**Atkļūdošana – Google Gemini**

Uzdevuma numurs tiek ņemts no Uzdevuma\_instrukcijas.md faila. Šādi var saprast kura uzdevuma kods tiks labots.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Uzdevuma nr.** | **Kļūdainais kods** | **Atkļūdotais kods** | | **Koda pārbaude** | | |
| 17. | 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 "" | import re  def get\_domain\_name(url):  """  Extracts the domain name from a given URL, handling various URL structures  and potential edge cases.  Args:  url (str): The URL from which to extract the domain name.  Returns:  str: The extracted domain name, or an empty string if no domain is found.  """  # Improved regex pattern for flexibility:  pattern = r"(?i)(?:https?://)?(?:www\.)?([^\/?#]+)" # Case-insensitive match  match = re.search(pattern, url)  if match:  domain = match.group(1)  # Handle edge cases (empty domain, single-level domain)  if not domain:  return ""  if '.' not in domain:  return domain # Return single-level domain  # Extract top-level domain (TLD) and second-level domain (SLD):  tld\_parts = domain.split('.')[-1:]  # Handle known Single-Level Domains (SLDs) with two or more dots  if len(tld\_parts) > 2 and tld\_parts[0] not in KNOWN\_SLDS:  return domain # Return entire domain for non-standard SLDs  return '.'.join(tld\_parts) # Return TLD and potentially SLD  return ""  # Define a list of known Single-Level Domains (SLDs) with two or more dots  KNOWN\_SLDS = [  "com", "net", "org", "gov", "edu", "mil", "int", "arpa", "aero", "biz",  "coop", "info", "museum", "name", "pro", "post", "tech", "travel"  ]  # Example usage:  urls = [  "https://www.google.com/search?q=domain+name",  "http://subdomain.example.co.uk/path/to/file",  "ftp://user:password@ftp.example.org",  "www.wikipedia.org", # Single-level domain  "very.long.subdomain.example.with.many.dots" # Non-standard SLD  ]  for url in urls:  domain = get\_domain\_name(url)  print(f"URL: {url}, Domain: {domain}") | |  | | |
| 18. | 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) | Kodam nebija vajadzīga atkļūdošana, jo kods faktiski darbojās pareizi, to vajadzēja pielāgot dotajiem testiem – samainīt funkciju nosaukumus un ievietot klasē. | | | | |
| 19. | 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 | 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.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]  # Add current to total after each word (except for multipliers)  else:  total += current  current = 0  # Add the final current value to the total  total += current  return total | |  | | |
| 7. | 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" | def likes(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:  # Use all provided names and indicate "others" only if there are more than 2 names provided  other\_count = length - len(names) if len(names) < length else 0  return f"{', '.join(names[:2])} and {other\_count} others like this" | | Šo kodu varēja arī nelabot, jo problēma ir tajā, ka ‘if’ ciklus var vienkārši pagarināt un apstrādāt vairākus vārdus. | | |
| 9. | 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]) | 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 ""  # Find the index where the position number ends (could be 1 or 2 digits)  for i in range(1, 3):  try:  int(sentence[:i]) # Attempt conversion to integer  break  except ValueError:  pass # Not a valid integer, try next index  # Extract position and actual word  position = int(sentence[:i])  actual\_word = sentence[i:]  # Rest of the code remains the same...  # Split the sentence into words and extract positions and actual words  words = []  for word in sentence.split():  position, actual\_word = extract\_position\_and\_word(word) # Use the new function  words.append((position, actual\_word))  # ... (rest of the code for sorting and joining)  def extract\_position\_and\_word(word):  """  Extracts the position number (up to two digits) and the actual word from a given word.  Args:  word: The word containing the position number.  Returns:  A tuple containing the position number (integer) and the actual word (string).  """  for i in range(1, 3):  try:  return int(word[:i]), word[i:]  except ValueError:  pass # Not a valid integer, try next index  raise ValueError("Invalid word format: {}".format(word)) # Raise error for invalid format | |  | | |
| 14. | 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) | 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])  # Check if the first letter has the expected difference from the implicit 'previous' letter (assuming 'A')  if ord(chars[0]) != ord('A') + (expected\_diff \* -1): # Check against implicit 'A' with reversed difference  return chr(ord(chars[0]) + expected\_diff) # Handle missing letter at the beginning  # Iterate through the rest of the array for other missing letters  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)  # If the loop completes without finding a missing letter, all letters are present  return "No letter missing" | |  | | | |
| 16. | 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] | 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  # Get the cycle length of the repeating last digits (consider 4 or power of 4)  cycle\_length = 4  while a % cycle\_length == 0:  cycle\_length \*= 4 # Keep increasing the cycle length if a is divisible by current cycle length  # Reduce b's exponent by finding the remainder when divided by the actual cycle length  b\_exp = b % cycle\_length  # Return the last digit based on the reduced exponent and the cycle  return last\_digit\_cycle[b\_exp] | | |  | |
| 17. | 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 "" | 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  # Check if there are multiple parts (domain and TLD)  if "." in domain\_name:  # Split and return the second-to-last element (domain name)  return domain\_name.split(".")[-2]  else:  # If no subdomain, return the entire domain name  return domain\_name  except ValueError:  # Handle potential parsing errors (invalid URL format)  return "" | | |  | |
| 18. | 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 | 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:  # Check if subtractive notation is applicable (e.g., 4 with IV)  if num >= value \* 5 and key in ("I", "X", "C"):  result += roman\_numeral\_map[key + key] # Add double character for subtraction  num -= value \* 4 # Subtract the combined value  else:  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 i, char in enumerate(roman\_num):  current = roman\_numeral\_map[char]  # Ensure subtractive notation is valid (larger value cannot precede smaller)  if current <= prev and i > 0:  return 0 # Indicate invalid Roman numeral  # Handle subtractive notation (IV, IX)  if current > prev:  sum = sum - prev + current  else:  sum += current  prev = current  return sum | | | Funkciju sākumā tika pievienota klase, lai var palaist testus. | |
| 19. | 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:]) | 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()  result = 0  i = 0  while i < len(words):  if words[i] in millions:  result += millions[words[i]] \* word\_to\_num(" ".join(words[i + 1:]))  i += 1  elif words[i] in thousands:  result += thousands[words[i]] \* word\_to\_num(" ".join(words[i + 1:]))  i += 1  elif words[i] in hundreds:  # Handle hundreds and optional "and"  result += hundreds[words[i]]  if i + 1 < len(words) and words[i + 1] == "and":  i += 2  i += 1  elif i + 1 < len(words) and words[i + 1] in tens: # Handle tens  result += tens[words[i + 1]]  if words[i] != "zero": # Handle "twenty one" vs. "one"  result += word\_to\_num(" ".join(words[i:]))  i += 2  elif i + 1 < len(words) and words[i + 1] in teens: # Handle teens  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]}")  return result  def int\_from\_group(group):  """  This function is no longer used as word\_to\_num handles the entire sentence now.  """  pass # This function can be removed | | |  | |
| 20. | 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] | 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  # Add leading zero for unit count if needed (e.g., "0 seconds")  results.append(f"{count:01} {unit + ('s' if count != 1 else '')}")  if not results:  return "now"  # Join all components with commas and spaces, including the last one  joined\_results = ", ".join(results)  # Ensure proper pluralization for the last component  return joined\_results.replace(" and 0 ", " and ") | | | Kods funkcionāli strādā pareizi, problēma ir tajā, ka pie izvades tiek prasīts vārds ‘and’, nevis ‘,’ starp atšķirīgajām vērtībām | |
| 18. | 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 | class RomanNumerals:  @staticmethod  def to\_roman(val: int) -> str:  """  Converts an integer to a Roman numeral string.  Args:  val: An integer between 1 and 3999.  Returns:  A string representing the Roman numeral equivalent of the integer.  """  # Define the mapping of integer values to Roman numerals (including subtractive cases)  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'  }  result = ''  while val > 0:  # Find the largest possible Roman numeral for the current value  largest\_key = max(key for key in int\_to\_roman\_map.keys() if key <= val)  result += int\_to\_roman\_map[largest\_key]  val -= largest\_key  return result | | Nepievienoja otru funkciju – nno romiešu uz parastajiem cipariem. | |
| 19. | 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 | Netika sniegta atbilde. Divas reizes tika uzdots atrast kļūdas dotajā kodā un arī tika pievienots uzdevums, lai saprastu kontekstu. Sniedza atbildi, ka nevar palīdzēt ar šo problēmu. | | | | |
| 17. | 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 | 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 = f"{uri.netloc}" # Corrected string formatting  return domain |  | | | |