

```

1  # String-related functions without using inbuilt functions
2  def string_length(s):
3      count = 0
4      for _ in s:
5          count += 1
6      return count
7
8  def count_char(s, target):
9      count = 0
10     for ch in s:
11         if ch == target:
12             count += 1
13     return count
14
15  def reverse_string(s):
16     reversed_s = ''
17     for i in range(string_length(s) - 1, -1, -1):
18         reversed_s += s[i]
19     return reversed_s
20
21  def is_palindrome(s):
22     n = string_length(s)
23     for i in range(n // 2):
24         if s[i] != s[n - i - 1]:
25             return False
26     return True
27
28  def to_lower_case(s):
29     result = ''
30     for ch in s:
31         if 'A' <= ch <= 'Z':
32             result += chr(ord(ch) + 32)
33         else:
34             result += ch
35     return result
36
37  def to_upper_case(s):
38     result = ''
39     for ch in s:
40         if 'a' <= ch <= 'z':
41             result += chr(ord(ch) - 32)
42         else:
43             result += ch
44     return result
45
46  def remove_spaces(s):
47     result = ''
48     for ch in s:
49         if ch != ' ':
50             result += ch
51     return result
52
53  def count_words(s):
54     count = 0
55     in_word = False
56     for ch in s:
57         if ch != ' ' and not in_word:
58             count += 1
59             in_word = True
60         elif ch == ' ':
61             in_word = False
62     return count
63
64
65
66
67
68
69

```

```

70 def replace_char(s, old, new):
71     result = ''
72     for ch in s:
73         if ch == old:
74             result += new
75         else:
76             result += ch
77     return result
78
79 def find_substring(s, sub):
80     n = string_length(s)
81     m = string_length(sub)
82     for i in range(n - m + 1):
83         match = True
84         for j in range(m):
85             if s[i + j] != sub[j]:
86                 match = False
87                 break
88         if match:
89             return i
90     return -1
91
92 def remove_vowels(s):
93     result = ''
94     for ch in s:
95         if ch not in 'aeiouAEIOU':
96             result += ch
97     return result
98
99 def get_unique_characters(s):
100     result = ''
101     for ch in s:
102         exists = False
103         for r in result:
104             if ch == r:
105                 exists = True
106                 break
107         if not exists:
108             result += ch
109     return result
110
111 def char_frequency(s):
112     freq = {}
113     for ch in s:
114         if ch in freq:
115             freq[ch] += 1
116         else:
117             freq[ch] = 1
118     return freq
119
120 def capitalize_first_letter(s):
121     result = ''
122     capitalize = True
123     for ch in s:
124         if ch == ' ':
125             result += ch
126             capitalize = True
127         elif capitalize and 'a' <= ch <= 'z':
128             result += chr(ord(ch) - 32)
129             capitalize = False
130         else:
131             result += ch
132             capitalize = False
133     return result
134
135
136
137
138

```

```
139 # List-related functions without using inbuilt functions
140
141 def list_length(lst):
142     count = 0
143     for _ in lst:
144         count += 1
145     return count
146
147 def list_sum(lst):
148     total = 0
149     for num in lst:
150         total += num
151     return total
152
153 def list_max(lst):
154     max_val = lst[0]
155     for num in lst:
156         if num > max_val:
157             max_val = num
158     return max_val
159
160 def list_min(lst):
161     min_val = lst[0]
162     for num in lst:
163         if num < min_val:
164             min_val = num
165     return min_val
166
167 def count_occurrences(lst, target):
168     count = 0
169     for item in lst:
170         if item == target:
171             count += 1
172     return count
173
174 def reverse_list(lst):
175     reversed_lst = []
176     for i in range(list_length(lst) - 1, -1, -1):
177         reversed_lst.append(lst[i])
178     return reversed_lst
179
180 def is_palindrome(lst):
181     n = list_length(lst)
182     for i in range(n // 2):
183         if lst[i] != lst[n - i - 1]:
184             return False
185     return True
186
187 def remove_duplicates(lst):
188     unique = []
189     for item in lst:
190         found = False
191         for u in unique:
192             if u == item:
193                 found = True
194                 break
195         if not found:
196             unique.append(item)
197     return unique
198
199 def sort_list(lst):
200     n = list_length(lst)
201     for i in range(n):
202         for j in range(i + 1, n):
203             if lst[i] > lst[j]:
204                 lst[i], lst[j] = lst[j], lst[i]
205     return lst
206
207
```

```

208 def merge_lists(lst1, lst2):
209     result = []
210     for item in lst1:
211         result.append(item)
212     for item in lst2:
213         result.append(item)
214     return result
215
216 def second_max(lst):
217     max1 = lst[0]
218     for num in lst:
219         if num > max1:
220             max1 = num
221     max2 = None
222     for num in lst:
223         if num != max1:
224             if max2 is None or num > max2:
225                 max2 = num
226     return max2
227
228 def common_elements(lst1, lst2):
229     common = []
230     for i in lst1:
231         for j in lst2:
232             if i == j:
233                 already_added = False
234                 for k in common:
235                     if k == i:
236                         already_added = True
237                 if not already_added:
238                     common.append(i)
239     return common
240
241 def rotate_left(lst, k=2):
242     n = list_length(lst)
243     rotated = []
244     for i in range(k, n):
245         rotated.append(lst[i])
246     for i in range(k):
247         rotated.append(lst[i])
248     return rotated
249
250 def split_even_odd(lst):
251     even = []
252     odd = []
253     for num in lst:
254         if num % 2 == 0:
255             even.append(num)
256         else:
257             odd.append(num)
258     return even, odd
259
260 def are_equal(lst1, lst2):
261     if list_length(lst1) != list_length(lst2):
262         return False
263     for i in range(list_length(lst1)):
264         if lst1[i] != lst2[i]:
265             return False
266     return True
267
268 def list_to_string(lst):
269     result = ''
270     for ch in lst:
271         result += ch
272     return result
273
274
275
276

```

```

277 def string_to_list(s):
278     result = []
279     for ch in s:
280         result.append(ch)
281     return result
282
283 def remove_duplicate_characters(s):
284     result = ''
285     for ch in s:
286         found = False
287         for r in result:
288             if ch == r:
289                 found = True
290                 break
291         if not found:
292             result += ch
293     return result
294
295 def string_length(s):
296     count = 0
297     for _ in s:
298         count += 1
299     return count
300
301 def alternate_case(s):
302     result = ''
303     upper = True
304     for ch in s:
305         if 'a' <= ch <= 'z' or 'A' <= ch <= 'Z':
306             if upper:
307                 if 'a' <= ch <= 'z':
308                     result += chr(ord(ch) - 32)
309                 else:
310                     result += ch
311                 upper = False
312             else:
313                 if 'A' <= ch <= 'Z':
314                     result += chr(ord(ch) + 32)
315                 else:
316                     result += ch
317                 upper = True
318             else:
319                 result += ch
320     return result
321
322 def reverse_each_word(s):
323     result = ''
324     word = ''
325     for ch in s + ' ':
326         if ch != ' ':
327             word += ch
328         else:
329             for i in range(string_length(word)-1, -1, -1):
330                 result += word[i]
331             result += ' '
332             word = ''
333     return result.strip()
334
335
336 # Tuple-related functions without using inbuilt functions
337 def tuple_length(t):
338     count = 0
339     for _ in t:
340         count += 1
341     return count
342
343 def count_occurrences(t, target):
344     count = 0
345     for item in t:

```

```

346         if item == target:
347             count += 1
348     return count
349
350 def find_max(t):
351     max_val = t[0]
352     for item in t:
353         if item > max_val:
354             max_val = item
355     return max_val
356
357 def find_min(t):
358     min_val = t[0]
359     for item in t:
360         if item < min_val:
361             min_val = item
362     return min_val
363 def tuple_sum(t):
364     total = 0
365     for item in t:
366         total += item
367     return total
368
369 def index_of_element(t, value):
370     index = 0
371     for item in t:
372         if item == value:
373             return index
374         index += 1
375     return -1
376 def reverse_tuple(t):
377     result = ()
378     for i in range(tuple_length(t) - 1, -1, -1):
379         result += (t[i],)
380     return result
381
382 def convert_list_to_tuple(lst):
383     result = ()
384     for item in lst:
385         result += (item,)
386     return result
387
388 def convert_tuple_to_list(t):
389     result = []
390     for item in t:
391         result.append(item)
392     return result
393
394 def are_tuples_equal(t1, t2):
395     if tuple_length(t1) != tuple_length(t2):
396         return False
397     for i in range(tuple_length(t1)):
398         if t1[i] != t2[i]:
399             return False
400     return True
401
402 # Dictionary-related functions without using inbuilt functions
403 def dict_length(d):
404     count = 0
405     for _ in d:
406         count += 1
407     return count
408
409 def get_keys(d):
410     keys = []
411     for key in d:
412         keys.append(key)
413     return keys
414

```

```
415 def get_values(d):
416     values = []
417     for key in d:
418         values.append(d[key])
419     return values
420
421 def get_items(d):
422     items = []
423     for key in d:
424         items.append((key, d[key]))
425     return items
426
427 def key_exists(d, target):
428     for key in d:
429         if key == target:
430             return True
431     return False
432
433 def value_exists(d, target):
434     for key in d:
435         if d[key] == target:
436             return True
437     return False
438
439 def count_value_occurrences(d, target):
440     count = 0
441     for key in d:
442         if d[key] == target:
443             count += 1
444     return count
445
446 def add_key_value(d, key, value):
447     d[key] = value
448     return d
449
450 def update_value(d, key, value):
451     if key_exists(d, key):
452         d[key] = value
453     return d
454
455 def delete_key(d, target):
456     new_dict = {}
457     for key in d:
458         if key != target:
459             new_dict[key] = d[key]
460     return new_dict
461
462 def merge_dicts(d1, d2):
463     merged = {}
464     for key in d1:
465         merged[key] = d1[key]
466     for key in d2:
467         merged[key] = d2[key]
468     return merged
469
470 def dict_from_list(pairs):
471     d = {}
472     for pair in pairs:
473         key, value = pair
474         d[key] = value
475     return d
476
477 def invert_dict(d):
478     inverted = {}
479     for key in d:
480         value = d[key]
481         inverted[value] = key
482     return inverted
483
```

```

484 def dict_get(d, key, default=None):
485     for k in d:
486         if k == key:
487             return d[k]
488     return default
489
490 def dict_has_key(d, key):
491     for k in d:
492         if k == key:
493             return True
494     return False
495
496 def dict_update(d1, d2):
497     for key in d2:
498         d1[key] = d2[key]
499     return d1
500
501 def dict_copy(d):
502     new_d = {}
503     for key in d:
504         new_d[key] = d[key]
505     return new_d
506
507 # Set-related functions without using inbuilt functions
508 def set_length(s):
509     count = 0
510     for _ in s:
511         count += 1
512     return count
513
514 def add_to_set(s, value):
515     exists = False
516     for item in s:
517         if item == value:
518             exists = True
519             break
520     if not exists:
521         s.add(value)
522     return s
523
524 def remove_from_set(s, value):
525     new_set = set()
526     for item in s:
527         if item != value:
528             new_set.add(item)
529     return new_set
530
531 def is_member(s, value):
532     for item in s:
533         if item == value:
534             return True
535     return False
536
537 def union_sets(s1, s2):
538     result = set()
539     for item in s1:
540         result.add(item)
541     for item in s2:
542         result.add(item)
543     return result
544
545 def intersection_sets(s1, s2):
546     result = set()
547     for item in s1:
548         for other in s2:
549             if item == other:
550                 result.add(item)
551     return result
552

```



```
553
554
555 def difference_sets(s1, s2):
556     result = set()
557     for item in s1:
558         found = False
559         for other in s2:
560             if item == other:
561                 found = True
562                 break
563         if not found:
564             result.add(item)
565     return result
566
567 def is_subset(s1, s2):
568     for item in s1:
569         found = False
570         for other in s2:
571             if item == other:
572                 found = True
573                 break
574         if not found:
575             return False
576     return True
577
578 def convert_list_to_set(lst):
579     result = set()
580     for item in lst:
581         result = add_to_set(result, item)
582     return result
583
584 def convert_set_to_list(s):
585     result = []
586     for item in s:
587         result.append(item)
588     return result
589
```