Lab 08: Loops

Make sure you read the PreLab08

Task 1: Factorization

Factorization is the decomposition of a number into a product of other prime numbers (factors), which when multiplied together give the original number. Examples:

- 3,5 are factors of 15, as 3×5=15
- 2,2,5 are factors of 20, as 2x2x5 = 20
- 2,2,5,5 are factors of 100, as 2x2x5x5 = 100

Your task is to implement a function that print out the prime factors for an input number.

One way to to think about how to get these factors is as follows: If we want to factorize 100, then we need to get all possible factors (2,3,4,5,6,7...). Then for each factor f we try to divide 100 by f using integer division, if there is no remainder from the division: div = 100/f, rem = 100%f i.e. rem equals zero, then f is a factor of 100. And then we try to divide by f again, i.e. div = div/f, rem = div%f. If there is no remainder (rem equals zero) then add f again to the factors of 100. We keep going like this until we get a remainder for the division (rem not equal zero). Then we move on to another possible factor f' and try to divide: div = div/f', rem=div/f' and repeat as we did with f.

Let's try this:

- starting with f = 2:
  - div = 100/2 = 50, rem = 100%2 = 0 -> add 2 to the factors of 100
  - div = 50/2 = 25, rem = 50%2 = 0 -> add 2 to the factors of 100
  - div = 25/2 = 12, rem = 25%2 = 1 -> We don't add 2 again because we have a remainder.

Move on to the next factor: - f = 3

- div = 25/3 = 8, rem = 25%3 = 1 -> We don't add 3 because we have a remainder.

Move on to the next factor: - f = 4

- div = 25/4 = 6, rem = 25%4 = 1 -> We don't add 4 because we have a remainder.

Move on to the next factor: - f = 5

- div = 25/5 = 5, rem = 25%5 = 0 -> add 5 to the factors of 100
- div = 5/5 = 1, rem = 5%5 = 0 -> add 5 to the factors of 100

And we will keep going on like this until we run out of factors. So the question now is how many factor
should we try? Let's consider the number 100. should we try factors 2, 3, 4, 5, ... 100? The answer is NO, because we know that we don't need to consider factors greater than div, because we know for sure we will have remainder. I.e. in our example, if we consider: f=6 and now we have div=1, as you can see f>div.

- f = 6

- div = 1/6 = 0, rem = 1%6= 1 -> We don't add 6 because we have a remainder.

And the same will happen with f = 7, f = 8 and so on because f>div.

Your task is to implement the function factorize, which takes an integer as input, and print the prime factors for that number.

```python
def factorize(n):
    #TODO
```

Sample input/outputs:

```bash
>>> factorize(1100)
2
2
5
5
11

>>> factorize(17)
17

>>> factorize(81)
3
3
3
3
```

Save the program as `factorize.py` in your lab08 directory. Make sure that it runs correctly.

**Task 2: Maximum Character Count**

In this task you are required to write a Python function that takes a string as an input parameters. Your function should return the character that appeared the maximum number of times in the input string. Note, the function should return a list of all characters of maximum occurrence with no duplicates.

For example:

- maxCharCount('apollo') should return a list of the characters ['o', 'l'] because each of these characters are repeated 2 times which is the maximum number of occurrences.
- maxCharCount('element') should return a list of the character ['e'] because this character is
def maxCharCount(myString):
    #TODO

Sample Input/Output

>>> print(maxCharCount('apollo'))
['o', 'l']
>>> print(maxCharCount('apple'))
['p']
>>> print(maxCharCount('iphone'))
['i', 'p', 'h', 'o', 'n', 'e']

Save the program as maxCharCount.py in your lab08 directory. Make sure that it runs correctly.

Task 3: Smart Input

In this task you are required to write the function readnumbers. The function takes two parameters n,k. The function reads from the user n numbers, one at a time and keep adding these numbers together, if the sum of the numbers is less than k, then the function should read n numbers again from the user. In other words, the function will keep reading n numbers from the user as long as their sum is less than k and the function only terminates when the sum is greater than or equal to k.

Note: Each batch of n numbers should be summed alone. This means, if a certain batch of n numbers had their sum less than k, the next batch of n numbers should not be be added to the sum of the previous batch. I.e. Each batch should start adding up from zero.

def readNumbers(n, k):
    #TODO

Sample Input/Output:

>>>readNumbers(3, 10)
Please Enter:  3  numbers
Enter a number: 1
Enter a number: 5
Enter a number: 2
Please Enter:  3  numbers
Enter a number: 3
Enter a number: 6
Enter a number: 1
The sum is >=  10

>>>readNumbers(2,5)
Please Enter:  2  numbers
Enter a number: 1
Enter a number: 1
Please Enter:  2  numbers
Enter a number: 2
Enter a number: 2
Please Enter:  2  numbers
Enter a number: 3
Enter a number: 3
The sum is >=  5

Save the program as **readNumbers.py** in your lab08 directory. Make sure that it runs correctly.

**Turnin Instructions**

Run putty and login to **data.cs.purdue.edu**. Turn in your lab by typing:

```
$ cd cs177
$ turnin –v -c cs177=COMMON –p lab08 lab08
```

**Grading Criteria**

<table>
<thead>
<tr>
<th>Task</th>
<th>%</th>
<th>Breakdown</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task 1</td>
<td>40</td>
<td>30% for printing the correct factors (10% for each test case), 10% for successful terminating condition for factors (which should not consider factors up to n)</td>
</tr>
<tr>
<td>Task 2</td>
<td>40</td>
<td>25% for picking the right characters, 5% for returning a list, 10% for not including duplicate characters in the list</td>
</tr>
<tr>
<td>Task 3</td>
<td>20</td>
<td>10% for successfully re-reading inputs when the condition holds, 5% for resetting the sum to zero between batch reading, 5% for printing as expected in the input/output sample.</td>
</tr>
</tbody>
</table>

From: [http://courses.cs.purdue.edu/](http://courses.cs.purdue.edu/) - Computer Science Courses


Last update: 2016/03/01 12:29