Homework 5

The due date for this homework is **Friday, March 29, 11pm EDT**.

Part A: The Game of Nim

Nim is a mathematical game of strategy in which two players take turns removing (or "nimming") objects from a set of piles. Variants of Nim have been played since ancient times, with the game likely originating in China.

At the 1940 New York World's Fair Westinghouse displayed a machine, the Nimatron, that played Nim. From May 11, 1940, to October 27, 1940, only a few people were able to beat the machine in that six-week period; if they did, they were presented with a coin that said Nim Champ. It was also one of the first-ever electronic computerized games. (Wikipedia)

Fig. 1: A woman playing against the Nimatron at the World's Fair

For this week's homework, we're going to create a simplified variant of Nim where there is only one pile of objects. The goals for this homework are to get you thinking about building more complex programs from smaller functions.

Game Rules

The rules for our simple version of Nim are as follows:

1. The game Nim starts out with a random number of stones on the table between 7 and 15 (inclusive).
2. Each player takes turns picking up 1, 2 or 3 stones.
3. Whoever is forced to pick up the last stone loses. In other words, the player that leaves 1 stones left for their opponent wins.

In this implementation, the (human) player will play against the computer. They should alternate turns, starting with the computer.

Some requirements:

1. The player's input should be validated (i.e., checked to see if it is a valid input). A valid input is a number. It is usually 1, 2, or 3, unless that number would leave less than 1 stone behind. (A user can't say they will take 3 stones away from a pile with only 3 stones left).
2. The computer's input should be randomly chosen between 1 and 3 (inclusive), but with the same restrictions (the number chosen must leave at least 1 stone behind).

Writing docstrings for functions

Docstrings allow us to write documentation for our functions. Documentation is important because it gives a high-level description of our code and its design. In `hw5_basic.py`, we have provided you with the docstrings for the `main` and `computer_turn` functions. For this homework, you are required to write docstrings for all the functions that you are going to implement. Note a few requirements when writing docstrings:

- the docstring must be writen at the beginning of the function, inside a beginning and ending triple double quotes (""") and the entire body of the docstring must be indented accordingly.
- the first phrase in the docstring gives a succinct overview of the function. Refrain from including code implementation details in the docstrings. For example, note that the docstring for `computer_turn` does not specify how the random numbers are generated or the details of how the code ensures that the randomly generated integer is less than the number of remaining stones.
- the docstring must include the section `Parameters` followed by the list of parameters, their type and a description of what the parameter represents. If the function has no parameters then write `None`.

- similarly, the docstring must include the section `Returns` followed by the list of return values, their type and a description of what the parameter represents. If the function has no return then write `None`.

Run `hw5_basic.py`, which runs `help(computer_turn)`. The `help` function is a built-in function that takes in the name of a function and returns the documentation (docstring) for that function. In this case, the output you should see in the terminal is the dosctring for the `computer_turn` function that we have provided you with. Comment out this line and proceed to implementing the code for the utility functions. When you are done writing the docstrings for all your functions, you may test that they are correctly implemented by using the `help` function.

Helper (Utility) Functions

To build this game, we'll be making use of a series of helper functions (described below). All these functions must be implemented in `hw5_basic.py`. For each function, **you must include a correctly formatted docstring**.

`print_header(stones_in_pile: int, turn: int) -> None` prints a header for each turn. One line of the header indicates the turn number (i.e., Turn 1, Turn 2, etc.). Another line draws the number of stones (`O`) in the pile. For example a pile of seven stones should be printed as: `OOOOOOO`

`single_game() -> None` executes an entire single game of Nim. This should call the other functions.

`player_turn(stones_in_pile: int) -> int` asks the user for the number of stones they would like to remove and checks to see if that input is valid (see restrictions above). If the input is invalid, it should keep asking until a valid input is provided. This function returns the valid user input.

`computer_turn(stones_in_pile: int) -> int` chooses a random number of stones to remove, given restrictions (see above).

Example Output

```
-- Turn 1 --
OOOOOOOOOOOOO

Computer removes 3

-- Turn 2 --
OOOOOOOOOO

How many stones would you like to remove? 4
Invalid input
How many stones would you like to remove? 3
Player removes 3

-- Turn 3 --
OOOOOOO

Computer removes 1

-- Turn 4 --
OOOOOO

How many stones would you like to remove? 3
Player removes 3

-- Turn 5 --
```

```
OOO

Computer removes 2

Game Over!
Computer wins
```

Part B: Testing Validation

Validating user input is an important part of this game. In a comment under the `player_turn()` function, give some examples of bad user input that you could use to test whether the validation works as expected. Describe why those would be good test cases.

Part C: Advanced Computer

As of now, our computer's strategy is pretty basic (it's completely random). To illustrate: If there were 4 stones remaining, a (smart) human player would obviously choose to take 3 so they would win, but our computer might choose randomly choose 1 or 2 instead.

Write a new, improved version of `computer_turn()` called `smart_computer_turn()` that will always win if there are 4 or fewer stones remaining in `hw5_advanced.py`. In `hw5_advanced.py` also implement another version of `single_game` that calls `smart_computer_turn()` instead of `computer_turn()`. For the other helper functions (`print_header` and `player_turn`), you can either import or copy over the functions in the `hw5_basic` module.

Grading

Your assignment will be graded on the following criteria:

1. [60%] Part A: Be sure to run your code before submitting to check the accuracy of it.
2. [10%] Part B
3. [15%] Part C:
4. [15%] Program Design and Style:
    - Variable names should be meaningful
    - Programs should contain at least a few descriptive comments. Do NOT comment every line of code with low level explanations of what each line does. Focus on high level ideas. For example, write only ONE comment for each loop or conditional statement.
    - The functions should be structured so that the logic is clear and easy to follow.
    - **All functions must have docstrings!**