# **Koda ģenerēšana – Meta AI**

|  |  |  |
| --- | --- | --- |
| Instruction | Output | Koda rezultāti |
| Please generate Python code that defines two integer variables and then does a simple addition operation. | # Define two integer variables  num1 = 5  num2 = 3  # Perform addition operation  sum = num1 + num2  # Print the result  print("The sum is:", 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):  """  Determine if a string is an isogram (has no repeating letters).  Args:  string (str): The input string.  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 from the string, which automatically removes duplicates  unique\_letters = set(string)  # Check if the length of the set is equal to the length of the original string  return len(unique\_letters) == len(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 has\_equal\_x\_and\_o(string):  """  Check if a string has the same amount of 'x's and 'o's.  Args:  string (str): The input string.  Returns:  bool: True if the string has equal 'x's and 'o's, False otherwise.  """  # Convert the string to lowercase to ignore case  string = string.lower()  # Count the number of 'x's and 'o's  x\_count = string.count('x')  o\_count = string.count('o')  # Return True if the counts are equal, False otherwise  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 | def is\_pangram(string):  """  Detect whether a string is a pangram (contains every letter of the alphabet at least once).  Args:  string (str): The input string.  Returns:  bool: True if the string is a pangram, False otherwise.  """  # Convert the string to lowercase and remove non-alphabetic characters  alphabet = "abcdefghijklmnopqrstuvwxyz"  string = "".join(filter(str.isalpha, string.lower()))  # Check if every letter of the alphabet is in the string  return set(alphabet) <= set(string) |  |
| 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): A positive integer.  Returns:  int: The multiplicative persistence of the number.  """  # Convert the number to a string to easily access its digits  num\_str = str(num)  # Initialize the persistence counter  persistence\_count = 0  # Loop until the number is reduced to a single digit  while len(num\_str) > 1:  # Initialize the product of the digits  product = 1  # Calculate the product of the digits  for digit in num\_str:  product \*= int(digit)  # Update the number string and increment the persistence counter  num\_str = str(product)  persistence\_count += 1  # Return the persistence count  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):  """  Calculate the maximum number of cakes that can be baked.  Args:  recipe (dict): A dictionary containing the recipe (e.g. {"flour": 500, "sugar": 200, "eggs": 1}).  available (dict): A dictionary containing the available ingredients (e.g. {"flour": 1200, "sugar": 1200, "eggs": 5, "milk": 3000}).  Returns:  int: The maximum number of cakes that can be baked.  """  # Initialize the maximum number of cakes to infinity  max\_cakes = float('inf')  # Iterate over the ingredients in the recipe  for ingredient, amount in recipe.items():  # Get the available amount of the ingredient (default to 0 if not present)  available\_amount = available.get(ingredient, 0)  # Calculate the number of cakes that can be baked with this ingredient  cakes\_with\_ingredient = available\_amount // amount  # Update the maximum number of cakes  max\_cakes = min(max\_cakes, cakes\_with\_ingredient)  # Return the maximum number of cakes  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):  """  Generate the display text for the "like" system.  Args:  names (list): A list of names of people who like the item.  Returns:  str: The display text.  """  if len(names) == 0:  return "no one likes this"  elif len(names) == 1:  return "{} likes this".format(names[0])  elif len(names) == 2:  return "{} and {} like this".format(names[0], names[1])  elif len(names) == 3:  return "{}, {} and {} like this".format(names[0], names[1], names[2])  else:  return "{}, {} and {} others like this".format(names[0], names[1], len(names) - 2) |  |
| 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 number.  Args:  n (int): A non-negative integer.  Returns:  int: The digital root of the number.  """  while n > 9:  n = sum(int(digit) for digit in str(n))  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\_words(s):  """  Sort a string based on the numbers in each word.  Args:  s (str): The input string.  Returns:  str: The sorted string.  """  if not s:  return ""  words = s.split()  sorted\_words = [""] \* len(words)  for word in words:  for char in word:  if char.isdigit():  sorted\_words[int(char) - 1] = word  break  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):  """  Convert an array of binary digits to an integer.  Args:  arr (list): A list of binary digits (0s and 1s).  Returns:  int: The equivalent integer value.  """  return int("".join(map(str, arr)), 2) |  |
| 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 representation.  Args:  r (int): The red value (0-255).  g (int): The green value (0-255).  b (int): The blue value (0-255).  Returns:  str: The hexadecimal representation of the RGB values.  """  # Ensure values are within the valid range (0-255)  r = min(max(r, 0), 255)  g = min(max(g, 0), 255)  b = min(max(b, 0), 255)    # Convert decimal values to hexadecimal  hex\_r = "{:02x}".format(r)  hex\_g = "{:02x}".format(g)  hex\_b = "{:02x}".format(b)    # Return the hexadecimal representation  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\_to\_end(arr):  """  Move all zeros to the end of an array, preserving the order of other elements.  Args:  arr (list): The input array.  Returns:  list: The modified array with zeros moved to the end.  """  # Separate the array into two lists: non-zeros and zeros  non\_zeros = [x for x in arr if x != 0]  zeros = [x for x in arr if x == 0]    # Combine the non-zeros and zeros lists  return non\_zeros + zeros |  |
| 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 a string to a new string where each character is "(" if it appears only once,  or ")" if it appears more than once.  Args:  word (str): The input string.  Returns:  str: The modified string with "(" and ")" characters.  """  word = word.lower()  count\_dict = {}  for char in word:  if char in count\_dict:  count\_dict[char] += 1  else:  count\_dict[char] = 1  result = ""  for char in word:  if count\_dict[char] == 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):  """  Find the missing letter in an array of consecutive letters.  Args:  chars (list): The input array of consecutive letters.  Returns:  str: The missing letter.  """  for i in range(len(chars) - 1):  if ord(chars[i + 1]) - ord(chars[i]) > 1:  return chr(ord(chars[i]) + 1) |  |
| Given an n x n array, return the array elements arranged from outermost elements to the middle element, traveling clockwise. | def arrange\_clockwise(arr):  """  Arrange the array elements from outermost to the middle element, traveling clockwise.  Args:  arr (list): The input 2D array.  Returns:  list: The arranged array elements.  """  result = []  while arr:  # Append the first row  result += arr.pop(0)  # Append the last column  if arr and arr[0]:  for row in arr:  result.append(row.pop())  # Append the last row in reverse order  if arr:  result += arr.pop()[::-1]  # Append the first column in reverse order  if arr and arr[0]:  for row in arr[::-1]:  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\_decimal\_digit(a, b):  """  Return the last decimal digit of a^b.  Args:  a (int): The base number.  b (int): The exponent.  Returns:  int: The last decimal digit of a^b.  """  # Calculate the last decimal digit using modular arithmetic  return pow(a, b, 10) |  |
| 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 extract\_domain(url):  """  Extract the domain name from a URL.  Args:  url (str): The input URL.  Returns:  str: The domain name.  """  parsed\_uri = urllib.parse.urlparse(url)  domain = '{uri.netloc}'.format(uri=parsed\_uri)  return domain |  |
| 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):  roman\_numerals = {'I': 1, 'V': 5, 'X': 10, 'L': 50, 'C': 100, 'D': 500, 'M': 1000}  int\_val = 0  for i in range(len(s)):  if i > 0 and roman\_numerals[s[i]] > roman\_numerals[s[i - 1]]:  int\_val += roman\_numerals[s[i]] - 2 \* roman\_numerals[s[i - 1]]  else:  int\_val += roman\_numerals[s[i]]  return int\_val  def int\_to\_roman(num):  val = [  1000, 900, 500, 400,  100, 90, 50, 40,  10, 9, 5, 4,  1  ]  syb = [  "M", "CM", "D", "CD",  "C", "XC", "L", "XL",  "X", "IX", "V", "IV",  "I"  ]  roman\_num = ''  i = 0  while num > 0:  for \_ in range(num // val[i]):  roman\_num += syb[i]  num -= val[i]  i += 1  return roman\_num |  |
| 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 word\_to\_num(s):  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, "hundred": 100, "thousand": 1000, "million": 1000000  }  s = s.replace("-", " ")  s = s.replace(" and ", " ")  words = s.split()  num = 0  temp = 0  for word in words:  if word in num\_dict:  if word == "hundred":  temp \*= num\_dict[word]  elif word == "thousand" or word == "million":  num += temp \* num\_dict[word]  temp = 0  else:  temp += num\_dict[word]  num += temp  return num |  |
| 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"  components = []  years, seconds = divmod(seconds, 31536000)  if years > 0:  components.append("{} year{}".format(years, "" if years == 1 else "s"))  days, seconds = divmod(seconds, 86400)  if days > 0:  components.append("{} day{}".format(days, "" if days == 1 else "s"))  hours, seconds = divmod(seconds, 3600)  if hours > 0:  components.append("{} hour{}".format(hours, "" if hours == 1 else "s"))  minutes, seconds = divmod(seconds, 60)  if minutes > 0:  components.append("{} minute{}".format(minutes, "" if minutes == 1 else "s"))  if seconds > 0:  components.append("{} second{}".format(seconds, "" if seconds == 1 else "s"))  if len(components) == 1:  return components[0]  last\_component = components.pop()  return ", ".join(components) + " and " + last\_component |  |