15-112 Fall 18

Home

Syllabus

Schedule

Staff

Gallery

Piazza

Autolab

OH Queue

CMU 15-112 Fall 2018: Fundamentals of Programming and Computer Science Homework 2 (Due Sunday 9-Sep, at 5pm)

- Reminder: all problems that are not explicitly marked COLLABORATIVE must be completed individually, as stated in the course syllabus.

- To start:
  1. Create a folder named 'week2'
  2. Download both hw2.py and cs112_f18_week2_linter.py to that folder
  3. Edit hw2.py using Pyzo
  4. When you are ready, submit hw2.py to Autolab. For this hw, you may submit up to 10 times, but only your last submission counts.
- Do not use string indexing, lists, or recursion this week.
- Do not hardcode the test cases in your solutions.

---

1. Group sessions [10pts]
   Attend and participate in at least one small group or large group or extensive review covering Week 2 content. One of the TAs will record your participation by hand.

2. Piazza Post [5pts]
   To succeed in 15-112, you'll need to understand how to use the course resources. Towards that end, by 5pm Sunday 9-Sep, you must ask a question on Piazza about the homework or coding in general. This has to be a real question! We will check for this after Sunday and will add points on Autolab manually at that point. If you have already asked a question on Piazza, we will count that question for credit (though you're still encouraged to ask more questions!).

3. Debugging: Debug the Programs [10pts]
   Each of the following three programs has one error (or bug) in it somewhere. Your task is to find and fix that bug without substantially rewriting the program. Specifically, each bug can be fixed by modifying only one of the program lines (which will result in full credit); if you modify more than one line, you'll only get partial credit (or no credit if the modifications are excessive). We define 'modify' to mean either changing one line, adding one line, or deleting one line.

   1. countEvenDigits(n)
      This program is supposed to count the number of even digits that appear in the parameter n. Remember, 0 is even!

```
def countEvenDigits(n):
    if n < 0:
        return countEvenDigits(-n)
    elif n == 0:
        return 1
    count = 0
    while n > 0:
        digit = n % 10
        if digit % 2 == 0
            count += 1
        n = n // 10
    return count
```

      Copy

   2. factorial(n)
      This program is supposed to return n!, i.e. n-factorial, which is defined for all non-negative integers. For example, 3!=3*2*1=6, 4!=4*3*2*1=24, and 5!=5*4*3*2*1=120. (Note that 0 is a special case, and 0!=1.)

```
#Assume n>=0
def factorial(n):
    if n==0:
        return 1
    result=n
    for smallerInteger in range(n):
        result=result*smallerInteger
    return result
```

Copy

3. intHasRepeatedDigits(n)
   This program is supposed to take any integer, positive or negative, and return True if the number has at least one pair of sequentially repeating digits (e.g. 55, 1223, 844, -993) and False otherwise (e.g. 5, 1234, 841, -903), accompanied by an appropriate print() message.

```
def intHasRepeatedDigits(n):
    if n>0:
        remainingDigits=n
    else:
        n=-n
    while remainingDigits//10>0:
        firstDigit=remainingDigits%10
        tensDigit=(remainingDigits//10)%10
        if firstDigit==tensDigit:
            print(n,"has repeated digits!")
            return True
        remainingDigits=remainingDigits//10
    print(n,"has no repeated digits...")
    return False
```

Copy

4. Test Writing: testIsEquidigital() [5pts]
   We say that a number is equidigital if it is positive and has the same number of digits as the number of digits in all of its unique prime factors combined. For example, 10 is equidigital because it has two digits and its prime factors, 2 and 5, have two digits combined. 3 is also equidigital because it has the same number of digits as its only prime factor, which is itself. 66 is not equidigital because its prime factors (2, 3, and 11) have four combined digits and 66 only has two. We want to write the function isEquidigital(n) which returns True if the number n if equidigital and False otherwise.

   We've made several attempts to solve this problem, each included in the starter file. We've given each a number (isEquidigital1, isEquidigital2, isEquidigital3...) to distinguish them. Only one of these implementations is correct according to the specification we've given above. Your task is to write a test function, testIsEquidigital(isEquidigital), which only passes when called on the correct function. On all other functions, it should raise an assertion error. The argument isEquidigital is actually a function; our own test function, testTestIsEquidigital (meta!), provides the different versions of isEquidigital for you to test. You can call the argument isEquidigital as you would a normal function name (we'll go over how this actually works closer to the end of the semester). For example, we would implement our example from before as:

```
def testIsEquidigital(isEquidigital):
    assert(isEquidigital(10) == True)
```

Copy

5. Code Writing: nthHappyPrime [15 pts]
   Write the function nthHappyPrime(n) that finds the nth happy prime. Prime we know already, but what is a happy number? To find out, read the first paragraph on the Wikipedia page. For

our purposes, we can simplify the process of finding a happy number by saying that a cycle which reaches 1 indicates a happy number, while a cycle which reaches 4 indicates a number that is unhappy. To solve this problem, you'll want to use isPrime and write three other functions:

1. sumOfSquaresOfDigits
   Write the function sumOfSquaresOfDigits(n) which takes a non-negative integer and returns the sum of the squares of its digits. For example, 123 would become 1^2 + 2^2 + 3^2 = 1 + 4 + 9 = 14. You must work with the number input directly instead of casting it to a string! Even if the linter allows your code, we will also check for this requirement in AutoLab!

2. isHappyNumber
   Write the function isHappyNumber(n) which takes a possibly-negative integer and returns True if it is happy and False otherwise. Note that all numbers less than 1 are not happy.

3. nthHappyPrime
   A happy prime is a number that is both happy and prime. Write the function nthHappyPrime(n) which takes a non-negative integer and returns the nth happy prime number (where the 0th happy prime number is 7).

6. Code Writing: printNumberTriangle(n) [5 pts]
   Instead of returning a value, in this function you'll be printing out results. Write the function printNumberTriangle that takes one non-negative integer, n, and prints out a number triangle based on n. For example, given the number 4, this function would print out

```
1
21
321
4321
```

   Copy

   Note: the autograder for this problem is very finicky about whitespace; it expects no spaces, and only one newline after each number line. Make sure your output matches what it expects!

7. Collaborative Code Writing: twoPlayerEggsGame() [10 pts]
   This function will allow two players to play the Egg Game. The rules of the game are:
   1. The game begins with 21 eggs.
   2. Player 1 is asked how many eggs she or he would like to remove, either 1, 2, or 3.
   3. The game prints the number of remaining eggs.
   4. Player 2 is asked how many eggs she or he would like to remove, either 1, 2, or 3.
   5. The game prints the new number of remaining eggs.
   6. Steps 2 through 5 are repeated until there are no eggs left.
   7. Whichever player took the last egg loses, and the other player wins!

   On a player's turn, that player MUST remove at least one (and no more than three) eggs!
   Note: As with playGuessingGame() in hw1, we will test this function by simulating user inputs and looking for correctly printed outputs. The autograder is very picky, so make sure your function delivers the input prompts exactly as specified in the homework! You will see that the file deals with inputs and outputs similarly. Here are some example interchanges. Make sure to copy the prompts exactly!

```
Let's play the egg game! There are 21 eggs left.
Player 1, how many eggs will you remove?3
There are 18 eggs left.
Player 2, how many eggs will you remove?3
There are 15 eggs left.
Player 1, how many eggs will you remove?3
There are 12 eggs left.
Player 2, how many eggs will you remove?3
There are 9 eggs left.
```

15-112
Fall 18

Home

Syllabus

Schedule

Staff

Gallery

Piazza

Autolab

OH Queue

```
Player 1, how many eggs will you remove?3
There are 6 eggs left.
Player 2, how many eggs will you remove?3
There are 3 eggs left.
Player 1, how many eggs will you remove?3
Player 1 took the last egg! Player 2 wins!
```

Pay atention to the slight change in the prompt where there is one egg left. Instead of "There are [x] eggs left" the prompt should be "There is 1 egg left" as seen in the following example. Note that it is OK if the last player tries to take more than the remaining number of eggs (as long as the number taken is still 1, 2, or 3) since they will still take the last one:

```
Let's play the egg game! There are 21 eggs left.
Player 1, how many eggs will you remove?3
There are 18 eggs left.
Player 2, how many eggs will you remove?3
There are 15 eggs left.
Player 1, how many eggs will you remove?3
There are 12 eggs left.
Player 2, how many eggs will you remove?3
There are 9 eggs left.
Player 1, how many eggs will you remove?3
There are 6 eggs left.
Player 2, how many eggs will you remove?3
There are 3 eggs left.
Player 1, how many eggs will you remove?2
There is 1 egg left.
Player 2, how many eggs will you remove?3
Player 2 took the last egg! Player 1 wins!
```

If a player does not enter a number that is 1, 2, or 3, print "Each player must take either 1, 2, or 3 eggs!" and prompt that player again as shown in the following example:

```
Let's play the egg game! There are 21 eggs left.
Player 1, how many eggs will you remove?3
There are 18 eggs left.
Player 2, how many eggs will you remove?3
There are 15 eggs left.
Player 1, how many eggs will you remove?3
There are 12 eggs left.
Player 2, how many eggs will you remove?3
There are 9 eggs left.
Player 1, how many eggs will you remove?3
There are 6 eggs left.
Player 2, how many eggs will you remove?1
There are 5 eggs left.
Player 1, how many eggs will you remove?4
Each player must take either 1, 2, or 3 eggs!
Player 1, how many eggs will you remove?1
There are 4 eggs left.
Player 2, how many eggs will you remove?3
There is 1 egg left.
Player 1, how many eggs will you remove?oh nooooo
Each player must take either 1, 2, or 3 eggs!
Player 1, how many eggs will you remove?1
Player 1 took the last egg! Player 2 wins!
```

REALLY IMPORTANT NOTE! You may get an EOF error when testing

twoPlayerEggsGame() and onePlayerEggsGame(), which stands for "End Of File". If you get an EOF error when testing, it probably means that your code is calling input() but the test case has already provided its entire input string! In other words, the test expects the game to be over, but your code is asking for more moves. Check your code carefully to make sure that your game (and code!) ends after the correct number of moves and does not call input() unnecessarily! You can also interactively try your game by calling it prior to the test. Your code should not crash if a player takes enough eggs to cause the remaining number to drop below zero; this just ends the game because the last egg was taken.

8. Collaborative Code Writing: onePlayerEggsGame() [10 pts]
   The rules are the same as in the previous problem: Start with 21 eggs and alternate turns taking 1, 2, or 3 away. If you take the last egg, you lose! Write a new function onePlayerEggsGame() where Player 2's moves are generated by your code. The human player always goes first and the computer player always goes second. In order to get credit for this problem, the computer must win EVERY time! Also note the possibility of EOF errors, as described in the previous problem! Write the helper function eggBot(n) that returns the number of eggs the computer player will take, where n is the number of eggs left at the beginning of the computer's turn. Here is a sample output:

```
Let's play the egg game! There are 21 eggs left.
Player 1, how many eggs will you remove?3
There are 18 eggs left.
The computer takes 1 egg.
There are 17 eggs left.
Player 1, how many eggs will you remove?3
There are 14 eggs left.
The computer takes 1 egg.
There are 13 eggs left.
Player 1, how many eggs will you remove?2
There are 11 eggs left.
The computer takes 2 eggs.
There are 9 eggs left.
Player 1, how many eggs will you remove?3
There are 6 eggs left.
The computer takes 1 egg.
There are 5 eggs left.
Player 1, how many eggs will you remove?1
There are 4 eggs left.
The computer takes 3 eggs.
There is 1 egg left.
Player 1, how many eggs will you remove?1
Player 1 took the last egg! The computer wins!
```

9. Collaborative Code Writing: nthKaprekarNumber [15pts]
   Background: a Kaprekar number is a non-negative integer, the representation of whose square can be split into two possibly-different-length parts (where the right part is not zero) that add up to the original number again. For instance, 45 is a Kaprekar number, because 45**2 = 2025 and 20+25 = 45. You can read more about Kaprekar numbers here. The first several Kaprekar numbers are: 1, 9, 45, 55, 99, 297, 703, 999 , 2223, 2728,...

   With this in mind, write the function nthKaprekarNumber(n) that takes a non-negative int n and returns the nth Kaprekar number. For example, nthKaprekarNumber(0) == 1.

10. Code Writing: nearestKaprekarNumber(n) [15 pts]
    Kaprekar numbers are described in the previous problem. Now, write the function nearestKaprekarNumber(n) that takes an int or float value n and returns the Kaprekar number closest to n, with ties going to smaller value. For example, nearestKaprekarNumber(49) returns 45, and nearestKaprekarNumber(51) returns 55. And since ties go to the smaller number, nearestKaprekarNumber(50) returns 45.

Note: as you probably guessed, this also cannot be solved by counting up from 0, as that will not be efficient enough to get past the autograder. Hint: one way to solve this is to start at n and grow in each direction until you find a Kaprekar number.

11. Bonus/Optional: Fun with generators! [3 pts; 1 pt each]
Note: if you attempt the optional bonus problems, please be sure to place your solutions above the #ignore_rest line, so the autograder can see them and test them.

First, do your own Google search to read about the "yield" statement and so-called "generators" in Python. Then, carefully look at this code, play around with it, and understand it:

```
def oddsGenerator():
    odd = 1
    while True:
        yield odd
        odd += 2


def oddsGeneratorDemo():
    print("Demonstrating use of oddsGenerator()...")
    print("   looping over oddGenerator():")
    for odd in oddsGenerator():
        # This is how generators are typically used
        print(odd)
        if (odd > 20): break
    print("Done!")
    print("   This time, using the next() function:")
    g = oddsGenerator()
    for i in range(11):
        # Explicitly calling the next() function is a less comm
        # way to use a generator, but it still works, of course.
        print(next(g))
    print("Done!")

oddsGeneratorDemo()
```

Copy

- Bonus/Optional: squaresGenerator()
Write the function squaresGenerator(), so that it passes the following tests. Your function must not use globals or other explicit non-locals (don't worry if you don't know what those are), and it must use "yield" properly.

```
def testSquaresGenerator():
    print("Testing squaresGenerator()...")
    # This time, we'll test twice.  First with next(),
    # then with a "for" loop
    g = squaresGenerator()
    assert(next(g) == 1)
    assert(next(g) == 4)
    assert(next(g) == 9)
    assert(next(g) == 16)

    # ok, now with a for loop.
    squares = ""
    for square in squaresGenerator():
        # we'll check the concatenation of the str's,
        # since we cannot use lists on this hw!
        if (squares != ""): squares += ", "
        squares += str(square)
        if (square >= 100): break
    assert(squares == "1, 4, 9, 16, 25, 36, 49, 64, 81, 100
```

15-112
Fall 18

Home

Syllabus

Schedule

Staff

Gallery

Piazza

Autolab

OH Queue

```
        )
    print("Passed!")
```

Copy

- Bonus/Optional: nswGenerator()
  First, read about Newman-Shanks-Williams primes here. (Hint: the recurrence relation is probably the most important part for this task.) We will build a generator for these! Before generating the NSW (Newman-Shanks-Williams) primes, we'll just generate all the values, prime or not, in the NSW series as described in that link. As noted, these values are also listed here.

  With this in mind, write the function nswGenerator(), so that it passes the following tests. Again, your function must not use globals or other explicit non-locals (don't worry if you don't know what those are), and it must use "yield" properly.

```
def testNswGenerator():
    print("Testing nswGenerator()...")
    nswNumbers = ""
    for nswNumber in nswGenerator():
        # we'll check the concatenation of the str's,
        # since we cannot use lists on this hw!
        if (nswNumbers != ""): nswNumbers += ", "
        nswNumbers += str(nswNumber)
        if (nswNumber >= 152139002499): break
    # from: http://oeis.org/A001333
    assert(nswNumbers == "1, 1, 3, 7, 17, 41, 99, 239, 577,
                         "19601, 47321, 114243, 275807, 665
                         "9369319, 22619537, 54608393, 1318
                         "768398401, 1855077841, 4478554083
                         "26102926097, 63018038201, 1521390
        )
    print("Passed!")
```

Copy

- Bonus/Optional: nswPrimesGenerator()
  Now we put it all together and make an nsw primes generator. Practically, we will be limited by the inefficiency of our isPrime (well, fasterIsPrime) function. So we will only test this function out to 63018038201, and not 489133282872437279 or beyond. Even so, be sure not to hardcode the answers, but instead solve this generally.

  With this in mind, write the function nswPrimesGenerator(), so that it passes the following tests. Again, your function must not use globals or other explicit non-locals (don't worry if you don't know what those are), and it must use "yield" properly.

```
def testNswPrimesGenerator():
    print("Testing nswPrimesGenerator()...")
    nswPrimes = ""
    for nswPrime in nswPrimesGenerator():
        # again, we'll check the concatenation of the str's
        # since we cannot use lists on this hw!
        if (nswPrimes != ""): nswPrimes += ", "
        nswPrimes += str(nswPrime)
        if (nswPrime >= 63018038201): break
    # from: http://oeis.org/A088165
    # print nswPrimes
    assert(nswPrimes == "7, 41, 239, 9369319, 63018038201"
                        #"489133282872437279"
        )
    print("Passed!")
```

Copy

15-112
Fall 18

Home

Syllabus

Schedule

Staff

Gallery

Piazza

Autolab

OH Queue