Lab 16, Part 1: Object-Oriented Programming

Instructions:

This worksheet serves as a guide and set of instructions to complete the lab.

- You must use the starter file, found here, to get credit for the lab.
- Additionally, here is the workbook that you can read through for further context and additional (non-required) material.
- All material was sourced from the CS10 version of The Beauty and Joy of Computing course.

Submitting:

For part 1 of Lab 16, you will need to complete **all *Book* class methods and 3 *TicTacToe* methods (9 total methods)** then submit this to Gradescope (Object Oriented Programming, Part 1).

- To receive full credit, you will need to complete all required methods, and the required methods must pass all tests from the autograder in Gradescope.
- For instructions on how to submit to labs to Gradescope, please see this document.

Please note, you must use the starter file, and you must NOT edit the name of any of the required functions. Failing to do either for these will result in the autograder failing.

Notes:

***This is Part 1 of Lab 16, meant for Thursday (7/18).* Part 2 is due on Monday (7/22) @ 2359hrs**

Objectives:

Object-Oriented Programming (OOP) is a programming paradigm based on the concept of objects. Objects which can contain data (we call these attributes) and code (we call these methods). OOP is a very important level of abstraction used commonly across programming languages In today's lab you will:

- Learn how to create Class methods
- Learn how to build a constructor for a Class
- Understand the difference between Class and instance attributes
    - Along with knowing when to properly use each
- Understand the differences between instances of the same Class
- Understand and create **str** and **repr** methods

Glossary

- **Object:** An object is a collection of data with its own methods, attributes, and identity.
- **Method:** A function that an object has. For example, the Book object has the method calculate_age that calculates the age of the book.
- **Attribute:** Basically, a variable that belongs to an object. For example, a Book object's genre is an attribute.
- **Class:** A class is the blueprint of an object; it is the precise definition of an object's methods and attributes.
- **Constructor:** The constructor is the method that Python uses when it creates (instantiates) an object. Not all attributes of a class are defined immediately, the constructor lets you define an object's attributes when you actually create the object.
- **Instance:** An instance is an object made from a specific class. For example, The Hunger Games could be an instance of the Book class.
- **Instantiation:** The creation of an instance. This is when the constructor actually creates the object.

Required Methods:

- Book Class
    - __init__(self, genre, title, author, publication_year)
    - __str__(self)
    - __rpr__(self)
    - calculate_age(self)
    - outdated(self, old_age)
    - add_to_genres(self)
- TicTacToe Class
    - do_move(self, x, y)
    - is_valid_move(self, x, y)
    - check_game_over(self)

Important Topics mentioned in the Workbook:
For better understanding of the lab we highly recommend going through these workbook pages! Topics that are important but not required for this lab will be indicated with an asterisk**. These topics are best reviewed in order and as you complete the lab.

- Constructors
- Class Attributes

Method 1.1: __init__(self, genre, title, author, publication_year)

- Objective:
    - Create a constructor that initializes a Book object with 4 attributes: genre, title, author, and publication year
- Notes:

    - Review Constructors page of workbook for guidance
- Inputs:
    - self = N/A
    - genre = string, the genre of the book
    - title = string, the title of the book
    - author = string, the author of the book
    - publication_year = integer, the publication year of the book
- Output:
    - Reports: None
    - Your output should be nothing. The constructor simply initializes an object without reporting anything
- Examples:
    - Doctests available
        - python3 -m doctest <labfilename>.py to run autograder
            - Must be in correct parent file

Method 1.2: __str__(self)

- Objective:
    - Create a str method that returns a custom string representation of a Book object
- Notes:
    - This is what's called when we use the print() function
- Inputs:
    - self = N/A
- Output:
    - Reports: String
    - Your output should be a string representing the Book object your calling on
- Examples:
    - Doctests available
        - python3 -m doctest <labfilename>.py to run autograder
            - Must be in correct parent file

Method 1.3: __rpr__(self)

- Objective:
    - Create a repr method that returns a custom string representation that can be used to recreate the object.
- Notes:
    - This is what is called when we call an object by itself
- Inputs:
    - self = N/A
- Output:

    - Reports: String
    - Your output should report a custom string representation
- Examples:
    - Doctests available
        - python3 -m doctest <labfilename>.py to run autograder
            - Must be in correct parent file

Method 1.4: calculate_age(self)

- Objective:
    - Write a method that calculates the age of a Book object
- Notes:
    - Every Book has a unique age attribute, how can you access and use that?
- Inputs:
    - self = N/A
- Output:
    - Reports: Integer
    - Your output should be the age of the Book you're evaluating
- Examples:
    - Doctests available
        - python3 -m doctest <labfilename>.py to run autograder
            - Must be in correct parent file

Method 1.5: outdated(self, old_age)

- Objective:
    - Write a method that determines whether a Book object is outdated or not based on its age
- Notes:
    - Every Book has a unique age attribute, how can you access and use that?
- Inputs:
    - self = N/A
    - old_age = Integer, the maximum age threshold for considering the book outdated
- Output:
    - Reports: Boolean
    - Your output should be True or False, depending on if the Book object is outdated or not
- Examples:
    - Doctests available
        - python3 -m doctest <labfilename>.py to run autograder
            - Must be in correct parent file

Method 1.6: add_to_genres(self)

- Objective:
    - Write a method that adds a Book object to a genres dictionary
- Notes:
    - If the book's genre isn't in the genres dictionary, add the genre and the book's information. Otherwise, make sure the book isn't already present in the genres dictionary before adding it.
    - The genres dictionary should be an attribute accessible to any Book object
    - .genre and .genres are *different attributes*
- Inputs:
    - self = N/A
- Output:
    - Reports: None
    - Your output should modify the genre dictionary and not report anything
- Examples:
    - Doctests available
        - python3 -m doctest <labfilename>.py to run autograder
            - Must be in correct parent file

Method 2.1: do_move(self, x, y)

- Objective:
    - Write a method that attempts to make a move on the board at the specified coordinates (x, y) for the current player.
- Notes:
    - You may use other TicTacToe Class methods inside of this method
    - Don’t make a move for the player before you know if it’s valid or not
- Inputs:
    - self = N/A
    - x = The row index (0-2) where the player wants to make a move
    - y = The column index (0-2) where the player wants to make a move
- Output:
    - Reports: Boolean
    - Your output should return True if the move is successfully made; False otherwise
- Examples:
    - Doctests available
        - python3 -m doctest <labfilename>.py to run autograder
            - Must be in correct parent file

Method 2.2: is_valid_move(self, x, y)

- Objective:

    - Write a method that checks if a move at the specified coordinates (x, y) is valid
- Notes:
    - There are multiple conditions you'll have to check
        - How do you enforce multiple conditions?
- Inputs:
    - self = N/A
    - x = The row index (0-2) where the player wants to check validity
    - y = The column index (0-2) where the player wants to check validity
- Output:
    - Reports: Boolean
    - Your output should return True if the move is valid to make; False otherwise
- Examples:
    - Doctests available
        - python3 -m doctest <labfilename>.py to run autograder
            - Must be in correct parent file

Method 2.3: check_game_over(self)

- Objective:
    - Write a method that determines the status of the game
- Notes:
    - You only need to fill in the ellipses (...) for this method, the rest is completed for you
- Inputs:
    - self = N/A
- Output:
    - Reports: Tuple
    - Your output should be a tuple containing a Boolean and a string, where the boolean indicates if the game is over and the str indicates the result
- Examples:
    - Doctests available
        - python3 -m doctest <labfilename>.py to run autograder
            - Must be in correct parent file

You can always check the validity of your solutions by using the local autograder. Remember to submit on Gradescope!