SOPH 303 The Digital World

Term 3. January 28-May 3. 2013

Problem Set Help Session

Feb 27, 2013. 2-5pm. Room: LT1

Most recent update: March 3, 2013

Objectives: The purpose of this problem set is to help students develop skills in writing simple Python programs involving conditions, loops, lists, functions, strings, dictionaries, files, and classes, etc.

Note:

1. The problems here are mostly simple and intended for a novice to practice programming in Python. It is highly recommended that you solve these problems using Idle and *without help from your friends*. You may certainly look up any internet site or a book or your notes for questions related to Python.

2. All floating point results given in the sample test cases are correct to two decimal places.

3. Test your programs using the given test cases. Problems in this set are not included in Tutor.

Summary:

Category A: Input/output from keyboard. Total problems: 4

1. Write a program that prompts a user as: "Enter an integer:" and reads an integer from the keyboard typed by a user. It then multiples the integer that is typed by the user by 9 and prints the product.

   Test Cases:

   Test case 1

   Input: 2
   Output: 18

   Test case 2

   Input: -5
   Output: -45

2. Write a program that prompts a user as: "Enter a number:" and reads the number from the keyboard typed by a user. It then divides this number by 9.0 and prints the result to two decimal places.

   Test Cases:

   Test case 1

   Input: 2
   Output: 0.22

   Test case 2

   Input: 25.25
   Output: 2.80

   Test case 3

   Input: -102
   Output: -11.33

3. Write a program that prompts the user to enter two numbers, one at a time; i.e., prompt for the first number and then for the second number. The numbers entered could be of type integer or floating point. Multiply the two numbers and print the result as shown in the test cases below.

   Test Cases:

   Test case 1

   Input: 4 16
   Output: 64

   Test case 2

   Input: -3 12.5
   Output: 37.5

Test case 3

Input: 0 19.3
Output: 0.0

4. Write a program that prompts the user to enter valid Python expression involving numbers and operators. Use the `eval()` function to evaluate the expression and print the result correct to at most 2-decimal places.

   Test Cases:

   Test case 1

   Input: 2*3.71+6
   Output: 13.42

   Test case 2

   Input: 4/(5-2.5*2)
   Output: ZeroDivisionError: float division by zero

   Test case 3

   Input: 9%2<<3
   Output: 8

   Test case 4

   Input: 2+3j*(3-4j)
   Output: (14+9j)

Category B: Conditionals. Total problems: 4

1. Write a program that prompts a user as: "Enter an integer:" and reads an integer from the keyboard typed by a user. If the input integer is greater than 0 then it is multiplied by 9 and the result printed. If the integer input is less than 0, then it is multiplied by -9 and the result printed. If the integer input is 0 then it multiplies the integer that is typed by the user by 9 and prints the product.

   Test Cases:

   Test case 1

   Input: 9
   Output: 81

   Test case 2

   Input: 0
   Output: 0

   Test case 3

   Input: -29.23
   Output: 263.07

2. Write a program that takes three integers as inputs, one at a time, and prints the largest of these.

   Test Cases:

   Test case 1
   Input: 3 9 -10
   Output: 9

   Test case 2
   Input: 0 0 19
   Output: 19

   Test case 3
   Input: -4 -5 -19
   Output: -4

3. Write a program that takes an integer as input by prompting the user. If the integer is less than 0 or greater than 6 then it prints "Incorrect input." If the input is between 0 and 6 (inclusive) then it prints the corresponding day of the week. Assume that 0 corresponds to Monday, 1 to Tuesday and on.

   Test Cases:

   Test case 1
   Input: 6
   Output: Sunday

   Test case 2
   Input: 0
   Output: Monday

   Test case 3
   Input: 7
   Output: Incorrect input

   Test case 4
   Input: 5
   Output: Saturday

4. Let `x`, `y`, and `z` denote three quantities. Write a program that prompts the user to input the values of `x`, `y`, and `z` . The program then checks if each of the following independent conditions is satisfied and prints out `Trie` or `False`.

   Condition 1 (C1): x<y and y>z+1 Condition 2 (C2): x<y and y>z+1 or x>0 Condition 3 (C3): not(x<y) or y>z or z>0

   Test Cases:

Test case 1

Input: x:5, y:6, z:7

Output: C1: False. C2: True. C3: True

Test case 1

Input: x:6, y:7, z:5

Output: C1: True. C2: True. C3: True

Category C: Functions. Total problems: 4

1. Write a function named `minMax()`. This function takes three inputs: `n1`, `n2`, and `m`. If `m`=0 then the function returns the larger of `n1` and `n2`. If `m`=1 then the function returns the smaller of `n1` and `n2`. If `m` is neither 0 nor 1 then the function returns the value `None`.

   Test Cases:

   Test case 1

   Input: 5 12 0

   Output: 12

   Test case 2

   Input: -15 12 1

   Output: -15

   Test case 3

   Input: 5 12 3

   Output: None

2. Write a function named `myEval()` that takes a string and two numbers as input. The string is an expression consisting of variables `x` and `y`. The first of the two values is of `x` and the second of `y`. The function should evaluate the expression for the given values of `x` and `y` and return its value. Use the `eval()` function in Python to evaluate the expression. Note that the string input to the function might involve functions from the `math` library.

   Test Cases:

   Test case 1

   Input: "x*2+y" 3 4

   Output: 10

   Test case 2

   Input: "x<y and x>1" 5 9

   Output: True

   Test case 3

   Input: "sin(x)**2+cos(x)**2" 3.14 0

   Output: 0.99

3. Write a function named `multGeneric()` that takes two inputs `x` and `y`. Each input could be a number or a string. If the inputs are numbers then the function returns their product. If any input is a string then it is converted to a number before multiplication. Assume that conversion from string to a number will not raise an exception.

   Test Cases:

   Test case 1
   Input: 5 12.0
   Output: 60.0

   Test case 2
   Input: "5" 12.0
   Output: 60.0

   Test case 3
   Input: "5" "12"
   Output: 60

4. Write a function named `defaultFunc()` that takes three inputs named `x`, `y`, and `z`. The default values for each of these three inputs are 5, 6, and 7. It should be possible to call `defaultFunc()` with all three arguments or with one or more missing. If an input argument is missing then its default value is used. The function returns the product of `x`, `y`, and `z`.

   Test Cases:

   Test case 1
   Input: `defaultFunc( 3, 4, 5)`
   Output: 60

   Test case 2
   Input: `defaultFunc( 4, y=5)`
   Output: 140

   Test case 3
   Input: `defaultFunc( z=1)`
   Output: 30

Category D: Lists and loops. Total problems: 9

1. Consider the following geometric series `s`.

   $$s = 1 + \frac{2}{3} + \frac{4}{9} + \frac{8}{27} \ldots$$

   Write a function that inputs `n`– the number of terms to be added– and returns the sum of the first `n` terms of `s`. The output given below is correct to 2-decimal places.

Test Cases:

Test case 1

Input: 10
Output: 2.95

Test case 2

Input: 0
Output: 0

2. Let `sPrev` denote the value of `s` obtained by adding the first $k > 0$ terms. Let `sNew` denote the value of `s` obtained by adding the $(k+1)^{th}$ term to `sPrev`. Modify your function in Problem 1 such that the summation stops when $\mid sPrev - sNew \mid < \epsilon$. Your program should get the value of ϵ by prompting the user. The function should return the sum.

Test Cases:

Test case 1

Input: 0.01
Output: 2.984585306741481

Test case 2

Input: 0.001
Output: 2.9986467210308017

Test case 3

Input: 0.00001
Output: 2.999984354714847

3. Write a function named `addEven()` that takes a list of integers as input, adds the elements at even indices and returns the sum. The function returns `None` is the list is empty.

Test Cases:

Test case 1

Input: [4, 2, 5, 9, 12]
Output: 21

Test case 2

Input: [-20]
Output: -20

Test case 3

Input: [5, 9]
Output: 5

Test case 4

Input: []
Output: `None`

4. Write a function named `countZeroOne()` that takes a list containing only 0's and 1's as input. It counts and returns the number of zeros and the number of ones. **Test Cases:**

   Test case 1
   Input: [0, 0, 0, 1, 1]
   Output: (3, 2)

   Test case 2
   Input: [1,0,1,0,1,1]
   Output: (2, 4)

   Test case 3
   Input: [1,1,1,1]
   Output: (0, 4)

   Test case 4
   Input: []
   Output: (0,0)

5. Write a function named `flip()` that takes a list containing 0'a and 1's. The function changes each 0 in the list to a 1 and each 1 in the list to a 0 and returns the new list.

   Test Cases:

   Test case 1
   Input: [1, 1, 1]
   Output: [0, 0, 0]

   Test case 2
   Input: [1, 0, 0, 1]
   Output: [0, 1, 1, 0]

   Test case 3
   Input: []
   Output: []

6. Write a function named `convertDecimal()` that takes a list of 0's and 1's as input. Assume that the first element in the list is the leftmost (most significant bit) of a binary number and the last element in the list is the least significant bit. Convert the binary number into its decimal equivalent and return the value. Assume that the number is positive, i.e., there is no sign bit.

   Test Cases:

   Test case 1
   Input: [1, 0, 1]
   Output: 5

   Test case 2

Input: [0, 1, 1, 0, 0]
Output: 12

Test case 3

Input: [0, 0, 0, 0]
Output: 0

Test case 4

Input: [1 1 1 1]
Output: 15

7. Write a function named `dnaBaseCount()`. This function takes as input a string containing letters A, C, T, and G. The function returns a list containing the counts of each letter in the string. The first element in the list must be the number of A's, the second the number of C's, the third the number of T's and the last the number of G's.

   Test Cases:

   Test case 1

   Input: "AACTTG"
   Output: [2, 1, 2, 1]

   Test case 2

   Input: "A"
   Output: [1, 0, 0, 0]

   Test case 3

   Input: ""
   Output: [0, 0, 0, 0]

8. Write a function named `gcd()` that takes two positive integers as input and returns their greatest common divisor. *Do not use recursion to solve this problem.* If any of the inputs is negative, the function returns `None`.

   Test Cases:

   Test case 1

   Input: 5 15
   Output: 5

   Test case 2

   Input: 3 13
   Output: 1

   Test case 3

   Input: -2 15
   Output: None

9. Write a function named `getData()`. This function prompts the user to input an integer between 1 and 5. If the user enters a number between 1 and 5 (inclusive) then this number is returned, otherwise the function prompts the user again. The function offers three tries to the user. After three unsuccessful tries the function returns `None`.

   Test Cases:

   Test case 1

   Computer: Please enter an integer between 1 and 5:

   User types: 3

   Output: 3 # This is the value returned by the function

   Test case 2

   Computer: Please enter an integer between 1 and 5:

   User types: 7

   Computer: Please enter an integer between 1 and 5:

   User: 4

   Output: 4 # This is the value returned by the function

   Test case 3

   Computer: Please enter an integer between 1 and 5:

   User types: 7

   Computer: Please enter an integer between 1 and 5:

   User: 9 Computer: Please enter an integer between 1 and 5:

   User types: 0

   Output: None # This is the value returned by the function

Category E: Exceptions (try-except). Total problems: 2

1. Write a function named `myEvalTwo()`. This function takes as input an expression as a string and returns its value computed using `eval()`. In case `eval()` raises an exception, the function returns `None`.

   Test Cases:

   Test case 1

   Input: "2*3+4"

   Output: 10

   Test case 2

   Input: "2*3+4/0"

   Output: None

2. Write two functions named `raiseEx()` and `testRaiseEx()`. `raise()` takes a list as input and returns `True` if all elements in the list are greater than 0. If any element in the list is 0 or less than 0 then a`Value Error` exception is raised. The `testRaiseEx()` function calls `raiseEx()` with a list of integers. If `raiseEx()` returns `True` then the message "List is correct." is printed, otherwise the message "List is incorrect." is printed.

   Test Cases:

   Test case 1
   Input: [1, 19, 98]
   Output: List is correct.

   Test case 2
   Input: [1, 19, -4, 98]
   Output: List is incorrect.

   Test case 2
   Input: [1, 0, -4, 98]
   Output: List is incorrect.

Category F: Nested lists. Total problems: 4

1. Write a function named `search()`. This function takes two inputs: a number x and a nested list l. The nested lists contains sublists where each sublist is a list of numbers. Your function should search for x is each sublist and return a list containing `True` or `False`. If x appears in sublist 1 then the first element of the returned list should be `True` else it should be `False`. Similarly, if x appears in sublist 2 then the second element of the returned list should be `True` else it should be `False`

   Test Cases:

   Test case 1
   Input: 4, [[3, 4], [1, 4, -2], [], [-9]]
   Output: [True, True, False, False]

   Test case 2
   Input: 5, [[3, 4, -9], [-8, 4, 17], [16, 2, 5]]
   Output: [False, False, True]

2. Write a function named `listAverage()` that takes a list of lists as input. Each element of a sublist in the list is a number. The function computes the average of the numbers in each sublist and returns the averages as a list. The first element of the returned list contains the average of numbers in the first sublist, the second element is the average of elements in the second sublist and so on. Use `None` as the average for an empty sublist.

   Test Cases:

Input: [[3, 4], [1, 4, -2], [], [-9]]
Output: [3.5, 1.0, None, -9]

3. Write a function named `matrixAdd()`. This function takes two $n \times n$ matrices as input. Each matrix is represented as a list of lists where each element of a list represents a row of the matrix. The function adds the two matrices as input and returns the sum of the two matrices as a list of lists. Here is an example of how a 3×3 matrix can be represented in Python as a nested list by using elements of each row as a list.

$$\begin{vmatrix} 3 & 4 & -9 \\ -8 & 4 & 17 \\ 15 & 2 & 5 \end{vmatrix} = [[3, 4, -9], [-8, 4, 17], [15, 2, 5]]$$

Test Cases:

Test case 1
Input: [[3, 4], [9, 11]], [[-2, 3], [5, 6]]
Output: [[1, 7], [14, 17]]

Test case 2
Input: [[3, 4, -9], [-8, 4, 17], [15, 2, 5]], [[0, 0, 1], [1, 1, 0], [0, 1, 0]]
Output: [[3, 4, -8], [-7, 5, 17], [15, 3, 5]]

4. Write a function named `compare()` that takes two nested lists as input. Each input list consists of sublists that contain numbers and strings. The function must return `tTrue` if the two input lists are identical, else it should return `False`. Try to solve this problem without using the equality operator (==).

Test Cases:

Test case 1
Input: [[1, 2], [”Hello”, 4, ”again”]], [[1, 2], [”Hello”, 4, ”again”]]
Output: True

Test case 2
Input: [[1, 4], [”Hello”, 4, ”again”]], [[1, 2], [”Hello”, 4, ”again”]]
Output: False

Category G: Dictionaries. Total problems: 4

1. Write a function named `createDict()` that prompts the user of key-value pairs as shown in the test cases below. The user indicates the end of input data by typing an asterisk (*) when prompted for a key. The function should return a dictionary containing all the key value pairs input by the user. Assume that each key is a string and the corresponding value is a number.

Test Scenario:

Begin user interaction:

Enter a key: SavingsAccnt
Enter a value: 10000
Enter a key: CheckingAccnt
Enter a value: 3500.85
Enter a key: VisaAccnt
Enter a value: 2349.45
Enter a key: *

End of interaction with the user.

Output: { "SavingsAccnt": 10000, "CheckingAccnt": 3500.85, "VisaAccnt": 2349.45}

2. Write a function named `dictSearch`. This function takes a dictionary as input. Each entry in the dictionary is a pair of country name and the corresponding capital. It then prompts the user for a key. It searches for the key in the dictionary and prints the corresponding value as shown below in the test scenario. If a key does not appear in the dictionary the function prints the message "Sorry, *** is not in my dictionary", where "***" denotes the key entered by the user. The function terminates when the user types an asterisk (*) as a key. It does not return any value.

   Test Scenario:

   Input: {"US": "Washington DC", "UK": "London", "Malaysia": "Kuala Lumpur"}

   Begin user interaction:

   Enter a key: UK
   Capital of UK is: London
   Enter a key: Indonesia
   Capital of Indonesia is: Kuala Lumpur
   Enter a key: Mali
   Sorry, Mali is not in my dictionary.

   End of interaction with the user.

3. Write a function named `updateDict()` that takes two inputs: a dictionary `d` and a list `l`. The list contains keys the entries corresponding to which must be deleted from `d`. If there is a key in `l` that does not exist in `d` then the function simply ignores such a key. The function returns the updated dictionary.

   Test Cases:

   Input: {"London": 129, "Chicago": 4, "Tokyo": 23, "Beijing": 230},
   ["Chicago", "Tokyo", "Frankfurt"]

   Output: { 'London": 129, "Beijing": 230}

4. Write a function named `splitDict()` that takes a dictionary as input and returns two lists one containing all the keys in the dictionary and the other the corresponding values.

   Test Cases:

Test case 1

Input: {1: "Novak", 2: "Federar", 3:" Murray", 4: "Ferrar", 5: "Nadal"}

Output: [1, 2, 3, 4, 5], ["Novak", "Federar", " Murray", "Ferrar", "Nadal"]

Category H: File I/O. Total problems: 4

1. Write a function named `countLines()` that takes the name of a file as input and return the number of lines in that file as its output.

   Test Cases:

   Test case 1

   Input: "courses.dat" # This is the file name; file contents are shown below.
   Design
   Digital World
   Physical World
   System World
   Biology

   Output: 5

   Test case 2

   Input: "capitals.dat" # This is the file name; file contents are shown below.
   Malaysia KualaLumpur
   India NewDelhi
   Indonesia Jakarta

   Output: 3

2. Write a function named `createDict()` that takes a file name as input. The function reads data from the file and creates and returns dictionary. On each line the file contains the name of a country and its capital; the two being separated by one or more spaces. The dictionary must contain the `country:capital` as the key value pairs input from the file. Your function must raise the exception `IOError` if the file does not exist and return an empty dictionary.

   Test Cases:

   Test case 1

   Input: "capitals.dat" # This is the file name; file contents are shown below.
   Malaysia KualaLumpur
   India NewDelhi
   Indonesia Jakarta
   Singapore Singapore

   Output: {"Malaysia": "KualaLumpur", "India": "NewDelhi", "Indonesia": "Jakarta", "Singapore": "Singapore"}

   Test case 2

   Input: "aFile.dat" # This file does not exist.

3. Write a function named `createFile()` that takes two inputs: a list of items that include numbers and strings and the name of a file. The function open the file and writes all elements in the list into the file with only one element on each line. It then closes the file. Assume that the list is not nested. Make sure you close the file before exiting from the function.

   Test Cases:

   Test case 1

   Input: [”Hello”, 2030, “Singapore!”, “6.3E06], list.dat
   Output: # Contents of list.dat
   “Hello”
   2030
   “Singapore!”
   6.3E06

4. Write a function named `readWriteFile()`. This function takes two filenames as input. The first argument is the name of the file from which data is to be read (input file) and the second is the name of the file into which data is to be written (output file). The input file contains exam scores of students organized by cohorts. For each cohort the fist line contains the cohort number. This is followed by several lines each containing the name of a student followed by the score obtained. The line following the last name in a cohort contains only an asterisk (*) indicating the end of data for a cohort. Assume that there are no spaces inside a name. Here is an example of data for cohort 1 that contains three students.

```
1
name1  10
name2  12
name3  8
*
```

   Your program should read the data from the input file and find the average of all scores for each cohort. It should then write the average of each cohort in the output file. Each line in the output file contains the cohort number and the average to 2-decimal places.

   Test Cases:

   Input: “datain.dat” “averages.dat” # Contents of “datain.dat” follow.

```
1
name1  10
name2  12
name3  8
*
2
name4  5
name6  10
*
```

```
3
name7  9
name8  12
name9  10
name10 6
```

Output: # Contents of "averages.dat" follow.

```
1 10.0
2  7.5
3  9.25
```

Category I: Classes. Total problems: 3

1. (a) Define a class named `Parent`. Include an `__init__()` method that takes two inputs: name of the mother and name of father. Include another method named `printNames()` that prints the mother and father's names. (b) Create an instance of the class named `myParents`. Use the `printNames()` method on object `myParents` to print the names of the parents.

   Test Cases:

   Test case 1

   Input: "Elizabeth, "Philip"
   Output: Elizabeth, Philip

   Test case 2

   Input: "Alice", 'Andrew"
   Output: Alice, Andrew

2. Write a class named `Feval`. The class has an `__init__()` method that takes a a function name `f` as input. The class also contains a `__call__()` method that takes `x` and `y` as inputs and returns the value of the function `f(x, y)`.

   Test the `Feval` class by writing a `test()` method outside the class. The `test()` method takes two parameters `x` and `y` as input. It create an object `t` from `Feval` and prints the value of `(t(x,y)`. Thus, the following code should work assuming that the `gcd()` function is available to calculate the GCD of two integers.

   ```
   def test(x,y):
       t=Feval(gcd)
       print t(x, y)
   ```

   Test Cases:

   Test case 1

   Input: 4, 6
   Output: 2

   Test case 2

   Input: 3, 13
   Output: 1

3. Write a program that defines a class named `shape`, and three functions (outside of the class) named `areaCircle()`, `areaSquare()` and `areaTriangle()`. Inside `shape` define an `__init__` function that takes two inputs: the name of a shape as a string and a list of its dimensions, e.g., for a circle the dimension is the radius of the circle and for a triangle it is a list containing the triangle's base and height. Also,define inside the class is a function named `area()` that takes a function as input and returns the value of this function when applied to the dimension. Note that this function is to calculate the area of a given shape.

   The three functions `areaCircle()`, `areaSquare()` and `areaTriangle()` calculate, respectively, the area of a circle, square, and triangle given the corresponding dimensions. Write a `test()` method that tests your class by creating three objects named `circle`, `square`, and `triangle`. In each case the `test` method should print out a suitable message indicating the area of the corresponding object.

   Partial code for the `test()` method is given below.

```
def test():
    circle=shape("circle", [1.0])
    print "For a ", circle.name, " with radius ",\
          circle.d[0], " the area is:", circle.area(areaCircle), "."
```

   Test Cases:

   Test case 1

   Input: "circle", [1.0]
   Output: For a circle with radius 1.0 the area is: 3.14.

   Test case 2

   Input: "square", [2.0]
   Output: For a square with width 2.0 the area is: 4.00.

   Test case 3

   Input: "triangle", [2.0, 3.0]
   Output: For a triangle with height and base 2.0 3.0 the area is: 3.00.

End of Problem Set of Help Session 1