RhysZH tests done 7bdc7a8 a day ago

1 contributor

488 lines (487 sloc) | 17.2 KB

Rhys Morgan - DAT9 - Project 1

Overall Assumption

I assume that the inputs for all functions strictly adhere to the types that each function is able to process (e.g. in the palindrome function the data type entered is an integer). I haven't made any qualifying statements to verify if the input is processable or not.

Challenge 1: Largest Palindrome

A palindromic number reads the same both ways. For example, 1234321 is a palindrome. The largest palindrome made from the product of two two-digit numbers is 9009 = 91 × 99. Find the largest palindrome made from the product of two three-digit numbers. Afterward, write a brief explanation walking through your code's logic in markdown.

In [3]:
```
def is_palindrome(n):
    # store n as a list of characters, and compare to its reverse.
    pal_n = list(str(n))
    if pal_n == list(reversed(pal_n)):
        return True
    else:
        return False

largest = 0
factor1 = 0
factor2 = 0
for i in range(100, 1000):
    for m in range(100, 1000):     #for all numbers 'i' in range,
        b = i*m                    #multiply each 'i' by every number 'm' in range.
        if is_palindrome(b) == True:
        #store 'b' as 'largest' and store its factors only if 'b' > current 'largest'
            if b > largest:
                largest = b
                fact1 = i
                fact2 = m

print("The largest palindromic product of two 3-digit numbers is {0} = {1} x {2}".format(largest,
fact1, fact2))
```

```
The largest palindromic product of two 3-digit numbers is 906609 = 913 x 993
```

1st Question Comments

1. Function "is_palindrome" determines whether a number 'n' is palindromic by:

   i. converting the number to a string, then storing each character in that string as an element in a list 'pal_n'.

   ii. comparing that list to its reverse using the 'reversed' method - if both are equal, 'n' is palindromic and returns True.

1. For all 3-digit numbers (100 to 999), each of these is multiplied by every number from 100 to 999 using nested 'for' loops. Each product iterated is stored in variable 'b'.

1. Each iterated 'b' value is evaluated to be palindromic (by way of the 'is_palindromic' function). If 'b' is palindromic, it is compared to 'largest', which is initially zero. If 'b' is greater than 'largest', 'largest' assumes the value of b, and whichever numbers produced b (its factors, i and m) are stored in 'factor1' and 'factor2'.

1. This carries on throughout the loop, with 'largest' and its factors only being reassigned if 'b' is larger than the previous palindromic number iterated.

1. By the end of the loop, 'largest' contains the value of the largest 'b' value, as well as its two factors, which are printed in the final 'print' statement.

Challenge 2: Summation of Primes

The sum of the primes below 10 is 2 + 3 + 5 + 7 = 17. Find the sum of all the primes below 2,000. Afterward, write a brief explanation walking through your code's logic in markdown.

In [5]:
```
def is_prime(n):
    # return False if n=1, as '1' will return True otherwise
    if n == 1:
        return False
    else:
        #checks for factors in specified range.
        for i in range(2,int(n**0.5)+1):
            if n%i==0:
                return False
        return True

def sum_of_prime(num):
    s = 0
    for m in range(0,num):
        if is_prime(m)== True:
            s = s + m
        else:
            s = s
    print("The sum of all primes below", num, "is", s)


sum_of_prime(2000)
```

```
The sum of all primes below 2000 is 277050
```

2nd Question Comments

1. First, a function 'is_prime' is defined for determining if a number 'n' is prime (unless n = '1', which is designated not prime immediately).

   > This is done by checking if all integers inclusive of 2 up to the square root of 'n' are factors (in that they divide with 0 remainder). If any are factors, the number is not prime (False). If none of these numbers are factors, it is prime (True).

1. We then define another function 'sum_of_prime' which will take a number 'num' and check through all numbers between 1 and 'num' (non-inclusive of num, so 'num+1' is not required), passing each through the 'is_prime' function.

1. If a number is prime, it is added to variable 's'. Once loop is finished, the final statement containing 's' is printed, which is our sum.

1. To answer the question, 'sum_of_prime' is called with argument '2000'.

   > NOTES ON "is_prime()":
   >
   > i. We start checking factors from '2' because '1' is a universal factor for integers (and would return False for every number if included in the check).
   >
   > ii. We only check up to the square root of 'n' because if any number higher than this is a factor, the number will also have another factor less than the square root.
   >
   > iii. We immediately return false if '1' is passed as an argument, as it would return True otherwise and sum_of_prime would return an incorrect sum. Though '1' meets the criteria for being prime, it is not treated as such due to its violation of the Fundamental Theorem of Arithmetic.

Challenge 3: Multiples of 3 and 5

If we list all of the natural numbers below 10 that are multiples of 3 or 5, we get 3, 5, 6, and 9. The sum of these multiples is 23. Find the sum of all the multiples of 3 and 5 below 1,000. Afterward, write a brief explanation walking through your code's logic in markdown.

In [11]:
```
def find_multiples(num):
    total = 0
    #add 'm' to 'total' if it divides by 3 or 5. If not, 'total' remains unchanged
    for m in range(3,num):
        if m%3==0:
            total = total + m
        elif m%5==0:
            total = total + m
        else:
            total = total

    print("The sum of all multiples of three and five below {0} is {1}".format(num, total))

find_multiples(1000)
```

```
The sum of all multiples of three and five below 1000 is 233168
```

3rd Question Comments

1. A function 'find_multiples' is defined with argument 'num'.

1. For all numbers from 3 up to 'num' (non-inclusive), add that number to variable 'total' if it divides by 3 or 5 with 0 remainder. After loop is finished,'total' and 'num' are then printed in statement.

   we start from '3' in 'range' because '3' is the smallest multiple of 3, which makes the function slightly more efficient.

1. Call the function with argument '1000' to solve the question

Challenge 4: String Compressor

Implement a method to perform basic string compression using the counts of repeated characters. (This is called run-length encoding.) For example, the string "aabccccccaaa" would become a2b1c5a3. If the “compressed” string would not become smaller than the original string, your method should return the original string. You can assume the string has only uppercase and lowercase letters (a–z). Specify whether your solution is case sensitive or case insensitive and what you would need to change to make it the other. Afterward, write a brief explanation walking through your code's logic in markdown.

In [15]:
```
def compress_string(letters):
    count = 0
    #convert lower case 'letters' string to list, and assign 1st element to 'compress'.
    letter_list = list(letters.lower())
    compress = [str(letter_list[0])]
    #compare element 'i' to last element in 'compress'. If equal, increase count
    for i in letter_list:
        if i == compress[-1]:
            count += 1
        #append count and 'i' to list if different char is reached
        #also, reset count to 1 to begin count for new char
        else:
            compress.append(str(count))
            compress.append(str(i))
            count = 1
    #once loop has finished, append final count to end of 'compress' list
    compress.append(str(count))

    #print 'compress' or arg depending on 'compress' length
    if len(compress) < len(letters):
        print (''.join(compress))
    else:
        print(letters, ": No point in compressing")

compress_string("aaaAbbBcCCCCAaaD")
```

```
a4b3c5a3d1
```

4th Question Comments

1. A function 'compress_string' is defined with argument 'letters'. A variable 'count' is declared within, of value 0, which will be used to count the number of repeated characters for later.

1. two other variables are declared:

   i. 'letter_list', which stores 'letters' as a list containing each character in 'letters' as an element. all characters are made lowercase before converting to list. This variable could also be a string, though it is easier to add and remove case sensitivity if declared as a list (no need to change variable names in rest of program).

   ii. 'compress', a list where we will store our compressed string. It initially has only one element, which is the first element in 'letter_list'.

1. A loop then iterates through each element 'i' in 'letter_list'. If that element is the same as the last element in 'compress', then 'count' is increased by one. This means that for successive identical characters in 'letters', the count is incremented by 1 for each one.

1. Once an element 'i' is detected that is different from the last element in 'compress', the value of 'count' is appended to 'compress'. Once this is done, 'i' is further appended to 'compress' as its latest element.

1. 'Count' is then re-set to '1' (as the latest element 'n' serves as the first occurence in a new count)

1. Once there are no elements left to iterate through the loop, the latest 'Count' is appended to the end of 'compress'. 'compress' is our completed compressed string.
2. The length of 'compress' is compared to the length of the original argument 'letters'. If compress is shorter, 'compress' is converted to a string using 'join' and printed. If not, a statement with the original argument 'letters' is printed.

NOTE: This solution is case insensitive, as the string argument is converted to lower case in line 3 as part of the function. Therefore, "aabcccccaaa" and "aAbCCcccaAa" will both produce the same result, "a2b1c5a2". To make the solution case-sensitive, remove ".lower()" in line 3 when declaring "letter_list".

BONUS Challenge: FizzBuzz

Write a program that prints all of the numbers from 1 to 100. For multiples of 3, instead of the number, print "Fizz;" for multiples of 5, print "Buzz." For numbers that are multiples of both 3 and 5, print "FizzBuzz." Afterward, write a brief explanation walking through your code's logic in markdown.

In [16]:
```
def fizzbuzz(n):
    #for all numbers 'i' in range, if i fulfils conditions, then print corresponding word.
    #if 'i' fulfils no conditions, print 'i' itself
    for i in range(1,n+1):
        if i%3 == 0 and i%5 == 0:
            print ("FizzBuzz")
        elif i%3 == 0:
            print ("Fizz")
        elif i%5 == 0:
            print ("Buzz")
        else:
            print (i)
fizzbuzz(100)
```

```
1
2
Fizz
4
Buzz
Fizz
7
8
Fizz
Buzz
11
Fizz
13
14
FizzBuzz
16
17
Fizz
19
Buzz
Fizz
22
23
Fizz
Buzz
26
Fizz
28
29
FizzBuzz
31
32
Fizz
34
Buzz
Fizz
37
38
Fizz
Buzz
41
Fizz
43
44
FizzBuzz
46
47
Fizz
49
Buzz
Fizz
52
53
Fizz
Buzz
56
```

```
Fizz
58
59
FizzBuzz
61
62
Fizz
64
Buzz
Fizz
67
68
Fizz
Buzz
71
Fizz
73
74
FizzBuzz
76
77
Fizz
79
Buzz
Fizz
82
83
Fizz
Buzz
86
Fizz
88
89
FizzBuzz
91
92
Fizz
94
Buzz
Fizz
97
98
Fizz
Buzz
```

Bonus Question Comments

1. A function 'fizzbuzz' is defined with argument 'num'.
1. Function iterates through all numbers from 1 to 'num' inclusive, and checks if it divides by 3 and 5 with 0 remainder. If so, it prints 'FizzBuzz'.
1. If a number does not divide by both 3 and 5, it checks if it divides by 3. If so, it prints 'Fizz'
1. Else, if a number does not divide by 3, it checks if it divides by 5. If so, it prints 'Buzz'
1. If none of these conditions are met, the number itself is printed. The end result of the function is a printed list of all numbers up to 'num', with the aforementioned replacements in steps 2-4.
1. To answer the question, 'fizzbuzz' is called with argument of 100.

> NOTE: We check if 'i' is a multiple of both 3 and 5 BEFORE checking for 3 and 5 individually. This is because qualifying 'fizzbuzz' numbers will also fulfil the conditions to print both 'fizz' and 'buzz' if we check for these individual multiples first. This is undesirable, as we only want one printed output per number.