

```
1 C:\Users\JOIN\anaconda3\envs\Multi-
  Reviewer_Governance_for_Code_Generation\python.exe C:\Users\JOIN\
  Downloads\Multi-Reviewer_Governance_for_Code_Generation\src\run.py
2 [1/164] HumanEval/0
3 =====
4 [TASK] HumanEval/0 | entry_point=has_close_elements
5 [PROMPT PREVIEW]
6 from typing import List
7
8
9 def has_close_elements(numbers: List[float], threshold: float) -> bool
10   """
11     """ Check if in given list of numbers, are any two numbers closer
12       to each other than
13       given threshold.
14     >>> has_close_elements([1.0, 2.0, 3.0], 0.5)
15     False
16     >>> has_close_elements([1.0, 2.8, 3.0, 4.0, 5.0, 2.0], 0.3)
17     True
18   """
19
20 -----
21 [INITIAL GENERATED CODE]
22 def has_close_elements(numbers: List[float], threshold: float) -> bool
23   :
24     for i in range(len(numbers)):
25       for j in range(i + 1, len(numbers)):
26         if abs(numbers[i] - numbers[j]) < threshold:
27           return True
28     return False
29 -----
30 [HumanEval/0] Attempt 0/3
31 [HumanEval/0] ✘ PASS (attempt=0)
32 =====
33 =====
34 [2/164] HumanEval/1
35 =====
36 [TASK] HumanEval/1 | entry_point=separate_paren_groups
37 [PROMPT PREVIEW]
38 from typing import List
39
40
41 def separate_paren_groups(paren_string: str) -> List[str]:
42   """
43     """ Input to this function is a string containing multiple groups
44       of nested parentheses. Your goal is to
45       separate those group into separate strings and return the list of
46       those.
47     Separate groups are balanced (each open brace is properly closed)
48     and not nested within each other
49     Ignore any spaces in t...
50 
```

```
41 -----
42 -----
42 [INITIAL GENERATED CODE]
43 def separate_paren_groups(paren_string: str) -> List[str]:
44     result = []
45     current_group = []
46     depth = 0
47
48     for char in paren_string.replace(" ", ""):
49         if char == '(':
50             depth += 1
51         elif char == ')':
52             depth -= 1
53             current_group.append(char)
54
55         if depth == 0:
56             result.append(''.join(current_group))
57             current_group = []
58
59     return result
60 -----
61 -----
61 [HumanEval/1] Attempt 0/3
62 [HumanEval/1] ⚡ PASS (attempt=0)
63 =====
63 =====
64 [3/164] HumanEval/2
65 =====
65 =====
66 [TASK] HumanEval/2 | entry_point=truncate_number
67 [PROMPT PREVIEW]
68
69
70 def truncate_number(number: float) -> float:
71     """ Given a positive floating point number, it can be decomposed
72     into
73     and integer part (largest integer smaller than given number) and
74     decimals
75     (leftover part always smaller than 1).
76
77     Return the decimal part of the number.
78     """
79
79 -----
79 -----
80 [INITIAL GENERATED CODE]
81 def truncate_number(number: float) -> float:
82     return number - int(number)
83 -----
83 -----
84 [HumanEval/2] Attempt 0/3
85 [HumanEval/2] ⚡ PASS (attempt=0)
```

```
86 =====
87 =====
87 [4/164] HumanEval/3
88 =====
88 =====
89 [TASK] HumanEval/3 | entry_point=below_zero
90 [PROMPT PREVIEW]
91 from typing import List
92
93
94 def below_zero(operations: List[int]) -> bool:
95     """ You're given a list of deposit and withdrawal operations on a
96     bank account that starts with
96     zero balance. Your task is to detect if at any point the balance
97     of account falls below zero, and
97     at that point function should return True. Otherwise it should
98     return False.
98     >>> below_zero([1, 2, 3])
99     False...
100 -----
100 -----
101 [INITIAL GENERATED CODE]
102 def below_zero(operations: List[int]) -> bool:
103     balance = 0
104     for operation in operations:
105         balance += operation
106         if balance < 0:
107             return True
108     return False
109 -----
109 -----
110 [HumanEval/3] Attempt 0/3
111 [HumanEval/3] 0 PASS (attempt=0)
112 =====
112 =====
113 [5/164] HumanEval/4
114 =====
114 =====
115 [TASK] HumanEval/4 | entry_point=mean_absolute_deviation
116 [PROMPT PREVIEW]
117 from typing import List
118
119
120 def mean_absolute_deviation(numbers: List[float]) -> float:
121     """ For a given list of input numbers, calculate Mean Absolute
121     Deviation
122     around the mean of this dataset.
123     Mean Absolute Deviation is the average absolute difference
123     between each
124     element and a centerpoint (mean in this case):
125     MAD = average | x - x_mean |
126     >>> mean_absolute_deviation([1.0, 2.....
127 -----
```

```
127 -----
128 [INITIAL GENERATED CODE]
129 from typing import List
130
131
132 def mean_absolute_deviation(numbers: List[float]) -> float:
133     """ For a given list of input numbers, calculate Mean Absolute
134     Deviation
135     around the mean of this dataset.
136     Mean Absolute Deviation is the average absolute difference
137     between each
138     element and a centerpoint (mean in this case):
139     MAD = average | x - x_mean |
140     >>> mean_absolute_deviation([1.0, 2.0, 3.0, 4.0])
141     1.0
142     """
143
144     if not numbers:
145         return 0.0
146
147 -----
148 -----
149 [HumanEval/4] Attempt 0/3
150 [HumanEval/4] 0 PASS (attempt=0)
151 =====
152 =====
153 [6/164] HumanEval/5
154 =====
155 from typing import List
156
157
158 def intersperse(numbers: List[int], delimiter: int) -> List[int]:
159     """ Insert a number 'delimiter' between every two consecutive
160     elements of input list `numbers`
161     >>> intersperse([], 4)
162     []
163     >>> intersperse([1, 2, 3], 4)
164     [1, 4, 2, 4, 3]
165     """
166 -----
167 [INITIAL GENERATED CODE]
168 def intersperse(numbers: List[int], delimiter: int) -> List[int]:
169     if not numbers:
170         return []
171
172     result = []
173     for i in range(len(numbers) - 1):
```

```
173         result.extend([numbers[i], delimiter])
174     result.append(numbers[-1])
175
176     return result
177 -----
178 -----
178 [HumanEval/5] Attempt 0/3
179 [HumanEval/5] Ø PASS (attempt=0)
180 =====
181 =====
181 [7/164] HumanEval/6
182 =====
183 [TASK] HumanEval/6 | entry_point=parse_nested_parens
184 [PROMPT PREVIEW]
185 from typing import List
186
187
188 def parse_nested_parens(paren_string: str) -> List[int]:
189     """ Input to this function is a string represented multiple
190     groups for nested parentheses separated by spaces.
191     For each of the group, output the deepest level of nesting of
192     parentheses.
193     E.g. ()() has maximum two levels of nesting while ((()) has
194     three.
195
196     >>> parse_nested_parens('(()()) ((())) () ((...'))
197 -----
198 -----
198 [INITIAL GENERATED CODE]
199 def parse_nested_parens(paren_string: str) -> List[int]:
200     result = []
201     for group in paren_string.split():
202         depth = max_depth = 0
203         for char in group:
204             if char == '(':
205                 depth += 1
206                 max_depth = max(max_depth, depth)
207             elif char == ')':
208                 depth -= 1
209             result.append(max_depth)
210
211     return result
212
213 -----
213 [8/164] HumanEval/7
214 =====
214 [TASK] HumanEval/7 | entry_point=filter_by_substring
215 [PROMPT PREVIEW]
```

```
216 from typing import List
217
218
219 def filter_by_substring(strings: List[str], substring: str) -> List[str]:
220     """ Filter an input list of strings only for ones that contain
221         given substring
222     >>> filter_by_substring([], 'a')
223     []
224     >>> filter_by_substring(['abc', 'bacd', 'cde', 'array'], 'a')
225     ['abc', 'bacd', 'array']
226     """
227 -----
228 -----
229 [INITIAL GENERATED CODE]
230 def filter_by_substring(strings: List[str], substring: str) -> List[str]:
231     return [s for s in strings if substring in s]
232 -----
233 -----
234 [HumanEval/7] Attempt 0/3
235 [HumanEval/7] 0 PASS (attempt=0)
236 =====
237 =====
238 [9/164] HumanEval/8
239 =====
240 =====
241 [TASK] HumanEval/8 | entry_point=sum_product
242 [PROMPT PREVIEW]
243 from typing import List, Tuple
244
245
246 def sum_product(numbers: List[int]) -> Tuple[int, int]:
247     """ For a given list of integers, return a tuple consisting of a
248         sum and a product of all the integers in a list.
249         Empty sum should be equal to 0 and empty product should be equal
250         to 1.
251     >>> sum_product([])
252     (0, 1)
253     >>> sum_product([1, 2, 3, 4])
254     (10, 24)
255     """
256 -----
257 -----
258 [INITIAL GENERATED CODE]
259 from typing import List, Tuple
260
261
262 def sum_product(numbers: List[int]) -> Tuple[int, int]:
263     """ For a given list of integers, return a tuple consisting of a
264         sum and a product of all the integers in a list.
265         Empty sum should be equal to 0 and empty product should be equal
266         to 1.
```

```
257     >>> sum_product([])
258     (0, 1)
259     >>> sum_product([1, 2, 3, 4])
260     (10, 24)
261     """
262     if not numbers:
263         return (0, 1)
264
265     total_sum = sum(numbers)
266     total_product = 1
267     for number in numbers:
268         total_product *= number
269
270     return (total_sum, total_product)
271 -----
-----
272 [HumanEval/8] Attempt 0/3
273 [HumanEval/8] ✓ PASS (attempt=0)
274 =====
=====
275 [10/164] HumanEval/9
276 =====
=====
277 [TASK] HumanEval/9 | entry_point=rolling_max
278 [PROMPT PREVIEW]
279 from typing import List, Tuple
280
281
282 def rolling_max(numbers: List[int]) -> List[int]:
283     """ From a given list of integers, generate a list of rolling
284     maximum element found until given moment
285     in the sequence.
286     >>> rolling_max([1, 2, 3, 2, 3, 4, 2])
287     [1, 2, 3, 3, 3, 4, 4]
288     """
289 -----
-----
290 [INITIAL GENERATED CODE]
291 from typing import List
292
293 def rolling_max(numbers: List[int]) -> List[int]:
294     if not numbers:
295         return []
296
297     result = []
298     current_max = float('-inf')
299     for number in numbers:
300         current_max = max(current_max, number)
301         result.append(current_max)
302
303     return result
304 -----
```

```
304 -----
305 [HumanEval/9] Attempt 0/3
306 [HumanEval/9] ✘ PASS (attempt=0)
307 =====
=====
308 [11/164] HumanEval/10
309 =====
=====
310 [TASK] HumanEval/10 | entry_point=make_palindrome
311 [PROMPT PREVIEW]
312
313
314 def is_palindrome(string: str) -> bool:
315     """ Test if given string is a palindrome """
316     return string == string[::-1]
317
318
319 def make_palindrome(string: str) -> str:
320     """ Find the shortest palindrome that begins with a supplied
321         string.
322         Algorithm idea is simple:
323             - Find the longest postfix of supplied string that is a
324                 palindrome.
325             - Append to the end of the string reverse of a string...
326 -----
327 -----
328 [INITIAL GENERATED CODE]
329 def make_palindrome(string: str) -> str:
330     """ Find the shortest palindrome that begins with a supplied
331         string.
332         Algorithm idea is simple:
333             - Find the longest postfix of supplied string that is a
334                 palindrome.
335             - Append to the end of the string reverse of a string prefix that
336                 comes before the palindromic suffix.
337             >>> make_palindrome('')
338             ''
339             >>> make_palindrome('cat')
340             'catac'
341             >>> make_palindrome('cata')
342             'catac'
343             """
344             if is_palindrome(string):
345                 return string
346                 for i in range(len(string), -1, -1):
347                     if is_palindrome(string[i:]):
348                         prefix = string[:i]
349                         return string + prefix[::-1]
350 -----
351 -----
352 [HumanEval/10] Attempt 0/3
353 [HumanEval/10] ✘ FAIL (attempt=0)
354 [TEST ERROR PREVIEW]
```

```
348 Traceback (most recent call last):
349   File "C:\Users\JOIN\Downloads\Multi-
Reviewer_Governance_for_Code_Generation\src\evaluator\
humaneval_evaluator.py", line 39, in _worker
350     ns["check"](candidate)
351   File "<string>", line 12, in check
352 AssertionError
353 -----
354 -----
354 [HumanEval/10] [REPAIRED CODE] (after attempt=0)
355 def make_palindrome(string: str) -> str:
356     """ Find the shortest palindrome that begins with a supplied
string.
357     Algorithm idea is simple:
358     - Find the longest postfix of supplied string that is a
palindrome.
359     - Append to the end of the string reverse of a string prefix that
comes before the palindromic suffix.
360     >>> make_palindrome('')
361     ''
362     >>> make_palindrome('cat')
363     'catac'
364     >>> make_palindrome('cata')
365     'catac'
366     """
367     if is_palindrome(string):
368         return string
369     for i in range(len(string), -1, -1):
370         if is_palindrome(string[i:]):
371             prefix = string[:i]
372             return string + prefix[::-1]
373
374 def is_palindrome(s: str) -> bool:
375     """ Check if a given string is a palindrome. """
376     return s == s[::-1]
377 -----
378 -----
378 [HumanEval/10] Attempt 1/3
379 [HumanEval/10] ⚡ FAIL (attempt=1)
380 [TEST ERROR PREVIEW]
381 Traceback (most recent call last):
382   File "C:\Users\JOIN\Downloads\Multi-
Reviewer_Governance_for_Code_Generation\src\evaluator\
humaneval_evaluator.py", line 39, in _worker
383     ns["check"](candidate)
384   File "<string>", line 12, in check
385 AssertionError
386 -----
387 -----
387 [HumanEval/10] [REPAIRED CODE] (after attempt=1)
388 def make_palindrome(string: str) -> str:
389     """ Find the shortest palindrome that begins with a supplied
string.
```

```
390     Algorithm idea is simple:
391     - Find the longest postfix of supplied string that is a
392       palindrome.
392     - Append to the end of the string reverse of a string prefix that
393       comes before the palindromic suffix.
393     >>> make_palindrome('')
394     ''
395     >>> make_palindrome('cat')
396     'catac'
397     >>> make_palindrome('cata')
398     'catac'
399     """
400     if is_palindrome(string):
401         return string
402     for i in range(len(string), -1, -1):
403         if is_palindrome(string[:i]):
404             prefix = string[i:]
405             return string + prefix[::-1]
406
407 def is_palindrome(s: str) -> bool:
408     """ Check if a given string is a palindrome. """
409     return s == s[::-1]
410 -----
411 -----
411 [HumanEval/10] Attempt 2/3
412 [HumanEval/10] ✘ FAIL (attempt=2)
413 [TEST ERROR PREVIEW]
414 Traceback (most recent call last):
415   File "C:\Users\JOIN\Downloads\Multi-
416     Reviewer_Governance_for_Code_Generation\src\evaluator\
417       humaneval_evaluator.py", line 39, in _worker
416       ns["check"](candidate)
417     File "<string>", line 12, in check
418     AssertionError
419 -----
420 -----
420 [HumanEval/10] [REPAIRED CODE] (after attempt=2)
421 def make_palindrome(string: str) -> str:
422     """ Find the shortest palindrome that begins with a supplied
423       string.
423     Algorithm idea is simple:
424     - Find the longest postfix of supplied string that is a
425       palindrome.
425     - Append to the end of the string reverse of a string prefix that
426       comes before the palindromic suffix.
426     >>> make_palindrome('')
427     ''
428     >>> make_palindrome('cat')
429     'catac'
430     >>> make_palindrome('cata')
431     'catac'
432     """
433     if is_palindrome(string):
```

```
434     return string
435     for i in range(len(string), -1, -1):
436         if is_palindrome(string[:i]):
437             prefix = string[i:]
438             return string + prefix[::-1]
439
440 def is_palindrome(s: str) -> bool:
441     """ Check if a given string is a palindrome. """
442     return s == s[::-1]
443 -----
-----
444 [HumanEval/10] Attempt 3/3
445 [HumanEval/10] ✘ FAIL (attempt=3)
446 [TEST ERROR PREVIEW]
447 Traceback (most recent call last):
448   File "C:\Users\JOIN\Downloads\Multi-
    Reviewer_Governance_for_Code_Generation\src\evaluator\
      humaneval_evaluator.py", line 39, in _worker
449     ns["check"](candidate)
450   File "<string>", line 12, in check
451 AssertionError
452 -----
-----
453 [HumanEval/10] ✘ FINAL FAIL after 3 retries
454 =====
=====
455 [12/164] HumanEval/11
456 =====
=====
457 [TASK] HumanEval/11 | entry_point=string_xor
458 [PROMPT PREVIEW]
459 from typing import List
460
461
462 def string_xor(a: str, b: str) -> str:
463     """ Input are two strings a and b consisting only of 1s and 0s.
464     Perform binary XOR on these inputs and return result also as a
        string.
465     >>> string_xor('010', '110')
466     '100'
467     """
468 -----
-----
469 [INITIAL GENERATED CODE]
470 def string_xor(a: str, b: str) -> str:
471     return ''.join(str(int(x) ^ int(y)) for x, y in zip(a, b))
472 -----
-----
473 [HumanEval/11] Attempt 0/3
474 [HumanEval/11] ✘ PASS (attempt=0)
475 =====
=====
476 [13/164] HumanEval/12
```

```
477 =====
478 =====
479 [TASK] HumanEval/12 | entry_point=longest
480 [PROMPT PREVIEW]
481 from typing import List, Optional
482
483 def longest(strings: List[str]) -> Optional[str]:
484     """ Out of list of strings, return the longest one. Return the
485     first one in case of multiple
486     strings of the same length. Return None in case the input list is
487     empty.
488     >>> longest([])
489
490     >>> longest(['a', 'b', 'c'])
491     'a'
492     >>> longest(['a', 'bb', 'ccc'])
493     'ccc'
494     """
495 -----
496 -----
497 [INITIAL GENERATED CODE]
498 from typing import List, Optional
499
500 def longest(strings: List[str]) -> Optional[str]:
501     """ Out of list of strings, return the longest one. Return the
502     first one in case of multiple
503     strings of the same length. Return None in case the input list is
504     empty.
505     >>> longest([])
506
507     >>> longest(['a', 'b', 'c'])
508     'a'
509     >>> longest(['a', 'bb', 'ccc'])
510     'ccc'
511     """
512     if not strings:
513         return None
514     return max(strings, key=len)
515 -----
516 -----
517 [HumanEval/12] Attempt 0/3
518 [HumanEval/12] ✘ PASS (attempt=0)
519 =====
520 =====
521 [TASK] HumanEval/13 | entry_point=greatest_common_divisor
522 [PROMPT PREVIEW]
523
524
```

```
521 def greatest_common_divisor(a: int, b: int) -> int:
522     """ Return a greatest common divisor of two integers a and b
523     >>> greatest_common_divisor(3, 5)
524     1
525     >>> greatest_common_divisor(25, 15)
526     5
527     """
528 -----
529 [INITIAL GENERATED CODE]
530 def greatest_common_divisor(a: int, b: int) -> int:
531     while b != 0:
532         a, b = b, a % b
533     return a
534 -----
535 [HumanEval/13] Attempt 0/3
536 [HumanEval/13] ✘ PASS (attempt=0)
537 =====
538 [15/164] HumanEval/14
539 =====
540 [TASK] HumanEval/14 | entry_point=all_prefixes
541 [PROMPT PREVIEW]
542 from typing import List
543
544
545 def all_prefixes(string: str) -> List[str]:
546     """ Return list of all prefixes from shortest to longest of the
547     input string
548     >>> all_prefixes('abc')
549     ['a', 'ab', 'abc']
550 """
551 [INITIAL GENERATED CODE]
552 return [string[:i] for i in range(1, len(string) + 1)]
553 -----
554 [HumanEval/14] Attempt 0/3
555 [HumanEval/14] ✘ FAIL (attempt=0)
556 [TEST ERROR PREVIEW]
557 Traceback (most recent call last):
558   File "C:\Users\JOIN\Downloads\Multi-
      Reviewer_Governance_for_Code_Generation\src\evaluator\
      humaneval_evaluator.py", line 27, in _worker
559     exec(prompt + "\n" + completion + "\n", ns, ns)
560   File "<string>", line 10
561 SyntaxError: 'return' outside function
562 -----
563 [HumanEval/14] [REPAIRED CODE] (after attempt=0)
```

```
564 def generate_prefixes(string):
565     return [string[:i] for i in range(1, len(string) + 1)]
566 -----
567 [HumanEval/14] Attempt 1/3
568 [HumanEval/14] ⚡ FAIL (attempt=1)
569 [TEST ERROR PREVIEW]
570 Traceback (most recent call last):
571   File "C:\Users\JOIN\Downloads\Multi-
      Reviewer_Governance_for_Code_Generation\src\evaluator\
      humaneval_evaluator.py", line 39, in _worker
572     ns["check"](candidate)
573   File "<string>", line 10, in check
574 AssertionError
575 -----
576 [HumanEval/14] [REPAIRED CODE] (after attempt=1)
577 def generate_prefixes(string):
578     return [string[:i] for i in range(len(string) + 1)]
579 -----
580 [HumanEval/14] Attempt 2/3
581 [HumanEval/14] ⚡ FAIL (attempt=2)
582 [TEST ERROR PREVIEW]
583 Traceback (most recent call last):
584   File "C:\Users\JOIN\Downloads\Multi-
      Reviewer_Governance_for_Code_Generation\src\evaluator\
      humaneval_evaluator.py", line 39, in _worker
585     ns["check"](candidate)
586   File "<string>", line 10, in check
587 AssertionError
588 -----
589 [HumanEval/14] [REPAIRED CODE] (after attempt=2)
590 def generate_prefixes(string):
591     return [string[:i] for i in range(len(string) + 1)]
592 -----
593 [HumanEval/14] Attempt 3/3
594 [HumanEval/14] ⚡ FAIL (attempt=3)
595 [TEST ERROR PREVIEW]
596 Traceback (most recent call last):
597   File "C:\Users\JOIN\Downloads\Multi-
      Reviewer_Governance_for_Code_Generation\src\evaluator\
      humaneval_evaluator.py", line 39, in _worker
598     ns["check"](candidate)
599   File "<string>", line 10, in check
600 AssertionError
601 -----
602 [HumanEval/14] ⚡ FINAL FAIL after 3 retries
603 =====
```

```
604 [16/164] HumanEval/15
605 =====
606 =====
606 [TASK] HumanEval/15 | entry_point=string_sequence
607 [PROMPT PREVIEW]
608
609
610 def string_sequence(n: int) -> str:
611     """ Return a string containing space-delimited numbers starting
       from 0 upto n inclusive.
612     >>> string_sequence(0)
613     '0'
614     >>> string_sequence(5)
615     '0 1 2 3 4 5'
616     """
617 -----
618 -----
618 [INITIAL GENERATED CODE]
619 def string_sequence(n: int) -> str:
620     return ' '.join(str(i) for i in range(n + 1))
621 -----
622 -----
622 [HumanEval/15] Attempt 0/3
623 [HumanEval/15] ✘ PASS (attempt=0)
624 =====
624 =====
625 [17/164] HumanEval/16
626 =====
626 =====
627 [TASK] HumanEval/16 | entry_point=count_distinct_characters
628 [PROMPT PREVIEW]
629
630
631 def count_distinct_characters(string: str) -> int:
632     """ Given a string, find out how many distinct characters (
       regardless of case) does it consist of
633     >>> count_distinct_characters('xyzXYZ')
634     3
635     >>> count_distinct_characters('Jerry')
636     4
637     """
638 -----
638 -----
639 [INITIAL GENERATED CODE]
640 def count_distinct_characters(string: str) -> int:
641     return len(set(string.lower()))
642 -----
642 -----
643 [HumanEval/16] Attempt 0/3
644 [HumanEval/16] ✘ PASS (attempt=0)
645 =====
645 =====
646 [18/164] HumanEval/17
```

```
647 =====
648 =====
649 [TASK] HumanEval/17 | entry_point=parse_music
650 [PROMPT PREVIEW]
651 from typing import List
652
653 def parse_music(music_string: str) -> List[int]:
654     """ Input to this function is a string representing musical notes
655         in a special ASCII format.
656         Your task is to parse this string and return list of integers
657         corresponding to how many beats does each
658             note last.
659
660             Here is a legend:
661                 'o' - whole note, lasts four beats
662                 'o|' - half note, lasts two beats
663                 '....'
664
665 [INITIAL GENERATED CODE]
666 def parse_music(music_string: str) -> List[int]:
667     """ Input to this function is a string representing musical notes
668         in a special ASCII format.
669         Your task is to parse this string and return list of integers
670             corresponding to how many beats does each
671                 note last.
672
673             Here is a legend:
674                 'o' - whole note, lasts four beats
675                 'o|' - half note, lasts two beats
676                 '.|' - quater note, lasts one beat
677
678                 >>> parse_music('o o| .| o| o| .| .| .| o o')
679                 [4, 2, 1, 2, 2, 1, 1, 1, 4, 4]
680
681             """
682
683             notes = music_string.split()
684             beats = []
685             for note in notes:
686                 if note == 'o':
687                     beats.append(4)
688                 elif note == 'o|':
689                     beats.append(2)
690                 elif note == '.|':
691                     beats.append(1)
692             return beats
693
694 [HumanEval/17] Attempt 0/3
695 [HumanEval/17] 0 PASS (attempt=0)
696 =====
697 =====
698 [19/164] HumanEval/18
```

```
692 =====
693 =====
693 [TASK] HumanEval/18 | entry_point=how_many_times
694 [PROMPT PREVIEW]
695
696
697 def how_many_times(string: str, substring: str) -> int:
698     """ Find how many times a given substring can be found in the
699     original string. Count overaloping cases.
700     >>> how_many_times('', 'a')
700     0
701     >>> how_many_times('aaa', 'a')
702     3
703     >>> how_many_times('aaaa', 'aa')
704     3
705     """
706 -----
707 -----
707 [INITIAL GENERATED CODE]
708 def how_many_times(string: str, substring: str) -> int:
709     count = 0
710     start = 0
711     while True:
712         start = string.find(substring, start)
713         if start == -1:
714             break
715         count += 1
716         start += 1
717     return count
718 -----
719 -----
719 [HumanEval/18] Attempt 0/3
720 [HumanEval/18] ✘ PASS (attempt=0)
721 =====
721 =====
722 [20/164] HumanEval/19
723 =====
723 =====
724 [TASK] HumanEval/19 | entry_point=sort_numbers
725 [PROMPT PREVIEW]
726 from typing import List
727
728
729 def sort_numbers(numbers: str) -> str:
730     """ Input is a space-delimited string of numberals from 'zero' to
731     'nine'.
732     Valid choices are 'zero', 'one', 'two', 'three', 'four', 'five',
733     'six', 'seven', 'eight' and 'nine'.
734     Return the string with numbers sorted from smallest to largest
735     >>> sort_numbers('three one five')
736     'one three five'
735     """
736 -----
```

```
736 -----
737 [INITIAL GENERATED CODE]
738 def sort_numbers(numbers: str) -> str:
739     num_map = {
740         "zero": 0, "one": 1, "two": 2, "three": 3, "four": 4,
741         "five": 5, "six": 6, "seven": 7, "eight": 8, "nine": 9
742     }
743     sorted_numbers = sorted(numbers.split(), key=lambda x: num_map[x])
744     return ' '.join(sorted_numbers)
745 -----
746 -----
747 [HumanEval/19] Attempt 0/3
748 [HumanEval/19] ⚡ PASS (attempt=0)
749 =====
750 =====
751 [21/164] HumanEval/20
752 =====
753 from typing import List, Tuple
754
755
756 def find_closest_elements(numbers: List[float]) -> Tuple[float, float]:
757     """ From a supplied list of numbers (of length at least two)
    select and return two that are the closest to each
758     other and return them in order (smaller number, larger number).
759     >>> find_closest_elements([1.0, 2.0, 3.0, 4.0, 5.0, 2.2])
760     (2.0, 2.2)
761     >>> find_closest_elements([1.0...
762 -----
763 -----
764 [INITIAL GENERATED CODE]
765 def find_closest_elements(numbers: List[float]) -> Tuple[float, float]:
766     numbers.sort()
767     closest_pair = (numbers[0], numbers[1])
768     min_diff = abs(numbers[1] - numbers[0])
769
770     for i in range(1, len(numbers) - 1):
771         diff = abs(numbers[i + 1] - numbers[i])
772         if diff < min_diff:
773             min_diff = diff
774             closest_pair = (numbers[i], numbers[i + 1])
775
776     return closest_pair
777 -----
778 [HumanEval/20] Attempt 0/3
779 [HumanEval/20] ⚡ PASS (attempt=0)
780 =====
```

```
779 =====
780 [22/164] HumanEval/21
781 =====
782 [TASK] HumanEval/21 | entry_point=rescale_to_unit
783 [PROMPT PREVIEW]
784 from typing import List
785
786
787 def rescale_to_unit(numbers: List[float]) -> List[float]:
788     """ Given list of numbers (of at least two elements), apply a
    linear transform to that list,
789     such that the smallest number will become 0 and the largest will
    become 1
790     >>> rescale_to_unit([1.0, 2.0, 3.0, 4.0, 5.0])
791     [0.0, 0.25, 0.5, 0.75, 1.0]
792     """
793 -----
794 -----
794 [INITIAL GENERATED CODE]
795 from typing import List
796
797
798 def rescale_to_unit(numbers: List[float]) -> List[float]:
799     """ Given list of numbers (of at least two elements), apply a
    linear transform to that list,
800     such that the smallest number will become 0 and the largest will
    become 1
801     >>> rescale_to_unit([1.0, 2.0, 3.0, 4.0, 5.0])
802     [0.0, 0.25, 0.5, 0.75, 1.0]
803     """
804     min_val = min(numbers)
805     max_val = max(numbers)
806     range_val = max_val - min_val
807     return [(x - min_val) / range_val for x in numbers]
808 -----
809 -----
809 [HumanEval/21] Attempt 0/3
810 [HumanEval/21] ⚡ PASS (attempt=0)
811 =====
812 =====
812 [23/164] HumanEval/22
813 =====
814 =====
814 [TASK] HumanEval/22 | entry_point=filter_integers
815 [PROMPT PREVIEW]
816 from typing import List, Any
817
818
819 def filter_integers(values: List[Any]) -> List[int]:
820     """ Filter given list of any python values only for integers
821     >>> filter_integers(['a', 3.14, 5])
822     [5]
```

```
823     >>> filter_integers([1, 2, 3, 'abc', {}, []])
824     [1, 2, 3]
825     """
826 -----
827 [INITIAL GENERATED CODE]
828 def filter_integers(values: List[Any]) -> List[int]:
829     return [value for value in values if isinstance(value, int)]
830 -----
831 [HumanEval/22] Attempt 0/3
832 [HumanEval/22] ✘ PASS (attempt=0)
833 =====
834 [24/164] HumanEval/23
835 =====
836 [TASK] HumanEval/23 | entry_point=strlen
837 [PROMPT PREVIEW]
838
839
840 def strlen(string: str) -> int:
841     """ Return length of given string
842     >>> strlen('')
843     0
844     >>> strlen('abc')
845     3
846     """
847 -----
848 [INITIAL GENERATED CODE]
849 return len(string)
850 -----
851 [HumanEval/23] Attempt 0/3
852 [HumanEval/23] ✘ FAIL (attempt=0)
853 [TEST ERROR PREVIEW]
854 Traceback (most recent call last):
855   File "C:\Users\JOIN\Downloads\Multi-
     Reviewer_Governance_for_Code_Generation\src\evaluator\
       humaneval_evaluator.py", line 27, in _worker
856     exec(prompt + "\n" + completion + "\n", ns, ns)
857   File "<string>", line 11
858 SyntaxError: 'return' outside function
859 -----
860 [HumanEval/23] [REPAIRED CODE] (after attempt=0)
861 def count_length(string):
862     return len(string)
863 -----
864 [HumanEval/23] Attempt 1/3
865 [HumanEval/23] ✘ FAIL (attempt=1)
```

```
866 [TEST ERROR PREVIEW]
867 Traceback (most recent call last):
868   File "C:\Users\JOIN\Downloads\Multi-
     Reviewer_Governance_for_Code_Generation\src\evaluator\
       humaneval_evaluator.py", line 39, in _worker
869     ns["check"](candidate)
870   File "<string>", line 10, in check
871 AssertionError
872 -----
-----
873 [HumanEval/23] [REPAIRED CODE] (after attempt=1)
874 def count_length(string):
875     return len(string)
876 -----
-----
877 [HumanEval/23] Attempt 2/3
878 [HumanEval/23] ⚡ FAIL (attempt=2)
879 [TEST ERROR PREVIEW]
880 Traceback (most recent call last):
881   File "C:\Users\JOIN\Downloads\Multi-
     Reviewer_Governance_for_Code_Generation\src\evaluator\
       humaneval_evaluator.py", line 39, in _worker
882     ns["check"](candidate)
883   File "<string>", line 10, in check
884 AssertionError
885 -----
-----
886 [HumanEval/23] [REPAIRED CODE] (after attempt=2)
887 def count_length(string):
888     return len(string)
889 -----
-----
890 [HumanEval/23] Attempt 3/3
891 [HumanEval/23] ⚡ FAIL (attempt=3)
892 [TEST ERROR PREVIEW]
893 Traceback (most recent call last):
894   File "C:\Users\JOIN\Downloads\Multi-
     Reviewer_Governance_for_Code_Generation\src\evaluator\
       humaneval_evaluator.py", line 39, in _worker
895     ns["check"](candidate)
896   File "<string>", line 10, in check
897 AssertionError
898 -----
-----
899 [HumanEval/23] ⚡ FINAL FAIL after 3 retries
900 =====
=====
901 [25/164] HumanEval/24
902 =====
=====
903 [TASK] HumanEval/24 | entry_point=largest_divisor
904 [PROMPT PREVIEW]
905
```

```
906
907 def largest_divisor(n: int) -> int:
908     """ For a given number n, find the largest number that divides n
909     evenly, smaller than n
910     >>> largest_divisor(15)
911     5
912     """
913 -----
914 [INITIAL GENERATED CODE]
915 def largest_divisor(n: int) -> int:
916     for i in range(n-1, 0, -1):
917         if n % i == 0:
918             return i
919     return 1
920 -----
921 [HumanEval/24] Attempt 0/3
922 [HumanEval/24] 0 PASS (attempt=0)
923 =====
924 =====
925 [26/164] HumanEval/25
926 =====
927 =====
928 [TASK] HumanEval/25 | entry_point=factorize
929 [PROMPT PREVIEW]
930 from typing import List
931
932 def factorize(n: int) -> List[int]:
933     """ Return list of prime factors of given integer in the order
934     from smallest to largest.
935     Each of the factors should be listed number of times
936     corresponding to how many times it appears in factorization.
937     Input number should be equal to the product of all factors
938     >>> factorize(8)
939     [2, 2, 2]
940     >>> factorize(25)
941     [5...
942     -----
943 [INITIAL GENERATED CODE]
944 from typing import List
945
946 def factorize(n: int) -> List[int]:
947     """ Return list of prime factors of given integer in the order
948     from smallest to largest.
949     Each of the factors should be listed number of times
950     corresponding to how many times it appears in factorization.
951     Input number should be equal to the product of all factors
952     >>> factorize(8)
953     [2, 2, 2]
```

```
949     >>> factorize(25)
950     [5, 5]
951     >>> factorize(70)
952     [2, 5, 7]
953     """
954
955     factors = []
956     # Start with the smallest prime number
957     divisor = 2
958     while n >= divisor * divisor:
959         if n % divisor == 0:
960             factors.append(divisor)
961             n //= divisor
962         else:
963             divisor += 1
964         if n > 1:
965             factors.append(n)
966     return factors
967 -----
968 -----
967 [HumanEval/25] Attempt 0/3
968 [HumanEval/25] 0 PASS (attempt=0)
969 =====
970 =====
970 [27/164] HumanEval/26
971 =====
972 =====
972 [TASK] HumanEval/26 | entry_point=remove_duplicates
973 [PROMPT PREVIEW]
974 from typing import List
975
976
977 def remove_duplicates(numbers: List[int]) -> List[int]:
978     """ From a list of integers, remove all elements that occur more
979     than once.
980     Keep order of elements left the same as in the input.
981     >>> remove_duplicates([1, 2, 3, 2, 4])
982     [1, 3, 4]
983     """
984 -----
985 -----
984 [INITIAL GENERATED CODE]
985 from typing import List
986
987
988 def remove_duplicates(numbers: List[int]) -> List[int]:
989     """ From a list of integers, remove all elements that occur more
990     than once.
991     Keep order of elements left the same as in the input.
992     >>> remove_duplicates([1, 2, 3, 2, 4])
993     [1, 3, 4]
994     """
994     seen = set()
995     result = []
```

```
996     for number in numbers:
997         if number not in seen:
998             seen.add(number)
999             result.append(number)
1000     return result
1001 -----
1002 -----
1002 [HumanEval/26] Attempt 0/3
1003 [HumanEval/26] ⚡ FAIL (attempt=0)
1004 [TEST ERROR PREVIEW]
1005 Traceback (most recent call last):
1006   File "C:\Users\JOIN\Downloads\Multi-
1007     Reviewer_Governance_for_Code_Generation\src\evaluator\
1008       humaneval_evaluator.py", line 39, in _worker
1009         ns["check"](candidate)
1010     File "<string>", line 12, in check
1011   AssertionError
1010 -----
1011 -----
1011 [HumanEval/26] [REPAIRED CODE] (after attempt=0)
1012 from typing import List
1013
1014
1015 def remove_duplicates(numbers: List[int]) -> List[int]:
1016     """ From a list of integers, remove all elements that occur more
1017     than once.
1018     Keep order of elements left the same as in the input.
1019     >>> remove_duplicates([1, 2, 3, 2, 4])
1020     [1, 3, 4]
1021     """
1021     seen = set()
1022     result = []
1023     for number in numbers:
1024         if number not in seen:
1025             seen.add(number)
1026             result.append(number)
1027         else:
1028             seen.remove(number)
1029     return result
1030 -----
1031 -----
1031 [HumanEval/26] Attempt 1/3
1032 [HumanEval/26] ⚡ FAIL (attempt=1)
1033 [TEST ERROR PREVIEW]
1034 Traceback (most recent call last):
1035   File "C:\Users\JOIN\Downloads\Multi-
1036     Reviewer_Governance_for_Code_Generation\src\evaluator\
1037       humaneval_evaluator.py", line 39, in _worker
1038         ns["check"](candidate)
1039     File "<string>", line 12, in check
1038 AssertionError
1039 -----
1039 -----
```

```
1040 [HumanEval/26] [REPAIRED CODE] (after attempt=1)
1041 from typing import List
1042
1043
1044 def remove_duplicates(numbers: List[int]) -> List[int]:
1045     """ From a list of integers, remove all elements that occur more
1046     than once.
1047     Keep order of elements left the same as in the input.
1048     >>> remove_duplicates([1, 2, 3, 2, 4])
1049     [1, 3, 4]
1050     """
1051     seen = set()
1052     result = []
1053     for number in numbers:
1054         if number not in seen:
1055             seen.add(number)
1056             result.append(number)
1057         else:
1058             continue
1059     return result
1060 -----
1061 -----
1060 [HumanEval/26] Attempt 2/3
1061 [HumanEval/26] ⚡ FAIL (attempt=2)
1062 [TEST ERROR PREVIEW]
1063 Traceback (most recent call last):
1064   File "C:\Users\JOIN\Downloads\Multi-
1065     Reviewer_Governance_for_Code_Generation\src\evaluator\
1066       humaneval_evaluator.py", line 39, in _worker
1067         ns["check"](candidate)
1066   File "<string>", line 12, in check
1067 AssertionError
1068 -----
1069 -----
1069 [HumanEval/26] [REPAIRED CODE] (after attempt=2)
1070 from typing import List
1071
1072
1073 def remove_duplicates(numbers: List[int]) -> List[int]:
1074     """ From a list of integers, remove all elements that occur more
1075     than once.
1076     Keep order of elements left the same as in the input.
1076     >>> remove_duplicates([1, 2, 3, 2, 4])
1077     [1, 3, 4]
1078     """
1079     seen = set()
1080     result = []
1081     for number in numbers:
1082         if number not in seen:
1083             seen.add(number)
1084             result.append(number)
1085         else:
1086             continue
```

```
1087     return result
1088 -----
1089 -----
1089 [HumanEval/26] Attempt 3/3
1090 [HumanEval/26] ⚡ FAIL (attempt=3)
1091 [TEST ERROR PREVIEW]
1092 Traceback (most recent call last):
1093   File "C:\Users\JOIN\Downloads\Multi-
1094     Reviewer_Governance_for_Code_Generation\src\evaluator\
1095       humaneval_evaluator.py", line 39, in _worker
1094       ns["check"](candidate)
1095     File "<string>", line 12, in check
1096   AssertionError
1097 -----
1098 -----
1098 [HumanEval/26] ⚡ FINAL FAIL after 3 retries
1099 =====
1100 =====
1100 [28/164] HumanEval/27
1101 =====
1101 =====
1102 [TASK] HumanEval/27 | entry_point=flip_case
1103 [PROMPT PREVIEW]
1104
1105
1106 def flip_case(string: str) -> str:
1107     """ For a given string, flip lowercase characters to uppercase
1107     and uppercase to lowercase.
1108     >>> flip_case('Hello')
1109     'hELLO'
1110     """
1111 -----
1112 -----
1112 [INITIAL GENERATED CODE]
1113 return ''.join(char.lower() if char.isupper() else char.upper() for
1113   char in string)
1114 -----
1115 -----
1115 [HumanEval/27] Attempt 0/3
1116 [HumanEval/27] ⚡ FAIL (attempt=0)
1117 [TEST ERROR PREVIEW]
1118 Traceback (most recent call last):
1119   File "C:\Users\JOIN\Downloads\Multi-
1120     Reviewer_Governance_for_Code_Generation\src\evaluator\
1121       humaneval_evaluator.py", line 27, in _worker
1120       exec(prompt + "\n" + completion + "\n", ns, ns)
1121     File "<string>", line 9
1122 SyntaxError: 'return' outside function
1123 -----
1124 -----
1124 [HumanEval/27] [REPAIRED CODE] (after attempt=0)
1125 def toggle_case(string):
1126     return ''.join(char.lower() if char.isupper() else char.upper()
```

```
1126 for char in string)
1127 -----
-----
1128 [HumanEval/27] Attempt 1/3
1129 [HumanEval/27] ⚡ FAIL (attempt=1)
1130 [TEST ERROR PREVIEW]
1131 Traceback (most recent call last):
1132   File "C:\Users\JOIN\Downloads\Multi-
    Reviewer_Governance_for_Code_Generation\src\evaluator\
      humaneval_evaluator.py", line 39, in _worker
1133     ns["check"](candidate)
1134   File "<string>", line 10, in check
1135 AssertionError
1136 -----
-----
1137 [HumanEval/27] [REPAIRED CODE] (after attempt=1)
1138 def toggle_case(string):
1139     return ''.join(char.lower() if char.islower() else char.upper()
        for char in string)
1140 -----
-----
1141 [HumanEval/27] Attempt 2/3
1142 [HumanEval/27] ⚡ FAIL (attempt=2)
1143 [TEST ERROR PREVIEW]
1144 Traceback (most recent call last):
1145   File "C:\Users\JOIN\Downloads\Multi-
    Reviewer_Governance_for_Code_Generation\src\evaluator\
      humaneval_evaluator.py", line 39, in _worker
1146     ns["check"](candidate)
1147   File "<string>", line 10, in check
1148 AssertionError
1149 -----
-----
1150 [HumanEval/27] [REPAIRED CODE] (after attempt=2)
1151 def toggle_case(string):
1152     return ''.join(char.lower() if char.isupper() else char.upper()
        for char in string)
1153 -----
-----
1154 [HumanEval/27] Attempt 3/3
1155 [HumanEval/27] ⚡ FAIL (attempt=3)
1156 [TEST ERROR PREVIEW]
1157 Traceback (most recent call last):
1158   File "C:\Users\JOIN\Downloads\Multi-
    Reviewer_Governance_for_Code_Generation\src\evaluator\
      humaneval_evaluator.py", line 39, in _worker
1159     ns["check"](candidate)
1160   File "<string>", line 10, in check
1161 AssertionError
1162 -----
-----
1163 [HumanEval/27] ⚡ FINAL FAIL after 3 retries
1164 =====
```

```
1164 =====
1165 [29/164] HumanEval/28
1166 =====
1167 [TASK] HumanEval/28 | entry_point=concatenate
1168 [PROMPT PREVIEW]
1169 from typing import List
1170
1171
1172 def concatenate(strings: List[str]) -> str:
1173     """ Concatenate list of strings into a single string
1174     >>> concatenate([])
1175     ''
1176     >>> concatenate(['a', 'b', 'c'])
1177     'abc'
1178     """
1179 -----
1180 -----
1180 [INITIAL GENERATED CODE]
1181 return ''.join(strings)
1182 -----
1183 -----
1183 [HumanEval/28] Attempt 0/3
1184 [HumanEval/28] ⚡ FAIL (attempt=0)
1185 [TEST ERROR PREVIEW]
1186 Traceback (most recent call last):
1187   File "C:\Users\JOIN\Downloads\Multi-
    Reviewer_Governance_for_Code_Generation\src\evaluator\
      humaneval_evaluator.py", line 27, in _worker
1188     exec(prompt + "\n" + completion + "\n", ns, ns)
1189   File "<string>", line 12
1190 SyntaxError: 'return' outside function
1191 -----
1192 -----
1192 [HumanEval/28] [REPAIRED CODE] (after attempt=0)
1193 def join_strings(strings):
1194     return ''.join(strings)
1195 -----
1196 -----
1196 [HumanEval/28] Attempt 1/3
1197 [HumanEval/28] ⚡ FAIL (attempt=1)
1198 [TEST ERROR PREVIEW]
1199 Traceback (most recent call last):
1200   File "C:\Users\JOIN\Downloads\Multi-
    Reviewer_Governance_for_Code_Generation\src\evaluator\
      humaneval_evaluator.py", line 39, in _worker
1201     ns["check"](candidate)
1202   File "<string>", line 10, in check
1203 AssertionError
1204 -----
1205 -----
1205 [HumanEval/28] [REPAIRED CODE] (after attempt=1)
1206 def join_strings(strings):
```

```
1207     return ''.join(str(s) for s in strings)
1208 -----
1209 -----
1209 [HumanEval/28] Attempt 2/3
1210 [HumanEval/28] ⚡ FAIL (attempt=2)
1211 [TEST ERROR PREVIEW]
1212 Traceback (most recent call last):
1213   File "C:\Users\JOIN\Downloads\Multi-
1214     Reviewer_Governance_for_Code_Generation\src\evaluator\
1215       humaneval_evaluator.py", line 39, in _worker
1216         ns["check"](candidate)
1215   File "<string>", line 10, in check
1216 AssertionError
1217 -----
1218 -----
1218 [HumanEval/28] [REPAIRED CODE] (after attempt=2)
1219 def join_strings(strings):
1220     return ''.join(str(s) for s in strings)
1221 -----
1222 -----
1222 [HumanEval/28] Attempt 3/3
1223 [HumanEval/28] ⚡ FAIL (attempt=3)
1224 [TEST ERROR PREVIEW]
1225 Traceback (most recent call last):
1226   File "C:\Users\JOIN\Downloads\Multi-
1227     Reviewer_Governance_for_Code_Generation\src\evaluator\
1228       humaneval_evaluator.py", line 39, in _worker
1227         ns["check"](candidate)
1228   File "<string>", line 10, in check
1229 AssertionError
1230 -----
1231 -----
1231 [HumanEval/28] ⚡ FINAL FAIL after 3 retries
1232 =====
1233 -----
1233 [30/164] HumanEval/29
1234 =====
1235 -----
1235 [TASK] HumanEval/29 | entry_point=filter_by_prefix
1236 [PROMPT PREVIEW]
1237 from typing import List
1238
1239
1240 def filter_by_prefix(strings: List[str], prefix: str) -> List[str]:
1241     """ Filter an input list of strings only for ones that start
1242     with a given prefix.
1243     >>> filter_by_prefix([], 'a')
1244     []
1245     >>> filter_by_prefix(['abc', 'bcd', 'cde', 'array'], 'a')
1246     ['abc', 'array']
1247     """
1248
1249 -----
```

```
1248 [INITIAL GENERATED CODE]
1249 def filter_by_prefix(strings: List[str], prefix: str) -> List[str]:
1250     return [s for s in strings if s.startswith(prefix)]
1251 -----
1252 -----
1252 [HumanEval/29] Attempt 0/3
1253 [HumanEval/29] ✘ PASS (attempt=0)
1254 =====
1255 -----
1255 [31/164] HumanEval/30
1256 =====
1256 -----
1257 [TASK] HumanEval/30 | entry_point=get_positive
1258 [PROMPT PREVIEW]
1259
1260
1261 def get_positive(l: list):
1262     """Return only positive numbers in the list.
1263     >>> get_positive([-1, 2, -4, 5, 6])
1264     [2, 5, 6]
1265     >>> get_positive([5, 3, -5, 2, -3, 3, 9, 0, 123, 1, -10])
1266     [5, 3, 2, 3, 9, 123, 1]
1267     """
1268 -----
1268 -----
1269 [INITIAL GENERATED CODE]
1270 def get_positive(l: list):
1271     return [x for x in l if x > 0]
1272 -----
1272 -----
1273 [HumanEval/30] Attempt 0/3
1274 [HumanEval/30] ✘ PASS (attempt=0)
1275 =====
1275 -----
1276 -----
1276 [32/164] HumanEval/31
1277 =====
1277 -----
1278 [TASK] HumanEval/31 | entry_point=is_prime
1279 [PROMPT PREVIEW]
1280
1281
1282 def is_prime(n):
1283     """Return true if a given number is prime, and false otherwise.
1284     >>> is_prime(6)
1285     False
1286     >>> is_prime(101)
1287     True
1288     >>> is_prime(11)
1289     True
1290     >>> is_prime(13441)
1291     True
1292     >>> is_prime(61)
1293     True
```

```
1294     >>> is_prime(4)
1295     False
1296     >>> is_prime(1)
1297     False
1298     """
1299 -----
1300 -----
1300 [INITIAL GENERATED CODE]
1301 def is_prime(n):
1302     if n <= 1:
1303         return False
1304     if n <= 3:
1305         return True
1306     if n % 2 == 0 or n % 3 == 0:
1307         return False
1308     i = 5
1309     while i * i <= n:
1310         if n % i == 0 or n % (i + 2) == 0:
1311             return False
1312         i += 6
1313     return True
1314 -----
1315 -----
1315 [HumanEval/31] Attempt 0/3
1316 [HumanEval/31] [] PASS (attempt=0)
1317 =====
1318 -----
1318 [33/164] HumanEval/32
1319 =====
1320 -----
1320 [TASK] HumanEval/32 | entry_point=find_zero
1321 [PROMPT PREVIEW]
1322 import math
1323
1324
1325 def poly(xs: list, x: float):
1326     """
1327     Evaluates polynomial with coefficients xs at point x.
1328     return xs[0] + xs[1] * x + xs[1] * x^2 + .... xs[n] * x^n
1329     """
1330     return sum([coeff * math.pow(x, i) for i, coeff in enumerate(xs
1331     )])
1332
1333 def find_zero(xs: list):
1334     """ xs are coefficients of a polynomial.
1335     find_zero find x such that poly(x) = 0.
1336     find_zero returns only only z...
1337 -----
1338 -----
1338 [INITIAL GENERATED CODE]
1339 def find_zero(xs: list):
1340     """ xs are coefficients of a polynomial.
```

```
1341     find_zero find x such that poly(x) = 0.
1342     find_zero returns only one zero point, even if there are many.
1343     Moreover, find_zero only takes list xs having even number of
1344     coefficients
1345     and largest non zero coefficient as it guarantees
1346     a solution.
1347     >>> round(find_zero([1, 2]), 2) # f(x) = 1 + 2x
1348     -0.5
1349     >>> round(find_zero([-6, 11, -6, 1]), 2) # (x - 1) * (x - 2) * (
1350     x - 3) = -6 + 11x - 6x^2 + x^3
1351     1.0
1352     """
1353     if len(xs) % 2 != 0:
1354         raise ValueError("The number of coefficients must be even.")
1355
1356     def f(x):
1357         return poly(xs, x)
1358
1359     a = -1000
1360     b = 1000
1361
1362     while abs(b - a) > 1e-6:
1363         mid = (a + b) / 2
1364         if f(mid) * f(a) < 0:
1365             b = mid
1366         else:
1367             a = mid
1368     -----
1369     -----
1370 [HumanEval/32] Attempt 0/3
1371 [HumanEval/32] ⚡ FAIL (attempt=0)
1372 [TEST ERROR PREVIEW]
1373 Traceback (most recent call last):
1374   File "C:\Users\JOIN\Downloads\Multi-
1375     Reviewer_Governance_for_Code_Generation\src\evaluator\
1376       humaneval_evaluator.py", line 39, in _worker
1377         ns["check"](candidate)
1378   File "<string>", line 20, in check
1379 AssertionException
1380 -----
1381 -----
1382 [HumanEval/32] [REPAIRED CODE] (after attempt=0)
1383 def find_zero(xs: list):
1384     """ xs are coefficients of a polynomial.
1385     find_zero find x such that poly(x) = 0.
1386     find_zero returns only one zero point, even if there are many.
1387     Moreover, find_zero only takes list xs having even number of
1388     coefficients
1389     and largest non zero coefficient as it guarantees
1390     a solution.
1391     >>> round(find_zero([1, 2]), 2) # f(x) = 1 + 2x
```

```

1387     -0.5
1388     >>> round(find_zero([-6, 11, -6, 1]), 2) # (x - 1) * (x - 2) * (
1389         x - 3) = -6 + 11x - 6x^2 + x^3
1390     1.0
1391     """
1392     if len(xs) % 2 != 0:
1393         raise ValueError("The number of coefficients must be even.")
1394
1395     def poly(coeffs, x):
1396         return sum(c * x**i for i, c in enumerate(reversed(coeffs)))
1397
1398     def f(x):
1399         return poly(xs, x)
1400
1401     a = -1000
1402     b = 1000
1403
1404     while abs(b - a) > 1e-6:
1405         mid = (a + b) / 2
1406         if f(mid) * f(a) < 0:
1407             b = mid
1408         else:
1409             a = mid
1410
1411     return round((a + b) / 2, 2)
1412 -----
1413 [HumanEval/32] Attempt 1/3
1414 [HumanEval/32] ✘ FAIL (attempt=1)
1415 [TEST ERROR PREVIEW]
1416 Traceback (most recent call last):
1417   File "C:\Users\JOIN\Downloads\Multi-
1418     Reviewer_Governance_for_Code_Generation\src\evaluator\
1419       humaneval_evaluator.py", line 39, in _worker
1420         ns["check"](candidate)
1421   File "<string>", line 20, in check
1422 AssertionException
1423 -----
1424 [HumanEval/32] [REPAIRED CODE] (after attempt=1)
1425 def find_zero(xs: list):
1426     """ xs are coefficients of a polynomial.
1427     find_zero find x such that poly(x) = 0.
1428     find_zero returns only one zero point, even if there are many.
1429     Moreover, find_zero only takes list xs having even number of
1430     coefficients
1431     and largest non zero coefficient as it guarantees
1432     a solution.
1433     >>> round(find_zero([1, 2]), 2) # f(x) = 1 + 2x
1434     -0.5
1435     >>> round(find_zero([-6, 11, -6, 1]), 2) # (x - 1) * (x - 2) * (
1436         x - 3) = -6 + 11x - 6x^2 + x^3
1437     1.0

```

```

1433     """
1434     if len(xs) % 2 != 0:
1435         raise ValueError("The number of coefficients must be even.")
1436
1437     def poly(coeffs, x):
1438         return sum(c * x**i for i, c in enumerate(reversed(coeffs)))
1439
1440     def f(x):
1441         return poly(xs, x)
1442
1443     a = -1000
1444     b = 1000
1445
1446     while abs(b - a) > 1e-6:
1447         mid = (a + b) / 2
1448         if f(mid) * f(a) < 0:
1449             b = mid
1450         else:
1451             a = mid
1452
1453     return round((a + b) / 2, 2)
1454 -----
-----
1455 [HumanEval/32] Attempt 2/3
1456 [HumanEval/32] ⚡ FAIL (attempt=2)
1457 [TEST ERROR PREVIEW]
1458 Traceback (most recent call last):
1459   File "C:\Users\JOIN\Downloads\Multi-
      Reviewer_Governance_for_Code_Generation\src\evaluator\
      humaneval_evaluator.py", line 39, in _worker
1460     ns["check"](candidate)
1461   File "<string>", line 20, in check
1462 Assertion
1463 -----
-----
1464 [HumanEval/32] [REPAIRED CODE] (after attempt=2)
1465 def find_zero(xs: list):
1466     """ xs are coefficients of a polynomial.
1467     find_zero find x such that poly(x) = 0.
1468     find_zero returns only one zero point, even if there are many.
1469     Moreover, find_zero only takes list xs having even number of
      coefficients
1470     and largest non zero coefficient as it guarantees
1471     a solution.
1472     >>> round(find_zero([1, 2]), 2) # f(x) = 1 + 2x
1473     -0.5
1474     >>> round(find_zero([-6, 11, -6, 1]), 2) # (x - 1) * (x - 2) * (
      x - 3) = -6 + 11x - 6x^2 + x^3
1475     1.0
1476     """
1477     if len(xs) % 2 != 0:
1478         raise ValueError("The number of coefficients must be even.")
1479

```

File - run

```
1480     def poly(coeffs, x):
1481         return sum(c * x**i for i, c in enumerate(reversed(coeffs)))
1482
1483     def f(x):
1484         return poly(xs, x)
1485
1486     a = -1000
1487     b = 1000
1488
1489     while abs(b - a) > 1e-6:
1490         mid = (a + b) / 2
1491         if f(mid) * f(a) < 0:
1492             b = mid
1493         else:
1494             a = mid
1495
1496     return round((a + b) / 2, 2)
1497 -----
1498 -----
1498 [HumanEval/32] Attempt 3/3
1499 [HumanEval/32] ⚡ FAIL (attempt=3)
1500 [TEST ERROR PREVIEW]
1501 Traceback (most recent call last):
1502   File "C:\Users\JOIN\Downloads\Multi-
1503     Reviewer_Governance_for_Code_Generation\src\evaluator\
1504       humaneval_evaluator.py", line 39, in _worker
1503       ns["check"](candidate)
1504     File "<string>", line 20, in check
1505   AssertionError
1506 -----
1507 -----
1507 [HumanEval/32] ⚡ FINAL FAIL after 3 retries
1508 =====
1509 =====
1509 [34/164] HumanEval/33
1510 =====
1511 =====
1511 [TASK] HumanEval/33 | entry_point=sort_third
1512 [PROMPT PREVIEW]
1513
1514
1515 def sort_third(l: list):
1516     """This function takes a list l and returns a list l' such that
1517     l' is identical to l in the indices that are not divisible by
1518     three, while its values at the indices that are divisible by three
1519     are equal
1520     to the values of the corresponding indices of l, but sorted.
1521     """
1522     >>> sort_third([1, 2, 3])
1523     [1, 2, 3]
1524     >>> sort_third([5, 6, 3, 4, 8, 9, 2])
1525     [2, ...
1526 -----
1527 -----
```

```
1524 [INITIAL GENERATED CODE]
1525 def sort_third(l: list):
1526     indices_divisible_by_three = [i for i in range(len(l)) if i % 3
1527         == 0]
1528     values_at_divisible_indices = sorted([l[i] for i in
1529         indices_divisible_by_three])
1530     result = l[:]
1531     for original_index, new_value in zip(indices_divisible_by_three,
1532         values_at_divisible_indices):
1533         result[original_index] = new_value
1534 -----
1535 -----
1535 [HumanEval/33] Attempt 0/3
1536 [HumanEval/33] [] PASS (attempt=0)
1537 =====
1538 -----
1538 [35/164] HumanEval/34
1539 =====
1540 -----
1540 [TASK] HumanEval/34 | entry_point=unique
1541 [PROMPT PREVIEW]
1542
1543
1544 def unique(l: list):
1545     """Return sorted unique elements in a list
1546     >>> unique([5, 3, 5, 2, 3, 3, 9, 0, 123])
1547     [0, 2, 3, 5, 9, 123]
1548     """
1549 -----
1550 -----
1550 [INITIAL GENERATED CODE]
1551 def unique(l: list):
1552     """Return sorted unique elements in a list
1553     >>> unique([5, 3, 5, 2, 3, 3, 9, 0, 123])
1554     [0, 2, 3, 5, 9, 123]
1555     """
1556     return sorted(set(l))
1557 -----
1558 -----
1558 [HumanEval/34] Attempt 0/3
1559 [HumanEval/34] [] PASS (attempt=0)
1560 =====
1561 -----
1561 [36/164] HumanEval/35
1562 =====
1563 -----
1563 [TASK] HumanEval/35 | entry_point=max_element
1564 [PROMPT PREVIEW]
1565
1566
```

```
1567 def max_element(l: list):
1568     """Return maximum element in the list.
1569     >>> max_element([1, 2, 3])
1570     3
1571     >>> max_element([5, 3, -5, 2, -3, 3, 9, 0, 123, 1, -10])
1572     123
1573     """
1574 -----
1575 -----
1575 [INITIAL GENERATED CODE]
1576 def max_element(l: list):
1577     return max(l)
1578 -----
1579 -----
1579 [HumanEval/35] Attempt 0/3
1580 [HumanEval/35] ✘ PASS (attempt=0)
1581 =====
1581 =====
1582 [37/164] HumanEval/36
1583 =====
1583 =====
1584 [TASK] HumanEval/36 | entry_point=fizz_buzz
1585 [PROMPT PREVIEW]
1586
1587
1588 def fizz_buzz(n: int):
1589     """Return the number of times the digit 7 appears in integers
1590     less than n which are divisible by 11 or 13.
1590     >>> fizz_buzz(50)
1591     0
1592     >>> fizz_buzz(78)
1593     2
1594     >>> fizz_buzz(79)
1595     3
1596     """
1597 -----
1597 -----
1598 [INITIAL GENERATED CODE]
1599 def fizz_buzz(n: int) -> int:
1600     count = 0
1601     for i in range(1, n):
1602         if i % 11 == 0 or i % 13 == 0:
1603             count += str(i).count('7')
1604     return count
1605 -----
1605 -----
1606 [HumanEval/36] Attempt 0/3
1607 [HumanEval/36] ✘ PASS (attempt=0)
1608 =====
1608 =====
1609 [38/164] HumanEval/37
1610 =====
1610 =====
```

```
1611 [TASK] HumanEval/37 | entry_point=sort_even
1612 [PROMPT PREVIEW]
1613
1614
1615 def sort_even(l: list):
1616     """This function takes a list l and returns a list l' such that
1617     l' is identical to l in the odd indices, while its values at
1618     the even indices are equal
1619     to the values of the even indices of l, but sorted.
1620     >>> sort_even([1, 2, 3])
1621     [1, 2, 3]
1622     >>> sort_even([5, 6, 3, 4])
1623     [3, 6, 5, 4]
1624     """
1625 -----
1626 -----
1627 [INITIAL GENERATED CODE]
1628 def sort_even(l: list):
1629     even_indices = [i for i in range(len(l)) if i % 2 == 0]
1630     sorted_evens = sorted([l[i] for i in even_indices])
1631     result = l[:]
1632     for i, value in zip(even_indices, sorted_evens):
1633         result[i] = value
1634     return result
1635 -----
1636 -----
1637 [HumanEval/37] Attempt 0/3
1638 [HumanEval/37] 0 PASS (attempt=0)
1639 =====
1640 =====
1641 =====
1642 =====
1643 [39/164] HumanEval/38
1644 =====
1645 =====
1646 [TASK] HumanEval/38 | entry_point=decode_cyclic
1647 [PROMPT PREVIEW]
1648
1649
1650 def encode_cyclic(s: str):
1651     """
1652     """
1653     returns encoded string by cycling groups of three characters.
1654     """
1655     # split string to groups. Each of length 3.
1656     groups = [s[(3 * i):min((3 * i + 3), len(s))] for i in range((len(s) + 2) // 3)]
1657     # cycle elements in each group. Unless group has fewer elements
1658     # than 3.
1659     groups = [(group[1:] + group[0]) if len(group) == 3 else group
1660               for group in...]
1661 -----
1662 -----
1663 [INITIAL GENERATED CODE]
1664 def decode_cyclic(s: str):
1665     """
```

```
1655     takes as input string encoded with encode_cyclic function.  
1656     Returns decoded string.  
1656     """  
1657     # split string to groups. Each of length 3.  
1658     groups = [s[(3 * i):min((3 * i + 3), len(s))] for i in range((  
1659         len(s) + 2) // 3)]  
1659     # cycle elements in each group back. Unless group has fewer  
1660     # elements than 3.  
1660     groups = [(group[-1] + group[:-1]) if len(group) == 3 else group  
1661         for group in groups]  
1661     return "".join(groups)  
1662 -----  
-----  
1663 [HumanEval/38] Attempt 0/3  
1664 [HumanEval/38] 0 PASS (attempt=0)  
1665 =====  
=====  
1666 [40/164] HumanEval/39  
1667 =====  
=====  
1668 [TASK] HumanEval/39 | entry_point=prime_fib  
1669 [PROMPT PREVIEW]  
1670  
1671  
1672 def prime_fib(n: int):  
1673     """  
1674     prime_fib returns n-th number that is a Fibonacci number and it'  
1674     s also prime.  
1675     >>> prime_fib(1)  
1676     2  
1677     >>> prime_fib(2)  
1678     3  
1679     >>> prime_fib(3)  
1680     5  
1681     >>> prime_fib(4)  
1682     13  
1683     >>> prime_fib(5)  
1684     89  
1685     """  
1686 -----  
-----  
1687 [INITIAL GENERATED CODE]  
1688 from sympy import isprime  
1689  
1690 def prime_fib(n: int) -> int:  
1691     def fib_generator():  
1692         a, b = 0, 1  
1693         while True:  
1694             yield b  
1695             a, b = b, a + b  
1696  
1697         count = 0  
1698         for fib_num in fib_generator():
```

```
1699         if isprime(fib_num):
1700             count += 1
1701             if count == n:
1702                 return fib_num
1703 -----
1704 -----
1704 [HumanEval/39] Attempt 0/3
1705 [HumanEval/39] ✘ FAIL (attempt=0)
1706 [TEST ERROR PREVIEW]
1707 Traceback (most recent call last):
1708   File "C:\Users\JOIN\Downloads\Multi-
Reviewer_Governance_for_Code_Generation\src\evaluator\
humaneval_evaluator.py", line 27, in _worker
1709     exec(prompt + "\n" + completion + "\n", ns, ns)
1710   File "<string>", line 18, in <module>
1711 ModuleNotFoundError: No module named 'sympy'
1712 -----
1713 -----
1713 [HumanEval/39] [REPAIRED CODE] (after attempt=0)
1714 import sympy
1715
1716 def prime_fib(n: int) -> int:
1717     def fib_generator():
1718         a, b = 0, 1
1719         while True:
1720             yield b
1721             a, b = b, a + b
1722
1723         count = 0
1724         for fib_num in fib_generator():
1725             if sympy.isprime(fib_num):
1726                 count += 1
1727                 if count == n:
1728                     return fib_num
1729 -----
1730 -----
1730 [HumanEval/39] Attempt 1/3
1731 [HumanEval/39] ✘ FAIL (attempt=1)
1732 [TEST ERROR PREVIEW]
1733 Traceback (most recent call last):
1734   File "C:\Users\JOIN\Downloads\Multi-
Reviewer_Governance_for_Code_Generation\src\evaluator\
humaneval_evaluator.py", line 27, in _worker
1735     exec(prompt + "\n" + completion + "\n", ns, ns)
1736   File "<string>", line 18, in <module>
1737 ModuleNotFoundError: No module named 'sympy'
1738 -----
1739 -----
1739 [HumanEval/39] [REPAIRED CODE] (after attempt=1)
1740 import sympy
1741
1742 def prime_fib(n: int) -> int:
1743     def fib_generator():
```

File - run

```
1744     a, b = 0, 1
1745     while True:
1746         yield b
1747         a, b = b, a + b
1748
1749     count = 0
1750     for fib_num in fib_generator():
1751         if sympy.isprime(fib_num):
1752             count += 1
1753             if count == n:
1754                 return fib_num
1755 -----
1756 -----
1756 [HumanEval/39] Attempt 2/3
1757 [HumanEval/39] ⚡ FAIL (attempt=2)
1758 [TEST ERROR PREVIEW]
1759 Traceback (most recent call last):
1760   File "C:\Users\JOIN\Downloads\Multi-
1761     Reviewer_Governance_for_Code_Generation\src\evaluator\
1762       humaneval_evaluator.py", line 27, in _worker
1761       exec(prompt + "\n" + completion + "\n", ns, ns)
1762   File "<string>", line 18, in <module>
1763 ModuleNotFoundError: No module named 'sympy'
1764 -----
1765 -----
1765 [HumanEval/39] [REPAIRED CODE] (after attempt=2)
1766 import sympy
1767
1768 def prime_fib(n: int) -> int:
1769     def fib_generator():
1770         a, b = 0, 1
1771         while True:
1772             yield b
1773             a, b = b, a + b
1774
1775         count = 0
1776         for fib_num in fib_generator():
1777             if sympy.isprime(fib_num):
1778                 count += 1
1779                 if count == n:
1780                     return fib_num
1781 -----
1782 -----
1782 [HumanEval/39] Attempt 3/3
1783 [HumanEval/39] ⚡ FAIL (attempt=3)
1784 [TEST ERROR PREVIEW]
1785 Traceback (most recent call last):
1786   File "C:\Users\JOIN\Downloads\Multi-
1787     Reviewer_Governance_for_Code_Generation\src\evaluator\
1788       humaneval_evaluator.py", line 27, in _worker
1787       exec(prompt + "\n" + completion + "\n", ns, ns)
1788   File "<string>", line 18, in <module>
1789 ModuleNotFoundError: No module named 'sympy'
```

```
1790 -----
-----
1791 [HumanEval/39] ⚡ FINAL FAIL after 3 retries
1792 =====
=====
1793 [41/164] HumanEval/40
1794 =====
=====
1795 [TASK] HumanEval/40 | entry_point=triples_sum_to_zero
1796 [PROMPT PREVIEW]
1797
1798
1799 def triples_sum_to_zero(l: list):
1800     """
1801         triples_sum_to_zero takes a list of integers as an input.
1802         it returns True if there are three distinct elements in the list
1803         that
1804             sum to zero, and False otherwise.
1805
1806         >>> triples_sum_to_zero([1, 3, 5, 0])
1807         False
1808         >>> triples_sum_to_zero([1, 3, -2, 1])
1809         True
1810         >>> triples_sum_to_zero([1, 2, 3, 7])
1811         False
1812         >>> triples_sum_to_z...
1813 -----
-----
1813 [INITIAL GENERATED CODE]
1814 def triples_sum_to_zero(l: list):
1815     n = len(l)
1816     for i in range(n-2):
1817         for j in range(i+1, n-1):
1818             for k in range(j+1, n):
1819                 if l[i] + l[j] + l[k] == 0:
1820                     return True
1821     return False
1822 -----
-----
1823 [HumanEval/40] Attempt 0/3
1824 [HumanEval/40] ⚡ PASS (attempt=0)
1825 =====
=====
1826 [42/164] HumanEval/41
1827 =====
=====
1828 [TASK] HumanEval/41 | entry_point=car_race_collision
1829 [PROMPT PREVIEW]
1830
1831
1832 def car_race_collision(n: int):
1833     """
1834         Imagine a road that's a perfectly straight infinitely long line.
```

```
1835     n cars are driving left to right; simultaneously, a different
1836     set of n cars
1836     are driving right to left. The two sets of cars start out
1837     being very far from
1837     each other. All cars move in the same speed. Two cars are said
1838     to collide
1838     when a car that's moving left to right h...
1839 -----
1839 -----
1840 [INITIAL GENERATED CODE]
1841 def car_race_collision(n: int) -> int:
1842     return n * n
1843 -----
1843 -----
1844 [HumanEval/41] Attempt 0/3
1845 [HumanEval/41] ✘ PASS (attempt=0)
1846 =====
1846 =====
1847 [43/164] HumanEval/42
1848 =====
1848 =====
1849 [TASK] HumanEval/42 | entry_point=incr_list
1850 [PROMPT PREVIEW]
1851
1852
1853 def incr_list(l: list):
1854     """Return list with elements incremented by 1.
1855     >>> incr_list([1, 2, 3])
1856     [2, 3, 4]
1857     >>> incr_list([5, 3, 5, 2, 3, 3, 9, 0, 123])
1858     [6, 4, 6, 3, 4, 4, 10, 1, 124]
1859     """
1860 -----
1860 -----
1861 [INITIAL GENERATED CODE]
1862 def incr_list(l: list):
1863     return [x + 1 for x in l]
1864 -----
1864 -----
1865 [HumanEval/42] Attempt 0/3
1866 [HumanEval/42] ✘ PASS (attempt=0)
1867 =====
1867 =====
1868 [44/164] HumanEval/43
1869 =====
1869 =====
1870 [TASK] HumanEval/43 | entry_point=pairs_sum_to_zero
1871 [PROMPT PREVIEW]
1872
1873
1874 def pairs_sum_to_zero(l):
1875     """
1876     pairs_sum_to_zero takes a list of integers as an input.
```

```
1877     it returns True if there are two distinct elements in the list
1878     that sum to zero, and False otherwise.
1879     >>> pairs_sum_to_zero([1, 3, 5, 0])
1880     False
1881     >>> pairs_sum_to_zero([1, 3, -2, 1])
1882     False
1883     >>> pairs_sum_to_zero([1, 2, 3, 7])
1884     False
1885     >>> pairs_sum_to_zero([2, 4, -5, 3, 5,...]
1886 -----
1887 -----
1888 [INITIAL GENERATED CODE]
1889 def pairs_sum_to_zero(l):
1890     seen = set()
1891     for num in l:
1892         if -num in seen:
1893             return True
1894         seen.add(num)
1895     return False
1896 -----
1897 [HumanEval/43] Attempt 0/3
1898 [HumanEval/43] 0 PASS (attempt=0)
1899 =====
1900 =====
1901 [45/164] HumanEval/44
1902 =====
1903 =====
1904
1905 def change_base(x: int, base: int):
1906     """Change numerical base of input number x to base.
1907     return string representation after the conversion.
1908     base numbers are less than 10.
1909     >>> change_base(8, 3)
1910     '22'
1911     >>> change_base(8, 2)
1912     '1000'
1913     >>> change_base(7, 2)
1914     '111'
1915     """
1916 -----
1917 -----
1918 [INITIAL GENERATED CODE]
1919 def change_base(x: int, base: int) -> str:
1920     if x == 0:
1921         return "0"
1922     digits = []
1923     while x:
1924         digits.append(int(x % base))
```

```
1924         x //= base
1925     return ''.join(str(x) for x in digits[::-1])
1926 -----
1927 -----
1927 [HumanEval/44] Attempt 0/3
1928 [HumanEval/44] ✘ PASS (attempt=0)
1929 =====
1930 [46/164] HumanEval/45
1931 =====
1932 [TASK] HumanEval/45 | entry_point=triangle_area
1933 [PROMPT PREVIEW]
1934
1935
1936 def triangle_area(a, h):
1937     """Given length of a side and high return area for a triangle.
1938     >>> triangle_area(5, 3)
1939     7.5
1940     """
1941 -----
1942 [INITIAL GENERATED CODE]
1943 def triangle_area(a, h):
1944     return 0.5 * a * h
1945 -----
1946 [HumanEval/45] Attempt 0/3
1947 [HumanEval/45] ✘ PASS (attempt=0)
1948 =====
1949 [47/164] HumanEval/46
1950 =====
1951 [TASK] HumanEval/46 | entry_point=fib4
1952 [PROMPT PREVIEW]
1953
1954
1955 def fib4(n: int):
1956     """The Fib4 number sequence is a sequence similar to the
1957     Fibonacci sequence that's defined as follows:
1958     fib4(0) -> 0
1959     fib4(1) -> 0
1960     fib4(2) -> 2
1961     fib4(3) -> 0
1962     fib4(n) -> fib4(n-1) + fib4(n-2) + fib4(n-3) + fib4(n-4).
1963     Please write a function to efficiently compute the n-th element
1964     of the fib4 number sequence. Do not use recursion.
1965     >>> fib4(5)
1966     4...
1967 -----
1968 [INITIAL GENERATED CODE]
```

```
1967 def fib4(n: int) -> int:
1968     if n == 0 or n == 1 or n == 3:
1969         return 0
1970     elif n == 2:
1971         return 2
1972
1973     a, b, c, d = 0, 0, 2, 0
1974     for _ in range(4, n + 1):
1975         a, b, c, d = b, c, d, a + b + c + d
1976     return d
1977 -----
1978 -----
1978 [HumanEval/46] Attempt 0/3
1979 [HumanEval/46] ✘ PASS (attempt=0)
1980 =====
1981 [48/164] HumanEval/47
1982 =====
1983 [TASK] HumanEval/47 | entry_point=median
1984 [PROMPT PREVIEW]
1985
1986
1987 def median(l: list):
1988     """Return median of elements in the list l.
1989     >>> median([3, 1, 2, 4, 5])
1990     3
1991     >>> median([-10, 4, 6, 1000, 10, 20])
1992     15.0
1993     """
1994 -----
1995 -----
1995 [INITIAL GENERATED CODE]
1996 def median(l: list):
1997     """Return median of elements in the list l.
1998     >>> median([3, 1, 2, 4, 5])
1999     3
2000     >>> median([-10, 4, 6, 1000, 10, 20])
2001     15.0
2002     """
2003     sorted_l = sorted(l)
2004     n = len(sorted_l)
2005     mid = n // 2
2006     if n % 2 == 1:
2007         return sorted_l[mid]
2008     else:
2009         return (sorted_l[mid - 1] + sorted_l[mid]) / 2
2010 -----
2011 -----
2011 [HumanEval/47] Attempt 0/3
2012 [HumanEval/47] ✘ PASS (attempt=0)
2013 =====
2014 [48/164] HumanEval/48
```

```
2014 [49/164] HumanEval/48
2015 =====
=====
2016 [TASK] HumanEval/48 | entry_point=is_palindrome
2017 [PROMPT PREVIEW]
2018
2019
2020 def is_palindrome(text: str):
2021     """
2022         Checks if given string is a palindrome
2023         >>> is_palindrome('')
2024         True
2025         >>> is_palindrome('aba')
2026         True
2027         >>> is_palindrome('aaaaa')
2028         True
2029         >>> is_palindrome('zbcd')
2030         False
2031     """
2032 -----
2033 -----
2033 [INITIAL GENERATED CODE]
2034 def is_palindrome(text: str) -> bool:
2035     return text == text[::-1]
2036 -----
2037 -----
2037 [HumanEval/48] Attempt 0/3
2038 [HumanEval/48] 0 PASS (attempt=0)
2039 =====
=====
2040 [50/164] HumanEval/49
2041 =====
=====
2042 [TASK] HumanEval/49 | entry_point=modp
2043 [PROMPT PREVIEW]
2044
2045
2046 def modp(n: int, p: int):
2047     """Return  $2^n \bmod p$  (be aware of numerics).
2048     >>> modp(3, 5)
2049     3
2050     >>> modp(1101, 101)
2051     2
2052     >>> modp(0, 101)
2053     1
2054     >>> modp(3, 11)
2055     8
2056     >>> modp(100, 101)
2057     1
2058     """
2059 -----
2060 -----
2060 [INITIAL GENERATED CODE]
```

```
2061 def modp(n: int, p: int) -> int:
2062     return pow(2, n, p)
2063 -----
2064 [HumanEval/49] Attempt 0/3
2065 [HumanEval/49] ✘ PASS (attempt=0)
2066 =====
2067 [51/164] HumanEval/50
2068 =====
2069 [TASK] HumanEval/50 | entry_point=decode_shift
2070 [PROMPT PREVIEW]
2071
2072
2073 def encode_shift(s: str):
2074     """
2075         returns encoded string by shifting every character by 5 in the
2076         alphabet.
2077     """
2078     return "".join([chr(((ord(ch) + 5 - ord("a")) % 26) + ord("a"))
2079                     for ch in s])
2080
2081 def decode_shift(s: str):
2082     """
2083         takes as input string encoded with encode_shift function.
2084         Returns decoded string.
2085     """
2086
2087 [INITIAL GENERATED CODE]
2088 return "".join([chr(((ord(ch) - 5 - ord("a")) % 26) + ord("a"))
2089                     for ch in s])
2090 -----
2091 [HumanEval/50] Attempt 0/3
2092 [HumanEval/50] ✘ FAIL (attempt=0)
2093 [TEST ERROR PREVIEW]
2094 Traceback (most recent call last):
2095   File "C:\Users\JOIN\Downloads\Multi-
2096       Reviewer_Governance_for_Code_Generation\src\evaluator\
2097           humaneval_evaluator.py", line 27, in _worker
2098             exec(prompt + "\n" + completion + "\n", ns, ns)
2099   File "<string>", line 15
2100 SyntaxError: 'return' outside function
2101 -----
2102 [HumanEval/50] [REPAIRED CODE] (after attempt=0)
2103 def rotate_string(s):
2104     return "".join([chr(((ord(ch) - 5 - ord("a")) % 26) + ord("a"))
2105                     for ch in s])
2106 -----
```

```
2100 -----
2101 [HumanEval/50] Attempt 1/3
2102 [HumanEval/50] ⚡ FAIL (attempt=1)
2103 [TEST ERROR PREVIEW]
2104 Traceback (most recent call last):
2105   File "C:\Users\JOIN\Downloads\Multi-
    Reviewer_Governance_for_Code_Generation\src\evaluator\
      humaneval_evaluator.py", line 39, in _worker
2106     ns["check"](candidate)
2107   File "<string>", line 15, in check
2108 AssertionError
2109 -----
2110 -----
2110 [HumanEval/50] [REPAIRED CODE] (after attempt=1)
2111 def rotate_string(s):
2112     return "".join([chr(((ord(ch) - 97 - 5) % 26) + 97) if 'a' <= ch
2113                   <= 'z' else ch for ch in s])
2114 -----
2114 -----
2114 [HumanEval/50] Attempt 2/3
2115 [HumanEval/50] ⚡ FAIL (attempt=2)
2116 [TEST ERROR PREVIEW]
2117 Traceback (most recent call last):
2118   File "C:\Users\JOIN\Downloads\Multi-
    Reviewer_Governance_for_Code_Generation\src\evaluator\
      humaneval_evaluator.py", line 39, in _worker
2119     ns["check"](candidate)
2120   File "<string>", line 15, in check
2121 AssertionError
2122 -----
2122 -----
2123 [HumanEval/50] [REPAIRED CODE] (after attempt=2)
2124 def rotate_string(s):
2125     return "".join([chr(((ord(ch) - 97 - 5) % 26) + 97) if 'a' <= ch
2126                   <= 'z' else chr(((ord(ch) - 65 - 5) % 26) + 65) if 'A' <= ch <= 'Z'
2127                   ' else ch for ch in s])
2128 -----
2128 -----
2127 [HumanEval/50] Attempt 3/3
2128 [HumanEval/50] ⚡ FAIL (attempt=3)
2129 [TEST ERROR PREVIEW]
2130 Traceback (most recent call last):
2131   File "C:\Users\JOIN\Downloads\Multi-
    Reviewer_Governance_for_Code_Generation\src\evaluator\
      humaneval_evaluator.py", line 39, in _worker
2132     ns["check"](candidate)
2133   File "<string>", line 15, in check
2134 AssertionError
2135 -----
2135 -----
2136 [HumanEval/50] ⚡ FINAL FAIL after 3 retries
2137 =====
```

```
2138 [52/164] HumanEval/51
2139 =====
2140 =====
2140 [TASK] HumanEval/51 | entry_point=remove_vowels
2141 [PROMPT PREVIEW]
2142
2143
2144 def remove_vowels(text):
2145     """
2146     remove_vowels is a function that takes string and returns string
2147     without vowels.
2148     >>> remove_vowels('')
2149     ''
2150     >>> remove_vowels("abcdef\nghijklm")
2151     'bcdf\nghijklm'
2152     >>> remove_vowels('abcdef')
2153     'bcdf'
2154     >>> remove_vowels('aaaaaa')
2155     ''
2156     >>> remove_vowels('aaBAA')
2157     'B'
2158     >>> remove_vowels('zbcd')
2159     'zbcd'
2160 -----
2161 -----
2161 [INITIAL GENERATED CODE]
2162 def remove_vowels(text):
2163     vowels = "aeiouAEIOU"
2164     return ''.join([char for char in text if char not in vowels])
2165 -----
2166 -----
2166 [HumanEval/51] Attempt 0/3
2167 [HumanEval/51] 0 PASS (attempt=0)
2168 =====
2169 =====
2169 [53/164] HumanEval/52
2170 =====
2170 =====
2171 [TASK] HumanEval/52 | entry_point=below_threshold
2172 [PROMPT PREVIEW]
2173
2174
2175 def below_threshold(l: list, t: int):
2176     """Return True if all numbers in the list l are below threshold
2177     t.
2178     >>> below_threshold([1, 2, 4, 10], 100)
2179     True
2180     >>> below_threshold([1, 20, 4, 10], 5)
2181     False
2182     """
2183 -----
2183 -----
```

```
2183 [INITIAL GENERATED CODE]
2184 def below_threshold(l: list, t: int) -> bool:
2185     return all(x < t for x in l)
2186 -----
2187 -----
2187 [HumanEval/52] Attempt 0/3
2188 [HumanEval/52] ✘ PASS (attempt=0)
2189 =====
2190 [54/164] HumanEval/53
2191 =====
2192 [TASK] HumanEval/53 | entry_point=add
2193 [PROMPT PREVIEW]
2194
2195
2196 def add(x: int, y: int):
2197     """Add two numbers x and y
2198     >>> add(2, 3)
2199     5
2200     >>> add(5, 7)
2201     12
2202     """
2203 -----
2204 -----
2204 [INITIAL GENERATED CODE]
2205 def add(x: int, y: int) -> int:
2206     return x + y
2207 -----
2208 -----
2208 [HumanEval/53] Attempt 0/3
2209 [HumanEval/53] ✘ PASS (attempt=0)
2210 =====
2211 [55/164] HumanEval/54
2212 =====
2213 [TASK] HumanEval/54 | entry_point=same_chars
2214 [PROMPT PREVIEW]
2215
2216
2217 def same_chars(s0: str, s1: str):
2218     """
2219     Check if two words have the same characters.
2220     >>> same_chars('eabcdzzz', 'ddzzzzzzddabc')
2221     True
2222     >>> same_chars('abcd', 'ddddddabc')
2223     True
2224     >>> same_chars('ddddddabc', 'abcd')
2225     True
2226     >>> same_chars('eabcd', 'dddddabc')
2227     False
2228     >>> same_chars('abcd', 'dddddabce')
```

```
2229     False
2230     >>> same_chars('eabcdzzz', 'dddzzzz...
2231 -----
2232 -----
2232 [INITIAL GENERATED CODE]
2233 def same_chars(s0: str, s1: str) -> bool:
2234     return set(s0) == set(s1)
2235 -----
2236 -----
2236 [HumanEval/54] Attempt 0/3
2237 [HumanEval/54] 0 PASS (attempt=0)
2238 =====
2239 =====
2239 [56/164] HumanEval/55
2240 =====
2241 [TASK] HumanEval/55 | entry_point=fib
2242 [PROMPT PREVIEW]
2243
2244
2245 def fib(n: int):
2246     """Return n-th Fibonacci number.
2247     >>> fib(10)
2248     55
2249     >>> fib(1)
2250     1
2251     >>> fib(8)
2252     21
2253     """
2254 -----
2255 -----
2255 [INITIAL GENERATED CODE]
2256 def fib(n: int) -> int:
2257     if n <= 1:
2258         return n
2259     a, b = 0, 1
2260     for _ in range(2, n + 1):
2261         a, b = b, a + b
2262     return b
2263 -----
2264 -----
2264 [HumanEval/55] Attempt 0/3
2265 [HumanEval/55] 0 PASS (attempt=0)
2266 =====
2267 =====
2267 [57/164] HumanEval/56
2268 =====
2269 [TASK] HumanEval/56 | entry_point=correct_bracketing
2270 [PROMPT PREVIEW]
2271
2272
2273 def correct_bracketing(brackets: str):
```

```
2274     """ brackets is a string of "<" and ">".  
2275     return True if every opening bracket has a corresponding closing  
2276     bracket.  
2277  
2278     >>> correctBracketing("<")  
2279     False  
2280     >>> correctBracketing("<>")  
2281     True  
2282     >>> correctBracketing("<<><>>")  
2283     True  
2284     >>> correctBracketing("><<>")  
2285     False  
2286     """  
2287 -----  
-----  
2287 [INITIAL GENERATED CODE]  
2288 def correctBracketing(brackets: str) -> bool:  
2289     balance = 0  
2290     for bracket in brackets:  
2291         if bracket == '<':  
2292             balance += 1  
2293         elif bracket == '>':  
2294             balance -= 1  
2295         if balance < 0:  
2296             return False  
2297     return balance == 0  
2298 -----  
-----  
2299 [HumanEval/56] Attempt 0/3  
2300 [HumanEval/56] ✘ PASS (attempt=0)  
2301 =====  
=====  
2302 [58/164] HumanEval/57  
2303 =====  
=====  
2304 [TASK] HumanEval/57 | entry_point=monotonic  
2305 [PROMPT PREVIEW]  
2306  
2307  
2308 def monotonic(l: list):  
2309     """Return True if list elements are monotonically increasing or  
     decreasing.  
2310     >>> monotonic([1, 2, 4, 20])  
2311     True  
2312     >>> monotonic([1, 20, 4, 10])  
2313     False  
2314     >>> monotonic([4, 1, 0, -10])  
2315     True  
2316     """  
2317 -----  
-----  
2318 [INITIAL GENERATED CODE]  
2319 def monotonic(l: list):
```

```
2320     return all(x <= y for x, y in zip(l, l[1:])) or all(x >= y for x
, y in zip(l, l[1:]))
2321 -----
-----
2322 [HumanEval/57] Attempt 0/3
2323 [HumanEval/57] ✘ PASS (attempt=0)
2324 =====
=====
2325 [59/164] HumanEval/58
2326 =====
=====
2327 [TASK] HumanEval/58 | entry_point=common
2328 [PROMPT PREVIEW]
2329
2330
2331 def common(l1: list, l2: list):
2332     """Return sorted unique common elements for two lists.
2333     >>> common([1, 4, 3, 34, 653, 2, 5], [5, 7, 1, 5, 9, 653, 121])
2334     [1, 5, 653]
2335     >>> common([5, 3, 2, 8], [3, 2])
2336     [2, 3]
2337
2338     """
2339 -----
-----
2340 [INITIAL GENERATED CODE]
2341 def common(l1: list, l2: list):
2342     """Return sorted unique common elements for two lists.
2343     >>> common([1, 4, 3, 34, 653, 2, 5], [5, 7, 1, 5, 9, 653, 121])
2344     [1, 5, 653]
2345     >>> common([5, 3, 2, 8], [3, 2])
2346     [2, 3]
2347
2348     """
2349     return sorted(set(l1) & set(l2))
2350 -----
-----
2351 [HumanEval/58] Attempt 0/3
2352 [HumanEval/58] ✘ PASS (attempt=0)
2353 =====
=====
2354 [60/164] HumanEval/59
2355 =====
=====
2356 [TASK] HumanEval/59 | entry_point=largest_prime_factor
2357 [PROMPT PREVIEW]
2358
2359
2360 def largest_prime_factor(n: int):
2361     """Return the largest prime factor of n. Assume n > 1 and is not
a prime.
2362     >>> largest_prime_factor(13195)
2363     29
```

```
2364     >>> largest_prime_factor(2048)
2365     2
2366     """
2367 -----
2368 [INITIAL GENERATED CODE]
2369 def largest_prime_factor(n: int) -> int:
2370     factor = 2
2371     while n % factor == 0:
2372         n //= factor
2373     for i in range(3, int(n**0.5) + 1, 2):
2374         while n % i == 0:
2375             n //= i
2376             factor = i
2377     if n > 2:
2378         factor = n
2379     return factor
2380 -----
2381 [HumanEval/59] Attempt 0/3
2382 [HumanEval/59] 0 PASS (attempt=0)
2383 =====
2384 [61/164] HumanEval/60
2385 =====
2386 [TASK] HumanEval/60 | entry_point=sum_to_n
2387 [PROMPT PREVIEW]
2388
2389
2390 def sum_to_n(n: int):
2391     """sum_to_n is a function that sums numbers from 1 to n.
2392     >>> sum_to_n(30)
2393     465
2394     >>> sum_to_n(100)
2395     5050
2396     >>> sum_to_n(5)
2397     15
2398     >>> sum_to_n(10)
2399     55
2400     >>> sum_to_n(1)
2401     1
2402     """
2403 -----
2404 [INITIAL GENERATED CODE]
2405 def sum_to_n(n: int) -> int:
2406     return n * (n + 1) // 2
2407 -----
2408 [HumanEval/60] Attempt 0/3
2409 [HumanEval/60] 0 PASS (attempt=0)
2410 =====
```

```
2410 =====
2411 [62/164] HumanEval/61
2412 =====
2413 [TASK] HumanEval/61 | entry_point=correctBracketing
2414 [PROMPT PREVIEW]
2415
2416
2417 def correctBracketing(brackets: str):
2418     """ brackets is a string of "(" and ")".
2419     return True if every opening bracket has a corresponding closing
2420     bracket.
2421
2422     >>> correctBracketing("(")
2423     False
2424     >>> correctBracketing("()")
2425     True
2426     >>> correctBracketing("((())")
2427     True
2428     >>> correctBracketing(")(()")
2429     False
2430     """
2431 -----
2432 [INITIAL GENERATED CODE]
2433 def correctBracketing(brackets: str) -> bool:
2434     balance = 0
2435     for bracket in brackets:
2436         if bracket == '(':
2437             balance += 1
2438         elif bracket == ')':
2439             balance -= 1
2440             if balance < 0:
2441                 return False
2442     return balance == 0
2443 -----
2444 [HumanEval/61] Attempt 0/3
2445 [HumanEval/61] ✘ PASS (attempt=0)
2446 =====
2447 [63/164] HumanEval/62
2448 =====
2449 [TASK] HumanEval/62 | entry_point=derivative
2450 [PROMPT PREVIEW]
2451
2452 def derivative(xs: list):
2453     """ xs represent coefficients of a polynomial.
2454     xs[0] + xs[1] * x + xs[2] * x^2 + ....
2455     Return derivative of this polynomial in the same form.
2456     >>> derivative([3, 1, 2, 4, 5])
```

```
2457 [1, 4, 12, 20]
2458 >>> derivative([1, 2, 3])
2459 [2, 6]
2460 """
2461 -----
2462 -----
2462 [INITIAL GENERATED CODE]
2463 def derivative(xs: list):
2464     return [i * x for i, x in enumerate(xs)][1:]
2465 -----
2466 -----
2466 [HumanEval/62] Attempt 0/3
2467 [HumanEval/62] ✘ PASS (attempt=0)
2468 =====
2469 =====
2469 [64/164] HumanEval/63
2470 =====
2471 =====
2471 [TASK] HumanEval/63 | entry_point=fibfib
2472 [PROMPT PREVIEW]
2473
2474
2475 def fibfib(n: int):
2476     """The FibFib number sequence is a sequence similar to the
2477     Fibonacci sequence that's defined as follows:
2477     fibfib(0) == 0
2478     fibfib(1) == 0
2479     fibfib(2) == 1
2480     fibfib(n) == fibfib(n-1) + fibfib(n-2) + fibfib(n-3).
2481     Please write a function to efficiently compute the n-th element
2482     of the fibfib number sequence.
2482     >>> fibfib(1)
2483     0
2484     >>> fibfib(5)
2485     4
2486     >>...
2487 -----
2488 -----
2488 [INITIAL GENERATED CODE]
2489 def fibfib(n: int) -> int:
2490     if n == 0 or n == 1:
2491         return 0
2492     elif n == 2:
2493         return 1
2494
2495     a, b, c = 0, 0, 1
2496     for _ in range(3, n + 1):
2497         a, b, c = b, c, a + b + c
2498     return c
2499 -----
2500 -----
2500 [HumanEval/63] Attempt 0/3
2501 [HumanEval/63] ✘ PASS (attempt=0)
```

```
2502 =====
=====
2503 [65/164] HumanEval/64
2504 =====
=====
2505 [TASK] HumanEval/64 | entry_point=vowels_count
2506 [PROMPT PREVIEW]
2507
2508 FIX = """
2509 Add more test cases.
2510 """
2511
2512 def vowels_count(s):
2513     """Write a function vowels_count which takes a string
representing
2514         a word as input and returns the number of vowels in the string.
2515         Vowels in this case are 'a', 'e', 'i', 'o', 'u'. Here, 'y' is
also a
2516         vowel, but only when it is at the end of the given word.
2517
2518     Example:
2519     >>> vowels_count("abcde")
2520     2
2521     >>> vowels_coun...
2522 -----
-----
2523 [INITIAL GENERATED CODE]
2524 def vowels_count(s):
2525     """Write a function vowels_count which takes a string
representing
2526         a word as input and returns the number of vowels in the string.
2527         Vowels in this case are 'a', 'e', 'i', 'o', 'u'. Here, 'y' is
also a
2528         vowel, but only when it is at the end of the given word.
2529
2530     Example:
2531     >>> vowels_count("abcde")
2532     2
2533     >>> vowels_count("ACEDY")
2534     3
2535     """
2536     s = s.lower()
2537     count = sum(1 for char in s if char in 'aeiou')
2538     if s.endswith('y'):
2539         count += 1
2540     return count
2541 -----
-----
2542 [HumanEval/64] Attempt 0/3
2543 [HumanEval/64] 0 PASS (attempt=0)
2544 =====
=====
2545 [66/164] HumanEval/65
```

```
2546 =====
2547 =====
2547 [TASK] HumanEval/65 | entry_point=circular_shift
2548 [PROMPT PREVIEW]
2549
2550 def circular_shift(x, shift):
2551     """Circular shift the digits of the integer x, shift the digits
2552     right by shift
2553     and return the result as a string.
2554     If shift > number of digits, return digits reversed.
2555     >>> circular_shift(12, 1)
2556     "21"
2557     >>> circular_shift(12, 2)
2558     "12"
2559     """
2559 -----
2560 -----
2560 [INITIAL GENERATED CODE]
2561 def circular_shift(x, shift):
2562     """Circular shift the digits of the integer x, shift the digits
2563     right by shift
2564     and return the result as a string.
2565     If shift > number of digits, return digits reversed.
2566     >>> circular_shift(12, 1)
2567     "21"
2568     >>> circular_shift(12, 2)
2569     "12"
2570     """
2570     x_str = str(x)
2571     n = len(x_str)
2572
2573     if shift >= n:
2574         return x_str[::-1]
2575
2576     shift %= n
2577     return x_str[-shift:] + x_str[:-shift]
2578 -----
2578 -----
2579 [HumanEval/65] Attempt 0/3
2580 [HumanEval/65] ⚡ FAIL (attempt=0)
2581 [TEST ERROR PREVIEW]
2582 Traceback (most recent call last):
2583   File "C:\Users\JOIN\Downloads\Multi-
2584     Reviewer_Governance_for_Code_Generation\src\evaluator\
2585       humaneval_evaluator.py", line 39, in _worker
2586       ns["check"](candidate)
2587       File "<string>", line 5, in check
2588     AssertionError
2587 -----
2587 -----
2588 [HumanEval/65] [REPAIRED CODE] (after attempt=0)
2589 def circular_shift(x, shift):
2590     """Circular shift the digits of the integer x, shift the digits
```

```
2590 right by shift
2591     and return the result as a string.
2592     If shift > number of digits, return digits reversed.
2593     >>> circular_shift(12, 1)
2594     "21"
2595     >>> circular_shift(12, 2)
2596     "12"
2597     """
2598     x_str = str(x)
2599     n = len(x_str)
2600
2601     if shift >= n:
2602         return x_str[::-1]
2603
2604     shift %= n
2605     return x_str[-shift:] + x_str[:-shift]
2606 -----
-----
2607 [HumanEval/65] Attempt 1/3
2608 [HumanEval/65] ✘ FAIL (attempt=1)
2609 [TEST ERROR PREVIEW]
2610 Traceback (most recent call last):
2611   File "C:\Users\JOIN\Downloads\Multi-
    Reviewer_Governance_for_Code_Generation\src\evaluator\
      humaneval_evaluator.py", line 39, in _worker
2612     ns["check"](candidate)
2613   File "<string>", line 5, in check
2614 AssertionError
2615 -----
-----
2616 [HumanEval/65] [REPAIRED CODE] (after attempt=1)
2617 def circular_shift(x, shift):
2618     """Circular shift the digits of the integer x, shift the digits
        right by shift
2619     and return the result as a string.
2620     If shift > number of digits, return digits reversed.
2621     >>> circular_shift(12, 1)
2622     "21"
2623     >>> circular_shift(12, 2)
2624     "12"
2625     """
2626     x_str = str(x)
2627     n = len(x_str)
2628
2629     if shift >= n:
2630         return x_str[::-1]
2631
2632     shift %= n
2633     return x_str[-shift:] + x_str[:-shift]
2634 -----
-----
2635 [HumanEval/65] Attempt 2/3
2636 [HumanEval/65] ✘ FAIL (attempt=2)
```

```
2637 [TEST ERROR PREVIEW]
2638 Traceback (most recent call last):
2639   File "C:\Users\JOIN\Downloads\Multi-
    Reviewer_Governance_for_Code_Generation\src\evaluator\
      humaneval_evaluator.py", line 39, in _worker
2640     ns["check"](candidate)
2641   File "<string>", line 5, in check
2642 AssertionError
2643 -----
-----
2644 [HumanEval/65] [REPAIRED CODE] (after attempt=2)
2645 def circular_shift(x, shift):
2646     """Circular shift the digits of the integer x, shift the digits
    right by shift
2647     and return the result as a string.
2648     If shift > number of digits, return digits reversed.
2649     >>> circular_shift(12, 1)
2650     "21"
2651     >>> circular_shift(12, 2)
2652     "12"
2653     """
2654     x_str = str(x)
2655     n = len(x_str)
2656
2657     if shift >= n:
2658         return x_str[::-1]
2659
2660     shift %= n
2661     return x_str[-shift:] + x_str[:-shift]
2662 -----
-----
2663 [HumanEval/65] Attempt 3/3
2664 [HumanEval/65] ⚡ FAIL (attempt=3)
2665 [TEST ERROR PREVIEW]
2666 Traceback (most recent call last):
2667   File "C:\Users\JOIN\Downloads\Multi-
    Reviewer_Governance_for_Code_Generation\src\evaluator\
      humaneval_evaluator.py", line 39, in _worker
2668     ns["check"](candidate)
2669   File "<string>", line 5, in check
2670 AssertionError
2671 -----
-----
2672 [HumanEval/65] ⚡ FINAL FAIL after 3 retries
2673 =====
=====
2674 [67/164] HumanEval/66
2675 =====
=====
2676 [TASK] HumanEval/66 | entry_point=digitSum
2677 [PROMPT PREVIEW]
2678
2679 def digitSum(s):
```

```
2680     """Task
2681     Write a function that takes a string as input and returns the
2682     sum of the upper characters only'
2683     ASCII codes.
2684
2685     Examples:
2686         digitSum("") => 0
2687         digitSum("abAB") => 131
2688         digitSum("abcCd") => 67
2689         digitSum("helloE") => 69
2690         digitSum("woArBld") => 131
2691         digitSum("aAaaaXa") => 153
2692     """
2693 -----
2694 [INITIAL GENERATED CODE]
2695 def digitSum(s):
2696     return sum(ord(c) for c in s if c.isupper())
2697 -----
2698 [HumanEval/66] Attempt 0/3
2699 [HumanEval/66] 0 PASS (attempt=0)
2700 =====
2701 =====
2702 [68/164] HumanEval/67
2703 =====
2704 =====
2705 [TASK] HumanEval/67 | entry_point=fruit_distribution
2706 [PROMPT PREVIEW]
2707
2708 def fruit_distribution(s,n):
2709     """
2710     In this task, you will be given a string that represents a
2711     number of apples and oranges
2712     that are distributed in a basket of fruit this basket contains
2713     apples, oranges, and mango fruits. Given the string that
2714     represents the total number of
2715     the oranges and apples and an integer that represent the total
2716     number of the fruits
2717     in the basket r...
2718
2719 -----
2720 [INITIAL GENERATED CODE]
2721 def fruit_distribution(s, n):
2722     parts = s.split()
2723     apples = int(parts[0])
2724     oranges = int(parts[3])
2725     return n - apples - oranges
2726 -----
2727 [HumanEval/67] Attempt 0/3
2728 [HumanEval/67] 0 PASS (attempt=0)
2729 =====
```

```
2722 =====
2723 [69/164] HumanEval/68
2724 =====
2725 [TASK] HumanEval/68 | entry_point=pluck
2726 [PROMPT PREVIEW]
2727
2728 def pluck(arr):
2729     """
2730     "Given an array representing a branch of a tree that has non-
2731     negative integer nodes
2732     your task is to pluck one of the nodes and return it.
2733     The plucked node should be the node with the smallest even value
2734     .
2735     If multiple nodes with the same smallest even value are found
2736     return the node that has smallest index.
2737
2738     The plucked node should be returned in a list,..."
```

---

```
2739 -----
```

```
2740 [INITIAL GENERATED CODE]
```

```
2741 def pluck(arr):
2742     if not arr:
2743         return []
2744
2745     min_val = float('inf')
2746     min_index = -1
2747
2748     for i, val in enumerate(arr):
2749         if val % 2 == 0 and val < min_val:
2750             min_val = val
2751             min_index = i
2752
2753     if min_index == -1:
2754         return []
2755
2756     return [min_val, min_index]
```

---

```
2757 -----
```

```
2758 [HumanEval/68] Attempt 0/3
2759 [HumanEval/68] ✘ PASS (attempt=0)
2760 =====
```

---

```
2761 =====
```

```
2762 [70/164] HumanEval/69
2763 =====
```

---

```
2764 =====
```

```
2765 [TASK] HumanEval/69 | entry_point=search
2766 [PROMPT PREVIEW]
2767
2768 def search(lst):
2769     '''
2770     You are given a non-empty list of positive integers. Return the
2771     greatest integer that is greater than
```

```
2766     zero, and has a frequency greater than or equal to the value of
the integer itself.
2767     The frequency of an integer is the number of times it appears in
the list.
2768     If no such a value exist, return -1.
2769     Examples:
2770         search([4, 1, 2, 2, 3, 1]) == 2...
2771 -----
-----
2772 [INITIAL GENERATED CODE]
2773 def search(lst):
2774     from collections import Counter
2775
2776     freq = Counter(lst)
2777     for num in sorted(freq.keys(), reverse=True):
2778         if freq[num] >= num:
2779             return num
2780     return -1
2781 -----
-----
2782 [HumanEval/69] Attempt 0/3
2783 [HumanEval/69] 0 PASS (attempt=0)
2784 =====
=====
2785 [71/164] HumanEval/70
2786 =====
=====
2787 [TASK] HumanEval/70 | entry_point=strange_sort_list
2788 [PROMPT PREVIEW]
2789
2790 def strange_sort_list(lst):
2791     '''
2792     Given list of integers, return list in strange order.
2793     Strange sorting, is when you start with the minimum value,
2794     then maximum of the remaining integers, then minimum and so on.
2795
2796     Examples:
2797     strange_sort_list([1, 2, 3, 4]) == [1, 4, 2, 3]
2798     strange_sort_list([5, 5, 5, 5]) == [5, 5, 5, 5]
2799     strange_sort_list([]) == []
2800     '''
2801 -----
-----
2802 [INITIAL GENERATED CODE]
2803 def strange_sort_list(lst):
2804     sorted_lst = sorted(lst)
2805     result = []
2806     while sorted_lst:
2807         if sorted_lst:
2808             result.append(sorted_lst.pop(0))
2809         if sorted_lst:
2810             result.append(sorted_lst.pop())
2811     return result
```

```
2812 -----
-----
2813 [HumanEval/70] Attempt 0/3
2814 [HumanEval/70] ✘ PASS (attempt=0)
2815 =====
=====
2816 [72/164] HumanEval/71
2817 =====
=====
2818 [TASK] HumanEval/71 | entry_point=triangle_area
2819 [PROMPT PREVIEW]
2820
2821 def triangle_area(a, b, c):
2822     '''
2823         Given the lengths of the three sides of a triangle. Return the
2824         area of
2825             the triangle rounded to 2 decimal points if the three sides form
2826             a valid triangle.
2827             Otherwise return -1
2828             Three sides make a valid triangle when the sum of any two sides
2829             is greater
2830                 than the third side.
2831             Example:
2832                 triangle_area(3, 4, 5) == 6.00
2833                 triangle_area(1, 2, ...
2834 -----
-----
2835 [INITIAL GENERATED CODE]
2836 import math
2837
2838 def triangle_area(a, b, c):
2839     if a + b > c and a + c > b and b + c > a:
2840         s = (a + b + c) / 2
2841         area = math.sqrt(s * (s - a) * (s - b) * (s - c))
2842         return round(area, 2)
2843     else:
2844         return -1
2845 -----
-----
2846 [HumanEval/71] Attempt 0/3
2847 [HumanEval/71] ✘ PASS (attempt=0)
2848 =====
=====
2849 [73/164] HumanEval/72
2850 =====
=====
2851 [TASK] HumanEval/72 | entry_point=will_it_fly
2852 [PROMPT PREVIEW]
2853
2854 def will_it_fly(q,w):
2855     '''
2856         Write a function that returns True if the object q will fly, and
2857         False otherwise.
```

```
2854     The object q will fly if it's balanced (it is a palindromic list
2855     ) and the sum of its elements is less than or equal the maximum
2856     possible weight w.
2857
2858     Example:
2859     will_it_fly([1, 2], 5) → False
2860     # 1+2 is less than the maximum possible weight, but it's
2861     unbalanced.
2862
2863     will...
2864
2865 -----
2866 [INITIAL GENERATED CODE]
2867 def will_it_fly(q, w):
2868     return q == q[::-1] and sum(q) <= w
2869
2870 -----
2871 [HumanEval/72] Attempt 0/3
2872 [HumanEval/72] 0 PASS (attempt=0)
2873 =====
2874 =====
2875 [74/164] HumanEval/73
2876 =====
2877 =====
2878 [TASK] HumanEval/73 | entry_point=smallest_change
2879 [PROMPT PREVIEW]
2880
2881 def smallest_change(arr):
2882     """
2883     Given an array arr of integers, find the minimum number of
2884     elements that
2885     need to be changed to make the array palindromic. A palindromic
2886     array is an array that
2887     is read the same backwards and forwards. In one change, you can
2888     change one element to any other element.
2889
2890     For example:
2891     smallest_change([1,2,3,5,4,7,9,6]) == 4
2892     smallest_change([1, 2, ...
2893
2894 -----
2895 [INITIAL GENERATED CODE]
2896 def smallest_change(arr):
2897     left = 0
2898     right = len(arr) - 1
2899     changes = 0
2900
2901     while left < right:
2902         if arr[left] != arr[right]:
2903             changes += 1
2904         left += 1
2905         right -= 1
2906
2907
2908
2909
2910
2911
2912
2913
2914
2915
2916
2917
2918
2919
2920
2921
2922
2923
2924
2925
2926
2927
2928
2929
2930
2931
2932
2933
2934
2935
2936
2937
2938
2939
2940
2941
2942
2943
2944
2945
2946
2947
2948
2949
2950
2951
2952
2953
2954
2955
2956
2957
2958
2959
2960
2961
2962
2963
2964
2965
2966
2967
2968
2969
2970
2971
2972
2973
2974
2975
2976
2977
2978
2979
2980
2981
2982
2983
2984
2985
2986
2987
2988
2989
2990
2991
2992
2993
2994
2995
2996
2997
2998
2999
3000
3001
3002
3003
3004
3005
3006
3007
3008
3009
3010
3011
3012
3013
3014
3015
3016
3017
3018
3019
3020
3021
3022
3023
3024
3025
3026
3027
3028
3029
3030
3031
3032
3033
3034
3035
3036
3037
3038
3039
3040
3041
3042
3043
3044
3045
3046
3047
3048
3049
3050
3051
3052
3053
3054
3055
3056
3057
3058
3059
3060
3061
3062
3063
3064
3065
3066
3067
3068
3069
3070
3071
3072
3073
3074
3075
3076
3077
3078
3079
3080
3081
3082
3083
3084
3085
3086
3087
3088
3089
3090
3091
3092
3093
3094
3095
3096
3097
3098
3099
3100
3101
3102
3103
3104
3105
3106
3107
3108
3109
3110
3111
3112
3113
3114
3115
3116
3117
3118
3119
3120
3121
3122
3123
3124
3125
3126
3127
3128
3129
3130
3131
3132
3133
3134
3135
3136
3137
3138
3139
3140
3141
3142
3143
3144
3145
3146
3147
3148
3149
3150
3151
3152
3153
3154
3155
3156
3157
3158
3159
3160
3161
3162
3163
3164
3165
3166
3167
3168
3169
3170
3171
3172
3173
3174
3175
3176
3177
3178
3179
3180
3181
3182
3183
3184
3185
3186
3187
3188
3189
3190
3191
3192
3193
3194
3195
3196
3197
3198
3199
3200
3201
3202
3203
3204
3205
3206
3207
3208
3209
3210
3211
3212
3213
3214
3215
3216
3217
3218
3219
3220
3221
3222
3223
3224
3225
3226
3227
3228
3229
3230
3231
3232
3233
3234
3235
3236
3237
3238
3239
3240
3241
3242
3243
3244
3245
3246
3247
3248
3249
3250
3251
3252
3253
3254
3255
3256
3257
3258
3259
3260
3261
3262
3263
3264
3265
3266
3267
3268
3269
3270
3271
3272
3273
3274
3275
3276
3277
3278
3279
3280
3281
3282
3283
3284
3285
3286
3287
3288
3289
3290
3291
3292
3293
3294
3295
3296
3297
3298
3299
3300
3301
3302
3303
3304
3305
3306
3307
3308
3309
3310
3311
3312
3313
3314
3315
3316
3317
3318
3319
3320
3321
3322
3323
3324
3325
3326
3327
3328
3329
3330
3331
3332
3333
3334
3335
3336
3337
3338
3339
3340
3341
3342
3343
3344
3345
3346
3347
3348
3349
3350
3351
3352
3353
3354
3355
3356
3357
3358
3359
3360
3361
3362
3363
3364
3365
3366
3367
3368
3369
3370
3371
3372
3373
3374
3375
3376
3377
3378
3379
3380
3381
3382
3383
3384
3385
3386
3387
3388
3389
3390
3391
3392
3393
3394
3395
3396
3397
3398
3399
3400
3401
3402
3403
3404
3405
3406
3407
3408
3409
3410
3411
3412
3413
3414
3415
3416
3417
3418
3419
3420
3421
3422
3423
3424
3425
3426
3427
3428
3429
3430
3431
3432
3433
3434
3435
3436
3437
3438
3439
3440
3441
3442
3443
3444
3445
3446
3447
3448
3449
3450
3451
3452
3453
3454
3455
3456
3457
3458
3459
3460
3461
3462
3463
3464
3465
3466
3467
3468
3469
3470
3471
3472
3473
3474
3475
3476
3477
3478
3479
3480
3481
3482
3483
3484
3485
3486
3487
3488
3489
3490
3491
3492
3493
3494
3495
3496
3497
3498
3499
3500
3501
3502
3503
3504
3505
3506
3507
3508
3509
3510
3511
3512
3513
3514
3515
3516
3517
3518
3519
3520
3521
3522
3523
3524
3525
3526
3527
3528
3529
3530
3531
3532
3533
3534
3535
3536
3537
3538
3539
3540
3541
3542
3543
3544
3545
3546
3547
3548
3549
3550
3551
3552
3553
3554
3555
3556
3557
3558
3559
3560
3561
3562
3563
3564
3565
3566
3567
3568
3569
3570
3571
3572
3573
3574
3575
3576
3577
3578
3579
3580
3581
3582
3583
3584
3585
3586
3587
3588
3589
3590
3591
3592
3593
3594
3595
3596
3597
3598
3599
3600
3601
3602
3603
3604
3605
3606
3607
3608
3609
3610
3611
3612
3613
3614
3615
3616
3617
3618
3619
3620
3621
3622
3623
3624
3625
3626
3627
3628
3629
3630
3631
3632
3633
3634
3635
3636
3637
3638
3639
3640
3641
3642
3643
3644
3645
3646
3647
3648
3649
3650
3651
3652
3653
3654
3655
3656
3657
3658
3659
3660
3661
3662
3663
3664
3665
3666
3667
3668
3669
3670
3671
3672
3673
3674
3675
3676
3677
3678
3679
3680
3681
3682
3683
3684
3685
3686
3687
3688
3689
3690
3691
3692
3693
3694
3695
3696
3697
3698
3699
3700
3701
3702
3703
3704
3705
3706
3707
3708
3709
3710
3711
3712
3713
3714
3715
3716
3717
3718
3719
3720
3721
3722
3723
3724
3725
3726
3727
3728
3729
3730
3731
3732
3733
3734
3735
3736
3737
3738
3739
3740
3741
3742
3743
3744
3745
3746
3747
3748
3749
3750
3751
3752
3753
3754
3755
3756
3757
3758
3759
3760
3761
3762
3763
3764
3765
3766
3767
3768
3769
3770
3771
3772
3773
3774
3775
3776
3777
3778
3779
3780
3781
3782
3783
3784
3785
3786
3787
3788
3789
3790
3791
3792
3793
3794
3795
3796
3797
3798
3799
3800
3801
3802
3803
3804
3805
3806
3807
3808
3809
3810
3811
3812
3813
3814
3815
3816
3817
3818
3819
3820
3821
3822
3823
3824
3825
3826
3827
3828
3829
3830
3831
3832
3833
3834
3835
3836
3837
3838
3839
3840
3841
3842
3843
3844
3845
3846
3847
3848
3849
3850
3851
3852
3853
3854
3855
3856
3857
3858
3859
3860
3861
3862
3863
3864
3865
3866
3867
3868
3869
3870
3871
3872
3873
3874
3875
3876
3877
3878
3879
3880
3881
3882
3883
3884
3885
3886
3887
3888
3889
3890
3891
3892
3893
3894
3895
3896
3897
3898
3899
3900
3901
3902
3903
3904
3905
3906
3907
3908
3909
3910
3911
3912
3913
3914
3915
3916
3917
3918
3919
3920
3921
3922
3923
3924
3925
3926
3927
3928
3929
3930
3931
3932
3933
3934
3935
3936
3937
3938
3939
3940
3941
3942
3943
3944
3945
3946
3947
3948
3949
3950
3951
3952
3953
3954
3955
3956
3957
3958
3959
3960
3961
3962
3963
3964
3965
3966
3967
3968
3969
3970
3971
3972
3973
3974
3975
3976
3977
3978
3979
3980
3981
3982
3983
3984
3985
3986
3987
3988
3989
3990
3991
3992
3993
3994
3995
3996
3997
3998
3999
3999
4000
```

```
2896     return changes
2897 -----
2898 -----
2898 [HumanEval/73] Attempt 0/3
2899 [HumanEval/73] 0 PASS (attempt=0)
2900 =====
2901 -----
2901 [75/164] HumanEval/74
2902 =====
2902 -----
2903 [TASK] HumanEval/74 | entry_point=total_match
2904 [PROMPT PREVIEW]
2905
2906 def total_match(lst1, lst2):
2907     """
2908         Write a function that accepts two lists of strings and returns
2909         the list that has
2910         total number of chars in the all strings of the list less than
2911         the other list.
2912
2913     if the two lists have the same number of chars, return the first
2914     list.
2915
2916     Examples
2917     total_match([], []) → []
2918     total_match(['hi', 'admin'], ['hI', 'Hi']) → ['hI', 'Hi']
2919     total_ma...
2920 -----
2921 -----
2922 [INITIAL GENERATED CODE]
2923 def total_match(lst1, lst2):
2924     len_lst1 = sum(len(s) for s in lst1)
2925     len_lst2 = sum(len(s) for s in lst2)
2926     return lst1 if len_lst1 <= len_lst2 else lst2
2927 -----
2928 -----
2929 [HumanEval/74] Attempt 0/3
2930 [HumanEval/74] 0 PASS (attempt=0)
2931 =====
2932 -----
2932 [76/164] HumanEval/75
2933 =====
2933 -----
2934 [TASK] HumanEval/75 | entry_point=is_multiply_prime
2935 [PROMPT PREVIEW]
2936
2937 def is_multiply_prime(a):
2938     """Write a function that returns true if the given number is the
2939     multiplication of 3 prime numbers
2940     and false otherwise.
2941     Knowing that (a) is less then 100.
2942     Example:
2943     is_multiply_prime(30) == True
```

```
2938     30 = 2 * 3 * 5
2939     """
2940 -----
2941 [INITIAL GENERATED CODE]
2942 def is_multiply_prime(a):
2943     def is_prime(n):
2944         if n <= 1:
2945             return False
2946         for i in range(2, int(n**0.5) + 1):
2947             if n % i == 0:
2948                 return False
2949         return True
2950
2951     primes = [i for i in range(2, 100) if is_prime(i)]
2952
2953     for i in range(len(primes)):
2954         for j in range(i, len(primes)):
2955             for k in range(j, len(primes)):
2956                 if primes[i] * primes[j] * primes[k] == a:
2957                     return True
2958     return False
2959 -----
2960 [HumanEval/75] Attempt 0/3
2961 [HumanEval/75] [] PASS (attempt=0)
2962 =====
2963 [77/164] HumanEval/76
2964 =====
2965 [TASK] HumanEval/76 | entry_point=is_simple_power
2966 [PROMPT PREVIEW]
2967
2968 def is_simple_power(x, n):
2969     """Your task is to write a function that returns true if a
2970     number x is a simple
2971     power of n and false in other cases.
2972     x is a simple power of n if n**int=x
2973     For example:
2974     is_simple_power(1, 4) => true
2975     is_simple_power(2, 2) => true
2976     is_simple_power(8, 2) => true
2977     is_simple_power(3, 2) => false
2978     is_simple_pow...
2979 -----
2980 [INITIAL GENERATED CODE]
2981 def is_simple_power(x, n):
2982     if n == 1:
2983         return x == 1
2984     power = 1
```

```
2985     while power < x:
2986         power *= n
2987     return power == x
2988 -----
2989 [HumanEval/76] Attempt 0/3
2990 [HumanEval/76] ⚡ PASS (attempt=0)
2991 =====
2992 [78/164] HumanEval/77
2993 =====
2994 [TASK] HumanEval/77 | entry_point=iscube
2995 [PROMPT PREVIEW]
2996
2997 def iscube(a):
2998     """
2999     Write a function that takes an integer a and returns True
3000     if this integer is a cube of some integer number.
3001     Note: you may assume the input is always valid.
3002     Examples:
3003     iscube(1) ==> True
3004     iscube(2) ==> False
3005     iscube(-1) ==> True
3006     iscube(64) ==> True
3007     iscube(0) ==> True
3008     iscube(180) ==> False
3009     """
3010 -----
3011 [INITIAL GENERATED CODE]
3012 def iscube(a):
3013     return round(a ** (1/3)) ** 3 == a
3014 -----
3015 [HumanEval/77] Attempt 0/3
3016 [HumanEval/77] ⚡ FAIL (attempt=0)
3017 [TEST ERROR PREVIEW]
3018 Traceback (most recent call last):
3019   File "C:\Users\JOIN\Downloads\Multi-
3020     Reviewer_Governance_for_Code_Generation\src\evaluator\
3021       humaneval_evaluator.py", line 39, in _worker
3022         ns["check"](candidate)
3023   File "<string>", line 6, in check
3024   File "<string>", line 17, in iscube
3025 TypeError: type complex doesn't define __round__ method
3026 -----
3027 [HumanEval/77] [REPAIRED CODE] (after attempt=0)
3028 def iscube(a):
3029     if isinstance(a, complex):
3030         return False
3031     return round(a ** (1/3)) ** 3 == a
```

```
3030 -----
-----
3031 [HumanEval/77] Attempt 1/3
3032 [HumanEval/77] ⚡ FAIL (attempt=1)
3033 [TEST ERROR PREVIEW]
3034 Traceback (most recent call last):
3035   File "C:\Users\JOIN\Downloads\Multi-
Reviewer_Governance_for_Code_Generation\src\evaluator\
humaneval_evaluator.py", line 39, in _worker
3036     ns["check"](candidate)
3037   File "<string>", line 6, in check
3038   File "<string>", line 19, in iscube
3039 TypeError: type complex doesn't define __round__ method
3040 -----
-----
3041 [HumanEval/77] [REPAIRED CODE] (after attempt=1)
3042 def iscube(a):
3043     if isinstance(a, complex):
3044         return False
3045     return round(abs(a) ** (1/3)) ** 3 == abs(a)
3046 -----
-----
3047 [HumanEval/77] Attempt 2/3
3048 [HumanEval/77] ⚡ PASS (attempt=2)
3049 =====
=====
3050 [79/164] HumanEval/78
3051 =====
=====
3052 [TASK] HumanEval/78 | entry_point=hex_key
3053 [PROMPT PREVIEW]
3054
3055 def hex_key(num):
3056     """You have been tasked to write a function that receives
3057     a hexadecimal number as a string and counts the number of
3058     hexadecimal
3059     digits that are primes (prime number, or a prime, is a natural
3060     number
3061     greater than 1 that is not a product of two smaller natural
3062     numbers).
3063     Hexadecimal digits are 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D
3064     , E, F.
3065     Prime numbers a...
3066
3067 -----
-----
3068 [INITIAL GENERATED CODE]
3069 def hex_key(num):
3070     prime_hex_digits = {'2', '3', '5', '7', 'B', 'D'}
3071     return sum(1 for digit in num if digit in prime_hex_digits)
3072 -----
-----
3073 [HumanEval/78] Attempt 0/3
3074 [HumanEval/78] ⚡ PASS (attempt=0)
```

```
3070 =====
=====
3071 [80/164] HumanEval/79
3072 =====
=====
3073 [TASK] HumanEval/79 | entry_point=decimal_to_binary
3074 [PROMPT PREVIEW]
3075
3076 def decimal_to_binary(decimal):
3077     """You will be given a number in decimal form and your task is
3078     to convert it to
3079     binary format. The function should return a string, with each
3080     character representing a binary
3081     number. Each character in the string will be '0' or '1'.
3082
3083     There will be an extra couple of characters 'db' at the
3084     beginning and at the end of the string.
3085     The extra character...
3086 -----
3087 -----
3088 [INITIAL GENERATED CODE]
3089 def decimal_to_binary(decimal):
3090     binary = bin(decimal)[2:]
3091     return f"db{binary}db"
3092 -----
3093 -----
3094 [HumanEval/79] Attempt 0/3
3095 [HumanEval/79] 0 PASS (attempt=0)
3096 =====
=====
3097 [81/164] HumanEval/80
3098 =====
=====
3099 [TASK] HumanEval/80 | entry_point=is_happy
3100 [PROMPT PREVIEW]
3101
3102 def is_happy(s):
3103     """You are given a string s.
3104     Your task is to check if the string is happy or not.
3105     A string is happy if its length is at least 3 and every 3
3106     consecutive letters are distinct
3107     For example:
3108     is_happy(a) => False
3109     is_happy(aa) => False
3110     is_happy(abcd) => True
3111     is_happy(aabb) => False
3112     is_happy(adb) => True
3113     is_happy(xyy) => False
3114     """
3115 -----
3116 -----
3117 [INITIAL GENERATED CODE]
3118 def is_happy(s):
```

```
3112     if len(s) < 3:
3113         return False
3114     for i in range(len(s) - 2):
3115         if s[i] == s[i+1] or s[i] == s[i+2] or s[i+1] == s[i+2]:
3116             return False
3117     return True
3118 -----
-----
3119 [HumanEval/80] Attempt 0/3
3120 [HumanEval/80] ✘ PASS (attempt=0)
3121 =====
=====
3122 [82/164] HumanEval/81
3123 =====
=====
3124 [TASK] HumanEval/81 | entry_point=numerical_letter_grade
3125 [PROMPT PREVIEW]
3126
3127 def numerical_letter_grade(grades):
3128     """It is the last week of the semester and the teacher has to
3129     give the grades
3130     to students. The teacher has been making her own algorithm for
3131     grading.
3132     The only problem is, she has lost the code she used for grading.
3133     She has given you a list of GPAs for some students and you have
3134     to write
3135     a function that can output a list of letter grades usi...
3136 -----
3137 -----
3138 [INITIAL GENERATED CODE]
3139 def numerical_letter_grade(grades):
3140     letter_grades = []
3141     for gpa in grades:
3142         if gpa == 4.0:
3143             letter_grades.append('A+')
3144         elif gpa > 3.7:
3145             letter_grades.append('A')
3146         elif gpa > 3.3:
3147             letter_grades.append('A-')
3148         elif gpa > 3.0:
3149             letter_grades.append('B+')
3150         elif gpa > 2.7:
3151             letter_grades.append('B')
3152         elif gpa > 2.3:
3153             letter_grades.append('B-')
3154         elif gpa > 2.0:
3155             letter_grades.append('C+')
3156         elif gpa > 1.7:
3157             letter_grades.append('C')
```

```
3158     elif gpa > 0.7:
3159         letter_grades.append('D')
3160     elif gpa > 0.0:
3161         letter_grades.append('D-')
3162     else:
3163         letter_grades.append('E')
3164 return letter_grades
3165 -----
3166 -----
3166 [HumanEval/81] Attempt 0/3
3167 [HumanEval/81] Ø PASS (attempt=0)
3168 =====
3169 =====
3169 [83/164] HumanEval/82
3170 =====
3170 =====
3171 [TASK] HumanEval/82 | entry_point=prime_length
3172 [PROMPT PREVIEW]
3173
3174 def prime_length(string):
3175     """Write a function that takes a string and returns True if the
3176     string
3176     length is a prime number or False otherwise
3177     Examples
3178     prime_length('Hello') == True
3179     prime_length('abcdcba') == True
3180     prime_length('kittens') == True
3181     prime_length('orange') == False
3182     """
3183 -----
3183 -----
3184 [INITIAL GENERATED CODE]
3185 def prime_length(string):
3186     length = len(string)
3187     if length < 2:
3188         return False
3189     for i in range(2, int(length ** 0.5) + 1):
3190         if length % i == 0:
3191             return False
3192     return True
3193 -----
3193 -----
3194 [HumanEval/82] Attempt 0/3
3195 [HumanEval/82] Ø PASS (attempt=0)
3196 =====
3196 -----
3197 =====
3197 [84/164] HumanEval/83
3198 =====
3198 -----
3199 [TASK] HumanEval/83 | entry_point=starts_one_ends
3200 [PROMPT PREVIEW]
3201
3202 def starts_one_ends(n):
```

```
3203     """
3204     Given a positive integer n, return the count of the numbers of n
3205     -digit
3206     positive integers that start or end with 1.
3207     """
3208 -----
3209 [INITIAL GENERATED CODE]
3210 def starts_one_ends(n):
3211     if n == 1:
3212         return 1
3213     else:
3214         return 2 * (10 ** (n - 1)) - 10 ** (n - 2)
3215 -----
3216 [HumanEval/83] Attempt 0/3
3217 [HumanEval/83] ⚡ FAIL (attempt=0)
3218 [TEST ERROR PREVIEW]
3219 Traceback (most recent call last):
3220   File "C:\Users\JOIN\Downloads\Multi-
3221     Reviewer_Governance_for_Code_Generation\src\evaluator\
3222       humaneval_evaluator.py", line 39, in _worker
3223         ns["check"](candidate)
3224   File "<string>", line 6, in check
3225 Assertion
3226 -----
3227 [HumanEval/83] [REPAIRED CODE] (after attempt=0)
3228 def starts_one_ends(n):
3229     if n == 1:
3230         return 1
3231     else:
3232         return 2 * (10 ** (n - 1)) - 10 ** (n - 1)
3233 -----
3234 [HumanEval/83] Attempt 1/3
3235 [HumanEval/83] ⚡ FAIL (attempt=1)
3236 [TEST ERROR PREVIEW]
3237 Traceback (most recent call last):
3238   File "C:\Users\JOIN\Downloads\Multi-
3239     Reviewer_Governance_for_Code_Generation\src\evaluator\
3240       humaneval_evaluator.py", line 39, in _worker
3241         ns["check"](candidate)
3242   File "<string>", line 6, in check
3243 Assertion
3244 -----
3245 [HumanEval/83] [REPAIRED CODE] (after attempt=1)
3246 def starts_one_ends(n):
3247     if n == 1:
3248         return 1
3249     else:
3250         return 2 * (10 ** (n - 1)) - 10 ** (n - 2)
```

```
3246 -----
-----
3247 [HumanEval/83] Attempt 2/3
3248 [HumanEval/83] ✘ FAIL (attempt=2)
3249 [TEST ERROR PREVIEW]
3250 Traceback (most recent call last):
3251   File "C:\Users\JOIN\Downloads\Multi-
    Reviewer_Governance_for_Code_Generation\src\evaluator\
      humaneval_evaluator.py", line 39, in _worker
3252     ns["check"](candidate)
3253   File "<string>", line 6, in check
3254 AssertionError
3255 -----
-----
3256 [HumanEval/83] [REPAIRED CODE] (after attempt=2)
3257 def starts_one_ends(n):
3258     if n == 1:
3259         return 1
3260     else:
3261         return 2 * (10 ** (n - 1)) - 10 ** (n - 2)
3262 -----
-----
3263 [HumanEval/83] Attempt 3/3
3264 [HumanEval/83] ✘ FAIL (attempt=3)
3265 [TEST ERROR PREVIEW]
3266 Traceback (most recent call last):
3267   File "C:\Users\JOIN\Downloads\Multi-
    Reviewer_Governance_for_Code_Generation\src\evaluator\
      humaneval_evaluator.py", line 39, in _worker
3268     ns["check"](candidate)
3269   File "<string>", line 6, in check
3270 AssertionError
3271 -----
-----
3272 [HumanEval/83] ✘ FINAL FAIL after 3 retries
3273 =====
=====
3274 [85/164] HumanEval/84
3275 =====
=====
3276 [TASK] HumanEval/84 | entry_point=solve
3277 [PROMPT PREVIEW]
3278
3279 def solve(N):
3280     """Given a positive integer N, return the total sum of its
    digits in binary.
3281
3282     Example
3283         For N = 1000, the sum of digits will be 1 the output should
    be "1".
3284         For N = 150, the sum of digits will be 6 the output should
    be "110".
3285         For N = 147, the sum of digits will be 12 the output should
```

```
3285 be "1100".
3286
3287     Variables:
3288         @N integer
3289         Con...
3290 -----
3291 -----
3291 [INITIAL GENERATED CODE]
3292 def solve(N):
3293     return bin(sum(int(digit) for digit in str(N)))[2:]
3294 -----
3295 -----
3295 [HumanEval/84] Attempt 0/3
3296 [HumanEval/84] ⚡ PASS (attempt=0)
3297 =====
3298 =====
3298 [86/164] HumanEval/85
3299 =====
3300 =====
3300 [TASK] HumanEval/85 | entry_point=add
3301 [PROMPT PREVIEW]
3302
3303 def add(lst):
3304     """Given a non-empty list of integers lst. add the even elements
3304      that are at odd indices..
3305
3306
3307     Examples:
3308         add([4, 2, 6, 7]) ==> 2
3309         """
3310 -----
3311 -----
3311 [INITIAL GENERATED CODE]
3312 def add(lst):
3313     return sum(x for i, x in enumerate(lst) if i % 2 == 1 and x % 2
3313      == 0)
3314 -----
3315 -----
3315 [HumanEval/85] Attempt 0/3
3316 [HumanEval/85] ⚡ PASS (attempt=0)
3317 =====
3318 =====
3318 [87/164] HumanEval/86
3319 =====
3320 =====
3320 [TASK] HumanEval/86 | entry_point=anti_shuffle
3321 [PROMPT PREVIEW]
3322
3323 def anti_shuffle(s):
3324     """
3325     Write a function that takes a string and returns an ordered
3325      version of it.
3326     Ordered version of string, is a string where all words (
```

```

3326 separated by space)
3327     are replaced by a new word where all the characters arranged in
3328     ascending order based on ascii value.
3329     Note: You should keep the order of words and blank spaces in the
3330     sentence.
3330
3331     For example:...
3332 -----
3333 -----
3333 [INITIAL GENERATED CODE]
3334 def anti_shuffle(s):
3335     words = s.split()
3336     ordered_words = [''.join(sorted(word)) for word in words]
3337     return ' '.join(ordered_words)
3338 -----
3339 -----
3339 [HumanEval/86] Attempt 0/3
3340 [HumanEval/86] ✘ PASS (attempt=0)
3341 =====
3342 [88/164] HumanEval/87
3343 =====
3344 [TASK] HumanEval/87 | entry_point=get_row
3345 [PROMPT PREVIEW]
3346
3347 def get_row(lst, x):
3348     """
3349         You are given a 2 dimensional data, as a nested lists,
3350         which is similar to matrix, however, unlike matrices,
3351         each row may contain a different number of columns.
3352         Given lst, and integer x, find integers x in the list,
3353         and return list of tuples, [(x1, y1), (x2, y2) ...] such that
3354         each tuple is a coordinate - (row, columns), starting with 0.
3355         So...
3356 -----
3357 -----
3357 [INITIAL GENERATED CODE]
3358 def get_row(lst, x):
3359     result = []
3360     for i, row in enumerate(lst):
3361         for j, value in enumerate(row[::-1]):
3362             if value == x:
3363                 result.append((i, len(row) - 1 - j))
3364     return sorted(result)
3365 -----
3366 -----
3366 [HumanEval/87] Attempt 0/3
3367 [HumanEval/87] ✘ FAIL (attempt=0)
3368 [TEST ERROR PREVIEW]
3369 Traceback (most recent call last):
3370   File "C:\Users\JOIN\Downloads\Multi-
Reviewer_Governance_for_Code_Generation\src\evaluator\

```

```
3370 humaneval_evaluator.py", line 39, in _worker
3371     ns["check"](candidate)
3372 File "<string>", line 4, in check
3373 AssertionError
3374 -----
3375 [HumanEval/87] [REPAIRED CODE] (after attempt=0)
3376 def get_row(lst, x):
3377     result = []
3378     for i, row in enumerate(lst):
3379         for j, value in enumerate(row):
3380             if value == x:
3381                 result.append((i, j))
3382     return sorted(result)
3383 -----
3384 [HumanEval/87] Attempt 1/3
3385 [HumanEval/87] ⚡ FAIL (attempt=1)
3386 [TEST ERROR PREVIEW]
3387 Traceback (most recent call last):
3388   File "C:\Users\JOIN\Downloads\Multi-
Reviewer_Governance_for_Code_Generation\src\evaluator\
humaneval_evaluator.py", line 39, in _worker
3389     ns["check"](candidate)
3390   File "<string>", line 4, in check
3391 AssertionError
3392 -----
3393 [HumanEval/87] [REPAIRED CODE] (after attempt=1)
3394 def get_row(lst, x):
3395     result = []
3396     for i, row in enumerate(lst):
3397         for j, value in enumerate(row):
3398             if value == x:
3399                 result.append((i, j))
3400     return sorted(result)
3401 -----
3402 [HumanEval/87] Attempt 2/3
3403 [HumanEval/87] ⚡ FAIL (attempt=2)
3404 [TEST ERROR PREVIEW]
3405 Traceback (most recent call last):
3406   File "C:\Users\JOIN\Downloads\Multi-
Reviewer_Governance_for_Code_Generation\src\evaluator\
humaneval_evaluator.py", line 39, in _worker
3407     ns["check"](candidate)
3408   File "<string>", line 4, in check
3409 AssertionError
3410 -----
3411 [HumanEval/87] [REPAIRED CODE] (after attempt=2)
3412 def get_row(lst, x):
3413     result = []
```

```
3414     for i, row in enumerate(lst):
3415         for j, value in enumerate(row):
3416             if value == x:
3417                 result.append((i, j))
3418     return sorted(result)
3419 -----
3420 -----
3420 [HumanEval/87] Attempt 3/3
3421 [HumanEval/87] ⚡ FAIL (attempt=3)
3422 [TEST ERROR PREVIEW]
3423 Traceback (most recent call last):
3424   File "C:\Users\JOIN\Downloads\Multi-
    Reviewer_Governance_for_Code_Generation\src\evaluator\
      humaneval_evaluator.py", line 39, in _worker
3425     ns["check"](candidate)
3426   File "<string>", line 4, in check
3427 AssertionError
3428 -----
3429 -----
3429 [HumanEval/87] ⚡ FINAL FAIL after 3 retries
3430 =====
3431 =====
3431 [89/164] HumanEval/88
3432 =====
3433 =====
3433 [TASK] HumanEval/88 | entry_point=sort_array
3434 [PROMPT PREVIEW]
3435
3436 def sort_array(array):
3437     """
3438         Given an array of non-negative integers, return a copy of the
            given array after sorting,
3439         you will sort the given array in ascending order if the sum(
            first index value, last index value) is odd,
3440         or sort it in descending order if the sum( first index value,
            last index value) is even.
3441
3442     Note:
3443         * don't change the given array.
3444
3445     Examples:
3446         * sort...
3447 -----
3448 -----
3448 [INITIAL GENERATED CODE]
3449 def sort_array(array):
3450     if not array:
3451         return []
3452
3453     sorted_array = array[:]
3454     if (sorted_array[0] + sorted_array[-1]) % 2 == 0:
3455         sorted_array.sort(reverse=True)
3456     else:
```

```
3457         sorted_array.sort()
3458
3459     return sorted_array
3460 -----
3461 -----
3461 [HumanEval/88] Attempt 0/3
3462 [HumanEval/88] ⚡ PASS (attempt=0)
3463 =====
3464 =====
3464 [90/164] HumanEval/89
3465 =====
3465 =====
3466 [TASK] HumanEval/89 | entry_point=encrypt
3467 [PROMPT PREVIEW]
3468
3469 def encrypt(s):
3470     """Create a function encrypt that takes a string as an argument
3470     and
3471     returns a string encrypted with the alphabet being rotated.
3472     The alphabet should be rotated in a manner such that the letters
3473     shift down by two multiplied to two places.
3474     For example:
3475     encrypt('hi') returns 'lm'
3476     encrypt('asdfghjkl') returns 'ewhjklnop'
3477     encrypt('gf') returns 'kj'...
3478 -----
3479 -----
3479 [INITIAL GENERATED CODE]
3480 def encrypt(s):
3481     result = []
3482     for char in s:
3483         if char.isalpha():
3484             shift = (2 * 2) % 26
3485             if char.islower():
3486                 result.append(chr((ord(char) - ord('a') + shift) %
3486                     26 + ord('a'))))
3487             else:
3488                 result.append(chr((ord(char) - ord('A') + shift) %
3488                     26 + ord('A'))))
3489             else:
3490                 result.append(char)
3491     return ''.join(result)
3492 -----
3493 -----
3493 [HumanEval/89] Attempt 0/3
3494 [HumanEval/89] ⚡ PASS (attempt=0)
3495 =====
3495 =====
3496 [91/164] HumanEval/90
3497 =====
3497 =====
3498 [TASK] HumanEval/90 | entry_point=next_smallest
```

```
3499 [PROMPT PREVIEW]
3500
3501 def next_smallest(lst):
3502     """
3503         You are given a list of integers.
3504         Write a function next_smallest() that returns the 2nd smallest
3505         element of the list.
3506         Return None if there is no such element.
3507
3508         next_smallest([1, 2, 3, 4, 5]) == 2
3509         next_smallest([5, 1, 4, 3, 2]) == 2
3510         next_smallest([]) == None
3511         next_smallest([1, 1]) == None
3512     -----
3513     -----
3514 [INITIAL GENERATED CODE]
3515 def next_smallest(lst):
3516     if len(lst) < 2:
3517         return None
3518     unique_lst = sorted(set(lst))
3519     return unique_lst[1] if len(unique_lst) > 1 else None
3520 -----
3521 [HumanEval/90] Attempt 0/3
3522 [HumanEval/90] 0 PASS (attempt=0)
3523 =====
3524 =====
3525 [92/164] HumanEval/91
3526 =====
3527 =====
3528 [TASK] HumanEval/91 | entry_point=is_bored
3529 [PROMPT PREVIEW]
3530
3531 def is_bored(s):
3532     """
3533         You'll be given a string of words, and your task is to count the
3534         number
3535         of boredoms. A boredom is a sentence that starts with the word "
3536         I".
3537         Sentences are delimited by '.', '?' or '!'.
3538
3539         For example:
3540         >>> is_bored("Hello world")
3541         0
3542         >>> is_bored("The sky is blue. The sun is shining. I love this
3543         weather")
3544         1
3545     """
3546 -----
3547     -----
3548 [INITIAL GENERATED CODE]
3549 def is_bored(s):
```

```
3543     sentences = S.split('.')
3544     boredom_count = sum(1 for sentence in sentences if sentence.
3545         strip().startswith('I'))
3545     return boredom_count
3546 -----
3547 [HumanEval/91] Attempt 0/3
3548 [HumanEval/91] ✘ FAIL (attempt=0)
3549 [TEST ERROR PREVIEW]
3550 Traceback (most recent call last):
3551   File "C:\Users\JOIN\Downloads\Multi-
3552     Reviewer_Governance_for_Code_Generation\src\evaluator\
3553       humaneval_evaluator.py", line 39, in _worker
3554       ns["check"](candidate)
3555     File "<string>", line 5, in check
3556     AssertionError: Test 2
3557 -----
3558 [HumanEval/91] [REPAIRED CODE] (after attempt=0)
3559 def is_bored(S):
3560     sentences = S.split('.')
3561     boredom_count = sum(1 for sentence in sentences if sentence.
3562         strip().startswith('I ') or sentence.strip().startswith('i '))
3563     return boredom_count
3564 -----
3565 [HumanEval/91] Attempt 1/3
3566 [HumanEval/91] ✘ PASS (attempt=1)
3567 =====
3568 =====
3569 [93/164] HumanEval/92
3570 =====
3571 =====
3572 [TASK] HumanEval/92 | entry_point=any_int
3573 [PROMPT PREVIEW]
3574
3575 def any_int(x, y, z):
3576     '''
3577     Create a function that takes 3 numbers.
3578     Returns true if one of the numbers is equal to the sum of the
3579     other two, and all numbers are integers.
3580     Returns false in any other cases.
3581
3582 Examples
3583 any_int(5, 2, 7) → True
3584
3585 any_int(3, 2, 2) → False
3586
3587 any_int(3, -2, 1) → True
3588
3589 any_int(3.6, -2.2, 2) → False
3590
3591
```

```
3586
3587     ''
3588 -----
3589 [INITIAL GENERATED CODE]
3590 def any_int(x, y, z):
3591     return all(isinstance(i, int) for i in [x, y, z]) and (x + y ==
3592     z or x + z == y or y + z == x)
3593 -----
3594 [HumanEval/92] Attempt 0/3
3595 [HumanEval/92] ✘ PASS (attempt=0)
3596 =====
3597 =====
3598 [94/164] HumanEval/93
3599 =====
3600 =====
3601 def encode(message):
3602     """
3603         Write a function that takes a message, and encodes in such a
3604         way that it swaps case of all letters, replaces all vowels in
3605         the message with the letter that appears 2 places ahead of that
3606         vowel in the english alphabet.
3607         Assume only letters.
3608
3609     Examples:
3610     >>> encode('test')
3611     'TGST'
3612     >>> encode('This is a message')
3613     'tHKS KS C MG...
3614 -----
3615 -----
3616 [INITIAL GENERATED CODE]
3617 def encode(message):
3618     vowels = "aeiou"
3619     shifted_vowels = {v: chr((ord(v) - 97 + 2) % 26 + 97) for v in
3620     vowels}
3621     shifted_vowels.update({v.upper(): shifted_vowels[v].upper() for
3622     v in vowels})
3623     return ''.join(shifted_vowels.get(c, c).swapcase() for c in
3624     message)
3625
3626 -----
3627 -----
3628 [HumanEval/93] Attempt 0/3
3629 [HumanEval/93] ✘ PASS (attempt=0)
3630 =====
3631 =====
3632 [95/164] HumanEval/94
3633 =====
```

```
3627 =====
3628 [TASK] HumanEval/94 | entry_point=skjkasdkd
3629 [PROMPT PREVIEW]
3630
3631
3632 def skjkasdkd(lst):
3633     """You are given a list of integers.
3634     You need to find the largest prime value and return the sum of
3635     its digits.
3636
3637     Examples:
3638     For lst = [0,3,2,1,3,5,7,4,5,5,5,2,181,32,4,32,3,2,32,324,4,3]
3639     the output should be 10
3640     For lst = [1,0,1,8,2,4597,2,1,3,40,1,2,1,2,4,2,5,1] the output
3641     should be 25
3642     For lst = [1,3,1,32,5107,34,83278,109,163,23,2323,32,30,1,9,3]
3643     the o...
3644 -----
3645 -----
3646 [INITIAL GENERATED CODE]
3647 def skjkasdkd(lst):
3648     def is_prime(n):
3649         if n <= 1:
3650             return False
3651         for i in range(2, int(n**0.5) + 1):
3652             if n % i == 0:
3653                 return False
3654         return True
3655
3656     max_prime = 0
3657     for num in lst:
3658         if is_prime(num) and num > max_prime:
3659             max_prime = num
3660
3661     return sum(int(digit) for digit in str(max_prime))
3662 -----
3663 -----
3664 [HumanEval/94] Attempt 0/3
3665 [HumanEval/94] 0 PASS (attempt=0)
3666 =====
3667 =====
3668 [96/164] HumanEval/95
3669 =====
3670 =====
3671
3672 =====
3673 [TASK] HumanEval/95 | entry_point=check_dict_case
3674 [PROMPT PREVIEW]
3675
3676 def check_dict_case(dict):
3677     """
3678     Given a dictionary, return True if all keys are strings in lower
3679     case or all keys are strings in upper case, else return False.
3680     The function should return False if the given dictionary is
```

```
3670 empty.
3671     Examples:
3672         check_dict_case({"a":"apple", "b":"banana"}) should return True.
3673         check_dict_case({"a":"apple", "A":"banana", "B":"banana"})
3674             should ret...
3674 -----
3674 -----
3675 [INITIAL GENERATED CODE]
3676 def check_dict_case(d):
3677     if not d:
3678         return False
3679
3680     keys = list(d.keys())
3681     all_lower = all(key.islower() for key in keys)
3682     all_upper = all(key.isupper() for key in keys)
3683
3684     return all_lower or all_upper
3685 -----
3685 -----
3686 [HumanEval/95] Attempt 0/3
3687 [HumanEval/95] ⚡ FAIL (attempt=0)
3688 [TEST ERROR PREVIEW]
3689 Traceback (most recent call last):
3690   File "C:\Users\JOIN\Downloads\Multi-
3691     Reviewer_Governance_for_Code_Generation\src\evaluator\
3692       humaneval_evaluator.py", line 39, in _worker
3693         ns["check"](candidate)
3694   File "<string>", line 6, in check
3695   File "<string>", line 20, in check_dict_case
3696   File "<string>", line 20, in <genexpr>
3697     AttributeError: 'int' object has no attribute 'islower'
3698 -----
3698 -----
3699 [HumanEval/95] [REPAIRED CODE] (after attempt=0)
3700 def check_dict_case(d):
3701     if not d:
3702         return False
3703
3704     keys = list(d.keys())
3705     all_lower = all(isinstance(key, str) and key.islower() for key
3706       in keys)
3707     all_upper = all(isinstance(key, str) and key.isupper() for key
3708       in keys)
3709
3710     return all_lower or all_upper
3711 -----
3711 -----
3712 [HumanEval/95] Attempt 1/3
3713 [HumanEval/95] ⚡ PASS (attempt=1)
3714 =====
3714 =====
3715 [97/164] HumanEval/96
3716 =====
```

```
3712 =====
3713 [TASK] HumanEval/96 | entry_point=count_up_to
3714 [PROMPT PREVIEW]
3715
3716 def count_up_to(n):
3717     """Implement a function that takes an non-negative integer and
3718     returns an array of the first n
3719     integers that are prime numbers and less than n.
3720     for example:
3721     count_up_to(5) => [2,3]
3722     count_up_to(11) => [2,3,5,7]
3723     count_up_to(0) => []
3724     count_up_to(20) => [2,3,5,7,11,13,17,19]
3725     count_up_to(1) => []
3726     count_up_to(18) => [2,3,5,7,11,13,17]
3727     """
3728 -----
3729 -----
3728 [INITIAL GENERATED CODE]
3729 def is_prime(num):
3730     if num <= 1:
3731         return False
3732     for i in range(2, int(num**0.5) + 1):
3733         if num % i == 0:
3734             return False
3735     return True
3736
3737 def count_up_to(n):
3738     primes = []
3739     for i in range(2, n):
3740         if is_prime(i):
3741             primes.append(i)
3742     return primes
3743 -----
3744 -----
3744 [HumanEval/96] Attempt 0/3
3745 [HumanEval/96] ✘ PASS (attempt=0)
3746 =====
3747 -----
3747 [98/164] HumanEval/97
3748 =====
3749 -----
3749 [TASK] HumanEval/97 | entry_point=multiply
3750 [PROMPT PREVIEW]
3751
3752 def multiply(a, b):
3753     """Complete the function that takes two integers and returns
3754     the product of their unit digits.
3755     Assume the input is always valid.
3756     Examples:
3757     multiply(148, 412) should return 16.
3758     multiply(19, 28) should return 72.
3759     multiply(2020, 1851) should return 0.
```

```
3760     multiply(14,-15) should return 20.
3761     """
3762 -----
3763 -----
3763 [INITIAL GENERATED CODE]
3764 def multiply(a, b):
3765     return (a % 10) * (b % 10)
3766 -----
3767 -----
3767 [HumanEval/97] Attempt 0/3
3768 [HumanEval/97] ⚡ PASS (attempt=0)
3769 =====
3770 =====
3770 [99/164] HumanEval/98
3771 =====
3771 =====
3772 [TASK] HumanEval/98 | entry_point=count_upper
3773 [PROMPT PREVIEW]
3774
3775 def count_upper(s):
3776     """
3777         Given a string s, count the number of uppercase vowels in even
3777         indices.
3778
3779         For example:
3780         count_upper('aBCdEf') returns 1
3781         count_upper('abcdefg') returns 0
3782         count_upper('dBBe') returns 0
3783         """
3784 -----
3784 -----
3785 [INITIAL GENERATED CODE]
3786 def count_upper(s):
3787     return sum(1 for i, c in enumerate(s) if i % 2 == 0 and c in 'AEIOU')
3788 -----
3788 -----
3789 [HumanEval/98] Attempt 0/3
3790 [HumanEval/98] ⚡ PASS (attempt=0)
3791 =====
3791 =====
3792 [100/164] HumanEval/99
3793 =====
3793 =====
3794 [TASK] HumanEval/99 | entry_point=closest_integer
3795 [PROMPT PREVIEW]
3796
3797 def closest_integer(value):
3798     '''
3799         Create a function that takes a value (string) representing a
3800         number
3800         and returns the closest integer to it. If the number is
3800         equidistant
```

```
3801     from two integers, round it away from zero.
3802
3803     Examples
3804     >>> closest_integer("10")
3805     10
3806     >>> closest_integer("15.3")
3807     15
3808
3809     Note:
3810     Rounding away from zero means that if the given number is
3811     equidist...
3812 -----
3813 [INITIAL GENERATED CODE]
3814 def closest_integer(value):
3815     num = float(value)
3816     if abs(num - round(num)) == 0.5:
3817         return int(num + (num > 0))
3818     else:
3819         return round(num)
3820 -----
3821 [HumanEval/99] Attempt 0/3
3822 [HumanEval/99] ✘ FAIL (attempt=0)
3823 [TEST ERROR PREVIEW]
3824 Traceback (most recent call last):
3825   File "C:\Users\JOIN\Downloads\Multi-
3826     Reviewer_Governance_for_Code_Generation\src\evaluator\
3827       humaneval_evaluator.py", line 39, in _worker
3828       ns["check"](candidate)
3829   File "<string>", line 6, in check
3830AssertionError: Test 3
3831 -----
3832 [HumanEval/99] [REPAIRED CODE] (after attempt=0)
3833 def closest_integer(value):
3834     num = float(value)
3835     if abs(num - round(num)) == 0.5:
3836         return int(num + (num > 0))
3837     else:
3838         return round(num)
3839 -----
3840 [HumanEval/99] Attempt 1/3
3841 [HumanEval/99] ✘ FAIL (attempt=1)
3842 [TEST ERROR PREVIEW]
3843 Traceback (most recent call last):
3844   File "C:\Users\JOIN\Downloads\Multi-
3845     Reviewer_Governance_for_Code_Generation\src\evaluator\
3846       humaneval_evaluator.py", line 39, in _worker
3847       ns["check"](candidate)
3848   File "<string>", line 6, in check
3849AssertionError: Test 3
```

```
3845 -----
-----
3846 [HumanEval/99] [REPAIRED CODE] (after attempt=1)
3847 def closest_integer(value):
3848     num = float(value)
3849     if abs(num - round(num)) == 0.5:
3850         return int(num + (num > 0))
3851     else:
3852         return round(num)
3853 -----
-----
3854 [HumanEval/99] Attempt 2/3
3855 [HumanEval/99] ⚡ FAIL (attempt=2)
3856 [TEST ERROR PREVIEW]
3857 Traceback (most recent call last):
3858   File "C:\Users\JOIN\Downloads\Multi-
    Reviewer_Governance_for_Code_Generation\src\evaluator\
      humaneval_evaluator.py", line 39, in _worker
3859     ns["check"](candidate)
3860   File "<string>", line 6, in check
3861 AssertionError: Test 3
3862 -----
-----
3863 [HumanEval/99] [REPAIRED CODE] (after attempt=2)
3864 def closest_integer(value):
3865     num = float(value)
3866     if abs(num - round(num)) == 0.5:
3867         return int(num + (num > 0))
3868     else:
3869         return round(num)
3870 -----
-----
3871 [HumanEval/99] Attempt 3/3
3872 [HumanEval/99] ⚡ FAIL (attempt=3)
3873 [TEST ERROR PREVIEW]
3874 Traceback (most recent call last):
3875   File "C:\Users\JOIN\Downloads\Multi-
    Reviewer_Governance_for_Code_Generation\src\evaluator\
      humaneval_evaluator.py", line 39, in _worker
3876     ns["check"](candidate)
3877   File "<string>", line 6, in check
3878 AssertionError: Test 3
3879 -----
-----
3880 [HumanEval/99] ⚡ FINAL FAIL after 3 retries
3881 =====
=====
3882 [101/164] HumanEval/100
3883 =====
=====
3884 [TASK] HumanEval/100 | entry_point=make_a_pile
3885 [PROMPT PREVIEW]
3886
```

```
3887 def make_a_pile(n):
3888     """
3889     Given a positive integer n, you have to make a pile of n levels
3890     of stones.
3891     The first level has n stones.
3892     The number of stones in the next level is:
3893         - the next odd number if n is odd.
3894         - the next even number if n is even.
3895     Return the number of stones in each level in a list, where
3896     element at index
3897         i represents the number of stones in th...
3896 -----
3897 -----
3897 [INITIAL GENERATED CODE]
3898 def make_a_pile(n):
3899     pile = []
3900     current_stones = n
3901     for i in range(n):
3902         pile.append(current_stones)
3903         if n % 2 == 0:
3904             current_stones += 2
3905         else:
3906             current_stones += 2
3907     return pile
3908 -----
3908 -----
3909 [HumanEval/100] Attempt 0/3
3910 [HumanEval/100] [] PASS (attempt=0)
3911 =====
3911 =====
3912 [102/164] HumanEval/101
3913 =====
3913 =====
3914 [TASK] HumanEval/101 | entry_point=words_string
3915 [PROMPT PREVIEW]
3916
3917 def words_string(s):
3918     """
3919     You will be given a string of words separated by commas or
3920     spaces. Your task is
3921     to split the string into words and return an array of the words.
3922
3923     For example:
3924     words_string("Hi, my name is John") == ["Hi", "my", "name", "is",
3925     ", "John"]
3924     words_string("One, two, three, four, five, six") == ["One", "two",
3925     ", "three", "four", "five", "six"]
3925     """
3926 -----
3926 -----
3927 [INITIAL GENERATED CODE]
3928 def words_string(s):
3929     return s.replace(',', ' ').split()
```

```
3930 -----
-----
3931 [HumanEval/101] Attempt 0/3
3932 [HumanEval/101] Ø PASS (attempt=0)
3933 =====
=====
3934 [103/164] HumanEval/102
3935 =====
=====
3936 [TASK] HumanEval/102 | entry_point=choose_num
3937 [PROMPT PREVIEW]
3938
3939 def choose_num(x, y):
3940     """This function takes two positive numbers x and y and returns
3941     the
3942     biggest even integer number that is in the range [x, y]
3943     inclusive. If
3944     there's no such number, then the function should return -1.
3945
3946     For example:
3947     choose_num(12, 15) = 14
3948     choose_num(13, 12) = -1
3949     """
3950
3951 [INITIAL GENERATED CODE]
3952 def choose_num(x, y):
3953     if x > y:
3954         return -1
3955     for num in range(y, x-1, -1):
3956         if num % 2 == 0:
3957             return num
3958     return -1
3959
3960 -----
-----
3961 [HumanEval/102] Attempt 0/3
3962 [HumanEval/102] Ø PASS (attempt=0)
3963 =====
=====
3964 [104/164] HumanEval/103
3965 =====
=====
3966 [TASK] HumanEval/103 | entry_point=rounded_avg
3967 [PROMPT PREVIEW]
3968
3969 def rounded_avg(n, m):
3970     """You are given two positive integers n and m, and your task is
3971     to compute the
3972     average of the integers from n through m (including n and m).
3973     Round the answer to the nearest integer and convert that to
3974     binary.
3975     If n is greater than m, return -1.
3976     Example:
```

```
3972     rounded_avg(1, 5) => "0b11"
3973     rounded_avg(7, 5) => -1
3974     rounded_avg(10, 20) => "0b1111"
3975     r...
3976 -----
3977 -----
3977 [INITIAL GENERATED CODE]
3978 def rounded_avg(n, m):
3979     if n > m:
3980         return -1
3981     avg = round((n + m) / 2)
3982     return bin(avg)
3983 -----
3984 -----
3984 [HumanEval/103] Attempt 0/3
3985 [HumanEval/103] ✘ PASS (attempt=0)
3986 =====
3986 =====
3987 [105/164] HumanEval/104
3988 =====
3988 =====
3989 [TASK] HumanEval/104 | entry_point=unique_digits
3990 [PROMPT PREVIEW]
3991
3992 def unique_digits(x):
3993     """Given a list of positive integers x. return a sorted list of
3993     all
3994     elements that hasn't any even digit.
3995
3996     Note: Returned list should be sorted in increasing order.
3997
3998     For example:
3999     >>> unique_digits([15, 33, 1422, 1])
4000     [1, 15, 33]
4001     >>> unique_digits([152, 323, 1422, 10])
4002     []
4003     """
4004 -----
4004 -----
4005 [INITIAL GENERATED CODE]
4006 def unique_digits(x):
4007     def has_no_even_digit(n):
4008         return all(int(digit) % 2 != 0 for digit in str(n))
4009
4010     result = [num for num in x if has_no_even_digit(num)]
4011     return sorted(result)
4012 -----
4012 -----
4013 [HumanEval/104] Attempt 0/3
4014 [HumanEval/104] ✘ PASS (attempt=0)
4015 =====
4015 =====
4016 [106/164] HumanEval/105
```

```
4017 =====
4018 =====
4019 [TASK] HumanEval/105 | entry_point=by_length
4020 [PROMPT PREVIEW]
4021 def by_length(arr):
4022     """
4023         Given an array of integers, sort the integers that are between 1
4024         and 9 inclusive,
4025         reverse the resulting array, and then replace each digit by its
4026         corresponding name from
4027         "One", "Two", "Three", "Four", "Five", "Six", "Seven", "Eight"
4028         ", "Nine".
4029
4030     For example:
4031     arr = [2, 1, 1, 4, 5, 8, 2, 3]
4032         -> sort arr -> [1, 1, 2, 2, 3, 4, 5, 8]...
4033 -----
4034 -----
4035 [INITIAL GENERATED CODE]
4036 def by_length(arr):
4037     digit_names = ["One", "Two", "Three", "Four", "Five", "Six", "
4038     Seven", "Eight", "Nine"]
4039     sorted_filtered = sorted([x for x in arr if 1 <= x <= 9])
4040     reversed_sorted = sorted_filtered[::-1]
4041     return [digit_names[x-1] for x in reversed_sorted]
4042 -----
4043 -----
4044 [HumanEval/105] Attempt 0/3
4045 [HumanEval/105] 0 PASS (attempt=0)
4046 =====
4047 =====
4048 [107/164] HumanEval/106
4049 =====
4050 =====
4051 [TASK] HumanEval/106 | entry_point=f
4052 [PROMPT PREVIEW]
4053
4054 def f(n):
4055     """
4056         Implement the function f that takes n as a parameter,
4057         and returns a list of size n, such that the value of the element
4058         at index i is the factorial of i if i is even
4059         or the sum of numbers from 1 to i otherwise.
4060         i starts from 1.
4061         the factorial of i is the multiplication of the numbers from 1
4062         to i (1 * 2 * ... * i).
4063         Example:
4064         f(5) == [1, 2, 6, 24, 15]
4065         """
4066 -----
4067 -----
4068 [INITIAL GENERATED CODE]
4069 def f(n):
```

```
4058     result = []
4059     for i in range(1, n + 1):
4060         if i % 2 == 0:
4061             factorial = 1
4062             for j in range(1, i + 1):
4063                 factorial *= j
4064             result.append(factorial)
4065         else:
4066             sum_numbers = sum(range(1, i + 1))
4067             result.append(sum_numbers)
4068     return result
4069 -----
4070 -----
4070 [HumanEval/106] Attempt 0/3
4071 [HumanEval/106] [] PASS (attempt=0)
4072 =====
4073 =====
4073 [108/164] HumanEval/107
4074 =====
4075 =====
4075 [TASK] HumanEval/107 | entry_point=even_odd_palindrome
4076 [PROMPT PREVIEW]
4077
4078 def even_odd_palindrome(n):
4079     """
4080     Given a positive integer n, return a tuple that has the number
4081     of even and odd
4082     integer palindromes that fall within the range(1, n), inclusive.
4083
4084     Example 1:
4085
4086         Input: 3
4087         Output: (1, 2)
4088         Explanation:
4089         Integer palindrome are 1, 2, 3. one of them is even, and two
4090         of them are odd.
4091
4092     Example 2:
4093
4094 -----
4095 -----
4095 [INITIAL GENERATED CODE]
4096 def even_odd_palindrome(n):
4097     def is_palindrome(num):
4098         return str(num) == str(num)[::-1]
4099
4100     even_count = 0
4101     odd_count = 0
4102
4103     for i in range(1, n + 1):
4104         if is_palindrome(i):
```

```
4105             if i % 2 == 0:
4106                 even_count += 1
4107             else:
4108                 odd_count += 1
4109
4110     return (even_count, odd_count)
4111 -----
4112 [HumanEval/107] Attempt 0/3
4113 [HumanEval/107] 0 PASS (attempt=0)
4114 =====
4115 [109/164] HumanEval/108
4116 =====
4117 [TASK] HumanEval/108 | entry_point=count_nums
4118 [PROMPT PREVIEW]
4119
4120 def count_nums(arr):
4121     """
4122         Write a function count_nums which takes an array of integers and
4123         returns
4124         the number of elements which has a sum of digits > 0.
4125         If a number is negative, then its first signed digit will be
4126         negative:
4127             e.g. -123 has signed digits -1, 2, and 3.
4128             >>> count_nums([]) == 0
4129             >>> count_nums([-1, 11, -11]) == 1
4130             >>> count_nums([1, 1, 2]) == 3
4131             """
4132
4133 [INITIAL GENERATED CODE]
4134 def count_nums(arr):
4135     def sum_of_digits(n):
4136         if n >= 0:
4137             return sum(int(digit) for digit in str(n))
4138         else:
4139             digits = [int(d) for d in str(n)[1:]]
4140             digits[0] *= -1
4141             return sum(digits)
4142
4143     return sum(1 for num in arr if sum_of_digits(num) > 0)
4144
4145 [HumanEval/108] Attempt 0/3
4146 [HumanEval/108] 0 PASS (attempt=0)
4147 =====
4148 [110/164] HumanEval/109
4149 =====
4150 [TASK] HumanEval/109 | entry_point=move_one_ball
```

```
4149 [PROMPT PREVIEW]
4150
4151 def move_one_ball(arr):
4152     """We have an array 'arr' of N integers arr[1], arr[2], ..., arr
4153     [N].The
4154         numbers in the array will be randomly ordered. Your task is to
4155         determine if
4156             it is possible to get an array sorted in non-decreasing order by
4157             performing
4158                 the following operation on the given array:
4159                     You are allowed to perform right shift operation any number
4160                     of times.
4161
4162     One...
4163 -----
4164 -----
4165 [INITIAL GENERATED CODE]
4166 def move_one_ball(arr):
4167     if not arr:
4168         return True
4169
4170     n = len(arr)
4171     sorted_arr = sorted(arr)
4172
4173     for i in range(n):
4174         if arr == sorted_arr:
4175             return True
4176         arr.insert(0, arr.pop())
4177
4178     return False
4179 -----
4180 -----
4181 [HumanEval/109] Attempt 0/3
4182 [HumanEval/109] 0 PASS (attempt=0)
4183 =====
4184 =====
4185 [111/164] HumanEval/110
4186 =====
4187 =====
4188 [TASK] HumanEval/110 | entry_point=exchange
4189 [PROMPT PREVIEW]
4190
4191 def exchange(lst1, lst2):
4192     """In this problem, you will implement a function that takes two
4193     lists of numbers,
4194     and determines whether it is possible to perform an exchange of
4195     elements
4196     between them to make lst1 a list of only even numbers.
4197     There is no limit on the number of exchanged elements between
4198     lst1 and lst2.
4199     If it is possible to exchange elements between the lst1 and lst2
4200     ...
4201
4202 -----
```

```
4189 -----
4190 [INITIAL GENERATED CODE]
4191 def exchange(lst1, lst2):
4192     odd_count = sum(1 for x in lst1 if x % 2 != 0)
4193     even_count_in_lst2 = sum(1 for x in lst2 if x % 2 == 0)
4194
4195     return "YES" if even_count_in_lst2 >= odd_count else "NO"
4196 -----
4197 -----
4198 [HumanEval/110] Attempt 0/3
4199 [HumanEval/110] ⚡ PASS (attempt=0)
4200 =====
4201 =====
4202 [112/164] HumanEval/111
4203 =====
4204 =====
4205 [TASK] HumanEval/111 | entry_point=histogram
4206 [PROMPT PREVIEW]
4207
4208 def histogram(test):
4209     """Given a string representing a space separated lowercase
4210     letters, return a dictionary
4211     of the letter with the most repetition and containing the
4212     corresponding count.
4213     If several letters have the same occurrence, return all of them.
4214
4215 Example:
4216     histogram('a b c') == {'a': 1, 'b': 1, 'c': 1}
4217     histogram('a b b a') == {'a': 2, 'b': 2}
4218     histogram('a b c...
4219 -----
4220 -----
4221 [INITIAL GENERATED CODE]
4222 def histogram(test):
4223     if not test:
4224         return {}
4225
4226     words = test.split()
4227     word_count = {}
4228
4229     for word in words:
4230         if word in word_count:
4231             word_count[word] += 1
4232         else:
4233             word_count[word] = 1
4234
4235     max_count = max(word_count.values(), default=0)
4236     result = {word: count for word, count in word_count.items() if
4237               count == max_count}
4238
4239     return result
4240 -----
4241 -----
```

```
4234 [HumanEval/111] Attempt 0/3
4235 [HumanEval/111] Ø PASS (attempt=0)
4236 =====
=====
4237 [113/164] HumanEval/112
4238 =====
=====
4239 [TASK] HumanEval/112 | entry_point=reverse_delete
4240 [PROMPT PREVIEW]
4241
4242 def reverse_delete(s,c):
4243     """Task
4244         We are given two strings s and c, you have to deleted all the
4245         characters in s that are equal to any character in c
4246         then check if the result string is palindrome.
4247         A string is called palindrome if it reads the same backward as
4248         forward.
4249         You should return a tuple containing the result string and True/
4250         False for the check.
4251     Example
4252     For s = "ab...
4250 -----
-----
4251 [INITIAL GENERATED CODE]
4252 def reverse_delete(s, c):
4253     result = ''.join([char for char in s if char not in c])
4254     return (result, result == result[::-1])
4255 -----
-----
4256 [HumanEval/112] Attempt 0/3
4257 [HumanEval/112] Ø PASS (attempt=0)
4258 =====
=====
4259 [114/164] HumanEval/113
4260 =====
=====
4261 [TASK] HumanEval/113 | entry_point=odd_count
4262 [PROMPT PREVIEW]
4263
4264 def odd_count(lst):
4265     """Given a list of strings, where each string consists of only
4266     digits, return a list.
4267     Each element i of the output should be "the number of odd
4268     elements in the
4269     string i of the input." where all the i's should be replaced by
4270     the number
4271     of odd digits in the i'th string of the input.
4272
4270     >>> odd_count(['1234567'])
4271     ["the number of odd elements 4n the str4ng 4...
4272 -----
-----
4273 [INITIAL GENERATED CODE]
```

```
4274 def odd_count(lst):
4275     return [f"the number of odd elements {sum(1 for char in s if int
4276     (char) % 2 != 0)} in the string {s.index(s)+1} of the {s.
4277     index(s)+1} input." for s in lst]
4278 -----
4279 -----
4280 [HumanEval/113] Attempt 0/3
4281 [HumanEval/113] ⚡ FAIL (attempt=0)
4282 [TEST ERROR PREVIEW]
4283 Traceback (most recent call last):
4284   File "C:\Users\JOIN\Downloads\Multi-
4285     Reviewer_Governance_for_Code_Generation\src\evaluator\
4286       humaneval_evaluator.py", line 39, in _worker
4287       ns["check"](candidate)
4288     File "<string>", line 4, in check
4289 AssertionTypeError: Test 1
4290 -----
4291 -----
4292 [HumanEval/113] [REPAIRED CODE] (after attempt=0)
4293 def odd_count(lst):
4294     return [f"the number of odd elements {sum(1 for char in s if int
4295     (char) % 2 != 0)} in the string {lst.index(s)+1} of the input." for
4296     s in lst]
4297 -----
4298 -----
4299 [HumanEval/113] [REPAIRED CODE] (after attempt=1)
4300 def odd_count(lst):
4301     return [f"the number of odd elements {sum(1 for char in s if int
4302     (char) % 2 != 0)} in the string {lst.index(s)+1} of the input." for
4303     s in lst]
4304 -----
4305 -----
4306 [HumanEval/113] Attempt 1/3
4307 [HumanEval/113] ⚡ FAIL (attempt=1)
4308 [TEST ERROR PREVIEW]
4309 Traceback (most recent call last):
4310   File "C:\Users\JOIN\Downloads\Multi-
4311     Reviewer_Governance_for_Code_Generation\src\evaluator\
4312       humaneval_evaluator.py", line 39, in _worker
4313       ns["check"](candidate)
4314     File "<string>", line 4, in check
```

```
4310 AssertionError: Test 1
4311 -----
-----
4312 [HumanEval/113] [REPAIRED CODE] (after attempt=2)
4313 def odd_count(lst):
4314     return [f"the number of odd elements {sum(1 for char in s if int(char) % 2 != 0)} in the string {lst.index(s)+1} of the input." for s in lst]
4315 -----
-----
4316 [HumanEval/113] Attempt 3/3
4317 [HumanEval/113] ⚠ FAIL (attempt=3)
4318 [TEST ERROR PREVIEW]
4319 Traceback (most recent call last):
4320   File "C:\Users\JOIN\Downloads\Multi-
Reviewer_Governance_for_Code_Generation\src\evaluator\
humaneval_evaluator.py", line 39, in _worker
4321     ns["check"](candidate)
4322   File "<string>", line 4, in check
4323 AssertionError: Test 1
4324 -----
-----
4325 [HumanEval/113] ⚠ FINAL FAIL after 3 retries
4326 =====
=====
4327 [115/164] HumanEval/114
4328 =====
=====
4329 [TASK] HumanEval/114 | entry_point=minSubArraySum
4330 [PROMPT PREVIEW]
4331
4332 def minSubArraySum(nums):
4333     """
4334         Given an array of integers nums, find the minimum sum of any non
-empty sub-array
4335         of nums.
4336     Example
4337     minSubArraySum([2, 3, 4, 1, 2, 4]) == 1
4338     minSubArraySum([-1, -2, -3]) == -6
4339     """
4340 -----
-----
4341 [INITIAL GENERATED CODE]
4342 def minSubArraySum(nums):
4343     min_sum = float('inf')
4344     current_sum = 0
4345     for num in nums:
4346         current_sum = min(num, current_sum + num)
4347         min_sum = min(min_sum, current_sum)
4348     return min_sum
4349 -----
-----
4350 [HumanEval/114] Attempt 0/3
```

```
4351 [HumanEval/114] ⚡ PASS (attempt=0)
4352 =====
4353 =====
4353 [116/164] HumanEval/115
4354 =====
4355 =====
4355 [TASK] HumanEval/115 | entry_point=max_fill
4356 [PROMPT PREVIEW]
4357
4358 def max_fill(grid, capacity):
4359     import math
4360     """
4361         You are given a rectangular grid of wells. Each row represents a
4362             single well,
4363                 and each 1 in a row represents a single unit of water.
4364             Each well has a corresponding bucket that can be used to extract
4365                 water from it,
4366                     and all buckets have the same capacity.
4367             Your task is to use the buckets to empty the wells.
4368             Output the numb...
4367 -----
4368 -----
4368 [INITIAL GENERATED CODE]
4369 def max_fill(grid, capacity):
4370     total_water = sum(sum(row) for row in grid)
4371     return math.ceil(total_water / capacity)
4372 -----
4373 -----
4373 [HumanEval/115] Attempt 0/3
4374 [HumanEval/115] ⚡ FAIL (attempt=0)
4375 [TEST ERROR PREVIEW]
4376 Traceback (most recent call last):
4377   File "C:\Users\JOIN\Downloads\Multi-
4378       Reviewer_Governance_for_Code_Generation\src\evaluator\
4379           humaneval_evaluator.py", line 39, in _worker
4380             ns["check"](candidate)
4381             File "<string>", line 6, in check
4382             File "<string>", line 40, in max_fill
4383 NameError: name 'math' is not defined
4382 -----
4383 -----
4383 [HumanEval/115] [REPAIRED CODE] (after attempt=0)
4384 import math
4385
4386 def max_fill(grid, capacity):
4387     total_water = sum(sum(row) for row in grid)
4388     return math.ceil(total_water / capacity)
4389 -----
4390 -----
4390 [HumanEval/115] Attempt 1/3
4391 [HumanEval/115] ⚡ FAIL (attempt=1)
4392 [TEST ERROR PREVIEW]
4393 Traceback (most recent call last):
```

```
4394 File "C:\Users\JOIN\Downloads\Multi-
    Reviewer_Governance_for_Code_Generation\src\evaluator\
        humaneval_evaluator.py", line 39, in _worker
4395     ns["check"]的文化)
4396 File "<string>", line 13, in check
4397 AssertionError: Error
4398 -----
-----
4399 [HumanEval/115] [REPAIRED CODE] (after attempt=1)
4400 import math
4401
4402 def max_fill(grid, capacity):
4403     total_water = sum(sum(row) for row in grid)
4404     if capacity == 0:
4405         raise ValueError("Capacity cannot be zero")
4406     return math.ceil(total_water / capacity)
4407 -----
-----
4408 [HumanEval/115] Attempt 2/3
4409 [HumanEval/115] 0 FAIL (attempt=2)
4410 [TEST ERROR PREVIEW]
4411 Traceback (most recent call last):
4412   File "C:\Users\JOIN\Downloads\Multi-
    Reviewer_Governance_for_Code_Generation\src\evaluator\
        humaneval_evaluator.py", line 39, in _worker
4413     ns["check"]的文化)
4414 File "<string>", line 13, in check
4415 AssertionError: Error
4416 -----
-----
4417 [HumanEval/115] [REPAIRED CODE] (after attempt=2)
4418 import math
4419
4420 def max_fill(grid, capacity):
4421     total_water = sum(sum(row) for row in grid)
4422     if capacity == 0:
4423         raise ValueError("Capacity cannot be zero")
4424     return math.ceil(total_water / capacity)
4425 -----
-----
4426 [HumanEval/115] Attempt 3/3
4427 [HumanEval/115] 0 FAIL (attempt=3)
4428 [TEST ERROR PREVIEW]
4429 Traceback (most recent call last):
4430   File "C:\Users\JOIN\Downloads\Multi-
    Reviewer_Governance_for_Code_Generation\src\evaluator\
        humaneval_evaluator.py", line 39, in _worker
4431     ns["check"]文化)
4432 File "<string>", line 13, in check
4433 AssertionError: Error
4434 -----
-----
4435 [HumanEval/115] 0 FINAL FAIL after 3 retries
```

```
4436 =====
4437 =====
4437 [117/164] HumanEval/116
4438 =====
4438 =====
4439 [TASK] HumanEval/116 | entry_point=sort_array
4440 [PROMPT PREVIEW]
4441
4442 def sort_array(arr):
4443     """
4444         In this Kata, you have to sort an array of non-negative integers
4445         according to
4446             number of ones in their binary representation in ascending order
4447
4448             For similar number of ones, sort based on decimal value.
4449
4450             It must be implemented like this:
4451             >>> sort_array([1, 5, 2, 3, 4]) == [1, 2, 3, 4, 5]
4452             >>> sort_array([-2, -3, -4, -5, -6]) == [-6, -5, -4, -3, -2...
4451 -----
4451 -----
4452 [INITIAL GENERATED CODE]
4453 def sort_array(arr):
4454     return sorted(arr, key=lambda x: (bin(x).count('1'), x))
4455 -----
4455 -----
4456 [HumanEval/116] Attempt 0/3
4457 [HumanEval/116] 0 PASS (attempt=0)
4458 =====
4458 =====
4459 [118/164] HumanEval/117
4460 =====
4460 =====
4461 [TASK] HumanEval/117 | entry_point=select_words
4462 [PROMPT PREVIEW]
4463
4464 def select_words(s, n):
4465     """Given a string s and a natural number n, you have been tasked
4466         to implement
4467         a function that returns a list of all words from string s that
4468             contain exactly
4469             n consonants, in order these words appear in the string s.
4470             If the string s is empty then the function should return an
4471             empty list.
4472             Note: you may assume the input string contains only letters and
4473             s...
4470 -----
4470 -----
4471 [INITIAL GENERATED CODE]
4472 def select_words(s, n):
4473     words = s.split()
4474     result = []
4475     for word in words:
```

```
4476         consonants = sum(1 for char in word if char.lower() not in 'aeiou' and char.isalpha())
4477         if consonants == n:
4478             result.append(word)
4479     return result
4480 -----
4481 -----
4481 [HumanEval/117] Attempt 0/3
4482 [HumanEval/117] [] PASS (attempt=0)
4483 =====
4484 =====
4484 [119/164] HumanEval/118
4485 =====
4486 [TASK] HumanEval/118 | entry_point=get_closest_vowel
4487 [PROMPT PREVIEW]
4488
4489 def get_closest_vowel(word):
4490     """You are given a word. Your task is to find the closest vowel
4491     that stands between
4492     two consonants from the right side of the word (case sensitive).
4493
4494     Vowels in the beginning and ending doesn't count. Return empty
4495     string if you didn't
4496     find any vowel met the above condition.
4497
4498     You may assume that the given string contains English letter
4499     only.....
4500
4501 [INITIAL GENERATED CODE]
4502 def get_closest_vowel(word):
4503     vowels = "aeiouAEIOU"
4504     for i in range(len(word) - 2, 0, -1):
4505         if word[i] in vowels and word[i-1] not in vowels and word[i+1] not in vowels:
4506             return word[i]
4507     return ""
4508 -----
4509 -----
4509 [HumanEval/118] Attempt 0/3
4510 [HumanEval/118] [] PASS (attempt=0)
4511 =====
4512 =====
4513 [TASK] HumanEval/119 | entry_point=match_parens
4514 [PROMPT PREVIEW]
4515
4516 def match_parens(lst):
4517     ''
4518     You are given a list of two strings, both strings consist of
```

```
4516 open
4517     parentheses '(' or close parentheses ')' only.
4518     Your job is to check if it is possible to concatenate the two
        strings in
4519     some order, that the resulting string will be good.
4520     A string S is considered to be good if and only if all
        parentheses in S
4521     are balanced. For example: the str...
4522 -----
-----
4523 [INITIAL GENERATED CODE]
4524 def match_parens(lst):
4525     def is_balanced(s):
4526         balance = 0
4527         for char in s:
4528             if char == '(':
4529                 balance += 1
4530             else:
4531                 balance -= 1
4532             if balance < 0:
4533                 return False
4534         return balance == 0
4535
4536     first_order = lst[0] + lst[1]
4537     second_order = lst[1] + lst[0]
4538     return 'Yes' if is_balanced(first_order) or is_balanced(
        second_order) else 'No'
4539 -----
-----
4540 [HumanEval/119] Attempt 0/3
4541 [HumanEval/119] [] PASS (attempt=0)
4542 =====
=====
4543 [121/164] HumanEval/120
4544 =====
=====
4545 [TASK] HumanEval/120 | entry_point=maximum
4546 [PROMPT PREVIEW]
4547
4548 def maximum(arr, k):
4549     """
4550         Given an array arr of integers and a positive integer k, return
        a sorted list
4551         of length k with the maximum k numbers in arr.
4552
4553     Example 1:
4554
4555         Input: arr = [-3, -4, 5], k = 3
4556         Output: [-4, -3, 5]
4557
4558     Example 2:
4559
4560         Input: arr = [4, -4, 4], k = 2
```

```
4561     Output: [4, 4]
4562
4563 Example 3:
4564
4565     Input: arr = [-3, 2, 1, 2, -1, -2, 1], k = 1...
4566 -----
4567 -----
4568 [INITIAL GENERATED CODE]
4569 def maximum(arr, k):
4570     return sorted(arr)[-k:]
4571 -----
4572 [HumanEval/120] Attempt 0/3
4573 [HumanEval/120] ⚡ FAIL (attempt=0)
4574 [TEST ERROR PREVIEW]
4575 Traceback (most recent call last):
4576   File "C:\Users\JOIN\Downloads\Multi-
4577     Reviewer_Governance_for_Code_Generation\src\evaluator\
4578       humaneval_evaluator.py", line 39, in _worker
4579         ns["check"](candidate)
4580   File "<string>", line 16, in check
4581 AssertionERROR
4582 -----
4583 -----
4584 [HumanEval/120] [REPAIRED CODE] (after attempt=0)
4585 def maximum(arr, k):
4586     return sorted(arr, reverse=True)[:k]
4587 -----
4588 [HumanEval/120] Attempt 1/3
4589 [HumanEval/120] ⚡ FAIL (attempt=1)
4590 [TEST ERROR PREVIEW]
4591 Traceback (most recent call last):
4592   File "C:\Users\JOIN\Downloads\Multi-
4593     Reviewer_Governance_for_Code_Generation\src\evaluator\
4594       humaneval_evaluator.py", line 39, in _worker
4595         ns["check"](candidate)
4596   File "<string>", line 4, in check
4597 AssertionERROR
4598 -----
4599 [HumanEval/120] [REPAIRED CODE] (after attempt=1)
4600 def maximum(arr, k):
4601     return sorted(arr, reverse=True)[:k]
4602 -----
4603 [HumanEval/120] Attempt 2/3
4604 [HumanEval/120] ⚡ FAIL (attempt=2)
4605 [TEST ERROR PREVIEW]
4606 Traceback (most recent call last):
4607   File "C:\Users\JOIN\Downloads\Multi-
4608     Reviewer_Governance_for_Code_Generation\src\evaluator\
4609       humaneval_evaluator.py", line 39, in _worker
```

```
4602     ns["check"](candidate)
4603 File "<string>", line 4, in check
4604 AssertionError
4605 -----
4606 [HumanEval/120] [REPAIRED CODE] (after attempt=2)
4607 def maximum(arr, k):
4608     return sorted(arr, reverse=True)[:k]
4609 -----
4610 [HumanEval/120] Attempt 3/3
4611 [HumanEval/120] ⚡ FAIL (attempt=3)
4612 [TEST ERROR PREVIEW]
4613 Traceback (most recent call last):
4614   File "C:\Users\JOIN\Downloads\Multi-
        Reviewer_Governance_for_Code_Generation\src\evaluator\
          humaneval_evaluator.py", line 39, in _worker
4615     ns["check"](candidate)
4616   File "<string>", line 4, in check
4617 AssertionError
4618 -----
4619 [HumanEval/120] ⚡ FINAL FAIL after 3 retries
4620 =====
4621 [122/164] HumanEval/121
4622 =====
4623 [TASK] HumanEval/121 | entry_point=solution
4624 [PROMPT PREVIEW]
4625
4626 def solution(lst):
4627     """Given a non-empty list of integers, return the sum of all of
        the odd elements that are in even positions.
4628
4629
4630     Examples
4631     solution([5, 8, 7, 1]) ==> 12
4632     solution([3, 3, 3, 3, 3]) ==> 9
4633     solution([30, 13, 24, 321]) ==>0
4634     """
4635 -----
4636 [INITIAL GENERATED CODE]
4637 def solution(lst):
4638     return sum(x for i, x in enumerate(lst) if i % 2 == 1 and x % 2
        == 1)
4639 -----
4640 [HumanEval/121] Attempt 0/3
4641 [HumanEval/121] ⚡ FAIL (attempt=0)
4642 [TEST ERROR PREVIEW]
4643 Traceback (most recent call last):
```

```
4644 File "C:\Users\JOIN\Downloads\Multi-
    Reviewer_Governance_for_Code_Generation\src\evaluator\
        humaneval_evaluator.py", line 39, in _worker
4645     ns["check"]的文化)
4646     File "<string>", line 4, in check
4647 AssertionError
4648 -----
-----
4649 [HumanEval/121] [REPAIRED CODE] (after attempt=0)
4650 def solution(lst):
4651     return sum(x for i, x in enumerate(lst) if i % 2 == 0 and x % 2
4652     == 1)
4653 -----
-----
4654 [HumanEval/121] Attempt 1/3
4655 [HumanEval/121] ⚡ PASS (attempt=1)
4656 =====
=====
4657 [123/164] HumanEval/122
4658 =====
=====
4659 [TASK] HumanEval/122 | entry_point=add_elements
4660 [PROMPT PREVIEW]
4661 def add_elements(arr, k):
4662     """
4663         Given a non-empty array of integers arr and an integer k, return
4664         the sum of the elements with at most two digits from the first k
4665         elements of arr.
4666
4667     Example:
4668         Input: arr = [111,21,3,4000,5,6,7,8,9], k = 4
4669         Output: 24 # sum of 21 + 3
4670
4671     Constraints:
4672         1. 1 <= len(arr) <= 100
4673         2. 1 <= k <= len(arr)
4674     """
4675 -----
-----
4676 [INITIAL GENERATED CODE]
4677 def add_elements(arr, k):
4678     return sum(x for x in arr[:k] if -99 <= x <= 99)
4679 -----
-----
4680 [HumanEval/122] Attempt 0/3
4681 [HumanEval/122] ⚡ PASS (attempt=0)
4682 =====
=====
4683 [124/164] HumanEval/123
4684 =====
=====
```

```
4685 [TASK] HumanEval/123 | entry_point=get_odd_collatz
4686 [PROMPT PREVIEW]
4687
4688 def get_odd_collatz(n):
4689     """
4690         Given a positive integer n, return a sorted list that has the
4691         odd numbers in collatz sequence.
4692
4693         The Collatz conjecture is a conjecture in mathematics that
4694         concerns a sequence defined
4695         as follows: start with any positive integer n. Then each term is
4696         obtained from the
4697         previous term as follows: if the previous term is even, the next
4698         term is one half of...
4699 -----
4700 -----
4701 [INITIAL GENERATED CODE]
4702 def get_odd_collatz(n):
4703     def collatz_sequence(num):
4704         sequence = []
4705         while num != 1:
4706             sequence.append(num)
4707             if num % 2 == 0:
4708                 num //= 2
4709             else:
4710                 num = 3 * num + 1
4711         sequence.append(1)
4712         return sequence
4713
4714 -----
4715 [HumanEval/123] Attempt 0/3
4716 [HumanEval/123] ⚡ PASS (attempt=0)
4717 =====
4718 =====
4719 [125/164] HumanEval/124
4720 =====
4721 =====
4722 [TASK] HumanEval/124 | entry_point=valid_date
4723 [PROMPT PREVIEW]
4724
4725 def valid_date(date):
4726     """
4727         You have to write a function which validates a given date
4728         string and
4729         returns True if the date is valid otherwise False.
4730         The date is valid if all of the following rules are satisfied:
4731         1. The date string is not empty.
4732         2. The number of days is not less than 1 or higher than 31 days
4733             for months 1,3,5,7,8,10,12. And the number of days is not less than
4734             1 or h...
```

```
4726 -----
-----
4727 [INITIAL GENERATED CODE]
4728 def valid_date(date):
4729     if len(date) != 10 or date[2] != '-' or date[5] != '-':
4730         return False
4731
4732     month, day, year = map(int, date.split('-'))
4733
4734     if month < 1 or month > 12:
4735         return False
4736
4737     if day < 1:
4738         return False
4739
4740     if month in [4, 6, 9, 11] and day > 30:
4741         return False
4742     elif month == 2 and day > 29:
4743         return False
4744     elif month in [1, 3, 5, 7, 8, 10, 12] and day > 31:
4745         return False
4746
4747     return True
4748 -----
-----
4749 [HumanEval/124] Attempt 0/3
4750 [HumanEval/124] Ø PASS (attempt=0)
4751 =====
=====
4752 [126/164] HumanEval/125
4753 =====
=====
4754 [TASK] HumanEval/125 | entry_point=split_words
4755 [PROMPT PREVIEW]
4756
4757 def split_words(txt):
4758     '''
4759         Given a string of words, return a list of words split on
        whitespace, if no whitespaces exists in the text you
4760         should split on commas ',' if no commas exists you should return
        the number of lower-case letters with odd order in the
4761         alphabet, ord('a') = 0, ord('b') = 1, ... ord('z') = 25
4762         Examples
4763         split_words("Hello world!") → ["Hello", "world!"]...
4764 -----
-----
4765 [INITIAL GENERATED CODE]
4766 def split_words(txt):
4767     if ' ' in txt:
4768         return txt.split()
4769     elif ',' in txt:
4770         return txt.split(',')
4771     else:
```

```
4772         return sum(1 for char in txt if char.islower() and (ord(char)
4773 ) - ord('a')) % 2 == 0)
4773 -----
4774 [HumanEval/125] Attempt 0/3
4775 [HumanEval/125] ⚡ FAIL (attempt=0)
4776 [TEST ERROR PREVIEW]
4777 Traceback (most recent call last):
4778   File "C:\Users\JOIN\Downloads\Multi-
4779     Reviewer_Governance_for_Code_Generation\src\evaluator\
4780       humaneval_evaluator.py", line 39, in _worker
4781         ns["check"](candidate)
4782   File "<string>", line 8, in check
4783 Assertion
4784 
4785 -----
4786 [HumanEval/125] [REPAIRED CODE] (after attempt=0)
4787 def split_words(txt):
4788     if ' ' in txt:
4789         return txt.split()
4790     elif ',' in txt:
4791         return txt.split(',')
4792     else:
4793         return [char for char in txt if char.islower() and (ord(char)
4794 ) - ord('a')) % 2 == 0]
4795 -----
4796 [HumanEval/125] Attempt 1/3
4797 [HumanEval/125] ⚡ FAIL (attempt=1)
4798 [TEST ERROR PREVIEW]
4799 Traceback (most recent call last):
4800   File "C:\Users\JOIN\Downloads\Multi-
4801     Reviewer_Governance_for_Code_Generation\src\evaluator\
4802       humaneval_evaluator.py", line 39, in _worker
4803         ns["check"](candidate)
4804   File "<string>", line 7, in check
4805 Assertion
4806 
4807 -----
4808 [HumanEval/125] [REPAIRED CODE] (after attempt=1)
4809 def split_words(txt):
4810     if ' ' in txt:
4811         return txt.split()
4812     elif ',' in txt:
4813         return txt.split(',')
4814     else:
4815         return [char for char in txt if char.islower() and (ord(char)
4816 ) - ord('a')) % 2 == 0]
4817 -----
4818 [HumanEval/125] Attempt 2/3
4819 [HumanEval/125] ⚡ FAIL (attempt=2)
4820 [TEST ERROR PREVIEW]
```

```
4813 Traceback (most recent call last):
4814   File "C:\Users\JOIN\Downloads\Multi-
    Reviewer_Governance_for_Code_Generation\src\evaluator\
      humaneval_evaluator.py", line 39, in _worker
4815     ns["check"](candidate)
4816   File "<string>", line 7, in check
4817 AssertionError
4818 -----
4819 [HumanEval/125] [REPAIRED CODE] (after attempt=2)
4820 def split_words(txt):
4821     if ' ' in txt:
4822         return txt.split()
4823     elif ',' in txt:
4824         return txt.split(',')
4825     else:
4826         return [char for char in txt if char.islower() and (ord(char)
4827 ) - ord('a')) % 2 == 0]
4828 -----
4829 [HumanEval/125] Attempt 3/3
4830 [HumanEval/125] Ø FAIL (attempt=3)
4831 [TEST ERROR PREVIEW]
4832 Traceback (most recent call last):
4833   File "C:\Users\JOIN\Downloads\Multi-
    Reviewer_Governance_for_Code_Generation\src\evaluator\
      humaneval_evaluator.py", line 39, in _worker
4834     ns["check"](candidate)
4835   File "<string>", line 7, in check
4836 AssertionError
4837 -----
4838 [HumanEval/125] Ø FINAL FAIL after 3 retries
4839 =====
4840 =====
4841 [127/164] HumanEval/126
4842 =====
4843 =====
4844 [TASK] HumanEval/126 | entry_point=is_sorted
4845 [PROMPT PREVIEW]
4846 =====
4847 =====
4848 def is_sorted(lst):
4849     """
4850     Given a list of numbers, return whether or not they are sorted
4851     in ascending order. If list has more than 1 duplicate of the
4852     same
4853     number, return False. Assume no negative numbers and only
4854     integers.
4855
4856 Examples
4857 is_sorted([5]) → True
4858 is_sorted([1, 2, 3, 4, 5]) → True
4859 is_sorted([1, 3, 2, 4, 5]) → False
```

```
4854     is_sorted([1, 2, 3, 4, 5, 6]) → True...
4855 -----
4856 -----
4856 [INITIAL GENERATED CODE]
4857 def is_sorted(lst):
4858     if len(lst) <= 1:
4859         return True
4860     for i in range(1, len(lst)):
4861         if lst[i] < lst[i - 1]:
4862             return False
4863         if i >= 2 and lst[i] == lst[i - 1] == lst[i - 2]:
4864             return False
4865     return True
4866 -----
4867 -----
4867 [HumanEval/126] Attempt 0/3
4868 [HumanEval/126] Ø PASS (attempt=0)
4869 =====
4870 -----
4870 [128/164] HumanEval/127
4871 =====
4872 -----
4872 [TASK] HumanEval/127 | entry_point=intersection
4873 [PROMPT PREVIEW]
4874
4875 def intersection(interval1, interval2):
4876     """You are given two intervals,
4877     where each interval is a pair of integers. For example, interval
4878     = (start, end) = (1, 2).
4879     The given intervals are closed which means that the interval (
4880     start, end)
4881     includes both start and end.
4882     For each given interval, it is assumed that its start is less or
4883     equal its end.
4884     Your task is to determine whet...
4885 -----
4886 -----
4883 [INITIAL GENERATED CODE]
4884 def is_prime(n):
4885     if n <= 1:
4886         return False
4887     for i in range(2, int(n**0.5) + 1):
4888         if n % i == 0:
4889             return False
4890     return True
4891
4892 def intersection(interval1, interval2):
4893     start = max(interval1[0], interval2[0])
4894     end = min(interval1[1], interval2[1])
4895     if start > end:
4896         return "NO"
4897     length = end - start + 1
4898     return "YES" if is_prime(length) else "NO"
```

```
4899 -----
4900 [HumanEval/127] Attempt 0/3
4901 [HumanEval/127] ⚡ FAIL (attempt=0)
4902 [TEST ERROR PREVIEW]
4903 Traceback (most recent call last):
4904   File "C:\Users\JOIN\Downloads\Multi-
    Reviewer_Governance_for_Code_Generation\src\evaluator\
      humaneval_evaluator.py", line 39, in _worker
4905     ns["check"](candidate)
4906   File "<string>", line 5, in check
4907 AssertionError
4908 -----
4909 [HumanEval/127] [REPAIRED CODE] (after attempt=0)
4910 def is_prime(n):
4911     if n <= 1:
4912         return False
4913     for i in range(2, int(n**0.5) + 1):
4914         if n % i == 0:
4915             return False
4916     return True
4917
4918 def intersection(interval1, interval2):
4919     start = max(interval1[0], interval2[0])
4920     end = min(interval1[1], interval2[1])
4921     if start > end:
4922         return "NO"
4923     length = end - start + 1
4924     return "YES" if is_prime(length) else "NO"
4925 -----
4926 [HumanEval/127] Attempt 1/3
4927 [HumanEval/127] ⚡ FAIL (attempt=1)
4928 [TEST ERROR PREVIEW]
4929 Traceback (most recent call last):
4930   File "C:\Users\JOIN\Downloads\Multi-
    Reviewer_Governance_for_Code_Generation\src\evaluator\
      humaneval_evaluator.py", line 39, in _worker
4931     ns["check"](candidate)
4932   File "<string>", line 5, in check
4933 AssertionError
4934 -----
4935 [HumanEval/127] [REPAIRED CODE] (after attempt=1)
4936 def is_prime(n):
4937     if n <= 1:
4938         return False
4939     for i in range(2, int(n**0.5) + 1):
4940         if n % i == 0:
4941             return False
4942     return True
4943
```

```
4944 def intersection(interval1, interval2):  
4945     start = max(interval1[0], interval2[0])  
4946     end = min(interval1[1], interval2[1])  
4947     if start > end:  
4948         return "NO"  
4949     length = end - start + 1  
4950     return "YES" if is_prime(length) else "NO"  
4951 -----  
-----  
4952 [HumanEval/127] Attempt 2/3  
4953 [HumanEval/127] ⚡ FAIL (attempt=2)  
4954 [TEST ERROR PREVIEW]  
4955 Traceback (most recent call last):  
4956   File "C:\Users\JOIN\Downloads\Multi-  
    Reviewer_Governance_for_Code_Generation\src\evaluator\  
    humaneval_evaluator.py", line 39, in _worker  
4957     ns["check"](candidate)  
4958   File "<string>", line 5, in check  
4959 AssertionException  
4960 -----  
-----  
4961 [HumanEval/127] [REPAIRED CODE] (after attempt=2)  
4962 def is_prime(n):  
4963     if n <= 1:  
4964         return False  
4965     for i in range(2, int(n**0.5) + 1):  
4966         if n % i == 0:  
4967             return False  
4968     return True  
4969  
4970 def intersection(interval1, interval2):  
4971     start = max(interval1[0], interval2[0])  
4972     end = min(interval1[1], interval2[1])  
4973     if start > end:  
4974         return "NO"  
4975     length = end - start + 1  
4976     return "YES" if is_prime(length) else "NO"  
4977 -----  
-----  
4978 [HumanEval/127] Attempt 3/3  
4979 [HumanEval/127] ⚡ FAIL (attempt=3)  
4980 [TEST ERROR PREVIEW]  
4981 Traceback (most recent call last):  
4982   File "C:\Users\JOIN\Downloads\Multi-  
    Reviewer_Governance_for_Code_Generation\src\evaluator\  
    humaneval_evaluator.py", line 39, in _worker  
4983     ns["check"](candidate)  
4984   File "<string>", line 5, in check  
4985 AssertionException  
4986 -----  
-----  
4987 [HumanEval/127] ⚡ FINAL FAIL after 3 retries  
4988 =====
```

```
4988 =====
4989 [129/164] HumanEval/128
4990 =====
4991 [TASK] HumanEval/128 | entry_point=prod_signs
4992 [PROMPT PREVIEW]
4993
4994 def prod_signs(arr):
4995     """
4996         You are given an array arr of integers and you need to return
4997         sum of magnitudes of integers multiplied by product of all signs
4998         of each number in the array, represented by 1, -1 or 0.
4999         Note: return None for empty arr.
5000
5001     Example:
5002     >>> prod_signs([1, 2, 2, -4]) == -9
5003     >>> prod_signs([0, 1]) == 0
5004     >>> prod_signs([]) == None
5005     """
5006 -----
5007 -----
5007 [INITIAL GENERATED CODE]
5008 def prod_signs(arr):
5009     if not arr:
5010         return None
5011
5012     sign_product = 1
5013     magnitude_sum = 0
5014
5015     for num in arr:
5016         if num == 0:
5017             return 0
5018         elif num < 0:
5019             sign_product *= -1
5020
5021         magnitude_sum += abs(num)
5022
5023     return magnitude_sum * sign_product
5024 -----
5025 -----
5025 [HumanEval/128] Attempt 0/3
5026 [HumanEval/128] [] PASS (attempt=0)
5027 =====
5028 =====
5028 [130/164] HumanEval/129
5029 =====
5029 =====
5030 [TASK] HumanEval/129 | entry_point=minPath
5031 [PROMPT PREVIEW]
5032
5033 def minPath(grid, k):
5034     """
5035         Given a grid with N rows and N columns (N >= 2) and a positive
```

```
5035 integer k,
5036     each cell of the grid contains a value. Every integer in the
      range [1, N * N]
5037     inclusive appears exactly once on the cells of the grid.
5038
5039     You have to find the minimum path of length k in the grid. You
      can start
5040     from any cell, and in each step you can move to any of the
      neighb...
5041 -----
5042 -----
5042 [INITIAL GENERATED CODE]
5043 from heapq import heappop, heappush
5044
5045 def minPath(grid, k):
5046     N = len(grid)
5047     directions = [(0, 1), (1, 0), (-1, 0), (0, -1)]
5048
5049     def is_valid(x, y):
5050         return 0 <= x < N and 0 <= y < N
5051
5052     heap = []
5053     for i in range(N):
5054         for j in range(N):
5055             heappush(heap, (grid[i][j], [i, j]))
5056
5057     while k > 0:
5058         value, (x, y) = heappop(heap)
5059         if k == 1:
5060             return [value]
5061         for dx, dy in directions:
5062             nx, ny = x + dx, y + dy
5063             if is_valid(nx, ny):
5064                 heappush(heap, (grid[nx][ny], [nx, ny]))
5065         k -= 1
5066 -----
5067 -----
5067 [HumanEval/129] Attempt 0/3
5068 [HumanEval/129] ⚡ FAIL (attempt=0)
5069 [TEST ERROR PREVIEW]
5070 Traceback (most recent call last):
5071   File "C:\Users\JOIN\Downloads\Multi-
      Reviewer_Governance_for_Code_Generation\src\evaluator\
      humaneval_evaluator.py", line 39, in _worker
5072     ns["check"](candidate)
5073   File "<string>", line 5, in check
5074 AssertionError
5075 -----
5076 -----
5076 [HumanEval/129] [REPAIRED CODE] (after attempt=0)
5077 from heapq import heappop, heappush
5078
5079 def minPath(grid, k):
```

```
5080     N = len(grid)
5081     directions = [(0, 1), (1, 0), (-1, 0), (0, -1)]
5082
5083     def is_valid(x, y):
5084         return 0 <= x < N and 0 <= y < N
5085
5086     heap = []
5087     for i in range(N):
5088         for j in range(N):
5089             heappush(heap, (grid[i][j], [i, j]))
5090
5091     while k > 0:
5092         value, (x, y) = heappop(heap)
5093         if k == 1:
5094             return [value]
5095         for dx, dy in directions:
5096             nx, ny = x + dx, y + dy
5097             if is_valid(nx, ny):
5098                 heappush(heap, (grid[nx][ny], [nx, ny]))
5099         k -= 1
5100
5101     return []
5102 -----
5103 [HumanEval/129] Attempt 1/3
5104 [HumanEval/129] [] FAIL (attempt=1)
5105 [TEST ERROR PREVIEW]
5106 Traceback (most recent call last):
5107   File "C:\Users\JOIN\Downloads\Multi-
      Reviewer_Governance_for_Code_Generation\src\evaluator\
      humaneval_evaluator.py", line 39, in _worker
5108     ns["check"](candidate)
5109   File "<string>", line 5, in check
5110 AssertionException
5111 -----
5112 [HumanEval/129] [REPAIRED CODE] (after attempt=1)
5113 from heapq import heappop, heappush
5114
5115 def minPath(grid, k):
5116     N = len(grid)
5117     directions = [(0, 1), (1, 0), (-1, 0), (0, -1)]
5118
5119     def is_valid(x, y):
5120         return 0 <= x < N and 0 <= y < N
5121
5122     heap = []
5123     for i in range(N):
5124         for j in range(N):
5125             heappush(heap, (grid[i][j], [i, j]))
5126
5127     while k > 0:
5128         value, (x, y) = heappop(heap)
```

```
5129         if k == 1:
5130             return [value]
5131         for dx, dy in directions:
5132             nx, ny = x + dx, y + dy
5133             if is_valid(nx, ny):
5134                 heappush(heap, (grid[nx][ny], [nx, ny]))
5135         k -= 1
5136
5137     return []
5138 -----
-----
5139 [HumanEval/129] Attempt 2/3
5140 [HumanEval/129] ⚡ FAIL (attempt=2)
5141 [TEST ERROR PREVIEW]
5142 Traceback (most recent call last):
5143   File "C:\Users\JOIN\Downloads\Multi-
      Reviewer_Governance_for_Code_Generation\src\evaluator\
        humaneval_evaluator.py", line 39, in _worker
5144     ns["check"](candidate)
5145   File "<string>", line 5, in check
5146 AssertionError
5147 -----
-----
5148 [HumanEval/129] [REPAIRED CODE] (after attempt=2)
5149 from heapq import heappop, heappush
5150
5151 def minPath(grid, k):
5152     N = len(grid)
5153     directions = [(0, 1), (1, 0), (-1, 0), (0, -1)]
5154
5155     def is_valid(x, y):
5156         return 0 <= x < N and 0 <= y < N
5157
5158     heap = []
5159     for i in range(N):
5160         for j in range(N):
5161             heappush(heap, (grid[i][j], [i, j]))
5162
5163     while k > 0:
5164         value, (x, y) = heappop(heap)
5165         if k == 1:
5166             return [value]
5167         for dx, dy in directions:
5168             nx, ny = x + dx, y + dy
5169             if is_valid(nx, ny):
5170                 heappush(heap, (grid[nx][ny], [nx, ny]))
5171         k -= 1
5172
5173     return []
5174 -----
-----
5175 [HumanEval/129] Attempt 3/3
5176 [HumanEval/129] ⚡ FAIL (attempt=3)
```

```
5177 [TEST ERROR PREVIEW]
5178 Traceback (most recent call last):
5179   File "C:\Users\JOIN\Downloads\Multi-
      Reviewer_Governance_for_Code_Generation\src\evaluator\
        humaneval_evaluator.py", line 39, in _worker
5180     ns["check"]的文化)
5181   File "<string>", line 5, in check
5182 AssertionError
5183 -----
-----
5184 [HumanEval/129] Ø FINAL FAIL after 3 retries
5185 =====
=====
5186 [131/164] HumanEval/130
5187 =====
=====
5188 [TASK] HumanEval/130 | entry_point=tri
5189 [PROMPT PREVIEW]
5190
5191 def tri(n):
5192     """Everyone knows Fibonacci sequence, it was studied deeply by
      mathematicians in
5193     the last couple centuries. However, what people don't know is
      Tribonacci sequence.
5194     Tribonacci sequence is defined by the recurrence:
5195     tri(1) = 3
5196     tri(n) = 1 + n / 2, if n is even.
5197     tri(n) = tri(n - 1) + tri(n - 2) + tri(n + 1), if n is odd.
5198     For example:
5199     tri(2) = 1 + (2 / 2)...
5200 -----
-----
5201 [INITIAL GENERATED CODE]
5202 def tri(n):
5203     if n == 0:
5204         return [1]
5205     elif n == 1:
5206         return [1, 3]
5207
5208     sequence = [1, 3, 2]
5209     for i in range(3, n + 1):
5210         if i % 2 == 0:
5211             next_value = 1 + i / 2
5212         else:
5213             next_value = sequence[i - 1] + sequence[i - 2] +
      sequence[i + 1]
5214         sequence.append(next_value)
5215
5216     return sequence
5217 -----
-----
5218 [HumanEval/130] Attempt 0/3
5219 [HumanEval/130] Ø FAIL (attempt=0)
```

```
5220 [TEST ERROR PREVIEW]
5221 Traceback (most recent call last):
5222   File "C:\Users\JOIN\Downloads\Multi-
      Reviewer_Governance_for_Code_Generation\src\evaluator\
        humaneval_evaluator.py", line 39, in _worker
5223     ns["check"](candidate)
5224   File "<string>", line 5, in check
5225   File "<string>", line 31, in tri
5226 IndexError: list index out of range
5227 -----
5228 -----
5229 [HumanEval/130] [REPAIRED CODE] (after attempt=0)
5230 def tri(n):
5231     if n == 0:
5232         return [1]
5233     elif n == 1:
5234         return [1, 3]
5235     sequence = [1, 3, 2]
5236     for i in range(3, n + 1):
5237         if i % 2 == 0:
5238             next_value = 1 + i / 2
5239         else:
5240             next_value = sequence[i - 1] + sequence[i - 2]
5241         sequence.append(next_value)
5242
5243     return sequence
5244 -----
5245 -----
5246 [HumanEval/130] Attempt 1/3
5247 [HumanEval/130] ⚡ FAIL (attempt=1)
5248 [TEST ERROR PREVIEW]
5249 Traceback (most recent call last):
5250   File "C:\Users\JOIN\Downloads\Multi-
      Reviewer_Governance_for_Code_Generation\src\evaluator\
        humaneval_evaluator.py", line 39, in _worker
5251     ns["check"](candidate)
5252   File "<string>", line 5, in check
5253 AssertionTypeError
5254 -----
5255 -----
5256 [HumanEval/130] [REPAIRED CODE] (after attempt=1)
5257 def tri(n):
5258     if n == 0:
5259         return [1]
5260     elif n == 1:
5261         return [1, 3]
5262     sequence = [1, 3, 2]
5263     for i in range(3, n + 1):
5264         if i % 2 == 0:
5265             next_value = int(1 + i / 2)
5266         else:
```

```
5266         next_value = sequence[i - 1] + sequence[i - 2]
5267         sequence.append(next_value)
5268
5269     return sequence
5270 -----
5271 [HumanEval/130] Attempt 2/3
5272 [HumanEval/130] ⚡ FAIL (attempt=2)
5273 [TEST ERROR PREVIEW]
5274 Traceback (most recent call last):
5275   File "C:\Users\JOIN\Downloads\Multi-
      Reviewer_Governance_for_Code_Generation\src\evaluator\
        humaneval_evaluator.py", line 39, in _worker
5276     ns["check"](candidate)
5277   File "<string>", line 5, in check
5278 AssertionError
5279 -----
5280 [HumanEval/130] [REPAIRED CODE] (after attempt=2)
5281 def tri(n):
5282     if n == 0:
5283         return [1]
5284     elif n == 1:
5285         return [1, 3]
5286
5287     sequence = [1, 3, 2]
5288     for i in range(3, n + 1):
5289         if i % 2 == 0:
5290             next_value = int(1 + i / 2)
5291         else:
5292             next_value = sequence[i - 1] + sequence[i - 2]
5293         sequence.append(next_value)
5294
5295     return sequence
5296 -----
5297 [HumanEval/130] Attempt 3/3
5298 [HumanEval/130] ⚡ FAIL (attempt=3)
5299 [TEST ERROR PREVIEW]
5300 Traceback (most recent call last):
5301   File "C:\Users\JOIN\Downloads\Multi-
      Reviewer_Governance_for_Code_Generation\src\evaluator\
        humaneval_evaluator.py", line 39, in _worker
5302     ns["check"](candidate)
5303   File "<string>", line 5, in check
5304 AssertionError
5305 -----
5306 [HumanEval/130] ⚡ FINAL FAIL after 3 retries
5307 =====
5308 [132/164] HumanEval/131
5309 =====
```

```
5309 =====
5310 [TASK] HumanEval/131 | entry_point=digits
5311 [PROMPT PREVIEW]
5312
5313 def digits(n):
5314     """Given a positive integer n, return the product of the odd
5315     digits.
5316     Return 0 if all digits are even.
5317     For example:
5318         digits(1) == 1
5319         digits(4) == 0
5320         digits(235) == 15
5321         """
5321 -----
5322 -----
5322 [INITIAL GENERATED CODE]
5323 def digits(n):
5324     product = 1
5325     has_odd = False
5326     for digit in str(n):
5327         if int(digit) % 2 != 0:
5328             product *= int(digit)
5329             has_odd = True
5330     return product if has_odd else 0
5331 -----
5332 -----
5332 [HumanEval/131] Attempt 0/3
5333 [HumanEval/131] [] PASS (attempt=0)
5334 =====
5335 -----
5335 [133/164] HumanEval/132
5336 =====
5336 -----
5337 [TASK] HumanEval/132 | entry_point=is_nested
5338 [PROMPT PREVIEW]
5339
5340 def is_nested(string):
5341     '''
5342     Create a function that takes a string as input which contains
5343     only square brackets.
5344     The function should return True if and only if there is a valid
5345     subsequence of brackets
5346     where at least one bracket in the subsequence is nested.
5347
5348     is_nested('[]') → True
5349     is_nested('[[[]]])])[[[[[]') → False
5350     is_nested('[][]') → False
5351     is_nested('[]') → Fals...
5350 -----
5351 -----
5351 [INITIAL GENERATED CODE]
5352 def is_nested(string):
5353     stack = []
```

```
5354     nested = False
5355     for char in string:
5356         if char == '[':
5357             stack.append(char)
5358         elif char == ']' and stack:
5359             stack.pop()
5360         if stack:
5361             nested = True
5362     return nested and not stack
5363 -----
-----
5364 [HumanEval/132] Attempt 0/3
5365 [HumanEval/132] ⚡ FAIL (attempt=0)
5366 [TEST ERROR PREVIEW]
5367 Traceback (most recent call last):
5368   File "C:\Users\JOIN\Downloads\Multi-
      Reviewer_Governance_for_Code_Generation\src\evaluator\
      humaneval_evaluator.py", line 39, in _worker
5369     ns["check"](candidate)
5370   File "<string>", line 13, in check
5371 AssertionError
5372 -----
-----
5373 [HumanEval/132] [REPAIRED CODE] (after attempt=0)
5374 def is_nested(string):
5375     stack = []
5376     nested = False
5377     for char in string:
5378         if char == '[':
5379             stack.append(char)
5380         elif char == ']' and stack:
5381             stack.pop()
5382         if stack:
5383             nested = True
5384     return nested and not stack
5385 -----
-----
5386 [HumanEval/132] Attempt 1/3
5387 [HumanEval/132] ⚡ FAIL (attempt=1)
5388 [TEST ERROR PREVIEW]
5389 Traceback (most recent call last):
5390   File "C:\Users\JOIN\Downloads\Multi-
      Reviewer_Governance_for_Code_Generation\src\evaluator\
      humaneval_evaluator.py", line 39, in _worker
5391     ns["check"](candidate)
5392   File "<string>", line 13, in check
5393 AssertionError
5394 -----
-----
5395 [HumanEval/132] [REPAIRED CODE] (after attempt=1)
5396 def is_nested(string):
5397     stack = []
5398     nested = False
```

```
5399     for char in string:
5400         if char == '[':
5401             stack.append(char)
5402         elif char == ']' and stack:
5403             stack.pop()
5404         if stack:
5405             nested = True
5406     return nested and len(stack) == 0
5407 -----
5408 -----
5408 [HumanEval/132] Attempt 2/3
5409 [HumanEval/132] ⚡ FAIL (attempt=2)
5410 [TEST ERROR PREVIEW]
5411 Traceback (most recent call last):
5412   File "C:\Users\JOIN\Downloads\Multi-
      Reviewer_Governance_for_Code_Generation\src\evaluator\
      humaneval_evaluator.py", line 39, in _worker
5413     ns["check"](candidate)
5414   File "<string>", line 13, in check
5415 AssertionError
5416 -----
5417 -----
5417 [HumanEval/132] [REPAIRED CODE] (after attempt=2)
5418 def is_nested(string):
5419     stack = []
5420     nested = False
5421     for char in string:
5422         if char == '[':
5423             stack.append(char)
5424         elif char == ']' and stack:
5425             stack.pop()
5426         if stack:
5427             nested = True
5428     return nested and len(stack) == 0
5429 -----
5430 -----
5430 [HumanEval/132] Attempt 3/3
5431 [HumanEval/132] ⚡ FAIL (attempt=3)
5432 [TEST ERROR PREVIEW]
5433 Traceback (most recent call last):
5434   File "C:\Users\JOIN\Downloads\Multi-
      Reviewer_Governance_for_Code_Generation\src\evaluator\
      humaneval_evaluator.py", line 39, in _worker
5435     ns["check"](candidate)
5436   File "<string>", line 13, in check
5437 AssertionError
5438 -----
5439 -----
5439 [HumanEval/132] ⚡ FINAL FAIL after 3 retries
5440 =====
5441 =====
5441 [134/164] HumanEval/133
5442 =====
```

```
5442 =====
5443 [TASK] HumanEval/133 | entry_point=sum_squares
5444 [PROMPT PREVIEW]
5445
5446
5447 def sum_squares(lst):
5448     """You are given a list of numbers.
5449     You need to return the sum of squared numbers in the given list,
5450     round each element in the list to the upper int(Ceiling) first.
5451     Examples:
5452         For lst = [1,2,3] the output should be 14
5453         For lst = [1,4,9] the output should be 98
5454         For lst = [1,3,5,7] the output should be 84
5455         For lst = [1.4,4.2,0] the output should b...
5456 -----
5457 -----
5458 [INITIAL GENERATED CODE]
5459 import math
5460
5461 def sum_squares(lst):
5462     return sum(math.ceil(x) ** 2 for x in lst)
5463 -----
5464 -----
5465 [HumanEval/133] Attempt 0/3
5466 [HumanEval/133] [] PASS (attempt=0)
5467 =====
5468 =====
5469 [135/164] HumanEval/134
5470 =====
5471 def check_if_last_char_is_a_letter(txt):
5472     '''
5473         Create a function that returns True if the last character
5474         of a given string is an alphabetical character and is not
5475         a part of a word, and False otherwise.
5476         Note: "word" is a group of characters separated by space.
5477
5478     Examples:
5479         check_if_last_char_is_a_letter("apple pie") → False
5480         check_if_last_char_is_a_letter("apple pi e") →...
5481 -----
5482 -----
5483 [INITIAL GENERATED CODE]
5484 def check_if_last_char_is_a_letter(txt):
5485     if not txt or not txt[-1].isalpha():
5486         return False
5487     if len(txt) == 1:
5488         return True
5489     return txt[-2] == ' '
```

```
5489 -----
5490 [HumanEval/134] Attempt 0/3
5491 [HumanEval/134] Ø PASS (attempt=0)
5492 =====
5493 =====
5493 [136/164] HumanEval/135
5494 =====
5494 =====
5495 [TASK] HumanEval/135 | entry_point=can_arrange
5496 [PROMPT PREVIEW]
5497
5498 def can_arrange(arr):
5499     """Create a function which returns the largest index of an
      element which
5500         is not greater than or equal to the element immediately
      preceding it. If
5501             no such element exists then return -1. The given array will not
      contain
5502                 duplicate values.
5503
5504     Examples:
5505         can_arrange([1,2,4,3,5]) = 3
5506         can_arrange([1,2,3]) = -1
5507         """
5508 -----
5508 -----
5509 [INITIAL GENERATED CODE]
5510 def can_arrange(arr):
5511     for i in range(len(arr) - 1, 0, -1):
5512         if arr[i] < arr[i - 1]:
5513             return i
5514     return -1
5515 -----
5515 -----
5516 [HumanEval/135] Attempt 0/3
5517 [HumanEval/135] Ø PASS (attempt=0)
5518 =====
5518 -----
5519 =====
5519 [137/164] HumanEval/136
5520 =====
5520 -----
5521 [TASK] HumanEval/136 | entry_point=largest_smallest_integers
5522 [PROMPT PREVIEW]
5523
5524 def largest_smallest_integers(lst):
5525     """
5526         Create a function that returns a tuple (a, b), where 'a' is
5527             the largest of negative integers, and 'b' is the smallest
5528                 of positive integers in a list.
5529             If there is no negative or positive integers, return them as
      None.
5530
5531     Examples:
```

```
5532     largest_smallest_integers([2, 4, 1, 3, 5, 7]) == (None, 1)
5533     largest_smallest_integers([]) == (None, ...)
5534 -----
5535 -----
5535 [INITIAL GENERATED CODE]
5536 def largest_smallest_integers(lst):
5537     negatives = [x for x in lst if x < 0]
5538     positives = [x for x in lst if x > 0]
5539
5540     largest_negative = max(negatives) if negatives else None
5541     smallest_positive = min(positives) if positives else None
5542
5543     return (largest_negative, smallest_positive)
5544 -----
5545 -----
5545 [HumanEval/136] Attempt 0/3
5546 [HumanEval/136] [] PASS (attempt=0)
5547 =====
5548 =====
5548 [138/164] HumanEval/137
5549 =====
5549 =====
5550 [TASK] HumanEval/137 | entry_point=compare_one
5551 [PROMPT PREVIEW]
5552
5553 def compare_one(a, b):
5554     """
5555         Create a function that takes integers, floats, or strings
5556         representing
5557         real numbers, and returns the larger variable in its given
5558         variable type.
5559         Return None if the values are equal.
5560         Note: If a real number is represented as a string, the floating
5561         point might be . or ,
5562         compare_one(1, 2.5) → 2.5
5563         compare_one(1, "2,3") → "2,3"
5564         compare_one("5,....")
5565 -----
5565 -----
5564 [INITIAL GENERATED CODE]
5565 def compare_one(a, b):
5566     def convert_to_float(value):
5567         if isinstance(value, str):
5568             value = value.replace(',', '.')
5569         return float(value)
5570
5571     a_float = convert_to_float(a)
5572     b_float = convert_to_float(b)
5573
5574     if a_float == b_float:
5575         return None
5576     elif a_float > b_float:
```

```
5577         return a
5578     else:
5579         return b
5580 -----
5581 [HumanEval/137] Attempt 0/3
5582 [HumanEval/137] Ø PASS (attempt=0)
5583 =====
5584 [139/164] HumanEval/138
5585 =====
5586 [TASK] HumanEval/138 | entry_point=is_equal_to_sum_even
5587 [PROMPT PREVIEW]
5588
5589 def is_equal_to_sum_even(n):
5590     """Evaluate whether the given number n can be written as the sum
5591     of exactly 4 positive even numbers
5592     Example
5593     is_equal_to_sum_even(4) == False
5594     is_equal_to_sum_even(6) == False
5595     is_equal_to_sum_even(8) == True
5596     """
5597 -----
5598 [INITIAL GENERATED CODE]
5599 def is_equal_to_sum_even(n):
5600     return n >= 8 and n % 2 == 0
5601 -----
5602 [HumanEval/138] Attempt 0/3
5603 [HumanEval/138] Ø PASS (attempt=0)
5604 =====
5605 [140/164] HumanEval/139
5606 =====
5607 [TASK] HumanEval/139 | entry_point=special_factorial
5608 [PROMPT PREVIEW]
5609
5610 def special_factorial(n):
5611     """The Brazilian factorial is defined as:
5612     brazilian_factorial(n) = n! * (n-1)! * (n-2)! * ... * 1!
5613     where n > 0
5614
5615     For example:
5616     >>> special_factorial(4)
5617     288
5618
5619     The function will receive an integer as input and should return
5620     the special
5621     factorial of this integer.
5622     """
5623
```

```
5621 -----
-----
5622 [INITIAL GENERATED CODE]
5623 import math
5624
5625 def special_factorial(n):
5626     result = 1
5627     for i in range(1, n + 1):
5628         result *= math.factorial(i)
5629     return result
5630 -----
-----
5631 [HumanEval/139] Attempt 0/3
5632 [HumanEval/139] Ø PASS (attempt=0)
5633 =====
=====
5634 [141/164] HumanEval/140
5635 =====
=====
5636 [TASK] HumanEval/140 | entry_point=fix_spaces
5637 [PROMPT PREVIEW]
5638
5639 def fix_spaces(text):
5640     """
5641         Given a string text, replace all spaces in it with underscores,
5642         and if a string has more than 2 consecutive spaces,
5643         then replace all consecutive spaces with -
5644
5645     fix_spaces("Example") == "Example"
5646     fix_spaces("Example 1") == "Example_1"
5647     fix_spaces(" Example 2") == "_Example_2"
5648     fix_spaces(" Example 3") == "_Example-3"
5649     """
5650 -----
-----
5651 [INITIAL GENERATED CODE]
5652 def fix_spaces(text):
5653     return text.replace(' ', '-').replace(' ', '_')
5654 -----
-----
5655 [HumanEval/140] Attempt 0/3
5656 [HumanEval/140] Ø PASS (attempt=0)
5657 =====
=====
5658 [142/164] HumanEval/141
5659 =====
=====
5660 [TASK] HumanEval/141 | entry_point=file_name_check
5661 [PROMPT PREVIEW]
5662
5663 def file_name_check(file_name):
5664     """Create a function which takes a string representing a file's
      name, and returns
```

```
5665     'Yes' if the file's name is valid, and returns 'No'
      otherwise.
5666     A file's name is considered to be valid if and only if all the
      following conditions
5667     are met:
5668         - There should not be more than three digits ('0'-'9') in the
            file's name.
5669         - The file's name contain...
5670 -----
-----
5671 [INITIAL GENERATED CODE]
5672 import re
5673
5674 def file_name_check(file_name):
5675     pattern = r'^[a-zA-Z][^\d]*\d{0,3}[^\d]*\.(txt|exe|dll)$'
5676     return 'Yes' if re.match(pattern, file_name) else 'No'
5677 -----
-----
5678 [HumanEval/141] Attempt 0/3
5679 [HumanEval/141] ⚡ FAIL (attempt=0)
5680 [TEST ERROR PREVIEW]
5681 Traceback (most recent call last):
5682   File "C:\Users\JOIN\Downloads\Multi-
      Reviewer_Governance_for_Code_Generation\src\evaluator\
      humaneval_evaluator.py", line 39, in _worker
5683     ns["check"](candidate)
5684   File "<string>", line 8, in check
5685 AssertionError
5686 -----
-----
5687 [HumanEval/141] [REPAIRED CODE] (after attempt=0)
5688 import re
5689
5690 def file_name_check(file_name):
5691     pattern = r'^[a-zA-Z][^\d]*\d{0,3}[^\d]*\.(txt|exe|dll)$'
5692     if re.match(pattern, file_name):
5693         return 'Yes'
5694     else:
5695         raise AssertionError(f"Invalid file name: {file_name}")
5696 -----
-----
5697 [HumanEval/141] Attempt 1/3
5698 [HumanEval/141] ⚡ FAIL (attempt=1)
5699 [TEST ERROR PREVIEW]
5700 Traceback (most recent call last):
5701   File "C:\Users\JOIN\Downloads\Multi-
      Reviewer_Governance_for_Code_Generation\src\evaluator\
      humaneval_evaluator.py", line 39, in _worker
5702     ns["check"](candidate)
5703   File "<string>", line 5, in check
5704   File "<string>", line 24, in file_name_check
5705 AssertionError: Invalid file name: 1example.dll
5706 -----
```

```
5706 -----
5707 [HumanEval/141] [REPAIRED CODE] (after attempt=1)
5708 import re
5709
5710 def file_name_check(file_name):
5711     pattern = r'^[a-zA-Z][^\d]*\d{0,3}[^\d]*\.(txt|exe|dll)$'
5712     if re.match(pattern, file_name):
5713         return 'Yes'
5714     else:
5715         raise AssertionError(f"Invalid file name: {file_name}")
5716 -----
5717 -----
5718 [HumanEval/141] Attempt 2/3
5719 [HumanEval/141] ⚡ FAIL (attempt=2)
5720 [TEST ERROR PREVIEW]
5721 Traceback (most recent call last):
5722   File "C:\Users\JOIN\Downloads\Multi-
      Reviewer_Governance_for_Code_Generation\src\evaluator\
        humaneval_evaluator.py", line 39, in _worker
5723     ns["check"](candidate)
5724   File "<string>", line 5, in check
5725   File "<string>", line 24, in file_name_check
5726 AssertionError: Invalid file name: 1example.dll
5727 -----
5728 -----
5729 [HumanEval/141] [REPAIRED CODE] (after attempt=2)
5730 import re
5731
5732 def file_name_check(file_name):
5733     pattern = r'^[a-zA-Z][^\d]*\d{0,3}[^\d]*\.(txt|exe|dll)$'
5734     if re.match(pattern, file_name):
5735         return 'Yes'
5736     else:
5737         raise AssertionError(f"Invalid file name: {file_name}")
5738 -----
5739 -----
5740 [HumanEval/141] Attempt 3/3
5741 [HumanEval/141] ⚡ FAIL (attempt=3)
5742 [TEST ERROR PREVIEW]
5743 Traceback (most recent call last):
5744   File "C:\Users\JOIN\Downloads\Multi-
      Reviewer_Governance_for_Code_Generation\src\evaluator\
        humaneval_evaluator.py", line 39, in _worker
5745   File "<string>", line 5, in check
5746   File "<string>", line 24, in file_name_check
5747 AssertionError: Invalid file name: 1example.dll
5748 -----
5749 [143/164] HumanEval/142
```

```
5750 =====
=====
5751 [TASK] HumanEval/142 | entry_point=sum_squares
5752 [PROMPT PREVIEW]
5753
5754
5755
5756 def sum_squares(lst):
5757     """
5758         This function will take a list of integers. For all entries in
5759         the list, the function shall square the integer entry if its index
5760         is a
5761             multiple of 3 and will cube the integer entry if its index is a
5762             multiple of 4 and not a multiple of 3. The function will not
5763             change the entries in the list whose indexes are not a multiple
5764             of 3 or 4. The function shal...
5765 -----
5766 -----
5767 [INITIAL GENERATED CODE]
5768 def sum_squares(lst):
5769     total_sum = 0
5770     for i, num in enumerate(lst):
5771         if i % 3 == 0:
5772             total_sum += num ** 2
5773         elif i % 4 == 0:
5774             total_sum += num ** 3
5775         else:
5776             total_sum += num
5777     return total_sum
5778 -----
5779 [HumanEval/142] Attempt 0/3
5780 [HumanEval/142] [] PASS (attempt=0)
5781 =====
=====
5782 [144/164] HumanEval/143
5783 =====
=====
5784 [TASK] HumanEval/143 | entry_point=words_in_sentence
5785 [PROMPT PREVIEW]
5786
5787 def words_in_sentence(sentence):
5788     """
5789         You are given a string representing a sentence,
5790         the sentence contains some words separated by a space,
5791         and you have to return a string that contains the words from the
5792         original sentence,
5793         whose lengths are prime numbers,
5794         the order of the words in the new string should be the same as
5795         the original one.
5796
5797         Example 1:
5798             Input: senten...
```

```
5792 -----
-----
5793 [INITIAL GENERATED CODE]
5794 def is_prime(n):
5795     if n <= 1:
5796         return False
5797     for i in range(2, int(n**0.5) + 1):
5798         if n % i == 0:
5799             return False
5800     return True
5801
5802 def words_in_sentence(sentence):
5803     words = sentence.split()
5804     prime_length_words = [word for word in words if is_prime(len(
5805         word))]
5806     return ' '.join(prime_length_words)
5807 -----
5808 [HumanEval/143] Attempt 0/3
5809 [HumanEval/143] 0 PASS (attempt=0)
5810 =====
5811 =====
5812 [145/164] HumanEval/144
5813 =====
5814 =====
5815 [TASK] HumanEval/144 | entry_point=simplify
5816 [PROMPT PREVIEW]
5817
5818 def simplify(x, n):
5819     """Your task is to implement a function that will simplify the
5820     expression
5821     x * n. The function returns True if x * n evaluates to a whole
5822     number and False
5823     otherwise. Both x and n, are string representation of a fraction
5824     , and have the following format,
5825     <numerator>/<denominator> where both numerator and denominator
5826     are positive whole numbers.
5827
5828 You can assume t...
5829
5830 -----
5831 [INITIAL GENERATED CODE]
5832 def simplify(x, n):
5833     numerator_x, denominator_x = map(int, x.split('/'))
5834     numerator_n, denominator_n = map(int, n.split('/'))
5835
5836     product_numerator = numerator_x * numerator_n
5837     product_denominator = denominator_x * denominator_n
5838
5839     return product_numerator % product_denominator == 0
5840
5841 -----
5842 [HumanEval/144] Attempt 0/3
```

```
5834 [HumanEval/144] Ø PASS (attempt=0)
5835 =====
5836 [146/164] HumanEval/145
5837 =====
5838 [TASK] HumanEval/145 | entry_point=order_by_points
5839 [PROMPT PREVIEW]
5840
5841 def order_by_points(nums):
5842     """
5843         Write a function which sorts the given list of integers
5844         in ascending order according to the sum of their digits.
5845         Note: if there are several items with similar sum of their
5846             digits,
5847             order them based on their index in original list.
5848
5849     For example:
5850     >>> order_by_points([1, 11, -1, -11, -12]) == [-1, -11, 1, -12,
5851           11]
5850     >>> order_by_points([])
5851 -----
5852 -----
5852 [INITIAL GENERATED CODE]
5853 def order_by_points(nums):
5854     return sorted(nums, key=lambda x: (sum(int(digit) for digit in
5855     str(abs(x))), nums.index(x)))
5855 -----
5856 -----
5856 [HumanEval/145] Attempt 0/3
5857 [HumanEval/145] Ø FAIL (attempt=0)
5858 [TEST ERROR PREVIEW]
5859 Traceback (most recent call last):
5860   File "C:\Users\JOIN\Downloads\Multi-
5861     Reviewer_Governance_for_Code_Generation\src\evaluator\
5862       humaneval_evaluator.py", line 39, in _worker
5863     ns["check"](candidate)
5862   File "<string>", line 4, in check
5863 AssertionError
5864 -----
5865 -----
5865 [HumanEval/145] [REPAIRED CODE] (after attempt=0)
5866 def order_by_points(nums):
5867     return sorted(nums, key=lambda x: (sum(int(digit) for digit in
5868     str(abs(x))), nums.index(x)))
5868 -----
5869 -----
5869 [HumanEval/145] Attempt 1/3
5870 [HumanEval/145] Ø FAIL (attempt=1)
5871 [TEST ERROR PREVIEW]
5872 Traceback (most recent call last):
5873   File "C:\Users\JOIN\Downloads\Multi-
      Reviewer_Governance_for_Code_Generation\src\evaluator\
```

```
5873 humaneval_evaluator.py", line 39, in _worker
5874     ns["check"](candidate)
5875 File "<string>", line 4, in check
5876 AssertionError
5877 -----
5878 [HumanEval/145] [REPAIRED CODE] (after attempt=1)
5879 def order_by_points(nums):
5880     return sorted(nums, key=lambda x: (sum(int(digit) for digit in
5881         str(abs(x))), nums.index(x)))
5882 -----
5883 [HumanEval/145] Attempt 2/3
5884 [HumanEval/145] ⚡ FAIL (attempt=2)
5885 [TEST ERROR PREVIEW]
5886 Traceback (most recent call last):
5887   File "C:\Users\JOIN\Downloads\Multi-
5888     Reviewer_Governance_for_Code_Generation\src\evaluator\
5889       humaneval_evaluator.py", line 39, in _worker
5890         ns["check"](candidate)
5891     File "<string>", line 4, in check
5892   AssertionError
5893 -----
5894 [HumanEval/145] [REPAIRED CODE] (after attempt=2)
5895 def order_by_points(nums):
5896     return sorted(nums, key=lambda x: (sum(int(digit) for digit in
5897         str(abs(x))), nums.index(x)))
5898 -----
5899 [HumanEval/145] Attempt 3/3
5900 [HumanEval/145] ⚡ FAIL (attempt=3)
5901 [TEST ERROR PREVIEW]
5902 Traceback (most recent call last):
5903   File "C:\Users\JOIN\Downloads\Multi-
5904     Reviewer_Governance_for_Code_Generation\src\evaluator\
5905       humaneval_evaluator.py", line 39, in _worker
5906         ns["check"](candidate)
5907     File "<string>", line 4, in check
5908   AssertionError
5909 -----
5910 [HumanEval/145] ⚡ FINAL FAIL after 3 retries
5911 =====
5912 =====
5913 [147/164] HumanEval/146
5914 =====
5915 =====
5916 [TASK] HumanEval/146 | entry_point=specialFilter
5917 [PROMPT PREVIEW]
5918
5919 def specialFilter(nums):
5920     """Write a function that takes an array of numbers as input and
```

```

5912 returns
5913     the number of elements in the array that are greater than 10 and
      both
5914     first and last digits of a number are odd (1, 3, 5, 7, 9).
5915     For example:
5916     specialFilter([15, -73, 14, -15]) => 1
5917     specialFilter([33, -2, -3, 45, 21, 109]) => 2
5918     """
5919 -----
-----
5920 [INITIAL GENERATED CODE]
5921 def specialFilter(nums):
5922     count = 0
5923     for num in nums:
5924         if num > 10:
5925             str_num = str(num)
5926             if int(str_num[0]) % 2 != 0 and int(str_num[-1]) % 2 !=
0:
5927                 count += 1
5928     return count
5929 -----
-----
5930 [HumanEval/146] Attempt 0/3
5931 [HumanEval/146] [] PASS (attempt=0)
5932 =====
=====
5933 [148/164] HumanEval/147
5934 =====
=====
5935 [TASK] HumanEval/147 | entry_point=get_max_triples
5936 [PROMPT PREVIEW]
5937
5938 def get_max_triples(n):
5939     """
5940     You are given a positive integer n. You have to create an
      integer array a of length n.
5941     For each i (1 ≤ i ≤ n), the value of a[i] = i * i - i + 1.
5942     Return the number of triples (a[i], a[j], a[k]) of a where i
      < j < k,
5943     and a[i] + a[j] + a[k] is a multiple of 3.
5944
5945     Example :
5946         Input: n = 5
5947         Output: 1
5948         Explanation:...
5949 -----
-----
5950 [INITIAL GENERATED CODE]
5951 def get_max_triples(n):
5952     a = [i * i - i + 1 for i in range(1, n + 1)]
5953     count = 0
5954     for i in range(n):
5955         for j in range(i + 1, n):

```

```
5956         for k in range(j + 1, n):
5957             if (a[i] + a[j] + a[k]) % 3 == 0:
5958                 count += 1
5959     return count
5960 -----
5961 -----
5961 [HumanEval/147] Attempt 0/3
5962 [HumanEval/147] Ø PASS (attempt=0)
5963 =====
5964 =====
5964 [149/164] HumanEval/148
5965 =====
5966 =====
5966 [TASK] HumanEval/148 | entry_point=bf
5967 [PROMPT PREVIEW]
5968
5969 def bf(planet1, planet2):
5970     '''
5971         There are eight planets in our solar system: the closerst to the
5971         Sun
5972         is Mercury, the next one is Venus, then Earth, Mars, Jupiter,
5972         Saturn,
5973         Uranus, Neptune.
5974         Write a function that takes two planet names as strings planet1
5974         and planet2.
5975         The function should return a tuple containing all planets whose
5975         orbits are
5976         located between the orbit of...
5977 -----
5978 -----
5978 [INITIAL GENERATED CODE]
5979 def bf(planet1, planet2):
5980     planets = ["Mercury", "Venus", "Earth", "Mars", "Jupiter", "
5980         Saturn", "Uranus", "Neptune"]
5981     if planet1 not in planets or planet2 not in planets:
5982         return ()
5983
5984     index1, index2 = planets.index(planet1), planets.index(planet2)
5985     if index1 > index2:
5986         index1, index2 = index2, index1
5987
5988     return tuple(planets[index1 + 1:index2])
5989 -----
5990 -----
5990 [HumanEval/148] Attempt 0/3
5991 [HumanEval/148] Ø PASS (attempt=0)
5992 =====
5993 =====
5993 [150/164] HumanEval/149
5994 =====
5995 =====
5995 [TASK] HumanEval/149 | entry_point=sorted_list_sum
5996 [PROMPT PREVIEW]
```

```
5997
5998 def sorted_list_sum(lst):
5999     """Write a function that accepts a list of strings as a
6000     parameter,
6001     deletes the strings that have odd lengths from it,
6002     and returns the resulted list with a sorted order,
6003     The list is always a list of strings and never an array of
6004     numbers,
6005     and it may contain duplicates.
6006     The order of the list should be ascending by length of each word
6007     , and you
6008     sho...
6009 -----
6010 -----
6011 [INITIAL GENERATED CODE]
6012 def sorted_list_sum(lst):
6013     return sorted([word for word in lst if len(word) % 2 == 0], key=
6014     lambda x: (len(x), x))
6015 -----
6016 -----
6017 [HumanEval/149] Attempt 0/3
6018 [HumanEval/149] 0 PASS (attempt=0)
6019 =====
6020 =====
6021 =====
6022 =====
6023 [151/164] HumanEval/150
6024 =====
6025 =====
6026 =====
6027 =====
6028 -----
6029 -----
6030 [INITIAL GENERATED CODE]
6031 def is_prime(num):
6032     if num <= 1:
6033         return False
6034     for i in range(2, int(num**0.5) + 1):
6035         if num % i == 0:
6036             return False
6037     return True
6038 -----
6039 def x_or_y(n, x, y):
6040     return x if is_prime(n) else y
6041 -----
```

```
6040 -----
6041 [HumanEval/150] Attempt 0/3
6042 [HumanEval/150] 0 PASS (attempt=0)
6043 =====
=====
6044 [152/164] HumanEval/151
6045 =====
=====
6046 [TASK] HumanEval/151 | entry_point=double_the_difference
6047 [PROMPT PREVIEW]
6048
6049 def double_the_difference(lst):
6050     '''
6051         Given a list of numbers, return the sum of squares of the
6052         numbers
6053         in the list that are odd. Ignore numbers that are negative or
6054         not integers.
6055
6056         double_the_difference([1, 3, 2, 0]) == 1 + 9 + 0 + 0 = 10
6057         double_the_difference([-1, -2, 0]) == 0
6058         double_the_difference([9, -2]) == 81
6059         double_the_difference([0]) == 0
6060     -----
6061     -----
6062     [INITIAL GENERATED CODE]
6063     def double_the_difference(lst):
6064         return sum(x**2 for x in lst if isinstance(x, int) and x >= 0
6065             and x % 2 != 0)
6066
6067     -----
6068     -----
6069     [HumanEval/151] Attempt 0/3
6070     [HumanEval/151] 0 PASS (attempt=0)
6071     =====
6072     =====
6073     [153/164] HumanEval/152
6074     =====
6075     =====
6076     [TASK] HumanEval/152 | entry_point=compare
6077     [PROMPT PREVIEW]
6078
6079     def compare(game, guess):
6080         """I think we all remember that feeling when the result of some
6081         long-awaited
6082         event is finally known. The feelings and thoughts you have at
6083         that moment are
6084         definitely worth noting down and comparing.
6085         Your task is to determine if a person correctly guessed the
6086         results of a number of matches.
6087         You are given two arrays of scores and guesses of equal length
6088         ...
6089     -----
```

```
6079 -----
6080 [INITIAL GENERATED CODE]
6081 def compare(game, guess):
6082     return [abs(g - s) for g, s in zip(guess, game)]
6083 -----
6084 -----
6084 [HumanEval/152] Attempt 0/3
6085 [HumanEval/152] Ø PASS (attempt=0)
6086 =====
6087 -----
6087 [154/164] HumanEval/153
6088 =====
6089 -----
6089 [TASK] HumanEval/153 | entry_point=Strongest_Extension
6090 [PROMPT PREVIEW]
6091
6092 def Strongest_Extension(class_name, extensions):
6093     """You will be given the name of a class (a string) and a list
6093     of extensions.
6094     The extensions are to be used to load additional classes to the
6094     class. The
6095     strength of the extension is as follows: Let CAP be the number
6095     of the uppercase
6096     letters in the extension's name, and let SM be the number of
6096     lowercase letters
6097     in the extension'...
6098 -----
6099 -----
6099 [INITIAL GENERATED CODE]
6100 def Strongest_Extension(class_name, extensions):
6101     max_strength = float('-inf')
6102     strongest_extension = ''
6103
6104     for extension in extensions:
6105         cap = sum(1 for char in extension if char.isupper())
6106         sm = sum(1 for char in extension if char.islower())
6107         strength = cap - sm
6108
6109         if strength > max_strength:
6110             max_strength = strength
6111             strongest_extension = extension
6112
6113     return f'{class_name}.{strongest_extension}'
6114 -----
6115 -----
6115 [HumanEval/153] Attempt 0/3
6116 [HumanEval/153] Ø PASS (attempt=0)
6117 =====
6118 -----
6118 [155/164] HumanEval/154
6119 =====
6120 -----
6120 [TASK] HumanEval/154 | entry_point=cycpattern_check
```

```
6121 [PROMPT PREVIEW]
6122
6123 def cycpattern_check(a , b):
6124     """You are given 2 words. You need to return True if the second
       word or any of its rotations is a substring in the first word
6125     cycpattern_check("abcd","abd") => False
6126     cycpattern_check("hello","ell") => True
6127     cycpattern_check("whassup","psus") => False
6128     cycpattern_check("abab","baa") => True
6129     cycpattern_check("efef","eef") => False
6130     cycpattern_c...
6131 -----
6132 -----
6133 [INITIAL GENERATED CODE]
6134 def cycpattern_check(a, b):
6135     return any(b in a[i:] + a[:i] for i in range(len(b)))
6136 -----
6137 [HumanEval/154] Attempt 0/3
6138 [HumanEval/154] ⚡ FAIL (attempt=0)
6139 [TEST ERROR PREVIEW]
6140 Traceback (most recent call last):
6141   File "C:\Users\JOIN\Downloads\Multi-
       Reviewer_Governance_for_Code_Generation\src\evaluator\
       humaneval_evaluator.py", line 39, in _worker
6142     ns["check"](candidate)
6143   File "<string>", line 11, in check
6144 AssertionException: test #3
6145 -----
6146 [HumanEval/154] [REPAIRED CODE] (after attempt=0)
6147 def cycpattern_check(a, b):
6148     return any(b in (a[i:] + a[:i]) for i in range(len(b)))
6149 -----
6150 [HumanEval/154] Attempt 1/3
6151 [HumanEval/154] ⚡ FAIL (attempt=1)
6152 [TEST ERROR PREVIEW]
6153 Traceback (most recent call last):
6154   File "C:\Users\JOIN\Downloads\Multi-
       Reviewer_Governance_for_Code_Generation\src\evaluator\
       humaneval_evaluator.py", line 39, in _worker
6155     ns["check"](candidate)
6156   File "<string>", line 11, in check
6157 AssertionException: test #3
6158 -----
6159 [HumanEval/154] [REPAIRED CODE] (after attempt=1)
6160 def cycpattern_check(a, b):
6161     return any(b in (a[i:] + a[:i]) for i in range(len(a)))
6162 [HumanEval/154] Attempt 2/3
```

```
6163 [HumanEval/154] ⚡ FAIL (attempt=2)
6164 [TEST ERROR PREVIEW]
6165 Traceback (most recent call last):
6166   File "C:\Users\JOIN\Downloads\Multi-
    Reviewer_Governance_for_Code_Generation\src\evaluator\
      humaneval_evaluator.py", line 39, in _worker
6167     ns["check"](candidate)
6168   File "<string>", line 11, in check
6169 AssertionError: test #3
6170 -----
-----
6171 [HumanEval/154] [REPAIRED CODE] (after attempt=2)
6172 def cycpattern_check(a, b):
6173     return any(b == (a[i:] + a[:i]) for i in range(len(a)))
6174 -----
-----
6175 [HumanEval/154] Attempt 3/3
6176 [HumanEval/154] ⚡ FAIL (attempt=3)
6177 [TEST ERROR PREVIEW]
6178 Traceback (most recent call last):
6179   File "C:\Users\JOIN\Downloads\Multi-
    Reviewer_Governance_for_Code_Generation\src\evaluator\
      humaneval_evaluator.py", line 39, in _worker
6180     ns["check"](candidate)
6181   File "<string>", line 9, in check
6182 AssertionError: test #1
6183 -----
-----
6184 [HumanEval/154] ⚡ FINAL FAIL after 3 retries
6185 =====
=====
6186 [156/164] HumanEval/155
6187 =====
=====
6188 [TASK] HumanEval/155 | entry_point=even_odd_count
6189 [PROMPT PREVIEW]
6190
6191 def even_odd_count(num):
6192     """Given an integer. return a tuple that has the number of even
       and odd digits respectively.
6193
6194     Example:
6195         even_odd_count(-12) ==> (1, 1)
6196         even_odd_count(123) ==> (1, 2)
6197     """
6198 -----
-----
6199 [INITIAL GENERATED CODE]
6200 def even_odd_count(num):
6201     num = abs(num)
6202     even_count = sum(1 for digit in str(num) if int(digit) % 2 == 0)
6203     odd_count = sum(1 for digit in str(num) if int(digit) % 2 != 0)
6204     return (even_count, odd_count)
```

```
6205 -----
-----
6206 [HumanEval/155] Attempt 0/3
6207 [HumanEval/155] Ø PASS (attempt=0)
6208 =====
=====
6209 [157/164] HumanEval/156
6210 =====
=====
6211 [TASK] HumanEval/156 | entry_point=int_to_mini_roman
6212 [PROMPT PREVIEW]
6213
6214 def int_to_mini_roman(number):
6215     """
6216         Given a positive integer, obtain its roman numeral equivalent as
6217         a string,
6218         and return it in lowercase.
6219         Restrictions: 1 <= num <= 1000
6220
6221         Examples:
6222             >>> int_to_mini_roman(19) == 'xix'
6223             >>> int_to_mini_roman(152) == 'cli'
6224             >>> int_to_mini_roman(426) == 'cdxxvi'
6225     """
6226 -----
-----
6226 [INITIAL GENERATED CODE]
6227 def int_to_mini_roman(number):
6228     val = [
6229         1000, 900, 500, 400,
6230         100, 90, 50, 40,
6231         10, 9, 5, 4,
6232         1
6233     ]
6234     syms = [
6235         "m", "cm", "d", "cd",
6236         "c", "xc", "l", "xl",
6237         "x", "ix", "v", "iv",
6238         "i"
6239     ]
6240     roman_numeral = ''
6241     i = 0
6242     while number > 0:
6243         for _ in range(number // val[i]):
6244             roman_numeral += syms[i]
6245             number -= val[i]
6246         i += 1
6247     return roman_numeral
6248 -----
-----
6249 [HumanEval/156] Attempt 0/3
6250 [HumanEval/156] Ø PASS (attempt=0)
6251 =====
```

```
6251 =====
6252 [158/164] HumanEval/157
6253 =====
6254 [TASK] HumanEval/157 | entry_point=right_angle_triangle
6255 [PROMPT PREVIEW]
6256
6257 def right_angle_triangle(a, b, c):
6258     """
6259         Given the lengths of the three sides of a triangle. Return True
6260         if the three
6261             sides form a right-angled triangle, False otherwise.
6262             A right-angled triangle is a triangle in which one angle is
6263             right angle or
6264                 90 degree.
6265             Example:
6266                 right_angle_triangle(3, 4, 5) == True
6267                 right_angle_triangle(1, 2, 3) == False
6268                 """
6269 -----
6270 -----
6271 [INITIAL GENERATED CODE]
6272 def right_angle_triangle(a, b, c):
6273     sides = sorted([a, b, c])
6274     return sides[0]**2 + sides[1]**2 == sides[2]**2
6275 -----
6276 -----
6277 [HumanEval/157] Attempt 0/3
6278 [HumanEval/157] ⚡ PASS (attempt=0)
6279 =====
6280 =====
6281 [159/164] HumanEval/158
6282 =====
6283 -----
6284 [TASK] HumanEval/158 | entry_point=find_max
6285 [PROMPT PREVIEW]
6286
6287 def find_max(words):
6288     """Write a function that accepts a list of strings.
6289         The list contains different words. Return the word with maximum
6290             number
6291             of unique characters. If multiple strings have maximum number of
6292             unique
6293             characters, return the one which comes first in lexicographical
6294             order.
6295
6296         find_max(["name", "of", "string"]) == "string"
6297         find_max(["name", "enam", "game"]) == "e...
6298 -----
6299 -----
6300 [INITIAL GENERATED CODE]
6301 def find_max(words):
6302     max_word = ""
```

```
6293     max_unique_chars = 0
6294
6295     for word in words:
6296         unique_chars = len(set(word))
6297         if unique_chars > max_unique_chars or (unique_chars ==
max_unique_chars and word < max_word):
6298             max_word = word
6299             max_unique_chars = unique_chars
6300
6301     return max_word
6302 -----
-----
6303 [HumanEval/158] Attempt 0/3
6304 [HumanEval/158] [] PASS (attempt=0)
6305 =====
=====
6306 [160/164] HumanEval/159
6307 =====
=====
6308 [TASK] HumanEval/159 | entry_point=eat
6309 [PROMPT PREVIEW]
6310
6311 def eat(number, need, remaining):
6312     """
6313     You're a hungry rabbit, and you already have eaten a certain
number of carrots,
6314     but now you need to eat more carrots to complete the day's meals
.
6315     you should return an array of [ total number of eaten carrots
after your meals,
6316                                         the number of carrots left after
your meals ]
6317     if there are not enough remaining...
6318 -----
-----
6319 [INITIAL GENERATED CODE]
6320 def eat(number, need, remaining):
6321     if need <= remaining:
6322         return [number + need, remaining - need]
6323     else:
6324         return [number + remaining, 0]
6325 -----
-----
6326 [HumanEval/159] Attempt 0/3
6327 [HumanEval/159] [] PASS (attempt=0)
6328 =====
=====
6329 [161/164] HumanEval/160
6330 =====
=====
6331 [TASK] HumanEval/160 | entry_point=do_algebra
6332 [PROMPT PREVIEW]
6333
```

```
6334 def do_algebra(operator, operand):
6335     """
6336     Given two lists operator, and operand. The first list has basic
6337     algebra operations, and
6338     the second list is a list of integers. Use the two given lists
6339     to build the algebraic
6340     expression and return the evaluation of this expression.
6341
6342     The basic algebra operations:
6343         Addition ( + )
6344         Subtraction ( - )
6345         Multiplication ( * )
6346         Floo...
6347 -----
6348 -----
6349 [INITIAL GENERATED CODE]
6350 def do_algebra(operator, operand):
6351     expression = str(operand[0])
6352     for i in range(len(operator)):
6353         expression += f" {operator[i]} {operand[i+1]}"
6354     return eval(expression)
6355 -----
6356 -----
6357 [HumanEval/160] Attempt 0/3
6358 [HumanEval/160] ∅ PASS (attempt=0)
6359 =====
6360 =====
6361 [162/164] HumanEval/161
6362 =====
6363 =====
6364 [TASK] HumanEval/161 | entry_point=solve
6365 [PROMPT PREVIEW]
6366
6367 def solve(s):
6368     """You are given a string s.
6369     if s[i] is a letter, reverse its case from lower to upper or
6370     vise versa,
6371     otherwise keep it as it is.
6372     If the string contains no letters, reverse the string.
6373     The function should return the resulted string.
6374     Examples
6375     solve("1234") = "4321"
6376     solve("ab") = "AB"
6377     solve("#a@C") = "#A@c"
6378     """
6379
6380 -----
6381 -----
6382 [INITIAL GENERATED CODE]
6383 def solve(s):
6384     if any(c.isalpha() for c in s):
6385         return ''.join(c.swapcase() if c.isalpha() else c for c in s
6386     )
6387     else:
```

```
6378         return s[::-1]
6379 -----
6380 -----
6380 [HumanEval/161] Attempt 0/3
6381 [HumanEval/161] Ø PASS (attempt=0)
6382 =====
6383 -----
6383 [163/164] HumanEval/162
6384 =====
6384 -----
6385 [TASK] HumanEval/162 | entry_point=string_to_md5
6386 [PROMPT PREVIEW]
6387
6388 def string_to_md5(text):
6389     """
6390         Given a string 'text', return its md5 hash equivalent string.
6391         If 'text' is an empty string, return None.
6392
6393         >>> string_to_md5('Hello world') == '
6393         3e25960a79dbc69b674cd4ec67a72c62'
6394     """
6395 -----
6395 -----
6396 [INITIAL GENERATED CODE]
6397 import hashlib
6398
6399 def string_to_md5(text):
6400     if not text:
6401         return None
6402     return hashlib.md5(text.encode()).hexdigest()
6403 -----
6403 -----
6404 [HumanEval/162] Attempt 0/3
6405 [HumanEval/162] Ø PASS (attempt=0)
6406 =====
6406 -----
6407 [164/164] HumanEval/163
6408 =====
6408 -----
6409 [TASK] HumanEval/163 | entry_point=generate_integers
6410 [PROMPT PREVIEW]
6411
6412 def generate_integers(a, b):
6413     """
6414         Given two positive integers a and b, return the even digits
6415         between a
6416         and b, in ascending order.
6417
6418         For example:
6419         generate_integers(2, 8) => [2, 4, 6, 8]
6420         generate_integers(8, 2) => [2, 4, 6, 8]
6421         generate_integers(10, 14) => []
6421     """
```

```
6422 -----
6423 -----
6423 [INITIAL GENERATED CODE]
6424 def generate_integers(a, b):
6425     start = min(a, b)
6426     end = max(a, b)
6427     return [i for i in range(start, end + 1) if i % 2 == 0]
6428 -----
6429 -----
6429 [HumanEval/163] Attempt 0/3
6430 [HumanEval/163] ⚡ FAIL (attempt=0)
6431 [TEST ERROR PREVIEW]
6432 Traceback (most recent call last):
6433   File "C:\Users\JOIN\Downloads\Multi-
       Reviewer_Governance_for_Code_Generation\src\evaluator\
       humaneval_evaluator.py", line 39, in _worker
6434     ns["check"](candidate)
6435   File "<string>", line 4, in check
6436 AssertionError: Test 1
6437 -----
6438 -----
6438 [HumanEval/163] [REPAIRED CODE] (after attempt=0)
6439 def generate_integers(a, b):
6440     start = min(a, b)
6441     end = max(a, b)
6442     return [i for i in range(start, end + 1) if i % 2 == 0]
6443 -----
6444 -----
6444 [HumanEval/163] Attempt 1/3
6445 [HumanEval/163] ⚡ FAIL (attempt=1)
6446 [TEST ERROR PREVIEW]
6447 Traceback (most recent call last):
6448   File "C:\Users\JOIN\Downloads\Multi-
       Reviewer_Governance_for_Code_Generation\src\evaluator\
       humaneval_evaluator.py", line 39, in _worker
6449     ns["check"](candidate)
6450   File "<string>", line 4, in check
6451 AssertionError: Test 1
6452 -----
6453 -----
6453 [HumanEval/163] [REPAIRED CODE] (after attempt=1)
6454 def generate_integers(a, b):
6455     start = min(a, b)
6456     end = max(a, b)
6457     return [i for i in range(start, end + 1) if i % 2 == 0]
6458 -----
6459 -----
6459 [HumanEval/163] Attempt 2/3
6460 [HumanEval/163] ⚡ FAIL (attempt=2)
6461 [TEST ERROR PREVIEW]
6462 Traceback (most recent call last):
6463   File "C:\Users\JOIN\Downloads\Multi-
       Reviewer_Governance_for_Code_Generation\src\evaluator\
```

```
6463 humaneval_evaluator.py", line 39, in _worker
6464     ns["check"](candidate)
6465 File "<string>", line 4, in check
6466 AssertionError: Test 1
6467 -----
6468 [HumanEval/163] [REPAIRED CODE] (after attempt=2)
6469 def generate_integers(a, b):
6470     start = min(a, b)
6471     end = max(a, b)
6472     return [i for i in range(start, end + 1) if i % 2 == 0]
6473 -----
6474 [HumanEval/163] Attempt 3/3
6475 [HumanEval/163] ⚡ FAIL (attempt=3)
6476 [TEST ERROR PREVIEW]
6477 Traceback (most recent call last):
6478   File "C:\Users\JOIN\Downloads\Multi-
       Reviewer_Governance_for_Code_Generation\src\evaluator\
       humaneval_evaluator.py", line 39, in _worker
6479     ns["check"](candidate)
6480   File "<string>", line 4, in check
6481 AssertionError: Test 1
6482 -----
6483 [HumanEval/163] ⚡ FINAL FAIL after 3 retries
6484 =====
6485 Baseline: 0.8475609756097561
6486
6487 Process finished with exit code 0
6488
```