Name: AndrewID:

15-112 Final Exam
Summer-1 2012 (Kesden)

1. Functions:
    a. Write a function, *concatString*, that takes, at a minimum, two strings, *string1* and *string2*, concatenates them, and returns the result. If an additional third argument, *case*, is supplied and is equal to "upper", it should return an all upper-case version of the concatenated string. If the optional third argument is supplied and is equal to "lower", it should return an all lowercase version of the string. Please neglect the situation where the *case* parameter is passed in, but it not correct.
    b. Call this function with your choice of two strings, and only two strings, and print the result.
    c. Call this function with your choice of any two strings – but in a way that will ensure that the result is all lower case – and print the result.
2. Generators:
    a. Write a generator that computes powers of two, starting with 2^x, and ending with 2^y where *x* and *y* are parameters to the generator
    b. Instantiate this generator to represent the powers of two from 2^0 through 2^5, inclusive.
    c. Use this generator to print the first 3 powers of two
    d. Use this generator and a *for loop* to print the powers of two from 2^0 through 2^{99}, inclusive.
3. Comprehensions
    a. Given a string, *sentence*, containing a sentence, use **list** comprehension to create a list of the words, *wordList*. You may neglect punctuation, etc.
    b. Given a string, *sentence*, containing a sentence, use **generator** comprehension to create a *wordGenerator*. You may neglect punctuation, etc.
    c. Iterate through your choice of *wordList* or *wordGenerator*
    d. Please identify two significant difference between the capabilities and/or behaviors and/or efficiencies of *wordList* and *wordGenerator*. In other words, can one do but not the other? Or what does one do more efficiently than the other?
4. Co-routines
    a. Please implement a coroutine, *sumNumber*, that begins with an initial value of *initalValue* and then accepts one **non-negative** integer, each time it is invoked. Each time it receives a **non-negative** integer, it should add it to the running sum. Upon receiving a **negative** integer, it should not add it but should, **instead** return the sum.
    b. Please initialize *sumNumber*, as above, for a starting value of *0*, and call it twice times with the inputs of your choice, followed by a fourth time with an input of *-1 (negative one)*. Print the last result.
    c. Please implement a coroutine, *tenTimes* that takes one arguments, *cr*, which is a reference to another co-routine. Given an input of *-1*, tenTimes should invoke the *cr* couroutine with which it was initialized with a value of -1 and return the result, otherwise is should invoke the *cr* coroutine with a value of *10* times whatever it was passed.
    d. Please initialize an instance of *sumNumber*, as above, with an initial value of *0*. Then, please initialize an instance of *tenTimes*, with a reference to the newly initialized instance of

sumNumber. Invoke *tenTimes* twice with the numbers of your choosing, and a third time with the number -1. Print the result of the third call.

5. Lambdas
    a. Define and instantiate a lambda that, given a string, returns the upper-case equivalent of the string.
    b. Use this lambda to define a function called *printUpperString* that takes a string as an argument and prints out an uppercase equivalent.
    c. Define a lamda that, given *x*, computes *x/n*
    d. Define a function, that accepts an integer argument *x*, that uses the lambda use created in part (c) to compute *x/5*.
6. Exceptions
    a. Please define a *CaseNotValid* exception
    b. Consider your solution to question #1, but this time consider the situation where *case* is passed in, but is neither "upper", nor "lower". Rewrite your solution to question #1, such that if should this occurs, it raises the *CaseNotValid* exception you defined above.
    c. Please call your *concatString* function from part (b) with the inputs *string1*, *string2*, and *caseSpec*, such that, should this exception occur, it will print "Case should be 'upper' or 'lower'".
7. Objects
    a. Please define an class of objects, *Person*, with the following properties
        i. It should keep track of each person's *firstName*, *lastName*, and *age*, which should be provided when the class is created.
        ii. It should provide an *incrementAge* method that increments the age by one year.
        iii. It should use a **class variable** to count the *numberOfPeople* in existence
        iv. It should use an *__del__ ()* method to decrement the *numberOfPeople*, as appropriate.
        v. It should have a **classmethod**, *getCount* that return the *numberOfPeople*
        vi. It should override *__str__()* to print out the attributes of a student, such as "Kesden, Gregory, 39"
    b. Please instantiate a new *Person* and assign it to the variable *greg*. The new person should be named "Gregory Kesden" and be age 39.
    c. Please print the total number of *Person*s via the *getCount* method.
8. Problem solving
    a. Please write a function *isPalindrome*, that accepts a single string argument and returns *True*, if and only if, the argument is a palindrome, or *false* otherwise. Recall that a palindrome is spelled the same forwards and backwards.
    b. Please use recursion to implement a function *remainder*, that takes two numbers *x* and *y*, and returns (x % y), through the use of repetitive subtraction.
    c. Please write a function *indexText* that accepts one argument, a **List** of strings. It should return a **Dictionary** that maps from each word to a **List** of the line numbers in which it appears. Feel free to start counting lines with your choice of *0* or *1*.
    d. Please write a function *findWord* that, given a **Dictionary** , dict, such as created by your solution to part (c) above, and a *word*, which is a string, will print out the line numbers within which *word* appears or will print out, "The word does not appear.", as appropriate. You do not need to be overly concerned about the formatting for your output – just get it out.
9. Quickies
    a. What function can be used to convert an integer into a string?
    b. What function can be used to convert a string into an integer?
    c. Please prompt the user to enter a name, read this name from the keyboard, and print it.