6.189 - Intro to Python
IAP 2008 - Class 3
Lead: Aseem Kishore

Lab 4: Control Flow with While

Problem 1 - Login security

One important aspect of security in computer science is the concept of *hashing*: taking some text, and somehow converting it to a number. This is needed because many security algorithms work through math, so numbers are needed.

Another important aspect is the use of the modulo operator (%). You've seen this -- it returns the remainder portion of a division. This is useful because unlike most other math operators, modulo is one-way. That is, I can tell you that I'm thinking of a number *x*, and when I mod it by 5, I get 3, but from this information alone, you don't know whether *x* is 3 or 8 or 13 or 18, or ...

In this problem, we'll create a login screen, where the user must enter a password in order to see a secret message. We will give the user 3 chances to get the password right, and either print the secret message or a failure message (after 3 chances).

First, define a function **encrypt** that takes one string. It will hash the string using the built-in Python function **hash** (try it on the shell) and modulo the value by a prime number (e.g. 541 -- this is very small in the computer science world but good enough for us). The function should then return this number.

e.g. **encrypt("mypassword") -> 283** (if you use 541 as the prime, for example)

At the top of the file, define a variable _**KEY** to be the result, e.g. _**KEY = 283**.

Now, write the rest of the program. Each time you ask the user for the password, call **encrypt** with the user's input, and compare the value to _**KEY**. If the two match, the user (most likely) entered the correct password, otherwise he loses one chance.

(see back for Problem 2)

Problem 2 - The game of Nims / Stones

In this game, two players sit in front of a pile of 100 stones. They take turns, each removing between 1 and 5 stones (assuming there are at least 5 stones left in the pile). The person who removes the last stone(s) wins.

Write a program to play this game. This may seem tricky, so break it down into parts. Like many programs, we have to use nested loops (one loop inside another).

In the outermost loop, we want to keep playing until we are out of stones.

Inside that, we want to keep alternating players. You have the option of either writing two blocks of code, or keeping a variable that tracks the current player. The second way is slightly trickier since we haven't learned lists yet, but it's definitely do-able!

Finally, we might want to have an innermost loop that checks if the user's input is valid. Is it a number? Is it a valid number (e.g. between 1 and 5)? Are there enough stones in the pile to take off this many? If any of these answers are no, we should tell the user and re-ask them the question.

So, the basic outline of the program should be something like this:

```
TOTAL = 100
MAX = 5

pile = TOTAL        # all stones are in the pile to start

while [pile is not empty]:

    while [player 1's answer is not valid]:
        [ask player 1]
        [check player 1's input... is it valid?]

    [same as above for player 2]
```

Note how the important numbers 100 and 5 are stored in a single variable at the top. This is good practice -- it allows you to easily change the constants of a program. For example, for testing, you may want to start with only 15 or 20 stones.

Be careful with the validity checks. Specifically, we want to keep asking player 1 for their choice as long as their answer is not valid, BUT we want to make sure we ask them at least ONCE. So, for example, we will want to keep a variable that tracks whether their answer is valid, and set it to False initially.

When you're finished, test each other's programs by playing them!