**Koda ģenerācija – Google Gemini**

|  |  |  |
| --- | --- | --- |
| 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  num1 = 2  num2 = 3  # Perform addition operation and store the result  sum = num1 + num2  # Print the sum  print(sum) |  |
| 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):  """  This function determines whether a string is an isogram.  Args:  string: The string to check.  Returns:  True if the string is an isogram, False otherwise.  """  # Convert the string to lowercase and remove non-letter characters  clean\_string = "".join(char.lower() for char in string if char.isalpha())  # Check if there are any duplicate characters using a set  return len(set(clean\_string)) == len(clean\_string) |  |
| 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 xo(string):  """  This function checks if a string has the same amount of 'x's and 'o's (case-insensitive).  Args:  string: The string to check.  Returns:  True if the string has the same amount of 'x's and 'o's, False otherwise.  """  # Count occurrences of 'x' and 'o' (case-insensitive)  x\_count = sum(char.lower() == 'x' for char in string)  o\_count = sum(char.lower() == 'o' for char in string)  return x\_count == o\_count |  |
| 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(sentence):  """  This function determines whether a sentence is a pangram.  Args:  sentence: The sentence to check.  Returns:  True if the sentence is a pangram, False otherwise.  """  # Convert the sentence to lowercase and remove non-alphanumeric characters  clean\_sentence = "".join(char.lower() for char in sentence if char.isalnum())  # Check if all letters from a to z are present in the set  alphabet = set(string.ascii\_lowercase)  return alphabet.issubset(set(clean\_sentence)) |  |
| 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):  """  This function calculates the multiplicative persistence of a number.  Args:  num: A positive integer.  Returns:  The number of times the digits in num must be multiplied to reach a single digit.  """  if num < 10:  return 0 # Single digit number has 0 persistence  count = 0  while num >= 10:  product = 1  for digit in str(num):  product \*= int(digit)  num = product  count += 1  return 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):  """  This function calculates the maximum number of cakes Pete can bake based on the recipe and available ingredients.  Args:  recipe: A dictionary containing the ingredients needed for one cake.  available: A dictionary containing the available quantities of each ingredient.  Returns:  The maximum number of cakes Pete can bake (integer).  """  # Find the ingredient with the least available quantity relative to the recipe  min\_ratio = float('inf')  for ingredient, amount in recipe.items():  if ingredient in available:  ratio = available[ingredient] / amount  else:  ratio = 0  min\_ratio = min(min\_ratio, ratio)  # Return the floor of the minimum ratio (maximum number of cakes possible)  return int(min\_ratio) |  |
| 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 who\_liked\_it(names):  """  This function formats the text displayed based on the number of people who liked an item.  Args:  names: A list of names of people who liked the item.  Returns:  The formatted text to be displayed.  """  length = len(names)  if length == 0:  return "no one likes this"  elif length == 1:  return f"{names[0]} likes this"  elif length == 2:  return f"{names[0]} and {names[1]} like this"  else:  return f"{names[0]}, {names[1]} and {length - 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):  """  This function calculates the digital root of a non-negative integer.  Args:  n: A non-negative integer.  Returns:  The digital root of the number.  """  while n >= 10:  sum\_of\_digits = 0  while n > 0:  sum\_of\_digits += n % 10  n //= 10  n = sum\_of\_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 sort\_by\_order(sentence):  """  This function sorts a string containing words with position numbers embedded.  Args:  sentence: The string to be sorted.  Returns:  The sorted string where each word appears in its designated position.  """  if not sentence:  return ""  # Split the sentence into words and extract positions and actual words  words = []  for word in sentence.split():  position, actual\_word = int(word[0]), word[1:]  words.append((position, actual\_word))  # Sort the list of tuples based on the position number  sorted\_words = sorted(words)  # Join the sorted words back into a string  return " ".join([word for \_, word in 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(binary\_array):  """  This function converts an array of ones and zeroes to an integer representing the binary value.  Args:  binary\_array: A list containing ones and zeroes.  Returns:  The integer equivalent of the binary value.  """  # Reverse the array to process bits from left (most significant) to right (least significant)  binary\_array = binary\_array[::-1]  # Initialize the decimal value to 0  decimal\_value = 0  # Iterate through each bit (element) in the reversed array  for i, bit in enumerate(binary\_array):  # Check if the bit is 1 and add the corresponding power of 2 to the decimal value  if bit == 1:  decimal\_value += 2\*\*i  return decimal\_value |  |
| 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):  """  This function converts RGB decimal values to a hexadecimal color representation.  Args:  r: Red value (0-255).  g: Green value (0-255).  b: Blue value (0-255).  Returns:  A string containing the hexadecimal color representation (always 6 characters long).  """  # Clamp RGB values to the valid range (0-255)  r = max(0, min(255, round(r)))  g = max(0, min(255, round(g)))  b = max(0, min(255, round(b)))  # Convert each value to hexadecimal string with zero-padding  hex\_r = hex(r)[2:].upper() # Remove "0x" prefix and convert to uppercase  hex\_g = hex(g)[2:].upper()  hex\_b = hex(b)[2:].upper()  # Ensure each component has two characters (zero-pad if necessary)  hex\_r = hex\_r.zfill(2)  hex\_g = hex\_g.zfill(2)  hex\_b = hex\_b.zfill(2)  # Combine RGB components and return the hexadecimal string  return f"{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\_two\_pointers(arr):  """  This function moves all zeros in an array to the end using two pointers (in-place).  Args:  arr: A list representing the array.  Returns:  The modified list with zeros at the end.  """  # Initialize two pointers: write\_index (to insert non-zero elements) and i (to iterate)  write\_index = 0  for i in range(len(arr)):  # If the current element is not zero, swap it with the element at write\_index  if arr[i] != 0:  arr[write\_index], arr[i] = arr[i], arr[write\_index]  write\_index += 1  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\_encoder(string):  """  This function encodes a string to indicate unique and duplicate characters.  Args:  string: The string to encode.  Returns:  A new string where "(" indicates a unique character and ")" indicates a duplicate.  """  # Convert the string to lowercase for case-insensitive comparison  lower\_string = string.lower()  # Create a dictionary to store character counts  char\_counts = {}  for char in lower\_string:  char\_counts[char] = char\_counts.get(char, 0) + 1  # Build the new string by iterating through the original string  result = ""  for char in string:  if char\_counts[char.lower()] == 1:  result += "("  else:  result += ")"  return result |  |
| 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):  """  This function finds the missing letter in an array of consecutive (increasing) letters.  Args:  chars: A list of consecutive (increasing) letters.  Returns:  The missing letter as a string.  """  # Get the expected difference between consecutive letters (always 1)  expected\_diff = ord(chars[1]) - ord(chars[0])  # Iterate through the array from the second element  for i in range(1, len(chars)):  # Check if the current difference is not equal to the expected difference  if ord(chars[i]) - ord(chars[i-1]) != expected\_diff:  # Calculate the missing character's ASCII code based on the expected difference  missing\_char\_code = ord(chars[i-1]) + expected\_diff  # Convert the ASCII code back to a character  return chr(missing\_char\_code) |  |
| Given an n x n array, return the array elements arranged from outermost elements to the middle element, traveling clockwise. | def spiral\_order(matrix):  """  This function returns the elements of an n x n matrix arranged in a spiral order (clockwise).  Args:  matrix: A list of lists representing the n x n matrix.  Returns:  A list containing the elements of the matrix arranged in a spiral order.  """  result = []  top, bottom, left, right = 0, len(matrix) - 1, 0, len(matrix[0]) - 1  direction = "right"  while top <= bottom and left <= right:  # Move in the current direction until reaching a boundary  if direction == "right":  for col in range(left, right + 1):  result.append(matrix[top][col])  top += 1  elif direction == "bottom":  for row in range(top, bottom + 1):  result.append(matrix[row][right])  right -= 1  elif direction == "left":  for col in range(right, left - 1, -1):  result.append(matrix[bottom][col])  bottom -= 1  elif direction == "top":  for row in range(bottom, top - 1, -1):  result.append(matrix[row][left])  left += 1  # Change direction for the next loop iteration  direction = {  "right": "bottom",  "bottom": "left",  "left": "top",  "top": "right",  }[direction]  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):  """  This function calculates the last decimal digit of a very large power a^b.  Args:  a: A non-negative integer base.  b: A non-negative integer exponent.  Returns:  The last decimal digit of a^b.  """  # Handle special cases (0^0 and anything to the power of 0)  if a == 0 and b == 0:  return 1  elif a == 0:  return 0  # Last digit of a can repeat in a cycle of {1, 0, 5, 6}  last\_digit\_cycle = [1, 0, 5, 6]  a\_last\_digit = a % 10 # Get the last digit of a  # Reduce b's exponent by finding the remainder when divided by the cycle length (4)  b\_exp = b % 4  # Return the last digit based on the reduced exponent and the cycle  return last\_digit\_cycle[b\_exp] |  |
| Write a function that when given a URL as a string, parses out just the domain name and returns it as a string. | import urllib.parse  def get\_domain\_name(url):  """  This function extracts the domain name from a URL string.  Args:  url: The URL string.  Returns:  The domain name as a string, or an empty string if parsing fails.  """  try:  # Parse the URL using urllib.parse  parsed\_url = urllib.parse.urlparse(url)  # Extract the netloc (domain name)  domain\_name = parsed\_url.netloc.lower() # Convert to lowercase  # Split and remove subdomain (if any)  parts = domain\_name.split(".")  return parts[-2] if len(parts) > 1 else domain\_name  except ValueError:  # Handle potential parsing errors (invalid URL format)  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"). | roman\_numeral\_map = {  "I": 1,  "V": 5,  "X": 10,  "L": 50,  "C": 100,  "D": 500,  "M": 1000,  }  def int\_to\_roman(num):  """  This function converts an integer to a Roman numeral string.  Args:  num: An integer between 1 and 3999.  Returns:  A string representing the Roman numeral equivalent of the integer.  """  result = ""  for key, value in sorted(roman\_numeral\_map.items(), reverse=True):  while num >= value:  result += key  num -= value  return result  def roman\_to\_int(roman\_num):  """  This function converts a Roman numeral string to an integer value.  Args:  roman\_num: A string representing a valid Roman numeral.  Returns:  An integer equivalent to the Roman numeral.  """  sum = 0  prev = 0  for char in roman\_num:  current = roman\_numeral\_map[char]  # Handle subtractive notation (IV, IX)  if current <= prev:  sum += current  else:  sum = sum - prev + current  prev = current  return sum | Kodu nācās papildināt: pievienot klasi, kas ietver abas šīs funkcijas: |
| 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 | ones = {  "zero": 0,  "one": 1,  "two": 2,  "three": 3,  "four": 4,  "five": 5,  "six": 6,  "seven": 7,  "eight": 8,  "nine": 9,  }  teens = {  "ten": 10,  "eleven": 11,  "twelve": 12,  "thirteen": 13,  "fourteen": 14,  "fifteen": 15,  "sixteen": 16,  "seventeen": 17,  "eighteen": 18,  "nineteen": 19,  }  tens = {  "twenty": 20,  "thirty": 30,  "forty": 40,  "fifty": 50,  "sixty": 60,  "seventy": 70,  "eighty": 80,  "ninety": 90,  }  hundreds = {  "hundred": 100,  }  thousands = {  "thousand": 1000,  }  millions = {  "million": 1000000,  }  def word\_to\_num(words):  """  This function converts a string in words to an integer.  Args:  words: A string representing a number in words.  Returns:  An integer equivalent to the number in words.  """  words = words.strip().lower().split() # Preprocess the string  result = 0  i = 0  while i < len(words):  if words[i] in millions:  result += millions[words[i]] \* int\_from\_group(words[i + 1:])  i += 2  elif words[i] in thousands:  result += thousands[words[i]] \* int\_from\_group(words[i + 1:])  i += 2  elif words[i] in hundreds:  result += hundreds[words[i]] \* int\_from\_group(words[i + 1:])  i += 2  elif i + 1 < len(words) and words[i + 1] in tens: # Handle tens (twenty, thirty, etc.)  result += tens[words[i + 1]]  if words[i] != "zero": # Handle "twenty one" vs. "one"  result += int\_from\_group(words[i:])  i += 2  elif i + 1 < len(words) and words[i + 1] in teens: # Handle teens (eleven, twelve, etc.)  result += teens[words[i + 1]]  i += 2  elif words[i] in ones:  result += ones[words[i]]  i += 1  else:  raise ValueError(f"Invalid word: {words[i]}") # Handle unexpected word  return result  def int\_from\_group(group):  """  This function processes a group of words (e.g., "one hundred") and returns the corresponding integer.  """  if len(group) == 0:  return 0  if len(group) == 1:  return ones[group[0]]  # Handle optional "and" between words  if group[1] == "and":  return ones[group[0]] + int\_from\_group(group[2:])  else:  return int\_from\_group(group[1:]) |  |
| 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):  """  This function formats a duration (given as seconds) in a human-friendly way.  Args:  seconds: A non-negative integer representing the duration in seconds.  Returns:  A string containing the human-readable duration.  """  units = [("year", 365 \* 24 \* 60 \* 60), ("day", 24 \* 60 \* 60), ("hour", 60 \* 60), ("minute", 60), ("second", 1)]  results = []  for unit, value in units:  if seconds >= value:  count = seconds // value  seconds -= count \* value  results.append(f"{count} {unit + ('s' if count > 1 else '')}")  return "now" if not results else ", ".join(results[:-1]) + " and " + results[-1] |  |