# **Koda ģenerēšana – ChatGPT 4**

|  |  |  |
| --- | --- | --- |
| Uzdevuma instrukcija  (angļu valodā) | Iegūtais kods | Koda rezultāti |
| Please generate Python code that defines two integer variables and then does a simple addition operation. | # Define two integer variables  a = 3  b = 2  # Perform addition  result = a + b  # Print the result  print("The sum of", a, "and", b, "is:", result) |  |
| 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(word):  """  Determine if the provided word is an isogram.    Args:  word (str): The word to check.  Returns:  bool: True if the word is an isogram, False otherwise.  """  seen = set()  for char in word.lower():  if char in seen:  return False  seen.add(char)  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\_xo\_count(s):  """  Check if the string has equal numbers of 'x's and 'o's.    Args:  s (str): The string to check.  Returns:  bool: True if the number of 'x's and 'o's are equal, False otherwise.  """  s = s.lower() # Convert the string to lowercase to make the check case-insensitive  count\_x = s.count('x')  count\_o = s.count('o')    return count\_x == count\_o |  |
| 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 | def is\_pangram(sentence):  """  Determine if the provided sentence is a pangram.    Args:  sentence (str): The sentence to check.  Returns:  bool: True if the sentence is a pangram, False otherwise.  """  # Convert to lowercase and filter out non-alphabetic characters  filtered\_sentence = [char for char in sentence.lower() if char.isalpha()]    # Use a set to find all unique letters  unique\_letters = set(filtered\_sentence)    # Check if we have all 26 letters of the alphabet  return len(unique\_letters) == 26 |  |
| 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):  """  Calculate the multiplicative persistence of a number.  Args:  num (int): The number to calculate the persistence for.  Returns:  int: The number of steps required to reduce num to a single digit by repeatedly multiplying its digits.  """  if num < 10:  return 0 # If the number is already a single digit, no steps are needed  steps = 0  while num >= 10: # Continue until num is a single digit  # Multiply all digits of the current num  product = 1  while num > 0:  product \*= num % 10  num //= 10  num = product  steps += 1  return steps |  |
| 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):  """  Calculate the maximum number of cakes that can be baked based on the available ingredients.  Args:  recipe (dict): A dictionary containing the ingredients and their quantities needed for one cake.  available (dict): A dictionary containing the ingredients and their quantities currently available.  Returns:  int: The maximum number of cakes that can be baked.  """  max\_cakes = float('inf') # Start with a theoretically infinite maximum  for ingredient, amount\_needed in recipe.items():  if ingredient in available:  # Calculate the maximum number of cakes for this ingredient  max\_cakes\_for\_ingredient = available[ingredient] // amount\_needed  else:  # Ingredient is missing, so no cakes can be made  return 0    # Update the overall maximum cakes if this ingredient is more limiting  if max\_cakes\_for\_ingredient < max\_cakes:  max\_cakes = max\_cakes\_for\_ingredient  return max\_cakes |  |
| 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):  """  Generates a string based on the number of likes from different people.    Args:  names (list): A list of strings containing the names of people who liked an item.  Returns:  str: A descriptive string about who likes the item.  """  n = len(names)  if n == 0:  return "no one likes this"  elif n == 1:  return f"{names[0]} likes this"  elif n == 2:  return f"{names[0]} and {names[1]} like this"  elif n == 3:  return f"{names[0]}, {names[1]} and {names[2]} like this"  else:  return f"{names[0]}, {names[1]} and {n - 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):  """  Calculate the digital root of a non-negative integer.    Args:  n (int): The non-negative integer to calculate the digital root of.  Returns:  int: The digital root of the integer.  """  while n >= 10: # Continue while the number has more than one digit  sum\_digits = 0  while n > 0:  sum\_digits += n % 10  n //= 10  n = sum\_digits # Update n to the sum of its digits  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):  """  Sorts words in a sentence based on the numeric digit found in each word.    Args:  sentence (str): The string containing words with embedded numbers.    Returns:  str: A string with the words sorted according to the embedded numbers.  """  words = sentence.split()  sorted\_words = [''] \* len(words) # Create a list with the same number of empty elements as words    for word in words:  for char in word:  if char.isdigit(): # Check if the character is a digit  # Place the word in the correct position (subtract 1 since lists are zero-indexed)  sorted\_words[int(char) - 1] = word  break # No need to check further characters once the digit is found    # Join the sorted words back into a single string  return ' '.join(sorted\_words) |  |
| 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):  """  Converts a list of binary digits (0s and 1s) to the corresponding decimal integer.    Args:  arr (list): List of integers (0s and 1s) representing a binary number.  Returns:  int: Decimal integer representation of the binary number.  """  result = 0  for digit in arr:  result = (result << 1) | digit # Shift result left by 1 and bitwise OR with the digit  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):  """  Convert RGB decimal values to a hexadecimal string.  Args:  r (int): Red component (0-255)  g (int): Green component (0-255)  b (int): Blue component (0-255)  Returns:  str: Hexadecimal string representation of the RGB color.  """  # Helper function to convert a single color component  def clamp\_and\_convert(x):  # Clamp x between 0 and 255  x = max(0, min(255, x))  # Convert to hexadecimal and return two character format  return f"{x:02X}"  # Convert each component and concatenate  return clamp\_and\_convert(r) + clamp\_and\_convert(g) + clamp\_and\_convert(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):  """  Moves all zeros in the list to the end while preserving the order of non-zero elements.    Args:  arr (list): List of integers which may contain zeros.  Returns:  list: The modified list with all zeros moved to the end.  """  zero\_count = arr.count(0) # Count how many zeros are in the list  # Create a new list without zeros  new\_arr = [x for x in arr if x != 0]  # Append the necessary amount of zeros at the end  new\_arr.extend([0] \* zero\_count)  return new\_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):  """  Encodes a string by replacing each character with '(' if it appears only once  in the original string, and ')' if it appears more than once. The comparison  is case-insensitive.    Args:  word (str): The input string to encode.    Returns:  str: The encoded string.  """  # Normalize the string to lowercase  normalized\_word = word.lower()  # Create a dictionary to count occurrences of each character  char\_count = {}  for char in normalized\_word:  if char in char\_count:  char\_count[char] += 1  else:  char\_count[char] = 1    # Construct the encoded string  encoded = ""  for char in normalized\_word:  if char\_count[char] > 1:  encoded += ")"  else:  encoded += "("    return encoded |  |
| 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):  """  Finds the missing letter in an array of consecutive letters.  Args:  chars (list): A list of consecutive (increasing) letters with exactly one missing.  Returns:  str: The missing letter.  """  # Convert characters to their ASCII values and find the missing one  previous = ord(chars[0])  for char in chars[1:]:  current = ord(char)  if current != previous + 1:  # Return the missing character  return chr(previous + 1)  previous = current  # Return None if no missing character is found (should not happen with valid input)  return None |  |
| Given an n x n array, return the array elements arranged from outermost elements to the middle element, traveling clockwise. | def spiral\_order(matrix):  """  Returns the elements of an n x n array arranged from outermost elements to the middle element, traveling clockwise.    Args:  matrix (list of list of int): An n x n 2D array.  Returns:  list: List of integers representing the spiral order of elements.  """  result = []    while matrix:  # Remove the first row  result.extend(matrix.pop(0))    if matrix and matrix[0]:  # Take the last element of each remaining row  for row in matrix:  result.append(row.pop())    if matrix:  # Remove the last row and add its elements in reverse order  result.extend(matrix.pop()[::-1])    if matrix and matrix[0]:  # Take the first element of each remaining row (going upward)  for row in reversed(matrix):  result.append(row.pop(0))  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):  """  Returns the last digit of a^b.  Args:  a (int): Base of the exponentiation.  b (int): Exponent in the exponentiation.  Returns:  int: Last digit of a^b.  """  if b == 0: # handling the special case where any number to the power of 0 is 1  return 1    # Cycles of last digits for bases 0-9  last\_digit\_cycles = {  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]  }    # Get last digit of a  last\_digit\_of\_a = a % 10  # Get the cycle based on the last digit of a  cycle = last\_digit\_cycles[last\_digit\_of\_a]  cycle\_length = len(cycle)    # Find the index in the cycle  # b % cycle\_length gives index in the cycle, subtract 1 because index is 0-based  index = (b % cycle\_length) - 1    # Handle the case where index is -1 (meaning b is exactly divisible by cycle\_length)  return cycle[index] |  |
| 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 get\_domain\_name(url):  """  Extracts the domain name from a given URL.  Args:  url (str): The URL from which to extract the domain name.  Returns:  str: The extracted domain name.  """  # Regex pattern to find the domain  pattern = r'(?:http[s]?://)?(?:www\.)?([^/:\?]+)'    # Search the URL with the regex pattern  match = re.search(pattern, url)  if match:  domain = match.group(1)  # Split by '.' and remove potential www or other subdomains to get the main domain  domain\_parts = domain.split('.')  if len(domain\_parts) > 2:  # Usually the main domain parts are the last two parts, except known SLDs  return '.'.join(domain\_parts[-2:])  return domain  return "" |  |
| 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"). | def roman\_to\_int(s):  """  Converts a Roman numeral string to an integer.  """  roman\_map = {'I': 1, 'V': 5, 'X': 10, 'L': 50, 'C': 100, 'D': 500, 'M': 1000}  integer\_value = 0  prev\_value = 0    for char in reversed(s):  int\_val = roman\_map[char]  if int\_val >= prev\_value:  integer\_value += int\_val  else:  integer\_value -= int\_val  prev\_value = int\_val    return integer\_value  def int\_to\_roman(num):  """  Converts an integer to a Roman numeral string.  """  roman\_pairs = [  (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")  ]  roman\_result = []    for value, numeral in roman\_pairs:  while num >= value:  roman\_result.append(numeral)  num -= value    return ''.join(roman\_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):  num\_dict = {  '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  }  multipliers = {  'hundred': 100,  'thousand': 1000,  'million': 1000000  }  words = string.replace('-', ' ').replace('and', '').split()  total = 0  current = 0  for word in words:  if word in num\_dict:  current += num\_dict[word]  elif word in multipliers:  current \*= multipliers[word]  if word in ['thousand', 'million']:  total += current  current = 0  return total + current |  |
| 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"  # Conversion factors  minute = 60  hour = 60 \* minute  day = 24 \* hour  year = 365 \* day  # Calculate each component  years = seconds // year  seconds %= year  days = seconds // day  seconds %= day  hours = seconds // hour  seconds %= hour  minutes = seconds // minute  seconds %= minute  # Build components list  components = []  if years: components.append(f"{years} year{'s' if years > 1 else ''}")  if days: components.append(f"{days} day{'s' if days > 1 else ''}")  if hours: components.append(f"{hours} hour{'s' if hours > 1 else ''}")  if minutes: components.append(f"{minutes} minute{'s' if minutes > 1 else ''}")  if seconds: components.append(f"{seconds} second{'s' if seconds > 1 else ''}")  # Format components into a string with commas and "and"  if len(components) == 1:  return components[0]  elif len(components) == 2:  return ' and '.join(components)  else:  return ', '.join(components[:-1]) + ' and ' + components[-1] |  |