CS2316 SPRING 2021 HOMEWORK 01: PYTHON REVIEW AND OOP

Revisions

Date: currently none

Purpose

Using object-oriented programming (OOP) when writing code reduces development time because you can reuse code and develop models based on previous objects. Objects also separate themselves to prevent accidental overwriting or influence from other programs. A good understanding of OOP basics, design principles, and design patterns will help programmers develop well-written, reusable code.

Skills

This assignment will give you practice with object oriented programming concepts. You will learn to write completely new classes and also create instances of those classes. You will learn how objects interact with each other, and improve your skills at determining which methods work with which types of objects.

Knowledge

You will gain knowledge in how to develop a new class and how to use objects that are of that class. Using methods from other classes will help you understand the underlying structure of a class. Understanding class structure will be helpful not only in developing new classes but also in using existing classes more efficiently.

Important

1. Due Date: **Sunday, 02/06/2022 at 11:59 pm**
2. This homework is graded out of **100** points. However, **the autograder will only show up to 75 points when you submit your code.** We are saving some test cases for the final autograder that we will use. This is to encourage you to start developing ways to test features of your code when you aren't given test cases in an autograder. We also reserve the right to change the test cases later if we need to. Therefore, the grade reflected in Gradescope does not reflect your final grade on HW01.py.
3. This is an individual assignment. You may collaborate with other students in this class. Collaboration means talking through problems, assisting with debugging, explaining a concept, etc. Students may only collaborate with fellow students currently taking CS 2316, the TAs, and the instructor. You should not exchange code or write code for others. For individual assignments, each student must turn in a unique program. Your submission must not be substantially similar to another student's submission. Collaboration at a reasonable level will not result in substantially similar code.
4. For Help:
   - TA Helpdesk / Ed Discussion Board
   - Python Fundamentals Handout, OOP Handout

    - http://openbookproject.net/thinkcs/python/english3e/

5. Comment out or delete all your function calls. Only global variables, and comments are okay to be outside the scope of a function. When your code is run, all it should do is run without any errors.
6. Do not wait until the last minute to do this assignment in case you run into problems.
7. **Read the entire specifications document before starting this assignment.**
8. **IF YOUR CODE CANNOT RUN BECAUSE OF AN ERROR, IT IS A 0%**

Introduction

The goal of this homework is to showcase your knowledge of Python techniques, including comprehensions and Object-Oriented Programming (OOP). You should test your classes and methods out first on your own computer. Then when you have one or more of them working, upload the entire file (which must be named **HW01.py**) to GradeScope to see how well your code performs on the test cases we have created. **You can submit the homework file as many times as you'd like before the deadline.**

Make sure you write your code in the template we provided to you (HW01.py) and follow the formatting requirements for one line functions.

One-line Formatting

Correct:

```
def function_a(param1):
     return [i for i in param1]
```

Correct:

```
def function_a(param1):
     return type(param1)
```

Incorrect:

```
def function_a(param1):
     val = [i for i in param1]
     return val
```

Incorrect:

```
def function_a(param1):
     return helper(param1)

def helper(param1):
     return [i for i in param1]
```

Testing your code:

We will walk through the process of testing your functions manually. We will use the add_digits function (the first homework question in this assignment) as an example.

1. Go to the add_digits function at the top of the HW01.py file. Delete the `'pass'` text, and replace it with your code for the function.
2. Make sure your function ends in a return statement.
3. When you are ready to test your function, scroll to the middle of the HW01.py file where it says `if __name__ == '__main__':`
4. Find the comment that applies to the function you have just written. In this case, we look for the `#Part 1: test add_digits` comment.
5. Uncomment any print statements in the code chunk that directly follows that comment. In this case, uncomment `print(add_digits(int_list))`. This will call the add_digits function and use int_list as the input parameter.
6. Save the file.
7. Go to terminal or command line.
8. Navigate to your CS2316 Homework directory using cd.
9. Run the homework file by typing '`python HW01.py`'
10. Check if your terminal output matches the example output for that function in the HW01 Prompt. In this case, if your terminal output for add_digits is `14381` then your add_digits function is correct!!! If not... happy debugging!

Functions and Classes

A grading rubric including point distribution is provided at the end of this document.

Part 1: Functions

Function name: `add_digits`
Parameter(s): `int_list` (list)
Return Type: int
Description: Write a function that passes in `int_list`, a list of whole numbers, and returns an integer. This integer should be created by adding each entry from `int_list` to the others as a new digit.
Hint: Try adding each number together as a string and then casting your result into an integer.

Python Test Cases:

```
>>> int_list = [1, 4, 3, 8, 1]
>>> add_digits(int_list)
14381

>>> int_list = [6, 1, 1, 1]
>>> add_digits(int_list)
6111
```

Function name: **`string_modifier`**
Parameter(s): `str_list` (list)
Return Type: list
Description: Write a function that passes in `str_list`, a list of strings containing any alphanumeric characters, and returns a list of the strings multiplied by their length. Only include the strings if they contain only **lowercase letters**.
Hint: `.islower()` and `.isalpha()` may be helpful.

Python Test Cases:

```
>>> str_list = ['ace', 'BLuE42', 'b4a', 'cs']
>>> string_modifier(str_list)
['aceaceace', 'cscs’]
>>> str_list =['Gabbagoo', 'gabbaGoo', 'gabbagOo', 'gabbagoO',
 'gabbagoo']
>>> string_modifier(str_list)
['gabbagoogabbagoogabbagoogabbagoogabbagoogabbagoogabbagoogabba
goo’]
```

Function name: **`contacts`**
Parameter(s): `phone_numbers`(list)
Return Type: int
Description: Write a function that parses through `phone_numbers` and returns the number of **unique** phone numbers in your contacts. You will need to remove all characters that are not digits and should only count phone numbers that start with 404. Valid phone numbers should contain exactly 10 digits.
Hint: `.isdigit()` may be helpful
Python Test Case:

```
>>> phone_numbers = ['4o04592,,,0000', '-%#3&0&3!0',
'4!0!4!5!9!2!0!0!0!0!','908', '4###04...-,,,,,78', '404$$$193--
8173', '3^^0--,30']
>>> contacts(phone_numbers)
2
>>> phone_numbers = ['404567891', ')))404(((', '$379XD',
'4o33,0', '714---615---0294']
>>> contacts(phone_numbers)
0
```

Function name: `shelf_books`
Parameter(s): `book_list` (list)
Return Type: list
Description: Georgia Tech finally decided to buy some books for the library, and they've asked you to create a shelfing system! Write a function that parses through `book_list` and sorts the books based off the author's last name. If any books are written by the same author, they should be sorted by their title as well. **Return a list of only the sorted book titles.**

There is a one-line maximum requirement for this function.

Python Test Case:

```
>>> shelf_books([('It', 'Stephen King'), ('Gone Girl', 'Gillian
Flynn'), ('Verity', 'Colleen Hoover')])
['Gone Girl','Verity', 'It']
>>> shelf_books([('The Lightning Thief', 'Rick Riordan'),
('Divergent', 'Veronica Roth'),('The Hunger Games', 'Suzanne
Colins'), ('Catching Fire', 'Suzanne Colins')])
['Catching Fire', 'The Hunger Games','The Lightning
Thief','Divergent']
```

Function name: `classics`
Parameter(s): `books` (list), `chapters` (list), `authors` (list)
Return Type: dict

Description: Given a list of book titles, a corresponding list of the total number of chapters in each book, and a list of their respective authors, come up with a function that creates a dictionary in which each book title is associated to whether the read is entertaining or not (**True** or **False**). A book is entertaining if and only if it has at **least 20 chapters** AND its **author is NOT James Joyce (J. Joyce)**.
Hint: Consider using `zip()` and keep in mind that it can take more than two lists as inputs.

There is a one-line maximum requirement for this function.

Python Test Case:

```
>>> classics([‘Around the Moon’, 'Ulysses', '1984', 'Don
Quixote'], [25, 45, 12, 90], ['J. Verne', 'J. Joyce', 'G.
Orwell', 'M. Cervantes'])
{'Around the Moon': True, 'Ulysses': False, '1984': False, 'Don
Quixote': True}
>>> classics([‘Frankenstein’, 'Araby', 'Doctor Sleep'], [30, 1,
20], ['M. Shelley', 'J. Joyce', 'S. King'])
{'Frankenstein': True, 'Araby': False, 'Doctor Sleep': True}
```

Function name: **writers**
Parameter(s): `authors` (list), `age_at_death` (list)
Return Type: dict
Description: Given a list of authors, along with their respective ages at the time of death, create a dictionary in which you rank them based on how long they lived. That is, create a dictionary whose keys are the natural numbers (starting at 1) and whose corresponding values are names of authors such that whoever lived the longest is ranked 1, whoever lived the second longest is ranked 2, and so on.

Hint: You might need to use `enumerate()` and `zip()`
Hint: If you use `enumerate()` , you will have to change the parameter `start` from its default value of zero.

There is a one-line maximum requirement for this function.

Python Test Case:

```
>>> writers([‘J. Swift’, ‘T. Paine’, ‘C. Dickens’, ‘H.
Melville’], [78, 73, 58, 72])
{1:‘J. Swift’, 2:‘T. Paine’, 3:‘H. Melville’, 4:‘C. Dickens’}
```

Part 2: Classes

For the Classes portion of this homework, you will be writing a Book and a Library class for a book collection database. You should write the Book class first, as the Library class will contain Book objects.

Class name: Book
Instance Attributes:

`name`	(str)	Name of the Book
`genre`	(str)	Genre of the Book
`year`	(int)	Year of publication of the Book
`pages`	(int)	Total pages of the Book

Description: Write an Book class with the above attributes. Write the appropriate methods to accomplish the following tasks:

Method 1:
Parameter(s): `name` (str), `genre` (str), `year` (int), `pages` (int)
Description: Write a method to initialize an instance of the Book class with the parameters above in order.

Method 2:
Description: Define a method that will make the Book sortable based on `year`. One Book is considered **less than** another if it's published earlier (i.e. 1885 < 2021).

Method 3:
Description: Define a method that will make the Book comparable based on `name`, `year`, and `pages`. One Book is considered the same as another if they have the same `name`, `year`, and `pages`.

Method 4:
Description: Define a method so that the following string is returned whenever a Book object is called in the python shell or printed: `'{name} is a {pages} pages long {genre} book published in {year}.'`

Python Test Case:

```
>>> Book1 = Book('Normal People', 'Fiction', 2018, 273)
>>> Book2 = Book('The Catcher in the Rye', 'Fiction', 1951, 234)
>>> Book3 = Book('Normal People', 'Some unknown genre', 2018, 273)
>>> Book1.name
'Normal People'
```

```
>>> Book1.genre
'Fiction'
>>> Book1 < Book2
False
>>> Book1 == Book2
False
>>> Book1 == Book3
True
>>> Book1
Normal People is a 273 pages long Fiction book published in 2018.
```

Class name: `Library`
Instance Attributes:

`name`	(str)	Name of the `Library`
`book_list`	(list of Book objects)	Collection of `Book` objects that make up the `Library`
`num_fiction`	(str)	The number of `Books` at the `Library` that are the genre `'Fiction'`

Description: Write a `Library` class with the above attributes. Write the appropriate methods to accomplish the following tasks:

Method 1:
Parameter(s): `name` (str), `book_list` (list)
Description: Write a method to initialize an instance of the `Library` class with the parameters above in order. **Be sure to initialize values for all 3 of the instance attributes described in the `Library` class description.** Make sure to create `Book` objects for the instance attributes given in the `book_list` parameter (i.e. the `book_list` *parameter* contains the *attribute parameters* needed to instantiate `Book` objects in the `book_list` *attribute*.)
Hint: Note that there is no parameter passed in for the `num_fiction` attribute. You will have to calculate this based on how many `Books` at the `Library` are fiction.

Method 2:
Description: Define a method so that the following string is returned whenever a `Library` object is called in the python shell or printed: `'{name} is a library where {num_fiction} of the {length of book_list} books are fiction.'`

Method 3:
Description: Define a method that will make `Library` sortable based on the total number of fiction books it contains. A `Library` is considered **less than** another if it has fewer total fiction books.

Method 4:
Method name: `add_book`
Parameter(s): `new_book` (list)
Return Type: NoneType
Description: Write a method to instantiate a new `Book` object and then add it to the `book_list` attribute.

Hint: Make sure to update **any** instance attributes that may change with the addition of a new book.

Python Test Cases:

```
>>> book_list = [['Normal People', 'Fiction', 2018, 273], ['The
Catcher in the Rye', 'Fiction', 1951, 234], ['The Goal',
'Fiction', 1984, 384], ['How to Think Like a Computer
Scientist', 'Textbook', 2002, 274]]

>>> Bohongs_Library = Library('Bohong’s Books', book_list)
>>> Bohongs_Library.name
'Bohong’s Books’
>>> Bohongs_Library
[Normal People is a 273 pages long Fiction book published in
2018., The Catcher in the Rye is a 234 pages long Fiction book
published in 1951., The Goal is a 384 pages long Fiction book
published in 1984., How to Think Like a Computer Scientist is a
274 pages long Textbook book published in 2002.]

>>> Erins_Library = Library('Erin’s Excellent Essays',
[['Arriving Today: From Factory to Front Door', 'Nonfiction',
2021, 384]])

>>> Bohongs_Library < Erins_Library
False

>>> book1 = ['The Hunger Games', 'Fiction', 2008, 374]
>>> Bohongs_Library.add_book(book1)
>>> Bohongs_Library
'Bohong’s Books is a library where 4 of the 5 books are
fiction.'
```

Gradescope Requirements

- **NO PRINT STATEMENTS.** This will break the autograder
- **NO FUNCTION CALLS** outside of function definitions. This will also cause the autograder to not be able to run your code.
- Make sure the file submitted is titled **HW01.py**
- **Do not import any modules, packages, or libraries**
- **All points of this grade will definitely be lost** if it is shown that you have collaborated in an unauthorized manner on this assignment or otherwise violated the Georgia Tech honor code.

Grading Rubric

`add_digits`	5	pts
`string_modifier`	5	pts
`contacts`	10	pts
`shelf_books`	10	pts
`classics`	10	pts
`writers`	10	pts
`Book class:`		
`Method #1`	2.5	pts
`Method #2`	2.5	pts
`Method #3`	2.5	pts
`Method #4`	2.5	pts
`Library class:`		
`Method #1`	10	pts
`Method #2`	10	pts
`Method #3`	10	pts
`Method #4`	10	pts
`Total`	100/100	pts

Note that the autograder on GradeScope does not include test cases for `string_modifier`, `shelf_books`, and `writers`. Therefore, you are able to get up to 75 points on GradeScope before the assignment is due. We are saving those test cases for later use. You are encouraged to manually test those functions.