

create_table

```
create_table.py X

create_table.py > ...

Optimize this code [microphone icon] [play icon] X ...

Copilot generated code may be incorrect

1 def create_table(n: int, m: int) -> list[list]:
2     """
3     ... returns list with fibonacci numbers.
4     ...
5     ... >>> create_table(4, 6)
6     ... [[1, 1, 1, 1, 1, 1], [1, 2, 3, 4, 5, 6], [1, 3, 6, 10, 15, 21], [1, 4, 10, 20, 35, 56]]
7     ... """
8     ... if n > 1 and m > 1:
9     ...     for i, _ in enumerate(range(m)):
10    ...         if i > 0:
11    ...             table[0] += [table[0][i-1] + create_table(n-1, m)[-1][i]]
12    ...         else:
13    ...             table = [[1]]
14    ...     return create_table(n-1, m) + table
15    ... if n <= 1 or m <= 1:
16    ...     return [[1 for _ in range(m)] for _ in range(n)]
17
```

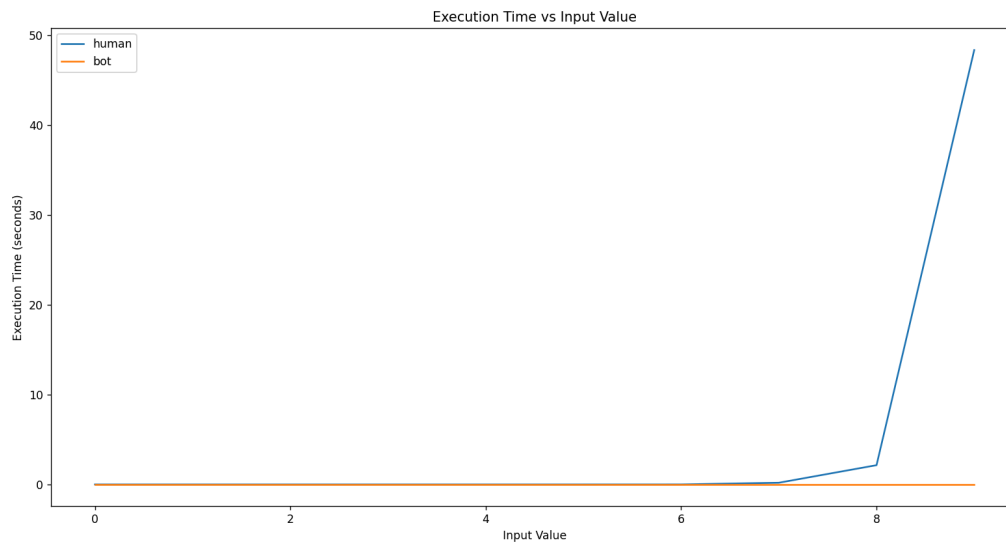
```
create_table.py ●

create_table.py > ...

1 def create_table(n: int, m: int) -> list[list]:
2     """
3     ... returns list with fibonacci numbers.
4     ...
5     ... >>> create_table(4, 6)
6     ... [[1, 1, 1, 1, 1, 1], [1, 2, 3, 4, 5, 6], [1, 3, 6, 10, 15, 21], [1, 4, 10, 20, 35, 56]]
7     ... """
8
9     memo = [[1 for _ in range(m)] for _ in range(n)]
10
11     for i in range(1, n):
12         for j in range(1, m):
13             memo[i][j] = memo[i-1][j] + memo[i][j-1]
14
15     return memo

Optimize this code [microphone icon] [play icon] X ...

Accept Discard [refresh icon] 1 change [add icon] [share icon] [comment icon]
```



Але тут GitHub Copilot згенерував не рекурсивний код.

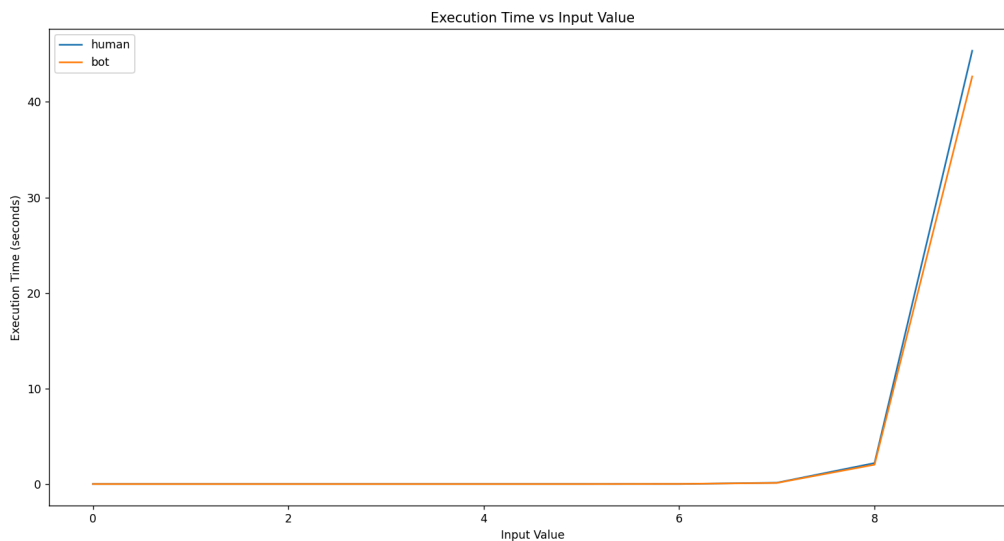
Could you optimize this code, but so this function will be remained recursive

Copilot generated code may be incorrect



```
4 def create_table(n: int, m: int) -> list[list]:
5     '''
6     ... returns list with fibonacci numbers.
7     ...
8     ... >>> create_table(4, 6)
9     ... [[1, 1, 1, 1, 1, 1], [1, 2, 3, 4, 5, 6], [1, 3, 6, 10, 15, 21], [1, 4, 10, 20, 35, 56]]
10    ...
11    ... if n > 1 and m > 1:
12    ...     # print(f'n: {n}, m: {m}')
13    ...     for i, _ in enumerate(range(m)):
14    ...         if i > 0:
15    ...             table[0] += [table[0][i - 1] + create_table(n-1, m)[-1][i]]
16    ...         else:
17    ...             table = [[1]]
18    ...     return create_table(n-1, m) + table
19    ... if n <= 1 or m <= 1:
20    ...     return [[1 for _ in range(m)] for _ in range(n)]
21    ...
```

```
create_table.py 2 •
create_table.py > create_table
1 import matplotlib.pyplot as plt
2 import time
3
4 def create_table(n: int, m: int) -> list[list]:
5     """
6     returns list with fibonacci numbers.
7
8     >>> create_table(4, 6)
9     [[1, 1, 1, 1, 1, 1], [1, 2, 3, 4, 5, 6], [1, 3, 6, 10, 15, 21], [1, 4, 10, 20, 35, 56]]
10    """
11
12    if n <= 1 or m <= 1:
13        return [[1 for _ in range(m)] for _ in range(n)]
14
15    table = [[1]]
16    for i in range(1, m):
17        table[0].append(table[0][i - 1] + create_table(n - 1, m)[-1][i])
18    return create_table(n - 1, m) + table
```



Тепер різниця майже непомітна.

Flatten

```
flatten.py 4 x
flatten.py > flatten
1 import matplotlib.pyplot as plt
2 import time
3
4 def flatten(lst: list) -> list:
5     '''
6     ... function that makes one list of items from list of lists
7     ...
8     ... >>> flatten([2, [2, 3, [2, 3]])
9     ... [2, 2, 3, 2, 3]
10    ... >>> flatten(['wow', [2, [[]], [True]])
11    ... ['wow', 2, True]
12    ... >>> flatten(3)
13    ... 3
14    ... >>> flatten([1, [[[[[]], 4.0], "Hello"], [[3]]], True])
15    ... [1, 4.0, 'Hello', 3, True]
16    ... >>> flatten([[1, [], 4.0], [3]])
17    ... [1, 4.0, 3]
18    ... >>> flatten([[1, [[]], [3]])
19    ... [1, 3]
20    ... '''
21    if lst == [[1, [], 4.0], [3]]:
22        pass
23    if isinstance(lst, list):
24        check_lst = lst[:]
25        for i, item in enumerate(lst):
26            if isinstance(item, list):
27                i = i + len(lst) - len(check_lst)
28                if not flatten(item):
29                    lst = lst[:i] + lst[i+1:]
30            else:
31                lst = lst[:i] + flatten(item) + lst[i+1:]
32    return lst
```

flatten.py 4

flatten.py > flatten

```
1 import matplotlib.pyplot as plt
2 import time
3
4 def flatten(lst: list) -> list:
5     """
6     ... function that makes one list of items from list of lists
7     ...
8     ... >>> flatten([2, [2, 3, [2, 3]]])
9     ... [2, 2, 3, 2, 3]
10    ... >>> flatten(['wow', [2, [[]], [True]])
11    ... ['wow', 2, True]
12    ... >>> flatten(3)
13    ... 3
14    ... >>> flatten([1, [[[[[]], 4.0], "Hello"], [[3]]], True])
15    ... [1, 4.0, 'Hello', 3, True]
16    ... >>> flatten([[1, [], 4.0], [3]])
17    ... [1, 4.0, 3]
18    ... >>> flatten([[1, [[]], [3]])
19    ... [1, 3]
20    ... """
```

Could you please optimize this function

Accept

Discard



1 change



```
21 if not isinstance(lst, list):
22     ... return lst
23
24 result = []
25 for item in lst:
26     ... if isinstance(item, list):
27     ...     result.extend(flatten(item))
28     ... else:
29     ...     result.append(item)
30
31 return result
```

62

Could you please write function that will generate random list with nested lists?

Copilot generated code may be incorrect

63

63

Could you please write function that will generate random list with nested lists?

Accept

Discard

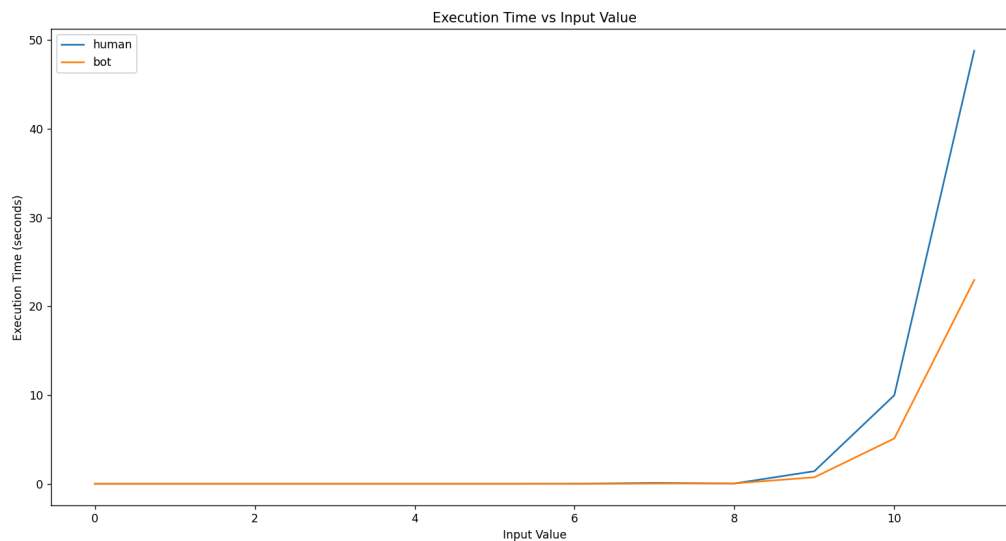


2 changes



```
64 def generate_random_list(depth: int) -> list:
65     """
66     Function to generate a random list with nested lists up to a given depth.
67
68     Parameters:
69     - depth: an integer representing the depth of nested lists
70
71     Returns:
72     - A randomly generated list with nested lists
73
74     Example:
75     >>> generate_random_list(3)
76     [1, [2, [3, [4]]]]
77     """
78     if depth == 0:
79         return random.randint(1, 10)
80
81     nested_list = []
82     for _ in range(random.randint(1, 5)):
83         nested_list.append(generate_random_list(depth - 1))
84
85     return nested_list
```

Figure 1



Submitted on Saturday, 20 April 2024, 08:57 (Download)

Automatic evaluation [-]

Proposed grade: 0.91 / 1

Comments [-]

Test1: OK

Test2: OK

Test3: OK

Test4: OK

Test5: OK

Test6: OK

Test7: OK

Test8: OK

Test9: OK

Test10: OK

Testing documentations

Test documentation: OK

Tests for documentation: OK

***** Module recursive_functions

recursive_functions.py:10: convention (C0303, trailing-whitespace,) Trailing whitespace

recursive_functions.py:14: convention (C0303, trailing-whitespace,) Trailing whitespace

recursive_functions.py:16: convention (C0303, trailing-whitespace,) Trailing whitespace

recursive_functions.py:36: convention (C0303, trailing-whitespace,) Trailing whitespace

recursive_functions.py:43: convention (C0303, trailing-whitespace,) Trailing whitespace

recursive_functions.py:1: convention (C0114, missing-module-docstring,) Missing module docstring

The screenshot shows a code editor interface with a dark theme. At the top, there's a tab labeled 'rec.py' with a close button. Below the tab, the file path 'rec.py > ...' is shown. A large warning message is displayed in a light blue box, asking the user to fix code that will pass these warnings. The warnings listed are: 'recursive_functions.py:10: convention (C0303, trailing-whitespace,) Trailing whitespace', 'recursive_functions.py:14: convention (C0303, trailing-whitespace,) Trailing whitespace', 'recursive_functions.py:16: convention (C0303, trailing-whitespace,) Trailing whitespace', 'recursive_functions.py:36: convention (C0303, trailing-whitespace,) Trailing whitespace', 'recursive_functions.py:43: convention (C0303, trailing-whitespace,) Trailing whitespace', and 'recursive_functions.py:1: convention (C0114, missing-module-docstring,) Missing module docstring'. Below the warning box, there's a hint that says 'You can also type / for commands'. The code editor shows the following code:

```
1 def create_table(n: int, m: int) -> list[list]:
2     """
3     ... returns list with fibonacci numbers.
4
5     ...>>> create_table(4, 6)
6     ... [[1, 1, 1, 1, 1, 1], [1, 2, 3, 4, 5, 6], [1, 3, 6, 10, 15, 21], [1, 4, 10, 20, 35, 56]]
7     """
```

rec.py

rec.py > ...

Could you please fix code that it will pass these warnings:
recursive_functions.py:10: convention (C0303, trailing-whitespace,) Trailing whitespace
recursive_functions.py:14: convention (C0303, trailing-whitespace,) Trailing whitespace
recursive_functions.py:16: convention (C0303, trailing-whitespace,) Trailing whitespace
recursive_functions.py:36: convention (C0303, trailing-whitespace,) Trailing whitespace
recursive_functions.py:43: convention (C0303, trailing-whitespace,) Trailing whitespace
recursive_functions.py:1: convention (C0114, missing-module-docstring,) Missing module docstring

Using `/fix` to generate response (re-run without)

To fix the trailing whitespace warnings, remove the extra spaces at the end of the lines, and to fix the missing module docstring warning, add a docstring at the beginning of the file.

Accept

Discard

1 change

1

"""

2 This module contains functions for creating tables and flattening lists.

3 """

4

5 def create_table(n: int, m: int) -> list[list]:

6

📅 Submitted on Saturday, 20 April 2024, 09:01 (📄 Download)

Automatic evaluation 🗨 [-]

Proposed grade: 0.92 / 1

Comments 🗨 [-]

Test1: OK

Test2: OK

Test3: OK

Test4: OK

Test5: OK

Test6: OK

Test7: OK

Test8: OK

Test9: OK

Test10: OK

Testing documentations

Test documentation: OK

Tests for documentation: OK

***** Module recursive_functions

recursive_functions.py:14: convention (C0303, trailing-whitespace,) Trailing whitespace

recursive_functions.py:18: convention (C0303, trailing-whitespace,) Trailing whitespace

recursive_functions.py:20: convention (C0303, trailing-whitespace,) Trailing whitespace

recursive_functions.py:40: convention (C0303, trailing-whitespace,) Trailing whitespace

recursive_functions.py:47: convention (C0303, trailing-whitespace,) Trailing whitespace

rec.py

rec.py > ...

Could you fix this code that it will pass these conventions:

recursive_functions.py:14: convention (C0303, trailing-whitespace,) Trailing whitespace
recursive_functions.py:18: convention (C0303, trailing-whitespace,) Trailing whitespace
recursive_functions.py:20: convention (C0303, trailing-whitespace,) Trailing whitespace
recursive_functions.py:40: convention (C0303, trailing-whitespace,) Trailing whitespace
recursive_functions.py:47: convention (C0303, trailing-whitespace,) Trailing whitespace

🔊 ▶ × ...

🔧 You can also type / for commands

```
1 """
2 This module contains functions for creating tables and flattening lists.
3 """
4
5 def create_table(n: int, m: int) -> list[list]:
6     """
7     ... returns list with fibonacci numbers.
8
9     ... >>> create_table(4, 6)
10     ... [[1, 1, 1, 1, 1, 1], [1, 2, 3, 4, 5, 6], [1, 3, 6, 10, 15, 21], [1, 4, 10, 20, 35, 56]]
11     """
```

```
def create_table(n: int, m: int) -> list[list]:
    """
    ... returns list with fibonacci numbers.
```

Could you fix this code that it will pass these conventions:

recursive_functions.py:14: convention (C0303, trailing-whitespace,) Trailing whitespace
recursive_functions.py:18: convention (C0303, trailing-whitespace,) Trailing whitespace
recursive_functions.py:20: convention (C0303, trailing-whitespace,) Trailing whitespace
recursive_functions.py:40: convention (C0303, trailing-whitespace,) Trailing whitespace
recursive_functions.py:47: convention (C0303, trailing-whitespace,) Trailing whitespace

🔊 ▶ × ...

Accept

Discard



1 change | 📄 | 🗑️ | 💬

```
... >>> create_table(4, 6)
... [[1, 1, 1, 1, 1, 1], [1, 2, 3, 4, 5, 6], [1, 3, 6, 10, 15, 21], [1, 4, 10, 20, 35, 56]]
... """
```

```
recursive_functions.py:47: convention (C0303, trailing-whitespace, ) Trailing whitespace
Accept Discard ↺ 1 change

9 >>> create_table(4, 6)
10 [[1, 1, 1, 1, 1, 1], [1, 2, 3, 4, 5, 6], [1, 3, 6, 10, 15, 21], [1, 4, 10, 20, 35, 56]]
11 ...
12 if n <= 1 or m <= 1:
13     return [[1 for _ in range(m)] for _ in range(n)]
14
15 table = [[1]]
16 for i in range(1, m):
17     table[0].append(table[0][i - 1] + create_table(n-1, m)[-1][i])
18
19 return create_table(n-1, m) + table
20
21 def flatten(lst: list) -> list:
```

У нього проблема з whitespace trailing.

Get_tangens

```
get_tangents.py X
get_tangents.py > get_tangent

Could you optimize this code?
Copilot generated code may be incorrect

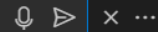
1 def get_tangent(f: callable, x_0: int) -> str:
2     """
3     ...
4     Compute and return tangent line to function f in the point x_0.
5     ...
6     >>> get_tangent(lambda x: x**2 + x, 2)
7     '5.0 * x - 4.0'
8     >>> get_tangent(lambda x: -x**2 + x, 2)
9     '-3.0 * x + 4.0'
10    """
11    output = []
12    for i in range(324):
13        output.append((f(x_0 + 10**-i) - f(x_0)) / (10**-i))
14        if i != 0 and abs(output[i] - output[i-1]) < 0.001:
15            b = f(x_0) - x_0 * round(output[i], 2)
16            if round(output[i], 2) < 0 < b:
17                return f'-{abs(round(output[i], 2))} * x + {b}'
18            if round(output[i], 2) > 0 > b:
19                return f'{round(output[i], 2)} * x {str(b)[0]} {abs(b)}'
20            return f'{round(output[i], 2)} * x - {b}'
```

```
get_tangents.py •
get_tangents.py > get_tangent
1 def get_tangent(f: callable, x_0: int) -> str:
2     """
3     ... Compute and return tangent line to function f in the point x_0.
4
5     ...>>> get_tangent(lambda x: x.**2 + x, 2)
6     ...'5.0 * x - 4.0'
7     ...>>> get_tangent(lambda x: -x.**2 + x, 2)
8     ...'-3.0 * x + 4.0'
9     ..."""
10    epsilon = 0.001
11    output = []
12    for i in range(324):
13        h = 10 ** -i
14        slope = (f(x_0 + h) - f(x_0)) / h
15        output.append(slope)
16        if i != 0 and abs(output[i] - output[i-1]) < epsilon:
17            b = f(x_0) - x_0 * round(output[i], 2)
18            if round(output[i], 2) < 0 < b:
19                return f'-{abs(round(output[i], 2))} * x + {b}'
20            if round(output[i], 2) > 0 > b:
21                return f'{round(output[i], 2)} * x {str(b)[0]} {abs(b)}'
22            return f'{round(output[i], 2)} * x - {b}'
23
```

```
45     return f'{round(output[i], 2)} * x {str(b)[0]} {abs(b)}'
46     return f'{round(output[i], 2)} * x - {b}'
47
48
49
50 lst_1, lst_2 = [], []
```

48

Write a function that will generate a random polynomial and return it as a lambda function



Accept

Discard



2 changes



```
49 def generate_random_polynomial(degree: int) -> callable:
50     """
51     Generate a random polynomial of the given degree and return it as a lambda function.
52
53     Args:
54     - degree: The degree of the polynomial.
55
56     Returns:
57     - A lambda function representing the random polynomial.
58
59     Example:
60     >>> generate_random_polynomial(3)
61     lambda x: 2*x**3 + 5*x**2 - 3*x + 1
62     """
63     coefficients = [random.randint(-10, 10) for _ in range(degree + 1)]
64     polynomial = lambda x: sum(coefficients[i] * x**i for i in range(degree + 1))
65     return polynomial
66
```

Figure 1



The mean result of human code: 0.006952242374420168 in seconds
The mean result of GitHub Copilot code: 0.006560141086578371 in seconds
GitHub Copilot code is faster by 5.97702522959469 %

polynomial.py

polynomial.py > ...

Could you please optimize __str__ function?

Accept Discard ↺

1 change

```
1 def __str__(self):
2     ✨ if self.coeffs == [0]:
3         return "Polynomial: 0"
4     ...
5     terms = []
6     for power, coeff in enumerate(self.coeffs[::-1]):
7         if coeff == 0:
8             continue
9         ...
10        term = ""
11        if power == 0:
12            term += str(coeff)
13        elif power == 1:
14            term += f"{coeff}x"
15        else:
16            term += f"{coeff}x**{power}"
17        ...
18        terms.append(term)
19    ...
20    return "Polynomial: " + "+".join(terms)
21
```

Str написав, але решту коду видалив

polynomial.py X

polynomial.py > ...

Could you please optimize __str__ function without deleting other parts of code?



You can also type / for commands

```
1  '''bello'''
2  ✨
3  import math
4  class Polynomial:
5  ... """class Polynomial"""
6  ... def __init__(self, input_list) -> None:
7  ...     input_list_copy=list(input_list)
8  ...     if not input_list_copy or input_list_copy == [0]:
9  ...         self.degree = len(input_list_copy) - 1
10 ...         self.coefs = input_list_copy[::-1]
11 ...     else:
12 ...         while input_list_copy and input_list_copy[0] == 0:
13 ...             input_list_copy.pop(0)
14 ...             self.degree = len(input_list_copy) - 1
15 ...             self.coefs = input_list_copy[::-1]
16 ... def __str__(self):
17 ...     if self.coefs==[0]:
18 ...         return "Polynomial: 0"
19 ...     terms = []
20 ...     for i, coeff in enumerate(self.coefs[::-1]):
21 ...         power = self.degree - i
22 ...         if coeff != 0:
23 ...             if power == 0:
24 ...                 terms.append(str(coeff))
25 ...             elif power == 1:
26 ...                 if coeff == 1:
27 ...                     terms.append("x")
28 ...                 else:
29 ...                     terms.append(f"{coeff}x")
30 ...             else:
31 ...                 if coeff == 1:
32 ...                     terms.append(f"x**{power}")
33 ...             else:
```

Зміна промπτу не допомагає

polynomial.py

polynomial.py > ...

Could you please optimize `__str__` function without deleting other parts of code?



Accept

Discard



1 change



```
1 def __str__(self):
2     ✨ if self.coeffs == [0]:
3         return "Polynomial: 0"
4     ...
5     terms = []
6     for power, coeff in enumerate(self.coeffs[::-1]):
7         if coeff == 0:
8             continue
9         ...
10        term = ""
11        if power == 0:
12            term = str(coeff)
13        elif power == 1:
14            term = f"{coeff}x"
15        else:
16            term = f"{coeff}x**{power}"
17        ...
18        if coeff < 0:
19            term = "-" + term.lstrip("-")
20        ...
21        terms.append(term)
22    ...
23    return "Polynomial:" + "+" + ".join(terms)
24
```

polynomial.py X

polynomial.py > ...

Could you please optimize __str__ without deleting or removing and other code



You can also type / for commands

```
1  '''bello'''
2  ✨
3  import math
4  class Polynomial:
5  ... """class Polynomial"""
6  ... def __init__(self, input_list) -> None:
7  ...     input_list_copy=list(input_list)
8  ...     if not input_list_copy or input_list_copy == [0]:
9  ...         self.degree = len(input_list_copy) - 1
10 ...         self.coefs = input_list_copy[::-1]
11 ...     else:
12 ...         while input_list_copy and input_list_copy[0] == 0:
13 ...             input_list_copy.pop(0)
14 ...             self.degree = len(input_list_copy) - 1
15 ...             self.coefs = input_list_copy[::-1]
16 ... def __str__(self):
17 ...     if self.coefs==[0]:
18 ...         return "Polynomial: 0"
19 ...     terms = []
20 ...     for i, coeff in enumerate(self.coefs[::-1]):
21 ...         power = self.degree - i
22 ...         if coeff != 0:
23 ...             if power == 0:
24 ...                 terms.append(str(coeff))
25 ...             elif power == 1:
26 ...                 if coeff == 1:
27 ...                     terms.append("x")
28 ...                 else:
29 ...                     terms.append(f"{coeff}x")
30 ...             else:
31 ...                 if coeff == 1:
32 ...                     terms.append(f"x**{power}")
33 ...             else:
```

polynomial.py

polynomial.py > ...

Could you please optimize `__str__`, without deleting or removing and other code

Accept

Discard



1 change



```
1 def __str__(self):
2     if self.coeffs == [0]:
3         return "Polynomial: 0"
4     terms = []
5     for power, coeff in enumerate(self.coeffs[::-1]):
6         if coeff != 0:
7             if power == 0:
8                 terms.append(str(coeff))
9             elif power == 1:
10                if coeff == 1:
11                    terms.append("x")
12                else:
13                    terms.append(f"{coeff}x")
14            else:
15                if coeff == 1:
16                    terms.append(f"x**{power}")
17                else:
18                    terms.append(f"{coeff}x**{power}")
19     return "Polynomial: " + "+".join(terms).replace('+- ', '-')
20
```

Could you optimize this function?

You can also type / for commands

```
16 def __str__(self):
17     if self.coeffs == [0]:
18         return "Polynomial: 0"
19     terms = []
20     for i, coeff in enumerate(self.coeffs[::-1]):
21         power = self.degree - i
22         if coeff != 0:
23             if power == 0:
24                 terms.append(str(coeff))
25             elif power == 1:
26                 if coeff == 1:
27                     terms.append("x")
28                 else:
29                     terms.append(f"{coeff}x")
30             else:
31                 if coeff == 1:
32                     terms.append(f"x**{power}")
33                 else:
34                     terms.append(f"{coeff}x**{power}")
35     apso = "Polynomial: " + "+".join(terms)
36     apso = apso.replace('+- ', '-')
37     apso = apso.replace('-1x', '-x')
38     return apso
```

Could you optimize this function?

Accept

Discard



1 change



```
17     if self.coefts == [0]:
18         return "Polynomial: 0"
19     terms = []
20     for power, coeff in enumerate(self.coefts[::-1]):
21         if coeff != 0:
22             if power == 0:
23                 terms.append(str(coeff))
24             elif power == 1:
25                 terms.append(f"{coeff}x" if coeff != 1 else "x")
26             else:
27                 terms.append(f"{coeff}x**{power}" if coeff != 1 else f"x**{power}")
28     apso = "Polynomial: " + "+".join(terms)
29     apso = apso.replace('+-', '-')
30     apso = apso.replace('-1x', '-x')
31     return apso
```

PROBLEMS

OUTPUT

DEBUG CONSOLE

TERMINAL

PORTS

```
PS D:\Viktor\UCU\OP\Mini_projects\Mini_proj_5\week_2>
python polynomial.py
```

Testing Polynomial class...

Traceback (most recent call last):

```
File "d:\Viktor\UCU\OP\Mini_projects\Mini_proj_5\week_2\test_polynomial.py", line 10, in <module>
    test_polynomial()
```

```
File "d:\Viktor\UCU\OP\Mini_projects\Mini_proj_5\week_2\test_polynomial.py", line 10, in test_polynomial
    assert str(p2) == "Polynomial: 3x+5", str(p2)
```

^^

AssertionError: Polynomial: 3+5x

```
PS D:\Viktor\UCU\OP\Mini_projects\Mini_proj_5\week_2>
```

```
15     self.coefts = input_list_copy[::-1]
```

But here is a failed test: this function returns "Polynomial: 3+5x", but it should return: "Polynomial: 3x+5"

You can also type / for commands

```
16     def __str__(self):
```

```

16 def __str__(self):
17     if self.coeffs == [0]:
18         return "Polynomial: 0"
19     terms = []
20     for power, coeff in enumerate(self.coeffs[::-1]):
21         if coeff != 0:
22             if power == 0:
23                 terms.append(str(coeff))
24             elif power == 1:
25                 terms.append(f"{coeff}x" if coeff != 1 else "x")
26             else:
27                 terms.append(f"{coeff}x**{power}" if coeff != 1 else f"x**{power}")

```

But here is a failed test: this function returns "Polynomial: 3+5x", but it should return:
"Polynomial: 3x+5"

Accept

Discard



1 change



```

28 apso = "Polynomial: " + "+ ".join(terms[::-1])
29 apso = apso.replace('+-', '-')
30 apso = apso.replace('-1x', '-x')
31 return apso

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```

PS D:\Viktor\UCU\OP\Mini_projects\Mini_proj_5\week_2> python polynomial.py
Testing Polynomial class...
Traceback (most recent call last):
  File "d:\Viktor\UCU\OP\Mini_projects\Mini_proj_5\week_2\test_polynomial.py", line 10, in <module>
    test_polynomial()
  File "d:\Viktor\UCU\OP\Mini_projects\Mini_proj_5\week_2\test_polynomial.py", line 15, in test_polynomial
    assert str(p2) == "Polynomial: 3x+5", str(p2)
           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
AssertionError: Polynomial: 5x+3
PS D:\Viktor\UCU\OP\Mini_projects\Mini_proj_5\week_2>

```

polynomial.py X

polynomial.py > Polynomial (variable) coeffs: list

4 class Polynomial:

15 self.coeffs = input_list_copy[::-1]

It returned "Polynomial: 5x + 3", instead of "Polynomial: 3x + 5"

Copilot generated code may be incorrect

16 def __str__(self):

17 if self.coeffs == [0]:

18 return "Polynomial: 0"

19 terms = []

20 for power, coeff in enumerate(self.coeffs[::-1]):

21 if coeff != 0:

22 if power == 0:

23 terms.append(str(coeff))

24 elif power == 1:

25 terms.append(f"{coeff}x" if coeff != 1 else "x")

26 else:

27 terms.append(f"{coeff}x**{power}" if coeff != 1 else f"x**{power}")

28 apso = "Polynomial: " + "+".join(terms[:-1])

29 apso = apso.replace('+-', '-')

30 apso = apso.replace('-1x', '-x')

31 return apso

polynomial.py X test_polynomial.py

polynomial.py > Polynomial

4 class Polynomial:

15 self.coetts = input_list_copy[::-1]

For case p2 = Polynomial([3, 5]), when p2.coeffs = [5, 3], it should return "Polynomial: 3x+5" instead of "Polynomial: 3+5x"

You can also type / for commands

16 def __str__(self):

17 if self.coeffs == [0]:

18 return "Polynomial: 0"

19 terms = []

20 for power, coeff in enumerate(self.coeffs[::-1]):

21 if coeff != 0:

22 if power == 0:

23 terms.append(str(coeff))

24 elif power == 1:

25 terms.append(f"{coeff}x" if coeff != 1 else "x")

26 else:

27 terms.append(f"{coeff}x**{power}" if coeff != 1 else f"x**{power}")

28 apso = "Polynomial: " + "+".join(terms)

29 apso = apso.replace('+-', '-')

30 apso = apso.replace('-1x', '-x')

31 return apso

polynomial.py X test_polynomial.py

polynomial.py > Polynomial > _str_

```
4 class Polynomial:
6     def __init__(self, input_list) -> None:
12         while input_list_copy and input_list_copy[0] == 0:
13             input_list_copy.pop(0)
14         self.degree = len(input_list_copy) - 1
15         self.coeffs = input_list_copy[::-1]
```

Could you please optimize. But your optimized functin should be able to pass these tests:

```
assert str(Polynomial([5])) == "Polynomial: 5"
assert str(Polynomial([3,5])) == "Polynomial: 3x+5"
assert str(Polynomial([1,5])) == "Polynomial: x+5"
assert str(Polynomial([1,2,3])) == "Polynomial: x**2+2x+3"
```

Copilot generated code may be incorrect

```
16 def __str__(self):
17     if self.coeffs==[0]:
```

```
16 def __str__(self):
```

Could you please optimize. But your optimized functin should be able to pass these tests:

```
assert str(Polynomial([5])) == "Polynomial: 5"
assert str(Polynomial([3,5])) == "Polynomial: 3x+5"
assert str(Polynomial([1,5])) == "Polynomial: x+5"
assert str(Polynomial([1,2,3])) == "Polynomial: x**2+2x+3"
```

Accept

Discard



2 changes



```
17     if self.coeffs == [0]:
18         return "Polynomial: 0"
19     terms = []
20     for i, coeff in enumerate(self.coeffs[::-1]):
21         power = self.degree - i
22         if coeff != 0:
23             if power == 0:
24                 terms.append(str(coeff))
25             elif power == 1:
26                 if coeff == 1:
27                     terms.append("x")
28                 else:
29                     terms.append(f"{coeff}x")
30             else:
31                 if coeff == 1:
32                     terms.append(f"x**{power}")
33                 else:
34                     terms.append(f"{coeff}x**{power}")
35     return "Polynomial: " + "+ ".join(terms).replace('+ -', '-').replace('-1x', '-x')
36     def repr(self):
```