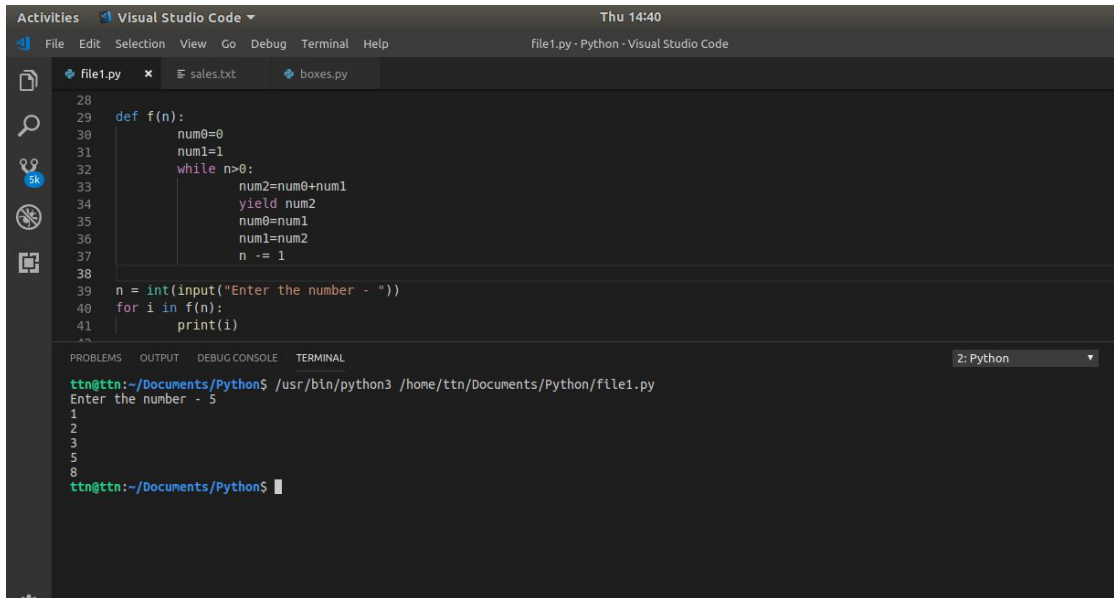


Exercise

1. Write a generator function to create fibonacci series up to n number. Get input n from the user.



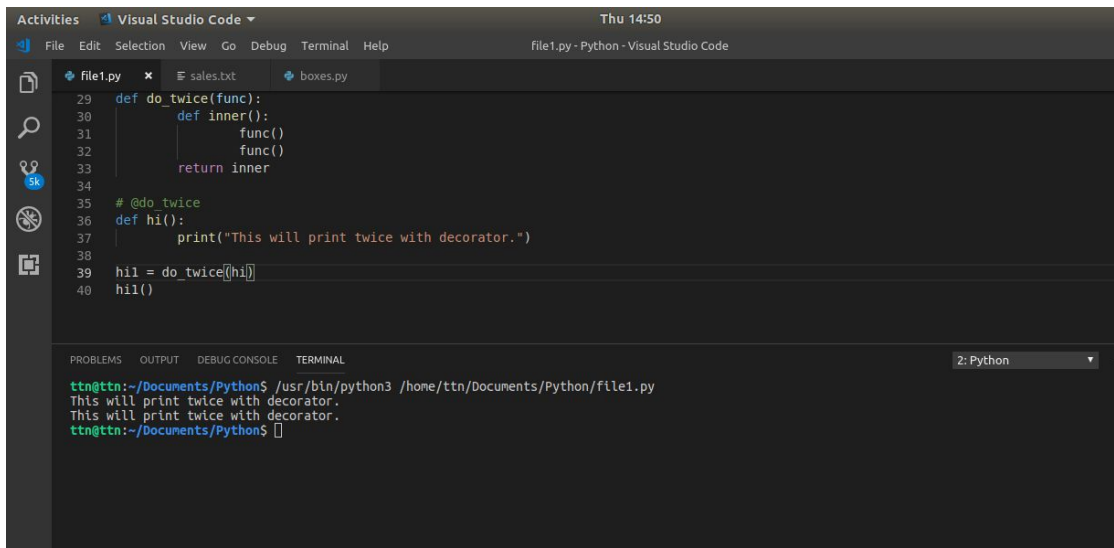
The screenshot shows the Visual Studio Code editor with a file named `file1.py`. The code defines a generator function `f(n)` that calculates the Fibonacci sequence up to the `n`th term. The function initializes `num0=0` and `num1=1`, then enters a `while` loop that yields the next number in the sequence and updates the previous two numbers. The user input is 5, and the output shows the first 5 terms of the Fibonacci sequence: 1, 1, 2, 3, 5.

```
28
29 def f(n):
30     num0=0
31     num1=1
32     while n>0:
33         num2=num0+num1
34         yield num2
35         num0=num1
36         num1=num2
37         n -= 1
38
39 n = int(input("Enter the number - "))
40 for i in f(n):
41     print(i)
```

Terminal output:

```
ttn@ttn:~/Documents/Python$ /usr/bin/python3 /home/ttn/Documents/Python/file1.py
Enter the number - 5
1
1
2
3
5
ttn@ttn:~/Documents/Python$
```

2. Create a decorator `do_twice` which provides any function capability to execute it twice.



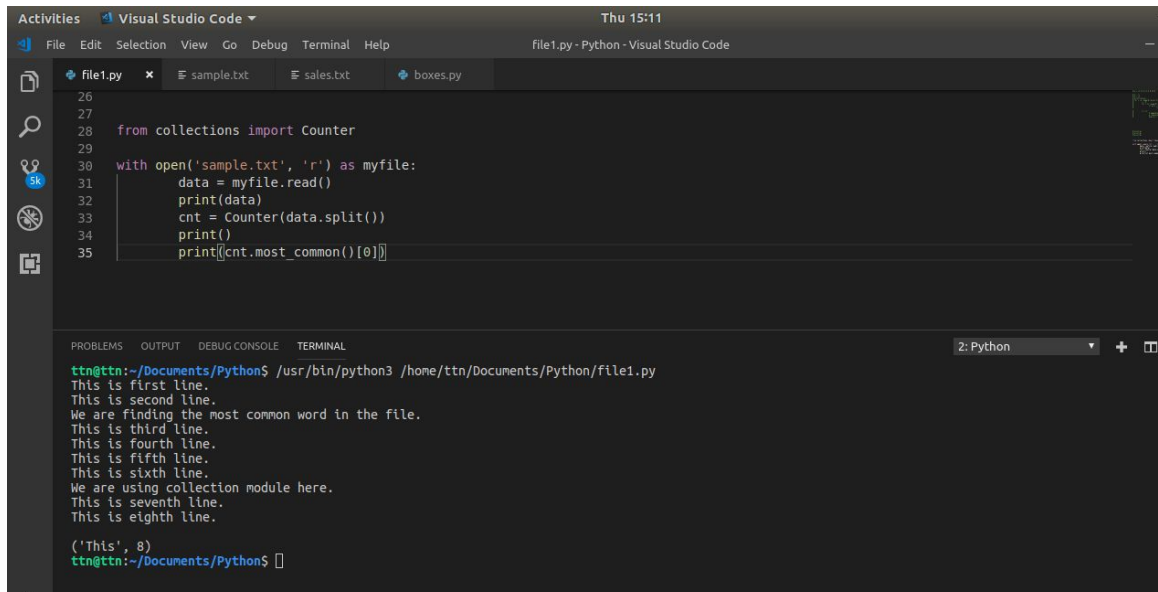
The screenshot shows the Visual Studio Code editor with a file named `file1.py`. The code defines a decorator function `do_twice` that takes a function `func` and returns an inner function that calls `func` twice. The `hi` function is decorated with `do_twice`, and the output shows the message "This will print twice with decorator." printed twice.

```
29 def do_twice(func):
30     def inner():
31         func()
32         func()
33     return inner
34
35 # @do_twice
36 def hi():
37     print("This will print twice with decorator.")
38
39 hi1 = do_twice(hi)
40 hi1()
```

Terminal output:

```
ttn@ttn:~/Documents/Python$ /usr/bin/python3 /home/ttn/Documents/Python/file1.py
This will print twice with decorator.
This will print twice with decorator.
ttn@ttn:~/Documents/Python$
```

3. Create a `sample.txt` file with 10-15 lines of text. Find out the most common word in the file



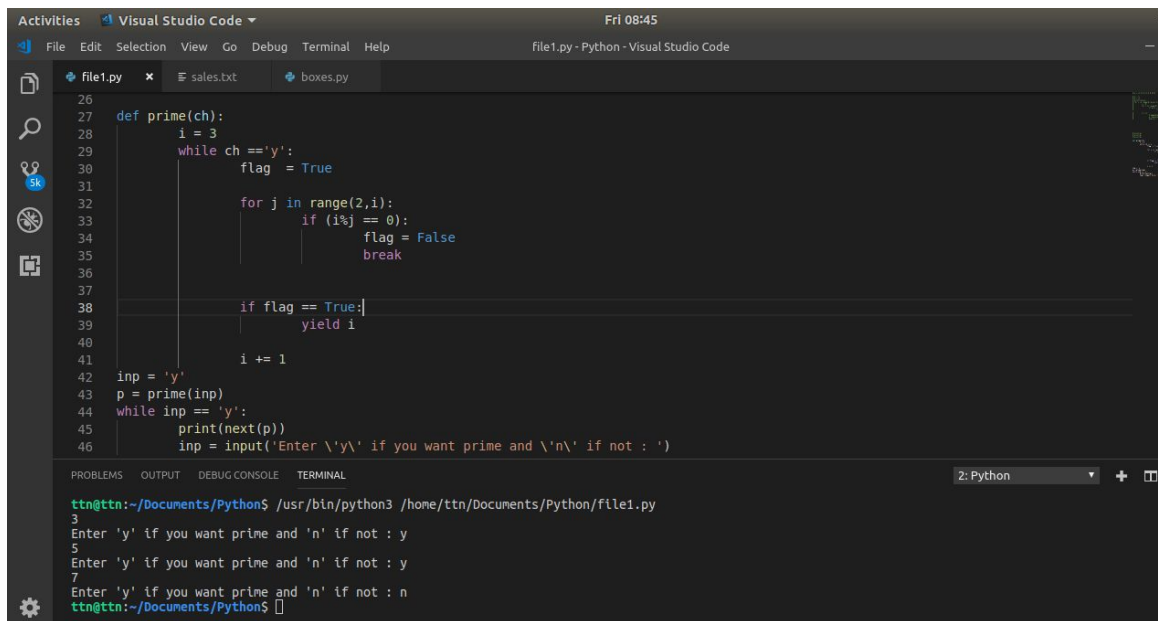
The screenshot shows the Visual Studio Code editor with a Python file named `file1.py`. The code reads a file `sample.txt` and uses the `collections.Counter` module to find the most common word. The terminal output shows the execution of the script, which prints the contents of the file and the most common word, 'This', which appears 8 times.

```
26
27
28 from collections import Counter
29
30 with open('sample.txt', 'r') as myfile:
31     data = myfile.read()
32     print(data)
33     cnt = Counter(data.split())
34     print()
35     print(cnt.most_common()[0])
```

```
ttn@ttn:~/Documents/Python$ /usr/bin/python3 /home/ttn/Documents/Python/file1.py
This is first line.
This is second line.
We are finding the most common word in the file.
This is third line.
This is fourth line.
This is fifth line.
This is sixth line.
We are using collection module here.
This is seventh line.
This is eighth line.

('This', 8)
ttn@ttn:~/Documents/Python$
```

4. Have the program find prime numbers until the user chooses to stop asking for the next one.

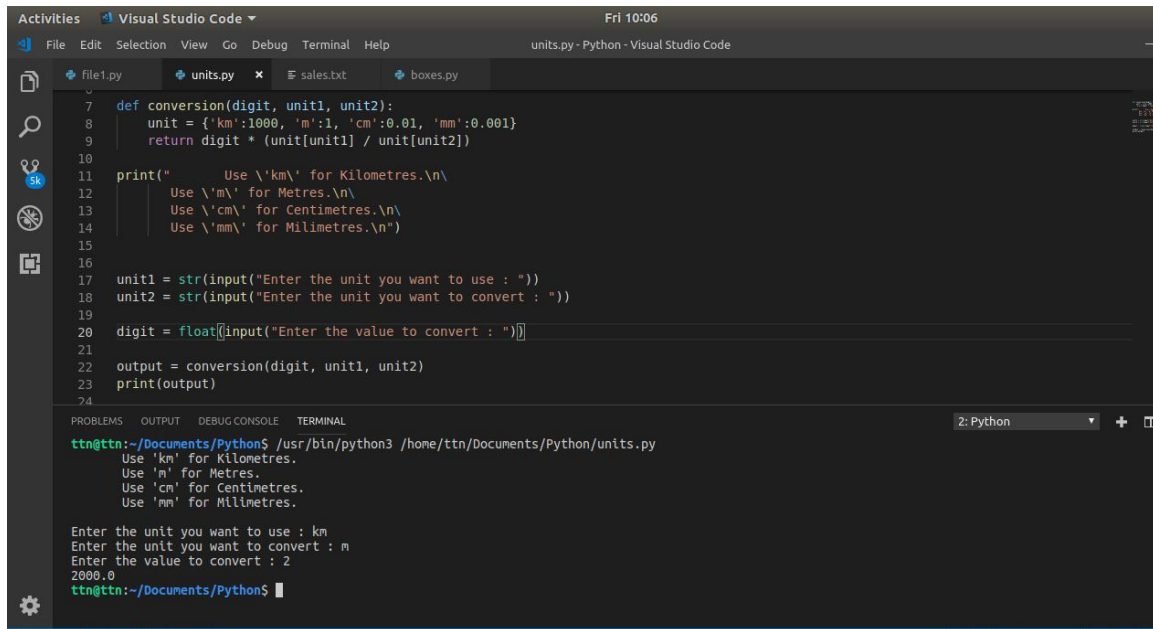


The screenshot shows the Visual Studio Code editor with a Python file named `file1.py`. The code defines a `prime` generator function and uses it to find prime numbers until the user chooses to stop. The terminal output shows the execution of the script, which prompts the user to enter 'y' for yes or 'n' for no, and prints the next prime number each time 'y' is entered.

```
26
27 def prime(ch):
28     i = 3
29     while ch == 'y':
30         flag = True
31
32         for j in range(2,i):
33             if (i%j) == 0:
34                 flag = False
35                 break
36
37         if flag == True:
38             yield i
39
40         i += 1
41
42 inp = 'y'
43 p = prime(inp)
44 while inp == 'y':
45     print(next(p))
46     inp = input('Enter \'y\' if you want prime and \'n\' if not : ')
```

```
ttn@ttn:~/Documents/Python$ /usr/bin/python3 /home/ttn/Documents/Python/file1.py
3
Enter 'y' if you want prime and 'n' if not : y
5
Enter 'y' if you want prime and 'n' if not : y
7
Enter 'y' if you want prime and 'n' if not : n
ttn@ttn:~/Documents/Python$
```

5. Converts various units between one another. The user enters the type of unit being entered, the type of unit they want to convert to and then the value. The program will then make the conversion



The screenshot shows the Visual Studio Code interface with a Python file named `units.py` open. The code defines a `conversion` function that takes a digit, a unit to convert from, and a unit to convert to. It uses a dictionary to map units to their conversion factors. The script prompts the user to enter the unit to use, the unit to convert to, and the value to convert. The terminal output shows the script being run, the prompts, and the resulting output.

```
def conversion(digit, unit1, unit2):
    unit = {'km':1000, 'm':1, 'cm':0.01, 'mm':0.001}
    return digit * (unit[unit1] / unit[unit2])

print("      Use \'km\' for Kilometres.\n\
      Use \'m\' for Metres.\n\
      Use \'cm\' for Centimetres.\n\
      Use \'mm\' for Millimetres.\n")

unit1 = str(input("Enter the unit you want to use : "))
unit2 = str(input("Enter the unit you want to convert : "))
digit = float(input("Enter the value to convert : "))

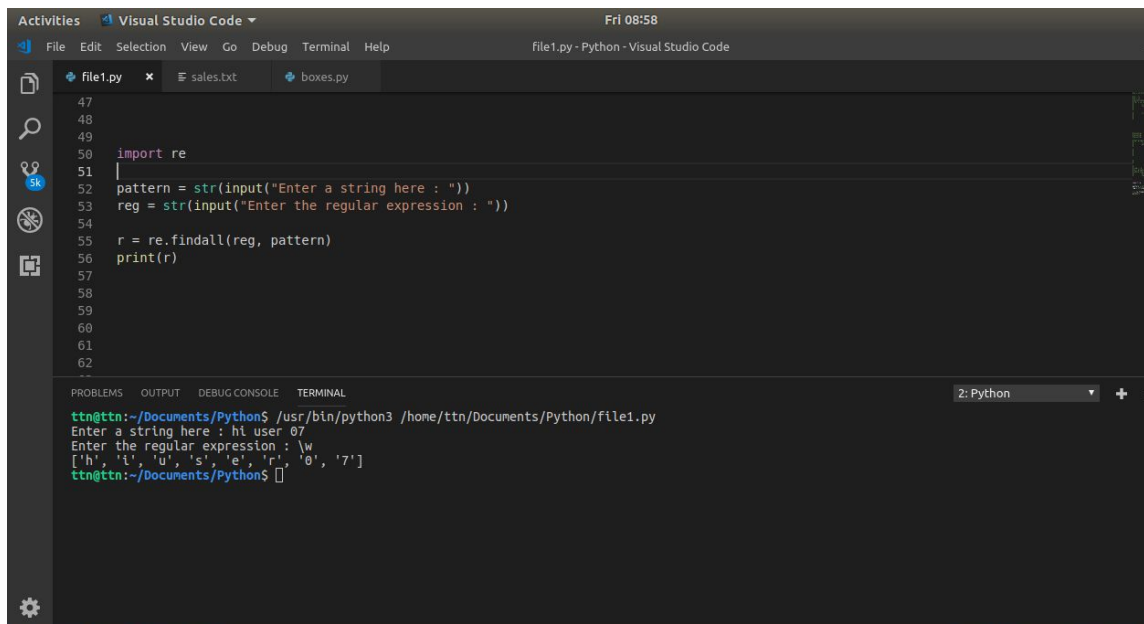
output = conversion(digit, unit1, unit2)
print(output)
```

Terminal Output:

```
ttn@ttn:~/Documents/Python$ /usr/bin/python3 /home/ttn/Documents/Python/units.py
      Use 'km' for Kilometres.
      Use 'm' for Metres.
      Use 'cm' for Centimetres.
      Use 'mm' for Millimetres.

Enter the unit you want to use : km
Enter the unit you want to convert : m
Enter the value to convert : 2
2000.0
ttn@ttn:~/Documents/Python$
```

6. A tool that allows the user to enter a text string and then in a separate control enter a regex pattern. It will run the regular expression against the source text and return any matches or flag errors in the regular expression.



The screenshot shows the Visual Studio Code interface with a Python file named `file1.py` open. The code imports the `re` module, prompts the user to enter a string and a regular expression, and then uses `re.findall` to find all matches. The terminal output shows the script being run, the prompts, and the resulting matches.

```
import re

pattern = str(input("Enter a string here : "))
reg = str(input("Enter the regular expression : "))

r = re.findall(reg, pattern)
print(r)
```

Terminal Output:

```
ttn@ttn:~/Documents/Python$ /usr/bin/python3 /home/ttn/Documents/Python/file1.py
Enter a string here : hi user 07
Enter the regular expression : \w
['h', 'i', 'u', 's', 'e', 'r', '0', '7']
ttn@ttn:~/Documents/Python$
```