Contents

1 Basic Test Results

```
Starting tests...
Sat Nov 26 23:24:14 IST 2016
0462e1c61cf04a9813b896a94b42a266d9aa8745  -


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

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There will be additional tests which will not be published in advance.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


===================
= Part D' Answers =
===================
1) cyclic('abcd', 'bcda')
- Returns True, since <class 'str'> just much like <class 'list'> is a
Sequence Type! That means it implements all the common sequence type
operations used within cyclic(), and can identify cyclic similarities between
strings just as it does with lists. Furthermore, it would also work on tuples
and other types, see https://docs.python.org/3/library/stdtypes.html#typesseq
since all the required methods for cyclic() implementation are inherited from
the sequence type.

2) histogram(3, [1,2,3,4])
IndexError: list index out of range
Since the number 4 in the list has no mapping in the list created for the
number 3 that was given.
Python attempts to access a position that is out of range, and exists with
error to prevent a buffer overflow.

3) prime_factors(0)
Returns [], since a list of primes factors was generated in the beggining of
the method but nothing was appended to it (0 is not in the range of the loop
and does not answere the special condition tested outside the loop).
A more interesting question would be prime_factors(4.5), the result would be
[4]. Why? ;-)

4) pairs(2, [0,0,1,1,2,2])
Returns [[0,2], [0,2], [0,2], [0,2], [1,1]]
while it might seem like the pair [0,2] is repeating 4 times, these are 4
different pairs!
[1st 0, 1st 2], [1st 0, 2nd 2], [2nd 0, 1st 2], [2nd 0, 2nd 2].

======================
=  Special Comments  =
======================
I had 2 ideas to improve my prime's exercise, by using a list of prime-only
numbers
between 2 and sqrt(n):
1) Store the prime numbers 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
```

```
60  methods, and it also seemed like an overkill, so I didn't implement that...
61  2) Search for prime numbers smaller than n on the internet (using urllib)
62  instead of calculating them the way I did. That also seemed like an overkill
63  and I could not assume the tester has authorized access to the outside www.
64  Also, on small numbers that would take way more time than just calculating.
```

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)
    for i in range(len(lst1)):
        shifted_lst1 = cyclic_shift(lst1, i)
        if (shifted_lst1 == lst2):
            return True

    # If we got here, no match was found for any cyclic permutation.
    return False


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).
    """
    # Divide n by every number lesser than it's root, as many times as
    # possible.
    # Append every number n is divided by (aka factor) to the list of prime
    # factors.
    # The factors are in fact primes, since we divide n by them as many
    # times as possible so no factor is divisible by any of it's precessors
    # (meaning it is a prime number).
    prime_factors = []
    for factor in range(2, math.ceil(math.sqrt(n))):
        while n % factor == 0:
            prime_factors.append(factor)
            n = n / factor
    # Finally, add what's left of n (either n its self if it was prime, or a
    # large factor greater than n's square root)
    if (n > 1):
        prime_factors.append(int(n))

```

```
128         return prime_factors
129
130
131     def cartesian(lst1, lst2):
132         """
133         Gets 2 lists - lst1, lst2
134         Returns a new list who's members are all the possible combinations of the
135         form (lst1_item, lst2_item)
136         """
137         cartesian_list = []
138         for lst2_item in lst2:
139             for lst1_item in lst1:
140                 cartesian_list.append((lst1_item, lst2_item))
141         return cartesian_list
142
143
144     def pairs(n, num_list):
145         """
146         Gets a number (n) and a list of numbers (num_list)
147         Returns a new list, containing all the possible lists of 2 numbers who's
148         sum is the number n.
149         """
150         sum_n_list = []
151         # For each number in the list, look forward through all the numbers
152         # proceeding it and check if their sum is n.
153         for i in range(len(num_list)):
154             for j in range(len(num_list) - (i+1)):
155                 if (num_list[i] + num_list[(i+1) + j] == n):
156                     sum_n_list.append([num_list[i], num_list[i+j+1]])
157         return sum_n_list
```