Black Problem 3: Nim! [35 points; individual or pair]

Copied from: https://www.cs.hmc.edu/twiki/bin/view/CS5/RealNimBlack on 3/22/2017

Please make sure that you have read the instructions on the Assignment 8 "Black" main page so that you understand the choices available to you this week.

In this problem you will implement the game of Nim. In particular, the user will play against the computer and **the computer will use the optimal "Nim Sum" algorithm described in class.**

The Functionality of Your Program

Your Nim program should have the following features:

- The program begins "automatically": We just include the line
- `if __name__ == "__main__":`
- `    main()`

  at the bottom of the file (anywhere *after* the definition of the `main()` function itself—typically at the very bottom of the file) and Python will automatically invoke the `main()` function when the program is invoked at the shell. Those long lines before and after `name` and `main` are pairs of underscore symbols: There are two consecutive underscores at the beginning and two consecutive ones at the end.

- The program begins by offering a welcome message and then asking the following questions in this order:
  - "How many piles do you want?" Any integer greater than 1 is valid. If the input is not an integer or is invalid (less than or equal to 1), the program must detect that fact and ask again. You will find the `try...except...` construct discussed in class useful here.
  - For each pile, the user should be asked "How many coins do you want on pile 0?" (and then on pile 1, and pile 2, etc.). Again, you must detect and reject all invalid inputs.
- The program now operates as follows until either the computer or player wins, in which case the winner is declared by the computer.

Note that the computer is "player 1" (the player that makes the first move):
    - The number of coins on each pile is shown.
    - If it's the computer's move, the computer indicates the number of coins removed from the selected pile.
    - If it's the player's move, the player is asked to specify a pile and a number of coins to remove from that pile. If the player tries to select a pile with no coins remaining, or in an invalid pile number, the player should be told that this is an invalid selection and asked to try again. Similarly, if the player tries to remove an invalid number of coins (0 coins or more coins than are in the pile) then the player is told that this is invalid and to try again. As before, you must also detect and reject non-integer inputs.
- When the game is over, the player is asked if he/she would like to play again. If so, the fun starts over.

Here is some sample input and output:

```
Welcome to Nim!  I'm probably going to win...
How many piles would like?  4
How many coins in pile 0? 2
How many coins in pile 1? 3
How many coins in pile 2? 7
How many coins in pile 3? 5
I remove 1 coins from pile 0
------------------------
Pile 0 has 1 coins
Pile 1 has 3 coins
Pile 2 has 7 coins
Pile 3 has 5 coins
Which pile would you like to remove from? 4
There's no such pile.  Try again.
Which pile would you like to remove from? 2
How many coins do you wish to remove from pile 2? -5
That's not a valid number of coins.  Try again.
How many coins do you wish to remove from pile 2? 4
------------------------
Pile 0 has 1 coins
Pile 1 has 3 coins
Pile 2 has 3 coins
Pile 3 has 5 coins
I remove 4 coins from pile 3
------------------------
Pile 0 has 1 coins
Pile 1 has 3 coins
Pile 2 has 3 coins
```

```
Pile 3 has 1 coins
Which pile would you like to remove from? 1
How many coins do you wish to remove from pile 1? 3
-----------------------
Pile 0 has 1 coins
Pile 1 has 0 coins
Pile 2 has 3 coins
Pile 3 has 1 coins
I remove 3 coins from pile 2
-----------------------
Pile 0 has 1 coins
Pile 1 has 0 coins
Pile 2 has 0 coins
Pile 3 has 1 coins
Which pile would you like to remove from? 0
How many coins do you wish to remove from pile 0? 1
-----------------------
Pile 0 has 0 coins
Pile 1 has 0 coins
Pile 2 has 0 coins
Pile 3 has 1 coins
I remove 1 coins from pile 3
-----------------------
Pile 0 has 0 coins
Pile 1 has 0 coins
Pile 2 has 0 coins
Pile 3 has 0 coins
I win!
Would you like to play again? n
Thanks for playing!  Bye.
```

The Design of Your Program

Your program will be evaluated carefully for quality of design, and approximately half of the score will be for following the good design principles outlined in class and summarized below.

- Think about the design of your program and lay out the functions that you'll need **on paper** before you do any actual programming. In particular, try to identify the separate logical parts of your program. For example, you might have a part of your program that prints the contents of the board, another part that choose a move, and so forth. For each part, ask yourself what arguments it will require and what result it will produce. These parts can then be translated into Python functions.

- Make sure that each of your Python functions really encapsulates one particular well-defined task. Then, write the functions one by one. For each function, include a docstring that explains the arguments that the function takes, what the function is "responsible for doing", and what result it produces. If your function is more than 10 or 12 lines of code, ask yourself if it makes sense to break it up into separate subfunctions. The answer may be "no"—but think about it.
- Think about your code carefully before you write it. There is usually more than one way to get a computational task done. Software takes a "short" time to write relative to the amount of time that it is used. It's worth programming slowly and making sure that the code that you write is clear, simple, and easy for you (or someone else) to go back and read and/or modify. If your code seems complicated, it probably can be rewritten in a simpler and more elegant way.
- Use comments when there is something in your code that is not self-evident. You don't need to have a comment for every line, but lines or blocks of code that do something that is not immediately obvious to the casual observer merit at least some documentation. You will find that it takes some experience to decide what is "not immediately obvious"; if in doubt, ask a professor or grutor.
- Test each function as you write it. Make sure that the function that you just wrote behaves correctly before moving on to the next function. *This will potentially save you enormous amounts of time and anguish when debugging.*
- Avoid global variables! `debug` is a notable exception, and on very rare occasions it makes sense to introduce others. In general though, it's cleaner and safer for functions to pass one another just what they need to do their jobs.
- Use descriptive function and variable names. A function name like `printBoard` is more descriptive than `pb`. Similarly, a variable name like `coins` is more descriptive than `c` or `x`.

For the sake of debugging, you should have the following line at the top of your code:

```
debug = True # debug is True if debugging info is to be
displayed, and False otherwise
```

Throughout your code, you should have conditional statements of the form:

```
if debug:
    print blah, blah, blah
```

that print out key information about the inner workings of your program.

In particular, if `debug` is `True`, your program should print out the XOR of the number of coins in each pile. This will allow you to ensure that the behavior

of your program is correct. Later, of course, you can set `debug` to `False` to deactivate the printing of this information.

Here are some specific observations for this problem:

- In some cases you may find yourself doing the same thing to each element in a list. In this case, it's much simpler and more elegant to use `map` (possibly with an anonymous function) than to build your own code to do this iteratively. **In general, use functional concepts where they make your code simpler.**
- You may wish to include code from an earlier assignment (in particular to convert between base 10 and base 2). You may find it preferable to modify that code to use lists rather than strings. It's up to you.
- Be sure to have a `debug` global variable (this should be your only global variable). When this variable is `True`, the program should show all kinds of internal details of the inner working of your program that could be useful during debugging. We'll test your program with this variable set to `True` and `False` to see its behavior in both cases.

Submit!