

```
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 0/6
31 [HumanEval/0] 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 and current_group:
56             result.append(''.join(current_group))
57             current_group = []
58
59     return result
60 -----
61 -----
61 [HumanEval/1] Attempt 0/6
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
71     into
72     and integer part (largest integer smaller than given number) and
72     decimals
73     (leftover part always smaller than 1).
74
75     Return the decimal part of the number.
76     >>> truncate_number(3.5)
77     0.5
78     """
79 -----
79 -----
80 [INITIAL GENERATED CODE]
81 def truncate_number(number: float) -> float:
82     return number - int(number)
83 -----
83 -----
84 [HumanEval/2] Attempt 0/6
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 0/6
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 0/6
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     result = [numbers[0]]
172     for number in numbers[1:]:
```

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

```
216
217
218 def filter_by_substring(strings: List[str], substring: str) -> List[
219     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[
231     str]:
232     return [s for s in strings if substring in s]
233
234 -----
235 [HumanEval/7] Attempt 0/6
236 [HumanEval/7] 0 PASS (attempt=0)
237 =====
238 =====
239 [9/164] HumanEval/8
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.
267     >>> sum_product([])
```

```
257     (0, 1)
258     >>> sum_product([1, 2, 3, 4])
259     (10, 24)
260     """
261     if not numbers:
262         return (0, 1)
263
264     total_sum = sum(numbers)
265     total_product = 1
266     for number in numbers:
267         total_product *= number
268
269     return (total_sum, total_product)
270 -----
271 -----
271 [HumanEval/8] Attempt 0/6
272 [HumanEval/8] ⚡ PASS (attempt=0)
273 =====
273 =====
274 [10/164] HumanEval/9
275 =====
275 =====
276 [TASK] HumanEval/9 | entry_point=rolling_max
277 [PROMPT PREVIEW]
278 from typing import List, Tuple
279
280
281 def rolling_max(numbers: List[int]) -> List[int]:
282     """ From a given list of integers, generate a list of rolling
283     maximum element found until given moment
284     in the sequence.
285     >>> rolling_max([1, 2, 3, 2, 3, 4, 2])
286     [1, 2, 3, 3, 3, 4, 4]
286     """
287 -----
287 -----
288 [INITIAL GENERATED CODE]
289 from typing import List
290
291
292 def rolling_max(numbers: List[int]) -> List[int]:
293     if not numbers:
294         return []
295
296     result = []
297     current_max = float('-inf')
298     for number in numbers:
299         current_max = max(current_max, number)
300         result.append(current_max)
301
302     return result
303 -----
```

```
304 [HumanEval/9] Attempt 0/6
305 [HumanEval/9] ✘ PASS (attempt=0)
306 =====
307 [11/164] HumanEval/10
308 =====
309 [TASK] HumanEval/10 | entry_point=make_palindrome
310 [PROMPT PREVIEW]
311
312
313 def is_palindrome(string: str) -> bool:
314     """ Test if given string is a palindrome """
315     return string == string[::-1]
316
317
318 def make_palindrome(string: str) -> str:
319     """ Find the shortest palindrome that begins with a supplied
320         string.
321         Algorithm idea is simple:
322             - Find the longest postfix of supplied string that is a
323                 palindrome.
324             - Append to the end of the string reverse of a string...
325 -----
326 -----
327 [INITIAL GENERATED CODE]
328 def make_palindrome(string: str) -> str:
329     """ Find the shortest palindrome that begins with a supplied
330         string.
331         Algorithm idea is simple:
332             - Find the longest postfix of supplied string that is a
333                 palindrome.
334             - Append to the end of the string reverse of a string prefix that
335                 comes before the palindromic suffix.
336             >>> make_palindrome('')
337             ''
338             >>> make_palindrome('cat')
339             'catac'
340             >>> make_palindrome('cata')
341             'catac'
342             """
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 0/6
353 [HumanEval/10] ✘ FAIL (attempt=0)
354 [TEST ERROR PREVIEW]
355 Traceback (most recent call last):
```

```
348 File "C:\Users\JOIN\Downloads\Multi-
    Reviewer_Governance_for_Code_Generation\src\evaluator\
        humaneval_evaluator.py", line 39, in _worker
349     ns["check"](candidate)
350 File "<string>", line 12, in check
351 AssertionError
352 -----
-----
353 [HumanEval/10] [REPAIRED CODE] (after attempt=0)
354 def make_palindrome(string: str) -> str:
355     """ Find the shortest palindrome that begins with a supplied
        string.
356     Algorithm idea is simple:
357         - Find the longest postfix of supplied string that is a
            palindrome.
358         - Append to the end of the string reverse of a string prefix that
            comes before the palindromic suffix.
359     >>> make_palindrome('')
360     ''
361     >>> make_palindrome('cat')
362     'catac'
363     >>> make_palindrome('cata')
364     'catac'
365     """
366     if is_palindrome(string):
367         return string
368     for i in range(len(string), -1, -1):
369         if is_palindrome(string[i:]):
370             prefix = string[:i]
371             return string + prefix[::-1]
372
373 def is_palindrome(s: str) -> bool:
374     """ Check if a given string is a palindrome. """
375     return s == s[::-1]
376 -----
-----
377 [HumanEval/10] Attempt 1/6
378 [HumanEval/10] ⚡ FAIL (attempt=1)
379 [TEST ERROR PREVIEW]
380 Traceback (most recent call last):
381     File "C:\Users\JOIN\Downloads\Multi-
        Reviewer_Governance_for_Code_Generation\src\evaluator\
            humaneval_evaluator.py", line 39, in _worker
382     ns["check"](candidate)
383 File "<string>", line 12, in check
384 AssertionError
385 -----
-----
386 [HumanEval/10] [REPAIRED CODE] (after attempt=1)
387 def make_palindrome(string: str) -> str:
388     """ Find the shortest palindrome that begins with a supplied
        string.
389     Algorithm idea is simple:
```

```
390     - Find the longest postfix of supplied string that is a
391         palindrome.
392     - Append to the end of the string reverse of a string prefix that
393         comes before the palindromic suffix.
394     >>> make_palindrome('')
395     ''
396     >>> make_palindrome('cat')
397     'catac'
398     >>> make_palindrome('cata')
399     'catac'
400     """
401     if is_palindrome(string):
402         return string
403     for i in range(len(string), -1, -1):
404         if is_palindrome(string[:i]):
405             prefix = string[i:]
406             return string + prefix[::-1]
407
408 def is_palindrome(s: str) -> bool:
409     """ Check if a given string is a palindrome. """
410     return s == s[::-1]
411 -----
412 -----
413 [HumanEval/10] Attempt 2/6
414 [HumanEval/10] ✘ FAIL (attempt=2)
415 [TEST ERROR PREVIEW]
416 Traceback (most recent call last):
417   File "C:\Users\JOIN\Downloads\Multi-
418       Reviewer_Governance_for_Code_Generation\src\evaluator\
419           humaneval_evaluator.py", line 39, in _worker
420             ns["check"](candidate)
421   File "<string>", line 12, in check
422 AssertionException
423 -----
424 -----
425 [HumanEval/10] [REPAIRED CODE] (after attempt=2)
426 def make_palindrome(string: str) -> str:
427     """ Find the shortest palindrome that begins with a supplied
428         string.
429     Algorithm idea is simple:
430         - Find the longest postfix of supplied string that is a
431             palindrome.
432         - Append to the end of the string reverse of a string prefix that
433             comes before the palindromic suffix.
434     >>> make_palindrome('')
435     ''
436     >>> make_palindrome('cat')
437     'catac'
438     >>> make_palindrome('cata')
439     'catac'
440     """
441     if is_palindrome(string):
442         return string
```

```
434     for i in range(len(string), -1, -1):
435         if is_palindrome(string[:i]):
436             prefix = string[i:]
437             return string + prefix[::-1]
438
439 def is_palindrome(s: str) -> bool:
440     """ Check if a given string is a palindrome. """
441     return s == s[::-1]
442 -----
-----
443 [HumanEval/10] Attempt 3/6
444 [HumanEval/10] ✘ FAIL (attempt=3)
445 [TEST ERROR PREVIEW]
446 Traceback (most recent call last):
447   File "C:\Users\JOIN\Downloads\Multi-
    Reviewer_Governance_for_Code_Generation\src\evaluator\
      humaneval_evaluator.py", line 39, in _worker
448     ns["check"](candidate)
449   File "<string>", line 12, in check
450 Assertion
451 -----
-----
452 [HumanEval/10] [REPAIRED CODE] (after attempt=3)
453 def make_palindrome(string: str) -> str:
454     """ Find the shortest palindrome that begins with a supplied
        string.
455     Algorithm idea is simple:
456     - Find the longest postfix of supplied string that is a
        palindrome.
457     - Append to the end of the string reverse of a string prefix that
        comes before the palindromic suffix.
458     >>> make_palindrome('')
459     ''
460     >>> make_palindrome('cat')
461     'catac'
462     >>> make_palindrome('cata')
463     'catac'
464     """
465     if is_palindrome(string):
466         return string
467     for i in range(len(string), -1, -1):
468         if is_palindrome(string[:i]):
469             prefix = string[i:]
470             return string + prefix[::-1]
471
472 def is_palindrome(s: str) -> bool:
473     """ Check if a given string is a palindrome. """
474     return s == s[::-1]
475 -----
-----
476 [HumanEval/10] Attempt 4/6
477 [HumanEval/10] ✘ FAIL (attempt=4)
478 [TEST ERROR PREVIEW]
```

```
479 Traceback (most recent call last):
480   File "C:\Users\JOIN\Downloads\Multi-
  Reviewer_Governance_for_Code_Generation\src\evaluator\
    humaneval_evaluator.py", line 39, in _worker
481     ns["check"](candidate)
482   File "<string>", line 12, in check
483 AssertionError
484 -----
485 -----
485 [HumanEval/10] [REPAIRED CODE] (after attempt=4)
486 def make_palindrome(string: str) -> str:
487     """ Find the shortest palindrome that begins with a supplied
        string.
488     Algorithm idea is simple:
489         - Find the longest postfix of supplied string that is a
            palindrome.
490         - Append to the end of the string reverse of a string prefix that
            comes before the palindromic suffix.
491     >>> make_palindrome('')
492     ''
493     >>> make_palindrome('cat')
494     'catac'
495     >>> make_palindrome('cata')
496     'catac'
497     """
498     if is_palindrome(string):
499         return string
500     for i in range(len(string), -1, -1):
501         if is_palindrome(string[:i]):
502             prefix = string[i:]
503             return string + prefix[::-1]
504
505 def is_palindrome(s: str) -> bool:
506     """ Check if a given string is a palindrome. """
507     return s == s[::-1]
508 -----
509 -----
509 [HumanEval/10] Attempt 5/6
510 [HumanEval/10] ⚡ FAIL (attempt=5)
511 [TEST ERROR PREVIEW]
512 Traceback (most recent call last):
513   File "C:\Users\JOIN\Downloads\Multi-
  Reviewer_Governance_for_Code_Generation\src\evaluator\
    humaneval_evaluator.py", line 39, in _worker
514     ns["check"](candidate)
515   File "<string>", line 12, in check
516 AssertionError
517 -----
518 -----
518 [HumanEval/10] [REPAIRED CODE] (after attempt=5)
519 def make_palindrome(string: str) -> str:
520     """ Find the shortest palindrome that begins with a supplied
        string.
```

```
521     Algorithm idea is simple:
522         - Find the longest postfix of supplied string that is a
523             palindrome.
524             - Append to the end of the string reverse of a string prefix that
525                 comes before the palindromic suffix.
526
527             >>> make_palindrome('')
528             ''
529
530             >>> make_palindrome('cat')
531             'catac'
532
533             >>> make_palindrome('cata')
534             'catac'
535             """
536
537             if is_palindrome(string):
538                 return string
539                 for i in range(len(string), -1, -1):
540                     if is_palindrome(string[:i]):
541                         prefix = string[i:]
542                         return string + prefix[::-1]
543
544 def is_palindrome(s: str) -> bool:
545     """ Check if a given string is a palindrome. """
546     return s == s[::-1]
547 -----
548 -----
549 [HumanEval/10] Attempt 6/6
550 [HumanEval/10] ✘ FAIL (attempt=6)
551 [TEST ERROR PREVIEW]
552 Traceback (most recent call last):
553   File "C:\Users\JOIN\Downloads\Multi-
554       Reviewer_Governance_for_Code_Generation\src\evaluator\
555           humaneval_evaluator.py", line 39, in _worker
556               ns["check"](candidate)
557   File "<string>", line 12, in check
558   AssertionError
559 -----
560 -----
561 [HumanEval/10] ✘ FINAL FAIL after 6 retries
562 =====
563 =====
564 [12/164] HumanEval/11
565 =====
566 =====
567 [TASK] HumanEval/11 | entry_point=string_xor
568 [PROMPT PREVIEW]
569 from typing import List
570
571
572 def string_xor(a: str, b: str) -> str:
573     """ Input are two strings a and b consisting only of 1s and 0s.
574     Perform binary XOR on these inputs and return result also as a
575     string.
576     >>> string_xor('010', '110')
577     '100'
```

```
565 """
566 -----
567 [INITIAL GENERATED CODE]
568 def string_xor(a: str, b: str) -> str:
569     return ''.join(str(int(x) ^ int(y)) for x, y in zip(a, b))
570 -----
571 [HumanEval/11] Attempt 0/6
572 [HumanEval/11] ✘ PASS (attempt=0)
573 =====
574 [13/164] HumanEval/12
575 =====
576 [TASK] HumanEval/12 | entry_point=longest
577 [PROMPT PREVIEW]
578 from typing import List, Optional
579
580
581 def longest(strings: List[str]) -> Optional[str]:
582     """ Out of list of strings, return the longest one. Return the
583     first one in case of multiple
584     strings of the same length. Return None in case the input list is
585     empty.
586     >>> longest([])
587
588     >>> longest(['a', 'b', 'c'])
589     'a'
590     >>> longest(['a', 'bb', 'ccc'])
591     'ccc'
592     """
593
594 [INITIAL GENERATED CODE]
595 def longest(strings: List[str]) -> Optional[str]:
596     if not strings:
597         return None
598     return max(strings, key=len)
599
600 -----
601 [HumanEval/12] Attempt 0/6
602 [HumanEval/12] ✘ PASS (attempt=0)
603 =====
604 [14/164] HumanEval/13
605 =====
606
607 def greatest_common_divisor(a: int, b: int) -> int:
```

```
608     """ Return a greatest common divisor of two integers a and b
609     >>> greatest_common_divisor(3, 5)
610     1
611     >>> greatest_common_divisor(25, 15)
612     5
613     """
614 -----
615 [INITIAL GENERATED CODE]
616 def greatest_common_divisor(a: int, b: int) -> int:
617     while b != 0:
618         a, b = b, a % b
619     return a
620 -----
621 [HumanEval/13] Attempt 0/6
622 [HumanEval/13] 0 PASS (attempt=0)
623 =====
624 [15/164] HumanEval/14
625 =====
626 [TASK] HumanEval/14 | entry_point=all_prefixes
627 [PROMPT PREVIEW]
628 from typing import List
629
630
631 def all_prefixes(string: str) -> List[str]:
632     """ Return list of all prefixes from shortest to longest of the
633     input string
634     >>> all_prefixes('abc')
635     ['a', 'ab', 'abc']
636     """
637 -----
638 [INITIAL GENERATED CODE]
639 return [string[:i] for i in range(1, len(string) + 1)]
640 -----
641 [HumanEval/14] Attempt 0/6
642 [HumanEval/14] 0 FAIL (attempt=0)
643 [TEST ERROR PREVIEW]
644 Traceback (most recent call last):
645   File "C:\Users\JOIN\Downloads\Multi-
646   Reviewer_Governance_for_Code_Generation\src\evaluator\
647   humaneval_evaluator.py", line 27, in _worker
648     exec(prompt + "\n" + completion + "\n", ns, ns)
649   File "<string>", line 10
650 SyntaxError: 'return' outside function
651 -----
652 [HumanEval/14] [REPAIRED CODE] (after attempt=0)
653 def generate_prefixes(string):
```

```
651     return [string[:i] for i in range(1, len(string) + 1)]
652 -----
653 [HumanEval/14] Attempt 1/6
654 [HumanEval/14] ✘ FAIL (attempt=1)
655 [TEST ERROR PREVIEW]
656 Traceback (most recent call last):
657   File "C:\Users\JOIN\Downloads\Multi-
    Reviewer_Governance_for_Code_Generation\src\evaluator\
      humaneval_evaluator.py", line 39, in _worker
658     ns["check"](candidate)
659   File "<string>", line 10, in check
660 AssertionError
661 -----
662 [HumanEval/14] [REPAIRED CODE] (after attempt=1)
663 def generate_prefixes(string):
664     return [string[:i] for i in range(1, len(string) + 1)]
665 -----
666 [HumanEval/14] Attempt 2/6
667 [HumanEval/14] ✘ FAIL (attempt=2)
668 [TEST ERROR PREVIEW]
669 Traceback (most recent call last):
670   File "C:\Users\JOIN\Downloads\Multi-
    Reviewer_Governance_for_Code_Generation\src\evaluator\
      humaneval_evaluator.py", line 39, in _worker
671     ns["check"](candidate)
672   File "<string>", line 10, in check
673 AssertionError
674 -----
675 [HumanEval/14] [REPAIRED CODE] (after attempt=2)
676 def generate_prefixes(string):
677     return [string[:i] for i in range(1, len(string) + 1)]
678 -----
679 [HumanEval/14] Attempt 3/6
680 [HumanEval/14] ✘ FAIL (attempt=3)
681 [TEST ERROR PREVIEW]
682 Traceback (most recent call last):
683   File "C:\Users\JOIN\Downloads\Multi-
    Reviewer_Governance_for_Code_Generation\src\evaluator\
      humaneval_evaluator.py", line 39, in _worker
684     ns["check"](candidate)
685   File "<string>", line 10, in check
686 AssertionError
687 -----
688 [HumanEval/14] [REPAIRED CODE] (after attempt=3)
689 def generate_prefixes(string):
690     return [string[:i] for i in range(1, len(string) + 1)]
691 -----
```

```
691 -----
692 [HumanEval/14] Attempt 4/6
693 [HumanEval/14] ⚡ FAIL (attempt=4)
694 [TEST ERROR PREVIEW]
695 Traceback (most recent call last):
696   File "C:\Users\JOIN\Downloads\Multi-
       Reviewer_Governance_for_Code_Generation\src\evaluator\
         humaneval_evaluator.py", line 39, in _worker
697     ns["check"](candidate)
698   File "<string>", line 10, in check
699 AssertionError
700 -----
701 -----
702 [HumanEval/14] [REPAIRED CODE] (after attempt=4)
703 def generate_prefixes(string):
704     return [string[:i] for i in range(1, len(string) + 1)]
705 -----
706 -----
707 [HumanEval/14] Attempt 5/6
708 [HumanEval/14] ⚡ FAIL (attempt=5)
709 [TEST ERROR PREVIEW]
710 Traceback (most recent call last):
711   File "C:\Users\JOIN\Downloads\Multi-
       Reviewer_Governance_for_Code_Generation\src\evaluator\
         humaneval_evaluator.py", line 39, in _worker
712     ns["check"](candidate)
713   File "<string>", line 10, in check
714 AssertionError
715 -----
716 -----
717 [HumanEval/14] [REPAIRED CODE] (after attempt=5)
718 def generate_prefixes(string):
719     return [string[:i] for i in range(1, len(string) + 1)]
720 -----
721 -----
722 [HumanEval/14] Attempt 6/6
723 [HumanEval/14] ⚡ FAIL (attempt=6)
724 [TEST ERROR PREVIEW]
725 Traceback (most recent call last):
726   File "C:\Users\JOIN\Downloads\Multi-
       Reviewer_Governance_for_Code_Generation\src\evaluator\
         humaneval_evaluator.py", line 39, in _worker
727     ns["check"](candidate)
728   File "<string>", line 10, in check
729 AssertionError
730 -----
731 -----
732 [HumanEval/14] ⚡ FINAL FAIL after 6 retries
733 =====
734 =====
735 [16/164] HumanEval/15
736 =====
737 =====
```

```
731 [TASK] HumanEval/15 | entry_point=string_sequence
732 [PROMPT PREVIEW]
733
734
735 def string_sequence(n: int) -> str:
736     """ Return a string containing space-delimited numbers starting
737         from 0 upto n inclusive.
738     >>> string_sequence(0)
739     '0'
740     >>> string_sequence(5)
741     '0 1 2 3 4 5'
742     """
743 -----
744 [INITIAL GENERATED CODE]
745 def string_sequence(n: int) -> str:
746     return ' '.join(str(i) for i in range(n + 1))
747 -----
748 [HumanEval/15] Attempt 0/6
749 [HumanEval/15] 0 PASS (attempt=0)
750 =====
751 =====
752 [17/164] HumanEval/16
753 =====
754 =====
755
756 def count_distinct_characters(string: str) -> int:
757     """ Given a string, find out how many distinct characters (
758         regardless of case) does it consist of
759     >>> count_distinct_characters('xyzXYZ')
760     3
761     >>> count_distinct_characters('Jerry')
762     4
763     """
764 -----
765 [INITIAL GENERATED CODE]
766 def count_distinct_characters(string: str) -> int:
767     return len(set(string.lower()))
768 -----
769 [HumanEval/16] Attempt 0/6
770 [HumanEval/16] 0 PASS (attempt=0)
771 =====
772 =====
773 [18/164] HumanEval/17
774 =====
775 =====
776 [TASK] HumanEval/17 | entry_point=parse_music
```

```
774 [PROMPT PREVIEW]
775 from typing import List
776
777
778 def parse_music(music_string: str) -> List[int]:
779     """ Input to this function is a string representing musical notes
    in a special ASCII format.
780     Your task is to parse this string and return list of integers
    corresponding to how many beats does each
781     note last.
782
783     Here is a legend:
784     'o' - whole note, lasts four beats
785     'o|' - half note, lasts two beats
786     '....'
787 -----
788 -----
788 [INITIAL GENERATED CODE]
789 def parse_music(music_string: str) -> List[int]:
790     """ Input to this function is a string representing musical notes
    in a special ASCII format.
791     Your task is to parse this string and return list of integers
    corresponding to how many beats does each
792     note last.
793
794     Here is a legend:
795     'o' - whole note, lasts four beats
796     'o|' - half note, lasts two beats
797     '.|' - quater note, lasts one beat
798
799     >>> parse_music('o o| .| o| o| .| .| .| o o')
800     [4, 2, 1, 2, 2, 1, 1, 1, 1, 4, 4]
801     """
802     notes = music_string.split()
803     beats = []
804     for note in notes:
805         if note == 'o':
806             beats.append(4)
807         elif note == 'o|':
808             beats.append(2)
809         elif note == '.|':
810             beats.append(1)
811     return beats
812 -----
813 -----
813 [HumanEval/17] Attempt 0/6
814 [HumanEval/17] 0 PASS (attempt=0)
815 =====
816 =====
816 [19/164] HumanEval/18
817 =====
817 =====
818 [TASK] HumanEval/18 | entry_point=how_many_times
```

```
819 [PROMPT PREVIEW]
820
821
822 def how_many_times(string: str, substring: str) -> int:
823     """ Find how many times a given substring can be found in the
824     original string. Count overlapping cases.
825     >>> how_many_times('', 'a')
826     0
827     >>> how_many_times('aaa', 'a')
828     3
829     >>> how_many_times('aaaa', 'aa')
830     3
831     """
832 -----
833 [INITIAL GENERATED CODE]
834 def how_many_times(string: str, substring: str) -> int:
835     count = 0
836     start = 0
837     while True:
838         start = string.find(substring, start)
839         if start == -1:
840             break
841         count += 1
842         start += 1
843     return count
844 -----
845 [HumanEval/18] Attempt 0/6
846 [HumanEval/18] ✘ PASS (attempt=0)
847 =====
848 =====
849 [20/164] HumanEval/19
850 =====
851 =====
852 [TASK] HumanEval/19 | entry_point=sort_numbers
853 [PROMPT PREVIEW]
854 from typing import List
855
856 def sort_numbers(numbers: str) -> str:
857     """ Input is a space-delimited string of numberals from 'zero' to
858     'nine'.
859     Valid choices are 'zero', 'one', 'two', 'three', 'four', 'five',
860     'six', 'seven', 'eight' and 'nine'.
861     Return the string with numbers sorted from smallest to largest
862     >>> sort_numbers('three one five')
863     'one three five'
864     """
865 -----
866 [INITIAL GENERATED CODE]
867 def sort_numbers(numbers: str) -> str:
```

```
864     num_map = {
865         "zero": 0, "one": 1, "two": 2, "three": 3, "four": 4,
866         "five": 5, "six": 6, "seven": 7, "eight": 8, "nine": 9
867     }
868     sorted_numbers = sorted(numbers.split(), key=lambda x: num_map[x]
869     ])
870 -----
871 -----
871 [HumanEval/19] Attempt 0/6
872 [HumanEval/19] 0 PASS (attempt=0)
873 =====
874 =====
874 [21/164] HumanEval/20
875 =====
875 =====
876 [TASK] HumanEval/20 | entry_point=find_closest_elements
877 [PROMPT PREVIEW]
878 from typing import List, Tuple
879
880
881 def find_closest_elements(numbers: List[float]) -> Tuple[float, float]:
882     """ From a supplied list of numbers (of length at least two)
883     select and return two that are the closest to each
884     other and return them in order (smaller number, larger number).
885     >>> find_closest_elements([1.0, 2.0, 3.0, 4.0, 5.0, 2.2])
886     (2.0, 2.2)
887     >>> find_closest_elements([1.0...
887 -----
888 -----
888 [INITIAL GENERATED CODE]
889 from typing import List, Tuple
890
891
892 def find_closest_elements(numbers: List[float]) -> Tuple[float, float]:
893     """ From a supplied list of numbers (of length at least two)
894     select and return two that are the closest to each
895     other and return them in order (smaller number, larger number).
896     >>> find_closest_elements([1.0, 2.0, 3.0, 4.0, 5.0, 2.2])
897     (2.0, 2.2)
898     >>> find_closest_elements([1.0, 2.0, 3.0, 4.0, 5.0, 2.0])
899     (2.0, 2.0)
900     """
901     numbers.sort()
902     closest_pair = (numbers[0], numbers[1])
903     min_diff = abs(numbers[1] - numbers[0])
904
905     for i in range(1, len(numbers) - 1):
906         diff = abs(numbers[i + 1] - numbers[i])
907         if diff < min_diff:
908             min_diff = diff
```

```
908         closest_pair = (numbers[i], numbers[i + 1])
909
910     return closest_pair
911 -----
912 [HumanEval/20] Attempt 0/6
913 [HumanEval/20] ✘ PASS (attempt=0)
914 =====
915 =====
916 [22/164] HumanEval/21
917 =====
918 =====
919 [TASK] HumanEval/21 | entry_point=rescale_to_unit
920 [PROMPT PREVIEW]
921 from typing import List
922
923 def rescale_to_unit(numbers: List[float]) -> List[float]:
924     """ Given list of numbers (of at least two elements), apply a
925     linear transform to that list,
926     such that the smallest number will become 0 and the largest will
927     become 1
928     >>> rescale_to_unit([1.0, 2.0, 3.0, 4.0, 5.0])
929     [0.0, 0.25, 0.5, 0.75, 1.0]
926     """
927
928 -----
929 [INITIAL GENERATED CODE]
930 def rescale_to_unit(numbers: List[float]) -> List[float]:
931     min_val = min(numbers)
932     max_val = max(numbers)
933     range_val = max_val - min_val
934     return [(x - min_val) / range_val for x in numbers]
935 -----
936 [HumanEval/21] Attempt 0/6
937 [HumanEval/21] ✘ PASS (attempt=0)
938 =====
939 =====
939 [23/164] HumanEval/22
940 =====
941 =====
941 [TASK] HumanEval/22 | entry_point=filter_integers
942 [PROMPT PREVIEW]
943 from typing import List, Any
944
945
946 def filter_integers(values: List[Any]) -> List[int]:
947     """ Filter given list of any python values only for integers
948     >>> filter_integers(['a', 3.14, 5])
949     [5]
950     >>> filter_integers([1, 2, 3, 'abc', {}, []])
951     [1, 2, 3]
```

```
952     """
953 -----
954 [INITIAL GENERATED CODE]
955 def filter_integers(values: List[Any]) -> List[int]:
956     return [value for value in values if isinstance(value, int)]
957 -----
958 [HumanEval/22] Attempt 0/6
959 [HumanEval/22] ✘ PASS (attempt=0)
960 =====
961 [24/164] HumanEval/23
962 =====
963 [TASK] HumanEval/23 | entry_point=strlen
964 [PROMPT PREVIEW]
965
966
967 def strlen(string: str) -> int:
968     """ Return length of given string
969     >>> strlen('')
970     0
971     >>> strlen('abc')
972     3
973     """
974 -----
975 [INITIAL GENERATED CODE]
976 return len(string)
977 -----
978 [HumanEval/23] Attempt 0/6
979 [HumanEval/23] ✘ FAIL (attempt=0)
980 [TEST ERROR PREVIEW]
981 Traceback (most recent call last):
982   File "C:\Users\JOIN\Downloads\Multi-
983     Reviewer_Governance_for_Code_Generation\src\evaluator\
984       humaneval_evaluator.py", line 27, in _worker
985         exec(prompt + "\n" + completion + "\n", ns, ns)
986   File "<string>", line 11
987 SyntaxError: 'return' outside function
988 -----
989 [HumanEval/23] [REPAIRED CODE] (after attempt=0)
990 def count_length(string):
991     return len(string)
992 -----
993 [HumanEval/23] Attempt 1/6
994 [HumanEval/23] ✘ FAIL (attempt=1)
995 [TEST ERROR PREVIEW]
996 Traceback (most recent call last):
```

```
995 File "C:\Users\JOIN\Downloads\Multi-
Reviewer_Governance_for_Code_Generation\src\evaluator\
humaneval_evaluator.py", line 39, in _worker
996     ns["check"](candidate)
997 File "<string>", line 10, in check
998 AssertionError
999 -----
-----
1000 [HumanEval/23] [REPAIRED CODE] (after attempt=1)
1001 def count_length(string):
1002     return len(string)
1003 -----
-----
1004 [HumanEval/23] Attempt 2/6
1005 [HumanEval/23] ⚡ FAIL (attempt=2)
1006 [TEST ERROR PREVIEW]
1007 Traceback (most recent call last):
1008     File "C:\Users\JOIN\Downloads\Multi-
Reviewer_Governance_for_Code_Generation\src\evaluator\
humaneval_evaluator.py", line 39, in _worker
1009     ns["check"](candidate)
1010 File "<string>", line 10, in check
1011 AssertionError
1012 -----
-----
1013 [HumanEval/23] [REPAIRED CODE] (after attempt=2)
1014 def count_length(string):
1015     return len(string)
1016 -----
-----
1017 [HumanEval/23] Attempt 3/6
1018 [HumanEval/23] ⚡ FAIL (attempt=3)
1019 [TEST ERROR PREVIEW]
1020 Traceback (most recent call last):
1021     File "C:\Users\JOIN\Downloads\Multi-
Reviewer_Governance_for_Code_Generation\src\evaluator\
humaneval_evaluator.py", line 39, in _worker
1022     ns["check"](candidate)
1023 File "<string>", line 10, in check
1024 AssertionError
1025 -----
-----
1026 [HumanEval/23] [REPAIRED CODE] (after attempt=3)
1027 def count_length(string):
1028     return len(string)
1029 -----
-----
1030 [HumanEval/23] Attempt 4/6
1031 [HumanEval/23] ⚡ FAIL (attempt=4)
1032 [TEST ERROR PREVIEW]
1033 Traceback (most recent call last):
1034     File "C:\Users\JOIN\Downloads\Multi-
Reviewer_Governance_for_Code_Generation\src\evaluator\
```

```
1034 humaneval_evaluator.py", line 39, in _worker
1035     ns["check"](candidate)
1036 File "<string>", line 10, in check
1037 AssertionError
1038 -----
1039 [HumanEval/23] [REPAIRED CODE] (after attempt=4)
1040 def count_length(string):
1041     return len(string)
1042 -----
1043 [HumanEval/23] Attempt 5/6
1044 [HumanEval/23] ⚡ FAIL (attempt=5)
1045 [TEST ERROR PREVIEW]
1046 Traceback (most recent call last):
1047     File "C:\Users\JOIN\Downloads\Multi-
        Reviewer_Governance_for_Code_Generation\src\evaluator\
        humaneval_evaluator.py", line 39, in _worker
1048     ns["check"](candidate)
1049     File "<string>", line 10, in check
1050 AssertionError
1051 -----
1052 [HumanEval/23] [REPAIRED CODE] (after attempt=5)
1053 def count_length(string):
1054     return len(string)
1055 -----
1056 [HumanEval/23] Attempt 6/6
1057 [HumanEval/23] ⚡ FAIL (attempt=6)
1058 [TEST ERROR PREVIEW]
1059 Traceback (most recent call last):
1060     File "C:\Users\JOIN\Downloads\Multi-
        Reviewer_Governance_for_Code_Generation\src\evaluator\
        humaneval_evaluator.py", line 39, in _worker
1061     ns["check"](candidate)
1062     File "<string>", line 10, in check
1063 AssertionError
1064 -----
1065 [HumanEval/23] ⚡ FINAL FAIL after 6 retries
1066 =====
1067 [25/164] HumanEval/24
1068 =====
1069 [TASK] HumanEval/24 | entry_point=largest_divisor
1070 [PROMPT PREVIEW]
1071
1072
1073 def largest_divisor(n: int) -> int:
1074     """ For a given number n, find the largest number that divides n
        evenly, smaller than n
```

```
1075     >>> largest_divisor(15)
1076     5
1077     """
1078 -----
1079 [INITIAL GENERATED CODE]
1080 def largest_divisor(n: int) -> int:
1081     for i in range(n-1, 0, -1):
1082         if n % i == 0:
1083             return i
1084     return 1
1085 -----
1086 [HumanEval/24] Attempt 0/6
1087 [HumanEval/24] 0 PASS (attempt=0)
1088 =====
1089 =====
1090 [26/164] HumanEval/25
1091 =====
1092 =====
1093 =====
1094 [TASK] HumanEval/25 | entry_point=factorize
1095 [PROMPT PREVIEW]
1096 from typing import List
1097
1098
1099 def factorize(n: int) -> List[int]:
1100     """ Return list of prime factors of given integer in the order
1101     from smallest to largest.
1102     Each of the factors should be listed number of times
1103     corresponding to how many times it appears in factorization.
1104     Input number should be equal to the product of all factors
1105     """
1106     >>> factorize(8)
1107     [2, 2, 2]
1108     >>> factorize(25)
1109     [5...
1110 -----
1111 [INITIAL GENERATED CODE]
1112 from typing import List
1113
1114
1115 def factorize(n: int) -> List[int]:
1116     """ Return list of prime factors of given integer in the order
1117     from smallest to largest.
1118     Each of the factors should be listed number of times
1119     corresponding to how many times it appears in factorization.
1120     Input number should be equal to the product of all factors
1121     """
1122     >>> factorize(8)
1123     [2, 2, 2]
1124     >>> factorize(25)
1125     [5, 5]
1126     >>> factorize(70)
1127     [2, 5, 7]
```

```
1119 """
1120     factors = []
1121     # Start with the smallest prime number
1122     divisor = 2
1123     while n >= divisor * divisor:
1124         if n % divisor == 0:
1125             factors.append(divisor)
1126             n //= divisor
1127         else:
1128             divisor += 1
1129     if n > 1:
1130         factors.append(n)
1131     return factors
1132 -----
1133 -----
1133 [HumanEval/25] Attempt 0/6
1134 [HumanEval/25] 0 PASS (attempt=0)
1135 =====
1136 -----
1136 [27/164] HumanEval/26
1137 =====
1137 -----
1138 [TASK] HumanEval/26 | entry_point=remove_duplicates
1139 [PROMPT PREVIEW]
1140 from typing import List
1141
1142
1143 def remove_duplicates(numbers: List[int]) -> List[int]:
1144     """ From a list of integers, remove all elements that occur more
1145     than once.
1146     Keep order of elements left the same as in the input.
1147     >>> remove_duplicates([1, 2, 3, 2, 4])
1148     [1, 3, 4]
1149 """
1149 -----
1150 -----
1150 [INITIAL GENERATED CODE]
1151 from typing import List
1152
1153
1154 def remove_duplicates(numbers: List[int]) -> List[int]:
1155     """ From a list of integers, remove all elements that occur more
1156     than once.
1157     Keep order of elements left the same as in the input.
1158     >>> remove_duplicates([1, 2, 3, 2, 4])
1159     [1, 3, 4]
1160     """
1160     seen = set()
1161     result = []
1162     for number in numbers:
1163         if number not in seen:
1164             seen.add(number)
1165             result.append(number)
```

```
1166     return result
1167 -----
1168 [HumanEval/26] Attempt 0/6
1169 [HumanEval/26] ⚡ FAIL (attempt=0)
1170 [TEST ERROR PREVIEW]
1171 Traceback (most recent call last):
1172   File "C:\Users\JOIN\Downloads\Multi-
    Reviewer_Governance_for_Code_Generation\src\evaluator\
      humaneval_evaluator.py", line 39, in _worker
1173     ns["check"](candidate)
1174   File "<string>", line 12, in check
1175 AssertionError
1176 -----
1177 [HumanEval/26] [REPAIRED CODE] (after attempt=0)
1178 from typing import List
1179
1180
1181 def remove_duplicates(numbers: List[int]) -> List[int]:
1182     """ From a list of integers, remove all elements that occur more
        than once.
1183     Keep order of elements left the same as in the input.
1184     >>> remove_duplicates([1, 2, 3, 2, 4])
1185     [1, 3, 4]
1186     """
1187     seen = set()
1188     result = []
1189     for number in numbers:
1190         if number not in seen:
1191             seen.add(number)
1192             result.append(number)
1193     return result
1194 -----
1195 [HumanEval/26] Attempt 1/6
1196 [HumanEval/26] ⚡ FAIL (attempt=1)
1197 [TEST ERROR PREVIEW]
1198 Traceback (most recent call last):
1199   File "C:\Users\JOIN\Downloads\Multi-
    Reviewer_Governance_for_Code_Generation\src\evaluator\
      humaneval_evaluator.py", line 39, in _worker
1200     ns["check"](candidate)
1201   File "<string>", line 12, in check
1202 AssertionError
1203 -----
1204 [HumanEval/26] [REPAIRED CODE] (after attempt=1)
1205 from typing import List
1206
1207
1208 def remove_duplicates(numbers: List[int]) -> List[int]:
1209     """ From a list of integers, remove all elements that occur more
```

```
1209 than once.
1210     Keep order of elements left the same as in the input.
1211     >>> remove_duplicates([1, 2, 3, 2, 4])
1212     [1, 3, 4]
1213     """
1214     seen = set()
1215     result = []
1216     for number in numbers:
1217         if number not in seen:
1218             seen.add(number)
1219             result.append(number)
1220     return result
1221 -----
1222 -----
1222 [HumanEval/26] Attempt 2/6
1223 [HumanEval/26] ✘ FAIL (attempt=2)
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
1229       ns["check"](candidate)
1230     File "<string>", line 12, in check
1231 Assertion
1230 -----
1231 -----
1231 [HumanEval/26] [REPAIRED CODE] (after attempt=2)
1232 from typing import List
1233
1234
1235 def remove_duplicates(numbers: List[int]) -> List[int]:
1236     """ From a list of integers, remove all elements that occur more
1237     than once.
1238     Keep order of elements left the same as in the input.
1239     >>> remove_duplicates([1, 2, 3, 2, 4])
1240     [1, 3, 4]
1241     """
1242     seen = set()
1243     result = []
1244     for number in numbers:
1245         if number not in seen:
1246             seen.add(number)
1247             result.append(number)
1248     return result
1249 -----
1249 [HumanEval/26] Attempt 3/6
1250 [HumanEval/26] ✘ FAIL (attempt=3)
1251 [TEST ERROR PREVIEW]
1252 Traceback (most recent call last):
1253   File "C:\Users\JOIN\Downloads\Multi-
1254     Reviewer_Governance_for_Code_Generation\src\evaluator\
1255       humaneval_evaluator.py", line 39, in _worker
```

```
1254     ns["check"](candidate)
1255 File "<string>", line 12, in check
1256 AssertionError
1257 -----
1258 [HumanEval/26] [REPAIRED CODE] (after attempt=3)
1259 from typing import List
1260
1261
1262 def remove_duplicates(numbers: List[int]) -> List[int]:
1263     """ From a list of integers, remove all elements that occur more
1264     than once.
1265     Keep order of elements left the same as in the input.
1266     >>> remove_duplicates([1, 2, 3, 2, 4])
1267     [1, 3, 4]
1268     """
1269     seen = set()
1270     result = []
1271     for number in numbers:
1272         if number not in seen:
1273             seen.add(number)
1274             result.append(number)
1275         else:
1276             seen.remove(number)
1277     return result
1278 -----
1279 [HumanEval/26] Attempt 4/6
1280 [HumanEval/26] ✘ FAIL (attempt=4)
1280 [TEST ERROR PREVIEW]
1281 Traceback (most recent call last):
1282   File "C:\Users\JOIN\Downloads\Multi-
1283     Reviewer_Governance_for_Code_Generation\src\evaluator\
1284       humaneval_evaluator.py", line 39, in _worker
1285         ns["check"](candidate)
1286   File "<string>", line 12, in check
1287 AssertionError
1288 -----
1289 [HumanEval/26] [REPAIRED CODE] (after attempt=4)
1290 from typing import List
1291
1292
1293 def remove_duplicates(numbers: List[int]) -> List[int]:
1294     """ From a list of integers, remove all elements that occur more
1295     than once.
1296     Keep order of elements left the same as in the input.
1297     >>> remove_duplicates([1, 2, 3, 2, 4])
1298     [1, 3, 4]
1299     """
1300     seen = set()
1301     result = []
1302     for number in numbers:
```

```
1300         if number not in seen:
1301             seen.add(number)
1302             result.append(number)
1303         else:
1304             continue
1305     return result
1306 -----
-----
1307 [HumanEval/26] Attempt 5/6
1308 [HumanEval/26] ⚡ FAIL (attempt=5)
1309 [TEST ERROR PREVIEW]
1310 Traceback (most recent call last):
1311   File "C:\Users\JOIN\Downloads\Multi-
    Reviewer_Governance_for_Code_Generation\src\evaluator\
      humaneval_evaluator.py", line 39, in _worker
1312     ns["check"](candidate)
1313   File "<string>", line 12, in check
1314 AssertionError
1315 -----
-----
1316 [HumanEval/26] [REPAIRED CODE] (after attempt=5)
1317 from typing import List
1318
1319
1320 def remove_duplicates(numbers: List[int]) -> List[int]:
1321     """ From a list of integers, remove all elements that occur more
    than once.
1322     Keep order of elements left the same as in the input.
1323     >>> remove_duplicates([1, 2, 3, 2, 4])
1324     [1, 3, 4]
1325     """
1326     seen = set()
1327     result = []
1328     for number in numbers:
1329         if number not in seen:
1330             seen.add(number)
1331             result.append(number)
1332         else:
1333             continue
1334     return result
1335 -----
-----
1336 [HumanEval/26] Attempt 6/6
1337 [HumanEval/26] ⚡ FAIL (attempt=6)
1338 [TEST ERROR PREVIEW]
1339 Traceback (most recent call last):
1340   File "C:\Users\JOIN\Downloads\Multi-
    Reviewer_Governance_for_Code_Generation\src\evaluator\
      humaneval_evaluator.py", line 39, in _worker
1341     ns["check"](candidate)
1342   File "<string>", line 12, in check
1343 AssertionError
1344 -----
```

```
1344 -----
1345 [HumanEval/26] ⚡ FINAL FAIL after 6 retries
1346 =====
1347 =====
1347 [28/164] HumanEval/27
1348 =====
1348 =====
1349 [TASK] HumanEval/27 | entry_point=flip_case
1350 [PROMPT PREVIEW]
1351
1352
1353 def flip_case(string: str) -> str:
1354     """ For a given string, flip lowercase characters to uppercase
1354     and uppercase to lowercase.
1355     >>> flip_case('Hello')
1356     'hELLO'
1357     """
1358 -----
1358 -----
1359 [INITIAL GENERATED CODE]
1360 def flip_case(string: str) -> str:
1361     return ''.join(char.lower() if char.isupper() else char.upper()
1361     for char in string)
1362 -----
1362 -----
1363 [HumanEval/27] Attempt 0/6
1364 [HumanEval/27] ⚡ PASS (attempt=0)
1365 =====
1365 =====
1366 [29/164] HumanEval/28
1367 =====
1367 =====
1368 [TASK] HumanEval/28 | entry_point=concatenate
1369 [PROMPT PREVIEW]
1370 from typing import List
1371
1372
1373 def concatenate(strings: List[str]) -> str:
1374     """ Concatenate list of strings into a single string
1375     >>> concatenate([])
1376     ''
1377     >>> concatenate(['a', 'b', 'c'])
1378     'abc'
1379     """
1380 -----
1380 -----
1381 [INITIAL GENERATED CODE]
1382 return ''.join(strings)
1383 -----
1383 -----
1384 [HumanEval/28] Attempt 0/6
1385 [HumanEval/28] ⚡ FAIL (attempt=0)
1386 [TEST ERROR PREVIEW]
```

```
1387 Traceback (most recent call last):
1388   File "C:\Users\JOIN\Downloads\Multi-
    Reviewer_Governance_for_Code_Generation\src\evaluator\
      humaneval_evaluator.py", line 27, in _worker
1389     exec(prompt + "\n" + completion + "\n", ns, ns)
1390   File "<string>", line 12
1391 SyntaxError: 'return' outside function
1392 -----
1393 -----
1393 [HumanEval/28] [REPAIRED CODE] (after attempt=0)
1394 def join_strings(strings):
1395     return ''.join(strings)
1396 -----
1397 -----
1397 [HumanEval/28] Attempt 1/6
1398 [HumanEval/28] ⚡ FAIL (attempt=1)
1399 [TEST ERROR PREVIEW]
1400 Traceback (most recent call last):
1401   File "C:\Users\JOIN\Downloads\Multi-
    Reviewer_Governance_for_Code_Generation\src\evaluator\
      humaneval_evaluator.py", line 39, in _worker
1402     ns["check"](candidate)
1403   File "<string>", line 10, in check
1404 AssertionError
1405 -----
1406 -----
1406 [HumanEval/28] [REPAIRED CODE] (after attempt=1)
1407 def join_strings(strings):
1408     return ''.join(str(s) for s in strings)
1409 -----
1410 -----
1410 [HumanEval/28] Attempt 2/6
1411 [HumanEval/28] ⚡ FAIL (attempt=2)
1412 [TEST ERROR PREVIEW]
1413 Traceback (most recent call last):
1414   File "C:\Users\JOIN\Downloads\Multi-
    Reviewer_Governance_for_Code_Generation\src\evaluator\
      humaneval_evaluator.py", line 39, in _worker
1415     ns["check"](candidate)
1416   File "<string>", line 10, in check
1417 AssertionError
1418 -----
1419 -----
1419 [HumanEval/28] [REPAIRED CODE] (after attempt=2)
1420 def join_strings(strings):
1421     return ''.join(str(s) for s in strings)
1422 -----
1423 -----
1423 [HumanEval/28] Attempt 3/6
1424 [HumanEval/28] ⚡ FAIL (attempt=3)
1425 [TEST ERROR PREVIEW]
1426 Traceback (most recent call last):
1427   File "C:\Users\JOIN\Downloads\Multi-
```

```
1427 Reviewer_Governance_for_Code_Generation\src\evaluator\  
    humaneval_evaluator.py", line 39, in _worker  
1428     ns["check"](candidate)  
1429     File "<string>", line 10, in check  
1430 AssertionError  
1431 -----  
-----  
1432 [HumanEval/28] [REPAIRED CODE] (after attempt=3)  
1433 def join_strings(strings):  
1434     return ''.join(str(s) for s in strings)  
1435 -----  
-----  
1436 [HumanEval/28] Attempt 4/6  
1437 [HumanEval/28] ⚡ FAIL (attempt=4)  
1438 [TEST ERROR PREVIEW]  
1439 Traceback (most recent call last):  
1440     File "C:\Users\JOIN\Downloads\Multi-  
        Reviewer_Governance_for_Code_Generation\src\evaluator\  
        humaneval_evaluator.py", line 39, in _worker  
1441     ns["check"](candidate)  
1442     File "<string>", line 10, in check  
1443 AssertionError  
1444 -----  
-----  
1445 [HumanEval/28] [REPAIRED CODE] (after attempt=4)  
1446 def join_strings(strings):  
1447     return ''.join(str(s) for s in strings)  
1448 -----  
-----  
1449 [HumanEval/28] Attempt 5/6  
1450 [HumanEval/28] ⚡ FAIL (attempt=5)  
1451 [TEST ERROR PREVIEW]  
1452 Traceback (most recent call last):  
1453     File "C:\Users\JOIN\Downloads\Multi-  
        Reviewer_Governance_for_Code_Generation\src\evaluator\  
        humaneval_evaluator.py", line 39, in _worker  
1454     ns["check"](candidate)  
1455     File "<string>", line 10, in check  
1456 AssertionError  
1457 -----  
-----  
1458 [HumanEval/28] [REPAIRED CODE] (after attempt=5)  
1459 def join_strings(strings):  
1460     return ''.join(str(s) for s in strings)  
1461 -----  
-----  
1462 [HumanEval/28] Attempt 6/6  
1463 [HumanEval/28] ⚡ FAIL (attempt=6)  
1464 [TEST ERROR PREVIEW]  
1465 Traceback (most recent call last):  
1466     File "C:\Users\JOIN\Downloads\Multi-  
        Reviewer_Governance_for_Code_Generation\src\evaluator\  
        humaneval_evaluator.py", line 39, in _worker
```

```
1467     ns["check"](candidate)
1468 File "<string>", line 10, in check
1469 AssertionError
1470 -----
1471 [HumanEval/28] ⚡ FINAL FAIL after 6 retries
1472 =====
1473 [30/164] HumanEval/29
1474 =====
1475 [TASK] HumanEval/29 | entry_point=filter_by_prefix
1476 [PROMPT PREVIEW]
1477 from typing import List
1478
1479
1480 def filter_by_prefix(strings: List[str], prefix: str) -> List[str]:
1481     """ Filter an input list of strings only for ones that start
1482         with a given prefix.
1483     >>> filter_by_prefix([], 'a')
1484     []
1485     >>> filter_by_prefix(['abc', 'bcd', 'cde', 'array'], 'a')
1486     ['abc', 'array']
1487     """
1488 -----
1489 [INITIAL GENERATED CODE]
1490 def filter_by_prefix(strings: List[str], prefix: str) -> List[str]:
1491     return [s for s in strings if s.startswith(prefix)]
1492 -----
1493 [HumanEval/29] Attempt 0/6
1494 [HumanEval/29] ⚡ PASS (attempt=0)
1495 =====
1496 [31/164] HumanEval/30
1497 =====
1498 [TASK] HumanEval/30 | entry_point=get_positive
1499 [PROMPT PREVIEW]
1500
1501 def get_positive(l: list):
1502     """Return only positive numbers in the list.
1503     >>> get_positive([-1, 2, -4, 5, 6])
1504     [2, 5, 6]
1505     >>> get_positive([5, 3, -5, 2, -3, 3, 9, 0, 123, 1, -10])
1506     [5, 3, 2, 3, 9, 123, 1]
1507     """
1508 -----
1509 [INITIAL GENERATED CODE]
1510 def get_positive(l: list):
```

```
1511     return [x for x in l if x > 0]
1512 -----
1513 -----
1513 [HumanEval/30] Attempt 0/6
1514 [HumanEval/30] 0 PASS (attempt=0)
1515 =====
1516 -----
1516 [32/164] HumanEval/31
1517 =====
1517 -----
1518 [TASK] HumanEval/31 | entry_point=is_prime
1519 [PROMPT PREVIEW]
1520
1521
1522 def is_prime(n):
1523     """Return true if a given number is prime, and false otherwise.
1524     >>> is_prime(6)
1525     False
1526     >>> is_prime(101)
1527     True
1528     >>> is_prime(11)
1529     True
1530     >>> is_prime(13441)
1531     True
1532     >>> is_prime(61)
1533     True
1534     >>> is_prime(4)
1535     False
1536     >>> is_prime(1)
1537     False
1538     """
1539 -----
1539 -----
1540 [INITIAL GENERATED CODE]
1541 def is_prime(n):
1542     if n <= 1:
1543         return False
1544     for i in range(2, int(n**0.5) + 1):
1545         if n % i == 0:
1546             return False
1547     return True
1548 -----
1548 -----
1549 [HumanEval/31] Attempt 0/6
1550 [HumanEval/31] 0 PASS (attempt=0)
1551 =====
1551 -----
1552 [33/164] HumanEval/32
1553 =====
1553 -----
1554 [TASK] HumanEval/32 | entry_point=find_zero
1555 [PROMPT PREVIEW]
1556 import math
```

```

1557
1558
1559 def poly(xs: list, x: float):
1560     """
1561         Evaluates polynomial with coefficients xs at point x.
1562         return xs[0] + xs[1] * x + xs[1] * x^2 + .... xs[n] * x^n
1563     """
1564     return sum([coeff * math.pow(x, i) for i, coeff in enumerate(xs
1565 )])
1565
1566
1567 def find_zero(xs: list):
1568     """ xs are coefficients of a polynomial.
1569     find_zero find x such that poly(x) = 0.
1570     find_zero returns only one zero point.
1571     -----
1572 [INITIAL GENERATED CODE]
1573 def find_zero(xs: list):
1574     """ xs are coefficients of a polynomial.
1575     find_zero finds x such that poly(x) = 0.
1576     find_zero returns only one zero point, even if there are many.
1577     Moreover, find_zero only takes list xs having even number of
1578     coefficients
1579     and largest non zero coefficient as it guarantees
1580     a solution.
1581     >>> round(find_zero([1, 2]), 2) # f(x) = 1 + 2x
1582     -0.5
1583     >>> round(find_zero([-6, 11, -6, 1]), 2) # (x - 1) * (x - 2) * (
1584     x - 3) = -6 + 11x - 6x^2 + x^3
1585     1.0
1586     """
1587
1588     if len(xs) % 2 != 0 or xs[-1] == 0:
1589         raise ValueError("Invalid input coefficients")
1590
1591     def f(x):
1592         return poly(xs, x)
1593
1594     a, b = -1000, 1000 # Initial bounds
1595     while abs(b - a) > 1e-6:
1596         mid = (a + b) / 2
1597         if f(mid) * f(a) < 0:
1598             b = mid
1599         else:
1600             a = mid
1601
1602     return round((a + b) / 2, 2)
1603
1604 -----
1605 [HumanEval/32] Attempt 0/6
1606 [HumanEval/32] ⚡ FAIL (attempt=0)
1607 [TEST ERROR PREVIEW]
1608 Traceback (most recent call last):

```

```
1605  File "C:\Users\JOIN\Downloads\Multi-
    Reviewer_Governance_for_Code_Generation\src\evaluator\
        humaneval_evaluator.py", line 39, in _worker
1606      ns["check"](candidate)
1607  File "<string>", line 20, in check
1608 AssertionException
1609 -----
-----
1610 [HumanEval/32] [REPAIRED CODE] (after attempt=0)
1611 def find_zero(xs: list):
1612     """ xs are coefficients of a polynomial.
1613     find_zero find x such that poly(x) = 0.
1614     find_zero returns only one zero point, even if there are many.
1615     Moreover, find_zero only takes list xs having even number of
1616     coefficients
1617     and largest non zero coefficient as it guarantees
1618     a solution.
1619     >>> round(find_zero([1, 2]), 2) # f(x) = 1 + 2x
1620     -0.5
1621     >>> round(find_zero([-6, 11, -6, 1]), 2) # (x - 1) * (x - 2) * (
1622         x - 3) = -6 + 11x - 6x^2 + x^3
1623     1.0
1624     """
1625
1626     if len(xs) % 2 != 0 or xs[-1] == 0:
1627         raise ValueError("Invalid input coefficients")
1628
1629     def poly(coeffs, x):
1630         return sum(c * x**i for i, c in enumerate(reversed(coeffs)))
1631
1632     def f(x):
1633         return poly(xs, x)
1634
1635     a, b = -1000, 1000 # Initial bounds
1636     while abs(b - a) > 1e-6:
1637         mid = (a + b) / 2
1638         if f(mid) * f(a) < 0:
1639             b = mid
1640         else:
1641             a = mid
1642
1643     return round((a + b) / 2, 2)
1644 -----
-----
1645 [HumanEval/32] Attempt 1/6
1646 [HumanEval/32] ⚡ FAIL (attempt=1)
1647 [TEST ERROR PREVIEW]
1648 Traceback (most recent call last):
1649   File "C:\Users\JOIN\Downloads\Multi-
    Reviewer_Governance_for_Code_Generation\src\evaluator\
        humaneval_evaluator.py", line 39, in _worker
1650      ns["check"](candidate)
1651  File "<string>", line 20, in check
1652 AssertionException
```

```

1650 -----
-----
1651 [HumanEval/32] [REPAIRED CODE] (after attempt=1)
1652 def find_zero(xs: list):
1653     """ xs are coefficients of a polynomial.
1654     find_zero find x such that poly(x) = 0.
1655     find_zero returns only one zero point, even if there are many.
1656     Moreover, find_zero only takes list xs having even number of
1657     coefficients
1658     and largest non zero coefficient as it guarantees
1659     a solution.
1660     >>> round(find_zero([1, 2]), 2) # f(x) = 1 + 2x
1661     -0.5
1662     >>> round(find_zero([-6, 11, -6, 1]), 2) # (x - 1) * (x - 2) * (
1663         x - 3) = -6 + 11x - 6x^2 + x^3
1664     1.0
1665     """
1666     if len(xs) % 2 != 0 or xs[-1] == 0:
1667         raise ValueError("Invalid input coefficients")
1668
1669     def poly(coeffs, x):
1670         return sum(c * x**i for i, c in enumerate(reversed(coeffs)))
1671
1672     def f(x):
1673         return poly(xs, x)
1674
1675     a, b = -1000, 1000 # Initial bounds
1676     while abs(b - a) > 1e-6:
1677         mid = (a + b) / 2
1678         if f(mid) * f(a) < 0:
1679             b = mid
1680         else:
1681             a = mid
1682
1683     return round((a + b) / 2, 2)
1684 -----
1685 [HumanEval/32] Attempt 2/6
1686 [HumanEval/32] ⚡ FAIL (attempt=2)
1687 [TEST ERROR PREVIEW]
1688 Traceback (most recent call last):
1689   File "C:\Users\JOIN\Downloads\Multi-
1690   Reviewer_Governance_for_Code_Generation\src\evaluator\
1691   humaneval_evaluator.py", line 39, in _worker
1692     ns["check"](candidate)
1693   File "<string>", line 20, in check
1694 Assertion
1695 Error
1696 -----
1697 [HumanEval/32] [REPAIRED CODE] (after attempt=2)
1698 def find_zero(xs: list):
1699     """ xs are coefficients of a polynomial.
1700     find_zero find x such that poly(x) = 0.

```

```

1696     find_zero returns only one zero point, even if there are many.
1697     Moreover, find_zero only takes list xs having even number of
1698     coefficients
1699     and largest non zero coefficient as it guarantees
1700     a solution.
1700     >>> round(find_zero([1, 2]), 2) # f(x) = 1 + 2x
1701     -0.5
1702     >>> round(find_zero([-6, 11, -6, 1]), 2) # (x - 1) * (x - 2) * (
1703     x - 3) = -6 + 11x - 6x^2 + x^3
1703     1.0
1704     """
1705     if len(xs) % 2 != 0 or xs[-1] == 0:
1706         raise ValueError("Invalid input coefficients")
1707
1708     def poly(coeffs, x):
1709         return sum(c * x**i for i, c in enumerate(reversed(coeffs)))
1710
1711     def f(x):
1712         return poly(xs, x)
1713
1714     a, b = -1000, 1000 # Initial bounds
1715     while abs(b - a) > 1e-6:
1716         mid = (a + b) / 2
1717         if f(mid) * f(a) < 0:
1718             b = mid
1719         else:
1720             a = mid
1721
1722     return round((a + b) / 2, 2)
1723 -----
-----
1724 [HumanEval/32] Attempt 3/6
1725 [HumanEval/32] ⚡ FAIL (attempt=3)
1726 [TEST ERROR PREVIEW]
1727 Traceback (most recent call last):
1728   File "C:\Users\JOIN\Downloads\Multi-
    Reviewer_Governance_for_Code_Generation\src\evaluator\
      humaneval_evaluator.py", line 39, in _worker
1729     ns["check"](candidate)
1730   File "<string>", line 20, in check
1731 Assertion
1732 -----
-----
1733 [HumanEval/32] [REPAIRED CODE] (after attempt=3)
1734 def find_zero(xs: list):
1735     """ xs are coefficients of a polynomial.
1736     find_zero find x such that poly(x) = 0.
1737     find_zero returns only one zero point, even if there are many.
1738     Moreover, find_zero only takes list xs having even number of
1739     coefficients
1740     and largest non zero coefficient as it guarantees
1741     a solution.
1741     >>> round(find_zero([1, 2]), 2) # f(x) = 1 + 2x

```

```

1742     -0.5
1743     >>> round(find_zero([-6, 11, -6, 1]), 2) # (x - 1) * (x - 2) * (
1744         x - 3) = -6 + 11x - 6x^2 + x^3
1744     1.0
1745     """
1746     if len(xs) % 2 != 0 or xs[-1] == 0:
1747         raise ValueError("Invalid input coefficients")
1748
1749     def poly(coeffs, x):
1750         return sum(c * x**i for i, c in enumerate(reversed(coeffs)))
1751
1752     def f(x):
1753         return poly(xs, x)
1754
1755     a, b = -1000, 1000 # Initial bounds
1756     while abs(b - a) > 1e-6:
1757         mid = (a + b) / 2
1758         if f(mid) * f(a) < 0:
1759             b = mid
1760         else:
1761             a = mid
1762
1763     return round((a + b) / 2, 2)
1764 -----
1765 -----
1765 [HumanEval/32] Attempt 4/6
1766 [HumanEval/32] ⚡ FAIL (attempt=4)
1767 [TEST ERROR PREVIEW]
1768 Traceback (most recent call last):
1769   File "C:\Users\JOIN\Downloads\Multi-
1770     Reviewer_Governance_for_Code_Generation\src\evaluator\
1771       humaneval_evaluator.py", line 39, in _worker
1770       ns["check"])(candidate)
1771   File "<string>", line 20, in check
1772 AssertionException
1773 -----
1774 -----
1774 [HumanEval/32] [REPAIRED CODE] (after attempt=4)
1775 def find_zero(xs: list):
1776     """ xs are coefficients of a polynomial.
1777     find_zero find x such that poly(x) = 0.
1778     find_zero returns only one zero point, even if there are many.
1779     Moreover, find_zero only takes list xs having even number of
1780     coefficients
1781     and largest non zero coefficient as it guarantees
1782     a solution.
1783     >>> round(find_zero([1, 2]), 2) # f(x) = 1 + 2x
1783     -0.5
1784     >>> round(find_zero([-6, 11, -6, 1]), 2) # (x - 1) * (x - 2) * (
1784         x - 3) = -6 + 11x - 6x^2 + x^3
1785     1.0
1786     """
1787     if len(xs) % 2 != 0 or xs[-1] == 0:

```

```
1788         raise ValueError("Invalid input coefficients")
1789
1790     def poly(coeffs, x):
1791         return sum(c * x**i for i, c in enumerate(reversed(coeffs)))
1792
1793     def f(x):
1794         return poly(xs, x)
1795
1796     a, b = -1000, 1000 # Initial bounds
1797     while abs(b - a) > 1e-6:
1798         mid = (a + b) / 2
1799         if f(mid) * f(a) < 0:
1800             b = mid
1801         else:
1802             a = mid
1803
1804     return round((a + b) / 2, 2)
1805 -----
1806 -----
1806 [HumanEval/32] Attempt 5/6
1807 [HumanEval/32] Ø FAIL (attempt=5)
1808 [TEST ERROR PREVIEW]
1809 Traceback (most recent call last):
1810   File "C:\Users\JOIN\Downloads\Multi-
1811     Reviewer_Governance_for_Code_Generation\src\evaluator\
1812       humaneval_evaluator.py", line 39, in _worker
1813         ns["check"](candidate)
1814   File "<string>", line 20, in check
1815 Assertion
1816 -----
1816 -----
1815 [HumanEval/32] [REPAIRED CODE] (after attempt=5)
1816 def find_zero(xs: list):
1817     """ xs are coefficients of a polynomial.
1818     find_zero find x such that poly(x) = 0.
1819     find_zero returns only one zero point, even if there are many.
1820     Moreover, find_zero only takes list xs having even number of
1821     coefficients
1822     and largest non zero coefficient as it guarantees
1823     a solution.
1824     >>> round(find_zero([1, 2]), 2) # f(x) = 1 + 2x
1825     -0.5
1826     >>> round(find_zero([-6, 11, -6, 1]), 2) # (x - 1) * (x - 2) * (
1827     x - 3) = -6 + 11x - 6x^2 + x^3
1828     1.0
1829     """
1830
1831     if len(xs) % 2 != 0 or xs[-1] == 0:
1832         raise ValueError("Invalid input coefficients")
1833
1834     def poly(coeffs, x):
1835         return sum(c * x**i for i, c in enumerate(reversed(coeffs)))
```

```
1835         return poly(xs, x)
1836
1837     a, b = -1000, 1000 # Initial bounds
1838     while abs(b - a) > 1e-6:
1839         mid = (a + b) / 2
1840         if f(mid) * f(a) < 0:
1841             b = mid
1842         else:
1843             a = mid
1844
1845     return round((a + b) / 2, 2)
1846 -----
-----
1847 [HumanEval/32] Attempt 6/6
1848 [HumanEval/32] ⚡ FAIL (attempt=6)
1849 [TEST ERROR PREVIEW]
1850 Traceback (most recent call last):
1851   File "C:\Users\JOIN\Downloads\Multi-
    Reviewer_Governance_for_Code_Generation\src\evaluator\
      humaneval_evaluator.py", line 39, in _worker
1852     ns["check"](candidate)
1853   File "<string>", line 20, in check
1854 AssertionError
1855 -----
-----
1856 [HumanEval/32] ⚡ FINAL FAIL after 6 retries
1857 =====
=====
1858 [34/164] HumanEval/33
1859 =====
=====
1860 [TASK] HumanEval/33 | entry_point=sort_third
1861 [PROMPT PREVIEW]
1862
1863
1864 def sort_third(l: list):
1865     """This function takes a list l and returns a list l' such that
1866     l' is identical to l in the indices that are not divisible by
1867     three, while its values at the indices that are divisible by three
1868     are equal
1869     to the values of the corresponding indices of l, but sorted.
1870     """
1871     >>> sort_third([1, 2, 3])
1872     [1, 2, 3]
1873     >>> sort_third([5, 6, 3, 4, 8, 9, 2])
1874     [2, ...
1875
1876 -----
-----
1877 [INITIAL GENERATED CODE]
1878 def sort_third(l: list):
1879     indices_divisible_by_three = [i for i in range(len(l)) if i % 3
1880                                   == 0]
1881     sorted_values = sorted([l[i] for i in indices_divisible_by_three
1882                           ])
```

```
1877     result = l[:]
1878     for original_index, new_value in zip(indices_divisible_by_three
1880         return result
1881 -----
1882 [HumanEval/33] Attempt 0/6
1883 [HumanEval/33] [] PASS (attempt=0)
1884 =====
1885 [35/164] HumanEval/34
1886 =====
1887 [TASK] HumanEval/34 | entry_point=unique
1888 [PROMPT PREVIEW]
1889
1890
1891 def unique(l: list):
1892     """Return sorted unique elements in a list
1893     >>> unique([5, 3, 5, 2, 3, 3, 9, 0, 123])
1894     [0, 2, 3, 5, 9, 123]
1895     """
1896 -----
1897 [INITIAL GENERATED CODE]
1898 def unique(l: list):
1899     """Return sorted unique elements in a list
1900     >>> unique([5, 3, 5, 2, 3, 3, 9, 0, 123])
1901     [0, 2, 3, 5, 9, 123]
1902     """
1903     return sorted(set(l))
1904 -----
1905 [HumanEval/34] Attempt 0/6
1906 [HumanEval/34] [] PASS (attempt=0)
1907 =====
1908 [36/164] HumanEval/35
1909 =====
1910 [TASK] HumanEval/35 | entry_point=max_element
1911 [PROMPT PREVIEW]
1912
1913
1914 def max_element(l: list):
1915     """Return maximum element in the list.
1916     >>> max_element([1, 2, 3])
1917     3
1918     >>> max_element([5, 3, -5, 2, -3, 3, 9, 0, 123, 1, -10])
1919     123
1920     """
1921 -----
```

```
1921 -----
1922 [INITIAL GENERATED CODE]
1923 def max_element(l: list):
1924     return max(l)
1925 -----
1926 -----
1926 [HumanEval/35] Attempt 0/6
1927 [HumanEval/35] ⚡ PASS (attempt=0)
1928 =====
1929 =====
1929 [37/164] HumanEval/36
1930 =====
1931 [TASK] HumanEval/36 | entry_point=fizz_buzz
1932 [PROMPT PREVIEW]
1933
1934
1935 def fizz_buzz(n: int):
1936     """Return the number of times the digit 7 appears in integers
1937     less than n which are divisible by 11 or 13.
1937     >>> fizz_buzz(50)
1938     0
1939     >>> fizz_buzz(78)
1940     2
1941     >>> fizz_buzz(79)
1942     3
1943     """
1944 -----
1945 -----
1945 [INITIAL GENERATED CODE]
1946 def fizz_buzz(n: int) -> int:
1947     count = 0
1948     for i in range(1, n):
1949         if i % 11 == 0 or i % 13 == 0:
1950             count += str(i).count('7')
1951     return count
1952 -----
1953 -----
1953 [HumanEval/36] Attempt 0/6
1954 [HumanEval/36] ⚡ PASS (attempt=0)
1955 =====
1956 =====
1956 [38/164] HumanEval/37
1957 =====
1958 [TASK] HumanEval/37 | entry_point=sort_even
1959 [PROMPT PREVIEW]
1960
1961
1962 def sort_even(l: list):
1963     """This function takes a list l and returns a list l' such that
1964     l' is identical to l in the odd indicies, while its values at
1964     the even indicies are equal
```

```
1965     to the values of the even indicies of l, but sorted.
1966     >>> sort_even([1, 2, 3])
1967     [1, 2, 3]
1968     >>> sort_even([5, 6, 3, 4])
1969     [3, 6, 5, 4]
1970     """
1971 -----
1972 -----
1972 [INITIAL GENERATED CODE]
1973 def sort_even(l: list):
1974     even_indices = [i for i in range(len(l)) if i % 2 == 0]
1975     sorted_evens = sorted([l[i] for i in even_indices])
1976     result = l[:]
1977     for i, value in zip(even_indices, sorted_evens):
1978         result[i] = value
1979     return result
1980 -----
1981 -----
1981 [HumanEval/37] Attempt 0/6
1982 [HumanEval/37] 0 PASS (attempt=0)
1983 =====
1984 [39/164] HumanEval/38
1985 =====
1986 [TASK] HumanEval/38 | entry_point=decode_cyclic
1987 [PROMPT PREVIEW]
1988
1989
1990 def encode_cyclic(s: str):
1991     """
1992     returns encoded string by cycling groups of three characters.
1993     """
1994     # split string to groups. Each of length 3.
1995     groups = [s[(3 * i):min((3 * i + 3), len(s))] for i in range((len(s) + 2) // 3)]
1996     # cycle elements in each group. Unless group has fewer elements
1997     # than 3.
1997     groups = [(group[1:] + group[0]) if len(group) == 3 else group
1997     for group in...]
1998 -----
1999 -----
1999 [INITIAL GENERATED CODE]
2000 def decode_cyclic(s: str):
2001     """
2002     takes as input string encoded with encode_cyclic function.
2002     Returns decoded string.
2003     """
2004     # split string to groups. Each of length 3.
2005     groups = [s[(3 * i):min((3 * i + 3), len(s))] for i in range((len(s) + 2) // 3)]
2006     # cycle elements in each group back. Unless group has fewer
2006     # elements than 3.
```

```
2007     groups = [(group[-1] + group[:-1]) if len(group) == 3 else group
2008         for group in groups]
2009     return "".join(groups)
2010 -----
2011 -----
2010 [HumanEval/38] Attempt 0/6
2011 [HumanEval/38] ✘ PASS (attempt=0)
2012 =====
2013 -----
2013 [40/164] HumanEval/39
2014 =====
2014 -----
2015 [TASK] HumanEval/39 | entry_point=prime_fib
2016 [PROMPT PREVIEW]
2017
2018
2019 def prime_fib(n: int):
2020     """
2021         prime_fib returns n-th number that is a Fibonacci number and it's
2021         also prime.
2022         >>> prime_fib(1)
2023         2
2024         >>> prime_fib(2)
2025         3
2026         >>> prime_fib(3)
2027         5
2028         >>> prime_fib(4)
2029         13
2030         >>> prime_fib(5)
2031         89
2032         """
2033 -----
2034 -----
2034 [INITIAL GENERATED CODE]
2035 from sympy import isprime
2036
2037 def prime_fib(n: int) -> int:
2038     def fib_generator():
2039         a, b = 0, 1
2040         while True:
2041             yield b
2042             a, b = b, a + b
2043
2044         count = 0
2045         for fib_num in fib_generator():
2046             if isprime(fib_num):
2047                 count += 1
2048                 if count == n:
2049                     return fib_num
2050 -----
2051 -----
2051 [HumanEval/39] Attempt 0/6
2052 [HumanEval/39] ✘ FAIL (attempt=0)
```

```
2053 [TEST ERROR PREVIEW]
2054 Traceback (most recent call last):
2055   File "C:\Users\JOIN\Downloads\Multi-
    Reviewer_Governance_for_Code_Generation\src\evaluator\
      humaneval_evaluator.py", line 27, in _worker
2056     exec(prompt + "\n" + completion + "\n", ns, ns)
2057   File "<string>", line 18, in <module>
2058 ModuleNotFoundError: No module named 'sympy'
2059 -----
-----
2060 [HumanEval/39] [REPAIRED CODE] (after attempt=0)
2061 import sympy
2062
2063 from sympy import isprime
2064
2065 def prime_fib(n: int) -> int:
2066     def fib_generator():
2067         a, b = 0, 1
2068         while True:
2069             yield b
2070             a, b = b, a + b
2071
2072         count = 0
2073         for fib_num in fib_generator():
2074             if isprime(fib_num):
2075                 count += 1
2076                 if count == n:
2077                     return fib_num
2078 -----
-----
2079 [HumanEval/39] Attempt 1/6
2080 [HumanEval/39] ⚡ FAIL (attempt=1)
2081 [TEST ERROR PREVIEW]
2082 Traceback (most recent call last):
2083   File "C:\Users\JOIN\Downloads\Multi-
    Reviewer_Governance_for_Code_Generation\src\evaluator\
      humaneval_evaluator.py", line 27, in _worker
2084     exec(prompt + "\n" + completion + "\n", ns, ns)
2085   File "<string>", line 18, in <module>
2086 ModuleNotFoundError: No module named 'sympy'
2087 -----
-----
2088 [HumanEval/39] [REPAIRED CODE] (after attempt=1)
2089 import sympy
2090
2091 from sympy import isprime
2092
2093 def prime_fib(n: int) -> int:
2094     def fib_generator():
2095         a, b = 0, 1
2096         while True:
2097             yield b
2098             a, b = b, a + b
```

```
2099
2100     count = 0
2101     for fib_num in fib_generator():
2102         if isprime(fib_num):
2103             count += 1
2104             if count == n:
2105                 return fib_num
2106 -----
2107 -----
2107 [HumanEval/39] Attempt 2/6
2108 [HumanEval/39] ⚡ FAIL (attempt=2)
2109 [TEST ERROR PREVIEW]
2110 Traceback (most recent call last):
2111   File "C:\Users\JOIN\Downloads\Multi-
2112     Reviewer_Governance_for_Code_Generation\src\evaluator\
2113       humaneval_evaluator.py", line 27, in _worker
2114       exec(prompt + "\n" + completion + "\n", ns, ns)
2115     File "<string>", line 18, in <module>
2116     ModuleNotFoundError: No module named 'sympy'
2117 -----
2118 -----
2116 [HumanEval/39] [REPAIRED CODE] (after attempt=2)
2117 import sympy
2118
2119 from sympy import isprime
2120
2121 def prime_fib(n: int) -> int:
2122     def fib_generator():
2123         a, b = 0, 1
2124         while True:
2125             yield b
2126             a, b = b, a + b
2127
2128         count = 0
2129         for fib_num in fib_generator():
2130             if isprime(fib_num):
2131                 count += 1
2132                 if count == n:
2133                     return fib_num
2134 -----
2135 -----
2135 [HumanEval/39] Attempt 3/6
2136 [HumanEval/39] ⚡ FAIL (attempt=3)
2137 [TEST ERROR PREVIEW]
2138 Traceback (most recent call last):
2139   File "C:\Users\JOIN\Downloads\Multi-
2140     Reviewer_Governance_for_Code_Generation\src\evaluator\
2141       humaneval_evaluator.py", line 27, in _worker
2142       exec(prompt + "\n" + completion + "\n", ns, ns)
2143     File "<string>", line 18, in <module>
2144     ModuleNotFoundError: No module named 'sympy'
2145 -----
2146 -----
```

```
2144 [HumanEval/39] [REPAIRED CODE] (after attempt=3)
2145 import sympy
2146
2147 from sympy import isprime
2148
2149 def prime_fib(n: int) -> int:
2150     def fib_generator():
2151         a, b = 0, 1
2152         while True:
2153             yield b
2154             a, b = b, a + b
2155
2156         count = 0
2157         for fib_num in fib_generator():
2158             if isprime(fib_num):
2159                 count += 1
2160                 if count == n:
2161                     return fib_num
2162 -----
2163 -----
2163 [HumanEval/39] Attempt 4/6
2164 [HumanEval/39] ⚡ FAIL (attempt=4)
2165 [TEST ERROR PREVIEW]
2166 Traceback (most recent call last):
2167   File "C:\Users\JOIN\Downloads\Multi-
2168     Reviewer_Governance_for_Code_Generation\src\evaluator\
2169     humaneval_evaluator.py", line 27, in _worker
2170       exec(prompt + "\n" + completion + "\n", ns, ns)
2171     File "<string>", line 18, in <module>
2170 ModuleNotFoundError: No module named 'sympy'
2171 -----
2172 -----
2172 [HumanEval/39] [REPAIRED CODE] (after attempt=4)
2173 import sympy
2174
2175 from sympy import isprime
2176
2177 def prime_fib(n: int) -> int:
2178     def fib_generator():
2179         a, b = 0, 1
2180         while True:
2181             yield b
2182             a, b = b, a + b
2183
2184         count = 0
2185         for fib_num in fib_generator():
2186             if isprime(fib_num):
2187                 count += 1
2188                 if count == n:
2189                     return fib_num
2190 -----
2191 -----
2191 [HumanEval/39] Attempt 5/6
```

```
2192 [HumanEval/39] ⚠ FAIL (attempt=5)
2193 [TEST ERROR PREVIEW]
2194 Traceback (most recent call last):
2195   File "C:\Users\JOIN\Downloads\Multi-
    Reviewer_Governance_for_Code_Generation\src\evaluator\
      humaneval_evaluator.py", line 27, in _worker
2196     exec(prompt + "\n" + completion + "\n", ns, ns)
2197   File "<string>", line 18, in <module>
2198 ModuleNotFoundError: No module named 'sympy'
2199 -----
-----
2200 [HumanEval/39] [REPAIRED CODE] (after attempt=5)
2201 import sympy
2202
2203 from sympy import isprime
2204
2205 def prime_fib(n: int) -> int:
2206     def fib_generator():
2207         a, b = 0, 1
2208         while True:
2209             yield b
2210             a, b = b, a + b
2211
2212         count = 0
2213         for fib_num in fib_generator():
2214             if isprime(fib_num):
2215                 count += 1
2216                 if count == n:
2217                     return fib_num
2218 -----
-----
2219 [HumanEval/39] Attempt 6/6
2220 [HumanEval/39] ⚠ FAIL (attempt=6)
2221 [TEST ERROR PREVIEW]
2222 Traceback (most recent call last):
2223   File "C:\Users\JOIN\Downloads\Multi-
    Reviewer_Governance_for_Code_Generation\src\evaluator\
      humaneval_evaluator.py", line 27, in _worker
2224     exec(prompt + "\n" + completion + "\n", ns, ns)
2225   File "<string>", line 18, in <module>
2226 ModuleNotFoundError: No module named 'sympy'
2227 -----
-----
2228 [HumanEval/39] ⚠ FINAL FAIL after 6 retries
2229 =====
=====
2230 [41/164] HumanEval/40
2231 =====
=====
2232 [TASK] HumanEval/40 | entry_point=triples_sum_to_zero
2233 [PROMPT PREVIEW]
2234
2235
```

```
2236 def triples_sum_to_zero(l: list):
2237     """
2238         triples_sum_to_zero takes a list of integers as an input.
2239         it returns True if there are three distinct elements in the list
2240         that
2241             sum to zero, and False otherwise.
2242
2243         >>> triples_sum_to_zero([1, 3, 5, 0])
2244         False
2245         >>> triples_sum_to_zero([1, 3, -2, 1])
2246         True
2247         >>> triples_sum_to_zero([1, 2, 3, 7])
2248         False
2249         >>> triples_sum_to_z...
2249 -----
2250 -----
2250 [INITIAL GENERATED CODE]
2251 def triples_sum_to_zero(l: list):
2252     n = len(l)
2253     for i in range(n-2):
2254         for j in range(i+1, n-1):
2255             for k in range(j+1, n):
2256                 if l[i] + l[j] + l[k] == 0:
2257                     return True
2258     return False
2259 -----
2259 -----
2260 [HumanEval/40] Attempt 0/6
2261 [HumanEval/40] ✘ PASS (attempt=0)
2262 =====
2262 =====
2263 [42/164] HumanEval/41
2264 =====
2264 =====
2265 [TASK] HumanEval/41 | entry_point=car_race_collision
2266 [PROMPT PREVIEW]
2267
2268
2269 def car_race_collision(n: int):
2270     """
2271         Imagine a road that's a perfectly straight infinitely long line.
2272         n cars are driving left to right; simultaneously, a different
2273             set of n cars
2274             are driving right to left. The two sets of cars start out
2275                 being very far from
2276                 each other. All cars move in the same speed. Two cars are said
2277                     to collide
2278                     when a car that's moving left to right h...
2279
2279 -----
2279 -----
2277 [INITIAL GENERATED CODE]
2278 def car_race_collision(n: int) -> int:
2279     return n * n
```

```
2280 -----
-----
2281 [HumanEval/41] Attempt 0/6
2282 [HumanEval/41] ✘ PASS (attempt=0)
2283 =====
=====
2284 [43/164] HumanEval/42
2285 =====
=====
2286 [TASK] HumanEval/42 | entry_point=incr_list
2287 [PROMPT PREVIEW]
2288
2289
2290 def incr_list(l: list):
2291     """Return list with elements incremented by 1.
2292     >>> incr_list([1, 2, 3])
2293     [2, 3, 4]
2294     >>> incr_list([5, 3, 5, 2, 3, 3, 9, 0, 123])
2295     [6, 4, 6, 3, 4, 4, 10, 1, 124]
2296     """
2297 -----
-----
2298 [INITIAL GENERATED CODE]
2299 def incr_list(l: list):
2300     return [x + 1 for x in l]
2301 -----
-----
2302 [HumanEval/42] Attempt 0/6
2303 [HumanEval/42] ✘ PASS (attempt=0)
2304 =====
=====
2305 [44/164] HumanEval/43
2306 =====
=====
2307 [TASK] HumanEval/43 | entry_point=pairs_sum_to_zero
2308 [PROMPT PREVIEW]
2309
2310
2311 def pairs_sum_to_zero(l):
2312     """
2313     pairs_sum_to_zero takes a list of integers as an input.
2314     it returns True if there are two distinct elements in the list
2315     that
2316         sum to zero, and False otherwise.
2317     >>> pairs_sum_to_zero([1, 3, 5, 0])
2318     False
2319     >>> pairs_sum_to_zero([1, 3, -2, 1])
2320     False
2321     >>> pairs_sum_to_zero([1, 2, 3, 7])
2322     False
2323     >>> pairs_sum_to_zero([2, 4, -5, 3, 5,...
```

```
2324 [INITIAL GENERATED CODE]
2325 def pairs_sum_to_zero(l):
2326     seen = set()
2327     for num in l:
2328         if -num in seen:
2329             return True
2330         seen.add(num)
2331     return False
2332 -----
2333 -----
2333 [HumanEval/43] Attempt 0/6
2334 [HumanEval/43] ✘ PASS (attempt=0)
2335 =====
2336 [45/164] HumanEval/44
2337 =====
2338 [TASK] HumanEval/44 | entry_point=change_base
2339 [PROMPT PREVIEW]
2340
2341
2342 def change_base(x: int, base: int):
2343     """Change numerical base of input number x to base.
2344     return string representation after the conversion.
2345     base numbers are less than 10.
2346     >>> change_base(8, 3)
2347     '22'
2348     >>> change_base(8, 2)
2349     '1000'
2350     >>> change_base(7, 2)
2351     '111'
2352     """
2353 -----
2354 -----
2354 [INITIAL GENERATED CODE]
2355 def change_base(x: int, base: int) -> str:
2356     if x == 0:
2357         return "0"
2358     digits = []
2359     while x:
2360         digits.append(int(x % base))
2361         x //= base
2362     return ''.join(str(x) for x in digits[::-1])
2363 -----
2364 -----
2364 [HumanEval/44] Attempt 0/6
2365 [HumanEval/44] ✘ PASS (attempt=0)
2366 =====
2367 [46/164] HumanEval/45
2368 =====
2369 [TASK] HumanEval/45 | entry_point=triangle_area
```

```
2370 [PROMPT PREVIEW]
2371
2372
2373 def triangle_area(a, h):
2374     """Given length of a side and high return area for a triangle.
2375     >>> triangle_area(5, 3)
2376     7.5
2377     """
2378 -----
2379 -----
2380 return 0.5 * a * h
2381 -----
2382 -----
2383 [HumanEval/45] Attempt 0/6
2384 [HumanEval/45] ⚡ FAIL (attempt=0)
2385 [TEST ERROR PREVIEW]
2386 Traceback (most recent call last):
2387   File "C:\Users\JOIN\Downloads\Multi-
2388     Reviewer_Governance_for_Code_Generation\src\evaluator\
2389       humaneval_evaluator.py", line 27, in _worker
2390         exec(prompt + "\n" + completion + "\n", ns, ns)
2391   File "<string>", line 9
2392 SyntaxError: 'return' outside function
2393 -----
2394 -----
2395 [HumanEval/45] [REPAIRED CODE] (after attempt=0)
2396 def calculate_area(a, h):
2397     return 0.5 * a * h
2398 -----
2399 -----
2400 [HumanEval/45] Attempt 1/6
2401 [HumanEval/45] ⚡ FAIL (attempt=1)
2402 [TEST ERROR PREVIEW]
2403 Traceback (most recent call last):
2404   File "C:\Users\JOIN\Downloads\Multi-
2405     Reviewer_Governance_for_Code_Generation\src\evaluator\
2406       humaneval_evaluator.py", line 39, in _worker
2407         ns["check"](candidate)
2408   File "<string>", line 7, in check
2409 AssertionException
2410 -----
2411 -----
2412 [HumanEval/45] [REPAIRED CODE] (after attempt=1)
2413 def calculate_area(a, h):
2414     return 0.5 * a * h
2415 -----
2416 -----
2417 [HumanEval/45] Attempt 2/6
2418 [HumanEval/45] ⚡ FAIL (attempt=2)
2419 [TEST ERROR PREVIEW]
2420 Traceback (most recent call last):
2421   File "C:\Users\JOIN\Downloads\Multi-
```

```
2412 Reviewer_Governance_for_Code_Generation\src\evaluator\  
    humaneval_evaluator.py", line 39, in _worker  
2413     ns["check"](candidate)  
2414     File "<string>", line 7, in check  
2415 AssertionError  
2416 -----  
-----  
2417 [HumanEval/45] [REPAIRED CODE] (after attempt=2)  
2418 def calculate_area(a, h):  
2419     return 0.5 * a * h  
2420 -----  
-----  
2421 [HumanEval/45] Attempt 3/6  
2422 [HumanEval/45] ⚡ FAIL (attempt=3)  
2423 [TEST ERROR PREVIEW]  
2424 Traceback (most recent call last):  
2425     File "C:\Users\JOIN\Downloads\Multi-  
        Reviewer_Governance_for_Code_Generation\src\evaluator\  
        humaneval_evaluator.py", line 39, in _worker  
2426     ns["check"](candidate)  
2427     File "<string>", line 7, in check  
2428 AssertionError  
2429 -----  
-----  
2430 [HumanEval/45] [REPAIRED CODE] (after attempt=3)  
2431 def calculate_area(a, h):  
2432     return 0.5 * a * h  
2433 -----  
-----  
2434 [HumanEval/45] Attempt 4/6  
2435 [HumanEval/45] ⚡ FAIL (attempt=4)  
2436 [TEST ERROR PREVIEW]  
2437 Traceback (most recent call last):  
2438     File "C:\Users\JOIN\Downloads\Multi-  
        Reviewer_Governance_for_Code_Generation\src\evaluator\  
        humaneval_evaluator.py", line 39, in _worker  
2439     ns["check"](candidate)  
2440     File "<string>", line 7, in check  
2441 AssertionError  
2442 -----  
-----  
2443 [HumanEval/45] [REPAIRED CODE] (after attempt=4)  
2444 def calculate_area(a, h):  
2445     return 0.5 * a * h  
2446 -----  
-----  
2447 [HumanEval/45] Attempt 5/6  
2448 [HumanEval/45] ⚡ FAIL (attempt=5)  
2449 [TEST ERROR PREVIEW]  
2450 Traceback (most recent call last):  
2451     File "C:\Users\JOIN\Downloads\Multi-  
        Reviewer_Governance_for_Code_Generation\src\evaluator\  
        humaneval_evaluator.py", line 39, in _worker
```

File - run

```
2452     ns["check"](candidate)
2453 File "<string>", line 7, in check
2454 AssertionError
2455 -----
2456 [HumanEval/45] [REPAIRED CODE] (after attempt=5)
2457 def calculate_area(a, h):
2458     return 0.5 * a * h
2459 -----
2460 [HumanEval/45] Attempt 6/6
2461 [HumanEval/45] ⚡ FAIL (attempt=6)
2462 [TEST ERROR PREVIEW]
2463 Traceback (most recent call last):
2464   File "C:\Users\JOIN\Downloads\Multi-
Reviewer_Governance_for_Code_Generation\src\evaluator\
humaneval_evaluator.py", line 39, in _worker
2465     ns["check"](candidate)
2466   File "<string>", line 7, in check
2467 AssertionError
2468 -----
2469 [HumanEval/45] ⚡ FINAL FAIL after 6 retries
2470 =====
=====
2471 [47/164] HumanEval/46
2472 =====
=====
2473 [TASK] HumanEval/46 | entry_point=fib4
2474 [PROMPT PREVIEW]
2475
2476
2477 def fib4(n: int):
2478     """The Fib4 number sequence is a sequence similar to the
Fibonacci sequence that's defined as follows:
2479     fib4(0) -> 0
2480     fib4(1) -> 0
2481     fib4(2) -> 2
2482     fib4(3) -> 0
2483     fib4(n) -> fib4(n-1) + fib4(n-2) + fib4(n-3) + fib4(n-4).
2484     Please write a function to efficiently compute the n-th element
of the fib4 number sequence. Do not use recursion.
2485     >>> fib4(5)
2486     4...
2487 -----
2488 [INITIAL GENERATED CODE]
2489 def fib4(n: int) -> int:
2490     if n == 0 or n == 1 or n == 3:
2491         return 0
2492     elif n == 2:
2493         return 2
2494
```

```
2495     a, b, c, d = 0, 0, 2, 0
2496     for _ in range(4, n + 1):
2497         a, b, c, d = b, c, d, a + b + c + d
2498     return d
2499 -----
2500 -----
2500 [HumanEval/46] Attempt 0/6
2501 [HumanEval/46] ✘ PASS (attempt=0)
2502 =====
2503 =====
2503 [48/164] HumanEval/47
2504 =====
2505 =====
2505 [TASK] HumanEval/47 | entry_point=median
2506 [PROMPT PREVIEW]
2507
2508
2509 def median(l: list):
2510     """Return median of elements in the list l.
2511     >>> median([3, 1, 2, 4, 5])
2512     3
2513     >>> median([-10, 4, 6, 1000, 10, 20])
2514     15.0
2515     """
2516 -----
2517 -----
2517 [INITIAL GENERATED CODE]
2518 def median(l: list):
2519     """Return median of elements in the list l.
2520     >>> median([3, 1, 2, 4, 5])
2521     3
2522     >>> median([-10, 4, 6, 1000, 10, 20])
2523     15.0
2524     """
2525     sorted_l = sorted(l)
2526     n = len(sorted_l)
2527     mid = n // 2
2528     if n % 2 == 0:
2529         return (sorted_l[mid - 1] + sorted_l[mid]) / 2
2530     else:
2531         return sorted_l[mid]
2532 -----
2533 -----
2533 [HumanEval/47] Attempt 0/6
2534 [HumanEval/47] ✘ PASS (attempt=0)
2535 =====
2536 =====
2536 [49/164] HumanEval/48
2537 =====
2538 =====
2538 [TASK] HumanEval/48 | entry_point=is_palindrome
2539 [PROMPT PREVIEW]
2540
```

```
2541
2542 def is_palindrome(text: str):
2543     """
2544         Checks if given string is a palindrome
2545         >>> is_palindrome('')
2546         True
2547         >>> is_palindrome('aba')
2548         True
2549         >>> is_palindrome('aaaaa')
2550         True
2551         >>> is_palindrome('zbcd')
2552         False
2553     """
2554 -----
2555 -----
2555 [INITIAL GENERATED CODE]
2556 def is_palindrome(text: str):
2557     return text == text[::-1]
2558 -----
2559 -----
2559 [HumanEval/48] Attempt 0/6
2560 [HumanEval/48] 0 PASS (attempt=0)
2561 =====
2562 -----
2562 [50/164] HumanEval/49
2563 =====
2563 -----
2564 [TASK] HumanEval/49 | entry_point=modp
2565 [PROMPT PREVIEW]
2566
2567
2568 def modp(n: int, p: int):
2569     """Return  $2^n \bmod p$  (be aware of numerics).
2570     >>> modp(3, 5)
2571     3
2572     >>> modp(1101, 101)
2573     2
2574     >>> modp(0, 101)
2575     1
2576     >>> modp(3, 11)
2577     8
2578     >>> modp(100, 101)
2579     1
2580     """
2581 -----
2581 -----
2582 [INITIAL GENERATED CODE]
2583 def modp(n: int, p: int) -> int:
2584     return pow(2, n, p)
2585 -----
2585 -----
2586 [HumanEval/49] Attempt 0/6
2587 [HumanEval/49] 0 PASS (attempt=0)
```

```
2588 =====
2589 =====
2589 [51/164] HumanEval/50
2590 =====
2591 =====
2591 [TASK] HumanEval/50 | entry_point=decode_shift
2592 [PROMPT PREVIEW]
2593
2594
2595 def encode_shift(s: str):
2596     """
2597         returns encoded string by shifting every character by 5 in the
2597         alphabet.
2598     """
2599     return "".join([chr(((ord(ch) + 5 - ord("a")) % 26) + ord("a")))
2599                     for ch in s])
2600
2601
2602 def decode_shift(s: str):
2603     """
2604         takes as input string encoded with encode_shift function.
2604         Returns decoded string.
2605     """
2606 -----
2607 -----
2607 [INITIAL GENERATED CODE]
2608 return "".join([chr(((ord(ch) - 5 - ord("a")) % 26) + ord("a")))
2608                 for ch in s])
2609 -----
2610 -----
2610 [HumanEval/50] Attempt 0/6
2611 [HumanEval/50] ✘ FAIL (attempt=0)
2612 [TEST ERROR PREVIEW]
2613 Traceback (most recent call last):
2614   File "C:\Users\JOIN\Downloads\Multi-
2614     Reviewer_Governance_for_Code_Generation\src\evaluator\
2614       humaneval_evaluator.py", line 27, in _worker
2615       exec(prompt + "\n" + completion + "\n", ns, ns)
2616   File "<string>", line 15
2617 SyntaxError: 'return' outside function
2618 -----
2619 -----
2619 [HumanEval/50] [REPAIRED CODE] (after attempt=0)
2620 def rotate_string(s):
2621     return "".join([chr(((ord(ch) - 5 - ord("a")) % 26) + ord("a")))
2621                     for ch in s])
2622 -----
2623 -----
2623 [HumanEval/50] Attempt 1/6
2624 [HumanEval/50] ✘ FAIL (attempt=1)
2625 [TEST ERROR PREVIEW]
2626 Traceback (most recent call last):
2627   File "C:\Users\JOIN\Downloads\Multi-
```

```
2627 Reviewer_Governance_for_Code_Generation\src\evaluator\  
    humaneval_evaluator.py", line 39, in _worker  
2628     ns["check"](candidate)  
2629     File "<string>", line 15, in check  
2630 AssertionError  
2631 -----  
-----  
2632 [HumanEval/50] [REPAIRED CODE] (after attempt=1)  
2633 def rotate_string(s):  
2634     return "".join([chr(((ord(ch) - 97 - 5) % 26) + 97) if 'a' <= ch  
        <= 'z' else ch for ch in s])  
2635 -----  
-----  
2636 [HumanEval/50] Attempt 2/6  
2637 [HumanEval/50] ✘ FAIL (attempt=2)  
2638 [TEST ERROR PREVIEW]  
2639 Traceback (most recent call last):  
2640     File "C:\Users\JOIN\Downloads\Multi-  
        Reviewer_Governance_for_Code_Generation\src\evaluator\  
        humaneval_evaluator.py", line 39, in _worker  
2641     ns["check"](candidate)  
2642     File "<string>", line 15, in check  
2643 AssertionError  
2644 -----  
-----  
2645 [HumanEval/50] [REPAIRED CODE] (after attempt=2)  
2646 def rotate_string(s):  
2647     return "".join([chr(((ord(ch) - 97 - 5) % 26) + 97) if 'a' <= ch  
        <= 'z' else ch for ch in s])  
2648 -----  
-----  
2649 [HumanEval/50] Attempt 3/6  
2650 [HumanEval/50] ✘ FAIL (attempt=3)  
2651 [TEST ERROR PREVIEW]  
2652 Traceback (most recent call last):  
2653     File "C:\Users\JOIN\Downloads\Multi-  
        Reviewer_Governance_for_Code_Generation\src\evaluator\  
        humaneval_evaluator.py", line 39, in _worker  
2654     ns["check"](candidate)  
2655     File "<string>", line 15, in check  
2656 AssertionError  
2657 -----  
-----  
2658 [HumanEval/50] [REPAIRED CODE] (after attempt=3)  
2659 def rotate_string(s):  
2660     return "".join([chr(((ord(ch) - 97 - 5) % 26) + 97) if 'a' <= ch  
        <= 'z' else ch for ch in s])  
2661 -----  
-----  
2662 [HumanEval/50] Attempt 4/6  
2663 [HumanEval/50] ✘ FAIL (attempt=4)  
2664 [TEST ERROR PREVIEW]  
2665 Traceback (most recent call last):
```

```
2666 File "C:\Users\JOIN\Downloads\Multi-
    Reviewer_Governance_for_Code_Generation\src\evaluator\
        humaneval_evaluator.py", line 39, in _worker
2667     ns["check"](candidate)
2668     File "<string>", line 15, in check
2669 AssertionError
2670 -----
-----
2671 [HumanEval/50] [REPAIRED CODE] (after attempt=4)
2672 def rotate_string(s):
2673     return "".join([chr(((ord(ch) - 97 - 5) % 26) + 97) if 'a' <= ch
2674                     <= 'z' else ch for ch in s])
2675 -----
-----
2676 [HumanEval/50] Attempt 5/6
2677 [HumanEval/50] ✘ FAIL (attempt=5)
2678 [TEST ERROR PREVIEW]
2679 Traceback (most recent call last):
2680     File "C:\Users\JOIN\Downloads\Multi-
    Reviewer_Governance_for_Code_Generation\src\evaluator\
        humaneval_evaluator.py", line 39, in _worker
2681     ns["check"](candidate)
2682     File "<string>", line 15, in check
2683 AssertionError
2684 -----
-----
2685 [HumanEval/50] [REPAIRED CODE] (after attempt=5)
2686 def rotate_string(s):
2687     return "".join([chr(((ord(ch) - 97 - 5) % 26) + 97) if 'a' <= ch
2688                     <= 'z' else ch for ch in s])
2689 -----
-----
2690 [HumanEval/50] Attempt 6/6
2691 [HumanEval/50] ✘ FAIL (attempt=6)
2692 [TEST ERROR PREVIEW]
2693 Traceback (most recent call last):
2694     File "C:\Users\JOIN\Downloads\Multi-
    Reviewer_Governance_for_Code_Generation\src\evaluator\
        humaneval_evaluator.py", line 39, in _worker
2695     ns["check"](candidate)
2696     File "<string>", line 15, in check
2697 AssertionError
2698 -----
-----
2699 [HumanEval/50] ✘ FINAL FAIL after 6 retries
2700 =====
=====
2701 [52/164] HumanEval/51
2702 =====
=====
2703 [TASK] HumanEval/51 | entry_point=remove_vowels
2704 [PROMPT PREVIEW]
2705
```

```
2704
2705 def remove_vowels(text):
2706     """
2707         remove_vowels is a function that takes string and returns string
2708         without vowels.
2709         >>> remove_vowels('')
2710         ''
2711         >>> remove_vowels("abcdef\nghijklm")
2712         'bcdf\nghijklm'
2713         >>> remove_vowels('abcdef')
2714         'bcdf'
2715         >>> remove_vowels('aaaaaa')
2716         ''
2717         >>> remove_vowels('aaBAA')
2718         'B'
2719         >>> remove_vowels('zbcd')
2720         'zbcd'
2721         """
2722 -----
2723 [INITIAL GENERATED CODE]
2724 def remove_vowels(text):
2725     vowels = "aeiouAEIOU"
2726     return ''.join([char for char in text if char not in vowels])
2727 -----
2728 [HumanEval/51] Attempt 0/6
2729 [HumanEval/51] 0 PASS (attempt=0)
2730 =====
2731 =====
2732 [53/164] HumanEval/52
2733 =====
2734 [TASK] HumanEval/52 | entry_point=below_threshold
2735 [PROMPT PREVIEW]
2736
2737 def below_threshold(l: list, t: int):
2738     """Return True if all numbers in the list l are below threshold
2739     t.
2740     >>> below_threshold([1, 2, 4, 10], 100)
2741     True
2742     >>> below_threshold([1, 20, 4, 10], 5)
2743     False
2744     """
2745 -----
2746 [INITIAL GENERATED CODE]
2747 def below_threshold(l: list, t: int) -> bool:
2748     return all(x < t for x in l)
2749 -----
2750 [HumanEval/52] Attempt 0/6
```

```
2749 [HumanEval/52] ✘ PASS (attempt=0)
2750 =====
2751 =====
2751 [54/164] HumanEval/53
2752 =====
2752 =====
2753 [TASK] HumanEval/53 | entry_point=add
2754 [PROMPT PREVIEW]
2755
2756
2757 def add(x: int, y: int):
2758     """Add two numbers x and y
2759     >>> add(2, 3)
2760     5
2761     >>> add(5, 7)
2762     12
2763     """
2764 -----
2764 -----
2765 [INITIAL GENERATED CODE]
2766 def add(x: int, y: int) -> int:
2767     return x + y
2768 -----
2768 -----
2769 [HumanEval/53] Attempt 0/6
2770 [HumanEval/53] ✘ PASS (attempt=0)
2771 =====
2771 =====
2772 [55/164] HumanEval/54
2773 =====
2773 =====
2774 [TASK] HumanEval/54 | entry_point=same_chars
2775 [PROMPT PREVIEW]
2776
2777
2778 def same_chars(s0: str, s1: str):
2779     """
2780     Check if two words have the same characters.
2781     >>> same_chars('eabcdzzz', 'dddzzzzzzddedab')
2782     True
2783     >>> same_chars('abcd', 'dddddabc')
2784     True
2785     >>> same_chars('dddddabc', 'abcd')
2786     True
2787     >>> same_chars('eabcd', 'dddddabc')
2788     False
2789     >>> same_chars('abcd', 'dddddabce')
2790     False
2791     >>> same_chars('eabcdzzz', 'ddz...'))
2792 -----
2792 -----
2793 [INITIAL GENERATED CODE]
2794 def same_chars(s0: str, s1: str) -> bool:
```

```
2795     return set(s0) == set(s1)
2796 -----
2797 -----
2797 [HumanEval/54] Attempt 0/6
2798 [HumanEval/54] ✘ PASS (attempt=0)
2799 =====
2800 =====
2800 [56/164] HumanEval/55
2801 =====
2801 =====
2802 [TASK] HumanEval/55 | entry_point=fib
2803 [PROMPT PREVIEW]
2804
2805
2806 def fib(n: int):
2807     """Return n-th Fibonacci number.
2808     >>> fib(10)
2809     55
2810     >>> fib(1)
2811     1
2812     >>> fib(8)
2813     21
2814     """
2815 -----
2815 -----
2816 [INITIAL GENERATED CODE]
2817 def fib(n: int) -> int:
2818     if n <= 0:
2819         return 0
2820     elif n == 1:
2821         return 1
2822     else:
2823         a, b = 0, 1
2824         for _ in range(2, n + 1):
2825             a, b = b, a + b
2826         return b
2827 -----
2827 -----
2828 [HumanEval/55] Attempt 0/6
2829 [HumanEval/55] ✘ PASS (attempt=0)
2830 =====
2830 -----
2831 [57/164] HumanEval/56
2832 =====
2832 -----
2833 [TASK] HumanEval/56 | entry_point=correct_bracketing
2834 [PROMPT PREVIEW]
2835
2836
2837 def correct_bracketing(brackets: str):
2838     """ brackets is a string of "<" and ">".
2839     return True if every opening bracket has a corresponding closing
bracket.
```

```
2840
2841     >>> correctBracketing("<")
2842     False
2843     >>> correctBracketing("<>")
2844     True
2845     >>> correctBracketing("<<><>>")
2846     True
2847     >>> correctBracketing("><<>")
2848     False
2849     """
2850 -----
2851 -----
2851 [INITIAL GENERATED CODE]
2852 def correctBracketing(brackets: str) -> bool:
2853     balance = 0
2854     for bracket in brackets:
2855         if bracket == '<':
2856             balance += 1
2857         elif bracket == '>':
2858             balance -= 1
2859         if balance < 0:
2860             return False
2861     return balance == 0
2862 -----
2863 -----
2863 [HumanEval/56] Attempt 0/6
2864 [HumanEval/56] 0 PASS (attempt=0)
2865 =====
2866 =====
2866 [58/164] HumanEval/57
2867 =====
2867 =====
2868 [TASK] HumanEval/57 | entry_point=monotonic
2869 [PROMPT PREVIEW]
2870
2871
2872 def monotonic(l: list):
2873     """Return True if list elements are monotonically increasing or
2874     decreasing.
2874     >>> monotonic([1, 2, 4, 20])
2875     True
2876     >>> monotonic([1, 20, 4, 10])
2877     False
2878     >>> monotonic([4, 1, 0, -10])
2879     True
2880     """
2881 -----
2881 -----
2882 [INITIAL GENERATED CODE]
2883 def monotonic(l: list):
2884     return all(x <= y for x, y in zip(l, l[1:])) or all(x >= y for x
2885     , y in zip(l, l[1:]))
2885 -----
```

```
2885 -----
2886 [HumanEval/57] Attempt 0/6
2887 [HumanEval/57] ✘ PASS (attempt=0)
2888 =====
2889 =====
2890 [59/164] HumanEval/58
2891 =====
2892 =====
2893
2894
2895 def common(l1: list, l2: list):
2896     """Return sorted unique common elements for two lists.
2897     >>> common([1, 4, 3, 34, 653, 2, 5], [5, 7, 1, 5, 9, 653, 121])
2898     [1, 5, 653]
2899     >>> common([5, 3, 2, 8], [3, 2])
2900     [2, 3]
2901
2902     """
2903 -----
2904 -----
2904 [INITIAL GENERATED CODE]
2905 def common(l1: list, l2: list):
2906     """Return sorted unique common elements for two lists.
2907     >>> common([1, 4, 3, 34, 653, 2, 5], [5, 7, 1, 5, 9, 653, 121])
2908     [1, 5, 653]
2909     >>> common([5, 3, 2, 8], [3, 2])
2910     [2, 3]
2911
2912     """
2913     return sorted(set(l1) & set(l2))
2914 -----
2915 -----
2915 [HumanEval/58] Attempt 0/6
2916 [HumanEval/58] ✘ PASS (attempt=0)
2917 =====
2918 =====
2918 [60/164] HumanEval/59
2919 =====
2920 =====
2920 [TASK] HumanEval/59 | entry_point=largest_prime_factor
2921 [PROMPT PREVIEW]
2922
2923
2924 def largest_prime_factor(n: int):
2925     """Return the largest prime factor of n. Assume n > 1 and is not
2926     a prime.
2927     >>> largest_prime_factor(13195)
2928     29
2929     >>> largest_prime_factor(2048)
2930     2
2931     """
```

```
2931 -----
-----
2932 [INITIAL GENERATED CODE]
2933 def largest_prime_factor(n: int) -> int:
2934     factor = 2
2935     while n % factor == 0:
2936         n //= factor
2937     for i in range(3, int(n**0.5) + 1, 2):
2938         while n % i == 0:
2939             n //= i
2940             factor = i
2941     if n > 2:
2942         factor = n
2943     return factor
2944 -----
-----
2945 [HumanEval/59] Attempt 0/6
2946 [HumanEval/59] ✘ PASS (attempt=0)
2947 =====
=====
2948 [61/164] HumanEval/60
2949 =====
=====
2950 [TASK] HumanEval/60 | entry_point=sum_to_n
2951 [PROMPT PREVIEW]
2952
2953
2954 def sum_to_n(n: int):
2955     """sum_to_n is a function that sums numbers from 1 to n.
2956     >>> sum_to_n(30)
2957     465
2958     >>> sum_to_n(100)
2959     5050
2960     >>> sum_to_n(5)
2961     15
2962     >>> sum_to_n(10)
2963     55
2964     >>> sum_to_n(1)
2965     1
2966     """
2967 -----
-----
2968 [INITIAL GENERATED CODE]
2969 def sum_to_n(n: int) -> int:
2970     return n * (n + 1) // 2
2971 -----
-----
2972 [HumanEval/60] Attempt 0/6
2973 [HumanEval/60] ✘ PASS (attempt=0)
2974 =====
=====
2975 [62/164] HumanEval/61
2976 =====
```

```
2976 =====
2977 [TASK] HumanEval/61 | entry_point=correctBracketing
2978 [PROMPT PREVIEW]
2979
2980
2981 def correctBracketing(brackets: str):
2982     """ brackets is a string of "(" and ")".
2983     return True if every opening bracket has a corresponding closing
2984     bracket.
2985
2986     >>> correctBracketing("(")
2987     False
2988     >>> correctBracketing("()")
2989     True
2990     >>> correctBracketing("((())")
2991     True
2992     >>> correctBracketing(")()()")
2993     False
2994     """
2995 -----
2996 -----
2997 [INITIAL GENERATED CODE]
2998 def correctBracketing(brackets: str) -> bool:
2999     balance = 0
3000     for bracket in brackets:
3001         if bracket == '(':
3002             balance += 1
3003         elif bracket == ')':
3004             balance -= 1
3005         if balance < 0:
3006             return False
3007     return balance == 0
3008 -----
3009 =====
3010 =====
3011 [HumanEval/61] Attempt 0/6
3012 [HumanEval/61] 0 PASS (attempt=0)
3013 =====
3014 =====
3015
3016 def derivative(xs: list):
3017     """ xs represent coefficients of a polynomial.
3018     xs[0] + xs[1] * x + xs[2] * x^2 + ....
3019     Return derivative of this polynomial in the same form.
3020     >>> derivative([3, 1, 2, 4, 5])
3021     [1, 4, 12, 20]
3022     >>> derivative([1, 2, 3])
3023     [2, 6]
```

```
3024     """
3025 -----
3026 -----
3026 [INITIAL GENERATED CODE]
3027 def derivative(xs: list):
3028     return [i * x for i, x in enumerate(xs)][1:]
3029 -----
3030 -----
3030 [HumanEval/62] Attempt 0/6
3031 [HumanEval/62] ✅ PASS (attempt=0)
3032 =====
3032 =====
3033 [64/164] HumanEval/63
3034 =====
3034 =====
3035 [TASK] HumanEval/63 | entry_point=fibfib
3036 [PROMPT PREVIEW]
3037
3038
3039 def fibfib(n: int):
3040     """The FibFib number sequence is a sequence similar to the
3040     Fibonacci sequence that's defined as follows:
3041     fibfib(0) == 0
3042     fibfib(1) == 0
3043     fibfib(2) == 1
3044     fibfib(n) == fibfib(n-1) + fibfib(n-2) + fibfib(n-3).
3045     Please write a function to efficiently compute the n-th element
3045     of the fibfib number sequence.
3046     >>> fibfib(1)
3047     0
3048     >>> fibfib(5)
3049     4
3050     >>...
3051 -----
3051 -----
3052 [INITIAL GENERATED CODE]
3053 def fibfib(n: int) -> int:
3054     if n == 0 or n == 1:
3055         return 0
3056     elif n == 2:
3057         return 1
3058
3059     a, b, c = 0, 0, 1
3060     for _ in range(3, n + 1):
3061         a, b, c = b, c, a + b + c
3062     return c
3063 -----
3063 -----
3064 [HumanEval/63] Attempt 0/6
3065 [HumanEval/63] ✅ PASS (attempt=0)
3066 =====
3066 =====
3067 [65/164] HumanEval/64
```

```

3068 =====
=====
3069 [TASK] HumanEval/64 | entry_point=vowels_count
3070 [PROMPT PREVIEW]
3071
3072 FIX = """
3073 Add more test cases.
3074 """
3075
3076 def vowels_count(s):
3077     """Write a function vowels_count which takes a string
       representing
3078     a word as input and returns the number of vowels in the string.
3079     Vowels in this case are 'a', 'e', 'i', 'o', 'u'. Here, 'y' is
       also a
3080     vowel, but only when it is at the end of the given word.
3081
3082     Example:
3083     >>> vowels_count("abcde")
3084     2
3085     >>> vowels_coun...
3086 -----
-----
3087 [INITIAL GENERATED CODE]
3088 def vowels_count(s):
3089     """Write a function vowels_count which takes a string
       representing
3090     a word as input and returns the number of vowels in the string.
3091     Vowels in this case are 'a', 'e', 'i', 'o', 'u'. Here, 'y' is
       also a
3092     vowel, but only when it is at the end of the given word.
3093
3094     Example:
3095     >>> vowels_count("abcde")
3096     2
3097     >>> vowels_count("ACEDY")
3098     3
3099     """
3100     vowels = "aeiouAEIOU"
3101     count = sum(1 for char in s if char in vowels)
3102     if s.endswith('y') or s.endswith('Y'):
3103         count += 1
3104     return count
3105 -----
-----
3106 [HumanEval/64] Attempt 0/6
3107 [HumanEval/64] 0 PASS (attempt=0)
3108 =====
=====
3109 [66/164] HumanEval/65
3110 =====
=====
3111 [TASK] HumanEval/65 | entry_point=circular_shift

```

```
3112 [PROMPT PREVIEW]
3113
3114 def circular_shift(x, shift):
3115     """Circular shift the digits of the integer x, shift the digits
3116     right by shift
3117     and return the result as a string.
3118     If shift > number of digits, return digits reversed.
3119     >>> circular_shift(12, 1)
3120     "21"
3121     >>> circular_shift(12, 2)
3122     "12"
3123     """
3124 -----
3125 [INITIAL GENERATED CODE]
3126 def circular_shift(x, shift):
3127     x_str = str(x)
3128     n = len(x_str)
3129     if shift >= n:
3130         return x_str[::-1]
3131     else:
3132         shift %= n
3133     return x_str[-shift:] + x_str[:-shift]
3134 -----
3135 [HumanEval/65] Attempt 0/6
3136 [HumanEval/65] ✘ FAIL (attempt=0)
3137 [TEST ERROR PREVIEW]
3138 Traceback (most recent call last):
3139   File "C:\Users\JOIN\Downloads\Multi-
3140     Reviewer_Governance_for_Code_Generation\src\evaluator\
3141       humaneval_evaluator.py", line 39, in _worker
3142       ns["check"](candidate)
3143     File "<string>", line 5, in check
3144     AssertionError
3145 -----
3146 [HumanEval/65] [REPAIRED CODE] (after attempt=0)
3147 def circular_shift(x, shift):
3148     x_str = str(x)
3149     n = len(x_str)
3150     if shift >= n:
3151         return x_str[::-1]
3152     else:
3153         shift %= n
3154     return x_str[-shift:] + x_str[:-shift]
3155 -----
3156 [HumanEval/65] Attempt 1/6
3157 [HumanEval/65] ✘ FAIL (attempt=1)
3158 [TEST ERROR PREVIEW]
3159 Traceback (most recent call last):
3160   File "C:\Users\JOIN\Downloads\Multi-
```

```
3157 Reviewer_Governance_for_Code_Generation\src\evaluator\  
    humaneval_evaluator.py", line 39, in _worker  
3158     ns["check"](candidate)  
3159     File "<string>", line 5, in check  
3160 AssertionError  
3161 -----  
-----  
3162 [HumanEval/65] [REPAIRED CODE] (after attempt=1)  
3163 def circular_shift(x, shift):  
3164     x_str = str(x)  
3165     n = len(x_str)  
3166     if shift >= n:  
3167         return x_str[::-1]  
3168     else:  
3169         shift %= n  
3170         return x_str[-shift:] + x_str[:-shift]  
3171 -----  
-----  
3172 [HumanEval/65] Attempt 2/6  
3173 [HumanEval/65] ⚡ FAIL (attempt=2)  
3174 [TEST ERROR PREVIEW]  
3175 Traceback (most recent call last):  
3176     File "C:\Users\JOIN\Downloads\Multi-  
        Reviewer_Governance_for_Code_Generation\src\evaluator\  
            humaneval_evaluator.py", line 39, in _worker  
3177     ns["check"](candidate)  
3178     File "<string>", line 5, in check  
3179 AssertionError  
3180 -----  
-----  
3181 [HumanEval/65] [REPAIRED CODE] (after attempt=2)  
3182 def circular_shift(x, shift):  
3183     x_str = str(x)  
3184     n = len(x_str)  
3185     if shift >= n:  
3186         return x_str[::-1]  
3187     else:  
3188         shift %= n  
3189         return x_str[-shift:] + x_str[:-shift]  
3190 -----  
-----  
3191 [HumanEval/65] Attempt 3/6  
3192 [HumanEval/65] ⚡ FAIL (attempt=3)  
3193 [TEST ERROR PREVIEW]  
3194 Traceback (most recent call last):  
3195     File "C:\Users\JOIN\Downloads\Multi-  
        Reviewer_Governance_for_Code_Generation\src\evaluator\  
            humaneval_evaluator.py", line 39, in _worker  
3196     ns["check"](candidate)  
3197     File "<string>", line 5, in check  
3198 AssertionError  
3199 -----  
-----
```

```
3200 [HumanEval/65] [REPAIRED CODE] (after attempt=3)
3201 def circular_shift(x, shift):
3202     x_str = str(x)
3203     n = len(x_str)
3204     if shift >= n:
3205         return x_str[::-1]
3206     else:
3207         shift %= n
3208         return x_str[-shift:] + x_str[:-shift]
3209 -----
-----
3210 [HumanEval/65] Attempt 4/6
3211 [HumanEval/65] ⚡ FAIL (attempt=4)
3212 [TEST ERROR PREVIEW]
3213 Traceback (most recent call last):
3214   File "C:\Users\JOIN\Downloads\Multi-
    Reviewer_Governance_for_Code_Generation\src\evaluator\
      humaneval_evaluator.py", line 39, in _worker
3215     ns["check"](candidate)
3216   File "<string>", line 5, in check
3217 AssertionError
3218 -----
-----
3219 [HumanEval/65] [REPAIRED CODE] (after attempt=4)
3220 def circular_shift(x, shift):
3221     x_str = str(x)
3222     n = len(x_str)
3223     if shift >= n:
3224         return x_str[::-1]
3225     else:
3226         shift %= n
3227         return x_str[-shift:] + x_str[:-shift]
3228 -----
-----
3229 [HumanEval/65] Attempt 5/6
3230 [HumanEval/65] ⚡ FAIL (attempt=5)
3231 [TEST ERROR PREVIEW]
3232 Traceback (most recent call last):
3233   File "C:\Users\JOIN\Downloads\Multi-
    Reviewer_Governance_for_Code_Generation\src\evaluator\
      humaneval_evaluator.py", line 39, in _worker
3234     ns["check"](candidate)
3235   File "<string>", line 5, in check
3236 AssertionError
3237 -----
-----
3238 [HumanEval/65] [REPAIRED CODE] (after attempt=5)
3239 def circular_shift(x, shift):
3240     x_str = str(x)
3241     n = len(x_str)
3242     if shift >= n:
3243         return x_str[::-1]
3244     else:
```

```
3245     shift %= n
3246     return x_str[-shift:] + x_str[:-shift]
3247 -----
3248 [HumanEval/65] Attempt 6/6
3249 [HumanEval/65] ⚡ FAIL (attempt=6)
3250 [TEST ERROR PREVIEW]
3251 Traceback (most recent call last):
3252   File "C:\Users\JOIN\Downloads\Multi-
    Reviewer_Governance_for_Code_Generation\src\evaluator\
      humaneval_evaluator.py", line 39, in _worker
3253     ns["check"](candidate)
3254   File "<string>", line 5, in check
3255 AssertionError
3256 -----
3257 [HumanEval/65] ⚡ FINAL FAIL after 6 retries
3258 =====
3259 [67/164] HumanEval/66
3260 =====
3261 [TASK] HumanEval/66 | entry_point=digitSum
3262 [PROMPT PREVIEW]
3263
3264 def digitSum(s):
3265     """Task
3266         Write a function that takes a string as input and returns the
3267         sum of the upper characters only'
3268         ASCII codes.
3269
3270     Examples:
3271         digitSum("") => 0
3272         digitSum("abAB") => 131
3273         digitSum("abcCd") => 67
3274         digitSum("helloE") => 69
3275         digitSum("woArBld") => 131
3276         digitSum("aAaaaXa") => 153
3277     """
3278 [INITIAL GENERATED CODE]
3279 def digitSum(s):
3280     return sum(ord(c) for c in s if c.isupper())
3281 -----
3282 [HumanEval/66] Attempt 0/6
3283 [HumanEval/66] ⚡ PASS (attempt=0)
3284 =====
3285 [68/164] HumanEval/67
3286 =====
```

```
3287 [TASK] HumanEval/67 | entry_point=fruit_distribution
3288 [PROMPT PREVIEW]
3289
3290 def fruit_distribution(s,n):
3291     """
3292         In this task, you will be given a string that represents a
3293             number of apples and oranges
3294                 that are distributed in a basket of fruit this basket contains
3295                     apples, oranges, and mango fruits. Given the string that
3296                         represents the total number of
3297                             the oranges and apples and an integer that represent the total
3298                                 number of the fruits
3299                                 in the basket r...
3300 -----
3301 -----
3302 [INITIAL GENERATED CODE]
3303 def fruit_distribution(s, n):
3304     parts = s.split()
3305     apples = int(parts[0])
3306     oranges = int(parts[3])
3307     return n - apples - oranges
3308 -----
3309 -----
3310 [HumanEval/67] Attempt 0/6
3311 [HumanEval/67] 0 PASS (attempt=0)
3312 =====
3313 =====
3314 [69/164] HumanEval/68
3315 =====
3316 =====
3317 [TASK] HumanEval/68 | entry_point=pluck
3318 [PROMPT PREVIEW]
3319
3320 def pluck(arr):
3321     """
3322         "Given an array representing a branch of a tree that has non-
3323             negative integer nodes
3324                 your task is to pluck one of the nodes and return it.
3325                     The plucked node should be the node with the smallest even value
3326 .
3327             If multiple nodes with the same smallest even value are found
3328                 return the node that has smallest index.
3329
3330         The plucked node should be returned in a list,....
3331 -----
3332 -----
3333 [INITIAL GENERATED CODE]
3334 def pluck(arr):
3335     if not arr:
3336         return []
3337
3338     min_val = float('inf')
3339     min_index = -1
```

```
3329
3330     for i, val in enumerate(arr):
3331         if val % 2 == 0 and val < min_val:
3332             min_val = val
3333             min_index = i
3334
3335     if min_index == -1:
3336         return []
3337
3338     return [min_val, min_index]
3339 -----
-----
3340 [HumanEval/68] Attempt 0/6
3341 [HumanEval/68] ✘ PASS (attempt=0)
3342 =====
=====
3343 [70/164] HumanEval/69
3344 =====
=====
3345 [TASK] HumanEval/69 | entry_point=search
3346 [PROMPT PREVIEW]
3347
3348 def search(lst):
3349     '''
3350         You are given a non-empty list of positive integers. Return the
3351         greatest integer that is greater than
3352         zero, and has a frequency greater than or equal to the value of
3353         the integer itself.
3354         The frequency of an integer is the number of times it appears in
3355         the list.
3356         If no such a value exist, return -1.
3357         Examples:
3358             search([4, 1, 2, 2, 3, 1]) == 2...
3359 -----
-----
3360 [INITIAL GENERATED CODE]
3361 def search(lst):
3362     from collections import Counter
3363
3364     freq = Counter(lst)
3365     for num in sorted(freq.keys(), reverse=True):
3366         if freq[num] >= num:
3367             return num
3368     return -1
3369 -----
-----
3370 [HumanEval/69] Attempt 0/6
3371 [HumanEval/69] ✘ PASS (attempt=0)
3372 =====
=====
3373 [71/164] HumanEval/70
3374 =====
=====
```

```
3372 [TASK] HumanEval/70 | entry_point=strange_sort_list
3373 [PROMPT PREVIEW]
3374
3375 def strange_sort_list(lst):
3376     """
3377         Given list of integers, return list in strange order.
3378         Strange sorting, is when you start with the minimum value,
3379         then maximum of the remaining integers, then minimum and so on.
3380
3381     Examples:
3382     strange_sort_list([1, 2, 3, 4]) == [1, 4, 2, 3]
3383     strange_sort_list([5, 5, 5, 5]) == [5, 5, 5, 5]
3384     strange_sort_list([]) == []
3385     """
3386 -----
3387 -----
3387 [INITIAL GENERATED CODE]
3388 def strange_sort_list(lst):
3389     sorted_lst = sorted(lst)
3390     result = []
3391     while sorted_lst:
3392         if sorted_lst:
3393             result.append(sorted_lst.pop(0))
3394         if sorted_lst:
3395             result.append(sorted_lst.pop())
3396     return result
3397 -----
3398 -----
3398 [HumanEval/70] Attempt 0/6
3399 [HumanEval/70] 0 PASS (attempt=0)
3400 =====
3401 =====
3401 [72/164] HumanEval/71
3402 =====
3402 =====
3403 [TASK] HumanEval/71 | entry_point=triangle_area
3404 [PROMPT PREVIEW]
3405
3406 def triangle_area(a, b, c):
3407     """
3408         Given the lengths of the three sides of a triangle. Return the
3409         area of
3410         the triangle rounded to 2 decimal points if the three sides form
3411         a valid triangle.
3412         Otherwise return -1
3413         Three sides make a valid triangle when the sum of any two sides
3414         is greater
3415         than the third side.
3416     Example:
3417     triangle_area(3, 4, 5) == 6.00
3418     triangle_area(1, 2, ...
3419 -----
3419 -----
```

```
3417 [INITIAL GENERATED CODE]
3418 import math
3419
3420 def triangle_area(a, b, c):
3421     if a + b > c and a + c > b and b + c > a:
3422         s = (a + b + c) / 2
3423         area = math.sqrt(s * (s - a) * (s - b) * (s - c))
3424         return round(area, 2)
3425     else:
3426         return -1
3427 -----
3428 -----
3429 [HumanEval/71] Attempt 0/6
3429 [HumanEval/71] 0 PASS (attempt=0)
3430 =====
3431 =====
3431 [73/164] HumanEval/72
3432 =====
3432 =====
3433 [TASK] HumanEval/72 | entry_point=will_it_fly
3434 [PROMPT PREVIEW]
3435
3436 def will_it_fly(q,w):
3437     '''
3438     Write a function that returns True if the object q will fly, and
3438     False otherwise.
3439     The object q will fly if it's balanced (it is a palindromic list
3439     ) and the sum of its elements is less than or equal the maximum
3439     possible weight w.
3440
3441     Example:
3442     will_it_fly([1, 2], 5) → False
3443     # 1+2 is less than the maximum possible weight, but it's
3443     unbalanced.
3444
3445     will...
3446 -----
3446 -----
3447 [INITIAL GENERATED CODE]
3448 def will_it_fly(q, w):
3449     return q == q[::-1] and sum(q) <= w
3450 -----
3450 -----
3451 [HumanEval/72] Attempt 0/6
3452 [HumanEval/72] 0 PASS (attempt=0)
3453 =====
3453 =====
3454 [74/164] HumanEval/73
3455 =====
3455 =====
3456 [TASK] HumanEval/73 | entry_point=smallest_change
3457 [PROMPT PREVIEW]
3458
```

```
3459 def smallest_change(arr):
3460     """
3461         Given an array arr of integers, find the minimum number of
3462         elements that
3463             need to be changed to make the array palindromic. A palindromic
3464             array is an array that
3465                 is read the same backwards and forwards. In one change, you can
3466                 change one element to any other element.
3467
3468 -----
3469 -----
3470 [INITIAL GENERATED CODE]
3471 def smallest_change(arr):
3472     left = 0
3473     right = len(arr) - 1
3474     changes = 0
3475
3476     while left < right:
3477         if arr[left] != arr[right]:
3478             changes += 1
3479         left += 1
3480         right -= 1
3481
3482 -----
3483 -----
3484 [HumanEval/73] Attempt 0/6
3485 [HumanEval/73] 0 PASS (attempt=0)
3486 =====
3487 =====
3488 [75/164] HumanEval/74
3489 =====
3490 =====
3491 [TASK] HumanEval/74 | entry_point=total_match
3492 [PROMPT PREVIEW]
3493
3494 def total_match(lst1, lst2):
3495     '''
3496         Write a function that accepts two lists of strings and returns
3497         the list that has
3498             total number of chars in the all strings of the list less than
3499             the other list.
3500
3501         if the two lists have the same number of chars, return the first
3502             list.
3503
3504         Examples
3505         total_match([], []) → []
3506         total_match(['hi', 'admin'], ['hI', 'Hi']) → ['hI', 'Hi']
3507         total_ma...
```

```
3502 -----
-----
3503 [INITIAL GENERATED CODE]
3504 def total_match(lst1, lst2):
3505     len_lst1 = sum(len(s) for s in lst1)
3506     len_lst2 = sum(len(s) for s in lst2)
3507     return lst1 if len_lst1 <= len_lst2 else lst2
3508 -----
-----
3509 [HumanEval/74] Attempt 0/6
3510 [HumanEval/74] 0 PASS (attempt=0)
3511 =====
=====
3512 [76/164] HumanEval/75
3513 =====
=====
3514 [TASK] HumanEval/75 | entry_point=is_multiply_prime
3515 [PROMPT PREVIEW]
3516
3517 def is_multiply_prime(a):
3518     """Write a function that returns true if the given number is the
3519     multiplication of 3 prime numbers
3520     and false otherwise.
3521     Knowing that (a) is less then 100.
3522     Example:
3523     is_multiply_prime(30) == True
3524     30 = 2 * 3 * 5
3525     """
3526 -----
-----
3526 [INITIAL GENERATED CODE]
3527 def is_multiply_prime(a):
3528     def is_prime(n):
3529         if n <= 1:
3530             return False
3531         for i in range(2, int(n**0.5) + 1):
3532             if n % i == 0:
3533                 return False
3534         return True
3535
3536     primes = [i for i in range(2, 100) if is_prime(i)]
3537
3538     for i in range(len(primes)):
3539         for j in range(i, len(primes)):
3540             for k in range(j, len(primes)):
3541                 if primes[i] * primes[j] * primes[k] == a:
3542                     return True
3543     return False
3544 -----
-----
3545 [HumanEval/75] Attempt 0/6
3546 [HumanEval/75] 0 PASS (attempt=0)
3547 =====
```

```
3547 =====
3548 [77/164] HumanEval/76
3549 =====
3550 =====
3551 [TASK] HumanEval/76 | entry_point=is_simple_power
3552 [PROMPT PREVIEW]
3553
3554 def is_simple_power(x, n):
3555     """Your task is to write a function that returns true if a
3556     number x is a simple
3557     power of n and false in other cases.
3558     x is a simple power of n if n**int=x
3559     For example:
3560     is_simple_power(1, 4) => true
3561     is_simple_power(2, 2) => true
3562     is_simple_power(8, 2) => true
3563     is_simple_power(3, 2) => false
3564     is_simple_power(3, 1) => false
3565     is_simple_power...
3566 -----
3567 -----
3568 [INITIAL GENERATED CODE]
3569 def is_simple_power(x, n):
3570     if n == 1:
3571         return x == 1
3572     power = 1
3573     while power < x:
3574         power *= n
3575     return power == x
3576 -----
3577 -----
3578 [HumanEval/76] Attempt 0/6
3579 [HumanEval/76] 0 PASS (attempt=0)
3580 =====
3581 =====
3582 [78/164] HumanEval/77
3583 =====
3584 =====
3585 [TASK] HumanEval/77 | entry_point=iscube
3586 [PROMPT PREVIEW]
3587
3588 def iscube(a):
3589     """
3590     Write a function that takes an integer a and returns True
3591     if this integer is a cube of some integer number.
3592     Note: you may assume the input is always valid.
3593     Examples:
3594     iscube(1) ==> True
3595     iscube(2) ==> False
3596     iscube(-1) ==> True
3597     iscube(64) ==> True
3598     iscube(0) ==> True
3599     iscube(180) ==> False
```

```
3594   ''
3595 -----
3596 [INITIAL GENERATED CODE]
3597 def iscude(a):
3598     return round(a ** (1/3)) ** 3 == a
3599 -----
3600 [HumanEval/77] Attempt 0/6
3601 [HumanEval/77] ⚡ FAIL (attempt=0)
3602 [TEST ERROR PREVIEW]
3603 Traceback (most recent call last):
3604   File "C:\Users\JOIN\Downloads\Multi-
Reviewer_Governance_for_Code_Generation\src\evaluator\
humaneval_evaluator.py", line 39, in _worker
3605     ns["check"](candidate)
3606   File "<string>", line 6, in check
3607   File "<string>", line 17, in iscude
3608 TypeError: type complex doesn't define __round__ method
3609 -----
3610 [HumanEval/77] [REPAIRED CODE] (after attempt=0)
3611 def iscude(a):
3612     if isinstance(a, complex):
3613         return False
3614     return round(a ** (1/3)) ** 3 == a
3615 -----
3616 [HumanEval/77] Attempt 1/6
3617 [HumanEval/77] ⚡ FAIL (attempt=1)
3618 [TEST ERROR PREVIEW]
3619 Traceback (most recent call last):
3620   File "C:\Users\JOIN\Downloads\Multi-
Reviewer_Governance_for_Code_Generation\src\evaluator\
humaneval_evaluator.py", line 39, in _worker
3621     ns["check"](candidate)
3622   File "<string>", line 6, in check
3623   File "<string>", line 19, in iscude
3624 TypeError: type complex doesn't define __round__ method
3625 -----
3626 [HumanEval/77] [REPAIRED CODE] (after attempt=1)
3627 def iscude(a):
3628     if isinstance(a, complex):
3629         return False
3630     return round(abs(a) ** (1/3)) ** 3 == abs(a)
3631 -----
3632 [HumanEval/77] Attempt 2/6
3633 [HumanEval/77] ⚡ PASS (attempt=2)
3634 =====
3635 [79/164] HumanEval/78
```

```
3636 =====
=====
3637 [TASK] HumanEval/78 | entry_point=hex_key
3638 [PROMPT PREVIEW]
3639
3640 def hex_key(num):
3641     """You have been tasked to write a function that receives
3642         a hexadecimal number as a string and counts the number of
3643             hexadecimal
3644             digits that are primes (prime number, or a prime, is a natural
3645                 number
3646                 greater than 1 that is not a product of two smaller natural
3647                     numbers).
3648             Hexadecimal digits are 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D
3649                 , E, F.
3650             Prime numbers a...
3651 -----
3652 -----
3653 [INITIAL GENERATED CODE]
3654 def hex_key(num):
3655     prime_hex_digits = {'2', '3', '5', '7', 'B', 'D'}
3656     return sum(1 for digit in num if digit in prime_hex_digits)
3657 -----
3658 [HumanEval/78] Attempt 0/6
3659 [HumanEval/78] 0 PASS (attempt=0)
3660 =====
=====
3661 [80/164] HumanEval/79
3662 -----
3663 [TASK] HumanEval/79 | entry_point=decimal_to_binary
3664 [PROMPT PREVIEW]
3665
3666 def decimal_to_binary(decimal):
3667     """You will be given a number in decimal form and your task is
3668         to convert it to
3669             binary format. The function should return a string, with each
3670                 character representing a binary
3671                 number. Each character in the string will be '0' or '1'.
3672
3673             There will be an extra couple of characters 'db' at the
3674                 beginning and at the end of the string.
3675             The extra character...
3676 -----
3677 -----
3678 [INITIAL GENERATED CODE]
3679 def decimal_to_binary(decimal):
3680     binary = bin(decimal)[2:]
3681     return f"db{binary}db"
3682 -----
3683 -----
3684 [HumanEval/79] Attempt 0/6
```

```
3675 [HumanEval/79] ✘ PASS (attempt=0)
3676 =====
3677 =====
3677 [81/164] HumanEval/80
3678 =====
3678 =====
3679 [TASK] HumanEval/80 | entry_point=is_happy
3680 [PROMPT PREVIEW]
3681
3682 def is_happy(s):
3683     """You are given a string s.
3684     Your task is to check if the string is happy or not.
3685     A string is happy if its length is at least 3 and every 3
3685     consecutive letters are distinct
3686     For example:
3687     is_happy(a) => False
3688     is_happy(aa) => False
3689     is_happy(abcd) => True
3690     is_happy(aabb) => False
3691     is_happy(adb) => True
3692     is_happy(xyy) => False
3693     """
3694 -----
3694 -----
3695 [INITIAL GENERATED CODE]
3696 def is_happy(s):
3697     if len(s) < 3:
3698         return False
3699     for i in range(len(s) - 2):
3700         if s[i] == s[i+1] or s[i] == s[i+2] or s[i+1] == s[i+2]:
3701             return False
3702     return True
3703 -----
3703 -----
3704 [HumanEval/80] Attempt 0/6
3705 [HumanEval/80] ✘ PASS (attempt=0)
3706 =====
3706 =====
3707 [82/164] HumanEval/81
3708 =====
3708 =====
3709 [TASK] HumanEval/81 | entry_point=numerical_letter_grade
3710 [PROMPT PREVIEW]
3711
3712 def numerical_letter_grade(grades):
3713     """It is the last week of the semester and the teacher has to
3713     give the grades
3714     to students. The teacher has been making her own algorithm for
3714     grading.
3715     The only problem is, she has lost the code she used for grading.
3716     She has given you a list of GPAs for some students and you have
3716     to write
3717     a function that can output a list of letter grades usi...
```

```
3718 -----
-----
3719 [INITIAL GENERATED CODE]
3720 def numerical_letter_grade(grades):
3721     letter_grades = []
3722     for gpa in grades:
3723         if gpa == 4.0:
3724             letter_grades.append('A+')
3725         elif gpa > 3.7:
3726             letter_grades.append('A')
3727         elif gpa > 3.3:
3728             letter_grades.append('A-')
3729         elif gpa > 3.0:
3730             letter_grades.append('B+')
3731         elif gpa > 2.7:
3732             letter_grades.append('B')
3733         elif gpa > 2.3:
3734             letter_grades.append('B-')
3735         elif gpa > 2.0:
3736             letter_grades.append('C+')
3737         elif gpa > 1.7:
3738             letter_grades.append('C')
3739         elif gpa > 1.3:
3740             letter_grades.append('C-')
3741         elif gpa > 1.0:
3742             letter_grades.append('D+')
3743         elif gpa > 0.7:
3744             letter_grades.append('D')
3745         elif gpa > 0.0:
3746             letter_grades.append('D-')
3747         else:
3748             letter_grades.append('E')
3749     return letter_grades
3750 -----
-----
3751 [HumanEval/81] Attempt 0/6
3752 [HumanEval/81] [] PASS (attempt=0)
3753 =====
=====
3754 [83/164] HumanEval/82
3755 =====
=====
3756 [TASK] HumanEval/82 | entry_point=prime_length
3757 [PROMPT PREVIEW]
3758
3759 def prime_length(string):
3760     """Write a function that takes a string and returns True if the
3761     string
3762     length is a prime number or False otherwise
3763     Examples
3764     prime_length('Hello') == True
3765     prime_length('abcdcba') == True
3766     prime_length('kittens') == True
```

```
3766     prime_length('orange') == False
3767     """
3768 -----
3769 [INITIAL GENERATED CODE]
3770 def prime_length(string):
3771     length = len(string)
3772     if length < 2:
3773         return False
3774     for i in range(2, int(length**0.5) + 1):
3775         if length % i == 0:
3776             return False
3777     return True
3778 -----
3779 [HumanEval/82] Attempt 0/6
3780 [HumanEval/82] Ø PASS (attempt=0)
3781 =====
3782 [84/164] HumanEval/83
3783 =====
3784 [TASK] HumanEval/83 | entry_point=starts_one_ends
3785 [PROMPT PREVIEW]
3786
3787 def starts_one_ends(n):
3788     """
3789     Given a positive integer n, return the count of the numbers of n
3790     -digit
3791     positive integers that start or end with 1.
3792     """
3793 [INITIAL GENERATED CODE]
3794 def starts_one_ends(n):
3795     if n == 1:
3796         return 1
3797     else:
3798         return 2 * (10 ** (n - 1)) - 10 ** (n - 2)
3799 -----
3800 [HumanEval/83] Attempt 0/6
3801 [HumanEval/83] Ø FAIL (attempt=0)
3802 [TEST ERROR PREVIEW]
3803 Traceback (most recent call last):
3804   File "C:\Users\JOIN\Downloads\Multi-
3805     Reviewer_Governance_for_Code_Generation\src\evaluator\
3806     humaneval_evaluator.py", line 39, in _worker
3807       ns["check"](candidate)
3808   File "<string>", line 6, in check
3809 Assertion
3810 -----
3811 -----
```

```
3809 [HumanEval/83] [REPAIRED CODE] (after attempt=0)
3810 def starts_one_ends(n):
3811     if n == 1:
3812         return 1
3813     else:
3814         return 2 * (10 ** (n - 1)) - 10 ** (n - 2)
3815 -----
3816 -----
3816 [HumanEval/83] Attempt 1/6
3817 [HumanEval/83] ✘ FAIL (attempt=1)
3818 [TEST ERROR PREVIEW]
3819 Traceback (most recent call last):
3820   File "C:\Users\JOIN\Downloads\Multi-
3821     Reviewer_Governance_for_Code_Generation\src\evaluator\
3822       humaneval_evaluator.py", line 39, in _worker
3823       ns["check"](candidate)
3824     File "<string>", line 6, in check
3825     AssertionError
3826 -----
3827 -----
3825 [HumanEval/83] [REPAIRED CODE] (after attempt=1)
3826 def starts_one_ends(n):
3827     if n == 1:
3828         return 1
3829     else:
3830         return 2 * (10 ** (n - 1)) - 10 ** (n - 2)
3831 -----
3832 -----
3832 [HumanEval/83] Attempt 2/6
3833 [HumanEval/83] ✘ FAIL (attempt=2)
3834 [TEST ERROR PREVIEW]
3835 Traceback (most recent call last):
3836   File "C:\Users\JOIN\Downloads\Multi-
3837     Reviewer_Governance_for_Code_Generation\src\evaluator\
3838       humaneval_evaluator.py", line 39, in _worker
3839       ns["check"](candidate)
3840     File "<string>", line 6, in check
3841     AssertionError
3842 -----
3843 -----
3841 [HumanEval/83] [REPAIRED CODE] (after attempt=2)
3842 def starts_one_ends(n):
3843     if n == 1:
3844         return 1
3845     else:
3846         return 2 * (10 ** (n - 1)) - 10 ** (n - 1)
3847 -----
3848 -----
3848 [HumanEval/83] Attempt 3/6
3849 [HumanEval/83] ✘ FAIL (attempt=3)
3850 [TEST ERROR PREVIEW]
3851 Traceback (most recent call last):
3852   File "C:\Users\JOIN\Downloads\Multi-
```

```
3852 Reviewer_Governance_for_Code_Generation\src\evaluator\  
    humaneval_evaluator.py", line 39, in _worker  
3853     ns["check"](candidate)  
3854     File "<string>", line 6, in check  
3855 AssertionError  
3856 -----  
-----  
3857 [HumanEval/83] [REPAIRED CODE] (after attempt=3)  
3858 def starts_one_ends(n):  
3859     if n == 1:  
3860         return 1  
3861     else:  
3862         return 2 * (10 ** (n - 1)) - 10 ** (n - 2)  
3863 -----  
-----  
3864 [HumanEval/83] Attempt 4/6  
3865 [HumanEval/83] ⚡ FAIL (attempt=4)  
3866 [TEST ERROR PREVIEW]  
3867 Traceback (most recent call last):  
3868     File "C:\Users\JOIN\Downloads\Multi-  
        Reviewer_Governance_for_Code_Generation\src\evaluator\  
        humaneval_evaluator.py", line 39, in _worker  
3869     ns["check"](candidate)  
3870     File "<string>", line 6, in check  
3871 AssertionError  
3872 -----  
-----  
3873 [HumanEval/83] [REPAIRED CODE] (after attempt=4)  
3874 def starts_one_ends(n):  
3875     if n == 1:  
3876         return 1  
3877     else:  
3878         return 2 * (10 ** (n - 1)) - 10 ** (n - 2)  
3879 -----  
-----  
3880 [HumanEval/83] Attempt 5/6  
3881 [HumanEval/83] ⚡ FAIL (attempt=5)  
3882 [TEST ERROR PREVIEW]  
3883 Traceback (most recent call last):  
3884     File "C:\Users\JOIN\Downloads\Multi-  
        Reviewer_Governance_for_Code_Generation\src\evaluator\  
        humaneval_evaluator.py", line 39, in _worker  
3885     ns["check"](candidate)  
3886     File "<string>", line 6, in check  
3887 AssertionError  
3888 -----  
-----  
3889 [HumanEval/83] [REPAIRED CODE] (after attempt=5)  
3890 def starts_one_ends(n):  
3891     if n == 1:  
3892         return 1  
3893     else:  
3894         return 2 * (10 ** (n - 1)) - 10 ** (n - 1)
```

```
3895 -----
-----
3896 [HumanEval/83] Attempt 6/6
3897 [HumanEval/83] ⚡ FAIL (attempt=6)
3898 [TEST ERROR PREVIEW]
3899 Traceback (most recent call last):
3900   File "C:\Users\JOIN\Downloads\Multi-
    Reviewer_Governance_for_Code_Generation\src\evaluator\
      humaneval_evaluator.py", line 39, in _worker
3901     ns["check"](candidate)
3902   File "<string>", line 6, in check
3903 AssertionError
3904 -----
-----
3905 [HumanEval/83] ⚡ FINAL FAIL after 6 retries
3906 =====
=====
3907 [85/164] HumanEval/84
3908 =====
=====
3909 [TASK] HumanEval/84 | entry_point=solve
3910 [PROMPT PREVIEW]
3911
3912 def solve(N):
3913     """Given a positive integer N, return the total sum of its
    digits in binary.
3914
3915     Example
3916         For N = 1000, the sum of digits will be 1 the output should
    be "1".
3917         For N = 150, the sum of digits will be 6 the output should
    be "110".
3918         For N = 147, the sum of digits will be 12 the output should
    be "1100".
3919
3920     Variables:
3921         @N integer
3922             Con...
3923 -----
-----
3924 [INITIAL GENERATED CODE]
3925 def solve(N):
3926     return bin(sum(int(digit) for digit in str(N)))[2:]
3927 -----
-----
3928 [HumanEval/84] Attempt 0/6
3929 [HumanEval/84] ⚡ PASS (attempt=0)
3930 =====
=====
3931 [86/164] HumanEval/85
3932 =====
=====
3933 [TASK] HumanEval/85 | entry_point=add
```

```
3934 [PROMPT PREVIEW]
3935
3936 def add(lst):
3937     """Given a non-empty list of integers lst. add the even elements
3938     that are at odd indices..
3939
3940     Examples:
3941         add([4, 2, 6, 7]) ==> 2
3942     """
3943 -----
3944 -----
3944 [INITIAL GENERATED CODE]
3945 def add(lst):
3946     return sum(x for i, x in enumerate(lst) if i % 2 == 1 and x % 2
3947     == 0)
3947 -----
3948 -----
3948 [HumanEval/85] Attempt 0/6
3949 [HumanEval/85] ✘ PASS (attempt=0)
3950 =====
3951 =====
3951 [87/164] HumanEval/86
3952 =====
3952 =====
3953 [TASK] HumanEval/86 | entry_point=anti_shuffle
3954 [PROMPT PREVIEW]
3955
3956 def anti_shuffle(s):
3957     """
3958     Write a function that takes a string and returns an ordered
3959     version of it.
3960     Ordered version of string, is a string where all words (
3961     separated by space)
3962     are replaced by a new word where all the characters arranged in
3963     ascending order based on ascii value.
3964     Note: You should keep the order of words and blank spaces in the
3965     sentence.
3966
3967     For example:...
3968
3969 -----
3966 [INITIAL GENERATED CODE]
3967 def anti_shuffle(s):
3968     words = s.split()
3969     ordered_words = [''.join(sorted(word)) for word in words]
3970     return ' '.join(ordered_words)
3971 -----
3972 -----
3972 [HumanEval/86] Attempt 0/6
3973 [HumanEval/86] ✘ PASS (attempt=0)
3974 =====
3974 =====
```

```
3975 [88/164] HumanEval/87
3976 =====
3977 =====
3977 [TASK] HumanEval/87 | entry_point=get_row
3978 [PROMPT PREVIEW]
3979
3980 def get_row(lst, x):
3981     """
3982         You are given a 2 dimensional data, as a nested lists,
3983         which is similar to matrix, however, unlike matrices,
3984         each row may contain a different number of columns.
3985         Given lst, and integer x, find integers x in the list,
3986         and return list of tuples, [(x1, y1), (x2, y2) ...] such that
3987         each tuple is a coordinate - (row, columns), starting with 0.
3988         So...
3989 -----
3990 -----
3990 [INITIAL GENERATED CODE]
3991 def get_row(lst, x):
3992     result = []
3993     for i, row in enumerate(lst):
3994         for j, value in enumerate(row[::-1]):
3995             if value == x:
3996                 result.append((i, len(row) - 1 - j))
3997     return sorted(result)
3998 -----
3999 -----
3999 [HumanEval/87] Attempt 0/6
4000 [HumanEval/87] ⚡ FAIL (attempt=0)
4001 [TEST ERROR PREVIEW]
4002 Traceback (most recent call last):
4003   File "C:\Users\JOIN\Downloads\Multi-
        Reviewer_Governance_for_Code_Generation\src\evaluator\
        humaneval_evaluator.py", line 39, in _worker
4004     ns["check"](candidate)
4005   File "<string>", line 4, in check
4006 AssertionError
4007 -----
4008 -----
4008 [HumanEval/87] [REPAIRED CODE] (after attempt=0)
4009 def get_row(lst, x):
4010     result = []
4011     for i, row in enumerate(lst):
4012         for j, value in enumerate(row):
4013             if value == x:
4014                 result.append((i, j))
4015     return sorted(result)
4016 -----
4017 -----
4017 [HumanEval/87] Attempt 1/6
4018 [HumanEval/87] ⚡ FAIL (attempt=1)
4019 [TEST ERROR PREVIEW]
4020 Traceback (most recent call last):
```

```
4021 File "C:\Users\JOIN\Downloads\Multi-
    Reviewer_Governance_for_Code_Generation\src\evaluator\
        humaneval_evaluator.py", line 39, in _worker
4022     ns["check"](candidate)
4023     File "<string>", line 4, in check
4024 AssertionError
4025 -----
-----
4026 [HumanEval/87] [REPAIRED CODE] (after attempt=1)
4027 def get_row(lst, x):
4028     result = []
4029     for i, row in enumerate(lst):
4030         for j, value in enumerate(row):
4031             if value == x:
4032                 result.append((i, j))
4033     return sorted(result)
4034 -----
-----
4035 [HumanEval/87] Attempt 2/6
4036 [HumanEval/87] ⚡ FAIL (attempt=2)
4037 [TEST ERROR PREVIEW]
4038 Traceback (most recent call last):
4039     File "C:\Users\JOIN\Downloads\Multi-
        Reviewer_Governance_for_Code_Generation\src\evaluator\
            humaneval_evaluator.py", line 39, in _worker
4040     ns["check"](candidate)
4041     File "<string>", line 4, in check
4042 AssertionError
4043 -----
-----
4044 [HumanEval/87] [REPAIRED CODE] (after attempt=2)
4045 def get_row(lst, x):
4046     result = []
4047     for i, row in enumerate(lst):
4048         for j, value in enumerate(row):
4049             if value == x:
4050                 result.append((i, j))
4051     return sorted(result)
4052 -----
-----
4053 [HumanEval/87] Attempt 3/6
4054 [HumanEval/87] ⚡ FAIL (attempt=3)
4055 [TEST ERROR PREVIEW]
4056 Traceback (most recent call last):
4057     File "C:\Users\JOIN\Downloads\Multi-
        Reviewer_Governance_for_Code_Generation\src\evaluator\
            humaneval_evaluator.py", line 39, in _worker
4058     ns["check"](candidate)
4059     File "<string>", line 4, in check
4060 AssertionError
4061 -----
-----
4062 [HumanEval/87] [REPAIRED CODE] (after attempt=3)
```

```
4063 def get_row(lst, x):
4064     result = []
4065     for i, row in enumerate(lst):
4066         for j, value in enumerate(row):
4067             if value == x:
4068                 result.append((i, j))
4069     return sorted(result)
4070 -----
4071 [HumanEval/87] Attempt 4/6
4072 [HumanEval/87] ⚡ FAIL (attempt=4)
4073 [TEST ERROR PREVIEW]
4074 Traceback (most recent call last):
4075   File "C:\Users\JOIN\Downloads\Multi-
    Reviewer_Governance_for_Code_Generation\src\evaluator\
      humaneval_evaluator.py", line 39, in _worker
4076     ns["check"](candidate)
4077   File "<string>", line 4, in check
4078 AssertionError
4079 -----
4080 [HumanEval/87] [REPAIRED CODE] (after attempt=4)
4081 def get_row(lst, x):
4082     result = []
4083     for i, row in enumerate(lst):
4084         for j, value in enumerate(row):
4085             if value == x:
4086                 result.append((i, j))
4087     return sorted(result)
4088 -----
4089 [HumanEval/87] Attempt 5/6
4090 [HumanEval/87] ⚡ FAIL (attempt=5)
4091 [TEST ERROR PREVIEW]
4092 Traceback (most recent call last):
4093   File "C:\Users\JOIN\Downloads\Multi-
    Reviewer_Governance_for_Code_Generation\src\evaluator\
      humaneval_evaluator.py", line 39, in _worker
4094     ns["check"](candidate)
4095   File "<string>", line 4, in check
4096 AssertionError
4097 -----
4098 [HumanEval/87] [REPAIRED CODE] (after attempt=5)
4099 def get_row(lst, x):
4100     result = []
4101     for i, row in enumerate(lst):
4102         for j, value in enumerate(row):
4103             if value == x:
4104                 result.append((i, j))
4105     return sorted(result)
4106 -----
```

```
4107 [HumanEval/87] Attempt 6/6
4108 [HumanEval/87] ⚡ FAIL (attempt=6)
4109 [TEST ERROR PREVIEW]
4110 Traceback (most recent call last):
4111   File "C:\Users\JOIN\Downloads\Multi-
    Reviewer_Governance_for_Code_Generation\src\evaluator\
      humaneval_evaluator.py", line 39, in _worker
4112     ns["check"](candidate)
4113   File "<string>", line 4, in check
4114 AssertionError
4115 -----
4116 -----
4116 [HumanEval/87] ⚡ FINAL FAIL after 6 retries
4117 =====
4118 [89/164] HumanEval/88
4119 =====
4120 [TASK] HumanEval/88 | entry_point=sort_array
4121 [PROMPT PREVIEW]
4122
4123 def sort_array(array):
4124     """
4125         Given an array of non-negative integers, return a copy of the
        given array after sorting,
4126         you will sort the given array in ascending order if the sum(
            first index value, last index value) is odd,
4127         or sort it in descending order if the sum( first index value,
            last index value) is even.
4128
4129     Note:
4130         * don't change the given array.
4131
4132     Examples:
4133         * sort...
4134 -----
4135 -----
4135 [INITIAL GENERATED CODE]
4136 def sort_array(array):
4137     if not array:
4138         return []
4139
4140     sorted_array = array[:]
4141     if (sorted_array[0] + sorted_array[-1]) % 2 == 0:
4142         sorted_array.sort(reverse=True)
4143     else:
4144         sorted_array.sort()
4145
4146     return sorted_array
4147 -----
4148 -----
4148 [HumanEval/88] Attempt 0/6
4149 [HumanEval/88] ⚡ PASS (attempt=0)
```

```
4150 =====
4151 =====
4151 [90/164] HumanEval/89
4152 =====
4152 =====
4153 [TASK] HumanEval/89 | entry_point=encrypt
4154 [PROMPT PREVIEW]
4155
4156 def encrypt(s):
4157     """Create a function encrypt that takes a string as an argument
4158     and
4159     returns a string encrypted with the alphabet being rotated.
4160     The alphabet should be rotated in a manner such that the letters
4161     shift down by two multiplied to two places.
4162     For example:
4163     encrypt('hi') returns 'lm'
4164     encrypt('asdfghjkl') returns 'ewhjklnop'
4165     encrypt('gf') returns 'kj'...
4165 -----
4165 -----
4166 [INITIAL GENERATED CODE]
4167 def encrypt(s):
4168     result = []
4169     for char in s:
4170         if char.isalpha():
4171             shift = 2 * 2
4172             base = ord('a') if char.islower() else ord('A')
4173             encrypted_char = chr((ord(char) - base + shift) % 26 +
4174             base)
4174             result.append(encrypted_char)
4175         else:
4176             result.append(char)
4177     return ''.join(result)
4178 -----
4178 -----
4179 [HumanEval/89] Attempt 0/6
4180 [HumanEval/89] 0 PASS (attempt=0)
4181 =====
4181 =====
4182 [91/164] HumanEval/90
4183 =====
4183 =====
4184 [TASK] HumanEval/90 | entry_point=next_smallest
4185 [PROMPT PREVIEW]
4186
4187 def next_smallest(lst):
4188     """
4189     You are given a list of integers.
4190     Write a function next_smallest() that returns the 2nd smallest
4191     element of the list.
4191     Return None if there is no such element.
4192
```

```
4193     next_smallest([1, 2, 3, 4, 5]) == 2
4194     next_smallest([5, 1, 4, 3, 2]) == 2
4195     next_smallest([]) == None
4196     next_smallest([1, 1]) == None
4197     """
4198 -----
4199 -----
4199 [INITIAL GENERATED CODE]
4200 def next_smallest(lst):
4201     if len(lst) < 2:
4202         return None
4203     unique_lst = sorted(set(lst))
4204     return unique_lst[1] if len(unique_lst) > 1 else None
4205 -----
4206 -----
4206 [HumanEval/90] Attempt 0/6
4207 [HumanEval/90] 0 PASS (attempt=0)
4208 =====
4209 -----
4209 [92/164] HumanEval/91
4210 =====
4211 -----
4211 [TASK] HumanEval/91 | entry_point=is_bored
4212 [PROMPT PREVIEW]
4213
4214 def is_bored(S):
4215     """
4216     You'll be given a string of words, and your task is to count the
4217     number
4218     of boredoms. A boredom is a sentence that starts with the word "I".
4219     Sentences are delimited by '.', '?' or '!'.
4220     For example:
4221     >>> is_bored("Hello world")
4222     0
4223     >>> is_bored("The sky is blue. The sun is shining. I love this
4224     weather")
4225     1
4226     """
4227 -----
4227 -----
4227 [INITIAL GENERATED CODE]
4228 def is_bored(S):
4229     sentences = S.split('.')
4230     count = 0
4231     for sentence in sentences:
4232         words = sentence.strip().split()
4233         if len(words) > 0 and words[0].lower() == "i":
4234             count += 1
4235     return count
4236 -----
```

```
4237 [HumanEval/91] Attempt 0/6
4238 [HumanEval/91] ✘ PASS (attempt=0)
4239 =====
=====
4240 [93/164] HumanEval/92
4241 =====
=====
4242 [TASK] HumanEval/92 | entry_point=any_int
4243 [PROMPT PREVIEW]
4244
4245 def any_int(x, y, z):
4246     """
4247         Create a function that takes 3 numbers.
4248         Returns true if one of the numbers is equal to the sum of the
4249         other two, and all numbers are integers.
4250         Returns false in any other cases.
4251
4252         Examples
4253             any_int(5, 2, 7) → True
4254
4255             any_int(3, 2, 2) → False
4256
4257             any_int(3, -2, 1) → True
4258
4259             any_int(3.6, -2.2, 2) → False
4260
4261
4262     """
4263 -----
-----
4264 [INITIAL GENERATED CODE]
4265 def any_int(x, y, z):
4266     return all(isinstance(i, int) for i in [x, y, z]) and (x + y ==
4267     z or x + z == y or y + z == x)
4268 -----
-----
4269 [HumanEval/92] Attempt 0/6
4270 [HumanEval/92] ✘ PASS (attempt=0)
4271 =====
=====
4272 [94/164] HumanEval/93
4273 =====
=====
4274 [TASK] HumanEval/93 | entry_point=encode
4275 [PROMPT PREVIEW]
4276
4277 def encode(message):
4278     """
4279         Write a function that takes a message, and encodes in such a
4280         way that it swaps case of all letters, replaces all vowels in
4281         the message with the letter that appears 2 places ahead of that
4282         vowel in the english alphabet.
4283
```

```
4282     Assume only letters.
4283
4284     Examples:
4285     >>> encode('test')
4286     'TGST'
4287     >>> encode('This is a message')
4288     'tHKS KS C MG...
4289 -----
4290 -----
4290 [INITIAL GENERATED CODE]
4291 def encode(message):
4292     vowels = "aeiou"
4293     shifted_vowels = {v: chr((ord(v) - 97 + 2) % 26 + 97) for v in
4294         vowels}
4294     shifted_vowels.update({v.upper(): shifted_vowels[v].upper() for
4295         v in vowels})
4295
4296     encoded_message = ""
4297     for char in message:
4298         if char.isalpha():
4299             if char.lower() in vowels:
4300                 encoded_message += shifted_vowels[char]
4301             else:
4302                 encoded_message += char.swapcase()
4303         else:
4304             encoded_message += char
4305
4306     return encoded_message
4307 -----
4308 -----
4308 [HumanEval/93] Attempt 0/6
4309 [HumanEval/93] ✘ FAIL (attempt=0)
4310 [TEST ERROR PREVIEW]
4311 Traceback (most recent call last):
4312   File "C:\Users\JOIN\Downloads\Multi-
4313     Reviewer_Governance_for_Code_Generation\src\evaluator\
4314       humaneval_evaluator.py", line 39, in _worker
4315         ns["check"](candidate)
4314   File "<string>", line 4, in check
4315 AssertionError: This prints if this assert fails 1 (good for
4316   debugging!)
4316 -----
4317 -----
4317 [HumanEval/93] [REPAIRED CODE] (after attempt=0)
4318 def encode(message):
4319     vowels = "aeiou"
4320     shifted_vowels = {v: chr((ord(v) - 97 + 2) % 26 + 97) for v in
4321         vowels}
4321     shifted_vowels.update({v.upper(): shifted_vowels[v].upper() for
4322         v in vowels})
4322
4323     encoded_message = ""
4324     for char in message:
```

```
4325         if char.isalpha():
4326             if char.lower() in vowels:
4327                 encoded_message += shifted_vowels[char]
4328             else:
4329                 encoded_message += char.swapcase()
4330         else:
4331             encoded_message += char
4332
4333     return encoded_message
4334 -----
-----
4335 [HumanEval/93] Attempt 1/6
4336 [HumanEval/93] ⚡ FAIL (attempt=1)
4337 [TEST ERROR PREVIEW]
4338 Traceback (most recent call last):
4339   File "C:\Users\JOIN\Downloads\Multi-
    Reviewer_Governance_for_Code_Generation\src\evaluator\
      humaneval_evaluator.py", line 39, in _worker
4340     ns["check"](candidate)
4341   File "<string>", line 4, in check
4342 AssertionError: This prints if this assert fails 1 (good for
  debugging!)
4343 -----
-----
4344 [HumanEval/93] [REPAIRED CODE] (after attempt=1)
4345 def encode(message):
4346     vowels = "aeiou"
4347     shifted_vowels = {v: chr((ord(v) - 97 + 2) % 26 + 97) for v in
    vowels}
4348     shifted_vowels.update({v.upper(): shifted_vowels[v].upper() for
    v in vowels})
4349
4350     encoded_message = ""
4351     for char in message:
4352         if char.isalpha():
4353             if char.lower() in vowels:
4354                 encoded_message += shifted_vowels[char]
4355             else:
4356                 encoded_message += char.swapcase()
4357         else:
4358             encoded_message += char
4359
4360     return encoded_message
4361 -----
-----
4362 [HumanEval/93] Attempt 2/6
4363 [HumanEval/93] ⚡ FAIL (attempt=2)
4364 [TEST ERROR PREVIEW]
4365 Traceback (most recent call last):
4366   File "C:\Users\JOIN\Downloads\Multi-
    Reviewer_Governance_for_Code_Generation\src\evaluator\
      humaneval_evaluator.py", line 39, in _worker
4367     ns["check"](candidate)
```

```
4368 File "<string>", line 4, in check
4369 AssertionError: This prints if this assert fails 1 (good for
debugging!)
4370 -----
4371 [HumanEval/93] [REPAIRED CODE] (after attempt=2)
4372 def encode(message):
4373     vowels = "aeiou"
4374     shifted_vowels = {v: chr((ord(v) - 97 + 2) % 26 + 97) for v in
vowels}
4375     shifted_vowels.update({v.upper(): shifted_vowels[v].upper() for
v in vowels})
4376
4377     encoded_message = ""
4378     for char in message:
4379         if char.isalpha():
4380             if char.lower() in vowels:
4381                 encoded_message += shifted_vowels[char]
4382             else:
4383                 encoded_message += char.swapcase()
4384         else:
4385             encoded_message += char
4386
4387     return encoded_message
4388 -----
4389 [HumanEval/93] Attempt 3/6
4390 [HumanEval/93] ⚡ FAIL (attempt=3)
4391 [TEST ERROR PREVIEW]
4392 Traceback (most recent call last):
4393   File "C:\Users\JOIN\Downloads\Multi-
Reviewer_Governance_for_Code_Generation\src\evaluator\
humaneval_evaluator.py", line 39, in _worker
4394     ns["check"](candidate)
4395   File "<string>", line 4, in check
4396 AssertionError: This prints if this assert fails 1 (good for
debugging!)
4397 -----
4398 [HumanEval/93] [REPAIRED CODE] (after attempt=3)
4399 def encode(message):
4400     vowels = "aeiou"
4401     shifted_vowels = {v: chr((ord(v) - 97 + 2) % 26 + 97) for v in
vowels}
4402     shifted_vowels.update({v.upper(): shifted_vowels[v].upper() for
v in vowels})
4403
4404     encoded_message = ""
4405     for char in message:
4406         if char.isalpha():
4407             if char.lower() in vowels:
4408                 encoded_message += shifted_vowels[char]
4409             else:
```

```
4410             encoded_message += char.swapcase()
4411         else:
4412             encoded_message += char
4413
4414     return encoded_message
4415 -----
-----
4416 [HumanEval/93] Attempt 4/6
4417 [HumanEval/93] ⚡ FAIL (attempt=4)
4418 [TEST ERROR PREVIEW]
4419 Traceback (most recent call last):
4420   File "C:\Users\JOIN\Downloads\Multi-
    Reviewer_Governance_for_Code_Generation\src\evaluator\
      humaneval_evaluator.py", line 39, in _worker
4421     ns["check"](candidate)
4422   File "<string>", line 4, in check
4423 AssertionError: This prints if this assert fails 1 (good for
  debugging!)
4424 -----
-----
4425 [HumanEval/93] [REPAIRED CODE] (after attempt=4)
4426 def encode(message):
4427     vowels = "aeiou"
4428     shifted_vowels = {v: chr((ord(v) - 97 + 2) % 26 + 97) for v in
    vowels}
4429     shifted_vowels.update({v.upper(): shifted_vowels[v].upper() for
    v in vowels})
4430
4431     encoded_message = ""
4432     for char in message:
4433         if char.isalpha():
4434             if char.lower() in vowels:
4435                 encoded_message += shifted_vowels[char]
4436             else:
4437                 encoded_message += char.swapcase()
4438         else:
4439             encoded_message += char
4440
4441     return encoded_message
4442 -----
-----
4443 [HumanEval/93] Attempt 5/6
4444 [HumanEval/93] ⚡ FAIL (attempt=5)
4445 [TEST ERROR PREVIEW]
4446 Traceback (most recent call last):
4447   File "C:\Users\JOIN\Downloads\Multi-
    Reviewer_Governance_for_Code_Generation\src\evaluator\
      humaneval_evaluator.py", line 39, in _worker
4448     ns["check"](candidate)
4449   File "<string>", line 4, in check
4450 AssertionError: This prints if this assert fails 1 (good for
  debugging!)
4451 -----
```

```
4451 -----
4452 [HumanEval/93] [REPAIRED CODE] (after attempt=5)
4453 def encode(message):
4454     vowels = "aeiou"
4455     shifted_vowels = {v: chr((ord(v) - 97 + 2) % 26 + 97) for v in
4456                          vowels}
4457     shifted_vowels.update({v.upper(): shifted_vowels[v].upper() for
4458                               v in vowels})
4459
4460     encoded_message = ""
4461     for char in message:
4462         if char.isalpha():
4463             if char.lower() in vowels:
4464                 encoded_message += shifted_vowels[char]
4465             else:
4466                 encoded_message += char.swapcase()
4467         else:
4468             encoded_message += char
4469 -----
4470 -----
4471 [HumanEval/93] Attempt 6/6
4472 [HumanEval/93] ✘ FAIL (attempt=6)
4473 [TEST ERROR PREVIEW]
4474 Traceback (most recent call last):
4475   File "C:\Users\JOIN\Downloads\Multi-
4476       Reviewer_Governance_for_Code_Generation\src\evaluator\
4477       humaneval_evaluator.py", line 39, in _worker
4478       ns["check"](candidate)
4479   File "<string>", line 4, in check
4480AssertionError: This prints if this assert fails 1 (good for
debugging!)
4481 -----
4482 -----
4483 [HumanEval/93] ✘ FINAL FAIL after 6 retries
4484 =====
4485 =====
4486 =====
4487 =====
4488 [95/164] HumanEval/94
4489 =====
4490 =====
4491 =====
4492 [TASK] HumanEval/94 | entry_point=skjkasdkd
4493 [PROMPT PREVIEW]
4494
4495
4496
4497 def skjkasdkd(lst):
4498     """You are given a list of integers.
4499     You need to find the largest prime value and return the sum of
its digits.
4500
4501 Examples:
4502 For lst = [0,3,2,1,3,5,7,4,5,5,2,181,32,4,32,3,2,32,324,4,3]
the output should be 10
```

```
4493     For lst = [1,0,1,8,2,4597,2,1,3,40,1,2,1,2,4,2,5,1] the output
4494         should be 25
4494     For lst = [1,3,1,32,5107,34,83278,109,163,23,2323,32,30,1,9,3]
4494         the o...
4495 -----
4495 -----
4496 [INITIAL GENERATED CODE]
4497 def is_prime(n):
4498     if n <= 1:
4499         return False
4500     for i in range(2, int(n**0.5) + 1):
4501         if n % i == 0:
4502             return False
4503     return True
4504
4505 def skjkasdkd(lst):
4506     max_prime = 0
4507     for num in lst:
4508         if is_prime(num) and num > max_prime:
4509             max_prime = num
4510     return sum(int(digit) for digit in str(max_prime))
4511 -----
4511 -----
4512 [HumanEval/94] Attempt 0/6
4513 [HumanEval/94] [] PASS (attempt=0)
4514 =====
4514 =====
4515 [96/164] HumanEval/95
4516 =====
4516 =====
4517 [TASK] HumanEval/95 | entry_point=check_dict_case
4518 [PROMPT PREVIEW]
4519
4520 def check_dict_case(dict):
4521     """
4522         Given a dictionary, return True if all keys are strings in lower
4523         case or all keys are strings in upper case, else return False.
4524         The function should return False if the given dictionary is
4524         empty.
4525         Examples:
4526             check_dict_case({"a":"apple", "b":"banana"}) should return True.
4527             check_dict_case({"a":"apple", "A":"banana", "B":"banana"})
4527             should ret...
4528 -----
4528 -----
4529 [INITIAL GENERATED CODE]
4530 def check_dict_case(d):
4531     if not d:
4532         return False
4533
4534     lower_keys = all(key.islower() for key in d if isinstance(key,
4534         str))
```

```
4535     upper_keys = all(key.isupper() for key in d if isinstance(key,
str))
4536
4537     return lower_keys or upper_keys
4538 -----
4539 [HumanEval/95] Attempt 0/6
4540 [HumanEval/95] ✘ FAIL (attempt=0)
4541 [TEST ERROR PREVIEW]
4542 Traceback (most recent call last):
4543   File "C:\Users\JOIN\Downloads\Multi-
Reviewer_Governance_for_Code_Generation\src\evaluator\
humaneval_evaluator.py", line 39, in _worker
4544     ns["check"](candidate)
4545   File "<string>", line 6, in check
4546 AssertionError: Third test error: True
4547 -----
4548 [HumanEval/95] [REPAIRED CODE] (after attempt=0)
4549 def check_dict_case(d):
4550     if not d:
4551         return False
4552
4553     lower_keys = any(key.islower() for key in d if isinstance(key,
str))
4554     upper_keys = any(key.isupper() for key in d if isinstance(key,
str))
4555
4556     return lower_keys or upper_keys
4557 -----
4558 [HumanEval/95] Attempt 1/6
4559 [HumanEval/95] ✘ FAIL (attempt=1)
4560 [TEST ERROR PREVIEW]
4561 Traceback (most recent call last):
4562   File "C:\Users\JOIN\Downloads\Multi-
Reviewer_Governance_for_Code_Generation\src\evaluator\
humaneval_evaluator.py", line 39, in _worker
4563     ns["check"](candidate)
4564   File "<string>", line 5, in check
4565 AssertionError: Second test error: True
4566 -----
4567 [HumanEval/95] [REPAIRED CODE] (after attempt=1)
4568 def check_dict_case(d):
4569     if not d:
4570         return False
4571
4572     lower_keys = any(key.islower() for key in d if isinstance(key,
str))
4573     upper_keys = any(key.isupper() for key in d if isinstance(key,
str))
4574
```

```
4575     return lower_keys and upper_keys
4576 -----
4577 -----
4577 [HumanEval/95] Attempt 2/6
4578 [HumanEval/95] ⚡ FAIL (attempt=2)
4579 [TEST ERROR PREVIEW]
4580 Traceback (most recent call last):
4581   File "C:\Users\JOIN\Downloads\Multi-
        Reviewer_Governance_for_Code_Generation\src\evaluator\
          humaneval_evaluator.py", line 39, in _worker
4582     ns["check"](candidate)
4583   File "<string>", line 4, in check
4584 AssertionError: First test error: False
4585 -----
4586 -----
4586 [HumanEval/95] [REPAIRED CODE] (after attempt=2)
4587 def check_dict_case(d):
4588     if not d:
4589         return False
4590
4591     lower_keys = any(key.islower() for key in d if isinstance(key,
        str))
4592     upper_keys = any(key.isupper() for key in d if isinstance(key,
        str))
4593
4594     return lower_keys and upper_keys
4595 -----
4596 -----
4596 [HumanEval/95] Attempt 3/6
4597 [HumanEval/95] ⚡ FAIL (attempt=3)
4598 [TEST ERROR PREVIEW]
4599 Traceback (most recent call last):
4600   File "C:\Users\JOIN\Downloads\Multi-
        Reviewer_Governance_for_Code_Generation\src\evaluator\
          humaneval_evaluator.py", line 39, in _worker
4601     ns["check"](candidate)
4602   File "<string>", line 4, in check
4603 AssertionError: First test error: False
4604 -----
4605 -----
4605 [HumanEval/95] [REPAIRED CODE] (after attempt=3)
4606 def check_dict_case(d):
4607     if not d:
4608         return False
4609
4610     lower_keys = any(key.islower() for key in d if isinstance(key,
        str))
4611     upper_keys = any(key.isupper() for key in d if isinstance(key,
        str))
4612
4613     return lower_keys and upper_keys
4614 -----
```

```
4615 [HumanEval/95] Attempt 4/6
4616 [HumanEval/95] ✘ FAIL (attempt=4)
4617 [TEST ERROR PREVIEW]
4618 Traceback (most recent call last):
4619   File "C:\Users\JOIN\Downloads\Multi-
    Reviewer_Governance_for_Code_Generation\src\evaluator\
      humaneval_evaluator.py", line 39, in _worker
4620     ns["check"](candidate)
4621   File "<string>", line 4, in check
4622 AssertionError: First test error: False
4623 -----
-----
4624 [HumanEval/95] [REPAIRED CODE] (after attempt=4)
4625 def check_dict_case(d):
4626     if not d:
4627         return False
4628
4629     lower_keys = any(key.islower() for key in d if isinstance(key,
        str))
4630     upper_keys = any(key.isupper() for key in d if isinstance(key,
        str))
4631
4632     return lower_keys and upper_keys
4633 -----
-----
4634 [HumanEval/95] Attempt 5/6
4635 [HumanEval/95] ✘ FAIL (attempt=5)
4636 [TEST ERROR PREVIEW]
4637 Traceback (most recent call last):
4638   File "C:\Users\JOIN\Downloads\Multi-
    Reviewer_Governance_for_Code_Generation\src\evaluator\
      humaneval_evaluator.py", line 39, in _worker
4639     ns["check"](candidate)
4640   File "<string>", line 4, in check
4641 AssertionError: First test error: False
4642 -----
-----
4643 [HumanEval/95] [REPAIRED CODE] (after attempt=5)
4644 def check_dict_case(d):
4645     if not d:
4646         return False
4647
4648     lower_keys = any(key.islower() for key in d if isinstance(key,
        str))
4649     upper_keys = any(key.isupper() for key in d if isinstance(key,
        str))
4650
4651     return lower_keys and upper_keys
4652 -----
-----
4653 [HumanEval/95] Attempt 6/6
4654 [HumanEval/95] ✘ FAIL (attempt=6)
4655 [TEST ERROR PREVIEW]
```

```
4656 Traceback (most recent call last):
4657   File "C:\Users\JOIN\Downloads\Multi-
    Reviewer_Governance_for_Code_Generation\src\evaluator\
      humaneval_evaluator.py", line 39, in _worker
4658     ns["check"](candidate)
4659   File "<string>", line 4, in check
4660 AssertionError: First test error: False
4661 -----
4662 -----
4662 [HumanEval/95] ✘ FINAL FAIL after 6 retries
4663 =====
4664 [97/164] HumanEval/96
4665 =====
4666 [TASK] HumanEval/96 | entry_point=count_up_to
4667 [PROMPT PREVIEW]
4668
4669 def count_up_to(n):
4670     """Implement a function that takes an non-negative integer and
        returns an array of the first n
4671     integers that are prime numbers and less than n.
4672     for example:
4673         count_up_to(5) => [2,3]
4674         count_up_to(11) => [2,3,5,7]
4675         count_up_to(0) => []
4676         count_up_to(20) => [2,3,5,7,11,13,17,19]
4677         count_up_to(1) => []
4678         count_up_to(18) => [2,3,5,7,11,13,17]
4679     """
4680 -----
4681 -----
4681 [INITIAL GENERATED CODE]
4682 def is_prime(num):
4683     if num <= 1:
4684         return False
4685     for i in range(2, int(num**0.5) + 1):
4686         if num % i == 0:
4687             return False
4688     return True
4689
4690 def count_up_to(n):
4691     primes = []
4692     for num in range(2, n):
4693         if is_prime(num):
4694             primes.append(num)
4695     return primes
4696 -----
4697 -----
4697 [HumanEval/96] Attempt 0/6
4698 [HumanEval/96] ✘ PASS (attempt=0)
4699 =====
4700 =====
```

```
4700 [98/164] HumanEval/97
4701 =====
4702 =====
4702 [TASK] HumanEval/97 | entry_point=multiply
4703 [PROMPT PREVIEW]
4704
4705 def multiply(a, b):
4706     """Complete the function that takes two integers and returns
4707     the product of their unit digits.
4708     Assume the input is always valid.
4709     Examples:
4710     multiply(148, 412) should return 16.
4711     multiply(19, 28) should return 72.
4712     multiply(2020, 1851) should return 0.
4713     multiply(14,-15) should return 20.
4714     """
4715 -----
4716 -----
4716 [INITIAL GENERATED CODE]
4717 def multiply(a, b):
4718     return (a % 10) * (b % 10)
4719 -----
4720 -----
4720 [HumanEval/97] Attempt 0/6
4721 [HumanEval/97] 0 PASS (attempt=0)
4722 =====
4722 =====
4723 [99/164] HumanEval/98
4724 =====
4724 =====
4725 [TASK] HumanEval/98 | entry_point=count_upper
4726 [PROMPT PREVIEW]
4727
4728 def count_upper(s):
4729     """
4730     Given a string s, count the number of uppercase vowels in even
4731     indices.
4732
4732     For example:
4733     count_upper('aBCdEf') returns 1
4734     count_upper('abcdefg') returns 0
4735     count_upper('dBBE') returns 0
4736     """
4737 -----
4737 -----
4738 [INITIAL GENERATED CODE]
4739 def count_upper(s):
4740     return sum(1 for i, c in enumerate(s) if i % 2 == 0 and c in 'AEIOU')
4741 -----
4741 -----
4742 [HumanEval/98] Attempt 0/6
4743 [HumanEval/98] 0 PASS (attempt=0)
```

```
4744 =====
4745 =====
4745 [100/164] HumanEval/99
4746 =====
4746 =====
4747 [TASK] HumanEval/99 | entry_point=closest_integer
4748 [PROMPT PREVIEW]
4749
4750 def closest_integer(value):
4751     """
4752         Create a function that takes a value (string) representing a
4753             number
4754             and returns the closest integer to it. If the number is
4755                 equidistant
4756                     from two integers, round it away from zero.
4755
4756 Examples
4757 >>> closest_integer("10")
4758 10
4759 >>> closest_integer("15.3")
4760 15
4761
4762 Note:
4763     Rounding away from zero means that if the given number is
4764         equidist...
4764 -----
4764 -----
4765 [INITIAL GENERATED CODE]
4766 def closest_integer(value):
4767     num = float(value)
4768     if abs(num - round(num)) == 0.5:
4769         return int(num + (num > 0))
4770     else:
4771         return round(num)
4772 -----
4772 -----
4773 [HumanEval/99] Attempt 0/6
4774 [HumanEval/99] ✘ FAIL (attempt=0)
4775 [TEST ERROR PREVIEW]
4776 Traceback (most recent call last):
4777   File "C:\Users\JOIN\Downloads\Multi-
        Reviewer_Governance_for_Code_Generation\src\evaluator\
          humaneval_evaluator.py", line 39, in _worker
4778     ns["check"](candidate)
4779   File "<string>", line 6, in check
4780 AssertionError: Test 3
4781 -----
4781 -----
4782 [HumanEval/99] [REPAIRED CODE] (after attempt=0)
4783 def closest_integer(value):
4784     num = float(value)
4785     if abs(num - round(num)) == 0.5:
4786         return int(num + (num > 0))
```

```
4787     else:
4788         return round(num)
4789 -----
4790 [HumanEval/99] Attempt 1/6
4791 [HumanEval/99] ✘ FAIL (attempt=1)
4792 [TEST ERROR PREVIEW]
4793 Traceback (most recent call last):
4794   File "C:\Users\JOIN\Downloads\Multi-
        Reviewer_Governance_for_Code_Generation\src\evaluator\
          humaneval_evaluator.py", line 39, in _worker
4795     ns["check"](candidate)
4796   File "<string>", line 6, in check
4797 AssertionError: Test 3
4798 -----
4799 [HumanEval/99] [REPAIRED CODE] (after attempt=1)
4800 def closest_integer(value):
4801     num = float(value)
4802     if abs(num - round(num)) == 0.5:
4803         return int(num + (num > 0))
4804     else:
4805         return round(num)
4806 -----
4807 [HumanEval/99] Attempt 2/6
4808 [HumanEval/99] ✘ FAIL (attempt=2)
4809 [TEST ERROR PREVIEW]
4810 Traceback (most recent call last):
4811   File "C:\Users\JOIN\Downloads\Multi-
        Reviewer_Governance_for_Code_Generation\src\evaluator\
          humaneval_evaluator.py", line 39, in _worker
4812     ns["check"](candidate)
4813   File "<string>", line 6, in check
4814 AssertionError: Test 3
4815 -----
4816 [HumanEval/99] [REPAIRED CODE] (after attempt=2)
4817 def closest_integer(value):
4818     num = float(value)
4819     if abs(num - round(num)) == 0.5:
4820         return int(num + (num > 0))
4821     else:
4822         return round(num)
4823 -----
4824 [HumanEval/99] Attempt 3/6
4825 [HumanEval/99] ✘ FAIL (attempt=3)
4826 [TEST ERROR PREVIEW]
4827 Traceback (most recent call last):
4828   File "C:\Users\JOIN\Downloads\Multi-
        Reviewer_Governance_for_Code_Generation\src\evaluator\
          humaneval_evaluator.py", line 39, in _worker
```

```
4829     ns["check"](candidate)
4830 File "<string>", line 6, in check
4831 AssertionError: Test 3
4832 -----
4833 [HumanEval/99] [REPAIRED CODE] (after attempt=3)
4834 def closest_integer(value):
4835     num = float(value)
4836     if abs(num - round(num)) == 0.5:
4837         return int(num + (num > 0))
4838     else:
4839         return round(num)
4840 -----
4841 [HumanEval/99] Attempt 4/6
4842 [HumanEval/99] ⚡ FAIL (attempt=4)
4843 [TEST ERROR PREVIEW]
4844 Traceback (most recent call last):
4845   File "C:\Users\JOIN\Downloads\Multi-
        Reviewer_Governance_for_Code_Generation\src\evaluator\
          humaneval_evaluator.py", line 39, in _worker
4846     ns["check"](candidate)
4847   File "<string>", line 6, in check
4848 AssertionError: Test 3
4849 -----
4850 [HumanEval/99] [REPAIRED CODE] (after attempt=4)
4851 def closest_integer(value):
4852     num = float(value)
4853     if abs(num - round(num)) == 0.5:
4854         return int(num + (num > 0))
4855     else:
4856         return round(num)
4857 -----
4858 [HumanEval/99] Attempt 5/6
4859 [HumanEval/99] ⚡ FAIL (attempt=5)
4860 [TEST ERROR PREVIEW]
4861 Traceback (most recent call last):
4862   File "C:\Users\JOIN\Downloads\Multi-
        Reviewer_Governance_for_Code_Generation\src\evaluator\
          humaneval_evaluator.py", line 39, in _worker
4863     ns["check"](candidate)
4864   File "<string>", line 6, in check
4865 AssertionError: Test 3
4866 -----
4867 [HumanEval/99] [REPAIRED CODE] (after attempt=5)
4868 def closest_integer(value):
4869     num = float(value)
4870     if abs(num - round(num)) == 0.5:
4871         return int(num + (num > 0))
4872     else:
```

```
4873     return round(num)
4874 -----
4875 -----
4875 [HumanEval/99] Attempt 6/6
4876 [HumanEval/99] ⚡ FAIL (attempt=6)
4877 [TEST ERROR PREVIEW]
4878 Traceback (most recent call last):
4879   File "C:\Users\JOIN\Downloads\Multi-
        Reviewer_Governance_for_Code_Generation\src\evaluator\
          humaneval_evaluator.py", line 39, in _worker
4880     ns["check"](candidate)
4881   File "<string>", line 6, in check
4882 AssertionError: Test 3
4883 -----
4884 -----
4884 [HumanEval/99] ⚡ FINAL FAIL after 6 retries
4885 =====
4886 -----
4886 [101/164] HumanEval/100
4887 =====
4888 -----
4888 [TASK] HumanEval/100 | entry_point=make_a_pile
4889 [PROMPT PREVIEW]
4890
4891 def make_a_pile(n):
4892     """
4893         Given a positive integer n, you have to make a pile of n levels
4893         of stones.
4894         The first level has n stones.
4895         The number of stones in the next level is:
4896             - the next odd number if n is odd.
4897             - the next even number if n is even.
4898         Return the number of stones in each level in a list, where
4898         element at index
4899             i represents the number of stones in th...
4900 -----
4901 -----
4901 [INITIAL GENERATED CODE]
4902 def make_a_pile(n):
4903     pile = []
4904     current_stones = n
4905     for i in range(n):
4906         pile.append(current_stones)
4907         if n % 2 == 0:
4908             current_stones += 2
4909         else:
4910             current_stones += 2
4911     return pile
4912 -----
4913 -----
4913 [HumanEval/100] Attempt 0/6
4914 [HumanEval/100] ⚡ PASS (attempt=0)
4915 =====
```

```
4915 =====
4916 [102/164] HumanEval/101
4917 =====
4918 [TASK] HumanEval/101 | entry_point=words_string
4919 [PROMPT PREVIEW]
4920
4921 def words_string(s):
4922     """
4923         You will be given a string of words separated by commas or
4924         spaces. Your task is
4925             to split the string into words and return an array of the words.
4926
4927             For example:
4928                 words_string("Hi, my name is John") == ["Hi", "my", "name", "is",
4929                     "John"]
4930                 words_string("One, two, three, four, five, six") == ["One", "two",
4931                     "three", "four", "five", "six"]
4932                     """
4933 -----
4934 -----
4935 [INITIAL GENERATED CODE]
4936 def words_string(s):
4937     return s.replace(',', ' ').split()
4938 -----
4939 [HumanEval/101] Attempt 0/6
4940 [HumanEval/101] 0 PASS (attempt=0)
4941 =====
4942 =====
4943 [103/164] HumanEval/102
4944 =====
4945 =====
4946 [TASK] HumanEval/102 | entry_point=choose_num
4947 [PROMPT PREVIEW]
4948
4949 def choose_num(x, y):
5000     """This function takes two positive numbers x and y and returns
5001         the
5002             biggest even integer number that is in the range [x, y]
5003             inclusive. If
5004                 there's no such number, then the function should return -1.
5005
5006             For example:
5007                 choose_num(12, 15) = 14
5008                 choose_num(13, 12) = -1
5009                 """
5010 -----
5011 -----
5012 [INITIAL GENERATED CODE]
5013 def choose_num(x, y):
5014     if x > y:
5015         return -1
```

```
4957     for num in range(y, x-1, -1):
4958         if num % 2 == 0:
4959             return num
4960     return -1
4961 -----
4962 -----
4962 [HumanEval/102] Attempt 0/6
4963 [HumanEval/102] Ø PASS (attempt=0)
4964 =====
4965 =====
4965 [104/164] HumanEval/103
4966 =====
4967 =====
4967 [TASK] HumanEval/103 | entry_point=rounded_avg
4968 [PROMPT PREVIEW]
4969
4970 def rounded_avg(n, m):
4971     """You are given two positive integers n and m, and your task is
4972     to compute the
4973     average of the integers from n through m (including n and m).
4974     Round the answer to the nearest integer and convert that to
4975     binary.
4976     If n is greater than m, return -1.
4977     Example:
4978     rounded_avg(1, 5) => "0b11"
4979     rounded_avg(7, 5) => -1
4980     rounded_avg(10, 20) => "0b1111"
4981     r...
4980 -----
4981 -----
4981 [INITIAL GENERATED CODE]
4982 def rounded_avg(n, m):
4983     if n > m:
4984         return -1
4985     avg = round((n + m) / 2)
4986     return bin(avg)
4987 -----
4988 -----
4988 [HumanEval/103] Attempt 0/6
4989 [HumanEval/103] Ø PASS (attempt=0)
4990 =====
4991 =====
4991 [105/164] HumanEval/104
4992 =====
4993 =====
4993 [TASK] HumanEval/104 | entry_point=unique_digits
4994 [PROMPT PREVIEW]
4995
4996 def unique_digits(x):
4997     """Given a list of positive integers x. return a sorted list of
4998     all
4999     elements that hasn't any even digit.
```

```
5000     Note: Returned list should be sorted in increasing order.
5001
5002     For example:
5003     >>> unique_digits([15, 33, 1422, 1])
5004     [1, 15, 33]
5005     >>> unique_digits([152, 323, 1422, 10])
5006     []
5007     """
5008 -----
5009 -----
5009 [INITIAL GENERATED CODE]
5010 def unique_digits(x):
5011     def has_no_even_digit(n):
5012         return all(int(digit) % 2 != 0 for digit in str(n))
5013
5014     result = [num for num in x if has_no_even_digit(num)]
5015     return sorted(result)
5016 -----
5017 -----
5017 [HumanEval/104] Attempt 0/6
5018 [HumanEval/104] [] PASS (attempt=0)
5019 =====
5020 =====
5020 [106/164] HumanEval/105
5021 =====
5021 =====
5022 [TASK] HumanEval/105 | entry_point=by_length
5023 [PROMPT PREVIEW]
5024
5025 def by_length(arr):
5026     """
5027         Given an array of integers, sort the integers that are between 1
5028         and 9 inclusive,
5029         reverse the resulting array, and then replace each digit by its
5030         corresponding name from
5031         "One", "Two", "Three", "Four", "Five", "Six", "Seven", "Eight"
5032         ", "Nine".
5033
5034     For example:
5035         arr = [2, 1, 1, 4, 5, 8, 2, 3]
5036         -> sort arr -> [1, 1, 2, 2, 3, 4, 5, 8]...
5037
5038 -----
5038 -----
5035 [INITIAL GENERATED CODE]
5036 def by_length(arr):
5037     digit_names = ["One", "Two", "Three", "Four", "Five", "Six", "
5038     Seven", "Eight", "Nine"]
5039     filtered_sorted_arr = sorted([num for num in arr if 1 <= num <=
5040     9])
5040     reversed_filtered_sorted_arr = filtered_sorted_arr[::-1]
5041     result = [digit_names[num - 1] for num in
5042     reversed_filtered_sorted_arr]
5041     return result
```

```
5042 -----
-----
5043 [HumanEval/105] Attempt 0/6
5044 [HumanEval/105] Ø PASS (attempt=0)
5045 =====
=====
5046 [107/164] HumanEval/106
5047 =====
=====
5048 [TASK] HumanEval/106 | entry_point=f
5049 [PROMPT PREVIEW]
5050
5051 def f(n):
5052     """ Implement the function f that takes n as a parameter,
5053     and returns a list of size n, such that the value of the element
5054     at index i is the factorial of i if i is even
5055     or the sum of numbers from 1 to i otherwise.
5056     i starts from 1.
5057     the factorial of i is the multiplication of the numbers from 1
5058     to i (1 * 2 * ... * i).
5059     Example:
5060     f(5) == [1, 2, 6, 24, 15]
5061     """
5060 -----
-----
5061 [INITIAL GENERATED CODE]
5062 def f(n):
5063     result = []
5064     for i in range(1, n + 1):
5065         if i % 2 == 0:
5066             factorial = 1
5067             for j in range(1, i + 1):
5068                 factorial *= j
5069             result.append(factorial)
5070         else:
5071             sum_numbers = sum(range(1, i + 1))
5072             result.append(sum_numbers)
5073     return result
5074 -----
-----
5075 [HumanEval/106] Attempt 0/6
5076 [HumanEval/106] Ø PASS (attempt=0)
5077 =====
=====
5078 [108/164] HumanEval/107
5079 =====
=====
5080 [TASK] HumanEval/107 | entry_point=even_odd_palindrome
5081 [PROMPT PREVIEW]
5082
5083 def even_odd_palindrome(n):
5084     """
5085     Given a positive integer n, return a tuple that has the number
```

```
5085 of even and odd
5086     integer palindromes that fall within the range(1, n), inclusive.
5087
5088     Example 1:
5089
5090         Input: 3
5091         Output: (1, 2)
5092         Explanation:
5093             Integer palindrome are 1, 2, 3. one of them is even, and two
5094             of them are odd.
5095
5096     Example 2:
5097
5098         Input: 12
5099         Outp...
5100 -----
5101 -----
5100 [INITIAL GENERATED CODE]
5101 def even_odd_palindrome(n):
5102     def is_palindrome(num):
5103         return str(num) == str(num)[::-1]
5104
5105     even_count = 0
5106     odd_count = 0
5107
5108     for i in range(1, n + 1):
5109         if is_palindrome(i):
5110             if i % 2 == 0:
5111                 even_count += 1
5112             else:
5113                 odd_count += 1
5114
5115     return (even_count, odd_count)
5116 -----
5117 -----
5117 [HumanEval/107] Attempt 0/6
5118 [HumanEval/107] 0 PASS (attempt=0)
5119 =====
5120 =====
5120 [109/164] HumanEval/108
5121 =====
5121 =====
5122 [TASK] HumanEval/108 | entry_point=count_nums
5123 [PROMPT PREVIEW]
5124
5125 def count_nums(arr):
5126     """
5127         Write a function count_nums which takes an array of integers and
5128         returns
5129             the number of elements which has a sum of digits > 0.
5130             If a number is negative, then its first signed digit will be
5131             negative:
5130             e.g. -123 has signed digits -1, 2, and 3.
```

```
5131     >>> count_nums([]) == 0
5132     >>> count_nums([-1, 11, -11]) == 1
5133     >>> count_nums([1, 1, 2]) == 3
5134     """
5135 -----
5136 -----
5136 [INITIAL GENERATED CODE]
5137 def count_nums(arr):
5138     def sum_of_digits(n):
5139         if n >= 0:
5140             return sum(int(digit) for digit in str(n))
5141         else:
5142             digits = [int(d) for d in str(n)[1:]]
5143             digits[0] *= -1
5144             return sum(digits)
5145
5146     return sum(1 for num in arr if sum_of_digits(num) > 0)
5147 -----
5148 -----
5148 [HumanEval/108] Attempt 0/6
5149 [HumanEval/108] [] PASS (attempt=0)
5150 =====
5151 =====
5151 [110/164] HumanEval/109
5152 =====
5152 =====
5153 [TASK] HumanEval/109 | entry_point=move_one_ball
5154 [PROMPT PREVIEW]
5155
5156 def move_one_ball(arr):
5157     """We have an array 'arr' of N integers arr[1], arr[2], ..., arr[N]. The
5158     numbers in the array will be randomly ordered. Your task is to
5159     determine if
5160     it is possible to get an array sorted in non-decreasing order by
5161     performing
5162     the following operation on the given array:
5163     You are allowed to perform right shift operation any number
5164     of times.
5165
5166     One...
5167 """
5168
5169
5170     n = len(arr)
5171     sorted_arr = sorted(arr)
5172
5173     for i in range(n):
5174         if arr == sorted_arr:
```

```
5175         return True
5176         arr.insert(0, arr.pop())
5177
5178     return False
5179 -----
5180 -----
5180 [HumanEval/109] Attempt 0/6
5181 [HumanEval/109] Ø PASS (attempt=0)
5182 =====
5183 =====
5183 [111/164] HumanEval/110
5184 =====
5185 [TASK] HumanEval/110 | entry_point=exchange
5186 [PROMPT PREVIEW]
5187
5188 def exchange(lst1, lst2):
5189     """In this problem, you will implement a function that takes two
5190     lists of numbers,
5190     and determines whether it is possible to perform an exchange of
5191     elements
5192     between them to make lst1 a list of only even numbers.
5193     There is no limit on the number of exchanged elements between
5194     lst1 and lst2.
5193     If it is possible to exchange elements between the lst1 and lst2
5194     ...
5194 -----
5194 -----
5195 [INITIAL GENERATED CODE]
5196 def exchange(lst1, lst2):
5197     odd_count = sum(1 for x in lst1 if x % 2 != 0)
5198     even_count_in_lst2 = sum(1 for x in lst2 if x % 2 == 0)
5199
5200     return "YES" if even_count_in_lst2 >= odd_count else "NO"
5201 -----
5201 -----
5202 [HumanEval/110] Attempt 0/6
5203 [HumanEval/110] Ø PASS (attempt=0)
5204 =====
5204 -----
5205 [112/164] HumanEval/111
5206 =====
5206 -----
5207 [TASK] HumanEval/111 | entry_point=histogram
5208 [PROMPT PREVIEW]
5209
5210 def histogram(test):
5211     """Given a string representing a space separated lowercase
5212     letters, return a dictionary
5212     of the letter with the most repetition and containing the
5213     corresponding count.
5213     If several letters have the same occurrence, return all of them.
5214
```

```

5215     Example:
5216     histogram('a b c') == {'a': 1, 'b': 1, 'c': 1}
5217     histogram('a b b a') == {'a': 2, 'b': 2}
5218     histogram('a b c...')
5219 -----
5220 -----
5220 [INITIAL GENERATED CODE]
5221 def histogram(test):
5222     if not test:
5223         return {}
5224
5225     words = test.split()
5226     count_dict = {}
5227
5228     for word in words:
5229         if word in count_dict:
5230             count_dict[word] += 1
5231         else:
5232             count_dict[word] = 1
5233
5234     max_count = max(count_dict.values(), default=0)
5235     result = {word: count for word, count in count_dict.items() if
5236               count == max_count}
5237
5238     return result
5239 -----
5239 [HumanEval/111] Attempt 0/6
5240 [HumanEval/111] [] PASS (attempt=0)
5241 =====
5241 =====
5242 [113/164] HumanEval/112
5243 =====
5243 =====
5244 [TASK] HumanEval/112 | entry_point=reverse_delete
5245 [PROMPT PREVIEW]
5246
5247 def reverse_delete(s,c):
5248     """Task
5249     We are given two strings s and c, you have to deleted all the
5250     characters in s that are equal to any character in c
5251     then check if the result string is palindrome.
5252     A string is called palindrome if it reads the same backward as
5253     forward.
5254     You should return a tuple containing the result string and True/
5255     False for the check.
5253     Example
5254     For s = "ab...
5255 -----
5255 -----
5256 [INITIAL GENERATED CODE]
5257 def reverse_delete(s, c):
5258     result = ''.join([char for char in s if char not in c])

```

```
5259     return (result, result == result[::-1])
5260 -----
5261 -----
5261 [HumanEval/112] Attempt 0/6
5262 [HumanEval/112] ✘ PASS (attempt=0)
5263 =====
5264 -----
5264 [114/164] HumanEval/113
5265 =====
5266 -----
5266 [TASK] HumanEval/113 | entry_point=odd_count
5267 [PROMPT PREVIEW]
5268
5269 def odd_count(lst):
5270     """Given a list of strings, where each string consists of only
5270     digits, return a list.
5271         Each element i of the output should be "the number of odd
5271         elements in the
5272             string i of the input." where all the i's should be replaced by
5272             the number
5273                 of odd digits in the i'th string of the input.
5274
5275     >>> odd_count(['1234567'])
5276     ["the number of odd elements 4n the str4ng 4...
5277 -----
5278 -----
5278 [INITIAL GENERATED CODE]
5279 def odd_count(lst):
5280     return [f"the number of odd elements {sum(1 for char in s if int
5280     (char) % 2 != 0)}n the str{s.index(s)+1}ng {s.index(s)+1} of the {s.
5280     index(s)+1}nput." for s in lst]
5281 -----
5282 -----
5282 [HumanEval/113] Attempt 0/6
5283 [HumanEval/113] ✘ FAIL (attempt=0)
5284 [TEST ERROR PREVIEW]
5285 Traceback (most recent call last):
5286   File "C:\Users\JOIN\Downloads\Multi-
5286   Reviewer_Governance_for_Code_Generation\src\evaluator\
5286   humaneval_evaluator.py", line 39, in _worker
5287     ns["check"](candidate)
5288   File "<string>", line 4, in check
5289 AssertionError: Test 1
5290 -----
5291 -----
5291 [HumanEval/113] [REPAIRED CODE] (after attempt=0)
5292 def odd_count(lst):
5293     return [f"the number of odd elements {sum(1 for char in s if int
5293     (char) % 2 != 0)} in the string {lst.index(s)+1} of the input." for
5293     s in lst]
5294 -----
5295 -----
5295 [HumanEval/113] Attempt 1/6
```

```
5296 [HumanEval/113] ⚡ FAIL (attempt=1)
5297 [TEST ERROR PREVIEW]
5298 Traceback (most recent call last):
5299   File "C:\Users\JOIN\Downloads\Multi-
    Reviewer_Governance_for_Code_Generation\src\evaluator\
      humaneval_evaluator.py", line 39, in _worker
5300     ns["check"](candidate)
5301   File "<string>", line 4, in check
5302 AssertionError: Test 1
5303 -----
-----
5304 [HumanEval/113] [REPAIRED CODE] (after attempt=1)
5305 def odd_count(lst):
5306     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]
5307 -----
-----
5308 [HumanEval/113] Attempt 2/6
5309 [HumanEval/113] ⚡ FAIL (attempt=2)
5310 [TEST ERROR PREVIEW]
5311 Traceback (most recent call last):
5312   File "C:\Users\JOIN\Downloads\Multi-
    Reviewer_Governance_for_Code_Generation\src\evaluator\
      humaneval_evaluator.py", line 39, in _worker
5313     ns["check"](candidate)
5314   File "<string>", line 4, in check
5315 AssertionError: Test 1
5316 -----
-----
5317 [HumanEval/113] [REPAIRED CODE] (after attempt=2)
5318 def odd_count(lst):
5319     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]
5320 -----
-----
5321 [HumanEval/113] Attempt 3/6
5322 [HumanEval/113] ⚡ FAIL (attempt=3)
5323 [TEST ERROR PREVIEW]
5324 Traceback (most recent call last):
5325   File "C:\Users\JOIN\Downloads\Multi-
    Reviewer_Governance_for_Code_Generation\src\evaluator\
      humaneval_evaluator.py", line 39, in _worker
5326     ns["check"](candidate)
5327   File "<string>", line 4, in check
5328 AssertionError: Test 1
5329 -----
-----
5330 [HumanEval/113] [REPAIRED CODE] (after attempt=3)
5331 def odd_count(lst):
5332     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
```

```
5332 s in lst]
5333 -----
-----
5334 [HumanEval/113] Attempt 4/6
5335 [HumanEval/113] ⚡ FAIL (attempt=4)
5336 [TEST ERROR PREVIEW]
5337 Traceback (most recent call last):
5338   File "C:\Users\JOIN\Downloads\Multi-
      Reviewer_Governance_for_Code_Generation\src\evaluator\
      humaneval_evaluator.py", line 39, in _worker
5339     ns["check"](candidate)
5340   File "<string>", line 4, in check
5341 AssertionError: Test 1
5342 -----
-----
5343 [HumanEval/113] [REPAIRED CODE] (after attempt=4)
5344 def odd_count(lst):
5345     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]
5346 -----
-----
5347 [HumanEval/113] Attempt 5/6
5348 [HumanEval/113] ⚡ FAIL (attempt=5)
5349 [TEST ERROR PREVIEW]
5350 Traceback (most recent call last):
5351   File "C:\Users\JOIN\Downloads\Multi-
      Reviewer_Governance_for_Code_Generation\src\evaluator\
      humaneval_evaluator.py", line 39, in _worker
5352     ns["check"](candidate)
5353   File "<string>", line 4, in check
5354 AssertionError: Test 1
5355 -----
-----
5356 [HumanEval/113] [REPAIRED CODE] (after attempt=5)
5357 def odd_count(lst):
5358     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]
5359 -----
-----
5360 [HumanEval/113] Attempt 6/6
5361 [HumanEval/113] ⚡ FAIL (attempt=6)
5362 [TEST ERROR PREVIEW]
5363 Traceback (most recent call last):
5364   File "C:\Users\JOIN\Downloads\Multi-
      Reviewer_Governance_for_Code_Generation\src\evaluator\
      humaneval_evaluator.py", line 39, in _worker
5365     ns["check"](candidate)
5366   File "<string>", line 4, in check
5367 AssertionError: Test 1
5368 -----
-----
```

```
5369 [HumanEval/113] ⚡ FINAL FAIL after 6 retries
5370 =====
5371 =====
5371 [115/164] HumanEval/114
5372 =====
5372 =====
5373 [TASK] HumanEval/114 | entry_point=minSubArraySum
5374 [PROMPT PREVIEW]
5375
5376 def minSubArraySum(nums):
5377     """
5378         Given an array of integers nums, find the minimum sum of any non
5379         -empty sub-array
5380         of nums.
5380         Example
5381         minSubArraySum([2, 3, 4, 1, 2, 4]) == 1
5382         minSubArraySum([-1, -2, -3]) == -6
5383     """
5384 -----
5384 -----
5385 [INITIAL GENERATED CODE]
5386 def minSubArraySum(nums):
5387     min_sum = float('inf')
5388     current_sum = 0
5389     for num in nums:
5390         current_sum = min(num, current_sum + num)
5391         min_sum = min(min_sum, current_sum)
5392     return min_sum
5393 -----
5393 -----
5394 [HumanEval/114] Attempt 0/6
5395 [HumanEval/114] ⚡ PASS (attempt=0)
5396 =====
5396 =====
5397 [116/164] HumanEval/115
5398 =====
5398 =====
5399 [TASK] HumanEval/115 | entry_point=max_fill
5400 [PROMPT PREVIEW]
5401
5402 def max_fill(grid, capacity):
5403     import math
5404     """
5405         You are given a rectangular grid of wells. Each row represents a
5406         single well,
5406         and each 1 in a row represents a single unit of water.
5407         Each well has a corresponding bucket that can be used to extract
5407         water from it,
5408         and all buckets have the same capacity.
5409         Your task is to use the buckets to empty the wells.
5410         Output the numb...
5411 -----
5411 -----
```

```
5412 [INITIAL GENERATED CODE]
5413 def max_fill(grid, capacity):
5414     total_water = sum(sum(row) for row in grid)
5415     return math.ceil(total_water / capacity)
5416 -----
5417 -----
5417 [HumanEval/115] Attempt 0/6
5418 [HumanEval/115] ⚡ FAIL (attempt=0)
5419 [TEST ERROR PREVIEW]
5420 Traceback (most recent call last):
5421   File "C:\Users\JOIN\Downloads\Multi-
      Reviewer_Governance_for_Code_Generation\src\evaluator\
      humaneval_evaluator.py", line 39, in _worker
5422     ns["check"](candidate)
5423   File "<string>", line 6, in check
5424   File "<string>", line 40, in max_fill
5425 NameError: name 'math' is not defined
5426 -----
5427 -----
5427 [HumanEval/115] [REPAIRED CODE] (after attempt=0)
5428 import math
5429
5430 def max_fill(grid, capacity):
5431     total_water = sum(sum(row) for row in grid)
5432     return math.ceil(total_water / capacity)
5433 -----
5434 -----
5434 [HumanEval/115] Attempt 1/6
5435 [HumanEval/115] ⚡ FAIL (attempt=1)
5436 [TEST ERROR PREVIEW]
5437 Traceback (most recent call last):
5438   File "C:\Users\JOIN\Downloads\Multi-
      Reviewer_Governance_for_Code_Generation\src\evaluator\
      humaneval_evaluator.py", line 39, in _worker
5439     ns["check"](candidate)
5440   File "<string>", line 13, in check
5441 AssertionError: Error
5442 -----
5443 -----
5443 [HumanEval/115] [REPAIRED CODE] (after attempt=1)
5444 import math
5445
5446 def max_fill(grid, capacity):
5447     total_water = sum(sum(row) for row in grid)
5448     if capacity == 0:
5449         raise ValueError("Capacity cannot be zero")
5450     return math.ceil(total_water / capacity)
5451 -----
5452 -----
5452 [HumanEval/115] Attempt 2/6
5453 [HumanEval/115] ⚡ FAIL (attempt=2)
5454 [TEST ERROR PREVIEW]
5455 Traceback (most recent call last):
```

```
5456 File "C:\Users\JOIN\Downloads\Multi-
    Reviewer_Governance_for_Code_Generation\src\evaluator\
        humaneval_evaluator.py", line 39, in _worker
5457     ns["check"]的文化)
5458 File "<string>", line 13, in check
5459 AssertionError: Error
5460 -----
-----
5461 [HumanEval/115] [REPAIRED CODE] (after attempt=2)
5462 import math
5463
5464 def max_fill(grid, capacity):
5465     total_water = sum(sum(row) for row in grid)
5466     if capacity == 0:
5467         raise ValueError("Capacity cannot be zero")
5468     return math.ceil(total_water / capacity)
5469 -----
-----
5470 [HumanEval/115] Attempt 3/6
5471 [HumanEval/115] 0 FAIL (attempt=3)
5472 [TEST ERROR PREVIEW]
5473 Traceback (most recent call last):
5474     File "C:\Users\JOIN\Downloads\Multi-
        Reviewer_Governance_for_Code_Generation\src\evaluator\
            humaneval_evaluator.py", line 39, in _worker
5475     ns["check"]的文化)
5476 File "<string>", line 13, in check
5477 AssertionError: Error
5478 -----
-----
5479 [HumanEval/115] [REPAIRED CODE] (after attempt=3)
5480 import math
5481
5482 def max_fill(grid, capacity):
5483     total_water = sum(sum(row) for row in grid)
5484     if capacity == 0:
5485         raise ValueError("Capacity cannot be zero")
5486     return math.ceil(total_water / capacity)
5487 -----
-----
5488 [HumanEval/115] Attempt 4/6
5489 [HumanEval/115] 0 FAIL (attempt=4)
5490 [TEST ERROR PREVIEW]
5491 Traceback (most recent call last):
5492     File "C:\Users\JOIN\Downloads\Multi-
        Reviewer_Governance_for_Code_Generation\src\evaluator\
            humaneval_evaluator.py", line 39, in _worker
5493     ns["check"]文化)
5494 File "<string>", line 13, in check
5495 AssertionError: Error
5496 -----
-----
5497 [HumanEval/115] [REPAIRED CODE] (after attempt=4)
```

```
5498 import math
5499
5500 def max_fill(grid, capacity):
5501     total_water = sum(sum(row) for row in grid)
5502     if capacity == 0:
5503         raise ValueError("Capacity cannot be zero")
5504     return math.ceil(total_water / capacity)
5505 -----
5506 -----
5506 [HumanEval/115] Attempt 5/6
5507 [HumanEval/115] ⚡ FAIL (attempt=5)
5508 [TEST ERROR PREVIEW]
5509 Traceback (most recent call last):
5510   File "C:\Users\JOIN\Downloads\Multi-
      Reviewer_Governance_for_Code_Generation\src\evaluator\
      humaneval_evaluator.py", line 39, in _worker
5511     ns["check"](candidate)
5512   File "<string>", line 13, in check
5513 AssertionError: Error
5514 -----
5515 -----
5515 [HumanEval/115] [REPAIRED CODE] (after attempt=5)
5516 import math
5517
5518 def max_fill(grid, capacity):
5519     total_water = sum(sum(row) for row in grid)
5520     if capacity == 0:
5521         raise ValueError("Capacity cannot be zero")
5522     return math.ceil(total_water / capacity)
5523 -----
5524 -----
5524 [HumanEval/115] Attempt 6/6
5525 [HumanEval/115] ⚡ FAIL (attempt=6)
5526 [TEST ERROR PREVIEW]
5527 Traceback (most recent call last):
5528   File "C:\Users\JOIN\Downloads\Multi-
      Reviewer_Governance_for_Code_Generation\src\evaluator\
      humaneval_evaluator.py", line 39, in _worker
5529     ns["check"](candidate)
5530   File "<string>", line 13, in check
5531 AssertionError: Error
5532 -----
5533 -----
5533 [HumanEval/115] ⚡ FINAL FAIL after 6 retries
5534 =====
5535 =====
5535 [117/164] HumanEval/116
5536 =====
5537 =====
5537 [TASK] HumanEval/116 | entry_point=sort_array
5538 [PROMPT PREVIEW]
5539
5540 def sort_array(arr):
```

```
5541 """
5542     In this Kata, you have to sort an array of non-negative integers
      according to
5543     number of ones in their binary representation in ascending order
5544
5545     For similar number of ones, sort based on decimal value.
5546
5547     It must be implemented like this:
5548     >>> sort_array([1, 5, 2, 3, 4]) == [1, 2, 3, 4, 5]
5549     >>> sort_array([-2, -3, -4, -5, -6]) == [-6, -5, -4, -3, -2...
5549 -----
5550 -----
5550 [INITIAL GENERATED CODE]
5551 def sort_array(arr):
5552     return sorted(arr, key=lambda x: (bin(x).count('1'), x))
5553 -----
5554 -----
5554 [HumanEval/116] Attempt 0/6
5555 [HumanEval/116] ⚡ PASS (attempt=0)
5556 =====
5556 =====
5557 [118/164] HumanEval/117
5558 =====
5558 =====
5559 [TASK] HumanEval/117 | entry_point=select_words
5560 [PROMPT PREVIEW]
5561
5562 def select_words(s, n):
5563     """Given a string s and a natural number n, you have been tasked
      to implement
5564     a function that returns a list of all words from string s that
      contain exactly
5565     n consonants, in order these words appear in the string s.
5566     If the string s is empty then the function should return an
      empty list.
5567     Note: you may assume the input string contains only letters and
      s...
5568 -----
5568 -----
5569 [INITIAL GENERATED CODE]
5570 def select_words(s, n):
5571     words = s.split()
5572     result = []
5573     for word in words:
5574         consonants = sum(1 for char in word if char.lower() not in "
      aeiou" and char.isalpha())
5575         if consonants == n:
5576             result.append(word)
5577     return result
5578 -----
5578 -----
5579 [HumanEval/117] Attempt 0/6
5580 [HumanEval/117] ⚡ PASS (attempt=0)
```

```
5581 =====
5582 =====
5582 [119/164] HumanEval/118
5583 =====
5583 =====
5584 [TASK] HumanEval/118 | entry_point=get_closest_vowel
5585 [PROMPT PREVIEW]
5586
5587 def get_closest_vowel(word):
5588     """You are given a word. Your task is to find the closest vowel
5589     that stands between
5590     two consonants from the right side of the word (case sensitive).
5591     Vowels in the beginning and ending doesn't count. Return empty
5592     string if you didn't
5593     find any vowel met the above condition.
5594
5594     You may assume that the given string contains English letter
5594     only.....
5595 -----
5595 -----
5596 [INITIAL GENERATED CODE]
5597 def get_closest_vowel(word):
5598     vowels = "aeiouAEIOU"
5599     for i in range(len(word) - 2, 0, -1):
5600         if word[i] in vowels and word[i-1] not in vowels and word[i+
5601             1] not in vowels:
5601             return word[i]
5602     return ""
5603 -----
5603 -----
5604 [HumanEval/118] Attempt 0/6
5605 [HumanEval/118] ⚡ PASS (attempt=0)
5606 =====
5606 =====
5607 [120/164] HumanEval/119
5608 =====
5608 =====
5609 [TASK] HumanEval/119 | entry_point=match_parens
5610 [PROMPT PREVIEW]
5611
5612 def match_parens(lst):
5613     '''
5614     You are given a list of two strings, both strings consist of
5614     open
5615     parentheses '(' or close parentheses ')' only.
5616     Your job is to check if it is possible to concatenate the two
5616     strings in
5617     some order, that the resulting string will be good.
5618     A string S is considered to be good if and only if all
5618     parentheses in S
5619     are balanced. For example: the str...
5620 -----
```

```
5620 -----
5621 [INITIAL GENERATED CODE]
5622 def match_parens(lst):
5623     def is_balanced(s):
5624         balance = 0
5625         for char in s:
5626             if char == '(':
5627                 balance += 1
5628             else:
5629                 balance -= 1
5630             if balance < 0:
5631                 return False
5632         return balance == 0
5633
5634     first_order = lst[0] + lst[1]
5635     second_order = lst[1] + lst[0]
5636     return 'Yes' if is_balanced(first_order) or is_balanced(
5637         second_order) else 'No'
5638 -----
5639 [HumanEval/119] Attempt 0/6
5640 =====
5641 [121/164] HumanEval/120
5642 =====
5643 [TASK] HumanEval/120 | entry_point=maximum
5644 [PROMPT PREVIEW]
5645
5646 def maximum(arr, k):
5647     """
5648         Given an array arr of integers and a positive integer k, return
5649         a sorted list
5650         of length k with the maximum k numbers in arr.
5651
5652     Example 1:
5653         Input: arr = [-3, -4, 5], k = 3
5654         Output: [-4, -3, 5]
5655
5656     Example 2:
5657         Input: arr = [4, -4, 4], k = 2
5658         Output: [4, 4]
5659
5660     Example 3:
5661
5662         Input: arr = [-3, 2, 1, 2, -1, -2, 1], k = 1...
5663
5664 -----
5665 [INITIAL GENERATED CODE]
5666 def maximum(arr, k):
```

```
5667     return sorted(arr)[-k:]
5668 -----
5669 [HumanEval/120] Attempt 0/6
5670 [HumanEval/120] ⚡ FAIL (attempt=0)
5671 [TEST ERROR PREVIEW]
5672 Traceback (most recent call last):
5673   File "C:\Users\JOIN\Downloads\Multi-
      Reviewer_Governance_for_Code_Generation\src\evaluator\
        humaneval_evaluator.py", line 39, in _worker
5674     ns["check"](candidate)
5675   File "<string>", line 16, in check
5676 AssertionError
5677 -----
5678 [HumanEval/120] [REPAIRED CODE] (after attempt=0)
5679 def maximum(arr, k):
5680     return sorted(arr, reverse=True)[:k]
5681 -----
5682 [HumanEval/120] Attempt 1/6
5683 [HumanEval/120] ⚡ FAIL (attempt=1)
5684 [TEST ERROR PREVIEW]
5685 Traceback (most recent call last):
5686   File "C:\Users\JOIN\Downloads\Multi-
      Reviewer_Governance_for_Code_Generation\src\evaluator\
        humaneval_evaluator.py", line 39, in _worker
5687     ns["check"](candidate)
5688   File "<string>", line 4, in check
5689 AssertionError
5690 -----
5691 [HumanEval/120] [REPAIRED CODE] (after attempt=1)
5692 def maximum(arr, k):
5693     return sorted(arr, reverse=True)[:k]
5694 -----
5695 [HumanEval/120] Attempt 2/6
5696 [HumanEval/120] ⚡ FAIL (attempt=2)
5697 [TEST ERROR PREVIEW]
5698 Traceback (most recent call last):
5699   File "C:\Users\JOIN\Downloads\Multi-
      Reviewer_Governance_for_Code_Generation\src\evaluator\
        humaneval_evaluator.py", line 39, in _worker
5700     ns["check"](candidate)
5701   File "<string>", line 4, in check
5702 AssertionError
5703 -----
5704 [HumanEval/120] [REPAIRED CODE] (after attempt=2)
5705 def maximum(arr, k):
5706     return sorted(arr, reverse=True)[:k]
5707 -----
```

```
5707 -----
5708 [HumanEval/120] Attempt 3/6
5709 [HumanEval/120] ⚠ FAIL (attempt=3)
5710 [TEST ERROR PREVIEW]
5711 Traceback (most recent call last):
5712   File "C:\Users\JOIN\Downloads\Multi-
      Reviewer_Governance_for_Code_Generation\src\evaluator\
        humaneval_evaluator.py", line 39, in _worker
5713     ns["check"](candidate)
5714   File "<string>", line 4, in check
5715 AssertionError
5716 -----
5717 -----
5717 [HumanEval/120] [REPAIRED CODE] (after attempt=3)
5718 def maximum(arr, k):
5719     return sorted(arr, reverse=True)[:k]
5720 -----
5721 -----
5721 [HumanEval/120] Attempt 4/6
5722 [HumanEval/120] ⚠ FAIL (attempt=4)
5723 [TEST ERROR PREVIEW]
5724 Traceback (most recent call last):
5725   File "C:\Users\JOIN\Downloads\Multi-
      Reviewer_Governance_for_Code_Generation\src\evaluator\
        humaneval_evaluator.py", line 39, in _worker
5726     ns["check"](candidate)
5727   File "<string>", line 4, in check
5728 AssertionError
5729 -----
5730 -----
5730 [HumanEval/120] [REPAIRED CODE] (after attempt=4)
5731 def maximum(arr, k):
5732     return sorted(arr, reverse=True)[:k]
5733 -----
5734 -----
5734 [HumanEval/120] Attempt 5/6
5735 [HumanEval/120] ⚠ FAIL (attempt=5)
5736 [TEST ERROR PREVIEW]
5737 Traceback (most recent call last):
5738   File "C:\Users\JOIN\Downloads\Multi-
      Reviewer_Governance_for_Code_Generation\src\evaluator\
        humaneval_evaluator.py", line 39, in _worker
5739     ns["check"](candidate)
5740   File "<string>", line 4, in check
5741 AssertionError
5742 -----
5743 -----
5743 [HumanEval/120] [REPAIRED CODE] (after attempt=5)
5744 def maximum(arr, k):
5745     return sorted(arr, reverse=True)[:k]
5746 -----
5747 -----
5747 [HumanEval/120] Attempt 6/6
```

```
5748 [HumanEval/120] ⚡ FAIL (attempt=6)
5749 [TEST ERROR PREVIEW]
5750 Traceback (most recent call last):
5751   File "C:\Users\JOIN\Downloads\Multi-
      Reviewer_Governance_for_Code_Generation\src\evaluator\
        humaneval_evaluator.py", line 39, in _worker
5752     ns["check"](candidate)
5753   File "<string>", line 4, in check
5754 AssertionError
5755 -----
-----
5756 [HumanEval/120] ⚡ FINAL FAIL after 6 retries
5757 =====
=====
5758 [122/164] HumanEval/121
5759 =====
=====
5760 [TASK] HumanEval/121 | entry_point=solution
5761 [PROMPT PREVIEW]
5762
5763 def solution(lst):
5764     """Given a non-empty list of integers, return the sum of all of
      the odd elements that are in even positions.
5765
5766
5767     Examples
5768     solution([5, 8, 7, 1]) ==> 12
5769     solution([3, 3, 3, 3, 3]) ==> 9
5770     solution([30, 13, 24, 321]) ==>0
5771     """
5772 -----
-----
5773 [INITIAL GENERATED CODE]
5774 def solution(lst):
5775     return sum(x for i, x in enumerate(lst) if i % 2 == 1 and x % 2
      == 1)
5776 -----
-----
5777 [HumanEval/121] Attempt 0/6
5778 [HumanEval/121] ⚡ FAIL (attempt=0)
5779 [TEST ERROR PREVIEW]
5780 Traceback (most recent call last):
5781   File "C:\Users\JOIN\Downloads\Multi-
      Reviewer_Governance_for_Code_Generation\src\evaluator\
        humaneval_evaluator.py", line 39, in _worker
5782     ns["check"](candidate)
5783   File "<string>", line 4, in check
5784 AssertionError
5785 -----
-----
5786 [HumanEval/121] [REPAIRED CODE] (after attempt=0)
5787 def solution(lst):
5788     return sum(x for i, x in enumerate(lst) if i % 2 != 0 and x % 2
```

```
5788 != 0)
5789 -----
-----
5790 [HumanEval/121] Attempt 1/6
5791 [HumanEval/121] ⚡ FAIL (attempt=1)
5792 [TEST ERROR PREVIEW]
5793 Traceback (most recent call last):
5794   File "C:\Users\JOIN\Downloads\Multi-
      Reviewer_Governance_for_Code_Generation\src\evaluator\
        humaneval_evaluator.py", line 39, in _worker
5795     ns["check"](candidate)
5796   File "<string>", line 4, in check
5797 AssertionError
5798 -----
-----
5799 [HumanEval/121] [REPAIRED CODE] (after attempt=1)
5800 def solution(lst):
5801     return sum(x for i, x in enumerate(lst) if i % 2 == 1 and x % 2
      != 0)
5802 -----
-----
5803 [HumanEval/121] Attempt 2/6
5804 [HumanEval/121] ⚡ FAIL (attempt=2)
5805 [TEST ERROR PREVIEW]
5806 Traceback (most recent call last):
5807   File "C:\Users\JOIN\Downloads\Multi-
      Reviewer_Governance_for_Code_Generation\src\evaluator\
        humaneval_evaluator.py", line 39, in _worker
5808     ns["check"](candidate)
5809   File "<string>", line 4, in check
5810 AssertionError
5811 -----
-----
5812 [HumanEval/121] [REPAIRED CODE] (after attempt=2)
5813 def solution(lst):
5814     return sum(x for i, x in enumerate(lst) if i % 2 == 0 and x % 2
      != 0)
5815 -----
-----
5816 [HumanEval/121] Attempt 3/6
5817 [HumanEval/121] ⚡ PASS (attempt=3)
5818 =====
=====
5819 [123/164] HumanEval/122
5820 =====
=====
5821 [TASK] HumanEval/122 | entry_point=add_elements
5822 [PROMPT PREVIEW]
5823
5824 def add_elements(arr, k):
5825     """
5826         Given a non-empty array of integers arr and an integer k, return
5827         the sum of the elements with at most two digits from the first k
```

```
5827 elements of arr.
5828
5829     Example:
5830
5831         Input: arr = [111,21,3,4000,5,6,7,8,9], k = 4
5832         Output: 24 # sum of 21 + 3
5833
5834     Constraints:
5835         1. 1 <= len(arr) <= 100
5836         2. 1 <= k <= len(arr)
5837     """
5838 -----
5839 [INITIAL GENERATED CODE]
5840 def add_elements(arr, k):
5841     return sum(x for x in arr[:k] if -99 <= x <= 99)
5842 -----
5843 [HumanEval/122] Attempt 0/6
5844 [HumanEval/122] [] PASS (attempt=0)
5845 =====
5846 [124/164] HumanEval/123
5847 =====
5848 [TASK] HumanEval/123 | entry_point=get_odd_collatz
5849 [PROMPT PREVIEW]
5850
5851 def get_odd_collatz(n):
5852     """
5853         Given a positive integer n, return a sorted list that has the
5854         odd numbers in collatz sequence.
5855
5856         The Collatz conjecture is a conjecture in mathematics that
5857         concerns a sequence defined
5858             as follows: start with any positive integer n. Then each term is
5859                 obtained from the
5860                     previous term as follows: if the previous term is even, the next
5861                     term is one half of...
5862
5863 [INITIAL GENERATED CODE]
5864 def get_odd_collatz(n):
5865     def collatz_sequence(num):
5866         sequence = []
5867         while num != 1:
5868             sequence.append(num)
5869             if num % 2 == 0:
5870                 num //= 2
5871             else:
5872                 num = 3 * num + 1
5873         sequence.append(1)
5874     return sequence
```

```
5871
5872     odd_numbers = [num for num in collatz_sequence(n) if num % 2 != 0]
5873     return sorted(odd_numbers)
5874 -----
5875 -----
5875 [HumanEval/123] Attempt 0/6
5876 [HumanEval/123] Ø PASS (attempt=0)
5877 =====
5878 =====
5878 [125/164] HumanEval/124
5879 =====
5880 =====
5880 [TASK] HumanEval/124 | entry_point=valid_date
5881 [PROMPT PREVIEW]
5882
5883 def valid_date(date):
5884     """You have to write a function which validates a given date
5885     string and
5886         returns True if the date is valid otherwise False.
5887         The date is valid if all of the following rules are satisfied:
5888             1. The date string is not empty.
5889             2. The number of days is not less than 1 or higher than 31 days
5890                 for months 1,3,5,7,8,10,12. And the number of days is not less than
5891                 1 or h...
5892 -----
5893 -----
5890 [INITIAL GENERATED CODE]
5891 import re
5892
5893 def valid_date(date):
5894     if not date:
5895         return False
5896
5897     pattern = r"^(0[1-9]|1[0-2])-(0[1-9]|1[2][0-9]|3[01])-\\d{4}$"
5898     if not re.match(pattern, date):
5899         return False
5900
5901     month, day, year = map(int, date.split('-'))
5902
5903     if month == 2 and day > 29:
5904         return False
5905     if month in [4, 6, 9, 11] and day > 30:
5906         return False
5907
5908     return True
5909 -----
5910 -----
5910 [HumanEval/124] Attempt 0/6
5911 [HumanEval/124] Ø PASS (attempt=0)
5912 =====
5913 =====
5913 [126/164] HumanEval/125
```

```
5914 =====
5915 =====
5915 [TASK] HumanEval/125 | entry_point=split_words
5916 [PROMPT PREVIEW]
5917
5918 def split_words(txt):
5919     """
5920         Given a string of words, return a list of words split on
5921         whitespace, if no whitespaces exists in the text you
5922         should split on commas ',' if no commas exists you should return
5923         the number of lower-case letters with odd order in the
5924         alphabet, ord('a') = 0, ord('b') = 1, ... ord('z') = 25
5925         Examples
5926             split_words("Hello world!") → ["Hello", "world!"]...
5925 -----
5925 -----
5926 [INITIAL GENERATED CODE]
5927 def split_words(txt):
5928     if ' ' in txt:
5929         return txt.split()
5930     elif ',' in txt:
5931         return txt.split(',')
5932     else:
5933         return sum(1 for char in txt if char.islower() and (ord(char)
5934 ) - ord('a')) % 2 == 0)
5934 -----
5934 -----
5935 [HumanEval/125] Attempt 0/6
5936 [HumanEval/125] ⚡ FAIL (attempt=0)
5937 [TEST ERROR PREVIEW]
5938 Traceback (most recent call last):
5939   File "C:\Users\JOIN\Downloads\Multi-
      Reviewer_Governance_for_Code_Generation\src\evaluator\
      humaneval_evaluator.py", line 39, in _worker
5940     ns["check"](candidate)
5941   File "<string>", line 8, in check
5942 AssertionError
5943 -----
5943 -----
5944 [HumanEval/125] [REPAIRED CODE] (after attempt=0)
5945 def split_words(txt):
5946     if ' ' in txt:
5947         return txt.split()
5948     elif ',' in txt:
5949         return txt.split(',')
5950     else:
5951         return [char for char in txt if char.islower() and (ord(char)
5952 ) - ord('a')) % 2 == 0]
5952 -----
5952 -----
5953 [HumanEval/125] Attempt 1/6
5954 [HumanEval/125] ⚡ FAIL (attempt=1)
5955 [TEST ERROR PREVIEW]
```

```
5956 Traceback (most recent call last):
5957   File "C:\Users\JOIN\Downloads\Multi-
      Reviewer_Governance_for_Code_Generation\src\evaluator\
        humaneval_evaluator.py", line 39, in _worker
5958     ns["check"](candidate)
5959   File "<string>", line 7, in check
5960 AssertionError
5961 -----
5962 [HumanEval/125] [REPAIRED CODE] (after attempt=1)
5963 def split_words(txt):
5964     if ' ' in txt:
5965         return txt.split()
5966     elif ',' in txt:
5967         return txt.split(',')
5968     else:
5969         return [char for char in txt if char.islower() and (ord(char)
5969 ) - ord('a')) % 2 == 0]
5970 -----
5971 [HumanEval/125] Attempt 2/6
5972 [HumanEval/125] Ø FAIL (attempt=2)
5973 [TEST ERROR PREVIEW]
5974 Traceback (most recent call last):
5975   File "C:\Users\JOIN\Downloads\Multi-
      Reviewer_Governance_for_Code_Generation\src\evaluator\
        humaneval_evaluator.py", line 39, in _worker
5976     ns["check"](candidate)
5977   File "<string>", line 7, in check
5978 AssertionError
5979 -----
5980 [HumanEval/125] [REPAIRED CODE] (after attempt=2)
5981 def split_words(txt):
5982     if ' ' in txt:
5983         return txt.split()
5984     elif ',' in txt:
5985         return txt.split(',')
5986     else:
5987         return [char for char in txt if char.islower() and (ord(char)
5987 ) - ord('a')) % 2 == 0]
5988 -----
5989 [HumanEval/125] Attempt 3/6
5990 [HumanEval/125] Ø FAIL (attempt=3)
5991 [TEST ERROR PREVIEW]
5992 Traceback (most recent call last):
5993   File "C:\Users\JOIN\Downloads\Multi-
      Reviewer_Governance_for_Code_Generation\src\evaluator\
        humaneval_evaluator.py", line 39, in _worker
5994     ns["check"](candidate)
5995   File "<string>", line 7, in check
5996 AssertionError
```

```
5997 -----
-----
5998 [HumanEval/125] [REPAIRED CODE] (after attempt=3)
5999 def split_words(txt):
6000     if ' ' in txt:
6001         return txt.split()
6002     elif ',' in txt:
6003         return txt.split(',')
6004     else:
6005         return [char for char in txt if char.islower() and (ord(char)
6006 ) - ord('a')) % 2 == 0]
6006 -----
-----
6007 [HumanEval/125] Attempt 4/6
6008 [HumanEval/125] ⚡ FAIL (attempt=4)
6009 [TEST ERROR PREVIEW]
6010 Traceback (most recent call last):
6011   File "C:\Users\JOIN\Downloads\Multi-
6012     Reviewer_Governance_for_Code_Generation\src\evaluator\
6013       humaneval_evaluator.py", line 39, in _worker
6014     ns["check"](candidate)
6015   File "<string>", line 7, in check
6016 Assertion
6017 -----
6018 -----
6019 [HumanEval/125] [REPAIRED CODE] (after attempt=4)
6020 def split_words(txt):
6021     if ' ' in txt:
6022         return txt.split()
6023     elif ',' in txt:
6024         return txt.split(',')
6025     else:
6026         return [char for char in txt if char.islower() and (ord(char)
6027 ) - ord('a')) % 2 == 0]
6028 -----
6029 -----
6030 [HumanEval/125] Attempt 5/6
6031 [HumanEval/125] ⚡ FAIL (attempt=5)
6032 [TEST ERROR PREVIEW]
6033 Traceback (most recent call last):
6034   File "C:\Users\JOIN\Downloads\Multi-
6035     Reviewer_Governance_for_Code_Generation\src\evaluator\
6036       humaneval_evaluator.py", line 39, in _worker
6037     ns["check"](candidate)
6038   File "<string>", line 7, in check
6039 Assertion
6040 -----
6041 -----
6042 [HumanEval/125] [REPAIRED CODE] (after attempt=5)
6043 def split_words(txt):
6044     if ' ' in txt:
6045         return txt.split()
6046     elif ',' in txt:
```

```
6039         return txt.split(',')
6040     else:
6041         return [char for char in txt if char.islower() and (ord(char)
6042 ) - ord('a')) % 2 == 0]
6042 -----
6043 -----
6043 [HumanEval/125] Attempt 6/6
6044 [HumanEval/125] ⚡ FAIL (attempt=6)
6045 [TEST ERROR PREVIEW]
6046 Traceback (most recent call last):
6047   File "C:\Users\JOIN\Downloads\Multi-
6048     Reviewer_Governance_for_Code_Generation\src\evaluator\
6049     humaneval_evaluator.py", line 39, in _worker
6048       ns["check"](candidate)
6049     File "<string>", line 7, in check
6050   AssertionError
6051 -----
6052 -----
6052 [HumanEval/125] ⚡ FINAL FAIL after 6 retries
6053 =====
6054 [127/164] HumanEval/126
6055 =====
6056 [TASK] HumanEval/126 | entry_point=is_sorted
6057 [PROMPT PREVIEW]
6058
6059 def is_sorted(lst):
6060     ''
6061     Given a list of numbers, return whether or not they are sorted
6062     in ascending order. If list has more than 1 duplicate of the
6063     same
6064     number, return False. Assume no negative numbers and only
6065     integers.
6066
6067     Examples
6068     is_sorted([5]) → True
6069     is_sorted([1, 2, 3, 4, 5]) → True
6070     is_sorted([1, 3, 2, 4, 5]) → False
6071     is_sorted([1, 2, 3, 4, 5, 6]) → True...
6072 -----
6073 -----
6071 [INITIAL GENERATED CODE]
6072 def is_sorted(lst):
6073     if len(lst) <= 1:
6074         return True
6075     for i in range(1, len(lst)):
6076         if lst[i] < lst[i - 1]:
6077             return False
6078         if i >= 2 and lst[i] == lst[i - 1] == lst[i - 2]:
6079             return False
6080     return True
6081 -----
```

```
6081 -----
6082 [HumanEval/126] Attempt 0/6
6083 [HumanEval/126] ✘ PASS (attempt=0)
6084 =====
6085 [128/164] HumanEval/127
6086 =====
6087 [TASK] HumanEval/127 | entry_point=intersection
6088 [PROMPT PREVIEW]
6089
6090 def intersection(interval1, interval2):
6091     """You are given two intervals,
6092         where each interval is a pair of integers. For example, interval
6093         = (start, end) = (1, 2).
6094         The given intervals are closed which means that the interval (
6095         start, end)
6096             includes both start and end.
6097             For each given interval, it is assumed that its start is less or
6098             equal its end.
6099             Your task is to determine whet...
6100 -----
6101 -----
6102 [INITIAL GENERATED CODE]
6103 def is_prime(n):
6104     if n <= 1:
6105         return False
6106     for i in range(2, int(n**0.5) + 1):
6107         if n % i == 0:
6108             return False
6109     return True
6110
6111 def intersection(interval1, interval2):
6112     start = max(interval1[0], interval2[0])
6113     end = min(interval1[1], interval2[1])
6114     if start > end:
6115         return "NO"
6116     length = end - start + 1
6117     return "YES" if is_prime(length) else "NO"
6118 -----
6119 -----
6120 [HumanEval/127] Attempt 0/6
6121 [HumanEval/127] ✘ FAIL (attempt=0)
6122 [TEST ERROR PREVIEW]
6123 Traceback (most recent call last):
6124   File "C:\Users\JOIN\Downloads\Multi-
6125       Reviewer_Governance_for_Code_Generation\src\evaluator\
6126           humaneval_evaluator.py", line 39, in _worker
6127             ns["check"](candidate)
6128   File "<string>", line 5, in check
6129 Assertion
6130 -----
6131 -----
```

```
6124 [HumanEval/127] [REPAIRED CODE] (after attempt=0)
6125 def is_prime(n):
6126     if n <= 1:
6127         return False
6128     for i in range(2, int(n**0.5) + 1):
6129         if n % i == 0:
6130             return False
6131     return True
6132
6133 def intersection(interval1, interval2):
6134     start = max(interval1[0], interval2[0])
6135     end = min(interval1[1], interval2[1])
6136     if start > end:
6137         return "NO"
6138     length = end - start + 1
6139     return "YES" if is_prime(length) else "NO"
6140 -----
-----
6141 [HumanEval/127] Attempt 1/6
6142 [HumanEval/127] ⚡ FAIL (attempt=1)
6143 [TEST ERROR PREVIEW]
6144 Traceback (most recent call last):
6145   File "C:\Users\JOIN\Downloads\Multi-
      Reviewer_Governance_for_Code_Generation\src\evaluator\
      humaneval_evaluator.py", line 39, in _worker
6146     ns["check"](candidate)
6147   File "<string>", line 5, in check
6148 AssertionError
6149 -----
-----
6150 [HumanEval/127] [REPAIRED CODE] (after attempt=1)
6151 def is_prime(n):
6152     if n <= 1:
6153         return False
6154     for i in range(2, int(n**0.5) + 1):
6155         if n % i == 0:
6156             return False
6157     return True
6158
6159 def intersection(interval1, interval2):
6160     start = max(interval1[0], interval2[0])
6161     end = min(interval1[1], interval2[1])
6162     if start > end:
6163         return "NO"
6164     length = end - start + 1
6165     return "YES" if is_prime(length) else "NO"
6166 -----
-----
6167 [HumanEval/127] Attempt 2/6
6168 [HumanEval/127] ⚡ FAIL (attempt=2)
6169 [TEST ERROR PREVIEW]
6170 Traceback (most recent call last):
6171   File "C:\Users\JOIN\Downloads\Multi-
```

```
6171 Reviewer_Governance_for_Code_Generation\src\evaluator\  
    humaneval_evaluator.py", line 39, in _worker  
6172     ns["check"]的文化  
6173     File "<string>", line 5, in check  
6174 AssertionError  
6175 -----  
-----  
6176 [HumanEval/127] [REPAIRED CODE] (after attempt=2)  
6177 def is_prime(n):  
6178     if n <= 1:  
6179         return False  
6180     for i in range(2, int(n**0.5) + 1):  
6181         if n % i == 0:  
6182             return False  
6183     return True  
6184  
6185 def intersection(interval1, interval2):  
6186     start = max(interval1[0], interval2[0])  
6187     end = min(interval1[1], interval2[1])  
6188     if start > end:  
6189         return "NO"  
6190     length = end - start + 1  
6191     return "YES" if is_prime(length) else "NO"  
6192 -----  
-----  
6193 [HumanEval/127] Attempt 3/6  
6194 [HumanEval/127] ⚡ FAIL (attempt=3)  
6195 [TEST ERROR PREVIEW]  
6196 Traceback (most recent call last):  
6197     File "C:\Users\JOIN\Downloads\Multi-  
        Reviewer_Governance_for_Code_Generation\src\evaluator\  
        humaneval_evaluator.py", line 39, in _worker  
6198     ns["check"]的文化  
6199     File "<string>", line 5, in check  
6200 AssertionError  
6201 -----  
-----  
6202 [HumanEval/127] [REPAIRED CODE] (after attempt=3)  
6203 def is_prime(n):  
6204     if n <= 1:  
6205         return False  
6206     for i in range(2, int(n**0.5) + 1):  
6207         if n % i == 0:  
6208             return False  
6209     return True  
6210  
6211 def intersection(interval1, interval2):  
6212     start = max(interval1[0], interval2[0])  
6213     end = min(interval1[1], interval2[1])  
6214     if start > end:  
6215         return "NO"  
6216     length = end - start + 1  
6217     return "YES" if is_prime(length) else "NO"
```

```
6218 -----
-----
6219 [HumanEval/127] Attempt 4/6
6220 [HumanEval/127] ⚡ FAIL (attempt=4)
6221 [TEST ERROR PREVIEW]
6222 Traceback (most recent call last):
6223   File "C:\Users\JOIN\Downloads\Multi-
Reviewer_Governance_for_Code_Generation\src\evaluator\
humaneval_evaluator.py", line 39, in _worker
6224     ns["check"](candidate)
6225   File "<string>", line 5, in check
6226 AssertionError
6227 -----
-----
6228 [HumanEval/127] [REPAIRED CODE] (after attempt=4)
6229 def is_prime(n):
6230     if n <= 1:
6231         return False
6232     for i in range(2, int(n**0.5) + 1):
6233         if n % i == 0:
6234             return False
6235     return True
6236
6237 def intersection(interval1, interval2):
6238     start = max(interval1[0], interval2[0])
6239     end = min(interval1[1], interval2[1])
6240     if start > end:
6241         return "NO"
6242     length = end - start + 1
6243     return "YES" if is_prime(length) else "NO"
6244 -----
-----
6245 [HumanEval/127] Attempt 5/6
6246 [HumanEval/127] ⚡ FAIL (attempt=5)
6247 [TEST ERROR PREVIEW]
6248 Traceback (most recent call last):
6249   File "C:\Users\JOIN\Downloads\Multi-
Reviewer_Governance_for_Code_Generation\src\evaluator\
humaneval_evaluator.py", line 39, in _worker
6250     ns["check"](candidate)
6251   File "<string>", line 5, in check
6252 AssertionError
6253 -----
-----
6254 [HumanEval/127] [REPAIRED CODE] (after attempt=5)
6255 def is_prime(n):
6256     if n <= 1:
6257         return False
6258     for i in range(2, int(n**0.5) + 1):
6259         if n % i == 0:
6260             return False
6261     return True
6262
```

```
6263 def intersection(interval1, interval2):  
6264     start = max(interval1[0], interval2[0])  
6265     end = min(interval1[1], interval2[1])  
6266     if start > end:  
6267         return "NO"  
6268     length = end - start + 1  
6269     return "YES" if is_prime(length) else "NO"  
6270 -----  
-----  
6271 [HumanEval/127] Attempt 6/6  
6272 [HumanEval/127] ⚡ FAIL (attempt=6)  
6273 [TEST ERROR PREVIEW]  
6274 Traceback (most recent call last):  
6275   File "C:\Users\JOIN\Downloads\Multi-  
     Reviewer_Governance_for_Code_Generation\src\evaluator\  
     humaneval_evaluator.py", line 39, in _worker  
6276     ns["check"](candidate)  
6277   File "<string>", line 5, in check  
6278 AssertionException  
6279 -----  
-----  
6280 [HumanEval/127] ⚡ FINAL FAIL after 6 retries  
6281 ======  
=====  
6282 [129/164] HumanEval/128  
6283 ======  
=====  
6284 [TASK] HumanEval/128 | entry_point=prod_signs  
6285 [PROMPT PREVIEW]  
6286  
6287 def prod_signs(arr):  
6288     """  
6289         You are given an array arr of integers and you need to return  
6290         sum of magnitudes of integers multiplied by product of all signs  
6291         of each number in the array, represented by 1, -1 or 0.  
6292         Note: return None for empty arr.  
6293  
6294         Example:  
6295         >>> prod_signs([1, 2, 2, -4]) == -9  
6296         >>> prod_signs([0, 1]) == 0  
6297         >>> prod_signs([]) == None  
6298         """  
6299 -----  
-----  
6300 [INITIAL GENERATED CODE]  
6301 def prod_signs(arr):  
6302     if not arr:  
6303         return None  
6304  
6305     product_of_signs = 1  
6306     sum_of_magnitudes = 0  
6307  
6308     for num in arr:
```

```

6309     if num == 0:
6310         return 0
6311     elif num < 0:
6312         product_of_signs *= -1
6313
6314     sum_of_magnitudes += abs(num)
6315
6316     return sum_of_magnitudes * product_of_signs
6317 -----
6318 -----
6318 [HumanEval/128] Attempt 0/6
6319 [HumanEval/128] [] PASS (attempt=0)
6320 =====
6321 =====
6321 [130/164] HumanEval/129
6322 =====
6323 =====
6323 [TASK] HumanEval/129 | entry_point=minPath
6324 [PROMPT PREVIEW]
6325
6326 def minPath(grid, k):
6327     """
6328     Given a grid with N rows and N columns (N >= 2) and a positive
6329     integer k,
6330     each cell of the grid contains a value. Every integer in the
6331     range [1, N * N]
6332     inclusive appears exactly once on the cells of the grid.
6333
6334     You have to find the minimum path of length k in the grid. You
6335     can start
6336     from any cell, and in each step you can move to any of the
6337     neigbh...
6338 -----
6339 -----
6335 [INITIAL GENERATED CODE]
6336 def minPath(grid, k):
6337     N = len(grid)
6338     directions = [(0, 1), (1, 0), (-1, 0), (0, -1)]
6339
6340     def dfs(x, y, path):
6341         if len(path) == k:
6342             return path
6343         min_path = None
6344         for dx, dy in directions:
6345             nx, ny = x + dx, y + dy
6346             if 0 <= nx < N and 0 <= ny < N:
6347                 new_path = dfs(nx, ny, path + [grid[nx][ny]])
6348                 if min_path is None or (new_path < min_path):
6349                     min_path = new_path
6350         return min_path
6351
6352     min_path = None
6353     for i in range(N):

```

```
6354         for j in range(N):
6355             path = dfs(i, j, [grid[i][j]])
6356             if min_path is None or (path < min_path):
6357                 min_path = path
6358     return min_path
6359 -----
6360 -----
6360 [HumanEval/129] Attempt 0/6
6361 [HumanEval/129] [] PASS (attempt=0)
6362 =====
6363 -----
6363 [131/164] HumanEval/130
6364 =====
6365 -----
6365 [TASK] HumanEval/130 | entry_point=tri
6366 [PROMPT PREVIEW]
6367
6368 def tri(n):
6369     """Everyone knows Fibonacci sequence, it was studied deeply by
6370     mathematicians in
6370     the last couple centuries. However, what people don't know is
6370     Tribonacci sequence.
6371     Tribonacci sequence is defined by the recurrence:
6372     tri(1) = 3
6373     tri(n) = 1 + n / 2, if n is even.
6374     tri(n) = tri(n - 1) + tri(n - 2) + tri(n + 1), if n is odd.
6375     For example:
6376     tri(2) = 1 + (2 / 2)...
6377 -----
6378 -----
6378 [INITIAL GENERATED CODE]
6379 def tri(n):
6380     if n == 0:
6381         return [1]
6382     elif n == 1:
6383         return [1, 3]
6384
6385     result = [1, 3, 2]
6386     for i in range(3, n + 1):
6387         if i % 2 == 0:
6388             result.append(1 + i / 2)
6389         else:
6390             result.append(result[i - 1] + result[i - 2] + result[i
+ 1])
6391
6392     return result
6393 -----
6394 -----
6394 [HumanEval/130] Attempt 0/6
6395 [HumanEval/130] [] FAIL (attempt=0)
6396 [TEST ERROR PREVIEW]
6397 Traceback (most recent call last):
6398   File "C:\Users\JOIN\Downloads\Multi-
```

```
6398 Reviewer_Governance_for_Code_Generation\src\evaluator\  
    humaneval_evaluator.py", line 39, in _worker  
6399     ns["check"](candidate)  
6400     File "<string>", line 5, in check  
6401     File "<string>", line 31, in tri  
6402 IndexError: list index out of range  
6403 -----  
-----  
6404 [HumanEval/130] [REPAIRED CODE] (after attempt=0)  
6405 def tri(n):  
6406     if n == 0:  
6407         return [1]  
6408     elif n == 1:  
6409         return [1, 3]  
6410  
6411     result = [1, 3, 2]  
6412     for i in range(3, n + 1):  
6413         if i % 2 == 0:  
6414             result.append(1 + i / 2)  
6415         else:  
6416             result.append(result[i - 1] + result[i - 2])  
6417  
6418     return result  
6419 -----  
-----  
6420 [HumanEval/130] Attempt 1/6  
6421 [HumanEval/130] ⚡ FAIL (attempt=1)  
6422 [TEST ERROR PREVIEW]  
6423 Traceback (most recent call last):  
6424     File "C:\Users\JOIN\Downloads\Multi-  
        Reviewer_Governance_for_Code_Generation\src\evaluator\  
        humaneval_evaluator.py", line 39, in _worker  
6425     ns["check"](candidate)  
6426     File "<string>", line 5, in check  
6427 AssertionError  
6428 -----  
-----  
6429 [HumanEval/130] [REPAIRED CODE] (after attempt=1)  
6430 def tri(n):  
6431     if n == 0:  
6432         return [1]  
6433     elif n == 1:  
6434         return [1, 3]  
6435  
6436     result = [1, 3, 2]  
6437     for i in range(3, n + 1):  
6438         if i % 2 == 0:  
6439             result.append(1 + i // 2)  
6440         else:  
6441             result.append(result[i - 1] + result[i - 2])  
6442  
6443     return result  
6444 -----
```

```
6444 -----
6445 [HumanEval/130] Attempt 2/6
6446 [HumanEval/130] ⚡ FAIL (attempt=2)
6447 [TEST ERROR PREVIEW]
6448 Traceback (most recent call last):
6449   File "C:\Users\JOIN\Downloads\Multi-
       Reviewer_Governance_for_Code_Generation\src\evaluator\
         humaneval_evaluator.py", line 39, in _worker
6450     ns["check"](candidate)
6451   File "<string>", line 5, in check
6452 AssertionError
6453 -----
6454 -----
6455 [HumanEval/130] [REPAIRED CODE] (after attempt=2)
6456 def tri(n):
6457     if n == 0:
6458         return [1]
6459     elif n == 1:
6460         return [1, 3]
6461     result = [1, 3, 2]
6462     for i in range(3, n + 1):
6463         if i % 2 == 0:
6464             result.append(1 + i // 2)
6465         else:
6466             result.append(result[i - 1] + result[i - 2])
6467
6468     return result
6469 -----
6470 -----
6471 [HumanEval/130] Attempt 3/6
6472 [HumanEval/130] ⚡ FAIL (attempt=3)
6473 [TEST ERROR PREVIEW]
6474 Traceback (most recent call last):
6475   File "C:\Users\JOIN\Downloads\Multi-
       Reviewer_Governance_for_Code_Generation\src\evaluator\
         humaneval_evaluator.py", line 39, in _worker
6476     ns["check"](candidate)
6477   File "<string>", line 5, in check
6478 AssertionError
6479 -----
6480 -----
6481 [HumanEval/130] [REPAIRED CODE] (after attempt=3)
6482 def tri(n):
6483     if n == 0:
6484         return [1]
6485     elif n == 1:
6486         return [1, 3]
6487     result = [1, 3, 2]
6488     for i in range(3, n + 1):
6489         if i % 2 == 0:
6490             result.append(1 + i // 2)
```

```
6490     else:
6491         result.append(result[i - 1] + result[i - 2])
6492
6493     return result
6494 -----
6495 [HumanEval/130] Attempt 4/6
6496 [HumanEval/130] ⚡ FAIL (attempt=4)
6497 [TEST ERROR PREVIEW]
6498 Traceback (most recent call last):
6499   File "C:\Users\JOIN\Downloads\Multi-
    Reviewer_Governance_for_Code_Generation\src\evaluator\
      humaneval_evaluator.py", line 39, in _worker
6500     ns["check"](candidate)
6501   File "<string>", line 5, in check
6502 AssertionError
6503 -----
6504 [HumanEval/130] [REPAIRED CODE] (after attempt=4)
6505 def tri(n):
6506     if n == 0:
6507         return [1]
6508     elif n == 1:
6509         return [1, 3]
6510
6511     result = [1, 3, 2]
6512     for i in range(3, n + 1):
6513         if i % 2 == 0:
6514             result.append(1 + i // 2)
6515         else:
6516             result.append(result[i - 1] + result[i - 2])
6517
6518     return result
6519 -----
6520 [HumanEval/130] Attempt 5/6
6521 [HumanEval/130] ⚡ FAIL (attempt=5)
6522 [TEST ERROR PREVIEW]
6523 Traceback (most recent call last):
6524   File "C:\Users\JOIN\Downloads\Multi-
    Reviewer_Governance_for_Code_Generation\src\evaluator\
      humaneval_evaluator.py", line 39, in _worker
6525     ns["check"](candidate)
6526   File "<string>", line 5, in check
6527 AssertionError
6528 -----
6529 [HumanEval/130] [REPAIRED CODE] (after attempt=5)
6530 def tri(n):
6531     if n == 0:
6532         return [1]
6533     elif n == 1:
6534         return [1, 3]
```

```
6535
6536     result = [1, 3, 2]
6537     for i in range(3, n + 1):
6538         if i % 2 == 0:
6539             result.append(1 + i // 2)
6540         else:
6541             result.append(result[i - 1] + result[i - 2])
6542
6543     return result
6544 -----
-----
6545 [HumanEval/130] Attempt 6/6
6546 [HumanEval/130] ⚡ FAIL (attempt=6)
6547 [TEST ERROR PREVIEW]
6548 Traceback (most recent call last):
6549   File "C:\Users\JOIN\Downloads\Multi-
Reviewer_Governance_for_Code_Generation\src\evaluator\
humaneval_evaluator.py", line 39, in _worker
6550     ns["check"](candidate)
6551   File "<string>", line 5, in check
6552 AssertionError
6553 -----
-----
6554 [HumanEval/130] ⚡ FINAL FAIL after 6 retries
6555 =====
=====
6556 [132/164] HumanEval/131
6557 =====
=====
6558 [TASK] HumanEval/131 | entry_point=digits
6559 [PROMPT PREVIEW]
6560
6561 def digits(n):
6562     """Given a positive integer n, return the product of the odd
       digits.
6563     Return 0 if all digits are even.
6564     For example:
6565     digits(1) == 1
6566     digits(4) == 0
6567     digits(235) == 15
6568     """
6569 -----
-----
6570 [INITIAL GENERATED CODE]
6571 def digits(n):
6572     product = 1
6573     has_odd = False
6574     for digit in str(n):
6575         if int(digit) % 2 != 0:
6576             product *= int(digit)
6577             has_odd = True
6578     return product if has_odd else 0
6579 -----
```

```
6579 -----
6580 [HumanEval/131] Attempt 0/6
6581 [HumanEval/131] ✘ PASS (attempt=0)
6582 =====
6583 [133/164] HumanEval/132
6584 =====
6585 [TASK] HumanEval/132 | entry_point=is_nested
6586 [PROMPT PREVIEW]
6587
6588 def is_nested(string):
6589     ''
6590     Create a function that takes a string as input which contains
6591     only square brackets.
6592     The function should return True if and only if there is a valid
6593     subsequence of brackets
6594     where at least one bracket in the subsequence is nested.
6595
6596     is_nested('[]') → True
6597     is_nested('[[[]]]])[[[[[]')) → False
6598     is_nested('[][]) → False
6599     is_nested('[]') → Fals...
6600 -----
6601 -----
6602 [INITIAL GENERATED CODE]
6603 def is_nested(string):
6604     stack = []
6605     nested = False
6606     for char in string:
6607         if char == '[':
6608             stack.append(char)
6609         elif char == ']' and stack:
6610             stack.pop()
6611             if stack:
6612                 nested = True
6613     return nested and not stack
6614 -----
6615 -----
6616 [HumanEval/132] Attempt 0/6
6617 [HumanEval/132] ✘ FAIL (attempt=0)
6618 [TEST ERROR PREVIEW]
6619 Traceback (most recent call last):
6620   File "C:\Users\JOIN\Downloads\Multi-
6621     Reviewer_Governance_for_Code_Generation\src\evaluator\
6622       humaneval_evaluator.py", line 39, in _worker
6623         ns["check"](candidate)
6624       File "<string>", line 13, in check
6625       AssertionError
6626 -----
6627 -----
6628 [HumanEval/132] [REPAIRED CODE] (after attempt=0)
6629 def is_nested(string):
```

```
6623     stack = []
6624     nested = False
6625     for char in string:
6626         if char == '[':
6627             stack.append(char)
6628         elif char == ']' and stack:
6629             stack.pop()
6630         if stack:
6631             nested = True
6632     return nested and len(stack) == 0
6633 -----
-----
6634 [HumanEval/132] Attempt 1/6
6635 [HumanEval/132] [] FAIL (attempt=1)
6636 [TEST ERROR PREVIEW]
6637 Traceback (most recent call last):
6638   File "C:\Users\JOIN\Downloads\Multi-
Reviewer_Governance_for_Code_Generation\src\evaluator\
humaneval_evaluator.py", line 39, in _worker
6639     ns["check"](candidate)
6640   File "<string>", line 13, in check
6641 AssertionError
6642 -----
-----
6643 [HumanEval/132] [REPAIRED CODE] (after attempt=1)
6644 def is_nested(string):
6645     stack = []
6646     nested = False
6647     for char in string:
6648         if char == '[':
6649             stack.append(char)
6650         elif char == ']' and stack:
6651             stack.pop()
6652         if stack:
6653             nested = True
6654     return nested and len(stack) == 0
6655 -----
-----
6656 [HumanEval/132] Attempt 2/6
6657 [HumanEval/132] [] FAIL (attempt=2)
6658 [TEST ERROR PREVIEW]
6659 Traceback (most recent call last):
6660   File "C:\Users\JOIN\Downloads\Multi-
Reviewer_Governance_for_Code_Generation\src\evaluator\
humaneval_evaluator.py", line 39, in _worker
6661     ns["check"](candidate)
6662   File "<string>", line 13, in check
6663 AssertionError
6664 -----
-----
6665 [HumanEval/132] [REPAIRED CODE] (after attempt=2)
6666 def is_nested(string):
6667     stack = []
```

```
6668     nested = False
6669     for char in string:
6670         if char == '[':
6671             stack.append(char)
6672         elif char == ']' and stack:
6673             stack.pop()
6674         if stack:
6675             nested = True
6676     return nested and len(stack) == 0
6677 -----
6678 -----
6678 [HumanEval/132] Attempt 3/6
6679 [HumanEval/132] ⚡ FAIL (attempt=3)
6680 [TEST ERROR PREVIEW]
6681 Traceback (most recent call last):
6682   File "C:\Users\JOIN\Downloads\Multi-
6683     Reviewer_Governance_for_Code_Generation\src\evaluator\
6684       humaneval_evaluator.py", line 39, in _worker
6685         ns["check"](candidate)
6686   File "<string>", line 13, in check
6687   AssertionError
6688 -----
6689 -----
6687 [HumanEval/132] [REPAIRED CODE] (after attempt=3)
6688 def is_nested(string):
6689     stack = []
6690     nested = False
6691     for char in string:
6692         if char == '[':
6693             stack.append(char)
6694         elif char == ']' and stack:
6695             stack.pop()
6696         if stack:
6697             nested = True
6698     return nested and len(stack) == 0
6699 -----
6700 -----
6700 [HumanEval/132] Attempt 4/6
6701 [HumanEval/132] ⚡ FAIL (attempt=4)
6702 [TEST ERROR PREVIEW]
6703 Traceback (most recent call last):
6704   File "C:\Users\JOIN\Downloads\Multi-
6705     Reviewer_Governance_for_Code_Generation\src\evaluator\
6706       humaneval_evaluator.py", line 39, in _worker
6707         ns["check"](candidate)
6708   File "<string>", line 13, in check
6709   AssertionError
6710 -----
6711 -----
6709 [HumanEval/132] [REPAIRED CODE] (after attempt=4)
6710 def is_nested(string):
6711     stack = []
6712     nested = False
```

```
6713     for char in string:
6714         if char == '[':
6715             stack.append(char)
6716         elif char == ']' and stack:
6717             stack.pop()
6718             if stack:
6719                 nested = True
6720     return nested and len(stack) == 0
6721 -----
6722 -----
6722 [HumanEval/132] Attempt 5/6
6723 [HumanEval/132] ⚡ FAIL (attempt=5)
6724 [TEST ERROR PREVIEW]
6725 Traceback (most recent call last):
6726   File "C:\Users\JOIN\Downloads\Multi-
Reviewer_Governance_for_Code_Generation\src\evaluator\
humaneval_evaluator.py", line 39, in _worker
6727     ns["check"](candidate)
6728   File "<string>", line 13, in check
6729 AssertionError
6730 -----
6731 -----
6731 [HumanEval/132] [REPAIRED CODE] (after attempt=5)
6732 def is_nested(string):
6733     stack = []
6734     nested = False
6735     for char in string:
6736         if char == '[':
6737             stack.append(char)
6738         elif char == ']' and stack:
6739             stack.pop()
6740             if stack:
6741                 nested = True
6742     return nested and len(stack) == 0
6743 -----
6744 -----
6744 [HumanEval/132] Attempt 6/6
6745 [HumanEval/132] ⚡ FAIL (attempt=6)
6746 [TEST ERROR PREVIEW]
6747 Traceback (most recent call last):
6748   File "C:\Users\JOIN\Downloads\Multi-
Reviewer_Governance_for_Code_Generation\src\evaluator\
humaneval_evaluator.py", line 39, in _worker
6749     ns["check"](candidate)
6750   File "<string>", line 13, in check
6751 AssertionError
6752 -----
6753 -----
6753 [HumanEval/132] ⚡ FINAL FAIL after 6 retries
6754 =====
6755 =====
6755 [134/164] HumanEval/133
6756 =====
```

```
6756 =====
6757 [TASK] HumanEval/133 | entry_point=sum_squares
6758 [PROMPT PREVIEW]
6759
6760
6761 def sum_squares(lst):
6762     """You are given a list of numbers.
6763         You need to return the sum of squared numbers in the given list,
6764         round each element in the list to the upper int(Ceiling) first.
6765         Examples:
6766             For lst = [1,2,3] the output should be 14
6767             For lst = [1,4,9] the output should be 98
6768             For lst = [1,3,5,7] the output should be 84
6769             For lst = [1.4,4.2,0] the output should b...
6770 -----
6771 -----
6771 [INITIAL GENERATED CODE]
6772 import math
6773
6774 def sum_squares(lst):
6775     return sum(math.ceil(x) ** 2 for x in lst)
6776 -----
6777 -----
6777 [HumanEval/133] Attempt 0/6
6778 [HumanEval/133] [] PASS (attempt=0)
6779 =====
6780 -----
6780 [135/164] HumanEval/134
6781 =====
6782 -----
6782 [TASK] HumanEval/134 | entry_point=check_if_last_char_is_a_letter
6783 [PROMPT PREVIEW]
6784
6785 def check_if_last_char_is_a_letter(txt):
6786     '''
6787         Create a function that returns True if the last character
6788         of a given string is an alphabetical character and is not
6789         a part of a word, and False otherwise.
6790         Note: "word" is a group of characters separated by space.
6791
6792         Examples:
6793             check_if_last_char_is_a_letter("apple pie") → False
6794             check_if_last_char_is_a_letter("apple pi e") →...
6795 -----
6796 -----
6796 [INITIAL GENERATED CODE]
6797 def check_if_last_char_is_a_letter(txt):
6798     if not txt or not txt[-1].isalpha():
6799         return False
6800     if len(txt) == 1:
6801         return True
6802     return txt[-2] == ' '
6803 -----
```

```
6803 -----
6804 [HumanEval/134] Attempt 0/6
6805 [HumanEval/134] Ø PASS (attempt=0)
6806 =====
6807 =====
6807 [136/164] HumanEval/135
6808 =====
6809 =====
6809 [TASK] HumanEval/135 | entry_point=can_arrange
6810 [PROMPT PREVIEW]
6811
6812 def can_arrange(arr):
6813     """Create a function which returns the largest index of an
6814     element which
6814     is not greater than or equal to the element immediately
6814     preceding it. If
6815     no such element exists then return -1. The given array will not
6815     contain
6816     duplicate values.
6817
6818     Examples:
6819     can_arrange([1,2,4,3,5]) = 3
6820     can_arrange([1,2,3]) = -1
6821     """
6822 -----
6823 -----
6823 [INITIAL GENERATED CODE]
6824 def can_arrange(arr):
6825     for i in range(len(arr) - 1, 0, -1):
6826         if arr[i] < arr[i - 1]:
6827             return i
6828     return -1
6829 -----
6830 -----
6830 [HumanEval/135] Attempt 0/6
6831 [HumanEval/135] Ø PASS (attempt=0)
6832 =====
6832 =====
6833 [137/164] HumanEval/136
6834 =====
6834 =====
6835 [TASK] HumanEval/136 | entry_point=largest_smallest_integers
6836 [PROMPT PREVIEW]
6837
6838 def largest_smallest_integers(lst):
6839     """
6840     Create a function that returns a tuple (a, b), where 'a' is
6841     the largest of negative integers, and 'b' is the smallest
6842     of positive integers in a list.
6843     If there is no negative or positive integers, return them as
6843     None.
6844
6845     Examples:
```

```
6846     largest_smallest_integers([2, 4, 1, 3, 5, 7]) == (None, 1)
6847     largest_smallest_integers([]) == (None, ...)
6848 -----
6849 [INITIAL GENERATED CODE]
6850 def largest_smallest_integers(lst):
6851     negatives = [x for x in lst if x < 0]
6852     positives = [x for x in lst if x > 0]
6853
6854     a = max(negatives) if negatives else None
6855     b = min(positives) if positives else None
6856
6857     return (a, b)
6858 -----
6859 [HumanEval/136] Attempt 0/6
6860 [HumanEval/136] Ø PASS (attempt=0)
6861 =====
6862 =====
6863 [138/164] HumanEval/137
6864 =====
6865 [TASK] HumanEval/137 | entry_point=compare_one
6866 [PROMPT PREVIEW]
6867
6868 def compare_one(a, b):
6869     """
6870         Create a function that takes integers, floats, or strings
6871         representing
6872         real numbers, and returns the larger variable in its given
6873         variable type.
6874         Return None if the values are equal.
6875         Note: If a real number is represented as a string, the floating
6876         point might be . or ,
6877     """
6878
6879 [INITIAL GENERATED CODE]
6880 def compare_one(a, b):
6881     def parse_value(val):
6882         if isinstance(val, str):
6883             val = val.replace(',', '.')
6884             return float(val)
6885
6886     a_parsed = parse_value(a)
6887     b_parsed = parse_value(b)
6888
6889     if a_parsed == b_parsed:
6890         return None
```

```
6891     elif a_parsed > b_parsed:
6892         return a
6893     else:
6894         return b
6895 -----
6896 -----
6896 [HumanEval/137] Attempt 0/6
6897 [HumanEval/137] Ø PASS (attempt=0)
6898 =====
6899 =====
6899 [139/164] HumanEval/138
6900 =====
6901 =====
6901 [TASK] HumanEval/138 | entry_point=is_equal_to_sum_even
6902 [PROMPT PREVIEW]
6903
6904 def is_equal_to_sum_even(n):
6905     """Evaluate whether the given number n can be written as the sum
6906     of exactly 4 positive even numbers
6907     Example
6907     is_equal_to_sum_even(4) == False
6908     is_equal_to_sum_even(6) == False
6909     is_equal_to_sum_even(8) == True
6910     """
6911 -----
6912 -----
6912 [INITIAL GENERATED CODE]
6913 def is_equal_to_sum_even(n):
6914     return n >= 8 and n % 2 == 0
6915 -----
6916 -----
6916 [HumanEval/138] Attempt 0/6
6917 [HumanEval/138] Ø PASS (attempt=0)
6918 =====
6919 =====
6919 [140/164] HumanEval/139
6920 =====
6921 =====
6921 [TASK] HumanEval/139 | entry_point=special_factorial
6922 [PROMPT PREVIEW]
6923
6924 def special_factorial(n):
6925     """The Brazilian factorial is defined as:
6926     brazilian_factorial(n) = n! * (n-1)! * (n-2)! * ... * 1!
6927     where n > 0
6928
6929     For example:
6930     >>> special_factorial(4)
6931     288
6932
6933     The function will receive an integer as input and should return
6934     the special
6934     factorial of this integer.
```

```
6935 """
6936 -----
6937 [INITIAL GENERATED CODE]
6938 import math
6939
6940 def special_factorial(n):
6941     result = 1
6942     for i in range(1, n + 1):
6943         result *= math.factorial(i)
6944     return result
6945 -----
6946 [HumanEval/139] Attempt 0/6
6947 [HumanEval/139] Ø PASS (attempt=0)
6948 =====
6949 =====
6949 [141/164] HumanEval/140
6950 =====
6951 =====
6951 [TASK] HumanEval/140 | entry_point=fix_spaces
6952 [PROMPT PREVIEW]
6953
6954 def fix_spaces(text):
6955     """
6956     Given a string text, replace all spaces in it with underscores,
6957     and if a string has more than 2 consecutive spaces,
6958     then replace all consecutive spaces with -
6959
6960     fix_spaces("Example") == "Example"
6961     fix_spaces("Example 1") == "Example_1"
6962     fix_spaces(" Example 2") == "_Example_2"
6963     fix_spaces(" Example 3") == "_Example-3"
6964     """
6965 -----
6966 [INITIAL GENERATED CODE]
6967 def fix_spaces(text):
6968     return text.replace(' ', '-').replace(' ', '_')
6969 -----
6970 -----
6970 [HumanEval/140] Attempt 0/6
6971 [HumanEval/140] Ø PASS (attempt=0)
6972 =====
6973 =====
6973 [142/164] HumanEval/141
6974 =====
6975 =====
6975 [TASK] HumanEval/141 | entry_point=file_name_check
6976 [PROMPT PREVIEW]
6977
6978 def file_name_check(file_name):
6979     """Create a function which takes a string representing a file's
```

```
6979 name, and returns
6980     'Yes' if the file's name is valid, and returns 'No'
6981     otherwise.
6982     A file's name is considered to be valid if and only if all the
6983     following conditions
6984     are met:
6985         - There should not be more than three digits ('0'-'9') in the
6986         file's name.
6987         - The file's name contain...
6988 -----
6989 [INITIAL GENERATED CODE]
6990 import re
6991
6992 def file_name_check(file_name):
6993     pattern = r'^[a-zA-Z][^\d]*\d{0,3}[^\d]*\.(txt|exe|dll)$'
6994     return 'Yes' if re.match(pattern, file_name) else 'No'
6995 -----
6996 [HumanEval/141] Attempt 0/6
6997 [HumanEval/141] ⚡ FAIL (attempt=0)
6998 [TEST ERROR PREVIEW]
6999 Traceback (most recent call last):
7000   File "C:\Users\JOIN\Downloads\Multi-
7001     Reviewer_Governance_for_Code_Generation\src\evaluator\
7002       humaneval_evaluator.py", line 39, in _worker
7003         ns["check"](candidate)
7004         File "<string>", line 8, in check
7005   AssertionError
7006 -----
7007 [HumanEval/141] [REPAIRED CODE] (after attempt=0)
7008 import re
7009
7010 def file_name_check(file_name):
7011     pattern = r'^[a-zA-Z][^\d]*\d{0,3}[^\d]*\.(txt|exe|dll)$'
7012     if re.match(pattern, file_name):
7013         return 'Yes'
7014     else:
7015         return 'No'
7016 -----
7017 [HumanEval/141] Attempt 1/6
7018 [HumanEval/141] ⚡ FAIL (attempt=1)
7019 [TEST ERROR PREVIEW]
7020 Traceback (most recent call last):
7021   File "C:\Users\JOIN\Downloads\Multi-
7022     Reviewer_Governance_for_Code_Generation\src\evaluator\
7023       humaneval_evaluator.py", line 39, in _worker
7024         ns["check"](candidate)
7025         File "<string>", line 8, in check
7026   AssertionError
7027 -----
```

```
7020 -----
7021 [HumanEval/141] [REPAIRED CODE] (after attempt=1)
7022 import re
7023
7024 def file_name_check(file_name):
7025     pattern = r'^[a-zA-Z][^\d]*\d{0,3}[^\d]*\.(txt|exe|dll)$'
7026     if re.match(pattern, file_name):
7027         return 'Yes'
7028     else:
7029         return 'No'
7030 -----
7031 -----
7032 [HumanEval/141] Attempt 2/6
7033 [HumanEval/141] ⚡ FAIL (attempt=2)
7034 [TEST ERROR PREVIEW]
7035 Traceback (most recent call last):
7036   File "C:\Users\JOIN\Downloads\Multi-
    Reviewer_Governance_for_Code_Generation\src\evaluator\
      humaneval_evaluator.py", line 39, in _worker
7037     ns["check"](candidate)
7038   File "<string>", line 8, in check
7039 Assertion
7040 -----
7041 -----
7042 [HumanEval/141] [REPAIRED CODE] (after attempt=2)
7043 import re
7044
7045 def file_name_check(file_name):
7046     pattern = r'^[a-zA-Z][^\d]*\d{0,3}[^\d]*\.(txt|exe|dll)$'
7047     if re.match(pattern, file_name):
7048         return 'Yes'
7049     else:
7050         return 'No'
7051 -----
7052 -----
7053 [HumanEval/141] Attempt 3/6
7054 [HumanEval/141] ⚡ FAIL (attempt=3)
7055 [TEST ERROR PREVIEW]
7056 Traceback (most recent call last):
7057   File "C:\Users\JOIN\Downloads\Multi-
    Reviewer_Governance_for_Code_Generation\src\evaluator\
      humaneval_evaluator.py", line 39, in _worker
7058     ns["check"](candidate)
7059   File "<string>", line 8, in check
7060 Assertion
7061 -----
7062 [HumanEval/141] [REPAIRED CODE] (after attempt=3)
7063 import re
7064
7065 def file_name_check(file_name):
7066     pattern = r'^[a-zA-Z][^\d]*\d{0,3}[^\d]*\.(txt|exe|dll)$'
7067     if re.match(pattern, file_name):
```

```
7065         return 'Yes'
7066     else:
7067         return 'No'
7068 -----
7069 [HumanEval/141] Attempt 4/6
7070 [HumanEval/141] ⚡ FAIL (attempt=4)
7071 [TEST ERROR PREVIEW]
7072 Traceback (most recent call last):
7073   File "C:\Users\JOIN\Downloads\Multi-
    Reviewer_Governance_for_Code_Generation\src\evaluator\
      humaneval_evaluator.py", line 39, in _worker
7074     ns["check"](candidate)
7075   File "<string>", line 8, in check
7076 AssertionError
7077 -----
7078 [HumanEval/141] [REPAIRED CODE] (after attempt=4)
7079 import re
7080
7081 def file_name_check(file_name):
7082     pattern = r'^[a-zA-Z][^\d]*\d{0,3}[^\d]*\.(txt|exe|dll)$'
7083     if re.match(pattern, file_name):
7084         return 'Yes'
7085     else:
7086         return 'No'
7087 -----
7088 [HumanEval/141] Attempt 5/6
7089 [HumanEval/141] ⚡ FAIL (attempt=5)
7090 [TEST ERROR PREVIEW]
7091 Traceback (most recent call last):
7092   File "C:\Users\JOIN\Downloads\Multi-
    Reviewer_Governance_for_Code_Generation\src\evaluator\
      humaneval_evaluator.py", line 39, in _worker
7093     ns["check"](candidate)
7094   File "<string>", line 8, in check
7095 AssertionError
7096 -----
7097 [HumanEval/141] [REPAIRED CODE] (after attempt=5)
7098 import re
7099
7100 def file_name_check(file_name):
7101     pattern = r'^[a-zA-Z][^\d]*\d{0,3}[^\d]*\.(txt|exe|dll)$'
7102     if re.match(pattern, file_name):
7103         return 'Yes'
7104     else:
7105         return 'No'
7106 -----
7107 [HumanEval/141] Attempt 6/6
7108 [HumanEval/141] ⚡ FAIL (attempt=6)
```

```
7109 [TEST ERROR PREVIEW]
7110 Traceback (most recent call last):
7111   File "C:\Users\JOIN\Downloads\Multi-
    Reviewer_Governance_for_Code_Generation\src\evaluator\
      humaneval_evaluator.py", line 39, in _worker
7112     ns["check"]的文化)
7113   File "<string>", line 8, in check
7114 AssertionError
7115 -----
    -----
7116 [HumanEval/141] ✘ FINAL FAIL after 6 retries
7117 =====
    =====
7118 [143/164] HumanEval/142
7119 =====
    =====
7120 [TASK] HumanEval/142 | entry_point=sum_squares
7121 [PROMPT PREVIEW]
7122
7123
7124
7125 def sum_squares(lst):
7126     """
7127         This function will take a list of integers. For all entries in
        the list, the function shall square the integer entry if its index
        is a
7128             multiple of 3 and will cube the integer entry if its index is a
            multiple of 4 and not a multiple of 3. The function will not
7129             change the entries in the list whose indexes are not a multiple
            of 3 or 4. The function shal...
7130 -----
    -----
7131 [INITIAL GENERATED CODE]
7132 def sum_squares(lst):
7133     total_sum = 0
7134     for i, num in enumerate(lst):
7135         if i % 3 == 0:
7136             total_sum += num ** 2
7137         elif i % 4 == 0:
7138             total_sum += num ** 3
7139         else:
7140             total_sum += num
7141     return total_sum
7142 -----
    -----
7143 [HumanEval/142] Attempt 0/6
7144 [HumanEval/142] ✘ PASS (attempt=0)
7145 =====
    =====
7146 [144/164] HumanEval/143
7147 =====
    =====
7148 [TASK] HumanEval/143 | entry_point=words_in_sentence
```

```
7149 [PROMPT PREVIEW]
7150
7151 def words_in_sentence(sentence):
7152     """
7153         You are given a string representing a sentence,
7154         the sentence contains some words separated by a space,
7155         and you have to return a string that contains the words from the
7156         original sentence,
7157         whose lengths are prime numbers,
7158         the order of the words in the new string should be the same as
7159         the original one.
7160
7161     Example 1:
7162         Input: senten...
7163 -----
7164 -----
7165 [INITIAL GENERATED CODE]
7166 def is_prime(n):
7167     if n < 2:
7168         return False
7169     for i in range(2, int(n**0.5) + 1):
7170         if n % i == 0:
7171             return False
7172     return True
7173
7174 def words_in_sentence(sentence):
7175     words = sentence.split()
7176     prime_length_words = [word for word in words if is_prime(len(
7177         word))]
7178     return ' '.join(prime_length_words)
7179 -----
7180 -----
7181 [HumanEval/143] Attempt 0/6
7182 [HumanEval/143] 0 PASS (attempt=0)
7183 =====
7184 =====
7185 [145/164] HumanEval/144
7186 =====
7187 =====
7188 [TASK] HumanEval/144 | entry_point=simplify
7189 [PROMPT PREVIEW]
7190
7191     Your task is to implement a function that will simplify the
7192     expression
7193      $x * n$ . The function returns True if  $x * n$  evaluates to a whole
7194     number and False
7195     otherwise. Both  $x$  and  $n$ , are string representation of a fraction
7196     , and have the following format,
7197     <numerator>/<denominator> where both numerator and denominator
7198     are positive whole numbers.
7199
7200     You can assume t...
```

```
7191 -----
-----
7192 [INITIAL GENERATED CODE]
7193 def simplify(x, n):
7194     num_x, denom_x = map(int, x.split('/'))
7195     num_n, denom_n = map(int, n.split('/'))
7196     return (num_x * num_n) % (denom_x * denom_n) == 0
7197 -----
-----
7198 [HumanEval/144] Attempt 0/6
7199 [HumanEval/144] 0 PASS (attempt=0)
7200 =====
=====
7201 [146/164] HumanEval/145
7202 =====
=====
7203 [TASK] HumanEval/145 | entry_point=order_by_points
7204 [PROMPT PREVIEW]
7205
7206 def order_by_points(nums):
7207     """
7208         Write a function which sorts the given list of integers
7209         in ascending order according to the sum of their digits.
7210         Note: if there are several items with similar sum of their
7211             digits,
7212                 order them based on their index in original list.
7213
7214     For example:
7215     >>> order_by_points([1, 11, -1, -11, -12,
7216     11])
7215     >>> order_by_points([])...
7216 -----
-----
7217 [INITIAL GENERATED CODE]
7218 def order_by_points(nums):
7219     return sorted(nums, key=lambda x: (sum(int(digit) for digit in
7220                                         str(abs(x))), nums.index(x)))
7220 -----
-----
7221 [HumanEval/145] Attempt 0/6
7222 [HumanEval/145] 0 FAIL (attempt=0)
7223 [TEST ERROR PREVIEW]
7224 Traceback (most recent call last):
7225   File "C:\Users\JOIN\Downloads\Multi-
    Reviewer_Governance_for_Code_Generation\src\evaluator\
      humaneval_evaluator.py", line 39, in _worker
7226     ns["check"](candidate)
7227   File "<string>", line 4, in check
7228 AssertionError
7229 -----
-----
7230 [HumanEval/145] [REPAIRED CODE] (after attempt=0)
7231 def order_by_points(nums):
```

```
7232     return sorted(nums, key=lambda x: (sum(int(digit) for digit in
str(abs(x))), nums.index(x)))
7233 -----
7234 [HumanEval/145] Attempt 1/6
7235 [HumanEval/145] ⚡ FAIL (attempt=1)
7236 [TEST ERROR PREVIEW]
7237 Traceback (most recent call last):
7238   File "C:\Users\JOIN\Downloads\Multi-
Reviewer_Governance_for_Code_Generation\src\evaluator\
humaneval_evaluator.py", line 39, in _worker
7239     ns["check"](candidate)
7240   File "<string>", line 4, in check
7241 AssertionError
7242 -----
7243 [HumanEval/145] [REPAIRED CODE] (after attempt=1)
7244 def order_by_points(nums):
7245     return sorted(nums, key=lambda x: (sum(int(digit) for digit in
str(abs(x))), nums.index(x)))
7246 -----
7247 [HumanEval/145] Attempt 2/6
7248 [HumanEval/145] ⚡ FAIL (attempt=2)
7249 [TEST ERROR PREVIEW]
7250 Traceback (most recent call last):
7251   File "C:\Users\JOIN\Downloads\Multi-
Reviewer_Governance_for_Code_Generation\src\evaluator\
humaneval_evaluator.py", line 39, in _worker
7252     ns["check"](candidate)
7253   File "<string>", line 4, in check
7254 AssertionError
7255 -----
7256 [HumanEval/145] [REPAIRED CODE] (after attempt=2)
7257 def order_by_points(nums):
7258     return sorted(nums, key=lambda x: (sum(int(digit) for digit in
str(abs(x))), nums.index(x)))
7259 -----
7260 [HumanEval/145] Attempt 3/6
7261 [HumanEval/145] ⚡ FAIL (attempt=3)
7262 [TEST ERROR PREVIEW]
7263 Traceback (most recent call last):
7264   File "C:\Users\JOIN\Downloads\Multi-
Reviewer_Governance_for_Code_Generation\src\evaluator\
humaneval_evaluator.py", line 39, in _worker
7265     ns["check"](candidate)
7266   File "<string>", line 4, in check
7267 AssertionError
7268 -----
7269 [HumanEval/145] [REPAIRED CODE] (after attempt=3)
```

```
7270 def order_by_points(nums):
7271     return sorted(nums, key=lambda x: (sum(int(digit) for digit in
7272 -----  
-----  
7273 [HumanEval/145] Attempt 4/6
7274 [HumanEval/145] ⚡ FAIL (attempt=4)
7275 [TEST ERROR PREVIEW]
7276 Traceback (most recent call last):
7277   File "C:\Users\JOIN\Downloads\Multi-
7278     Reviewer_Governance_for_Code_Generation\src\evaluator\
7279       humaneval_evaluator.py", line 39, in _worker
7280         ns["check"](candidate)
7281     File "<string>", line 4, in check
7282 Assertion
7283 Error
7284 -----  
-----  
7285 [HumanEval/145] [REPAIRED CODE] (after attempt=4)
7286 def order_by_points(nums):
7287     return sorted(nums, key=lambda x: (sum(int(digit) for digit in
7288 str(abs(x))), nums.index(x)))
7289 -----  
-----  
7290 [HumanEval/145] Attempt 5/6
7291 [HumanEval/145] ⚡ FAIL (attempt=5)
7292 [TEST ERROR PREVIEW]
7293 Traceback (most recent call last):
7294   File "C:\Users\JOIN\Downloads\Multi-
7295     Reviewer_Governance_for_Code_Generation\src\evaluator\
7296       humaneval_evaluator.py", line 39, in _worker
7297         ns["check"](candidate)
7298     File "<string>", line 4, in check
7299 Assertion
7300 Error
7301 -----  
-----  
7302 [HumanEval/145] [REPAIRED CODE] (after attempt=5)
7303 def order_by_points(nums):
7304     return sorted(nums, key=lambda x: (sum(int(digit) for digit in
7305 str(abs(x))), nums.index(x)))
7306 -----  
-----  
7307 [HumanEval/145] Attempt 6/6
7308 [HumanEval/145] ⚡ FAIL (attempt=6)
7309 [TEST ERROR PREVIEW]
7310 Traceback (most recent call last):
7311   File "C:\Users\JOIN\Downloads\Multi-
7312     Reviewer_Governance_for_Code_Generation\src\evaluator\
7313       humaneval_evaluator.py", line 39, in _worker
7314         ns["check"](candidate)
7315     File "<string>", line 4, in check
7316 Assertion
7317 Error
7318 -----  
-----
```

```

7308 [HumanEval/145] ⚡ FINAL FAIL after 6 retries
7309 =====
    =====
7310 [147/164] HumanEval/146
7311 =====
    =====
7312 [TASK] HumanEval/146 | entry_point=specialFilter
7313 [PROMPT PREVIEW]
7314
7315 def specialFilter(nums):
7316     """Write a function that takes an array of numbers as input and
    returns
7317     the number of elements in the array that are greater than 10 and
    both
7318     first and last digits of a number are odd (1, 3, 5, 7, 9).
7319     For example:
7320     specialFilter([15, -73, 14, -15]) => 1
7321     specialFilter([33, -2, -3, 45, 21, 109]) => 2
7322     """
7323 -----
    -----
7324 [INITIAL GENERATED CODE]
7325 def specialFilter(nums):
7326     count = 0
7327     for num in nums:
7328         if num > 10:
7329             str_num = str(num)
7330             if int(str_num[0]) % 2 != 0 and int(str_num[-1]) % 2 !=
    0:
    count += 1
7332     return count
7333 -----
    -----
7334 [HumanEval/146] Attempt 0/6
7335 [HumanEval/146] ⚡ PASS (attempt=0)
7336 =====
    =====
7337 [148/164] HumanEval/147
7338 =====
    =====
7339 [TASK] HumanEval/147 | entry_point=get_max_triples
7340 [PROMPT PREVIEW]
7341
7342 def get_max_triples(n):
7343     """
7344     You are given a positive integer n. You have to create an
    integer array a of length n.
7345     For each i ( $1 \leq i \leq n$ ), the value of  $a[i] = i * i - i + 1$ .
7346     Return the number of triples  $(a[i], a[j], a[k])$  of a where  $i < j < k$ ,
7347     and  $a[i] + a[j] + a[k]$  is a multiple of 3.
7348
7349     Example :
```

```
7350      Input: n = 5
7351      Output: 1
7352      Explanation:...
7353 -----
7354 [INITIAL GENERATED CODE]
7355 def get_max_triples(n):
7356     a = [i * i - i + 1 for i in range(1, n + 1)]
7357     count = 0
7358     for i in range(n):
7359         for j in range(i + 1, n):
7360             for k in range(j + 1, n):
7361                 if (a[i] + a[j] + a[k]) % 3 == 0:
7362                     count += 1
7363     return count
7364 -----
7365 [HumanEval/147] Attempt 0/6
7366 [HumanEval/147] Ø PASS (attempt=0)
7367 =====
7368 [149/164] HumanEval/148
7369 =====
7370 [TASK] HumanEval/148 | entry_point=bf
7371 [PROMPT PREVIEW]
7372
7373 def bf(planet1, planet2):
7374     '''
7375     There are eight planets in our solar system: the closerst to the
7376     Sun
7377     is Mercury, the next one is Venus, then Earth, Mars, Jupiter,
7378     Saturn,
7379     Uranus, Neptune.
7380     Write a function that takes two planet names as strings planet1
7381     and planet2.
7382     The function should return a tuple containing all planets whose
7383     orbits are
7384     located between the orbit of...
7385
7386 -----
7387
7388 [INITIAL GENERATED CODE]
7389 def bf(planet1, planet2):
7390     planets = ["Mercury", "Venus", "Earth", "Mars", "Jupiter", "
7391     Saturn", "Uranus", "Neptune"]
7392     if planet1 not in planets or planet2 not in planets:
7393         return ()
7394
7395     index1, index2 = planets.index(planet1), planets.index(planet2)
7396     if index1 > index2:
7397         index1, index2 = index2, index1
7398
7399     return tuple(planets[index1 + 1:index2])
```

```
7393 -----
-----
7394 [HumanEval/148] Attempt 0/6
7395 [HumanEval/148] Ø PASS (attempt=0)
7396 =====
=====
7397 [150/164] HumanEval/149
7398 =====
=====
7399 [TASK] HumanEval/149 | entry_point=sorted_list_sum
7400 [PROMPT PREVIEW]
7401
7402 def sorted_list_sum(lst):
7403     """Write a function that accepts a list of strings as a
    parameter,
7404     deletes the strings that have odd lengths from it,
7405     and returns the resulted list with a sorted order,
7406     The list is always a list of strings and never an array of
    numbers,
7407     and it may contain duplicates.
7408     The order of the list should be ascending by length of each word
    , and you
7409     sho...
7410 -----
-----
7411 [INITIAL GENERATED CODE]
7412 def sorted_list_sum(lst):
7413     return sorted([word for word in lst if len(word) % 2 == 0], key=
    lambda x: (len(x), x))
7414 -----
-----
7415 [HumanEval/149] Attempt 0/6
7416 [HumanEval/149] Ø PASS (attempt=0)
7417 =====
=====
7418 [151/164] HumanEval/150
7419 =====
=====
7420 [TASK] HumanEval/150 | entry_point=x_or_y
7421 [PROMPT PREVIEW]
7422
7423 def x_or_y(n, x, y):
7424     """A simple program which should return the value of x if n is
7425     a prime number and should return the value of y otherwise.
7426
7427     Examples:
7428     for x_or_y(7, 34, 12) == 34
7429     for x_or_y(15, 8, 5) == 5
7430
7431     """
7432 -----
-----
7433 [INITIAL GENERATED CODE]
```

```
7434 def is_prime(num):  
7435     if num <= 1:  
7436         return False  
7437     for i in range(2, int(num**0.5) + 1):  
7438         if num % i == 0:  
7439             return False  
7440     return True  
7441  
7442 def x_or_y(n, x, y):  
7443     return x if is_prime(n) else y  
7444 -----  
-----  
7445 [HumanEval/150] Attempt 0/6  
7446 [HumanEval/150] [] PASS (attempt=0)  
7447 ======  
=====  
7448 [152/164] HumanEval/151  
7449 ======  
=====  
7450 [TASK] HumanEval/151 | entry_point=double_the_difference  
7451 [PROMPT PREVIEW]  
7452  
7453 def double_the_difference(lst):  
7454     '''  
7455         Given a list of numbers, return the sum of squares of the  
    numbers  
7456         in the list that are odd. Ignore numbers that are negative or  
    not integers.  
7457  
7458         double_the_difference([1, 3, 2, 0]) == 1 + 9 + 0 + 0 = 10  
7459         double_the_difference([-1, -2, 0]) == 0  
7460         double_the_difference([9, -2]) == 81  
7461         double_the_difference([0]) == 0  
7462  
7463         If the input...  
7464 -----  
-----  
7465 [INITIAL GENERATED CODE]  
7466 def double_the_difference(lst):  
7467     return sum(x**2 for x in lst if isinstance(x, int) and x >= 0  
    and x % 2 != 0)  
7468 -----  
-----  
7469 [HumanEval/151] Attempt 0/6  
7470 [HumanEval/151] [] PASS (attempt=0)  
7471 ======  
=====  
7472 [153/164] HumanEval/152  
7473 ======  
=====  
7474 [TASK] HumanEval/152 | entry_point=compare  
7475 [PROMPT PREVIEW]  
7476
```

```
7477 def compare(game,guess):
7478     """I think we all remember that feeling when the result of some
    long-awaited
7479     event is finally known. The feelings and thoughts you have at
    that moment are
7480     definitely worth noting down and comparing.
7481     Your task is to determine if a person correctly guessed the
    results of a number of matches.
7482     You are given two arrays of scores and guesses of equal length
    ,...
7483 -----
    -----
7484 [INITIAL GENERATED CODE]
7485 def compare(game, guess):
7486     return [abs(g - s) for g, s in zip(guess, game)]
7487 -----
    -----
7488 [HumanEval/152] Attempt 0/6
7489 [HumanEval/152] Ø PASS (attempt=0)
7490 =====
    =====
7491 [154/164] HumanEval/153
7492 =====
    =====
7493 [TASK] HumanEval/153 | entry_point=Strongest_Extension
7494 [PROMPT PREVIEW]
7495
7496 def Strongest_Extension(class_name, extensions):
7497     """You will be given the name of a class (a string) and a list
    of extensions.
7498     The extensions are to be used to load additional classes to the
    class. The
7499     strength of the extension is as follows: Let CAP be the number
    of the uppercase
7500     letters in the extension's name, and let SM be the number of
    lowercase letters
7501     in the extension'...
7502 -----
    -----
7503 [INITIAL GENERATED CODE]
7504 def Strongest_Extension(class_name, extensions):
7505     max_strength = float('-inf')
7506     strongest_extension = ''
7507
7508     for extension in extensions:
7509         cap = sum(1 for char in extension if char.isupper())
7510         sm = sum(1 for char in extension if char.islower())
7511         strength = cap - sm
7512
7513         if strength > max_strength:
7514             max_strength = strength
7515             strongest_extension = extension
7516
```

```
7517     return f'{class_name}.{strongest_extension}'  
7518 -----  
-----  
7519 [HumanEval/153] Attempt 0/6  
7520 [HumanEval/153] ✘ PASS (attempt=0)  
7521 ======  
=====  
7522 [155/164] HumanEval/154  
7523 ======  
=====  
7524 [TASK] HumanEval/154 | entry_point=cycpattern_check  
7525 [PROMPT PREVIEW]  
7526  
7527 def cycpattern_check(a , b):  
7528     """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  
7529     cycpattern_check("abcd","abd") => False  
7530     cycpattern_check("hello","ell") => True  
7531     cycpattern_check("whassup","psus") => False  
7532     cycpattern_check("abab","baa") => True  
7533     cycpattern_check("efef","eeff") => False  
7534     cycpattern_c...  
7535 -----  
-----  
7536 [INITIAL GENERATED CODE]  
7537 def cycpattern_check(a, b):  
7538     return any(b in a[i:] + a[:i] for i in range(len(b)))  
7539 -----  
-----  
7540 [HumanEval/154] Attempt 0/6  
7541 [HumanEval/154] ✘ FAIL (attempt=0)  
7542 [TEST ERROR PREVIEW]  
7543 Traceback (most recent call last):  
7544   File "C:\Users\JOIN\Downloads\Multi-  
      Reviewer_Governance_for_Code_Generation\src\evaluator\  
      humaneval_evaluator.py", line 39, in _worker  
7545     ns["check"](candidate)  
7546   File "<string>", line 11, in check  
7547 AssertionError: test #3  
7548 -----  
-----  
7549 [HumanEval/154] [REPAIRED CODE] (after attempt=0)  
7550 def cycpattern_check(a, b):  
7551     return any(b in (a[i:] + a[:i]) for i in range(len(b)))  
7552 -----  
-----  
7553 [HumanEval/154] Attempt 1/6  
7554 [HumanEval/154] ✘ FAIL (attempt=1)  
7555 [TEST ERROR PREVIEW]  
7556 Traceback (most recent call last):  
7557   File "C:\Users\JOIN\Downloads\Multi-  
      Reviewer_Governance_for_Code_Generation\src\evaluator\  
      humaneval_evaluator.py", line 39, in _worker
```

```
7558     ns["check"](candidate)
7559 File "<string>", line 11, in check
7560 AssertionError: test #3
7561 -----
7562 [HumanEval/154] [REPAIRED CODE] (after attempt=1)
7563 def cycpattern_check(a, b):
7564     return any(b in (a[i:] + a[:i]) for i in range(len(a)))
7565 -----
7566 [HumanEval/154] Attempt 2/6
7567 [HumanEval/154] ⚡ FAIL (attempt=2)
7568 [TEST ERROR PREVIEW]
7569 Traceback (most recent call last):
7570   File "C:\Users\JOIN\Downloads\Multi-
    Reviewer_Governance_for_Code_Generation\src\evaluator\
      humaneval_evaluator.py", line 39, in _worker
7571     ns["check"](candidate)
7572   File "<string>", line 11, in check
7573 AssertionError: test #3
7574 -----
7575 [HumanEval/154] [REPAIRED CODE] (after attempt=2)
7576 def cycpattern_check(a, b):
7577     return any(b in (a[i:] + a[:i]) for i in range(len(a)))
7578 -----
7579 [HumanEval/154] Attempt 3/6
7580 [HumanEval/154] ⚡ FAIL (attempt=3)
7581 [TEST ERROR PREVIEW]
7582 Traceback (most recent call last):
7583   File "C:\Users\JOIN\Downloads\Multi-
    Reviewer_Governance_for_Code_Generation\src\evaluator\
      humaneval_evaluator.py", line 39, in _worker
7584     ns["check"](candidate)
7585   File "<string>", line 11, in check
7586 AssertionError: test #3
7587 -----
7588 [HumanEval/154] [REPAIRED CODE] (after attempt=3)
7589 def cycpattern_check(a, b):
7590     return any(b in (a[i:] + a[:i]) for i in range(len(a)))
7591 -----
7592 [HumanEval/154] Attempt 4/6
7593 [HumanEval/154] ⚡ FAIL (attempt=4)
7594 [TEST ERROR PREVIEW]
7595 Traceback (most recent call last):
7596   File "C:\Users\JOIN\Downloads\Multi-
    Reviewer_Governance_for_Code_Generation\src\evaluator\
      humaneval_evaluator.py", line 39, in _worker
7597     ns["check"](candidate)
7598   File "<string>", line 11, in check
```

```
7599 AssertionError: test #3
7600 -----
    -----
7601 [HumanEval/154] [REPAIRED CODE] (after attempt=4)
7602 def cycpattern_check(a, b):
7603     return any(b == (a[i:] + a[:i]) for i in range(len(a)))
7604 -----
    -----
7605 [HumanEval/154] Attempt 5/6
7606 [HumanEval/154] ⚡ FAIL (attempt=5)
7607 [TEST ERROR PREVIEW]
7608 Traceback (most recent call last):
7609     File "C:\Users\JOIN\Downloads\Multi-
    Reviewer_Governance_for_Code_Generation\src\evaluator\
        humaneval_evaluator.py", line 39, in _worker
7610     ns["check"](candidate)
7611     File "<string>", line 9, in check
7612 AssertionError: test #1
7613 -----
    -----
7614 [HumanEval/154] [REPAIRED CODE] (after attempt=5)
7615 def cycpattern_check(a, b):
7616     return any(b == (a[i:] + a[:i]) for i in range(len(a) + 1))
7617 -----
    -----
7618 [HumanEval/154] Attempt 6/6
7619 [HumanEval/154] ⚡ FAIL (attempt=6)
7620 [TEST ERROR PREVIEW]
7621 Traceback (most recent call last):
7622     File "C:\Users\JOIN\Downloads\Multi-
    Reviewer_Governance_for_Code_Generation\src\evaluator\
        humaneval_evaluator.py", line 39, in _worker
7623     ns["check"](candidate)
7624     File "<string>", line 9, in check
7625 AssertionError: test #1
7626 -----
    -----
7627 [HumanEval/154] ⚡ FINAL FAIL after 6 retries
7628 =====
    =====
7629 [156/164] HumanEval/155
7630 =====
    =====
7631 [TASK] HumanEval/155 | entry_point=even_odd_count
7632 [PROMPT PREVIEW]
7633
7634 def even_odd_count(num):
7635     """Given an integer. return a tuple that has the number of even
    and odd digits respectively.
7636
7637     Example:
7638         even_odd_count(-12) ==> (1, 1)
7639         even_odd_count(123) ==> (1, 2)
```

```
7640     """
7641 -----
7642 [INITIAL GENERATED CODE]
7643 def even_odd_count(num):
7644     num = abs(num)
7645     even_count = sum(1 for digit in str(num) if int(digit) % 2 == 0)
7646     odd_count = sum(1 for digit in str(num) if int(digit) % 2 != 0)
7647     return (even_count, odd_count)
7648 -----
7649 [HumanEval/155] Attempt 0/6
7650 [HumanEval/155] 0 PASS (attempt=0)
7651 =====
7652 [157/164] HumanEval/156
7653 =====
7654 [TASK] HumanEval/156 | entry_point=int_to_mini_roman
7655 [PROMPT PREVIEW]
7656
7657 def int_to_mini_roman(number):
7658     """
7659         Given a positive integer, obtain its roman numeral equivalent as
7660         a string,
7661         and return it in lowercase.
7662         Restrictions: 1 <= num <= 1000
7663
7664         Examples:
7665         >>> int_to_mini_roman(19) == 'xix'
7666         >>> int_to_mini_roman(152) == 'cli'
7667         >>> int_to_mini_roman(426) == 'cdxxvi'
7668 """
7669 -----
7670 [INITIAL GENERATED CODE]
7671 def int_to_mini_roman(number):
7672     val = [
7673         1000, 900, 500, 400,
7674         100, 90, 50, 40,
7675         10, 9, 5, 4,
7676         1
7677     ]
7678     syms = [
7679         "m", "cm", "d", "cd",
7680         "c", "xc", "l", "xl",
7681         "x", "ix", "v", "iv",
7682         "i"
7683     ]
7684     roman_numeral = ''
7685     i = 0
7686     while number > 0:
7687         for _ in range(number // val[i]):
```

```
7687         roman_numeral += syms[i]
7688         number -= val[i]
7689         i += 1
7690     return roman_numeral
7691 -----
7692 -----
7692 [HumanEval/156] Attempt 0/6
7693 [HumanEval/156] Ø PASS (attempt=0)
7694 =====
7695 =====
7695 [158/164] HumanEval/157
7696 =====
7697 =====
7697 [TASK] HumanEval/157 | entry_point=right_angle_triangle
7698 [PROMPT PREVIEW]
7699
7700 def right_angle_triangle(a, b, c):
7701     '''
7702         Given the lengths of the three sides of a triangle. Return True
7702         if the three
7703             sides form a right-angled triangle, False otherwise.
7704             A right-angled triangle is a triangle in which one angle is
7704             right angle or
7705                 90 degree.
7706             Example:
7707                 right_angle_triangle(3, 4, 5) == True
7708                 right_angle_triangle(1, 2, 3) == False
7709                 '''
7710 -----
7711 -----
7711 [INITIAL GENERATED CODE]
7712 def right_angle_triangle(a, b, c):
7713     sides = sorted([a, b, c])
7714     return sides[0]**2 + sides[1]**2 == sides[2]**2
7715 -----
7716 -----
7716 [HumanEval/157] Attempt 0/6
7717 [HumanEval/157] Ø PASS (attempt=0)
7718 =====
7719 =====
7719 [159/164] HumanEval/158
7720 =====
7721 -----
7721 [TASK] HumanEval/158 | entry_point=find_max
7722 [PROMPT PREVIEW]
7723
7724 def find_max(words):
7725     """Write a function that accepts a list of strings.
7726         The list contains different words. Return the word with maximum
7726         number
7727             of unique characters. If multiple strings have maximum number of
7727             unique
7728                 characters, return the one which comes first in lexicographical
```

```
7728 order.
7729
7730     find_max(["name", "of", "string"]) == "string"
7731     find_max(["name", "enam", "game"]) == "e...
7732 -----
7733 -----
7733 [INITIAL GENERATED CODE]
7734 def find_max(words):
7735     max_word = ""
7736     max_unique_chars = 0
7737
7738     for word in words:
7739         unique_chars = len(set(word))
7740         if unique_chars > max_unique_chars or (unique_chars ==
max_unique_chars and word < max_word):
7741             max_word = word
7742             max_unique_chars = unique_chars
7743
7744     return max_word
7745 -----
7746 -----
7746 [HumanEval/158] Attempt 0/6
7747 [HumanEval/158] Ø PASS (attempt=0)
7748 =====
7749 =====
7749 [160/164] HumanEval/159
7750 =====
7750 =====
7751 [TASK] HumanEval/159 | entry_point=eat
7752 [PROMPT PREVIEW]
7753
7754 def eat(number, need, remaining):
7755     """
7756     You're a hungry rabbit, and you already have eaten a certain
    number of carrots,
7757     but now you need to eat more carrots to complete the day's meals
    .
7758     you should return an array of [ total number of eaten carrots
    after your meals,
7759                                         the number of carrots left after
    your meals ]
7760     if there are not enough remaining...
7761 -----
7761 -----
7762 [INITIAL GENERATED CODE]
7763 def eat(number, need, remaining):
7764     total_eaten = number + min(need, remaining)
7765     carrots_left = max(remaining - need, 0)
7766     return [total_eaten, carrots_left]
7767 -----
7767 -----
7768 [HumanEval/159] Attempt 0/6
7769 [HumanEval/159] Ø PASS (attempt=0)
```

```
7770 =====
=====
7771 [161/164] HumanEval/160
7772 =====
=====
7773 [TASK] HumanEval/160 | entry_point=do_algebra
7774 [PROMPT PREVIEW]
7775
7776 def do_algebra(operator, operand):
7777     """
7778     Given two lists operator, and operand. The first list has basic
    algebra operations, and
7779     the second list is a list of integers. Use the two given lists
    to build the algebraic
7780     expression and return the evaluation of this expression.
7781
7782     The basic algebra operations:
7783     Addition ( + )
7784     Subtraction ( - )
7785     Multiplication ( * )
7786     Floo...
7787 -----
    -----
7788 [INITIAL GENERATED CODE]
7789 def do_algebra(operator, operand):
7790     expression = str(operand[0])
7791     for i in range(len(operator)):
7792         expression += f" {operator[i]} {operand[i+1]}"
7793     return eval(expression)
7794 -----
    -----
7795 [HumanEval/160] Attempt 0/6
7796 [HumanEval/160] Ø PASS (attempt=0)
7797 =====
=====
7798 [162/164] HumanEval/161
7799 =====
=====
7800 [TASK] HumanEval/161 | entry_point=solve
7801 [PROMPT PREVIEW]
7802
7803 def solve(s):
7804     """You are given a string s.
7805     if s[i] is a letter, reverse its case from lower to upper or
    vise versa,
7806     otherwise keep it as it is.
7807     If the string contains no letters, reverse the string.
7808     The function should return the resulted string.
7809     Examples
7810     solve("1234") = "4321"
7811     solve("ab") = "AB"
7812     solve("#a@C") = "#A@c"
7813     """

```

```
7814 -----
-----
7815 [INITIAL GENERATED CODE]
7816 def solve(s):
7817     if any(c.isalpha() for c in s):
7818         return ''.join(c.swapcase() if c.isalpha() else c for c in s
    )
7819     else:
7820         return s[::-1]
7821 -----
-----
7822 [HumanEval/161] Attempt 0/6
7823 [HumanEval/161] Ø PASS (attempt=0)
7824 =====
=====
7825 [163/164] HumanEval/162
7826 =====
=====
7827 [TASK] HumanEval/162 | entry_point=string_to_md5
7828 [PROMPT PREVIEW]
7829
7830 def string_to_md5(text):
7831     """
7832         Given a string 'text', return its md5 hash equivalent string.
7833         If 'text' is an empty string, return None.
7834
7835     >>> string_to_md5('Hello world') == '
7836         3e25960a79dbc69b674cd4ec67a72c62'
7837 -----
-----
7838 [INITIAL GENERATED CODE]
7839 import hashlib
7840
7841 def string_to_md5(text):
7842     if not text:
7843         return None
7844     return hashlib.md5(text.encode()).hexdigest()
7845 -----
-----
7846 [HumanEval/162] Attempt 0/6
7847 [HumanEval/162] Ø PASS (attempt=0)
7848 =====
=====
7849 [164/164] HumanEval/163
7850 =====
=====
7851 [TASK] HumanEval/163 | entry_point=generate_integers
7852 [PROMPT PREVIEW]
7853
7854 def generate_integers(a, b):
7855     """
7856         Given two positive integers a and b, return the even digits
```

```
7856 between a
7857     and b, in ascending order.
7858
7859     For example:
7860     generate_integers(2, 8) => [2, 4, 6, 8]
7861     generate_integers(8, 2) => [2, 4, 6, 8]
7862     generate_integers(10, 14) => []
7863     """
7864 -----
7865 -----
7865 [INITIAL GENERATED CODE]
7866 def generate_integers(a, b):
7867     start = min(a, b)
7868     end = max(a, b)
7869     return [i for i in range(start, end + 1) if i % 2 == 0]
7870 -----
7871 -----
7871 [HumanEval/163] Attempt 0/6
7872 [HumanEval/163] ⚡ FAIL (attempt=0)
7873 [TEST ERROR PREVIEW]
7874 Traceback (most recent call last):
7875   File "C:\Users\JOIN\Downloads\Multi-
    Reviewer_Governance_for_Code_Generation\src\evaluator\
      humaneval_evaluator.py", line 39, in _worker
7876     ns["check"](candidate)
7877   File "<string>", line 4, in check
7878 AssertionError: Test 1
7879 -----
7880 -----
7880 [HumanEval/163] [REPAIRED CODE] (after attempt=0)
7881 def generate_integers(a, b):
7882     start = min(a, b)
7883     end = max(a, b)
7884     return [i for i in range(start, end + 1) if i % 2 == 0]
7885 -----
7886 -----
7886 [HumanEval/163] Attempt 1/6
7887 [HumanEval/163] ⚡ FAIL (attempt=1)
7888 [TEST ERROR PREVIEW]
7889 Traceback (most recent call last):
7890   File "C:\Users\JOIN\Downloads\Multi-
    Reviewer_Governance_for_Code_Generation\src\evaluator\
      humaneval_evaluator.py", line 39, in _worker
7891     ns["check"](candidate)
7892   File "<string>", line 4, in check
7893 AssertionError: Test 1
7894 -----
7895 -----
7895 [HumanEval/163] [REPAIRED CODE] (after attempt=1)
7896 def generate_integers(a, b):
7897     start = min(a, b)
7898     end = max(a, b)
7899     return [i for i in range(start, end + 1) if i % 2 == 0]
```

```
7900 -----
-----
7901 [HumanEval/163] Attempt 2/6
7902 [HumanEval/163] ⚡ FAIL (attempt=2)
7903 [TEST ERROR PREVIEW]
7904 Traceback (most recent call last):
7905   File "C:\Users\JOIN\Downloads\Multi-
    Reviewer_Governance_for_Code_Generation\src\evaluator\
      humaneval_evaluator.py", line 39, in _worker
7906     ns["check"](candidate)
7907   File "<string>", line 4, in check
7908 AssertionError: Test 1
7909 -----
-----
7910 [HumanEval/163] [REPAIRED CODE] (after attempt=2)
7911 def generate_integers(a, b):
7912     start = min(a, b)
7913     end = max(a, b)
7914     return [i for i in range(start, end + 1) if i % 2 == 0]
7915 -----
-----
7916 [HumanEval/163] Attempt 3/6
7917 [HumanEval/163] ⚡ FAIL (attempt=3)
7918 [TEST ERROR PREVIEW]
7919 Traceback (most recent call last):
7920   File "C:\Users\JOIN\Downloads\Multi-
    Reviewer_Governance_for_Code_Generation\src\evaluator\
      humaneval_evaluator.py", line 39, in _worker
7921     ns["check"](candidate)
7922   File "<string>", line 4, in check
7923 AssertionError: Test 1
7924 -----
-----
7925 [HumanEval/163] [REPAIRED CODE] (after attempt=3)
7926 def generate_integers(a, b):
7927     start = min(a, b)
7928     end = max(a, b)
7929     return [i for i in range(start, end + 1) if i % 2 == 0]
7930 -----
-----
7931 [HumanEval/163] Attempt 4/6
7932 [HumanEval/163] ⚡ FAIL (attempt=4)
7933 [TEST ERROR PREVIEW]
7934 Traceback (most recent call last):
7935   File "C:\Users\JOIN\Downloads\Multi-
    Reviewer_Governance_for_Code_Generation\src\evaluator\
      humaneval_evaluator.py", line 39, in _worker
7936     ns["check"](candidate)
7937   File "<string>", line 4, in check
7938 AssertionError: Test 1
7939 -----
-----
7940 [HumanEval/163] [REPAIRED CODE] (after attempt=4)
```

```
7941 def generate_integers(a, b):
7942     start = min(a, b)
7943     end = max(a, b)
7944     return [i for i in range(start, end + 1) if i % 2 == 0]
7945 -----
7946 -----
7946 [HumanEval/163] Attempt 5/6
7947 [HumanEval/163] ⚡ FAIL (attempt=5)
7948 [TEST ERROR PREVIEW]
7949 Traceback (most recent call last):
7950   File "C:\Users\JOIN\Downloads\Multi-
    Reviewer_Governance_for_Code_Generation\src\evaluator\
      humaneval_evaluator.py", line 39, in _worker
7951     ns["check"](candidate)
7952   File "<string>", line 4, in check
7953 AssertionError: Test 1
7954 -----
7955 -----
7955 [HumanEval/163] [REPAIRED CODE] (after attempt=5)
7956 def generate_integers(a, b):
7957     start = min(a, b)
7958     end = max(a, b)
7959     return [i for i in range(start, end + 1) if i % 2 == 0]
7960 -----
7961 -----
7961 [HumanEval/163] Attempt 6/6
7962 [HumanEval/163] ⚡ FAIL (attempt=6)
7963 [TEST ERROR PREVIEW]
7964 Traceback (most recent call last):
7965   File "C:\Users\JOIN\Downloads\Multi-
    Reviewer_Governance_for_Code_Generation\src\evaluator\
      humaneval_evaluator.py", line 39, in _worker
7966     ns["check"](candidate)
7967   File "<string>", line 4, in check
7968 AssertionError: Test 1
7969 -----
7970 -----
7970 [HumanEval/163] ⚡ FINAL FAIL after 6 retries
7971 =====
7972 Baseline: 0.8414634146341463
7973
7974 Process finished with exit code 0
7975
```