Contents

1 Basic Test Results

```
Starting tests...
Wed Nov 23 14:21:15 IST 2016
1985e88ae0c983c71df1fc8d38b52ebaba6ae337  -


Archive:  /tmp/bodek.KDn6AM/intro2cs/ex3/razkarl/presubmission/submission
  inflating: src/ex3.py
  inflating: src/README

Testing README...
Done testing README...

Running presubmit tests...
8 passed tests out of 8
result_code    ex3    8    1
Done running presubmit tests

Tests completed

Additional notes:

There will be additional tests which will not be published in advance.
```

2 README

```
razkarl
311143127
Raz Karl

I discussed the exercise with:

====================================
=  README for ex3: Loops 'N Lists  =
====================================


==================
=  Description:  =
==================
Exercises using loops and lists.

======================
=  Special Comments  =
======================
I had 2 ideas to improve my prime's exercise:
1) Store the prime number calculated each run in a persistent way for re-use
without calculation in future runs (using python shelve for example), I
didn't know if it is permited to create files on the testing server and did
not want to get in trouble in case I caused the tester to call unaothorized
methods, and it also seemed like an overkill, so I didn't implement that...
2) Search for prime numbers smaller than n on the internet (using urllib)
instead of calculating them the way I did. That also seemed like an overkill
and I could not assume the tester has authorized access to the outside www.
```

3 ex3.py

```
#!/usr/bin/env python3
################################################################
# FILE : ex3.py
# WRITER : Raz Karl , razkarl , 311143127
# EXERCISE : intro2cs ex3 2016-2017
# DESCRIPTION: An exercise about loops and lists
################################################################
import math


def create_list():
    """
    Reads multiple inputs from the user.
    Creats a list of all the inputs given until an empty string was recieved.
    Returns a list of all the inputs prior to the empty string.
    """
    inputs_list = []
    user_input = input()
    while user_input != "":
        inputs_list.append(user_input)
        user_input = input()

    return inputs_list


def concat_list(str_list):
    """
    Gets a list of strings.
    Concatenates all the strings to a single string (no spaces or seperators).
    Returns the concatenated list.
    """
    concated_list = ""
    for string in str_list:
        concated_list += string
    return concated_list


def average(num_list):
    """
    Gets a list of numbers.
    Calculates and returns their average as a float.
    """
    if len(num_list) == 0:
        return None
    else:
        sum = 0
        for num in num_list:
            sum += num
        average = float(sum / len(num_list))
        return average


def cyclic(lst1, lst2):
    """
    Checks if 2 lists are a cyclic permutation of each other
    Returns true if they are, otherwise false.
    """

    # Lists with different lengths cannot be cyclic permutaions
```

```
    if len(lst1) != len(lst2):
        return False

    # Two empty lists are a cyclic permutation
    if len(lst1) == 0:
        return True

    # Compare lst2 against all possible cyclic permutations of lst1 until a
    # match is found (or not)
    cycle_found = False
    for i in range(len(lst1)):
        shifted_lst1 = cyclic_shift(lst1, i)
        if (shifted_lst1 == lst2):
            cycle_found = True
            break

    return cycle_found


def cyclic_shift(list, shift):
    """
    Gets a list and an integer (shift)
    moves each item in the list <shift> steps forward, in a cyclic manner.
    """
    return list[-shift:] + list[:-shift]


def histogram(n, num_list):
    """
    Gets a non-negative integer (n) and a list of non negative numbers
    between 0 and n-1 (num_list).
    Returns a list of occurrences of each number in the range where the index
    symbloises the number counted, and the value is the actual count of
    occurrences.
    """
    # Initialize an empty histogram for all the numbers 0-(n-1)
    histogram = [0]*n

    # Count occurences for every number in the list
    for num in num_list:
        histogram[num] += 1

    return histogram


def prime_factors(n):
    """
    Gets an integer (n) greater or equal to 1.
    Returns a list of all the prime factors of that integer (so that
    multiplying all the factors in that list gives back the number. That
    implies repitions are possible).
    """
    # Create a list of all prime numbers lesser than or equal to n (potential
    # candidates for being it's factors)
    prime_candidates = []
    for num in range(2, n+1):
        if is_prime(num):
            prime_candidates.append(num)

    # Check which primes are actually factors of n, add them to the list
    prime_factors = []
    for prime in prime_candidates:
        while n % prime == 0:
            prime_factors.append(prime)
            n = n / prime

    return prime_factors

```

```
128
129 def is_prime(num):
130     """
131     Gets a number and returns wether it's prime or not.
132     """
133     if(num == 2):
134         return True
135
136     # Scan for factors of num between 2 and its root (rounded up)
137     for factor in range(2, math.ceil(math.sqrt(num))):
138         if num % factor == 0:
139             return False
140
141     # No factors at all? Prime!
142     return True
143
144
145 def cartesian(lst1, lst2):
146     """
147     Gets 2 lists - lst1, lst2
148     Returns a new list who's members are all the possible combinations of the
149     form (lst1_item, lst2_item)
150     """
151     cartesian_list = []
152     for lst2_item in lst2:
153         for lst1_item in lst1:
154             cartesian_list.append((lst1_item, lst2_item))
155     return cartesian_list
156
157
158 def pairs(n, num_list):
159     """
160     Gets a number (n) and a list of numbers (num_list)
161     Returns a new list, containing all the possible lists of 2 numbers who's
162     sum is the number n.
163     """
164     sum_n_list = []
165     # For each number in the list, look forward through all the numbers
166     # proceeding it and check if their sum is n.
167     for i in range(len(num_list)):
168         for j in range(len(num_list) - (i+1)):
169             if (num_list[i] + num_list[(i+1) + j] == n):
170                 sum_n_list.append([num_list[i], num_list[i+j+1]])
171     return sum_n_list
```