CMPSC 1500 - Midterm #2

2019/4/15

Choose any 2 questions to answer. Submit your answers as Q1.py, Q2.py, etc.

Make sure to submit both the script you write for the answer and its unittest script.

You correctly submitting your scripts in the proper folders and being named correctly according to the questions is part of the grade.

1. Write a **recursive** function that calculates the greatest common divisor for two variables a and b using Euclid’s algorithm, which is as follows:

gcd(a,0) = a

gcd(a,b) = gcd(b, a mod b)

if both a and b are greater than zero.

Your function should be declared as

def gcd(a,b):

return result

Use the test\_gcd.py to test your functions.

1. Create a script that has two functions, one called *has\_any\_duplicates* and another called *num\_duplicates*.

*has\_any\_duplicates* takes a list as an input, and returns true if there are any duplicates in that list. Otherwise is should False.

For instance, [1,2,3,4,5] would return False

[1,1,2,3,4] would return True

[1,1,2,4,4] would return True

*num\_duplicates* returns the number of items have duplicates.

For instance, the list [1,2,2,3,4,5,5,6] would return 2 because there are 2 unique values that have occur 2 or more times.

>> num\_duplicates([1,2,2,3,4,5,5,6])

>> 2

*# x is a list of values***def** has\_any\_duplicates(x):  
 …your code…

return True or False

**def num\_**duplicates(x):  
 …your code…

return result

Use the test\_dup.py to test your functions.

1. Write a **recursive** function that generates Pell numbers up to the argument given.

Pell numbers are generated as:

P(n) = 0 if n==0;

1 if n==1;

Else 2\*P(n-1) + P(n-2)

Your function should be declared as

def pell(n):

return result # this is the Pell number for n

Then have the main portion of the script generate the Pell numbers up to the 50th number in the sequence, and print out the 49th and 50th numbers.

**Bonus**: 5pts

This series is famously used to approximate a fairly common irrational number.

Print out  until you can see it converge. What is the value it converges to?

1. Examine the Vehicles.py files and the classes within.

There are several types of vehicles that are all subclasses of the more generic Vehicle class. For the sake of this exercise, we only separate vehicles by type, name and size.

You need to add some class member functions for the unittests in test\_vehicles.py to run successfully. The initial error messages should guide you to what functions are needed.

You will likely notice that the child classes each call a function called super().\_\_init\_\_()

Recall that the \_\_init\_\_ function is the class constructor that gets called on the creation of each instance of a class. The super().\_\_init\_\_() will call the parent class’s \_\_init\_\_ function with the provided arguments. That is needed here because the \_\_init\_\_ function is declared (overloaded) in the child classes with default parameters that make sense for that particular child class. If a child class doesn’t overload a parent member function, it will automatically inherit and call the parent’s functions. For instance, in the following code

a = Motorcycle(“Suzuki”,2)

a.set\_dim(5,2,3)

will call the set\_dim function in the Vehicle parent/super class because the Motorcycle class did not overload it, but will have inherited it.

1. The Caesar cipher is shifting each letter of the message by a constant number of places with wrapping. The Vigenere cipher is several Caesar ciphers in sequence, where the amount each encrypted letter is shifted is determined by the letters in a repeating keyword. This is a cipher technique that was used by the confederate army in the American Civil War, and regularly cracked because of it’s simplistic nature and the confederate’s reuse of keys.

A keyword letter ‘A’ would shift the plaintext character by zero places, a ‘B’ would shift the plaintext character by 1, a ‘C’ would shift by 2, and so on.

For example, suppose that the plaintext to be encrypted is

ATTACKATDAWN

The person sending the message chooses a keyword and repeats it until it matches the length of the plaintext, for example, the keyword "LEMON":

LEMONLEMONLE

The letter ‘L’ would shift the plaintext ‘A’ by 11 places to become an ‘L’, the letter ‘E’ would shift the plaintext ‘T’ by 4 places to become an ‘X’, and so on as depected below.

Plaintext: ATTACKATDAWN

Key: LEMONLEMONLE

Ciphertext: LXFOPVEFRNHR

The included crypto class needs to have the encrypt and decrypt functions implemented.

For our purposes, we will only deal with uppercase characters and can ignore all spaces and punctuation as above. Use the file test\_crypto.py for unittesting.

Hint: indexing the key with a mod operator gives the same results as repeating it for the entire length of the plaintext.

For instance, in the above example, note that for index 5, plaintext[5]=’K’ and

key[5%len(key)] = key[5%5] = key[0] = ’L’

key[6%len(key)] = key[6%5] = key[1] = ’E’

etc

Use the test\_crypto.py file to unittest your file.

Trivia Bonus: 5 pts

Due to an amusingly coincidental octal-decimal equivalency, computer scientists tend to conflate what two holidays?