A5: The Game of Nim

- Assignment A5 should be complete individually or with one partner.
- Be sure to read the entire prompt and understand the problem before beginning coding.

Objectives

- Practice breaking a larger problem down into smaller "mental chunks"using functions
- Gain practice using loops and modulus (%)

Introduction to Top-Down Design

A good way to design large programming problems is to use functions to break them down into a series of smaller sub-problems, which can be solved somewhat independently and then combined to arrive at a complete solution. You can essentially focus your attention on solving each piece of the program, and not get confused with the overall challenge of the whole program. After checking that each function works correctly (i.e., for a set of inputs, it produces the expected output), you can then start to stitch the pieces together, now factoring in how they interlock. Such a process is methodical and reduces the chances of unexpected errors appearing, because something you expected for one function had unintended consequences when working with another function. This is known as **coupling**, and is best to have a program with minimum coupling between functions.

This technique of breaking a program down into manageable mental chunks is called **top-down design**. The programmer typically breaks the algorithm down into smaller and smaller sub-algorithms. A solution derived from such a planned-out design will tend to be composed of small, manageable code segments which:

- tackle a specific smaller task,
- are self-contained, and only the inputs and outputs matter,
- will likely be easier to write, and
- will be easier to test and debug, so will save you time in the long run!

One way of testing these small, manageable code segments is known as **black-box testing**. In **black-box testing** of a function, the tester only knows the desired behavior of the function, namely what the inputs and anticipated expected outcome should be, but not how the function computes the output. You've already seen black-box testing in T4, when you created test cases for the `i_steal_pennies()` function:

```
testit(i_steal_pennies(0.88)==[3, 1, 0, 3])
```

Pseudocode is an outline of the logic of a program in regular English. It uses the structural conventions of a programming language but is intended for a human, not a machine, to read. In the process of creating the pseudocode, the comments describing each function will be easier for others to understand and later modify the code as necessary (a vital skill in the programming industry!). Writing the pseudocode for a program is a good practice before writing code, because it's easy to ignore syntax problems and focus on creating a good design to the program. Here is an example pseudocode for the is_even.py code:

```
function is_even(y):
      if y is divisible by 2:
            return True
      else:
            return False

function main():
      year = a random integer between 0 and 2015
      if is_even(year):
            print "Year is even"
      else:
            print "Year is odd"
main()
```

Notice the syntax is not Python, but it reads like code. Pseudocode is intended to be readable by a human, but also convertible to code by a programmer.

The Game of Nim

Nim is a game of logic and strategy. The goal of Nim is to be the player who removes the last of some group of items (e.g., balls) from a pile. A player must remove some number during their turn. The player who removes the last one wins. You can try a version of the game at: http://education.jlab.org/nim/index.html against the computer. In this version, there are 9 or 10 protons in a pile, and you and the computer are each allowed to take either 1 or 2 protons each time. The player who takes the last proton wins!

The instructions

A key idea in this teamwork is to use fruitful functions for returning values. Note that when a function returns a value, it is typical to put the value into a variable or use it directly.

The following examples illustrate these ideas and also demonstrate the use of modulus:

- peel_digits.py
- is_even.py

Your task is to create the game of Nim that works as follows (which is slightly different than the pepper example done in class):

- The human player will give the program the number of balls to start.
- The starting number of balls must be 15 or higher.
- The player will then play against the computer, with the human player going first.
- The player and the computer will then take turns choosing between 1 to 4 balls to remove from the pile.
- During the human player's turn, your program will need to ask the human player to input the number of balls to remove.
- During the computer's turn, your program will remove a number of balls. More on how your algorithm determines this number below… for now, assume it's between 1 and 4 balls.
- **The player who takes the last ball wins**! Inform the user who wins at the end of the program.

Hints and Requirements

- Use functions to break code into smaller logical groupings for easier debugging.
- After asking the human player for the number of balls to start, test it to make sure it is 15 or larger. Use a loop to allow the player re-enter the number until it is 15 or larger.
- You will also need to use a loop in order to have play continue until the last ball is picked.
- In the loop, you will need to alternate between the human player and the computer, but you may assume the human always plays first.
- When it is the human player's turn, your program will ask him/her to input the number of balls they wish to remove. Test this input to make sure it is between 1 and 4, using a loop to continually ask for the number of balls until it is valid (i.e. between 1 and 4).
- Create a fruitful function which determine how many balls the computer will pick up using the following strategy (NOTE: The following reveals how to maximize the computer's wins. Try to solve it first on your own, based on what you learned in class. The answer is provided here if you can't come up with a solution, because I am more interested in your implementation than your ability to solve this puzzle):
  - [redacted]
    - Return the number of balls the computer will take from your fruitful function.

- Be sure to announce who has won!
- Be sure to include our standard header at the top of your program with name, username, assignment number, purpose and acknowledgements. Be sure to give credit where credit is due, including your partner (if you worked with one), and any help you received.
- Be sure to include good coding practices as we've discussed in class, including following coding standards (see Chapter 7 on Iteration) and the use of a `main()` function.
- (Optional) You may optionally use graphics, but please make the logic work first because debugging with graphics is harder. Please be sure to include comments for the sections in your code which does graphics.

Design Write-up

This assignment requires more planning and design than previous assignments. As such, I'd like for you to have a process of documenting your thought process throughout this assignment. Use the space below to document your process. You'll fill out parts of this table at different points in the process of thinking about your design and creating code.

INITIAL DESIGN PLAN: Design a pseudocode plan which meets the computational requirements to solve the problem. Your pseudocode does not need to be syntactically correct. It needs to capture the flow of logic in a human readable format.	1. First use a loop to ask the player how many balls to start with until they pick a number that is 15 or higher. 2. Have a loop so that the game will keep playing until the ball count is 0. 3. Have the player take a number of balls 1-4 and have the computer take a number of balls that will allow it to win. If it cannot win based off of the player's previous move, take a random number of balls.
IMPLEMENTATIONS: A list in bullet form of specifically what was accomplished, including any challenges overcome and innovations that were not specifically required by the assignment.	1. I made it to where the game will play based on any starting number higher than 15. 2. I made it to where any chance the computer has to win, it will. 3. I made the program restrict what you could input when you got down to less than 4 balls left, to ensure you didn't go into negative numbers. 4. Had the end result print if you won or lost.
TESTING: A list in bullet form of all input values used for testing. Here you should be careful to select representative input cases, including	For starting balls I had 4 different values. ● 15 ● 16

both representative typical cases as well as extreme cases.	• 150 • 151 This was to test both the always winning side of the computer and the random integer side of the computer at both the minimum side, as well as an extreme side. Before I implemented the computer, I needed to test to make sure my limitations on the values that could be inputed when you reached less than 4 worked. To do this I once I reached 3, 2, and 1, I used the values 4, 3, and 2 respectively. So, • 4 when it is 3 • 3 when it is 2 • 2 when it is 1
ERRORS: A list in bullet form of all known errors and deficiencies in your implementation.	• If you put anything other than an integer when prompted, you will receive an error.
SUMMARY: A brief summary description of the design and implementation, including how much your initial design plan evolved, the final result you achieved, and the amount of time you spent as a programmer in accomplishing these results. This should be no more than two paragraphs. Consider this like a report of what you did.	My initial design was just really broad. I implemented everything in my plan, but there were some things that I had to add in that I did not account for, such as limiting input values when you reached less than 4 so you did not go below 0. The final result was a fairly good working nim game. If you make a mistake the computer will always win, but if you know what you are doing, you can always win. I spent about 2 hours making this program.
COMMENTS: A paragraph or so of your own comments on and reactions to this assignment. Consider this like a reflection.	I didn't think this assignment was too bad. I am just not sure how specific I need to be when doing the pseudo code. As previously mentioned, I was able to implement everything I said prior to coding, however, there were some extra things that I needed to add on later to make everything function properly.

Submission Instructions

1. Review the requirements above to ensure you completed everything that was expected of you.
2. Save your code as **A5_Nim_*username*.py**. Replace *username* with your Berea usernames. For example, the TA Bianca Marrero's file would be **A5_Nim_marrerob.py.**
3. Save this document at **A5_Nim_*username*.pdf.** Replace *username* with your Berea usernames.
   NOTE: Incorrect filenames will automatically reduce your grade by 1 point. Fortunately, the format is always the same no matter what the assignment.
4. Zip the two files together.
5. Upload the Zip file to Moodle by the due date listed on the course website: https://trello.com/b/w7bIrLoV/.