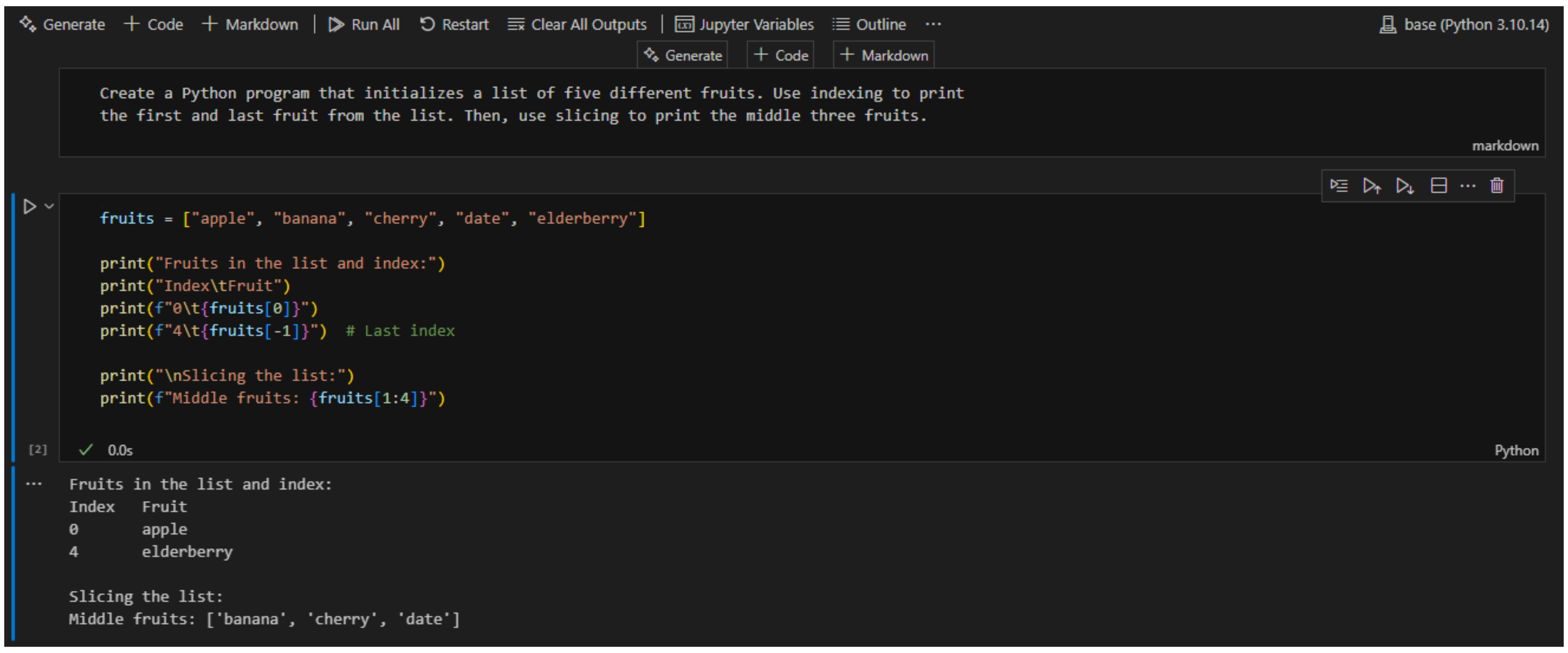
W2_OpSrc_Avni_23103028

▨ Week Number:	2

Ques 1:

```
fruits = ["apple", "banana", "cherry", "date", "elderberry"]

print("Fruits in the list and index:")
print("Index\tFruit")
print(f"0\t{fruits[0]}")
print(f"4\t{fruits[-1]}")  # Last index

print("\nSlicing the list:")
print(f"Middle fruits: {fruits[1:4]}")
```



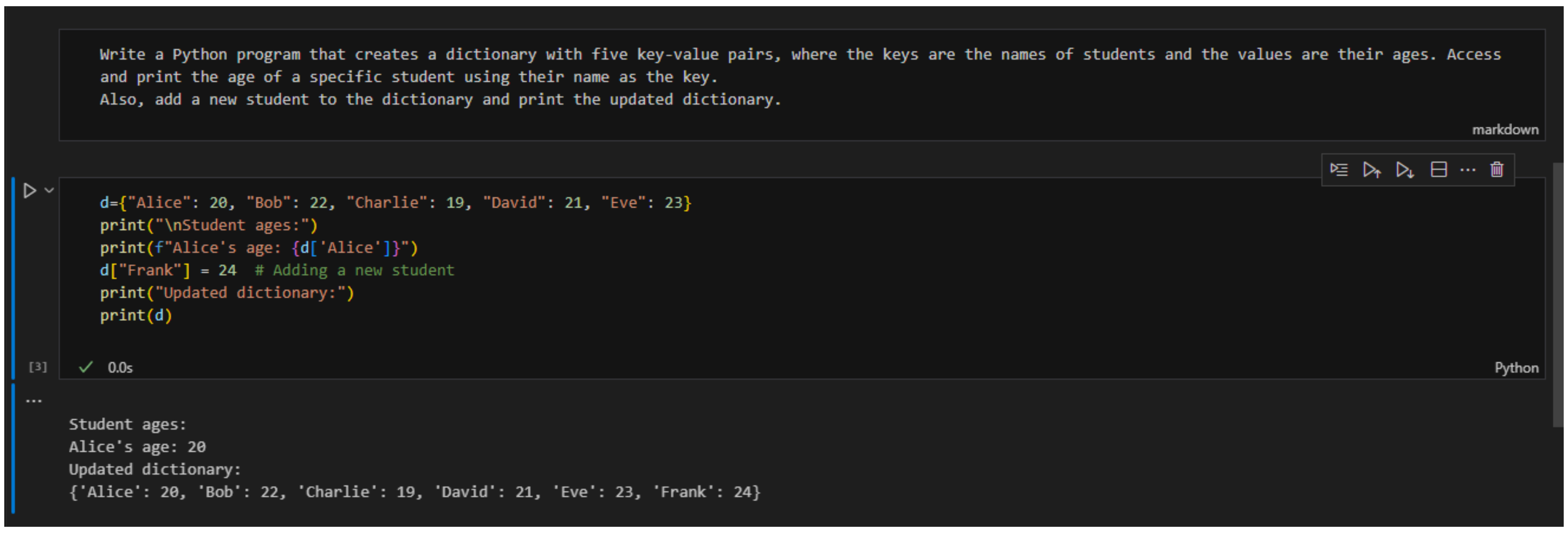
Create a Python program that initializes a list of five different fruits. Use indexing to print the first and last fruit from the list. Then, use slicing to print the middle three fruits.

```
fruits = ["apple", "banana", "cherry", "date", "elderberry"]

print("Fruits in the list and index:")
print("Index\tFruit")
print(f"0\t{fruits[0]}")
print(f"4\t{fruits[-1]}")  # Last index

print("\nSlicing the list:")
print(f"Middle fruits: {fruits[1:4]}")
```

```
Fruits in the list and index:
Index   Fruit
0       apple
4       elderberry

Slicing the list:
Middle fruits: ['banana', 'cherry', 'date']
```

Ques 2:

```
d={"Alice": 20, "Bob": 22, "Charlie": 19, "David": 21, "Eve": 23}
print("\nStudent ages:")
print(f"Alice's age: {d['Alice']}")
d["Frank"] = 24  # Adding a new student
print("Updated dictionary:")
print(d)
```



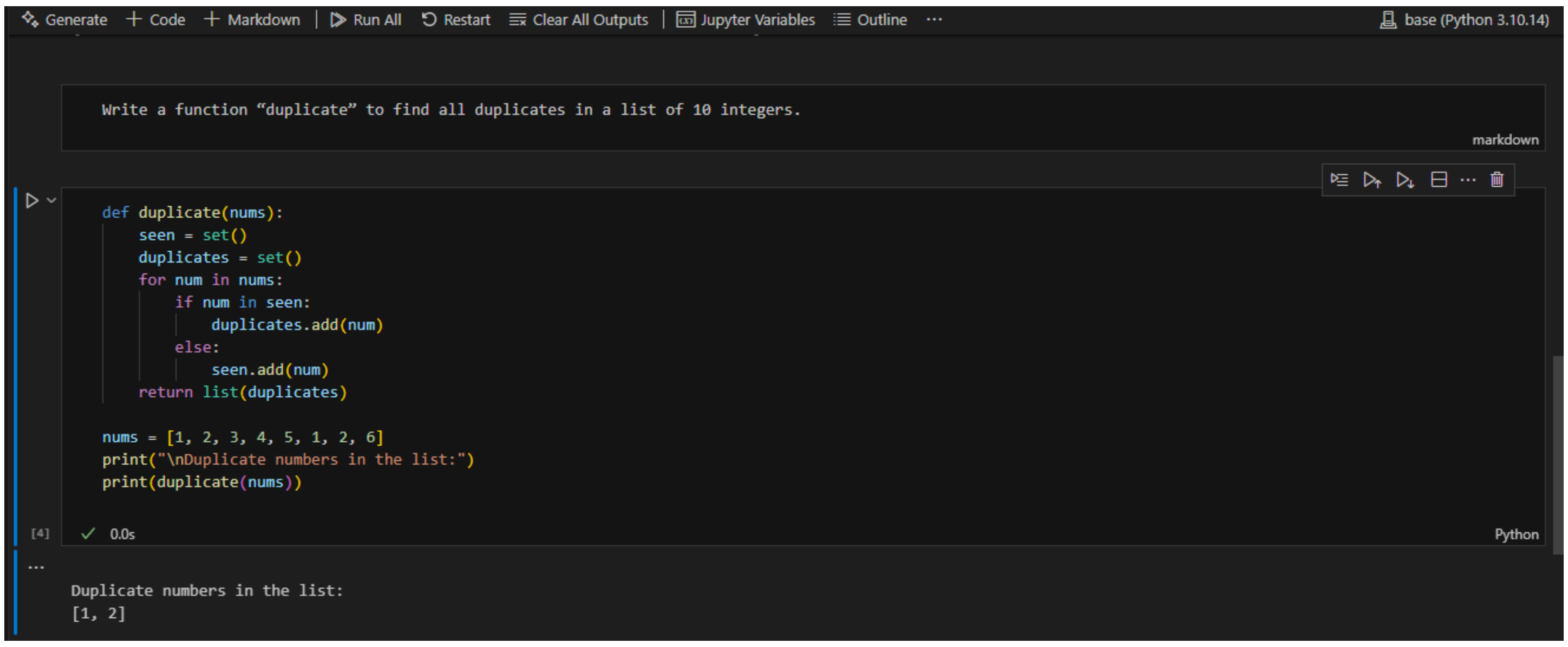
Write a Python program that creates a dictionary with five key-value pairs, where the keys are the names of students and the values are their ages. Access and print the age of a specific student using their name as the key.
Also, add a new student to the dictionary and print the updated dictionary.

```
d={"Alice": 20, "Bob": 22, "Charlie": 19, "David": 21, "Eve": 23}
print("\nStudent ages:")
print(f"Alice's age: {d['Alice']}")
d["Frank"] = 24  # Adding a new student
print("Updated dictionary:")
print(d)
```

```

Student ages:
Alice's age: 20
Updated dictionary:
{'Alice': 20, 'Bob': 22, 'Charlie': 19, 'David': 21, 'Eve': 23, 'Frank': 24}
```

Ques 3:

```
def duplicate(nums):
    seen = set()
    duplicates = set()
    for num in nums:
        if num in seen:
            duplicates.add(num)
        else:
            seen.add(num)
    return list(duplicates)

nums = [1, 2, 3, 4, 5, 1, 2, 6]
print("\nDuplicate numbers in the list:")
print(duplicate(nums))
```



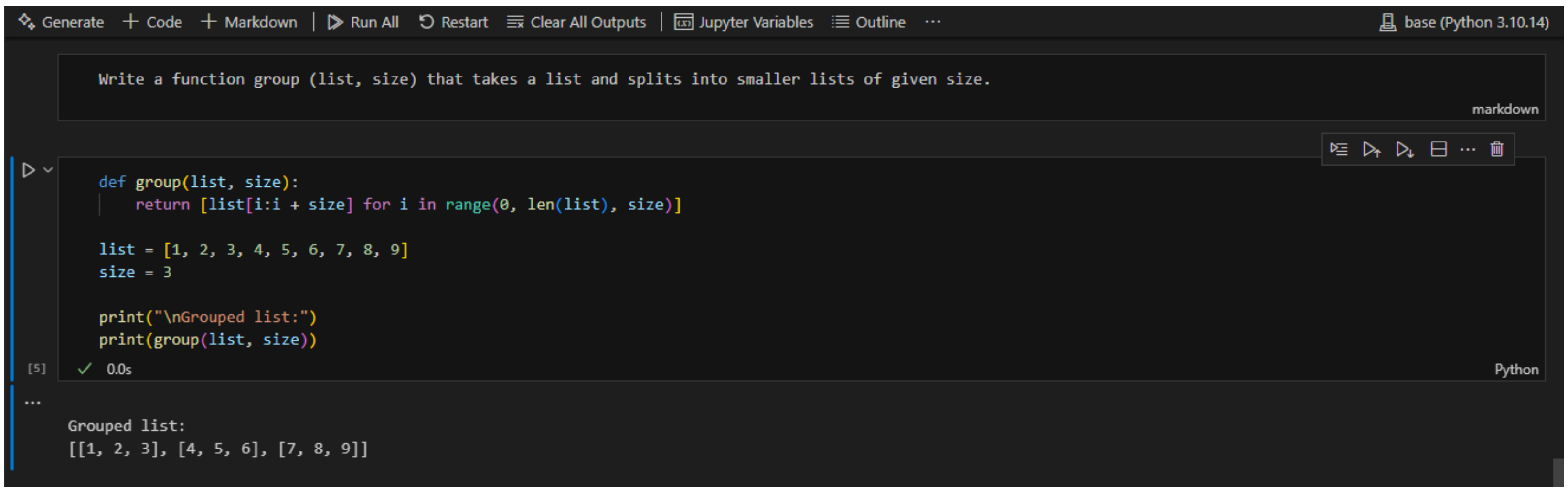
Write a function "duplicate" to find all duplicates in a list of 10 integers.

```
def duplicate(nums):
    seen = set()
    duplicates = set()
    for num in nums:
        if num in seen:
            duplicates.add(num)
        else:
            seen.add(num)
    return list(duplicates)

nums = [1, 2, 3, 4, 5, 1, 2, 6]
print("\nDuplicate numbers in the list:")
print(duplicate(nums))
```

```
Duplicate numbers in the list:
[1, 2]
```

Ques 4:

```
def group(list, size):
    return [list[i:i + size] for i in range(0, len(list), size)]

list = [1, 2, 3, 4, 5, 6, 7, 8, 9]
size = 3

print("\nGrouped list:")
print(group(list, size))
```



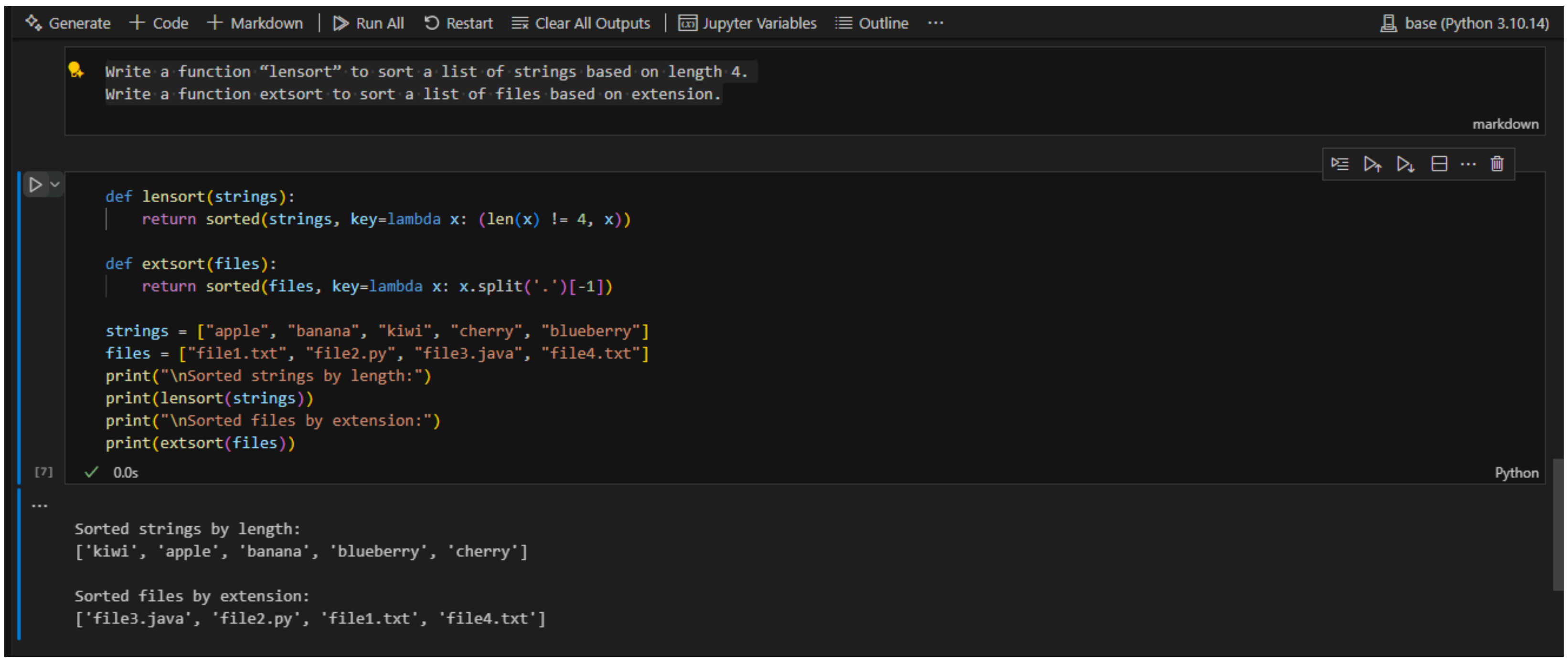
Write a function group (list, size) that takes a list and splits into smaller lists of given size.

```
def group(list, size):
    return [list[i:i + size] for i in range(0, len(list), size)]

list = [1, 2, 3, 4, 5, 6, 7, 8, 9]
size = 3

print("\nGrouped list:")
print(group(list, size))
```

```
Grouped list:
[[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

Ques 5:

```
def lensort(strings):
    return sorted(strings, key=lambda x: (len(x) != 4, x))

def extsort(files):
    return sorted(files, key=lambda x: x.split('.')[-1])

strings = ["apple", "banana", "kiwi", "cherry", "blueberry"]
files = ["file1.txt", "file2.py", "file3.java", "file4.txt"]
print("\nSorted strings by length:")
print(lensort(strings))
print("\nSorted files by extension:")
print(extsort(files))
```



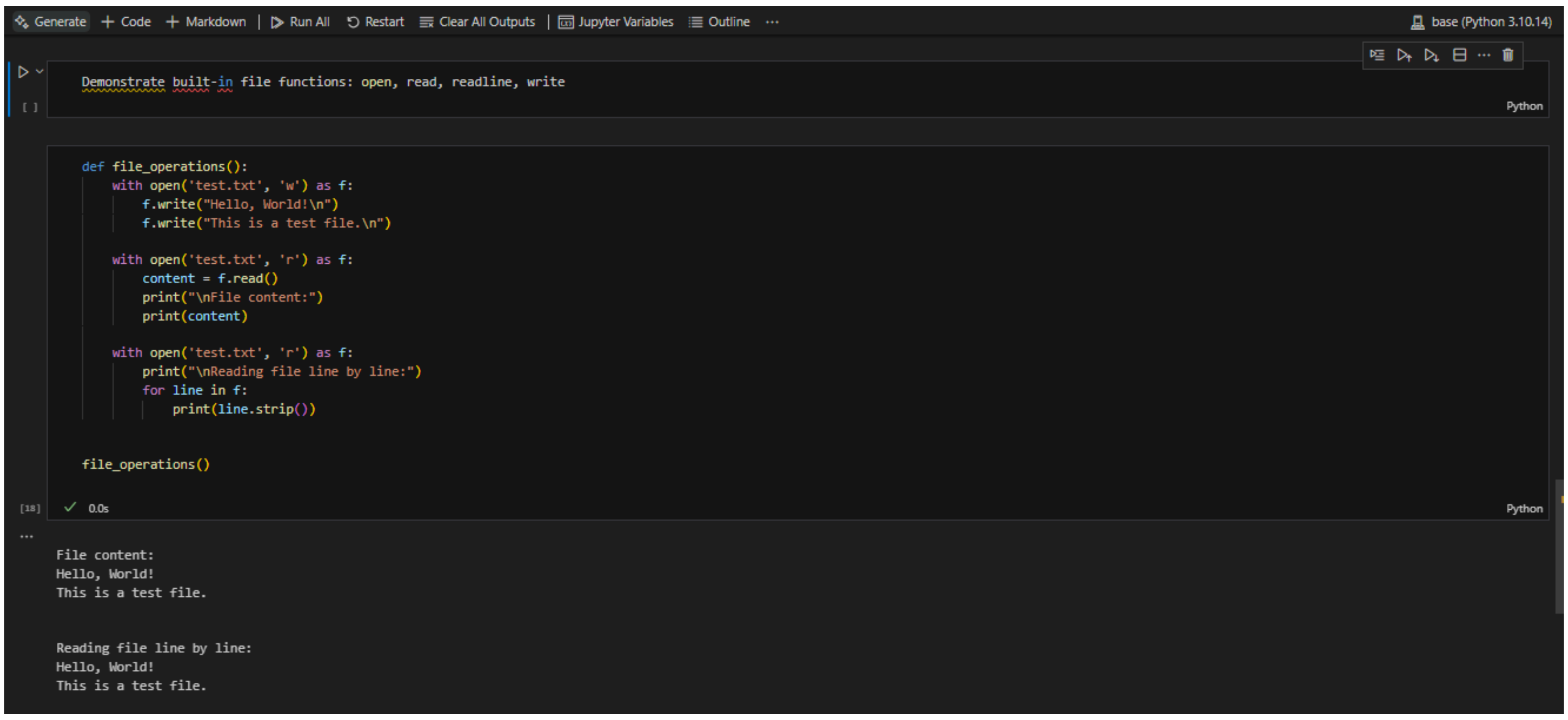
Write a function “lensort” to sort a list of strings based on length 4.
Write a function extsort to sort a list of files based on extension.

```
def lensort(strings):
    return sorted(strings, key=lambda x: (len(x) != 4, x))

def extsort(files):
    return sorted(files, key=lambda x: x.split('.')[-1])

strings = ["apple", "banana", "kiwi", "cherry", "blueberry"]
files = ["file1.txt", "file2.py", "file3.java", "file4.txt"]
print("\nSorted strings by length:")
print(lensort(strings))
print("\nSorted files by extension:")
print(extsort(files))
```

```
Sorted strings by length:
['kiwi', 'apple', 'banana', 'blueberry', 'cherry']

Sorted files by extension:
['file3.java', 'file2.py', 'file1.txt', 'file4.txt']
```

Ques 6:

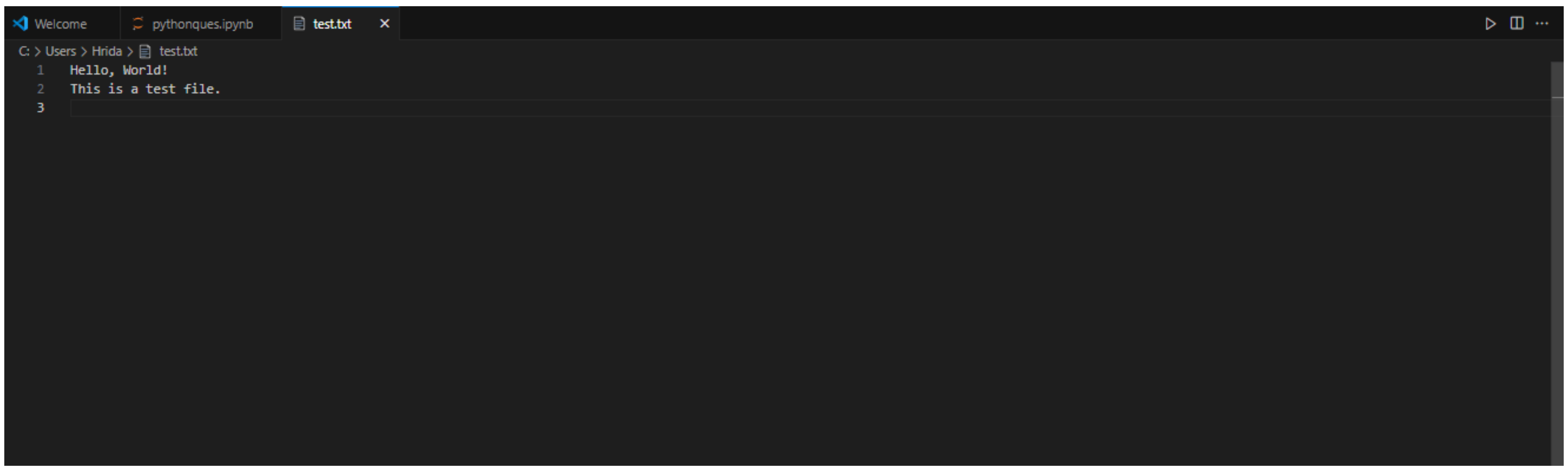
```
def file_operations():
    with open('test.txt', 'w') as f:
        f.write("Hello, World!\n")
        f.write("This is a test file.\n")

    with open('test.txt', 'r') as f:
        content = f.read()
        print("\nFile content:")
        print(content)

    with open('test.txt', 'r') as f:
        print("\nReading file line by line:")
        for line in f:
            print(line.strip())


file_operations()
```



Demonstrate built-in file functions: open, read, readline, write

```
def file_operations():
    with open('test.txt', 'w') as f:
        f.write("Hello, World!\n")
        f.write("This is a test file.\n")

    with open('test.txt', 'r') as f:
        content = f.read()
        print("\nFile content:")
        print(content)

    with open('test.txt', 'r') as f:
        print("\nReading file line by line:")
        for line in f:
            print(line.strip())


file_operations()
```

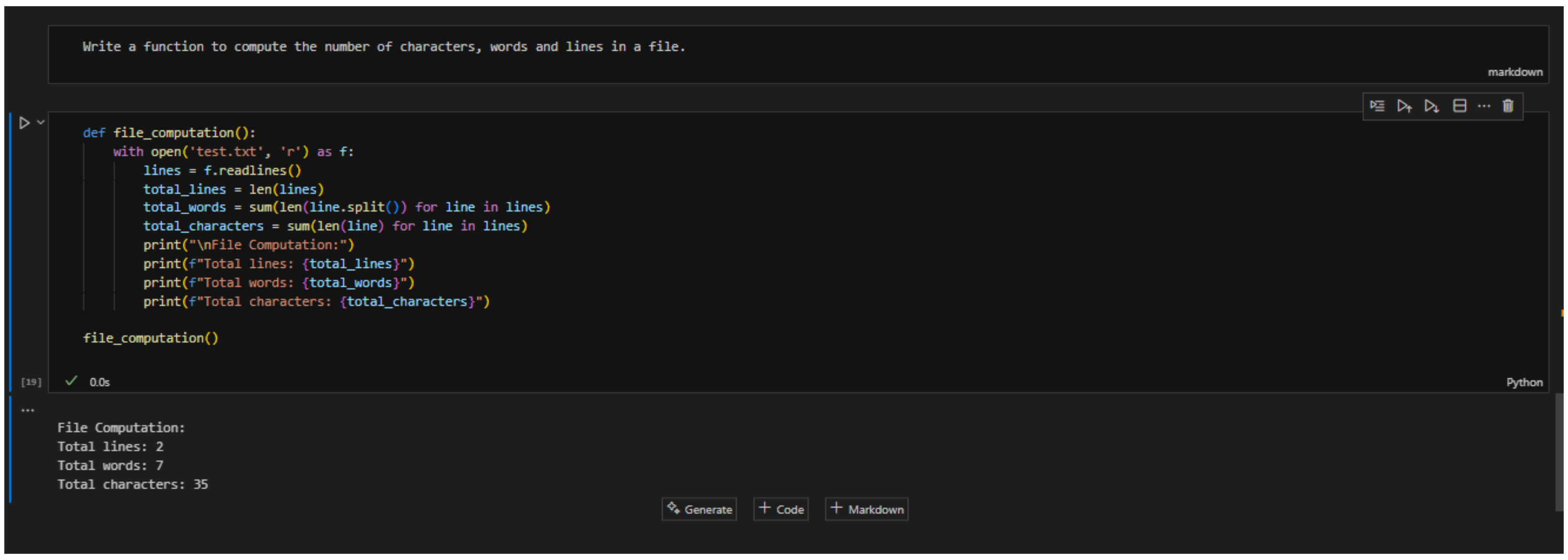
```
File content:
Hello, World!
This is a test file.


Reading file line by line:
Hello, World!
This is a test file.
```



```
Hello, World!
This is a test file.

```

Ques 7:

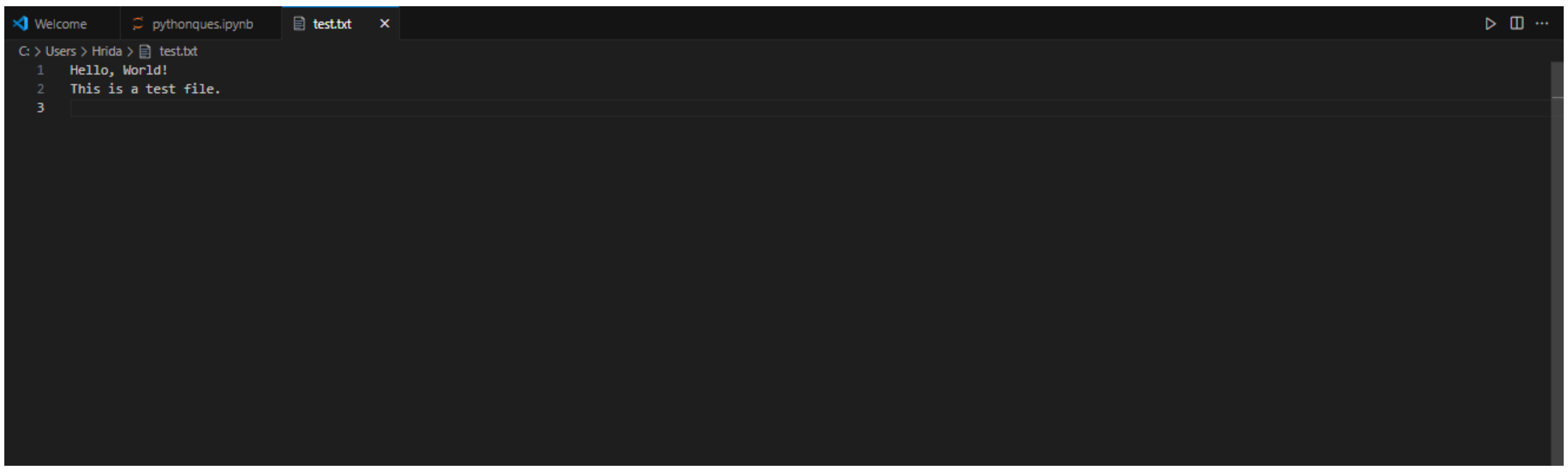
```
def file_computation():
    with open('test.txt', 'r') as f:
        lines = f.readlines()
        total_lines = len(lines)
        total_words = sum(len(line.split()) for line in lines)
        total_characters = sum(len(line) for line in lines)
        print("\nFile Computation:")
        print(f"Total lines: {total_lines}")
        print(f"Total words: {total_words}")
        print(f"Total characters: {total_characters}")

file_computation()
```



Write a function to compute the number of characters, words and lines in a file.

```
def file_computation():
    with open('test.txt', 'r') as f:
        lines = f.readlines()
        total_lines = len(lines)
        total_words = sum(len(line.split()) for line in lines)
        total_characters = sum(len(line) for line in lines)
        print("\nFile Computation:")
        print(f"Total lines: {total_lines}")
        print(f"Total words: {total_words}")
        print(f"Total characters: {total_characters}")

file_computation()
```

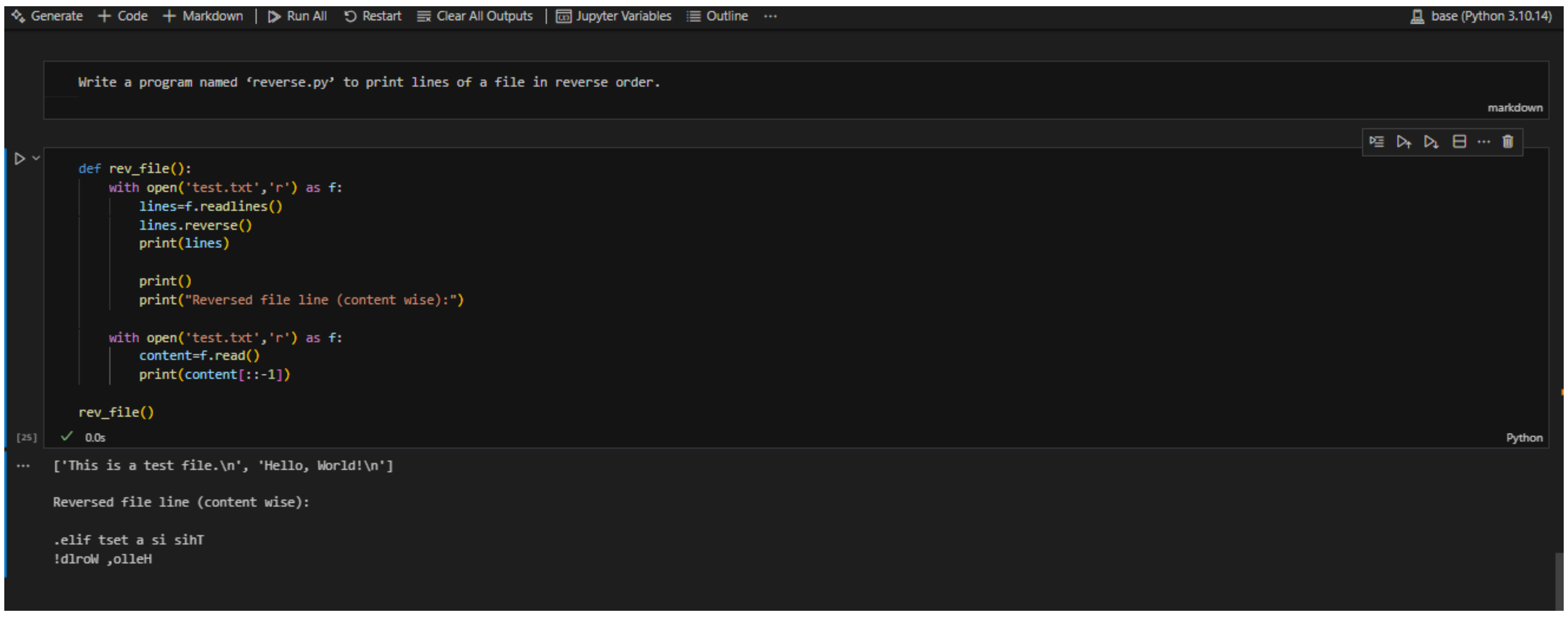
```
File Computation:
Total lines: 2
Total words: 7
Total characters: 35
```



```
Hello, World!
This is a test file.
```

Ques 8:

```
def rev_file():
    with open('test.txt','r') as f:
        lines=f.readlines()
        lines.reverse()
        print(lines)

        print()
        print("Reversed file line (content wise):")

    with open('test.txt','r') as f:
        content=f.read()
        print(content[::-1])

rev_file()
```



Write a program named ‘reverse.py’ to print lines of a file in reverse order.

```
def rev_file():
    with open('test.txt','r') as f:
        lines=f.readlines()
        lines.reverse()
        print(lines)

        print()
        print("Reversed file line (content wise):")

    with open('test.txt','r') as f:
        content=f.read()
        print(content[::-1])

rev_file()
```

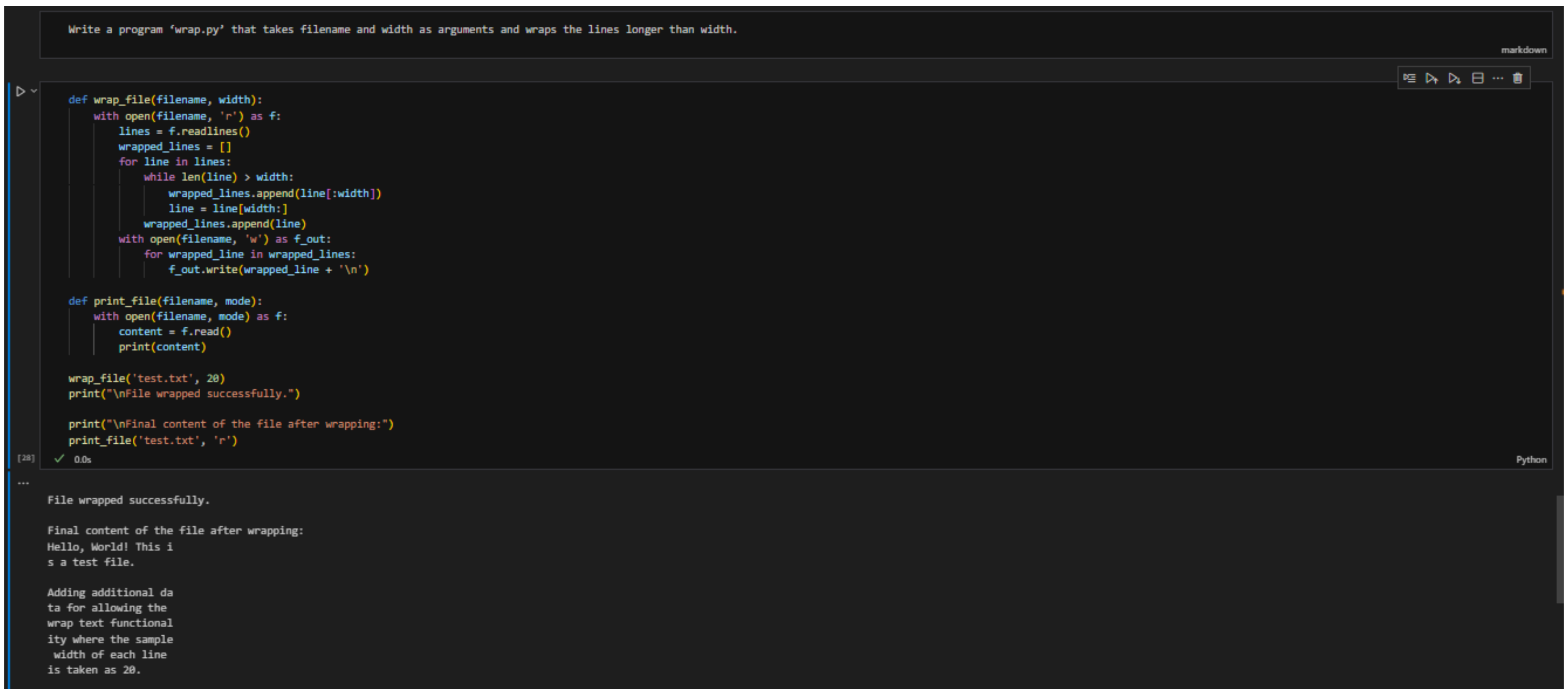
```
['This is a test file.\n', 'Hello, World!\n']

Reversed file line (content wise):

.elif tset a si sihT
!dlroW ,olleH
```

Ques 9:

DONE WITH QUES 8 ONLY.

Ques 10:

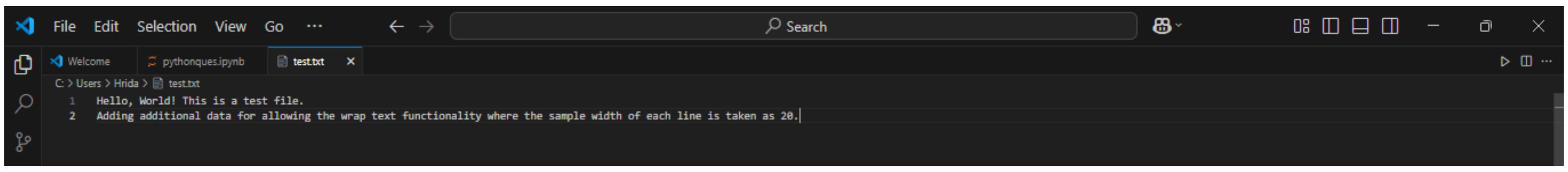
```
def wrap_file(filename, width):
    with open(filename, 'r') as f:
        lines = f.readlines()
        wrapped_lines = []
        for line in lines:
            while len(line) > width:
                wrapped_lines.append(line[:width])
                line = line[width:]
            wrapped_lines.append(line)
        with open(filename, 'w') as f_out:
            for wrapped_line in wrapped_lines:
                f_out.write(wrapped_line + '\n')

def print_file(filename, mode):
    with open(filename, mode) as f:
        content = f.read()
        print(content)

wrap_file('test.txt', 20)
print("\nFile wrapped successfully.")

print("\nFinal content of the file after wrapping:")
print_file('test.txt', 'r')
```



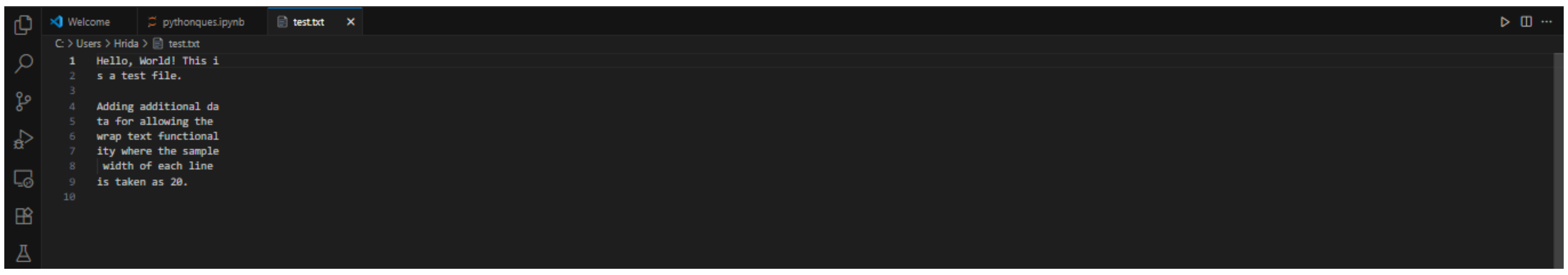
Write a program 'wrap.py' that takes filename and width as arguments and wraps the lines longer than width.

```
def wrap_file(filename, width):
    with open(filename, 'r') as f:
        lines = f.readlines()
        wrapped_lines = []
        for line in lines:
            while len(line) > width:
                wrapped_lines.append(line[:width])
                line = line[width:]
            wrapped_lines.append(line)
        with open(filename, 'w') as f_out:
            for wrapped_line in wrapped_lines:
                f_out.write(wrapped_line + '\n')

def print_file(filename, mode):
    with open(filename, mode) as f:
        content = f.read()
        print(content)

wrap_file('test.txt', 20)
print("\nFile wrapped successfully.")

print("\nFinal content of the file after wrapping:")
print_file('test.txt', 'r')
```

```
File wrapped successfully.

Final content of the file after wrapping:
Hello, World! This i
s a test file.

Adding additional da
ta for allowing the
wrap text functional
ity where the sample
 width of each line
is taken as 20.
```



```
Hello, World! This is a test file.
Adding additional data for allowing the wrap text functionality where the sample width of each line is taken as 20.
```

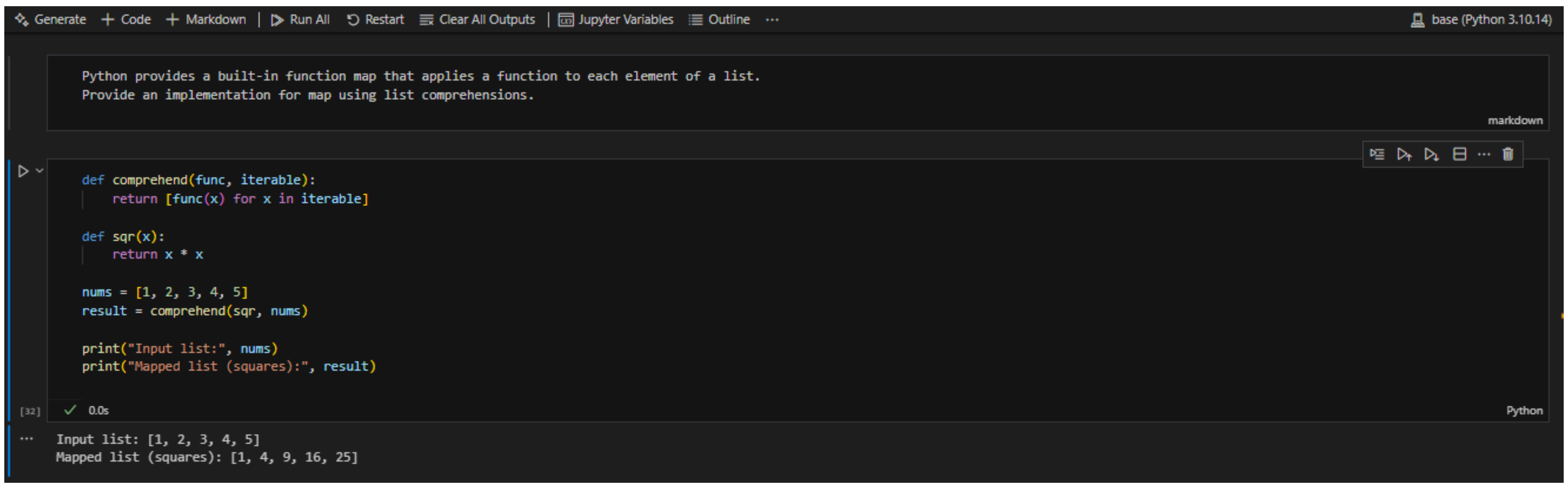


```
Hello, World! This i
s a test file.

Adding additional da
ta for allowing the
wrap text functional
ity where the sample
 width of each line
is taken as 20.

```

Ques 11:

```
def comprehend(func, iterable):
    return [func(x) for x in iterable]

def sqr(x):
    return x * x

nums = [1, 2, 3, 4, 5]
result = comprehend(sqr, nums)
```

```
print("Input list:", nums)
print("Mapped list (squares):", result)
```



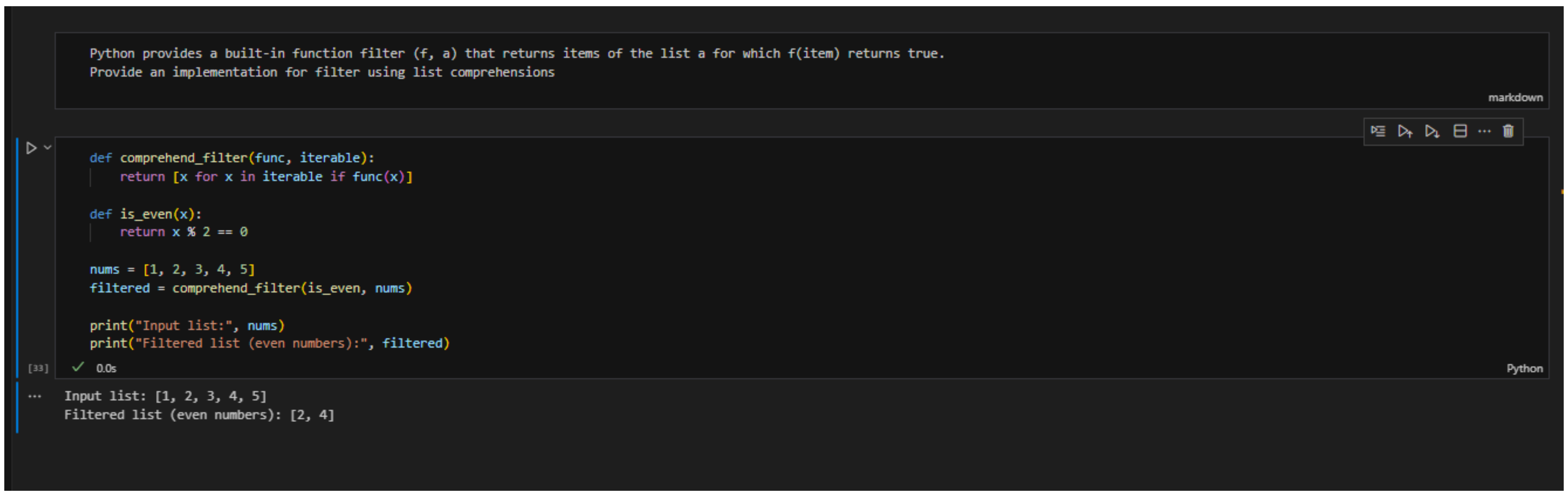
Python provides a built-in function map that applies a function to each element of a list.
Provide an implementation for map using list comprehensions.

```
def comprehend(func, iterable):
    return [func(x) for x in iterable]

def sqr(x):
    return x * x

nums = [1, 2, 3, 4, 5]
result = comprehend(sqr, nums)

print("Input list:", nums)
print("Mapped list (squares):", result)
```

```
Input list: [1, 2, 3, 4, 5]
Mapped list (squares): [1, 4, 9, 16, 25]
```

Ques 12:

```
def comprehend_filter(func, iterable):
    return [x for x in iterable if func(x)]

def is_even(x):
    return x % 2 == 0

nums = [1, 2, 3, 4, 5]
filtered = comprehend_filter(is_even, nums)

print("Input list:", nums)
print("Filtered list (even numbers):", filtered)
```



Python provides a built-in function filter (f, a) that returns items of the list a for which f(item) returns true.
Provide an implementation for filter using list comprehensions

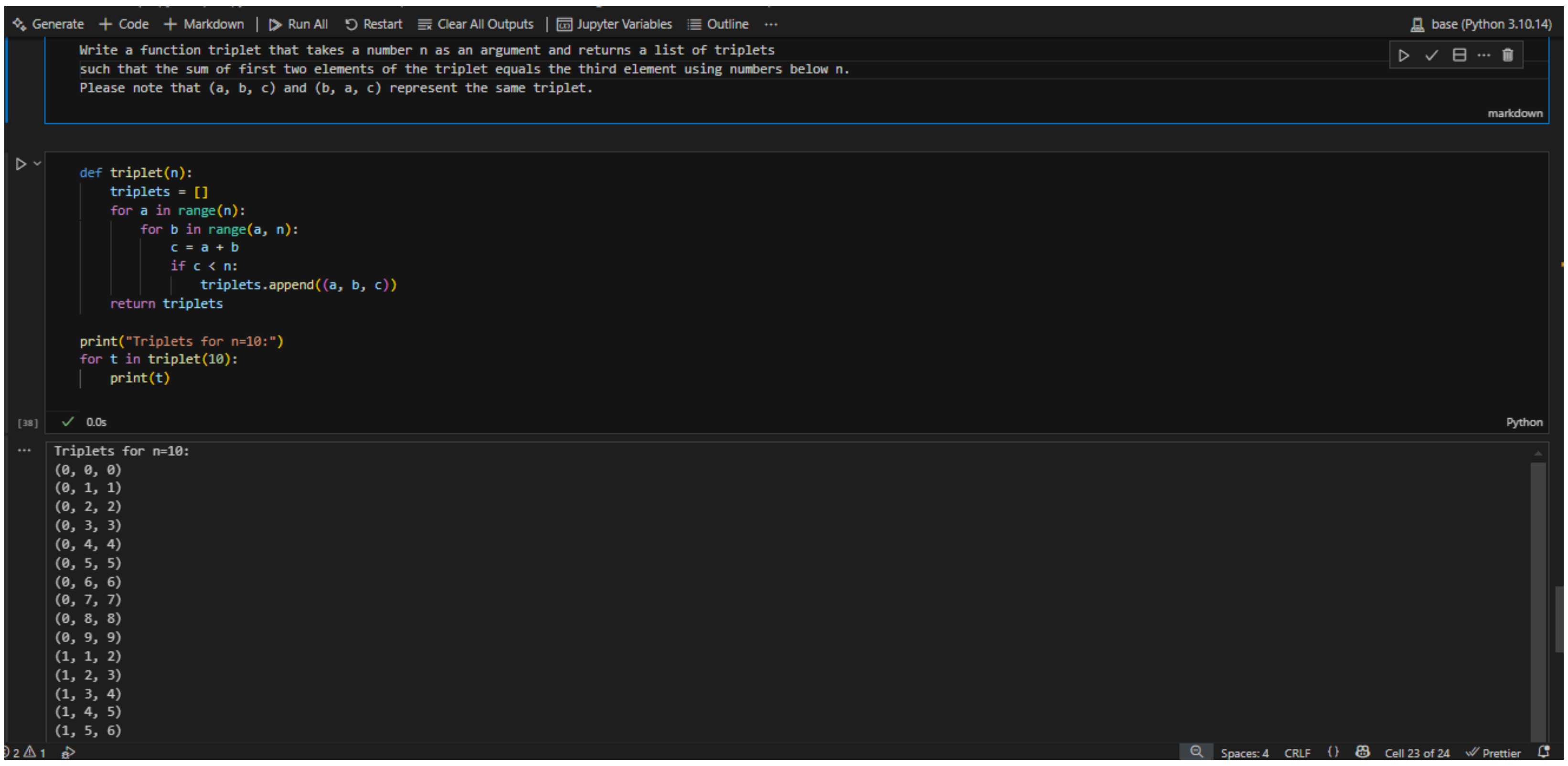
```
def comprehend_filter(func, iterable):
    return [x for x in iterable if func(x)]

def is_even(x):
    return x % 2 == 0

nums = [1, 2, 3, 4, 5]
filtered = comprehend_filter(is_even, nums)

print("Input list:", nums)
print("Filtered list (even numbers):", filtered)
```

```
Input list: [1, 2, 3, 4, 5]
Filtered list (even numbers): [2, 4]
```

Ques 13:

```
def triplet(n):
    triplets = []
    for a in range(n):
        for b in range(a, n):
            c = a + b
            if c < n:
                triplets.append((a, b, c))
    return triplets

print("Triplets for n=10:")
```

```
for t in triplet(10):
    print(t)
```

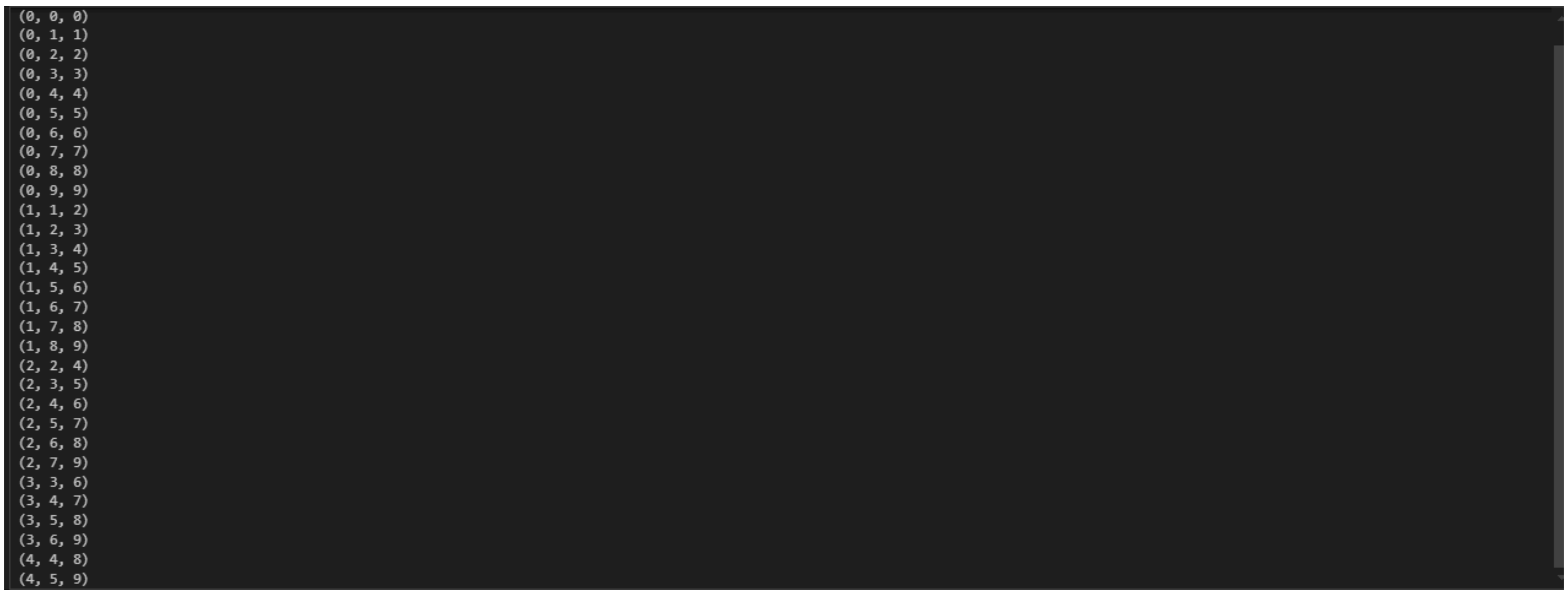


```
Write a function triplet that takes a number n as an argument and returns a list of triplets
such that the sum of first two elements of the triplet equals the third element using numbers below n.
Please note that (a, b, c) and (b, a, c) represent the same triplet.
```

```
def triplet(n):
    triplets = []
    for a in range(n):
        for b in range(a, n):
            c = a + b
            if c < n:
                triplets.append((a, b, c))
    return triplets

print("Triplets for n=10:")
for t in triplet(10):
    print(t)
```

[38] ✓ 0.0s

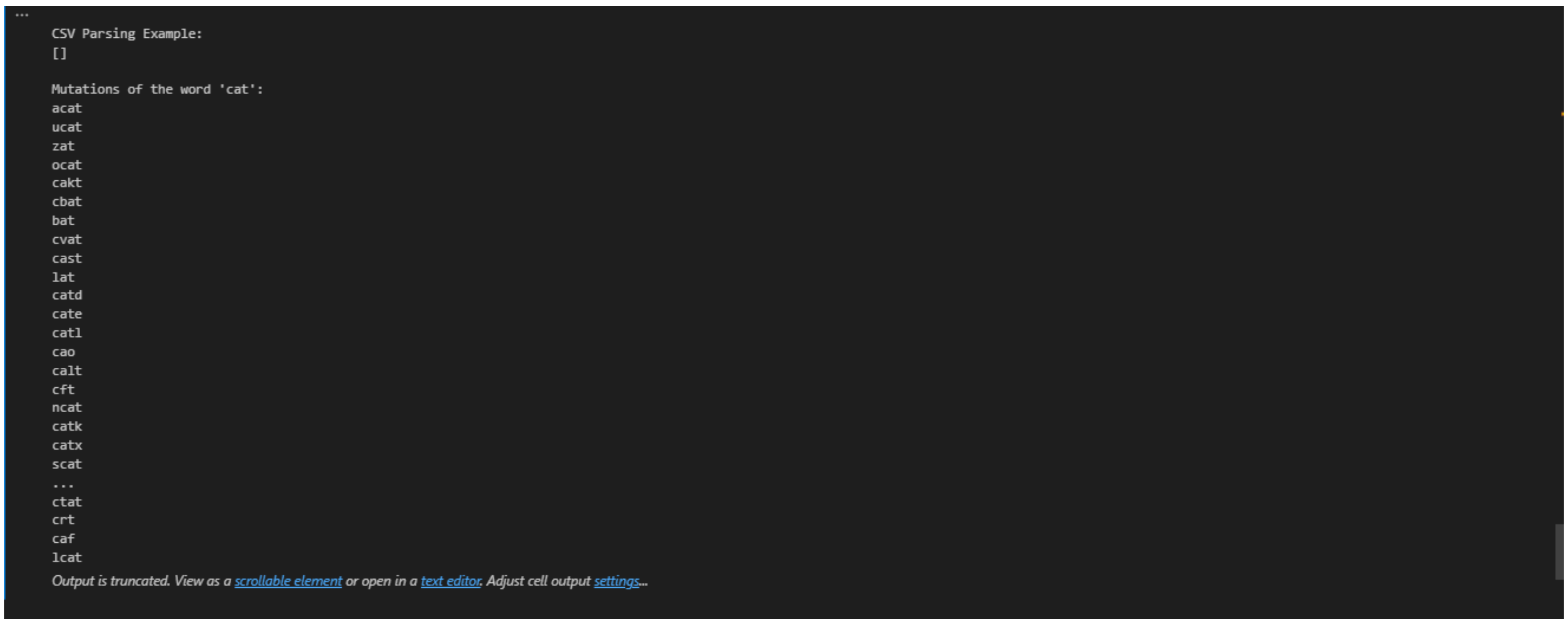
```
Triplets for n=10:
(0, 0, 0)
(0, 1, 1)
(0, 2, 2)
(0, 3, 3)
(0, 4, 4)
(0, 5, 5)
(0, 6, 6)
(0, 7, 7)
(0, 8, 8)
(0, 9, 9)
(1, 1, 2)
(1, 2, 3)
(1, 3, 4)
(1, 4, 5)
(1, 5, 6)
```



```
(0, 0, 0)
(0, 1, 1)
(0, 2, 2)
(0, 3, 3)
(0, 4, 4)
(0, 5, 5)
(0, 6, 6)
(0, 7, 7)
(0, 8, 8)
(0, 9, 9)
(1, 1, 2)
(1, 2, 3)
(1, 3, 4)
(1, 4, 5)
(1, 5, 6)
(1, 6, 7)
(1, 7, 8)
(1, 8, 9)
(2, 2, 4)
(2, 3, 5)
(2, 4, 6)
(2, 5, 7)
(2, 6, 8)
(2, 7, 9)
(3, 3, 6)
(3, 4, 7)
(3, 5, 8)
(3, 6, 9)
(4, 4, 8)
(4, 5, 9)
```

Ques 14:

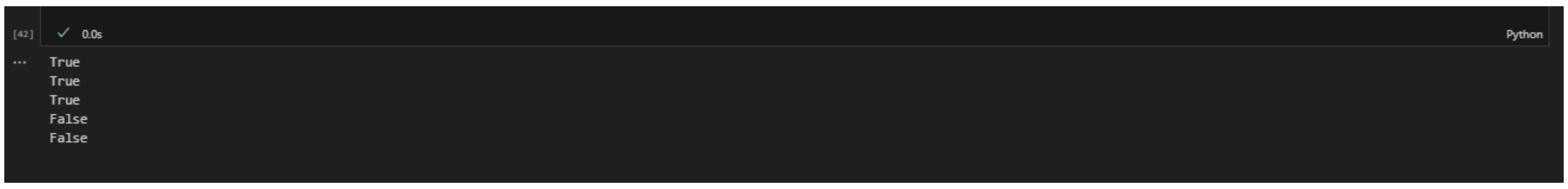
```
import csv

def parse_csv(filename):
    with open(filename, 'r') as f:
        lines = f.readlines()
        data = [line.strip().split(',') for line in lines]
    return data

def mutate(word):
    mutations = set()
    alphabet = 'abcdefghijklmnopqrstuvwxyz'

    for i in range(len(word) + 1):
        for char in alphabet:
            mutations.add(word[:i] + char + word[i:])

    for i in range(len(word)):
        mutations.add(word[:i] + word[i+1:])

    for i in range(len(word)):
        for char in alphabet:
            mutations.add(word[:i] + char + word[i+1:])

    for i in range(len(word) - 1):
        mutations.add(word[:i] + word[i+1] + word[i] + word[i+2:])

    return mutations


open("test.csv", "w").close()
print("\nCSV Parsing Example:")
csv_data = parse_csv('test.csv')
print(csv_data)
print("\nMutations of the word 'cat':")
mutated_words = mutate('cat')

for word in mutated_words:
    print(word)
```



```
...
    CSV Parsing Example:
    []

    Mutations of the word 'cat':
    acat
    ucat
    zat
    ocat
    cakt
    cbat
    bat
    cvat
    cast
    lat
    catd
    cate
    catl
    cao
    calt
    cft
    ncat
    catk
    catx
    scat
    ...
    ctat
    crt
    caf
    lcat
    Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
```

Ques 15:

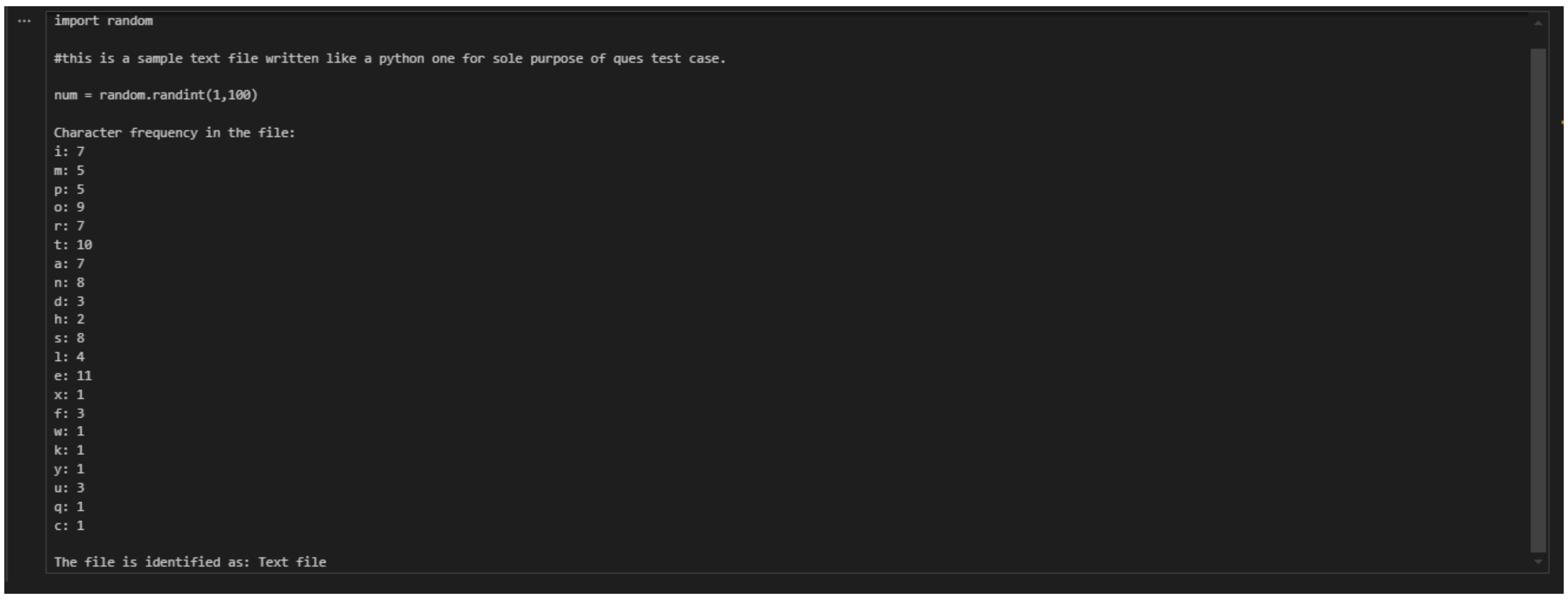
```
def nearly_equal(a, b):
    if a == b:
        return False

    La= len(a)
    Lb = len(b)

    if La==Lb:
        diff=sum(1 for x,y in zip(a,b) if x!=y)
        return diff==1

    if abs(La-Lb)==1:
        if La>Lb:
            a,b = b,a

        i=j=diff=0
        while i<len(a) and j<len(b):
            if a[i]!=b[j]:
                if diff:
                    return False
                diff+=1
                j+=1
            else:
                i+=1
                j+=1
        return True

    return False

print(nearly_equal("cat", "bat"))
print(nearly_equal("cat", "cats"))
print(nearly_equal("cats", "cat"))
print(nearly_equal("cat", "dog"))
print(nearly_equal("cat", "cat"))
```



```
True
True
True
False
False
```

Ques 16:

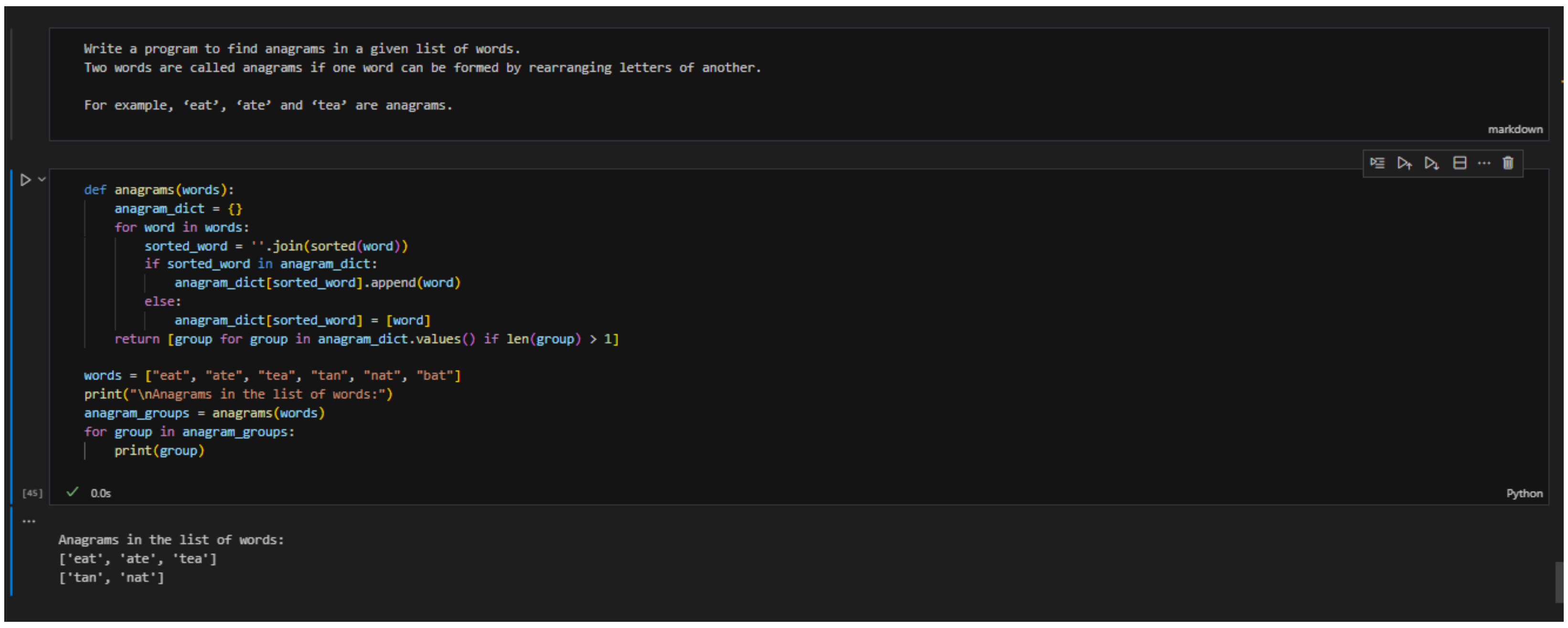
```
def char_frequency(filename):
    with open(filename, 'r') as f:
        content = f.read()
        freq = {}
        for char in content:
            if char.isalpha():
                freq[char] = freq.get(char,0)+1
    return freq

def identify_file_type(freq):
    if 'def' in freq and 'import' in freq:
        return "Python program file"
    elif 'int' in freq or 'float' in freq:
        return "C program file"
    else:
        return "Text file"

def file_print(filename, mode):
    with open(filename, mode) as f:
        content = f.read()
        print(content)

filename = 'test.txt'
print("File Content:")
file_print(filename, 'r')
freq = char_frequency(filename)
print("\nCharacter frequency in the file:")
for char, count in freq.items():
    print(f"{char}: {count}")
file_type = identify_file_type(freq)
print(f"\nThe file is identified as: {file_type}")
```



```
import random

#this is a sample text file written like a python one for sole purpose of ques test case.

num = random.randint(1,100)

Character frequency in the file:
i: 7
m: 5
p: 5
o: 9
r: 7
t: 10
a: 7
n: 8
d: 3
h: 2
s: 8
l: 4
e: 11
x: 1
f: 3
w: 1
k: 1
y: 1
u: 3
q: 1
c: 1

The file is identified as: Text file
```

Ques 17:

```
def anagrams(words):
    anagram_dict = {}
    for word in words:
        sorted_word = ''.join(sorted(word))
        if sorted_word in anagram_dict:
            anagram_dict[sorted_word].append(word)
        else:
            anagram_dict[sorted_word] = [word]
    return [group for group in anagram_dict.values() if len(group) > 1]

words = ["eat", "ate", "tea", "tan", "nat", "bat"]
print("\nAnagrams in the list of words:")
anagram_groups = anagrams(words)
for group in anagram_groups:
    print(group)
```



Write a program to find anagrams in a given list of words.
Two words are called anagrams if one word can be formed by rearranging letters of another.

For example, ‘eat’, ‘ate’ and ‘tea’ are anagrams.

```
def anagrams(words):
    anagram_dict = {}
    for word in words:
        sorted_word = ''.join(sorted(word))
        if sorted_word in anagram_dict:
            anagram_dict[sorted_word].append(word)
        else:
            anagram_dict[sorted_word] = [word]
    return [group for group in anagram_dict.values() if len(group) > 1]

words = ["eat", "ate", "tea", "tan", "nat", "bat"]
print("\nAnagrams in the list of words:")
anagram_groups = anagrams(words)
for group in anagram_groups:
    print(group)
```

```
Anagrams in the list of words:
['eat', 'ate', 'tea']
['tan', 'nat']
```