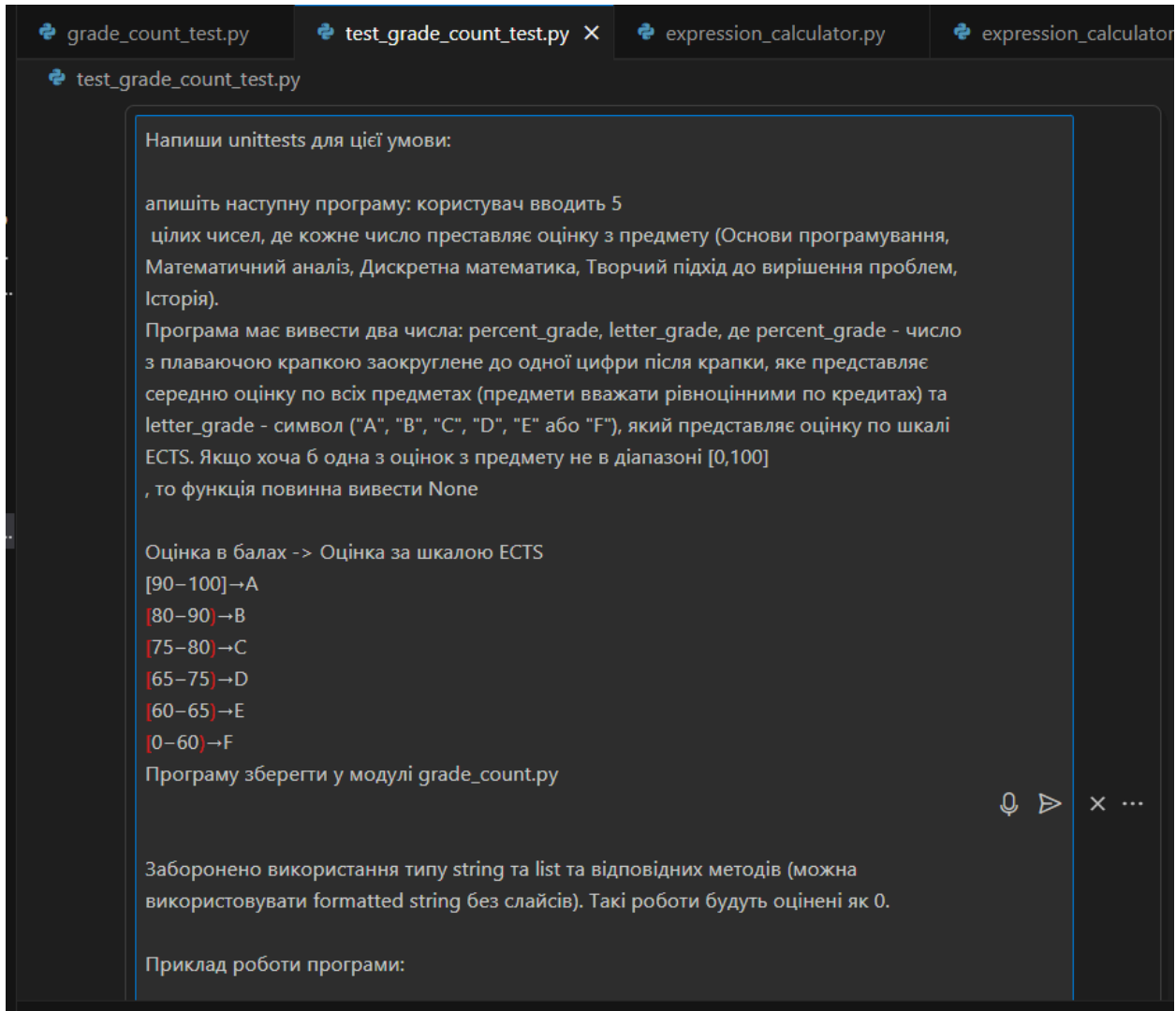


Grade to letter:



The screenshot shows a code editor with four tabs: `grade_count_test.py`, `test_grade_count_test.py` (active), `expression_calculator.py`, and `expression_calculator`. The active tab contains the following text:

```
test_grade_count_test.py
```

Напиши unittests для цієї умови:

напишіть наступну програму: користувач вводить 5 цілих чисел, де кожне число представляє оцінку з предмету (Основи програмування, Математичний аналіз, Дискретна математика, Творчий підхід до вирішення проблем, Історія).

Програма має вивести два числа: `percent_grade`, `letter_grade`, де `percent_grade` - число з плаваючою крапкою заокруглене до одної цифри після крапки, яке представляє середню оцінку по всіх предметах (предмети вважати рівноцінними по кредитах) та `letter_grade` - символ ("A", "B", "C", "D", "E" або "F"), який представляє оцінку по шкалі ECTS. Якщо хоча б одна з оцінок з предмету не в діапазоні [0,100], то функція повинна вивести `None`

Оцінка в балах -> Оцінка за шкалою ECTS

[90–100]	→A
[80–90]	→B
[75–80]	→C
[65–75]	→D
[60–65]	→E
[0–60]	→F

Програму зберегти у модулі `grade_count.py`

Заборонено використання типу `string` та `list` та відповідних методів (можна використовувати `formatted string` без слайсів). Такі роботи будуть оцінені як 0.

Приклад роботи програми:

Accept Discard ↺ 1 change

```
1 import unittest
2 from grade_count import calculate_grade
3
4 class TestGradeCount(unittest.TestCase):
5     def test_average_grade_A(self):
6         result = calculate_grade(85, 90, 67, 70, 87)
7         self.assertEqual(result, "Average grade = 79.8 -> C")
8
9     def test_average_grade_B(self):
10        result = calculate_grade(97, 93, 84, 78, 80)
11        self.assertEqual(result, "Average grade = 86.4 -> B")
12
13    def test_invalid_grade(self):
14        result = calculate_grade(85, 90, 67, 70, 105)
15        self.assertIsNone(result)
16
17 if __name__ == '__main__':
18     unittest.main()
```

Запустив тест, один тест впав там, де ввів 105

```
grade_count.py  test_grade_count_test.py X  expression_calculator.py

test_grade_count_test.py > ...
1  import unittest
2  from grade_count import calculate_grade
3
4  class TestGradeCount(unittest.TestCase):
5      def test_average_grade_A(self):
6          result = calculate_grade(85, 90, 67, 70, 87)
7          self.assertEqual(result, "Average grade = 79.8 -> C")
8
9      def test_average_grade_B(self):
10         result = calculate_grade(97, 93, 84, 78, 80)
11         self.assertEqual(result, "Average grade = 86.4 -> B")
12
13     def test_invalid_grade(self):
14         result = calculate_grade(85, 90, 67, 70, 105)
15         self.assertIsNone(result)
16
17 if __name__ == '__main__':
18     unittest.main()
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
rade_count_test.py
..F
=====
FAIL: test_invalid_grade (__main__.TestGradeCount.test_invalid_grade)
-----
Traceback (most recent call last):
  File "d:\Viktor\UCU\OP\Mini_projects\Mini_proj_5\week_3\test_grade_count_
    self.assertIsNone(result)
AssertionError: 'Average grade = 83.4 -> B' is not None
-----

Ran 3 tests in 0.001s

FAILED (failures=1)
PS D:\Viktor\UCU\OP\Mini_projects\Mini_proj_5\week_3> |
```

Тут стає зрозуміло, що я зробив в коді не так та, що варто виправити

```
if a < 0 or a > 100 or b < 0 or b > 100 or c < 0 or c > 100 or d < 0 or d > 100 or e < 0 or e > 100:  
    return
```

Поміняв перевірку вводу

Походить всі тести

```
OK  
PS D:\Viktor\UCU\OP\Mini_p  
rade_count_test.py  
...  
-----  
Ran 3 tests in 0.001s  
  
OK  
PS D:\Viktor\UCU\OP\Mini_p
```

📅 Submitted on Saturday, 27 April 2024, 21:35 (📄 Download)

Automatic evaluation 🔄 [-]

Proposed grade: 1 / 1

Comments 🗨 [-]

[+]Summary of tests

grade_count.py

```
1 a=int(input())  
2 b=int(input())  
3 c=int(input())  
4 d=int(input())  
5 e=int(input())  
6 ▾ if a < 0 or a > 100 or b < 0 or b > 100 or c < 0 or c > 100 or d < 0 or d > 100 or e < 0 or e > 100:  
7     print(None)  
8     exit()  
9  
10 grade=round(((a+b+c+d+e)/5), 1)  
11  
12 ▾ if 100>=grade>=90:  
13     print(f"Average grade = {grade} -> A")  
14 ▾ elif 90>grade>=80:  
15     print(f"Average grade = {grade} -> B")  
16 ▾ elif 80>grade>=70:  
17     print(f"Average grade = {grade} -> C")  
18 ▾ elif 70>grade>=60:  
19     print(f"Average grade = {grade} -> D")  
20 ▾ else:  
21     print(f"Average grade = {grade} -> F")  
22
```

Calculate expression

Напиши unittests для цієї умови:

У модулі expression_calculator.py напишіть функцію calculate_expression, яка буде приймати єдиний аргумент - стрічку з простим формулюванням математичного виразу та повертати ціле число - результат цього виразу.

Наприклад: параметр функції – «Скільки буде 5 додати 5?», результат функції – 10; або «Скільки буде 2 помножити на 10 додати 7?», результат функції – 27. Для простоти функція не має враховувати пріоритетність операцій, тобто результатом виразу «Скільки буде 2 додати 10 помножити на 7?» буде 84, а не 72.

Операції, які підтримує функція – "додати"/"плюс", "відняти"/"мінус", "помножити на", "поділити на". Функція також має вертати стрічку «Неправильний вираз!», якщо вираз не правильний, а саме:

Операції які не підтримуються («Скільки буде 3 в квадраті?»)

Не математичні вирази («Скільки сезонів в році?»)

Математичні операції з неправильним синтаксисом («Скільки буде 2 2 додати?»)

Приклад

```
>>> calculate_expression("Скільки буде 8 відняти 3?")
```

```
5
```

```
>>> calculate_expression("Скільки буде 7 додати 3 помножити на 5?")
```

```
50
```

```
>>> calculate_expression("Скільки буде 10 поділити на -2 додати 11 мінус -3?")
```

```
9
```

```
>>> calculate_expression("Скільки буде 3 в кубі?")
```

```
'Неправильний вираз!'
```



Copilot generated code may be incorrect

Створив новий файл з тестами

```
grade_count.py  test_grade_count_test.py  expression_calculator.py  test_expression_calculator.py X
test_expression_calculator.py > ...
1  import unittest
2  from expression_calculator import calculate_expression
3
4  class TestExpressionCalculator(unittest.TestCase):
5      def test_addition(self):
6          result = calculate_expression("Скільки буде 5 додати 5?")
7          self.assertEqual(result, 10)
8
9      def test_multiplication_and_addition(self):
10         result = calculate_expression("Скільки буде 2 помножити на 10 додати 7?")
11         self.assertEqual(result, 27)
12
13     def test_division_and_subtraction(self):
14         result = calculate_expression("Скільки буде 10 поділити на -2 додати 11 мінус -3?")
15         self.assertEqual(result, 9)
16
17     def test_invalid_expression(self):
18         result = calculate_expression("Скільки буде 3 в кубі?")
19         self.assertEqual(result, "Неправильний вираз!")
20
21 if __name__ == '__main__':
22     unittest.main()
```

Запустив тести, видно, які моменти не пройшли та, які варто допрацювати

history_analysis.py × history_analysis_test.py × expression_calculator_test.py

history_analysis_test.py > ...

Напиши unittests для цієї умови:

Requested files: history_analysis.py, get_browser_history.py (Download)

Type of work: Individual work

Ваше завдання - спробувати опрацювати дані з історії переглядів браузера Google Chrome. Вам надається готовий модуль `get_browser_history.py`, який згенерує список з історії відвідувань та запише його у файл. Для роботи з модулем завантажте його, виберіть вашу ОС та введіть ім'я користувача, щоб утворити коректний шлях до бази даних браузера. Ці значення передавайте як аргументи при виклику функцій модуля. Також, якщо у вашого браузера кілька профілів, Ви можете змінити значення `Default` на `Profile 1` або інші. Зауважте, що відповідний браузер має бути закритим, коли Ви стягуєте історію. Також список може бути згенерований не у хронологічному порядку.

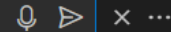
Ви повинні імплементувати наступні функції у модулі `history_analysis.py`:

`sites_on_date(visits, date)` - отримує список-історію браузера та дату у форматі стрічки "yyyy-mm-dd", повертає множину url-адрес, які були відвідані в цей день

`most_frequent_sites(visits, number)` - отримує список-історію браузера та число, а повертає множину найбільш частих url-адрес. Аргумент `number` - число, яке задає, скільки саме url-адрес потрібно повернути. Якщо це число більше за загальну кількість різних url-адрес в `visits`, тоді функція повертає множину усіх відвіданих адрес

`get_url_info(visits, url)` - отримує список-історію браузера та url-адресу. Для заданої адреси повертає заголовок сайту, дату останнього відвідування у форматі "yyyy-mm-dd", час останнього відвідування у форматі "hh:mm:ss.ms", кількість відвідувань, середню тривалість перебування на цій адресі.

Для виконання завдання вартує використати дані, отримані завдяки модулю `get_browser_history.py`. Він генерує файл, в якому в кожному рядку розташована інформація про одне відвідування певного сайту. Дані про кожне відвідування збережені у форматі tuple в наступному порядку:



Accept

Discard



1 change



```
1 import unittest
2 from history_analysis import sites_on_date, most_frequent_sites, get_url_info
3
4 class TestHistoryAnalysis(unittest.TestCase):
5
6     def setUp(self):
7         self.visits = [
8             ('https://youtube.com/', 'YouTube', '2023-10-23', '11:55:31.705320', 0),
9             ('https://youtube.com/', 'YouTube', '2023-10-22', '11:55:31.705320', 10834),
10            ('https://www.instagram.com/', 'Instagram', '2023-10-23', '11:55:39.190008', 1),
11            ('https://cms.ucu.edu.ua/my/index.php', 'Dashboard', '2023-10-23', '11:56:00.000000', 1),
12            ('https://www.instagram.com/', 'Instagram', '2023-10-23', '11:55:39.655681', 1),
13            ('https://youtube.com/', 'YouTube', '2023-10-22', '11:55:34.348591', 10061)
14        ]
15
16     def test_sites_on_date(self):
17         self.assertEqual(sites_on_date(self.visits, '2023-10-22'), {'https://www.youtube.com/': 10834})
18         self.assertEqual(sites_on_date(self.visits, '2023-10-23'), {'https://youtube.com/': 1, 'https://www.instagram.com/': 2, 'https://cms.ucu.edu.ua/my/index.php': 1})
19
20     def test_most_frequent_sites(self):
21         self.assertEqual(most_frequent_sites(self.visits, 10), {'https://www.youtube.com/': 10834})
22         self.assertEqual(most_frequent_sites(self.visits, 2), {'https://www.instagram.com/': 2, 'https://youtube.com/': 1})
23
24     def test_get_url_info(self):
25         self.assertEqual(get_url_info(self.visits, 'https://youtube.com/'), ('YouTube', 'https://youtube.com/'))
26         self.assertEqual(get_url_info(self.visits, 'https://www.instagram.com/'), ('Instagram', 'https://www.instagram.com/'))
27         self.assertEqual(get_url_info(self.visits, 'https://cms.ucu.edu.ua/my/index.php'), ('Dashboard', 'https://cms.ucu.edu.ua/my/index.php'))
28
29 if __name__ == '__main__':
30     unittest.main()
31
```

```

=====
FAIL: test_most_frequent_sites (__main__.TestHistoryAnalysis.test_most_frequent_sites)
-----
Traceback (most recent call last):
  File "d:\Viktor\UCU\OP\Mini_projects\Mini_proj_5\week_3\history_analysis_test.py", line 22, in test_most_frequent_sites
    self.assertEqual(most_frequent_sites(self.visits, 2), {'https://www.instagram.com/', 'https://youtube.com/'})
AssertionError: Items in the first set but not the second:
'https://cms.ucu.edu.ua/my/index.php'
Items in the second set but not the first:
'https://www.instagram.com/'

=====
FAIL: test_sites_on_date (__main__.TestHistoryAnalysis.test_sites_on_date)
-----
Traceback (most recent call last):
  File "d:\Viktor\UCU\OP\Mini_projects\Mini_proj_5\week_3\history_analysis_test.py", line 17, in test_sites_on_date
    self.assertEqual(sites_on_date(self.visits, '2023-10-22'), {'https://www.youtube.com/'})
AssertionError: Items in the first set but not the second:
'https://youtube.com/'
Items in the second set but not the first:
'https://www.youtube.com/'

-----
Ran 3 tests in 0.002s

```

Забув написати www для деяких посилань на ютубі

```

history_analysis.py  history_analysis_test.py  expression_calculator_test.py
history_analysis_test.py > ...
You forgot to add "www" to some links
You can also type / for commands

1 import unittest
2 from history_analysis import sites_on_date, most_frequent_sites, get_url_info
3
4 class TestHistoryAnalysis(unittest.TestCase):
5
6     def setUp(self):
7         self.visits = [
8             ('https://youtube.com/', 'YouTube', '2023-10-23', '11:55:31.705320', 0),
9             ('https://youtube.com/', 'YouTube', '2023-10-22', '11:55:31.705320', 1083473),
10            ('https://www.instagram.com/', 'Instagram', '2023-10-23', '11:55:39.190008', 467439),
11            ('https://cms.ucu.edu.ua/my/index.php', 'Dashboard', '2023-10-23', '11:56:03.912323', 0),
12            ('https://www.instagram.com/', 'Instagram', '2023-10-23', '11:55:39.655687', 414254256),
13            ('https://youtube.com/', 'YouTube', '2023-10-22', '11:55:34.348591', 1006174780)
14        ]
15
16     def test_sites_on_date(self):
17         self.assertEqual(sites_on_date(self.visits, '2023-10-22'), {'https://www.youtube.com/'})
18         self.assertEqual(sites_on_date(self.visits, '2023-10-23'), {'https://youtube.com/', 'https://www.instagram.com/', 'https://cms.ucu.edu.ua/
19

```

Поміняв для ютубу не для cms, але видалив частину коду

```
history_analysis.py history_analysis_test.py expression_calculator_test.py
history_analysis_test.py > ...
1 import unittest
2 from history_analysis import sites_on_date, most_frequent_sites, get_url_info
3
4 class TestHistoryAnalysis(unittest.TestCase):
5
6     def setUp(self):
7         self.visits = [
8             ('https://www.youtube.com/', 'YouTube', '2023-10-23', '11:55:31.705320', 0),
9             ('https://www.youtube.com/', 'YouTube', '2023-10-22', '11:55:31.705320', 1083473),
10            ('https://www.instagram.com/', 'Instagram', '2023-10-23', '11:55:39.190008', 467439),
11            ('https://cms.ucu.edu.ua/my/index.php', 'Dashboard', '2023-10-23', '11:56:03.912323', 0),
12            ('https://www.instagram.com/', 'Instagram', '2023-10-23', '11:55:39.655687', 41425425),
13            ('https://www.youtube.com/', 'YouTube', '2023-10-22', '11:55:34.348591', 1006174780)
14        ]
15
16     # Rest of the code remains the same
17
```

You forget to add "www" to some links

Accept Discard 1 change

Не помогло

```
history_analysis.py history_analysis_test.py X expression_calculator_test.py
history_analysis_test.py > TestHistoryAnalysis
1 import unittest
2 from history_analysis import sites_on_date, most_frequent_sites, get_url_info
3
4 class TestHistoryAnalysis(unittest.TestCase):
5
6     def setUp(self):
7         self.visits = [
8             ('https://www.youtube.com/', 'YouTube', '2023-10-23', '11:55:31.705320', 0),
9             ('https://www.youtube.com/', 'YouTube', '2023-10-22', '11:55:31.705320', 1083473),
10            ('https://www.instagram.com/', 'Instagram', '2023-10-23', '11:55:39.190008', 467439),
11            ('https://cms.ucu.edu.ua/my/index.php', 'Dashboard', '2023-10-23', '11:56:03.912323', 0),
12            ('https://www.instagram.com/', 'Instagram', '2023-10-23', '11:55:39.655687', 414254256),
13            ('https://www.youtube.com/', 'YouTube', '2023-10-22', '11:55:34.348591', 1006174780)
14        ]
15
16     def test_sites_on_date(self):
17         self.assertEqual(sites_on_date(self.visits, '2023-10-22'), {'https://www.youtube.com/'})
18         self.assertEqual(sites_on_date(self.visits, '2023-10-23'), {'https://youtube.com/', 'https://www
```

You forget to add www to cms and youtube links for setup and other tests

Copilot generated code may be incorrect

Додав з декількох спроб

```

10         ('https://www.instagram.com/', 'Instagram', '2023-10-23', '11:55:39.190008', 467439),
11     ]
12     ('https://www.cms.ucu.edu.ua/my/index.php', 'Dashboard', '2023-10-23', '11:56:03.912323', 0),
13     ('https://www.instagram.com/', 'Instagram', '2023-10-23', '11:55:39.655687', 414254256),
14     ('https://www.youtube.com/', 'YouTube', '2023-10-22', '11:55:34.348591', 1006174780)
15 ]
16
17 def test_sites_on_date(self):
18     self.assertEqual(sites_on_date(self.visits, '2023-10-22'), {'https://www.youtube.com/'})
19     self.assertEqual(sites_on_date(self.visits, '2023-10-23'), {'https://www.youtube.com/', 'https://www.instagram.com/', 'https://www.cms.ucu.edu.ua/my/index.php'})
20
21 def test_most_frequent_sites(self):
22     self.assertEqual(most_frequent_sites(self.visits, 10), {'https://www.instagram.com/', 'https://www.youtube.com/', 'https://www.cms.ucu.edu.ua/my/index.php'})
23     self.assertEqual(most_frequent_sites(self.visits, 2), {'https://www.instagram.com/', 'https://www.youtube.com/'})
24
25 def test_get_url_info(self):
26     self.assertEqual(get_url_info(self.visits, 'https://www.youtube.com/'), ('YouTube', '2023-10-23', '11:55:31.705320', 2, 542369736.5))
27     self.assertEqual(get_url_info(self.visits, 'https://www.instagram.com/'), ('Instagram', '2023-10-23', '11:55:39.655687', 2, 207827347.5))
28     self.assertEqual(get_url_info(self.visits, 'https://www.cms.ucu.edu.ua/my/index.php'), ('Dashboard', '2023-10-23', '11:56:03.912323', 1, 0))
29
30 if __name__ == '__main__':
31     unittest.main()

```

Прогнав тести, зрозумів в чому проблеми

```

=====
FAIL: test_get_url_info (__main__.TestHistoryAnalysis.test_get_url_info)
-----
Traceback (most recent call last):
  File "d:\Viktor\UCU\OP\Mini_projects\Mini_proj_5\week_3\history_analysis_test.py", line 25, in test_get_url_info
    self.assertEqual(get_url_info(self.visits, 'https://www.youtube.com/'), ('YouTube', '2023-10-23', '11:55:31.705320', 2, 542369736.5))
AssertionError: None != ('YouTube', '2023-10-23', '11:55:31.705320', 2, 542369736.5)

=====
FAIL: test_most_frequent_sites (__main__.TestHistoryAnalysis.test_most_frequent_sites)
-----
Traceback (most recent call last):
  File "d:\Viktor\UCU\OP\Mini_projects\Mini_proj_5\week_3\history_analysis_test.py", line 22, in test_most_frequent_sites
    self.assertEqual(most_frequent_sites(self.visits, 2), {'https://www.instagram.com/', 'https://www.youtube.com/'})
AssertionError: Items in the first set but not the second:
'https://www.cms.ucu.edu.ua/my/index.php'
Items in the second set but not the first:
'https://www.instagram.com/'

=====
Ran 3 tests in 0.002s

```

Зробив помилку в цьому докесті:

```

history_analysis.py  history_analysis_test.py x  line.py
history_analysis_test.py > TestHistoryAnalysis > setUp
4 class TestHistoryAnalysis(unittest.TestCase):
5
6     def setUp(self):
7         self.visits = [
8             ('https://www.youtube.com/', 'YouTube', '2023-10-23', '11:55:31.705320', 0),
9             ('https://www.youtube.com/', 'YouTube', '2023-10-22', '11:55:31.705320', 1083473),
10            ('https://www.instagram.com/', 'Instagram', '2023-10-23', '11:55:39.190008', 467439),
11            ('https://www.cms.ucu.edu.ua/my/index.php', 'Dashboard', '2023-10-23', '11:56:03.912323', 0),
12            ('https://www.instagram.com/', 'Instagram', '2023-10-23', '11:55:39.655687', 414254256),
13            ('https://www.youtube.com/', 'YouTube', '2023-10-22', '11:55:34.348591', 1006174780)
14        ]
15
16     def test_sites_on_date(self):
17         self.assertEqual(sites_on_date(self.visits, '2023-10-22'), {'https://www.youtube.com/'})
18         self.assertEqual(sites_on_date(self.visits, '2023-10-23'), {'https://www.youtube.com/', 'https://www.instagram.com/', 'https://www.cms.ucu.edu.ua/my/index.php'})
19
20     def test_most_frequent_sites(self):
21         self.assertEqual(most_frequent_sites(self.visits, 10), {'https://www.instagram.com/', 'https://www.youtube.com/', 'https://www.cms.ucu.edu.ua/my/index.php'})
22         self.assertEqual(most_frequent_sites(self.visits, 2), {'https://www.instagram.com/', 'https://www.youtube.com/'})
23
24     def test_get_url_info(self):
25         self.assertEqual(get_url_info(self.visits, 'https://www.youtube.com/'), ('YouTube', '2023-10-23', '11:55:31.705320', 2, 542369736.5))
26         self.assertEqual(get_url_info(self.visits, 'https://www.instagram.com/'), ('Instagram', '2023-10-23', '11:55:39.655687', 2, 207827347.5))

```

Адже ютуб траплявся три рази, та й середня довжина мала б бути меншою

Line;

The screenshot shows an IDE with two tabs: `test_line.py` and `test_line_git.py`. The active tab is `test_line_git.py`. A task description box is overlaid on the editor, containing the following text:

Напиши unittests для цієї умови:

Requested files: `line.py`, `test_line.py`.

Розробіть множину unittest тестів для методу `intersect`, який призначений для пошуку точки перетину двох прямих і реалізований в класі `Line`. Аргументами методу є дві прямі. Прямі в класі `Line` представляються парою точок (екземпляри класу `Point`), через які вони проходять, наприклад `Line(Point(0.0, 0.0), Point(1.0, 3.0))`.

Набір тестів повинен містити мінімум шість тестових випадків та враховувати наступні особливості:

- а. Якщо прямі не перетинаються, то метод `intersect` повертає `None`.
- б. Якщо прямі перетинаються, то метод `intersect` повертає точку (екземпляр класу `Point`) перетину, наприклад `Point(0.5, 0.75)`.
- в. Якщо прямі збігаються, то метод повертає перший аргумент.

Тести потрібно оформити в модулі `test_line.py` в класі `TestLine`, а клас `Line` та метод `intersect` в модулі `line.py`. При розробці тестів зверніть увагу (перечитайте `help`) по використанню методів `setUp` та `tearDown` класу `TestCase`.

At the bottom of the task box, there is a small icon and the text: "You can also select code to make an edit".

Below the task box, in the IDE's status bar area, the number "1" is displayed.

```

1 import unittest
2 from line import Line, Point
3
4 class TestLine(unittest.TestCase):
5     def setUp(self):
6         # Create two lines for testing
7         self.line1 = Line(Point(0.0, 0.0), Point(1.0, 3.0))
8         self.line2 = Line(Point(2.0, 2.0), Point(4.0, 6.0))
9
10    def tearDown(self):
11        # Clean up any resources used for testing
12        pass
13
14    def test_no_intersection(self):
15        # Test case for lines that do not intersect
16        self.assertIsNone(self.line1.intersect(self.line2))
17
18    def test_intersection(self):
19        # Test case for lines that intersect
20        expected_point = Point(0.5, 0.75)
21        self.assertEqual(self.line1.intersect(self.line1), expected_point)
22
23    def test_coincident_lines(self):
24        # Test case for coincident lines
25        self.assertEqual(self.line1.intersect(self.line1), self.line1.start)
26
27    # Add more test cases as needed
28
29 if __name__ == '__main__':
30     unittest.main()
31

```

Вирішив, що у нашого коду повинен бути атрибут старт

test_line_git.py × line.py

line > test_line_git.py > ...

Our code doesn't have attribute start

🔊 ▶ × ...

⚠ Copilot generated code may be incorrect

```
1 import unittest
2 from line import Line, Point
3
4 class TestLine(unittest.TestCase):
5     def setUp(self):
6         # Create two lines for testing
7         self.line1 = Line(Point(0.0, 0.0), Point(1.0, 3.0))
8         self.line2 = Line(Point(2.0, 2.0), Point(4.0, 6.0))
9
10    def tearDown(self):
11        # Clean up any resources used for testing
12        pass
13
14    def test_no_intersection(self):
15        # Test case for lines that do not intersect
16        self.assertIsNone(self.line1.intersect(self.line2))
17
18    def test_intersection(self):
19        # Test case for lines that intersect
20        expected_point = Point(0.5, 0.75)
21        self.assertEqual(self.line1.intersect(self.line1), expected_point)
22
```

```
test_line_git.py • line.py
line > test_line_git.py > ...
4 class TestLine(unittest.TestCase):
5     ... def setUp(self):
6         ... # Create two lines for testing
7         ... self.line1 = Line(Point(0.0, 0.0), Point(1.0, 3.0))
8         ... self.line2 = Line(Point(2.0, 2.0), Point(4.0, 6.0))
9
10    ... def tearDown(self):
11        ... # Clean up any resources used for testing
12        ... pass
13
14    ... def test_no_intersection(self):
15        ... # Test case for lines that do not intersect
16        ... self.assertIsNotNone(self.line1.intersect(self.line2))
17
18    ... def test_intersection(self):
19        ... # Test case for lines that intersect
20        ... expected_point = Point(0.5, 0.75)
21        ... self.assertEqual(self.line1.intersect(self.line1), expected_point)
22
23    ... def test_coincident_lines(self):
24        ... # Test case for coincident lines
25        ... self.assertEqual(self.line1.intersect(self.line1), self.line1.start)
26
27    ... # Add more test cases as needed
28
29    if __name__ == '__main__':
```

Нічого не змінив

test_line_git.py • line.py

line > test_line_git.py > ...

```
4 class TestLine(unittest.TestCase):
5     ... def setUp(self):
6         ... # Create two lines for testing
7         ... self.line1 = Line(Point(0.0, 0.0), Point(1.0, 3.0))
8         ... self.line2 = Line(Point(2.0, 2.0), Point(4.0, 6.0))
9
10    ... def tearDown(self):
11        ... # Clean up any resources used for testing
12        ... pass
13
14    ... def test_no_intersection(self):
15        ... # Test case for lines that do not intersect
16        ... self.assertIsNotNone(self.line1.intersect(self.line2))
17
18    ... def test_intersection(self):
19        ... # Test case for lines that intersect
20        ... expected_point = Point(0.5, 0.75)
21        ... self.assertEqual(self.line1.intersect(self.line1), expected_point)
22
23    ... def test_coincident_lines(self):
24        ... # Test case for coincident lines
25        ... self.assertEqual(self.line1.intersect(self.line1), self.line1.start)
26
27    ... # Add more test cases as needed
28
29 if __name__ == '__main__':
```