Ask the user for the name of a pet
    - Increment the age of the pet (must call the pet's `birthday` method)
- `treat`
    - Ask the user for the name of a pet and the name of a treat.
    - Adds the treat to the pet's `fav_treats` list (must call the pet's `add_treat` method)
- `pets`
    - Displays every pet's information (must use `print` on each pet in the pets list)

A sample output of the entire program should look something like this:

```
Enter a command: adopt
What is the name of the pet? Momo
What type of pet is it? Dog
How old is the pet? 1

Enter a command: treat
Who is this treat for? Momo
What is the name of the treat? Biscuits

Enter a command: treat
Who is this treat for? Momo
What is the name of the treat? Bone

Enter a command: adopt
What is the name of the pet? Peluchito
What type of pet is it? Dog
How old is the pet? 13

Enter a command: rename
What is the current name of the pet? Peluchito
What is the new name of the pet? Peluche
```

```
Enter a command: birthday
Whose birthday is it? Peluche

Enter a command: pets
Momo is a Dog that is 1 years old.
Favorite treats:
        Biscuits
        Bone

Peluche is a Dog that is 14 years old.
Favorite treats:

Enter a command: q
```

Problem 6: *Strawhat Sailing*

This problem will model a pirate world where there exists `Pirate`s and `Crew`s that `Pirate`s can be a part of.

Part 1A: *Creating the* `Pirate` *Class*

The `Pirate` class will have the following attributes:

```
name (str)              # The Pirate's name
status (str)            # Can be "Captain", "Member", or "Solo"
collected_loot (int)    # The total loot a Pirate owns
```

The `Pirate` class should implement the following methods:

```
def __init__(self, name, loot = 0):
```

Note: Every `Pirate` ***is assumed to have a status of*** `"Solo"` ***at initialization***

```
def __str__(self):
    """
    Returns a string representation of a Pirate
    """
```

The string representation should be:

```
<status> <name>
```

```
def update_loot(self, loot_change):
    """
    Updates the value of the pirate's collected loot.
    loot_change may be positive or negative
    collected_loot cannot go below 0
    """
```

```
def update_status(self, status):
    """
    Updates the pirate's status to the given status
    """
```

```
def get_loot(self):
    """
    Returns the pirate's collected loot
    """
```

Part 1B: *Creating the* `Crew` *Class*

The `Crew` class will have the following attributes:

```
name (str)                  # Name of the Crew
pirates (list of Pirates)   # All pirate crew members
captain (Pirate)            # Pirate that's the captain of the
Crew (only 1 captain allowed per Crew)
total_loot (int)            # Total loot based on sum of each
pirate's collected loot
```

The `Crew` class should implement the following methods:

```
def __init__(self, name):
```

Note: `Crew`'s are assumed to have no pirate members nor a captain at initialization. This also means the total loot will be 0 at initialization.

```
def __str__(self):
    """
    Returns a string representation of a Pirate Crew
    """
```

The string representation should be:

```
<name> under <captain> with the following crew members:
<Pirate in pirates OR "No crew members yet." if no pirates>
With a total loot of: <total_loot>
```

You may look at the example output for further examples on `Crew`'s string representation.

```
def add_crew_member(self, pirate, role):
    """
    Adds the given pirate to this crew's member list according to the
given role ("Captain" or "Member").
    Updates the crew's total loot with the new pirate's loot and
appropriate pirate attributes.
    Returns False if adding a second captain, True if the operation
is successful.
    """
```

```
def remove_crew_member(self, pirate):
    """
    Removes the given pirate from this crew's member list.
    Updates the crew's total loot and appropriate pirate attributes.
    Returns False if removing a pirate not in the crew, True if the
operation is successful
    """
```

Note: With each method, think about what attributes need to be updated in either `Crew` *OR* `Pirate` *to keep the information consistent.*

You may use the following `main()` definition to help test your code:

```
def main():
    # Creating our pirates
    pirate_luffy = Pirate("Monkey D. Luffy")
    pirate_zoro = Pirate("Roronoa Zoro", 5)
    pirate_sanji = Pirate("Vinsmoke Sanji", 100)
    pirate_whitebeard = Pirate("Edward Newgate")
    pirate_marco = Pirate("Marco The Phoenix")
    pirate_ace = Pirate("Portgas D. Ace", 10)

    # Updating the loot of some pirates
```

```
    pirate_whitebeard.update_loot(40)
    pirate_sanji.update_loot(-20)
    pirate_marco.update_loot(-10)

    # Printing the current status of all our pirates
    all_pirates = [pirate_luffy, pirate_zoro, pirate_sanji,
pirate_whitebeard, pirate_marco, pirate_ace]
    for pirate in all_pirates:
        print(pirate, pirate.get_loot())
    print()

    # Creating our 2 crews
    crew1 = all_pirates[:3]
    crew2 = all_pirates[3:]

    strawhat_crew = Crew("Strawhat Crew")
    whitebeard_crew = Crew("Whitebeard Crew")

    # Print the empty crews
    print("--------------------------Printing empty
crews--------------------------")
    print(strawhat_crew)
    print()
    print(whitebeard_crew)
    print()

    # Add members to each crew
    for pirate in crew1:
        is_pirate_added = strawhat_crew.add_crew_member(pirate, "Captain")

        if not is_pirate_added:
            # Unable to add pirate due to attempting to add a 2nd captain
            # Intead, add the pirate as a member
            strawhat_crew.add_crew_member(pirate, "Member")

    for pirate in crew2:
        is_pirate_added = whitebeard_crew.add_crew_member(pirate, "Captain")

        if not is_pirate_added:
            # Unable to add pirate due to attempting to add a 2nd captain
            # Instead, add the pirate as a member
```

```
            whitebeard_crew.add_crew_member(pirate, "Member")

    # Print the newly made crews
    print("--------------------------Printing the newly made
crews--------------------------")
    print(strawhat_crew)
    print()
    print(whitebeard_crew)
    print()

    # Adding and removing a Pirate to a crew
    pirate_blackbeard = Pirate("Blackbeard", 50)

    # Removing a pirate not part of the crew
    whitebeard_crew.remove_crew_member(pirate_blackbeard)

    # Adding a Pirate to the crew then removing the pirate
    whitebeard_crew.remove_crew_member(pirate_whitebeard)
    whitebeard_crew.add_crew_member(pirate_blackbeard, "Captain")
    print("--------------------------Printing the new crew after captain
changes--------------------------")
    print(whitebeard_crew)
    print()
    print(pirate_whitebeard)
```

Whose expected output should be:

```
Solo Monkey D. Luffy 0
Solo Roronoa Zoro 5
Solo Vinsmoke Sanji 80
Solo Edward Newgate 40
Solo Marco The Phoenix 0
Solo Portgas D. Ace 10

--------------------------Printing empty
crews--------------------------
Strawhat Crew under None with the following crew members:
No crew members yet.
```

```
With a total loot of: 0

Whitebeard Crew under None with the following crew members:
No crew members yet.
With a total loot of: 0

--------------------------Printing the newly made
crews--------------------------
Strawhat Crew under Captain Monkey D. Luffy with the following
crew members:
Captain Monkey D. Luffy
Member Roronoa Zoro
Member Vinsmoke Sanji
With a total loot of: 85

Whitebeard Crew under Captain Edward Newgate with the following
crew members:
Captain Edward Newgate
Member Marco The Phoenix
Member Portgas D. Ace
With a total loot of: 50

--------------------------Printing the new crew after captain
changes--------------------------
Whitebeard Crew under Captain Blackbeard with the following
crew members:
Member Marco The Phoenix
Member Portgas D. Ace
Captain Blackbeard
With a total loot of: 60

Solo Edward Newgate
```