PSG COLLEGE OF TECHNOLOGY, COIMBATORE – 641004

DEPARTMENT OF INFORMATION TECHNOLOGY

Ex- 6 CLASSES AND OBJECTS

1. Write a class called `Investment` with fields called `principal` and `interest`. The constructor should set the values of those fields. There should be a method called `value_after` that returns the value of the investment after *n* years. The formula for this is $p(1+i)^n$, where *p* is the principal, and *i* is the interest rate. It should also use the special method `__str__` so that printing the object will result in something like below:

```
Principal - $1000.00, Interest rate - 5.12%
```

2. Write a class called `Product`. The class should have fields called `name`, `amount`, and `price`, holding the product's name, the number of items of that product in stock, and the regular price of the product. There should be a method `get_price` that receives the number of items to be bought and returns a the cost of buying that many items, where the regular price is charged for orders of less than 10 items, a 10% discount is applied for orders of between 10 and 99 items, and a 20% discount is applied for orders of 100 or more items. There should also be a method called `make_purchase` that receives the number of items to be bought and decreases `amount` by that much.

3. Write a class called `Password_manager`. The class should have a list called `old_passwords` that holds all of the user's past passwords. The last item of the list is the user's current password. There should be a method called `get_password` that returns the current password and a method called `set_password` that sets the user's password. The `set_password` method should only change the password if the attempted password is different from all the user's past passwords. Finally, create a method called `is_correct` that receives a string and returns a boolean `True` or `False` depending on whether the string is equal to the current password or not.

4. Write a class called `Time` whose only field is a time in seconds. It should have a method called `convert_to_minutes` that returns a string of minutes and seconds formatted as in the following example: if seconds is 230, the method should return `'5:50'`. It should also have a method called `convert_to_hours` that returns a string of hours, minutes, and seconds formatted analogously to the previous method.

5. Write a class called `Wordplay`. It should have a field that holds a list of words. The user of the class should pass the list of words they want to use to the class. There should be the following methods:
    - `words_with_length(length)` — returns a list of all the words of length `length`
    - `starts_with(s)` — returns a list of all the words that start with `s`

- `ends_with(s)` — returns a list of all the words that end with `s`
- `palindromes()` — returns a list of all the palindromes in the list
- `only(L)` — returns a list of the words that contain only those letters in `L`
- `avoids(L)` — returns a list of the words that contain none of the letters in `L`

6. Write a class called `Converter`. The user will pass a length and a unit when declaring an object from the class—for example, `c = Converter(9, 'inches')`. The possible units are inches, feet, yards, miles, kilometers, meters, centimeters, and millimeters. For each of these units there should be a method that returns the length converted into those units. For example, using the `Converter` object created above, the user could call `c.feet()` and should get `0.75` as the result.

7. Use the `Standard_deck` class of this section to create a simplified version of the game *War*. In this game, there are two players. Each starts with half of a deck. The players each deal the top card from their decks and whoever has the higher card wins the other player's cards and adds them to the bottom of his deck. If there is a tie, the two cards are eliminated from play (this differs from the actual game, but is simpler to program). The game ends when one player runs out of cards.

8. Write a class that inherits from the `Card_group` class of this chapter. The class should represent a deck of cards that contains only hearts and spaces, with only the cards 2 through 10 in each suit. Add a method to the class called `next2` that returns the top two cards from the deck.

9. Write a class called `Rock_paper_scissors` that implements the logic of the game Rock-paper-scissors. For this game the user plays against the computer for a certain number of rounds. Your class should have fields for the how many rounds there will be, the current round number, and the number of wins each player has. There should be methods for getting the computer's choice, finding the winner of a round, and checking to see if someone has one the (entire) game. You may want more methods.

10.
    a. Write a class called `Connect4` that implements the logic of a Connect4 game. Use the `Tic_tac_toe` class from this chapter as a starting point.
    b. Use the `Connect4` class to create a simple text-based version of the game.

11. Write a class called `Poker_hand` that has a field that is a list of `Card` objects. There should be the following self-explanatory methods:

```
has_royal_flush, has_straight_flush, has_four_of_a_kind,
has_full_house, has_flush, has_straight,
has_three_of_a_kind, has_two_pair, has_pair
```

    There should also be a method called `best` that returns a string indicating what the best hand is that can be made from those cards.