

1. What is a lambda function in Python, and how does it differ from a regular function?

Ans:- A lambda function in Python is a small, anonymous, and inline function that can have any number of arguments, but can only have one expression. It is defined using the lambda keyword, followed by the arguments and the expression. Lambda functions are often used when we need a simple function for a short period of time .

Here are some key points about lambda functions in Python:

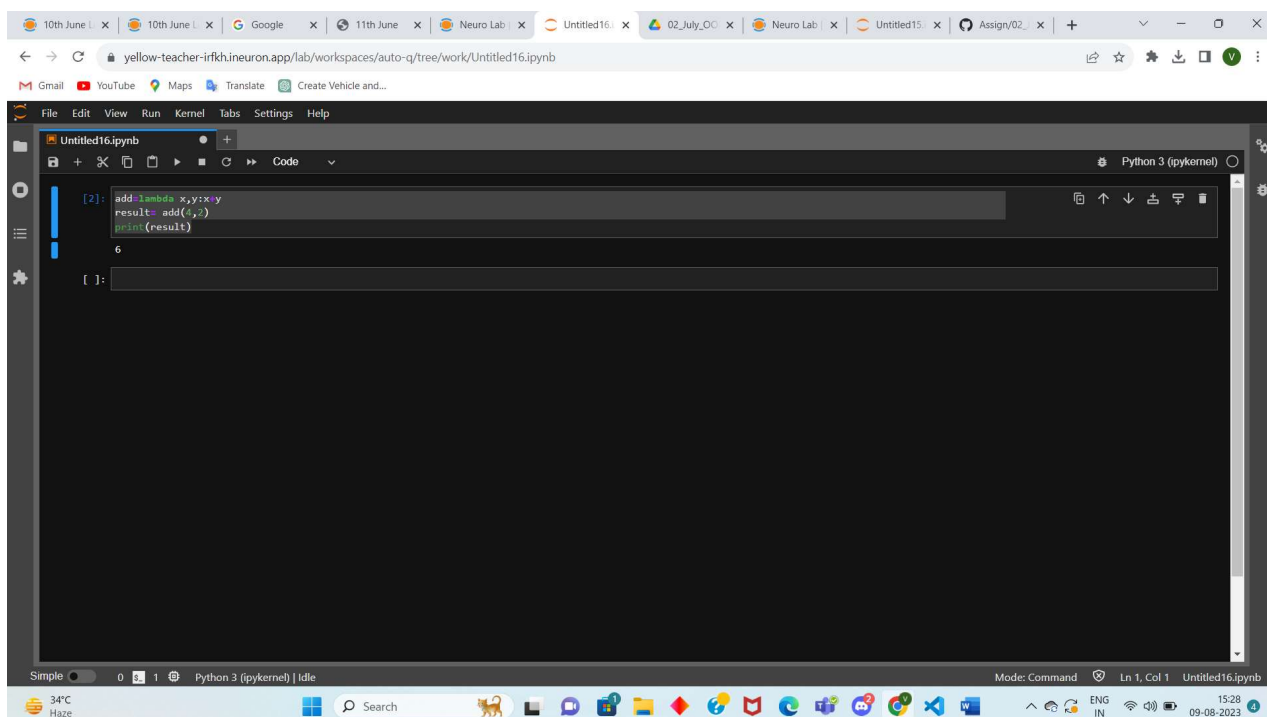
1. Lambda functions are small, anonymous functions defined with the lambda keyword.
2. They can take any number of arguments but can only have one expression.
3. The expression is evaluated and returned when the function is called.
4. Lambda functions do not require a return statement, the expression is implicitly returned.
5. You can't include statements like loops, if or else in lambda functions; only expressions are allowed.
6. Lambda functions are useful for small tasks that are not reused throughout your code.
7. They can be assigned to variables and used like regular functions.

Difference :- Lambda functions are typically more concise & are defined in a single line . Regular functions have more structured syntax with a formal header & the use of return keywords.

Lambda functions can only contain a single expression, while regular functions can have multiple statement & more complex logic.

2. Can a lambda function in Python have multiple arguments? If yes, how can you define and use them?

Ans:- Yes, a lambda function in Python can have multiple arguments. The syntax for defining a lambda function with multiple arguments is as follows:



The screenshot shows a web browser window with a Jupyter Notebook interface. The browser's address bar displays the URL: `yellow-teacher-irfkh.neuron.app/lab/workspaces/auto-q/tree/work/Untitled16.ipynb`. The notebook itself is titled "Untitled16.ipynb" and is running on a "Python 3 (ipykernel)" environment. In the code editor, the following code is entered:

```
[2]: add=lambda x,y:x+y
    result= add(4,2)
    print(result)

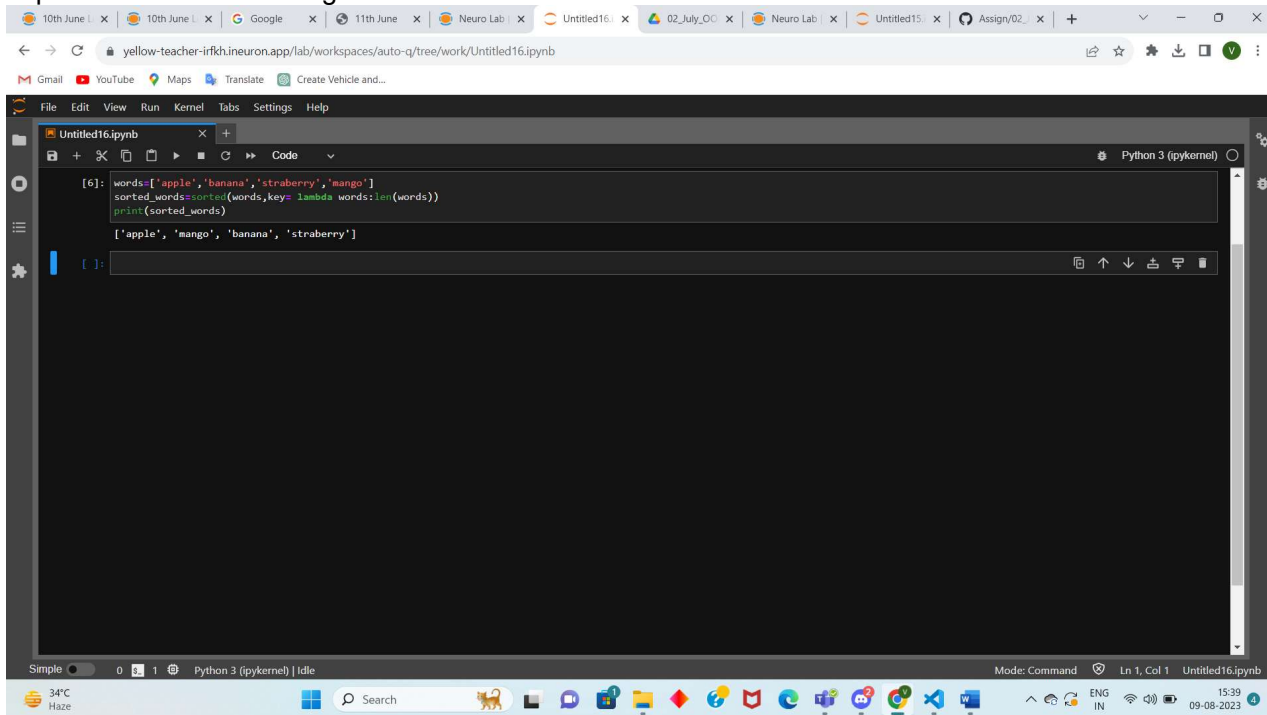
6
```

The output of the code execution is displayed below the code cell, showing the value `6`. The notebook interface includes a menu bar with options like File, Edit, View, Run, Kernel, Tabs, Settings, and Help. The bottom status bar shows the current mode as "Command" and the file name as "Untitled16.ipynb". The Windows taskbar at the very bottom indicates the system temperature is 34°C, the time is 15:28, and the date is 09-08-2023.

3. How are lambda functions typically used in Python? Provide an example use case.

Ans:-

Lambda functions are typically used in Python for situations where we need a small, anonymous function to perform a specific task, especially when we are working with higher-order functions that expect a function as an argument.



The screenshot shows a web browser with multiple tabs, including Google, Neuro Lab, and several Untitled files. The active tab is a Jupyter Notebook titled 'Untitled16.ipynb'. The notebook interface includes a menu bar (File, Edit, View, Run, Kernel, Tabs, Settings, Help) and a toolbar. The code cell shows the following Python code:

```
[6]: words=['apple','banana','strawberry','mango']
sorted_words=sorted(words,key= lambda words:len(words))
print(sorted_words)

['apple', 'mango', 'banana', 'strawberry']
```

The output of the code is displayed below the code cell: `['apple', 'mango', 'banana', 'strawberry']`. The bottom status bar indicates the kernel is 'Python 3 (ipykernel)' and is in 'Idle' mode.

4. What are the advantages and limitations of lambda functions compared to regular functions in Python?

Ans -Advantages of lambda functions in python:-

- This is particularly useful for short and straightforward operations.
- They return automatically.
- They can't have a docstring and they don't have a name.

Limitations of lambda functions:-

- Lambda functions can only consist of a single expression, which means they are not suitable for more complex logic that requires multiple statements or control structures.
- Lambda functions do not support docstrings, which are important for providing information about a function's purpose, parameters, and usage.
- Due to their anonymous nature, lambda functions are less reusable compared to named functions.

5. Are lambda functions in Python able to access variables defined outside of their own scope? Explain with an example.

Ans:- Yes, lambda functions in Python are able to access variables defined outside of their own scope.

The screenshot shows a Jupyter Notebook titled 'Untitled16.ipynb' in a web browser. The code in the cell is as follows:

```
[7]: def outer_function(x):  
    multiplier = 2  
  
    lambda_function = lambda y: y * multiplier  
  
    result = lambda_function(x)  
    return result  
  
value = 5  
output = outer_function(value)  
print(output)  
10
```

The output of the cell is 10. The interface includes a menu bar (File, Edit, View, Run, Kernel, Tabs, Settings, Help) and a status bar at the bottom showing 'Python 3 (ipykernel) | Idle'.

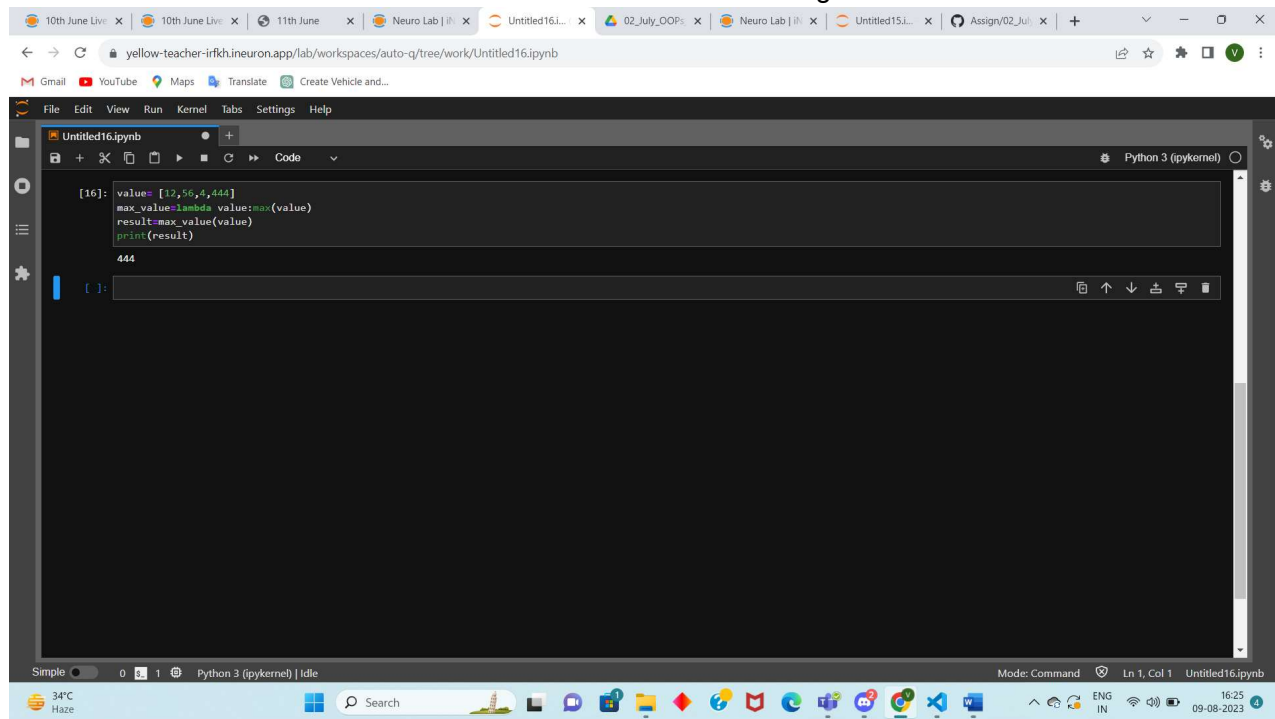
6. Write a lambda function to calculate the square of a given number.

The screenshot shows a Jupyter Notebook titled 'Untitled16.ipynb' in a web browser. The code in the cell is as follows:

```
[12]: square=lambda x : x**2  
result= square(4)  
print(result)  
16
```

The output of the cell is 16. The interface includes a menu bar (File, Edit, View, Run, Kernel, Tabs, Settings, Help) and a status bar at the bottom showing 'Python 3 (ipykernel) | Idle'.

## 7. Create a lambda function to find the maximum value in a list of integers.



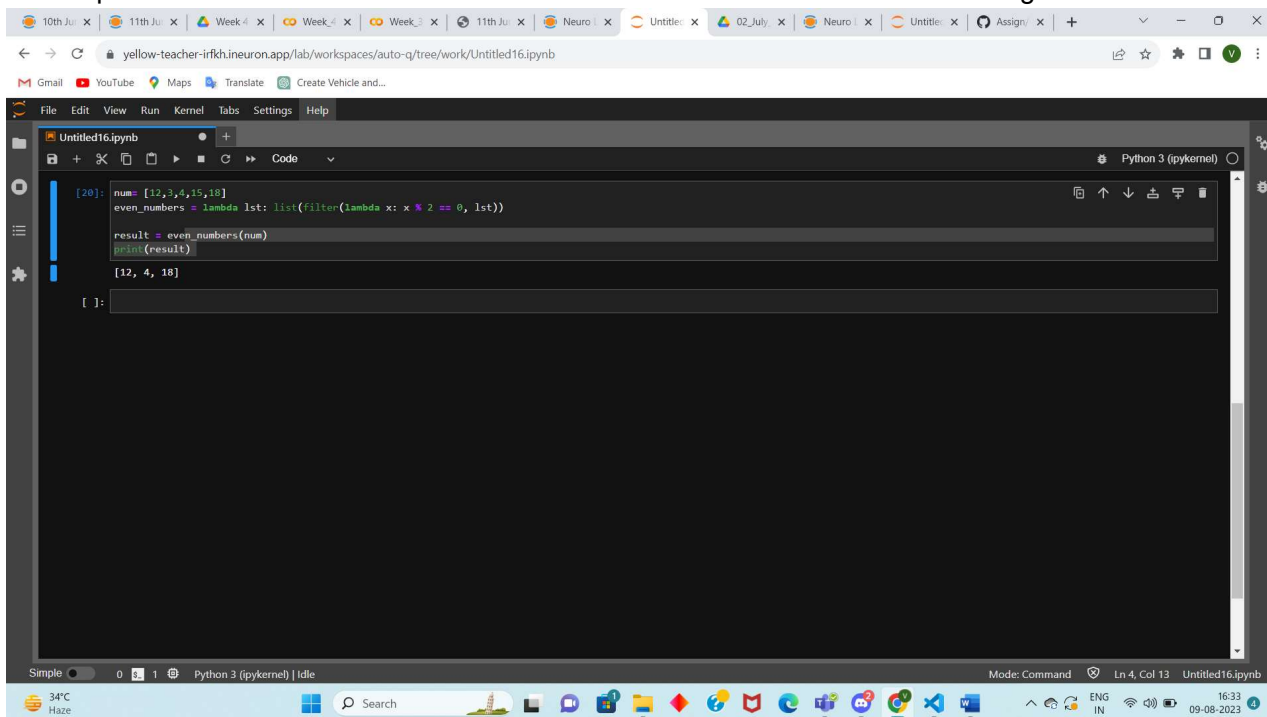
The screenshot shows a Jupyter Notebook interface with a single code cell. The code defines a list of integers, a lambda function to find the maximum, and prints the result.

```
[16]: values = [12,56,4,444]
      max_value = lambda value: max(value)
      result = max_value(values)
      print(result)

      444
```

The output of the cell is 444. The notebook is titled 'Untitled16.ipynb' and is running on a Python 3 (ipykernel) environment.

## 8. Implement a lambda function to filter out all the even numbers from a list of integers.



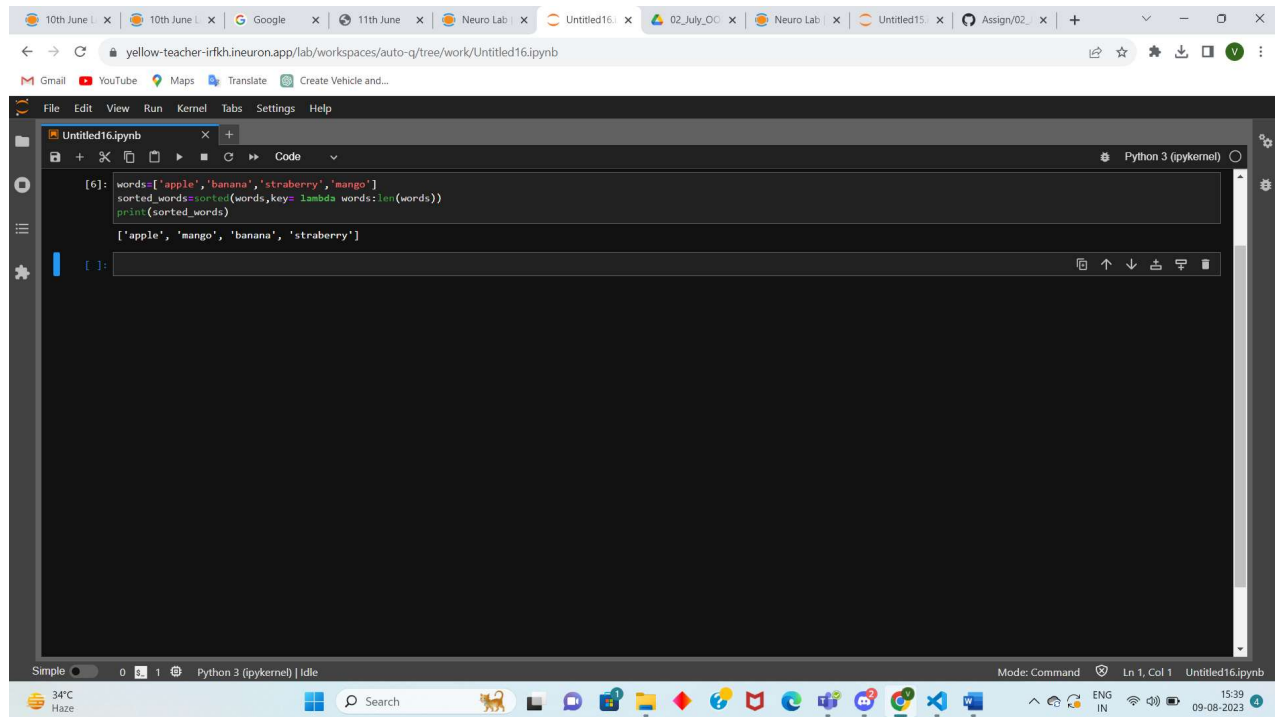
The screenshot shows a Jupyter Notebook interface with a single code cell. The code defines a list of integers, a lambda function to filter even numbers, and prints the result.

```
[20]: num = [12,3,4,15,18]
      even_numbers = lambda lst: list(filter(lambda x: x % 2 == 0, lst))
      result = even_numbers(num)
      print(result)

      [12, 4, 18]
```

The output of the cell is [12, 4, 18]. The notebook is titled 'Untitled16.ipynb' and is running on a Python 3 (ipykernel) environment.

9. Write a lambda function to sort a list of strings in ascending order based on the length of each string.



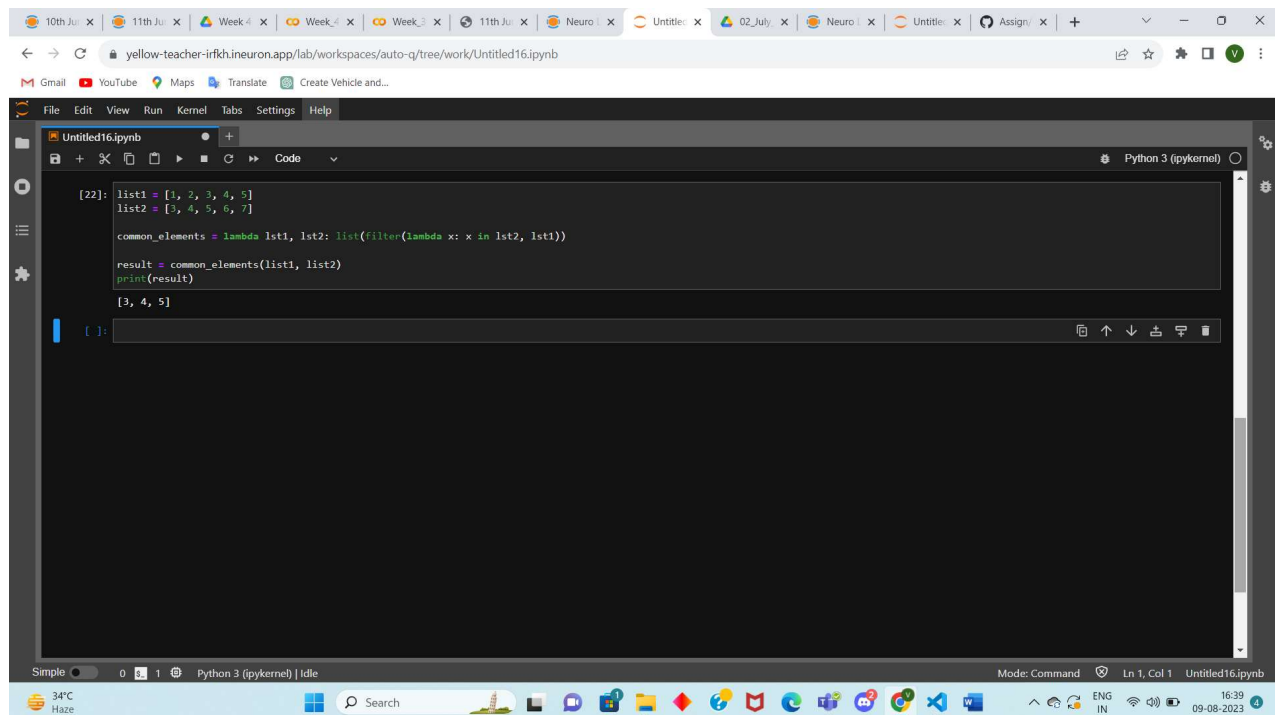
The screenshot shows a Jupyter Notebook interface with a code cell containing the following Python code:

```
[6]: words=['apple','banana','strawberry','mango']
sorted_words=sorted(words,key=lambda words:len(words))
print(sorted_words)

['apple', 'mango', 'banana', 'strawberry']
```

The output of the code is displayed below the code cell: `['apple', 'mango', 'banana', 'strawberry']`. The notebook is titled "Untitled16.ipynb" and is running on a Python 3 (ipykernel) environment.

10. Create a lambda function that takes two lists as input and returns a new list containing the common elements between the two lists.



The screenshot shows a Jupyter Notebook interface with a code cell containing the following Python code:

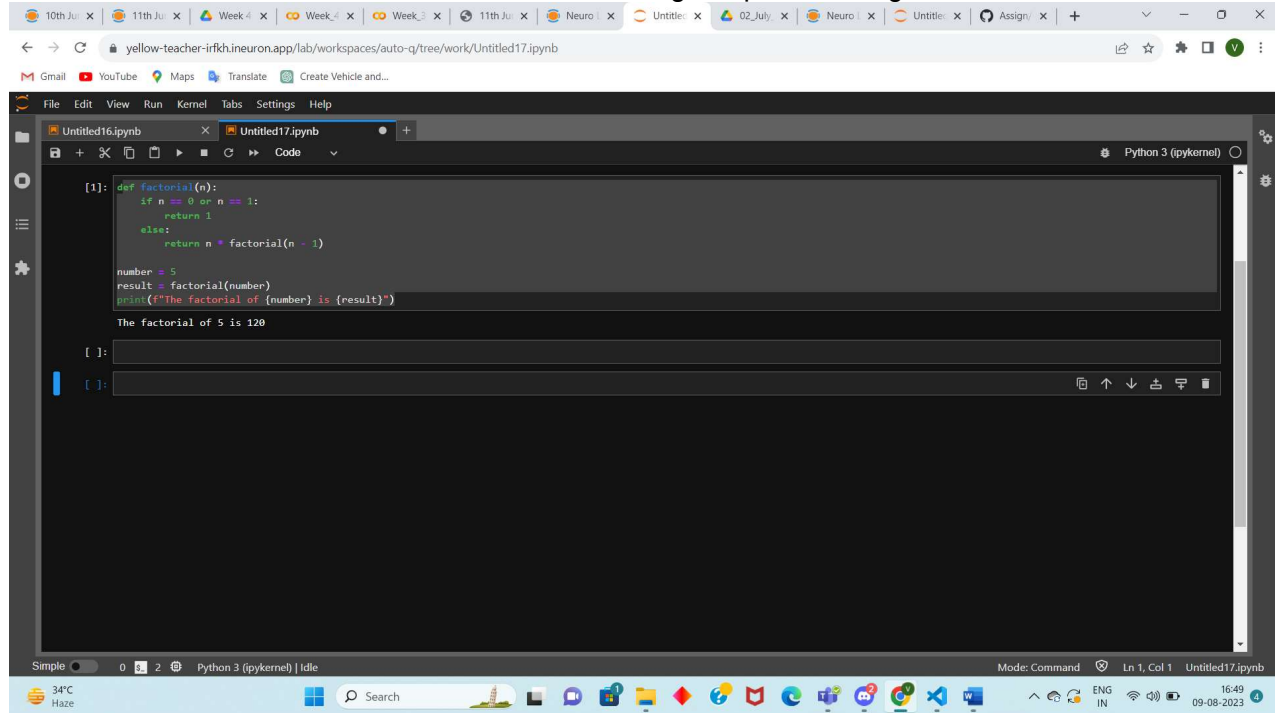
```
[22]: list1 = [1, 2, 3, 4, 5]
list2 = [3, 4, 5, 6, 7]

common_elements = lambda lst1, lst2: list(filter(lambda x: x in lst2, lst1))
result = common_elements(list1, list2)
print(result)

[3, 4, 5]
```

The output of the code is displayed below the code cell: `[3, 4, 5]`. The notebook is titled "Untitled16.ipynb" and is running on a Python 3 (ipykernel) environment.

11. Write a recursive function to calculate the factorial of a given positive integer.

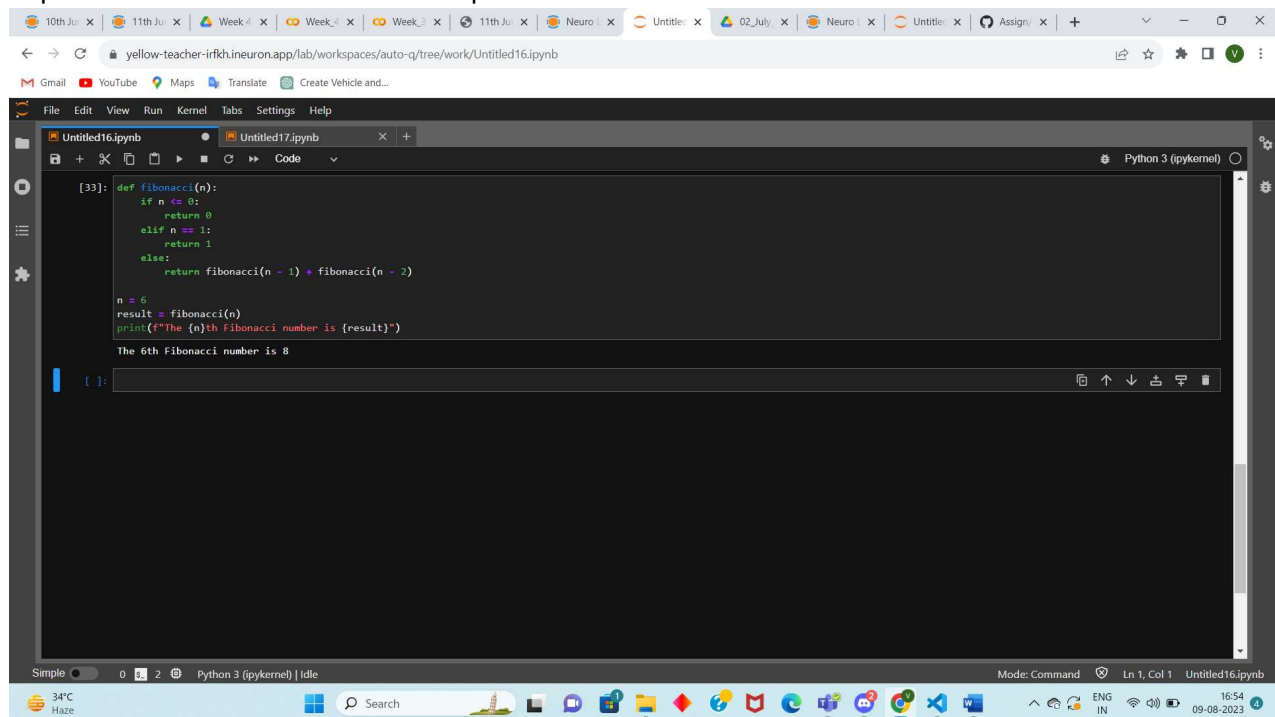


The screenshot shows a Jupyter Notebook interface with a browser window at the top displaying the URL `yellow-teacher-irfkh.neuron.app/lab/workspaces/auto-q/tree/work/Untitled17.ipynb`. The notebook has two tabs: `Untitled16.ipynb` and `Untitled17.ipynb`. The active tab, `Untitled17.ipynb`, contains the following Python code in a cell:

```
[1]: def factorial(n):  
    if n == 0 or n == 1:  
        return 1  
    else:  
        return n * factorial(n - 1)  
  
    number = 5  
    result = factorial(number)  
    print(f"The factorial of {number} is {result}")  
  
The factorial of 5 is 120
```

The output of the code is displayed below the cell: `The factorial of 5 is 120`. The notebook interface includes a menu bar (File, Edit, View, Run, Kernel, Tabs, Settings, Help) and a status bar at the bottom showing `Simple`, `0`, `2`, `Python 3 (pykernel)`, `Idle`, `Mode: Command`, `Ln 1, Col 1`, and `Untitled17.ipynb`. The system tray at the bottom shows the temperature as `34°C`, weather as `Haze`, and the date as `09-08-2023`.

12. Implement a recursive function to compute the nth Fibonacci number.

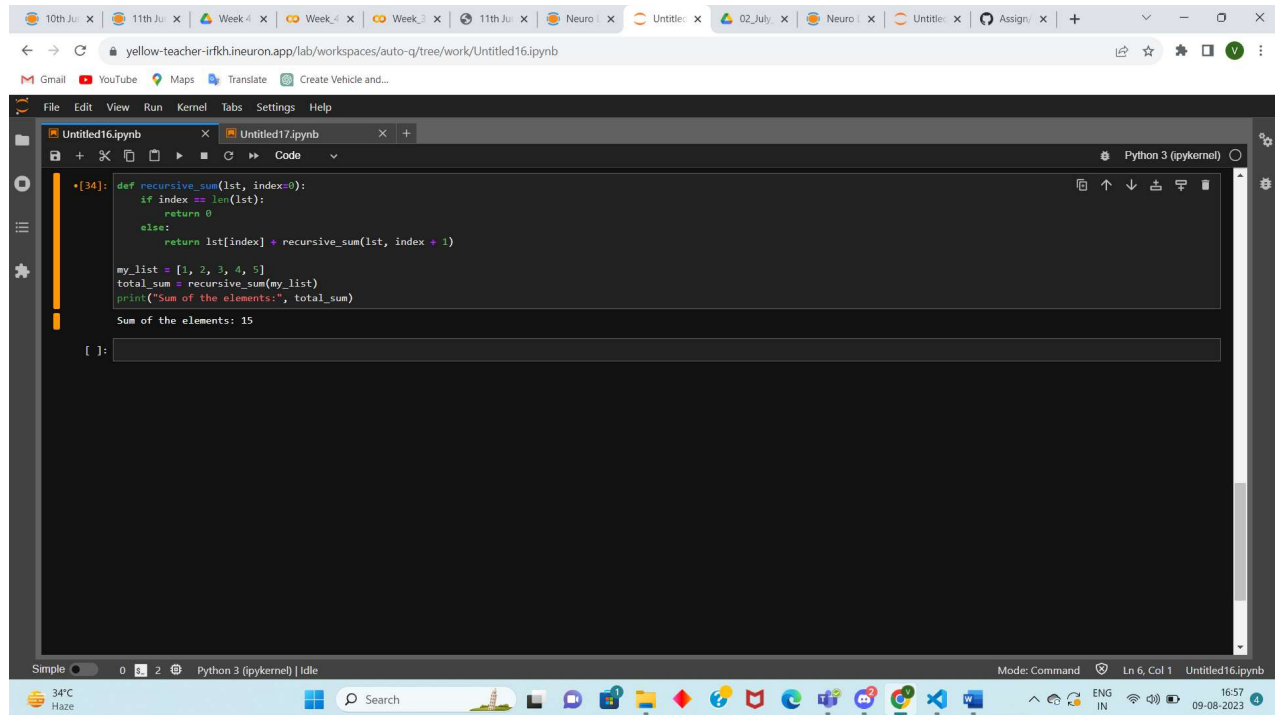


The screenshot shows a Jupyter Notebook interface with a browser window at the top displaying the URL `yellow-teacher-irfkh.neuron.app/lab/workspaces/auto-q/tree/work/Untitled16.ipynb`. The notebook has two tabs: `Untitled16.ipynb` and `Untitled17.ipynb`. The active tab, `Untitled16.ipynb`, contains the following Python code in a cell:

```
[33]: def fibonacci(n):  
    if n <= 0:  
        return 0  
    elif n == 1:  
        return 1  
    else:  
        return fibonacci(n - 1) + fibonacci(n - 2)  
  
    n = 6  
    result = fibonacci(n)  
    print(f"The {n}th Fibonacci number is {result}")  
  
The 6th Fibonacci number is 8
```

The output of the code is displayed below the cell: `The 6th Fibonacci number is 8`. The notebook interface includes a menu bar (File, Edit, View, Run, Kernel, Tabs, Settings, Help) and a status bar at the bottom showing `Simple`, `0`, `2`, `Python 3 (pykernel)`, `Idle`, `Mode: Command`, `Ln 1, Col 1`, and `Untitled16.ipynb`. The system tray at the bottom shows the temperature as `34°C`, weather as `Haze`, and the date as `09-08-2023`.

13. Create a recursive function to find the sum of all the elements in a given list.



The screenshot shows a Jupyter Notebook with a single code cell. The code defines a recursive function `recursive_sum` that takes a list `lst` and an index `index` as arguments. The function checks if the index is equal to the length of the list. If so, it returns 0. Otherwise, it returns the element at the current index plus the result of the recursive call with the index incremented by 1. Below the function definition, a list `my_list = [1, 2, 3, 4, 5]` is created, the function is called with `recursive_sum(my_list)`, and the result is printed. The output of the cell shows the sum of the elements: 15.

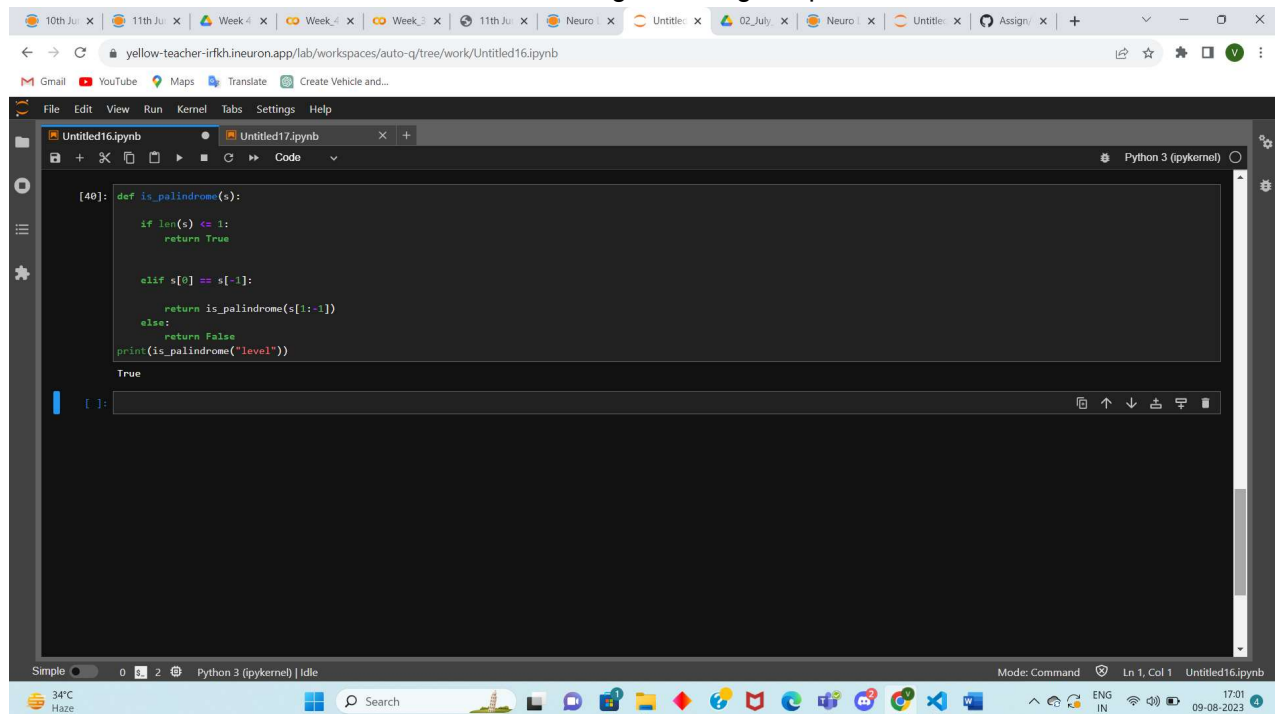
```
[34]: def recursive_sum(lst, index=0):
      if index == len(lst):
          return 0
      else:
          return lst[index] + recursive_sum(lst, index + 1)

      my_list = [1, 2, 3, 4, 5]
      total_sum = recursive_sum(my_list)
      print("Sum of the elements:", total_sum)

      Sum of the elements: 15

      [ ]:
```

14. Write a recursive function to determine whether a given string is a palindrome.



The screenshot shows a Jupyter Notebook with a single code cell. The code defines a recursive function `is_palindrome` that takes a string `s` as an argument. The function checks if the length of the string is less than or equal to 1. If so, it returns True. Otherwise, it checks if the first and last characters are equal. If they are, it returns the result of the recursive call with the string sliced from the second character to the second-to-last character. If they are not equal, it returns False. Below the function definition, the function is called with `is_palindrome("level")`, and the result is printed. The output of the cell shows the result: True.

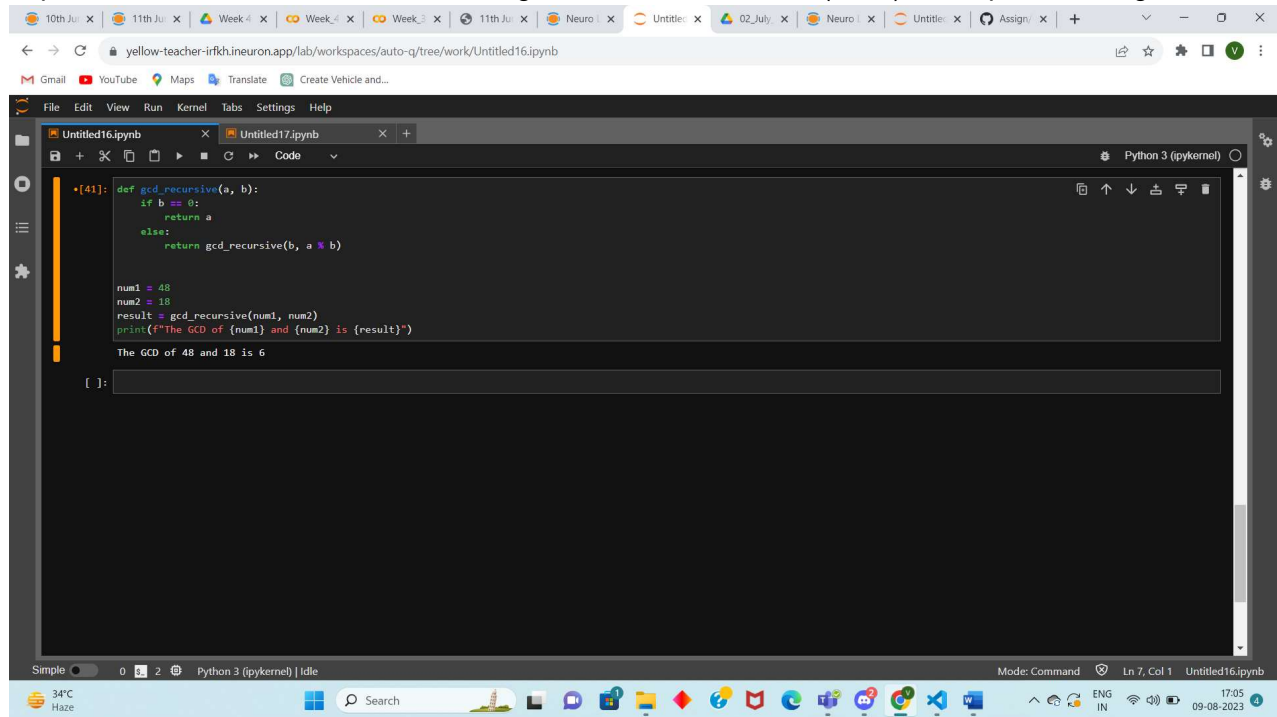
```
[40]: def is_palindrome(s):
      if len(s) <= 1:
          return True

      elif s[0] == s[-1]:
          return is_palindrome(s[1:-1])
      else:
          return False
      print(is_palindrome("level"))

      True

      [ ]:
```

15. Implement a recursive function to find the greatest common divisor (GCD) of two positive integers.



The screenshot shows a web browser window with a Jupyter Notebook interface. The browser's address bar displays the URL: `yellow-teacher-irfkh.neuron.app/lab/workspaces/auto-q/tree/work/Untitled16.ipynb`. The notebook has two tabs: `Untitled16.ipynb` (active) and `Untitled17.ipynb`. The active tab contains the following Python code:

```
[41]: def gcd_recursive(a, b):  
    if b == 0:  
        return a  
    else:  
        return gcd_recursive(b, a % b)  
  
num1 = 48  
num2 = 18  
result = gcd_recursive(num1, num2)  
print(f"The GCD of {num1} and {num2} is {result}")  
  
The GCD of 48 and 18 is 6  
  
[ ]:
```

The notebook's status bar at the bottom indicates the kernel is `Python 3 (pykernel)` and is in `Idle` mode. The system tray at the very bottom shows the temperature as `34°C` with `Haze` weather, the time as `17:05`, and the date as `09-08-2023`.