Chris Piech and Mehran Sahami   Assignment #2
Code in Place   April 22, 2020

Assignment #2: Khan-sole Academy
Due: 11:59pm Anywhere on Earth, May 1st

Based on problems by Chris Piech, Mehran Sahami, Nick Bowman, Sonja Johnson-Yu, Kylie Jue, and more

This assignment consists of multiple short problems to give you practice with several different concepts in Python (e.g., variables, control structures, etc.) as well as two longer programs. You can download the starter code for this project under the "Assignments" tab on the Code in Place website. The starter project will provide Python files for you to write your programs in. PyCharm not working on your computer? Ed has an online IDE where you can solve these problems.

Everyone does Part 1 (Sandcastles). **Part 2 and 3 are optional but recommended**.

(1)   (2)

(3)

This assignment has three parts:
1. Sandcastles
2. Khan-sole Academy (Optional)
3. The Ancient Game of Nimm (Optional)

Part 1: Sandcastles

"Sandcastle" problems are meant to be fairly straightforward and make sure you're solid on particular concepts (like control flow, variables, and functions) before moving on to writing larger programs. They're kind of like building sandcastles in a sandbox – they're meant to be fun to do and no one gets hurt. Do these problems first!

A. Write a program in the file `subtract_numbers.py` that reads two ***real numbers*** from the user and prints the first number minus the second number. You can assume the user will always enter valid real numbers as input (negative values are fine). Yes, we know this problem is really similar to a problem we did in class – that's why this problem is a sandcastle! A sample run of the program is shown below:

```
This program subtracts one number from another.
Enter first number: 5.5
Enter second number: 2.1
The result is 3.4
```

B. Write a program in the file `random_numbers.py` that prints 10 random integers (each random integer should have a value between 0 and 100, inclusive). Your program should use a constant named `NUM_RANDOM`, which determines the number of random numbers to print (with a value of 10). It should also use constants named `MIN_RANDOM` and `MAX_RANDOM` to determine the minimal and maximal values of the random numbers generated (with respective values 0 and 100). To generate random numbers, you should use the function `random.randint()` from Python's `random` library:

```
35
10
45
59
45
100
8
31
48
6
```

C. Write a program in the file `liftoff.py` that prints out the calls for a spaceship that is about to launch. Countdown the numbers from 10 to 1 and then write "Liftoff!" Your program should include a `for` loop using `range`. A sample run of the program is below.

```
10
9
8
7
6
5
4
3
2
1
Liftoff!
```

Part 2: Khan-Sole Academy

Now that you've seen how programming can help us in a number of different areas, it's time for you to implement Khan-sole Academy—a program that helps other people learn! In this problem, you'll write a program in the file `khansole_academy.py` that randomly generates simple addition problems for the user, reads in the answer from the user, and then checks to see if they got it right or wrong, until the user appears to have mastered the material. Note that "console" is another name for "terminal" :-).

More specifically, your program should be able to generate simple addition problems that involve adding two 2-digit integers (i.e., the numbers 10 through 99). The user should be asked for an answer to each problem. Your program should determine if the answer was correct or not, and give the user an appropriate message to let them know. Your program should keep giving the user problems until the user has gotten 3 problems **correct *in a row***. (Note: the number of problems the user needs to get correctly in a row to complete the program is just one example of a good place to specify a constant in your program).

A sample run of the program is shown below (user input is in *italics*).

```
What is 51 + 79?
Your answer: 120
Incorrect.  The expected answer is 130
What is 33 + 19?
Your answer: 42
Incorrect.  The expected answer is 52
What is 55 + 11?
Your answer: 66
Correct!  You've gotten 1 correct in a row.
What is 84 + 25?
Your answer: 109
Correct!  You've gotten 2 correct in a row.
What is 26 + 58?
Your answer: 74
Incorrect.  The expected answer is 84
What is 98 + 85?
Your answer: 183
Correct!  You've gotten 1 correct in a row.
What is 79 + 66?
Your answer: 145
Correct!  You've gotten 2 correct in a row.
What is 97 + 20?
Your answer: 117
Correct!  You've gotten 3 correct in a row.
Congratulations!  You mastered addition.
```
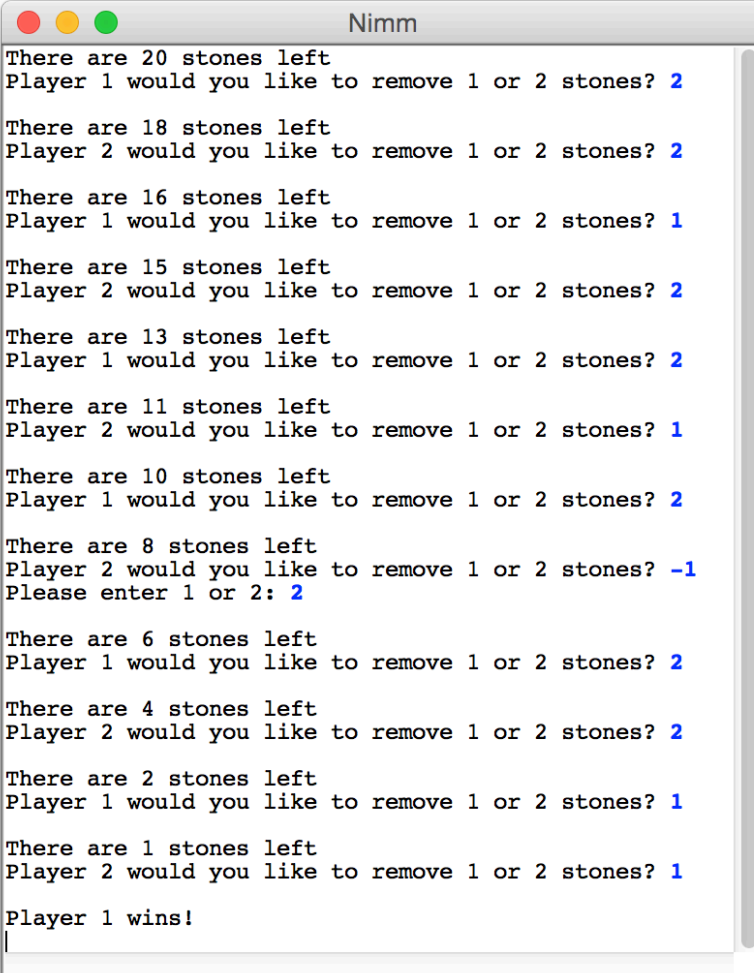
As a side note, one of the earliest programs Mehran wrote (with his friend Matthew) when he was first learning how to program was very similar to Khansole Academy. It was called "M&M's Math Puzzles." It was written in a language named BASIC on a computer that had 4K of memory (that's 4 Kilobytes) and used cassette tapes (the same kind used for music in the 1970's) to store information. Yeah, Mehran is old.

Part 3: Ancient Game of Nimm

Nimm is an ancient game of strategy that is named after the old German word for "take." It is also called Tiouk Tiouk in West Africa and Tsynshidzi in China. Players alternate taking stones until there are zero left. The game of Nimm goes as follows:

1. The game starts with a pile of 20 stones between the players
2. The two players alternate turns
3. On a given turn, a player may take either 1 or 2 stone from the center pile
4. The two players continue until the center pile has run out of stones.

The last player to take a stone loses. Here is a screenshot from an example execution:

```
●  ●  ●                  Nimm
There are 20 stones left
Player 1 would you like to remove 1 or 2 stones? 2

There are 18 stones left
Player 2 would you like to remove 1 or 2 stones? 2

There are 16 stones left
Player 1 would you like to remove 1 or 2 stones? 1

There are 15 stones left
Player 2 would you like to remove 1 or 2 stones? 2

There are 13 stones left
Player 1 would you like to remove 1 or 2 stones? 2

There are 11 stones left
Player 2 would you like to remove 1 or 2 stones? 1

There are 10 stones left
Player 1 would you like to remove 1 or 2 stones? 2

There are 8 stones left
Player 2 would you like to remove 1 or 2 stones? -1
Please enter 1 or 2: 2

There are 6 stones left
Player 1 would you like to remove 1 or 2 stones? 2

There are 4 stones left
Player 2 would you like to remove 1 or 2 stones? 2

There are 2 stones left
Player 1 would you like to remove 1 or 2 stones? 1

There are 1 stones left
Player 2 would you like to remove 1 or 2 stones? 1

Player 1 wins!
```

Write a program to play Nimm. To make your life easier we have broken the problem down into smaller milestones. You have a lot of time for this program. Take it slowly, piece by piece.
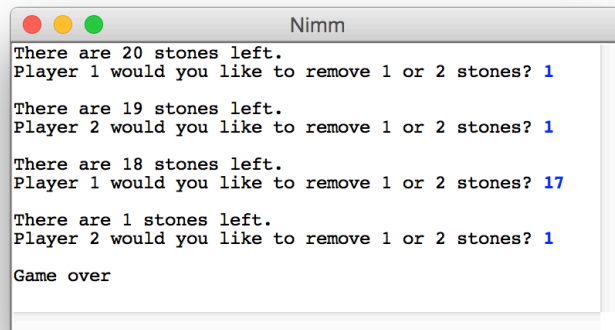
Milestone 1

Start with 20 stones. Repeat the process of removing stones and printing out how many stones are left until there are zero. Don't worry about whose turn it is. Don't worry about making sure only one or two stones are removed. Use the method input(*msg*) which prints msg and waits for the user to enter an input. Make sure to convert the input into an int.

```
●  ●  ●                    Nimm
There are 20 stones left
Would you like to remove 1 or 2 stones? 2

There are 18 stones left
Would you like to remove 1 or 2 stones? 17

There are 1 stones left
Would you like to remove 1 or 2 stones? 3

Game over
```
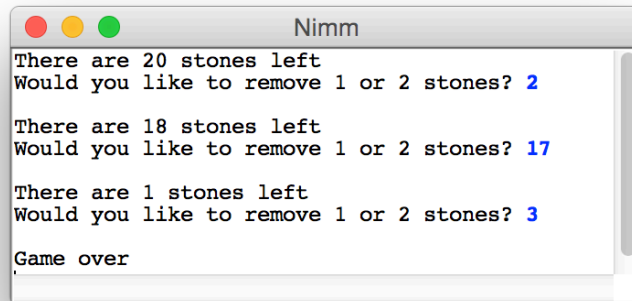
Milestone 2

Create a variable of type int to keep track of whose turn it is (remember there are two players). Tell the user whose turn it is. Each time someone picks up stones, change the player number.

```
●  ●  ●                    Nimm
There are 20 stones left.
Player 1 would you like to remove 1 or 2 stones? 1

There are 19 stones left.
Player 2 would you like to remove 1 or 2 stones? 1

There are 18 stones left.
Player 1 would you like to remove 1 or 2 stones? 17

There are 1 stones left.
Player 2 would you like to remove 1 or 2 stones? 1

Game over
```

Milestone 3

Make sure that each turn only one or two stones are removed. After you read a number of stones to remove from a user (their input), you can use the following pattern to check if it was valid and keep asking until it is valid.

```
while inputisinvalid :
    input = int(input("Please enter 1 or 2: "))
```

As a final touch, announce the winner after the game is over.

Part X: Optional Extensions

Once you've completed all the required parts of the assignment, you might want to consider adding some extensions. Extensions, you may recall, are things that are totally optional. Here are some extra programs to write if you are interested – but feel free to just make something cool!

Extend Khansole Academy

You could consider extending your Khansole Academy program to, for example, add more problem types (subtraction, multiplication, division, and more). You could also consider problems beyond arithmetic. If you could build your own version of Khansole Academy, what would you use it to help people learn? Be creative and enjoy.

AI for Game of Nimm

Can you make a computer player that can always win in a game of Nimm?

Hailstones

A separate (optional) problem you could consider writing is based on a problem in Douglas Hofstadter's Pulitzer-prize-winning book _Gödel, Escher, Bach_. That book contains many interesting mathematical puzzles, many of which can be expressed in the form of computer programs. In Chapter XII, Hofstadter mentions a wonderful problem that is well within the scope of what you know. The problem can be expressed as follows:

> Pick some positive integer and call it n.
> If n is even, divide it by two.
> If n is odd, multiply it by three and add one.
> Continue this process until n is equal to one.

On page 401 of the Vintage edition of his book, Hofstadter illustrates this process with thfollowing example, starting with the number 15:

15	is odd, so I make $3n+1$:	46
46	is even, so I take half:	23
23	is odd, so I make $3n+1$:	70
70	is even, so I take half:	35
35	is odd, so I make $3n+1$:	106
106	is even, so I take half:	53
53	is odd, so I make $3n+1$:	160
160	is even, so I take half:	80
80	is even, so I take half:	40
40	is even, so I take half:	20
20	is even, so I take half:	10
10	is even, so I take half:	5
5	is odd, so I make $3n+1$:	16
16	is even, so I take half:	8
8	is even, so I take half:	4
4	is even, so I take half:	2
2	is even, so I take half:	1

As you can see from this example, the numbers go up and down, but eventually—at least for all numbers that have ever been tried—comes down to end in 1. In some respects, this process is reminiscent of the formation of hailstones, which get carried upward by the

winds over and over again before they finally descend to the ground. Because of this analogy, this sequence of numbers is usually called the **Hailstone sequence,** although it goes by many other names as well.

You might want to write a Python program that reads in a number from the user and then displays the Hailstone sequence for that number, just as in Hofstadter's book, followed by a line showing the number of steps taken to reach 1. For example, here's a sample run of what such a program might look like (user input is in *italics*):

```
Enter a number: 17
17 is odd, so I make 3n + 1: 52
52 is even, so I take half: 26
26 is even, so I take half: 13
13 is odd, so I make 3n + 1: 40
40 is even, so I take half: 20
20 is even, so I take half: 10
10 is even, so I take half: 5
5 is odd, so I make 3n + 1: 16
16 is even, so I take half: 8
8 is even, so I take half: 4
4 is even, so I take half: 2
2 is even, so I take half: 1
The process took 12 steps to reach 1
```