Задача 1.Написать функцию на вход которой подается строка, состоящая из латинских букв. Функция должна вернуть количество гласных букв (a, e, i, o, u) в этой строке. Написать тесты для кода

def count\_vowels(input\_string):

count = 0

for char in input\_string:

if char in "aeiouAEIOU":

count += 1

return count

def test\_count\_vowels():

assert count\_vowels("hello") == 2

assert count\_vowels("world") == 1

assert count\_vowels("aEiOu") == 5

assert count\_vowels("") == 0

assert count\_vowels("xyz") == 0

Задача 2. Написать функцию на вход, которой подается строка. Функция должна вернуть true, если каждый символ в строке встречается только 1 раз, иначе должна вернуть false. Написать тесты для кода

def unique\_characters(input\_string):

return len(set(input\_string)) == len(input\_string)

def test\_unique\_characters():

assert unique\_characters("abcdefg") == True

assert unique\_characters("hello") == False

assert unique\_characters("world") == True

assert unique\_characters("aabbcc") == False

assert unique\_characters("12345") == True

assert unique\_characters("") == True

Задача 3. Написать функцию, которая принимает положительное число и возвращает количество бит равных 1 в этом числе. Написать тесты для кода  
def count\_bits(num):

count = 0

while num:

count += num & 1

num >>= 1

return count

# Тесты

def test\_count\_bits():

assert count\_bits(0) == 0

assert count\_bits(1) == 1

assert count\_bits(2) == 1

assert count\_bits(3) == 2

assert count\_bits(8) == 1

assert count\_bits(15) == 4

assert count\_bits(255) == 8

Задача 4. Написать функцию, которая принимает положительное число. Функция должна вернуть то, сколько раз необходимо перемножать цифры числа или результат перемножения, чтобы получилось число состоящее из одной цифры. Например, для входного числа: · 39 функция должна вернуть 3, так как 39=27 => 27=14 => 14=4 · 4 функция должна вернуть 0, так как число уже состоит из одной цифры · 999 функция должна вернуть 4, так как 999=729 => 729=126 => 126=12 => 12=2. Написать тесты для кода

def magic(n):

if n < 10:

return 0

else:

count = 0

while n >= 10:

product = 1

while n > 0:

product \*= n % 10

n = n // 10

n = product

count += 1

return count

# Тесты

print(magic(39))

print(magic(4))

print(magic(999))

print(magic(12345))

Задача 5. Написать функцию, которая принимает два целочисленных вектора одинаковой длины и возвращает среднеквадратическое отклонение двух векторов. Написать тесты для кода

def mse(pred, true):

if len(pred) != len(true):

raise ValueError("Vectors must have the same length")

sum\_of\_squares = sum((p - t) \*\* 2 for p, t in zip(pred, true))

return (sum\_of\_squares / len(pred)) \*\* 0.5

# Тесты

assert mse([1, 2, 3], [4, 5, 6]) == 3.0

assert mse([0, 0, 0, 0, 0], [1, 1, 1, 1, 1]) == 1.0

assert mse([5, 5, 5], [5, 5, 3]) == 1.0

Задача 6. Написать функцию, принимающая целое положительное число. Функция должна вернуть строку вида “(n1p1)(n2p2)…(nkpk)” представляющая разложение числа на простые множители (если pi == 1, то выводить только ni). Например, для числа 86240 функция должна вернуть “(25)(5)(7\*\*2)(11)”. Написать тесты для кода

def prime\_factorization(n):

result = ""

divisor = 2

while n > 1:

count = 0

while n % divisor == 0:

n //= divisor

count += 1

if count > 0:

result += "(" + str(divisor)

if count > 1:

result += "\*\*" + str(count)

result += ")"

divisor += 1

return result

# Тесты

assert prime\_factorization(86240) == "(2\*\*5)(5)(7\*\*2)(11)"

assert prime\_factorization(30) == "(2)(3)(5)"

assert prime\_factorization(1) == ""

assert prime\_factorization(13) == "(13)"

Задача 7. Написать функцию, принимающая целое число n, задающее количество кубиков. Функция должна определить, можно ли из данного кол-ва кубиков построить пирамиду, то есть можно ли представить число n как 1^2+2^2+3^2+…+k^2. Если можно, то функция должна вернуть k, иначе строку “It is impossible”. Написать тесты для кода

def pyramid(number):

sum = 0

k = 1

while sum < number:

sum += k \*\* 2

if sum > number:

return "It is impossible"

k += 1

return k

# Тесты

print(pyramid(1))

print(pyramid(5))

print(pyramid(12))

print(pyramid(14))

Задача 8. Функция принимает на вход положительное число и определяет является ли оно сбалансированным, т.е. сумма цифр до средних равна сумме цифр после. Средними в случае нечетного числа цифр считать одну цифру, в случае четного - две средних. Написать тесты для кода

def is\_balanced\_number(n):

digits = [int(d) for d in str(n)]

if len(digits) % 2 == 0:

left\_sum = sum(digits[:len(digits)//2])

right\_sum = sum(digits[len(digits)//2:])

else:

left\_sum = sum(digits[:len(digits)//2])

right\_sum = sum(digits[len(digits)//2+1:])

return left\_sum == right\_sum

# Тесты

def test\_is\_balanced\_number():

assert is\_balanced\_number(121) == True

assert is\_balanced\_number(12321) == True

assert is\_balanced\_number(12345) == False

assert is\_balanced\_number(1234567) == True

assert is\_balanced\_number(111222333) == False

test\_is\_balanced\_number()