```
##************************************************************
## C.Bruni (instructor)
## CS 116 Fall 2022
## Assignment 03 Problem 1
##************************************************************

import check

def is_anagram(s, t):
    '''
    Returns whether or not s can be formed by the anagramming a
    subset of letters of t. This is case sensitive.

    is_anagram: Str Str -> Bool

    Examples:
       is_anagram("meat", "team") => True
       is_anagram("meal", "alma") => False
    '''
    pos = t.find(s[0:1])
    if pos == -1 or len(s) != len(t):
        return False
    if len(s) == 0:
        return True
    return is_anagram(s[1:], t[:pos] + t[pos+1:])
    # or t.replace(s[0],'',1))


## Alternate Solution
# def same_characters(s, t, pos):
#     if pos < len(s):
#         if s.count(s[pos]) != t.count(s[pos]):
#             return False
#         if s.count(s[pos]) == t.count(s[pos]):
#             return same_characters(s, t, pos + 1)
#     return True

# def is_anagram(s, t):
#     if len(s) != len(t):
#         return False
#     return same_characters(s, t, 0)


## Examples:
check.expect("Ex 1", is_anagram("meat", "team"), True)
check.expect("Ex 2", is_anagram("meal", "alma"), False)
```

```
## Tests:
check.expect("Test 1", is_anagram("meal", "teams"), False)
check.expect("Test 2", is_anagram("meal", "team"), False)
check.expect("Test 3", is_anagram("lie", "lei"), True)
check.expect("Test 4 identical", is_anagram("popcorn",
"popcorn"), True)
check.expect("Test 5 anagram", is_anagram("marson", "ransom"),
True)
check.expect("Test 6 fail len", is_anagram("abcde", "abcd"),
False)
check.expect("Test 7 fail end", is_anagram("abc", "abd"), False)
check.expect("Test 8 fail first", is_anagram("dbc", "abc"),
False)
```

```
##*************************************************************
## C.Bruni (instructor)
## CS 116 Fall 2022
## Assignment 03 Problem 2
##*************************************************************

import check

num_prompt = "Enter a number of directions to be entered: "
dir_prompt = "Enter a direction (N, E, W, S): "
final_answer = "The distance from the origin is {0:0.3f} units."

def get_directions(n):
  '''
  Returns a string consisting of the directional responses (N,
  E, W, S) from the user called n times.

  get_directions: Nat -> Str

  Effects: Reads input from keyboard.
           Prints to screen
  '''
  if (n > 0):
    d = input(dir_prompt)
    return d + get_directions(n-1)
  return ""


def distance_from_origin():
  '''
  Returns the distance from the origin from user input of number
  of cardinal directions and the cardinal directions that
  followed as input.

  distance_from_origin: None -> Float

  Effects: Reads input from keyboard.
           Prints to screen

  Examples:
     distance_from_origin() => 0.0

  Assuming the following:
     The prompt "Enter a number of directions to be entered: "
     is displayed.
     Assume the user types '0'.
```

```
    then "The distance from the origin is 0.0 units." Is
    printed

    distance_from_origin() => 2.236066797749979

Assuming the following:
    The prompt "Enter a number of directions to be entered: "
    is displayed.
    Assume the user types '5'.
    The prompt "Enter a direction (N, E, W, S): " is then
    printed 5 times.
    If the given input is 'N','N','S','E','E',
    then "The distance from the origin is 2.236 units." is
    printed
    '''
    n = int(input(num_prompt))
    s = get_directions(n)
    north = s.count('N')
    south = s.count('S')
    east = s.count('E')
    west = s.count('W')
    dist = ((north-south)**2 + (east - west)**2)**0.5
    print(final_answer.format(dist))
    return dist


EPSILON = 0.00001


## Examples:
check.set_input('0')
check.set_print_exact(final_answer.format(0.0))
check.within("Example 5 No direction", distance_from_origin(),
0.0, EPSILON)

check.set_input('5','N','N','S','E','E')
check.set_print_exact(final_answer.format(2.236))
check.within("Example 1", distance_from_origin(),
2.236066797749979, EPSILON)

## Tests:

check.set_input('6','N','N','N','N','N','N')
check.set_print_exact(final_answer.format(6.000))
```

```
check.within("Test 2 all N", distance_from_origin(), 6.000,
EPSILON)

check.set_input('6','N','N','N','S','S','S')
check.set_print_exact(final_answer.format(0.0))
check.within("Test 3 0", distance_from_origin(), 0.0, EPSILON)

check.set_input('1','E')
check.set_print_exact(final_answer.format(1.0))
check.within("Test 4 Single direction", distance_from_origin(),
1.0, EPSILON)

check.set_input('4','N','E','W','S')
check.set_print_exact(final_answer.format(0.000))
check.within("Test 6 Test all dirs", distance_from_origin(),
0.000, EPSILON)
```

```
##*************************************************************
## C.Bruni (instructor)
## CS 116 Fall 2022
## Assignment 03 Problem 3
##*************************************************************

import check

first_row_oo = "mnp"
first_row_ob = "aehklou"
first_row_bo = "iw"
first_row_bb = ""
second_row_oo = "hw"
second_row_ob = "ilp"
second_row_bo = "eno"
second_row_bb = "akmu"
third_row_oo = 'u'
third_row_ob = "klmnop"
third_row_bo = 'w'
third_row_bb = "aehi"

def print_row(s, oo, ob, bo, bb):
  '''
  Prints out the string row according to what row
  we are on as indicated by the 4 parameters

  third_row: Str Str Str Str Str -> None
  Requires:
     All parameters only have letters from "aehiklmnopuw"
     oo ob bo and bb should be constants from above

  Effects: Prints to screen
  '''
  if s == "":
    return print("")
  if s[0] in oo:
    print("oo", end = '')
  elif s[0] in ob:
    print("ob", end = '')
  elif s[0] in bo:
    print("bo", end = '')
  elif s[0] in bb:
    print("bb", end = '')
  if len(s) != 1:
    print(" ", end = '')
  print_row(s[1:], oo, ob, bo, bb)
```

```
def letter_print(s):
  '''
  Prints out a Hawaiian word in braille with dots as 'o' and
  blanks as 'b'with spaces between letters and lines ending in
  newline characters.

  letter_print: Str -> None
  Requires:
    s is nonempty and only has letters from "aehiklmnopuw'"

  Effects: Prints to screen

  Example:
    letter_print("hawaiian") => None
    and the following is printed:

    ob ob bo ob bo bo ob oo
    oo bb oo bb ob ob bb bo
    bb bb bo bb bb bb bb ob

  '''
  print_row(s, first_row_oo, first_row_ob, first_row_bo,
first_row_bb)
  print_row(s, second_row_oo, second_row_ob, second_row_bo,
second_row_bb)
  print_row(s, third_row_oo, third_row_ob, third_row_bo,
third_row_bb)

def hawaiian_braille(s):
  '''
  Prints out a Hawaiian word in braille with dots as 'o' and
  blanks as 'b'with spaces between letters.

  hawaiian_braille: Str -> None
  Requires:
     s is nonempty and only has letters from "aehiklmnopuw"

  Effects: Prints to screen

  Example:
    hawaiian_braille("a") => None
    and the following is printed:

     ob
     bb
```

```
    bb


    hawaiian_braille("hawaiian") => None
    and the following is printed:

    ob ob bo ob bo bo ob oo
    oo bb oo bb ob ob bb bo
    bb bb bo bb bb bb bb ob
'''
s = s.replace("'","")
letter_print(s)

## Examples:
check.set_print_exact("ob","bb","bb")
check.expect("Example", hawaiian_braille("a"), None)

check.set_print_exact("ob ob bo ob bo bo ob oo",
                      "oo bb oo bb ob ob bb bo",
                      "bb bb bo bb bb bb bb ob")
check.expect("Example", hawaiian_braille("hawaiian"), None)


## Tests:
check.set_print_exact("ob","bo","bb")
check.expect("Test single", hawaiian_braille("e"), None)

check.set_print_exact("ob ob ob bo ob ob oo oo ob oo ob bo",
                      "bb bo oo ob bb ob bb bo bo ob bb oo",
                      "bb bb bb bb ob ob ob ob ob ob oo bo")
check.expect("Test all", hawaiian_braille("aehiklmnopuw"), None)

check.set_print_exact("ob ob ob ob","bb bb bb bb","bb bb bb bb")
check.expect("Test repeating and okina",
hawaiian_braille("aa'aa"), None)

check.set_print_exact("oo ob ob bo", "bb bb bb ob", "ob bb oo
bb")
check.expect("Test real", hawaiian_braille("maui"), None)
```