# **Koda ģenerācija – ChatGPT 3.5**

|  |  |  |
| --- | --- | --- |
| Uzdevuma instrukcija  (angļu valodā) | Iegūtais kods | Koda rezultāti |
| Please generate Python code that defines two integer variables and then does a simple addition operation. | num1 = 10  num2 = 20  # Perform addition  result = num1 + num2  # Print the result  print("The result of addition is:", result) | Šis uzdevums nebija ‘codewars’ vietnē, tāpēc šo kodu var manuāli pārbaudīt. |
| An isogram is a word that has no repeating letters, consecutive or non-consecutive. Implement a function that determines whether a string that contains only letters is an isogram. Assume the empty string is an isogram. Ignore letter case. | def is\_isogram(string):  """  Check if a string is an isogram.  Args:  string (str): The input string to check.  Returns:  bool: True if the string is an isogram, False otherwise.  """  # Convert the string to lowercase to ignore letter case  string = string.lower()    # Create a set to store unique letters  seen = set()    # Iterate through each character in the string  for char in string:  # Check if the character is already in the set  if char in seen:  return False  # Add the character to the set  seen.add(char)    # If no repeating letters found, return True  return True |  |
| Check to see if a string has the same amount of 'x's and 'o's. The method must return a boolean and be case insensitive. The string can contain any char. | def equal\_x\_and\_o(string):  # Count the occurrences of 'x' and 'o' in the string  count\_x = string.lower().count('x')  count\_o = string.lower().count('o')    # Return True if the counts are equal, False otherwise  return count\_x == count\_o  # Example usage:  print(equal\_x\_and\_o("xooxxoX")) # Output: True  print(equal\_x\_and\_o("xooxX")) # Output: False |  |
| A pangram is a sentence that contains every single letter of the alphabet at least once. For example, the sentence "The quick brown fox jumps over the lazy dog" is a pangram, because it uses the letters A-Z at least once (case is irrelevant).  Given a string, detect whether or not it is a pangram. Return True if it is, False if not. Ignore numbers and punctuation | import string  def is\_pangram(s):  alphabet = set(string.ascii\_lowercase) # Set of all lowercase letters    # Iterate through each character in the string  for char in s.lower():  if char.isalpha(): # Check if the character is a letter  alphabet.discard(char) # Remove the letter from the set if found    # If the set is empty, all letters have been found  return not alphabet |  |
| Write a function, persistence, that takes in a positive parameter num and returns its multiplicative persistence, which is the number of times you must multiply the digits in num until you reach a single digit. | def persistence(num):  # Convert the number to a string to access its digits  num\_str = str(num)  # Initialize persistence count  persistence\_count = 0    # Keep looping until the number becomes a single digit  while len(num\_str) > 1:  # Multiply the digits  result = 1  for digit in num\_str:  result \*= int(digit)  # Update the number and increment the persistence count  num\_str = str(result)  persistence\_count += 1    return persistence\_count |  |
| Write a function cakes(), which takes the recipe (object) and the available ingredients (also an object) and returns the maximum number of cakes Pete can bake (integer). For simplicity there are no units for the amounts (e.g. 1 lb of flour or 200 g of sugar are simply 1 or 200). Ingredients that are not present in the objects, can be considered as 0. | def cakes(recipe, available):  # Initialize a list to store the ratios of available ingredients to required ingredients  ratios = []    # Iterate through each ingredient in the recipe  for ingredient, amount in recipe.items():  # If the ingredient is not available, return 0 cakes  if ingredient not in available:  return 0  # Calculate the ratio of available ingredient to required ingredient  ratio = available[ingredient] // amount  # Add the ratio to the list  ratios.append(ratio)    # Return the minimum ratio, which represents the maximum number of cakes that can be made  return min(ratios) |  |
| You probably know the "like" system from Facebook and other pages. People can "like" blog posts, pictures or other items. We want to create the text that should be displayed next to such an item.  Implement the function which takes an array containing the names of people that like an item. It must return the display text as shown in the examples | def likes(names):  if len(names) == 0:  return "no one likes this"  elif len(names) == 1:  return f"{names[0]} likes this"  elif len(names) == 2:  return f"{names[0]} and {names[1]} like this"  elif len(names) == 3:  return f"{names[0]}, {names[1]} and {names[2]} like this"  else:  return f"{names[0]}, {names[1]} and {len(names) - 2} others like this" |  |
| Digital root is the recursive sum of all the digits in a number.  Given n, take the sum of the digits of n. If that value has more than one digit, continue reducing in this way until a single-digit number is produced. The input will be a non-negative integer. | def digital\_root(n):  # Keep looping until n is a single-digit number  while n >= 10:  # Convert n to a string to access its digits  n\_str = str(n)  # Calculate the sum of digits  digit\_sum = sum(int(digit) for digit in n\_str)  # Update n with the sum of digits  n = digit\_sum  return n |  |
| Your task is to sort a given string. Each word in the string will contain a single number. This number is the position the word should have in the result.  Note: Numbers can be from 1 to 9. So 1 will be the first word (not 0).  If the input string is empty, return an empty string. The words in the input String will only contain valid consecutive numbers. | def order(sentence):  if not sentence:  return ""    # Split the sentence into words  words = sentence.split()    # Sort the words based on the number within each word  sorted\_words = sorted(words, key=lambda x: int(''.join(filter(str.isdigit, x))))    # Join the sorted words into a single string  sorted\_sentence = ' '.join(sorted\_words)    return sorted\_sentence |  |
| Given an array of ones and zeroes, convert the equivalent binary value to an integer.  Eg: [0, 0, 0, 1] is treated as 0001 which is the binary representation of 1. | def binary\_array\_to\_number(arr):  # Initialize the result as 0  result = 0    # Iterate through the array in reverse order  for i in range(len(arr)):  # Multiply the digit by its corresponding power of 2 and add it to the result  result += arr[len(arr) - i - 1] \* (2 \*\* i)    return result |  |
| The rgb function is incomplete. Complete it so that passing in RGB decimal values will result in a hexadecimal representation being returned. Valid decimal values for RGB are 0 - 255. Any values that fall out of that range must be rounded to the closest valid value.  Note: Your answer should always be 6 characters long, the shorthand with 3 will not work here. | def rgb(r, g, b):  # Function to clamp a value within the range [0, 255]  def clamp(x):  return max(0, min(x, 255))  # Convert decimal values to hexadecimal representation and ensure each component is 2 characters long  hex\_r = format(clamp(r), '02X')  hex\_g = format(clamp(g), '02X')  hex\_b = format(clamp(b), '02X')  # Concatenate the hexadecimal components  return hex\_r + hex\_g + hex\_b |  |
| Write an algorithm that takes an array and moves all of the zeros to the end, preserving the order of the other elements. | def move\_zeros(arr):  # Initialize a variable to keep track of the count of non-zero elements  non\_zero\_count = 0    # Iterate through the array and move non-zero elements to the front  for num in arr:  if num != 0:  arr[non\_zero\_count] = num  non\_zero\_count += 1    # Fill the remaining positions with zeros  for i in range(non\_zero\_count, len(arr)):  arr[i] = 0    return arr |  |
| The goal of this exercise is to convert a string to a new string where each character in the new string is "(" if that character appears only once in the original string, or ")" if that character appears more than once in the original string. Ignore capitalization when determining if a character is a duplicate. | def duplicate\_encode(word):  # Convert the word to lowercase to ignore capitalization  word = word.lower()    # Initialize a dictionary to store the count of each character  char\_count = {}    # Count the occurrences of each character in the word  for char in word:  char\_count[char] = char\_count.get(char, 0) + 1    # Construct the new string based on the counts  new\_word = ''  for char in word:  if char\_count[char] == 1:  new\_word += '('  else:  new\_word += ')'    return new\_word |  |
| Write a method that takes an array of consecutive (increasing) letters as input and that returns the missing letter in the array.  You will always get an valid array. And it will be always exactly one letter be missing. The length of the array will always be at least 2. The array will always contain letters in only one case. | def find\_missing\_letter(chars):  # Iterate through the array  for i in range(len(chars) - 1):  # Check if the ASCII value of the next character is not consecutive  if ord(chars[i + 1]) - ord(chars[i]) != 1:  # Return the missing letter  return chr(ord(chars[i]) + 1)  # Example:  print(find\_missing\_letter(['a', 'b', 'c', 'd', 'f'])) # Output: 'e' |  |
| Given an n x n array, return the array elements arranged from outermost elements to the middle element, traveling clockwise. | def spiral\_order(matrix):  result = []  if not matrix:  return result  # Define initial boundaries  top, bottom, left, right = 0, len(matrix) - 1, 0, len(matrix[0]) - 1  while top <= bottom and left <= right:  # Traverse top row  for j in range(left, right + 1):  result.append(matrix[top][j])  top += 1  # Traverse rightmost column  for i in range(top, bottom + 1):  result.append(matrix[i][right])  right -= 1  # Check if there is a bottom row  if top <= bottom:  # Traverse bottom row  for j in range(right, left - 1, -1):  result.append(matrix[bottom][j])  bottom -= 1  # Check if there is a leftmost column  if left <= right:  # Traverse leftmost column  for i in range(bottom, top - 1, -1):  result.append(matrix[i][left])  left += 1  return result |  |
| Define a function that takes in two non-negative integers �a and �b and returns the last decimal digit of ��ab. Note that �a and �b may be very large!  For example, the last decimal digit of 9797 is 99, since 97=478296997=4782969. The last decimal digit of (2200)2300(2200)2300, which has over 10921092 decimal digits, is 66. Also, please take 0000 to be 11.  You may assume that the input will always be valid. | def last\_digit(a, b):  if b == 0:  return 1    # Map the last digit of a to the possible last digits of a^b  digit\_to\_power = {  0: [0],  1: [1],  2: [2, 4, 8, 6],  3: [3, 9, 7, 1],  4: [4, 6],  5: [5],  6: [6],  7: [7, 9, 3, 1],  8: [8, 4, 2, 6],  9: [9, 1]  }    # Find the last digit of a  last\_digit\_a = a % 10    # Find the possible last digits of a^b  possible\_last\_digits = digit\_to\_power[last\_digit\_a]    # Find the position of b in the cycle of possible last digits  position\_in\_cycle = (b - 1) % len(possible\_last\_digits)    # Return the last digit of a^b  return possible\_last\_digits[position\_in\_cycle]  # Example usage:  print(last\_digit(3, 3)) # Output: 7  print(last\_digit(4, 2)) # Output: 6  print(last\_digit(7, 8)) # Output: 1  print(last\_digit(0, 0)) # Output: 1 |  |
| Write a function that when given a URL as a string, parses out just the domain name and returns it as a string. | import re  def domain\_name(url):  # Regular expression pattern to match the domain name  pattern = r'(?:http[s]?://)?(?:www\.)?([a-zA-Z0-9-]+)\.'    # Use regular expression to find the domain name in the URL  match = re.search(pattern, url)    # Return the domain name if found, otherwise return an empty string  return match.group(1) if match else '' |  |
| Write two functions that convert a roman numeral to and from an integer value. Multiple roman numeral values will be tested for each function.  Modern Roman numerals are written by expressing each digit separately starting with the left most digit and skipping any digit with a value of zero. In Roman numerals:  1990 is rendered: 1000=M, 900=CM, 90=XC; resulting in MCMXC  2008 is written as 2000=MM, 8=VIII; or MMVIII  1666 uses each Roman symbol in descending order: MDCLXVI.  Input range : 1 <= n < 4000  In this kata 4 should be represented as IV, NOT as IIII (the "watchmaker's four"). | class RomanNumerals:  @staticmethod  def to\_roman(val: int) -> str:  # Define the mapping of integer values to Roman numerals  int\_to\_roman\_map = {  1000: 'M',  900: 'CM',  500: 'D',  400: 'CD',  100: 'C',  90: 'XC',  50: 'L',  40: 'XL',  10: 'X',  9: 'IX',  5: 'V',  4: 'IV',  1: 'I'  }    # Initialize the result string  result = ''    # Iterate through the integer values in descending order  for integer in sorted(int\_to\_roman\_map.keys(), reverse=True):  # Append the corresponding Roman numeral to the result while subtracting the integer value  while val >= integer:  result += int\_to\_roman\_map[integer]  val -= integer    return result |  |
| In this kata we want to convert a string into an integer. The strings simply represent the numbers in words.  Examples:  "one" => 1  "twenty" => 20  "two hundred forty-six" => 246  "seven hundred eighty-three thousand nine hundred and nineteen" => 783919  Additional Notes:  The minimum number is "zero" (inclusively)  The maximum number, which must be supported is 1 million (inclusively)  The "and" in e.g. "one hundred and twenty-four" is optional, in some cases it's present and in others it's not  All tested numbers are valid, you don't need to validate them | def parse\_int(string):  # Define a mapping between words and their numerical values  word\_to\_number = {  'zero': 0,  'one': 1,  'two': 2,  'three': 3,  'four': 4,  'five': 5,  'six': 6,  'seven': 7,  'eight': 8,  'nine': 9,  'ten': 10,  'eleven': 11,  'twelve': 12,  'thirteen': 13,  'fourteen': 14,  'fifteen': 15,  'sixteen': 16,  'seventeen': 17,  'eighteen': 18,  'nineteen': 19,  'twenty': 20,  'thirty': 30,  'forty': 40,  'fifty': 50,  'sixty': 60,  'seventy': 70,  'eighty': 80,  'ninety': 90,  'hundred': 100,  'thousand': 1000,  'million': 1000000  }    # Split the input string into individual words  words = string.replace('-', ' ').split()    # Initialize variables to track the total value and the current value  total = 0  current = 0    # Iterate through the words  for word in words:  # If the word represents a numerical value, add it to the current value  if word in word\_to\_number:  current += word\_to\_number[word]  # If the word represents a multiple of 100, multiply the current value by it  elif word == 'hundred':  current \*= 100  # If the word represents a multiple of 1000 or 1000000, multiply the current value by it and add it to the total  elif word in ['thousand', 'million']:  total += current \* word\_to\_number[word]  current = 0    # Add the remaining current value to the total  total += current    return total |  |
| Your task in order to complete this Kata is to write a function which formats a duration, given as a number of seconds, in a human-friendly way.  The function must accept a non-negative integer. If it is zero, it just returns "now". Otherwise, the duration is expressed as a combination of years, days, hours, minutes and seconds.  It is much easier to understand with an example:  \* For seconds = 62, your function should return  "1 minute and 2 seconds"  \* For seconds = 3662, your function should return  "1 hour, 1 minute and 2 seconds"  For the purpose of this Kata, a year is 365 days and a day is 24 hours.  Note that spaces are important.  Detailed rules  The resulting expression is made of components like 4 seconds, 1 year, etc. In general, a positive integer and one of the valid units of time, separated by a space. The unit of time is used in plural if the integer is greater than 1.  The components are separated by a comma and a space (", "). Except the last component, which is separated by " and ", just like it would be written in English.  A more significant units of time will occur before than a least significant one. Therefore, 1 second and 1 year is not correct, but 1 year and 1 second is.  Different components have different unit of times. So there is not repeated units like in 5 seconds and 1 second.  A component will not appear at all if its value happens to be zero. Hence, 1 minute and 0 seconds is not valid, but it should be just 1 minute.  A unit of time must be used "as much as possible". It means that the function should not return 61 seconds, but 1 minute and 1 second instead. Formally, the duration specified by of a component must not be greater than any valid more significant unit of time. | def format\_duration(seconds):  if seconds == 0:  return "now"    # Define the units of time and their corresponding values in seconds  units = [  ('year', 365 \* 24 \* 60 \* 60),  ('day', 24 \* 60 \* 60),  ('hour', 60 \* 60),  ('minute', 60),  ('second', 1)  ]    # Initialize an empty list to store the formatted components  formatted\_components = []    # Iterate through the units of time  for unit, value in units:  # Calculate the number of occurrences of the current unit in the remaining seconds  count = seconds // value  # If the count is greater than zero, add the formatted component to the list  if count > 0:  formatted\_components.append(f"{count} {unit}" + ('' if count == 1 else 's'))  # Subtract the corresponding amount from the remaining seconds  seconds -= count \* value    # Join the formatted components with commas and 'and' where necessary  formatted\_duration = ', '.join(formatted\_components[:-1])  if formatted\_duration:  formatted\_duration += ' and ' + formatted\_components[-1]  else:  formatted\_duration = formatted\_components[-1]    return formatted\_duration |  |