COMP 1510 Object oriented programming 1
Assignment #2: More Dungeons and Dragons!

Christopher Thompson
cthompson98@bcit.ca

Due **Saturday October 19th at or before 15:59:59 (that's 3:59:59 pm)**

Welcome!

For your second assignment I'd like you to build upon some of the code you wrote for lab 5. Now that we know all the important control structures – sequence, selection, repetition, and indirection – we can start building interesting collections of functions that work together. Programs!

Developing software is often a highly iterative process. Iterative software development involves repeated cycles of designing, coding, testing, and integrating small sections or increments of a program. We build a little, test it, make adjustments, and continue.

By deconstructing a large project into smaller, more manageable tasks, we can abstract away portions of the application and focus on separate, solvable problems. The knowledge we gain from developing and testing these smaller modules can be applied to the development of other parts of the project. Ultimately we generate correct, working, and maintainable software efficiently and quickly.

1 Submission Requirements

1. **This take home assignment is due no later than Saturday October 19th at or before 15:59:59**.
2. Late assignments will not be accepted for any reason. This assignment is due in the middle of the afternoon so you can focus on your exams.
3. This is an individual assignment. I strongly encourage you to share ideas and concepts, but sharing code or submitting someone else's work is not allowed.

2 Project Setup

Please complete the following:

1. Open the PyCharm project you created for the first assignment. Recall we made folders called A1, A2, ..., A5. Recall we will use this project for ALL of our assignments this term.
2. For your second assignment, all of your code must go into the A2 folder. After you complete each function, commit and push your change to version control. In order to earn full marks, you must commit and push after each function is complete.
3. When you are finished, you do not need to add me as a collaborator, because you already invited me to collaborate for A1.

3 Style Requirements

1. **Your source code must all be placed inside a file called dungeonsanddragons.py**.
2. **You may not use global variables**. All variables must be local, i.e., they must be inside functions.
3. Functions must be short and must only do one thing. If a function does more than one thing, break it down into two or more functions that work together. Remember that helper functions can help more than one function, so we want to use generic names as much as possible.
4. **You must comment each function** you implement with correctly formatted docstrings.
5. **Include informative doctests where necessary** to demonstrate to the user how the function is meant to be used, and what the expected output will be when input is provided.
6. Ensure that the docstring for each function you write has the following components (in this order):
   (a) Short one-sentence description that begins with a verb in imperative tense
   (b) One blank line
   (c) Additional comments if the single line description is not sufficient (this is a good place to describe in detail how the function is meant to be used)
   (d) One blank line
   (e) PARAM statement for each parameter which describes what the user should pass to the function.
   (f) PRECONDITION statement for each precondition which the user promises to meet before using the function
   (g) POSTCONDITION statement for each postcondition which the function promises to meet if the precondition is met
   (h) RETURN statement which describes what will be returned from the function if the preconditions are met
   (i) One blank line
   (j) And finally, the doctests. Here is an example:

```
def my_factorial(number):
    """Calculate factorial.

    A simple function that demonstrates good comment construction.

    :param number: a positive integer
    :precondition: number must be a positive integer
    :postcondition: calculates the correct factorial
    :return: factorial of number

    >>> my_factorial(0)
    1
    >>> my_factorial(1)
    1
    >>> my_factorial(5)
    120
    """
    return math.factorial(number)
```

7. **Each function must also be tested in a separate unit test file** by a separate collection of unit tests. Ensure your test cases and functions have descriptive, meaningful names, and ensure that every possible valid input is tested for each function. Remember we want to create disjointed equivalency partitions out of all possible output, and test a representative from each partition.

4 Functions

Please implement the following functions. For some functions, you may wish to (and perhaps even should) create helper functions:

1. Start by copying the following functions (and their helper functions) from the Lab05 package in your labs project to dungeonsanddragons.py:
   (a) roll_die
   (b) create_name
   (c) choose_inventory
   (d) create_character
   (e) print_character.
2. **Let's modify the create_character function**:
   (a) Instead of creating a list, modify the create_character function so it creates a dictionary to store the character. The contents should be modified as follows:
      i. The first key:value pair should be 'Name' (key) and the character's name (value).
      ii. The next six items should be key:value pairs for the six attributes (the keys are strings, and the values are ints).
      iii. Then inventory should be a key:value pair too. The key should be the word 'Inventory', and the value should be a list of inventory items. A brand new player who has not acquired any items must have an empty list for an inventory.
   (b) Every character has experience points. Players gain experience points by solving problems, killing monsters, etc. Let's add experience points to the dictionary that represents a character. Let's insert it right after the six attributes, before the inventory. A new character starts with 0 experience points, or 'XP' as we will call them.
   (c) When we create a new character, we should select a class. Create a function called select_class which prints out a list of classes to the user, and asks the user to select the class they want to play. Classes to choose from include fighter, wizard, cleric, rogue, ranger, etc. A full list of the twelve (12) main classes in D&D can be found at `https://www.dndbeyond.com/characters/classes`. You can invoke this function from inside generate_character, and use the player's response to populate the character.
   (d) Let's decide to insert the character's class right after the character's name in the character dictionary. Store it as a key:value pair. The key is the string 'Class' and the value is the class name (all lower case please).
   (e) We can select our race in D&D too. There are nine (9) basic races available to players, including such favourites as elf, halfling, and tiefling. Create a function called select_race which prints out a list of races to the user, and asks the user to select the race they want to play. A full list of the races in D&D can be found at `https://www.dndbeyond.com/characters/races`. You can invoke this function from inside generate_character, and use the player's response to populate the character.
   (f) Let's decide to insert the character's race right after the character's name in the character dictionary, before the class. Store it as a key:value pair. The key is the string 'Race' and the value is the race name (all lower case please).
   (g) Every character needs some health, or as we call it in D&D, hit points. The number of hit points depends on the character's class.
   (h) Each new character gets to roll a die once to determine their initial HP. A barbarian can roll a d12 once, for example. You can find the initial die to roll HP on the same page: `https://www.dndbeyond.com/characters/classes`.
   (i) Ensure HP is placed after the class in the character dictionary object. It makes sense that the name, class, and remaining health are near the top. When you store the HP, you need to store two values: the maximum HP, and the current HP. Store these as a key:value pair where the key is 'HP' and the value is a list of length two, where the first element is the maximum HP, and the second element is the current HP.

3. What about outfitting our character with weaponry and supplies? **Modify the choose_inventory function**:
   (a) The choose_inventory function must print a list of goods to the screen and ask the player what they want to buy. The list of goods should be printed like this:

```
Welcome to the Olde Tyme Merchant!

Here is what we have for sale:

1. sword
2. dagger
3. chalice of becoming
4. orb of discovery
5. and so on and so on...

What would you like to buy (-1 to finish):
```

   (b) If the user enters a number that corresponds to a number on the list, add that item to a temporary local list and then print the menu again and wait for the user to respond
   (c) If the user enters a -1, exit the function by returning the temporary local list of purchases
   (d) If the user enters anything else, print a helpful message that tells the user what they can enter, and print the menu again.
4. What about combat? **Let's create a simple function called combat_round** that represents a single round of combat.
   (a) In order for combat to take place, we need two combatants. The combat function accepts two character dictionary objects as parameters, opponent_one and opponent_two. This function has a precondition – it will not work unless both parameters are well-formed dictionaries each containing a correct character. We are not responsible for what happens if the user of this function passes anything else to it.
   (b) Each character starts by rolling 1d20. Whoever rolls the highest number strikes first. If the characters roll the same number, roll again to get a winner. Keep rolling until someone rolls a higher value.
   (c) The first attacker rolls a 1d20 and compares the rolled number to the other character's dexterity. If the attacker rolled a number higher than the victim's dexterity, the attacker has successfully struck their opponent!
   (d) Of course IRL damage depends on the attacker's strength, armour, and weapon, but let's begin with something simple. Let's assume that the damage a character inflicts is equivalent to their hit dice. That is, if we rolled a 1d10 to determine a character's hit points (HP), that character will roll a 1d10 when metering out damage. Reduce the other character's HP by that amount.
   (e) Print a meaningful message to the screen. If the other combatant was killed, that's definitely some information we should know.
   (f) If the second combatant is still alive, they must roll 1d20 and try to get a value greater than the other player's dexterity. If they do, roll the die they used to determine their HP to find out how much damage they inflicted.
   (g) The function must print useful, interesting information during the round. Make me fear for my life! Make my blood sing in my ears! Most importantly, tell me how much damage was inflicted, and if someone dies, that's useful too.
   (h) The function returns nothing.
5. Create an `if __name__ == "__main__"` block in dungeonsanddragons.py and add code that invokes a main function.
6. In the main function, create a short interactive program for me that demonstrates how your code works. Every function must be showcased. Ensure the functions roll_die and print_character are used. They may need to be modified too.
7. That's it. Not so hard, was it?

5 Grading

Your second assignment will be marked out of 10. I will randomly select some functions from this assignment and mark them. For full marks, you must:

1. (5) Correctly implement the requirements described in this document for each function I mark.
2. (3) Correctly write and execute doctests for each function I mark that unambiguously describe the function's correct use, and correctly write and execute unit tests to thoroughly test each function I mark.
3. (1) Correctly commit and push your code to git and GitHub at regular intervals, i.e., after you complete each function.
4. (1) Correctly format and comment your code. Eliminate all warnings offered by PyCharm, use good function and variable names, write code that is easy to understand, use whitespace wisely, employ good grammar in your comments, use inline comments (comments that start with #) inside functions to describe complicated code blocks, **make sure functions are brief and only do one thing (I'm looking at combat_round for example and my first thought is it should have a helper function called attack)**, etc.

Please remember that this is an individual assignment. I strongly encourage you to share ideas and concepts (please!), but sharing code or submitting someone else's work is not allowed.

Good luck, and have fun!