Project 1: Python Coding Exercises

Authors: Joseph Nelson (DC)

The following code challenges are drawn from common exercises used in technical interviews.

Please note that there may be several ways to approach each challenge. If you get stuck, try mapping out your approach in pseudocode first. Finally, while solutions to problems like these may be found online, remember that if you copy/paste code that you can't explain, you'll be missing out on the point of the project. The only way to truly learn a new skill is through practice, trial, and error - we can only help you improve by understanding where you are having trouble.

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 [4]:
```
palin_factors = []
i = 1
for u in range (100,1000):
    for y in range (100,1000):
        if str(u*y) == str(u*y)[::-1]:
            if u*y > i:
                i=u*y
```

```
                palin_factors.append(u)
                palin_factors.append(y)
print('The largest Palindrome number is ' + str(i) + " and is the product of " + str(palin_factors
[-2]) + " and " + str(palin_factors[-1]) + ".")
```

```
The largest Palindrome number is 906609 and is the product of 913 and 993.
```

Step 1: I first set an empty list which will be used in later steps to capture the factors that make up the palindrome product.

Step 2: Since we are only looking at three-digit numbers, I specify the range of factors from 100-999(inclusive).

Step 3: Using an IF function, the condition only works when the product of two numbers reads the same both ways, which is done by setting the string of numbers to be TRULY equal forward and backwards[::-1].

Step 4: Once we identify a palindrome product, only if the product is greater than the previous number, then will the new product be assigned to "i", such that "i" takes the value of the greatest number.

Step 5: After which, the factors are appended to, "palin_factors", so that the last two numbers in the list should give me the factors that make up the largest palindrome product, which in this case is "i".

Step 6: Lastly, the statement prints out the value of "i" and the 2 factors which make up "i", by referencing to the last 2 values in the "palin_factors" list.

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 sumprimes(n):
    primelist = [2]
    j = 2
    if n <= 1:
        print("There are no prime numbers.")
    elif n == 2:
        primelist
        print(primelist)
        print("The sum of all prime numbers up to " + str(n) + " is " + str(n) + ".")
    else:
        for i in range(3,n+1,2):
            factors = []
            for k in range(3,i,2):
                if i%k == 0:
                    factors.append(k)
                    break

            if factors == []:
                primelist.append(i)
                j += i
        print(primelist)
        print("The sum of all prime numbers up to " + str(n) + " is " + str(j) + ".")
```

In [6]:
```
sumprimes(2000)
```

```
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97, 1
01, 103, 107, 109, 113, 127, 131, 137, 139, 149, 151, 157, 163, 167, 173, 179, 181, 191, 193, 197,
199, 211, 223, 227, 229, 233, 239, 241, 251, 257, 263, 269, 271, 277, 281, 283, 293, 307, 311, 313
, 317, 331, 337, 347, 349, 353, 359, 367, 373, 379, 383, 389, 397, 401, 409, 419, 421, 431, 433, 4
39, 443, 449, 457, 461, 463, 467, 479, 487, 491, 499, 503, 509, 521, 523, 541, 547, 557, 563, 569,
571, 577, 587, 593, 599, 601, 607, 613, 617, 619, 631, 641, 643, 647, 653, 659, 661, 673, 677, 683
, 691, 701, 709, 719, 727, 733, 739, 743, 751, 757, 761, 769, 773, 787, 797, 809, 811, 821, 823, 8
27, 829, 839, 853, 857, 859, 863, 877, 881, 883, 887, 907, 911, 919, 929, 937, 941, 947, 953, 967,
971, 977, 983, 991, 997, 1009, 1013, 1019, 1021, 1031, 1033, 1039, 1049, 1051, 1061, 1063, 1069, 1
087, 1091, 1093, 1097, 1103, 1109, 1117, 1123, 1129, 1151, 1153, 1163, 1171, 1181, 1187, 1193, 120
1, 1213, 1217, 1223, 1229, 1231, 1237, 1249, 1259, 1277, 1279, 1283, 1289, 1291, 1297, 1301, 1303,
1307, 1319, 1321, 1327, 1361, 1367, 1373, 1381, 1399, 1409, 1423, 1427, 1429, 1433, 1439, 1447, 14
51, 1453, 1459, 1471, 1481, 1483, 1487, 1489, 1493, 1499, 1511, 1523, 1531, 1543, 1549, 1553, 1559
, 1567, 1571, 1579, 1583, 1597, 1601, 1607, 1609, 1613, 1619, 1621, 1627, 1637, 1657, 1663, 1667,
1669, 1693, 1697, 1699, 1709, 1721, 1723, 1733, 1741, 1747, 1753, 1759, 1777, 1783, 1787, 1789, 18
01, 1811, 1823, 1831, 1847, 1861, 1867, 1871, 1873, 1877, 1879, 1889, 1901, 1907, 1913, 1931, 1933
, 1949, 1951, 1973, 1979, 1987, 1993, 1997, 1999]
The sum of all prime numbers up to 2000 is 277050.
```

I used a function to find the sum of all prime numbers below "n". The function also allows for the input "n" to be flexible and not be limited to the 2000 as stated in the question.

Step 1: As the first prime number is 2, anything below 2 will not have any prime numbers and will thus print "There are no prime numbers".

Step 2: As all prime numbers other than 2 are odd, I carved out an exceptional case for 2 under the elif statement that sum of primes up to 2 is 2.

Step 3: For any "n" stated that is 3 or greater, I stated the range of numbers as starting from 3 up to "n"(inclusive) with the code running for every

other number, skipping all even numbers (to save on run time).

Step 4: Setting "factors" as an empty list at the start, I append any number that is NOT prime to the list. This is derived by appending any number that gives a remainder of zero when divided by any number other than 1 and itself. Therefore, if the "factors" list remains empty after trying to divide, this will indicate that the tested number is prime.

Step 5: If a number is identified as prime, it will then be appended to the list "primelist" which starts at 2 as indicated at the start of the code. For the "factors" list, it will always reset back to an empty set under the ELSE function so that it can validate every number accurately.

Step 6: If a number is identified as prime, it will also be added to the summation of all primes as indicated by "j". Where the += function represents a summation of all identified numbers before that.

Step 7: The function then prints the list of all prime numbers identified and also prints the statement for the sum of all prime numbers below the stated "n".

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 [7]:
```
def sum_multiples(n,j,k):
    sum = 0
    multiples = []
    for i in range (1,n):
        if i % j == 0 or i % k == 0:
            sum += i
            multiples.append(i)
    print("The sum of all multiples of " + str(j) + " and " + str(k) + " below " + str(n) + " is "
+ str(sum) + ".")
    print(multiples)
```

sum_multiples function works on 3 arguments. sum_multiples(n,j,k) where,

n is the range of numbers that you want to find the sum of all multiples (not including n) j is the first multiple that you want to indicate k is the second multiple that you want to indicate

In [8]:
```
sum_multiples(1000,3,5)
```

```
The sum of all multiples of 3 and 5 below 1000 is 233168.
[3, 5, 6, 9, 10, 12, 15, 18, 20, 21, 24, 25, 27, 30, 33, 35, 36, 39, 40, 42, 45, 48, 50, 51, 54, 5
5, 57, 60, 63, 65, 66, 69, 70, 72, 75, 78, 80, 81, 84, 85, 87, 90, 93, 95, 96, 99, 100, 102, 105,
108, 110, 111, 114, 115, 117, 120, 123, 125, 126, 129, 130, 132, 135, 138, 140, 141, 144, 145, 147
, 150, 153, 155, 156, 159, 160, 162, 165, 168, 170, 171, 174, 175, 177, 180, 183, 185, 186, 189, 1
90, 192, 195, 198, 200, 201, 204, 205, 207, 210, 213, 215, 216, 219, 220, 222, 225, 228, 230, 231,
234, 235, 237, 240, 243, 245, 246, 249, 250, 252, 255, 258, 260, 261, 264, 265, 267, 270, 273, 275
, 276, 279, 280, 282, 285, 288, 290, 291, 294, 295, 297, 300, 303, 305, 306, 309, 310, 312, 315, 3
18, 320, 321, 324, 325, 327, 330, 333, 335, 336, 339, 340, 342, 345, 348, 350, 351, 354, 355, 357,
360, 363, 365, 366, 369, 370, 372, 375, 378, 380, 381, 384, 385, 387, 390, 393, 395, 396, 399, 400
, 402, 405, 408, 410, 411, 414, 415, 417, 420, 423, 425, 426, 429, 430, 432, 435, 438, 440, 441, 4
44, 445, 447, 450, 453, 455, 456, 459, 460, 462, 465, 468, 470, 471, 474, 475, 477, 480, 483, 485,
486, 489, 490, 492, 495, 498, 500, 501, 504, 505, 507, 510, 513, 515, 516, 519, 520, 522, 525, 528
, 530, 531, 534, 535, 537, 540, 543, 545, 546, 549, 550, 552, 555, 558, 560, 561, 564, 565, 567, 5
70, 573, 575, 576, 579, 580, 582, 585, 588, 590, 591, 594, 595, 597, 600, 603, 605, 606, 609, 610,
612, 615, 618, 620, 621, 624, 625, 627, 630, 633, 635, 636, 639, 640, 642, 645, 648, 650, 651, 654
, 655, 657, 660, 663, 665, 666, 669, 670, 672, 675, 678, 680, 681, 684, 685, 687, 690, 693, 695, 6
96, 699, 700, 702, 705, 708, 710, 711, 714, 715, 717, 720, 723, 725, 726, 729, 730, 732, 735, 738,
740, 741, 744, 745, 747, 750, 753, 755, 756, 759, 760, 762, 765, 768, 770, 771, 774, 775, 777, 780
, 783, 785, 786, 789, 790, 792, 795, 798, 800, 801, 804, 805, 807, 810, 813, 815, 816, 819, 820, 8
22, 825, 828, 830, 831, 834, 835, 837, 840, 843, 845, 846, 849, 850, 852, 855, 858, 860, 861, 864,
865, 867, 870, 873, 875, 876, 879, 880, 882, 885, 888, 890, 891, 894, 895, 897, 900, 903, 905, 906
, 909, 910, 912, 915, 918, 920, 921, 924, 925, 927, 930, 933, 935, 936, 939, 940, 942, 945, 948, 9
50, 951, 954, 955, 957, 960, 963, 965, 966, 969, 970, 972, 975, 978, 980, 981, 984, 985, 987, 990,
993, 995, 996, 999]
```

I used a function to find the sum of multiples of 2 arguments within a certain range.

Step 1: Set the arguments within the function that I want to use.

Step 2: Set "n" as the range of all numbers that I want to identify below "n".

Step 3: Set "j" and "k" as the arguments that I want to find the multiples of.

Step 4: Set the range using a FOR function between 1 and "n".

Step 5: Use an IF function to indicate that if any number leaves no remainder (modulus with answer 0) when dividing by either "j" OR "k", then it is identified as a multiple.

Step 6: This will then be added to the sum (+=) so that it adds all newly identified multiples to the previous ones that were already summed up.

Step 7: As an added step, I also add the multiples to an empty list so that we can see which multiples were added to the total sum.

Step 8: The print statement just states that multiples of "j" and "k" under "n" and also shows all the multiples that contributed to the overall summation.

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 [9]:
```
def str_com(s,p):
    if p == 0:
        n = s
    elif p == 1:
        n = s.lower()
    else:
        n = s.upper()

    com = n[0]
    count = 1

    for i in range(1,len(n)):
        if(n[i-1] == n[i]):
            count += 1
        else:
            if(count > 1):
                com += str(count)
            com += n[i]
            count = 1

    if(count > 1):
        com += str(count)

    print("The compressed string is " + com + ".")
```

In [10]:
```
str_com("aabccccccaaa",0)
str_com("aabcCCccAaa",0)
str_com("aabccccccaaa",1)
str_com("aabcCCccAaa",1)
str_com("aabccccccaaa",2)
str_com("aabcCCccAaa",2)
```

```
The compressed string is a2bc5a3.
The compressed string is a2bcC2c2Aa2.
The compressed string is a2bc5a3.
The compressed string is a2bc5a3.
The compressed string is A2BC5A3.
The compressed string is A2BC5A3.
```

I used a function to express 2 arguments where the first argument acts as the input string and the second acts as the signal of whether the output is lowercase, uppercase or BOTH.

Step 1: Using p as the second argument, I first convert the input string into all lowercase, uppercase or remain the same with BOTH lowercase and uppercase.

Step 2: Using a string with the first element n[0], this forms the base of the output string for the later steps. I also include a count as a counter for the number of consecutive repeated elements.

Step 3: Since the output string already has the first element, I start my range from n[1], the second element, and run it all the way through the length of the input string.

Step 4: If the first element is the same as the second element, the count of repeated element adds 1. This is repeated for all elements which are identical to the next element, and continues to add to the count of the repeated elements.

Step 5: If the next element is different from the prior element, it first checks if there are repeated elements previously. If there are, it will add on the count number to the output string. If there isn't, it will just add the next element to the output string.

Step 6: One important step here would be to reset the count number to 1 for the next input element that it goes through so that the count of CONSECUTIVE repeated elements is accurate.

Step 7: If the last or last few elements in the input string is repeated, this is where the last step would need to make sure that this is also captured and included into the output string.

Step 8: Print the statement with the final output that is a compressed string of the input.

Extra Challenge 4: Tried an additional problem which I term as a string expander.

In [11]:
```
def str_exp(n):

    exp = n[0]
    factor = ""

    for i in range(1,len(n)):

        if n[i].isdigit() == True:
            factor += n[i]

        else:
            if factor == "":
                exp += n[i]
            else:
                exp = exp + exp[-1]*(int(factor)-1)
                factor = ""
                exp += n[i]


    if n[-1].isdigit() == True:
        exp = exp + exp[-1]*(int(factor)-1)

    print(exp)
```

In [12]:
```
str_exp("abcd")
str_exp("a1b1c1d1")
str_exp("a2bcd4")
str_exp("a11b2c12d")
```

```
abcd
abcd
aabcdddd
aaaaaaaaaaabbccccccccccccd
```

I used a function to expand the input string which was previously compressed. However, as it is only expanding based on the count given, it is not possible to identify which were previously lowercase or uppercase.

The concept is very similar to the string compressor, so I will not repeat most of the steps as described under the string compressor.

Step 1: We need to identify where in the input string which is listed as a digit, using the .isdigit() method.

Step 2: As the the digits are recognised as strings as well, it is important to take case of cases where the count number is greater than 9 (i.e. 2-digit numbers or more).

Step 3: If an element is identified as a digit, it will be added to the factor string where it will be used to multiply against the last element added to the output string. This ensures that the factor is applied to the correct element which was last added to the output string.

Step 4: Of course, if the element is not a digit, it will just be added to the output string.

Step 5: Similarly, it is important that the factor element is RESET so that the next count of any factor is accurate.

Step 6: If the last element is a digit, this would need to be catered for and applied to the output string as well.

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 [13]:
```
def fizzbuzz(n,j,k):
    sum = 0
    multiples = []
    for i in range (1,n+1):
        if i % (j*k) == 0:
            multiples.append("FizzBuzz")
        elif i % j == 0:
            multiples.append("Fizz")
        elif i % k == 0:
            multiples.append("Buzz")
        else:
            multiples.append(i)
    print(multiples)
```

In [14]:
```
fizzbuzz(100,3,5)
```

```
[1, 2, 'Fizz', 4, 'Buzz', 'Fizz', 7, 8, 'Fizz', 'Buzz', 11, 'Fizz', 13, 14, 'FizzBuzz', 16, 17, 'F
```

```
izz', 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, 'B
uzz', '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']
```

I used a function to list down all the identified multiples of 2 arguments within a certain range.

Step 1: Set the arguments within the function that I want to use.

Step 2: Set "n" as the range of all numbers that I want to identify below "n".

Step 3: Set "j" and "k" as the arguments that I want to find the multiples of.

Step 4: Set the range using a FOR function between 1 and "n".

Step 5: Use an IF function to indicate that if any number leaves no remainder (modulus with answer 0) when dividing by the product of "j" OR "k", then it is identified as a multiple and will print out FizzBuzz. This is the first condition that should be fulfilled because it takes precedence over all other conditions.

Step 6: Use an ELIF function to indicate that if any number leaves no remainder (modulus with answer 0) when dividing by the "j", then it is identified as a multiple and will print out Fizz. Use an ELIF function to indicate that if any number leaves no remainder (modulus with answer 0) when dividing by the "k", then it is identified as a multiple and will print out Buzz.