```
##*************************************************************
## C.Bruni (instructor)
## CS 116 Fall 2022
## Assignment 05 Problem 1
##*************************************************************


import check


vowels = "aeiou"

def consonant_run_word(s, acc, cur):
  '''
  Determines the consonant run of a word using accumulative recursion;
  cur keeps track of the current run and acc is the largest overall.

  consonant_run_word: Str Nat Nat -> (listof Str)
  '''
  if s == "":
    return max(acc, cur)
  if s[0] not in vowels:
    acc = acc + 1
  else:
    cur = max(cur, acc)
    acc = 0
  return consonant_run_word(s[1:], acc, cur)

def consonant_run_acc(L, acc, cur_max):
  '''
  Returns all strings in L that have the longest consonant run
  using accumulative recursion with acc and cur_max the current best

  consonant_run_acc: (listof Str) (listof Str) Nat -> (listof Str)
  '''
  if L == []:
    return acc
  run = consonant_run_word(L[0], 0, 0)
  if run == cur_max:
    acc.append(L[0])
  if run > cur_max:
    acc = [L[0]]
    cur_max = run
  return consonant_run_acc(L[1:], acc, cur_max)

def max_consonant_run(L):
  '''
  Returns all strings in L that have the longest consonant run

  max_consonant_run: (listof Str) -> (listof Str)
```

This document is for the exclusive use of billy.chow

```
  Examples:
     max_consonant_run([]) => []

max_consonant_run(["watchstrap","monkey","banana","cookies","mmmmmmore
"])
              => ["watchstrap","mmmmmmore"]
     max_consonant_run(["xyz","ccc","civility", "abracadabra"])
              => ["xyz","ccc"]
  '''
  return consonant_run_acc(L, [], 0)

##Examples:
check.expect("Ex 1", max_consonant_run(
  ["watchstrap","monkey","banana","cookies","mmmmmmore"]),
              ["watchstrap","mmmmmmore"])
check.expect("Ex 2", max_consonant_run(
  ["xyz","ccc","civility", "abracadabra"]), ["xyz","ccc"])

##Tests:
check.expect("Test empty", max_consonant_run([]), [])
check.expect("Test empty string", max_consonant_run(['', '', '']),
['', '', ''])
check.expect("Test empty and vowels",
             max_consonant_run(['', 'aeiou']), ['', 'aeiou'])
check.expect("Test single consonant", max_consonant_run(
  ["eaoiueoiau","c","ioeuaoieu", "oeiaueoiau"]), ["c"])
check.expect("Test multiple words different letters",
max_consonant_run(
  ["she","aaacaaa","the", "oeiaueoiau", "act"]), ["she", "the",
"act"])
check.expect("Test long runs", max_consonant_run(
  ["c"*20, "a"*30, "a"*10 + "c"*20 + "a"*10]),
              ["c"*20, "a"*10 + "c"*20 + "a"*10])

##################################################################
## C.Bruni (instructor)
## CS 116 Fall 2022
## Assignment 05 Problem 2
##################################################################

import check
import check


def has_factor(n, div):
  '''
  Returns True if n has a factor between div and mx,
  starting from div and False otherwise
```

This document is for the exclusive use of billy.chow

```
  has_factor: Nat Nat Nat -> Bool
  Requires:
     0 < n
     0 < mx
     2 <= div
  '''
  if div * div > n:
    return False
  elif n % div == 0:
    return True
  else:
    return has_factor(n, div + 1)

def is_prime(n):
  '''
  Returns True if n is prime, False otherwise

  is_prime: Nat -> Bool

  Examples:
     is_prime(11) => True
     is_prime(100) => False
  '''
  if n < 2:
    return False
  else:
    return not(has_factor(n, 2))

##End code from module 5



def is_slime_breaks(num, pos):
  '''
  Returns True if and only if num is slime based on
  slicing the number into parts based on pos.

  is_slime_breaks: Nat Nat -> Bool
  '''
  num = str(num)
  if pos >= len(num):
    return is_prime(int(num))
  return (is_prime(int(num[:pos])) and is_slime(int(num[pos:]))) \
         or is_slime_breaks(num, pos+1)


def is_slime(num):
  '''
  Returns True if and only if num is slime and False otherwise.
  (Slime numbers are concatenations of prime numbers).
```

This document is for the exclusive use of billy.chow

```
  is_slime: Nat -> Bool

  Examples:
     is_slime(0) => False
     is_slime(1) => False
     is_slime(2) => True
     is_slime(91) => False
     is_slime(10137) => True
  '''
  return is_slime_breaks(num, 1)


##Examples:
check.expect("Example 1", is_slime(10137), True)
check.expect("Example 2", is_slime(91), False)
check.expect("Example 3", is_slime(101), True)


##Tests:
check.expect("Test 1 Repeated Prime", is_slime(3737), True)
check.expect("Test 2 Repeated Prime", is_slime(222222222), True)
check.expect("Test 3 Repeated Prime", is_slime(10110110110110101),
True)
check.expect("Test 4 Simple Prime", is_slime(2), True)
check.expect("Test 5 Not slime", is_slime(9936), False)
check.expect("Test 6 Not slime", is_slime(971038), False)
check.expect("Test 7 Slime", is_slime(9710383), True)

check.expect("Test 8 Small case", is_slime(0), False)
check.expect("Test 9 Small case", is_slime(1), False)
check.expect("Test 10 Repeated composite case",
is_slime(99999999999999), False)

check.expect("Test 11 Repeated Prime", is_slime(551551), True)
check.expect("Test 12 Initial Prime", is_slime(243), True)


**********************************************************
## C.Bruni (instructor)
## CS 116 Fall 2022
## Assignment 05 Problem 3
##**********************************************************

import check


def near_palindrome(s, k):
  '''
  Returns True if and only if s is a palindrome after removing
```

This document is for the exclusive use of billy.chow

```
at most k elements from string s.

near_palindrome: Str Nat -> Bool

Examples:
   near_palindrome("abcda", 2) => True
   near_palindrome("abcda", 1) => False
   near_palindrome("", 0) =>  True
'''
if len(s) == 0 or len(s) <= k:
  return True
if k == 0:
  return s == s[::-1]
if s[0] == s[-1]:
  return near_palindrome(s[1:-1], k)
  return near_palindrome(s[1:], k-1) or near_palindrome(s[:-1], k-1)

##Examples:
check.expect("Example 1", near_palindrome("abcda", 2), True)
check.expect("Example 2", near_palindrome("abcda", 1), False)

##Tests:
check.expect("Test 0", near_palindrome("", 0), True)
check.expect("Test 1", near_palindrome("radar", 2), True)
check.expect("Test 2", near_palindrome("abcdefghij"*50, 500), True)
check.expect("Test 3", near_palindrome("abcdefghij"*5, 49), True)
check.expect("Test 4 No", near_palindrome("abcdefghij", 8), False)
check.expect("Test 5 Palindrome", near_palindrome("radar", 0), True)
check.expect("Test 6 bad beginning", near_palindrome("abcddd", 3),
True)
check.expect("Test 7 bad end", near_palindrome("dddbca", 3), True)
check.expect("Test 8 bad middle", near_palindrome("dabdcd", 3), True)
check.expect("Test 9 not enough", near_palindrome("dabdcd", 2), False)
check.expect("Test 10 test large k", near_palindrome("radar", 100),
True)
```

This document is for the exclusive use of billy.chow