|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Uzdevuma nr.  Uzdevuma numurs tiek ņemts no Uzdevuma\_instrukcijas.md faila. Šādi var saprast kura uzdevuma kods tiks labots. | Kļūdains kods  **Kļūdu atrašana – Meta AI** | Iespējamās kļūdainās vietas | Meta AI atzīmētās rindas ar kļūdām | Meta AI pamatojums kāpēc šeit ir kļūdas |
| 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 "" | 1) Šī vieta iespējams atgriež arī subdomēnu: # Regex pattern to find the domain  pattern = r'(?:http[s]?://)?(?:www\.)?([^/:\?]+)'  2) Šeit skatās uz ne tikai domēnu, bet arī subdomēnu, bet ir situācijas, kad tāds nav: if len(domain\_parts) > 2:  3) Kods nepareizi uzbūvēts – tas nevis izvada domēnus, bet gan pēc domēna doto informāciju, piemēram, no <http://www.example.co.uk> tiek izvadīts ‘co.uk’, nevis ‘example’. | Tiek norādīts, ka kodam vajadzētu darboties kā paredzēts. Taču, tiek norādītas vairākas problēmas, kuru dēļ var arī nedarboties kods:   * Kods atbalsta tikai ASCII rakstzīmes. Ja tiks dota hipersaite, kurā nebūs ASCII rakstzīmes, tad var iegūt nepareizu rezultātu * Kods var neatpazīt nestandarta hipersaites, piemēram, ja hipersaite sākas ar ‘ftp’ vai ‘mailto’, nevis ‘http’, tad kods neuztvers domēna nosaukumu * Ja ir dota hipersaite, kurā nav domēns, tad tiks atgriezta tukša virkne. | |
| 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) | Lai palaistu veiksmīgi visus testus ‘codewars’ vietnē, abas funkcijas vajag ievietot klasē.  Funkciju nosaukumi nav pareizi. | Tika pateikts, ka viss darbojas. Šajā situācijā kods tiešām arī strādā pareizi, tikai vajadzēja to pielāgot situācijai un pievienot/izmainīt funkciju nosaukumus un klasi. Tā kā netika iedots testu konteksts, tad netika saņemts izlabots kods vai konkrēti ieteikumi, ko uzlabot. | |
| 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 | Domājams, ka kļūda ir koda loģikā, jo tas spēj izpildīt aptuveni pusi no testiem, taču pie sarežģītākiem cipariem, tas izpilda nepareizi. Visticamāk netiek pareizi nolasīti cipari. | Netiek norādītas konkrētas koda rindiņas, kurās būtu kļūdas, taču tiek norādīti potenciāli kļūdu cēloņi:   * Kods nepareizi apstrādā vērtību ‘simts’ (hundred). Ja tiek konstatēta vērtība simts, tad pašreizējā vērtība ir jāreizina ar simts, taču kodā pašreizējā vērtība tiek pieskaitīta kopsummai, kas ir jādara tikai tad, ja ir konstatēts ‘tūkstotis’ vai ‘miljons’ (thousand or million) * Ja tiek konstatēts ‘tūkstotis’ vai ‘miljons’, tad pašreizējā vērtība tiek pieskaitīta pie kopsummas, bet pēc tam netiek atiestatīta uz 0, kas nozīmē, ka nākamais skaitlis tiks pievienots iepriekšējai kopsummai. Pašreizējā vērtība ir jāiestata uz 0 pēc tam, kad tā ir jau pievienota kopsummai. | |
| 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" | Koda loģika strādā, taču dažus testus tas neiziet, šajā gadījumā problēma ir šeit:  if length == 0:  elif length == 1:  elif length == 2: else: Šīs rindiņas norāda, cik vārdi tiks izvadīti secīgi pēc kārtas. Tā kā vienā no testa gadījumiem vajag izvadīt trīs vārdus un tad ziņojumu ‘others like this’, tad šeit nokļūdās, jo tiek izvadīti tikai divi vārdi un ziņojums. | 1) "and {length - 2} others like this" | 1) Tiek norādīts, ka ir izvēlēts nepareizs formatējums. Konkrētā koda rindiņa ir jāizmaina uz "{length - 2} others like this".  2) Tiek norādīts, ka vajag apstrādāt gadījumus, kad ir tieši trīs vārdi, nevis pārstāt pie diviem. |
| 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]) | Koda loģika nav pareiza. Kods darbojas pēc principa, ka cipars vārdā vienmēr būs pirmais, taču tā var nebūt. | Tiek norādīts, ka kļūdas var radīt:   * Kodā šobrīd tiek pieņemts, ka pozīcijas numurs vienmēr ir vārda pirmā rakstzīme, taču var būt gadījumi, kad tā nav. * Netiek apstrādāti gadījumi, ja ir vairāk nekā viens numurs, tad tiek tikai izvēlēts pirmais pēc pozīcijas cipars. | |
| 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) | Koda loģikas kļūda. | 1) ‘expected\_diff’ | 1) Iesaka nomainīt doto nosaukumu uz saprotamāku, piemēram, ‘ letter\_offset’ vai ‘letter\_step’.  2) Iesaka izmantot efektīvāku ciklu jeb cilpu (loop). Šobrīd tiek atkārtots viss masīvs, to varētu aizvietot ar to, ka ja tiklīdz tiek atrasts trūkstošais burts, uzreiz izkļūst no cilpas. |
| 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] | 1) Kopsavilkumā koda loģika nav līdz galam pareizi uztaisīta. Ir uztaisīts, ka kods darbojas tikai pēc konkrēta cikla (b\_exp un last\_digit\_cycle). | Netiek norādītas konkrētas koda rindiņas, kuras būtu nepareizas. Tiek norādīts, ka koda loģikā ir trūkumi:   * Nepareizi tiek apstrādāti īpašie gadījumi – kods apstrādā, kad a un b ir 0, taču neapstrādā kad b ir 0, bet a nav 0. Ja b ir 0, tad rezultātam vienmēr jābūt 1, neatkarīgi no a vērtības * Nepareizs pēdējā cipara aprēķins. Šobrīd kodā tiek izmantots cikls (1, 0, 5, 6), lai aprēķinātu pēdējo ciparu, taču tas nav atbilstošs visām a vērtībām. Pareizais cikls ir atkarīgs no a vērtības pēdējā cipara. | |
| 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 "" | Nepareiza koda loģika. Ja ir kaut kas pēc domēna, tad tas tiks izvadīts. Ja vietnes sākumā nav ‘https:’ tad izvada tukšu vērtību. | Netiek dotas konkrētas koda rindiņas, kuras nedarbotos, taču tiek norādīts uz vairākām koda loģikas problēmām:   * Kods sadala domēna nosaukumu vairākās daļās un atgriež otro nosaukumu, pieņemot, ka tas ir jāizvada. Tomēr tas neattiecas uz gadījumiem, kad ir vairāki apakšdomēni. * Kods neņem vērā valsts koda augstākā līmeņa domēnus (ccTLD). Piemēram, ‘.us’. | |
| 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 | Testa ietvaros nav pievienota klase, kas iekļautu abas funkcijas. 1) ‘Roman\_numeral\_map’ – neiekļauj pareizi tādus ciparus, kā 9 vai 4. | Tiek norādīts, ka netiek pareizi apstrādāti romiešu cipari IV (4) un IX (9) – tie tiek nepareizi gan veidoti, gan darbības ar tiem nepareizi tiek veidotas. | |
| 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:]) | 1) Testu ietvaros visu vajadzētu rakstīt vienā funkcijā. 2) ‘tens’ un ‘teens’ nav loģiski iesaistīts funkcijās. | Tiek norādītas vairākas koda loģikas problēmas:   * Netiek pareizi apstrādāti gadījumi, kad vārdu savienojums sastāv no desmitiem un vieniem (tens and ones). Kā arī tiek norādīts, ka netiek apstrādāti gadījumi, kad ir konkrēti desmiti, piemēram, ‘eleven’. * Netiek korekti apstrādāti gadījumi, kad vārdu savienojums sastāv no simts (hundred) un kaut kā cita. * Netiek korekti apstrādāti gadījumi, kad vārdu savienojums sastāv no tūkstots (thousand) vai miljons (million) un kaut kā cita. * Netiek korekti apstrādāti gadījumi, kad vārdu savienojumus parādās starpvārds ‘and’. | |
| 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] | Neloģiski izveidota beigu daļa kodam, var pārstrukturēt. | Tiek norādītas koda loģikas problēmas:   * Nepareizi apstrādāti gadījumi ar nulles vērtību – ja skaits ir nulle, tad komponenta pievienošana sarakstam ir jāizlaiž. * Nepareiza pēdēja komponenta apstrāde – vienmēr tiek pievienots vārds ‘and’ pirms pēdējā komponenta, pat ja ir tikai viens komponents. Šo vajag pievienot tikai tad, ja ir vairāk nekā viena komponente. | |
| 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 | 1) Nav funkcija, kur tiek pārveidotas vērtības no romiešu cipariem uz parastiem cipariem. | Tiek norādīti vairāki trūkumi kodam:   * Dotajā kodā trūkst funkcija, kurā cipari tiek pārveidoti no romiešu cipariem. * Nepareiza ievades diapazona apstrāde – kods nepārbauda vai ievades vērtība ir norādītajā diapazonā ( 1<= n < 4000 ) | |
| 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 | 1) Nepareiza loģika pie lielākiem cipariem – simtiem, tūkstošiem utt. 2) Nenolasa pareizi vērtības. | Tiek norādīti vairāki iemesli kļūdām:   * ‘And’ apstrāde – netiek apstrādāts neobligātais ‘and’ ievades virknē. To vajadzētu ignorēt, kad ar to saskaras. * Vairāku tūkstošu vai miljonu apstrāde (thousands or millions). Netiek apstrādāti gadījumi, kad virknē ir vairāki tūkstoši vai miljoni. Tai ir jāpievieno pašreizējā vērtība kopsummai un jāatiestata pašreizējā vērtība, kad tiek konstatēts jauns reizinājums. | |
| 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 | Kods strādā pareizi, tas izvada domēnu, taču tas izvada arī neprasīto informāciju – papildus domēnus no dotās hipersaites. | Netiek norādītas konkrētas koda rindiņas.  Tiek norādīts, ka kods neapstrādā pareizi gadījumus, kad ir dota hipersaite ar ‘portu’ (portnumber). Kā arī tiek norādīts, ka tiek nepareizi apstrādāti gadījumi, kad ir dota hipersaite, kurā ir iekļauti lietotājvārdi un paroles. | |